Linux Fun

Paul Cobbaut

About

3 October 2021

Hello and welcome to this pdf, I really hope you enjoy it and learn something.

This pdf is **not finished**, but I decided to release it anyway because the parts that are written are much better than the last release from 25 May 2015. I believe there is some useful stuff up until page 600 or so. After that it is only a draft.

I wrote this pdf between June 2019 and January 2020. I have not found the energy to finish this work, maybe I will one day, maybe not.

You are free to give copies of this pdf to people close to you, but please refer them to **http://linux-training.be** as this website will always contain the latest updates.

Some people are selling my work, I have no relation to any of them, I make no money of this. This work is free for everyone. If you want to reward me or pay me, then do something nice for a person. It may make their day and give them a valuable memory.

Also, there is no advertising on this website and no registration of any kind.

This book is dedicated to all the nice and honest people on this planet.

Contents

Part I

Introduction to Linux

Chapter 1

Linux history

todo

Chapter 2

Linux distributions

todo

Chapter 3

Installing Linux

todo

Chapter 4

Start using Linux

4.1 Who am I?

4.1.1 who

The **who** command displays a list of logged on users. The screenshot below shows two logged on users, named **root** and **paul**.

```
paul@debian10:~$ who
root     tty1         2019-07-24 18:49
paul     pts/0        2019-07-24 21:12 (192.168.56.1)
paul@debian10:~$
```

4.1.2 who am i

The **who am i** command will display your line in the list of logged on users, as shown in the next screenshot.

```
paul@debian10:~$ who am i
paul     pts/0        2019-07-24 21:12 (192.168.56.1)
paul@debian10:~$
```

4.2 Where am I?

4.2.1 hostname

The **hostname** command will display the name of the computer (on which you are logged on). By default this name is also displayed in your command prompt.

```
paul@debian10:~$ hostname
debian10
paul@debian10:~$
```

4.3 Change your password

4.3.1 passwd

You can use the **passwd** command to change your password (for that user, on that computer).

```
paul@debian10:~$ passwd
Changing password for paul.
Current password:
New password:
Retype new password:
passwd: password updated successfully
paul@debian10:~$
```

Note
When typing the old or new password, nothing is displayed on the screen.

4.4 When are we?

4.4.1 date

The **date** command will display the current date and time.

```
paul@debian10:~$ date
Wed 24 Jul 2019 09:55:20 PM CEST
paul@debian10:~$
```

There are many options possible to the **date** command. But this is a humble introduction so we will keep it simple. Try executing **date --date="Next Sunday"** like in this screenshot.

```
paul@debian10:~$ date --date="Next Sunday"
Sun 28 Jul 2019 12:00:00 AM CEST
paul@debian10:~$
```

4.4.2 cal

You can use the **cal** command to display a calendar of the current month.

```
paul@debian10:~$ cal
     July 2019
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

paul@debian10:~$
```

And this calendar goes way back in time. In this screenshot we ask for February of 1970.

```
paul@debian10:~$ cal 02 1970
   February 1970
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28


paul@debian10:~$
```

Tip
I was born on a Friday, on which day of the week were you born?

4.5 What was I doing?

Instead of typing a command multiple times, you can use the **arrow up** key to see and execute previously entered commands.

Try now to use **arrow up** and execute some of your previous commands.

4.6 Get me out of here.

You can **exit** from this Debian Linux computer by typing **exit** or **Ctrl-d**. Both commands have the same effect.

Note
Technically **Ctrl-d** sends an ASCII EOF character to the shell.

4.7 Turn the computer off

If you created a virtual machine for this book, then you can turn it off by typing **poweroff** as **root**. That means you need to log on as the **root** user, since only **root** has the power to turn a computer off.

```
root@debian10:~# poweroff
```

4.8 Cheat sheet

Table 4.1: Start using Linux

command	explanation
who	list all users logged on the system
who am i	list your logon information
hostname	display the name of the computer
passwd	change your password
date	display the current time and date
cal	display this month's calendar
cal 01 1970	display a calendar of January 1970
arrow up key	go to previous command(s)
exit	log out of your current session
Ctrl-d	send an EOF character (log out of your current session)

4.9 Practice

1. Log on to a Linux computer and execute the **who** command.
2. Now execute the **who am i** command.
3. Display the name of the computer on which you logged on.
4. Change your password (and remember it!)
5. Display the current date and time.
6. Display the calender of the year 2019.
7. Exit the Linux computer and log back on again.

4.10 Solution

1. Log on to a Linux computer and execute the **who** command.

```
who
```

2. Now execute the **who am i** command.

```
who am i
```

3. Display the name of the computer on which you logged on.

```
hostname
```

4. Change your password (and remember it!)

```
passwd
```

After typing **passwd** type your current password, then enter, then your new password, then enter again, and again your new password.

5. Display the current date and time.

```
date
```

6. Display the calender of the year 2019.

```
cal 2019
```

7. Exit the Linux computer and log back on again.

```
exit
```

or

```
Ctrl-d
```

Chapter 5

Directories

5.1 Creating directories

5.1.1 ls

Files are always stored in a **directory** and there is always a **current directory**. To see the *list* of files in the current directory type **ls**.

```
paul@debian10:~$ ls
paul@debian10:~$
```

There are likely no files yet in your **home directory**, so there is no output to this command.

Note
Later in this book we will look at the *hidden files* in this directory.

5.1.2 mkdir

You can create a new empty directory with the **mkdir** command. In this screenshot we create two directories.

```
paul@debian10:~$ mkdir data
paul@debian10:~$ mkdir backup
paul@debian10:~$ ls
backup  data
paul@debian10:~$
```

The two new directories that you created are empty. You can verify this by executing **ls** with the name of the directory as an argument (aka parameter). Just like the two **mkdir** commands, there is no output when entering these two **ls** commands.

```
paul@debian10:~$ ls data
paul@debian10:~$ ls backup
paul@debian10:~$
```

Tip
In general, when there is no output to a Linux command, you may assume that everything worked fine!

5.2 Display the current directory

Your current directory can be displayed with the **pwd** (print working directory) command.

```
paul@debian10:~$ pwd
/home/paul
paul@debian10:~$
```

5.3 Entering directories

The **cd** (change directory) command allows you to enter an existing directory. This command will change your current directory.

```
paul@debian10:~$ cd data
paul@debian10:~/data$ pwd
/home/paul/data
paul@debian10:~/data$
```

Typing **cd** without any parameter will put you back in your **home directory**.

```
paul@debian10:~/data$ pwd
/home/paul/data
paul@debian10:~/data$ cd
paul@debian10:~$ pwd
/home/paul
paul@debian10:~$
```

Here is another example of changing directories (to /etc) and then getting back to the **home directory**.

```
paul@debian10:~$ cd /etc
paul@debian10:/etc$ pwd
/etc
paul@debian10:/etc$ cd
paul@debian10:~$ pwd
/home/paul
paul@debian10:~$
```

5.4 Current directory in prompt

You may have noticed by now that your current directory is shown in your prompt. This is default behaviour on Debian Linux. We will see later on this book how you change the appearance of your prompt.

Note
Note that the **home directory** is abbreviated to a ~ (tilde). So when we write ~/**data** we mean the **data** directory in your home directory.

Tip
Entering the **cd** command is identical to entering the **cd** ~ command.

5.5 Path completion

Linux people are lazy, so they developed **path completion**. In this case this means that you can type the first letter of a directory, followed by the **tab key** when using the **cd** command.

Try this with the two directories in your home directory by typing **"cd d tab key"** or **"cd b tab key"**.

```
paul@debian10:~$ cd data/
paul@debian10:~/data$ cd
paul@debian10:~$ cd backup/
paul@debian10:~/backup$
```

Note
Notice how the **tab key** path completion puts a **"/"** at the end of the directory. This is to indicate that it is a directory. Later we will see **file completion** which lacks this trailing slash.

5.6 Removing directories

You can remove *empty* directories with the **rmdir** command. In the screenshot below we remove the **data** directory.

```
paul@debian10:~$ rmdir data/
paul@debian10:~$ ls
backup
paul@debian10:~$
```

Note
Did you use the **tab key** when removing this directory?

For **rmdir** to work, the directory must be empty! In the following screenshot we create a directory **"pictures"** inside the directory **"data"**. This means the **"data"** directory is not empty, so the **rmdir** command fails.

```
paul@debian10:~$ mkdir data
paul@debian10:~$ mkdir data/pictures
paul@debian10:~$ ls
backup  data
paul@debian10:~$ ls data/
pictures
paul@debian10:~$ rmdir data/
rmdir: failed to remove data/: Directory not empty
paul@debian10:~$
```

5.7 Case sensitive

The name of a directory is **case sensitive**! In the screenshot below we create three different directories.

```
paul@debian10:~$ mkdir music
paul@debian10:~$ mkdir Music
paul@debian10:~$ mkdir MUSIC
paul@debian10:~$ ls
backup  data  music  Music  MUSIC
paul@debian10:~$
```

Let us quickly remove two of those three directories.

```
paul@debian10:~$ rmdir MUSIC/ Music/
paul@debian10:~$
```

5.8 Creating the necessary parent directories

The following command fails because the parent directory **"cars"** does not exist.

```
paul@debian10:~$ <b>mkdir cars/mercedes</b>
mkdir: cannot create directory ‘cars/mercedes’: No such file or directory
paul@debian10:~$
```

But we can add the **"-p"** option to **mkdir** to force the creation of the parent directory **cars**.

```
paul@debian10:~$ mkdir -p cars/mercedes
paul@debian10:~$ ls
backup  cars  data  music
paul@debian10:~$ ls cars
mercedes
paul@debian10:~$
```

5.9 Searching for directories

By now we have forgotten where we put the **"pictures"** directory so we have to search for it. The screenshot below uses the **find** command to search for a directory named **pictures**.

```
paul@debian10:~$ find -name pictures
./data/pictures
paul@debian10:~$
```

The **find** command will by default look in the **current directory** and in **all subdirectories** of the current directory.

5.10 Entering a parent directory

You can use **cd ..** to enter the parent directory of the current directory. In the example below we use **cd ..** to go from **/home/paul/cars/mercedes** to **/home/paul/cars**.

```
paul@debian10:~$ cd cars/mercedes/
paul@debian10:~/cars/mercedes$ pwd
/home/paul/cars/mercedes
paul@debian10:~/cars/mercedes$ cd ..
paul@debian10:~/cars$ pwd
/home/paul/cars
paul@debian10:~/cars$
```

Note
Note that there is a space between **cd** and **..** !

5.11 The root directory

What happens if we keep going to the parent directory? Let us test.

```
paul@debian10:~/cars$ pwd
/home/paul/cars
paul@debian10:~/cars$ cd ..
paul@debian10:~$ pwd
/home/paul
paul@debian10:~$ cd ..
paul@debian10:/home$ pwd
/home
paul@debian10:/home$ cd ..
paul@debian10:/$ pwd
/
paul@debian10:/$ cd ..
paul@debian10:/$ pwd
/
paul@debian10:/$
```

You can see that we reach the end when we are in the **"/"** directory. This **"/"** is called the **root directory**. The **root directory** is the start of all files and directories on your Linux computer.

You cannot remove directories in the root directory, they are protected by permissions (which we will discuss later).

```
paul@debian10:~$ rmdir /etc
rmdir: failed to remove /etc: Permission denied
paul@debian10:~$ rmdir /bin
rmdir: failed to remove /bin: Permission denied
paul@debian10:~$
```

5.12 Absolute and relative paths

5.12.1 absolute path

Whenever you start the name of a directory (or a file) with a **"/"** , then the computer will look for that directory (or file) in the **root directory**. In the example below we enter three directories using an **absolute path**.

```
paul@debian10:~$ cd /var
paul@debian10:/var$ cd /etc
paul@debian10:/etc$ cd /usr/bin
paul@debian10:/usr/bin$
```

The commands will fail if you forget the starting **"/"** !

```
paul@debian10:~$ cd var
-bash: cd: var: No such file or directory
paul@debian10:~$ cd etc
-bash: cd: etc: No such file or directory
paul@debian10:~$ cd usr/bin
-bash: cd: usr/bin: No such file or directory
paul@debian10:~$
```

5.12.2 relative path

Whenever you use the name of a directory (or file) without a starting **"/"** , then you are using a **relative** path. Using a **relative path** only works if the target directory is in the current directory.

```
paul@debian10:~$ ls
backup  cars  data  music
paul@debian10:~$ cd cars
paul@debian10:~/cars$ cd data
-bash: cd: data: No such file or directory
paul@debian10:~/cars$ cd /home/paul/data
paul@debian10:~/data$
```

5.13 Cheat sheet

Table 5.1: Directories

command	**explanation**
ls	list the contents of the current directory
ls foo	list the contents of the **foo** directory
ls /bar	list the contents of the **/bar** directory
mkdir foo	make (create) a new directory named **foo**
mkdir foo/bar	create the **bar** directory in the **foo** directory
mkdir -p foo/bar	create both **foo** and **foo/bar** if they don't exist
pwd	print the name of the current (working) directory
cd	change into your home directory
cd foo	change into the **foo** directory
cd /bar	change into the **/bar** directory
cd ..	change to the parent directory of the current directory
rmdir foo	remove (delete) the **empty** directory **foo**
find -name foo	search for the directory (or file) named **foo**
find -type d -name foo	search for the directory named **foo**
/	/ is the name of the **root** directory
cd /foo	use an **absolute** path to enter the **/foo** directory
cd foo	use a **relative** path to enter the **foo** directory
tab key	complete the path you are typing

5.14 Practice

1. Display the name of your current directory.
2. Change your current directory to your home directory.
3. Create the directories **~/data** and **~/backup** .
4. Remove the directory **~/data** .
5. In one command create the directory **~/data/payable/invoices** .
6. Remove the **invoices** directory.
7. Enter the **/var** directory and display its contents.
8. Go back to your home directory using an **absolute path**.
9. Enter the **~/data** directory using a **relative path**.
10. Enter your home directory and then search for the **payable** directory.

5.15 Solution

1. Display the name of your current directory.

```
pwd
```

2. Change your current directory to your home directory.

```
cd
```

3. Create the directories **~/data** and **~/backup** .

```
mkdir ~/data ~/backup
```

There are other solutions to this, for example:

```
cd
mkdir data
mkdir backup
```

4. Remove the directory **~/data** .

```
rmdir ~/data
```

5. In one command create the directory **~/data/payable/invoices** .

```
mkdir -p ~/data/payable/invoices
```

6. Remove the **invoices** directory.

```
rmdir ~/data/payable/invoices
```

Or you could also do it like this:

```
cd data
cd payable
rmdir invoices
```

7. Enter the **/var** directory and display its contents.

```
cd /var
ls
```

8. Go back to your home directory using an **absolute path**.

```
cd /home/paul
```

9. Enter the **~/data** directory using a **relative path**.

```
cd data/
```

10. Enter your home directory and then search for the **payable** directory.

```
cd
find -name payable
```

Chapter 6

Files

6.1 Listing files

Files are always stored in a **directory** and there is always a **current directory**. To see the *list* of files in the current directory type **ls**.

```
paul@debian10:~$ ls
backup  cars  data  music
paul@debian10:~$
```

Maybe you have some directories from the previous chapter. If not, then create some with **mkdir**.

When executing a simple **ls** command, you cannot see the difference between a file and a directory. To see the difference, you can use **ls -l** (that is a lowercase letter l from **l**ong list).

```
paul@debian10:~$ ls -l
total 16
drwxr-xr-x 2 paul paul 4096 Jul 24 23:07 backup
drwxr-xr-x 3 paul paul 4096 Jul 25 01:24 cars
drwxr-xr-x 3 paul paul 4096 Jul 25 01:00 data
drwxr-xr-x 2 paul paul 4096 Jul 25 01:16 music
paul@debian10:~$
```

Notice the first character of each line is a letter **d**. This is the **d** from **directory**. (We will explain the rest later in this book).

6.2 Creating files

Most applications on your Linux computer will be able to create files. We will start with the very simple **touch** command to create empty files. The example below creates three empty files in the **cars** directory and then lists the contents of the **cars** directory.

```
paul@debian10:~$ cd cars
paul@debian10:~/cars$ touch bmw
paul@debian10:~/cars$ touch fiat kia
paul@debian10:~/cars$ ls -l
total 4
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 bmw
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 fiat
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 kia
drwxr-xr-x 2 paul paul 4096 Jul 25 01:24 mercedes
paul@debian10:~/cars$
```

Notice that the three files have size 0, and that their lines start with a **"-"** (called a hyphen or a dash). The **"-"** tells you that they are **regular files**.

There is a time travel trick to **touch**; when using **touch -t** you can create files with a custom time stamp. See this example to create a file with the 1970-Feb-27 at 11:30 time stamp. (Later in the backup chapter is an example of how this can be useful.)

```
paul@debian10:~$ touch -t 197002271130 oldfile
paul@debian10:~$ ls -l oldfile
-rw-r--r-- 1 paul paul 0 Feb 27  1970 oldfile
paul@debian10:~$
```

6.3 Removing files

Now let us remove two files in the **cars** directory. To remove files we use the **rm** command.

```
paul@debian10:~/cars$ ls -l
total 4
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 bmw
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 fiat
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 kia
drwxr-xr-x 2 paul paul 4096 Jul 25 01:24 mercedes
paul@debian10:~/cars$ rm bmw kia
paul@debian10:~/cars$ ls -l
total 4
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 fiat
drwxr-xr-x 2 paul paul 4096 Jul 25 01:24 mercedes
paul@debian10:~/cars$
```

Notice that you cannot remove a directory with **rm**.

```
paul@debian10:~/cars$ ls -l
total 4
-rw-r--r-- 1 paul paul    0 Jul 25 03:52 fiat
drwxr-xr-x 2 paul paul 4096 Jul 25 01:24 mercedes
paul@debian10:~/cars$ rm mercedes/
rm: cannot remove mercedes/: Is a directory
paul@debian10:~/cars$
```

Tip
Did you use tab-completion when typing mercedes?

6.4 Removing multiple files

The **rm** command can remove many files at once. We will see later in the book how to use *globbing* with commands. In this example we create three files and then we remove two of them. Notice that **rm** with the option **-i** will ask for confirmation before erasing a file.

```
paul@debian10:~$ mkdir files
paul@debian10:~$ cd files
paul@debian10:~/files$ touch one two three
paul@debian10:~/files$ ls
one  three  two
paul@debian10:~/files$ rm -i one two three
rm: remove regular empty file one? y
rm: remove regular empty file two? n
rm: remove regular empty file three? y
paul@debian10:~/files$ ls
two
paul@debian10:~/files$
```

6.5 Recursive force removal of a directory

Are you ready for the most dangerous command on Linux? Because we can give two options to the **rm** command to **f**orce it to **r**ecursively remove all files and directories in a directory.

See this example for the power of **rm -rf**.

```
paul@debian10:~/cars$ ls
fiat  mercedes
paul@debian10:~/cars$ cd ..
paul@debian10:~$ ls
backup  cars  data  music
paul@debian10:~$ rm -rf cars
paul@debian10:~$ ls
backup  data  music
paul@debian10:~$
```

As you can see, no warning is given. The **rm -rf** command will remove a directory and everything in it. There is no warning given, no confirmation asked, there is also no recovery from this (except for forensic tools).

6.6 Downloading files with wget

Download the following five files from the internet with **wget**. Make sure that your home directory is your current directory!

Tip
Use the **arrow-up key** instead of re-writing the same URL five times.

```
paul@debian10:~$ <1b>cd</b>
paul@debian10:~$ <1b>wget http://linux-training.be/all.txt.gz</b>
--2019-07-27 13:16:59--  http://linux-training.be/all.txt.gz
Resolving linux-training.be (linux-training.be)... 88.151.243.8
Connecting to linux-training.be (linux-training.be)|88.151.243.8|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 332067 (324K) [application/x-gzip]
Saving to: 'all.txt.gz'

all.txt.gz             100%[=========================>] 324.28K  --.-KB/s    in 0.08s

2019-07-27 13:16:59 (4.06 MB/s) - 'all.txt.gz' saved [332067/332067]

paul@debian10:~$ <1b>wget http://linux-training.be/linuxfun.pdf</b>
--2019-07-27 13:17:08--  http://linux-training.be/linuxfun.pdf
Resolving linux-training.be (linux-training.be)... 88.151.243.8
Connecting to linux-training.be (linux-training.be)|88.151.243.8|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7022363 (6.7M) [application/pdf]
Saving to: 'linuxfun.pdf'

linuxfun.pdf           100%[=========================>]   6.70M  6.54MB/s    in 1.0s

2019-07-27 13:17:09 (6.54 MB/s) - 'linuxfun.pdf' saved [7022363/7022363]

paul@debian10:~$ <1b>wget http://linux-training.be/dates.txt</b>
<cut output>
paul@debian10:~$ <1b>wget http://linux-training.be/wolf.png</b>
<cut output>
paul@debian10:~$ <1b>wget http://linux-training.be/studentfiles.html</b>
<cut output>
paul@debian10:~$
```

Once downloaded you should be able to find them in your home directory.

```
paul@debian10:~$ ls
all.txt.gz  backup  data  dates.txt  linuxfun.pdf  music  studentfiles.html  wolf.png
paul@debian10:~$
```

6.7 Identifying files

There is a command named **file** that can help in identifying files. For example in the screenshot below we use the **file** command to obtain information about **all.txt.gz**.

```
paul@debian10:~$ file all.txt.gz
all.txt.gz: gzip compressed data, was "allfiles.txt", last modified: Thu Jul 25 02:59:57  ↩
    2019, from Unix, original size 2903154
paul@debian10:~$
```

As you can see the **file** command gives a lot of information; the file is **gzip compressed**, it was **allfiles.txt** and its original size was **2903154** (bytes). Let's do the same for the **wolf.png** file.

```
paul@debian10:~$ file wolf.png
wolf.png: PNG image data, 1280 x 960, 8-bit/color RGB, non-interlaced
paul@debian10:~$
```

As expected, the **file** command gives us PNG image information. Try for yourself on the other files, and on a directory. The output should be similar to this screenshot.

```
paul@debian10:~$ file dates.txt
dates.txt: ASCII text
paul@debian10:~$ file linuxfun.pdf
linuxfun.pdf: PDF document, version 1.4
paul@debian10:~$ file studentfiles.html
studentfiles.html: HTML document, ASCII text
paul@debian10:~$ file music/
music/: directory
paul@debian10:~$
```

Tip
Instead of typing **file studentfiles.html** you can type **file s** *tab key*.

6.8 Copying files

6.8.1 cp

Copying files is the simplest form of a backup. In this example we use the **cp** command to copy two files to the **backup** directory.

```
paul@debian10:~$ ls backup/
paul@debian10:~$ cp dates.txt backup/
paul@debian10:~$ cp linuxfun.pdf backup/
paul@debian10:~$ ls backup/
dates.txt  linuxfun.pdf
paul@debian10:~$
```

Notice that we use two arguments to the **cp** command. The first one is the file to copy, and the second one is the target directory.

You can copy multiple files in one command. To do this you have to list all the files and provide the target directory as the very last argument. In the example below we copy three files to the backup directory.

```
paul@debian10:~$ cp all.txt.gz studentfiles.html wolf.png backup/
paul@debian10:~$
```

6.8.2 cp rename

You can also use the **cp** command to give the copied file a different name. In this screenshot we copy **linuxfun.pdf** and rename the copy to **linux.pdf**.

```
paul@debian10:~$ cp linuxfun.pdf linux.pdf
paul@debian10:~$ ls
all.txt.gz  data       linuxfun.pdf  music              wolf.png
backup      dates.txt  linux.pdf     studentfiles.html
paul@debian10:~$
```

6.8.3 cp -i

The **cp** command has a **-i** option to prevent accidentally overwriting existing files. In the example below we try to copy a file twice to the same destination directory. Thanks to the **-i** option this fails the second time.

```
paul@debian10:~$ ls data/
pictures
paul@debian10:~$ cp -i wolf.png data/pictures/
paul@debian10:~$ cp -i wolf.png data/pictures/
cp: overwrite data/pictures/wolf.png? n
paul@debian10:~$
```

6.8.4 cp -p

The **cp** command also has a **-p** option to preserve the **time stamp** on files copied, as can be seen in this screenshot.

```
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 paul paul 1118 Jul 27 13:12 dates.txt
paul@debian10:~$ cp dates.txt dates2.txt
paul@debian10:~$ cp -p dates.txt dates3.txt
paul@debian10:~$ ls -l dates2.txt dates3.txt
-rw-r--r-- 1 paul paul 1118 Aug 14 01:55 dates2.txt
-rw-r--r-- 1 paul paul 1118 Jul 27 13:12 dates3.txt
paul@debian10:~$
```

6.9 Renaming files with mv

The easy way to rename a file is to use the **mv** command. The example below renames the **studentfiles.html** file to **index.htm**.

```
paul@debian10:~$ mv studentfiles.html index.htm
paul@debian10:~$
```

The **mv** command has a **-i** option to prevent accidental overwriting of an existing file.

```
paul@debian10:~$ mv -i backup/studentfiles.html index.htm
mv: overwrite index.htm? n
paul@debian10:~$
```

You can also use **mv** to move files from one directory to another. The first line in this screenshots moves a file from **~/data/pictures** to **~/music**, while the second line moves a file into the **current directory** (represented as a dot).

```
paul@debian10:~$ mv data/pictures/wolf.png music/
paul@debian10:~$ mv backup/studentfiles.html .
paul@debian10:~$
```

Tip
The **"."** notation for the *current directory* can be used in many commands, for example with **cp** and **mv**.

6.10 Renaming files with rename

The **mv** command is easy to use when you want to rename one or only a couple of files. To rename many (millions if you like) files in one command you will need the **rename** command (or the **find** command). The syntax of the **rename** command uses **regular expressions** which we will discuss later in this book.

Here is a very simple example that renames all **linux*** files to **Linux***.

```
paul@debian10:~$ ls
all.txt.gz  data       index.htm     linux.pdf  studentfiles.html
backup      dates.txt  linuxfun.pdf  music      wolf.png
paul@debian10:~$ rename 's/^linux/Linux/' *
paul@debian10:~$ ls
all.txt.gz  data       index.htm     Linux.pdf  studentfiles.html
backup      dates.txt  Linuxfun.pdf  music      wolf.png
paul@debian10:~$
```

Warning
The rename command is not installed by default on Debian Linux. Ask an administrator to execute **apt-get install rename** if you want to test this now.

6.11 Hidden files

We mentioned before that there are **hidden files** in the home directory. These **hidden files** start with a dot **"."** . In the example below we create a hidden file named **.hidden** and we verify with **ls** that it is indeed not visible in the list of files.

```
paul@debian10:~$ ls
all.txt.gz  data       index.htm     Linux.pdf  studentfiles.html
backup      dates.txt  Linuxfun.pdf  music      wolf.png
paul@debian10:~$ touch .hidden
paul@debian10:~$ ls
all.txt.gz  data       index.htm     Linux.pdf  studentfiles.html
backup      dates.txt  Linuxfun.pdf  music      wolf.png
paul@debian10:~$
```

To see the **hidden files** we need to give the **-a** option to **ls** (where a is short for **all** files).

```
paul@debian10:~$ ls -a
.           backup         .bashrc    .hidden    Linuxfun.pdf  .profile
..          .bash_history  data       index.htm  Linux.pdf     studentfiles.html
all.txt.gz  .bash_logout   dates.txt  .lesshst   music         wolf.png
paul@debian10:~$
```

Can you find the six hidden files in the output above? Do not count the first two special files named **"."** and **".."**. We will explain those two in the Links chapter.

Note
Note that options like **-l** for long listing and **-a** for all can be combined. Try the following commands "**ls -a -l**" and "**ls -al**".

6.12 Finding files

Just like with directories, you can use the **find** command to search for files. The example below enters the home directory and then searches for all instances of **dates.txt** .

```
paul@debian10:~$ cd
paul@debian10:~$ find -name dates.txt
./backup/dates.txt
./dates.txt
paul@debian10:~$
```

You can force the **find** command to search for only directories or only files using the **-type** option. If we only want directories in our result, then we use **-type d** .

First we create a file named **pictures**, then we see the **find** command displaying just the directory named **pictures** and not the file.

```
paul@debian10:~$ touch pictures
paul@debian10:~$ find -type d -name pictures
./data/pictures
paul@debian10:~$
```

Use **find -type f** to receive files but not directories.

```
paul@debian10:~$ find -type f -name pictures
./pictures
paul@debian10:~$
```

6.13 File extensions

The **file** command is really recognising the file type. It does not use any extension like **.pdf** or **.htm**. The following example shows how extensions have no meaning on the Linux command line, except for human readability.

```
paul@debian10:~$ file wolf.png
wolf.png: PNG image data, 1280 x 960, 8-bit/color RGB, non-interlaced
paul@debian10:~$ cp wolf.png test.txt
paul@debian10:~$ file test.txt
test.txt: PNG image data, 1280 x 960, 8-bit/color RGB, non-interlaced
paul@debian10:~$
```

6.14 Cheat sheet

Table 6.1: Files

command	explanation
ls	List the files in the current directory.
ls -l	Long list of files in the current directory.
ls -a	List also the hidden files in the current directory.
touch foo	Create an empty file named **foo**.
touch .foo	Create a hidden file named **.foo**.
touch -t 197002271130 bar	Create a file named **bar** with a specific time stamp.
rm foo	Remove the **foo** file.
rm -i foo	Ask for permission to remove the **foo** file.
rm -rf foo	Remove **foo** recursively and forced.
wget http://linux-training.be/foo	Download **foo** from http://linux-training.be.
file bar	Identify the type of file for **bar**.
cp foo bar	Copy the **foo** file to **bar**.
cp foo1 foo2 foo3 bar/	Copy multiple files to the **bar** directory.
cp -i foo bar	Copy **foo** to **bar** but ask permission before overwriting a file.
cp -p foo bar	Copy and preserve the time stamp.
mv foo bar	Rename the **foo** file to **bar**.
mv foo1 foo2 foo3 bar/	Move multiple files to the **bar** directory.
rename 's/foo/bar/' *	Rename all files in the current directory replacing the string **foo** with the string **bar**.
find -name foo	Search for the file (or directory) named **foo**.
find -type f -name foo	Search for the file (not directory) named **foo**.
find -type d -name foo	Search for the directory named **foo**.

6.15 Practice

1. List the contents of your home directory.
2. Create an empty file named **empty.txt** .
3. List all the files in your home directory, including the hidden files.
4. Create and then remove the directory **test** .
5. Remove the **empty.txt** file.
6. Remove the **~/backup** directory and all files in it.
7. Download the http://linux-training.be/dates.txt file from the internet into your home directory.
8. Identify the type of file for **dates.txt** .
9. Copy the **dates.txt** file to **data.txt** .
10. Rename the **data.txt** file to **DATA.TXT** .
11. Find all the files (not the directories) named **Linuxfun.pdf** in your home directory.

6.16 Solution

1. List the contents of your home directory.

```
cd
ls
```

or

```
ls &#126;
```

2. Create an empty file named **empty.txt** .

```
touch empty.txt
```

3. List all the files in your home directory, including the hidden files.

```
ls -a
```

4. Create and then remove the directory **test** .

```
mkdir test
rmdir test
```

5. Remove the **empty.txt** file.

```
rm empty.txt
```

6. Remove the **~/backup** directory and all files in it.

```
rm -rf ~/backup
```

7. Download the http://linux-training.be/dates.txt file from the internet into your home directory.

```
cd
wget http://linux-training.be/dates.txt
```

Note
If the file is already there, then the download will get a **.1** extension.

---

8. Identify the type of file for **dates.txt**

```
file dates.txt
```

9. Copy the **dates.txt** file to **data.txt** .

```
cp dates.txt data.txt
```

10. Rename the **data.txt** file to **DATA.TXT** .

```
mv data.txt DATA.TXT
```

11. Find all the files (not the directories) named **Linuxfun.pdf** in your home directory.

```
cd
find -type f -name Linuxfun.pdf
```

Chapter 7

File contents

Note
This chapter uses some of the files that were downloaded in the previous chapter.

7.1 head

The Linux command line is good at dealing with text files. The first command we will look at is called **head**. This command will display the first ten lines of a file.

Warning
First verify with the **file** command that the file is indeed a text file. Running text commands on non-text files can have awkward results.

```
paul@debian10:~$ file dates.txt
dates.txt: ASCII text
paul@debian10:~$ head dates.txt
 AL Albania        1912-11-30      IT Italy          1582-10-04
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
 BE Belgium        1582-12-14      LN Latin          9999-05-31
 BG Bulgaria       1916-03-18      LU Luxembourg     1582-12-14
 CA Canada         1752-09-02      LV Latvia         1918-02-01
 CH Switzerland    1655-02-28      NL Netherlands    1582-12-14
 CN China          1911-12-18      NO Norway         1700-02-18
 CZ Czech Republic 1584-01-06      PL Poland         1582-10-04
 DE Germany        1700-02-18      PT Portugal       1582-10-04
paul@debian10:~$
```

You can change the default of 10 lines to any other number by writing the desired number as an option to the **head** command. The example below displays the first three line of a text file.

```
paul@debian10:~$ file dates.txt
dates.txt: ASCII text
paul@debian10:~$ head -3 dates.txt
 AL Albania        1912-11-30      IT Italy          1582-10-04
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
paul@debian10:~$
```

7.2 tail

The **tail** command is complementary to **head** in that it displays the last ten lines of a text file. And just like with **head**, you can choose another number.

```
paul@debian10:~$ tail -4 dates.txt
 GB United Kingdom 1752-09-02      TR Turkey         1926-12-18
 GR Greece         1924-03-09      US United States  1752-09-02
 HU Hungary        1587-10-21      YU Yugoslavia     1919-03-04
 IS Iceland        1700-11-16
paul@debian10:~$
```

7.3 cat

The **cat** command will display the full text file on the screen. Even for very long text files. This might not seem very useful now, but we will use **cat** a lot in this book.

```
paul@debian10:~$ cat dates.txt
 AL Albania         1912-11-30      IT Italy           1582-10-04
 AT Austria         1583-10-05      JP Japan           1918-12-18
 AU Australia       1752-09-02      LI Lithuania       1918-02-01
 BE Belgium         1582-12-14      LN Latin           9999-05-31
 BG Bulgaria        1916-03-18      LU Luxembourg      1582-12-14
 CA Canada          1752-09-02      LV Latvia          1918-02-01
 CH Switzerland     1655-02-28      NL Netherlands     1582-12-14
 CN China           1911-12-18      NO Norway          1700-02-18
 CZ Czech Republic 1584-01-06      PL Poland          1582-10-04
 DE Germany         1700-02-18      PT Portugal        1582-10-04
 DK Denmark         1700-02-18      RO Romania         1919-03-31
 ES Spain           1582-10-04      RU Russia          1918-01-31
 FI Finland         1753-02-17      SI Slovenia        1919-03-04
 FR France          1582-12-09      SE Sweden          1753-02-17
 GB United Kingdom 1752-09-02      TR Turkey          1926-12-18
 GR Greece          1924-03-09      US United States   1752-09-02
 HU Hungary         1587-10-21      YU Yugoslavia      1919-03-04
 IS Iceland         1700-11-16
paul@debian10:~$
```

Warning
Again, don't use these commands on random files, always verify with the **file** command that the target file is a text file.

7.4 Creating files with cat

The **cat** command can be used to create simple text files. In this example we create a text file that contains four words, one on each line. We will explain how this works in the Redirection chapter.

Note
After typing **four** and the enter key, type **Ctrl d** to send an EOF character to the **cat** command.

```
paul@debian10:~$ cat > count.txt
one
two
three
four
paul@debian10:~$
```

Take a look at the file with **head** and **tail** to discover its contents.

```
paul@debian10:~$ head -3 count.txt
one
two
three
paul@debian10:~$ tail -3 count.txt
two
three
four
paul@debian10:~$
```

7.5 tac

Complementary to **cat** is **tac**. The **tac** command is the reverse of the **cat** command.

```
paul@debian10:~$ cat count.txt
one
two
three
four
paul@debian10:~$ tac count.txt
four
three
two
one
paul@debian10:~$
```

Important
In Linux there are a lot of simple small commands that do only one thing, but they do that one thing very efficiently. Consider these commands as **building blocks** for more complex solutions. More about this in the Filters chapter.

7.6 wc

The **wc** (short for **w**ord **c**ount) command can count characters, words and lines in a (text) file. In the example below we count the number of **l**ines of our humble **count.txt** file and of the **/etc/services** file.

```
paul@debian10:~$ wc -l count.txt
4 count.txt
paul@debian10:~$ file /etc/services
/etc/services: ASCII text
paul@debian10:~$ wc -l /etc/services
578 /etc/services
paul@debian10:~$
```

The example below shows the **-w** option to count the number of words in a file.

```
paul@debian10:~$ wc -w index.htm
149 index.htm
paul@debian10:~$
```

And finally we count the number of characters in the **dates.txt** file using the **-c** option.

```
paul@debian10:~$ wc -l dates.txt
18 dates.txt
paul@debian10:~$ wc -c dates.txt
1118 dates.txt
paul@debian10:~$
```

7.7 grep

As you can see in the previous example, the **/etc/services** file has too many lines to display on the screen. The first tool that we will see to cope with this is the **grep** command. With **grep** you can select lines from a text file.

In this example we select all the lines containing the string **http**.

```
paul@debian10:~$ grep http /etc/services
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names- ↩
    port-numbers.xhtml .
http            80/tcp          www             # WorldWideWeb HTTP
https           443/tcp                         # http protocol over TLS/SSL
http-alt        8080/tcp        webcache        # WWW caching service
http-alt        8080/udp
paul@debian10:~$
```

Note
Notice that with **grep** the target file is the last argument.

7.8 more

Another way to display parts of or the whole file is using the **more** command. With the **more** command you can display text files page by page (using the space key) or even line by line (using the enter key).

```
paul@debian10:~$ more /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-p
ort-numbers.xhtml .
#
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
--More--(4%)
```

Tip
Try using the enter key and the space bar to advance in this file. Press **q** to quit the **more** command.

7.9 echo

Another very simple command is **echo**. The **echo** command will display whatever you give it as arguments. In combination with the **greater than** sign, you can create a simple file with **echo**.

```
paul@debian10:~$ echo hello world
hello world
paul@debian10:~$ echo hello world > hello.txt
paul@debian10:~$ cat hello.txt
hello world
paul@debian10:~$
```

We can expand the **hello.txt** file using two **greater than** signs. See the example below.

```
paul@debian10:~$ cat hello.txt
hello world
paul@debian10:~$ echo hello again >> hello.txt
paul@debian10:~$ cat hello.txt
hello world
hello again
paul@debian10:~$
```

7.10 Cheat sheet

Table 7.1: File contents

command	explanation
head foo	Show the top ten lines of the text file **foo**.
head -5	Show the top 5 lines.
tail	Show the last ten lines.
tail -42	Show the last 42 lines.
cat	Show the complete file.
tac	Reverse of **cat**, shows last line first.
wc foo	Count lines, words and characters in (text) file **foo**.
wc -l	Count lines.
wc -w	Count words.
wc -c	Count characters.
grep foo bar	Show the lines containing **foo** in the (text) file **bar**.
more foo	Show the (text) file **foo** page by page.
echo foo	Display the string **foo**.
cat > foo	Create (or empty) a file named **foo**, end input with **Ctrl-d**.
head -33 foo > bar	Create (or empty) the **bar** file, then put the first 33 lines of **foo** in it.
tail -1201 foo > bar	Create (or empty) the **bar** file, then put the last 1201 lines of **foo** in it.
echo foo > bar	Put the string **foo** in the file named **bar** after clearing **bar**.
echo foo >> bar	Append the string **foo** to the file named **bar**.

7.11 Practice

1. Display the top five lines of the **dates.txt** file.
2. Display the bottom eleven lines of **/etc/services**.
3. Use **cat** to display the **studentfiles.html** file.
4. Use **more** to display the **/etc/services** file, advance in the file with the **enter** key and the **space** bar.
5. Use **cat** to create a file that contains five lines, name the file **five.txt**.
6. Display the **five.txt** file in reverse.
7. Count the number of lines in **five.txt**.
8. Count the number of characters in **five.txt**
9. Use echo to create a file named **abc.txt** that contains "**The quick brown fox jumps over the lazy dog.**"
10. Use echo to expand the file named **abc.txt** with this line "**The answer to life, the universe, and everything is 42.**"

7.12 Solution

1. Display the top five lines of the **dates.txt** file.

```
head -5 dates.txt
```

2. Display the bottom eleven lines of **/etc/services**.

```
tail -11 /etc/services
```

3. Use **cat** to display the **studentfiles.html** file.

```
cat studentfiles.html
```

4. Use **more** to display the **/etc/services** file, advance in the file with the **enter** key and the **space** bar.

```
more /etc/services
```

5. Use **cat** to create a file that contains five lines, name the file **five.txt**.

```
cat > five.txt
one
two
three
four
five
```

   Followed by a **Ctrl d** key.

6. Display the **five.txt** file in reverse.

```
tac five.txt
```

7. Count the number of lines in **five.txt**.

```
wc -l five.txt
```

8. Count the number of characters in **five.txt**

```
wc -c five.txt
```

9. Use echo to create a file named **abc.txt** that contains "**The quick brown fox jumps over the lazy dog.**"

```
echo The quick brown fox jumps over the lazy dog. > abc.txt
```

10. Use echo to expand the file named **abc.txt** with this line "**The answer to life, the universe, and everything is 42.**"

```
echo The answer to life, the universe and everything is 42. >> abc.txt
```

Chapter 8

Compression

8.1 About compression

Files can be compressed in two distinct forms named **lossy compressions** and **lossless compression**. With **lossy compression** it is impossible to recreate the original file, an example of this is a **.mp3** file or a **.jpg** file.

With **lossless compression** it is always possible to recreate the original file (or files). Examples of **lossless compression** are **.gz** files and **.flac** files.

In this chapter we only discuss **lossless compression** of files.

8.2 gzip

One of the most popular tools for compressing files on Linux is **gzip**. In the example below we use the **gzip** command on a copy of the **/etc/services** file.

```
paul@debian10:~$ cp /etc/services .
paul@debian10:~$ ls -l services
-rw-r--r-- 1 paul paul 18774 Jul 28 15:09 services
paul@debian10:~$ gzip services
paul@debian10:~$ ls -l services.gz
-rw-r--r-- 1 paul paul 7307 Jul 28 15:09 services.gz
paul@debian10:~$
```

Using **gzip** on this small text file made it shrink in size from 18774 bytes to 7307 bytes, as shown by the two **ls -l** commands.

Note
Notice that when using **gzip** on a file, the name of the file is appended with **.gz** .

8.3 zmore

You cannot use tools like **cat**, **more**, **head**, **tail** or **grep** on the compressed file (please don't try this). But you can use **zmore** as shown in this example.

```
paul@debian10:~$ file services.gz
services.gz: gzip compressed data, was "services", last modified: Sun Jul 28 13:09:58 2019, ↩
     from Unix, original size 18774
paul@debian10:~$ zmore services.gz
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-p
ort-numbers.xhtml .
#
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
```

```
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
--More--
```

8.4 zcat

You can also use **zcat** to have the whole file scrolling by on the display. Luckily we can combine the **zcat** command with commands like **head** and **tail**. See the example below for how to use **tail** with **zcat**.

```
paul@debian10:~$ zcat services.gz | tail -4
tfido           60177/tcp                       # fidonet EMSI over telnet
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
paul@debian10:~$
```

We will explain the **"|"** symbol in the Redirection chapter. For now understand that the screenshot above shows the last four lines of the compressed **services** file.

8.5 zgrep

And you can use the **zgrep** command to search for strings in the compressed **services.gz** file, as is shown in this example.

```
paul@debian10:~$ zgrep http services.gz
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names- ↩
    port-numbers.xhtml .
http            80/tcp          www             # WorldWideWeb HTTP
https           443/tcp                         # http protocol over TLS/SSL
http-alt        8080/tcp        webcache        # WWW caching service
http-alt        8080/udp
paul@debian10:~$
```

8.6 gzip -l

The **gzip -l** option (that is the lowercase letter L from **l**ist) allows you to view the uncompressed size of a **.gz** file before uncompressing it. In this example we use **gzip -l** on the **all.txt.gz** file that was downloaded some chapters ago.

```
paul@debian10:~$ gzip -l all.txt.gz
         compressed        uncompressed  ratio uncompressed_name
             332067             2903154  88.6% all.txt
paul@debian10:~$
```

Note
Compressing multiple files in one file (with **tar** for example) is discussed in the backup chapter.

8.7 gunzip

Compressed files can be uncompressed again with **gunzip**. The screenshot below shows how to uncompress the **services.gz** file back to its original state. The **.gz** file extension is removed by **gunzip**.

```
paul@debian10:~$ ls -l services.gz
-rw-r--r-- 1 paul paul 7307 Jul 28 15:09 services.gz
paul@debian10:~$ gunzip services.gz
paul@debian10:~$ ls -l services
-rw-r--r-- 1 paul paul 18774 Jul 28 15:09 services
paul@debian10:~$
```

8.8 bzip2 - bunzip2 - bzcat - bzmore - bzgrep

Another popular compressions tool on Linux is **bzip2**. It is generally slower than **gzip** but compresses better. In the screenshot below we use **bzip2** and its associated commands **bunzip2**, **bzcat**, **bzmore**, and **bzgrep**.

```
paul@debian10:~$ bzip2 services
paul@debian10:~$ ls -l services.bz2
-rw-r--r-- 1 paul paul 6966 Jul 28 15:09 services.bz2
paul@debian10:~$ bzcat services.bz2 | tail -4
tfido           60177/tcp                       # fidonet EMSI over telnet
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
paul@debian10:~$ bzgrep http services.bz2
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names- ↩
    port-numbers.xhtml .
http            80/tcp          www             # WorldWideWeb HTTP
https           443/tcp                         # http protocol over TLS/SSL
http-alt        8080/tcp        webcache        # WWW caching service
http-alt        8080/udp
paul@debian10:~$ bunzip2 services.bz2
paul@debian10:~$ ls -l services
-rw-r--r-- 1 paul paul 18774 Jul 28 15:09 services
paul@debian10:~$
```

8.9 Cheat sheet

Table 8.1: Compression

command	**explanation**
gzip foo	Compress the **foo** file with gzip.
gzip -l foo.gz	Display information about the compressed **foo.gz** file.
zmore foo.gz	Display the compressed **foo.gz** file page by page.
zcat foo.gz	Display the complete **foo.gz** file.
zgrep foo bar.gz	Show the lines containing the string **foo** from the compressed **bar.gz** file.
gunzip foo.gz	Uncompress the **foo.gz** file.
bzip2 foo	Compress the **foo** file with bzip2.
bzmore foo.bz2	Display the compressed **foo.bz2** file page by page.
bzcat foo.bz2	Display the complete **foo.bz2** file.
bzgrep foo bar.bz2	Show the lines containing the string **foo** from the compressed **bar.bz2** file.
bunzip2 foo.bz2	Uncompress the **foo.bz2** file.

8.10 Practice

1. Copy the **/usr/share/dict/words** file to your home directory.
2. Verify the size of the **~/words** file.
3. Use **gzip** to compress the **~/words** file.
4. Again verify the size of the file.
5. Is the word **evening** present in this compressed file?
6. Display the last ten lines of **~/words.gz** .
7. Uncompress the **~/words.gz** file.
8. Repeat all of the above but use **bzip2** instead of **gzip**.

8.11 Solution

1. Copy the **/usr/share/dict/words** file to your home directory.

```
cp /usr/share/dict/words ~
```

2. Verify the size of the **~/words** file.

```
ls -l ~/words
```

3. Use **gzip** to compress the **~/words** file.

```
gzip ~/words
```

4. Again verify the size of the file.

```
ls -l ~/words.gz
```

5. Is the word **evening** present in this compressed file?

```
zgrep evening ~/words.gz
```

6. Display the last ten lines of **~/words.gz** .

```
zcat ~/words.gz | tail
```

7. Uncompress the **~/words.gz** file.

```
gunzip ~/words.gz
```

8. Repeat all of the above but use **bzip2** tools instead of **gzip**.

```
ls -l ~/words
bzip2 ~/words
ls -l ~/words.bz2
bzgrep evening ~/words.bz2
bzcat ~/words.bz2 | tail
bunzip2 ~/words.bz2
```

Chapter 9

Manual pages

9.1 man

Most Linux commands come with an excellent manual in the form of a **man page**. This **man page** can usually be accessed by typing **man** followed by the command. The example below enters the **man page** of the **tail** command.

```
paul@debian10:~$ man tail
TAIL(1)                          User Commands                          TAIL(1)

NAME
       tail - output the last part of files

SYNOPSIS
       tail [OPTION]... [FILE]...

DESCRIPTION
output truncated
```

When inside the man page, you have several options that are the same as when using the **more** command.

Table 9.1: man page keys

key	action
enter	continue one line
space bar	next page
q	quit

On Debian 10 you can also use the arrow keys to browse up and down in the **man page**.

9.2 Searching the man page

You can use the slash key **"/"** to search for strings in the **man page**. The **/** will appear at the bottom of the screen. The search is case insensitive by default. You can jump to the next occurrence using the **n** key and to the previous occurrence using the **p** key. And remember type **q** to quit the **man page**.

Table 9.2: man page keys

key	action
/string	search for string
n	jump to next occurrence of string
p	jump to previous occurrence of string

9.3 man man

There is a **man page** for the **man** command itself. In this (rather long) **man page** you can read about different sections (see the screenshot below).

```
       The table below shows the section numbers of the manual followed by the types of
       pages they contain.
```

```
       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
       5   File formats and conventions eg /etc/passwd
       6   Games
       7   Miscellaneous  (including  macro  packages  and  conventions),  e.g. man(7),
           groff(7)
       8   System administration commands (usually only for root)
       9   Kernel routines [Non standard]
```

The vast majority of **man page** readings for a system administrator will involve either **section 1** on commands or **section 5** on files, and the correct section will automatically be chosen.

But there are cases where there is a command and a file with the same name. This is the case for example with the **passwd** command and the **passwd** file. To select the correct **man page** you can type the section as an argument.

The following command opens the **man page** for the **passwd** *command*.

```
paul@debian10:~$ man 1 passwd
PASSWD(1)                          User Commands                          PASSWD(1)

NAME
       passwd - change user password

SYNOPSIS
       passwd [options] [LOGIN]

DESCRIPTION
output truncated
```

And here we show how to open the **man page** for the **passwd** *file*.

```
paul@debian10:~$ man 5 passwd
PASSWD(5)                   File Formats and Conversions                   PASSWD(5)

NAME
       passwd - the password file

DESCRIPTION
output truncated
```

9.4 apropos

It can happen that you are looking for a **man page** or even a **command** but you forgot the exact name. Then the **apropos** command can help you. In the example below we use **apropos** to discover commands related to **USB** devices.

```
paul@debian10:~$ apropos usb
lsusb (8)            - list USB devices
usb-devices (1)      - print USB device details
usbhid-dump (8)      - dump USB HID device report descriptors and streams
paul@debian10:~$
```

The **apropos** command will do a case insensitive search in the name and the short description of all **man pages**.

Note
Typing **apropos** is identical to typing **man -k**.

9.5 man -wK

If you want to search the content of all man pages, then you can use **man -wK**. Because this can take a long time, it is advised to limit the search to one section. The screenshot below demonstrates searching section 1 for the string **SIGHUP**.

```
paul@debian10:~$ man -wK -s 1 SIGHUP
/usr/share/man/man1/bash.1.gz
/usr/share/man/man1/systemd-run.1.gz
/usr/share/man/man1/systemd.1.gz
/usr/share/man/man1/nano.1.gz
/usr/share/man/man1/nano.1.gz
/usr/share/man/man1/nano.1.gz
/usr/share/man/man1/tar.1.gz
/usr/share/man/man1/dbus-daemon.1.gz
paul@debian10:~$
```

9.6 whatis

You can use the **whatis** command to see the short description of **files** or **commands**. The example shows the short description of the **head** command and of the **services** file.

```
paul@debian10:~$ whatis head
head (1)             - output the first part of files
paul@debian10:~$ whatis services
services (5)         - Internet network services list
paul@debian10:~$
```

9.7 whereis

You can find the location of **commands** and **files** and their respective **man pages** in the file system with the **whereis** command. The example does this for the **tail** command and for the **lsusb** command.

Note
Notice that the **man page** is stored as a gzipped **.gz** file.

```
paul@debian10:~$ whereis tail
tail: /usr/bin/tail /usr/share/man/man1/tail.1.gz
paul@debian10:~$ whereis lsusb
lsusb: /usr/bin/lsusb /usr/share/man/man8/lsusb.8.gz
paul@debian10:~$
```

9.8 mandb

It should not happen, but in case you have freshly installed programs that you cannot find with the **man** commands, then it may help to run the **mandb** command. This command will regenerate all man page index caches.

```
paul@debian10:~$ mandb
0 man subdirectories contained newer manual pages.
0 manual pages were added.
0 stray cats were added.
0 old database entries were purged.
paul@debian10:~$
```

Unfortunately there is no **man page** for **woman**.

```
paul@debian10:~$ man woman
No manual entry for woman
paul@debian10:~$
```

9.9 Cheat sheet

Table 9.3: man pages

command	**explanation**
man foo	Display the manual page of **foo**.
man man	Display the manual of the manual.
man 1 foo	Display the section 1 manual page of **foo**.
man 5 foo	Display the section 5 manual page of **foo**.
apropos foo	List manual pages containing the string **foo**.
man -k foo	List manual pages containing the string **foo**.
whatis foo	Shows the short description of **foo**.
whereis foo	Shows the location of **foo** and its manual.
mandb	Regenerate the man index caches.
	When inside a man page
enter key	Go to the next line
space bar	Go to the next page
q	quit the manual
/foo	Search for the string foo
n	jump to next occurrence of string
p	jump to previous occurrence of string

9.10 Practice

1. Open the man page for **cp** and search for the option to recursively copy directories.
2. Open the man page for the **passwd** file.
3. Open the man page of **bash** and search for **noclobber**
4. List all the man pages that have something to do with **passwords**.
5. List the location of the **cat** command and its manual page.
6. Display the short description of the **grep** command.

9.11 Solution

1. Open the man page for **cp** and search for the option to recursively copy directories.

```
man cp
```

   The option you were looking for is **-r**.

2. Open the man page for the **passwd** file.

```
man 5 passwd
```

3. Open the man page of **bash** and search for **noclobber**

```
man bash
```

   followed by

```
/noclobber
n n n
```

> **Note**
> We will use the **noclobber** option later in this book.

4. List all the man pages that have something to do with **passwords**.

```
apropos password
```

5. List the location of the **cat** command and its manual page.

```
whereis cat
```

6. Display the short description of the **grep** command.

```
whatis grep
```

Chapter 10

File system tree

10.1 Filesystem Hierarchy Standard

All files on a Linux system should have a well defined directory to reside in. Debian Linux conforms to many parts of the Filesystem Hierarchy Standard (FHS) which can be found at http://www.pathname.com/fhs/ .

There is an alphabetical list of directories (describing their purpose) in the **man hier** manual (*hier* is short for hierarchy). In this chapter we discuss the most common directories on a Debian Linux system.

10.2 Data directories

10.2.1 /home

The first directory to explore is the **/home** directory. In this directory there can be a sub-directory for each user, usually identical in name to the user name. Each user can put personal files in his or her **home directory**.

In the screenshot below we list the contents of the **/home** directory on a Debian system with five users.

Note
Chances are that on your system there is only one user home directory, namely yours.

```
paul@debian10:~$ ls -l /home
total 20
drwxr-xr-x 2 annik annik 4096 Jul 29 14:31 annik
drwxr-xr-x 2 david david 4096 Jul 29 14:31 david
drwxr-xr-x 2 geert geert 4096 Jul 29 14:31 geert
drwxr-xr-x 2 linda linda 4096 Jul 29 14:31 linda
drwxr-xr-x 5 paul  paul  4096 Jul 29 13:52 paul
paul@debian10:~$
```

Caution
You cannot (and should not) store files in /**home**/ directly. Always store personal files in /**home**/**yourusername**.

10.2.2 /root

Each Linux system has a unique user named **root**. This **root** user can manage the system, install software, add users and much more. Since this is a special user, he or she also has a special **home directory** named **/root**.

As a normal user, you do not have access to the **/root** directory.

```
paul@debian10:~$ ls /root
ls: cannot open directory /root: Permission denied
paul@debian10:~$
```

Note
The real reason that /**root** is not in /**home** is that /**home** can be mounted from the network and the network is not available when troubleshooting a system in *single user mode*.

10.2.3 /srv

File servers can share many files (and directories) on the network. Whenever a directory is shared, it should reside in the **/srv** directory. You can read the **/srv** directory as *served by your system*.

The screenshot below shows an empty **/srv** directory, so I assume this computer is not sharing directories on the (local) network.

```
paul@debian10:~$ ls -l /srv/
total 0
paul@debian10:~$
```

Warning
I have often encountered organisations that use /**mnt** for this purpose. Don't do that, it is wrong.

10.2.4 /mnt

The **/mnt** directory should be empty. It serves as a **temporary** mount point for filesystems.

```
paul@debian10:~$ ls /mnt
paul@debian10:~$
```

Note
We will discuss **mounting** of filesystems in the storage section of this book.

10.2.5 /media

The **/media** directory is used to **mount** removable media such as compact discs or USB sticks. On a graphical interface of an end user Linux computer this will happen automatically so the user can access the media. On a Linux server this directory can be empty.

This Debian 10 server has a CD-ROM drive, so the directory **/media/cdrom** is present.

```
paul@debian10:~$ ls -l /media/
total 4
lrwxrwxrwx 1 root root    6 Jul 24 18:07 cdrom -> cdrom0
drwxr-xr-x 2 root root 4096 Jul 24 18:07 cdrom0
paul@debian10:~$
```

Tip
The cdrom file is a **symbolic link**, we will discuss **links** in the Links chapter.

10.2.6 /tmp

Both users and applications can store **temporary** data in **/tmp**. A regular Debian Linux will empty this directory each time it is booted.

10.3 Binary directories

10.3.1 /bin

According to the FHS **/bin** holds commands for normal users and for the administrator. In Debian 10 the **/bin** directory is a link to **/usr/bin** (We will discuss links in the Links chapter).

```
paul@debian10:~$ ls -ld /bin
lrwxrwxrwx 1 root root 7 Jul 24 18:07 /bin -> usr/bin
paul@debian10:~$
```

The **/bin** directory contains many simple commands like **head**, **tail**, **tac**, **cat**, **ls**, and **pwd** (and many more).

```
paul@debian10:~$ cd /bin
paul@debian10:/bin$ ls -l head tail tac cat ls pwd
-rwxr-xr-x 1 root root  43744 Feb 28 16:30 cat
-rwxr-xr-x 1 root root  47840 Feb 28 16:30 head
-rwxr-xr-x 1 root root 138856 Feb 28 16:30 ls
-rwxr-xr-x 1 root root  39616 Feb 28 16:30 pwd
-rwxr-xr-x 1 root root  43744 Feb 28 16:30 tac
-rwxr-xr-x 1 root root  72608 Feb 28 16:30 tail
paul@debian10:/bin$
```

10.3.2 /sbin

The **/sbin** directory contains utilities for the **root** user to manage the system. On Debian 10 this directory is a link to **/usr/sbin**.

```
paul@debian10:~$ ls -ld /sbin
lrwxrwxrwx 1 root root 8 Jul 24 18:07 /sbin -> usr/sbin
paul@debian10:~$
```

We will use many of the commands in **/sbin** in this book when we reach the system administration chapters.

10.3.3 /lib

The **/lib** directory (again a symbolic link to **/usr/lib**) contains libraries. Libraries are executables that do not run by themselves, but instead are called by other executables.

The screenshot below shows information about the **ssh-keysign** library file.

```
paul@debian10:~$ file /lib/openssh/ssh-keysign
/lib/openssh/ssh-keysign: setuid ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),  ↩
    dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,  ↩
    BuildID[sha1]=d3eb93acd1467d5e04b65d5bcc35ceb5b0d1ff90, stripped
paul@debian10:~$
```

10.3.4 /opt

The **/opt** directory is there for *optional* packages. This means packages that are not part of Debian (they are not in the Debian repositories), but can be installed on Debian. Examples are **/opt/ora** for Oracle software and **/opt/ibm** for IBM packages.

By default on Debian **/opt** is empty.

```
paul@debian10:~$ ls -l /opt
total 0
paul@debian10:~$
```

10.4 Configuration directory

10.4.1 /etc

The **/etc** directory contains configuration files, in fact it contains almost all configuration files for the local system. There are many sub-directories in **/etc** to group configuration files by application or by protocol.

Below is a small view of some of the configuration files in **/etc**.

```
paul@debian10:~$ ls /etc/*.conf
/etc/adduser.conf            /etc/kernel-img.conf          /etc/reportbug.conf
/etc/ca-certificates.conf    /etc/ld.so.conf               /etc/resolv.conf
/etc/debconf.conf            /etc/libaudit.conf            /etc/rsyslog.conf
/etc/deluser.conf            /etc/logrotate.conf           /etc/sysctl.conf
/etc/discover-modprobe.conf  /etc/mke2fs.conf              /etc/ucf.conf
/etc/gai.conf                /etc/nsswitch.conf            /etc/xattr.conf
/etc/hdparm.conf             /etc/pam.conf
/etc/host.conf               /etc/popularity-contest.conf
paul@debian10:~$
```

10.5 Boot directory

10.5.1 /boot

The **/boot** directory contains the necessary files to **boot** the system from **power on** to a running **Linux kernel**. The **Linux kernel** file is called **vmlinuz** followed by a version number. We will discuss these files in the Booting Linux chapter.

```
paul@debian10:~$ ls -l /boot
total 33576
-rw-r--r-- 1 root root   206212 Jul 19 10:45 config-4.19.0-5-amd64
drwxr-xr-x 5 root root     4096 Jul 24 18:16 grub
-rw-r--r-- 1 root root 25563597 Jul 24 18:12 initrd.img-4.19.0-5-amd64
-rw-r--r-- 1 root root  3371025 Jul 19 10:45 System.map-4.19.0-5-amd64
-rw-r--r-- 1 root root  5225712 Jul 19 10:45 vmlinuz-4.19.0-5-amd64
paul@debian10:~$
```

10.6 Directories in RAM memory

10.6.1 /dev

The **/dev** directory is populated with real and pseudo **devices** when the computer is starting. You can for example find information about attached hard disk devices in **/dev/disk/**.

```
paul@debian10:~$ ls /dev/disk
by-id  by-partuuid  by-path  by-uuid
paul@debian10:~$
```

10.6.1.1 /dev/random

One useful *pseudo* device in **/dev** is **/dev/random**. This file is a random number generator that will generate random numbers if there is enough entropy. This file can be used to generate secure private keys.

10.6.1.2 /dev/urandom

You can also use **/dev/urandom** which will continuously generate random numbers that can be used for games and other non-security related actions. It will spawn random numbers even if there is not enough entropy on the system.

10.6.1.3 /dev/zero

The **/dev/zero** *pseudo* device will generate an endless amount of zeroes when used. This is useful for example when overwriting a hard disk before you sell it on eBay.

10.6.1.4 /dev/null

The **/dev/null** *pseudo* device is often used as a target file for data that has to be discarded immediately.

10.6.2 /proc

The **/proc** directory provides a means to talk with the kernel and contains a directory for every **process** on the system. We will discuss processes in the Processes chapters.

```
paul@debian10:~$ ls /proc
1     135  21   322  41   6     buddyinfo    interrupts   misc          sysrq-trigger
10    14   22   33   42   65    bus          iomem        modules       sysvipc
1065  15   24   34   43   66    cgroups      ioports      mounts        thread-self
1068  16   241  35   431  76    cmdline      irq          mtrr          timer_list
1069  17   25   36   432  8     consoles     kallsyms     net           tty
11    171  258  365  44   9     cpuinfo      kcore        pagetypeinfo  uptime
114   176  26   369  46   967   crypto       keys         partitions    version
115   177  27   37   481  969   devices      key-users    sched_debug   vmallocinfo
117   19   289  374  482  974   diskstats    kmsg         schedstat     vmstat
118   2    29   376  486  977   dma          kpagecgroup  self          zoneinfo
119   20   3    378  492  978   driver       kpagecount   slabinfo
12    203  30   38   493  982   execdomains  kpageflags   softirqs
129   205  31   39   500  987   fb           loadavg      stat
13    206  32   4    546  988   filesystems  locks        swaps
134   207  321  40   58   acpi  fs           meminfo      sys
paul@debian10:~$
```

Using **/proc** you can query for a lot of information about the computer like its **interrupts**, its **uptime** or **cpuinfo**.

```
paul@debian10:~$ head /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 70
model name      : Intel(R) Core(TM) i7-4770HQ CPU @ 2.20GHz
stepping        : 1
cpu MHz         : 2194.918
cache size      : 6144 KB
physical id     : 0
siblings        : 4
paul@debian10:~$
```

10.6.3 /sys

The **/sys** directory has similar information about the kernel like **/proc** but is structured differently. When hot plugging devices then the necessary files are automatically created in **/sys**. This directory is not part of FHS, but is always present in RAM memory on a Debian Linux system.

```
paul@debian10:~$ ls -l /sys
total 0
drwxr-xr-x   2 root root 0 Jul 29 16:27 block
drwxr-xr-x  29 root root 0 Jul 29 16:27 bus
drwxr-xr-x  46 root root 0 Jul 29 16:27 class
drwxr-xr-x   4 root root 0 Jul 29 16:27 dev
drwxr-xr-x  15 root root 0 Jul 29 16:27 devices
drwxr-xr-x   5 root root 0 Jul 29 16:27 firmware
drwxr-xr-x   6 root root 0 Jul 29 16:27 fs
drwxr-xr-x   2 root root 0 Jul 29 21:02 hypervisor
drwxr-xr-x  12 root root 0 Jul 29 16:27 kernel
drwxr-xr-x 101 root root 0 Jul 29 16:27 module
drwxr-xr-x   2 root root 0 Jul 29 16:27 power
paul@debian10:~$
```

10.6.4 /usr

The **/usr** directory, short for Unix System Resources (a backronym) stores read-only, shareable data. It is possible to mount **/usr** from the network and to share it across multiple Linux computers.

```
paul@debian10:~$ ls /usr
bin  games  include  lib  lib32  lib64  libx32  local  sbin  share  src
paul@debian10:~$
```

10.6.4.1 /usr/bin and /usr/sbin

We have already seen that Debian by default will create a link between **/bin** and **/usr/bin** and between **/sbin** and **/usr/sbin**. So naturally these directories contain the same commands.

```
paul@debian10:~$ ls -ld /bin
lrwxrwxrwx 1 root root 7 Jul 24 18:07 /bin -> usr/bin
paul@debian10:~$ ls -ld /sbin
lrwxrwxrwx 1 root root 8 Jul 24 18:07 /sbin -> usr/sbin
paul@debian10:~$
```

10.6.4.2 /usr/include and /usr/src

Source code should be placed in the **/usr/src** directory and include files in the **/usr/include** directory.

Below a screenshot of some header files in **/usr/include**.

```
paul@debian10:~$ find /usr/include/
/usr/include/
/usr/include/reglib
/usr/include/reglib/reglib.h
/usr/include/reglib/nl80211.h
/usr/include/reglib/regdb.h
/usr/include/iproute2
/usr/include/iproute2/bpf_elf.h
/usr/include/clif.h
paul@debian10:~$
```

The **/usr/src** directory can be empty until you install source code from the repository.

```
paul@debian10:~$ ls /usr/src/
paul@debian10:~$
```

10.6.4.3 /usr/local

The **/usr/local** directory can be used by software that is installed locally (often after local compilation) by the **root** user.

You may recognise some of the directories in **/usr/local** and they can be empty on a fresh install of Debian 10.

```
paul@debian10:~$ ls /usr/local/
bin  etc  games  include  lib  man  sbin  share  src
paul@debian10:~$
```

10.6.4.4 /usr/share

The **/usr/share** directory contains shareable architecture independent files. For example the **man pages** are in **/usr/share**.

```
paul@debian10:~$ ls /usr/share/man
cs  de  fi  fr.ISO8859-1  hu  it  ko    man2  man4  man6  man8  pl  pt_BR  sl  sv  zh_CN
da  es  fr  fr.UTF-8      id  ja  man1  man3  man5  man7  nl    pt  ru     sr  tr  zh_TW
paul@debian10:~$
```

10.6.5 /var

The **/var** directory contains files that are unpredictable in size. In the past this used to be a separate partition on the hard disk, to prevent filling the root file system to 100 percent.

```
paul@debian10:~$ ls /var
backups  cache  lib  local  lock  log  mail  opt  run  spool  tmp
paul@debian10:~$
```

The **/var** directory contains several notable sub-directories.

10.6.5.1 /var/cache

Applications that use a cache should store this in **/var/cache/**. For example the cache of a DNS server or the cache of the **apt** system (see Software Management).

```
paul@debian10:~$ ls /var/cache/
apparmor  apt  debconf  dictionaries-common  ldconfig  man  private
paul@debian10:~$
```

10.6.5.2 /var/log

Log files should be stored in **/var/log**, either in a **.log** file or in a sub-directory. We will see later how to access these log files.

```
paul@debian10:~$ ls /var/log
alternatives.log  daemon.log  installer  popularity-contest.0     private
apt               debug       kern.log   popularity-contest.1.gz  syslog
auth.log          dpkg.log    lastlog    popularity-contest.2.gz  user.log
btmp              faillog     messages   popularity-contest.new   wtmp
paul@debian10:~$
```

10.6.5.3 /var/run

The **/var/run** directory is a symbolic link to **/run**.

```
paul@debian10:~$ ls -ld /var/run
lrwxrwxrwx 1 root root 4 Jul 24 18:07 /var/run -> /run
paul@debian10:~$
```

This directory contains run-state information about programs that are running (since the system was booted). For example **.pid** files should be stored here.

```
paul@debian10:~$ ls /var/run
agetty.reload  dbus                  lock          network          sshd.pid    user
console-setup  dhclient.enp0s3.pid   log           sendsigs.omit.d  systemd     utmp
crond.pid      initctl               motd.dynamic  shm              tmpfiles.d
crond.reboot   initramfs             mount         sshd             udev
paul@debian10:~$
```

10.6.5.4 /var/spool

The **/var/spool** directory contains data that is waiting to be processed, like for example a print queue or an outgoing mail queue.

```
paul@debian10:~$ ls /var/spool/
anacron  cron  mail  rsyslog
paul@debian10:~$
```

10.7 Cheat sheet

Table 10.1: Filesystem hierarchy

location	explanation
/home	location for home directories
/home/paul	home directory for **paul**
/root	home directory for the **root** user
/srv	location for directories that are **served** on the network
/mnt	temporary mount point
/media	location for removable media
/media/cdrom	mount point for CD-ROM
/tmp	location for temporary files
/bin	location for commands (executables)
/sbin	location for commands to manage the system
/lib	location for libraries (shared objects)
/opt	location for **optional** software from outside the repository
/etc	contains configuration
/boot	contains files necessary to boot the system
/dev	contains real and pseudo devices
/dev/random	secure random number generator
/dev/urandom	quick random number generator
/dev/zero	endless source of zeroes
/dev/null	black hole (anything copied to it is discarded)
/proc	contains kernel information (interfaces)
/proc/cpuinfo	contains CPU information
/proc/meminfo	contains memory information
/sys	system information (since kernel 2.6)
/usr	read only shared data
/usr/bin	/bin links to this location
/usr/sbin	/sbin links to this location
/usr/games	location for games
/usr/lib	/lib links to this location
/usr/local	default location for locally compiled files
/usr/share	data that can be shared across architectures
/usr/share/dict	dictionaries location
/usr/share/man	location for man pages
/usr/src	location for system source files
/var	directory for files variable in size
/var/cache	location for cache files
/var/log	location for log files
/var/run	location for run-state information
/var/spool	location for spool directories (e.g. mail or print queue)

10.8 Practice

1. List the contents of the ssh configuration directory.
2. List the contents of your home directory.
3. Try to enter the home directory of the root user.
4. List the data directories that are shared on the network by this computer.
5. Create a temporary test directory.
6. List the commands to manage the Linux system.
7. List the man-db libraries.
8. List third party package providers.
9. List the size of the kernel in human readable format.

10.9 Solution

1. List the contents of the ssh configuration directory.

```
ls /etc/ssh
```

2. List the contents of your home directory.

```
ls ~
```

3. Try to enter the home directory of the root user.

```
cd /root
```

4. List the data directories that are shared on the network by this computer.

```
ls /srv
```

5. Create a temporary test directory.

```
mkdir /tmp/test
```

6. List the commands to manage the Linux system.

```
ls /sbin
ls /usr/sbin
```

7. List the man-db libraries.

```
ls /lib/man-db
```

8. List third party package providers.

```
ls /opt
```

9. List the size of the kernel in human readable format.

```
ls -lh /boot/vmlinuz*
```

Part II

Introduction to the shell

Chapter 11

Shell commands

11.1 /bin/bash

The command prompt that you are typing in, is called a **shell**. If you use **ssh** or **putty** to connect to a remote computer, then you are using a **remote shell**. In both (local and remote) cases you are probably using the **bash** shell since this is the default on Debian 10.

You can verify the shell by typing **echo $SHELL** in the command prompt.

```
paul@debian10:~$ echo $SHELL
/bin/bash
paul@debian10:~$
```

This **bash** shell is doing a lot of work to make your live easier, so it is important to explain its workings.

11.2 which

The **which** command tells you where the shell is searching for a command when you type it. This allows you to type **cat** instead of **/usr/bin/cat**.

```
paul@debian10:~$ which cat head tail find
/usr/bin/cat
/usr/bin/head
/usr/bin/tail
/usr/bin/find
paul@debian10:~$
```

The **which** command is looking in the **$PATH** for commands. This **$PATH** is a list of directories, separated by a colon **":"** .

```
paul@debian10:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
paul@debian10:~$
```

But the **which** command is incomplete since there is no answer for commands like **cd** and **popd**.

```
paul@debian10:~$ which cd popd
paul@debian10:~$
```

11.3 type

The **type** command is less known, but more complete than **which**. It will also look in the **$PATH** directories, but before that it will look at shell **built-in** commands.

```
paul@debian10:~$ type cat head cd popd
cat is /usr/bin/cat
head is /usr/bin/head
cd is a shell builtin
popd is a shell builtin
paul@debian10:~$
```

Note
Notice that **cd** and **popd** are built-in to the bash shell.

Some commands exist as both a **built-in** command and as en external command. For example the **echo** command shown in this screenshot.

```
paul@debian10:~$ which echo
/usr/bin/echo
paul@debian10:~$ type echo
echo is a shell builtin
paul@debian10:~$
```

So the question is, which will be executed when you type **echo**? The answer is that the **built-in** command gets priority to the external command. So when you use **echo** it is the **built-in** echo of the **bash shell** that is used. If you want to use the external **echo** command then you have to type **/bin/echo** or **/usr/bin/echo**.

Commands that you have recently used are still in memory. The **type** command will show these as **hashed** as can be seen in this screenshot.

```
paul@debian10:~$ type which
which is hashed (/usr/bin/which)
paul@debian10:~$
```

11.4 alias

The bash shell allows you to create **aliases** for commands. This allows you for example to abbreviate commands.

```
paul@debian10:~$ t cat head find
-bash: t: command not found
paul@debian10:~$ alias t=type
paul@debian10:~$ t cat head find
cat is /usr/bin/cat
head is /usr/bin/head
find is hashed (/usr/bin/find)
paul@debian10:~$
```

But what happens when an **alias** is identical to a **built-in** command? The answer is that the **alias** gets priority.

```
paul@debian10:~$ alias cd=type
paul@debian10:~$ cd cat head find
cat is /usr/bin/cat
head is /usr/bin/head
find is hashed (/usr/bin/find)
paul@debian10:~$
```

To summarise, the **alias** gets priority to the **built-in** which gets priority to **external** commands.

One advantage that is often used, is that with **aliases** you can define default options for commands. For example instead of always typing **mv -i**, you can define an **alias** for this.

```
paul@debian10:~$ alias mv='mv -i'
paul@debian10:~$ mv studentfiles.html index.htm
mv: overwrite 'index.htm'? n
paul@debian10:~$
```

To see a list of all **aliases** just type **alias** at the command prompt.

```
paul@debian10:~$ alias
alias cd='type'
alias ls='ls --color=auto'
alias mv='mv -i'
alias t='type'
paul@debian10:~$
```

11.5 unalias

When you no longer need an **alias**, then you can remove it with **unalias**. See the example below where we remove two aliases.

```
paul@debian10:~$ unalias cd
paul@debian10:~$ unalias t
paul@debian10:~$ alias
alias ls='ls --color=auto'
alias mv='mv -i'
paul@debian10:~$
```

11.6 exit code

A command typed in the shell will (hopefully) end and then it returns an **exit code**. You can query this **exit code** with the **echo $?** command.

```
paul@debian10:~$ cp Linuxfun.pdf Linux.pdf
paul@debian10:~$ echo $?
0
paul@debian10:~$
```

Note
An **exit code** of 0 means the command executed successfully.

11.7 Double ampersand &&

You can use this **exit code** as a condition to execute another command using the **&&** control operator. Read this as "If the first command succeeds, then execute the second command.".

In the screenshot below we execute the **rm** command only if the **cp** command returned an **exit code** of 0.

```
paul@debian10:~$ cp Linuxfun.pdf Linux.pdf && rm Linuxfun.pdf
paul@debian10:~$
```

11.8 Double vertical bar ||

You can also use the **exit code** to execute another command if the first command **fails**. Use the double pipe symbol **"||"** for this, as shown in the example below.

```
paul@debian10:~$ rm test || echo it failed
rm: cannot remove 'test': No such file or directory
it failed
paul@debian10:~$
```

11.9 Combining the two

You can use both the **&&** and the || to do one thing when the command succeeds and another thing when the command fails. See this for example.

```
paul@debian10:~$ rm fun.pdf && echo it worked || echo it failed
it worked
paul@debian10:~$ rm fun.pdf && echo it worked || echo it failed
rm: cannot remove 'fun.pdf': No such file or directory
it failed
paul@debian10:~$
```

11.10 Operator ;

When you want to run two or more commands after each other, without interpreting the **exit code**, then you can use the semicolon **";"** operator. The result is identical to typing these commands one by one on a separate line.

```
paul@debian10:~$ rm index.htm ; echo hello ; alias
hello
alias ls='ls --color=auto'
paul@debian10:~$
```

11.11 man bash

The man page for **bash** can be reached by typing **man bash**. This man page however is a long read. The next chapters are a summary of what is the most needed Linux knowledge in medium to large organisations.

11.12 Cheat sheet

Table 11.1: Commands

command	**explanation**
/bin/bash	this is the bash shell executable
echo $SHELL	displays your current shell
which	searches the $PATH for a command
echo $PATH	displays the PATH variable (a colon separated list of directories)
type foo	displays the type of the command **foo**
alias	displays a list of all aliases
alias foo=bar	defines the alias **foo** as **bar**
alias cp='cp -i'	defines a default option for **cp**
unalias foo	unsets the alias **foo**
echo $?	displays the last error code
foo && bar	executes **bar** if **foo** succeeds
foo || bar	executes **bar** if **foo** fails
foo ; bar	executes **foo**, then executes **bar**
man bash	a long but interesting read

11.13 Practice

1. Display the external location of the **mv** and **cp** commands.
2. Display the name of the current shell.
3. Is the **pushd** command external or built-in?
4. Create an alias for **cp** so the **-i** option is always used when typing **cp**.
5. List all aliases in your current shell.
6. Remove the **cp** alias.
7. Copy Linuxfun.pdf to fun.pdf and then display the **exit code**.
8. Run **echo it works** but only if **rm fun.pdf** succeeds.
9. Run **echo it failed** but only if **rm fun.pdf** fails.
10. Open the manual of bash and search for the **xtrace** option, is it the same as **-x**?

11.14 Solution

1. Display the external location of the **mv** and **cp** commands.

```
which mv cp
```

2. Display the name of the current shell.

```
echo $SHELL
```

3. Is the **pushd** command external or built-in?

```
type pushd
```

4. Create an alias for **cp** so the **-i** option is always used when typing **cp**.

```
alias cp='cp -i'
```

5. List all aliases in your current shell.

```
alias
```

6. Remove the **cp** alias.

```
unalias cp
```

7. Copy Linuxfun.pdf to fun.pdf and then display the **exit code**.

```
cp Linufun.pdf fun.pdf
echo $?
```

8. Run **echo it works** but only if **rm fun.pdf** succeeds.

```
rm fun.pdf && echo it works
```

9. Run **echo it failed** but only if **rm fun.pdf** fails.

```
rm fun.pdf || echo it failed
```

10. Open the manual of bash and search for the **xtrace** option, is it the same as **-x**?

```
man bash
/xtrace
n n n n
```

Chapter 12

Shell arguments

12.1 White space removal

The shell will remove all whitespace and tabs from your command line. See for example the following three identical command lines.

```
paul@debian10:~$ echo hello world
hello world
paul@debian10:~$ echo    hello    world
hello world
paul@debian10:~$      echo       hello       world
hello world
paul@debian10:~$
```

The shell is effectively cutting your command line into distinct **arguments**. For the lines above the first **argument** is always **echo**. We call this **argument zero**. The second argument is always **hello** without any spaces, we call this **argument one**. And **world** is **argument two**.

The **echo** command will display all the arguments that it receives, with a space added between the arguments. This is true even in this example.

```
paul@debian10:~$ echo hello world
hello world
paul@debian10:~$
```

In the example above the shell removes the space between **echo** and **hello** and the space between **hello** and **world**. The **echo** command then adds a space between **argument one** (hello) and **argument two** (world).

12.2 Single quotes

When using single quotes on the command line, you tell the shell that the stuff between single quotes is **one single argument**. The following screenshot shows a command line that contains only **two arguments** ; namely **argument zero** is **echo** and **argument one** is **hello world** with the six spaces included.

```
paul@debian10:~$ echo 'hello      world'
hello      world
paul@debian10:~$
```

Since the **echo** command only receives one argument, it displays that one argument as is.

Warning
When you fail to enter the second quote, then the shell will assume you have not finished your command line yet. You must enter the second quote, or type **Ctrl-c** for an interrupt.

```
paul@debian10:~$ echo 'hello
> world'
```

12.3 Double quotes

In this chapter using **double quotes** is essentially identical to using **single quotes** ; namely you are telling the shell that the stuff between the **double quotes** is one single argument. Later in the book we will discuss differences between using single quotes and double quotes.

```
paul@debian10:~$ echo "Hello      world"
Hello      world
paul@debian10:~$
```

12.4 Pound sign

The pound sign **"#"** is a control operator where the shell stops interpreting. This means that anything after this **pound sign** is ignored (same as with a hashtag ;-).

```
paul@debian10:~$ echo hello # this is not shown
hello
paul@debian10:~$ echo hello # rm -rf /home is not executed
hello
paul@debian10:~$
```

12.5 Escaping with \

You can **escape** all the control operators and quotes with the **backslash "\"**. This prevents the shell from interpreting the special character.

```
paul@debian10:~$ echo a hashtag looks like this \# yes
a hashtag looks like this # yes
paul@debian10:~$ echo here\'s a quote
here's a quote
paul@debian10:~$
```

12.6 End of line \

If you end your command line with a **** followed directly by an **enter** key, then the **enter** key is ignored. This means you can continue writing your command line, on several lines.

The next screenshot contains a command line over four lines.

```
paul@debian10:~$ echo Hello \
> world \
> and also Mars,\
> and Venus
Hello world and also Mars,and Venus
paul@debian10:~$
```

12.7 set -x

You can verify what the shell is doing with your command line by typing **set -x** to activate the shell **xtrace** option. (Yes the shell has options, more on that later in this book).

In the example below we use this option to see that **echo** first gets two arguments and in the second command gets only one argument. (Look behind the + sign).

```
paul@debian10:~$ set -x
paul@debian10:~$ echo hello world
+ echo hello world
hello world
paul@debian10:~$ echo 'hello world'
+ echo 'hello world'
hello world
paul@debian10:~$
```

You can disable the **-x** option by typing **set +x** or **set +o xtrace**.

12.8 Cheat sheet

Table 12.1: Arguments

command	**explanation**
echo hello world	this is three arguments, with argument zero being **echo**
echo 'hello world'	this is two arguments, the second is **hello world**
echo "hello world"	this is two arguments, the second is **hello world**
foo #bar	**bar** is a comment (not interpreted by the shell)
echo \#	this will display a # because it is escaped (so not interpreted)
set -x	activate the shell's xtrace option

12.9 Practice

1. Create one file named **foo bar** (do not create two files foo and bar).
2. Use tab-completion to remove this file.
3. Echo a single and a double quote to the screen.
4. Find three different ways to display a pound sign # on the screen.
5. Write a command line on three lines.
6. Activate the **xtrace** option in bash.
7. Execute **echo $SHELL** while the **xtrace** is active and try to explain what happens.
8. Disable the **xtrace** option.

12.10 Solution

1. Create one file named **foo bar** (do not create two files foo and bar).

```
touch 'foo bar'
```

2. Use tab-completion to remove this file.

```
rm fo tab-key
```

3. Echo a single and a double quote to the screen.

```
echo \' \"
```

4. Find three different ways to display a pound sign # on the screen.

```
echo \#
echo "#"
echo '#'
```

5. Write a command line on three lines.

```
echo hello \
world \
and Mars
```

6. Activate the **xtrace** option in bash.

```
set -x
```

or

```
set -o xtrace
```

7. Execute **echo $SHELL** while the **xtrace** is active and try to explain what happens. We will see the solution in the next chapter.
8. Disable the **xtrace** option.

```
set +o xtrace
```

or

```
set +x
```

Chapter 13

Shell variables

13.1 Variables in the shell

Variables in the shell always start with a dollar sign **"$"**. Applications can use variables to define certain settings (for example: a path to a directory).

In this chapter we discuss variables in the shell, later in this book we will see how applications define and use variables.

13.2 declare

You can define a variable by **declaring** it. In the screenshot below we declare two variables; the first explicitly, the second implicitly.

```
paul@debian10:~$ declare var1=33
paul@debian10:~$ var2=42
paul@debian10:~$
```

You can now use echo to use these variables.

```
paul@debian10:~$ echo The answer is $var2
The answer is 42
paul@debian10:~$ echo Why $var1 seconds
Why 33 seconds
paul@debian10:~$
```

You can also put a string (like a path) in a variable.

```
paul@debian10:~$ var3=/var/tmp
paul@debian10:~$ cd $var3
paul@debian10:/var/tmp$
```

Note
Notice that when declaring a variable you do not use the dollar sign **$**.

13.3 Variables and quotes

Variables between **double quotes** are still parsed by the shell and thus are replaced by their value.

```
paul@debian10:~$ echo "The answer is $var2"
The answer is 42
paul@debian10:~$
```

Variables between **single quotes** are not parsed, this is a big difference between **single** and **double** quotes.

```
paul@debian10:~$ echo 'Why $var1 seconds?'
Why $var1 seconds?
paul@debian10:~$
```

Note
Single quotes are often called **full quotes**.

13.4 Parsed by the shell?

Yes, the **bash shell** is replacing variables with their value. Consider this example with **xtrace** active.

```
paul@debian10:~$ set -x
paul@debian10:~$ echo "The answer is $var2"
+ echo 'The answer is 42'
The answer is 42
paul@debian10:~$
```

The **echo** command in the example above does not see **$var2**, it receives **42** instead. The **echo** command doesn't even know how to handle variables, it simply *echoes* what it receives from the bash shell.

13.5 Escaping the $

Another way to prevent the parsing of variables by the shell is to escape the dollar sign as shown in this screenshot.

```
paul@debian10:~$ echo Why \$var1 seconds?
Why $var1 seconds?
paul@debian10:~$
```

13.6 set

You can see the list of all variables in the shell by typing the **set** command. There are many variables so we truncate the screenshot to the top nine.

```
paul@debian10:~$ set | head -9
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote: ↩
    force_fignore:globasciiranges:histappend:interactive_comments:login_shell:progcomp: ↩
    promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=([0]="0")
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=([0]="2" [1]="8")
BASH_LINENO=()
BASH_SOURCE=()
paul@debian10:~$
```

The **set** command will also list all **functions**, more on **functions** in the Scripting chapters.

You can use **grep** in combination with **set** to find the variables that we just created.

```
paul@debian10:~$ set | grep var1
var1=33
paul@debian10:~$ set | grep var2
var2=42
paul@debian10:~$
```

Note
We will explain the | symbol in the redirection chapter.

13.7 unset

You can **unset** a variable with the **unset** command. The variable then no longer exists. The example below shows what happens when you use an non-existing variable.

```
paul@debian10:~$ echo The answer is $var2
The answer is 42
paul@debian10:~$ unset var2
paul@debian10:~$ echo The answer is $var2
The answer is
paul@debian10:~$
```

As you can see the non-existing variable is replaced with *nothing*.

Warning
Do not use the $ when unsetting a variable.

13.8 Unbound variables

You may want an error message when you use a non-existing (=unbound) variable. You can achieve this by setting the **nounset** shell option either with **set -u** or with **set -o nounset**.

```
paul@debian10:~$ echo The answer is $var2
The answer is
paul@debian10:~$ set -o nounset
paul@debian10:~$ echo The answer is $var2
-bash: var2: unbound variable
paul@debian10:~$
```

13.9 Delineate variables

Consider the following example where we create a **$pre** variable with a value of **Super**.

```
paul@debian10:~$ pre=Super
paul@debian10:~$ echo $preman is stronger than $pregirl
is stronger than
paul@debian10:~$
```

The **$pre** is not recognised in the example above, this can be solved in several ways.

```
paul@debian10:~$ echo ${pre}man is stronger than $pre'girl'
Superman is stronger than Supergirl
paul@debian10:~$ echo "$pre"man is stronger than $pre"girl"
Superman is stronger than Supergirl
paul@debian10:~$
```

13.10 $SHELL

The **$SHELL** variable is initialised by the **bash** shell when the shell is started. It is set to **/bin/bash**. If, one day, the **bash** shell is replaced with another shell, then this variable will probably be changed.

```
paul@debian10:~$ echo $SHELL
/bin/bash
paul@debian10:~$
```

13.11 $PS1

The **$PS1** variable determines the look of the prompt. In the example below we set the prompt to the text **prompt>** . Everything else is unaffected, the shell works exactly as before.

```
paul@debian10:~$ PS1='prompt>'
prompt>
prompt>file studentfiles.html
studentfiles.html: HTML document, ASCII text
prompt>
```

We can set it back to something useful using the following shortcuts.

Table 13.1: PS1 backslash-escaped shortcuts

shortcut	decoding
\u	name of current user
\h	hostname
\w	current directory
\$	$ for users and # for root
\t	current time

So let us set the prompt to include these shortcuts. To do this we simply set the **$PS1** variable.

```
prompt>
prompt>PS1='\u \h \w \t '
paul debian10 ~ 07:32:38
paul debian10 ~ 07:32:39
```

Almost good, so let us include the @ and the **:** and the **$**, and do not forget to add a space at the end.

```
paul debian10 ~ 07:32:39 PS1=\'\u@\h:\w@\t\$ '
paul@debian10:~@07:34:21$
paul@debian10:~@07:34:27$
```

Later in this book we will see how to add a *function* to the prompt.

13.12 $PATH

We have discussed the **$PATH** variable before, together with the **which** command. This variable contains a list of directories, separated by a colon **:** .

```
paul@debian10:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
paul@debian10:~$
```

These directories are checked in order when executing a command (the **which** command also checks them in order). So if a command with the same name exists in two of these directories, then only the first will be executed.

You can add or remove directories from the **$PATH** by setting the variable. In the example below we add the **/home/paul/bin** directory at the end of the **$PATH**.

```
paul@debian10:~$ PATH=$PATH:/home/paul/bin
paul@debian10:~$
```

You may have noticed that the **current directory** is not in the **$PATH**. This is for security reasons. You can however add it, in front of all the other directories, if you desire to live dangerously.

```
paul@debian10:~$ PATH=.:$PATH
paul@debian10:~$ echo $PATH
.:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/home/paul/bin
paul@debian10:~$
```

Note
Remember the **current directory** is represented by a simple dot.

13.13 Cheat sheet

Table 13.2: Variables

command	explanation
declare foo=42	Declares and initialises a variable named **foo**
echo $foo	Displays the value of the variable **foo**
echo "$foo"	Displays the value of the variable **foo**
echo '$foo'	Displays $foo
set | grep foo	Searches in all variable names for the string **foo**
unset foo	Unsets (destroys) the variable named **foo**
set -o nounset	Sets the shell option to error when using non-existing variables.
echo $foo"bar"	Displays the contents of **foo** followed by the string **bar**.
echo "$foo"bar	Displays the contents of **foo** followed by the string **bar**.
echo $foo'bar'	Displays the contents of **foo** followed by the string **bar**.
echo ${foo}bar	Displays the contents of **foo** followed by the string **bar**.
$PATH	This variable contains directories where commands can be found.
$SHELL	This variable contains the name of the current shell.
$PS1	This variable contains the prompt in the bash shell.

13.14 Practice

1. Declare a variable named **$varone** with a value of **100** .
2. Declare a variable named **$mypath** with a value of **/home/yourname** .
3. Display both your variables on one line.
4. Display both your variables on one line with six spaces between them.
5. Display **$varone=100** by escaping the first dollar sign.
6. Use the **set** command to display your variables.
7. Delete the **$mypath** variable.
8. Make sure you get an error when using **$mypath** .
9. Remove the games directories from the **$PATH** .
10. Log out and log back on, are your variables still there?

13.15 Solution

1. Declare a variable named **$varone** with a value of **100** .

```
varone=100
```

2. Declare a variable named **$mypath** with a value of **/home/yourname** .

```
mypath=/home/paul
```

3. Display both your variables on one line.

```
echo $mypath and $varone
```

4. Display both your variables on one line with six spaces between them.

```
echo "$mypath      $varone"
```

or

```
echo $mypath'      '$varone
```

5. Display **$varone=100** by escaping the first dollar sign.

```
echo \$varone=$varone
```

6. Use the **set** command to display your variables.

```
set | grep varone
set | grep mypath
```

7. Delete the **$mypath** variable.

```
unset mypath
```

8. Make sure you get an error when using **$mypath** .

```
set -u
echo $mypath
```

9. Remove the games directories from the **$PATH** .

```
PATH=/usr/local/bin:/usr/bin:/bin
```

10. Log out and log back on, are your variables still there?

```
no
```

Chapter 14

Shell embedding and child shells

14.1 Shell embedding

Shell **embedding** is a useful trick to save time and effort, remember Linux people are lazy and lazy people are efficient. Consider the following two statements, where we copy the output from the **which** command and use it as input for the **ls -l** command.

```
paul@debian10:~$ which cat
/usr/bin/cat
paul@debian10:~$ ls -l /usr/bin/cat
-rwxr-xr-x 1 root root 43744 Feb 28 16:30 /usr/bin/cat
paul@debian10:~$
```

We can shorten this by embedding a **bash** shell in the command with the **$( )** notation.

```
paul@debian10:~$ ls -l $(which cat)
-rwxr-xr-x 1 root root 43744 Feb 28 16:30 /usr/bin/cat
paul@debian10:~$
```

When the **bash** shell is evaluating your command, it encounters the **$(** and there it starts a new **bash** shell. That new **embedded** shell then executes the **which cat** command and gives the result back to our **bash** shell.

Nothing remains of the embedded shell once it executed its command(s). A variable defined inside the embedded shell does not exist outside of that shell.

```
paul@debian10:~$ echo $(var5=500 ; echo \$var5=$var5 )
$var5=500
paul@debian10:~$ echo $var5
-bash: var5: unbound variable
paul@debian10:~$
```

Note
You can *nest* embedded shells, sometimes this is useful!

14.2 Backticks

Some people use **backticks** instead of the **$( )** notation for embedded shells. This can be confusing to new users who mistake them for **single quotes**. The result is quite different when using single quotes.

```
paul@debian10:~$ ls -l `which cat`
-rwxr-xr-x 1 root root 43744 Feb 28 16:30 /usr/bin/cat
paul@debian10:~$ ls -l 'which cat'
ls: cannot access which cat: No such file or directory
paul@debian10:~$
```

Note
You cannot nest embedded shells with backticks.

14.3 Child shells

Besides embedding shells, you can also create **child** shells. A **child shell** is created when you type **bash** at the command line. And you can exit this **child shell** by typing **exit** at the command line.

```
paul@debian10:~$ bash
paul@debian10:~$ echo inside the child shell now
inside the child shell now
paul@debian10:~$ exit
exit
paul@debian10:~$
```

In this **child shell** you can do a lot of things without influencing your **parent shell**. For example you can create variables, change them, and destroy them, without changing any of the variables of the **parent shell**.

```
paul@debian10:~$ varans=42
paul@debian10:~$ bash
paul@debian10:~$ varans=8472
paul@debian10:~$ echo $varans
8472
paul@debian10:~$ exit
exit
paul@debian10:~$ echo $varans
42
paul@debian10:~$
```

14.4 Verifying that shell is child shell

You can verify that you are in a child shell by displaying the **$SHLVL** variable.

```
paul@debian10:~$ echo $SHLVL
1
paul@debian10:~$ bash
paul@debian10:~$ echo $SHLVL
2
paul@debian10:~$ exit
exit
paul@debian10:~$ echo $SHLVL
1
paul@debian10:~$
```

14.5 Exporting variables

A **child shell** will inherit all **exported** variables. See the example below where we use the **export** command to export the **$varans** variable.

```
paul@debian10:~$ varmin=33
paul@debian10:~$ varans=42
paul@debian10:~$ export varans
paul@debian10:~$ bash
paul@debian10:~$ echo $varmin does not exist
does not exist
paul@debian10:~$ echo $varans is the answer
42 is the answer
paul@debian10:~$
```

Tip
You can type **export varans=42** to export and initialise a variable.

14.6 env - printenv - export

There is, of course, a list of all exported variables. You can display this list by typing **env**, or **printenv**, or **export** without any arguments.

```
paul@debian10:~$ env | tail -3
MAIL=/var/mail/paul
SSH_TTY=/dev/pts/0
_=/usr/bin/env
paul@debian10:~$ printenv | tail -3
MAIL=/var/mail/paul
SSH_TTY=/dev/pts/0
_=/usr/bin/printenv
paul@debian10:~$ export | tail -3
declare -x XDG_SESSION_ID="54"
declare -x XDG_SESSION_TYPE="tty"
declare -x varans="42"
paul@debian10:~$
```

Note
Note that **export** gives you a **declare -x** command to **declare** and **export** variables in one command.

14.7 env

You can also use the **env** command to execute a new shell (or any other command) in a new environment. In the screenshot below we start a new child **bash** shell, but the **-i** option prevents inheritance of even **exported** variables.

```
paul@debian10:~$ export varans=42
paul@debian10:~$ varmin=33
paul@debian10:~$ env -i varspe=8472  bash
paul@debian10:/home/paul$ echo $varspe $varans $varmin
8472
paul@debian10:/home/paul$ exit
exit
paul@debian10:~$
```

14.8 Cheat sheet

Table 14.1: Variables

command	**explanation**
foo $(bar)	first execute **bar**, then do **foo**
foo `bar`	first execute **bar**, then do **foo**
bash	starts a new bash shell (with its own environment)
exit	exits the current bash shell
Ctrl-d	exits the current bash shell (by sending a EOF character)
echo $SHLVL	displays how many shells deep you are
export foo	exports the variable foo to child shells
env	prints all **exported** variables
printenv	prints all **exported** variables
export	prints all **exported** variables
env foo	execute the **foo** command in a default environment
env -i foo	execute the **foo** command in an empty environment

14.9 Practice

1. Verify that you are not in a **child shell** . Exit if necessary.
2. Create and echo a variable in an **embedded** shell.
3. Export the **varmin=33** variable.
4. Start a **child** shell. Then start another **child** shell.
5. Verify that **$SHLVL** is indeed 3.
6. List all the exported variables.
7. Verify that your **$varmin** exists and is **33** .
8. Start a new bash shell in a clean environment.

14.10 Solution

1. Verify that you are not in a **child shell** . Exit if necessary.

```
echo $SHLVL
exit
```

2. Create and echo a variable in an **embedded** shell.

```
echo $( varans=42; echo The value is $varans )
```

or

```
echo ` varans=42; echo The value is $varans `
```

3. Export the **varmin=33** variable.

```
export varmin=33
```

or

```
declare -x varmin=33
```

4. Start a **child** shell. Then start another **child** shell.

```
bash
bash
```

5. Verify that **$SHLVL** is indeed 3.

```
echo $SHLVL
```

6. List all the exported variables.

```
env
printenv
export
```

7. Verify that your **$varmin** exists and is **33** .

```
echo $varmin
```

8. Start a new bash shell in a clean environment.

```
env -i bash
```

Chapter 15

Shell history

15.1 history

The **history** command shows you a list of all the commands that you typed. By default the last one thousand commands are kept in this list.

```
paul@debian10:~$ history | grep export
  825  export varans
  830  export
  834  which export
  841  export | tail -3
  849  export varans=42
  852  export varans=42
  857  export varans=42
  861  history | grep export
paul@debian10:~$
```

You can add a number to see only the last number of commands. In the screenshot below we show the last five commands.

```
paul@debian10:~$ history 5
  858  varmin=33
  859  env -i varspe=8472  bash
  860  history | grep varans
  861  history | grep export
  862  history 5
paul@debian10:~$
```

15.2 !n

You can execute any command from the history list by typing an exclamation mark followed by the line number.

```
paul@debian10:~$ !858
varmin=33
paul@debian10:~$
```

You can also execute a command from history by typing the exclamation mark followed by the first letter(s) of the command you want to repeat.

```
paul@debian10:~$ !ex
export varans=42
paul@debian10:~$
```

Warning
Make sure you know exactly which command you are repeating with this last option.

To execute the last command again simply type **!!** (often called bang bang), as seen in the screenshot below.

```
paul@debian10:~$ !!
export varans=42
paul@debian10:~$
```

15.3 Ctrl-r

You can also search backwards in your history by typing **Ctrl-r** followed by the string you are looking for. In the screenshot below we type **Ctrl-r** followed by **de**.

```
paul@debian10:~$ !858
varmin=33
(reverse-i-search)de: declare -x varmin=33
```

Tip
You can search further down the history by typing **Ctrl-r** several times.

15.4 ~/.bash_history

The list of commands that forms your **history** is stored in a hidden file in your home directory named **.bash_history** .

```
paul@debian10:~$ file .bash_history
.bash_history: ASCII text
paul@debian10:~$ tail -4 .bash_history
echo $varans $varmin
exit
echo $varspe $varans $varmin
exit
paul@debian10:~$
```

Warning
The **.bash_history** file is updated with your current history every time you **properly** exit the bash shell (by typing **exit** or **Ctrl-d**)!

15.5 $HISTSIZE

By default the **$HISTSIZE** variable is set to a value of **1000** in Debian. You can change this to have the **active shell** retain more (or less) commands. We will see later in this book how to make this change permanent.

```
paul@debian10:~$ echo $HISTSIZE
1000
paul@debian10:~$ HISTSIZE=4000
paul@debian10:~$ echo $HISTSIZE
4000
paul@debian10:~$
```

15.6 $HISTFILESIZE

The number of commands stored in the **.bash_history** file is by default set to **2000**. In the screenshot below we set this to **9000**. (Again we will see later in this book how to make this change permanent for a user.)

```
paul@debian10:~$ echo $HISTFILESIZE
2000
paul@debian10:~$ HISTFILESIZE=9000
paul@debian10:~$ echo $HISTFILESIZE
9000
paul@debian10:~$
```

15.7 $HISTCONTROL

By default the $HISTCONTROL variable is set to **ignoreboth** which means that command lines that start with a space character are not stored in the shell history, and identical repeating commands are only stored once in the history.

```
paul@debian10:~$ echo $HISTCONTROL
ignoreboth
paul@debian10:~$
```

15.8 Ctrl-a Ctrl-e

Whenever you have text on the command line, you can use **Ctrl-a** to get to the start of your command line, and **Ctrl-e** to get to the end of your command line.

15.9 Cheat sheet

Table 15.1: Shell history

command	explanation
history	Display all the commands you entered.
history 5	Display the last 5 commands you entered.
!42	Execute the 42nd command again.
!ex	Execute the last command that started with **ex**.
!!	Execute the last command again.
Ctrl-r	Search back in your command history.
~/.bash_history	The file that contains your history of entered commands.
$HISTSIZE	This variable contains the maximum number of commands in your history.
$HISTFILESIZE	The maximum number of commands in the $HISTFILE.
$HISTFILE	Contains the name of the history file.
Ctrl-a	Jump to the start of the current command line.
Ctrl-e	Jump to the end of the current command line.

15.10 Practice

1. Display all the previous commands you entered.
2. Display the last 10 commands you entered.
3. Find your last **echo** commands.
4. Use the number in front of the last **echo** command to re-execute it.
5. Repeat the last echo command using only two characters.
6. Search your history for the last **export** command.
7. Display the first five lines of your all-time history on this computer.
8. Set the number of commands remembered to 5000.
9. Set the number of commands in the history file to 8000.
10. Repeat the last command.

15.11 Solution

1. Display all the previous commands you entered.

```
history
```

2. Display the last 10 commands you entered.

```
history 10
```

3. Find your last **echo** commands.

```
history | grep echo
```

4. Use the number in front of the last **echo** command to re-execute it.

```
!874 (or something)
```

5. Repeat the last echo command using only two characters.

```
!ec
```

6. Search your history for the last **export** command.

```
Ctrl-r ex
```

7. Display the first five lines of your all-time history on this computer.

```
head -5 .bash_history
```

8. Set the number of commands remembered to 5000.

```
HISTSIZE=5000
```

9. Set the number of commands in the history file to 8000.

```
HISTFILESIZE=8000
```

10. Repeat the last command.

```
!!
```

Chapter 16

Shell file globbing

16.1 About globbing

File globbing is the use of wildcards to specify a set of filenames. There are several of these wildcards in use in Linux (or rather in the bash shell).

All the examples here are shown in the **~/globbing** directory.

16.2 asterisk *

The first wildcard we will use is the asterisk ***** . This asterisk stands for **zero, one or any number of characters**. Note that in the screenshot below all four files match the pattern.

```
paul@debian10:~$ mkdir globbing
paul@debian10:~$ cd globbing/
paul@debian10:~/globbing$ touch file file2 fileA fileABC
paul@debian10:~/globbing$ ls file*
file  file2  fileA  fileABC
paul@debian10:~/globbing$
```

We get of course the same result using **f*** instead of using **file*** .

```
paul@debian10:~/globbing$ ls f*
file  file2  fileA  fileABC
paul@debian10:~/globbing$
```

16.3 question mark?

The second wildcard character to show is the question mark **?** . This question mark represents **exactly one character**. Note that only two out of four files in the next screenshot match this pattern.

```
paul@debian10:~/globbing$ ls file?
file2  fileA
paul@debian10:~/globbing$ ls
file  file2  fileA  fileABC
paul@debian10:~/globbing$
```

Here is another example that uses multiple question marks, each one representing one single character.

```
paul@debian10:~/globbing$ touch first
paul@debian10:~/globbing$ ls
file  file2  fileA  fileABC  first
paul@debian10:~/globbing$ ls fi???
file2  fileA  first
paul@debian10:~/globbing$
```

16.4 square brackets [ ]

You can put a selection of characters between square brackets to represent an **exactly one character** wildcard. For example [ab] means one character a or one character b. Similarly [5d] means exactly one 5 or exactly one d.

```
paul@debian10:~/globbing$ ls
file  file2  fileA  fileABC  first
paul@debian10:~/globbing$ ls file[2A]
file2  fileA
```

```
paul@debian10:~/globbing$ ls file[ace9]
ls: cannot access file[ace9]: No such file or directory
paul@debian10:~/globbing$
```

Tip
The order between square brackets is not important.

16.5 square brackets series

You can put series inside the square brackets. For example [a-z] means **exactly one letter between a and z** . Similarly [0-9] means **exactly one number between 0 and 9** .

```
paul@debian10:~/globbing$ touch file23
paul@debian10:~/globbing$ ls
file  file2  file23  fileA  fileABC  first
paul@debian10:~/globbing$ ls file[0-9]
file2
paul@debian10:~/globbing$ ls file[0-9][0-9]
file23
paul@debian10:~/globbing$
```

16.6 ascii series

You can put any series between brackets that is present in the ASCII table. So the **space to tilde [-~]** series contains every readable ASCII character. (Remember to escape the space, otherwise the shell will interpret it as two separate arguments.)

```
paul@debian10:~/globbing$ ls
file  file2  file23  fileA  fileABC  first
paul@debian10:~/globbing$ ls file[\ -~]
file2  fileA
paul@debian10:~/globbing$
```

16.7 multiple series

You can put multiple series between one pair of square brackets. The **[0-9A-Z]** means exactly one number or uppercase letter.

```
paul@debian10:~/globbing$ ls
file  file2  file23  fileA  fileABC  first
paul@debian10:~/globbing$ ls file[0-9A-Z]
file2  fileA
paul@debian10:~/globbing$
```

16.8 exclude letters

Using the **exclamation mark** inside the square brackets means the inverse, so **not** the characters listed. In the example we ask for the fifth character to **not** be a number.

```
paul@debian10:~/globbing$ ls
file  file2  file23  fileA  fileABC  first
paul@debian10:~/globbing$ ls file[!0-9]
fileA
paul@debian10:~/globbing$
```

16.9 Cheat sheet

Table 16.1: file globbing

glob	**explanation**
*	zero, one or any number of characters.
ls f*	list all files starting with f (including a file named **f**).
?	exactly one character
ls f?	list all files that start with f and have exactly two characters
ls?????	list all files with exactly five characters
[abc]	Exactly one character a, b, or c.
[z2a]	Exactly one character z, 2 or a.
[0-9]	Exactly one digit.
[a-z][a-z][0-9]	Exactly two letters and one digit (three characters in total)
[!0-9]	Not a digit.

16.10 Practice

1. Create and enter a **~/globbing** directory.
2. Create the following files: **text text4 textB text.txt** .
3. List all files starting with **te** .
4. List all files ending in a number.
5. List all files of exactly 8 characters.
6. Create a file named 5test.txt, then list all files not starting with a letter.

16.11 Solution

1. Create and enter a **~/globbing** directory.

```
mkdir ~/globbing
cd ~/globbing
```

2. Create the following files: **text text4 textB text.txt** .

```
touch text text4 textB text.txt
```

3. List all files starting with **te** .

```
ls te*
```

4. List all files ending in a number.

```
ls *[0-9]
```

5. List all files of exactly 8 characters.

```
ls????????
```

6. Create a file named 5test.txt, then list all files not starting with a letter.

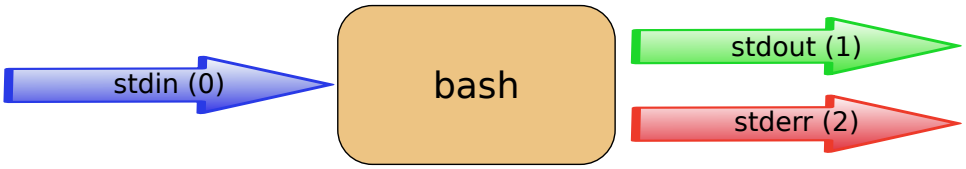
```
ls [!A-Za-z]*
```

Chapter 17

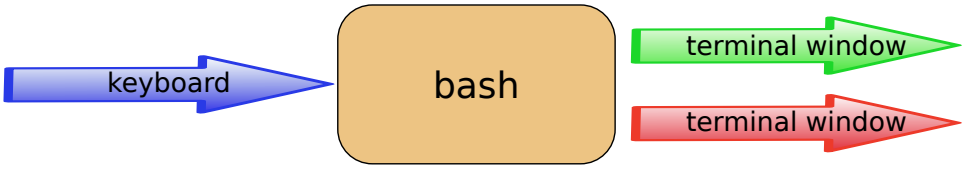
Shell redirection

17.1 stdin, stdout, stderr

The **bash** shell has three default streams to work with. Stream 0 is named **stdin**, stream 1 is named **stdout** and stream 2 is named **stderr**.

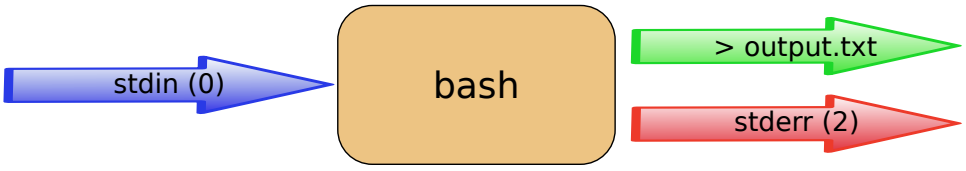


By default **stream 0** is connected to the **keyboard**, while **stream 1 and 2** are both directed to your terminal window.



17.2 > stdout

Using the **bigger than** sign **>** we can redirect **stream 1** to something else than the terminal window. In fact **>** is identical to **1>** because we are redirecting **stream 1** .



In the image above and in the screenshot below we redirect **stream 1** to a file named **output.txt** .

```
paul@debian10:~$ ls *.gz > output.txt
paul@debian10:~$ cat output.txt
all.txt.gz
services.gz
words.gz
paul@debian10:~$
```

Note that **stream 2** a.k.a. **stderr** still goes to the terminal window, as shown in the following screenshot.

```
paul@debian10:~$ ls *.no > output.txt
ls: cannot access '*.no': No such file or directory
paul@debian10:~$
```

17.3 >> stdout append

Data can be appended to a file when redirecting **stdout** using the **>>** or **1>>** notation, as is shown in the next screenshot.

```
paul@debian10:~$ ls *.gz > output.txt
paul@debian10:~$ cat output.txt
all.txt.gz
services.gz
words.gz
```

```
paul@debian10:~$ ls *.pdf >> output.txt
paul@debian10:~$ cat output.txt
all.txt.gz
services.gz
words.gz
Linux.pdf
paul@debian10:~$
```

17.4 noclobber

There is the danger, when using **stdout redirection**, of overwriting an existing file. In the example below we lose the contents of the **output.txt** file.

```
paul@debian10:~$ cat output.txt
all.txt.gz
services.gz
words.gz
Linux.pdf
paul@debian10:~$ echo hello > output.txt
paul@debian10:~$ cat output.txt
hello
paul@debian10:~$
```

This danger can be mitigated by setting the **noclobber** option (**set -C** or **set -o noclobber**). This shell option prevents accidental overwriting of an existing file.

```
paul@debian10:~$ cat output.txt
hello
paul@debian10:~$ set -C
paul@debian10:~$ echo hello > output.txt
-bash: output.txt: cannot overwrite existing file
paul@debian10:~$
```

But even with the **noclobber** option active, a file can be overwritten using the **>|** notation.

```
paul@debian10:~$ cat output.txt
hello
paul@debian10:~$ set -C
paul@debian10:~$ echo hello > output.txt
-bash: output.txt: cannot overwrite existing file
paul@debian10:~$ echo hello >| output.txt
paul@debian10:~$
```

17.5 quick file clear

The above explanation has as a consequence that **> file** is a very quick way to clear a file (or create an empty file if it does not exist).

```
paul@debian10:~$ > empty.txt
paul@debian10:~$
```

Using **>|** of course when you suspect that the **noclobber** option is active.

```
paul@debian10:~$ set -o noclobber
paul@debian10:~$ >| empty.txt
paul@debian10:~$
```

17.6 redirection by the shell

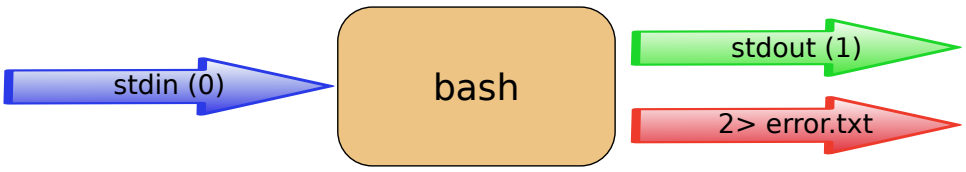
The **bash shell** is redirecting **stdout** when using **>** or **>>** and the **bash shell** is effectively removing the redirection from the command line. This allows you to put the redirection anywhere in the command line. The following three command lines are identical, except that the first one is more readable.

```
paul@debian10:~$ echo hello > output.txt
paul@debian10:~$ > output.txt echo hello
paul@debian10:~$ echo > output.txt hello
paul@debian10:~$
```

The same is true for redirection of **stderr** using **2>** or **2>>**.

17.7 2> stderr

In the image below we redirect **stream 2** to a file named **error.txt** using **2>** . In this way we redirect all errors that would appear in the terminal window instead to the file **error.txt**.



On the command line this looks like this.

```
paul@debian10:~$ ls *.no 2> error.txt
paul@debian10:~$
```

Note that normal output (a.k.a. stdout) is still directed to the terminal window, as shown in the example below.

```
paul@debian10:~$ ls *.gz 2> error.txt
all.txt.gz  services.gz  words.gz
paul@debian10:~$
```

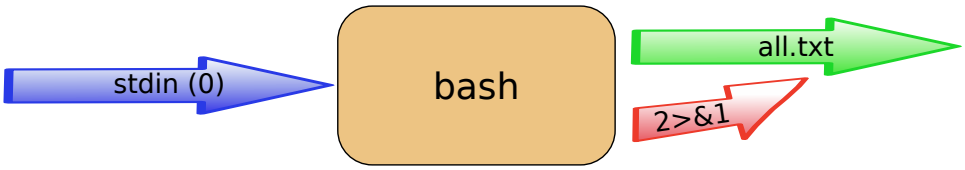
17.8 2>> stderr append

Errors can be appended to an existing file using the **2>>** notation, as is shown in the screenshot below.

```
paul@debian10:~$ ls *.no 2>> error.txt
paul@debian10:~$ cat error.txt
ls: cannot access '*.no': No such file or directory
paul@debian10:~$ ls *.no 2>> error.txt
paul@debian10:~$ cat error.txt
ls: cannot access '*.no': No such file or directory
ls: cannot access '*.no': No such file or directory
paul@debian10:~$
```

17.9 2>&1 combining stdout and stderr

You can combine **stdout** and **stderr** in the same file using the **2>&1** notation. See the example below where we redirect both **streams** to **all.txt** .



In the command line this looks like this.

```
paul@debian10:~$ find / > all.txt 2>&1
paul@debian10:~$
```

Tip
There is no output on your terminal window when redirecting both **stdout** and **stderr** to one file, or to two separate files.

Warning
In the case above the order matters, **> all.txt** must come before **2>&1** !

17.10 &> combining stdout and stderr

You can also combine both **stdout** and **stderr** using the **&>** notation. Both statements in the next screenshot are identical.

```
paul@debian10:~$ find / > all.txt 2>&1
paul@debian10:~$ find / &> all.txt
paul@debian10:~$
```

17.11 swapping stdout and stderr

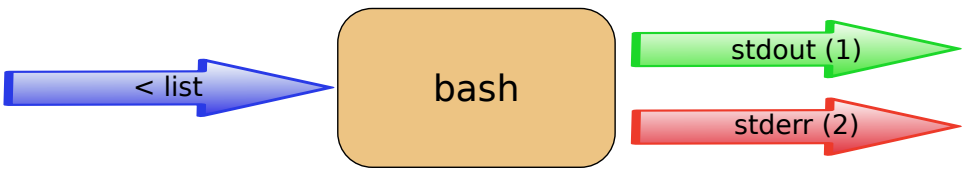
You can swap **stdout** and **stderr** using **stream 3** (streams 3 to 9 are undefined until you use them). The swap goes like this; redirect stream 3 to 1, then stream 1 to 2, then stream 2 to 3. On the command line this looks like this:

```
paul@debian10:~$ echo hello 3>&1 1>&2 2>&3
hello
paul@debian10:~$
```

This swap can be useful when you want to do actions like **grep**, **head** or other filters on **stderr** instead of on **stdout** .

17.12 < redirecting stdin

In the example below we redirect **stdin** to come from a file (instead of the keyboard), the file is named **list** and the redirection symbol is a **smaller than** sign **<** .



On the command line it looks like the screenshot below. We use **cat** and **grep** in this screenshot because we have not seen **filters** yet that cannot read from a file (which **cat** and **grep** can).

```
paul@debian10:~$ ls *.txt > list
paul@debian10:~$ cat < list
all.txt
combi.txt
count.txt
dates.txt
empty.txt
error.txt
hello.txt
output.txt
test.txt
paul@debian10:~$ grep dates < list
dates.txt
paul@debian10:~$
```

17.13 << here document

The **here document** construct is a way to send text (from the keyboard) to a running program until it receives the **limit string**. For example let us create a music artist list, until we type **stop**.

```
paul@debian10:~$ cat <<stop
> Abba
> Bowie
> Queen
> stop
Abba
Bowie
Queen
paul@debian10:~$
```

We can keep the result in a file using **stdout** redirection. The rest is identical to the example above.

```
paul@debian10:~$ cat <<stop > music.txt
> Queen
> Abba
> Bowie
> stop
paul@debian10:~$ head music.txt
Queen
Abba
Bowie
paul@debian10:~$
```

17.14 <<< here string

In the example below we send the value of a variable to the **grep** command. You can put any string here.

```
paul@debian10:~$ vartext="The answer is 42."
paul@debian10:~$ grep 42 <<< $vartext
The answer is 42.
paul@debian10:~$ grep 33 <<< $vartext
paul@debian10:~$
```

17.15 | pipe symbol

In the previous chapters we already used the pipe symbol | to combine two commands, for example **cat** and **grep** as seen in this example.

```
paul@debian10:~$ cat dates.txt | grep France
 FR France          1582-12-09      SE Sweden          1753-02-17
paul@debian10:~$
```

What actually happens is the **stdout** of **cat dates.txt** will be used as **stdin** for **grep France**. So the pipe symbol takes **stdout** from a command and serves it as **stdin** for the next command.

Note
The pipe symbol does not forward **stderr**.

17.16 Cheat sheet

Table 17.1: Shell redirection

redirection	**explanation**
stdin	standard input
stream 0	standard input
stdout	standard output
stream 1	standard output
stderr	standard error
stream 2	standard error
>	redirect stdout
1>	redirect stdout
>>	append redirect stdout
set -C	activate the **noclobber** option
> foo	quickly clear (or create) the file named **foo**
2>	redirect stderr
2>>	append redirect stderr
2>&1	redirect stderr to stream 1 (stdout)
&>foo	redirect stream 1 and 2 into file **foo**
<foo	redirect stdin from the file named **foo**
<<	here document (until the **limit string**)
<<<	here string (redirect input from a string)
foo|bar	pipe stdout of **foo** to stdin of **bar**

17.17 Practice

1. Execute the **find /** command and notice that you get a lot of output in your terminal window.
2. Execute the same command, but redirect standard output to a file named **all.txt**. You should see different and less output on the screen.
3. Now while redirecting output, also redirect the errors to a file named **error.txt** .
4. Verify with **head** and **tail** that there is a list of files in **all.txt** .
5. Verify with **grep** that **error.txt** is filled with **Permission denied**.
6. Quickly empty the **error.txt** file.
7. Activate the **noclobber** shell option.
8. Quickly empty the **all.txt** file (with the noclobber active).
9. Use cat to display the **dates.txt** file. Then redirect **stdin** to display this file.
10. Use **cat** to create a file listing tennis players, use **quit** as a **limit string**.

17.18 Solution

1. Execute the **find /** command and notice that you get a lot of output in your terminal window.

```
find /
```

2. Execute the same command, but redirect standard output to a file named **all.txt**. You should see different and less output on the screen.

```
find / > all.txt
```

3. Now while redirecting output, also redirect the errors to a file named **error.txt** .

```
find / > all.txt 2> error.txt
```

4. Verify with **head** and **tail** that there is a list of files in **all.txt** .

```
head all.txt
tail all.txt
```

5. Verify with **grep** that **error.txt** is filled with **Permission denied**.

```
grep Permission error.txt
```

6. Quickly empty the **error.txt** file.

```
> error.txt
```

7. Activate the **noclobber** shell option.

```
set -o noclobber
```

or

```
set -C
```

8. Quickly empty the **all.txt** file (with the noclobber active).

```
>| all.txt
```

9. Use cat to display the **dates.txt** file. Then redirect **stdin** to display this file.

```
cat dates.txt
cat < dates.txt
```

10. Use **cat** to create a file listing tennis players, use **quit** as a **limit string**.

```
cat > tennis <<quit
Venus Williams
Serena Williams
Justine Henin
quit
```

Chapter 18

Filters

18.1 Filters

Programs, like **grep**, **cat**, **head**, **tail**, and many more, that take text input from **stdin** and give text output to **stdout** are called **filters** . These **filters** can be used to manipulate and transform text files into usable data files.

In this chapter we will take a look at a limited set of **filters** .

18.2 | pipe symbol

We use the pipe symbol | between two filters to use **stdout** from the first filter as **stdin** to the second filter. In the screenshot below we send the **stdout** from the **cat** command via the pipe symbol and as **stdin** to the **grep** command.

```
paul@debian10:~$ cat dates.txt | grep Turkey
 GB United Kingdom 1752-09-02      TR Turkey         1926-12-18
paul@debian10:~$
```

Note
Note that **stderr** is *not* passed through the | pipe symbol.

18.3 cat

The simplest filter of all is the **cat** command, because it does nothing. It takes **stdin** and puts it on **stdout**. Text is not changed when piping it through the **cat** command.

```
paul@debian10:~$ cat count.txt | cat | cat | cat | cat
one
two
three
four
paul@debian10:~$
```

18.4 tac

The **tac** filter also takes **stdin**, but it puts the last line first in **stdout**. It reverses the lines of a file.

```
paul@debian10:~$ cat count.txt | tac
four
three
two
one
paul@debian10:~$
```

18.5 tee

The **tee** filter takes **stdin** and creates two outputs, one to **stdout** and one to a file. This filter can help in troubleshooting long command lines with many pipes, or just to keep intermediate results of a complex text manipulation.

```
paul@debian10:~$ cat count.txt | tac | tee temp.txt | tac
one
two
three
four
paul@debian10:~$ cat temp.txt
four
three
two
one
paul@debian10:~$
```

Note
Note that both **cat** and **tee** do not change anything in the pipe line.

18.6 grep

The **grep** filter can select lines that contain a certain **string**. For example in the screenshot below we filter for **Belgium**. Note that **grep** is case sensitive by default.

```
paul@debian10:~$ cat dates.txt | grep Belgium
 BE Belgium        1582-12-14      LN Latin          9999-05-31
paul@debian10:~$ cat dates.txt | grep belgium
paul@debian10:~$
```

You can force **grep** to be case insensitive by using the **-i** option.

```
paul@debian10:~$ cat dates.txt | grep -i belgium
 BE Belgium        1582-12-14      LN Latin          9999-05-31
paul@debian10:~$ cat dates.txt | grep -i CHINA
 CN China          1911-12-18      NO Norway         1700-02-18
paul@debian10:~$
```

And you can select lines **not** containing a certain string using the **-v** option.

```
paul@debian10:~$ head -3 dates.txt
 AL Albania        1912-11-30      IT Italy          1582-10-04
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
paul@debian10:~$ head -3 dates.txt | grep -v Italy
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
paul@debian10:~$
```

And you can, of course, combine the two options.

```
paul@debian10:~$ head -3 dates.txt | grep -iv ITALY
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
paul@debian10:~$ head -3 dates.txt | grep -v -i italy
 AT Austria        1583-10-05      JP Japan          1918-12-18
 AU Australia      1752-09-02      LI Lithuania      1918-02-01
paul@debian10:~$
```

The **grep** command is often used to search for text in **log files**, and then it can be useful to have some context. You can use the **-A** option to add lines following the grepped string. The screenshot below uses **grep** to find **Canada** and the two lines following right behind the line containing **Canada**.

```
paul@debian10:~$ grep -A2 Canada dates.txt
 CA Canada          1752-09-02      LV Latvia          1918-02-01
 CH Switzerland     1655-02-28      NL Netherlands     1582-12-14
 CN China           1911-12-18      NO Norway          1700-02-18
paul@debian10:~$
```

Similarly you can **grep** for lines **before** a certain string using the **-B** option. This screenshot uses **grep** to find **Canada** and the three lines before the line with **Canada**.

```
paul@debian10:~$ grep -B3 Canada dates.txt
 AU Australia       1752-09-02      LI Lithuania       1918-02-01
 BE Belgium         1582-12-14      LN Latin           9999-05-31
 BG Bulgaria        1916-03-18      LU Luxembourg      1582-12-14
 CA Canada          1752-09-02      LV Latvia          1918-02-01
paul@debian10:~$
```

You can combine the **-A** and **-B** options, but you can also use **-C** (which stands for **c**ontext). Using **-C** gives you a number of lines before and after the searched string.

```
paul@debian10:~$ grep -C1 Spain dates.txt
 DK Denmark         1700-02-18      RO Romania         1919-03-31
 ES Spain           1582-10-04      RU Russia          1918-01-31
 FI Finland         1753-02-17      SI Slovenia        1919-03-04
paul@debian10:~$
```

We will come back to the **grep** command in the **regular expressions** chapter.

18.7 cut

The **cut** filter can select columns from a text file. Columns can be delimited by many different characters, or even by position in the text file. We use a file called **tennis** which can be downloaded from linux-training.be (wget http://linux-training.be/tennis).

In the screenshot below we use the **cut** filter with a comma as **delimiter -d,** to receive the text of each line until the first comma **-f1** . So the **-d** is short for **delimiter** and the **-f** is short for **field**.

```
paul@debian10:~$ cat tennis
Venus Williams, USA
Justine Henin, Bel
Serena Williams, Usa
Martina Hingis, SUI
Kim Clijsters, Bel
paul@debian10:~$ cat tennis | cut -d, -f1
Venus Williams
Justine Henin
Serena Williams
Martina Hingis
Kim Clijsters
paul@debian10:~$
```

Using **-f2** we get the field right behind the first comma, as is shown in this screenshot.

```
paul@debian10:~$ cat tennis | cut -d, -f2
 USA
 Bel
 Usa
 SUI
 Bel
paul@debian10:~$
```

Instead of the comma as **delimiter** we can also use a space (remember to escape or quote the space because the shell removes all spaces from the command line). In the screenshot below we ask for the third field.

```
paul@debian10:~$ cat tennis | cut -d" " -f3
USA
Bel
Usa
SUI
Bel
paul@debian10:~$
```

Using spaces to **cut** the **dates.txt** file is not very useful (try if you like to get the country names out of it, this fails for United *space* Kingdom). Luckily we can **cut** by position. In the example below we cut **byte** (or character) 5 to 19 from the **dates.txt** file.

```
paul@debian10:~$ cat dates.txt | cut -b 5-19 | tail -4
United Kingdom
Greece
Hungary
Iceland
paul@debian10:~$
```

Note
The **tail** filter is here to limit the size of the screenshots, you don't need to type it in your terminal window.

We can cut both **country** columns by counting the characters in the file. See this example, note that we keep **two** spaces between the country columns.

```
paul@debian10:~$ cat dates.txt | cut -b 5-19,38-52 | tail -4
United Kingdom  Turkey
Greece          United States
Hungary         Yugoslavia
Iceland
paul@debian10:~$
```

18.8 paste

The complement of **cut** is **paste**. The **paste** filter can take separate files (or text streams) and join them together by columns. By default **paste** will put a **tab** between columns, as can be seen in this screenshot.

```
paul@debian10:~$ cat music.txt | head -2
Queen
Abba
paul@debian10:~$ cat tennis | head -2
Venus Williams, USA
Justine Henin, Bel
paul@debian10:~$ paste music.txt tennis | head -2
Queen   Venus Williams, USA
Abba    Justine Henin, Bel
paul@debian10:~$
```

The **delimiter** can be adjusted to any ASCII character. In this example we use the comma.

```
paul@debian10:~$ paste -d, music.txt tennis | head -2
Queen,Venus Williams, USA
Abba,Justine Henin, Bel
paul@debian10:~$
```

18.9 tr

The **tr** filter can **translate** characters. In the example below we first translate all comma's to a space, and the second **tr** translates all spaces to comma's.

```
paul@debian10:~$ cat tennis
Venus Williams, USA
Justine Henin, Bel
Serena Williams, Usa
Martina Hingis, SUI
Kim Clijsters, Bel
paul@debian10:~$ cat tennis | tr ',' ' '
Venus Williams  USA
Justine Henin  Bel
Serena Williams  Usa
Martina Hingis  SUI
Kim Clijsters  Bel
paul@debian10:~$ cat tennis | tr ' ' ','
Venus,Williams,,USA
Justine,Henin,,Bel
Serena,Williams,,Usa
Martina,Hingis,,SUI
Kim,Clijsters,,Bel
paul@debian10:~$
```

We can define ranges to translate with **tr**, for example to convert all letters to uppercase as shown in this example.

```
paul@debian10:~$ cat tennis | tr 'a-z' 'A-Z'
VENUS WILLIAMS, USA
JUSTINE HENIN, BEL
SERENA WILLIAMS, USA
MARTINA HINGIS, SUI
KIM CLIJSTERS, BEL
paul@debian10:~$
```

And **tr** can also be used to **squeeze** consecutive characters to one character. In the example below we make sure that consecutive spaces are reduced to one space.

```
paul@debian10:~$ cat tennis | tr ',' ' ' | tr -s ' '
Venus Williams USA
Justine Henin Bel
Serena Williams Usa
Martina Hingis SUI
Kim Clijsters Bel
paul@debian10:~$
```

And **tr** can also **delete** characters from a file, as shown here.

```
paul@debian10:~$ cat tennis | tr -d ,
Venus Williams USA
Justine Henin Bel
Serena Williams Usa
Martina Hingis SUI
Kim Clijsters Bel
paul@debian10:~$
```

18.10 wc

The **wc** filter is a simple command that can count the number of words, lines or characters in a file. We use the **all.txt** and **error.txt** files from **find / > all.txt 2> error.txt**.

In the first example we count the number of lines in **all.txt**, using the **-l** option.

```
paul@debian10:~$ find / > all.txt 2> error.txt
paul@debian10:~$ cat all.txt | wc -l
73573
paul@debian10:~$
```

And here we count the number of words in a text stream, using the **-w** option.

```
paul@debian10:~$ cat tennis | tr -d , | wc -w
15
paul@debian10:~$
```

18.11 sort

We can **sort** words according to the alphabet using the **sort** command. In this example we **sort** each line.

```
paul@debian10:~$ cat tennis | sort
Justine Henin, Bel
Kim Clijsters, Bel
Martina Hingis, SUI
Serena Williams, Usa
Venus Williams, USA
paul@debian10:~$
```

This looks correct, but maybe we want to sort on the second column using the **-k** option.

```
paul@debian10:~$ cat tennis | sort -k2
Kim Clijsters, Bel
Justine Henin, Bel
Martina Hingis, SUI
Serena Williams, Usa
Venus Williams, USA
paul@debian10:~$
```

Consider the following sort on the numbers column in this **cities** file.

```
paul@debian10:~$ wget http://linux-training.be/cities
paul@debian10:~$ cat cities
Brussels, 5000
Shanghai, 200000
Antwerp, 40000
LA, 600000
Perth, 30000
paul@debian10:~$ cat cities | sort -k2
Shanghai, 200000
Perth, 30000
Antwerp, 40000
Brussels, 5000
LA, 600000
paul@debian10:~$
```

Maybe that alphabetical **sort** on numbers is not what we want. So let us do a **numerical sort** using the **-n** option.

```
paul@debian10:~$ cat cities | sort -n -k2
Brussels, 5000
Perth, 30000
Antwerp, 40000
Shanghai, 200000
LA, 600000
paul@debian10:~$
```

Let us expand the **music.txt** file so we have a double.

```
paul@debian10:~$ cat >>music.txt <<stop
> Abba
> Led Zeppelin
> stop
paul@debian10:~$ cat music.txt
Queen
Abba
Bowie
Abba
Led Zeppelin
paul@debian10:~$
```

Now we can **sort** it, and then **sort** and output unique **-u** values.

```
paul@debian10:~$ cat music.txt | sort
Abba
Abba
Bowie
Led Zeppelin
Queen
paul@debian10:~$ cat music.txt | sort -u
Abba
Bowie
Led Zeppelin
Queen
paul@debian10:~$
```

18.12 uniq

We can also use the **uniq** command instead of **sort -u**. But **uniq** only works on a sorted list.

```
paul@debian10:~$ cat music.txt | sort | uniq
Abba
Bowie
Led Zeppelin
Queen
paul@debian10:~$
```

And with **uniq** we can also **count -c** the number of occurrences of each name.

```
paul@debian10:~$ cat music.txt | sort | uniq -c
      2 Abba
      1 Bowie
      1 Led Zeppelin
      1 Queen
paul@debian10:~$
```

18.13 comm

The last command of this chapter is **comm**, it compares two sorted files (or one file and stdin). In the screenshot below we first **sort** and **uniq** the music.txt file and then compare it to the sorted music2.txt file.

Note
The **-** after the **comm** command refers to **stdin** .

```
paul@debian10:~$ cat music.txt
Queen
Abba
Bowie
Abba
Led Zeppelin
paul@debian10:~$ cat music2.txt
Abba
Bowie
Led Zeppelin
Sting
paul@debian10:~$ cat music.txt | sort | uniq | comm - music2.txt
                Abba
                Bowie
                Led Zeppelin
Queen
        Sting
paul@debian10:~$
```

The result of the **comm** command are three columns; the first column contains lines unique to **stdin** , the second column are lines unique to **music2.txt** and the third column are lines that appear in both columns.

18.14 What about sed?

The **stream editor sed** is indeed a commonly used filter on Linux, we discuss how to use **sed** in the **regular expressions** chapter.

18.15 Cheat sheet

Table 18.1: Filters

command	**explanation**
cat	copy **stdin** to **stdout**
tac	reverse of **cat**, last line first
tee	copy **stdin** to a file and to **stdout**
grep	filter lines from **stdin** or a file that contain a string
grep	filter lines that match a regular expression (see later)
grep -i	case insensitive filter
grep -v	filter the lines not matching a string/regex
grep -A2	filter the lines and also output the two next (After) lines
grep -B2	filter the lines and also output the two previous (Before) lines
grep -C2	filter the lines and also output the two next and previous (Context) lines
cut	select columns from **stdin** or from a file
cut -dx -f2	separate columns by x and display the second column (field)
cut -b5-42	display the fifth till 42nd character of each line
paste	join lines by column (the reverse of cut)
tr	translate characters
tr a b	translate each a to a b (from **stdin** to **stdout**)
tr 'a-z' 'A-Z'	translate text to uppercase
tr -d x	remove all occurrences of x
wc	count lines, words or characters
sort	sort by alphabet (or by ASCII value)
sort -n	numerical sort
sort -u	remove doubles (and triples etc) from a list
uniq	remove doubles from a sorted list
comm	compare sorted lists

18.16 Practice

1. Do a case insensitive **grep** for 'Bel' in the **tennis** file.
2. Filter the lines containing 'Henin' and the line before and after from the **tennis** file.
3. Filter the surname column from the **tennis** file.
4. Again filter the surname from the **tennis** file, but remove the comma's.
5. Filter the dates from the **dates.txt** file and put each date on a separate line.
6. Count the number of files and directories in **/etc** .
7. Sort the **cities** file and put the result in **sorted_cities.txt**
8. Knowing that **/usr/share/dict/words** is a dictionary, write a simple spell checker on the command line.

18.17 Solution

1. Do a case insensitive **grep** for 'Bel' in the **tennis** file.

```
grep -i Bel tennis
```

2. Filter the lines containing 'Henin' and the line before and after from the **tennis** file.

```
cat tennis | grep -C1 Henin
```

3. Filter the surname column from the **tennis** file.

```
cut -d\' ' -f2 tennis
```

4. Again filter the surname from the **tennis** file, but remove the comma's.

```
cat tennis | cut -d\  -f2 | tr -d ,
```

5. Filter the dates from the **dates.txt** file and put each date on a separate line.

```
cat dates.txt | cut -b 20-29,53-63 | tr ' ' '\n'
```

6. Count the number of files and directories in **/etc** .

```
ls /etc | wc -l
```

7. Sort the **cities** file and put the result in **sorted_cities.txt**

```
sort cities > sorted_cities.txt
```

8. Knowing that **/usr/share/dict/words** is a dictionary, write a simple spell checker on the command line.

```
echo "The zun is shining today!" > text
sort /usr/share/dict/words > dict
cat text | tr -d '!?' | tr 'A-Z' 'a-z' | tr ' ' '\n' | sort | uniq | comm -23 - dict
```

Chapter 19

Regular expressions

19.1 What are regular expressions?

A regular expression is a string of characters defining a search pattern. A lot of info can be found on the Wikipedia page https://en.wikipedia.org/wiki/Regular_expression . In this chapter we discuss commands that use regular expressions and learn by example.

Regular expressions sometimes use special characters like a **$** or a ***** or a **?**. To prevent the shell from interpreting these characters we will enclose the regular expression in single quotes when necessary. To play safe, you may choose to always enclose a **regex** in single quotes.

19.2 grep

We have already discussed **grep** in the previous chapters. This time we will take it a step further by providing examples of **grep** with regular expressions.

19.2.1 ^ starts with

When using regular expressions the ^ sign means **starts with**. In the screenshot below we **grep** for all lines starting with an uppercase letter **K**.

```
paul@debian10:~$ cat tennis
Venus Williams, USA
Justine Henin, Bel
Serena Williams, Usa
Martina Hingis, SUI
Kim Clijsters, Bel
paul@debian10:~$ cat tennis | grep ^K
Kim Clijsters, Bel
paul@debian10:~$
```

19.2.2 $ ends with

The dollar sign can be used to search for lines that **end with** a string of letters. Here we ask for all lines ending with **sa**.

```
paul@debian10:~$ cat tennis | grep sa$
Serena Williams, Usa
```

Remember that **grep** has the **-i** option for a case insensitive search? Well this can still be used, as seen in this example.

```
paul@debian10:~$ cat tennis | grep -i sa$
Venus Williams, USA
Serena Williams, Usa
paul@debian10:~$
```

19.2.3 ^ $ Match the whole line

When you want to **grep** for a whole line, then you can start your regex with the ^ and end it with the **$**. Remember to escape the spaces!

```
paul@debian10:~$ cat tennis | grep ^Justine\ Henin,\ Bel$
Justine Henin, Bel
paul@debian10:~$
```

19.2.4 . single character

The dot **.** matches any single character. This includes punctuation characters. In the example below we need to match the exact number of characters between **Ve** and **sa**.

```
paul@debian10:~$ cat tennis | grep -i ^Ve.............sa$
Venus Williams, USA
```

One dot less, and there is no match.

```
paul@debian10:~$ cat tennis | grep -i ^Ve............sa$
paul@debian10:~$
```

One dot too many, and again there is no match.

```
paul@debian10:~$ cat tennis | grep -i ^Ve..............sa$
paul@debian10:~$
```

19.2.5 * zero, one, or more

In the example below we put a ***** behind the **.** meaning the previous (this is the .) any number of times. So we read this **regex** as **starts with Ve, then any number of characters, then ends with sa**.

```
paul@debian10:~$ cat tennis | grep -i ^Ve.*sa$
Venus Williams, USA
paul@debian10:~$
```

19.2.6 [ ] single character

Just like with **file globbing** the **[ ]** notation denotes a single character, with the list of characters between the square brackets. In the example below we ask for exactly three characters at the end, the last character being uppercase A or lowercase a.

```
paul@debian10:~$ cat tennis | grep U[sS][Aa]$
Venus Williams, USA
Serena Williams, Usa
paul@debian10:~$
```

19.2.7 [^] not

Remember that the ^ meant **at the start of the line**, well enclosed in square brackets the ^ means **not this list**. In the example below we ask for no uppercase A or I at the end of the line.

```
paul@debian10:~$ cat tennis | grep [^AI]$
Justine Henin, Bel
Serena Williams, Usa
Kim Clijsters, Bel
paul@debian10:~$
```

Here is another example of the **not** ^ in combination with the **at the start of** ^ . The following **regex** can be read as **at the start of the line, not an A or a** as first character and **not an A or e** as the second character.

```
paul@debian10:~$ cat cities
Brussels, 5000
Shanghai, 200000
Antwerp, 40000
LA, 600000
Perth, 30000
```

```
paul@debian10:~$ cat cities | grep ^[^Aa][^Ae]
Brussels, 5000
Shanghai, 200000
paul@debian10:~$
```

19.2.8 [a-z] [0-9] series

Just like with **file globbing** we can put series inside the square brackets. In the example below we ask for no uppercase letter at the end of the line.

```
paul@debian10:~$ cat tennis | grep [^A-Z]$
Justine Henin, Bel
Serena Williams, Usa
Kim Clijsters, Bel
paul@debian10:~$
```

19.3 rename

Warning
This section is about the **rename** command on Debian and derivatives like Ubuntu, Mint and Raspbian. The **rename** command on RHEL and CentOS is a completely different story.

The **rename** command in Debian 10 is not installed by default, the **root** user can install it with **apt-get install rename**. The **rename** command is installed as a symbolic link to a symbolic link to a **Perl script**.

```
paul@debian10:~$ file $(readlink -f $(which rename))
/usr/bin/file-rename: Perl script text executable
paul@debian10:~$
```

Note
Links and **readlink -f** will be explained in the Links chapter.

In this section we will use a **renamedir** in our home directory containing the following files.

```
paul@debian10:~$ mkdir renamedir
paul@debian10:~$ cp error.txt music*.txt dates.txt renamedir/
paul@debian10:~$ cd renamedir/
paul@debian10:~/renamedir$ ls
dates.txt  error.txt  music2.txt  music.txt
paul@debian10:~/renamedir$
```

19.3.1 replace string

The **rename** command is often used to replace a string of characters in file names to another string. For example this screenshot where we replace **music** with **artist** for all files matching the ***.txt** glob.

```
paul@debian10:~/renamedir$ rename 's/music/artist/' *.txt
paul@debian10:~/renamedir$ ls
artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$
```

Remember the **bash shell** is expanding this glob, **rename** receives all the .txt files and verifies for each one whether it can do the **artist-to-music** switch.

```
paul@debian10:~/renamedir$ ls
dates.txt  error.txt  music2.txt  music.txt
paul@debian10:~/renamedir$ set -x
paul@debian10:~/renamedir$ rename 's/music/artist/' *.txt
+ rename s/music/artist/ dates.txt error.txt music2.txt music.txt
paul@debian10:~/renamedir$ set +x
+ set +x
paul@debian10:~/renamedir$ ls
artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$
```

19.3.2 Renaming file 'extensions'

File extensions have no meaning for the **bash** shell, but can be informative to end users. In this section we explain how to **rename** file extensions.

At first this seems simple, in the example below we change all **.txt** files to **.TXT**.

```
paul@debian10:~/renamedir$ ls
artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$ rename 's/txt/TXT/' *.txt
paul@debian10:~/renamedir$ ls
artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$
```

Except that the above example is wrong. Can you find the mistake?

The mistake becomes clear when we add the alltxt.txt file, as shown in the following screenshot.

```
paul@debian10:~/renamedir$ ls
artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$ touch alltxt.txt
paul@debian10:~/renamedir$ rename 's/txt/TXT/' *.txt
paul@debian10:~/renamedir$ ls
allTXT.txt  artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$
```

To rename extensions we will have to look at replacing strings **at the end** of a filename. For this we can use the **dollar sign $** as can be seen in this screenshot.

```
paul@debian10:~/renamedir$ rename 's/TXT/txt/' *
paul@debian10:~/renamedir$ ls
alltxt.txt  artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$ rename 's/txt$/TXT/' *.txt
paul@debian10:~/renamedir$ ls
alltxt.TXT  artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$
```

19.3.3 global rename

The **s/string/replace/** will only replace the first occurrence of **string** in **replace**, as can be seen in this example. Note the second **rename** command and the resulting **allTXT.txt** file.

```
paul@debian10:~/renamedir$ rename 's/TXT$/txt/' *
paul@debian10:~/renamedir$ ls
alltxt.txt  artist2.txt  artist.txt  dates.txt  error.txt
```

```
paul@debian10:~/renamedir$ rename 's/txt/TXT/' *
paul@debian10:~/renamedir$ ls
allTXT.txt  artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$
```

You can use **rename** to **globally** replace all occurrences of the **string** using the **/g** at the end of the regex.

```
paul@debian10:~/renamedir$ ls
alltxt.txt  artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$ rename 's/txt/TXT/g' *
paul@debian10:~/renamedir$ ls
allTXT.TXT  artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$
```

19.3.4 case insensitive replace

You can have **rename** ignore the case of the searched **string** by adding a **/i** at the end of the regex. (This can be in combination with /g and then becomes /gi).

In the screenshot below we **rename** all the .txt .TXT .Txt and variations to **.text**.

```
paul@debian10:~/renamedir$ ls
allTXT.TXT  artist2.TXT  artist.TXT  dates.TXT  error.TXT
paul@debian10:~/renamedir$ mv artist.TXT artist.txt
paul@debian10:~/renamedir$ rename 's/txt$/text/i' *
paul@debian10:~/renamedir$ ls
allTXT.text  artist2.text  artist.text  dates.text  error.text
paul@debian10:~/renamedir$
```

19.3.5 a dry run

You can test a rename command before executing it by performing a dry run using the **-n** option. It will then display the names instead of renaming files.

```
paul@debian10:~/renamedir$ ls
allTXT.txt  artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$ rename -n 's/txt$/TXT/' *
rename(allTXT.txt, allTXT.TXT)
rename(artist2.txt, artist2.TXT)
rename(artist.txt, artist.TXT)
rename(dates.txt, dates.TXT)
rename(error.txt, error.TXT)
paul@debian10:~/renamedir$
```

19.3.6 removing extensions

Until now, we have been ignoring the **.** part of a file 'extension'. We cannot simply use a **.** in the expression since that means **any character**. Luckily we can escape the **.** and thus make it a literal **.** , as seen in the example below where we remove **.txt** from all files.

```
paul@debian10:~/renamedir$ mv allTXT.txt alltxt
paul@debian10:~/renamedir$ ls
alltxt  artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$ rename 's/\.txt$//' *
paul@debian10:~/renamedir$ ls
alltxt  artist  artist2  dates  error
paul@debian10:~/renamedir$
```

19.3.7 adding extensions

Files without an extension can have one added **at the end** (represented by a $ sign). See this screenshot for an example on how to do this.

```
paul@debian10:~/renamedir$ ls
alltxt  artist  artist2  dates  error
paul@debian10:~/renamedir$ rename 's/$/.txt/' *
paul@debian10:~/renamedir$ ls
alltxt.txt  artist2.txt  artist.txt  dates.txt  error.txt
paul@debian10:~/renamedir$
```

19.4 sed

The **sed** command is a **stream editor**, it is created to edit a stream of text. This first screenshot will look familiar (if you just studied **rename**).

```
paul@debian10:~$ echo Sunday | sed 's/Sun/Mon/'
Monday
paul@debian10:~$
```

The forward slashes in the above example can be replaced by colons, underscores, vertical bars or commas to improve readability of the **regex**.

```
paul@debian10:~$ echo Sunday | sed 's:Sun:Mon:'
Monday
paul@debian10:~$ echo Sunday | sed 's|Sun|Mon|'
Monday
paul@debian10:~$ echo Sunday | sed 's_Sun_Mon_'
Monday
paul@debian10:~$ echo Sunday | sed 's,Sun,Mon,'
Monday
paul@debian10:~$
```

The **search** and **replace** strings do not need to be the same length.

```
paul@debian10:~$ cat tennis | sed 's/ SUI/ Switzerland/'
Venus Williams, USA
Justine Henin, Bel
Serena Williams, Usa
Martina Hingis, Switzerland
Kim Clijsters, Bel
paul@debian10:~$
```

19.4.1 interactive sed

While **sed** was designed as a stream editor, it can be used **in-place** using the **-i** option, as seen in this screenshot.

```
paul@debian10:~$ echo Sunday > today
paul@debian10:~$ cat today
Sunday
paul@debian10:~$ sed –i 's/Sun/Mon/' today
paul@debian10:~$ cat today
Monday
paul@debian10:~$
```

19.4.2 global /g

You may want **sed** to replace the **regex** in the whole stream (instead of just the first occurrence of each line). To do this, apply the **/g** option.

```
paul@debian10:~$ cat tennis | sed 's/i/XX/g'
Venus WXXllXXams, USA
JustXXne HenXXn, Bel
Serena WXXllXXams, Usa
MartXXna HXXngXXs, SUI
KXXm ClXXjsters, Bel
paul@debian10:~$
```

19.4.3 case insensitive /i

For a **case insensitive** replace, use the **/i** option (here together with **/g**).

```
paul@debian10:~$ cat tennis | sed 's/ USA/ United States/gi'
Venus Williams, United States
Justine Henin, Bel
Serena Williams, United States
Martina Hingis, SUI
Kim Clijsters, Bel
paul@debian10:~$
```

19.4.4 remove white space

A common use of **sed** is to remove spaces, in this screenshot at the beginning of every line. Remember to escape the space character. Read the **regex** as **beginning of the line**, **space**, **any number of times**, and replace with nothing.

```
paul@debian10:~$ cat > white
     There are spaces before this line.
   And also before this one.
paul@debian10:~$ cat white
     There are spaces before this line.
   And also before this one.
paul@debian10:~$ cat white | sed 's/^\ *//'
There are spaces before this line.
And also before this one.
paul@debian10:~$
```

Tip
Instead of using \ to denote spaces, you can also use **\s** which includes spaces and tabs.

19.4.5 \(\) marked subexpressions

You can add escaped **parentheses** around part of your regex to mark it as a **block**. Using the **\n** notation you can refer to these **blocks** (using the numbers 1 to 9 in order). For example in this screenshot we mark **Sun** and recall it thrice with the **\1** notation.

```
paul@debian10:~$ echo Sunday | sed 's/\(Sun\)/\1\1\1/'
SunSunSunday
paul@debian10:~$
```

Remember the **dates.txt** file and the **cut** filter to get the countries from that file? We left two spaces between the countries, for a reason.

```
paul@debian10:~$ cat dates.txt | cut -b 5-19,38-52 | tail -5
France          Sweden
United Kingdom  Turkey
Greece          United States
Hungary         Yugoslavia
Iceland
paul@debian10:~$
```

Well we can use these two consecutive spaces followed by an uppercase letter and replace them with a comma (followed by that same uppercase letter). To do this we mark the search for that uppercase letter with **escaped parentheses** so we can refer to those letters with **\1** (and put them back right after the commas).

```
paul@debian10:~$ cat dates.txt | cut -b 5-19,38-52 | sed s/\ \ \([A-Z]\)/,\1/ | head -5
Albania       ,Italy
Austria       ,Japan
Australia     ,Lithuania
Belgium       ,Latin
Bulgaria      ,Luxembourg
paul@debian10:~$
```

In this screenshot we do the same thing, except that we put **newlines** instead of **commas**. So we get a country on each line.

```
paul@debian10:~$ cat dates.txt | cut -b 5-19,38-52 | sed s/\ \ \([A-Z]\)/\n\1/ | head -5
Albania
Italy
Austria
Japan
Australia
paul@debian10:~$
```

Here is the country list sorted. And we also removed the *useless use* of **cat** though some people find it easier to read with the **cat** command included.

```
paul@debian10:~$ cut -b 5-19,38-52 dates.txt | sed s/\ \ \([A-Z]\)/\n\1/ | sort | head -5
Albania
Australia
Austria
Belgium
Bulgaria
paul@debian10:~$
```

19.4.6 multiple back referencing

In this example we convert an American date format (MM-DD-YYYY) to an international date format which is YYYY-MM-DD.

```
paul@debian10:~$ echo 12-31-2019 | sed s/\(..\)-\(..\)-\(....\)/\3-\1-\2/
2019-12-31
paul@debian10:~$
```

19.5 bash history

The **bash shell** can also interpret regular expressions. In the screenshot below we use a **s/search/replace** on the previous command.

```
paul@debian10:~$ touch file1
paul@debian10:~$ !t:s/1/42/
touch file42
paul@debian10:~$ ls file*
file1  file42
paul@debian10:~$
```

This also works when repeating a command via its history number.

```
paul@debian10:~$ history 5
 1265  ls file*
 1266  touch file1
 1267  touch file42
 1268  ls file*
 1269  history 5
paul@debian10:~$ !1267:s/42/33/
touch file33
paul@debian10:~$ ls file*
file1  file33  file42
paul@debian10:~$
```

NOTE There are several other tools on Linux that use regular expressions, like **vi**, **awk** and **more** for example.

19.6 Cheat sheet

Table 19.1: Regular expressions

command	explanation
grep ^foo	grep for lines that start with **foo**
grep foo$	grep for lines that end with **foo**
grep ^foo$	grep for lines that contain just and exactly **foo**
grep ^.$	grep for lines containing a single character
grep ^...$	grep for lines containing exactly three characters
grep ^A.*	grep for lines starting with A, then zero one or more characters
grep [a2e]$	grep for lines ending with a single character, namely a or 2 or e
grep [^x7]$	grep for lines ending with a single character, but not x or 7
grep [^A-Z]$	not an uppercase letter at the end of the line
rename 's/foo/bar/' *	Rename all files(*) replacing the first occurrence of **foo** with *bar*.
rename 's/foo/bar/i' *	Rename all files(*) replacing the first occurrence of case insensitive **foo** with *bar*.
rename 's/foo/bar/g' *	Rename all files(*) replacing all occurrences of **foo** with *bar*.
rename 's/foo$/bar/' *	Rename all files replacing **foo** at the end of the filename with **bar**.
rename 's/$/.foo/' *	Rename all files adding **.foo** as an extension.
rename 's/\.foo//' *	Rename all files removing **.foo** as an extension.
rename -n	a dry run, just show what would be done
sed 's/foo/bar/'	replace first occurrence of **foo** with **bar** on each line
sed 's:foo:bar:'	replace first occurrence of **foo** with **bar** on each line
sed 's_foo_bar_'	replace first occurrence of **foo** with **bar** on each line
sed 's,foo,bar,'	replace first occurrence of **foo** with **bar** on each line
sed 's/foo/bar/g'	replace all occurrences of **foo** with **bar**
sed 's/foo/bar/gi'	replace all occurrences of case insensitive **foo** with **bar**
sed 's/^\ *//'	remove all white space at the start of each line
sed 's/\(foo\)/\1\1\1/'	mark **foo** as subexpression 1, repeat it three times
sed 's/\(foo\)\(bar\)/\2\1'	mark **foo** as subexpression 1, **bar** as subexpression 2, then reverse the two
sed -i regex foo	change the file **foo** (instead of writing to **stdout**)
!foo:s/33/42/	repeat the last command that starts with **foo** replacing 33 with 42
!42:s/foo/bar/	repeat command number 42, replacing **foo** with **bar**

19.7 Practice

1. List all lines of the **tennis** file that start with **Ma** .
2. List all lines of the **tennis** file that end with **UI** .
3. List all lines of the **tennis** file starting with **Se**, then any characters, then ending with **SA** .
4. List all lines of the **tennis** file **not** ending in **a** or **A**.
5. Rename all files ending in **.txt** to **.TXT** .
6. Use **sed** in a stream to change all vowels in the tennis file to X (output on the terminal).

19.8 Solution

1. List all lines of the **tennis** file that start with **Ma** .

```
grep ^Ma tennis
```

2. List all lines of the **tennis** file that end with **UI** .

```
grep 'UI$' tennis
```

3. List all lines of the **tennis** file starting with **Se**, then any characters, then ending with **SA** .

```
cat tennis | grep '^Se.*SA$'
```

4. List all lines of the **tennis** file **not** ending in **a** or **A**.

```
cat tennis | grep '[^aA]$'
```

5. Rename all files ending in **.txt** to **.TXT** .

```
rename 's/\.txt$/\.TXT/' *.txt
```

6. Use **sed** in a stream to change all vowels in the tennis file to X (output on the terminal).

```
cat tennis | sed 's/[aeiou]/X/g'
```

Chapter 20

Useful tools

20.1 find

The **find** command is very versatile, as you will *find* out in this section. In its simplest form the **find** command will list all files in the current directory.

```
paul@debian10:~$ find | more
.
./renamedir
./renamedir/alltxt
./renamedir/error
./renamedir/dates
./renamedir/artist
./renamedir/artist2
./white
./error.txt
./.bashrc
./cities
./textfile
./tennis
./Linux.pdf
./music.txt
./services.gz
./.bash_logout
./count.txt
./pictures
./wolf.png
--More--
```

20.1.1 find location

Usually administrators like to specify where **find** needs to look for files. This is the first argument to find in the examples below.

```
find .    --> searches in current directory and all subdirectories
find /etc --> searches in /etc (and all subdirectories)
find /tmp --> searches in /tmp (and all subdirectories)
find /    --> searches in the complete file system
```

20.1.2 find -name

The next argument for **find** that we will discuss is **-name**. Using this we can let find search for files with a specific name, as this screenshot shows.

```
paul@debian10:~$ find . -name dates.txt
./backup/dates.txt
./dates.txt
paul@debian10:~$
```

Wild cards can be used to find files by name, but not like this screenshot. Do you know why this wild card fails?

```
paul@debian10:~$ find . -name music*
find: paths must precede expression: music2.txt
find: possible unquoted pattern after predicate -name?
paul@debian10:~$
```

It fails because the **bash shell** is expanding the music* to list all the files that start with music. We need to quote the wild card so the shell does not touch it. In other words, we need to make sure that the **find** command receives the wild card. The next screenshot is much better.

```
paul@debian10:~$ find . -name 'music*'
./music.txt
./music2.txt
./music
paul@debian10:~$
```

20.1.3 find -type

We saw before that we can specify the file type for **files** or **directories** as the screenshot below shows.

```
paul@debian10:~$ find . -name 'music*' -type d
./music
paul@debian10:~$ find . -name 'music*' -type f
./music.txt
./music2.txt
paul@debian10:~$
```

20.1.4 find -newer

You can use **find** to search for files that are newer than a certain file with the **-newer** argument.

```
paul@debian10:~$ find . -name '*.txt' -newer music.txt
./all.txt
./music2.txt
paul@debian10:~$
```

20.1.5 find -exec

We can use find to execute a command on the files that it finds. We can for example search for all **.txt** files in our home directory and copy them in a temporary **/tmp/textfiles** directory.

```
paul@debian10:~$ mkdir /tmp/textfiles
paul@debian10:~$ find . -name '*.txt' -exec cp {} /tmp/textfiles/ \;
paul@debian10:~$ ls /tmp/textfiles/
5test.txt  combi.txt  dates.txt  error.txt  music2.txt  output.txt  test.txt
all.txt    count.txt  empty.txt  hello.txt  music.txt   temp.txt
paul@debian10:~$
```

The **-exec** ends at the **\;** . The **{ }** means **the file we just found**. So for every file that **find** finds that matches the name, it will execute a **cp** of that file to **/tmp/textfiles**. In this case the **cp** command was executed thirteen times.

20.1.6 2> /dev/null

When you execute a **find /** , then you will get a lot of **Permission denied** errors. You can mitigate this problem by redirecting **stderr** to a file, or even to **/dev/null**.

```
paul@debian10:~$ find / > all.txt 2> /dev/null
paul@debian10:~$
```

Nothing will appear on the screen when executing the above command, and all errors are immediately discarded. The *pseudo* device **/dev/null** is sometimes called **the black hole**, because it absorbs everything and nothing can be retrieved from it.

20.2 locate

The **locate** command is not installed by default on Debian 10, so ask an administrator to perform **apt-get install locate && updatedb** if you want to try this out. The **locate** command works much faster than **find** because it uses an index (created by the **updatedb** command).

In the screenshot below we ask **locate** for all files that contain **resolv.conf** in their name.

```
paul@debian10:~$ locate resolv.conf
/etc/resolv.conf
/usr/lib/systemd/resolv.conf
/usr/share/man/man5/resolv.conf.5.gz
paul@debian10:~$
```

The **locate** command doesn't know about recently created files, because the index is only updated once a day.

```
paul@debian10:~$ touch haha
paul@debian10:~$ locate haha
paul@debian10:~$
```

20.3 watch

The **watch** command will by default execute a command every two seconds. This allows you to **watch** for changes in the command's output. In the screenshot below we execute the **ls -l *.txt** command every two seconds.

```
paul@debian10:~$ watch ls -l *.txt
paul@debian10:~$
Every 2.0s: ls -l all.txt combi.txt count.txt dates.txt e...  debian10: Mon Aug  5 05:28:38 ↩
     2019

-rw-r--r-- 1 paul paul 2900463 Aug  5 04:57 all.txt
-rw-r--r-- 1 paul paul     198 Aug  3 21:09 combi.txt
-rw-r--r-- 1 paul paul      19 Jul 27 23:42 count.txt
-rw-r--r-- 1 paul paul    1118 Jul 27 13:12 dates.txt
-rw-r--r-- 1 paul paul       0 Aug  3 21:45 empty.txt
-rw-r--r-- 1 paul paul   31537 Aug  4 14:59 error.txt
-rw-r--r-- 1 paul paul      24 Jul 28 13:56 hello.txt
-rw-r--r-- 1 paul paul      30 Aug  4 16:02 music2.txt
-rw-r--r-- 1 paul paul      35 Aug  4 15:29 music.txt
-rw-r--r-- 1 paul paul       6 Aug  3 20:57 output.txt
-rw-r--r-- 1 paul paul      19 Aug  4 13:15 temp.txt
-rw-r--r-- 1 paul paul 2062776 Jul 28 14:48 test.txt
```

You can change the refresh by add **-n seconds** to the **watch** command. Quit the **watch** command with the **Ctrl-c** keyboard interrupt.

20.4 time

The **time** utility tells you how long it took to execute a command. This means you don't have to wait for a command to finish to know how long it took. This can be useful to schedule intensive jobs like backups.

In the screenshot below we measure the time it takes to **gzip** a text file. It took 0.062 real seconds (most of which was in user space).

```
paul@debian10:~$ find / > all.txt 2>/dev/null
paul@debian10:~$ time gzip all.txt
```

```
real    0m0.062s
user    0m0.058s
sys     0m0.004s
paul@debian10:~$
```

We mentioned before that **bzip2** takes longer than **gzip**. With the **time** command you can compare them. We will revisit this when discussing backups!

```
paul@debian10:~$ time bzip2 all.txt

real    0m0.293s
user    0m0.269s
sys     0m0.022s
paul@debian10:~$
```

20.5 date

Let's revisit the **date** command that we saw in one of the first chapters. We start with **date +%s** which shows us the count of seconds since the beginning of time.

```
paul@debian10:~$ date +%s
1564977087
paul@debian10:~$ date +%s
1564977110
paul@debian10:~$
```

In the Linux world, time began on January 1st 1970. The first second of that day, is second number 0 for Linux. This is why we have a Year 2038 problem, because then this signed 32-bit integer number will exceed 2147483647, which it cannot (at least not in 32 bits). You can read more about this here https://en.wikipedia.org/wiki/Year_2038_problem .

The **date** string can be customised to the format of your choice. The format is described in the man page and comes after a **+** sign.

```
paul@debian10:~$ date
Mon 05 Aug 2019 06:56:07 AM CEST
paul@debian10:~$ date +'%A %d-%m-%Y'
Monday 05-08-2019
paul@debian10:~$
```

20.6 od

The **od** command, short for **o**ctal **d**isplay, is able to print the content of files in different formats. Consider the file **test** which contains the ASCII characters 0 to 9 followed by a newline.

```
paul@debian10:~$ echo 0123456789 > test
paul@debian10:~$ cat test
0123456789
paul@debian10:~$ od -c test
0000000   0   1   2   3   4   5   6   7   8   9  \n
0000013
paul@debian10:~$ od -b test
0000000 060 061 062 063 064 065 066 067 070 071 012
0000013
paul@debian10:~$ od -x test
0000000 3130 3332 3534 3736 3938 000a
0000013
paul@debian10:~$
```

The **od -c** command will show the ASCII characters inside the file, including the **newline** at the end. With **od -b** we see the **octal** numbers corresponding to those characters and for the **newline** (012). And with **od -x** we see the same data in **hexadecimal** format (in little endian).

Unicode characters require two bytes per character, as can be seen in this screenshot.

```
paul@debian10:~$ <b>cat test</b>
μωΩπρφετη
paul@debian10:~$ <b>od -b test</b>
0000000 316 274 317 211 316 251 317 200 317 201 317 206 316 265 317 204
0000020 316 267 012
0000026
paul@debian10:~$
```

20.7 dd

The **dd** command will copy bytes from one file to another. Since everything on Linux is a file (see the chapter), this means that this command can copy anything to anything.

For example in this screenshot we use **/dev/zero** (an endless source of zeroes) to create a file of 1024 bytes, with each byte set to zero.

```
paul@debian10:~$ dd if=/dev/zero of=test bs=1024 count=1
1+0 records in
1+0 records out
1024 bytes (1.0 kB, 1.0 KiB) copied, 0.000525708 s, 1.9 MB/s
paul@debian10:~$
```

The **if** means **i**nput **f**ile, **of** is **o**utput **f**ile and **bs** is **b**lock **s**ize. We can verify with **od** that all bytes are set to zero.

```
paul@debian10:~$ od -b test
0000000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
*
0002000
paul@debian10:~$
```

With **root** privileges you can erase a complete disk using **dd**. Not even forensic tools can recover the data when executing the following command.

```
dd if=/dev/zero of=/dev/sdb
```

Note
We will discuss hard disks in the storage chapters.

In the example below we use **dd** to create an ISO file from a Compact Disc. When not giving it a **bs** and **count** then **dd** will copy until it gets an **EOF** byte.

```
paul@debian10:~$ dd if=/dev/cdrom of=cd.iso
684032+0 records in
684032+0 records out
350224384 bytes (350 MB, 334 MiB) copied, 2.04172 s, 172 MB/s
paul@debian10:~$
```

The **dd** tool can be used to copy a file, just like **cp** it requires a source and a destination (input file and output file).

```
paul@debian10:~$ dd if=services.gz of=copy.gz
14+1 records in
14+1 records out
7307 bytes (7.3 kB, 7.1 KiB) copied, 0.000486533 s, 15.0 MB/s
paul@debian10:~$
```

20.8 bc

The **bc** command, which is short for **b**asic **c**alculator, is not installed by default. Ask an administrator to execute **apt-get install bc** if you want to try this.

In short you can give calculations to **bc** and it will output the answer.

```
paul@debian10:~$ echo 4+3 | bc
7
paul@debian10:~$
```

If you want answers that are more precise than integer numbers, then use the **scale=** statement.

```
paul@debian10:~$ echo "42.2 / 33.3" | bc
1
paul@debian10:~$ echo "scale=2; 42.2 / 33.3" | bc
1.26
paul@debian10:~$
```

The **bc** command can work with *extremely* large numbers, much larger than common calculators can. See for example this question for 10 to the power of 128. (10 to the power 4096 also works).

```
paul@debian10:~$ echo 10^128 | bc
100000000000000000000000000000000000000000000000000000000000000000000000000000000\
000000000000000000000000000000000000000000000000000000000000000000000000
paul@debian10:~$
```

You can use C-style syntax when working with variables and **bc**, as shown in this example.

```
paul@debian10:~$ echo "var=10;++var;--var" | bc
11
10
paul@debian10:~$
```

20.9 sleep

The **sleep** command is probably the simplest command of all. You give it an argument in seconds and it will do nothing for that amount of seconds.

```
paul@debian10:~$ sleep 3
paul@debian10:~$
```

20.10 script

The **script** tool allows you to record sessions in your terminal window. In the screenshot below we start the **script** tool by typing **script** and end it with the **exit** command.

```
paul@debian10:~$ script
Script started, file is typescript
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 paul paul 1118 Jul 27 13:12 dates.txt
paul@debian10:~$ cp Linux.pdf Fun.pdf
paul@debian10:~$ exit
exit
Script done, file is typescript
paul@debian10:~$
```

Use **cat** to see the recorded session.

```
paul@debian10:~$ cat typescript
Script started on 2019-08-08 15:35:25+02:00 [TERM="screen-256color" TTY="/dev/pts/0"  ↩
    COLUMNS="96" LINES="29"]
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 paul paul 1118 Jul 27 13:12 dates.txt
paul@debian10:~$ cp Linux.pdf Fun.pdf
paul@debian10:~$ exit
exit

Script done on 2019-08-08 15:35:44+02:00 [COMMAND_EXIT_CODE="0"]
paul@debian10:~$
```

20.11 tmux

To finish this chapter I have to mention **tmux**. Personally I use it all the time, but it can be a lot to learn for people that are new to Linux. **tmux** is a **terminal multiplexer**, which means it allows for many terminals inside one terminal. The **root** user can install it with **apt-get install tmux**.

By default you can send commands to **tmux** using Ctrl-b. For example **Ctrl-b-"** and **Ctrl-b-%** will split the screen in two horizontal or two vertical panes. Exit a pane with the **exit** command (or Ctrl-d). Move from one pane to the next using **Ctrl-b-o**.

20.12 Cheat sheet

Table 20.1: Useful tools

command	explanation
find .	List all files in the current directory (and all subdirectories).
find /etc	List all files in **/etc**.
find . -name foo	List all files named **foo**.
find . -name 'foo*'	List all files of which the name starts with **foo**.
find . -type d	List only directories.
find . -type f	List only regular files.
find . -exec foo {} \;	Execute the **foo** command on all files found.
find / 2>/dev/null	List all files on the system, while discarding errors.
locate	Use an index to search for files.
updatedb	Update the index with all current filenames.
locate foo	List all files (in the index) that contain **foo** in their name.
watch foo	Repeat the **foo** command every two seconds.
time foo	Give a summary of how long it took to run the **foo** command.
date	Display the current time and date.
date +%s	Display the number of seconds since the epoch (Jan 1st 1970, 00:00:00 UTC).
od -c foo	Display the file **foo** as ASCII characters.
od -b foo	Display the file **foo** as octal numbers.
od -x foo	Display the file **foo** as hexadecimal little endian numbers.
dd if=foo of=bar	Copy **foo** to **bar**.
dd if=/dev/zero of=/dev/sdb	Wipe the **/dev/sdb** device (hard disk or usb stick or . . .)
echo 42+33 | bc	Calculate the sum of these numbers.
echo "scale=4; 42/33" | bc	Calculate the division with four decimal places.
echo 10^10 | bc	Calculate 10 to the power 10.
sleep	Do nothing for x seconds.
script foo	Start recording the terminal session in the file named **foo**.
exit	End the recording of the terminal session.
Ctrl-d	Send an ASCII EOF character (end recording of terminal session).
tmux	

20.13 Practice

1. Search in **/etc** for all **files** named **resolv.conf**, discarding all errors.
2. Search **/var** for all files newer than yesterday 11h (11 a.m.)
3. Search **/etc** for all files named ***.conf** and copy all those files to ~/backup , and discard all errors.
4. Repeat the **file *** command every five seconds.
5. How long does it take to execute **find / >/dev/null 2>&1**?
6. Create a transcript of your session in the file **mysession**.
7. Create a 30 bytes file with random numbers.
8. Create a 1MiB (one mebibyte) file with pseudo random numbers.
9. Sleep for 42 seconds.
10. Interrupt the sleep command.
11. End your **mysession** transcript.
12. Play with **tmux**.

20.14 Solution

1. Search in **/etc** for all **files** named **resolv.conf**, discarding all errors.

```
find /etc -type f -name resolv.conf 2> /dev/null
```

2. Search **/var** for all files newer than yesterday 11h (11 a.m.)

```
touch -t 201908071100 yesterday
find /var -newer yesterday 2>/dev/null
```

3. Search **/etc** for all files named ***.conf** and copy all those files to ~/backup , and discard all errors.

```
find /etc -name '*.conf' -exec cp {} ~/backup \;
```

4. Repeat the **file *** command every five seconds.

```
watch -n5 file \*
```

5. How long does it take to execute **find / >/dev/null 2>&1**?

```
time find / >/dev/null 2>&1
```

6. Create a transcript of your session in the file **mysession**.

```
script mysession
```

7. Create a 30 bytes file with random numbers.

```
dd if=/dev/random of=randomfile count=30 bs=1
```

If this stalls then open another terminal to this computer and make some stuff happen.

8. Create a 1MiB (one mebibyte) file with pseudo random numbers.

```
dd if=/dev/urandom of=urandomfile count=1024 bs=1024
```

9. Sleep for 42 seconds.

```
sleep 42
```

10. Interrupt the sleep command.

```
Ctrl-c
```

11. End your **mysession** transcript.

```
exit (or Ctrl-d)
more mysession
```

12. Play with **tmux**.

```
tmux
Ctrl-b %
Ctrl-b "
Ctrl-b o
```

Part III

Introduction to vi and scripting

Chapter 21

Introduction to vi

21.1 vim

Most Linux administrators use **vim** (**v**i **im**proved) rather than the old **vi** itself. An administrator can install **vim** by typing **apt-get install vim**. The **vi** command will then start **vim** instead. Whenever we say **vi** we actually mean **vim** (vi saves a letter).

People often say that **vi** is hard to use. It is not! It is however true that **vi** requires an effort to learn, but once you know it, it is easy to use. (This book and all its predecessors are written in **vi**.)

This introduction to **vi** chapter teaches you the minimal knowledge you need to do simple edits in **vi**. The next chapter, if you persevere, teaches you all the basics of the **vi** editor.

21.2 :q! exiting vi

It can happen, as a junior, that you open a file in **vi** for editing... and that you completely mess up the file. No worries, just hit the **Esc** key followed by **:q!** and enter and you will quit **vi** without saving changes to the opened file.

Do it now, open the tennis file with **vi** and quit it immediately.

```
vim tennis
:q!
```

Practice this on some of the other text files in your home directory.

21.3 a for insert mode

We are now ready to edit a file for real. Open the tennis file in **vi** and walk around with the arrow keys. Don't type anything yet, you are still in **command mode** which means anything you type will be interpreted as a command.

Type **a** to go from **command mode** to **insert mode**. The **a** will not appear on the screen (because it was a command), but now you can type anything. Try to add another tennis (or football) player to the list. When you are finished typing hit **Esc** to get back to **command mode**.

Remember to hit **a** if you want to type some more.

21.4 :w write to file

When you are happy with your changes to the **tennis** file, hit **Esc** followed by **:w**. You have now saved the file. You can verify with **head** or **cat** that the changes are written to the file.

21.5 0 and $ start and end of line

Open a file in **vi** again and while you are in the middle of a line, press the **0** key. You jump to the start of the line. Press the **$** key and you jump to the end of the line.

21.6 w and b word and back

While a file is open in **vi** use the **w** key to jump one word ahead and the **b** key to jump one word back.

21.7 d for delete

The **d** key can be combined with the previous commands to delete text in **vi**. For example, when in the middle of a line of text, press **d$** to delete all characters until the end of the line. Similarly press **d0** to delete all until the start of the line.

Pressing **w** jumps one word ahead, and pressing **dw** will delete a word. Pressing **db** will delete a word back.

21.8 dd delete line and p P paste

Pressing **dd** will delete a line of text. Pressing **p** will paste that line after the current line. Pressing uppercase **P** will paste that line before the current line.

The pasting with **p** and **P** also works after **dw**, **db**, **d0**, and **d$**.

21.9 y for yank

The **yanking** or copying of text can be done with **y**. Press **yy** to yank a line and then use **p** or **P** to paste it. Similarly you can **y0**, **y$**, **yw**, and **yb**.

21.10 repeat command

When in command mode, before giving any of the previous commands, you can type a number to repeat the command. For example **5dw** will delete the next five words and **3yy** will yank three lines. **5p** will paste five times.

21.11 choice

This is it for this humble introduction to **vi**. You now face a choice, either you make the effort to learn **vi** or you go back to **nano** or something to edit text files on Linux. The next chapter digs further into **vi** and can be considered optional.

21.12 Practice

Execute the **vimtutor** command. This is a 30 minute introduction to **vi**.

Chapter 22

More vi

22.1 About this chapter

Congratulations on starting this chapter which goes deeper into **vi** and assumes you have read and understood the previous chapter (including the **vimtutor** practice).

We start the chapter with a couple of tables explaining **vi** commands, some of which are already familiar to you. Don't learn these commands by heart, learn by using them in **vi**. It is okay as a junior Linux administrator to use only some of the commands discussed in these tables.

22.2 a A i I o O

These six commands **a A i I o** and **O** put **vi** in insert mode. Each differ on the position where you can start typing.

Table 22.1: a A i I o O

command	short	action
a	append	Start writing after the current character.
A	Append	Start writing at the end of the current line.
i	insert	Start writing before the current character.
I	Insert	Start writing at the start of the current line.
o	open	Start writing on a new line after the current line.
O	Open	Start writing on a new line above the current line.

22.3 r R x X

The **x** key can be used to delete the character underneath the cursor, while the uppercase **X** key deletes the character just before the cursor. The **r** key will replace one character underneath the cursor, while the uppercase **R** key will allow you to keep replacing characters on the current line until you hit **Esc**.

Table 22.2: x X r R

command	short	action
x	delete	Delete the current character.
X	delete	Delete the character before the current character.
p	paste	Paste the last deleted character.
xp	switch	switch the current and the next character.
r	replace	Replace the current character.
R	Replace	Replace characters on this line until **Esc**.

22.4 u . Ctrl-r

The **u** command will undo you last command in **vi**, if you keep using **u** then you end up with the original document. With **Ctrl-r** you can undo the undo commands until the most recent change. And the **.** command will redo your last command.

Table 22.3: u . Ctrl-r

command	short	action
u	undo	Undo the last command.
uu	undo	Undo the last two commands.
Ctrl-r	undo the undo	Undo the last **u** command.
.	redo	Redo the last command.

22.5 dd yy p P

The **dd** command deletes a line, and **yy** will copy a line. Both commands can be preceded by a number to repeat the command.

Table 22.4: dd yy p P

command	short	action
dd	delete	Delete the current line.
yy	yank	Yank (copy) the current line.
300dd		Delete the next 300 lines.
5yy		Copy the next five lines.
p	paste	Paste (after dd or yy) as the next line.
P	Paste	Paste (after dd or yy) as the previous line.
yyp		Duplicate the current line.
ddp		Switch two lines.

22.6 w b dw db p P

With **w** and **b** you can jump forward and back in the text by word. In combination with **d** this allows for deletion of words.

Table 22.5: w b dw dp p P

command	short	action
w	word	Move forward one word.
b	back	Move back one word.
3w		Move forward three words.
dw	delete word	Delete the current word (at the start of the word!).
d5w or 5dw		Delete five words.
4db	delete back	Delete four words back.
p	paste	Paste (after dw or db) after the current character.
P	Paste	Paster (after dw or db) before the current character.

22.7 0 $ d0 d$

With **0** you can jump to the start of the current line and with **$** you jump to the end of the current line. Both can be combined with **y**ank and **d**elete.

Table 22.6: 0 $ d0 d$ y0 y$

command	short	action
0	start	Jump to start if the current line.
$	end	Jump to end if the current line.
d0	delete	Delete from before the cursor to the start of the line.
d$	delete	Delete from the cursor to the end of the line.
y0	yank	Copy from before the cursor to the start of the line.
y$	yank	Copy from the cursor to the end of the line.

22.8 J G H

The **J** command allows you to **join** the current line with the next line. While **42J** will join the next 42 lines on the current line.

With **G** you jump to last line of the file. Putting a number before **G** will jump to that line number. So **5G** jumps to the fifth line of the document and **8472G** will jump to the 8472nd line in the current file.

The **H** command will jump to the top of the current screen, while **4H** will jump to the fourth line of the current screen.

Table 22.7: J G H

command	short	action
J	Join	Join the next line with the current.
5J	Join	Join the next five lines with the current line.
G	Go	Go to the last line of the current file.
gg	Go	Go to the first line of the current file.
1G	Go	Go to the first line of the current file.
33G	Go	Go to line 33 of the current file.
dG	delete	Delete up to the end of the file.
H		Go to the top of the current screen.
5H		Go to the fifth line of the current screen.

22.9 v V Ctrl-v

Using the **v** or **V** or **Ctrl-v** commands you can highlight a portion of the text for **yanking** or **deleting**.

Table 22.8: v V Ctrl-v

command	action
v	highlight by character
p	Paste after the current character (after v)
V	highlight by line

Table 22.8: (continued)

command	action
p	Paste after the current line (after V)
Ctrl-v	highlight by column

22.10 Writing and quitting

Whenever you are in command mode (after pressing **Esc**), you can press the colon **:** which will appear at the bottom of the screen. After this colon you can type **w** for writing to a file and/or **q** to quit **vi**.

By default **vi** will not allow you to quit without saving the file, this can be overruled by typing **:q!** .

This table gives an overview of **colon** commands.

Table 22.9: Writing and quitting

command	short	action
:w	write	Write to current file.
:w fname	write	Write to the file named fname.
:wq	write and quit	Write to the current file and quit **vi**.
:q	quit	Quit **vi**.
:q!	quit	Quit **vi** without saving changes to the file.
ZZ	write and quit	Same as **:wq** .
:w!	write	Try to write to a non-writable file (see permissions chapter).

22.11 Searching

When you are in command, then you can type a **forward slash /** to start searching the current file. The **/** will appear at the bottom of the screen, and you can search using strings or even using regular expressions. Using the question mark **?** you can search backwards in the current file.

Table 22.10: Searching

command	short	action
/string	search	Case insensitive forward search for **string** in the file.
?string	search	Case insensitive back search for **string** .
n	next	Jump to next occurrence of **string** .
/^string	search	Search for lines starting with **string** .
/string$	search	Search for lines ending with **string** .
/br[aeio]l	search	Search for **bral**, **brel**, **bril**, and **brol**.
/\<he\>	search	Search for the word **he** and not **here** or **the**.

22.12 Search and replace

You may want to replace (all) occurrences of a string with another string. You could do this with **sed -i** as we have seen before, or you can do it from within **vi**. In **vi** you can type the **colon** followed by the **s/foo/bar/** notation.

Table 22.11: Search and replace

command	short	action
:s/foo/bar/	replace	Replace **foo** with **bar** once, in the current line.
:s/foo/bar/g	replace	Replace all occurrences of **foo** with **bar** in the current line.
:s/foo/bar/gi	replace	Do a case insensitive replace of all **foo** with **bar** .
:4,8 s/foo/bar/g	replace	Replace all **foo** with **bar** on lines 4 through 8.
:1,$ s/foo/bar/g	replace	Replace all **foo** with **bar** in the whole file.
:%s/foo/bar/g	replace	Replace all **foo** with **bar** in the whole file.

22.13 Reading input

You can type text in **vi**, but you can also **read** input from other files or read output from commands that you execute.

The first action is performed by typing **:r** followed by a space and a filename. The content of that file will then be pasted, starting from the next line. Make sure you import a text file!

The second action is to type **:r !** followed by a command. This command will be executed in **bash** and the output (stdout and stderr) will be transferred to the file you are editing, again starting from the next line.

Table 22.12: Read

command	short	action
:r fname	read	Read the file named **fname** and insert its content after the current line.
:r !ls	read	Put the output of **ls** after the current line.
:r	read	Read the current file again, inserting it after the current line.

22.14 Multiple files

You can edit multiple files in **vi**, just open them by typing **vi textfile1 textfile2** at the command line. Here are some hints on working with multiple files.

Table 22.13: Open multiple files

command	short	action
:n	next	Go to the next file.
:rew	rewind	Go to the first file.
:args	arguments	List the open files, marking the active one.

22.15 Digraphs

Digraphs are characters that are not present on your keyboard like **omega** or **epsilon** . These characters can be formed in **vi** by type **Ctrl-k** followed by a combination of two characters.

Table 22.14: Some digraphs

command	short	action
Ctrl-k-a-*	alfa	Writes α.
Ctrl-k-b-*	beta	Writes β.
Ctrl-k-p-*	pi	Writes π.
Ctrl-k-m-*	mu	Writes μ.
Ctrl-k-w-*	omega	Writes ω.
Ctrl-k-W-*	Omega	Writes Ω.
Ctrl-k-*-e	epsilon	Writes ε.
:digraph		Displays all digraphs and their shortcut.

22.16 Macro's

You can record several macro's in **vi** by typing **q** followed by a letter to name the macro. Anything you type will then be recorded until you hit **q** again to end the macro. You can call on this macro by typing @ followed by the name of the macro. This @ sign can be preceded by a number for multiple executions of this macro.

For example **q a $ a e n d Esc j q** will record a macro named **a** that adds **end** at the end of the line and then goes to the next line.

The **q b A e n d Esc j q** will create a macro named **b** that does the same thing. Typing **400@b** afterwards will execute this macro 400 times (adding **end** at the end of 400 lines).

22.17 Setting options

There are many options that you can set in **vi**. For example the **set number** option will display line numbers and **syntax on** will recognise syntax for most programming languages. Options can be set on the fly by typing **:** followed by the option.

Most options, when setting them on the fly, can be abbreviated. For example **se nu** is sufficient for **set number**. So is **sy on** for **syntax on**.

Table 22.15: Some vim options

command	action
set number	Display line numbers.
set nonumber	Remove line numbers.
set relativenumber	Display relative line numbers.
syntax on	Enable syntax highlighting.
syntax off	Disable syntax highlighting.
set tabstop=4	Tabs are four spaces wide.
set nobackup	Disable creating a backup file.
set noswapfile	Disable creating a swap file.
set spell	Enable spell checker.
set nospell	Disable spell checker.

22.18 .vimrc

These options can be automatically set by typing them in the **.vimrc** file. For example here is a simple **.vimrc** setting some options.

```
paul@debian10~$ cat .vimrc
set number
syntax on

set background=dark

set nobackup
set nowb
set noswapfile

paul@debian10~$
```

22.19 Practice

1. Read through this chapter and try most of the commands. Use the **tennis** and the **cities** file for practice. They can be downloaded again if required using **wget http://linux-training.be/tennis** and **wget http://linux-training.be/cities** .

Chapter 23

Introduction to scripting

23.1 About scripting chapters

The next four chapters discuss **bash** scripting on Debian Linux. The goal of these chapters is not to become a skilled programmer, but rather to be able to read and understand scripts. As an administrator you will frequently encounter (simple) scripts in Debian Linux.

23.2 Hello world

A script is a text file that can be executed line by line by (in this case) the **bash shell**. Below is an example of a **Hello world** script in bash.

```
paul@debian10:~$ vi first.sh
paul@debian10:~$ cat first.sh
echo Hello world
paul@debian10:~$
```

A text file is not executable by default. We have to manually set the **x** flag for **executable** on the file. We can do this with the **chmod +x** command (which will be discussed in detail in the Permissions chapter).

```
paul@debian10:~$ chmod +x first.sh
paul@debian10:~$
```

To execute the script we wrote, we have to either use an absolute path **/home/paul/first.sh** or use a relative path starting from the current directory as in **./first.sh**. Remember that the current directory is by default not in $PATH so typing just **first.sh** will result in a **command not found** error.

```
paul@debian10:~$ /home/paul/first.sh
Hello world
paul@debian10:~$ ./first.sh
Hello world
paul@debian10:~$ first.sh
-bash: first.sh: command not found
paul@debian10:~$
```

23.3 Sha-bang

It is advised to start any script (bash or Perl or any other scripting language) with **#!** as the first two characters, followed by the interpreter. In our case the interpreter will always be **/bin/bash**.

The **#!** is called a **sha-bang** or **she-bang**. The next screenshot shows how this looks.

```
paul@debian10:~$ vi first.sh
paul@debian10:~$ cat first.sh
#!/bin/bash
echo Hello world
paul@debian10:~$
```

When launching the script, the Linux kernel will start a new process and will read the **sha-bang** to know which exec it should perform for that new process. This new process will end when the script ends.

In this book we only use **bash** for scripting, so the kernel will always be starting **/bin/bash** for our scripts.

23.4 Comment

Except for the **sha-bang**, any line that starts with **#** is ignored by the **bash shell**. You can use the **#** to write comment about the instruction(s).

In the example below we added some comments to our script.

```
paul@debian10:~$ vi first.sh
paul@debian10:~$ cat first.sh
#!/bin/bash

# This command displays Hello world
echo Hello world

# rm -rf ~ is not executed

# The script will end here
paul@debian10:~$
```

23.5 Variables in a script

You can, of course, use variables in a script. In our example we create a variable named **var_s** and we give it a value.

```
paul@debian10:~$ vi vars.sh
paul@debian10:~$ chmod +x vars.sh
paul@debian10:~$ cat vars.sh
#!/bin/bash

var_s=200

echo Our var_s is $var_s

paul@debian10:~$ ./vars.sh
Our var_s is 200
paul@debian10:~$
```

But this variable was created in a **child shell** to the current shell, so it does not exist in our current shell. (You can verify the **child shell** status using **echo $SHLVL**.)

```
paul@debian10:~$ echo $var_s

paul@debian10:~$
```

This means that a script that is executed has no influence on the environment of our current shell. In the script you can create, use and delete variables, all without touching any of the variables in our current shell.

23.6 Sourcing a script

If you **want** your script to change the current environment, then you can, using the **source** command. When **sourcing** a script then the script is run in the current shell (instead of in a child shell).

```
paul@debian10:~$ source vars.sh
Our var_s is 200
paul@debian10:~$ echo $var_s
200
paul@debian10:~$
```

Note
A **sourced** script does not need to be executable!

Since **source** is a long word to type you can abbreviate it as a simple dot. The screenshot below shows two identical commands.

```
paul@debian10:~$ source vars.sh
Our var_s is 200
paul@debian10:~$ . vars.sh
Our var_s is 200
paul@debian10:~$
```

23.7 Reading input

You can ask the user of the script to give **runtime input** by using the **read** statement. The following screenshot asks for a number and then displays that number.

```
paul@debian10:~$ cat ask.sh
#!/bin/bash

echo -n "Give a number between 1 and 9: "
read number
echo You gave $number
paul@debian10:~$ ./ask.sh
Give a number between 1 and 9: 4
You gave 4
paul@debian10:~$
```

Warning
There is no input validation in the previous example!

23.8 Troubleshooting a script

You can force a script to run in a **bash** shell by tying **bash** with the name of the script as the first argument.

```
paul@debian10:~$ bash vars.sh
Our var_s is 200
paul@debian10:~$
```

You can give options to **bash** this way, for example the **xtrace** option to have bash list all the commands that it executes. This can be a way to troubleshoot a script.

This screenshot shows how bash writes every line it will execute behind a **+** sign.

```
paul@debian10:~$ bash -x vars.sh
+ var_s=200
+ echo Our var_s is 200
Our var_s is 200
paul@debian10:~$
```

23.9 Cheat sheet

Table 23.1: Intro scripting

command	**explanation**
chmod +x foo	Make **foo** executable.
./foo	Execute **foo** in the current directory.
#!	Denotes a sha-bang (if these are the first two characters of a script).
#!/bin/bash	Denotes a sha-bang for the **bash** shell.
#	Starts a comment in a script (or on the command line).
foo=bar	Declare a variable named **foo** with a value of **bar**.
source foo	Source the script named **foo**.
. foo	Source the script named **foo**.
read bar	Read runtime input in the variable **bar**.
bash foo	Run the script **foo** in a child bash shell.
bash -x foo	Run the script **foo** in a child bash shell with **xtrace** on.

23.10 Practice

1. Write a script that displays "The answer is 42." and execute it the normal way.
2. Write a script that sets and echoes a variable, execute it and verify that your variable did not survive the script.
3. Write some comments in your second script.
4. Source your last script and verify that the variable now survives the script.
5. Have the **bash** shell show every command of your script that it executes.
6. Write a script with **cd /etc** in it. Is there a difference between normal execution and sourcing this script?

23.11 Solution

1. Write a script that displays "The answer is 42." and execute it the normal way.

```
paul@debian10:~$ vi first.sh
paul@debian10:~$ cat first.sh
#!/bin/bash
echo "The answer is 42."
paul@debian10:~$ chmod +x first.sh
paul@debian10:~$ ./first.sh
The answer is 42.
paul@debian10:~$
```

2. Write a script that sets and echoes a variable, execute it and verify that your variable did not survive the script.

```
paul@debian10:~$ vi second.sh
paul@debian10:~$ cat second.sh
#!/bin/bash
varA=42
echo $varA
paul@debian10:~$ chmod +x second.sh
paul@debian10:~$ ./second.sh
42
paul@debian10:~$ echo $varA

paul@debian10:~$
```

3. Write some comments in your second script.

```
paul@debian10:~$ cat second.sh
#!/bin/bash

# declare a variable
varA=42

# Use a variable
echo $varA
paul@debian10:~$
```

4. Source your last script and verify that the variable now survives the script.

```
paul@debian10:~$ . second.sh
42
paul@debian10:~$ echo $varA
42
paul@debian10:~$
```

5. Have the **bash** shell show every command of your script that it executes.

```
paul@debian10:~$ bash -x second.sh
+ varA=42
+ echo 42
42
paul@debian10:~$
```

6. Write a script with **cd /etc** in it. Is there a difference between normal execution and sourcing this script?

```
paul@debian10:~$ cat third.sh
#!/bin/bash
cd /etc
```

```
pwd
paul@debian10:~$ ./third.sh
/etc
paul@debian10:~$ source third.sh
/etc
paul@debian10:/etc$
```

Chapter 24

Scripting loops

24.1 test

Script loops will run while or until a certain condition is met. This condition can be written with the **test** command. In this screenshot we first test whether 42 is greater than 33 and subsequently we test whether 42 is less than 33. The second **test** fails (with error code 1).

```
paul@debian10:~$ test 42 -gt 33 ; echo $?
0
paul@debian10:~$ test 42 -lt 33 ; echo $?
1
paul@debian10:~$
```

The **test** command can also be written using square brackets, as the following screenshot shows. This is identical to writing **test** followed by an expression.

```
paul@debian10:~$ [ 42 -gt 33 ] && echo true || echo false
true
paul@debian10:~$ [ 42 -lt 33 ] && echo true || echo false
false
paul@debian10:~$
```

Both **test** and **[** are commands in **/usr/bin/**, but both are also builtin to the bash shell.

```
paul@debian10:~$ which test [
/usr/bin/test
/usr/bin/[
paul@debian10:~$ type test [
test is a shell builtin
[ is a shell builtin
paul@debian10:~$
```

Below is a table with some common **test** cases. See the **man test** for a complete list.

Table 24.1: test command

test	**explanation**
-d foo	Does the directory **foo** exist?
-e bar	Does the file **bar** exist?
-f foo	Is **foo** a regular file?
-r bar	Is **bar** a readable file?
'/var' == $path	Is the string **'/var'** equal to the value of **$path**?
42 -lt $foo	Is 42 less than the value of **$foo**?
$foo -ge 8472	Is the value of **$foo** greater than or equal to 8472?
"abc" < $bar	Does **"abc"** sort before the value of **$bar**?
foo -nt bar	Is file **foo** newer than file **bar**?
-o nounset	Is the shell option **nounset** active?

Further more it is possible to combine these expression with **-a** for a logical AND and **-o** for a logical OR.

```
paul@debian10:~$ [ 42 -gt 33 -a 8472 -gt 42 ] && echo true || echo false
true
paul@debian10:~$ [ 42 -gt 33 -o 8472 -gt 42 ] && echo true || echo false
true
paul@debian10:~$ [ 42 -gt 33 -o 8472 -lt 42 ] && echo true || echo false
true
paul@debian10:~$
```

24.2 if

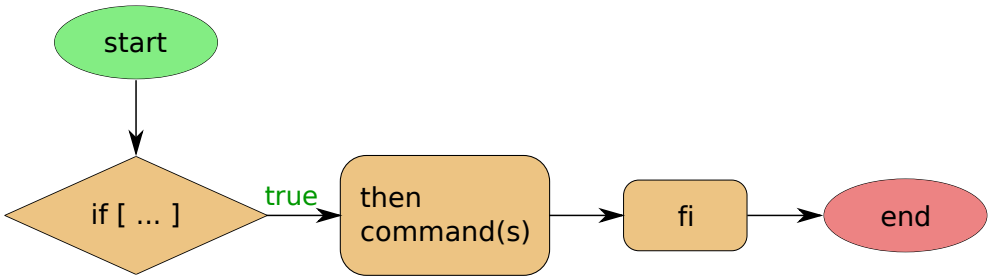
The reserved keywords **if**, **then**, **else**, **elif**, and **fi** can be used to execute commands depending on one or more **test** statements. Let us start with a simple example.

24.2.1 if then fi

In this example we **test** whether a file named **file.txt** exists, and only if the **test** statement evaluates to **true** the statement after **then** will be executed.

```
paul@debian10:~$ cat choice.sh
#!/bin/bash

if [ -f file.txt ]
then echo the file exists
fi
paul@debian10:~$
```

In a diagram the above example looks like this.



You can also write this as a one-liner in the bash shell.

```
paul@debian10:~$ if [ -f file.txt ] ; then echo the file exists ; fi
the file exists
paul@debian10:~$
```

The above program can also be written with the bash logical AND.

```
paul@debian10:~$ [ -f file.txt ] && echo the file exists
the file exists
paul@debian10:~$
```

24.2.2 if then else fi

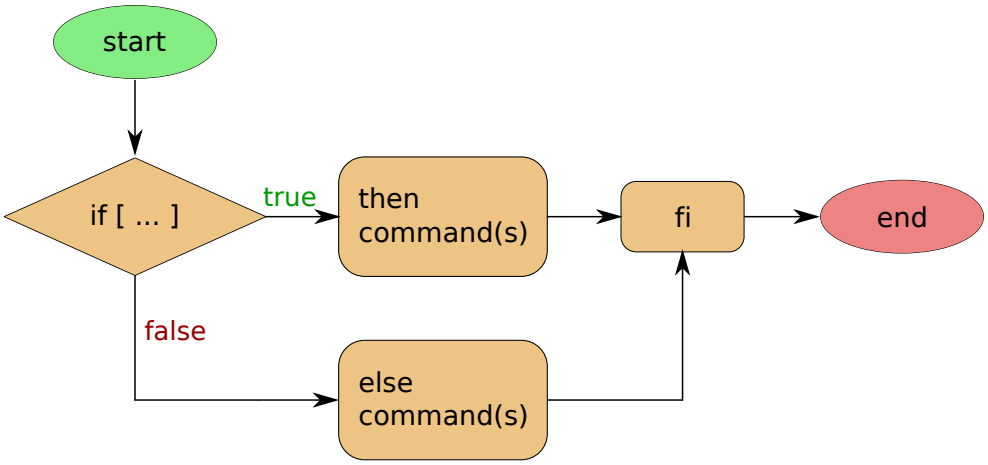
You may want to execute some commands when the **test** succeeds, and some other commands when the **test** fails. This can be done by adding an **else** statement to the above program.

```
paul@debian10:~$ cat choice.sh
#!/bin/bash

if [ -f file.txt ]
then echo the file exists
else echo the file is gone
fi
paul@debian10:~$
```

The screenshot below shows the two possible outputs of the little script.

```
paul@debian10:~$ ./choice.sh
the file is gone
paul@debian10:~$ touch file.txt
paul@debian10:~$ ./choice.sh
the file exists
paul@debian10:~$
```

In a diagram the above program looks like this.



You can again also write this as a one-liner on the bash command prompt.

```
paul@debian10:~$ if [ -f file.txt ] ; then echo the file exists ; else echo the file is  ↩
    gone ; fi
the file exists
paul@debian10:~$
```

The above program can also be written with the bash logical AND and the bash logical OR.

```
paul@debian10:~$ [ -f file.txt ] && echo the file exists || echo the file is gone
the file exists
paul@debian10:~$
```

You can execute multiple commands after the **then** keyword and after the **else** keyword, as this example shows.

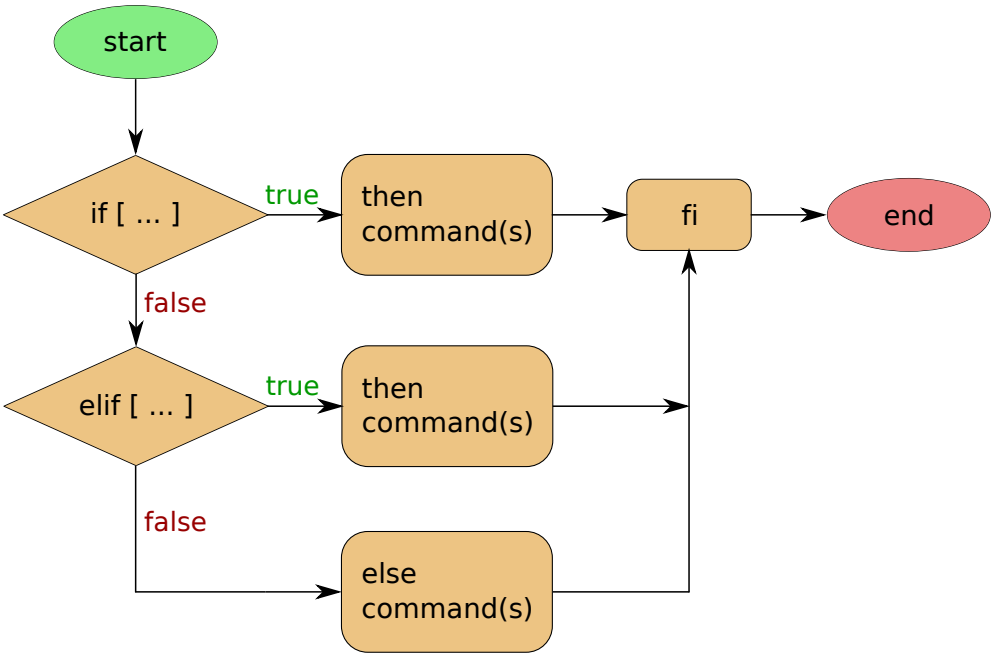
```
paul@debian10:~$ cat choice2.sh
#!/bin/bash

if [ -f file.txt ]
then
     echo the file exists
     echo the file really exists
else
     echo the file is gone
     echo the file is really gone
fi
paul@debian10:~$
```

24.2.3 if then elif then else fi

An **if** statement can be nested using the **elif** keyword. Avoid nesting of **if** statements since this is not beneficial to the readability of your script.

```
paul@debian10:~$ cat choice.sh
#!/bin/bash

if [ -f file.txt ]
then echo the file exists
elif [ -f file2.txt ]
  then echo file2.txt is there
  else echo the files are both gone
fi
paul@debian10:~$
```

In a diagram this code snippet looks like this.



This is not very readable as a bash one-liner on the command prompt.

```
paul@debian10:~$ if [ -f file.txt ] ; then echo the file exists ; elif [ -f file2.txt ] ;  ↩
    then echo file2.txt is there; else echo the files are both gone ; fi
the files are both gone
paul@debian10:~$
```

Though you could write it as a multi-part one-liner, as this screenshot shows.

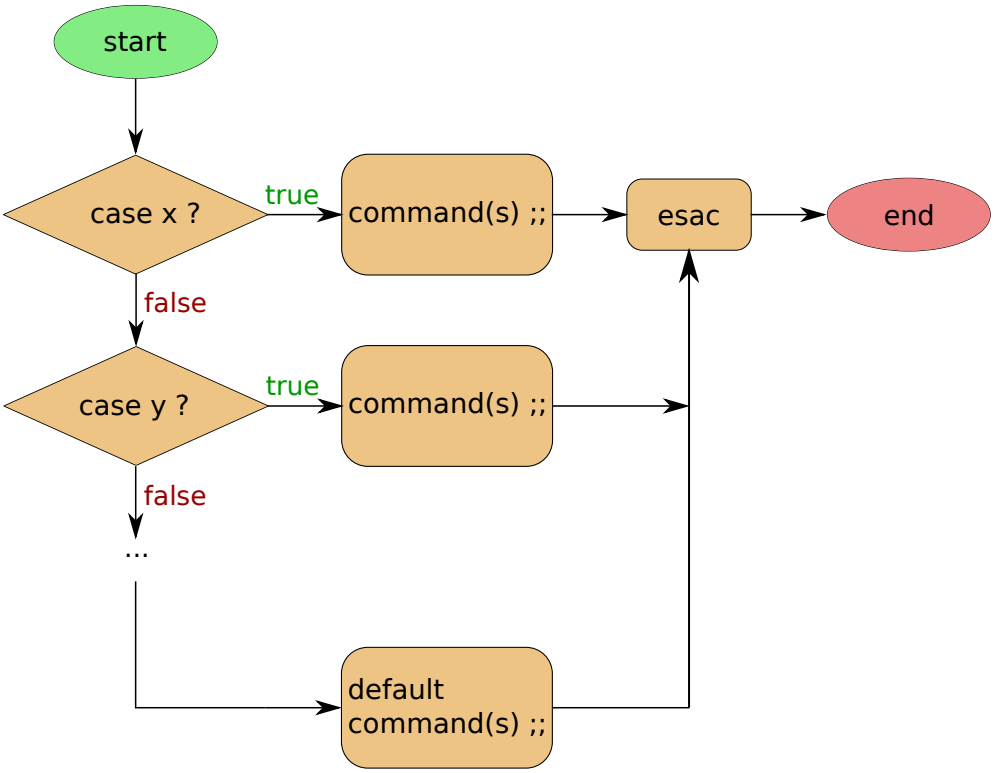
```
paul@debian10:~$ if [ -f file.txt ]
> then echo file exists
> elif [ -f file2.txt ]
> then echo file2.txt is there
> else echo the files are both gone
> fi
the files are both gone
paul@debian10:~$
```

24.3 case

It is often easier to read and write a **case** statement instead of several nested **if** and **elif** statements. Below is an example of a **case** statement with many options.

```
paul@debian10:~$ cat help.sh
#!/bin/bash
#
# Wild Animals Helpdesk
#
echo -n "What animal did you see? "
read animal
case $animal in
  "lion" | "tiger")
    echo "You better start running fast!"
    ;;
  "cat")
    echo "Let that mouse go."
    ;;
  "dog")
    echo "Don't worry, give it a cookie."
    ;;
  "chicken" | "goose" | "duck")
    echo "Yeah, eggs for breakfast!"
    ;;
  "liger")
    echo "Approach and say 'Ah you big fluffy kitty...'"
    ;;
  "babelfish")
    echo "Did it fall out of your ear?"
    ;;
   *)
    echo "You discovered an new animal, name it!"
    ;;
esac
paul@debian10:~$
```

A **case** statement can be represented in a diagram like this. The diagram resembles the **if-then-elif-then-fi** diagram, but a **case** statement in a bash script is easier to read.

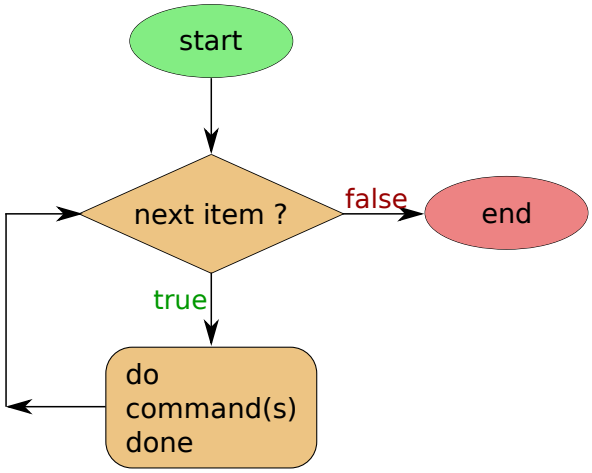


24.4 for loop

A **for** loop will loop through a list and execute all the statements between **do** and **done** for each item of the list.

```
paul@debian10:~$ cat forloop.sh
#!/bin/bash

for i in 33 42 8472
do
        echo the number is $i
done
paul@debian10:~$ ./forloop.sh
the number is 33
the number is 42
the number is 8472
paul@debian10:~$
```

A diagram of a **for** loop looks like this. The **"next item?"** question can be read as **"Is there still an item in the list that was not processed?"**.



Here is an example of a **for** loop but with the list created by an **embedded** shell. Can you figure out what the **seq** command is doing?

```
paul@debian10:~$ cat forloop2.sh
#!/bin/bash

for i in $(seq 1 5)
do
        echo We are at number $i
done
paul@debian10:~$ ./forloop2.sh
We are at number 1
We are at number 2
We are at number 3
We are at number 4
We are at number 5
paul@debian10:~$
```

The following **for** loop has the same output. The **bash** shell is expanding **{1..5}** to **1 2 3 4 5**.

```
paul@debian10:~$ cat forloop3.sh
#!/bin/bash

for counter in {1..5}
```

```
do
        echo We are at number $counter
done
paul@debian10:~$ ./forloop3.sh
We are at number 1
We are at number 2
We are at number 3
We are at number 4
We are at number 5
paul@debian10:~$
```

You can also use file **globbing** when writing a **for** loop. This example walks through all the **.txt** files in the current directory.

```
paul@debian10:~$ cat forloop4.sh
#!/bin/bash

for tfile in *.txt
do
        echo We can see the $tfile file here.
done
paul@debian10:~$ ./forloop4.sh
We can see the combi.txt file here.
We can see the count.txt file here.
We can see the dates.txt file here.
We can see the error.txt file here.
We can see the file.txt file here.
We can see the hello.txt file here.
We can see the music2.txt file here.
We can see the music.txt file here.
We can see the temp.txt file here.
We can see the test.txt file here.
paul@debian10:~$
```

A **for** loop can also be written directly on the command line. This screenshot shows a very similar **for** loop as the one above.

```
paul@debian10:~$ for tf in *.txt; do echo We see $tf; done
We see combi.txt
We see count.txt
We see dates.txt
We see error.txt
We see file.txt
We see hello.txt
We see music2.txt
We see music.txt
We see temp.txt
We see test.txt
paul@debian10:~$
```

And of course you can substitute the **echo** command in the examples with something more useful like **cp** to copy all **.txt** files to the backup directory.

```
paul@debian10:~$ for tf in *.txt; do cp $tf backup/ ; done
paul@debian10:~$
```

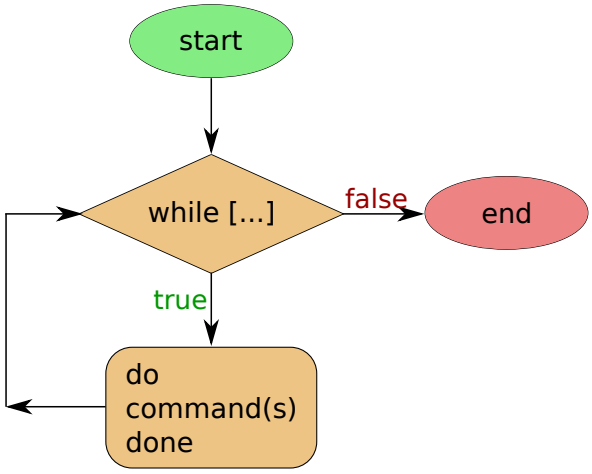
24.5 while loop

A **while** loop will verify a condition and will then execute all commands between **do** and **done** and will then go back to checking the condition, lather, rinse, repeat. Here is an example.

```
paul@debian10:~$ cat while1.sh
#!/bin/bash

i=5
while [ $i -ge 0 ] ;
do
        echo Counting down, from 5 to 0, now at $i
        let i--
done
paul@debian10:~$ ./while1.sh
Counting down, from 5 to 0, now at 5
Counting down, from 5 to 0, now at 4
Counting down, from 5 to 0, now at 3
Counting down, from 5 to 0, now at 2
Counting down, from 5 to 0, now at 1
Counting down, from 5 to 0, now at 0
paul@debian10:~$
```

Note
Note that the above **while** loop is executed **six** times.

The diagram of a **while** loop in bash looks like this (Hey this diagram is identical to the diagram for a **for** loop!). Notice how the **test** statement is evaluated before any command is executed, so it is possible that the command(s) between **do** and **done** are never executed.



Note that you can write a **while** loop as a one-liner on the command line, as this example shows.

```
paul@debian10:~$ i=5; while [ $i -ge 0 ] ; do echo $i ; let i-- ; done
5
4
3
2
1
0
paul@debian10:~$
```

A **while** loop can be endless so be careful when writing the **test** condition. In the screenshot below you can see an endless **while** loop. The script is interrupted by typing **Ctrl-c** (which sends a keyboard interrupt to the script).

```
paul@debian10:~$ cat while2.sh
#!/bin/bash

while :
do
        echo hello
        sleep 3
done
paul@debian10:~$ ./while2.sh
hello
hello
hello
^C
paul@debian10:~$
```

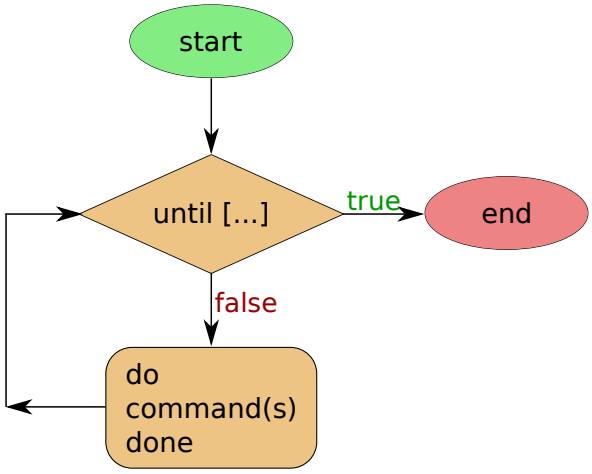
An endless **while** loop can easily be achieved writing **while :** or **while true**. The **:** basically means **no operation** and evaluates to **true**.

24.6 until loop

For completeness, here is an example of an **until** loop. The loop will check the **test** statement and run everything between **do** and **done** . Note that the loop below runs only **five** times.

```
paul@debian10:~$ cat until1.sh
#!/bin/bash

i=5
until [ $i -le 0 ]
do
        echo Counting down now at $i
        let i--
done
paul@debian10:~$ ./until1.sh
Counting down now at 5
Counting down now at 4
Counting down now at 3
Counting down now at 2
Counting down now at 1
paul@debian10:~$
```

In a diagram the **until** loop looks similar to a while loop, except that the **true** and **false** checks have switched places.



As a one-liner this statement becomes this.

```
paul@debian10:~$ i=5; until [ $i -le 0 ] ; do echo $i ; let i-- ; done
5
4
3
2
1
paul@debian10:~$
```

24.7 Cheat sheet

Table 24.2: Scripting loops

command	explanation
test $foo -gt $bar	Compare two variables with "greater than".
test $foo -lt $bar	Compare two variables with "less than".
[$foo -gt $bar]	Compare two variables with "greater than".
[$foo -lt $bar]	Compare two variables with "less than".
-a	A logical AND to combine two test expressions.
-o	A logical OR to combine two test expressions.
if foo ; then bar; fi	Execute **bar** if **foo** is true.
if foo ; then bar1; else bar2; fi	Execute **bar2** if **foo** is false.
case $foo in ... esac	Keywords for a **case** statement.
for var in foo; do bar; done	Execute **bar** for each item in the list **foo**.
while foo; do bar; done	Execute **bar** as long as **foo** is true.
until foo; do bar; done	Execute **bar** until **foo** is true.

24.8 Practice

1. Write a script that uses a **for** loop to count from 3 to 7.
2. Write a script that uses a **for** loop to count from 42 to 8472.
3. Write a script that uses a **while** loop to count from 3 to 7.
4. Write a script that uses an **until** loop to count from 8 to 4.
5. Write a script that counts the number of **.txt** files in the current directory.
6. Write an **if** statement around the script so it is also correct when there are zero files ending in **.txt** .

24.9 Solution

1. Write a script that uses a **for** loop to count from 3 to 7.

```
#!/bin/bash

for i in 3 4 5 6 7
do
  echo Counting from 3 to 7, now at $i
done
```

or as a one liner

```
for i in 3 4 5 6 7; do echo "Counting from 3 to 7, now at $i" ; done
```

2. Write a script that uses a **for** loop to count from 42 to 8472.

```
#!/bin/bash

for i in $(seq 42 8472)
do
  echo Counting from 42 to 8472, now at $i
done
```

3. Write a script that uses a **while** loop to count from 3 to 7.

```
#!/bin/bash

i=3
while [ $i -le 7 ]
do
 echo Counting from 3 to 7, now at $i
 let i=i+1
done
```

4. Write a script that uses an **until** loop to count from 8 to 4.

```
#!/bin/bash

i=8
until [ $i -lt 4 ]
do
  echo Counting from 8 to 4, now at $i
  let i=i-1
done
```

5. Write a script that counts the number of **.txt** files in the current directory.

```
#!/bin/bash

let i=0
for file in *.txt
do
  let i++
done
echo "There are $i files ending in .txt"
```

6. Write an **if** statement around the script so it is also correct when there are zero files ending in **.txt** .

```
#!/bin/bash

ls *.txt > /dev/null 2>&1
if [ $? -ne 0 ]
then echo "There are 0 files ending in .txt"
else
        let i=0
        for file in *.txt
        do
                  let i++
        done
        echo "There are $i files ending in .txt"
fi
```

Chapter 25

Script parameters

25.1 script parameters

A **bash** script can be started with one or more **parameters** a.k.a. arguments. You can refer to these arguments within the script. The argument zero, referred to as **$0**, is the name of the script itself.

The script in the screenshot below explains some of the **parameters** you can use.

```
paul@debian10:~$ cat params.sh
#!/bin/bash

echo my name is $0
echo The first argument is $1
echo The second argument is $2
echo The third argument is $3

echo \$\$ $$ is the PID of this script
echo \$\# $# is the number of arguments
echo \$\* $* is a list of all the arguments
paul@debian10:~$
```

In the screenshot below we execute the script. Note that we pass three arguments to the script. These are **positional** arguments.

```
paul@debian10:~$ ./params.sh one two three
my name is ./params.sh
The first argument is one
The second argument is two
The third argument is three
$$ 628 is the PID of this script
$# 3 is the number of arguments
$* one two three is a list of all the arguments
paul@debian10:~$
```

25.2 shift through parameters

You can shift through all **parameters** using the **shift** command. The script below shows how you can achieve this.

```
paul@debian10:~$ cat shift.sh
#!/bin/bash

if [ "$#" == "0" ]
then
        echo You have to give at least one parameter.
        exit 1
fi

while [ "$#" != "0" ]
do
        echo You gave $1
        shift
done
paul@debian10:~$
```

The screenshot below tests the script first without any arguments, and then with four arguments.

```
paul@debian10:~$ ./shift.sh
You have to give at least one parameter.
paul@debian10:~$ ./shift.sh 33 42 8472 31337
You gave 33
You gave 42
```

```
You gave 8472
You gave 31337
paul@debian10:~$
```

25.3 Parsing options

Scripts can receive **options** in the form of letters preceded by a **-**. These options can be parsed using the **getopts** bash built-in. The script below demonstrates how to do this.

```
paul@debian10:~$ cat options.sh
#!/bin/bash

while getopts ":afz" option
do
        case $option in
                a)
                        echo Received -a
                        ;;
                f)
                        echo Received -f
                        ;;
                z)
                        echo Received -z
                        ;;
                *)
                        echo "Invalid option -$OPTARG"
                        ;;
        esac
done
paul@debian10:~$
```

Running the script with several valid and invalid options, results in the following output.

```
paul@debian10:~$ ./options.sh -a
Received -a
paul@debian10:~$ ./options.sh -zaf
Received -z
Received -a
Received -f
paul@debian10:~$ ./options.sh -f -i
Received -f
Invalid option -i
paul@debian10:~$ ./options.sh -fia
Received -f
Invalid option -i
Received -a
paul@debian10:~$
```

25.4 Options with an argument

Some options may require an argument, for example a path or a filename. The script below requires an argument with the **-f** option.

```
paul@debian10:~$ cat options2.sh
#!/bin/bash

while getopts ":af:z" option
```

```
do
        case $option in
                a)
                        echo Received -a
                        ;;
                f)
                        echo Received -f with $OPTARG
                        ;;
                z)
                        echo Received -z
                        ;;
                :)
                        echo "$OPTARG needs an argument"
                        ;;
                *)
                        echo "Invalid option -$OPTARG"
                        ;;
        esac
done
paul@debian10:~$
```

When running the script, you will notice that the **-f** option needs an argument, but it will accept any argument.

```
paul@debian10:~$ ./options2.sh -az
Received -a
Received -z
paul@debian10:~$ ./options2.sh -af go
Received -a
Received -f with go
paul@debian10:~$ ./options2.sh -z -f go
Received -z
Received -f with go
paul@debian10:~$ ./options2.sh -z -f -a
Received -z
Received -f with -a
paul@debian10:~$ ./options2.sh -f
f needs an argument
paul@debian10:~$
```

25.5 Sourcing a configuration file

An application on Linux usually has a configuration file (often in /etc). Below we show a configuration file for a humble script. This configuration file will be sourced by our script.

```
paul@debian10:~$ cat myapp.conf
# this is the config file for myApp.sh
#

# Enter the path here
myAppPath=/var/myapp

# Enter the number of squirrels
squirrels=42
paul@debian10:~$
```

Below is the actual application that sources the configuration file above, and then uses one of the values set in the configuration file.

```
paul@debian10:~$ cat myApp.sh
```

```
#!/bin/bash
#
# Welcome to myApp

. ./myapp.conf

echo There are $squirrels squirrels
paul@debian10:~$
```

The running application (script in this case) can use the variable as if it was defined in the application script itself.

```
paul@debian10:~$ ./myApp.sh
There are 42 squirrels
paul@debian10:~$
```

25.6 Cheat sheet

Table 25.1: Parameters

parameter	**explanation**
$0	argument 0, the name of the script itself
$1	argument 1, the first positional parameter given to the script
$2	argument 2, the second positional parameter
$#	the number of parameters received (not counting argument 0)
$$	PID of the current script
$*	A string of all the parameters (excluding argument 0)
$?	The last error code
$-	Expands to a list of active shell options
$!	Expands to the last job you put in background
shift	Removes $1 and shifts all $n to $n-1
getopts	filters options from a list of options

25.7 Practice

1. Write a shell script that receives four parameters, and outputs them in reverse order.
2. Write a script that receives two filenames and outputs whether those files exist.
3. Write a script that receives any number of filenames and outputs whether those files exist.
4. Same as 3. , but write the script without an **if** statement.
5. Look at **/etc/init.d/ssh** and determine whether it **sources** a configuration file.

25.8 Solution

1. Write a shell script that receives four parameters, and outputs them in reverse order.

```
#!/bin/bash
echo $4 $3 $2 $1
```

2. Write a script that receives two filenames and outputs whether those files exist.

```
#!/bin/bash
if [ -f $1 ]
then echo $1 exists
else echo $1 not found
fi
if [ -f $2 ]
then echo $2 exists
else echo $2 not found
fi
```

3. Write a script that receives any number of filenames and outputs whether those files exist.

```
#!/bin/bash

while [ $# -gt "0" ]
do
  if [ -f $1 ]
   then echo $1 exists
   else echo $1 not found
  fi
  shift
done
```

4. Same as 3. , but write the script without an **if** statement.

```
#!/bin/bash

while [ $# -gt "0" ]
do
  [ -f $1 ] && echo $1 exists || echo $1 not found
  shift
done
```

5. Look at **/etc/init.d/ssh** and determine whether it **sources** a configuration file.

```
Yes it does, line 22 (as of this writing) says ". /etc/default/ssh" .
```

Chapter 26

More scripting

26.1 eval

The **eval** builtin command will concatenate a string with spaces and execute this as a command in the current **bash** shell. So it is similar to **bash -c** but runs in the current environment.

```
paul@debian10:~$ var1='echo'
paul@debian10:~$ var2='hello'
paul@debian10:~$ var3='world'
paul@debian10:~$ eval $var1 $var2 $var3
hello world
paul@debian10:~$
```

This **eval** notation also allows to use the value of a variable as a variable. The following screenshot shows what we mean by this.

```
paul@debian10:~$ answer=42
paul@debian10:~$ word=answer
paul@debian10:~$ eval echo \$$word
42
paul@debian10:~$
```

This allows also to set complete commands in a variable, you cannot execute the variable, but you can **eval** it, as we show in this screenshot.

```
paul@debian10:~$ <b>lastweek='date --date="1 week ago"'</b>
paul@debian10:~$ <b>$lastweek</b>
date: extra operand ‘ago"’
Try <i>date --help</i> for more information.
paul@debian10:~$ <b>eval $lastweek</b>
Thu 08 Aug 2019 01:04:21 PM CEST
paul@debian10:~$
```

26.2 expr

The **expr** command prints the value of an (often mathematical) expression to standard output, as shown in the examples below.

```
paul@debian10:~$ expr 33 + 42
75
paul@debian10:~$ expr 42 \* 8472
355824
paul@debian10:~$ expr 355824 / 8472
42
paul@debian10:~$
```

You can combine the **expr** command with embedded shells, as shown in this example. Note that **date +%s** gives the number of seconds since the epoch Jan 1st, 1970. Dividing this by 86400, the number of seconds in a day, gives us the number of days passed since the Linux epoch.

```
paul@debian10:~$ expr $(date +%s) / 86400
18123
paul@debian10:~$
```

26.3 let

The **let** command will evaluate an arithmetic expression and will return 0 (true) unless the last argument evaluates to 0, then it will return 1 (false). See these examples.

```
paul@debian10:~$ let x='33 + 42' && echo $x
75
paul@debian10:~$ let x='10 + 100/10' && echo $x
20
paul@debian10:~$ let x='10-2+100/10' && echo $x
18
paul@debian10:~$ let x='10*2+100/10' && echo $x
30
paul@debian10:~$
```

Tip
It is August 2019 as of this writing and the following math problem is trending: 8/2(2+2) is it 1 or 16. Can you find out?

```
paul@debian10:~$ let y='2+2' && let x="8 / 2 * $y" && echo $x
16
paul@debian10:~$
```

Note also that the **let "expression"** is the exact same as **((expression))**. The latter is often used in a **while** statement as in the screenshot below. The expression will remain **true** until $# becomes 0.

```
#!/bin/bash

while (( $# ))
do
  [ -f $1 ] && echo $1 exists || echo $1 not found
  shift
done
```

Note
This script is an answer to the previous practice question 4.

The **let** command can work with different numeral systems (bases). Here is an example of first and second in **hexadecimal** and then third and fourth in **octal**.

```
paul@debian10:~$ let x="0xFF" && echo $x
255
paul@debian10:~$ let x="16#FF" && echo $x
255
paul@debian10:~$ let x="8#77" && echo $x
63
paul@debian10:~$ let x="8#100" && echo $x
64
paul@debian10:~$
```

Beware, **let** and **declare** for variables are different from each other.

```
paul@debian10:~$ dec=15 ; oct=017 ; hex=0x0F
paul@debian10:~$ echo $dec $oct $hex
15 017 0x0F
paul@debian10:~$ let dec=15 ; let oct=017 ; let hex=0x0F
paul@debian10:~$ echo $dec $oct $hex
15 15 15
paul@debian10:~$
```

26.4 functions

Functions are a collection of commands that can be called by the name of the function as one command. Here is a simple example of a function that combines two **echo** commands into one new **greetings** command.

```
paul@debian10:~$ cat functions.sh
#!/bin/bash

function greetings {
echo Hello world.
echo Hello to $USER.
}

echo We will now call a function.
greetings
echo The end.
paul@debian10:~$ ./functions.sh
We will now call a function.
Hello world.
Hello to paul.
The end.
paul@debian10:~$
```

A function can also receive parameters, for example this function which adds two numbers receives both numbers as a parameter. Sourcing this function would make the **plus** command valid on the command line.

```
paul@debian10:~$ cat functions2.sh
#!/bin/bash

function plus {
let result="$1 + $2"
echo $1 + $2 = $result
}

plus 33 42
plus 42 33
plus 8472 33
paul@debian10:~$ ./functions2.sh
33 + 42 = 75
42 + 33 = 75
8472 + 33 = 8505
paul@debian10:~$
```

This function can also be written as a one-liner on the command line. Note the extra semicolon after the last command (echo) inside the function.

```
paul@debian10:~$ function plus { let r="$1 + $2" ; echo $r ; } ; plus 33 42
75
paul@debian10:~$
```

26.5 exit code in $PS1

Some Linux administrators put a lot of information in the command prompt **$PS1**, sometimes spanning multiple lines. One way to do this is using the **$PROMPT_COMMAND** variable of bash. If you assign a function to this variable, then that function will be executed each time the prompt is displayed. This allows you to put the output of any script in your command prompt.

As an example, we put the last error code in red in the command prompt. It displays a space when there is no error code. We also simplified the prompt colour usage. Source this script, or put the function and the **export** in your **~/.bashrc**.

```
paul@debian10~$ cat prompt.sh
#!/bin/bash

prompt_command () {
 [ $? -eq 0 ] && ERR=" " || ERR='==$?== '
 RED="\[\033[01;31m\]"
 GREEN="\[\033[01;32m\]"
 BLUE="\[\033[01;34m\]"
 WHITE="\[\033[01;37m\]"
 DEFAULT="\[\033[0;39m\]"
 export PS1="$GREEN\u$WHITE@$BLUE\h$WHITE\w\$$RED$ERR$DEFAULT"
}

export PROMPT_COMMAND=prompt_command
paul@debian10~$
```

Typing a command that produces an error will have an output like this. The **==1==** is displayed in red.

```
paul@debian10~$ rm g
rm: cannot remove g: No such file or directory
paul@debian10~$==1==
paul@debian10~$==1==
```

This method allows you to put cpu load, current time, number of httpd processes, current weather, or anything in your command prompt.

26.6 Cheat sheet

Table 26.1: More scripting

command	**explanation**
eval	Concatenate to a string with spaces and execute as a command.
expr	Print the value of an expression.
let	Evaluate an arithmetic expression.
let i--	Decrease the value of i with 1.
let i++	Increase the value of i with 1.
let x="0x42"	Let x be the **hexadecimal** value 42.
let x="16#42"	Let x be the **hexadecimal** value 42.
let x="8#42"	Let x be the **octal** value 42.
((expression))	This is identical to **let expression**.
function foo { bar ; }	Declare a function named **foo** which executes **bar**.
foo	Call the function named **foo**.

26.7 Practice

1. Read and understand this chapter, then put the red-coloured error code in your prompt.

Part IV

Introduction to Linux system management

Chapter 27

Introduction to users

27.1 who am i

We already saw the **who am i** command in the *Start* chapter. This command displays the user that we used to log on to this system.

```
paul@debian10:~$ who am i
paul     pts/0        2019-08-11 11:44 (192.168.1.28)
paul@debian10:~$
```

This user, **paul** in my case, does not exist by default. An administrator has created this user on this system.

27.2 whoami

There is a simple command named **whoami** which will show you your current **username**. By default this is also displayed in your command prompt.

```
paul@debian10:~$ whoami
paul
paul@debian10:~$
```

27.3 id

The **id** command will give you a lot more information. We will look at the **groups** later, for now we can see that the user **paul** has **uid** 1000 and **gid** 1000. So besides a username every user also has a **uid** and **gid** (as in a primary group).

```
paul@debian10:~$ id
uid=1000(paul) gid=1000(paul) groups=1000(paul),24(cdrom),25(floppy),29(audio),30(dip),44( ↩
    video),46(plugdev),109(netdev),111(bluetooth)
paul@debian10:~$
```

27.4 su

If you know the password of another **user account** on this computer, then you can issue an **su** command to switch to this user. In the screenshot below we **switch user** from **paul** to **annik**.

```
paul@debian10:~$ whoami
paul
paul@debian10:~$ su annik
Password:
annik@debian10:/home/paul$ whoami
annik
annik@debian10:/home/paul$
```

When you type **exit** or **Ctrl-d** then you go back to the previous user.

Note
After the **su** we did not arrive in the correct home directory.

27.5 su -

You can also use **su -** instead of **su** to switch to another user. The difference (for now) is that with **su -** you also arrive in the *environment* of the target user. See the screenshot below.

```
paul@debian10:~$ whoami
paul
paul@debian10:~$ su - annik
Password:
annik@debian10:~$ whoami
annik
annik@debian10:~$
```

We will see later what exactly differs when executing **su** or **su -**.

27.6 root

There is a special user account, named **root**, on every Debian Linux system. This is the user with **uid 0** and is created at installation time. This **root** user has the power to create other users (and to manage all aspects of the Linux system).

If you know the **root** password, then you can **su -** to this special user.

```
paul@debian10:~$ su - root
Password:
root@debian10:~# whoami
root
root@debian10:~#
```

There is no need to specify the username when you **su** to **root**, as can be seen in this screenshot.

```
paul@debian10:~$ su -
Password:
root@debian10:~#
```

27.7 sudo

Debian does not install **sudo** by default, but a lot of derivatives of Debian, like Ubuntu do install **sudo**. In short this **sudo** command allows you to execute a program as another user (often this is the root user).

When using **sudo** you need to provide *your own* password.

27.8 ssh

When you use **ssh** (or **putty**) to connect to a Linux computer then you need to provide a hostname and a username. For example **ssh paul@herrdebby** will attempt to connect to the **herrdebby** computer with the username **paul**. It will then ask for my password on that computer.

The screenshot below shows how to use **ssh** to connect to another computer. This only works if you have a user account on that computer.

```
paul@debian10:~$ ssh paul@herrdebby
paul@herrdebby's password:
Linux herrdebby 4.19.0-5-amd64 #1 SMP Debian 4.19.37-5+deb10u1 (2019-07-19) x86_64

The programs included with the Debian GNU/Linux system are free software;
```

```
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jul 21 00:29:03 2019 from 109.130.74.182
paul@herrdebby~$ whoami
paul
paul@herrdebby~$ hostname
herrdebby2
paul@herrdebby~$
```

27.9 w

To finish this introductory chapter on users we show you the **w** command. It will show all logged on users and the command they are executing.

```
paul@debian10:~$ w
 16:12:13 up  4:29,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -                16:12   13.00s  0.03s  0.01s -bash
paul     pts/0    192.168.1.28     11:44    0.00s  0.46s  0.00s w
paul@debian10:~$
```

27.10 Cheat sheet

Table 27.1: Introduction to users

command	**explanation**
who am i	Display your login information.
whoami	Display your current username.
id	Display your username and userid, and list group membership details.
su foo	Switch to the **foo** user account.
su - foo	Switch to the **foo** user account and the **foo** environment.
su - root	Switch to the **root** user account and the **root** environment.
sudo	Execute a command as another user (not by default in Debian).
ssh foo@bar	Connect to a user account **foo** on another computer named **bar**.
w	Display all logged on users on this computer and their current command.

27.11 Practice

1. Display a list of logged on users.
2. Display your username.
3. Display your **uid** and the list of groups that you are a member of.
4. If you have the password of another user, then switch to that user.
5. If you have the password of another user, then switch to that user and its environment.

27.12 Solution

1. Display a list of logged on users.

```
w
who
who | cut -d' ' -f1
```

2. Display your username.

```
whoami
```

3. Display your **uid** and the list of groups that you are a member of.

```
id
```

4. If you have the password of another user, then switch to that user.

```
su username
```

5. If you have the password of another user, then switch to that user and its environment.

```
su - username
```

Chapter 28

User management

Warning
For this chapter you need your own (virtual) Debian Linux server, as you will be executing commands as **root**.

28.1 useradd

The **useradd** command is used to add users to a Debian Linux computer. There are several options that can be used while creating a user.

Table 28.1: useradd options

option	explanation
-c	Set an optional comment for this user.
-e	Set an expiry date for this account.
-m	Create a home directory for this user.
-p	Set an **encrypted** password.
-s	Set the default shell for this user.
-u	Set the **uid** for this user.

In the screenshot below we create three users, and use some of the options from the table above. It is a custom in Linux to use lowercase characters for the username.

```
root@debian10:~# useradd -m -s /bin/bash tania
root@debian10:~# useradd -m laura
root@debian10:~# useradd valentina
root@debian10:~#
```

When verifying the **home directories** we can see that there is no directory created for **valentina** (because the **-m** option was not used).

```
root@debian10:~# ls /home
annik  david  geert  laura  linda  paul  tania
root@debian10:~#
```

28.2 su - as root

When you are the **root** user then you can **su** to any other account, without knowing the password. In the screenshot below we use **su -** to verify that the users have the bash shell.

```
root@debian10:~# su - tania
tania@debian10:~$ echo $SHELL
/bin/bash
tania@debian10:~$ exit
logout
root@debian10:~# su - laura
$ echo $SHELL
/bin/sh
$ exit
root@debian10:~#
```

As you can see the **bash** shell was correctly set for **tania**, but not for **laura**.

28.3 usermod

Luckily we can correct our omissions with the **usermod** command. The **usermod** command works on existing user accounts and has the same options as **useradd**, so the solution is trivial.

```
root@debian10:~# usermod -s /bin/bash laura
root@debian10:~# usermod -d /home/valentina -m -s /bin/bash valentina
root@debian10:~#
```

28.4 userdel

A user account can be deleted using the **userdel** command. By default the **home directory** of that user will remain. You can force deletion of the home directory using the **-r** option.

```
root@debian10:~# userdel -r tania
userdel: tania mail spool (/var/mail/tania) not found
root@debian10:~#
```

Note
The **-r** option will also (try to) remove the user's mail spool.

28.5 /etc/passwd

Information about users on a Debian Linux system is kept in the **/etc/passwd** file. The screenshot below shows the last three lines of this file.

```
root@debian10:~# tail -3 /etc/passwd
annik:x:1004:1004::/home/annik:/bin/bash
laura:x:1006:1006::/home/laura:/bin/bash
valentina:x:1007:1007::/home/valentina:/bin/bash
root@debian10:~#
```

There are seven fields in this file, separated by a colon. Below is a summary of the seven fields in order of appearance.

Table 28.2: /etc/passwd fields

field	explanation
annik	the username
x	x (long ago the encrypted password was here)
1004	the uid
1004	the gid
	an optional comment
/home/annik	the name of the home directory
/bin/bash	the default shell for this user

28.6 chsh

Any user can change their own shell with the **chsh** command. The root user can, of course, change anyone's shell, as can be seen in this screenshot.

```
root@debian10:~# chsh -s /bin/sh laura
root@debian10:~# grep laura /etc/passwd
```

```
laura:x:1006:1006::/home/laura:/bin/sh
root@debian10:~# chsh -s /bin/bash laura
root@debian10:~#
```

28.7 home directories

When using the **useradd -m** option, Debian Linux will create a home directory for the user in the **/home** parent directory. The default home directory appears empty, but there are hidden files.

```
root@debian10:~# useradd -m -s /bin/bash tania
root@debian10:~# ls -la /home/tania/
total 20
drwxr-xr-x 2 tania tania 4096 Aug 11 17:43 .
drwxr-xr-x 9 root  root  4096 Aug 11 17:43 ..
-rw-r--r-- 1 tania tania  220 Apr 18 06:12 .bash_logout
-rw-r--r-- 1 tania tania 3526 Apr 18 06:12 .bashrc
-rw-r--r-- 1 tania tania  807 Apr 18 06:12 .profile
root@debian10:~#
```

28.8 /etc/skel

There is a skeleton directory named **/etc/skel** on Debian which contains the default files that are copied into any new home directory (when using **useradd -m**).

```
root@debian10:~# ls -la /etc/skel/
total 20
drwxr-xr-x  2 root root 4096 Jul 24 18:08 .
drwxr-xr-x 74 root root 4096 Aug 11 17:43 ..
-rw-r--r--  1 root root  220 Apr 18 06:12 .bash_logout
-rw-r--r--  1 root root 3526 Apr 18 06:12 .bashrc
-rw-r--r--  1 root root  807 Apr 18 06:12 .profile
root@debian10:~#
```

If you place files in **/etc/skel** then those files will be copied to the home directory of any user you create. This copy happens at creation time, files will not be copied to existing users!

28.9 adduser

Some administrators prefer to use the **adduser** tool instead of **useradd**. The end result is the same, a user will be created in **/etc/passwd**.

28.10 Cheat sheet

Table 28.3: User management

command	**explanation**
useradd foo	Create a new user named **foo** on this computer.
useradd -m foo	Create a new user with a home directory (/home/foo).
useradd -s /bin/bash foo	Create a new user with the **bash** shell as default shell.
usermod -s /bin/bash foo	Change the default shell for the user **foo** to bash.
userdel foo	Delete the user **foo**, the home directory is untouched.
userdel -r foo	Delete the user **foo**, also delete the home directory.
chsh -s /bin/bash foo	Change the default shell for user **foo**.
/etc/passwd	The list of all users on this computer.
/home	The default location for home directories.
/etc/skel	The skeleton (template) directory for home directories.

28.11 Practice

1. Create a user named **serena** with a home directory, a comment and the bash shell.
2. Verify in **/etc/passwd** that this user was created.
3. Use **su -** to verify that this user has the bash shell and a home directory.
4. Change the comment for the **serena** user to **tennis** .
5. Verify that there is a **serena** home directory in **/home** .
6. Delete the **serena** user and the user's home directory.

28.12 Solution

1. Create a user named **serena** with a home directory, a comment and the bash shell.

```
useradd -m -s /bin/bash -c Serena Williams serena
```

2. Verify in **/etc/passwd** that this user was created.

```
grep serena /etc/passwd
```

3. Use **su -** to verify that this user has the bash shell and a home directory.

```
su - serena
echo $SHELL
ls -la
exit
```

4. Change the comment for the **serena** user to **tennis** .

```
usermod -c tennis serena
```

5. Verify that there is a **serena** home directory in **/home** .

```
ls -l /home
```

or

```
ls -ld /home/serena/
```

6. Delete the **serena** user and the user's home directory.

```
userdel -r serena
```

Chapter 29

Password management

29.1 passwd

The most common way to set a password for a user is to use the **passwd** command as **root**. The screenshot below sets a password for the user **tania**. The password is set to **hunter2**, this is not displayed on the terminal window.

```
root@debian10:~# passwd tania
New password:
Retype new password:
passwd: password updated successfully
root@debian10:~#
```

As a normal user you can change your password using the **passwd** command, but then you need to know (and type) your old password, otherwise you cannot change your password.

```
tania@debian10:~$ passwd
Changing password for tania.
Current password:
New password:
Retype new password:
passwd: password updated successfully
tania@debian10:~$
```

29.2 /etc/shadow

Passwords are encrypted and the encrypted string is kept in the **/etc/shadow** file. The screenshot below shows that the users **laura** and **valentina** do not have a password yet. These users cannot login using a password (but there are other ways of logging in).

```
root@debian10:~# tail -3 /etc/shadow
laura:!:18119:0:99999:7:::
valentina:!:18119:0:99999:7:::
tania:$6$fG8oDKeooCJBfo5W$Mc.As85/mr3iblJku7n37yWaiwtWPj9EftfmX2VicrjPqE6NcX2qzSV7GOU/ ↩
    SzKHY6dcriYd0XWUrf2mgi1lF.:18120:0:99999:7:::
root@debian10:~#
```

The fields in **/etc/shadow** are described in the table below. The encrypted password starts with **6**, this identifies the SHA-512 algorithm.

Table 29.1: /etc/shadow fields

field	**explanation**
tania	username
6...	encrypted password
18120	the day of last password change
0	minimum password age
99999	password expiry day
7	warning days before expire
	password inactivity period
	account expiration date
	reserved for future use

The numbers 18120 and 99999 represent days since the epoch. Which means day 0 is January 1st, 1970 (hence day 18120 is August 12th, 2019).

```
paul@debian10:~$ expr $(date +%s) / 86400
18120
paul@debian10:~$
```

29.3 chage

The fields in the **/etc/shadow** file can also be viewed (and changed) using the **chage** command. The example below shows how to list the values for your own user. The **root** user can use **chage** to edit all these values.

```
paul@debian10:~$ chage -l paul
Last password change                                    : Jul 24, 2019
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
paul@debian10:~$
```

29.4 /etc/login.defs

The **/etc/login.defs** file contains some default settings for user passwords, like password aging and length. You can also find the numerical limits for uid's and gid's and whether or not a home directory should be created by default.

The screenshot shows some of the password settings available in **/etc/login.defs** .

```
paul@debian10:~$ grep PASS /etc/login.defs
#       PASS_MAX_DAYS   Maximum number of days a password may be used.
#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
#       PASS_WARN_AGE   Number of days warning given before a password expires.
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_WARN_AGE   7
#PASS_CHANGE_TRIES
#PASS_ALWAYS_WARN
#PASS_MIN_LEN
#PASS_MAX_LEN
# NO_PASSWORD_CONSOLE
paul@debian10:~$
```

29.5 openssl

You can set a password when adding the user with **useradd**, but the **-p** option of this command requires an encrypted password. To calculate the encrypted password, you can use **openssl** as shown in this screenshot.

```
paul@debian10:~$ openssl passwd hunter2
cTpJqYxZIIR8g
paul@debian10:~$ openssl passwd -6 hunter2
$6$GSsWZvofqKRDYoql$Q5Z9tFFH3zvapvitzBFBK9.rp48zvQ4U5UEhbJwoQgV5ILAw4dUv.RoQlaUTBCZ9fqJxMC. ↩
    zhQBjgiJGAtdMo/
paul@debian10:~$
```

The first **openssl** command will generate a valid password hash, but this is no longer considered secure. The second **openssl** command will use SHA-512. Both are valid arguments to the **useradd -p** option.

Warning
The following statement is not very secure, because you have a password in clear text in the process listing and in your bash history.

```
root@debian10:~# useradd -m -s /bin/bash -p $(openssl passwd -6 hunter2) mohamed
root@debian10:~#
```

29.6 crypt

A third and final option to generate password hashes is to use the **crypt** function. The screenshot below shows how to do this in **Python**, then in **Perl** and then in **C**. The examples below use **salt** as the salt, this can be changed!

```
paul@debian10:~$ python -c 'import crypt; print crypt.crypt("hunter2", "$6$salt")'
$6$salt$wAH.7/lcHCMJGFjRlqUqFxcc7YA/UTYSAsXRVlygbexkXyhkV2uwiJtwTlie0Eo8qerF9f.eK6LEsR.Tlhp ↩
    /k0
paul@debian10:~$ perl -e 'print crypt("hunter2","\$6\$salt\$") . "\n"'
$6$salt$wAH.7/lcHCMJGFjRlqUqFxcc7YA/UTYSAsXRVlygbexkXyhkV2uwiJtwTlie0Eo8qerF9f.eK6LEsR.Tlhp ↩
    /k0
paul@debian10:~$ ./MyCrypt hunter2 '$6$salt'
$6$salt$wAH.7/lcHCMJGFjRlqUqFxcc7YA/UTYSAsXRVlygbexkXyhkV2uwiJtwTlie0Eo8qerF9f.eK6LEsR.Tlhp ↩
    /k0
paul@debian10:~$
```

The **C** program looks like this, compiling can be done after an **apt-get install gcc**.

```
paul@debian10:~$ cat MyCrypt.c
#include <stdio.h>
#define __USE_XOPEN
#include <unistd.h>

int main(int argc, char** argv)
{
if(argc==3)
 {
  printf("%s\n", crypt(argv[1],argv[2]));
 }
else
 {
 printf("Usage: MyCrypt $password $salt\n" );
 }
return 0;
}
paul@debian10:~$ gcc MyCrypt.c -o MyCrypt -lcrypt
paul@debian10:~$
```

29.7 Generating good passwords

There are several tools available to generate a good password. In general a good password is one that is not easy to guess, for example a long password with uppercase and lowercase letters, some numbers and some other characters.

Below are some Debian tools that generate passwords.

29.7.1 pwgen

The **pwgen** tool (apt-get install pwgen) is a simple tool that generates passwords of a given length using letters and numbers.

```
paul@debian10:~$ pwgen 20 1
ooSah6jie6ab3OoPheer
paul@debian10:~$ pwgen 20 1
Eesei7chahriiFaegeiT
paul@debian10:~$
```

If these passwords are not complex enough, then you can use some options to **pwgen** (read the manual) to add other characters besides letters and numbers.

```
paul@debian10:~$ pwgen -ysBv 20 1
(WW}cX]|%Xgx>c4.pKkx
paul@debian10:~$
```

29.7.2 gpw

The **gpw** tool (apt-get install gpw) will generate *pronounceable* passwords consisting of only lowercase letters.

```
paul@debian10:~$ gpw 1 40
cationspingkostoickshagregriousightsomsc
paul@debian10:~$ gpw 1 40
ttliciessalingolonstaticialomillepribin
paul@debian10:~$
```

29.7.3 diceware

The **diceware** tool (apt-get install diceware) will generate pronounceable passwords from a word list. These passwords are longer but easier to remember.

```
paul@debian10:~$ diceware
StatisticCoasterCaliberPlayhouseResaleWham
paul@debian10:~$ diceware
CrossCoherenceSubsystemArmoredReveredObstacle
paul@debian10:~$
```

29.7.4 xkcdpass

The last honorable mention for a password generator goes to https://www.xkcd.com/936/ and the tool named **xkcdpass** which is based on the comic.

```
paul@debian10:~$ xkcdpass
unsterile jawed bagpipe froth moonshine discuss
paul@debian10:~$ xkcdpass
buckshot undertake vowel radiance trodden crestless
paul@debian10:~$
```

29.8 Locking an account

A user account can be locked by **root** using the **passwd -l** command. The screenshot below shows the locking of the **tania** account, done by adding an **!** at the start of the encrypted password.

```
root@debian10:~# passwd -l tania
passwd: password expiry information changed.
root@debian10:~# grep tania /etc/shadow | cut -b1-60
tania:!$6$JyPEIqaSBpYgRZaW$lldoHK0bBHwUEywTnFTomLQ2kUhdCHIKD
root@debian10:~#
```

Warning
The user **tania** can still log on using other means, for example with **ssh-keys**.

Passwords can be **unlocked** again using the **passwd -u** or **usermod -U** commands. The exclamation mark will then be removed and the old password can be used again.

29.9 Editing local files

Being **root** on the system allows you to edit any file, so this includes **/etc/passwd** and **/etc/shadow**. This is not advised, but just in case you do, always use **vipw** since it will verify that no other **vipw** has the file open for editing.

29.10 Cheat sheet

Table 29.2: Passwords

command	**explanation**
passwd	Change your password
passwd foo	Change the password of the user named **foo**.
chage -l foo	List the fields in /etc/shadow (except the hash) for user **foo**.
openssl passwd hunter2	Display a hash for the password **hunter2**.
openssl passwd -6 hunter2	Display a strong hash for the password **hunter2**.
passwd -l **foo**	Lock the account **foo**.
vipw	Edit /etc/passwd or /etc/shadow, knowing you are the only one using **vipw**.
pwgen	Generates passwords.
gpw	Generates passwords.
diceware	Generates passwords consisting of words.
xkcdpass	Generates passwords consisting of words.
/etc/shadow	The file with all password hashes.
/etc/login.defs	This file contains default settings for new user accounts.

29.11 Practice

1. Create a user named **tania** with bash shell and home directory.
2. Verify in **/etc/shadow** that **tania** has no password.
3. Set the password for **tania** to **hunter2** .
4. As a normal user, verify that you can use **su - tania** with the password.
5. As **root**, lock the **tania** account.
6. Verify in **/etc/shadow** that the account is locked.
7. Verify that **su - tania** as a normal user no longer works.
8. Unlock the **tania** account.

29.12 Solution

1. Create a user named **tania** with bash shell and home directory.

```
useradd -m -s /bin/bash tania
```

2. Verify in **/etc/shadow** that **tania** has no password.

```
grep tania /etc/shadow
```

3. Set the password for **tania** to **hunter2** .

```
passwd tania
```

4. As a normal user, verify that you can use **su - tania** with the password.

```
su - tania
exit
```

5. As **root**, lock the **tania** account.

```
passwd -l tania
```

6. Verify in **/etc/shadow** that the account is locked.

```
grep tania /etc/shadow
```

7. Verify that **su - tania** as a normal user no longer works.

```
su - tania
```

8. Unlock the **tania** account.

```
passwd -u tania
```

Chapter 30

User profiles

30.1 /etc/profile

User profiles are scripts that run before the user receives a command prompt. The first script that Debian 10 runs is **/etc/profile**. This script is the same for every user!

The **/etc/profile** scripts starts by setting the **$PATH** for normal user or for **root** user.

```
paul@debian10:~$ head -9 /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "`id -u`" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
fi
export PATH
paul@debian10:~$
```

The **/etc/profile** script then does some checking for interactive **bash** shell and sources **/etc/bash.bashrc** if it exists. If it is not in **bash**, then it sets a simple prompt.

```
paul@debian10:~$ tail -n +11 /etc/profile | head -16
if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1=# '
    else
      PS1='$ '
    fi
  fi
fi

paul@debian10:~$
```

The last part of this script will look for ***.sh** files in the **/etc/profile.d/** directory and run them (in alphabetical order).

```
paul@debian10:~$ tail -8 /etc/profile
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
paul@debian10:~$
```

It is advised to not edit this file. Instead put your custom changes (aliases, variables, and functions) in a .sh file in **/etc/profile.d**. Remember these aliases, variables and functions will be active for **all** users.

30.2 ~/.bash_profile

The **~/.bash_profile** file does not exist by default on Debian 10. But it is important to know that, should you create this file, then **~/.profile** is no longer executed!

```
paul@debian10:~$ ls . tab tab
./             .bash_history  .bashrc        .hidden        .profile       .viminfo
../            .bash_logout   .config/       .lesshst       .ssh/          .vimrc
paul@debian10:~$ ls .
```

30.3 ~/.bash_login

This script does not exist by default on Debian 10, but if you create it and **~/.bash_profile** does not exist, then this script is run instead of **~/.profile**.

30.4 ~/.profile

The **~/.profile** script is executed only if **~/.bash_profile** and **~/.bash_login** do not exist. The first thing this script will do is check for the existence of **~/.bashrc** and source this file.

```
paul@debian10:~$ tail +11 .profile |head -7
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi
paul@debian10:~$
```

Then it checks for the existence of **~/bin** and **~/local/bin** and adds them to the $PATH if they exist.

30.5 ~/.bashrc

The **~/.bashrc** file does a lot of things, like setting variables, aliases, shell options and possible a color prompt. It also explains every action it takes in comment lines. This script also checks for the existence of **~/.bash_aliases** (which does not exist by default).

Changing the $HISTSIZE and $HISTFILESIZE variables in this file will influence how much history is kept by bash (for your personal user account).

```
paul@debian10:~$ grep HIST .bashrc
HISTCONTROL=ignoreboth
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=20000
HISTFILESIZE=20000
paul@debian10:~$
```

30.6 ~/.bash_logout

When you properly exit the shell (using **exit** or **Ctrl-d**) then **~/.bash_logout** will run. The only thing it does is clear the terminal window (if it is a console).

```
paul@debian10:~$ cat .bash_logout
# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy
```

```
if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
paul@debian10:~$
```

30.7 su and su -

We briefly discussed the difference between **su** and **su -**, explaining that **su -** gets *the environment* of the target user.

When executing **su** without the **-**, then only the **/etc/bash.bashrc** and **~/.bashrc** are sourced in your environment. So you will see aliases, variables and functions from these files.

When executing **su -**, then **/etc/profile** and **~/.profile** are also executed, so you will get aliases, variables and functions from all four profile scripts.

30.8 ssh (and putty)

When using ssh (or putty) to remotely log on to a Debian 10 Linux computer, then all four profile scripts **/etc/profile**, **/etc/bash.bashrc**, **~/.bashrc**, and **~/.profile** are sourced into your environment.

30.9 graphical login via xfce

When logging in to a graphical desktop environment, in this case XFCE, then only the **/etc/bash.bashrc** and **~/.bashrc** are executed. This was tested by setting an **alias** in each of the profile scripts.

```
paul@Debian10~$ alias | grep yes
alias _bash_rc=yes
alias etc_bash.bashrc=yes
paul@Debian10~$
```

30.10 tmux

When executing **tmux** on the command line, then all four profile scripts are executed (sourced in fact). Starting **tmux** does start a brand new **bash** shell.

```
paul@Debian10~$ alias | grep yes
alias _bash_rc=yes
alias _etc_profile=yes
alias _profile=yes
alias etc_bash.bashrc=yes
paul@Debian10~$
```

30.11 Cheat sheet

Table 30.1: Profiles

file	explanation
/etc/profile	First script to run for every user on this computer.
/etc/bash.bashrc	General profile script for all users, called by **/etc/profile**.
~/.bash_profile	If this script exists, it is run instead of **~/.profile**.
~/.bash_login	If this script exists, and **~/.bash_profile** does not, it is run instead of **~/.profile**.
~/.profile	Standard profile script on Debian, exists per user.
~/.bashrc	Standard profile script on Debian, called by **~/.profile.**
~/.bash_logout	Script that is executed when properly logging out.

30.12 Practice

1. Read **/etc/profile** and add an alias as the last line.
2. Verify with **su** and **su -** whether your alias exists.
3. Do the same with an alias in **/etc/bash.bashrc**.
4. Make sure that 5000 lines of history are remembered and stored for your user.
5. Display 'Sad to see you go…' for 10 seconds when logging out.
6. Install **bash-doc** and read the Bash_aliases file in /usr/share/doc/bash-doc/examples/startup-files/ .

30.13 Solution

1. Read **/etc/profile** and add an alias as the last line.

```
su -
vi /etc/profile
G
alias _etc_profile=yes
```

2. Verify with **su** and **su -** whether your alias exists.

```
su paul
alias
exit
su - paul
alias
exit
```

3. Do the same with an alias in **/etc/bash.bashrc**.

```
su -
vi /etc/bash.bashrc
G
alias _etc_bash.bashrc=yes
su paul
alias
exit
su - paul
alias
exit
```

4. Make sure that 5000 lines of history are remembered and stored for your user.

```
cd
vi .bashrc
/HIST
n
HISTSIZE=5000
HISTFILESIZE=5000
```

5. Display 'Sad to see you go...' for 10 seconds when logging out.

```
cd
vi .bash_logout
G
echo 'Sad to see you go...' && sleep 10
```

6. Install **bash-doc** and read the Bash_aliases file in /usr/share/doc/bash-doc/examples/startup-files/ .

```
su -
apt-get install bash-doc
exit
more /usr/share/doc/bash-doc/examples/startup-files/Bash_aliases
```

Chapter 31

Group management

31.1 groups

By default any user created on Debian Linux 10 will belong to one group. This group has the same name as the user and is considered the *primary group* of that user.

```
tania@debian10:~$ groups
tania
tania@debian10:~$
```

An exception to this rule is the very first user, which is created at installation time. This user belongs to a number of groups, as shown in this screenshot.

```
paul@debian10:~$ groups
paul cdrom floppy audio dip video plugdev netdev bluetooth
paul@debian10:~$
```

31.2 groupadd

To create a group, as root, issue the **groupadd** command followed by the name of the group. The example below creates two new *empty* groups.

```
root@debian10:~# groupadd tennis
root@debian10:~# groupadd football
root@debian10:~#
```

31.3 /etc/group

Groups are stored by the Linux system in the **/etc/group** file. Every line in this file has four fields separated by a colon. The first field is the group name, then there is room for a group password (I have never seen this in use), then follows the group id and last is the list of members of this group.

```
paul@debian10:~$ tail -2 /etc/group
tennis:x:1009:
football:x:1010:
paul@debian10:~$
```

31.4 groupmod

You can change the **gid** or the name of the group with the **groupmod** command. Both actions are shown in the screenshot below.

```
root@debian10:~# groupmod -g 2000 tennis
root@debian10:~# groupmod -n soccer football
root@debian10:~# tail -2 /etc/group
tennis:x:2000:
soccer:x:1010:
root@debian10:~#
```

31.5 groepdel

Groups can be deleted with the **groupdel** command.

```
root@debian10:~# groupdel soccer
root@debian10:~# tail -2 /etc/group
tania:x:1008:
tennis:x:2000:
root@debian10:~#
```

31.6 usermod

The **usermod** command can be used to add a member to a list of groups. Be careful when using **usermod -G** as you will have to list all groups of which you want the user to be a member. The user will be removed from groups not listed. The screenshot below shows how user **tania** is removed from the group **tennis**.

```
root@debian10:~# groupadd football
root@debian10:~# tail -2 /etc/group
tennis:x:2000:
football:x:2001:
root@debian10:~# usermod -G tennis tania
root@debian10:~# usermod -G football tania
root@debian10:~# tail -2 /etc/group
tennis:x:2000:
football:x:2001:tania
root@debian10:~#
```

You can prevent this by using **usermod** with the **-a -G** options.

```
root@debian10:~# usermod -a -G tennis tania
root@debian10:~# tail -2 /etc/group
tennis:x:2000:tania
football:x:2001:tania
root@debian10:~#
```

31.7 newgrp

This topic touches file ownership, which will be discussed later in this book. When you are creating files (for example with **touch** or with **>** redirection), then you are the **owner** of the file, and your primary group is the **group owner** of the file. But if you issue the **newgrp tennis** command, then a new shell is started with **tennis** as your primary group. See the screenshot below, in particular the ownership of the files.

```
root@debian10:~# touch file33
root@debian10:~# ls -l file33
-rw-r--r-- 1 root root 0 Aug 13 14:21 file33
root@debian10:~# newgrp tennis
root@debian10:~# echo $SHLVL
2
root@debian10:~# touch file42
root@debian10:~# ls -l file42
-rw-r--r-- 1 root tennis 0 Aug 13 14:21 file42
root@debian10:~#
```

Note
By default only the **root** user can do this.

31.8 gpasswd

The last command of this little chapter is **gpasswd**. It allows for delegation of groups to a user, so that user can add and remove members to and from a group. The user has to use **gpasswd** for this, not **usermod**.

In the screenshot below **tania** is the delegated account for the group **tennis**. She can then add users to that group with **gpasswd -a**.

```
root@debian10:~# gpasswd -A tania tennis
root@debian10:~# su - tania
tania@debian10:~$ grep tennis /etc/group
tennis:x:2000:tania
tania@debian10:~$ gpasswd -a valentina tennis
Adding user valentina to group tennis
tania@debian10:~$ grep tennis /etc/group
tennis:x:2000:tania,valentina
tania@debian10:~$
```

31.9 backups

Note that the files **/etc/passwd**, **/etc/shadow**, **/etc/group** and **/etc/gshadow** are backed up every day to **/var/backup**. Only **root** can access these backups.

```
paul@debian10:~$ ls -l /var/backups/*.bak
-rw------- 1 root root    960 Sep  5 22:30 /var/backups/group.bak
-rw------- 1 root shadow  803 Sep  5 22:30 /var/backups/gshadow.bak
-rw------- 1 root root   1837 Sep  5 22:30 /var/backups/passwd.bak
-rw------- 1 root shadow 1515 Sep  5 22:30 /var/backups/shadow.bak
paul@debian10:~$
```

31.10 Cheat sheet

Table 31.1: Groups

command	**explanation**
groups	List your group memberships.
groupadd foo	Adds an empty group named **foo** to this computer.
groupmod	Tool to modify properties of groups.
groupdel foo	Delete the group named **foo**.
usermod -G foo bar	Add the user **bar** to the group **foo** (and remove **bar** from all other groups).
usermod -a -G foo bar	Add the user **bar** to the group **foo**.
newgrp	Change the primary group in your current environment.
gpasswd -A foo bar	Make user **foo** an administrator of group **bar**.
/etc/group	Contains the list of all groups and their members on this computer.
/var/backups	Contains backups of /etc/passwd, /etc/group and more.
/etc/cron.daily/passwd	The file that performs the backup (see the Scheduling chapter).

31.11 Practice

1. Verify your group membership on this computer.
2. Add two groups named **music** and **arts**
3. Verify that both groups were created and contain no members.
4. Modify the **gid** of arts to 3000.
5. Modify the name of the group **music** to **artists**.
6. Delete the group artists.
7. Add the user **tania** to the group **arts**, without removing her from other groups.

31.12 Solution

1. Verify your group membership on this computer.

```
groups
```

2. Add two groups named **music** and **arts**

```
su -
groupadd music
groupadd arts
```

3. Verify that both groups were created and contain no members.

```
tail -2 /etc/group
```

4. Modify the **gid** of arts to 3000.

```
groupmod -g 3000 arts
```

5. Modify the name of the group **music** to **artists**.

```
groupmod -n music artists
```

6. Delete the group artists.

```
groupdel artists
```

7. Add the user **tania** to the group **arts**, without removing her from other groups.

```
usermod -a -G arts tania
```

Chapter 32

Everything is a file

32.1 commands are files

When we look at a command with **ls -l**, then we see that the first character is a **-** . This **-** signifies that the file is a regular file. Many commands are regular files, even the bash shell is a regular file.

```
paul@debian10:~$ ls -l $(which cp)
-rwxr-xr-x 1 root root 146880 Feb 28 16:30 /usr/bin/cp
paul@debian10:~$ ls -l $(which bash)
-rwxr-xr-x 1 root root 1168776 Apr 18 06:12 /usr/bin/bash
paul@debian10:~$
```

32.2 directories are files

Directories are a special kind of file, but they are files nonetheless (you can open a directory in **vi**, though I wouldn't recommend changing anything). Directories start with a **d** in an **ls -l** listing.

```
paul@debian10:~$ ls -ld backup/
drwxr-xr-x 2 paul paul 4096 Aug 13 16:52 backup/
paul@debian10:~$ ls -ld .
drwxr-xr-x 10 paul paul 4096 Aug 13 16:52 .
paul@debian10:~$
```

32.3 terminals are files

You can issue the **who am i** command to see the terminal on which you are logged on. A terminal window is a character device (its input and/or output are a stream of characters). Character devices start with a **c** in the output of **ls -l**.

```
paul@debian10:~$ who am i
paul     pts/0        2019-08-13 12:49 (192.168.56.1)
paul@debian10:~$ ls -l /dev/pts/0
crw------- 1 paul tty 136, 0 Aug 13 16:57 /dev/pts/0
paul@debian10:~$ ls -l /dev/tty1
crw------- 1 root tty 4, 1 Aug 13 12:33 /dev/tty1
paul@debian10:~$
```

32.4 block devices are files

In Debian 10 Linux any block device will get a corresponding file in **/dev**. So the hard disk, even an SSD, a USB stick, and the CD-ROM will be visible as a block device. The screenshot shows that the first hard disk, the first partition and the CD-ROM in this computer are **b**lock devices.

```
paul@debian10:~$ ls -l /dev/sda /dev/sda1 /dev/sr0
brw-rw---- 1 root disk   8, 0 Aug 13 10:41 /dev/sda
brw-rw---- 1 root disk   8, 1 Aug 13 10:41 /dev/sda1
brw-rw---- 1 root cdrom 11, 0 Aug 13 10:41 /dev/sr0
paul@debian10:~$
```

Note
A **block device** has an address for each block. An addressed block can be retrieved or written to, this is the opposite of the continuous stream of a **character device**.

Tip
Behind each character or block device is a kernel device driver.

32.5 symbolic links are files

Symbolic links will be explained later in this book, but we can already say that a symlink is also a file. There are many **symbolic links** on a Linux computer.

```
paul@debian10:/lib$ find /usr/lib -exec ls -l {} \; 2>/dev/null | grep ^l | head -3
lrwxrwxrwx  1 root root    21 Aug 12 19:01 cpp -> /etc/alternatives/cpp
lrwxrwxrwx  1 root root    20 Jan 14  2018 libdiscover.so.2 -> libdiscover.so.2.0.1
lrwxrwxrwx  1 root root    19 Apr  8 12:13 sftp-server -> openssh/sftp-server
paul@debian10:/lib$
```

32.6 pipes are files

A **pipe** can be used as way to communicate between processes. We will introduce **named pipes** in the processes chapter. They can be recognised by the letter **p** in **ls -l**. There aren't many **named pipes** on a default Debian 10.

```
paul@debian10:/lib$ find / -exec ls -l {} \; 2>/dev/null | grep ^p
prw-------  1 root root    0 Aug 13 10:41 initctl
prw------- 1 root root 0 Aug 13 10:41 /run/initctl
prw------- 1 root root   0 Aug 13 12:32 12.ref
prw------- 1 root root   0 Aug 13 12:49 16.ref
prw------- 1 root root 0 Aug 13 12:49 /run/systemd/sessions/16.ref
prw------- 1 root root 0 Aug 13 12:32 /run/systemd/sessions/12.ref
paul@debian10:/lib$
```

Note
The above is a very slow **find** command, because it does an **ls -l** on every file.

32.7 IPC sockets are files

Linux domain sockets can be used by processes for inter-process communication. For example the **systemd-journald** will listen to **/run/systemd/journal/dev-log**, which is a socket.

```
paul@debian10:/lib$ ls -l /dev/log
lrwxrwxrwx 1 root root 28 Aug 13 10:41 /dev/log -> /run/systemd/journal/dev-log
paul@debian10:/lib$ ls -l /run/systemd/journal/dev-log
srw-rw-rw- 1 root root 0 Aug 13 10:41 /run/systemd/journal/dev-log
paul@debian10:/lib$
```

32.8 Summary

Below is a table of all the Linux special files that can be recognised by the first character of an **ls -l** command. Note that we did not discuss a **door** because it does not exist in Debian 10.

Table 32.1: List special files

first character	**special file type**
-	regular file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	IPC socket
D	door (Created by Solaris or Linux kernel 2.4.18.)

This list is very similar to the **-type** option of the **find** command. Note however the difference for regular files.

Table 32.2: Find special files

find -type option	**special file type**
f	regular file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	IPC socket
D	door

32.9 Cheat sheet

Table 32.3: Everything is a file

first character in ls -l	**explanation**
-	This file is a regular file.
d	This is a directory.
c	This is a character device.
b	This is a block device.
l	This is a symbolic link.
p	This is a named pipe.
s	This is a socket.
D	This is a door (cannot be created on Debian 10, but some commands like **find** have options for it).

32.10 Practice

1. Log on twice with the same user on the same computer and identify the terminal in each.
2. Use **echo** to send a message from one terminal to the other, as if it was a file.
3. List all character devices in **/dev**.
4. Compare the output of **lsblk** with **ls -l /dev | grep ^b** .
5. Is **/run/initctl** a pipe?

32.11 Solution

1. Log on twice with the same user on the same computer and identify the terminal in each.

```
who am i
```

2. Use **echo** to send a message from one terminal to the other, as if it was a file.

```
echo hello > /dev/pts/1
```

3. List all character devices in **/dev**.

```
ls -l /dev | grep ^c
```

4. Compare the output of **lsblk** with **ls -l /dev | grep ^b** .

```
lsblk
ls -l /dev | grep ^b
```

5. Is **/run/initctl** a pipe?

```
ls -l /run/initctl
```

Chapter 33

File permissions

33.1 File ownership

Every file on Debian Linux has two owners, named a user owner and a group owner. The screenshot below shows the **dates.txt** file being owned by the user **paul** and by the group **paul**.

```
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 paul paul 1118 Jul 27 13:12 dates.txt
paul@debian10:~$
```

This screenshot shows the owners of the **/etc/shadow** file, with **root** as the user owner and the group **shadow** as the group owner.

```
paul@debian10:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1377 Aug 13 12:59 /etc/shadow
paul@debian10:~$
```

33.2 chgrp

The **root** user can use the **chgrp** command to change the **group owner** of a file to another group. The group must exist in **/etc/group**.

```
root@debian10:~# chgrp tennis /home/paul/dates.txt
root@debian10:~# ls -l /home/paul/dates.txt
-rw-r--r-- 1 paul tennis 1118 Jul 27 13:12 /home/paul/dates.txt
root@debian10:~#
```

33.3 chown

The **root** user can use **chown** to change the user owner and to change the group owner in one command. The screenshot below changes both owners of a file.

```
root@debian10:~# chown laura:football /home/paul/dates.txt
root@debian10:~# ls -l /home/paul/dates.txt
-rw-r--r-- 1 laura football 1118 Jul 27 13:12 /home/paul/dates.txt
root@debian10:~#
```

The **paul** user cannot take ownership of the file, even if it is in his home directory, as the following screenshot shows.

```
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 laura football 1118 Jul 27 13:12 dates.txt
paul@debian10:~$ chown paul:paul dates.txt
chown: changing ownership of dates.txt: Operation not permitted
paul@debian10:~$
```

33.4 Regular files

Regular files (those that start with a **-** in **ls -l**) have permissions in the form of **rwxrwxrwx**. Three triplets of **r**ead, **w**rite, and e**x**ecute permissions.

The first triplet are the permissions for the **user owner**. The second triplet are the permissions for the **group owner** (for all the members of that group). The third triplet are the permissions for **other** users.

Table 33.1: file permissions table

letter	short	permission
r	read	Can read the file with **cat**, **more**, ...
w	write	Can write to the file with **vi**, **>**, ...
x	execute	Can execute (run) the file.

In the screenshot below the **laura** user has read and write permissions, the members of the **football** group have read permissions, and the others also have read.

```
paul@debian10:~$ ls -l dates.txt
-rw-r--r-- 1 laura football 1118 Jul 27 13:12 dates.txt
paul@debian10:~$
```

In the screenshot below the **paul** user has read, write and execute permissions, the **paul** group has read and execute permissions, and the others have no permissions (others cannot read, write or execute the **forloop4.sh** file).

```
paul@debian10:~$ ls -l forloop4.sh
-rwxr-x--- 1 paul paul 79 Aug  9 13:54 forloop4.sh
paul@debian10:~$
```

In case you are the user owner, and you are a member of the group owner, then the user owner permissions prevail.

33.5 Directories

Permissions on directories (those starting with **d** in **ls -l**) have a different meaning than permissions on regular files. Generally **r** and **x** are given to be able to enter the directory with **cd** and list the files with **ls**. The **w** permission on directories is given when you need to be able to create files in the directory.

Table 33.2: directory permissions table

letter	short	permission
r	read	Together with **x** allows for reading with **ls**, ...
w	write	Can create files in the directory.
x	execute	Can enter the directory with **cd**.

33.6 chmod

Permissions on a regular file or directory can be changed with the **chmod** command (which is short for **ch**ange **mod**e). The **chmod** command can add, remove or set permissions for the **u**ser owner, **g**roup owner and **o**thers.

Table 33.3: chmod permissions

character	short	permission
u	user	user owner
g	group	group owner
o	other	others
+	add	add permission(s) to file or directory

Table 33.3: (continued)

character	short	permission
-	remove	remove permission(s) from file or directory
=	set	set permissions on file or directory
r	read	read permission
w	write	write permission
x	execute	execute permission

For example to set permissions to rwxr-xr-- the following command can be used.

```
root@debian10:/home/paul# ls -l dates.txt
-rw-r--r-- 1 laura football 1118 Jul 27 13:12 dates.txt
root@debian10:/home/paul#
root@debian10:/home/paul# chmod u=rwx,g=rx,o=r dates.txt
root@debian10:/home/paul# ls -l dates.txt
-rwxr-xr-- 1 laura football 1118 Jul 27 13:12 dates.txt
root@debian10:/home/paul#
```

To simply add permission for the **others**, you can issue this command.

```
root@debian10:/home/paul# chmod o+rwx dates.txt
root@debian10:/home/paul# ls -l dates.txt
-rwxr-xrwx 1 laura football 1118 Jul 27 13:12 dates.txt
root@debian10:/home/paul#
```

You can combine the owners in the **chmod** command, if they need the same permission change. And you can clear permissions by omitting them, as shown in this screenshot.

```
root@debian10:/home/paul# chmod ug=rw,o= dates.txt
root@debian10:/home/paul# ls -l dates.txt
-rw-rw---- 1 laura football 1118 Jul 27 13:12 dates.txt
root@debian10:/home/paul#
```

And instead of writing **ugo** for all three permissions sets, you can simply write **a** for all.

```
root@debian10:/home/paul# chmod a+rwx dates.txt
root@debian10:/home/paul# ls -l dates.txt
-rwxrwxrwx 1 laura football 1118 Jul 27 13:12 dates.txt
root@debian10:/home/paul#
```

33.7 old school chmod

While the previous section with **chmod ugo+-=rwx** seems straightforward, most people on Linux use **octal numbers** to set permissions. This is quicker to type and easy to get used to.

Table 33.4: chmod octal permissions

octal number	rwx equivalent
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

For regular files the most common use cases are 6, 4 and 0. We use these three in the following screenshot. The **6** means **rw** for user owner, the **4** means **r** for group owner and the **0** removes all permissions for others.

```
paul@debian10:~$ chmod 640 error.txt
paul@debian10:~$ ls -l error.txt
-rw-r----- 1 paul paul 31537 Aug  4 14:59 error.txt
paul@debian10:~$
```

And for directories the most common use cases are 7, 5 and 0. See the following example which grants **rwx** for the user owner, **r-x** for the group owner and nothing for the others.

```
paul@debian10:~$ chmod 750 renamedir/
paul@debian10:~$ ls -ld renamedir/
drwxr-x--- 2 paul paul 4096 Aug  4 21:39 renamedir/
paul@debian10:~$
```

The **chmod** command can be used to **recursively** change permissions on all files in all subdirectories using the **-R** option, as seen in this example.

```
paul@debian10:~/recu$ find . -type f -exec ls -l {} \;
-rw-r--r-- 1 paul paul 0 Aug 14 00:34 ./subdir/file33
-rw-r--r-- 1 paul paul 0 Aug 14 00:34 ./file42
paul@debian10:~/recu$ chmod -R 770 *
paul@debian10:~/recu$ find . -type f -exec ls -l {} \;
-rwxrwx--- 1 paul paul 0 Aug 14 00:34 ./subdir/file33
-rwxrwx--- 1 paul paul 0 Aug 14 00:34 ./file42
paul@debian10:~/recu$
```

33.8 stat

The **stat** command will display permissions in octal and in human readable form, together with the uid and gid.

```
paul@debian10~$ stat recu/file42 | grep Access | head -1
Access: (0770/-rwxrwx---)  Uid: ( 1000/    paul)   Gid: ( 1000/    paul)
paul@debian10~$
```

33.9 umask

Where do the default permissions come from? For example in the following screenshot we create a new file and a new directory, and they get default permissions.

```
paul@debian10:~/umask$ touch file42
paul@debian10:~/umask$ mkdir newdir
paul@debian10:~/umask$ ls -l
total 4
-rw-r--r-- 1 paul paul    0 Aug 14 00:39 file42
drwxr-xr-x 2 paul paul 4096 Aug 14 00:39 newdir
paul@debian10:~/umask$
```

They come from the **umask** value, which can be displayed with the **umask** command.

```
paul@debian10:~/umask$ umask
0022
paul@debian10:~/umask$
```

The first zero in the **umask** is to signify that this is an **octal** number, so the octal number we receive is **022**. These are the permissions that you **do not** get when creating files and directories.

Let's verify this for the directory we just created. The maximum permission is 777, minus the **umask** 022 equals 755, which is exactly what we got when creating the **newdir** above.

It is the same for regular files, except that the maximum permission for files is 666, because files are **never** executable by default.

The **umask** value is set in **/etc/login.defs**, but can be changed in any of the profile scripts or even on the fly by typing **umask** followed by the desired value.

```
paul@debian10:~$ umask
0022
paul@debian10:~$ umask 002
paul@debian10:~$ umask
0002
paul@debian10:~$
```

33.10 mkdir -m

There is an option **-m** to **mkdir** which allows for you to specify the permissions on that directory at creation time. See this for example.

```
paul@debian10:~/umask$ mkdir -m 700 mydir
paul@debian10:~/umask$ ls -ld mydir/
drwx------ 2 paul paul 4096 Aug 14 00:50 mydir/
paul@debian10:~/umask$
```

33.11 root and permissions

Permissions do not apply to the **root** user. The **root** user can always read from and write to regular files and directories.

33.12 Cheat sheet

Table 33.5: Permissions

command	explanation
ls -l foo	Display user owner and group owner of a file named **foo**.
chgrp bar foo	Change the group owner of a file named **foo** to **bar**.
chown bar:bar foo	Change both owners of the file named **foo** to **bar**.
chmod	Tool to change the permissions (the **mod**e) of a file.
chmod u=r foo	Sets the **user owner** permission to **read** (for the file **foo**)
chmod g+rw foo	Adds **read** and **write** permissions for the **group** owner.
chmod o-x foo	Removes the **execute** permission for **others**.
chmod 777 foo	Sets 777 octal permissions on the file named **foo**.
chmod -R	Recursively sets permissions for all files in all subdirectories.
stat foo	Display owners and permissions (and more) information about the file **foo**.
umask	Display the default permission mask (for your current environment!).
umask 002	Set the default permission mask (for your current environment!).
mkdir -m 700 foo	Create the directory **foo** with octal 700 permissions.

33.13 Practice

1. Display the user owner and group owner of the **/var/log/auth.log** file .
2. Change the group owner of the **wolf.png** file to the group **tennis**.
3. Change the user owner of the **wolf.png** file to the user **root**.
4. Change the permissions on the **Linux.pdf** file to **r--r-----** .
5. Change the permissions on the **Linux.pdf** file to **r--r-----** using octal notation.
6. Create a directory **~/newdir** with 700 permissions.
7. Display the **umask**.
8. Does the **~/.profile** script change the **umask**?

33.14 Solution

1. Display the user owner and group owner of the **/var/log/auth.log** file .

```
ls -l /var/log/auth.log
```

2. Change the group owner of the **wolf.png** file to the group **tennis**.

```
su -
chgrp tennis wolf.png
```

3. Change the user owner of the **wolf.png** file to the user **root**.

```
su -
chown root wolf.png
```

4. Change the permissions on the **Linux.pdf** file to **r--r-----** .

```
chmod ug=r,o= Linux.pdf
```

5. Change the permissions on the **Linux.pdf** file to **r--r-----** using octal notation.

```
chmod 440 Linux.pdf
```

6. Create a directory **~/newdir** with 700 permissions.

```
mkdir -m 700 ~/newdir
```

7. Display the **umask**.

```
umask
```

8. Does the **~/.profile** script change the **umask**?

```
grep umask ~/.profile
```

Chapter 34

Advanced file permissions

34.1 sticky bit

A **sticky bit** is a special permission bit that you can set on a directory so that users can only remove files of which they are the user owner. This effectively prevents users from deleting files from other users, even if they have 777 permissions on the directory and on the file.

The **sticky bit** is set using **chmod +t** on the directory.

```
root@debian10:~# mkdir /srv/project42
root@debian10:~# ls -l /srv
total 4
drwxr-xr-x 2 root root 4096 Aug 14 02:23 project42
root@debian10:~# chmod +t /srv/project42
root@debian10:~# ls -l /srv
total 4
drwxr-xr-t 2 root root 4096 Aug 14 02:23 project42
root@debian10:~#
```

You can see the **sticky bit** as a letter **t** (if the x is also there) or as an uppercase **T** (in case x is not set). You can also use the octal notation **1** to set permission and sticky bit in one command.

```
root@debian10:~# chmod 1750 /srv/project42/
root@debian10:~# ls -l /srv
total 4
drwxr-x--T 2 root root 4096 Aug 14 02:23 project42
root@debian10:~#
```

By default the **sticky bit** is set on the **/tmp** directory. You can find all directories with a sticky bit using this **find** command.

```
root@debian10:~# find / -type d -perm -1000 2>/dev/null
```

34.2 setgid directory

The **setgid bit** is a special permission bit that can be set on a directory to force group ownership on all files created in that directory. The purpose is to share this directory in a group project, where all files created in this directory belong to the group project.

The **setgid bit** on a directory can be set using **chmod g+s**, as seen in this screenshot.

```
root@debian10:~# ls -l /srv
total 4
drwxrwx--T 2 root root 4096 Aug 14 02:23 project42
root@debian10:~# chmod g+s /srv/project42/
root@debian10:~# ls -l /srv
total 4
drwxrws--T 2 root root 4096 Aug 14 02:23 project42
root@debian10:~#
```

You can **find** all directories with a **setgid bit** using this find command.

```
root@debian10:~# find / -type d -perm -2000 2>/dev/null
```

34.3 project directories

It is common for project directories (that are shared over the network) to have a common group owner and to not be able to remove other files (as in both **sticky bit** and **setgid bit** set on the directory). This can be done using a one **chmod** command.

```
root@debian10:~# chmod 3770 /srv/project42/
root@debian10:~# ls -l /srv/
total 4
drwxrxs--T 2 root root 4096 Aug 14 02:23 project42
root@debian10:~#
```

Of course to make this example complete, we also need to create a group for this **project42** and make that group the owner of the directory.

```
root@debian10:~# groupadd project42
root@debian10:~# chgrp project42 /srv/project42/
root@debian10:~# ls -l /srv
total 4
drwxrxs--T 2 root project42 4096 Aug 14 02:23 project42
root@debian10:~#
```

Now when we make a user a member of the group **project42** then that user can create files in **/srv/project42** that automatically belong to the group **project42**. And she can only delete her own files.

```
root@debian10:~# usermod -a -G project42 valentina
root@debian10:~# su - valentina
valentina@debian10:~$ touch /srv/project42/test.txt
valentina@debian10:~$ ls -l /srv/project42/
total 0
-rw-r--r-- 1 valentina project42 0 Aug 14 03:05 test.txt
valentina@debian10:~$
```

34.4 setuid binary

The **setuid bit** is a special permission bit that can be set on an executable file. The **setuid bit** is visible as an **s** in the user owner triplet. It is set by default on some binaries in **/usr/bin/**.

```
valentina@debian10:~$ find /usr/bin/ -type f -perm -4000 -exec ls -l {} \;
-rwsr-xr-x 1 root root 34888 Jan 10  2019 /usr/bin/umount
-rwsr-xr-x 1 root root 84016 Jul 27  2018 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 63568 Jan 10  2019 /usr/bin/su
-rwsr-xr-x 1 root root 63736 Jul 27  2018 /usr/bin/passwd
-rwsr-xr-x 1 root root 44440 Jul 27  2018 /usr/bin/newgrp
-rwsr-xr-x 1 root root 54096 Jul 27  2018 /usr/bin/chfn
-rwsr-xr-x 1 root root 44528 Jul 27  2018 /usr/bin/chsh
-rwsr-xr-x 1 root root 51280 Jan 10  2019 /usr/bin/mount
valentina@debian10:~$
```

Setting this bit will make sure that a binary is started with the credentials of the **owner** of the binary instead of the user starting the binary. Normally any program you start will run with the credentials of your user account. Not so for **setuid bit** files, they **always** run with owner credentials.

For example **/usr/bin/passwd**, the command to change your password. This change has to be done in **/etc/shadow**, but a normal user has no permissions to write changes to this file. So when you type **passwd**, you become **root** on the computer for as long as the **passwd** command is running.

```
valentina@debian10:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1482 Aug 14 02:43 /etc/shadow
valentina@debian10:~$ passwd
Changing password for valentina.
Current password:
New password:
Retype new password:
```

```
passwd: password updated successfully
valentina@debian10:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1482 Aug 14 03:24 /etc/shadow
valentina@debian10:~$
```

Note
Note the time stamp of the **shadow file** in the above screenshot. The **valentina** user changed a file without having permissions on that file.

34.5 setgid binary

The **setgid bit** is equivalent to the **setuid bit** on binaries in that it preserves group credentials when executing the file in question. The **setgid bit** is visible as an **s** in the group owner triplet.

You can **find** all the **setgid** executables using this command.

```
root@debian10:~# find / -type f -perm -2000 2>/dev/null
/usr/bin/ssh-agent
/usr/bin/bsd-write
/usr/bin/wall
/usr/bin/chage
/usr/bin/crontab
/usr/bin/dotlockfile
/usr/bin/expiry
/usr/sbin/unix_chkpwd
root@debian10:~#
```

We will look at **crontab** in the Scheduling chapter. You should remember **chage** from the Password management chapter. The **chage** command can read fields in the **/etc/shadow** file thanks to its **setgid bit**.

```
paul@debian10:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1482 Aug 14 03:24 /etc/shadow
paul@debian10:~$ ls -l /usr/bin/chage
-rwxr-sr-x 1 root shadow 71816 Jul 27  2018 /usr/bin/chage
paul@debian10:~$
```

34.6 attributes

Files on a Debian Linux have **attributes** which can have several purposes. A list is available in the man page of the **chattr** command. We will discuss two attribute flags.

34.6.1 immutable

The **root** user can decide to set the **immutable** flag on a file. This file can no longer be renamed, moved or changed in any way.

```
root@debian10:~# touch imfile
root@debian10:~# chattr +i imfile
root@debian10:~# ls -l imfile
-rw-r--r-- 1 root root 0 Aug 14 04:10 imfile
root@debian10:~# lsattr imfile
----i---------e---- imfile
root@debian10:~# rm -rf imfile
rm: cannot remove imfile: Operation not permitted
root@debian10:~#
```

Use **chattr -i** to remove the immutable flag.

```
root@debian10:~# chattr -i imfile
root@debian10:~# rm -rf imfile
root@debian10:~#
```

34.6.2 appendable

Another flag that may be useful is the **appendable** flag. With this **appendable** flag the file can not be renamed or moved, but data can be appended to it, so this can be set on a log file.

```
root@debian10:~# touch apfile
root@debian10:~# chattr +a apfile
root@debian10:~# mv apfile test
mv: cannot move 'apfile' to 'test': Operation not permitted
root@debian10:~# echo "The answer is 42" > apfile
-bash: apfile: Operation not permitted
root@debian10:~# echo "The answer is 42" >> apfile
root@debian10:~# rm -rf apfile
rm: cannot remove 'apfile': Operation not permitted
root@debian10:~#
```

Warning
Don't randomly set this on existing log files as **logrotate** will fail.

Note
By default there are no files on Debian Linux with **appendable** or **immutable** flag.

34.7 Cheat sheet

Table 34.1: Advanced permissions

command	**explanation**
chmod +t	Set the sticky bit.
chmod g+s	Set the setgid bit.
chmod 3770	Set the sticky bit and the setgid bit and 770 octal permissions.
chmod ug+s	Set the setuid and setgid bit (on an executable file).
find / -perm 1000	Find all files with sticky bit set.
find / -perm 2000	Find all files with setgid bit set.
find / -perm 4000	Find all files with setuid bit set.
chattr	Tool to set attributes on a file.
chattr +i	Set the **immutable** attribute.
chattr -i	Unset the **immutable** attribute.
chattr +a	Set the **appendable** attribute.

34.8 Practice

1. Create a **project33** directory and make sure users can only delete their own files.
2. Make sure the group owner **pro33** is set for all files created in the **project33** directory.
3. Test with the **laura** user that all files created in the **project33** directory have the correct group owner.
4. Find all **setuid** binaries in **/usr/bin** .
5. Display the **setgid** bit on a binary in **/usr/bin** .
6. List the attributes of **/etc/passwd** .
7. What is the result of setting the **immutable** flag on **/etc/shadow**.

34.9 Solution

1. Create a **project33** directory and make sure users can only delete their own files.

```
mkdir /srv/project33
chmod +t /srv/project33
```

2. Make sure the group owner **pro33** is set for all files created in the **project33** directory.

```
groupadd pro33
chgrp pro33 /srv/project33
chmod g+s /srv/project33
```

3. Test with the **laura** user that all files created in the **project33** directory have the correct group owner.

```
usermod -a -G pro33 laura
su - laura
touch /srv/project33/test
ls -l /srv/project33/
```

4. Find all **setuid** binaries in **/usr/bin** .

```
find /usr/bin -type f -perm -4000 \;
```

5. Display the **setgid** bit on a binary in **/usr/bin** .

```
find /usr/bin -type f -perm -2000 \;
ls -l /usr/bin/crontab
```

6. List the attributes of **/etc/passwd** .

```
lsattr /etc/passwd
```

7. What is the result of setting the **immutable** flag on **/etc/shadow**.

```
Passwords cannot be changed (and no new user passwords can be added).
```

Chapter 35

Access control lists

Warning
Debian 10 Linux has no **ACLs** enabled after a default installation. You will have to perform some commands as **root** to enable **ACLs**. These commands are explained in the Storage chapters and in the Managing software chapter. You may want to study the Storage chapters and come back here when you can mount filesystems. Or you may continue here with this chapter, after blindly executing these three commands as **root** .

```
apt-get update
apt-get install acl
mount -o remount,acl /
```

35.1 About

With **ACLs** enabled on a filesystem, you can assign permissions to many different users and groups on individual files or directories. We will first take a look at recognising **ACLs** and then explain how to set them.

35.2 ls

Consider the example in the screenshot below and the question: Why can the user **tania** still read this file? Can you see the difference with all screenshots from before this chapter?

```
paul@debian10~$ chmod 770 newfile
paul@debian10~$ ls -l newfile
-rwxrwx---+ 1 paul paul 0 Aug 15 17:53 newfile
paul@debian10~$
```

There is new character in the screenshot above, one that we have not seen yet. After the permissions there is a **+**-sign. This means that beside the standard Linux permissions, there is also an **ACL** active on this file.

Note that the **stat** command has no info at all on the presence of an **ACL**.

```
paul@debian10~$ stat newfile
  File: newfile
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 393399      Links: 1
Access: (0770/-rwxrwx---)  Uid: ( 1000/    paul)   Gid: ( 1000/    paul)
Access: 2019-08-15 18:34:52.854530983 +0200
Modify: 2019-08-15 17:53:42.610530930 +0200
Change: 2019-08-15 18:19:40.866530983 +0200
 Birth: -
paul@debian10~$
```

35.3 getfacl

The **getfacl** command will display the **ACL** that is present on this file. The example shows an entry for the user **tania** and two entries for the groups **tennis** and **football**.

```
paul@debian10~$ getfacl newfile
# file: newfile
# owner: paul
# group: paul
user::rwx
user:tania:r-x
group::r--
group:tennis:r-x
group:football:rwx
mask::rwx
other::---

paul@debian10~$
```

The above screenshot means that **tania** can **r-x newfile**, same as the group members of the **tennis** group. And the **football** group gets all permissions on this **newfile**.

35.4 setfacl

The **setfacl** command allows us to set **ACLs** on files and directories. Again we learn by example, so the first screenshot will set 7 permissions (=rwx) for a user named **valentina**.

```
paul@debian10~$ echo hello > oldfile
paul@debian10~$ ls -l oldfile
-rw-r--r-- 1 paul paul 6 Aug 15 19:06 oldfile
paul@debian10~$ setfacl -m u:valentina:7 oldfile
paul@debian10~$ getfacl oldfile | grep valentina
user:valentina:rwx
paul@debian10~$
```

The **-m** option to **setfacl** is to **modify** an ACL, which means existing ACL antries will remain. Now even after a **chmod 770** the user **valentina** will have access to the file, as can be seen in this screenshot.

```
paul@debian10~$ chmod 770 oldfile
paul@debian10~$ su - valentina
Password:
valentina@debian10:~$ cat /home/paul/oldfile
hello
valentina@debian10:~$
```

Note that **setfacl** can also be used to set the standard **rwx** permissions for user, group and other. The screenshot below uses **setfacl** but no ACL is present.

```
root@debian10:~# mkdir /srv/project33
root@debian10:~# chown root:pro33 /srv/project33/
root@debian10:~# setfacl -m u::rwx,g::rwx,o:- /srv/project33/
root@debian10:~# ls -ld /srv/project33/
drwxrwx--- 2 root pro33 4096 Aug 17 18:24 /srv/project33/
root@debian10:~#
```

The **setfacl** command above is identical to the one here below, which uses octal permissions.

```
root@debian10:~# setfacl -m u::7,g::7,o:0 /srv/project33/
root@debian10:~# ls -ld /srv/project33/
drwxrwx--- 2 root pro33 4096 Aug 17 18:24 /srv/project33/
root@debian10:~#
```

35.5 setfacl --set

We can set all permissions, the classic rwx ones, and all the ACL entries in one command with **setfacl --set**. This command will erase all existing ACL entries, so it has to be complete.

```
root@debian10:~# setfacl --set u::7,g::7,u:tania:r,o::-,g:pro33:7 /srv/project33/
root@debian10:~# getfacl /srv/project33/
getfacl: Removing leading / from absolute path names
# file: srv/project33/
# owner: root
# group: root
user::rwx
user:tania:r--
group::rwx
group:pro33:rwx
mask::rwx
other::---

root@debian10:~#
```

The example above is equal to **chmod 770 /srv/project33** followed by **setfacl -m u:tania:r,g:pro33:7 /srv/project33** .

35.6 mask

You may have noticed the **mask** entry in the output of the previous **getfacl** commands. This **mask** is set automatically when using **setfacl**, but can be changed afterwards by a **chmod** or a **setfacl** command.

Below a **chmod 750**, which sets the **mask** to **r-x**. The output of **getfacl** will then list the ACL entries, followed by an **effective** mask.

```
root@debian10:~# chmod 750 /srv/project33/
root@debian10:~# getfacl /srv/project33/
getfacl: Removing leading / from absolute path names
# file: srv/project33/
# owner: root
# group: root
user::rwx
user:tania:r--
group::rwx                      #effective:r-x
group:pro33:rwx                 #effective:r-x
mask::r-x
other::---

root@debian10:~#
```

Using **setfacl** once is enough to recalculate the **mask** so all ACLs apply. So be careful when mixing **chmod** and **setfacl** commands.

```
root@debian10:~# setfacl -m u:tania:7 /srv/project33/
root@debian10:~# getfacl /srv/project33/
getfacl: Removing leading / from absolute path names
# file: srv/project33/
# owner: root
# group: root
user::rwx
user:tania:rwx
group::rwx
group:pro33:rwx
mask::rwx
other::---

root@debian10:~#
```

35.7 default acls

You may want to set an ACL on a directory, and have all the files and subdirectories automatically inherit the ACL. This can be done by marking the ACL as **default**, using the **-d** option to **setfacl**. Consider this example.

```
root@debian10:~# rm -rf /srv/project33/
root@debian10:~# mkdir /srv/project33/
root@debian10:~# chown root:pro33 /srv/project33/
root@debian10:~# setfacl -d --set u::7,g::7,o::-,g:pro33:7,u:tania:7 /srv/project33/
root@debian10:~# mkdir /srv/project33/newdir
root@debian10:~# getfacl /srv/project33/newdir
getfacl: Removing leading / from absolute path names
# file: srv/project33/newdir
# owner: root
# group: root
user::rwx
user:tania:rwx
group::rwx
```

```
group:pro33:rwx
mask::rwx
other::---
default:user::rwx
default:user:tania:rwx
default:group::rwx
default:group:pro33:rwx
default:mask::rwx
default:other::---

root@debian10:~#
```

35.8 getfacl | setfacl

The output of the **getfacl** command can serve as input for the **setfacl** command, which allows for copying of the ACL from one file or directory to another.

```
root@debian10:~# getfacl /srv/project33/ | setfacl --set-file=- /srv/project42/
getfacl: Removing leading / from absolute path names
root@debian10:~# getfacl /srv/project42/
getfacl: Removing leading / from absolute path names
# file: srv/project42/
# owner: root
# group: project42
# flags: -st
user::rwx
group::rwx
other::---
default:user::rwx
default:user:tania:r--
default:group::rwx
default:group:pro33:rwx
default:mask::rwx
default:other::---

root@debian10:~#
```

Note
setfacl --set-file is similar to **-d --set** in that it has to be complete because it erases an existing ACL and it sets a default ACL.

You can also put the output of **getfacl** in a file, and use that file later with **setfacl** to set the same permissions and ACL. This works recursively when using the **-R** option to **getfacl**.

```
root@debian10:~# getfacl -R /srv > srv.acls
getfacl: Removing leading / from absolute path names
root@debian10:~# cd /
root@debian10:/# setfacl --restore /root/srv.acls
root@debian10:/#
```

35.9 /etc/fstab

To be able to use **ACLs** the filesystem has to be mounted with the **acl** mount option. Mount options are explained in the Storage chapter.

35.10 Cheat sheet

Table 35.1: Access control lists

command	**explanation**
+	The sign that **ls** shows when there is an ACL.
getfacl foo	Read the ACL of the file named **foo**.
setfacl -m u:bar:7 foo	Set octal 7 permissions for user **bar** on file **foo**.
setfacl --set	Set all permissions and ACL's.
setfacl -d	Set a default (=inheritable) ACL.

35.11 Practice

1. Create a directory for **project8472**. Also create a group named **pro8472** for this project and make the group owner of the directory.
2. Create a group named **pro42** and give them read and execute access on the **project8472** directory, while retaining access for the **pro8472** group.
3. Verify with **ls -l** that there is a plus-sign behind the permissions.
4. Verify with **getfacl** that the ACL contains two different groups.
5. Use **chmod 700** on the project8472 directory, is the ACL still active?
6. Is the ACL restored after a **chmod 770** on the project8472 directory?
7. Store the ACL in a file.
8. Remove the ACL from the project8472 directory.
9. Restore the ACL using the backup file.

35.12 Solution

1. Create a directory for **project8472**. Also create a group named **pro8472** for this project and make the group owner of the directory.

```
mkdir /srv/project8472
groupadd pro8472
chgrp pro8472 /srv/project8472
```

2. Create a group named **pro42** and give them read and execute access on the **project8472** directory, while retaining access for the **pro8472** group.

```
groupadd pro42
setfacl -m g:pro42:rx /srv/project8472
```

3. Verify with **ls -l** that there is a plus-sign behind the permissions.

```
ls -l /srv/project8472
```

4. Verify with **getfacl** that the ACL contains two different groups.

```
getfacl /srv/project8472
```

5. Use **chmod 700** on the project8472 directory, is the ACL still active?

```
chmod 700 /srv/project8472
You should see 'effective:---' when using getfacl on this directory.
```

6. Is the ACL restored after a **chmod 770** on the project8472 directory?

```
chmod 770 /srv/project8472
getfacl /srv/project8472
```

7. Store the ACL in a file.

```
getfacl /srv/project8472 > file.acl
```

8. Remove the ACL from the project8472 directory.

```
setfacl -b /srv/project8472
```

9. Restore the ACL using the backup file.

```
setfacl --set-file=file.acl /srv/project8472
```

Chapter 36

Links

36.1 inode

Every file on an **ext4** filesystem, which is the default on Debian 10 Linux, has an **inode** in the **inode table**. Each **inode** on a filesystem has a unique identification number, called the **inode number**. In this **inode** there is meta-information about the file like the permissions, the owners, the last access time, a pointer to the content of the file; basically every piece of meta information about the file, except for the file name (and the actual file contents).

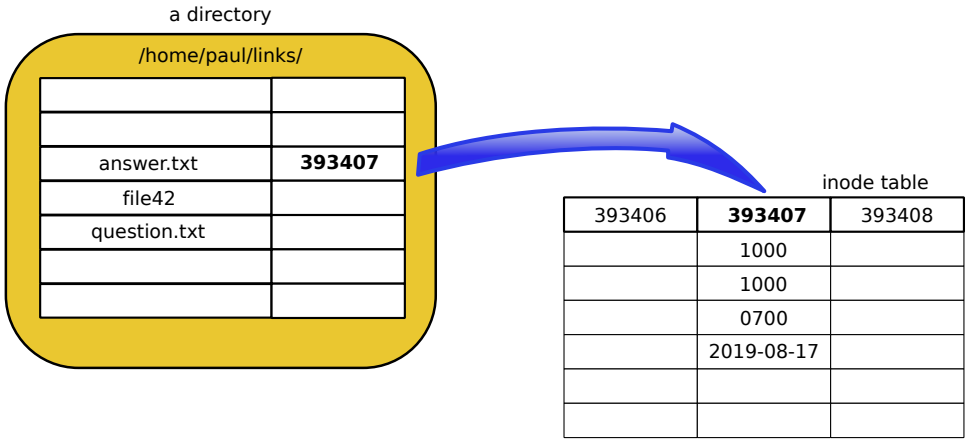
Consider this example of a directory **/home/paul/links** with three files.

```
paul@debian10:~/links$ ls
answer.txt  file42  question.txt
paul@debian10:~/links$
```

The **inode** number of a file can be seen using **ls -li** . The **inode** number of the **answer.txt** file is 393407 in this screenshot.

```
paul@debian10:~/links$ ls -li answer.txt
393407 -rwx------ 2 paul paul 56 Aug 17 21:33 answer.txt
paul@debian10:~/links$
```

The file name is located in the **directory**, together with its **inode number**. A **directory** is in fact a list of filenames with **inode numbers** as pointers to files. So in this case the **/home/paul/links/** directory has an entry that reads **answer.txt 393407**.

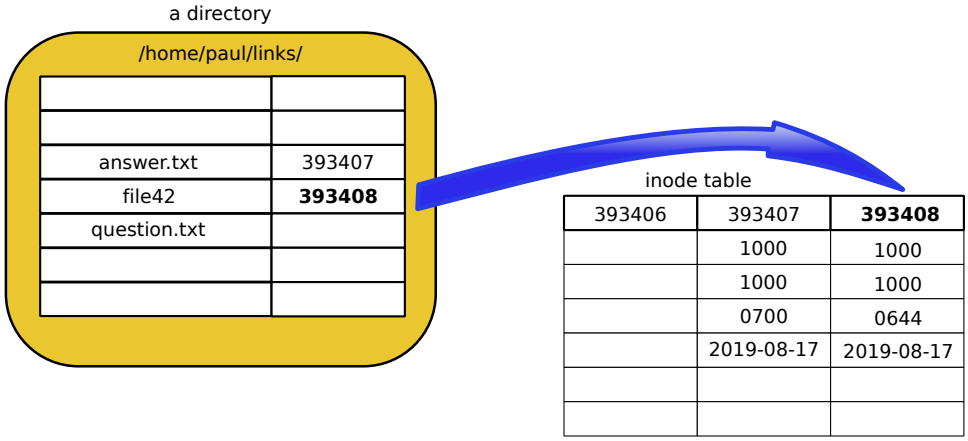


Many fields in the **inode** can be seen with the **stat** command, only some fields are shown in the drawing above.

```
paul@debian10:~/links$ stat answer.txt
  File: answer.txt
  Size: 56              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 393407      Links: 2
Access: (0700/-rwx------)  Uid: ( 1000/    paul)   Gid: ( 1000/    paul)
Access: 2019-08-17 21:33:26.380957381 +0200
Modify: 2019-08-17 21:33:26.380957381 +0200
Change: 2019-08-17 21:35:28.096364280 +0200
 Birth: -
paul@debian10:~/links$
```

The file named **file42** was created right after **answer.txt** and it got the next available **inode**, as can be seen in this screenshot.

```
paul@debian10:~/links$ ls -li file42
393408 -rw-r--r-- 1 paul paul 0 Aug 17 23:10 file42
paul@debian10:~/links$
```



As you can see here, the permissions **644** are stored in the **inode** together with all the other meta-information.

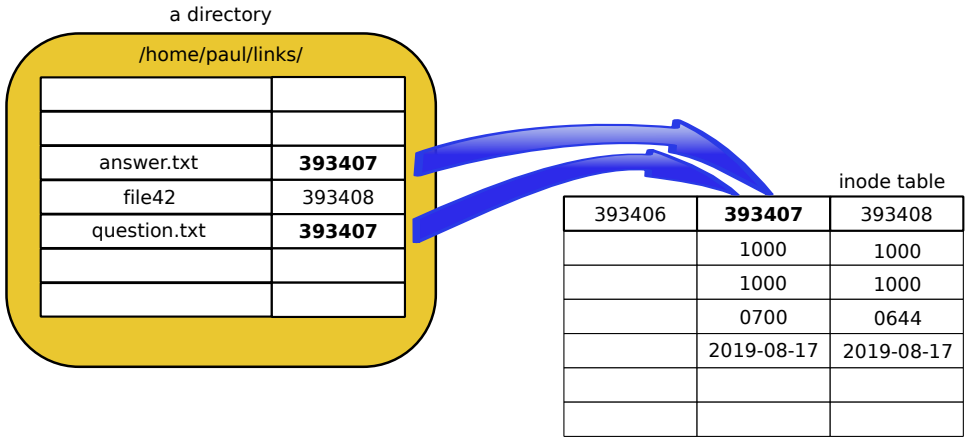
36.2 hard link

Every file has a unique **inode** number, except for hard links. When two distinct filenames point to the same **inode**, then we call this two hard links to the same file. Remember all the information about the file is in the **inode**, so when two filenames point to the same **inode**, then they share permissions, owners, content, access time etcetera, only the filename is different.

Consider this example of two files with the same **inode**. These two files share everything.

```
paul@debian10:~/links$ ls -li answer.txt question.txt
393407 -rwx------ 2 paul paul 56 Aug 17 21:33 answer.txt
393407 -rwx------ 2 paul paul 56 Aug 17 21:33 question.txt
paul@debian10:~/links$
```

In a drawing this looks like this in the directory, with two pointers pointing to the same **inode**.



When we execute a **chmod** command on one of them, then the other will also change, because it points to the same **inode** and the permissions are inside the **inode**.

```
paul@debian10:~/links$ chmod 640 question.txt
paul@debian10:~/links$ ls -li answer.txt question.txt
393407 -rw-r----- 2 paul paul 56 Aug 17 21:33 answer.txt
393407 -rw-r----- 2 paul paul 56 Aug 17 21:33 question.txt
paul@debian10:~/links$
```

There is also no difference between the two files. One is not pointing to the other, they are both pointing to the same **inode**.

36.3 ln

Hard links to an existing **inode** can be created with the **ln** command. In the screenshot below we create a third hard link for the existing **answer.txt** and **question.txt** files.

```
paul@debian10:~/links$ ln answer.txt third
paul@debian10:~/links$ ls -li answer.txt question.txt third
393407 -rw-r----- 3 paul paul 56 Aug 17 21:33 answer.txt
393407 -rw-r----- 3 paul paul 56 Aug 17 21:33 question.txt
393407 -rw-r----- 3 paul paul 56 Aug 17 21:33 third
paul@debian10:~/links$
```

It doesn't matter whether we use **answer.txt** or **question.txt** to create the new hard link, because all hard links are equal. Data added to one of the three hard linked files will be added to all hard linked files.

```
paul@debian10:~/links$ echo "The quick brown fox jumped over the lazy dog." >> third
paul@debian10:~/links$ ls -l answer.txt question.txt third
-rw-r----- 3 paul paul 102 Aug 18 15:23 answer.txt
-rw-r----- 3 paul paul 102 Aug 18 15:23 question.txt
-rw-r----- 3 paul paul 102 Aug 18 15:23 third
paul@debian10:~/links$
```

Did you notice that the number before the owners in **ls -l** is counting the number of **hard links**? Let's create a hard link to these three files in another directory.

```
paul@debian10:~/links$ ln question.txt /home/paul/backup/fourth_hardlink
paul@debian10:~/links$ ls -li answer.txt question.txt third
393407 -rw-r----- 4 paul paul 102 Aug 18 15:23 answer.txt
393407 -rw-r----- 4 paul paul 102 Aug 18 15:23 question.txt
393407 -rw-r----- 4 paul paul 102 Aug 18 15:23 third
paul@debian10:~/links$
```

Voila, it says four hard links exist now to these files.

36.4 find hard links

To **find** all hard links to a file, you can use the **find** command with the **-inum** option, as this screenshot shows.

```
paul@debian10:~/links$ find ~ -inum 393407
/home/paul/links/question.txt
/home/paul/links/answer.txt
/home/paul/links/third
/home/paul/backup/fourth_hardlink
paul@debian10:~/links$
```

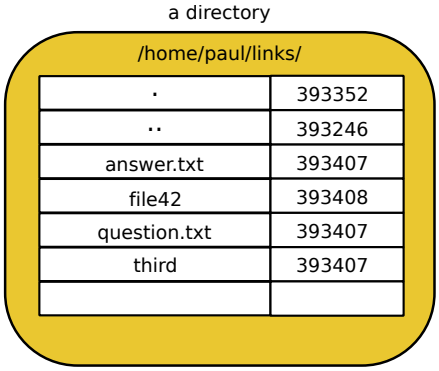
36.5 . and ..

If you look at the contents of a (newly created) directory, then you will notice two files named **.** and **..** are always there. We know that **.** points to the current directory, and **..** points to the parent directory. This works via hard links, as we can see when executing **ls -lai**.

```
paul@debian10:~/links$ ls -lai
total 20
393352 drwxr-xr-x  2 paul paul 4096 Aug 18 15:20 .
393246 drwxr-xr-x 14 paul paul 4096 Aug 17 21:32 ..
393407 -rw-r-----  4 paul paul  102 Aug 18 15:23 answer.txt
```

```
393408 -rw-r--r--  1 paul paul    0 Aug 17 23:10 file42
393407 -rw-r-----  4 paul paul  102 Aug 18 15:23 question.txt
393407 -rw-r-----  4 paul paul  102 Aug 18 15:23 third
paul@debian10:~/links$
```

So the **inode** of this directory is **393352** and the **inode** of the parent directory named **/home/paul** is **393246**. All files and directories form a giant tree of **inodes** and file names.



36.6 symbolic link

There is a limitation to **hard links** in that they have to exist on the same filesystem. To create links between different filesystems we need the **ln -s** command to create a **symbolic link**. A **symbolic link** is sometimes called a **symlink** or even a **soft link**.

In the screenshot below we create a **symbolic link** to the **file42** file.

```
paul@debian10:~/links$ ln -s file42 symlink.txt
paul@debian10:~/links$ ls -l
total 12
-rw-r----- 4 paul paul 102 Aug 18 15:23 answer.txt
-rw-r--r-- 1 paul paul   0 Aug 17 23:10 file42
-rw-r----- 4 paul paul 102 Aug 18 15:23 question.txt
lrwxrwxrwx 1 paul paul   6 Aug 18 15:38 symlink.txt -> file42
-rw-r----- 4 paul paul 102 Aug 18 15:23 third
paul@debian10:~/links$
```

Now we can say that **symlink.txt** is pointing to **file42**, and not the other way around. Notice that **symbolic links** all have an **l** as the first character in **ls -l**, and that permissions don't apply, the permissions of the target file apply.

36.7 readlink -f

There is a simple tool called **readlink** that follows a **symbolic link** to its target. But **readlink** becomes useful with the **-f** switch to keep on following **symbolic links** to **symbolic links** until it reaches the end file.

```
paul@debian10:~/links$ readlink /usr/bin/rename
/etc/alternatives/rename
paul@debian10:~/links$ readlink -f /usr/bin/rename
/usr/bin/file-rename
paul@debian10:~/links$
```

36.8 Cheat sheet

Table 36.1: Links

command	**explanation**
ls -il	Show **i**node number in long listing output.
ln foo bar	Create a hard link named **bar** (linked with **foo**).
ln -s foo bar	Create a symbolic link from **bar** to **foo**.
find / -inum 42	Find all files with **inode number** 42.
.	This is a hard link to the current directory.
..	This is a hard link to the parent directory.
readlink	Follows a symbolic link.
readlink -f	Recursively follows all symbolic links.

36.9 Practice

1. Create a new directory, then create two empty files in it.
2. Look at the inode numbers of the two files.
3. Create a hard link to the second file.
4. Change the permissions on one of the hard links, verify that both change.
5. How many hard links are there to the parent directory? Explain.
6. Find all hard links to the second file.
7. Create a symbolic link to the second file.

36.10 Solution

1. Create a new directory, then create two empty files in it.

```
mkdir ~/links
cd ~/links
> one
> two
```

2. Look at the inode numbers of the two files.

```
ls -li
```

3. Create a hard link to the second file.

```
ln two three
```

4. Change the permissions on one of the hard links, verify that both change.

```
chmod 600 two
ls -l
```

5. How many hard links are there to the parent directory? Explain.

```
ls -lid ..
Each subdirectory of /home/paul has a .. entry.
```

6. Find all hard links to the second file.

```
find ~ -inum 393407
```

7. Create a symbolic link to the second file.

```
ln -s symlinkname two
```

Chapter 37

Introduction to processes

37.1 process

A **process** is a piece of machine code that is running on one or more CPU's. A program becomes a **process** when it is executed. There are several processes running on your Debian Linux system. In this chapter we will take a look at the terminology around processes and how to identify them.

37.2 /proc

All processes are listed as a directory in **/proc**. Each number represents a process, there are 92 processes running in this screenshot.

```
paul@debian10:~$ ls /proc
1    132  205  3    389  453  80           fb           loadavg       stat
10   14   206  30   39   454  9            filesystems  locks         swaps
11   15   21   31   391  46   acpi         fs           meminfo       sys
116  16   22   32   392  462  buddyinfo    interrupts   misc          sysrq-trigger
119  17   23   33   4    463  bus          iomem        modules       sysvipc
12   171  239  34   40   467  cgroups      ioports      mounts        thread-self
122  173  24   341  41   5    cmdline      irq          mtrr          timer_list
123  175  25   342  418  6    consoles     kallsyms     net           tty
124  177  257  35   42   61   cpuinfo      kcore        pagetypeinfo  uptime
125  18   26   36   43   65   crypto       keys         partitions    version
126  19   269  361  432  66   devices      key-users    sched_debug   vmallocinfo
127  2    27   37   439  67   diskstats    kmsg         schedstat     vmstat
129  20   28   38   44   7    dma          kpagecgroup  self          zoneinfo
13   200  283  386  45   76   driver       kpagecount   slabinfo
131  203  29   388  450  8    execdomains  kpageflags   softirqs
paul@debian10:~$
```

37.3 PID and pidof

Every process on Debian Linux gets a unique **process identifier** or **PID**. This **PID** stays the same during the lifetime of the process. The **PID** is visible as a directory in **/proc**. The **PID** of a program can be found with the **pidof** command. For example the **PID** of the current bash shell is displayed when typing **pidof bash** or when executing **echo $$**.

```
paul@debian10:~$ pidof bash
463
paul@debian10:~$ echo $$
463
paul@debian10:~$
```

Tip
If you get more than one number after **pidof bash** then there are more **bash shells** running on the computer.

37.4 PPID

Every process also has a **parent process identifier** or **PPID**. This means that each process has a parent process, and that all processes are organised in a tree. The **PPID** of the current bash shell is displayed when echoing **$PPID**.

```
paul@debian10:~$ echo $PPID
462
paul@debian10:~$
```

37.5 init

Every process has a parent process, except **init** because **init** has **PID 1**. The **init** process is the very first process that is started by the kernel, at boot time. It is **init**'s job then to start other processes.

```
paul@debian10:~$ pidof init
1
paul@debian10:~$
```

37.6 ps

The **ps** command, short for **process snapshot**, can display information about some or all processes running on the system. Here we ask for all processes with **ps -ef**, but we **grep** for our bash shell's **PID**.

```
paul@debian10:~$ ps -ef | head -1
UID        PID  PPID  C STIME TTY          TIME CMD
paul@debian10:~$ ps -ef | grep $$
paul       463   462  0 11:36 pts/0    00:00:00 -bash
paul       469   463  0 11:38 pts/0    00:00:00 ps -ef
paul       470   463  0 11:38 pts/0    00:00:00 grep 463
paul@debian10:~$
```

We see the bash shell running with **PID** 463, and having a **PPID** of 462. We also see the **ps -ef** running with **PID** 469, having the bash shell as parent. And at the same time the **grep $$** is running as **grep 463** (remember shell expansion).

37.7 daemon

Some processes are started when the system is starting, and continue to run forever, or until the system shuts down. These processes are called **daemon** processes or **daemons**. For example when using **ssh** to connect to a computer, you make contact with **sshd**, which is the **secure shell daemon**.

```
paul@debian10:~$ ps -ef | grep ssh
root       432     1  0 11:36?        00:00:00 /usr/sbin/sshd -D
root       450   432  0 11:36?        00:00:00 sshd: paul [priv]
paul       462   450  0 11:36?        00:00:00 sshd: paul@pts/0
paul       472   463  0 11:39 pts/0    00:00:00 grep ssh
paul@debian10:~$
```

In the screenshot above we see **sshd** running with PID 432, having PID 1 as parent. This is the **ssh daemon**. Further we see another **sshd** process running with PID 462 having a connection with **paul@pts/0**. This last one will disappear when logging out of the computer.

37.8 kill

They say a process **dies** when it is no longer running. And to communicate with a process, we use the **kill** command. The next chapter is about the **kill** command.

37.9 zombie

When a process dies, but something still remains in **/proc**, then we call this a **zombie process**. **Zombies** take no resources and take no CPU time, so you generally do not have to worry about them. Unless there are many **zombies**. Also, you cannot **kill** a **zombie** because it is already dead.

37.10 fork - exec

A new process is created by a **fork** (copy) of an existing process, usually followed by an **exec** of a program in that new process. When you use the **exec** command, then you replace a process with a new program.

If you use **exec** in the bash shell, for example with **cp** then the shell will be replaced with **cp**. The **bash shell** effectively ceases to exist when executing **exec cp**.

```
paul@debian10:~$ exec cp dates.txt dates2.txt
Connection to 192.168.56.101 closed.
```

Note
The copy of dates.txt to dates2.txt happens, then the **cp** command ends and there is no **bash** to return to.

37.11 ps fax

All processes together form a tree with **init** at the top. You can see the whole tree using **ps fax**. In the screenshot below we see part of the tree, for example **ps fax** and **tail -8** are child processes of **bash**.

```
paul@debian10:~$ ps fax | tail -8
  432?        Ss     0:00 /usr/sbin/sshd -D
  543?        Ss     0:00  \_ sshd: paul [priv]
  558?        S      0:00      \_ sshd: paul@pts/0
  559 pts/0    Ss     0:00          \_ -bash
  613 pts/0    R+     0:00              \_ ps fax
  614 pts/0    S+     0:00              \_ tail -8
  547?        Ss     0:00 /lib/systemd/systemd --user
  549?        S      0:00  \_ (sd-pam)
paul@debian10:~$
```

Note
The **PID** of bash changed because we had to login again after the **exec cp** command.

37.12 top

The **top** command is a useful way to look at processes because **top** will by default order the top processes that use most of the CPU. It also tells you how many processes are active as **Tasks:** and how many **zombies** there are. (Together with memory information which we see in a later chapter.)

The **top** command can be stopped using the **q** key or by typing **Ctrl-c**.

```
paul@debian10:~$ top
top - 13:08:38 up  1:32,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  86 total,   1 running,  85 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  1.6 sy,  0.0 ni, 98.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    987.0 total,    834.9 free,     73.9 used,     78.2 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    802.9 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0   21860   9776   7724 S   0.0   1.0   0:01.10 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
```

```
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0   0.0   0:00.00 ksoftirqd/0
   10 root      20   0       0      0      0 I   0.0   0.0   0:02.38 rcu_sched
   11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_bh
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/0
   13 root      20   0       0      0      0 I   0.0   0.0   0:04.34 kworker/0:1-events
   14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   16 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/1
paul@debian10:~$
```

Tip

Looking at **PID 1** which is it now, **init** or **systemd**? The answer lies in the /**sbin**/**init** command. Debian 10 uses **systemd**, which has its own chapter.

```
paul@debian10:~$ file /sbin/init
/sbin/init: symbolic link to /lib/systemd/systemd
paul@debian10:~$
```

37.13 ps -eo

With **ps -eo** you can select which fields you want to see in the output of the **ps** command. For example in this screenshot we ask for the **PID**, the **CMD** (the command, with arguments, that was used to start this process) and the **COMMAND** (the binary that is actually running). All fields are explained in **man ps**, under STANDARD FORMAT SPECIFIERS.

```
paul@debian10:~$ ps -eo pid,cmd,comm | head -2
  PID CMD                         COMMAND
    1 /sbin/init                  systemd
paul@debian10:~$
```

37.14 Cheat sheet

Table 37.1: Processes terminology

term	**explanation**
PID	Process Identification or Process ID.
PPID	Parent PID.
daemon	A *guardian angel* process that never dies.
die	What a process does when it no longer exists.
zombie	A process that was killed, but something still remains in /proc.
/proc	A directory where all processes have a directory.
fork	All processes are created as a fork (copy) of an existing process (except init).
/sbin/init	The process with PID 1.

Table 37.2: Processes commands

command	**explanation**
pidof foo	Gives the PID of **foo**.
echo $$	Gives the PID of the current bash shell.
echo $PPID	Gives the PPID of the current bash shell.
ps -ef	Display a list of all processes.
ps fax	Display a list of all processes.
ps -eo ...	A custom display of all processes.
kill	A tool to communicate with processes or the kernel.
top	A tool that displays system info and a list of processes.

37.15 Practice

1. Issue a **cat /proc/1/cmdline** to see the command that started the first process on this computer.
2. What type of file is this command from question 1.?
3. Display the **PID** of your bash shell.
4. Display the **PID** of all bash shells.
5. Look at the complete process tree, page by page.
6. Are there any zombies on your computer?
7. Press **h** when in top to get some help on shortcuts in top. Can you sort the list by memory usage?

37.16 Solution

1. Issue a **cat /proc/1/cmdline** to see the command that started the first process on this computer.

```
cat /proc/1/cmdline
```

2. What type of file is this command from question 1.?

```
ls -l /sbin/init
```

or

```
file /sbin/init
```

or

```
ls -l $(cat /proc/1/cmdline)
file $(cat /proc/1/cmdline)
```

3. Display the **PID** of your bash shell.

```
echo $$
```

4. Display the **PID** of all bash shells.

```
pidof bash
```

5. Look at the complete process tree, page by page.

```
ps fax | more
ps -ef | more
```

6. Are there any zombies on your computer?

```
top
```

7. Press **h** when in top to get some help on shortcuts in top. Can you sort the list by memory usage?

```
>
```

Chapter 38

Signalling processes

38.1 kill -l

The **kill** command can be used to send a signal to a process. The list of signals can be obtained by typing **kill -l** (that is the letter l from **l**ist), as shown in this screenshot.

```
paul@debian10:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
paul@debian10:~$
```

38.2 kill -1

The first signal in the list, **SIGHUP**, is short for **signal a hangup** and is used to tell daemons they should reload their configuration file. So a **kill -1 1** will signal the systemd daemon that is running with PID 1 to reload its configuration.

```
root@debian10:~# kill -1 1
root@debian10:~#
```

The same is true for many other daemons, like in this case **sshd**.

```
root@debian10:~# ps fax | grep /usr/sbin/sshd
  432?        Ss     0:00 /usr/sbin/sshd -D
 7950 pts/0    S+     0:00                    \_ grep /usr/sbin/sshd
root@debian10:~# kill -1 432
root@debian10:~#
```

38.3 kill -2

The **kill -2** command, or SIGINT, is the same as a keyboard interrupt when typing **Ctrl-c**. The default action for the receiving process is to terminate. A developer can however choose what to do when receiving this signal.

```
paul@debian10:~$ find / >/dev/null 2>&1
^C
paul@debian10:~$ sleep 10
^C
paul@debian10:~$
```

38.4 kill -15

The **kill -15** command is equivalent to **kill** because SIGTERM is the default request sent to a process. It is a polite request to stop executing. Again, as a developer you can choose what to do when receiving a SIGTERM.

We are thrown out of the computer when we kill our sshd connection. Not even the **newline** after **kill 8820** gets displayed.

```
paul@debian10:~$ ps fax | grep ssh
 8808?        Ss     0:00 sshd: paul [priv]
 8820?        S      0:00  \_ sshd: paul@pts/0
 8833 pts/0    S+     0:00          \_ grep ssh
paul@debian10:~$
paul@debian10:~$
paul@debian10:~$ kill 8820Connection to 192.168.56.101 closed by remote host.
Connection to 192.168.56.101 closed.
```

38.5 kill -9

The **kill -9** command is different than the others in that it sends a SIGKILL request to the **Linux kernel**. It tells the kernel to stop a process. As a developer you cannot decide on an action for a **kill -9** because the process never sees this request.

```
paul@debian10:~$ kill -9 $$
Connection to 192.168.56.101 closed.
```

38.6 kill -18 -19 -20

The **kill -19** or **kill -20** command will stop a process and put it suspended in background. The process is not terminated it is stopped (as if frozen in time). The process can continue to execute when it receives a **kill -18** or SIGCONT. Typing **Ctrl-z** is equivalent to **kill -20**

In this screenshot we start a **sleep 42** process and use **Ctrl-z** to stop it in background.

```
paul@debian10:~$ sleep 8472
^Z
[1]+  Stopped                 sleep 8472
paul@debian10:~$
```

Then we look for the PID and send a SIGCONT to the sleep command. It now continues in background and will notify you when finished. Background processes are discussed in the next chapter.

```
paul@debian10:~$ ps fax | grep sleep
 8931 pts/0    T      0:00              \_ sleep 8472
 8933 pts/0    S+     0:00              \_ grep sleep
paul@debian10:~$ kill -18 8931
paul@debian10:~$ ps fax | grep sleep
 8931 pts/0    S      0:00              \_ sleep 8472
 8935 pts/0    S+     0:00              \_ grep sleep
paul@debian10:~$
```

Note the little status change from T to S in the screenshot above.

38.7 pkill

Killing by numbers may seem cumbersome. Luckily there is **pkill** to kill by name. But be careful, the command in the next screenshot will kill **all** sleep processes (that belong to the user **paul**). As **root** this can also terminate scripts that momentarily execute a **sleep 1**.

```
paul@debian10:~$ pkill sleep
[1]+  Terminated              sleep 8472
paul@debian10:~$
```

By the way, typing **pkill slee** is enough to kill the sleep process, but the command just became more dangerous.

You can add the **-c** option to get a count and a list of kills. As this screenshot shows.

```
paul@debian10:~$ pkill -c slee
1
[1]+  Terminated              sleep 8472
paul@debian10:~$
```

And you can of course change the default **-15** signal to something else. The example below sends a **-9** request to the kernel.

```
paul@debian10:~$ pkill -c -9 slee
1
[1]+  Killed                  sleep 8472
paul@debian10:~$
```

38.8 killall

With **killall** you can kill a process by name, this is identical to **pkill** when using no options.

```
paul@debian10:~$ sleep 8472 &
[1] 7999
paul@debian10:~$ killall sleep
[1]+  Terminated              sleep 8472
paul@debian10:~$
```

But **killall** has an interesting **-i** option to **interactively** select which processes you want to kill with this tool.

```
paul@debian10:~$ sleep 8472 &
[1] 8001
paul@debian10:~$ killall -i sleep
Kill sleep(8001)? (y/N) y
[1]+  Terminated              sleep 8472
paul@debian10:~$
```

38.9 top k

Within the **top** command there is also an option to signal a process. Just type **k** when in **top** followed by the PID to signal.

```
paul@debian10:~$ top
top - 19:34:55 up  7:58,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:  89 total,   1 running,  88 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :    987.0 total,    755.3 free,     86.4 used,    145.2 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    756.9 avail Mem
PID to signal/kill [default pid = 269]
```

38.10 Cheat sheet

Table 38.1: Killing processes

command	explanation
kill -l (letter L)	List all signals.
kill -1 (digit One)	Send signal 1 SIGHUP to a process, often the process re-reads a configuration.
kill -2	Send a signal 2 SIGINT to a process, interrupts a process.
Ctrl-c	Send a signal 2 SIGINT to a process.
kill 15	Send a signal 15 SIGTERM to a process, this kills a process nicely.
kill -9	Send a signal 9 SIGKILL to the Linux kernel, this sure kills a process.
kill -19	Send a signal 19 to the kernel to suspend a process.
kill -20	Send a signal 20 to the kernel to suspend a process.
Ctrl-z	Send a signal 20 to the kernel to suspend a process.
kill -18	Send a signal 18 to the kernel to continue a suspended process.
pkill foo	Kill (SIGTERM) all processes with **foo** in their name.
pkill -c foo	Also displays a count of all kills.
pkill -9 foo	Sure kill of all processes with **foo** in their name.

38.11 Practice

1. List all the signals that you can send.
2. Tell the sshd daemon to reload its configuration file.
3. Tell the kernel to kill your bash shell.
4. Put a **sleep 8472** in the background, stopped.
5. Start the **sleep 8472** again in background.
6. Kill all sleep processes by name, while displaying a count.

38.12 Solution

1. List all the signals that you can send.

```
kill -l
```

2. Tell the sshd daemon to reload its configuration file.

```
su -
ps fax | grep sshd
kill -1 xyz (where xyz is the pid of the sshd daemon)
```

3. Tell the kernel to kill your bash shell.

```
kill -9 $$
```

4. Put a **sleep 8472** in the background, stopped.

```
sleep 8472
Ctrl-z
```

5. Start the **sleep 8472** again in background.

```
kill -18 xyz (where xyz is the pid of the sleep process)
```

6. Kill all sleep processes by name, while displaying a count.

```
pkill -c sleep
```

Chapter 39

Processing jobs

39.1 sleep &

Any process that is managed by your current shell is called a job. Jobs can be put in **background** by typing an **ampersand** at the end of the command line. In this screenshot we put a **sleep 8472** command as a job in the background.

```
paul@debian10:~$ sleep 8472 &
[1] 4601
paul@debian10:~$
```

39.2 jobs

You can see all your **jobs** with the **jobs** command.

```
paul@debian10:~$ jobs
[1]+  Running                 sleep 8472 &
paul@debian10:~$
```

39.3 Ctrl-z

Another way to put a job in the background is by typing **Ctrl-z** when a foreground job is running. This will freeze the job in background. The screenshot starts a process and then shows what happens when pressing **Ctrl-z**.

```
paul@debian10:~$ sleep 31337
^Z
[2]+  Stopped                 sleep 31337
paul@debian10:~$
```

39.4 bg

With the **bg** command you can (re)start a job that is suspended in background by **Ctrl-z**. Typing **bg %1** refers to job number one, and **bg %2** refers to job number two, and so on.

```
paul@debian10:~$ jobs
[1]-  Running                 sleep 8472 &
[2]+  Stopped                 sleep 31337
paul@debian10:~$ bg %2
[2]+ sleep 31337 &
paul@debian10:~$ jobs
[1]-  Running                 sleep 8472 &
[2]+  Running                 sleep 31337 &
paul@debian10:~$
```

Another way to (re)start a job in background is using the **%1 &** notation. The result is the same as with **bg %1**.

```
paul@debian10:~$ jobs
[1]+  Stopped                 sleep 8472
paul@debian10:~$ %1 &
[1]+ sleep 8472 &
paul@debian10:~$ jobs
[1]+  Running                 sleep 8472 &
paul@debian10:~$
```

39.5 jobs -p

You can view the PIDs of the jobs in background with **jobs -p**, but one may prefer to run **ps -o pid,cmd** to better distinguish the jobs. Both commands are shown in this screenshot.

```
paul@debian10:~$ jobs -p
4601
4602
paul@debian10:~$ ps -o pid,cmd
  PID CMD
  464 -bash
 4601 sleep 8472
 4602 sleep 31337
 4618 ps -o pid,cmd
paul@debian10:~$
```

39.6 fg

Jobs that are running in background can be called to run in foreground with the **fg** command. So **fg %1** will put job number one in foreground.

```
paul@debian10:~$ jobs
[1]-  Running                 sleep 8472 &
[2]+  Running                 sleep 31337 &
paul@debian10:~$ fg %2
sleep 31337
```

And here also you can just use **%1** to call a job to the foreground.

```
paul@debian10:~$ jobs
[1]-  Running                 sleep 8472 &
[2]+  Running                 sleep 31337 &
paul@debian10:~$ %1
sleep 8472
```

39.7 kill

Jobs can be killed by a regular **kill** command followed by the PID. The shell will tell you the job is **terminated** when you next type enter.

```
paul@debian10:~$ ps -o pid,cmd
  PID CMD
  464 -bash
 4601 sleep 8472
 4622 ps -o pid,cmd
paul@debian10:~$ kill 4601
paul@debian10:~$
[1]+  Terminated              sleep 8472
paul@debian10:~$
```

The **kill** command will also accept the **job id** as an argument. In this screenshot we kill job number one with **kill %1**.

```
paul@debian10:~$ jobs
[1]-  Running                 sleep 8472 &
[2]+  Running                 sleep 31337 &
paul@debian10:~$ kill %1
```

```
paul@debian10:~$
[1]-  Terminated              sleep 8472
paul@debian10:~$
```

39.8 nohup

Jobs that belong to your shell will stop running when you quit the shell. You can prevent this by using **nohup** to start the job. The sleep command in the following screenshot will get **init** as a foster parent when the bash shell exits.

```
paul@debian10:~$ nohup sleep 9000 &
[1] 4712
paul@debian10:~$ nohup: ignoring input and appending output to nohup.out

paul@debian10:~$ jobs
[1]+  Running                 nohup sleep 9000 &
paul@debian10:~$
```

39.9 disown

You can use **disown** to make sure that a running job is not killed when exiting the shell. When you **disown** a job, then you can no longer manage it with **fg** and **bg**, and it will no longer appear in the **jobs** list.

```
paul@debian10:~$ sleep 9000 &
[1] 4723
paul@debian10:~$ jobs
[1]+  Running                 sleep 9000 &
paul@debian10:~$ disown %1
paul@debian10:~$ jobs
paul@debian10:~$
```

39.10 huponexit

There is a shell option called **huponexit** that will enable the survival of all your jobs when properly exiting the shell (by typing **exit** or **Ctrl-d**). This option is off by default, the screenshot below turns it on.

```
paul@debian10:~$ shopt huponexit
huponexit       off
paul@debian10:~$ shopt -s huponexit
paul@debian10:~$ shopt huponexit
huponexit       on
paul@debian10:~$
```

39.11 Cheat sheet

Table 39.1: Jobs

command	**explanation**
foo &	Run the **foo** command in background.
jobs	List all jobs in background.
jobs -p	List all jobs in background using their PID.
ps -o pid,cmd	List all processes linked to this shell (includes background jobs).
Ctrl-z	Put the foreground process **suspended** in background.
bg %42	Start job number 42 in background.
fg %42	Start (or put) job number 42 in foreground.
kill %42	Kill job number 42.
disown %42	Disown job number 42 from this shell (it will not die with the shell).
nohup foo &	Start the **foo** command in background such that it will not die with the current shell.

39.12 Practice

1. Execute **find / >/dev/null 2>&1** and immediately put it suspended in background.
2. Execute **sleep 8472** directly in background.
3. List you background jobs.
4. List the PID of each background job.
5. Put the **sleep** job in foreground.
6. Send a keyboard interrupt to the **sleep** process.
7. Start the **find** job in background.
8. Terminate the find job (if it has not stopped already).
9. Start a **sleep 9000** process that will not be killed when the shell exits.
10. Start a **sleep 9000** process in background. Then decide to not have it killed when you exit the shell.
11. Verify whether the **huponexit** shell option is active.

39.13 Solution

1. Execute **find / >/dev/null 2>&1** and immediately put it suspended in background.

```
find / >/dev/null 2>&1  # followed by Ctrl-z
```

2. Execute **sleep 8472** directly in background.

```
sleep 8472 &
```

3. List you background jobs.

```
jobs
ps
```

4. List the PID of each background job.

```
jobs -p
ps
```

5. Put the **sleep** job in foreground.

```
fg %2
```

6. Send a keyboard interrupt to the **sleep** process.

```
Ctrl-c
```

7. Start the **find** job in background.

```
bg
```

8. Terminate the find job (if it has not stopped already).

```
kill %1
```

9. Start a **sleep 9000** process that will not be killed when the shell exits.

```
nohup sleep 9000 &
```

10. Start a **sleep 9000** process in background. Then decide to not have it killed when you exit the shell.

```
sleep 9000 &
disown %1
```

11. Verify whether the **huponexit** shell option is active.

```
shopt huponexit
```

Chapter 40

Process priorities

40.1 priorities

All processes that are running on a Debian Linux computer have a **priority** value and a **nice** value. The higher the priority, the more time a process gets on the CPU. The highest priority is **0**. See the **PR** column in this screenshot from **top**.

```
paul@debian10:~$ top
top - 15:04:27 up  3:27,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:  92 total,   1 running,  91 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.1 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :    987.0 total,    797.5 free,     80.2 used,    109.2 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    781.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4848 paul      20   0   10956   3368   2960 R   0.3   0.3   0:00.02 top
    1 root      20   0   21864   9820   7756 S   0.0   1.0   0:01.71 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0   0.0   0:00.29 ksoftirqd/0
   10 root      20   0       0      0      0 I   0.0   0.0   0:04.71 rcu_sched
   11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_bh
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.07 migration/0
   13 root      20   0       0      0      0 I   0.0   0.0   0:00.32 kworker/0:1-events
   14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   16 root      rt   0       0      0      0 S   0.0   0.0   0:00.15 migration/1
paul@debian10:~$
```

Under **NI** in the screenshot above a **nice** value is listed. The **nice** value is added to the priority value to decrease (or increase with negative nice values) the priority. In this chapter we will start with **nice** values and how to set them.

Next we will look at setting limits to the allowed CPU time and the number of processes with **ulimit**, **/etc/security/limits.conf** and **prlimit**.

To finish this chapter we will introduce the use of **cgroups** to limit the CPU time for a process.

40.2 top -p

The **top** command will by default list all processes, organised by their current CPU usage. But you can start **top** with a list of PIDs to display. This can be useful to avoid clutter or just to display the processes that you choose to watch.

```
paul@debian10:~$ sleep 8472 &
[1] 4801
paul@debian10:~$ sleep 31337 &
[2] 4802
paul@debian10:~$ top -p 4801,4802
top - 14:48:55 up  3:11,  2 users,  load average: 0.00, 0.01, 0.00
Tasks:   2 total,   0 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.5 id,  0.2 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :    987.0 total,    798.4 free,     79.7 used,    108.9 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    781.8 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4801 paul      20   0    5260    744    680 S   0.0   0.1   0:00.00 sleep
 4802 paul      20   0    5260    684    620 S   0.0   0.1   0:00.00 sleep
```

40.3 mkfifo

Each process that starts gets a **virtual memory** space that allows it to use virtual addresses which are translated to real addresses by the kernel. Processes cannot read memory from other processes.

We can use **mkfifo** to create **named pipes** that can facilitate communication between processes. We will use these pipes together with **cat** to put some strain on our CPU's.

```
paul@debian10:~/pipes$ mkfifo pipe1
paul@debian10:~/pipes$ mkfifo pipe2
paul@debian10:~/pipes$ ls -l
total 0
prw-r--r-- 1 paul paul 0 Aug 21 14:53 pipe1
prw-r--r-- 1 paul paul 0 Aug 21 14:53 pipe2
paul@debian10:~/pipes$
```

40.4 loading the CPU with cat

To simulate a load on the CPU we use the **cat** command to **cat** one character between two pipes. We will rename the cat command several times to be able to distinguish between the processes (because looking at the several close-to-each-other PIDs is cumbersome).

Warning
This may take your CPU load to 100%, I took these screenshots on a throttled VPS.

In the screenshot below we copy **cat**, we create two pipes and we use this copied **cat** to swap **x** between the two pipes.

40.4.1 creating the cat commands

```
paul@debian10:~/pipes$ cp /bin/cat pro33
paul@debian10:~/pipes$ mkfifo pipe33a pipe33b
paul@debian10:~/pipes$ echo -n x | ./pro33 - pipe33a > pipe33b &
[1] 454
paul@debian10:~/pipes$ ./pro33 <pipe33b >pipe33a &
[2] 455
paul@debian10:~/pipes$
```

We do the same for **project 42**, this will put the load on a two core CPU to 100%.

```
paul@debian10:~/pipes$ mkfifo pipe42a
paul@debian10:~/pipes$ mkfifo pipe42b
paul@debian10:~/pipes$ cp /bin/cat pro42
paul@debian10:~/pipes$ echo -n x | ./pro42 - pipe42a > pipe42b &
[3] 457
paul@debian10:~/pipes$ ./pro42 <pipe42b >pipe42a &
[4] 458
paul@debian10:~/pipes$
```

And a third time for **project 9000**, since my VPS has only two cores, the six **cat** processes will have to fight for CPU time.

```
paul@debian10:~/pipes$ mkfifo pipe9000a pipe9000b
paul@debian10:~/pipes$ cp /bin/cat pro9000
paul@debian10:~/pipes$ echo -n x | ./pro9000 - pipe9000a > pipe9000b &
```

```
[5] 461
paul@debian10:~/pipes$ ./pro9000 <pipe9000b >pipe9000a &
[6] 462
paul@debian10:~/pipes$
```

Note
If you run this on a computer with more cores or more CPU's you may have to repeat the procedure above several times.

40.4.2 Viewing the CPU battle

The **top -p** command will show our six processes battling for CPU time. It is possible that a process that is running longer, gets some more CPU time. You should see something similar to this screenshot with %CPU jumping around 33% for all six processes.

```
paul@debian10:~/pipes$ top -p 454,455,457,458,461,462
top - 15:45:41 up 4 min,  1 user,  load average: 3.02, 0.95, 0.34
Tasks:   6 total,   3 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.9 us, 67.4 sy,  0.0 ni, 23.1 id,  0.0 wa,  0.0 hi,  0.6 si,  0.0 st
MiB Mem :    987.3 total,    856.2 free,     63.5 used,     67.6 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    819.5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  457 paul      20   0    5400   1632   1552 S  36.7   0.2   0:41.51 pro42
  455 paul      20   0    5400    684    620 S  35.0   0.1   0:42.00 pro33
  458 paul      20   0    5400    748    684 R  29.0   0.1   0:39.95 pro42
  454 paul      20   0    5400   1696   1608 R  27.0   0.2   0:38.81 pro33
  461 paul      20   0    5400   1664   1580 R  22.0   0.2   0:13.27 pro9000
  462 paul      20   0    5400    744    680 S  18.3   0.1   0:13.84 pro9000
```

40.5 renice

All processes got a priority of **20**. They are all equal. Now let's say that **project 9000** is less important, then we can use the **renice** command to lower its priority (higher number is less priority on the CPU).

```
paul@debian10:~/pipes$ renice 15 461
461 (process ID) old priority 0, new priority 15
paul@debian10:~/pipes$ renice 15 462
462 (process ID) old priority 0, new priority 15
paul@debian10:~/pipes$
```

After a little while, you see something like the screenshot below. The **pro9000** cat command gets a lot less CPU time. Its priority is changed to **35** because of our **renice** command. This is a simple way to manage process priorities for running processes.

```
paul@debian10:~/pipes$ top -p 454,455,457,458,461,462
top - 15:46:38 up 5 min,  1 user,  load average: 4.19, 1.65, 0.61
Tasks:   6 total,   4 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s): 16.5 us, 82.6 sy,  0.3 ni,  0.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    987.3 total,    855.6 free,     64.0 used,     67.7 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    819.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  458 paul      20   0    5400    748    684 R  50.5   0.1   0:59.89 pro42
  454 paul      20   0    5400   1696   1608 S  50.2   0.2   1:00.35 pro33
  455 paul      20   0    5400    684    620 R  46.2   0.1   1:03.35 pro33
  457 paul      20   0    5400   1632   1552 R  41.2   0.2   1:02.43 pro42
  461 paul      35  15    5400   1664   1580 S   6.0   0.2   0:26.65 pro9000
  462 paul      35  15    5400    744    680 R   4.7   0.1   0:26.59 pro9000
```

Note
Always lower the priority of less important processes, never increase priority with negative nice values. When there is no competition, then even a process with nice value of 20 will use 100% CPU.

You can kill all the **cat** processes now.

40.6 nice

Compressed and possibly encrypted backups are probably very important, so we start them normally. Other CPU intensive processes, like say a SETI search can then be started with **nice** in front of the command. As mentioned before, even a **nice** process can take 100% CPU if there is no competing process.

By default the **nice** command will start a process with a **nice value** of 10. As a normal user the **nice values** range from 1 to 19. Only **root** can use negative nice values.

```
paul@debian10:~/pipes$ nice sleep 8472 &
[1] 509
paul@debian10:~/pipes$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   448   447  0  80   0 -  2053 -      pts/0    00:00:00 bash
0 S  1000   509   448  0  90  10 -  1315 hrtime pts/0    00:00:00 sleep
0 R  1000   516   448  0  80   0 -  2637 -      pts/0    00:00:00 ps
paul@debian10:~/pipes$
```

40.7 ulimit

The **ulimit** bash built-in command can set some limits on CPU usage. Consider this screenshot with possible limits for the total CPU time and for the number of concurrent processes.

```
paul@debian10:~$ ulimit -a | grep -A1 cpu
cpu time               (seconds, -t) unlimited
max user processes              (-u) 3833
paul@debian10:~$
```

These limits can be changed with the **ulimit** command. For example to limit the **cpu time** just issue a **ulimit -t 40** command to set it to 40 seconds.

```
paul@debian10:~$ ulimit -t 40
paul@debian10:~$ ulimit -a | grep -A1 cpu
cpu time               (seconds, -t) 40
max user processes              (-u) 3833
paul@debian10:~$
```

That is 40 seconds total CPU time, not 40 seconds real time. A **sleep 31337** will run fine because it uses less than 40 seconds on the CPU.

You can limit the number of processes that your shell can start in the same way, by using **ulimit -u 100** to limit to 100 concurrent processes.

```
paul@debian10:~$ ulimit -u 100
paul@debian10:~$ ulimit -a | grep -A1 cpu
cpu time               (seconds, -t) 40
max user processes              (-u) 100
paul@debian10:~$
```

Note
These limits are gone when you log out and log back in.

40.8 /etc/security/limits.conf

The **root** user can set **soft** and **hard** limits using the **/etc/security/limits.conf** file. These limits can not be increased then by the user. The **man limits.conf** has several examples, we use just two.

The screenshot shows two limits set for the user **tania**. She can have a maximum of 99 concurrent processes and a process can run for maximum one day (CPU time!).

```
root@debian10:~# grep tania /etc/security/limits.conf
tania           hard    cpu     1440
tania           hard    nproc   99
root@debian10:~# su - tania
tania@debian10:~$ ulimit -a | grep -A1 cpu
cpu time               (seconds, -t) 86400
max user processes              (-u) 99
tania@debian10:~$
```

40.9 prlimit

Another useful tool to limit CPU usage is **prlimit**. It has the same settings as **ulimit**, as can be seen in this screenshot.

```
paul@debian10:~$ prlimit | grep -e CPU -e NPROC
CPU        CPU time                          unlimited unlimited seconds
NPROC      max number of processes                3833      3833 processes
paul@debian10:~$ su - tania
Password:
tania@debian10:~$ prlimit  | grep -e CPU -e NPROC
CPU        CPU time                            86400     86400 seconds
NPROC      max number of processes                99        99 processes
tania@debian10:~$
```

But it can serve as starter for commands (or scripts). For example this screenshot shows how to start the **bzip2** command but with a limit of one hour (3600 seconds) CPU time.

```
paul@debian10:~$ prlimit --cpu=3600 bzip2 ReallyHugeFile &
[1] 1012
paul@debian10:~$
```

40.10 cgroups

The Linux kernel provides **control groups** or **cgroups** to limit resource usage for resources like CPU, memory, network and so on. The resources are listed in **/sys/fs/cgroup**.

```
paul@debian10:~$ ls /sys/fs/cgroup/
blkio  cpuacct      cpuset   freezer  net_cls           net_prio    pids  systemd
cpu    cpu,cpuacct  devices  memory   net_cls,net_prio  perf_event  rdma  unified
paul@debian10:~$
```

The presence of the **unified** directory signifies that Debian 10 uses **hybrid cgroups**, this may change in future versions of Debian Linux to full **cgroupsv2**.

In this section we will show you how to start a process with limited CPU access. First we need to create a **cgroup** that limits CPU usage to 10%.

```
root@debian10:~# cgcreate -g cpu:cpu10
root@debian10:~# ls /sys/fs/cgroup/cpu/cpu10/
cgroup.clone_children  cpuacct.usage_all          cpuacct.usage_sys   cpu.shares
cgroup.procs           cpuacct.usage_percpu       cpuacct.usage_user  cpu.stat
cpuacct.stat           cpuacct.usage_percpu_sys   cpu.cfs_period_us   notify_on_release
cpuacct.usage          cpuacct.usage_percpu_user  cpu.cfs_quota_us    tasks
root@debian10:~#
```

Then we limit our new cgroup to 100 shares (of the available 1024).

```
root@debian10:~# cat /sys/fs/cgroup/cpu/cpu10/cpu.shares
1024
root@debian10:~# echo 100 > /sys/fs/cgroup/cpu/cpu10/cpu.shares
root@debian10:~#
```

Then we start two CPU intensive tasks, the first one is free to use the CPU, and the second one is confined to our cgroup.

```
root@debian10:~# cat /dev/urandom > /dev/null &
[3] 600
root@debian10:~# cgexec -g cpu:cpu10 cat /dev/urandom > /dev/null &
[4] 601
root@debian10:~#
```

Using **top** we can see that the second **cat** with pid 601 is nicely confined to 10% CPU.

```
root@debian10:~# top -p 600,601
top - 13:06:33 up 25 min,  2 users,  load average: 1.91, 1.27, 1.11
Tasks:   2 total,   2 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,100.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    987.5 total,    847.7 free,     68.1 used,     71.8 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.    813.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  600 root      20   0    5400    740    680 R  91.0   0.1   2:47.12 cat
  601 root      20   0    5400    748    684 R   9.0   0.1   0:15.80 cat
```

The command **cgcreate** does not make a permanent change, but it is easy to script, as the screenshot below shows.

```
root@debian10:~# cat cgroupsdemo.sh
#!/bin/bash

# This process will take 100% CPU
cat /dev/urandom > /dev/null &

# Create a cgroup to limit the CPU to 10%
cgcreate -g cpu:cpu10

# Default 100%=1024 shares
echo 100 > /sys/fs/cgroup/cpu/cpu10/cpu.shares

# This process will take 10% CPU while the previous cat is running at 90%
cgexec -g cpu:cpu10 cat /dev/urandom > /dev/null &
root@debian10:~#
```

Important
The **cgroup** demo was done on a VPS with one CPU. You may need more processes before one gets limited to 10%.

40.11 Cheat sheet

Table 40.1: Priorities

command	**explanation**
top	Tool that displays processes with their priority and nice value.
top -p 33,42	Display only processes with PID 33 or PID 42.
mkfifo foo	Create a named pipe **foo**.
renice 42	Change the nice value of PID 42 to 10.
renice 15 42	Change the nice value of PID 42 to 15.
nice foo	Start the **foo** command with a nice value of 10.
ulimit	Tool to set limits on CPU usage.
prlimit	Tool to start programs with limits on CPU usage.
/etc/security/limits.conf	File which contains settings for CPU (and other) limits.
cgcreate	Tool to create **cgroups** that can set limits.
cgexec	Tool used to execute a command in a certain cgroup.

40.12 Practice

1. Use **top** and take a look at the PR and NI column. Some kernel processes are not nice.
2. Start two **sleep** processes in background and use **top** to view just those two processes.
3. Figure out how many CPU's or cores are on your computer.
4. Optional: Recreate **"Loading the cpu with cat"**, but use one more cat-duos than the number of processors you have in the previous question.
5. Use the **renice** command to lower the priority of a **cat duo** from the previous question. If there are not enough cat processes, then you will see no effect.
6. Start an extra **cat duo** with the nice command.
7. Type **ulimit -a** and consider the options. Would you start a script with some of these limits?
8. Which limit would stop a bash **fork bomb** (that you find on the internet).
9. Open the man page of **limits.conf** and consider the options.
10. Start a process with **prlimit** that gets a maximum of 10 minutes on the CPU.
11. Create a **cgroup** to limit CPU usage to 20% and start two processes in this **cgroup**.

40.13 Solution

1. Use **top** and take a look at the PR and NI column. Some kernel processes are not nice.

```
top
```

2. Start two **sleep** processes in background and use **top** to view just those two processes.

```
sleep 8472 &
sleep 31337 &
top -p 4801,4802 # replace with the PIDs you received
```

3. Figure out how many CPU's or cores are on your computer.

```
grep processor /proc/cpuinfo
```

4. Optional: Recreate **"Loading the cpu with cat"**, but use one more cat-duos than the number of processors you have in the previous question.

```
mkfifo pipe1 pipe2
cp /bin/cat pro1
echo -n x | ./pro1 - pipe1 > pipe2 &
./pro1 <pipe2 > pipe1 &


mkfifo pipe3 pipe4
cp /bin/cat pro3
echo -n x | ./pro3 - pipe3 > pipe4 &
./pro3 <pipe4 > pipe3 &

...
```

5. Use the **renice** command to lower the priority of a **cat duo** from the previous question. If there are not enough cat processes, then you will see no effect.

```
renice 461 # replace with your PIDs
renice 462
```

6. Start an extra **cat duo** with the nice command.

```
mkfifo pipe3 pipe4
cp /bin/cat pro3
echo -n x | nice ./pro3 - pipe3 > pipe4 &
nice ./pro3 <pipe4 > pipe3 &
```

7. Type **ulimit -a** and consider the options. Would you start a script with some of these limits?

```
ulimit -a
```

8. Which limit would stop a bash **fork bomb** (that you find on the internet).

```
max user processes
```

9. Open the man page of **limits.conf** and consider the options.

```
man limits.conf
```

10. Start a process with **prlimit** that gets a maximum of 10 minutes on the CPU.

```
prlimit --cpu=600 myscript.sh
```

11. Create a **cgroup** to limit CPU usage to 20% and start two processes in this **cgroup**.

```
# Create a cgroup to limit the CPU to 20%
cgcreate -g cpu:cpu20

# Default 100%=1024 shares
echo 200 > /sys/fs/cgroup/cpu/cpu20/cpu.shares

# This processes will take 20% CPU together, while the free cat is running at 80%
cgexec -g cpu:cpu20 cat /dev/urandom > /dev/null &
cgexec -g cpu:cpu20 cat /dev/urandom > /dev/null &
root@debian10:~#
```

Chapter 41

cpu monitoring

41.1 top

41.2 ps

41.3 glances

client/server glances -s glaces -c @server

glances -w (webserver) http://@server:61208/30 http://@server:61208/50

41.4 mpstat

mpstat -P ALL mpstat -P ALL 2 4

41.5 sar

sar 2 3 sar -u 2 3

41.6 iostat -c

iostat -xtc 5 2

41.7 vmstat

vmstat 1 5

41.8 htop

41.9 nmon

41.10 cpustat

41.11 perf

41.12 tiptop

41.13 dstat

41.14 Practice

41.15 Solution

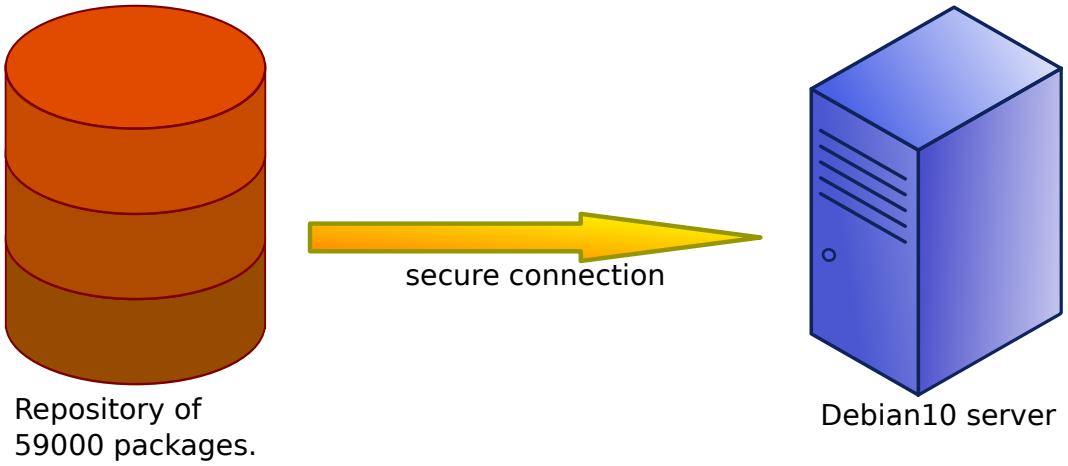
Chapter 42

Software management

42.1 Repositories

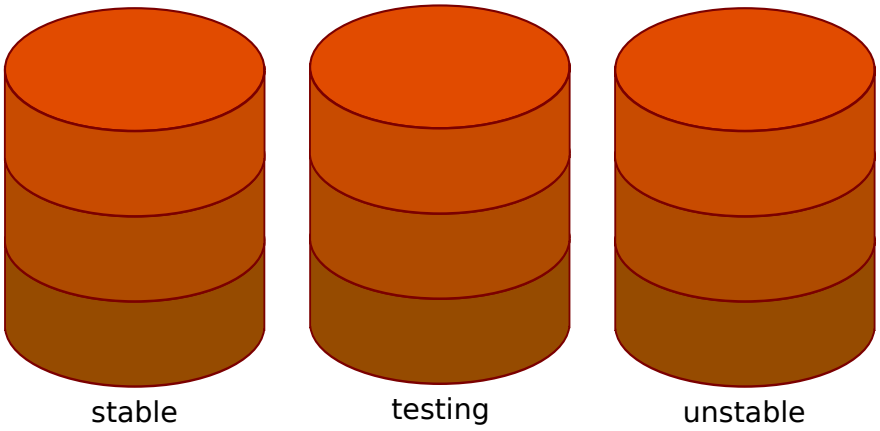
Most of the time Debian applications are installed from the repository of 59.000 (sept 2019) packages. This is done over a secure connection. If a hacker manages to redirect you to another repository then your Debian will refuse to install packages from it.

This repository is well maintained, every package, every application has one or more dedicated maintainers. Thousands of people take care of this repository. The packages in the repository are well tested.



42.1.1 Debian distributions

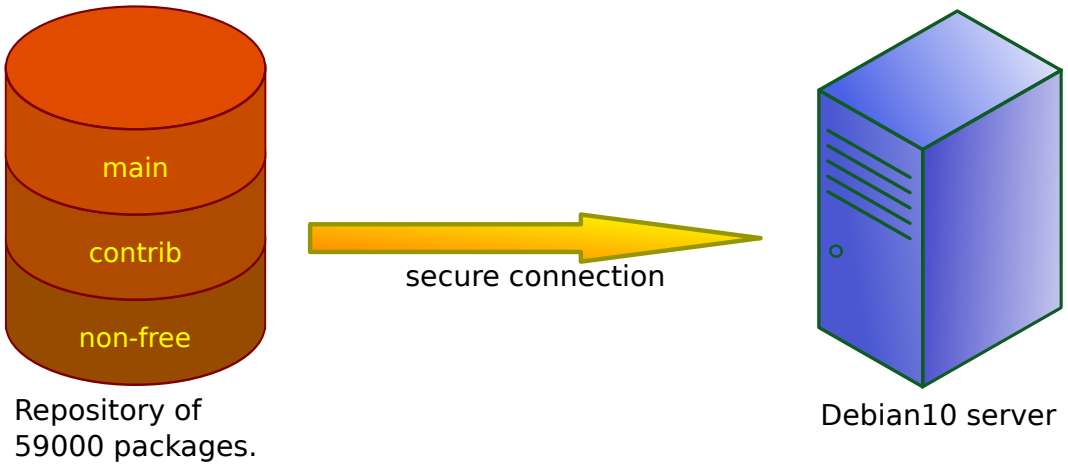
There are always three main Debian distributions to choose from; there is **stable** which only contains well tested packages, **testing** which contains packages that are intended for **stable**, and **unstable** which contains packages that are new but have met the criterion for stability and quality of packaging.



You can choose one of these three repositories. Debian servers usually run the **stable** distribution. **Debian 10** nicknamed **buster** is the stable distribution since July 2019, and will likely remain so until several months into 2021. Even the **stable** distribution gets regular updates so you are not stuck with old packages.

42.1.2 main, contrib and non-free

You can decide, inside one repository, whether to include only **main** packages, or also **contrib** packages or even **non-free** packages. The **main** distribution contains packages that fully adhere to the **debian free software guidelines**. The **contrib** distribution has free packages that depend on **non-free** software. And **non-free** are packages that have restrictive licensing.



By default only the **main** distribution is used, this contains just over 58000 packages. The **contrib** contains almost 300 and **non-free** about 600.

42.1.3 /etc/apt/sources.list

Software on Debian 10 is managed via **apt**, with **/etc/apt/sources.list** as its main configuration file. In this file you will find a line like this.

```
deb http://deb.debian.org/debian/ buster main
```

This line names a repository **http://deb.debian.org/debian/**, then names a Debian release named **buster**, this is the stable release. And in this repository we only connect to the **main** software packages.

Next you will find a line like this. This is the same as above, but for the **source code** of these packages. Debian provides source code for all its packages!

```
deb-src http://deb.debian.org/debian/ buster main
```

If you want to include packages from **contrib** then you can change these lines like in the screenshot below, simply by adding a space and the word **contrib**. Similarly appending **non-free** will also include the **non-free** packages.

```
deb http://deb.debian.org/debian/ buster main contrib
deb-src http://deb.debian.org/debian/ buster main contrib
```

Next in this **/etc/apt/sources.list** file you will find **security updates**. These come from an extra secure location. And there are the **volatile** updates, which are necessary updates to keep stable packages functional (for example with recent antivirus signatures).

42.1.4 apt-get update

The **apt-get update** command will download meta-information about all packages in the **repository**. It will not download any packages, nor will it update any applications, it is just information about all packages that is updated.

```
root@debian10:~# apt-get update
Hit:1 http://security.debian.org/debian-security buster/updates InRelease
Hit:2 http://deb.debian.org/debian buster InRelease
Hit:3 http://deb.debian.org/debian buster-updates InRelease
Reading package lists... Done
root@debian10:~#
```

42.1.5 apt-cache search

After an **apt-get update** your Debian server is up-to-date on information about packages. You can query this information with an **apt-cache search** command. In this screenshot we search for packages containing the string **net-tools** in their name or description. This search happens locally on your server.

```
root@debian10:~# apt-cache search net-tools
ddnet-tools - Tools for DDNet
hobbit-plugins - plugins for the Xymon network monitor
iproute2 - networking and traffic control tools
atm-tools - Base programs for ATM in Linux, the net-tools for ATM
net-tools - NET-3 networking toolkit
root@debian10:~#
```

42.1.6 apt-get install

Use **apt-get install** to install a package. This will first download the package, it will verify the integrity of the package and it will perform all actions necessary to install the package.

```
root@debian10:~# apt-get install net-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  net-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/248 kB of archives.
After this operation, 1002 kB of additional disk space will be used.
Selecting previously unselected package net-tools.
(Reading database ... 40886 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20180626.aebd88e-1_amd64.deb ...
Unpacking net-tools (1.60+git20180626.aebd88e-1) ...
Setting up net-tools (1.60+git20180626.aebd88e-1) ...
Processing triggers for man-db (2.8.5-2) ...
root@debian10:~#
```

42.1.7 apt-get remove

If you no longer need a package, then you can remove it in two distinct ways. The first is a simple **apt-get remove** of the package. This will remove the package, but some configuration may remain.

```
root@debian10:~# apt-get remove net-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  net-tools
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 1002 kB disk space will be freed.
Do you want to continue? [Y/n]
(Reading database ... 40943 files and directories currently installed.)
Removing net-tools (1.60+git20180626.aebd88e-1) ...
Processing triggers for man-db (2.8.5-2) ...
root@debian10:~#
```

If you want to remove a package and all its configuration, then you can use **apt-get purge** on the package. Nothing of the package remains.

42.1.8 upgrade everything

There is a simple command to **upgrade** all your packages to the latest version, it is **apt-get upgrade**. This command is usually preceded by **apt-get update** to obtain the latest information about all packages. In the screenshot everything was already up-to-date.

```
root@debian10:~# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@debian10:~#
```

When a package upgrade requires new packages to be installed, then this upgrade will be withheld until you perform an **apt-get dist-upgrade**.

Tip
You should regularly run **apt-get update && apt-get upgrade**.

42.2 aptitude

Some administrators prefer the **aptitude** command to the **apt-get** command. The **aptitude** command uses the same configuration and can be mixed with the **apt-get** command. Here are some example commands with **aptitude** and the **net-tools** package.

```
aptitude update
aptitude upgrade
aptitude search net-tools
aptitude install net-tools
aptitude remove net-tools
aptitude purge net-tools
```

42.3 apt

If you prefer to type less, then you can use the **apt** tool, which is newer and entirely compatible with **apt-get** and with **aptitude**. Here are some example commands with **apt**.

```
apt update
apt upgrade
apt search net-tools
apt install net-tools
apt remove net-tools
apt purge net-tools
```

42.4 dpkg

Debian packages come in the **.deb** format. The **depian package manager** tool named **dpkg** is the back-end to **aptitude** that installs and removes these **.deb** packages. We list some useful options to this tool here.

42.4.1 dpkg -l

The **dpkg -l $somepackage** command will display the desired state of a package and whether the package matches the desired state. For example **ii** signifies the package is desired to be installed (the first i) and it is effectively installed (the second i).

```
root@debian10:~# dpkg -l net-tools
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version                   Architecture Description
+++-==============-=========================-============-===============================
ii  net-tools      1.60+git20180626.aebd88e-1 amd64        NET-3 networking toolkit
root@debian10:~#
```

The **dpkg -l** command without package argument will list all installed packages. This is a long list, so it is often combined with **grep** to quickly see the installed state of certain packages.

```
root@debian10:~# dpkg -l | wc -l
442
root@debian10:~# dpkg -l | grep openssh | cut -b-84
ii  openssh-client                 1:7.9p1-10                   amd64        secure s
ii  openssh-server                 1:7.9p1-10                   amd64        secure s
ii  openssh-sftp-server            1:7.9p1-10                   amd64        secure s
root@debian10:~#
```

42.4.2 dpkg -s

Use the **dpkg -s** command to get detailed information about a package, like the name and e-mail of the maintainer and whether it is installed.

```
root@debian10:~# dpkg -s net-tools | head -6
Package: net-tools
Status: install ok installed
Priority: important
Section: net
Installed-Size: 979
Maintainer: net-tools Team <team+net-tools@tracker.debian.org>
root@debian10:~#
```

42.4.3 dpkg -S

Using **dpkg -S** you can query to which package a certain file belongs. The screenshot proves that **/bin/netstat** is installed via the **net-tools** package.

```
root@debian10:~# dpkg -S /bin/netstat
net-tools: /bin/netstat
root@debian10:~#
```

Note
Remember that /**bin** is a link to /**usr**/**bin**, so you may have to double check some binaries.

42.4.4 dpkg -L

You can list all the files that are installed by a package with **dpkg --listfiles**, as this screenshot shows fro the **net-tools** package.

```
root@debian10:~# dpkg -L net-tools | head -6
/.
/bin
/bin/netstat
/sbin
/sbin/ifconfig
/sbin/ipmaddr
root@debian10:~#
```

42.5 Third party deb

With **fifty-nine thousand packages** available it is rare to require software from outside Debian. But there are packages that come in **.deb** format and that are not available in any of the Debian repositories.

42.5.1 dpkg --info

Downloading and installing a third party **.deb** package is at your own risk. It is up to you to trust the third party. With **dpkg --info** you get some information about a downloaded **.deb** package.

```
root@MBDebian~# dpkg --info heimer-1.11.0-Ubuntu-18.04_amd64.deb
 new Debian package, version 2.0.
 size 248770 bytes: control archive=538 bytes.
     367 bytes,    10 lines      control
     336 bytes,     5 lines      md5sums
 Package: heimer
 Version: 1.11.0
 Section: Education
 Priority: optional
 Architecture: amd64
 Depends: libc6 (>= 2.14), libgcc1 (>= 1:3.0), libqt5core5a (>= 5.9.0~beta), libqt5gui5 (>= ↩
      5.2.0), libqt5widgets5 (>= 5.2.0), libqt5xml5 (>= 5.0.2), libstdc++6 (>= 5.2)
 Installed-Size: 705
 Maintainer: Jussi Lind <jussi.lind@iki.fi>
 Description: Simple mind map creation tool.

root@MBDebian~#
```

42.5.2 dpkg -i

The **.deb** can be installed with **dpkg -i**. In this screenshot we install the **heimer** mindmap package for Ubuntu on a laptop with Debian 10.

```
root@MBDebian~# dpkg -i heimer-1.11.0-Ubuntu-18.04_amd64.deb
Selecting previously unselected package heimer.
(Reading database ... 196317 files and directories currently installed.)
Preparing to unpack heimer-1.11.0-Ubuntu-18.04_amd64.deb ...
Unpacking heimer (1.11.0) ...
Setting up heimer (1.11.0) ...
Processing triggers for desktop-file-utils (0.23-4) ...
Processing triggers for mime-support (3.62) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
root@MBDebian~#
```

42.5.3 dpkg -r

Removing a manually installed package can be done with **dpkg -r**, as this screenshot shows the removal of the **heimer** mindmap package.

```
root@MBDebian~# dpkg -r heimer
(Reading database ... 196321 files and directories currently installed.)
Removing heimer (1.11.0) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for desktop-file-utils (0.23-4) ...
Processing triggers for mime-support (3.62) ...
root@MBDebian~#
```

42.6 Third party source

It is entirely possible that an application is not available in Debian, but that the source code is published online. In this case there is hopefully a README or INSTALL file available to help you compile and install the application.

Sometimes installing from source code is a simple as three commands: **configure** followed by **make** and then as root **make install**.

42.6.1 example Heimer

The **Heimer** application required a bit more work to install from source. First we installed the prerequisites with **aptitude**, then as a normal user we ran **cmake** and **make**, and to finish we ran **make install** as root.

```
aptitude install qt5-default qttools5-dev-tools qttools5-dev cmake build-essential
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr
make
make install
```

Warning
Tools like **dpkg** and **apt-get** are not aware of applications installed from source.

42.7 Cheat sheet

Table 42.1: Software management

command	explanation
/etc/apt/sources.list	File that contains URL's to repositories.
/etc/apt/sources.d/	Directory that contains extra repository files.
apt-get update	Update the local information with the remote repositories.
apt-cache search foo	Search the repositories for packages with **foo** in their name.
apt-get install foo	Download and install package **foo**.
apt-get remove foo	Remove (=Uninstall) the package **foo**.
apt-get purge foo	Remove the package **foo** and all its configuration.
apt-get upgrade	Upgrade all installed packages to the latest version in the repo.
aptitude update	Update the local information with the remote repositories.
aptitude upgrade	Upgrade all packages to the latest version.
aptitude search foo	Search for all packages with **foo** in their name.
aptitude install foo	Download and install the package **foo**.
aptitude remove foo	Remove the package **foo**.
aptitude purge foo	Remove the package **foo** and all its configuration.
apt update	Update the local information with the remote repositories.
apt upgrade	Upgrade all packages to the latest version.
apt search foo	Search for all packages with **foo** in their name.
apt install foo	Download and install the package **foo**.
apt remove foo	Remove the package **foo**.
apt purge foo	Remove the package **foo** and all its configuration.
dpkg -l foo	Display information about package **foo**.
dpkg -l	List all installed and/or configured packages.
dpkg -s foo	Display detailed information about package **foo**.
dpkg -S bar	Query to which package the file **bar** belongs.
dpkg -L foo	List all files installed by the package **foo**.
dpkg -i foo.deb	Install the **foo.deb** package.
dpkg -r foo	Uninstall the **foo.deb** package.

42.8 Practice

1. Update the meta-information about all packages in the repository to your server.
2. Download and upgrade all packages on your server to the newest version.
3. Search for all packages containing the string zfs.
4. Take a backup of **/etc/apt/sources.list**.
5. Add the **contrib** packages to **/etc/apt/sources.list** .
6. Update all repository information, then search for zfs again.
7. Verify with **dpkg** whether any of the zfs packages are installed.
8. Verify which package installed **/bin/mkdir** .

42.9 Solution

1. Update the meta-information about all packages in the repository to your server.

```
apt update
apt-get update
aptitude update
```

2. Download and upgrade all packages on your server to the newest version.

```
apt-get upgrade
```

3. Search for all packages containing the string zfs.

```
aptitude search zfs
```

4. Take a backup of **/etc/apt/sources.list**.

```
cp /etc/apt/sources.list /etc/apt/sources.list.20190914
```

5. Add the **contrib** packages to **/etc/apt/sources.list** .

```
root@debian10:~# vi /etc/apt/sources.list
root@debian10:~# grep contrib /etc/apt/sources.list
deb http://deb.debian.org/debian/ buster main contrib
deb-src http://deb.debian.org/debian/ buster main contrib
root@debian10:~#
```

6. Update all repository information, then search for zfs again.

```
aptitude update
aptitude search zfs    # You should see more zfs packages.
```

7. Verify with **dpkg** whether any of the zfs packages are installed.

```
dpkg -l | grep -i zfs
```

8. Verify which package installed **/bin/mkdir** .

```
dpkg -S /bin/mkdir
```

Part V

Linux storage

Chapter 43

Introduction to storage

43.1 Introduction

There are four major components to managing local storage, each component has its own chapter. Each component is mandatory prerequisite knowledge for the other storage chapters.

- Detecting hardware (this chapter)
- Creating partitions (next chapter)
- Creating filesystems
- Mounting filesystems

The following chapters will continue this storage story.

- Troubleshooting storage
- UUIDs
- RAID
- LVM
- iSCSI
- multipath devices

Note
For the screenshots we installed Debian 10 on a server named **server2**. This server has three extra 144GiB hard disks.

43.2 hard disk

A hard disk can either be a spinning disk, with tracks, heads and cylinders, or a Solid State Drive. A spinning hard disk has several platters with data in concentric circles called tracks. All tracks on top of each other are called cylinders. The magnetic reading and writing is done by heads that almost touch each side of the platters (except maybe the outside of the stack).

An SSD or Solid State Drive is new technology that contains no moving parts. Currently in 2019 SSDs are more expensive per megabyte than spinning disks.

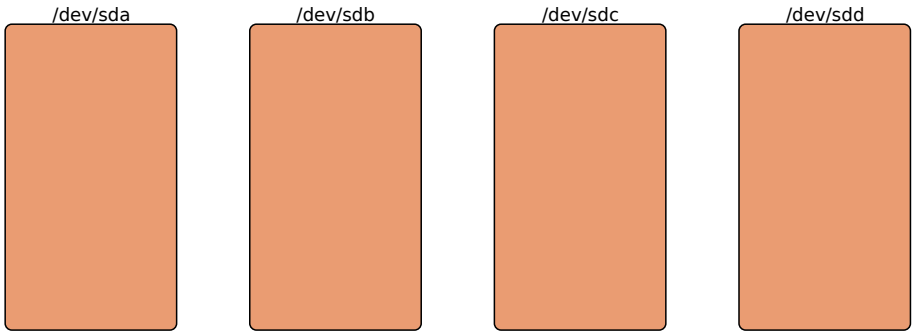
43.3 block device

A block device is a storage device with random access. Each block has an address and can be read and/or written to. Block devices can be recognised by the letter **b** as the first character in the output of **ls -l**.

```
root@server2:~# ls -l /dev | grep ^b
brw-rw---- 1 root disk      8,   0 Aug 22 19:12 sda
brw-rw---- 1 root disk      8,   1 Aug 22 19:12 sda1
brw-rw---- 1 root disk      8,   2 Aug 22 19:12 sda2
brw-rw---- 1 root disk      8,   5 Aug 22 19:12 sda5
brw-rw---- 1 root disk      8,  16 Aug 22 19:12 sdb
brw-rw---- 1 root disk      8,  32 Aug 22 19:12 sdc
brw-rw---- 1 root disk      8,  48 Aug 22 19:12 sdd
root@server2:~#
```

Note
A hard disk is a block device, and so is a **RAID**, **LVM**, **iSCSI** or **multipath** device. What we learn here about simple hard disk devices will be applicable later for all block devices.

In this book we picture these four hard disk (or any block devices) as follows.



43.4 lsblk

All block devices on a Debian Linux computer can be displayed using the **lsblk** command. Compare the output with the screenshot from **/dev/** from above.

```
root@server2:~# lsblk -i
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   16G  0 disk
|-sda1   8:1    0   15G  0 part /
|-sda2   8:2    0    1K  0 part
`-sda5   8:5    0 1022M  0 part [SWAP]
sdb      8:16   0  144G  0 disk
sdc      8:32   0  144G  0 disk
sdd      8:48   0  144G  0 disk
root@server2:~#
```

Now that we know the sizes of these hard disks, we can update our drawing. You may want to browse Wikipedia on **kibi**byte versus **kilo**byte here https://en.wikipedia.org/wiki/Kibibyte .

43.5 /dev/sd*

In Debian 10 almost all block devices use the **sd** driver for SCSI disk devices and thus get a **/dev/sd** device name. This includes extra hard disks, extra USB sticks, well every new block device.

You can look in **/proc/devices** to verify the driver for a major number. The **major number** of a device is visible in the **ls -l** output and corresponds to the driver in use for this device.

```
root@server2:/proc# grep sd /proc/devices | head -5
  8 sd
 65 sd
 66 sd
 67 sd
 68 sd
root@server2:/proc#
```

In our Debian 10 server the first hard disk is known as **/dev/sda**, the second to fourth are known as **/dev/sdb**, **/dev/sdc**, and **/dev/sdd**.

```
root@server2:~# ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda5  /dev/sdb  /dev/sdc  /dev/sdd
root@server2:~#
```

43.6 fdisk -l

The tool most people use to manage disks is called **fdisk**. Typing **fdisk -l** will list the attached block devices. The names of the block devices should be familiar by now.

```
root@server2:~# fdisk -l | grep sd
Disk /dev/sda: 16 GiB, 17179869184 bytes, 33554432 sectors
/dev/sda1  *        2048 31457279 31455232   15G 83 Linux
/dev/sda2       31459326 33552383  2093058 1022M  5 Extended
/dev/sda5       31459328 33552383  2093056 1022M 82 Linux swap / Solaris
Disk /dev/sdb: 144 GiB, 154618822656 bytes, 301989888 sectors
Disk /dev/sdc: 144 GiB, 154618822656 bytes, 301989888 sectors
Disk /dev/sdd: 144 GiB, 154618822656 bytes, 301989888 sectors
root@server2:~#
```

43.7 dmesg

The **dmesg** command will display messages from the kernel (including when it was booting) and can provide some information on attached storage devices. In the screenshot we query **dmesg** for messages regarding **sdd**.

```
root@server2:~# dmesg | grep sdd
[    3.052894] sd 5:0:0:0: [sdd] 301989888 512-byte logical blocks: (155 GB/144 GiB)
[    3.052896] sd 5:0:0:0: [sdd] Write Protect is off
[    3.052897] sd 5:0:0:0: [sdd] Mode Sense: 00 3a 00 00
[    3.052901] sd 5:0:0:0: [sdd] Write cache: enabled, read cache: enabled, doesn't support ↩
     DPO or FUA
[    3.053928] sd 5:0:0:0: [sdd] Attached SCSI disk
root@server2:~#
```

43.8 lshw

The **lshw** tool can be installed with **apt-get install lshw**. It will give information on all hardware and how it is connected to your computer. Look for **disk** and find the SCSI connections.

```
root@server2:~# lshw | grep -A1 -i SCSI
             logical name: scsi0
             logical name: scsi1
             logical name: scsi2
```

```
             logical name: scsi3
             version: 02
--
                bus info: scsi@0:0.0.0
                logical name: /dev/sda
--
                   bus info: scsi@0:0.0.0,1
                   logical name: /dev/sda1
--
                   bus info: scsi@0:0.0.0,2
                   logical name: /dev/sda2
--
                bus info: scsi@1:0.0.0
                logical name: /dev/sdb
--
                bus info: scsi@2:0.0.0
                logical name: /dev/sdc
--
                bus info: scsi@3:0.0.0
                logical name: /dev/sdd
root@server2:~#
```

43.9 lsscsi

The **lsscsi** tool can be installed with **apt-get install lsscsi**. This tool can display SCSI devices in several output formats.

```
root@server2:~# lsscsi --brief
[0:0:0:0]    /dev/sda
[1:0:0:0]    /dev/sdb
[2:0:0:0]    /dev/sdc
[3:0:0:0]    /dev/sdd
root@server2:~#
```

43.10 Cheat sheet

Table 43.1: Introduction to storage

command	**explanation**
lsblk	List block devices.
fdisk -l	List block devices with more information.
dmesg	Display kernel messages (including detection of block devices)..
lshw	List hardware (including block devices).
lsscsi	List SCSI devices.
/proc/devices	Location that shows the link between major numbers and drivers.

43.11 Practice

1. Prepare a (virtual) server with at least three extra disks.
2. List all block devices in **/dev**.
3. List all block devices on the computer (not using ls).
4. Verify that the block devices use the **sd** driver.
5. Use **fdisk** to list all disks.
6. Use **lsscsi** to list all SCSI devices.
7. Use **lshw** and look at the complete device tree.

43.12 Solution

1. Prepare a (virtual) server with at least three extra disks.

```
If you don't have a real server then you can use Vmware or Virtualbox,
and add three extra virtual disks.
Or use a Raspberry Pi with three old USB sticks.
```

2. List all block devices in **/dev**.

```
ls -l /dev | grep ^b
```

3. List all block devices on the computer (not using ls).

```
lsblk
```

4. Verify that the block devices use the **sd** driver.

```
Check the major number (probably 8) in cat /proc/devices.
```

5. Use **fdisk** to list all disks.

```
fdisk -l
```

6. Use **lsscsi** to list all SCSI devices.

```
apt-get install lsscsi
lsscsi
```

7. Use **lshw** and look at the complete device tree.

```
apt-get install lshw
lshw
```

Chapter 44

Partitioning block devices

44.1 Primary partition

To prepare a disk (or any block device) for use, you have to partition it. Even if you create one partition for the whole disk, this step is necessary. The boot disk (named **/dev/sda**) of this server has one primary partition.

The first primary partition is named **/dev/sda1**. In theory there could have been a second, a third and a fourth primary partition, named **/dev/sda2**, **/dev/sda3** and **/dev/sda4**. Four is the maximum number of primary partitions.

```
paul@server2:~$ ls -l /dev/sda*
brw-rw---- 1 root disk 8, 0 Aug 24 11:21 /dev/sda
brw-rw---- 1 root disk 8, 1 Aug 24 11:21 /dev/sda1
brw-rw---- 1 root disk 8, 2 Aug 24 11:21 /dev/sda2
brw-rw---- 1 root disk 8, 5 Aug 24 11:21 /dev/sda5
paul@server2:~$
```

44.2 /proc/partitions

The partitions are also visible in **/proc/partitions**, together with their major number (8 is a SCSI disk driver).

```
paul@server2:~$ cat /proc/partitions | grep sda
   8        0   16777216 sda
   8        1   15727616 sda1
   8        2          1 sda2
   8        5    1046528 sda5
paul@server2:~$
```

Tip
The grep command is here to limit the size of the screenshots, you can execute the command without grep.

44.3 master boot record

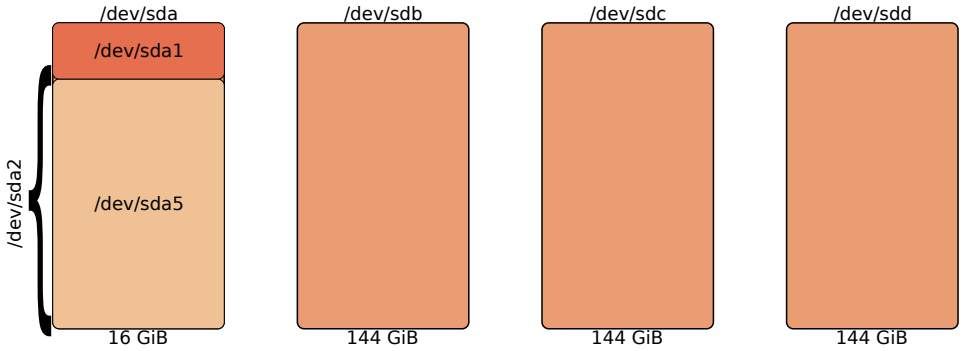
Information about primary partitions are stored in the **master boot record** or **MBR**. The **master boot record** is the first sector of the disk, so the first 512 bytes.

You can take a backup of the **master boot record** with the **dd** command, as shown in this screenshot.

```
root@server2:~# dd if=/dev/sda of=sda.mbr count=1 bs=512
1+0 records in
1+0 records out
512 bytes copied, 0.000827762 s, 619 kB/s
root@server2:~# file sda.mbr
sda.mbr: DOS/MBR boot sector
root@server2:~#
```

44.4 extended partitions

You can sacrifice one of the four primary partitions to create an **extended partition**. In this extended partition you can create many **logical drives**. The use of a **logical drive** is almost identical to using a **primary partition**.



Our server here has an **extended partition** named **/dev/sda2** and one **logical drive** named **/dev/sda5**. Logical drive naming always starts at 5, since 1 to 4 are reserved for primary partitions. You can create only one extended partition!

```
paul@server2:~$ ls -l /dev/sda*
brw-rw---- 1 root disk 8, 0 Aug 24 11:21 /dev/sda
brw-rw---- 1 root disk 8, 1 Aug 24 11:21 /dev/sda1
brw-rw---- 1 root disk 8, 2 Aug 24 11:21 /dev/sda2
brw-rw---- 1 root disk 8, 5 Aug 24 11:21 /dev/sda5
paul@server2:~$
```

44.5 fdisk

One tool to manage partitions on a fixed disk (or any block device) is **fdisk**. Typing **fdisk -l /dev/sda** will provide a listing of the **/dev/sda** disk.

```
root@server2:~# fdisk -l /dev/sda | tail
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xd5c7efec

Device     Boot    Start      End  Sectors  Size Id Type
/dev/sda1  *        2048 31457279 31455232   15G 83 Linux
/dev/sda2       31459326 33552383  2093058 1022M  5 Extended
/dev/sda5       31459328 33552383  2093056 1022M 82 Linux swap / Solaris
root@server2:~#
```

Notice that the one logical drive spans the whole extended partition. No extra partitions or logical drives can be created on this disk.

44.5.1 Creating primary partitions with fdisk

Typing **fdisk /dev/sdb** will open an interactive menu to manage the **/dev/sdb** disk. From this menu we can create partitions and logical drives.

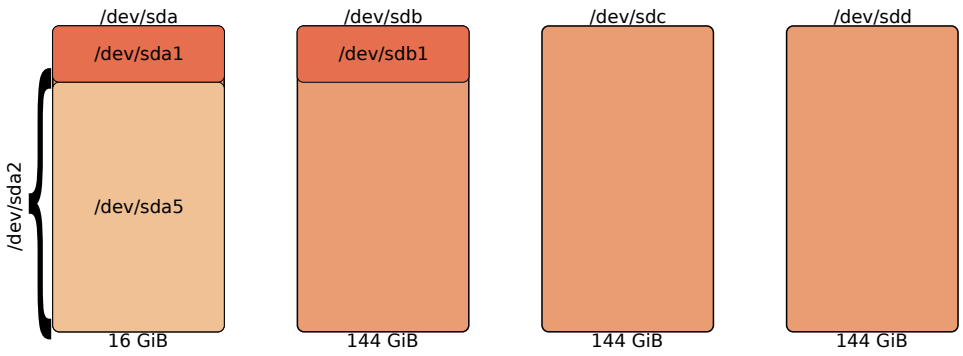
```
root@server2:~# fdisk /dev/sdb

Welcome to fdisk (util-linux 2.33.1).
```

```
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x45dca5cf.

Command (m for help):
```

You can type **m** for help with the commands. We will type **n** here for a **n**ew partition and continue with creating a 20 Gibibyte partition.

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-301989887, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-301989887, default 301989887): +20G

Created a new partition 1 of type 'Linux' and of size 20 GiB.

Command (m for help):
```

And here is a picture of the device we just created in **fdisk**.



We can now use **p** to print the partition information and verify our work.

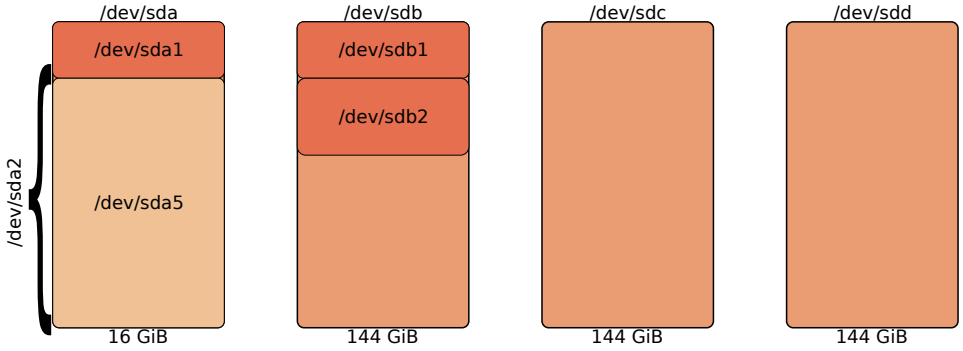
```
Command (m for help): p
Disk /dev/sdb: 144 GiB, 154618822656 bytes, 301989888 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x45dca5cf

Device     Boot Start      End  Sectors Size Id Type
/dev/sdb1        2048 41945087 41943040  20G 83 Linux

Command (m for help):
```

We will use **fdisk** to create one more primary partition of 40Gib size.

```
Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
```

```
Partition number (2-4, default 2):
First sector (41945088-301989887, default 41945088):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (41945088-301989887, default 301989887): +40G

Created a new partition 2 of type 'Linux' and of size 40 GiB.

Command (m for help):
```

We can now update our drawing with the new primary partition.

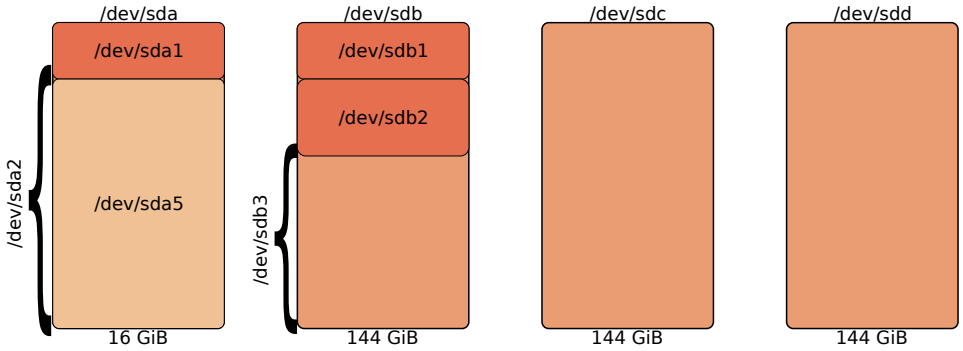


44.5.2 Creating logical drives with fdisk

We will use the rest of the disk to create an **extended partition**. When you don't enter any size, then the rest of the disk will be used.

```
Command (m for help): n
Partition type
   p   primary (2 primary, 0 extended, 2 free)
   e   extended (container for logical partitions)
Select (default p): e
Partition number (3,4, default 3):
First sector (125831168-301989887, default 125831168):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (125831168-301989887, default 301989887):

Created a new partition 3 of type 'Extended' and of size 84 GiB.

Command (m for help):
```

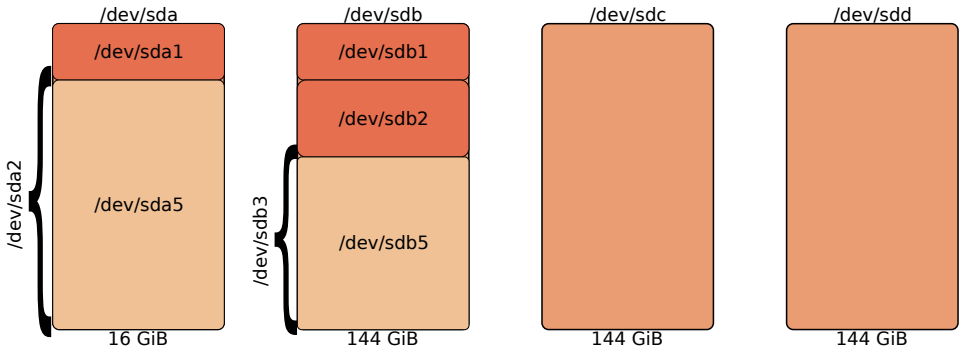
We picture the extended partition outside of the disk, because we want to put **logical drives** inside it. But it is part of the disk of course.



And inside the **extended partition** we will create one **logical drive** spanning the whole extended partition. You could create several smaller logical drives.

```
Command (m for help): n
All space for primary partitions is in use.
Adding logical partition 5
First sector (125833216-301989887, default 125833216):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (125833216-301989887, default 301989887):

Created a new partition 5 of type Linux and of size 84 GiB.

Command (m for help):
```

We update our drawing to include the new logical drive.



44.5.3 writing changes to disk with fdisk

The final step in **fdisk** is to decide whether to write your changes to disk or to quit without changing anything on the disk. We decide to write our changes to disk.

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@server2:~#
```

Important
When typing **w** the fdisk tool will do **two** actions; one it will write the changes to disk, and two it will tell the kernel to reload the partition table of the disk.

We can now verify our work with **fdisk -l /dev/sdb**.

```
root@server2:~# fdisk -l /dev/sdb | tail
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x45dca5cf

Device     Boot     Start       End   Sectors Size Id Type
/dev/sdb1            2048  41945087  41943040  20G 83 Linux
/dev/sdb2        41945088 125831167  83886080  40G 83 Linux
/dev/sdb3       125831168 301989887 176158720  84G  5 Extended
/dev/sdb5       125833216 301989887 176156672  84G 83 Linux
root@server2:~#
```

44.6 parted

You can install **parted** with **apt-get install parted**. The **parted** command is preferred by some administrators to **fdisk**.

44.6.1 Viewing partition tables with parted

You can start an interactive **parted** with a disk as the argument. Here we use are **/dev/sdb** that we prepared with **fdisk**.

```
root@server2:~# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

You can use **?** in parted for a list of commands.

We can verify our work with the **print** command. Notice how **parted** uses gigabytes by default instead of **fdisk**'s gibibytes.

```
(parted) print
Disk /dev/sdb: 155GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  21.5GB  21.5GB  primary
 2      21.5GB  64.4GB  42.9GB  primary
 3      64.4GB  155GB   90.2GB  extended
 5      64.4GB  155GB   90.2GB  logical

(parted)
```

We can select another disk within **parted**. In the screenshot below we switch to the **/dev/sdc** disk and verify that no partitions were created on it.

```
(parted) select /dev/sdc
Using /dev/sdc
(parted) print
Error: /dev/sdc: unrecognised disk label
Disk /dev/sdc: 155GB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
(parted)
```

44.6.2 Creating primary partitions with parted

Warning
The **parted** tool immediately writes changes to the disk.

In **parted** you have to label a disk before creating partitions. Labels are discussed later in this chapter.

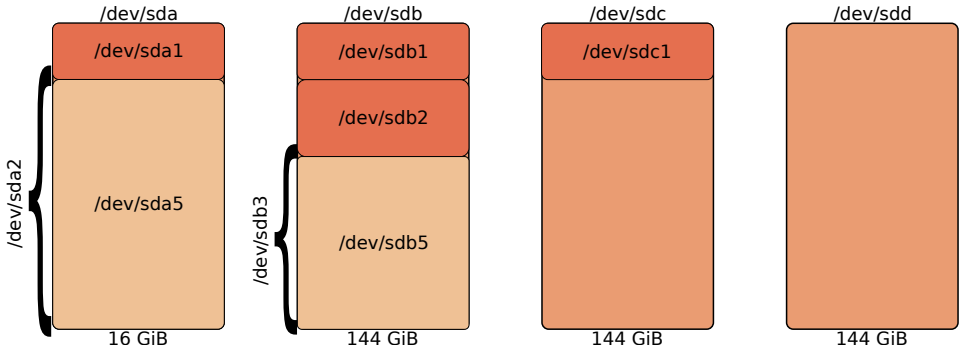
```
(parted) mklabel msdos
(parted) print
Disk /dev/sdc: 155GB
```

```
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End  Size  Type  File system  Flags

(parted)
```

In this screenshot we create a 20 gigabyte primary partition. The warning we get is only valid for **spinning** disks, not for SSDs.

```
(parted) mkpart primary 0 20GB
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
(parted) print
Disk /dev/sdc: 155GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End     Size    Type     File system  Flags
 1      512B   20.0GB  20.0GB  primary               lba

(parted)
```

In our drawing we can now update the third disk device.



44.6.3 Resizing partitions with parted

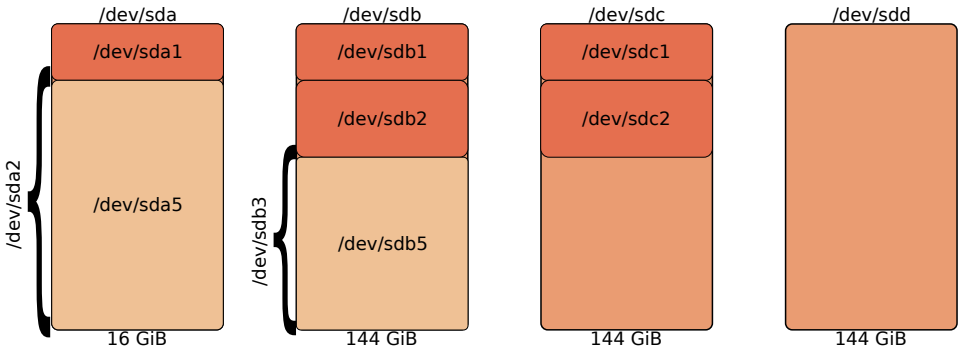
We just created a smaller partition than before, just 20 gigabytes instead of 20 gibibytes. Luckily **parted** allows for resizing of partitions, and it also understands **GiB**.

```
(parted) resizepart
Partition number? 1
End?  [20.0GB]? 20GiB
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sdc: 155GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End     Size    Type     File system  Flags
 1      512B   21.5GB  21.5GB  primary               lba

(parted)
```

44.6.4 Creating a second primary partition

So now let us create the second primary partition of 40 Gibibytes.

```
(parted) mkpart primary 20GiB 60GiB
Warning: You requested a partition from 21.5GB to 64.4GB (sectors 41943040..125829119).
The closest location we can manage is 21.5GB to 64.4GB (sectors 41943041..125829119).
Is this still acceptable to you?
Yes/No? Yes
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
(parted)
```

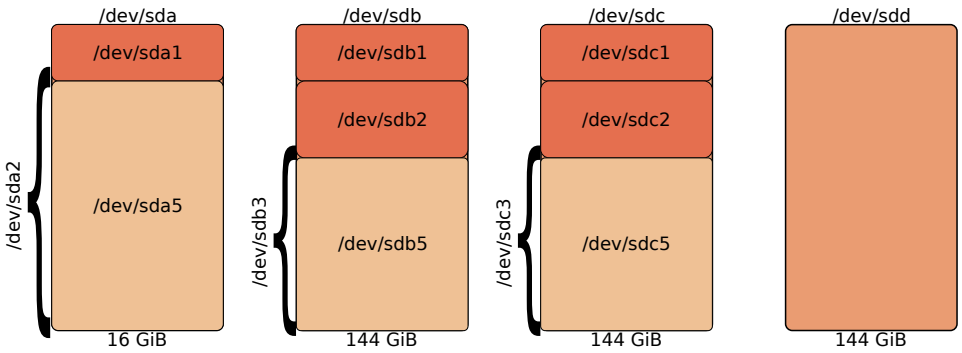


44.6.5 Creating logical drives with parted

To finish we will create an extended partition that encompasses the rest of the disk and fill it with one logical drive.

```
(parted) mkpart extended 60GiB 155GB
(parted) mkpart logical 60GiB 155GB
Warning: You requested a partition from 64.4GB to 155GB (sectors 125829120..301989887).
The closest location we can manage is 64.4GB to 155GB (sectors 125829121..301989887).
Is this still acceptable to you?
Yes/No? Y
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
(parted)
```

This finalises our drawing for **/dev/sdc**, which we partitioned with **parted**.

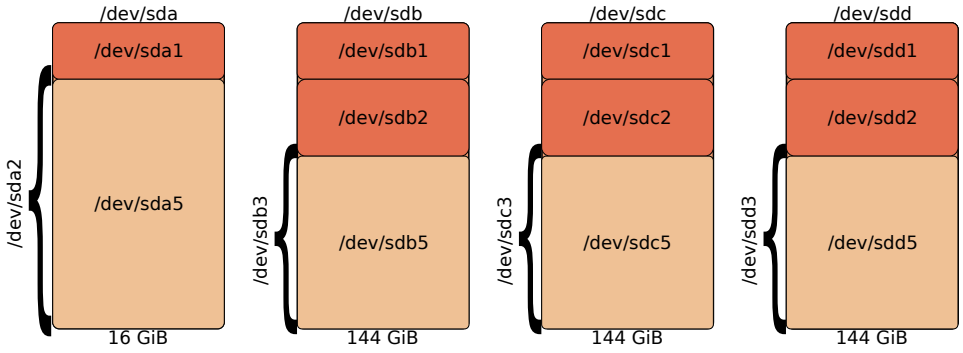


44.7 parted one liners

You can use **parted** one liners to create a similar layout on **/dev/sdd**. See the screenshot below.

```
root@server2:~# parted /dev/sdd mklabel msdos mkpart primary 0 20Gib
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server2:~# parted /dev/sdd mkpart primary 20Gib 60Gib
Information: You may need to update /etc/fstab.

root@server2:~# parted /dev/sdd mkpart extended 60Gib 100%
Information: You may need to update /etc/fstab.

root@server2:~# parted /dev/sdd mkpart logical 60Gib 100%
Warning: You requested a partition from 64.4GB to 155GB (sectors 125829120..301989887).
The closest location we can manage is 64.4GB to 155GB (sectors 125829121..301989887).
Is this still acceptable to you?
Yes/No? Y
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server2:~#
```

And here is a picture of what we did on **/dev/sdd**.



44.8 Labeling disks

Up until now we have been using disks with a DOS aka MBR label. With a **label** we mean the partition table type. Ever since the 1980s it was possible to use this partition table type and Linux today still supports it. But MBR partitions cannot be larger than two tebibyte.

Modern computer systems use GPT, short for **GUID partition table**, which is part of the UEFI standard since 2010. GPT allows for more and for much larger partitions, up to 8 zebibyte. A GPT partition table will put an MBR in the first 512-byte sector.

More info on GPT can be found here https://en.wikipedia.org/wiki/GUID_Partition_Table .

Debian Linux can boot from and use MBR and GPT partitions (and others but that is out of the scope of this book).

44.8.1 Creating a GPT label with fdisk

The **fdisk** tool has support for GPT labels. In the screenshot below we show how to set a GPT label on a disk. This erases all existing partitions and an existing GPT or MBR label.

```
root@server2:~# fdisk /dev/sdd

Welcome to fdisk (util-linux 2.33.1).
Changes will remain in memory only, until you decide to write them.
```

```
Be careful before using the write command.


Command (m for help): g
Created a new GPT disklabel (GUID: 2937367A-C265-6045-BA7F-E65AD6128940).

Command (m for help):
```

When typing **n** for a new partition, then we are greeted with the option to create 128 partitions. There is no primary and extended, it is just partitions.

```
Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-301989854, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-301989854, default 301989854): +20GiB

Created a new partition 1 of type 'Linux filesystem' and of size 20 GiB.

Command (m for help):
```

44.8.2 Create a GPT label with parted

The **parted** tool also has support for a GPT label. It will warn you that existing partition information will be destroyed.

```
root@server2:~# parted /dev/sdc mklabel gpt
Warning: The existing disk label on /dev/sdc will be destroyed and all data on this disk  ↩
    will
be lost. Do you want to continue?
Yes/No? y
Information: You may need to update /etc/fstab.

root@server2:~#
```

The **mkpart** command in parted now expects a name for each partition.

```
root@server2:~# parted /dev/sdc
GNU Parted 3.2
Using /dev/sdc
Welcome to GNU Parted! Type help to view a list of commands.
(parted) mkpart first 0% 20GiB
(parted)
```

44.9 sfdisk, cfdisk

There are some other tools to manage partitions like **sfdisk** and **cfdisk**, and also some graphical tools like **gparted**. Using these tools is left as an exercise for the reader.

44.10 partprobe

If you are unsure that the kernel knows about a certain partition table on a disk, then you can use **partprobe** to tell the kernel about it. By default **partprobe** will scan all disk labels and tell the kernel about them.

```
root@server2:~# partprobe -s
/dev/sda: msdos partitions 1 2 <5>
/dev/sdb: msdos partitions 1 2 3 <5>
```

```
/dev/sdc: gpt partitions 1
/dev/sdd: gpt partitions 1
root@server2:~#
```

Tip
To avoid this scan of all disks with **partprobe**, you can you can specify the device to scan.

44.11 Cheat sheet

Table 44.1: Partitioning

command	**explanation**
fdisk -l /dev/sda	List partition information on /dev/sda.
fdisk /dev/sda	Open the **fdisk** partitioning tool for /dev/sda.
parted /dev/sda	Open the **parted** partitioning tool for /dev/sda.
partprobe	Tell the kernel to read all disk labels (with partition info).
/proc/partitions	Contains kernel info on partitions.

44.12 Practice

1. Make sure you identify your boot disk (which already has partitions) and don't touch this disk. (It is probably **/dev/sda**.)
2. Use **fdisk** to create a primary partition on the first extra disk.
3. Verify your work from question 2.
4. Use the rest of the second disk as an extended partition.
5. Create a logical drive in the extended partitions.
6. Use **parted** to create a primary partition on the second extra disk.
7. Use the rest of the disk as an extended partition.
8. Create a logical drive with parted.
9. Open the manual of **fdisk** and **parted** and read about labels (partition types).

44.13 Solution

1. Make sure you identify your boot disk (which already has partitions) and don't touch this disk. (It is probably **/dev/sda**.)

```
fdisk -l
```

2. Use **fdisk** to create a primary partition on the first extra disk.

```
fdisk /dev/sdb    ## replace with your first extra disk
n
p
+2G
w
```

3. Verify your work from question 2.

```
fdisk -l /dev/sdb
```

4. Use the rest of the second disk as an extended partition.

```
fdisk /dev/sdb
n
e
w
```

5. Create a logical drive in the extended partitions.

```
fdisk /dev/sdb
n
l
+1GB
w
```

6. Use **parted** to create a primary partition on the second extra disk.

```
parted /dev/sdc ## replace with your second extra disk
mklabel msdos
mkpart primary 0 2GiB
I
print
```

7. Use the rest of the disk as an extended partition.

```
parted /dev/sdc
mkpart extended 2GiB 100%
```

8. Create a logical drive with parted.

```
mkpart logical 2GiB 4GiB
```

9. Open the manual of **fdisk** and **parted** and read about labels (partition types).

```
man fdisk
man parted
```

Chapter 45

Introduction to file systems

45.1 man fs

You need to put a **filesystem** on a partition before you can put files and directories on it. Without a filesystem a partition is useless.

There are many **filesystems** supported in Linux, see **man fs** for a list. The most common filesystem on Linux today is **ext4**, It is also the default on Debian 10. Our **server2** has the **ext4** filesystem on /dev/sda1.

```
root@server2:~# parted /dev/sda print
Model: ATA (scsi)
Disk /dev/sda: 17.2GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system     Flags
 1      1049kB  16.1GB  16.1GB  primary   ext4            boot
 2      16.1GB  17.2GB  1072MB  extended
 5      16.1GB  17.2GB  1072MB  logical   linux-swap(v1)

root@server2:~#
```

45.2 /proc/filesystems

The Linux kernel supports many filesystems, but only some are compiled in the kernel. The rest will be loaded as a module when needed. You can see a list of currently supported filesystems in **/proc/filesystems** .

```
root@server2:~# grep -v nodev /proc/filesystems
        ext3
        ext2
        ext4
root@server2:~#
```

45.3 ext4

Since **ext4** is the default and the most common, it is appropriate to talk about some of its features. The **fourth extended file system** can grow to 1 Exbibyte with file size up to 16 tebibyte with the standard 4KiB block size. It has journaling, a fast file system check, it can have 6 billion entries in a directory, and timestamps are in nanoseconds. See the man page for more details or visit Wikipedia here https://en.wikipedia.org/wiki/Ext4 .

But **ext4** is still a basic filesystem, ideal and default for the root partition on most Linux distributions. It does not have features like a **volume manager**, **integrity checking** and it doesn't scale well beyond 100 TiB.

```
root@server2:~# man ext4
root@server2:~#
```

45.4 File Allocation Table

The **FAT** filesystem has a history since 1977, with prominence in the DOS and Windows 95 era. It is supported by all operating systems and its derivatives like **FAT32** and **exFAT** are still in use today. **exFAT** notably on SDXC cards larger than 32GiB and on newer USB sticks. There is no security on these filesystems, and files do not have owners.

Below a screenshot on what happens when you insert an old 32GB SD card. The **vfat** driver is loaded and **fdisk** will see a new *disk* with a **fat32** filesystem.

```
paul@MBDebian~$ grep -v nodev /proc/filesystems
  ext3
  ext2
  ext4
  vfat
paul@MBDebian~$ su -
Password:
root@MBDebian~# fdisk -l /dev/sdb
Disk /dev/sdb: 29.9 GiB, 32094814208 bytes, 62685184 sectors
Disk model: SD Card Reader
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device     Boot Start      End  Sectors  Size Id Type
/dev/sdb1        8192 62685183 62676992 29.9G  c W95 FAT32 (LBA)
root@MBDebian~#
```

The same happens when inserting a 128GB USB stick. The **vfat** driver will handle the FAT32 filesystem on the stick.

```
root@MBDebian~# fdisk -l /dev/sdc | grep sdc
Disk /dev/sdc: 114.6 GiB, 123010547712 bytes, 240254976 sectors
/dev/sdc1          32 240254975 240254944 114.6G  c W95 FAT32 (LBA)
root@MBDebian~#
```

Much more information on FAT and its derivatives here https://en.wikipedia.org/wiki/File_Allocation_Table and also here https://en.wiki
wiki/ExFAT .

45.5 ISO9660 and UDF

Mounting a CD-ROM will add the **ISO9660** filesystem to **/proc/filesystems**. A CD-ROM does not show up in **fdisk** but will get a line in **df -h**.

```
root@server2:~# df -hT | grep iso
/dev/sr0       iso9660   334M  334M     0 100% /mnt
root@server2:~# grep -v nodev /proc/filesystems
        ext3
        ext2
        ext4
        iso9660
root@server2:~#
```

Mounting a DVD will add the **UDF** filesystem to **/proc/filesystems**. Similar to a CD-ROM it will show up in **df** when mounted, but not in **fdisk**.

```
oot@server2:~# grep udf /proc/filesystems
        udf
root@server2:~# df -hT | grep udf
/dev/sr0       udf       3.7G  3.7G     0 100% /mnt
root@server2:~#
```

45.6 ZFS

The **zettabyte file system** or ZFS was developed by Sun and part of OpenSolaris in 2005. ZFS is a complete filesystem with volume manager, redundancy, encryption, networking and much more. The problem is that ZFS is licensed under the CDDL, which may be incompatible with the GPL License from the Linux kernel.

We will discuss all aspects of ZFS in a separate chapter.

45.7 mkfs.ext4

It is time now to put a filesystem on our partitions from the previous chapter. We will start with a partition on **/dev/sdb** which uses the MBR partition table type. Recreate these partitions if necessary.

```
root@server2:~# fdisk -l /dev/sdb | tail -5
Device     Boot     Start       End   Sectors Size Id Type
/dev/sdb1            2048  41945087  41943040  20G 83 Linux
/dev/sdb2        41945088 125831167  83886080  40G 83 Linux
/dev/sdb3       125831168 301989887 176158720  84G  5 Extended
/dev/sdb5       125833216 301989887 176156672  84G 83 Linux
root@server2:~#
```

The **mkfs.ext4** command will create an **ext4** filesystem on the **/dev/sdb1** partition when executing **mkfs.ext4 /dev/sdb1** . You can see in the screenshot that more than five million 4K blocks and more than one million **inodes** were created.

```
root@server2:~# mkfs.ext4 /dev/sdb1
mke2fs 1.44.5 (15-Dec-2018)
Creating filesystem with 5242880 4k blocks and 1310720 inodes
Filesystem UUID: cfcbcfa8-9df4-4ecd-b24e-a6adcc42b4d3
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

root@server2:~#
```

You can add a lot of options when using **mkfs.ext4**. For example for the filesystem on **/dev/sdb2** we will create far fewer **inodes** (because we plan to only put large files on it). The **-i** option specifies the number of bytes per inode.

```
root@server2:~# mkfs.ext4 -i 1024K /dev/sdb2
mke2fs 1.44.5 (15-Dec-2018)
Creating filesystem with 10485760 4k blocks and 40960 inodes
Filesystem UUID: 83ae9263-bf4c-4385-9050-e556d2459382
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624

Allocating group tables: done
Writing inode tables: done
Creating journal (65536 blocks): done
Writing superblocks and filesystem accounting information: done

root@server2:~#
```

Look in the **man mkfs.ext4** for many more options. Note also that **mkfs.ext2**, **mkfs.ext3**, and **mkfs.ext4** are all links to the same program.

```
paul@debian10:~$ ls -l /sbin/mkfs.ext*
lrwxrwxrwx 1 root root 6 Sep 25 19:37 /sbin/mkfs.ext2 -> mke2fs
lrwxrwxrwx 1 root root 6 Sep 25 19:37 /sbin/mkfs.ext3 -> mke2fs
lrwxrwxrwx 1 root root 6 Sep 25 19:37 /sbin/mkfs.ext4 -> mke2fs
paul@debian10:~$
```

45.8 tune2fs

With **tune2fs** you can list properties and options of existing file systems. For example if we want to know how many blocks are reserved for the root user, then we execute **tune2fs -l** on the device and grep for our property.

```
root@server2:~# tune2fs -l /dev/sdb2 | grep 'Reserved block '
Reserved block count:     524288
root@server2:~#
```

We can change this with **tune2fs** to zero, as is shown in this screenshot.

```
root@server2:~# tune2fs -m 0 /dev/sdb2
tune2fs 1.44.5 (15-Dec-2018)
Setting reserved blocks percentage to 0% (0 blocks)
root@server2:~# tune2fs -l /dev/sdb2 | grep 'Reserved block '
Reserved block count:     0
root@server2:~#
```

45.9 resize2fs

In later chapters we will see how a block device can grow in size while the filesystem is in use. When this happens then we also need to resize the **ext4** filesystem to match the size of the block device. This is done with **resize2fs**.

```
root@server2:~# resize2fs /dev/md2
resize2fs 1.44.5 (15-Dec-2018)
Filesystem at /dev/md2 is mounted on /srv/pro42; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/md2 is now 523264 (4k) blocks long.

root@server2:~#
```

45.10 fsck

The computer will from time to time do an automated **fsck** or **file system check**, so this is not a common task. But you can initiate a **file system check** manually using the **fsck /device** command shown in this screenshot.

```
root@server2:~# fsck /dev/sdb2
fsck from util-linux 2.33.1
e2fsck 1.44.5 (15-Dec-2018)
/dev/sdb2: clean, 11/40960 files, 81102/10485760 blocks
root@server2:~#
```

45.11 Cheat sheet

Table 45.1: File systems

command	**explanation**
man fs	Information about file systems.
parted /dev/foo print	Partition and file system information for **/dev/foo**.
fdisk -l /dev/foo	Partition information for **/dev/foo**.
cat /proc/filesystems	All kernel loaded file systems.
mkfs.ext2	Create (make) an ext2 file system.
mkfs.ext3	Create (make) an ext3 file system.
mkfs.ext4	Create (make) an ext4 file system.
tune2fs	Adjust parameters of an ext2/3/4 file system.
resize2fs	Resize an ext2/3/4 file system.
fsck	Run a file system check.
ext4	Default and most common file system on Debian Linux.
iso9660	CD-ROM file system.
udf	DVD file system.
zfs	Zettabyte File System (has its own chapter).

45.12 Practice

1. Open **man fs** and have a brief look at the filesystems mentioned.
2. Put an **ext4** filesystem on /dev/sdb1 (replace sdb with your first extra disk).
3. Verify your work with **fdisk** and with **lsblk** .
4. Also create an **ext4** on /dev/sdb2, with 0% reserved blocks for the root user.
5. Verify the reserved blocks with **tune2fs**.
6. Do a filesystem check on /dev/sdb2 .
7. Do the same exercises on a GPT label on /dev/sdd.

45.13 Solution

1. Open **man fs** and have a brief look at the filesystems mentioned.

```
man fs
```

2. Put an **ext4** filesystem on /dev/sdb1 (replace sdb with your first extra disk).

```
mkfs.ext4 /dev/sdb1
```

3. Verify your work with **fdisk** and with **lsblk** .

```
fdisk -l /dev/sdb
lsblk -f /dev/sdb
```

4. Also create an **ext4** on /dev/sdb2, with 0% reserved blocks for the root user.

```
mkfs.ext4 -m 0 /dev/sdb2
```

5. Verify the reserved blocks with **tune2fs**.

```
tune2fs -l /dev/sdb2
```

6. Do a filesystem check on /dev/sdb2 .

```
fsck /dev/sdb2
```

7. Do the same exercises on a GPT label on /dev/sdd.

```
fdisk /dev/sdd
mkfs.ext4 /dev/sdd1
mkfs.ext4 -m 0 /dev/sdd2
lsblk -f /dev/sdd
fdisk -l /dev/sdd
tune2fs -l /dev/sdd2
```

Chapter 46

File system mounting

This chapter is the fourth and final component of **basic storage introduction**. The first is recognising hardware, the second is partitioning and the third is filesystems.

46.1 mount

To be able to use a filesystem, we need to mount it to a **mount point**. The **mount point** is a directory that we create for this purpose. For example, if we want to share the filesystem on /dev/sdb1 for the **project 42** group, then we create **/srv/pro42**.

```
root@server2:~# mkdir /srv/pro42
root@server2:~#
```

The next step is to link this directory to the filesystem on /dev/sdb1, in other words, we **mount** the filesystem on **/srv/pro42** .

```
root@server2:~# mount /dev/sdb1 /srv/pro42/
root@server2:~#
```

From now on, whenever your create a file or directory in **/srv/pro42/**, then you create these on the filesystem on /dev/sdb1.

```
root@server2:~# echo Hello > /srv/pro42/welcome.txt
root@server2:~#
```

46.2 df

You can see mounted filesystems with the **df** tool. The **-h** option gives a human readable output for the sizes in multiples of kibibytes. Using **-H** will use multiples of kilobytes.

```
root@server2:~# df -h | grep sdb
/dev/sdb1        20G   45M   19G   1% /srv/pro42
root@server2:~#
```

The **-T** option will add the filesystem to the output of **df** .

```
root@server2:~# df -Th | grep sdb
/dev/sdb1      ext4       20G   45M   19G   1% /srv/pro42
root@server2:~#
```

46.3 du -sh

You can use **du -sh** to obtain the total size of a directory. In this case the directory is a mount point, so you get the total size of all files on the mounted filesystem.

```
root@server2:~# du -sh /srv/pro42/
24K     /srv/pro42/
root@server2:~#
```

46.4 /proc/mounts

Your **mount** is visible also in **/proc/mounts** and in **/etc/mtab** (the latter is a link to the former now). The **/proc/mounts** file holds information about what the **kernel** thinks is mounted.

```
root@server2:~# grep sdb /proc/mounts
/dev/sdb1 /srv/pro42 ext4 rw,relatime 0 0
root@server2:~# grep sdb /etc/mtab
/dev/sdb1 /srv/pro42 ext4 rw,relatime 0 0
root@server2:~# file /etc/mtab
/etc/mtab: symbolic link to ../proc/self/mounts
root@server2:~#
```

46.5 umount

A **mounted** filesystem can be unmounted with the **umount** command. This can be done either by using the mount point or by using the device.

```
root@server2:~# umount /srv/pro42
root@server2:~# mount /dev/sdb1 /srv/pro42/
root@server2:~# umount /dev/sdb1
root@server2:~#
```

46.6 /etc/fstab

The **file system table** can contain an entry for each filesystem that can be mounted. The required fields are **device, mountpoint, filesystem type, options, dump, and fsck**. The device is of course /dev/sdb1, the mountpoint is /srv/pro42, the filesystem type is ext4 (but you can put **auto** here), we use no options, so the **defaults** keyword is mandatory, and we don't want backups, nor filesystem checks at boot time.

```
root@server2:~# echo "/dev/sdb1 /srv/pro42 ext4 defaults 0 0" >> /etc/fstab
root@server2:~#
```

This line will make sure the filesystem is mounted automatically at boot. It also enables us to type **mount /srv/pro42** or **mount /dev/sdb1** as the mount command will look in **/etc/fstab**.

```
root@server2:~# mount /srv/pro42/
root@server2:~#
```

46.7 mount options

Several **mount options** can be used when mounting a filesystem. These options can be typed with a mount command, or can be set in **/etc/fstab**. A common option for troubleshooting a filesystem is **ro** to mount it as **read only**.

```
root@server2:~# umount /srv/pro42
root@server2:~# mount -o ro /srv/pro42/
root@server2:~# touch /srv/pro42/nope
touch: cannot touch '/srv/pro42/nope': Read-only file system
root@server2:~#
```

The **umount** and then **mount** combo can be avoided with the **remount** option. Using **remount** will keep the filesystem mounted while changing the options.

```
root@server2:~# mount -o remount,ro /srv/pro42/
root@server2:~#
```

An option that we discussed in the **ACLs** chapter is to mount a filesystem with ACL support. In this screenshot we add a line to **/etc/fstab** to automatically **mount** the /dev/sdd1 device with ACL support. (The defaults keyword can then be omitted.)

```
root@server2:~# echo '/dev/sdd1 /srv/pro33 auto acl 0 0' >> /etc/fstab
root@server2:~#
```

46.8 securing mounts

We already discussed the **ro** option to mount a filesystem **read only**. Not even root can then write to the filesystem. There are some other interesting security options.

The **noexec** option prevents execution of binaries or scripts from the filesystem. In the screenshot below we mount with this option and verify that not even root can start a script from this filesystem.

```
root@server2:~# mount -o noexec,remount /srv/pro33/
root@server2:~# echo echo hello > /srv/pro33/script.sh
root@server2:~# chmod +x /srv/pro33/script.sh
root@server2:~# /srv/pro33/script.sh
-bash: /srv/pro33/script.sh: Permission denied
root@server2:~#
```

In much the same way you can use the **nosuid** option and the **noacl** option.

46.9 Cheat sheet

Table 46.1: Mounting

command	explanation
mount foo bar	Mount the **foo** file system on the **bar** directory.
mount -o ro foo bar	Mount the **foo** file system read only on the **bar** directory.
mount -o noexec foo bar	Mount the **foo** file system on the **bar** directory, but prevent execution on that mount.
df -h	Display information about mounted file systems.
du -sh foo	Display the total size of the **foo** directory/mount point.
mount	Display information about all mounts.
umount foo	Unmount the **foo** file system/directory.
cat /proc/mounts	Display kernel information about all mounts.
/etc/fstab	File that contains information about file systems to mount at boot.

46.10 Practice

1. Choose a free disk, create a GPT label on it and five partitions.
2. Create an **ext4** filesystem on two of the partitions.
3. Create two mount points, one for project42 shared, one for project33 local.
4. Mount the two **ext4** filesystems on the mount points.
5. Add two lines for these mounts to **/etc/fstab**.
6. Test the lines in **/etc/fstab** by mounting then with one parameter to the **mount** command.
7. View your mounts with the **df** and the **mount** command.

46.11 Solution

1. Choose a free disk, create a GPT label on it and five partitions.

```
fdisk /dev/sdd
g
n n n n n
w
```

2. Create an **ext4** filesystem on two of the partitions.

```
mkfs.ext4 /dev/sdd1
mkfs.ext4 /dev/sdd2
```

3. Create two mount points, one for project42 shared, one for project33 local.

```
mkdir /srv/pro42
mkdir /home/pro33
```

4. Mount the two **ext4** filesystems on the mount points.

```
mount /dev/sdd1 /srv/pro42
mount /dev/sdd2 /home/pro33
```

5. Add two lines for these mounts to **/etc/fstab**.

```
echo '/dev/sdd1  /srv/pro42  auto defaults 0 0' >> /etc/fstab
echo '/dev/sdd2  /home/pro33  auto defaults 0 0' >> /etc/fstab
```

6. Test the lines in **/etc/fstab** by mounting then with one parameter to the **mount** command.

```
umount /dev/sdd1
mount /dev/sdd1
umount /dev/sdd1
mount /srv/pro42
umount /dev/sdd2
mount /dev/sdd2
umount /dev/sdd2
mount /home/pro33
```

7. View your mounts with the **df** and the **mount** command.

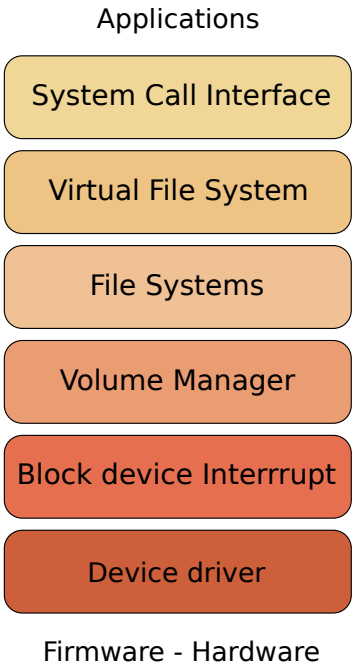
```
df -h
mount
```

Chapter 47

Monitoring and troubleshooting storage

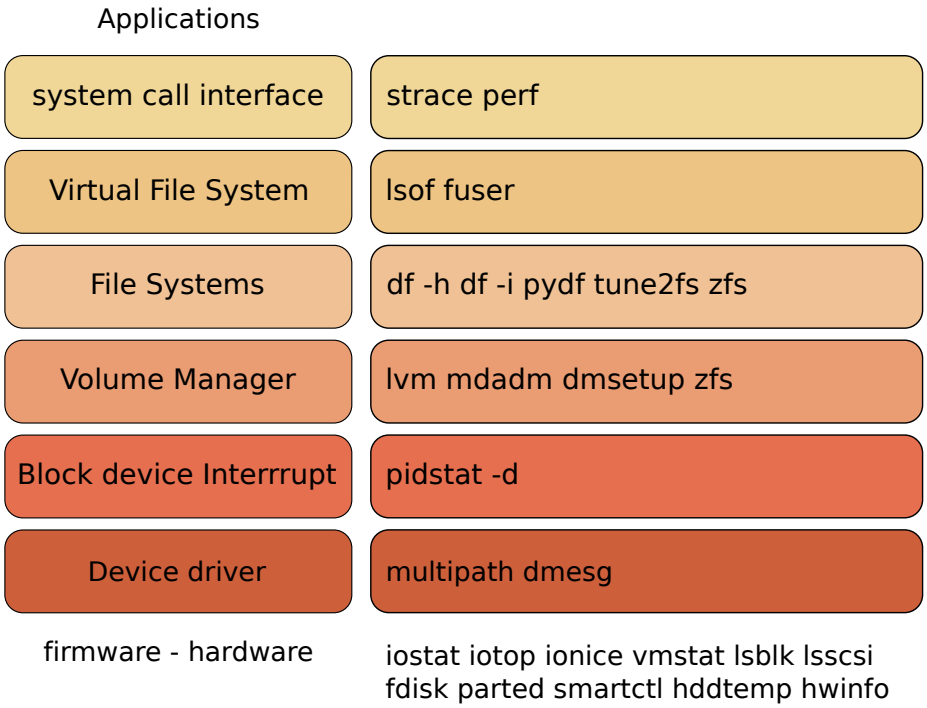
47.1 Operating system layers

There are six operating system layers between the Applications and the Hardware. These operating system layers are colored in the picture below.



Applications use the **System Call Interface** to open, read, write etc to files on a file system. These calls are the same calls for all file systems. In case of file access the **System Call Interface** will connect to the **Virtual File System** which interfaces with the **File System** (think for example **ext4**). In the end a kernel **device driver** will operate the hardware (including the firmware).

There are many tools to monitor and troubleshoot storage. The picture below tries to link the tools to the correct operating system layer.



47.2 System Call Interface

47.2.1 strace

The **strace** command can be installed with **apt-get install strace**. It traces all system function calls for a given command. A simple **ls** will generate close to one hundred calls whereas a humble **vi** goes well over a thousand calls.

In this small example we change a value in **/etc/shadow** with **vi** and save this change. We use **strace** on **vi** and write the **strace** output to a file named **output**.

```
root@debian10:~# ls -l /etc/shadow
-rw-r----- 1 root shadow 1568 Oct 14 11:31 /etc/shadow
root@debian10:~# strace 2> output vi /etc/shadow
root@debian10:~# ls -l /etc/shadow
-rw-r----- 1 root shadow 1568 Oct 14 11:31 /etc/shadow
root@debian10:~#
```

We get 1438 lines of output to analyse. In this example we look at the call to **chmod** to set the permissions of the file back to **640** (after writing to it).

```
root@debian10:~# wc -l output
1438 output
root@debian10:~# grep chmod output
chmod("/etc/.shadow.swp", 0640)         = 0
fchmod(3, 0100640)                      = 0
root@debian10:~#
```

One more small example of **strace** shows you how to attach to a running program using the **-p** option followed by a PID. In this example it is the PID of a bash shell and we execute **ls** during the trace.

```
root@debian10:~# strace -p 17394 -o output
strace: Process 17394 attached
^Cstrace: Process 17394 detached

root@debian10:~# grep read output
read(0, "l", 1)                         = 1
read(0, "s", 1)                         = 1
read(0, "\r", 1)                        = 1
root@debian10:~#
```

You can see that the **bash shell** reads the command line character by character.

47.2.2 perf

This tool depends on the kernel version, so you will have to install a new version if you use a different kernel with **apt-get install linux-tools-4.19** for kernel 4.19.

The **perf** tool can monitor almost two thousand events for performance analysis.

```
root@debian10:~# perf list | wc -l
1976
root@debian10:~#
```

For example to monitor context switches we first grep for the correct name of the event to monitor. Then we start the monitoring with **perf record**. This recording goes on until you press **Ctrl-C**. You can view the result with **perf report**.

```
root@debian10:~# perf list | grep context
  context-switches OR cs                            [Software event]
  filelock:locks_get_lock_context                   [Tracepoint event]
root@debian10:~# perf record -e cs -a
```

```
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.104 MB perf.data (33 samples) ]

root@debian10:~# perf report
root@debian10:~#
```

The **perf** tool can also be used as a one liner to execute a command and then give a little summary on the command's performance.

```
root@debian10:~# perf stat sleep 3

 Performance counter stats for sleep 3:

              0.89 msec task-clock                #    0.000 CPUs utilized
                 1      context-switches          #    0.000 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                57      page-faults               #    0.000 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       3.001848171 seconds time elapsed

       0.001571000 seconds user
       0.000000000 seconds sys

root@debian10:~#
```

There are hundreds of examples on using this command at http://www.brendangregg.com/perf.html .

47.3 Virtual File System

47.3.1 lsof

The **lsof** tool, short for **l**i**s** of **o**pen **f**iles, will list open files for a directory (or a filesystem). In the example below we query the **mount point /srv/pro33** .

```
root@server2:~# lsof /srv/pro33/
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
more    24324 paul    3r   REG   8,49    18774   13 /srv/pro33/services
root@server2:~#
```

When you query **lsof** for a directory that contains **mount points** further in the tree, then those are not followed. See the example below.

```
root@server2:~# lsof /srv/pro33/
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
more    24324 paul    3r   REG   8,49    18774   13 /srv/pro33/services
root@server2:~# lsof /srv/
root@server2:~#
```

You can also give a device as argument to **lsof**. Note that we need to specify a partition (with a filesystem), listing just the device **/dev/sdd** yields no results.

```
root@server2:~# lsof /dev/sdd1
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
more    24324 paul    3r   REG   8,49    18774   13 /srv/pro33/services
root@server2:~#
```

47.3.2 fuser

The **fuser** tool is part of the **psmisc** package, install it with **apt-get install psmisc** . With **fuser** you can find the processes that are using a file (or directory). Here is a simple example.

```
root@server2:~# fuser .
/root:                450c
root@server2:~# echo $$
450
root@server2:~#
```

You can use the **-u** option to immediately get the username associated with the process, as this screenshot shows.

```
root@server2:~# fuser -u /srv/pro33/
/srv/pro33:          24131c(paul) 24142c(paul)
root@server2:~#
```

Using the **-k** option will (attempt to) kill the processes it finds. Running as root this is a quick and dirty way to throw everyone off of a filesystem.

```
root@server2:~# fuser -uk /srv/pro33/
/srv/pro33:          24131c(paul) 24142c(paul)
Could not kill process 24142: No such process
root@server2:~#
```

47.4 File Systems

We already covered **df -h** and **tune2fs** to monitor the file system. For a **ZFS** file system there is the **zfs** command (see the ZFS chapter).

47.4.1 df -i

We have seen the **df** tool before, here we use it to look at the number of free **inodes**, because if you run out of **inodes**, then you cannot create files anymore.

```
root@server2:~# df -i /dev/sdd1
Filesystem     Inodes IUsed  IFree IUse% Mounted on
/dev/sdd1      655360    13 655347    1% /srv/pro33
root@server2:~#
```

47.4.2 pydf

The **pydf** tool is installed with **apt-get install pydf** and works much the same as **df**, but with a nicer output.

```
root@server2:~# pydf
Filesystem Size  Used Avail Use%                      Mounted on
/dev/sda1   15G 1186M   13G  7.9 [#.................] /
/dev/sdd1   10G   37M 9450M  0.4 [..................] /srv/pro33
root@server2:~#
```

The **pydf** tool will default to multiples of kibibytes. Using the **-H** option will use multiples of kilobytes.

```
root@server2:~# pydf -H
Filesystem Size  Used Avail Use%                      Mounted on
/dev/sda1   16G 1244M   14G  7.9 [#.................] /
/dev/sdd1   11G   39M   10G  0.4 [..................] /srv/pro33
root@server2:~#
```

47.5 Volume Manager

The **LVM volume manager** can be monitored with the **pv* vg* lv*** commands (see the LVM chapter) and the **mdadm** RAID configurations can be monitored with **mdadm** and **dmsetup** and also with **cat /proc/mdstat**, see the RAID chapter for details.

47.6 Block Device Interrupt

47.6.1 pidstat

The **pidstat** tool can be installed with **apt-get install sysstat**. This tool will report statitics for Linux tasks. The **-d** option gives I/O statitics, the example below uses two second intervals.

```
root@MBDebian~# pidstat -d 2
Linux 4.19.0-6-amd64 (MBDebian)         10/14/2019      x86_64        (8 CPU)

02:37:00 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
02:37:02 PM     0        67      0.00      0.00      0.00       1  kworker/3:1-mm_percpu_wq

02:37:02 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

02:37:04 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command

02:37:06 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
02:37:08 PM     0       264      0.00      2.00      0.00       2  jbd2/sda4-8
02:37:08 PM  1000      2816      0.50     24.50      0.00       0  VirtualBoxVM
q^C

Average:      UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
Average:        0        67      0.00      0.00      0.00       0  kworker/3:1-mm_percpu_wq
Average:        0       264      0.00      0.50      0.00       0  jbd2/sda4-8
Average:     1000      2816      0.12      6.12      0.00       0  VirtualBoxVM
root@MBDebian~# pidstat -d 2
```

47.7 Device Driver

47.7.1 dmesg

For device driver information related to storage you can use the **grep** command on the **dmesg** output. In this example we **grep** for SCSI and can see that the **sd** driver is used for /dev/sda and /dev/sdb.

```
root@MBDebian~# dmesg | grep SCSI
[    1.151005] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 247)
[    1.502503] SCSI subsystem initialized
[    1.858803] sd 0:0:0:0: [sda] Attached SCSI disk
[    2.900073] sd 1:0:0:0: [sdb] Attached SCSI removable disk
root@MBDebian~#
```

The device driver for a storage device can also be found using its major number. In this screenshot the major umber for /dev/sda is **8**. Which corresponds to the **sd** driver according to **/proc/devices**.

```
root@MBDebian~# ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 Oct 14 10:00 /dev/sda
root@MBDebian~# grep 8 /proc/devices
128 ptm
180 usb
```

```
189 usb_device
248 watchdog
  8 sd
 68 sd
128 sd
root@MBDebian~#
```

47.8 Firmware - Hardware

47.8.1 iostat

The **iostat** tool is part of the **sysstat** package, so you will need to do an **apt-get install sysstat** to obtain this tool. It will read information from **/proc/diskstat** and format it nicely with colours.

Just typing **iostat** will give a brief summary of disk usage for all disks (or all block devices), as seen in this screenshot. Use **iostat -x** for more statistics.

```
root@server2:~# iostat
Linux 4.19.0-5-amd64 (server2)  08/27/2019      x86_64        (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.39    0.00    0.90    0.06    0.00   98.65

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.89        17.55         5.68     228339      73936
sdb               0.02         0.64         0.00       8354          0
sdc               0.01         0.33         0.00       4248          0
sdd               0.12         1.99        23.35      25849     303680

root@server2:~#
```

For real time statistics, for example every 3 seconds, use the **-d 3** option. We also add the **-h** option for human readable sizes. And you can specify a block device to follow. You can also use **-p sda** for all partitions on **sda**.

```
root@server2:~# iostat -h -d 3 sda1 | grep sda1
    15.64       235.0k       261.4k       2.1G       2.4G sda1
   345.67         1.9M         0.0k       5.6M       0.0k sda1
   333.67         1.6M         0.0k       4.7M       0.0k sda1
    69.67       294.7k         0.0k     884.0k       0.0k sda1
   166.00       664.0k         0.0k       1.9M       0.0k sda1
   171.33       685.3k         0.0k       2.0M       0.0k sda1
   380.00         7.5M         0.0k      22.5M       0.0k sda1
   382.00         3.0M         6.7k       9.0M      20.0k sda1
   395.33         5.6M         0.0k      16.9M       0.0k sda1
^C
root@server2:~#
```

47.8.2 iotop

The **iotop** tool can be installed with **apt-get install iotop**. It works very much like **top** but organises processes by disk usage. The **jbd2** process at the top is the **ext4 journaling block device** which sits between the **ext4** filesystem and the applications.

```
root@server2:~# iotop

Total DISK READ:         0.00 B/s | Total DISK WRITE:        15.39 K/s
Current DISK READ:       0.00 B/s | Current DISK WRITE:      26.93 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
```

```
  193 be/3 root        0.00 B/s   15.39 K/s  0.00 %  1.53 % [jbd2/sda1-8]
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_gp]
    4 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_par_gp]
    6 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kworker/0:0H-kblockd]
    8 be/0 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [mm_percpu_wq]
    9 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
   10 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_sched]
   11 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [rcu_bh]
  keys:  any: refresh  q: quit  i: ionice  o: active  p: procs  a: accum
  sort:  r: asc  left: SWAPIN  right: COMMAND  home: TID  end: COMMAND
```

You can show only processes that are doing disk I/O by using **iostat -oPa** . The **find** command in the screenshots copies every file on the system.

```
root@server2:~# iotop -oPa

Total DISK READ:       103.83 K/s | Total DISK WRITE:         0.00 B/s
Current DISK READ:     280.73 K/s | Current DISK WRITE:    1942.03 K/s
  PID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 3416 be/4 root        548.00 K      0.00 B  0.00 %  1.10 % find / -exec~/srv/pro33/tmp ;
 2287 be/3 root          0.00 B      4.00 K  0.00 %  0.41 % [jbd2/sdd1-8]




  keys:  any: refresh  q: quit  i: ionice  o: all  p: threads  a: bandwidth
  sort:  r: asc  left: SWAPIN  right: COMMAND  home: PID  end: COMMAND
```

47.8.3 ionice

The **ionice** tool is similar to **nice** but regulates block device I/O priorities. A common use of **ionice** is with the **-c2 -n7** arguments for **class 2** and **be the nicest** to other I/O tasks.

In the screenshot below we execute a **find** command with **ionice**.

```
root@server2:~# ionice -c2 -n7 find /srv/pro42 -exec cp {} /srv/backup \; &
root@server2:~#
```

47.8.4 vmstat

The **vmstat** gives virtual memory statistics, and includes by default I/O for block devices as **bi** (blocks in) and **bo** (blocks out). In the screenshot we ask for statistics every two seconds, 100 times.

```
root@server2:~# vmstat 2 100
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 839572  11792  91524    0    0   140    71  285   94  0  0 99  0  0
 1  0      0 834856  11944  96352    0    0  1922  2288  920 2092 17 30  0 53  0
 3  0      0 830368  11992 100720    0    0  1652  2004  935 2020 15 30  0 55  0
 1  0      0 825628  12504 104984    0    0  1810  1994  971 2251 18 28  0 54  0
 3  0      0 822464  12744 108140    0    0  1144  1690  874 1990 16 37  0 47  0
 1  0      0 819392  13612 110132    0    0   836  1810  755 1913 23 50  0 27  0
 4  0      0 816196  13672 113216    0    0  1024  1608  830 1871 17 32  0 51  0
^C
root@server2:~#
```

You can use the **-d** option to display statistics for each disk (each block device).

```
root@server2:~# vmstat -d 2 3
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors      ms  total merged sectors      ms    cur    sec
sda    34731     11 1811494   43006  31145   1965 2248696    5674      0     22
sdb      272      0   16708      96      0      0       0       0      0      0
sdc      166      0    8496      34      0      0       0       0      0      0
sdd      390      0   25136     242      0      0       0       0      0      0
sda    35364     14 1818422   43893  31893   1972 2264448    5759      0     23
sdb      272      0   16708      96      0      0       0       0      0      0
sdc      166      0    8496      34      0      0       0       0      0      0
sdd      390      0   25136     242      0      0       0       0      0      0
sda    35955     14 1824150   44874  32726   1972 2273640    5823      0     24
sdb      272      0   16708      96      0      0       0       0      0      0
sdc      166      0    8496      34      0      0       0       0      0      0
sdd      390      0   25136     242      0      0       0       0      0      0
root@server2:~#
```

47.8.5 S.M.A.R.T. tools

You can install **hddtemp** with **apt-get install hddtemp**. The **hddtemp** tool supports many disks and can read the temperature of the disk using S.M.A.R.T. technology. The screenshot is from an Apple Macbookpro running Debian 10 Linux.

```
root@MBDebian~# hddtemp /dev/sda
/dev/sda: APPLE SSD SM0256G: 31C
root@MBDebian~#
```

Our server runs in **QEMU** so the temperature is not available, run this tool directly on the hardware. We will discuss QEMU and virtualisation later in this book.

```
root@server2~# hddtemp /dev/sda
/dev/sda: QEMU QEMU HARDDISK: S.M.A.R.T. not available
root@server2:~#
```

On a real server you can also install **hwinfo** with **apt-get install hwinfo** and **smartctl** with **apt-get install smartmontools**. These tools will give you a lot of hardware information about your disks.

47.9 Cheat sheet

Table 47.1: Monitoring and troubleshooting storage

command	explanation
strace	Trace system function calls for a command.
perf stat foo	Gather performance statistics on command **foo**.
lsof /foo	List open files on the **/foo** mount point.
lsof /dev/bar1	List open files on the **/dev/bar1** partition.
fuser foo	List processes that use the **foo** directory.
fuser -u foo	List processes and usernames that use the **foo** directory.
fuser -uk foo	List and kill processes that use the **foo** directory.
df -i	List inode usage for mounted file systems.
pydf	List usage of mounted file systems.
pidstat -d 2	Display statistics on disk I/O.
dmesg | grep SCSI	List kernel messages about SCSI devices.
/proc/devices	Contains major numbers and their driver.
iostat	List a summary of disk usage.
iotop	Display processes by disk I/O.
ionice	Execute a command with low disk I/O priority.
vmstat	Display virtual memory, swap and disk I/O statistics.

47.10 Practice

1. Start a disk-intensive command in background and then monitor the disk(s) with **iostat** every five seconds.
2. Start a disk intensive command in background and monitor the processes that do most of the I/O with **iotop**.
3. Start a read-intensive task in background and monitor it with **vmstat**.
4. Mount a filesystem on /srv/pro33. Copy a file (for example /etc/services) to /srv/pro33. Open the file (for example with **more /srv/pro33/services**). Open a second terminal and list all processes that have files open on **/srv/pro33** . You should find one or two processes, list the command of those two processes.
5. Kill the above processes with **fuser**.
6. (In the other terminal) open the file again with **more**. Then use **lsof** to find who has the file open and with which command.
7. Verify the amount of free **inodes** on /dev/sdd1.
8. Install **pydf** and list all mounted partitions with it.
9. If you have a real (non virtual) server, then install **hddtemp**, **hwinfo** and **smartctl** and discover your hardware with these tools.

47.11 Solution

1. Start a disk-intensive command in background and then monitor the disk(s) with **iostat** every five seconds.

```
mount /dev/sdd1 /srv/pro33
find /usr/ -exec cp {} /srv/pro33/tmp \; >/dev/null 2>&1 &
iostat -d 5 /dev/sdd1
```

2. Start a disk intensive command in background and monitor the processes that do most of the I/O with **iotop**.

```
find /usr/ -exec cp {} /srv/pro33/tmp \; >/dev/null 2>&1 &
iotop -oa
```

3. Start a read-intensive task in background and monitor it with **vmstat**.

```
find /usr/ -exec cat {} \; >/dev/null 2>&1 &
vmstat 2 100    # You should see a lot of bi = read
```

4. Mount a filesystem on /srv/pro33. Copy a file (for example /etc/services) to /srv/pro33. Open the file (for example with **more /srv/pro33/services**). Open a second terminal and list all processes that have files open on **/srv/pro33** . You should find one or two processes, list the command of those two processes.

```
fuser -u /srv/pro33/
ps f 2779 2818     ## replace with the processes you received from fuser.
```

5. Kill the above processes with **fuser**.

```
fuser -uk /src/pro33/
```

6. (In the other terminal) open the file again with **more**. Then use **lsof** to find who has the file open and with which command.

```
lsof /srv/pro33/
```

7. Verify the amount of free **inodes** on /dev/sdd1.

```
df -i /dev/sdd1
```

8. Install **pydf** and list all mounted partitions with it.

```
apt-get install pydf
pydf
```

9. If you have a real (non virtual) server, then install **hddtemp**, **hwinfo** and **smartctl** and discover your hardware with these tools.

```
apt-get install hddtemp hwinfo smartmontools
```

Chapter 48

Network mounts

todo

Chapter 49

UUID's

49.1 Problem with /dev/sd*

Up until now we have been using **/dev/sd*** to refer to disks and partitions. There is problem with this, namely that those names are not persistent. What is **/dev/sdc** today may be **/dev/sdb** tomorrow (by removing the old /dev/sdb for example).

Luckily there is a solution for this because every filesystem has a unique identifier.

49.2 What is a UUID?

A **UUID** is a Universally Unique Identifier. In other words it is a very big unique number. There are many tools to create a **UUID**, on Debian Linux we have **dbus-uuidgen**.

```
root@server2:~# dbus-uuidgen
afac8abf786bfaa2fe2e17d25d668787
root@server2:~# dbus-uuidgen
81f63c400031e7ea4f07b6075d668789
root@server2:~# dbus-uuidgen
6a00dc58cba70ea97e6ccd185d66878f
root@server2:~#
```

49.3 tune2fs

Every filesystem that is created in Debian Linux will get a UUID. You can find the UUID using **tune2fs** as shown in this screenshot.

```
root@server2:~# tune2fs -l /dev/sdd1 | grep UUID
Filesystem UUID:          9d27f1b0-65f5-44ed-a3c4-2efbffbf6d94
root@server2:~#
```

49.4 blkid and lsblk

You can also use **blkid** or **lsblk** to find the UUID. Note that the UUID for the partition is different from the UUID for the filesystem.

```
root@server2:~# blkid | grep sdd1
/dev/sdd1: UUID="9d27f1b0-65f5-44ed-a3c4-2efbffbf6d94" TYPE="ext4" PARTUUID="19637ce0-f08d ↩
    -6144-90af-d981c6b69c73"
root@server2:~# lsblk -f | grep sdd1
|-sdd1 ext4         9d27f1b0-65f5-44ed-a3c4-2efbffbf6d94    9.2G     0% /srv/pro33
root@server2:~#
```

49.5 /etc/fstab

You may have noticed that there are already UUIDs in the **/etc/fstab** file. This is to make sure that the correct filesystem is mounted at the correct mount point.

```
root@server2:~# tail -7 /etc/fstab
# / was on /dev/sda1 during installation
UUID=9ea6c2b3-1ad2-4209-ba06-19af4027b6f8 /               ext4    errors=remount-ro 0  ↩
          1
# swap was on /dev/sda5 during installation
```

```
UUID=89854b55-bfab-4c61-8bbb-aaa40c7154aa none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/sdb1 /srv/pro42 ext4 defaults 0 0
/dev/sdd1 /srv/pro33 auto acl 0 0
root@server2:~#
```

The line that starts with **/dev/sdd1** can be improved by using the filesystem's UUID instead of **/dev/sdd1**. We found the UUID in the screenshots above, so the last line in **/etc/fstab** becomes this.

```
root@server2:~# tail -7 /etc/fstab
# / was on /dev/sda1 during installation
UUID=9ea6c2b3-1ad2-4209-ba06-19af4027b6f8 /               ext4    errors=remount-ro 0  ↩
          1
# swap was on /dev/sda5 during installation
UUID=89854b55-bfab-4c61-8bbb-aaa40c7154aa none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/sdb1 /srv/pro42 ext4 defaults 0 0
UUID="9d27f1b0-65f5-44ed-a3c4-2efbffbf6d94" /srv/pro33 auto acl 0 0
root@server2:~#
```

Tip
You can get the UUID in vi by typing **Esc :r !blkid /dev/sdd1** .

Note that you can still use the device name with the **mount** command, even though the device is not mentioned in **/etc/fstab**. The **mount** tool will search for this device. It may be safer to use the mount point.

```
root@server2:/boot/grub# umount /dev/sdd1
root@server2:/boot/grub# mount /dev/sdd1
root@server2:/boot/grub# umount /dev/sdd1
root@server2:/boot/grub# mount /srv/pro33/
root@server2:/boot/grub#
```

49.6 grub2

We will see later in this book that Debian Linux boots from a UUID. So even if **/dev/sda** changes, the system will still boot properly.

49.7 Cheat sheet

Table 49.1: UUID's

command	**explanation**
dbus-uuidgen	Create a UUID.
tune2fs -l /dev/foo1	List UUID (and other parameters) of the partition **/dev/foo1** partition.
blkid	Lists partition UUID (and other information).
lsblk -f	Lists file system UUID's.
/etc/fstab	Can contain UUID's as identifier to mount a file system.
grub2	Tool to boot Debian Linux which identifies the root file system by UUID.

49.8 Practice

1. Use **tune2fs** and **blkid** to find the UUID of a filesystem. Verify that both UUIDs are identical.
2. Add two of your filesystems with their UUID to **/etc/fstab** . Make sure to remove the old entry with **/dev/sd**.
3. Test with the mount command that your entries in **/etc/fstab** work properly.

49.9 Solution

1. Use **tune2fs** and **blkid** to find the UUID of a filesystem. Verify that both UUIDs are identical.

```
tune2fs -l /dev/sdc1
blkid | grep sdc1
```

2. Add two of your filesystems with their UUID to **/etc/fstab** . Make sure to remove the old entry with **/dev/sd**.

```
vi /etc/fstab
```

3. Test with the mount command that your entries in **/etc/fstab** work properly.

```
umount /srv/pro33
mount /srv/pro33
umount /srv/pro42
mount /srv/pro42
```

Chapter 50

RAID storage

50.1 RAID 1 and 5

RAID is short for **Redundant Array of Inexpensive Disks**, more recently changed to **Redundant Array of Independent Disks**. RAID is a system of combining several disks to have larger (virtual) disks and/or redundant storage.

RAID 1 is very simple, you take one disk and have a second disk as an identical copy. This is also called **mirroring** a disk. The end result is that **one disk may fail** but there is no data loss. The disadvantage is that you have to buy double the disk space that you need. RAID 1 can also be done with more mirror disks.

We will create a RAID 1 setup in this chapter.

RAID 5 is a bit more complex. In essence it requires at least three disk, and one of the disks can fail without data loss. This is done by spreading parity information across all disks. RAID 5 can also be done with more disks, you still *lose* the space of one disk.

We will create a RAID 5 setup with three disks in this chapter.

There are more RAID levels available, for a complete list you can take a look at Wikipedia here https://en.wikipedia.org/wiki/-Standard_RAID_levels .

50.2 hardware or software

RAID levels can be achieved through hardware or software. The verdict used to be "**hardware is best**" because of the speed, and because of the software overhead for the operating system. But lately software is preferred because of the flexibility of managing the disks with software (and the independence from sometimes expensive disks for hardware controllers).

In this chapter we only use software RAID levels.

50.3 mdadm

After **apt-get install mdadm**, we are ready to create and use **multiple devices**. We will start with creating a RAID 5 device spanning three (equally sized) disks. If you remove **mdadm** then the **md** devices will be gone at the next reboot.

50.4 Creating a RAID 5

In this section we will go through the steps of creating and using a **/dev/md0** device that is a software RAID 5 array. We will use the disks sdb, sdc and sdd for this, so unmount all filesystems and remove all partitions from these devices

50.4.1 Wipe existing drives

These steps here show how to remove all partition information from **/dev/sdb**. Repeat these steps for **/dev/sdc** and **/dev/sdd**.

```
fdisk /dev/sdb
o
w
dd if=/dev/zero of=/dev/sdb count=1 bs=512
```

50.4.2 Create the RAID 5 array

The next command is **mdadm --create** to create a RAID 5 with the three existing devices. This will create the device and make sure it is activated when the system boots.

```
root@server2:~# mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdb /dev/sdc /dev/ ↩
    sdd
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
root@server2:~#
```

Note
We created the array with sdb, sdc and sdd devices. It is possible to create a primary partition on each device, spanning the whole device and marking this partition as type **fd**. These **fd** partitions can then be used instead of the parent devices.

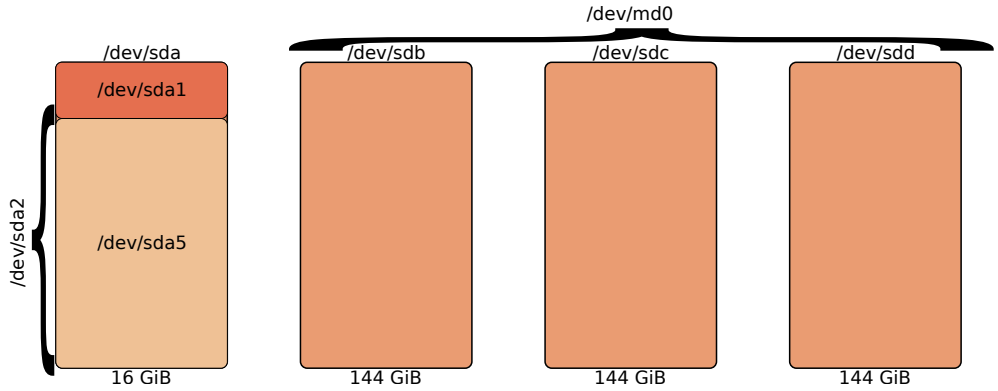
50.4.3 Verify in /proc/mdstat

The device is now visible in **/proc/mdstat**. It is synchronising the third disk to the other two, this can take some time. You can watch it sync with **watch cat /proc/mdstat** .

```
root@server2:~# cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdd[3] sdc[1] sdb[0]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2] [UU_]
      [=>...................]  recovery =  6.1% (9248188/150862848) finish=11.7min speed ↩
          =200983K/sec
      bitmap: 0/2 pages [0KB], 65536KB chunk

unused devices: <none>
root@server2:~#
```

50.4.4 Update the drawing of our four disks

The drawing of the storage on our server can now be updated to include **/dev/md0**.



50.4.5 For fdisk this is a normal block device

We can check the new **/dev/md0** device with **fdisk**. As expected the size is 288GiB, we *lose* the disk space of one device. You could decide to partition this device and use these partitions like any other partitions.

```
root@server2:~# fdisk -l /dev/md0
Disk /dev/md0: 287.8 GiB, 308967112704 bytes, 603451392 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 524288 bytes / 1048576 bytes
root@server2:~#
```

50.4.6 Put a filesystem on it

Next we create an **ext4** filesystem on the **/dev/md0** device. There is no need to create a partition (there are also no partitions on the individual devices).

```
root@server2:~# mkfs.ext4 /dev/md0
mke2fs 1.44.5 (15-Dec-2018)
Creating filesystem with 75431424 4k blocks and 18857984 inodes
Filesystem UUID: f2dcd28e-4f05-48d7-acee-54bca2cd191a
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872, 71663616

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done

root@server2:~#
```

50.4.7 Mount the filesystem

Time to **mount** the filesystem on our favorite **/srv/pro33** mount point. We can now use the ext4 filesystem on a software RAID 5 device.

```
root@server2:~# mount /dev/md0 /srv/pro33
root@server2:~# df -h /srv/pro33/
Filesystem      Size  Used Avail Use% Mounted on
/dev/md0        283G   65M  268G   1% /srv/pro33
root@server2:~#
```

50.4.8 Add the device to /etc/fstab

And you can add it to **/etc/fstab** for a permanent mount on this computer. The screenshot uses the UUID method to link the filesystem to the mount point.

```
root@server2:~# tail -1 /etc/fstab
UUID="f2dcd28e-4f05-48d7-acee-54bca2cd191a" TYPE="ext4" /srv/pro33 auto defaults 0 0
root@server2:~#
```

Warning
Do not create doubles in /**etc**/**fstab** ; remove the existing /**srv**/**pro33** line!

50.4.9 Persistent names across reboots

There is just one little issue when you do a **reboot** now. The **md** device will be automatically activated at boot time, but with a different name. It will be visible as **/dev/md127**. You don't have to care about this (since the mount will be correct), but if you do, then follow the instructions in this screenshot.

```
root@server2:~# mdadm --detail --scan
ARRAY /dev/md0 metadata=1.2 spares=1 name=server2:0 UUID=4c467627:b9e4b5c2:1ffb5876: ↩
    c6dea03b
root@server2:~# vi /etc/mdadm/mdadm.conf
```

```
root@server2:~# tail -1 /etc/mdadm/mdadm.conf
ARRAY /dev/md0 UUID=4c467627:b9e4b5c2:1ffb5876:c6dea03b
root@server2:~# update-initramfs -u
update-initramfs: Generating /boot/initrd.img-4.19.0-5-amd64
root@server2:~#
```

We will discuss **initramfs** in the Booting Linux chapter.

50.4.10 Sync should be ready by now

After a while the three disks should be in sync and marked as **active** in **/proc/mdstat**. Always verify this file when troubleshooting **md** devices.

```
root@server2:~# cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdd[3] sdc[1] sdb[0]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]
      bitmap: 0/2 pages [0KB], 65536KB chunk

unused devices: $060;none>
root@server2:~
```

50.4.11 Deleting the md device

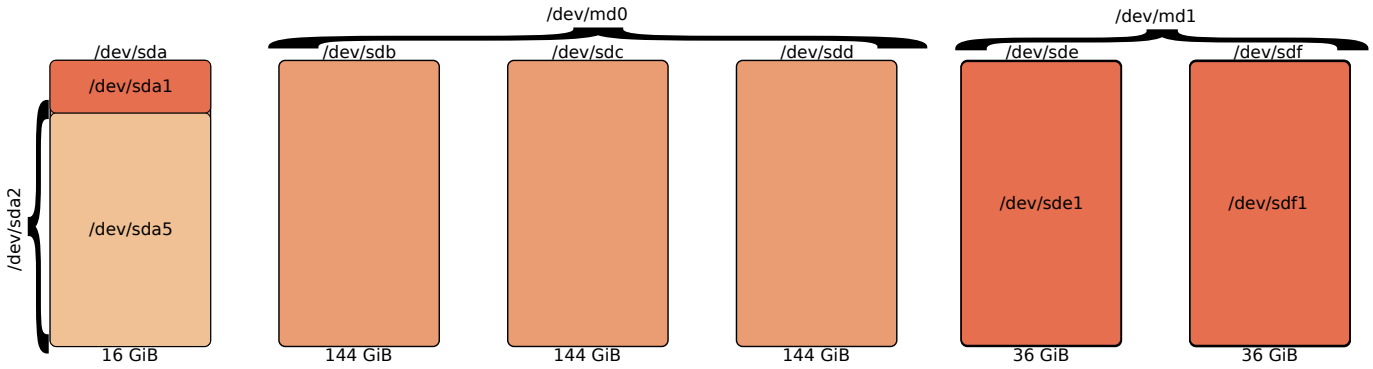
To delete the **/dev/md0** device you first have to unmount it. The **--stop** option will stop the array, but it will reappear after a reboot because **mdadm** will recognise its signature on the disks (or partitions).

```
root@server2:~# umount /dev/md0
root@server2:~# mdadm --stop /dev/md0
mdadm: stopped /dev/md0
root@server2:~#
```

So remove the entry from **/etc/fstab** and from **/etc/mdadm/mdadm.conf** and update the **initramfs**. And then either wipe the disks, or remove the **mdadm** package.

50.5 Creating a RAID 1 mirror

We will create a two disk mirror device named **/dev/md1**. For this exercise we added two 36GB hard disks to our Debian Linux **server2**. The names of the hard disk are **/dev/sde** and **/dev/sdf**.



We start by creating primary partitions on a standard msdos label. This step is not mandatory as the raw devices will also do.

```
root@server2:~# parted /dev/sde mklabel msdos mkpart primary 0 100%
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server2:~# parted /dev/sdf mklabel msdos mkpart primary 0 100%
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server2:~#
```

Then we use **mdadm** to create the **/dev/md1** device. See the **Note** in the output, booting from a mirror requires some extra steps. We will discuss booting from a mirror in the Booting Linux chapter.

```
root@server2:~# mdadm --create /dev/md1 --level=1 --raid-devices=2 /dev/sde1 /dev/sdf1
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store /boot on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use
    --metadata=0.90
Continue creating array? Y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md1 started.
root@server2:~#
```

The **/dev/md1** device is now visible in **/proc/mdstat** and resyncing.

```
root@server2:~# head -5 /proc/mdstat
Personalities : [raid6] [raid5] [raid4] [linear] [multipath] [raid0] [raid1] [raid10]
md1 : active raid1 sdf1[1] sde1[0]
      37714880 blocks super 1.2 [2/2] [UU]
      [===>.................]  resync = 19.6% (7403776/37714880) finish=2.5min speed=200056 ↩
          K/sec
root@server2:~#
```

The rest of the story is identical to the RAID 5 device. The new **/dev/md1** device can be partitioned if you want to, or can be used as a whole. There is no practical difference with *real* hard disk devices or partitions.

50.6 Recovery

The goal of RAID 1 and RAID 5 is to be able to fully recover from a disk crash. Suppose we have this working situation (see the screenshot below). There is a RAID 5 named **md0** consisting of sdb, sdc and sdd; and there is a RAID 1 named **md1** consisting of sde and sdf.

```
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [raid6] [raid5] [raid4] [linear] [multipath] [raid0] [raid10]
md0 : active raid5 sdd[3] sdc[1] sdb[0]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]
      bitmap: 0/2 pages [0KB], 65536KB chunk

md1 : active (auto-read-only) raid1 sde1[0] sdf1[1]
      37714880 blocks super 1.2 [2/2] [UU]

unused devices: <none>
root@server2:~#
```

Suppose /dev/sdc crashes and we reboot the machine and insert a new disk. This new disk becomes **/dev/sdg** (if we leave /dev/sdc in the computer, other wise /dev/sdd to sdf shift up one letter). So we need to add **sdg** to the **md0** device, and then it syncs the new disk.

The **/proc/mdstat** file will show U_U to indicate the **/dev/sdc** device is no longer part of the array.

```
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active (auto-read-only) raid5 sdb[0] sdd[3]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2] [U_U]
      bitmap: 0/2 pages [0KB], 65536KB chunk

md1 : active (auto-read-only) raid1 sde1[0] sdf1[1]
      37714880 blocks super 1.2 [2/2] [UU]

unused devices: <none>
root@server2:~#
```

Note
Without a reboot you can achieve the same result by executing a **mdadm --fail /dev/md0 /dev/sdc** followed by **mdadm --remove /dev/md0 /dev/sdc**.

You can use the **sfdisk** tool to copy the partition table to the new disk.

```
root@server2:~# sfdisk -d /dev/sdb | sfdisk /dev/sdg
sfdisk: /dev/sdb: does not contain a recognized partition table
Checking that no-one is using this disk right now ... OK

Disk /dev/sdg: 144 GiB, 154618822656 bytes, 301989888 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

>>> Done.

New situation:
root@server2:~#
```

Then we can add the new device to the RAID 5 array with **mdadm -a**. The syncing of the new device starts automatically.

```
root@server2:~# mdadm /dev/md0 -a /dev/sdg
mdadm: added /dev/sdg
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdg[4] sdb[0] sdd[3]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2] [U_U]
      [>....................]  recovery =  0.5% (775036/150862848) finish=16.1min speed ↩
          =155007K/sec
      bitmap: 0/2 pages [0KB], 65536KB chunk

md1 : active (auto-read-only) raid1 sde1[0] sdf1[1]
      37714880 blocks super 1.2 [2/2] [UU]

unused devices: <none>
root@server2:~#
```

50.7 Growing an md device

Let's say we have a two-way mirror and want a three-way mirror. So three disks with identical data. We start with our situation from above.

```
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdg[4] sdb[0] sdd[3]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]
      bitmap: 0/2 pages [0KB], 65536KB chunk

md1 : active raid1 sde1[0] sdf1[1]
      37714880 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

We copy the partition table from **/dev/sdf** to **/dev/sdc** (let's say by magic that the faulty part is beyond the 36GiB mark). This copy will create an identical 36GiB partition on /dev/sdc.

```
root@server2:~# sfdisk -d /dev/sdf | sfdisk /dev/sdc
Checking that no-one is using this disk right now ... OK

Disk /dev/sdc: 144 GiB, 154618822656 bytes, 301989888 sectors
<output truncated>
```

Then we add the **/dev/sdc1** device to the **/dev/md1** array. It will by default be added as a spare. The next command tells **mdadm** that it should be a three-way mirror.

```
root@server2:~# mdadm -a /dev/md1 /dev/sdc1
mdadm: added /dev/sdc1
root@server2:~# mdadm --grow --raid-devices=3 /dev/md1
raid_disks for /dev/md1 set to 3
```

And **mdadm** automatically begins syncing the new device, this can take a while but the device can be used while syncing.

```
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [linear] [multipath] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdg[4] sdb[0] sdd[3]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]
      bitmap: 0/2 pages [0KB], 65536KB chunk

md1 : active raid1 sdc1[2] sde1[0] sdf1[1]
      37714880 blocks super 1.2 [3/2] [UU_]
      [>....................]  recovery =  1.5% (601856/37714880) finish=2.0min speed ↩
          =300928K/sec

unused devices: <none>
root@server2:~#
```

50.8 Growing from RAID 1 to RAID 5

We can *grow* our three-way mirror to a RAID 5. The procedure starts by *growing* our three-way mirror to a two-way mirror. First we remove **/dev/sdc1** from our md1 device.

```
root@server2:~# mdadm --fail /dev/md1 /dev/sdc1
mdadm: set /dev/sdc1 faulty in /dev/md1
root@server2:~# mdadm --remove /dev/md1 /dev/sdc1
mdadm: hot removed /dev/sdc1 from /dev/md1
root@server2:~#
```

Then we set our md device to a two-way mirror. If we add our **/dev/sdc1** now, then it would just start syncing again to a three-way mirror.

```
root@server2:~# mdadm --grow /dev/md1 --level=1 --raid-devices=2
raid_disks for /dev/md1 set to 2
root@server2:~#
```

Now we can add the **/dev/sdc1** device as a spare and *grow* to a RAID 5 with three devices. The syncing starts immediately.

```
root@server2:~# mdadm --add /dev/md1 /dev/sdc1
mdadm: added /dev/sdc1
root@server2:~# mdadm --grow /dev/md1 --level=5 --raid-devices=3
mdadm: level of /dev/md1 changed to raid5
root@server2:~# cat /proc/mdstat
Personalities : [raid1] [raid6] [raid5] [raid4] [linear] [multipath] [raid0] [raid10]
md1 : active raid5 sdc1[2] sdf1[1] sde1[0]
      37714880 blocks super 1.2 level 5, 64k chunk, algorithm 2 [3/3] [UUU]
      [>....................]  reshape =  0.7% (301720/37714880) finish=8.2min speed=75430K ↩
          /sec

md0 : active (auto-read-only) raid5 sdg[4] sdb[0] sdd[3]
      301725696 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3] [UUU]
      bitmap: 0/2 pages [0KB], 65536KB chunk

unused devices: <none>
root@server2:~#
```

50.9 Cheat sheet

Table 50.1: RAID storage

command	explanation
mdadm -C /dev/md0 -l=5 -n=3 /dev/sdb /dev/sdc /dev/sdd	Create a software RAID 5.
mdadm -C /dev/md1 -l=1 -n=2 /dev/sde1 /dev/sdf1	Create a software RAID 1 (using partitions).
cat /proc/mdstat	Verify the status of your RAID devices.
fdisk /dev/md0	(Optionally) manage the RAID 5 device as a normal block device.
mkfs.ext4 /dev/md0	Put a file system directly on the RAID 5 device.
mount /dev/md0 /foo	Mount the file system on the RAID 5 device.
/etc/mdadm/mdadm.conf	Can contain persistent names for RAID devices.
mdadm --stop /dev/md0	Stop the RAID device (after unmounting it).
mdadm --fail /dev/md0 /dev/sdc	Mark /dev/sdc as faulty.
mdadm --remove /dev/md0 /dev/sdc	Remove /dev/sdc from the RAID device.
mdadm /dev/md0 --add /dev/sdg	Add /dev/sdg to the RAID device.
mdadm --grow --raid-devices=3 /dev/md1	Grow a RAID device with the last added block device.

50.10 Practice

1. Choose three disk devices and create one primary partition, spanning the whole disk, on each of them.
2. Mark each primary partition as type **fd**.
3. Create a RAID 5 device using the three primary partitions.
4. Verify your work with **lsblk**.
5. Make sure the **md0** name survives a reboot.
6. Test the reboot and verify with **lsblk**.
7. Remove all partition info from the three disks and destroy your md0 array. Don't forget **/etc/fstab**, **mdadm.conf** and **update-initramfs**.
8. Create a two-way mirror, mount it under **/srv/pro42**.
9. Convert the two-way mirror to a three disk RAID 5. Then grow the ext4 filesystem.

50.11 Solution

1. Choose three disk devices and create one primary partition, spanning the whole disk, on each of them.

```
fdisk /dev/sdb o n p w
fdisk /dev/sdc o n p w
fdisk /dev/sdd o n p w
```

2. Mark each primary partition as type **fd**.

```
fdisk /dev/sdb l t fd w
fdisk /dev/sdc t fd w
fdisk /dev/sdd t fd w
```

3. Create a RAID 5 device using the three primary partitions.

```
root@server2:~# mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdb1 /dev/sdc1  ↩
    /dev/s
dd1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
root@server2:~#
```

4. Verify your work with **lsblk**.

```
lsblk
```

5. Make sure the **md0** name survives a reboot.

```
root@server2:~# mdadm --detail --scan
ARRAY /dev/md0 metadata=1.2 spares=1 name=server2:0 UUID=8975585e:12fc7627:195c5d2b ↩
    :49088394
root@server2:~# vi /etc/mdadm/mdadm.conf
root@server2:~# grep md0 /etc/mdadm/mdadm.conf
ARRAY /dev/md0 UUID=8975585e:12fc7627:195c5d2b:49088394
root@server2:~# update-initramfs -u
update-initramfs: Generating /boot/initrd.img-4.19.0-5-amd64
root@server2:~#
```

6. Test the reboot and verify with **lsblk**.

```
reboot
lsblk
```

7. Remove all partition info from the three disks and destroy your md0 array. Don't forget **/etc/fstab**, **mdadm.conf** and **update-initramfs**.

```
mdadm --stop /dev/md0
dd if=/dev/zero of=/dev/sdb bs=512 count=100  ## repeat for sdc,sdd
vi /etc/fstab  ## remove all /dev/md and/or /dev/sd[bcd] entries
vi /etc/mdadm/mdadm.conf  ## remove /dev/md0 ARRAY
update-initramfs -u
```

8. Create a two-way mirror, mount it under **/srv/pro42**.

```
mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb /dev/sdc
mkfs.ext4 /dev/md0
mount /dev/md0 /srv/pro42
watch /proc/mdstat
```

9. Convert the two-way mirror to a three disk RAID 5. Then grow the ext4 filesystem.

```
mdadm --add /dev/md0 /dev/sdd
mdadm --grow /dev/md0 --level=3 --raid-devices=3
resize2fs /dev/md0
```

Chapter 51

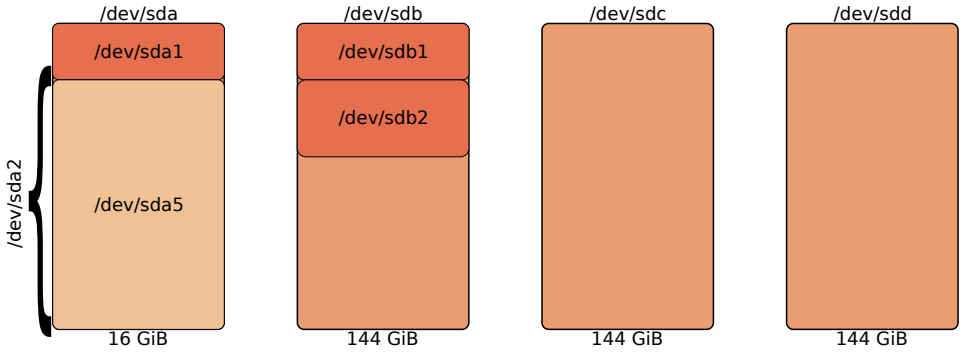
LVM storage

At the start of this chapter all **md** devices from the previous chapter have been removed as were all partitions on /dev/sd[bcdefg]. We also removed **mdadm** with **apt-get remove mdadm**. We start with a clean environment of six empty hard disks.

51.1 About Logical Volume Management

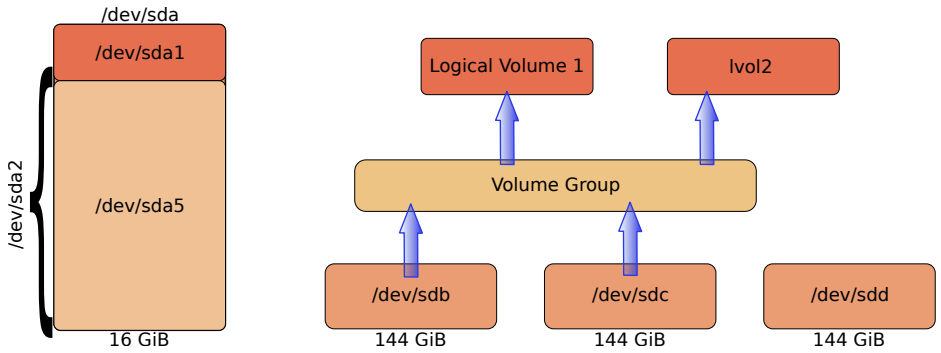
The problem with the classical scheme of putting a filesystem on a partition is that it is not easy to enlarge the partition, even if there is free space on the disk. It is also not easy to move the filesystem to another disk (for example when the disk is failing).

Consider this drawing with primary partitions **/dev/sdb1** and **/dev/sdb2**. You cannot enlarge the partition **/dev/sdb1** without taking the filesystem offline, even though there is still free space on the disk. Also you cannot move **/dev/sdb2** to the empty disk **/dev/sdc** without taking the filesystem offline. The direct link between partition and filesystem is not flexible.



Enter **Logical Volume Management** which puts an abstraction layer between the hardware and the filesystems. Instead of putting a filesystem directly on a partition (or block device), we group the block devices (called Physical Volumes) in a virtual layer (called a Volume Group) and put our filesystems on Logical Volumes (that reside in a Volume Group).

Consider this drawing where **/dev/sdb** and **/dev/sdc** are added to a Volume Group, and two Logical Volumes are created in the Volume Group. The Logical Volumes are like partitions, so you can put a filesystem on them.



With **LVM** it is easy to enlarge a Volume Group (by adding **/dev/sdd**), to remove a faulty drive (like **/dev/sdc**) and to migrate a Logical Volume to other disks. All this without unmounting the filesystem. So let's install **LVM**.

```
apt-get install lvm2
```

51.2 Creating a basic LVM setup.

51.2.1 Adding Physical Volumes to LVM

We start by telling LVM, with the **pvcreate** command, that we have two disks that can be used. It is advised to always create one primary partition spanning the whole disk, instead of adding the disk device itself.

```
root@server2:~# parted /dev/sdb mklabel msdos mkpart primary 0 100%
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.
```

```
root@server2:~# parted /dev/sdc mklabel msdos mkpart primary 0 100%
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server2:~# pvcreate /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sdb1" successfully created.
  Physical volume "/dev/sdc1" successfully created.
root@server2:~#
```

51.2.2 Creating a Volume Group

The next step is to create a volume group, with **vgcreate**, and to add our two disks to the volume group. The volume group is the layer between our disks and our filesystems. We can verify with **vgs** that the size is indeed 288GiB.

```
root@server2:~# vgcreate VG42 /dev/sdb1 /dev/sdc1
  Volume group "VG42" successfully created
root@server2:~# vgs
  VG   PV #LV #SN Attr   VSize   VFree
  VG42   2   0   0 wz--n- 287.99g 287.99g
root@server2:~
```

51.2.3 Creating a Logical Volume

The final step in LVM is to create a logical volume. We don't supply a name, so the default name **lvol0** is chosen. We can verify the logical volume with **lvs**. The logical volume resides in the volume group **VG42**, we don't need to care whether it is on /dev/sdb1 or /dev/sdc1.

```
root@server2:~# lvcreate VG42 --size=1G
  Logical volume "lvol0" created.
root@server2:~# lvs
  LV    VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  lvol0 VG42 -wi-a----- 1.00g
root@server2:~#
```

51.2.4 Using the Logical Volume

The rest of the story should be familiar. Put **ext4** on the logical volume as if it was a regular partition, create a mount point and mount it.

```
root@server2:~# mkfs.ext4 /dev/VG42/lvol0
mke2fs 1.44.5 (15-Dec-2018)
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: 342b8820-ef60-49d1-9768-c3b00aa74ac8
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

root@server2:~# mkdir /srv/project42
root@server2:~# mount /dev/VG42/lvol0 /srv/project42/
root@server2:~# df -h | grep project42
/dev/mapper/VG42-lvol0  976M  2.6M  907M   1% /srv/project42
root@server2:~#
```

51.3 Growing the Logical Volume

The logical volume can be extended easily using **lvextend** followed by **resize2fs**. In the screenshot we resize **VG42/lvol0** from 1GB to 3GB. There is no need to unmount the active filesystem.

```
root@server2:~# lvextend -L 3GB VG42/lvol0
  Size of logical volume VG42/lvol0 changed from 1.00 GiB (256 extents) to 3.00 GiB (768 ↩
      extents).
  Logical volume VG42/lvol0 successfully resized.
root@server2:~# resize2fs /dev/VG42/lvol0
resize2fs 1.44.5 (15-Dec-2018)
Filesystem at /dev/VG42/lvol0 is mounted on /srv/project42; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/VG42/lvol0 is now 786432 (4k) blocks long.

root@server2:~# df -h | grep project42
/dev/mapper/VG42-lvol0  3.0G  3.0M  2.8G   1% /srv/project42
root@server2:~#
```

51.4 More information about PV, VG and LV

We can obtain more information about the physical volumes with **pvdisplay**, about the volume group with **vgdisplay** and about the logical volume with **lvdisplay**.

```
root@server2:~# lvdisplay
  --- Logical volume ---
  LV Path                /dev/VG42/lvol0
  LV Name                lvol0
  VG Name                VG42
  LV UUID                Y3jxZv-ePyv-sEG6-BaV7-5tsN-ZXQ8-5oc9Nc
  LV Write Access        read/write
  LV Creation host, time server2, 2019-08-30 17:45:39 +0200
  LV Status              available
  # open                 1
  LV Size                3.00 GiB
  Current LE             768
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:0

root@server2:~#
```

51.5 Creating a RAID1 Logical Volume

LVM supports RAID1 logical volumes. This means that the logical volume is spread equally across two disks, each disk holding a copy of the data. In the screenshot below we use **lvcreate** to create a 2GB RAID1 device, and name it **lvproject42**.

```
root@server2:~# lvcreate -m1 -L 2G -n lvproject42 VG42
  Logical volume "lvproject42" created.
root@server2:~#
```

You can now put a filesystem on this logical volume and mount it, as if it was a regular partition. Except that it is a RAID1 device. The **lvs** command now shows two logical volumes.

```
root@server2:~# lvs
  LV          VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  lvol0       VG42 -wi-ao---- 3.00g
  lvproject42 VG42 rwi-a-r--- 2.00g                                    100.00
root@server2:~#
```

51.6 Expanding the Volume Group

We want to grow our **VG42** volume group with another 144GiB disk named **/dev/sdd**. We copy the partition information from **/dev/sdb**, use **pvcreate** on the partition and then extend our **VG42** with **vgextend**.

```
root@server2:~# sfdisk -d /dev/sdb | sfdisk /dev/sdd
Checking that no-one is using this disk right now ... OK
<output truncated>
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
root@server2:~# pvcreate /dev/sdd1
  Physical volume "/dev/sdd1" successfully created.
root@server2:~# vgextend VG42 /dev/sdd1
  Volume group "VG42" successfully extended
root@server2:~#
```

Our VG42 volume group now has three disks in it, totalling 432GiB.

```
root@server2:~# vgs
  VG   PV #LV #SN Attr   VSize    VFree
  VG42   3   2   0 wz--n- <431.99g 424.98g
root@server2:~ vgdisplay | grep PV
  Max PV                0
  Cur PV                3
  Act PV                3
root@server2:~#
```

51.7 lsblk

The **lsblk** tool gives a very nice view of the LVM volume groups and logical volumes.

```
root@server2:~# lsblk /dev/sdb /dev/sdc
NAME                          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb                             8:16   0  144G  0 disk
`-sdb1                          8:17   0  144G  0 part
  |-VG42-lvol0                254:0    0    3G  0 lvm  /srv/project42
  |-VG42-lvproject42_rmeta_0  254:1    0    4M  0 lvm
  | `-VG42-lvproject42        254:5    0    2G  0 lvm
  `-VG42-lvproject42_rimage_0 254:2    0    2G  0 lvm
    `-VG42-lvproject42        254:5    0    2G  0 lvm
sdc                             8:32   0  144G  0 disk
`-sdc1                          8:33   0  144G  0 part
  |-VG42-lvproject42_rmeta_1  254:3    0    4M  0 lvm
  | `-VG42-lvproject42        254:5    0    2G  0 lvm
  `-VG42-lvproject42_rimage_1 254:4    0    2G  0 lvm
    `-VG42-lvproject42        254:5    0    2G  0 lvm
root@server2:~#
```

51.8 Creating a RAID5 logical volume

Creating a RAID5 volume requires three physical volumes, luckily we just added one. The command is a very straightforward **lvcreate**. You can verify with **lsblk** that project33 resides on three different disks.

```
root@server2:~# lvcreate --type raid5 -L 10G -n lvproject33 VG42
  Using default stripesize 64.00 KiB.
  Logical volume "lvproject33" created.
root@server2:~#
```

51.9 Replacing a faulty disk.

In our example here we will assume that **/dev/sdc** is failing and that **/dev/sdg** is the replacement disk. The first step is then to add **/dev/sdg** to the volume group.

```
root@server2:~# sfdisk -d /dev/sdb | sfdisk /dev/sdg
<output truncated>
root@server2:~# pvcreate /dev/sdg1
  Physical volume "/dev/sdg1" successfully created.
root@server2:~# vgextend VG42 /dev/sdg1
  Volume group "VG42" successfully extended
root@server2:~#
```

Then we have to tell LVM that **/dev/sdc1** is about to be removed from the volume group. The **pvmove** command will (try to) move all data to the other disks. Filesystems can stay mounted, files can be open.

```
root@server2:~# pvmove /dev/sdc1
  /dev/sdc1: Moved: 0.39%
  /dev/sdc1: Moved: 28.54%
  /dev/sdc1: Moved: 28.60%
  /dev/sdc1: Moved: 99.94%
  /dev/sdc1: Moved: 100.00%
root@server2:~#
```

You can verify with **lsblk** that all logical volumes were moved off of **/dev/sdc1**. We can now remove this physical volume from LVM with **vgreduce** and **pvremove**.

```
root@server2:~# vgreduce VG42 /dev/sdc1
  Removed "/dev/sdc1" from volume group "VG42"
root@server2:~# pvremove /dev/sdc1
  Labels on physical volume "/dev/sdc1" successfully wiped.
root@server2:~#
```

51.10 Creating a snapshot

You may require snapshots for consistent backups. A snapshot of a logical volume is easily created. In the screenshot we create a 300MB snapshot of our 3GB **lvol0**. Snapshots in LVM are copy-on-write.

```
root@server2:~# lvcreate --snapshot --size 300M -n snaplvol0 VG42/lvol0
  Logical volume "snaplvol0" created.
root@server2:~# lvs
  LV          VG   Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  lvol0       VG42 owi-aos---   3.00g
  lvproject33 VG42 rwi-a-r---  10.00g                                     100.00
  lvproject42 VG42 rwi-a-r---   2.00g                                     100.00
  snaplvol0   VG42 swi-a-s--- 300.00m      lvol0  0.01
root@server2:~#
```

Tip
Snapshots are an important concept in storage backups, if this is unclear then visit the Wikipedia page https://en.wikipedia.org/-wiki/Snapshot_(computer_storage) .

51.11 Thin provisioning

LVM allows for thin provisioning of logical volumes. This enables us to create a 2TB logical volume, named **lvollie**, using only 2GB of actual space. We obviously get a number of warnings on disk space.

```
root@server2:~# lvcreate --type thin --name lvollie -V 2T -L 2G VG42
  Thin pool volume with chunk size 64.00 KiB can address at most 15.81 TiB of data.
  WARNING: Sum of all thin volume sizes (2.00 TiB) exceeds the size of thin pool VG42/lvol2 ↩
       and the size of whole volume group (<431.99 GiB).
  WARNING: You have not turned on protection against thin pools running out of space.
  WARNING: Set activation/thin_pool_autoextend_threshold below 100 to trigger automatic  ↩
      extension of thin pools before they get full.
  Logical volume "lvollie" created.
root@server2:~#
```

Yeah, we have a two tebibyte volume on our server2.

```
root@server2:~# mkfs.ext4 /dev/VG42/lvollie
<output truncated>
root@server2:~# mkdir /srv/lie
root@server2:~# mount /dev/VG42/lvollie /srv/lie/
root@server2:~# df -h | grep lie
/dev/mapper/VG42-lvollie  2.0T   81M  1.9T   1% /srv/lie
root@server2:~#
```

51.12 mdadm or lvm?

The question, of which of these two to use for example for RAID5, is a valid question. Performance wise it should be the same since lvm and mdadm use the same kernel driver for RAID5. So pick whichever you feel comfortable with.

51.13 Cheat sheet

Table 51.1: LVM

command	explanation
pvcreate /dev/sda /dev/sdb	Add physical volumes to LVM.
pvcreate /dev/sdc1 /dev/sdd3	Add physical volumes to LVM (partitions).
vgcreate foo /dev/sda /dev/sdd3	Create a volume group named **foo**.
lvcreate foo --size=1G	Create a one gigabyte logical volume in volume group **foo**.
pvs	Display information about physical volumes.
vgs	Display information about volume groups.
lvs	Display information about logical volumes.
pvdisplay	Display detailed information about physical volumes.
vgdisplay	Display detailed information about volume groups.
lvdisplay	Display detailed information about logical volumes.
mkfs.ext4 /dev/foo/lvol0	Put a file system on logical volume lvol0 in volume group **foo**.
lvextend -L 100GB foo/lvol0	Extend the lvol0 logical volume with 100 gibibyte.
lvcreate -m1 -L 100G foo	Create a 100 gibibyte mirrored (RAID 1) logical volume in volume group **foo**.
lvcreate --type raid5 ...	Create a RAID 5 logical volume.
vgextend foo /dev/sdd1	Extend the voluem group **foo** with device /dev/sdd1.
lsblk	List block devices and LVM information.
pvmove /dev/bar	Move all LVM date off off the device **/dev/foo**.
vgreduce foo /dev/bar	Remove the **/dev/bar** device from volume group **foo**.
pvremove /dev/bar	Remove the **/dev/bar** device from LVM.
lvcreate --snapshot ...	Create a snapshot of a logical volume.
lvcreate --type thin ...	Create a thin provisioned logical volume.

51.14 Practice

1. Make sure (with **lsblk** for example) that there are no lingering **md** devices or partitions on your extra disks. Unmount it all, wipe the extra disks and remove entries from **/etc/fstab** and **mdadm.conf**. You can even remove **mdadm**.
2. Add two disks (their 100% primary partition) to LVM.
3. Create a volume group named VG33 on /dev/sdb1.
4. Extend the volume group named VG33 with /dev/sdc1.
5. Create a 500MB logical volume and a 1000MB logical volume in VG33.
6. Verify your work with lvs, pvs, vgs, lvdisplay, pvdisplay, vgdisplay, lsblk.
7. Extend the first logical volume with 700M.
8. Extend the volume group with /dev/sdd1.
9. Create a mirrored logical volume of 4GB, name it lvmirror.
10. Create a 1GB snapshot of the mirrored logical volume.

51.15 Solution

1. Make sure (with **lsblk** for example) that there are no lingering **md** devices or partitions on your extra disks. Unmount it all, wipe the extra disks and remove entries from **/etc/fstab** and **mdadm.conf**. You can even remove **mdadm**.

```
lsblk
dd if=/dev/zero of=/dev/sdb count=100 bs=512  # for all devices
vi /etc/fstab
vi /etc/mdadm/mdadm.conf
```

2. Add two disks (their 100% primary partition) to LVM.

```
parted /dev/sdb mklabel msdos mkpart primary 0 100%
parted /dev/sdc mklabel msdos mkpart primary 0 100%
pvcreate /dev/sdb1 /dev/sdb1
```

3. Create a volume group named VG33 on /dev/sdb1.

```
vgcreate VG33 /dev/sdb1
```

4. Extend the volume group named VG33 with /dev/sdc1.

```
vgextend VG33 /dev/sdc1
```

5. Create a 500MB logical volume and a 1000MB logical volume in VG33.

```
lvcreate -L 500M VG33
lvcreate -L 1000M VG33
```

6. Verify your work with lvs, pvs, vgs, lvdisplay, pvdisplay, vgdisplay, lsblk.

```
pvs
vgs
lvs
pvdisplay
vgdisplay
lvdisplay
lsblk
```

7. Extend the first logical volume with 700M.

```
lvextend -L +700 VG33/lvol0
```

8. Extend the volume group with /dev/sdd1.

```
sfdik -d /dev/sdb | sfdisk /dev/sdd
vgextend VG33 /dev/sdd1
```

9. Create a mirrored logical volume of 4GB, name it lvmirror.

```
lvcreate -m 1 -L 4G -n lvmirror VG33
```

10. Create a 1GB snapshot of the mirrored logical volume.

```
lvcreate --snapshot --size 1G VG33/lvmirror
```

Chapter 52

ZFS

52.1 Installing ZFS

Warning
Debian 10 Linux does not provide kernel modules for ZFS by default. **The reason is there may be a licensing conflict between GPL (the kernel license) and CDDL (the ZFS license).** This chapter is entirely optional!

If you still want to proceed, then ZFS is available in **contrib** (see the Installing Software chapter). To install software from **contrib** you need to change two lines in **/etc/apt/sources.list**. Take a backup of this file, then add the word **contrib** to the following two lines.

```
root@server2:/etc/apt# cp /etc/apt/sources.list /etc/apt/sources.original
root@server2:/etc/apt# vi /etc/apt/sources.list
root@server2:/etc/apt# grep contrib /etc/apt/sources.list
deb http://deb.debian.org/debian/ buster main contrib
deb-src http://deb.debian.org/debian/ buster main contrib
root@server2:/etc/apt#
```

Then you can do **apt-get update** followed by **apt-get upgrade**. This will refresh the list of all Debian packages available in **main** and in **contrib**, followed by an installation of all available upgrades (if there are any).

Then issue these two commands to install the **ZFS** kernel modules. These commands may take a while to complete since a lot of stuff is compiled from source.

```
apt-get install spl-dkms
apt-get install zfs-dkms
```

Note
ZFS can take a lot of RAM memory, at least 4GB is recommended.

52.2 Creating a basic ZFS pool

ZFS is a filesystem and a volume manager and a mount command. In the screenshot below we use **zpool create** to create a striped volume, a ZFS filesystem, a mount point and the ZFS volume is mounted on that mount point all in one command.

```
root@server2:~# modprobe zfs
root@server2:~# zpool create mypool /dev/sdb /dev/sdc
root@server2:~# df -h | grep mypool
mypool          278G     0  278G   0% /mypool
root@server2:~#
```

52.3 Verifying a ZFS pool

Two useful commands for verifying a ZFS pool are **zpool list** for a nice summary, and **zpool status** for device and data integrity information.

```
root@server2:~# zpool list
NAME     SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
mypool   286G  1.19G   285G         -     0%     0%  1.00x  ONLINE  -
root@server2:~# zpool status mypool
  pool: mypool
 state: ONLINE
```

```
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          sdb       ONLINE       0     0     0
          sdc       ONLINE       0     0     0

errors: No known data errors
root@server2:~#
```

The status of all pools can be quickly verified with **zpool status -x**.

```
root@server2:~# zpool status -x
all pools are healthy
root@server2:~#
```

52.4 Destroying a ZFS pool

The **zpool destroy** command will unmount the ZFS filesystem, remove the mount point, and destroy the ZFS pool. All data on the ZFS pool will be lost.

```
root@server2:~# zpool destroy mypool
root@server2:~#
```

52.5 Creating a ZFS RAID5 pool

The next **zpool create** command will create a RAID5 (called a RAIDZ by ZFS) over three disks (/dev/sdb /dev/sdc /dev/sdd), will named the pool **pro42pool**, and it will also create a custom mount point and mount the filesystem on it.

```
root@server2:~# zpool create -m /srv/project42 pro42pool raidz sdb sdc sdd
root@server2:~# df -h | grep project42
pro42pool       277G     0  277G   0% /srv/project42
root@server2:~#
```

52.6 Setting ZFS disk quota

In this example we create a ZFS filesystem (and mount point) and set a 30GB quota on it. Note that we use the **zfs** command instead of **zpool**.

```
root@server2:~# zfs create pro42pool/div01
root@server2:~# zfs set quota=30G pro42pool/div01
root@server2:~# zfs list
NAME              USED  AVAIL  REFER  MOUNTPOINT
pro42pool         151K   276G  30.6K  /srv/project42
pro42pool/div01  30.6K  30.0G  30.6K  /srv/project42/div01
root@server2:~#
```

52.7 ZFS properties

There are many properties that you can set on a ZFS filesystem. You can see them all with the **zfs get all** command as shown in this example.

```
root@server2:~# zfs get all pro42pool/div01 | head
NAME             PROPERTY              VALUE                  SOURCE
pro42pool/div01  type                  filesystem             -
pro42pool/div01  creation              Sat Aug 31 17:45 2019  -
pro42pool/div01  used                  30.6K                  -
pro42pool/div01  available             30.0G                  -
pro42pool/div01  referenced            30.6K                  -
pro42pool/div01  compressratio         1.00x                  -
pro42pool/div01  mounted               yes                    -
pro42pool/div01  quota                 30G                    local
pro42pool/div01  reservation           none                   default
root@server2:~#
```

52.7.1 query individual properties

Using the **zfs get** command you can query for individual properties on a zfs filesystem. In the example here we query for the **quota** we set before.

```
root@server2:~# zfs get quota pro42pool/div01
NAME             PROPERTY  VALUE  SOURCE
pro42pool/div01  quota     30G    local
root@server2:~#
```

52.7.2 Setting reserved disk space

One useful property is to set a minimum reserved space for a zfs filesystem. Notice how the pool available space shrink 100GB when reserving this space.

```
root@server2:~# zfs create pro42pool/div02
root@server2:~# zfs set reservation=100G pro42pool/div02
root@server2:~# zfs list
NAME              USED  AVAIL  REFER  MOUNTPOINT
pro42pool         100G   176G  32.0K  /srv/project42
pro42pool/div01  30.6K  30.0G  30.6K  /srv/project42/div01
pro42pool/div02  30.6K   276G  30.6K  /srv/project42/div02
root@server2:~#
```

52.7.3 Activating compression

You can activate compression to save disk space. Don't forget that compressing twice is a waste of CPU power. The screenshot shows the **zfs set compression=on** command.

```
root@server2:~# zfs get compression pro42pool/div01
NAME             PROPERTY     VALUE     SOURCE
pro42pool/div01  compression  off       local
root@server2:~# zfs set compression=on pro42pool/div01
root@server2:~# zfs get compression pro42pool/div01
NAME             PROPERTY     VALUE     SOURCE
pro42pool/div01  compression  on        local
root@server2:~#
```

52.8 Monitoring ZFS

There is **zpool iostat** command to monitor a zpool every x seconds. In the screenshot we ask for statistics every two seconds.

```
root@server2:~# zpool iostat 2
               capacity     operations     bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
pro42pool    210M   428G      0      1     52  19.6K
pro42pool    634M   427G      0  18.9K      0   170M
pro42pool    869M   427G      0  7.11K      0  85.7M
```

Use **zpool iostat -v** for detailed statistics about each hard disk in the pool.

```
root@server2:~# zpool iostat -v 5
               capacity     operations     bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
pro42pool    887M   427G      0      4     51  47.7K
  raidz1     887M   427G      0      4     51  47.7K
    sdb         -      -      0      1     25  15.9K
    sdc         -      -      0      1     25  15.9K
    sdd         -      -      0      1      1  15.9K
----------  -----  -----  -----  -----  -----  -----
^C
root@server2:~#
```

52.9 Creating ZFS snapshots

It is easy to create copy-on-write snapshots with the **zfs snapshot** command.

```
root@server2:~# zfs snapshot pro42pool/div02@2019aug31
root@server2:~# zfs list -t all
NAME                          USED  AVAIL  REFER  MOUNTPOINT
pro42pool                     101G   176G  32.0K  /srv/project42
pro42pool/div01               591M  29.4G   591M  /srv/project42/div01
pro42pool/div02              30.6K   276G  30.6K  /srv/project42/div02
pro42pool/div02@2019aug31        0B      -  30.6K  -
root@server2:~#
```

52.10 ZFS send and receive

You can send a ZFS filesystem over the network with **zfs send** and **ssh**, and receive it on another server with **zfs receive**. This will be discussed later in the Backup chapter.

```
root@server2:~# zfs send pro42pool/div02@2019aug31 | ssh ba@server3 'zfs receive backuppool ↩
    /div02@2019aug31'
ba@server3's password:
root@server2:~#
```

52.11 ZFS network sharing

We will discuss ZFS **NFS sharing** in the NFS chapter and **SMB sharing** in one of the Samba chapters.

52.12 Cheat sheet

Table 52.1: ZFS

command	explanation
zpool create foo /dev/sdb /dev/sdc	Create a new zpool named **foo** using two block devices.
zpool list	List information about zpools.
zpool status foo	Display information about the zpool named **foo**.
zpool status -x	Succinct information about all zpools.
zpool destroy foo	Delete the zpool named **foo**.
zpool create foo raidz sdb sdc sdd	Create a raidz (RAID 5) zpool named **foo**.
zfs create foo/bar	Create a zfs file system named **bar** in the zpool named **foo**.
zfs set quota=100G foo/bar	Set a 100 gigabyte quota for **foo/bar**.
zfs set reservation=42G foo/bar	Reserve 42 gigabyte for **foo/bar**.
zfs set compression=on foo/bar	compress the zfs file system **foo/bar**.
zfs get all foo/bar	List all properties for the zfs file system **foo/bar**.
zpool iostat 2	Display statistics every two seconds.
zfs snapshot foo/bar@date	Create a snapshot of **foo/bar**.
zfs send foo/bar@date | ssh	Send the snapshot to another server.
zfs receive backupfoo/bar@date	Receive a snapshot from a server.

52.13 Practice

1. Follow the instructions in the theory to install ZFS.
2. Create a zpool RAID1 mirror named **mymirpool**.
3. Create a **division01** zfs filesystem in this pool with a 10GB quota.
4. Verify your work with **zfs list** .
5. Read the **zfs** man page about **deduplication** and set this property to on for **division01**.
6. Create a snapshot from the **division01** ZFS filesystem.
7. Create a disk I/O heavy job in the background and monitor the **zpool** every three seconds.
8. Destroy your pool and all its subdirectories.

52.14 Solution

1. Follow the instructions in the theory to install ZFS.

```
cp /etc/apt/sources.list /etc/apt/sources.original
vi /etc/apt/sources.list
apt-get install spl-dkms
apt-get install zfs-dkms
```

2. Create a zpool RAID1 mirror named **mymirpool**.

```
zpool create mymirpool mirror sde sdf
```

3. Create a **division01** zfs filesystem in this pool with a 10GB quota.

```
zfs create -o quota=10G mymirpool/division01
```

4. Verify your work with **zfs list** .

```
zfs list
```

5. Read the **zfs** man page about **deduplication** and set this property to on for **division01**.

```
man zfs
zfs set dedup=on mymirpool/division01
```

6. Create a snapshot from the **division01** ZFS filesystem.

```
zfs snapshot mymirpool/division01@2019aug31
```

7. Create a disk I/O heavy job in the background and monitor the **zpool** every three seconds.

```
cp -r /usr /mymirpool/division01 &
zpool iostat 3
```

8. Destroy your pool and all its subdirectories.

```
zpool destroy mymirpool
```

Chapter 53

iSCSI storage

53.1 SCSI over TCP/IP

SCSI or Small Computer System Interface is a protocol to locally connect hard disks (and other devices) to the computer. **iSCSI** enables SCSI-storage devices (everything that uses the kernel **sd** module) over the TCP/IP network.

This allows for dedicated storage servers (often SAN) that provide storage over the network to application servers. **iSCSI** is a widely adopted standard among storage providers.

The **iSCSI** server is called a **target**. We will start by configuring a **target** server named **server2**. The **iSCSI** client is called an **initiator**. The **initiator** in this chapter is called **server3**.

53.2 iSCSI target

The **iSCSI target** software is built into the Linux kernel. It can be installed with **apt-get install targetcli-fb** (the fb means free branch, there once was another targetcli).

```
root@server2:~# apt-get install targetcli-fb
```

After the installation, you get a **targetcli** command that opens an interactive command line interface.

```
root@server2:~# targetcli
Warning: Could not load preferences file /root/.targetcli/prefs.bin.
targetcli shell version 2.1.fb48
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type help.

/>
```

53.2.1 targetcli

In this **targetcli** you can type **ls** to get a directory structure that you can navigate with **cd**. You can also use **Ctrl-r** and **arrow-up** like in the bash shell.

You should see four different **backstores**, we will focus on the **block** backstore, this takes a block device and shares it as a block device.

```
/> ls
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block .................................................. [Storage Objects: 0]
  | o- fileio ................................................. [Storage Objects: 0]
  | o- pscsi .................................................. [Storage Objects: 0]
  | o- ramdisk ................................................ [Storage Objects: 0]
  o- iscsi ............................................................ [Targets: 0]
  o- loopback ......................................................... [Targets: 0]
  o- vhost ............................................................ [Targets: 0]
  o- xen-pvscsi ....................................................... [Targets: 0]
/>
```

53.2.2 Adding a block device

We will start by adding **/dev/sdb** as a block device for iSCSI. We name it **iSCSIpro42** . This is the block device that we will share over the network.

```
/> cd backstores/block
/backstores/block> create iSCSIpro42 /dev/sdb
Created block storage object iSCSIpro42 using /dev/sdb.
/backstores/block> ls
o- block ..................................................... [Storage Objects: 1]
  o- iSCSIpro42 ........................... [/dev/sdb (144.0GiB) write-thru deactivated]
    o- alua ....................................................... [ALUA Groups: 1]
      o- default_tg_pt_gp ............................... [ALUA state: Active/optimized]
/backstores/block>
```

53.2.3 Creating a target

The next step is to create a **target** for the **initiator**. This **target** has to be an **IQN**. An iSCSI Qualified Name has the format **iqn.YYYY-MM.naming-authority:unique name** .

```
/backstores/block> cd /iscsi
/iscsi> create iqn.2008-08.be.linux-training.server2:iTargetpro42
Created target iqn.2008-08.be.linux-training.server2:itargetpro42.
Created TPG 1.
Global pref auto_add_default_portal=true
Created default portal listening on all IPs (0.0.0.0), port 3260.
/iscsi>
```

53.2.4 Linking block device with target

The next step is to link the **block device** that we named **iSCSIpro42** with the **target** (which we gave the long IQN name).

```
/iscsi> cd iqn.2008-08.be.linux-training.server2:itargetpro42/tpg1/luns
/iscsi/iqn.20...o42/tpg1/luns> create /backstores/block/iSCSIpro42
Created LUN 0.
/iscsi/iqn.20...o42/tpg1/luns>
```

53.2.5 Setting CHAP authentication

Next we configure the access for user **root** with password **hunter2**.

```
/iscsi/iqn.20...o42/tpg1/luns> cd ../acls
/iscsi/iqn.20...o42/tpg1/acls> create iqn.2008-08.be.linux-training.server3:iInitpro42
Created Node ACL for iqn.2008-08.be.linux-training.server3:iinitpro42
Created mapped LUN 0.
/iscsi/iqn.20...o42/tpg1/acls> cd iqn.2008-08.be.linux-training.server3:iinitpro42/
/iscsi/iqn.20...r3:iinitpro42> set auth userid=root
Parameter userid is now root.
/iscsi/iqn.20...r3:iinitpro42> set auth password=hunter2
Parameter password is now hunter2.
/iscsi/iqn.20...r3:iinitpro42>
```

53.2.6 Verifying the configuration

The complete configuration should look similar to this screenshot.

```
/> ls
o- / ..................................................................... [...]
  o- backstores ......................................................... [...]
```

```
  | o- block ..................................................... [Storage Objects: 1]
  | | o- iSCSIpro42 ......................... [/dev/sdb (144.0GiB) write-thru activated]
  | |   o- alua ........................................................ [ALUA Groups: 1]
  | |     o- default_tg_pt_gp ........................... [ALUA state: Active/optimized]
  | o- fileio .................................................... [Storage Objects: 0]
  | o- pscsi ..................................................... [Storage Objects: 0]
  | o- ramdisk ................................................... [Storage Objects: 0]
  o- iscsi ................................................................ [Targets: 1]
  | o- iqn.2008-08.be.linux-training.server2:itargetpro42 ..................... [TPGs: 1]
  |   o- tpg1 ................................................... [no-gen-acls, no-auth]
  |     o- acls .............................................................. [ACLs: 1]
  |     | o- iqn.2008-08.be.linux-training.server3:iinitpro42 .......... [Mapped LUNs: 1]
  |     |   o- mapped_lun0 ................................ [lun0 block/iSCSIpro42 (rw)]
  |     o- luns .............................................................. [LUNs: 1]
  |     | o- lun0 ..................... [block/iSCSIpro42 (/dev/sdb) (default_tg_pt_gp)]
  |     o- portals ........................................................ [Portals: 1]
  |       o- 0.0.0.0:3260 ......................................................... [OK]
  o- loopback ............................................................. [Targets: 0]
  o- vhost ................................................................ [Targets: 0]
  o- xen-pvscsi ........................................................... [Targets: 0]
/>
```

53.2.7 Repeating it all

Doing the same process again with **/dev/sdc** and **Project33** we end up with this configuration.

```
/> ls
o- / ..................................................................... [...]
  o- backstores ......................................................... [...]
  | o- block ..................................................... [Storage Objects: 2]
  | | o- iSCSIpro33 ......................... [/dev/sdc (144.0GiB) write-thru activated]
  | | | o- alua ........................................................ [ALUA Groups: 1]
  | | |   o- default_tg_pt_gp ........................... [ALUA state: Active/optimized]
  | | o- iSCSIpro42 ......................... [/dev/sdb (144.0GiB) write-thru activated]
  | |   o- alua ........................................................ [ALUA Groups: 1]
  | |     o- default_tg_pt_gp ........................... [ALUA state: Active/optimized]
  | o- fileio .................................................... [Storage Objects: 0]
  | o- pscsi ..................................................... [Storage Objects: 0]
  | o- ramdisk ................................................... [Storage Objects: 0]
  o- iscsi ................................................................ [Targets: 2]
  | o- iqn.2008-08.be.linux-training.server2:itargetpro33 ..................... [TPGs: 1]
  | | o- tpg1 ................................................... [no-gen-acls, no-auth]
  | |   o- acls .............................................................. [ACLs: 1]
  | |   | o- iqn.2008-08.be.linux-training.server3:iinitpro33 .......... [Mapped LUNs: 1]
  | |   |   o- mapped_lun0 ................................ [lun0 block/iSCSIpro33 (rw)]
  | |   o- luns .............................................................. [LUNs: 1]
  | |   | o- lun0 ..................... [block/iSCSIpro33 (/dev/sdc) (default_tg_pt_gp)]
  | |   o- portals ........................................................ [Portals: 1]
  | |     o- 0.0.0.0:3260 ......................................................... [OK]
  | o- iqn.2008-08.be.linux-training.server2:itargetpro42 ..................... [TPGs: 1]
  |   o- tpg1 ................................................... [no-gen-acls, no-auth]
  |     o- acls .............................................................. [ACLs: 1]
  |     | o- iqn.2008-08.be.linux-training.server3:iinitpro42 .......... [Mapped LUNs: 1]
  |     |   o- mapped_lun0 ................................ [lun0 block/iSCSIpro42 (rw)]
  |     o- luns .............................................................. [LUNs: 1]
  |     | o- lun0 ..................... [block/iSCSIpro42 (/dev/sdb) (default_tg_pt_gp)]
  |     o- portals ........................................................ [Portals: 1]
  |       o- 0.0.0.0:3260 ......................................................... [OK]
  o- loopback ............................................................. [Targets: 0]
  o- vhost ................................................................ [Targets: 0]
```

```
  o- xen-pvscsi ......................................................... [Targets: 0]
/>
```

53.2.8 Saving the configuration

To finish just type **saveconfig** and then **exit** in the **targetcli**. The configuration is then saved and activated. The iSCSI target is listening on port 3260.

```
root@server2:~# ss -anpt | grep 3260
LISTEN    0         256                0.0.0.0:3260            0.0.0.0:*
root@server2:~#
```

Important
After a **reboot** the configuration on our server required restoring. We automate this restoration in the next section.

53.3 Automating the restoration of the target configuration

When you reboot the **iSCSI target**, then the saved configuration is not active. It can be activated by running **targetctl restore**. We put this command in **/etc/rc.local** and make this file executable. Then this script will run at the end of a system startup.

```
root@server2:~# cat /etc/rc.local
#!/bin/bash
targetctl restore
exit 0
root@server2:~# chmod +x /etc/rc.local
root@server2:~#
```

53.4 iSCSI Initiator

The iSCSI initiator is the *client*. We use the **open-iscsi** package to configure a client on **server3**. So **server3** will receive a block device over TCP/IP from **server2**.

Start by installing the iSCSI client package **open-iscsi**.

```
apt-get install open-iscsi
```

53.4.1 /etc/iscsi/iscsi.conf

The configuration of the iSCSI initiator happens in the **/etc/iscsi/iscsi.conf** file. For example you can set **node.startup** to manual or automatic.

```
root@server3:~# grep node.startup /etc/iscsi/iscsid.conf
# node.startup = automatic
node.startup = manual
root@server3:~#
```

53.4.2 Setting CHAP authentication

We shared the block device with CHAP authentication, so this information will also need to be configured in **/etc/iscsi/iscsid.conf** .

```
root@server3:~# grep auth /etc/iscsi/iscsid.conf | grep ^[^#]
node.session.auth.authmethod = CHAP
node.session.auth.username = root
node.session.auth.password = hunter2
root@server3:~#
```

53.4.3 Setting initiator name

The initiator name that we configured on **server2** will need to be set in **/etc/iscsi/initiatorname.iscsi** .

```
root@server3:~# tail -1 /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2008-08.be.linux-training.server3:iinitpro42
root@server3:~# service iscsid restart
root@server3:~#
```

53.4.4 Discovering the target

We can now take a look at the targets that are shared at our **server2**. 192.168.1.22 is the ip-address of **server2**. We see both the shared devices.

```
root@server3:~# iscsiadm -m discovery -t st -p 192.168.1.22
192.168.1.22:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro42
192.168.1.22:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro33
root@server3:~#
```

53.4.5 Logging in to the target

Since **server3** is the Project42 server it will connect to the Project42 shared device. If this succeeds than the block device should appear on the initiator.

```
root@server3:~# iscsiadm -m node --targetname "iqn.2008-08.be.linux-training.server2: ↩
    itargetpro42" --portal "192.168.1.22:3260" --login
Logging in to [iface: default, target: iqn.2008-08.be.linux-training.server2:itargetpro42,  ↩
    portal: 192.168.1.22,3260] (multiple)
Login to [iface: default, target: iqn.2008-08.be.linux-training.server2:itargetpro42,  ↩
    portal: 192.168.1.22,3260] successful.
root@server3:~#
```

53.4.6 Looking at the block device

We now have a new block device on the **initiator** that is named **/dev/sdb** . It is by chance that this is the same name as the shared block device on the **target**, the **sd** driver just takes the next available letter.

```
root@server3:~# lsblk | grep 144
sdb                         8:16   0  144G  0 disk
root@server3:~#
```

53.4.7 Setting automatic login

You can change the **node.startup** line in the **/etc/iscsi/nodes/** subdirectory to **automatic** to have the block device available right after booting the server.

```
root@server3:~# vi /etc/iscsi/nodes/iqn.2008-08.be.linux-training.server2\:itargetpro42 ↩
    /192.168.1.22\,3260\,1/default
root@server3:~# grep auto /etc/iscsi/nodes/iqn.2008-08.be.linux-training.server2\: ↩
    itargetpro42/192.168.1.22\,3260\,1/default
node.startup = automatic
root@server3:~#
```

53.4.8 Removing a node

If you no longer want to connect to a **target** at boot time, then you can delete the node.

```
root@server3:~# iscsiadm -m node -o delete -T iqn.2008-08.be.linux-training.server2: ↩
    itargetpro42
root@server3:~#
```

53.5 Cheat sheet

Table 53.1: iSCSI

command	explanation
apt-get install targetcli-fb	Install the iSCSI target software.
targetcli	Start the interactive iSCSI target configuration.
ss -anpt | grep 3260	Verify that the iSCSI target daemon is listening on port 3260.
apt-get install open-iscsi	Install the iSCSI initiator software.
/etc/iscsi/iscsid.conf	File with iSCSI initiator configuration.
/etc/iscsi/initiatorname.iscsi	File with iSCSI initiator name.
iscsiadm -m discovery -t st -p 10.0.0.1	Discover the iSCSI target at 10.0.0.1.
iscsiadm -m node --targetname "iqn.name" --portal "10.0.0.1:3260" --login	Log in from initiator to target 10.0.0.1.
iscsiadm -m node -o delete -T iqn.name	Delete an iSCSI node.

53.6 Practice

1. Make sure you have root access to two Debian 10 Linux servers that are connected over a network. Note the name and IP of the server with the disks, that will be the **target**, and note the name and IP from the server that will receive the **block devices**, that will be the **initiator**.
2. Configure the **target** server to share two hard disks.
3. Configure the other computer as an **initiator** for this target.
4. If both are configured correctly, then the discover from the initiator to the target will succeed. Try the discover.
5. Log in to the target and verify that you received two block devices.
6. Set the login to automatic. Reboot the initiator and test that the block devices are there after startup.

53.7 Solution

1. Make sure you have root access to two Debian 10 Linux servers that are connected over a network. Note the name and IP of the server with the disks, that will be the **target**, and note the name and IP from the server that will receive the **block devices**, that will be the **initiator**.

```
hostname
ip a       ## on each server
```

2. Configure the **target** server to share two hard disks.

```
apt-get install targetcli-fb
targetcli
 cd backstores/block
 create disk1 /dev/sde
 create disk2 /dev/sdf
 cd /iscsi
 create iqn.2008-08.be.linux-training.be:target1
 cd iqn.2008-08.be.linux-training.be:target1
 cd tpg1/luns
 create /backstores/block/disk1
 create /backstores/block/disk2
 cd ../acls
 create iqn.2008-08.be.linux-training.be:init1
 cd iqn.2008-08.be.linux-training.be:init1
 set auth userid=paul
 set auth password=hunter2
 saveconfig
 exit
vi /etc/rc.local      ## Do not forget this before you reboot.
```

3. Configure the other computer as an **initiator** for this target.

```
apt-get install open-iscsi
vi /etc/iscsi/iscsi.conf   # set auth for paul/hunter2
vi /etc/iscsi/initiatorname.iscsi   ## set the init1 iqn
service iscsid restart
```

4. If both are configured correctly, then the discover from the initiator to the target will succeed. Try the discover.

```
iscsiadm -m discovery -t st -p 10.0.2.6  ## ip from target
```

5. Log in to the target and verify that you received two block devices.

```
iscsiadm -m node --targetname "iqn.2008-08.be.linux-training:target1 --portal  ↩
    "10.0.2.6:3260" --login
lsblk
```

6. Set the login to automatic. Reboot the initiator and test that the block devices are there after startup.

```
vi /etc/iscsi/nodes/iqn.2008-08.be.linux-training:target1\:target1/10.0.2.6\,3260\,1/ ↩
    default
reboot
lsblk
```

Chapter 54

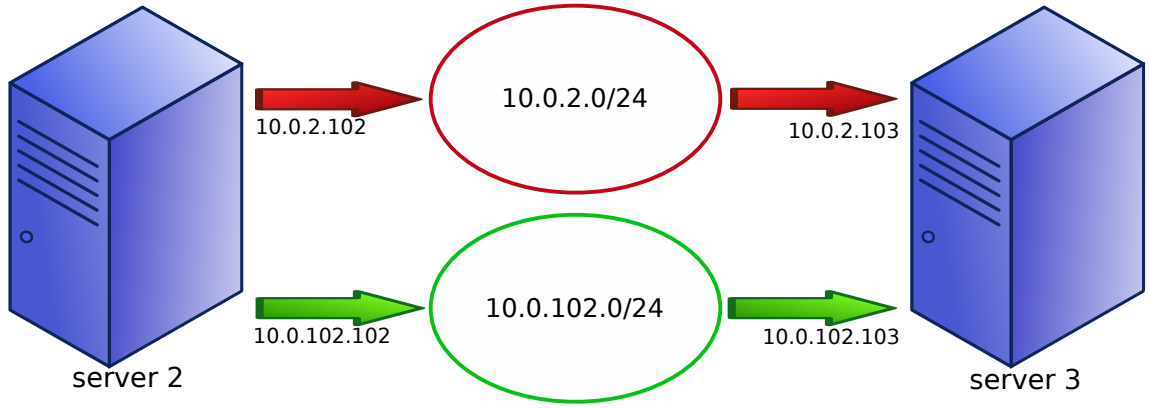
Multipath storage

54.1 About Multipath

The **iSCSI** setup from the last chapter is very nice for remote storage, but what happens when a **switch** fails, or a network cable is accidentally unplugged? Then you lose your storage.

Enter **multipath** storage. The idea is simple, an iSCSI block device is accessible over the network via several distinct paths (subnets). An interface is generated for this device to automatically fail-over to another path.

The drawing shows two paths. On the left is **server2** with two network interfaces, one interface is connected to the red path (subnet 10.0.2.0/24), another interface is connected to the green path (subnet 10.0.102.0/24). Both servers have two ip-addresses.



Note
Multipath is not limited to two distinct paths.

54.2 Configure the network

For **multipath** to work, there have to be multiple paths. In our setup the **iSCSI target** (still server2) was available on the 10.0.2.0/24 network with 10.0.2.102, and on the 10.0.102.0/24 network with 10.0.102.102.

Tip
Networking and ip-addresses are explained in the Networking chapters. The name of the **iface** is probably different in your setup. Maybe study networking first and then come back here.

```
root@server2:~# tail -12 /etc/network/interfaces
# Network1
allow-hotplug enp0s8
iface enp0s8 inet static
address 10.0.2.102
netmask 255.255.255.0

# Network2
allow-hotplug enp0s9
iface enp0s9 inet static
address 10.0.102.102
netmask 255.255.255.0
root@server2:~#
```

54.3 Test the connection on both networks

On the **iSCSI initiator** we test the connection with two distinct discovery commands. Look at the ip-addresses in the screenshot.

```
root@server3:~# iscsiadm -m discovery -t st -p 10.0.102.102
10.0.102.102:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro33
10.0.102.102:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro42
root@server3:~# iscsiadm -m discovery -t st -p 10.0.2.102
10.0.2.102:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro33
10.0.2.102:3260,1 iqn.2008-08.be.linux-training.server2:itargetpro42
root@server3:~#
```

You can now login over both these networks.

```
root@server3~# iscsiadm -m node --targetname "iqn.2008-08.be.linux-training.server2: ↩
    itargetpro42" --portal "10.0.2.102:3260" --login
<output truncated>
root@server3~# iscsiadm -m node --targetname "iqn.2008-08.be.linux-training.server2: ↩
    itargetpro42" --portal "10.0.102.102:3260" --login
<output truncated>
```

Note
Note that we are only using the **itargetpro42** IQN.

54.4 Installing multipath

```
root@server3:~# apt-get install multipath-tools
<output truncated>
root@server3:~#
```

54.5 Creating a multipath.conf file

You will need to create a minimal **multipath.conf** file. The **user_friendly_names** setting to **yes** will provide the **mpathX** names instead of the device wwn. The **find_multipaths yes** setting allows multipath to automatically create multipath devices (see **man multipath.conf** for an elaborate explanation).

```
root@server3:~# cat /etc/multipath.conf
defaults {
        user_friendly_names yes
        find_multipaths yes
}
root@server3:~#
```

54.6 Restarting the service

No more configuration is necessary, just a (re)start of the service with **systemctl restart multipathd**.

```
root@server3:~# systemctl restart multipathd
root@server3:~#
```

You should see the multipath devices with **lsblk**. Never use **sdb** or **sdc** in this scenario, only use the **/dev/mapper/mpatha** devices (for partitioning etcetera).

```
root@server3:~# lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                         8:0    0   16G  0 disk
|-sda1                      8:1    0  243M  0 part  /boot
|-sda2                      8:2    0    1K  0 part
`-sda5                      8:5    0 15.8G  0 part
  |-debianlvm--vg-root    254:0    0 14.8G  0 lvm   /
  `-debianlvm--vg-swap_1  254:1    0 1020M  0 lvm   [SWAP]
sdb                         8:16   0  144G  0 disk
`-mpatha                  254:2    0  144G  0 mpath
sdc                         8:32   0  144G  0 disk
`-mpatha                  254:2    0  144G  0 mpath
sr0                        11:0    1 1024M  0 rom
root@server3:~#
```

54.7 Verifying the mpath device

```
root@server3:~# multipath -ll
mpatha (3600140527c48bce7f774074a88f6c4b9) dm-2 LIO-ORG,iSCSIpro42
size=144G features=2 queue_mode mq hwhandler=1 alua wp=rw
|-+- policy=service-time 0 prio=50 status=active
| `- 3:0:0:0 sdb 8:16 active ready running
`-+- policy=service-time 0 prio=50 status=enabled
  `- 4:0:0:0 sdc 8:32 active ready running
root@server3:~#
```

54.8 Using the mpath device

You can use the **/dev/mapper/maptha** device just like any other block device. In the example below we create one partition on it and an **ext4** filesystem on that partition. And we mount it on our favorite mount point.

```
root@server3:~# parted /dev/mapper/mpatha mklabel msdos mkpart primary 0 50%
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? I
Information: You may need to update /etc/fstab.

root@server3:~# mkfs.ext4 /dev/mapper/mpatha-part1
<output truncated>
root@server3:~# mkdir /srv/project42
root@server3:~# mount /dev/mapper/mpatha-part1 /srv/project42/
root@server3:~# df -h | grep project42
/dev/mapper/mpatha-part1         71G   53M   67G   1% /srv/project42
root@server3:~#
```

54.9 Creating an alias for a WWN

You can copy the WWN from the output of **multipath -l** and assign it a user friendly alias. In this example we use **project42** as an alias for our multipath.

```
root@server3:~# cat /etc/multipath.conf
defaults {
        user_friendly_names yes
        find_multipaths yes
```

```
}

multipaths {
        multipath {
                wwid 3600140527c48bce7f774074a88f6c4b9
                alias project42
                }
}
root@server3:~#
```

Tip
Remember that you can do **Esc :r !multipath -l** from within vi.

Just restart the service and the new alias will be active.

```
root@server3:~# multipath -l
project42 (3600140527c48bce7f774074a88f6c4b9) dm-2 LIO-ORG,iSCSIpro42
size=144G features=2 queue_mode mq hwhandler=1 alua wp=rw
|-+- policy=service-time 0 prio=0 status=active
| `- 3:0:0:0 sdb 8:16 active undef running
`-+- policy=service-time 0 prio=0 status=enabled
  `- 4:0:0:0 sdc 8:32 active undef running
root@server3:~#
```

Even **lsblk** is up-to-date on the structure and names of our multipath device. Notice that the name of the first partition ends in **-part1**.

```
root@server3:~# lsblk | tail -9
sdb                       8:16   0  144G  0 disk
|-sdb1                    8:17   0   72G  0 part
`-project42             254:2    0  144G  0 mpath
  `-project42-part1     254:3    0   72G  0 part
sdc                       8:32   0  144G  0 disk
|-sdc1                    8:33   0   72G  0 part
`-project42             254:2    0  144G  0 mpath
  `-project42-part1     254:3    0   72G  0 part
sr0                      11:0    1 1024M  0 rom
root@server3:~#
```

54.10 Cheat sheet

Table 54.1: Multipath

command	explanation
apt-get install multipath-tools	Install the multipath software.
/etc/multipath.conf	Contains the multipath configuration.
systemctl restart multipathd	(Re)starts the multipath software.
lsblk	Displays block devices, including multipath devices.
multipath -ll	Displays details about multipath devices.

54.11 Practice

1. Add a network card to two of your servers. Create two subnets and place one network card of each server in each subnet. Test with **ping** that the connections work.
2. Create a new **iSCSI target** (or use the one from the previous chapter).
3. Verify on the **iSCSI initiator** that you can access the **iSCSI target** on both network interfaces.
4. Login to the **iSCSI target** via both subnets.
5. Install **multipath**, create **multipath.conf** and restart the service.
6. Verify with **lsblk** and with **multipath -ll** that a new **mpatha** device is created.
7. Create four new Debian 10 servers with six extra hard disks each. Create an mdadm RAID5 over the six disks. Share this **md** device with iSCSI. Create three subnets on each server. Use multipath over the twelve networks to connect to all the RAID5 devices. You should receive four. Add all four devices to an **LVM Volume group** and create a logical volume. Put ZFS on the logical volume. Just kidding, though it would be a fun exercise ;-)

54.12 Solution

1. Add a network card to two of your servers. Create two subnets and place one network card of each server in each subnet. Test with **ping** that the connections work.

```
root@server3:~# ping -c1 10.0.2.102
PING 10.0.2.102 (10.0.2.102) 56(84) bytes of data.
64 bytes from 10.0.2.102: icmp_seq=1 ttl=64 time=0.696 ms

--- 10.0.2.102 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.696/0.696/0.696/0.000 ms
root@server3:~# ping -c1 10.0.102.102
PING 10.0.102.102 (10.0.102.102) 56(84) bytes of data.
64 bytes from 10.0.102.102: icmp_seq=1 ttl=64 time=0.651 ms

--- 10.0.102.102 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.651/0.651/0.651/0.000 ms
root@server3:~#
```

2. Create a new **iSCSI target** (or use the one from the previous chapter).

```
targetcli
```

3. Verify on the **iSCSI initiator** that you can access the **iSCSI target** on both network interfaces.

```
iscsiadm -m discovery -t st -p 10.0.2.102
iscsiadm -m discovery -t st -p 10.0.102.102
```

4. Login to the **iSCSI target** via both subnets.

```
iscsiadm -m node --targetname "iqn.2008-08.be.linux-training.server2:itargetpro42" -- ↩
    portal "10.0.2.2:3260" --login
iscsiadm -m node --targetname "iqn.2008-08.be.linux-training.server2:itargetpro42" -- ↩
    portal "10.0.2.102:3260" --login
```

5. Install **multipath**, create **multipath.conf** and restart the service.

```
apt-get install multipath
vi /etc/multipath.conf
systemctl restart multipathd
```

6. Verify with **lsblk** and with **multipath -ll** that a new **mpatha** device is created.

```
lsblk
multipath -ll
```

7. Create four new Debian 10 servers with six extra hard disks each. Create an mdadm RAID5 over the six disks. Share this **md** device with iSCSI. Create three subnets on each server. Use multipath over the twelve networks to connect to all the RAID5 devices. You should receive four. Add all four devices to an **LVM Volume group** and create a logical volume. Put ZFS on the logical volume. Just kidding, though it would be a fun exercise ;-)

Part VI

More Linux system management

Chapter 55

Booting Linux

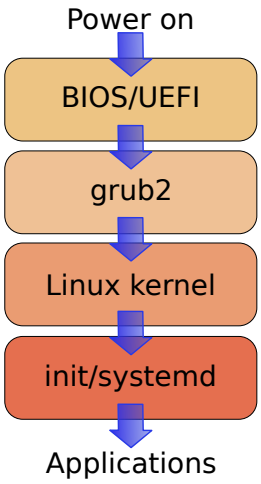
55.1 From power on to applications

When you power on a Debian Linux server then the first thing that will run is the BIOS or the UEFI firmware. Both are embedded software in the hardware. The goal of the BIOS or UEFI is to load a **boot loader**.

The default **boot loader** on Debian Linux is **grub2**, though others like **grub** and **lilo** are still supported. The goal of the **boot loader** is to load a **Linux kernel**.

The **Linux kernel** runs the computer, it is the core of the operating system. When the kernel has detected and configured the hardware it will start its first process named **init**.

We have seen in the processes chapter that **init** has PID 1, and that **init** is a symbolic link to **systemd**. Both will be discussed in a separate chapter. The goal of **init** or **systemd** is to start applications on the server.



55.2 BIOS

The BIOS, short for **Basic Input/Output System**, is since the 1970s the default firmware on Intel and compatible computers to test and configure hardware and to load a boot manager. In the BIOS there is a **boot order** defined which usually sets an **optical drive** as the first item to boot. Inserting a CD or DVD before power on will have the BIOS try to boot from this medium.

Since the mid 1990s you could press Del, F2 or F12 (this depends on the hardware) during power on to get you into an interactive shell of the BIOS. Chances are you never have to do this anymore, since most servers are virtual or (will) boot from UEFI.

55.3 UEFI

The UEFI, short for **Unified Extensible Firmware Interface**, replaces the BIOS on modern computers. UEFI can handle larger disks and more partitions than BIOS, thanks to the GPT support.

In Linux you can mount the UEFI partition and browse around in it. It is a FAT32 partition that holds **grub2** as an EFI application that is started automatically.

```
root@server6:~# fdisk -l | grep EFI
/dev/sda1      2048  1050623  1048576  512M EFI System
root@server6:~# mount /dev/sda1 /mnt
root@server6:~# mount | grep mnt
/dev/sda1 on /mnt type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=ascii ↩
    ,shortname=mixed,utf8,errors=remount-ro)
root@server6:~# cat /mnt/startup.nsh
\EFI\debian\grubx64.efi
root@server6:~#
```

More information on UEFI and Debian can be found here https://wiki.debian.org/UEFI .

55.4 BIOS or UEFI

You may ask whether there is a way to find out whether the server boots from BIOS or UEFI without rebooting it. You can, and it is easy. If **/sys/firmware/efi** exists then your server booted from UEFI.

```
root@server6:~# ls /sys/firmware/efi/
config_table  efivars  fw_platform_size  fw_vendor  runtime  runtime-map  systab  vars
root@server6:~#
```

55.5 grub2

55.5.1 grub2 and UEFI

As stated before, the job of the BIOS or UEFI is to load a boot loader. The default bootloader for Debian 10 is **grub2**. Debian 10 will install it's EFI bootloader in **\EFI\debian** . The name of the **grub2** EFI application depends on the system architecture.

Table 55.1: grub2 EFI application name

Architecture	Path and name
amd64	\EFI\debian\grubx64.efi
i386	\EFI\debian\grubia32.efi
arm64	\EFI\debian\grubaa64.efi
armhf	\EFI\debian\grubarm.efi

55.5.2 grub2 and BIOS

If you have a BIOS system, then **grub2** will install a stage loader in the **Master Boot Record** to load **grub2**. There is no dedicated partition to the bootloader when using the BIOS.

The **grub-install** program should only be run once, often at install time, to install grub in the MBR. You probably never have to run this.

55.5.3 Configuring grub2

The **grub v2** bootloader should be configured through its configuration file **/etc/default/grub** . For advanced configuration there are scripts in **/etc/grub.d**. The **update-grub** command will then generate **/boot/grub/grub.cfg** file. Do not edit this last file!

The **/etc/default/grub** file explains all its options. In the screenshot below we change the GRUB_TIMEOUT value to 2 and then we run **update-grub**.

```
root@server6:~# vi /etc/default/grub
root@server6:~# grep TIME /etc/default/grub
GRUB_TIMEOUT=2
root@server6:~# update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.19.0-5-amd64
Found initrd image: /boot/initrd.img-4.19.0-5-amd64
Adding boot menu entry for EFI firmware configuration
done
root@server6:~#
```

55.5.4 Changing the grub2 font

On modern laptops or desktops you often have 4K or higher resolutions and the grub font can be really tiny on those screens. You can increase the font size by supplying grub with a new font file.

```
root@debian10~# grub-mkfont -s 32 -o /boot/grub/deja.pf2 /usr/share/fonts/truetype/dejavu/ ↩
    DejaVuSansMono.ttf
root@debian10~# grep FONT /etc/default/grub
GRUB_FONT=/boot/grub/deja.pf2
root@debian10~# update-grub
<output truncated>
root@debian10~#
```

55.6 /boot

There are four files present in the **/boot** directory. The **vmlinuz** file is the compressed Linux kernel. This file gets loaded and uncompressed into RAM by the **grub2** boot loader. This is the core of your operating system, the Linux kernel manages the hardware and the software.

```
root@server6:~# ls -lh /boot/vmlinuz-4.19.0-5-amd64
-rw-r--r-- 1 root root 5.0M Aug  8 04:02 /boot/vmlinuz-4.19.0-5-amd64
root@server6:~#
```

There is also an **initrd.img** image file. This is a small collection of directories and files that the kernel needs before it can mount the **root** file system on **/**.

You can view the contents of this file by uncompressing it and extracting the files with **cpio** (discussed in the backup chapter).

```
root@server6:~/test# cp /boot/initrd.img-4.19.0-5-amd64 initrd.gz
root@server6:~/test# zcat initrd.gz | cpio -i
171869 blocks
root@server6:~/test# rm initrd.gz
root@server6:~/test# du -sh
87M     .
root@server6:~/test# ls
bin  conf  etc  init  lib  lib32  lib64  libx32  run  sbin  scripts  usr
root@server6:~/test#
root@server6:~/test#
```

There is also a **config** file in the **/boot** directory. This file contains the configuration that was used to compile this kernel. This can serve as a starting point when you want to compile a kernel yourself (which is rare in 2019).

```
root@server6:~# grep ^[^#] /boot/config-4.19.0-5-amd64 | head -6
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80300
CONFIG_CLANG_VERSION=0
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y
root@server6:~#
```

The last file is a **System.map** file for the kernel to look-up symbol names and their addresses in memory. This can be useful for debugging kernel panics.

```
root@server6:~# head -6 /boot/System.map-4.19.0-5-amd64
0000000000000000 D __per_cpu_start
0000000000000000 D irq_stack_union
00000000000001e0 A kexec_control_code_size
0000000000004000 D cpu_debug_store
```

```
0000000000005000 D cpu_tss_rw
0000000000008000 D gdt_page
root@server6:~#
```

55.7 Adding a grub2 menu entry

In case you *are* compiling a new kernel (or if you really want to dual-boot Linux), then you can use the **40_custom** script to add a menu entry. We add an entry here for a fictitious **kernel-6.0** .

```
root@server6:/etc/grub.d# vi 40_custom
root@server6:/etc/grub.d# tail -5 40_custom
# the 'exec tail' line above.
menuentry 'My Linux menu' {
        linux /boot/vmlinuz-6.0 root=/dev/sda2
        initrd /boot/initrd-6.0
}
root@server6:/etc/grub.d#
```

We need to copy the files to match the names we gave in our **menuentry** above. And we need to run **update-grub** to update the grub configuration. After this you can reboot and enjoy your custom **grub2** entry.

```
root@server6:/etc/grub.d# cd /boot
root@server6:/boot# cp vmlinuz-4.19.0-5-amd64 vmlinuz-6.0
root@server6:/boot# cp initrd.img-4.19.0-5-amd64 initrd-6.0
root@server6:/boot# update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.0
Found initrd image: /boot/initrd-6.0
Found linux image: /boot/vmlinuz-4.19.0-5-amd64
Found initrd image: /boot/initrd.img-4.19.0-5-amd64
Adding boot menu entry for EFI firmware configuration
done
root@server6:/boot#
```

55.8 Rescue mode

You can boot from the optical drive or USB key and choose **Rescue mode** in the **grub** menu. This will start a mini-Debian in RAM and enables a command prompt without root filesystem. You can use this to troubleshoot a broken system, and to reset the root password (if it is lost).

In the screenshot here we are in **rescue mode**. Some commands may work, others don't. We mount the **/dev/sda2** partition to **/mnt**, this gives us access to the root filesystem of the installation.

```
~ # pwd
/
~ # mount /dev/sda2 /mnt
~ # du -sh /
sh: du: not found
~ # /mnt/usr/bin/du -sh /
</proc errors truncated>
1.7G    /
~ #
```

55.9 chroot

While in this **rescue mode** we are running without a real root filesystem. So typing **passwd** to change the root password will not change anything on the hard disk. You could edit **/mnt/etc/shadow** but that is dangerous.

It is easier to use the **chroot** command to change the root directory. By typing **chroot /mnt** you can change to the root filesystem on the disk (the grub interface will also propose this by the way).

In the screenshot here we **chroot** to the hard disk, **du** now works and **passwd** will change the root password on the disk. Typing **exit** will exit the chrooted environment.

```
~ # chroot /mnt
# du -sh /
1.1G    /
# passwd
New password:
Retype new password:
passwd: password updated successfully
# exit
~ #
```

55.10 Cheat sheet

Table 55.2: Booting Linux

command	**explanation**
/sys/firmware/efi	Only exists on UEFI booted Debian computers.
grub-install	Installs **grub2** on the hard disk (at Debian install time).
update-grub	Updates the configuration of **grub2**.
grub-mkfont	Creates a font file for use with **grub2**.
/boot/vmlinuz-*	This is the compressed Linux kernel.
/boot/initrd-*	This is a mini file system for use at boot time.
/boot/config-*	This is the kernel configuration (at compile time).
/boot/System.map-*	This is the kernel symbol table (for debugging).
/etc/grub.d/	Contains **grub2** configuration.
chroot /foo	Change root file system to **/foo**.

55.11 Practice

1. Did your system boot from UEFI or BIOS?
2. Change the **grub2** timeout to 30 seconds.
3. Add a stanza to **grub2** to boot *your* Linux 7.0 kernel.
4. Use an optical drive or USB stick to boot in **rescue mode**. Take note of the options that **grub** presents.

55.12 Solution

1. Did your system boot from UEFI or BIOS?

```
ls /sys/firmware/efi && echo UEFI || echo BIOS
```

2. Change the **grub2** timeout to 30 seconds.

```
vi /etc/default/grub
update-grub
reboot
```

3. Add a stanza to **grub2** to boot *your* Linux 7.0 kernel.

```
vi /etc/grub.d/40_custom
cp /boot/vmlinuz* /boot/vmlinuz-7.0
cp /boot/initrd* /boot/initrd-7.0
update-grub
reboot
```

4. Use an optical drive or USB stick to boot in **rescue mode**. Take note of the options that **grub** presents.

Chapter 56

init

56.1 About System-V style init

The **init** system discussed in this chapter is (or rather was) since the 1980s a standard way to start several daemons on SysV Unix and derivatives. The **Linux** project was started in 1991 and adopted this **init** style.

The past couple of years most Linux distributions have migrated to **systemd** (this is the next chapter). Debian 10 is still backwards compatible with **services** that use **init** instead of **systemd**, so we will first discuss this **init** system.

The Linux kernel still starts **/sbin/init** as PID 1, but this is a link to **systemd**.

```
root@server2:~# ls -l /sbin/init
lrwxrwxrwx 1 root root 20 May 24 22:58 /sbin/init -> /lib/systemd/systemd
root@server2:~#
```

56.2 /etc/init.d

Services or daemons that still use **init** have a script in **/etc/init.d**. This script expects a **start** or **stop** as parameter to start or stop the service. Often keywords like **status** and **restart** are also supported.

Most services however are integrated in **systemd** even if they have a script here.

```
root@server2:~# <b>/etc/init.d/ssh status | head -2</b>
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
root@server2:~#
```

Typing **service ssh restart** used to just run the script in **/etc/init.d**, but is now captured by **systemctl** if the service is present in **systemd**.

56.3 runlevel

Typical for **init** style systems is that they are running in a **runlevel**. Runlevel 0 is shutdown, runlevel 6 is reboot, runlevel S and 1 are single user mode, runlevel 2 to 5 are multiuser modes. The two runlevel commands, **runlevel** and **who -r** still report this runlevel.

```
root@server2:~# runlevel
N 5
root@server2:~# who -r
         run-level 5  2019-09-04 22:50
root@server2:~#
```

56.4 /etc/rc?.d

To see which init scripts are run in **runlevel 5**, execute an **ls** in **/etc/rc5.d**. In that directory are symbolic links to **/etc/init.d**.

```
root@server2:~# ls -l /etc/rc5.d/ | head -6
total 0
lrwxrwxrwx 1 root root 17 Aug 22 17:42 S01anacron -> ../init.d/anacron
lrwxrwxrwx 1 root root 19 Aug 22 17:42 S01bluetooth -> ../init.d/bluetooth
lrwxrwxrwx 1 root root 26 Aug 22 17:37 S01console-setup.sh -> ../init.d/console-setup.sh
lrwxrwxrwx 1 root root 14 Aug 22 17:36 S01cron -> ../init.d/cron
lrwxrwxrwx 1 root root 14 Aug 22 17:42 S01dbus -> ../init.d/dbus
root@server2:~#
```

The **init runlevel** is replaced in **systemd** by a **systemd target**.

56.5 init 6 and reboot

Commands like **init 1** to go to single user mode or **init 6** to reboot are still supported. But remember that **/sbin/init** is a symbolic link to **systemd** so it is **systemd** that is performing these **runlevel** changes.

```
root@server2:~# file $(which init)
/usr/sbin/init: symbolic link to /lib/systemd/systemd
root@server2:~# init 6
Connection to 192.168.56.102 closed by remote host.
Connection to 192.168.56.102 closed.
```

The commands **reboot**, **shutdown** and **poweroff** are also performed by **systemd** now.

56.6 update-rc.d

The **update-rc.d** tool manages the runlevels in which daemons are started or stopped. It will do this by creating (and removing) links in the **/etc/rc?.d** directories.

```
root@server2:~# find /etc/rc?.d/  -name '*ssh' -exec ls -l {} \;
lrwxrwxrwx 1 root root 13 Aug 22 17:43 /etc/rc2.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Aug 22 17:43 /etc/rc3.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Aug 22 17:43 /etc/rc4.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Aug 22 17:43 /etc/rc5.d/S01ssh -> ../init.d/ssh
root@server2:~# update-rc.d ssh remove
root@server2:~# find /etc/rc?.d/  -name '*ssh' -exec ls -l {} \;
root@server2:~# update-rc.d ssh defaults
root@server2:~# find /etc/rc?.d/  -name '*ssh' -exec ls -l {} \;
lrwxrwxrwx 1 root root 13 Sep  5 11:41 /etc/rc2.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Sep  5 11:41 /etc/rc3.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Sep  5 11:41 /etc/rc4.d/S01ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 13 Sep  5 11:41 /etc/rc5.d/S01ssh -> ../init.d/ssh
root@server2:~#
```

Warning
Disabling ssh with **update-rc.d** has no effect anymore since the ssh service is still enabled via **systemd**, while enabling ssh with **update-rc.d** will also enable ssh with **systemd**.

Note
This chapter will be removed in future editions of this book.

56.7 Cheat sheet

Table 56.1: init

command	**explanation**
/sbin/init	The first process started by the kernel.
/etc/init.d	Legacy location for daemon/application startup scripts.
runlevel	Legacy command to display the current runlevel.
who -r	Legacy command to display the current runlevel.
init 6	Legacy **reboot** command.
update-rc.d	Legacy command to configure runlevels.

Chapter 57

systemd

57.1 About systemd

The **systemd software suite** replaces the **SysV Init system** on most modern Linux distributions and this includes Debian 10. We will see that **systemd** does a lot more than **init**.

```
root@debian10:~# systemd-analyze
Startup finished in 2.382s (kernel) + 489ms (userspace) = 2.871s
graphical.target reached after 480ms in userspace
root@debian10:~#
```

It's a bit strange that **systemd** reports **graphical.target reached** on a server without any graphical applications. It is nice though that you can **blame** the slow services from when the system started.

```
root@debian10:~# systemd-analyze blame | head -5
           261ms dev-sda1.device
           132ms systemd-timesyncd.service
           118ms networking.service
            77ms keyboard-setup.service
            66ms apparmor.service
root@debian10:~#
```

57.2 targets for runlevels

Instead of **runlevels** in **init**, there are now many **targets** in **systemd**. You can list all targets with the **systemctl list-units -t target -a** command. These target files are located in the **/usr/lib/systemd/system/** directory.

```
root@debian10:~# systemctl list-units -t target -a | head -6
UNIT                   LOAD   ACTIVE   SUB    DESCRIPTION
basic.target           loaded active   active Basic System
cryptsetup.target      loaded active   active Local Encrypted Volumes
emergency.target       loaded inactive dead   Emergency Mode
getty-pre.target       loaded inactive dead   Login Prompts (Pre)
getty.target           loaded active   active Login Prompts
root@debian10:~#
```

The default target can be displayed with **systemctl get-default**.

```
root@debian10:~# systemctl get-default
graphical.target
root@debian10:~#
```

57.3 dependencies

There were no real dependencies with the legacy **init** system, just an order for scripts to start. With **systemd** now a target can require one or more targets or services. The legacy runlevels, for example, all require a systemd target or service.

```
root@debian10:/usr/lib/systemd/system# grep 'Requires=' runlevel?.target
runlevel0.target:Requires=systemd-poweroff.service
runlevel1.target:Requires=sysinit.target rescue.service
runlevel2.target:Requires=basic.target
runlevel3.target:Requires=basic.target
runlevel4.target:Requires=basic.target
runlevel5.target:Requires=multi-user.target
runlevel6.target:Requires=systemd-reboot.service
root@debian10:/usr/lib/systemd/system#
```

Dependencies can also be defined in the **services** themselves. See for example the dependencies of the **ssh.service**.

```
root@debian10:/usr/lib/systemd/system# grep target ssh.service
After=network.target auditd.service
WantedBy=multi-user.target
root@debian10:/usr/lib/systemd/system#
```

All dependencies for a services can be listed with the **systemctl list-dependencies** command, followed by the service name. On modern terminals you get a Unicode colour output to quickly see whether a dependency is properly active.

```
root@debian10:~# systemctl list-dependencies ssh | head -6
ssh.service
* |--.mount
* |-system.slice
* `-sysinit.target
*   |-apparmor.service
*   |-dev-hugepages.mount
root@debian10:~#
```

57.4 systemctl enable

To **enable** a service, this means to have it automatically start at boot time and to have it stay active, you can issue the **systemctl enable** command followed by the service name. The service name can be abbreviated.

```
root@debian10:~# systemctl enable ssh.service
Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-sysv- ↩
    install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
root@debian10:~# systemctl enable ssh
Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-sysv- ↩
    install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
root@debian10:~#
```

Note that an **enable** command will not immediately start the service.

57.5 systemctl start

To **start** a service there is **systemctl start** followed by the name of the service. This will start the service immediately, and is independent of the **enabled** or **disabled** state of this service. The same is true when **stopping** a service.

```
root@debian10:~# systemctl stop ssh
root@debian10:~# systemctl start ssh
root@debian10:~#
```

Tip
Remember when there is no output after executing a command, then all went well.

57.6 systemctl status

The **systemctl status** command will give a lot of information about the status of a service, including the last logging messages for this service.

```
root@debian10:~# systemctl status ssh
* ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2019-09-05 13:41:22 CEST; 2h 37min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 1133 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 1134 (sshd)
    Tasks: 1 (limit: 1149)
   Memory: 2.8M
   CGroup: /system.slice/ssh.service
           `-1134 /usr/sbin/sshd -D

Sep 05 13:41:22 debian10 systemd[1]: Starting OpenBSD Secure Shell server...
Sep 05 13:41:22 debian10 sshd[1134]: Server listening on 0.0.0.0 port 22.
Sep 05 13:41:22 debian10 sshd[1134]: Server listening on :: port 22.
Sep 05 13:41:22 debian10 systemd[1]: Started OpenBSD Secure Shell server.
Sep 05 16:16:57 debian10 sshd[1318]: Accepted password for paul from 192.168.56.1 port 39
Sep 05 16:16:57 debian10 sshd[1318]: pam_unix(sshd:session): session opened for user paul
root@debian10:~#
```

57.7 systemctl poweroff

The SysV init commands **halt**, **poweroff**, **shutdown**, and **reboot** can still be used, but are handled by **systemd** now. Remember also that **reboot -h** does not print a help message ;-)

```
root@debian10:~# reboot -h
Connection to 192.168.56.101 closed by remote host.
Connection to 192.168.56.101 closed.
```

57.8 remote systemd

The **systemctl** tool can access remote hosts and execute **systemctl** commands there. You can also start, stop, enable and disable remote services. And of course you can also remotely reboot etcetera.

```
root@debian10:~# systemctl -H root@10.0.2.102 status ssh
root@10.0.2.102's password:
* ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2019-09-05 16:36:31 CEST; 49s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 551 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 559
    Tasks: 1 (limit: 2359)
   Memory: 5.8M
   CGroup: /system.slice/ssh.service
           `-559 /usr/sbin/sshd -D
root@debian10:~#
```

57.9 creating a service

This humble process of creating your own service explains a bit how the systemd service model works. First we create a script that logs some messages, you can put this script anywhere.

```
root@debian10:~# cat /etc/init.d/myservice.sh
echo Starting my service...

while true
do
        echo "The service still runs..."
        sleep 60
done
root@debian10:~#
```

Then we need to create a **.service** file in **/etc/systemd/system/**. Do not put your file in **/lib/systemd/system/**. Then we enable and start the service.

```
root@debian10:~# cat /etc/systemd/system/myservice.service
[Unit]
Description=My Service

[Service]
Type=simple
ExecStart=/bin/bash /etc/init.d/myservice.sh

[Install]
WantedBy=multi-user.target
root@debian10:~# systemctl enable myservice
Created symlink /etc/systemd/system/multi-user.target.wants/myservice.service -> /etc/ ↩
    systemd/system/myservice.service.
root@debian10:~# systemctl start myservice
root@debian10:~#
```

The last action is to check that the service is running properly.

```
root@debian10:~# systemctl status myservice
* myservice.service - My Service
   Loaded: loaded (/etc/systemd/system/myservice.service; enabled; vendor preset: enabled
   Active: active (running) since Thu 2019-09-05 17:21:11 CEST; 4s ago
 Main PID: 957 (bash)
    Tasks: 2 (limit: 1149)
   Memory: 720.0K
   CGroup: /system.slice/myservice.service
           |-957 /bin/bash /etc/init.d/myservice.sh
           `-958 sleep 60

Sep 05 17:21:11 debian10 systemd[1]: Started My Service.
Sep 05 17:21:11 debian10 bash[957]: Starting my service...
Sep 05 17:21:11 debian10 bash[957]: The service still runs...
root@debian10:~#
```

Stopping and disabling the service is trivial.

```
root@debian10:~# systemctl stop myservice
root@debian10:~# systemctl disable myservice
Removed /etc/systemd/system/multi-user.target.wants/myservice.service.
root@debian10:~#
```

57.10 Cheat sheet

Table 57.1: systemd

command	**explanation**
systemd-analyze blame	Display statistics about the last boot.
systemctl list-units -t target -a	Lists all systemd **targets**.
systemctl get-default	Name the default target.
systemctl list-dependencies foo	Show all dependencies for service **foo**.
systemctl enable foo	Enable the **foo** service.
systemctl disable foo	Disable the **foo** service.
systemctl start foo	Start the **foo** service.
systemctl stop foo	Stop the **foo** service.
systemctl status foo	Show the status of the service **foo**.
poweroff	Poweroff handled by systemd.
reboot	Reboot handled by systemd.

57.11 Practice

1. Display the top ten services that delayed the boot sequence.
2. List all **systemd** targets.
3. List all dependencies of the **cron** service.
4. List all services that are dependant on **cron**. (Hint: This is the reverse of the previous question.)
5. Stop the cron service. Check its status. Start the cron service again. And check its status again.
6. Query the status of the cron service on a remote server.
7. Create you own service, enable it and start it and check its status.

57.12 Solution

1. Display the top ten services that delayed the boot sequence.

```
systemd-analyze blame | head
```

2. List all **systemd** targets.

```
systemctl list-units -t target -a
```

3. List all dependencies of the **cron** service.

```
systemctl list-dependencies cron
```

4. List all services that are dependant on **cron**. (Hint: This is the reverse of the previous question.)

```
systemctl --reverse list-dependencies cron
```

5. Stop the cron service. Check its status. Start the cron service again. And check its status again.

```
systemctl stop cron
systemctl status cron
systemctl start cron
systemctl status cron
```

6. Query the status of the cron service on a remote server.

```
systemctl -H root@10.0.42.42 status cron   ## Use a relevant ip-address
```

7. Create you own service, enable it and start it and check its status.

```
See the last part of the theory.
```

Chapter 58

Scheduling

58.1 About scheduling

Historically the **at** and **batch** commands were used to schedule tasks. Today **cron** is the most used method of scheduling. And more recently the **systemd timer** has gained popularity. We will briefly take a look at **at** and then continue with a longer explanation of **cron** and finish with an example of a **systemd timer** .

58.2 at

The **at** command is no longer installed by default, but it is still available in the Debian repository. Install it with **apt-get install at** .

```
apt-get install at
```

The **at** command understands some common English words like **today**, **tomorrow**, **next week** and **teatime** and can be used to schedule tasks once in the future. Press **Ctrl-d** (as End Of Transmission) to end the **at** command.

```
paul@debian10:~$ at teatime tomorrow
warning: commands will be executed using /bin/sh
at> echo It is teatime.
at> <EOT>
job 1 at Sat Sep  7 16:00:00 2019
paul@debian10:~$
```

You can also type **at 7am next week** or **at 12h50 today**, and you can schedule a script to run in a **bash** shell, instead of just one command.

```
paul@debian10:~$ at 12h50 today
warning: commands will be executed using /bin/sh
at> bash /home/paul/backup.sh
at> <EOT>
job 6 at Fri Sep  6 12:50:00 2019
paul@debian10:~$
```

58.2.1 atq and atrm

You can view the **at** queue with the **atq** command, and you can remove items from it with the **atrm** command. The screenshot removes our teatime command.

```
paul@debian10:~$ atq
1       Sat Sep  7 16:00:00 2019 a paul
2       Fri Sep  6 12:50:00 2019 a paul
paul@debian10:~$ atrm 1
paul@debian10:~$ atq
2       Fri Sep  6 12:50:00 2019 a paul
paul@debian10:~$
```

58.2.2 at.allow and at.deny

The **/etc/at.allow** and **/etc/at.deny** files determine who can submit jobs using the **at** or **batch** commands. If **at.allow** exists then only usernames that are in it can submit jobs. If **at.allow** does not exist, then **at.deny** is checked. The **at.allow** file does not exist by default.

```
root@debian10:~# head -5 /etc/at.deny
alias
backup
```

```
bin
daemon
ftp
root@debian10:~#
```

58.2.3 batch

The **batch** command is part of the **at suite** of commands. A **batch** will run when the load on the server is less than 1.5. The load can be checked with the **uptime** command.

```
paul@debian10~$ uptime
 11:34:07 up 11 days, 15:07,  1 user,  load average: 0.12, 0.03, 0.01
paul@debian10~$ batch
warning: commands will be executed using /bin/sh
at> echo hello > testbatch.txt
at> <EOT>
job 1 at Fri Sep  6 11:34:00 2019
paul@debian10~$ cat testbatch.txt
hello
paul@debian10~$
```

58.3 cron

The **cron** daemon is installed and activated by default and is the most used program to schedule tasks on Debian Linux. The history of **cron** goes back to the 1970s (like many Linux programs it is older than Linux itself). Most of today's front-end backup tools will use **cron** as a back-end scheduler.

The **cron** daemon reads **cron tables** and executes scheduled commands. It is named after Chronos, the Greek god of time, or after Cronus the Greek titan. Both deities are depicted with a scythe.

```
paul@debian10:~$ pidof cron
410
paul@debian10:~$
```

58.3.1 cron.allow and cron.deny

The **/etc/cron.allow** and **/etc/cron.deny** files work the same way as the **at** files. If **cron.allow** exists, then you must be named in it. If **cron.allow** does not exist, which is the default, then **cron.deny** is checked.

The screenshot shows a command so that only **root** will be able to use **cron**. All other users are blocked from the scheduler. Note that **root** can always use **cron**.

```
root@debian10:~# echo root > /etc/cron.allow
root@debian10:~#
```

58.3.2 crontab

You need to use **crontab -e** to edit a **crontab** file. The files are located in **/var/spool/cron**, but do no edit these files directly. The **crontab -e** command will open your favorite editor as defined by the **VISUAL** or **EDITOR** variable.

```
paul@debian10:~$ tail -2 .bashrc
export VISUAL=vi
export EDITOR=vi
paul@debian10:~$ cronttab -e
```

```
You (paul) are not allowed to use this program (crontab)
See crontab(1) for more information
paul@debian10:~$
```

You get the message shown here above is **cron.allow** exists and you are not listed in it. Either remove that file or add your user account to it.

58.3.3 Using a crontab table

A **crontab** table has five time fields followed by the scheduled command. The five fields control the minutes, hours, days-of-the-month, month, and days-of-the-week the job has to run.

```
paul@debian10:~$ crontab -e
crontab: installing new crontab
paul@debian10:~$
```

You can list your own crontab with **crontab -l**. Below is a sample crontab with explanations of the timing. You can also replace the five fields with one of @yearly, @monthly, @weekly, @daily, @hourly or @reboot.

```
paul@debian10:~$ crontab -l
# m h  dom mon dow   command

0 5 * * 1  weekly.sh  # runs every Monday at five in the morning
10 6 * * 2 weekly2.sh # runs every Tuesday at 6h10 in the morning
*/10 * * * * often.sh # runs every ten minutes all days
0 20 1,15 * * biw.sh  # runs on the first and the fifteenth of the month at 8pm.
30 17 14 2 * val.sh   # runs every year on February 14th at 17h30.

paul@debian10:~$
```

With **crontab -r** you can remove your personal crontab. It will also restore the explanation of the file in a comment section.

```
paul@debian10:~$ crontab -r
paul@debian10:~$ crontab -l
no crontab for paul
paul@debian10:~$
```

58.3.4 /etc/cron.hourly daily weekly monthly

Some scripts should run every day, others every week or every hour or every month. Because this is so common there are directories created for this. When the **root** user copies a script in any of these directories, then the script will run at said interval.

```
paul@debian10:~$ ls -ld /etc/cron.*
drwxr-xr-x 2 root root 4096 Jul 24 18:14 /etc/cron.d
drwxr-xr-x 2 root root 4096 Sep  5 22:30 /etc/cron.daily
drwxr-xr-x 2 root root 4096 Jul 24 18:09 /etc/cron.hourly
drwxr-xr-x 2 root root 4096 Jul 24 18:14 /etc/cron.monthly
drwxr-xr-x 2 root root 4096 Jul 24 18:14 /etc/cron.weekly
paul@debian10:~$
```

Often scripts are put in those directories by applications that need a regular task to be performed. This screenshot shows some scripts on a default Debian 10 system in the **/etc/cron.daily** directory.

```
paul@debian10:~$ ls /etc/cron.daily/
0anacron    bsdmainutils  exim4-base  logrotate  passwd
apt-compat  dpkg          locate      man-db     popularity-contest
paul@debian10:~$
```

Scripts that require some other schedule besides the above can put an entry in **/etc/cron.d** with their custom schedule as in a regular **crontab** file.

58.3.5 /etc/crontab

When do the hourly, daily, weekly and monthly scripts run? This is answered in the **/etc/crontab** file, which you can customise. Note that the format of this file is a bit different than regular **crontab** files; there is a username just before the command to run. The cron jobs will run as this user.

```
root@debian10:~# tail -5 /etc/crontab
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron ↩
    .daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron ↩
    .weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron ↩
    .monthly )
#
root@debian10:~#
```

58.3.6 cron output

Output of **cron** jobs are mailed to the user. The user can read these mails with the **mail** command. You can prevent this output by using **>/dev/null 2>&1** after the commands, or by redirecting to a log file.

The output of **cron** can also be mailed to an e-mail address using the **MAILTO=** directive and a properly set up mail server. See the mail server chapter.

58.4 systemd timer

Some services, like **logrotate** for example have been migrated from **cron** to a **systemd timer** . You can list all **systemd timers** with **systemctl list-timers**.

```
root@debian10:~# systemctl list-timers | cut -b-90
NEXT                         LEFT          LAST                         PASSED    UNIT
Wed 2019-09-11 20:31:21 CEST  30min left    Wed 2019-09-11 19:31:50 CEST  29min ago anacro
Thu 2019-09-12 00:00:00 CEST  3h 58min left Wed 2019-09-11 12:14:40 CEST  7h ago    logrot
Thu 2019-09-12 00:00:00 CEST  3h 58min left Wed 2019-09-11 12:14:40 CEST  7h ago    man-db
Thu 2019-09-12 06:25:11 CEST  10h left      Wed 2019-09-11 12:14:40 CEST  7h ago    apt-da
Thu 2019-09-12 12:29:59 CEST  16h left      Wed 2019-09-11 12:29:59 CEST  7h ago    system
Thu 2019-09-12 13:51:23 CEST  17h left      Wed 2019-09-11 19:49:22 CEST  11min ago apt-da

6 timers listed.
Pass --all to see loaded but inactive timers, too.
root@debian10:~#
```

We can create a small timer for the **myservice.service** from the previous chapter. We can link the service and the timer by naming the file for the timer **myservice.timer**. The line that starts with **OnCalendar** has the format **Year-Month-Day hour:minutes:seconds** .

```
root@debian10:~# vi /etc/systemd/system/myservice.timer
root@debian10:~# cat /etc/systemd/system/myservice.timer
[Unit]
Description=Timer for My Service

[Timer]
OnCalendar=--* 19:55:00

[Install]
WantedBy=timers.target
root@debian10:~# systemctl enable myservice.timer
```

```
root@debian10:~# systemctl start myservice.timer
root@debian10:~#
```

This timer, when started will start the **myservice.service** at 19h55. We can verify the status with **systemctl status myservice.timer** and with **systemctl list-timers** .

```
root@debian10:~# systemctl status myservice.timer
* myservice.timer - Timer for My Service
   Loaded: loaded (/etc/systemd/system/myservice.timer; enabled; vendor preset: enabled)
   Active: active (waiting) since Wed 2019-09-11 19:50:46 CEST; 59s ago
  Trigger: Wed 2019-09-11 19:55:00 CEST; 3min 13s left

Sep 11 19:50:46 debian10 systemd[1]: Started Timer for My Service.
root@debian10:~# systemctl list-timers
NEXT                         LEFT          LAST                         PASSED       UN
Wed 2019-09-11 19:55:00 CEST  1min 55s left n/a                         n/a          my
<output truncated>
root@debian10:~#
```

The **myservice.service** started exactly on time, which we could watch with **watch systemctl status myservice.service** .

58.5 Cheat sheet

Table 58.1: Scheduling

command	**explanation**
apt-get install at	Install the **at** scheduler.
at teatime tomorrow	Schedule a task with **at** at 16:00h tomorrow.
atq	List all **at** tasks.
atrm 1	Remove the first **at** task.
/etc/at.allow	Lists users that are allowed to use **at**.
/etc/at.deny	Lists users that are not allowed to use **at**.
batch	Schedule a job when server load is below 1.5.
/etc/cron.allow	Lists users that are allowed to use **cron**.
/etc/cron.deny	Lists users that are not allowed to use **cron**.
crontab -e	Edit your **crontab** table.
crontab -l	Display your **crontab** table.
crontab -r	Remove your crontab table.
/etc/cron*	Location for system **crontab** tables and jobs.
systemctl list-timers	List systemd timers.

58.6 Practice

1. Install the **at** command and verify the contents of **at.deny**.
2. Deny all users the use of **at** and **cron**.
3. Schedule an **at** job that writes to a log file, at noon tomorrow. Verify that your job is in the **at queue**.
4. Remove your job from the **at** queue and then remove **at** from your system.
5. Make sure only root and paul can use **cron** .
6. Schedule a cron job (backup.sh) to run every half hour.
7. Schedule a cron job to run during weekdays at 22h00.
8. Remove your **crontab** and restore the help comments.
9. Read and understand the **/etc/cron.daily/passwd** .

58.7 Solution

1. Install the **at** command and verify the contents of **at.deny**.

```
apt-get install at
more /etc/at.deny
```

2. Deny all users the use of **at** and **cron**.

```
touch /etc/at.allow
touch /etc/cron.allow
```

3. Schedule an **at** job that writes to a log file, at noon tomorrow. Verify that your job is in the **at queue**.

```
root@debian10:~# at noon tomorrow
warning: commands will be executed using /bin/sh
at> echo hello > noonjob.log
at> <EOT>
job 7 at Sun Sep  8 12:00:00 2019
root@debian10:~# atq
7       Sun Sep  8 12:00:00 2019 a root
4       Fri Sep 13 07:00:00 2019 a paul
root@debian10:~#
```

4. Remove your job from the **at** queue and then remove **at** from your system.

```
atrm 7
apt-get remove at
```

5. Make sure only root and paul can use **cron** .

```
echo paul > /etc/cron.allow
```

6. Schedule a cron job (backup.sh) to run every half hour.

```
crontab -e
1,31 * * * * backup.sh
```

7. Schedule a cron job to run during weekdays at 22h00.

```
crontab -e
0 22 * * 1-5 backup.sh
```

8. Remove your **crontab** and restore the help comments.

```
crontab -r
```

9. Read and understand the **/etc/cron.daily/passwd** .

```
more /etc/cron.daily/passwd
```

Chapter 59

Login logging

59.1 local and remote logins

Local logins can be on a text based terminal that is directly attached to the Debian 10 server. This will show up as a **tty** connection. There are six of these **tty** terminals available on Debian 10 and you can reach them by typing **alt-F1** to **alt-F6** on the terminal keyboard. The process that is performing these logins on a **tty** is called **/bin/login**.

If you install Debian on a desktop or laptop computer then you will be presented with a graphical login on **tty7** . Using **Ctrl-alt-F1** to **Ctrl-alt-F6** enables you to switch to text based terminals. Typing **alt-F7** brings you back to the GUI.

Remote logins (via ssh or putty) will connect to a **pseudo terminal service** or **pts**. These connections are numbered starting from **/pts/0**.

Connecting to embedded devices with Debian 10 is often done via a serial cable to the **serial console**. This will connect to **/dev/ttyS0**.

59.2 who

The **who** command reads **/var/log/wtmp** and shows you the currently logged on users, their terminal, and the location (ip) from which they logged on.

```
paul@debian10:~$ who
root     tty1         2019-09-07 13:29
paul     pts/0        2019-09-07 15:17 (192.168.1.28)
paul@debian10:~$
```

If you have a Debian 10 computer with a graphical login, then the logged on terminal will show as **tty7** and the origin as **:0** . You can reach **tty7** by typing **Ctrl-Alt-F7**, and likewise **tty1** can be reached by typing **Ctrl-Alt-F1** .

```
paul@debian10~$ who
paul     tty7         2019-09-07 11:13 (:0)
```

Typing **watch who** will constantly display who is logged on to the computer.

59.3 last

The **last** command will also read **/var/log/wtmp** to display the last successful logons to the system. It will also show the duration of the logon.

```
paul@debian10:~$ last | head -6
paul     pts/0        192.168.1.28     Sat Sep  7 16:14   still logged in
paul     pts/0        192.168.1.28     Sat Sep  7 15:17 - 15:18  (00:00)
paul     pts/0        192.168.1.28     Sat Sep  7 13:29 - 15:07  (01:37)
root     tty1                          Sat Sep  7 13:29   still logged in
reboot   system boot  4.19.0-5-amd64   Sat Sep  7 13:29   still running
paul     pts/0        192.168.56.1     Sat Sep  7 11:45 - 13:28  (01:43)
paul@debian10:~$
```

59.4 lastb

The **lastb** command reads from **/var/log/btmp** and shows you the last bad login attempts. The screenshot shows a server under a **brute force** attack, over 100 attempts per minute to log on as root.

```
root@debian10:/var/log# lastb | head
root     ssh:notty    35.0.127.52      Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    218.92.0.175     Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    35.0.127.52      Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    218.92.0.175     Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    218.92.0.175     Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    89.234.157.254   Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    218.92.0.175     Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    89.234.157.254   Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    89.234.157.254   Thu Aug 22 23:21 - 23:21  (00:00)
root     ssh:notty    218.92.0.175     Thu Aug 22 23:21 - 23:21  (00:00)
root@debian10:/var/log#
```

Logging of **lastb** can be disabled by removing the **/var/log/btmp** file. This is done because people sometimes type their password instead of their userid, and a file containing clear text passwords is never a good idea.

59.5 lastlog

The **lastlog** command will read the **/var/log/lastlog** file and display the most recent logons of all registered users. You may wonder why some users have an account on this system (or maybe you see a laid off user still logging on?).

```
paul@debian10:~$ lastlog | tail
paul             pts/0    192.168.1.28     Sat Sep  7 16:14:46 +0200 2019
systemd-coredump                           **Never logged in**
geert                                      **Never logged in**
david                                      **Never logged in**
linda                                      **Never logged in**
annik                                      **Never logged in**
laura                                      **Never logged in**
tania                                      **Never logged in**
valentina                                  **Never logged in**
Debian-exim                                **Never logged in**
paul@debian10:~$
```

59.6 ssh

When using **ssh** (or putty) to log on remotely to a Debian 10 Linux computer, then your login (attempt) is registered in the **/var/log/auth.log** file.

```
root@debian10:~# tail /var/log/auth.log | grep ssh
Sep  7 16:14:46 debian10 sshd[2170]: Accepted password for paul from 192.168.1.28 port  ↩
    57952 ssh2
Sep  7 16:14:46 debian10 sshd[2170]: pam_unix(sshd:session): session opened for user paul  ↩
    by (uid=0)
root@debian10:~#
```

59.7 su

When switching user account with **su** or **su -**, then this is also logged in the **/var/log/auth.log** file. Failed attempts with **su** are likewise logged in this file.

```
root@debian10:~# tail /var/log/auth.log | grep 'su:'
Sep  7 16:25:59 debian10 su: (to root) paul on pts/0
```

```
Sep  7 16:25:59 debian10 su: pam_unix(su-l:session): session opened for user root by paul( ↩
    uid=1000)
root@debian10:~#
```

59.8 loginctl

The **loginctl** utility, which is part of **systemd**, displays logged on users including a new **seat** field. I guess this is useful on **multi-seat** computers.

```
root@debian10:~# loginctl
SESSION  UID USER SEAT  TTY
      1    0 root seat0 tty1
     19 1000 paul       pts/0

2 sessions listed.
root@debian10:~#
```

There are many arguments that you can give to **loginctl** (see the man page). The **user-state** argument can be useful to see a (process) tree concerning your user (for example the fact the user **paul** did an **su -** to root).

```
root@debian10:~# loginctl user-status | head -18
paul (1000)
           Since: Sat 2019-09-07 17:36:53 CEST; 12s ago
           State: active
        Sessions: *19
          Linger: no
            Unit: user-1000.slice
                  |-session-19.scope
                  | |-2495 sshd: paul [priv]
                  | |-2508 sshd: paul@pts/0
                  | |-2509 -bash
                  | |-2512 su -
                  | |-2513 -bash
                  | |-2516 loginctl user-status
                  | `-2517 head -19
                  `-user@1000.service
                    `-init.scope
                      |-2498 /lib/systemd/systemd --user
                      `-2499 (sd-pam)
root@debian10:~#
```

59.9 getent

You can query for user information in the **passwd** and **shadow** files with the **getent** command. The screenshot shows a query in the **passwd** file, but as **root** you can also type **getent shadow paul** .

```
root@debian10:~# getent passwd paul
paul:x:1000:1000:Paul Cobbaut,,,:/home/paul:/bin/bash
root@debian10:~#
```

59.10 PAM

PAM is short for **Pluggable Authentication Module** and is responsible for a high-level API to low-level authentication schemes. PAM can be configured via **/etc/pam.conf** and the files in **/etc/pam.d/**, but in my 20-plus years of Linux consulting I never needed to change PAM configuration. Sometimes stuff just works.

```
root@debian10:~# ls /etc/pam.*
/etc/pam.conf

/etc/pam.d:
atd       common-account   common-session-noninteractive  other      sshd
chfn      common-auth      cron                           passwd     su
chpasswd  common-password  login                          runuser    su-l
chsh      common-session   newusers                       runuser-l  systemd-user
root@debian10:~#
```

59.11 Cheat sheet

Table 59.1: Login logging

command	**explanation**
who	Display a list of logged on users.
last	Display the last successful logons on this computer.
lastb	Display the last failed logon attempts.
touch /var/log/btmp	Enable logging of failed logon attempts.
lastlog	Display the last login of every user account.
/var/log/auth.log	Contains authentication messages.
loginctl	The **systemd** command to display logged on users.
getent	Get information about a user in a database.
getent passwd paul	Get information about user **paul** in the **passwd** file.

59.12 Practice

1. If you have local access to a Debian computer, then login on **tty1** and **tty3** and issue a **who** command.
2. Display the last ten successful logins.
3. Display the last ten failed logins.
4. Stop the logging of bad login attempts.
5. Display the last login of the **david** user.
6. Log in with **ssh** to your computer and show the logging of this login.
7. Look at the man page of **loginctl** and explore the many options.

59.13 Solution

1. If you have local access to a Debian computer, then login on **tty1** and **tty3** and issue a **who** command.

```
alt-F1            # login here
alt-F3            # also login here
who
```

   Or in case of a laptop or desktop

```
Ctrl-alt-F1       # login here
alt-F3            # also login here
who
alt-F7            # back to the GUI
```

2. Display the last ten successful logins.

```
last | head
```

3. Display the last ten failed logins.

```
lastb | head
```

4. Stop the logging of bad login attempts.

```
rm /var/log/btmp
```

5. Display the last login of the **david** user.

```
lastlog | grep david
```

6. Log in with **ssh** to your computer and show the logging of this login.

```
ssh user@ip
tail /var/log/auth.log | grep ssh
```

7. Look at the man page of **loginctl** and explore the many options.

```
man loginctl
```

Chapter 60

Logging

60.1 About rsyslog

The **r**ocket-fast **sys**tem for **log** processing or **rsyslog** has replaced the legacy **syslog** server since Debian 5 Lenny. It allows for local and secure remote logging from a lot of different sources.

```
root@debian10:~# pgrep -a rsyslog
387 /usr/sbin/rsyslogd -n -iNONE
root@debian10:~#
```

60.2 /etc/rsyslog.conf

Configuration of **rsyslogd** happens in **/etc/rsyslog.conf** and in the **/etc/rsyslog.d** directories. The **rsyslog.conf** file is backwards compatible with the legacy **syslog.conf** in that this configuration can be copied to the end of the **rsyslog.conf** file.

```
root@debian10:~# wc -l /etc/rsyslog.conf
92 /etc/rsyslog.conf
root@debian10:~# ls /etc/rsyslog.d/
root@debian10:~#
```

60.3 modules in rsyslog

Many features of **rsyslog** are provided by **modules**. The first part of **rsyslog.conf** loads some of these modules, as can be seen in this screenshot.

```
root@debian10:~# head -21 /etc/rsyslog.conf | tail -11
module(load="imuxsock") # provides support for local system logging
module(load="imklog")   # provides kernel logging support
#module(load="immark")  # provides --MARK-- message capability

# provides UDP syslog reception
#module(load="imudp")
#input(type="imudp" port="514")

# provides TCP syslog reception
#module(load="imtcp")
#input(type="imtcp" port="514")
root@debian10:~#
```

Module names that start with **im** are **input modules**. The five examples in the screenshot above are all input modules. Likewise modules that start with **om** are **output modules**, for example **ommysql** will output logging to a **mysql** database. We will set this up in the **mysql** chapter.

The **-r** option of the legacy **syslog** daemon is replaced by the **imudp** module, as can be seen in the screenshot above. Uncomment these lines instead of using **-r** .

60.4 facility, priority, action

The rest of the **rsyslog.conf** file is identical to the legacy **syslog.conf** file in that it expects a **facility**, a **priority** and an **action** on each line.

The **facility** is the source of the message and can be one of the following keywords: auth, authpriv, cron, daemon, kern, lpr, mail, mark, news, syslog, user, uucp and local0 through local7. The **mark** keyword is for internal **rsyslog** use only.

The **priority** is the severity of the message and in ascending order consists of the following keywords: debug, info, notice, warn(ing), err(or), crit, alert, emerg (=panic).

The **facility** and **priority** are separated by a dot and form the **selector field** together. An **asterisk** stands for all facilities or all priorities. The **selector field** is seperated from the **action** with tabs or spaces.

The **action** defines whether to send the message to: a file, a named pipe, a terminal or the console, a remote machine, a list of users, everyone who is logged on, a database table, or to discard the message.

Here are some examples of **facility.priority action** .

```
kern.*     -/var/log/kern.log   ## send all messages from kernel to kern.log
mail.warn  -/var/log/mail.warn  ## all warning and worse messages to mail.warn
mail.err   /var/log/mail.err    ## all error and worse messages to mail.err
```

The line in this next screenshot is to send all info, notice and warning messages, but none from auth, authpriv, cron, daemon, mail, and news to **/var/log/messages** .

```
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

And this last line is to send all **emergency** (=the most severe priority) messages to all logged on users. The list of users is preceded by **:omusrmsg:** , which is the output module for a list of users. The asterisk denotes all logged on users.

```
*.emerg                         :omusrmsg:*
```

60.5 local0 to local7

There are eight facilities that you can customise for your own purpose. They are **local0** to **local7** . You can use them for applications or for network devices like routers and switches. Consider for example these three lines.

```
root@debian10:~# vi /etc/rsyslog.conf
root@debian10:~# grep local4 /etc/rsyslog.conf
local4.warn     /var/log/local4.warn_and_above.log
local4.=warn    /var/log/local4.warn.log
local4.=crit    /var/log/local4.crit.log
root@debian10:~# systemctl restart rsyslog
root@debian10:~# systemctl status rsyslog
* rsyslog.service - System Logging Service
   Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-09-11 13:32:19 CEST; 10s ago
<output truncated>
root@debian10:~#
```

60.5.1 logger

The **logger** tool allows you to test your **rsyslog** configuration. We will test our three **local4** lines by sending one **warning** message and one **critical** message.

```
root@debian10:~# logger -p "local4.crit" critical
root@debian10:~# logger -p "local4.warn" warning
root@debian10:~# cat /var/log/local4.crit.log
Sep 11 13:38:27 debian10 paul: critical
root@debian10:~# cat /var/log/local4.warn.log
Sep 11 13:38:37 debian10 paul: warning
root@debian10:~# cat /var/log/local4.warn_and_above.log
Sep 11 13:38:27 debian10 paul: critical
Sep 11 13:38:37 debian10 paul: warning
root@debian10:~#
```

60.6 tail -f

The **tail -f** command can be used to display the last ten lines of a log file, and to keep displaying lines that are added to the logfile in real time. You can use this to immediately see the result of your **logger** command.

60.7 logrotate

Log files tend to grow, and grow. To prevent them taking up all your disk space there is a **logrotate** command that will rotate (and compress) log files. Its configuration file is **/etc/logrotate.conf** and the files in **/etc/logrotate.d/** .

```
root@debian10:~# head -6 /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4
root@debian10:~#
```

The **logrotate** program still has a file in **/etc/cron.daily/**, but this exits in favour of the **systemd** timer.

```
root@debian10:~# systemctl list-timers | grep logrotate
Thu 2019-09-12 00:00:00 CEST  9h left       Wed 2019-09-11 12:14:40 CEST  2h 43min ago  ↩
    logrotate.timer              logrotate.service
root@debian10:~#
```

60.8 Remote logging

With **rsyslogd** you can send log files to other **rsyslogd** servers. In our example here we configure **server2** to send the **local3** facility logs to the server named **debian10** at ip-address **10.0.2.101**. The screenshot below shows the configuration of **server2**.

```
root@server2:~# vi /etc/rsyslog.conf
root@server2:~# tail -1 /etc/rsyslog.conf
local3.*                        @10.0.2.101
root@server2:~# systemctl restart rsyslog.service
root@server2:~#
```

And here is the configuration on the server named **debian10**. We uncomment the two lines under **provides UDP syslog reception** and create an entry for **local3** .

```
root@debian10:~# vi /etc/rsyslog.conf
root@debian10:~# grep imudp /etc/rsyslog.conf
module(load="imudp")
input(type="imudp" port="514")
root@debian10:~# tail -1 /etc/rsyslog.conf
local3.*        /var/log/from_server2.log
root@debian10:~# systemctl restart rsyslog
root@debian10:~#
```

You can verify that **rsyslog** is listening on port 514 with **netstat** (apt-get install net-tools) or **ss**, as shown in this screenshot.

```
root@debian10:~# ss -anpu | grep 514
UNCONN    0         0                  0.0.0.0:514              0.0.0.0:*        users:(("  ↩
    rsyslogd",pid=1551,fd=6))
UNCONN    0         0                     [::]:514                 [::]:*        users:(("  ↩
    rsyslogd",pid=1551,fd=7))
```

```
root@debian10:~#
root@debian10:~# netstat -nlup | grep 514
udp        0      0 0.0.0.0:514           0.0.0.0:*                           1551/rsyslogd
udp6       0      0 :::514                :::*                                1551/rsyslogd
root@debian10:~#
```

When this is done, then we can use the **logger** tool to generate a **local3** message on **server2**. This message will be transported over the network over port UDP 514.

```
root@server2:~# logger -p "local3.crit" 'critical from server2'
root@server2:~#
```

Now we can verify that the message is received on the server named **debian10** .

```
root@debian10:~# tail /var/log/from_server2.log
Sep 11 16:03:38 server2 paul: critical from server2
root@debian10:~#
```

60.9 Logging by hostname

The previous section works, but what if you have many **rsyslogd** servers sending messages to a central **syslogd** server. Is there a way to separate the logs by hostname? Of course there is, by using a **DynaFile** template on the receiving server, as the screenshot shows.

```
root@debian10:~# grep -n DynaFile /etc/rsyslog.conf
53:$template DynaFile,"/var/log/%HOSTNAME%.log"
99:local3.*     -?DynaFile
root@debian10:~# systemctl restart rsyslog
root@debian10:~#
```

There is no extra configuration on the sending servers. We test the template use by sending a critical message from **server2** and a warning message from **server3** . Two separate logfiles are created on the server named **debian10** .

```
root@debian10:~# cat /var/log/server2.log
Sep 11 16:31:11 server2 paul: critical from server2
root@debian10:~# cat /var/log/server3.log
Sep 11 16:33:41 server3 paul: warning from server3
root@debian10:~#
```

60.10 Cheat sheet

Table 60.1: Logging

command	explanation
/usr/sbin/rsyslogd	This is the **rsyslog** daemon.
/etc/rsyslog.conf	This is the **rsyslog** configuration file.
logger	A tool to test **rsyslog** configuration (or to log messages).
tail -f	Live following of log files.
logrotate	A tool to rotate log files.
/etc/logrotate.conf	The configuration file for **logrotate**.
/etc/lograte.d/	A directory for **logrotate** configurations.

60.11 Practice

1. Verify that **rsyslogd** is running.
2. Open the **rsyslog** configuration file and look at the default modules, and understand them (by reading the manual if necessary).
3. Add a line for **local4.warning** writing to the **/var/log/l4.log** file, and test with **logger** that it works.
4. Use the **tail -f** command on your logfile and test with **logger** (from a second terminal).
5. Set up logging from one server to another for **local2**. Test with the **logger** tool.
6. Set up logging by hostname with a **DynaFile** template.

60.12 Solution

1. Verify that **rsyslogd** is running.

```
ps fax | grep rsyslogd
systemctl status rsyslog
```

2. Open the **rsyslog** configuration file and look at the default modules, and understand them (by reading the manual if necessary).

```
more /etc/rsyslog.conf
man rsyslog.conf
```

3. Add a line for **local4.warning** writing to the **/var/log/l4.log** file, and test with **logger** that it works.

```
echo "local4.warning   /var/log/l4.log" >> /etc/rsyslog.conf
systemctl restart rsyslog
logger -p "local4.warning" "test my log file"
cat /var/log/l4.log
```

4. Use the **tail -f** command on your logfile and test with **logger** (from a second terminal).

```
tail -f /var/log/l4.log          ## On one terminal
logger -p "local4.crit" test     ## On another terminal
```

5. Set up logging from one server to another for **local2**. Test with the **logger** tool.

```
local2.*    @10.0.2.101                       ## on server1 to ip-address server2
systemctl restart rsyslog

module(load="imudp")                          ## on server2
input(type="imudp" port="514")
local2.*    /var/log/from_server1.log
systemctl restart rsyslog

logger -p "local2.crit" "test from server1"    ## on server1

tail /var/log/from_server1.log                 ## on server2
```

6. Set up logging by hostname with a **DynaFile** template.

```
See the last part of the theory.
```

Chapter 61

systemd logging

61.1 About systemd journal

The executable that collects all messages and stores them in a binary format is **systemd-journald**. The **systemd journal** has the same functionality as **rsyslog** and may replace it in the future.

```
root@debian10:~# systemctl status systemd-journald | head -3 | cut -b-84
* systemd-journald.service - Journal Service
   Loaded: loaded (/lib/systemd/system/systemd-journald.service; static; vendor pres
   Active: active (running) since Thu 2019-09-12 12:37:56 CEST; 6h ago
root@debian10:~# ps fax | grep journald
  240?        Ss     0:00 /lib/systemd/systemd-journald
 1266 pts/0    S+     0:00                      \_ grep journald
root@debian10:~#
```

61.2 Reading journals

The command to read the **systemd journal** is **journalctl**. You can use commands like **grep** after it to filter the log, or you can use one of the many arguments that **journalctl** supports.

```
root@debian10:~# journalctl | head -4 | cut -b-84
-- Logs begin at Thu 2019-09-12 10:43:22 CEST, end at Thu 2019-09-12 10:44:15 CEST.
Sep 12 10:43:22 debian10 kernel: Linux version 4.19.0-6-amd64 (debian-kernel@lists.d
Sep 12 10:43:22 debian10 kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-4.19.0-6-amd
Sep 12 10:43:22 debian10 kernel: x86/fpu: Supporting XSAVE feature 0x001: 'x87 float
root@debian10:~#
```

61.2.1 Boot messages

The **journalctl -b** command will display messages from this boot, and is identical to **journalctl -b 0** . To see messages from the previous boot type **journalctl -b 1** and from the boot before that it is **journalctl -b 2** and so on. By default on Debian the **journalctl** logs are volatile so unless you set the logs to be persistent you will be greeted by the same message as this screenshot.

```
root@debian10:~# journalctl -b -1
Specifying boot ID or boot offset has no effect, no persistent journal was found.
root@debian10:~#
```

61.2.2 Recent messages

You can query the **systemd journal** to display recent messages, like this screenshot shows.

```
root@debian10:~# journalctl --since "10 min ago" | cut -b-84
-- Logs begin at Thu 2019-09-12 11:41:24 CEST, end at Thu 2019-09-12 12:52:51 CEST.
Sep 12 12:46:40 debian10 dhclient[366]: DHCPREQUEST for 192.168.56.101 on enp0s3 to
Sep 12 12:46:40 debian10 dhclient[366]: DHCPACK of 192.168.56.101 from 192.168.56.10
Sep 12 12:46:40 debian10 dhclient[366]: bound to 192.168.56.101 -- renewal in 581 se
Sep 12 12:52:51 debian10 systemd[1]: Starting Cleanup of Temporary Directories...
Sep 12 12:52:51 debian10 systemd[1]: systemd-tmpfiles-clean.service: Succeeded.
Sep 12 12:52:51 debian10 systemd[1]: Started Cleanup of Temporary Directories.
root@debian10:~#
```

61.2.3 Messages since a timestamp

You can query the **systemd journal** for all messages since a certain timestamp, as this screenshot shows.

```
root@debian10:~# journalctl --since "2019-09-12 19:00:00" | cut -b-84
-- Logs begin at Thu 2019-09-12 12:37:38 CEST, end at Thu 2019-09-12 19:09:32 CEST.
Sep 12 19:01:06 debian10 dhclient[366]: DHCPREQUEST for 192.168.56.101 on enp0s3 to
Sep 12 19:01:06 debian10 dhclient[366]: DHCPACK of 192.168.56.101 from 192.168.56.10
Sep 12 19:01:06 debian10 dhclient[366]: bound to 192.168.56.101 -- renewal in 506 se
Sep 12 19:09:32 debian10 dhclient[366]: DHCPREQUEST for 192.168.56.101 on enp0s3 to
Sep 12 19:09:32 debian10 dhclient[366]: DHCPACK of 192.168.56.101 from 192.168.56.10
Sep 12 19:09:32 debian10 dhclient[366]: bound to 192.168.56.101 -- renewal in 590 se
root@debian10:~#
```

61.2.4 Messages until a timestamp

And you can query messages that appeared before a certain timestamp with **--until** . The screenshot combines the **--since** and **--until** arguments.

```
root@debian10:~# journalctl --since "2019-09-12 19:08:00" --until "2019-09-12 19:12:00" |  ↩
    cut -b-84
-- Logs begin at Thu 2019-09-12 12:37:38 CEST, end at Thu 2019-09-12 19:09:32 CEST.
Sep 12 19:09:32 debian10 dhclient[366]: DHCPREQUEST for 192.168.56.101 on enp0s3 to
Sep 12 19:09:32 debian10 dhclient[366]: DHCPACK of 192.168.56.101 from 192.168.56.10
Sep 12 19:09:32 debian10 dhclient[366]: bound to 192.168.56.101 -- renewal in 590 se
root@debian10:~#
```

You can also use **--until "1 hour ago"** or **--since yesterday** .

61.2.5 Messages concerning a PID

You can query for all messages that concern a certain process by filtering by PID, as this screenshot shows.

```
root@debian10:~# journalctl _PID=490
-- Logs begin at Thu 2019-09-12 11:41:24 CEST, end at Thu 2019-09-12 18:00:52 CEST. --
Sep 12 12:37:56 debian10 sshd[490]: Server listening on 0.0.0.0 port 22.
Sep 12 12:37:56 debian10 sshd[490]: Server listening on :: port 22.
root@debian10:~#
```

61.2.6 Messages concerning a UID

You can filter messages bu UID (and combine this filter with all the previous arguments to **journalctl**). In this screenshot 1000 is the UID of the **paul** user account.

```
root@debian10:~# journalctl _UID=1000 | tail -2
Sep 12 12:38:05 debian10 su[746]: (to root) paul on pts/0
Sep 12 12:38:05 debian10 su[746]: pam_unix(su-l:session): session opened for user root by  ↩
    paul(uid=1000)
root@debian10:~#
```

You can also get a list of all UIDs that have an entry in the **systemd journal** using the **-F** option of **journalctl**.

```
root@debian10:~# journalctl -F _UID
1000
0
101
root@debian10:~#
```

61.2.7 Messages concerning a unit

You can filter messages to a **systemd unit**. In this screenshot this is combined with **-b** to only show messages from this boot concerning **ssh.service** .

```
root@debian10:~# journalctl -b -u ssh
-- Logs begin at Thu 2019-09-12 11:41:24 CEST, end at Thu 2019-09-12 18:00:52 CEST. --
Sep 12 12:37:56 debian10 systemd[1]: Starting OpenBSD Secure Shell server...
Sep 12 12:37:56 debian10 sshd[490]: Server listening on 0.0.0.0 port 22.
Sep 12 12:37:56 debian10 systemd[1]: Started OpenBSD Secure Shell server.
Sep 12 12:37:56 debian10 sshd[490]: Server listening on :: port 22.
Sep 12 12:38:01 debian10 sshd[730]: Accepted password for paul from 192.168.56.1 port 419
Sep 12 12:38:01 debian10 sshd[730]: pam_unix(sshd:session): session opened for user paul
root@debian10:~#
```

61.2.8 Messages concerning a binary

There is a quick way now to see all the messages concerning **su** (or rather **/usr/bin/su**), by giving the executable as an argument to **journalctl**.

```
root@debian10:~# journalctl -b /usr/bin/su  | cut -b-84
-- Logs begin at Thu 2019-09-12 12:37:38 CEST, end at Thu 2019-09-12 19:37:18 CEST.
Sep 12 12:38:05 debian10 su[746]: (to root) paul on pts/0
Sep 12 12:38:05 debian10 su[746]: pam_unix(su-l:session): session opened for user ro
root@debian10:~#
```

61.3 Persistent journal

The **systemd journal** can be made persistent by changing its configuration file **/etc/systemd/journald.conf** . Uncomment the **Storage=** and set it to **persistent**. After restarting the service there will be a **/var/log/journal** directory.

```
root@debian10:~# vi /etc/systemd/journald.conf
root@debian10:~# grep persistent /etc/systemd/journald.conf
Storage=persistent
root@debian10:~# systemctl restart systemd-journald
root@debian10:~# ls /var/log/journal/
0c7ce7595d4f4fcabe9b03cfac39a88c
root@debian10:~#
```

61.4 Priority level

The **systemd journal** uses the same priorities as **rsyslog** and **syslog**, which are defined in RFC 5424.

Table 61.1: syslog priorities

Code	Priority
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

61.4.1 Messages of certain severity

And you can filter by syslog priority (or severity) using the **journalctl -p** option followed by a priority, or followed by a range as in this screenshot.

```
root@debian10:~# journalctl -p err..emerg
-- Logs begin at Thu 2019-09-12 11:41:24 CEST, end at Thu 2019-09-12 18:00:52 CEST. --
-- No entries --
root@debian10:~#
```

61.5 Facilities

The **systemd journal** also uses the same facilities as **rsyslog** and **syslog**, which are defined in the same section of RFC 5424.

Table 61.2: syslog facilities

Code	keyword	Facility
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9	cron	clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12		NTP subsystem
13		log audit
14		log alert
15	cron	clock daemon (on other OS)
16	local0	local use 0 (local0)
17	local1	local use 1 (local1)
18	local2	local use 2 (local2)
19	local3	local use 3 (local3)
20	local4	local use 4 (local4)
21	local5	local use 5 (local5)
22	local6	local use 6 (local6)
23	local7	local use 7 (local7)

For example to query log entries concerning the **auth** facility use **journactl SYSLOG_FACILITY=4** as in this screenshot.

```
root@debian10:~# journalctl -b SYSLOG_FACILITY=4 | cut -b-84 | tail -5
Sep 12 12:37:56 debian10 systemd-logind[387]: Watching system buttons on /dev/input/
Sep 12 12:37:56 debian10 systemd-logind[387]: Watching system buttons on /dev/input/
Sep 12 12:38:01 debian10 sshd[730]: Accepted password for paul from 192.168.56.1 por
Sep 12 12:38:01 debian10 systemd-logind[387]: New session 1 of user paul.
Sep 12 12:38:05 debian10 su[746]: (to root) paul on pts/0
```

```
root@debian10:~#
```

You can also query syslog for a list of all facilities for which it has an entry in the **systemd journal** using the **-F** option of the **journalctl** command.

```
root@debian10:~# journalctl -F SYSLOG_FACILITY
4
3
10
9
root@debian10:~#
```

61.6 Cheat sheet

Table 61.3: Systemd logging

command	**explanation**
systemd-journald	This is the **systemd** logging daemon.
journalctl	Systemd tool to read log files.
journalctl -b	Read the **systemd** boot log.
journalctl -S "ten min ago"	Show only recent messages.
journalctl -S "2020-01-01 19:00:00"	Show messages since a certain date-time.
journalctl -U "2020-01-01 19:00:00"	Show messages until a certain date-time.
journalctl _PID=42	Show messages related to a PID.
journalctl _UID=42	Show messages related to a UID.
journalctl -u foo	Show messages related to a **systemd** unit.
journalctl SYSLOG_FACILITY=4	Show messages related to a **syslog** facility.

61.7 Practice

1. Is the **systemd-journald.service** active?
2. How many lines are there in the complete journal?
3. Make the **systemd journal** persistent.
4. Use **journalctl** to *list boots*.
5. Display all messages of the last twenty minutes.
6. Display all messages concerning your UID.

61.8 Solution

1. Is the **systemd-journald.service** active?

```
systemctl status systemd-journald
```

2. How many lines are there in the complete journal?

```
journalctl | wc -l
```

3. Make the **systemd journal** persistent.

```
vi /etc/systemd/journald.conf
Storage=persistent
systemctl restart systemd-journald
```

4. Use **journalctl** to *list boots*.

```
journalctl --list-boots
```

5. Display all messages of the last twenty minutes.

```
journalctl --since "20 min ago"
```

6. Display all messages concerning your UID.

```
journalctl _UID=1000
```

Chapter 62

Memory management

62.1 About memory

Back when computers were 32-bit every application received 2GB (in some cases 3 or 4GB) of virtual memory. Now with 64-bit computers applications get many terabytes of virtual memory to use. This is independent of how much physical memory is installed on the computer. Linux will map the virtual addresses to physical addresses.

Each application has its own virtual memory and cannot access the address space of another application.

62.1.1 physical memory

Information about physical memory installed on a computer can only be obtained on non-virtual machines. The screenshot here is from **dmidecode** running on a laptop (since all my servers are virtual). The laptop has two 8GB memory modules.

```
root@MBDebian~# dmidecode -t memory | grep -A1 Size
        Size: 8192 MB
        Form Factor: SODIMM
--
        Size: 8192 MB
        Form Factor: SODIMM
root@MBDebian~#
```

62.1.2 /proc/meminfo

Many tools will read memory information in **/proc/meminfo** and display it in a nicer format. In the screenshot you can see that this server named **debian10** has 4GB of RAM, of which most is unused.

```
root@debian10:~# grep Mem /proc/meminfo
MemTotal:        4041188 kB
MemFree:         3886672 kB
MemAvailable:    3811160 kB
root@debian10:~#
```

62.1.3 free

The **free** command here was run a bit later when the computer was using 651MB of memory. The **-m** option is used to display megabytes, you can use **-g** for gigabytes. This **free** command also gives a summary on swap space.

```
paul@debian10:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3946         651        3040           0         253        3084
Swap:          1021           0        1021
paul@debian10:~$
```

62.1.4 pmap

The **pmap** tool will give an overview of the memory map of a process. In this example we display the memory map of our **bash shell**. If you have a wide terminal then you can try -X or even -XX.

```
root@debian10:~# pmap -x $$ | head -20
806:   -bash
Address           Kbytes     RSS   Dirty Mode  Mapping
0000562e9f5a9000     180     180       0 r---- bash
0000562e9f5d6000     696     696       0 r-x-- bash
0000562e9f684000     216     108       0 r---- bash
0000562e9f6bb000      12      12      12 r---- bash
```

```
0000562e9f6be000      36      36      36 rw--- bash
0000562e9f6c7000      40      28      28 rw---   [ anon ]
0000562e9fa8a000    1704    1564    1564 rw---   [ anon ]
00007f7a18cf4000      24      24       0 r---- libpthread-2.28.so
00007f7a18cfa000      60      60       0 r-x-- libpthread-2.28.so
00007f7a18d09000      24       0       0 r---- libpthread-2.28.so
00007f7a18d0f000       4       4       4 r---- libpthread-2.28.so
00007f7a18d10000       4       4       4 rw--- libpthread-2.28.so
00007f7a18d11000      16       4       4 rw---   [ anon ]
00007f7a18d15000       8       8       0 r---- librt-2.28.so
00007f7a18d17000      16      16       0 r-x-- librt-2.28.so
00007f7a18d1b000       8       0       0 r---- librt-2.28.so
00007f7a18d1d000       4       4       4 r---- librt-2.28.so
00007f7a18d1e000       4       4       4 rw--- librt-2.28.so
root@debian10:~#
```

62.2 Swap space

A portion of the hard disk can be reserved as extra memory, this is called **swap space** and can be a partition or a regular file. Linux will use this **swap space** to swap pages of RAM to disk, to free up RAM for applications.

```
root@debian10:~# grep Swap /proc/meminfo
SwapCached:            0 kB
SwapTotal:       1046524 kB
SwapFree:        1046524 kB
root@debian10:~#
```

62.2.1 /proc/swaps

The **/proc/swaps** file will list the current **swap space**. In the screenshot below we see that **/dev/sda5** is a swap partition of 1GB size.

```
root@debian10:~# cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       1046524 0       -2
root@debian10:~#
```

62.2.2 Creating a swap partition

You can create extra **swap partitions** by first creating a partition and then running the **mkswap** command on said partition.

```
root@debian10:~# mkswap /dev/sdb1
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=255ca455-da29-4bac-a4b6-01c97e22cf5c
root@debian10:~#
```

62.2.3 swapon

You still need to add the **swap partition** to the **swap space**, after the **mkswap** command, with a **swapon** command followed by the partition. The **swapon -s** command is equivalent to **cat /proc/swaps** .

```
root@debian10:~# swapon /dev/sdb1
root@debian10:~# swapon -s
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       1046524 0       -2
/dev/sdb1                               partition       2097148 0       -3
root@debian10:~#
```

62.2.4 Creating a swap file

Swap partitions are preferred over swap files, but if there is no free disk space to create a partition, then a swap file can be a (temporary) solution. The screenshot creates a two gibibyte swap file named **/swapfile** .

```
root@debian10:~# fallocate -l 2G /swapfile
root@debian10:~# ls -lh /swapfile
-rw-r--r-- 1 root root 2.0G Sep 13 14:11 /swapfile
root@debian10:~# chmod 600 /swapfile
root@debian10:~# mkswap /swapfile
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=05cc69e6-502f-4294-b669-ef10b68dc366
root@debian10:~#
```

You can activate this swap file in the same way as a partition, with the **swapon** command. We now have a total of 5 gibibyte of swap space.

```
root@debian10:~# swapon /swapfile
root@debian10:~# swapon -s
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       1046524 0       -2
/dev/sdb1                               partition       2097148 0       -3
/swapfile                               file            2097148 0       -4
root@debian10:~#
```

62.2.5 Swap space in /etc/fstab

If you want this swap partition and this swap file to be enabled by default at boot, then you have to add them to the **/etc/fstab** file.

```
root@debian10:~# vi /etc/fstab
root@debian10:~# tail -2 /etc/fstab
/dev/sdb1 swap swap defaults 0 0
/swapfile swap swap defaults 0 0
root@debian10:~#
```

62.2.6 vmstat swap io

You can watch the moving of pages in and out of swap space by using the **vmstat** tool. For this occasion I limited the RAM memory of this server to 1GB, as can be seen here by this **free -tm** command.

```
root@debian10:~# free -tm
              total        used        free      shared  buff/cache   available
Mem:            986          53         881           0          51         838
Swap:          5117          15        5102
Total:         6104          68        5984
root@debian10:~#
```

And then I started some memory hungry scripts. You see a lot of **so** or **swap out** writing to swap when the free memory goes below 70MB.

```
root@debian10:~# vmstat 2 200
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 4  0      0 112408  10128  78468    0    0    88     7  219  409  6  2 92  0  0
 4  0   1280  70264   9140  61908    0  590    94   590 6856 14368 60 14 27  0  0
 3  0  18432  67264    436  27676   18 8562    80  8562 7539 14028 59 13 28  0  0
 4  0  68096  66476    200  18904    0 24716   134 24716 7291 13781 56 17 27  0  0
 5  0 133376  71796    200  18360   10 32550    12 32550 7356 13562 59 15 26  0  0
 3  0 186112  69424    200  18572   62 26394   266 26394 7471 13496 56 16 28  0  0
 4  0 251648  70208    200  18480   10 26016    12 26016 7397 14126 58 17 26  0  0
^C
root@debian10:~#
```

62.2.7 swapoff

If you no longer need the extra swap space, then this can be removed with the **swapoff** command, as shown in this screenshot. Do not forget to remove the entries from the **/etc/fstab** file.

```
root@debian10:~# swapoff /swapfile
root@debian10:~# swapoff /dev/sdb1
root@debian10:~# swapon -s
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       1046524 0       -2
root@debian10:~#
```

62.3 top

The **top** command also displays a summary of memory and swap usage in mebibytes. Note that it is perfectly normal for a Linux server to **use** all of its memory. As long as there is no extensive swapping going on, everything is fine.

```
paul@debian10:~$ top
top - 20:53:35 up 14 min,  4 users,  load average: 1.80, 0.69, 0.27
Tasks: 104 total,   4 running,  99 sleeping,   1 stopped,   0 zombie
%Cpu(s): 58.9 us, 13.6 sy,  0.0 ni, 27.4 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :   3946.4 total,   2465.6 free,   1227.5 used,    253.4 buff/cache
MiB Swap:   1022.0 total,   1022.0 free,      0.0 used.   2508.7 avail Mem
```

62.4 other tools

Several other tools like **htop** (apt-get install htop), **atop** (apt-get install atop) and **glances** (apt-get install glances) are available that give a summary of memory and swap usage.

62.5 Cheat sheet

Table 62.1: Memory

command	explanation
dmidecode -t memory	Obtain information about physical memory in a Debian computer.
/proc/meminfo	Contains kernel information about memory (including swap).
free	This tool displays a summary of physical, free and swap memory.
top	This tool displays a summary of physical, free and swap memory.
pmap -x 42	Display the memory map of a process with PID 42.
/proc/swaps	Kernel information about swap space.
swapon -s	Display information about swap space.
mkswap /dev/foo	Create a swap partition.
mkswap /foo	Create a swap file.
swapoff /foo	Turn off a swap file or swap partition.
/etc/fstab	Can contain swap partitions or files to mount at boot.
vmstat 2 200	Display virtual memory statistics.

62.6 Practice

1. If you are on a bare metal server, then run **dmidecode** to see the installed memory modules.
2. Display the total amount of memory, the used memory, the free memory and the cached memory.
3. Same as the previous question, but adding the **Total** line.
4. Display the memory map of the **init** process.
5. Display the currently activated swap space.
6. Create and activate a swap partition on an extra disk.
7. Create and activate a two gibibyte swap file.
8. Optional: Search a memory allocating script on the internet and watch your memory being consumed. Maybe also watch swap space being used.
9. Deactivate your swap file and swap partition.
10. Optional: use top, atop, htop and glances to look at memory usage.

62.7 Solution

1. If you are on a bare metal server, then run **dmidecode** to see the installed memory modules.

```
dmidecode -t memory
```

2. Display the total amount of memory, the used memory, the free memory and the cached memory.

```
free -m
```

3. Same as the previous question, but adding the **Total** line.

```
free -mt
```

4. Display the memory map of the **init** process.

```
pmap -x 1
```

5. Display the currently activated swap space.

```
cat /proc/swaps
swapon -s
```

6. Create and activate a swap partition on an extra disk.

```
fdisk /dev/sdb
mkswap /dev/sdb1
swapon /dev/sdb1
```

7. Create and activate a two gibibyte swap file.

```
fallocate -l 2G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
```

8. Optional: Search a memory allocating script on the internet and watch your memory being consumed. Maybe also watch swap space being used.

```
watch free -om
vmstat 2 100
```

9. Deactivate your swap file and swap partition.

```
swapoff /swapfile
swapoff /dev/sdb1
```

10. Optional: use top, atop, htop and glances to look at memory usage.

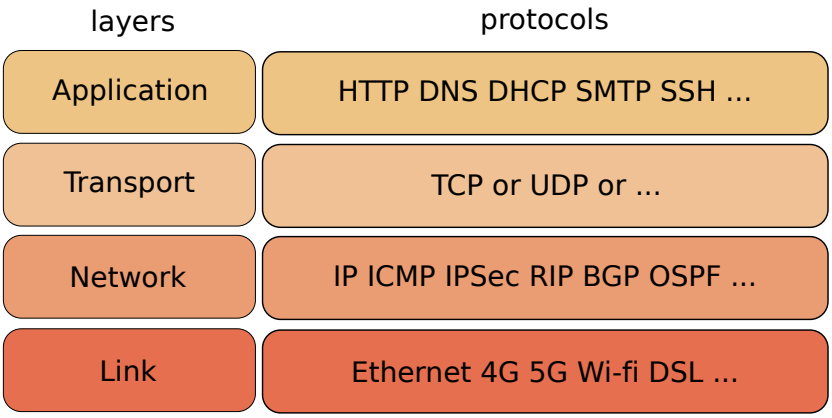
```
top
atop
htop
glances
```

Chapter 63

Introduction to networks

63.1 TCP/IP

In the 1980s and 1990s there were a lot of protocols in use to construct networks. IBM had SNA, Apple had Appletalk, Novell had Netware and so on. Nowadays all these protocols are replaced by **TCP/IP**. The **Transmission Control Protocol/Internet Protocol** is used everywhere, from the smallest home networks to the worldwide Internet.



Explaining networking is today in 2019 the same as explaining **TCP/IP**.

63.1.1 RFC

TCP/IP is a protocol stack with more than a thousand protocols. You may know some of them like HTTP, or DHCP, or DNS just to name a few. These protocols are all defined in **RFCs**. For example RFC 2616 is about HTTP and RFC 2131 describes DHCP.

The official website that hosts all RFCs is https://www.rfc-editor.org/ . You can find any RFC here using this example URL:

```
https://www.rfc-editor.org/rfc/rfc2131.txt .
```

63.1.2 the Internet

The history of the Internet begins in 1969, which is also the birth year of Unix and is also the year in which the RFCs were started. While all this history is very interesting, it is out of the scope of this book.

Today the Internet is one giant worldwide TCP/IP network.

63.1.3 an intranet

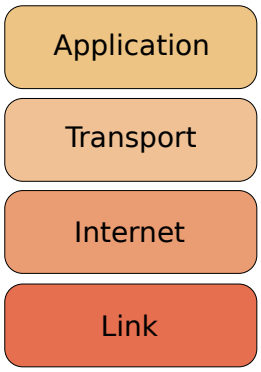
Many organisations have an **intranet** that is only accessible to certain groups of people. This too is a TCP/IP network, it is just restricted in use. An intranet relies on the same stack of protocols that the Internet does.

63.1.4 4G, 5G, Wi-fi, hotspot

So what about 4G and Wi-fi and a hotspot? Well 4G and 5G is TCP/IP via you smartphone, Wi-fi is TCP/IP without network cables and a hotspot is just a restricted Wi-fi network. It is all TCP/IP today.

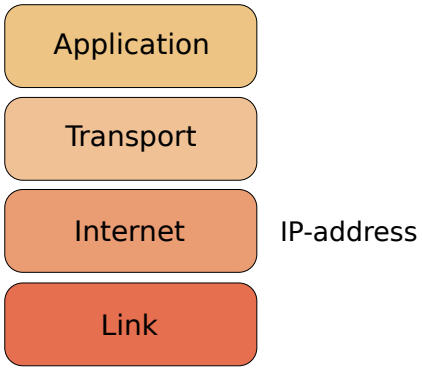
63.2 TCP/IP Layers

The TCP/IP protocol works in layers. There are four distinct layers in this protocol: at the top is the **Application** layer, just underneath is the **Transport** layer, below that is the **Internet** layer and at the bottom is the **Link** layer.



63.2.1 IP-address

An IP-address is part of the **Internet** layer. Every computer on the network, be it the Internet or an intranet, has an IP-address.



Today there are two forms of IP-addresses in use. There are the old **IPv4** (version 4) addresses and the new **IPv6** addresses.

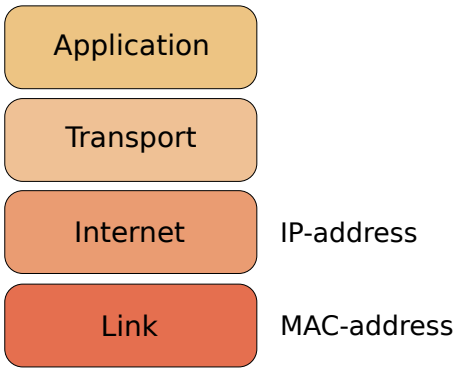
IPv4 addresses are 32-bit in length and are usually written in four decimal numbers separated by a dot, for example 10.0.2.1 or 192.168.56.101 or 88.151.243.8 . There are a little more than four billion IPv4 addresses.

IPv6 addresses are 128-bit in length and are usually written in hexadecimal form, for example: fe80::1a65:90ff:fedb:cb89 or 2a00:1450:400e:80b::200e . There are about 340 billion billion billion billion IPv6 addresses.

In **IPv4** the ranges 10.0.0.0/8 and 192.168.0.0/16 are reserved for private use, these adresses cannot exist on the Internet. The 169.254.0.0/16 is reserved for **zeroconf** (see the DHCP chapter) and also cannot exist on the Internet.

63.2.2 MAC-address

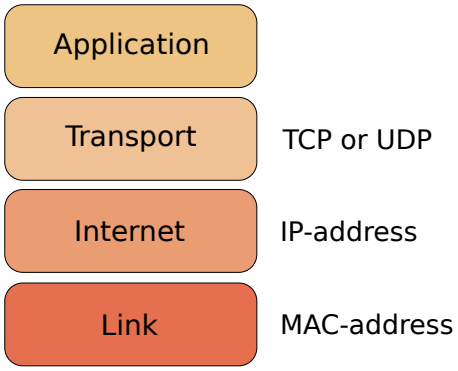
The hardware that is being used for network connectivity has a **MAC-address**. Each **MAC-address** is unique in its local environment. The **MAC-address** is part of the **Link** layer.



A **MAC-address** is 48-bits (or six bytes) in length and is usually written in hexadecimal form, for example : 08:00:27:3f:27:a6 or 18:65:90:dc:cb:49.

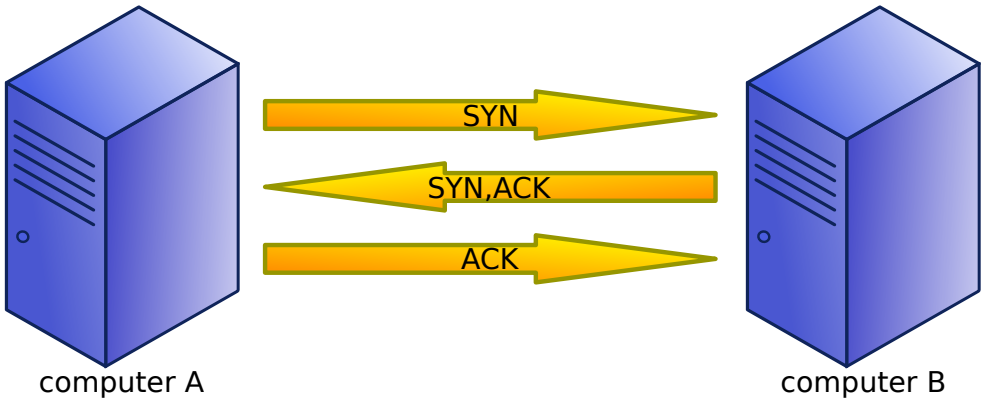
63.2.3 TCP or UDP

The two **Transport** protocols that we will discuss the most are **TCP** and **UDP**. In short **TCP** is a reliable protocol with a lot of overhead, and **UDP** is a much simpler, faster, but unreliable protocol.



63.2.3.1 TCP

The **TCP** protocol will set up a connection between two computers with a triple handshake. First part is a **SYN** packet sent from **computer A** to **computer B**, this packet basically asks "Will you connect with me?". The reply to this packet is called a **SYN,ACK** and is the second part of the triple handshake and goes from **computer B** to **computer A** basically saying "Yes, I would connect with you. Will you connect with me?" The third part is a simple **ACK** from **computer A** to confirm a TCP connection is set.



TCP will then number all packets from computer A to computer B and will confirm whether all packets were received, re-sending them if necessary. When the TCP transfer is done a **FIN** packet will be sent, which will receive a final **ACK**.

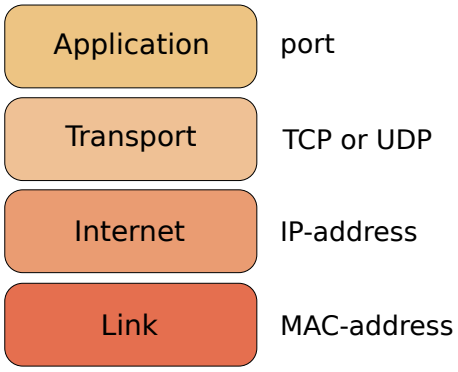
Protocols like SMTP (for e-mail) and HTTP (for websites) rely on a TCP connection.

63.2.3.2 UDP

The **UDP** protocol will not set up a connection, it will just send the data and hope that it arrives. This is much simpler and faster than TCP on reliable network connections. For example DNS queries on your local subnet are done with UDP.

63.2.4 port

Every networked application will use a **port** to send and/or receive data. Many **ports** are **well known** like 22 for SSH, 53 for DNS and 80 for HTTP.



A TCP connection is initiated from a client application and will use a **source** port and a **destination** port. The destination port is to connect to the server application, and the source port is there because traffic has to be able to return to the client application.

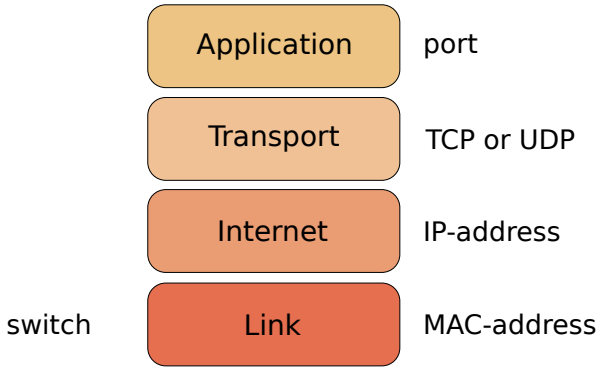
63.3 TCP/IP devices

63.3.1 hub

A **hub** is a device that receives and transmit electrical signals. It may be aware of zeroes and ones and retransmit those. A **hub** is invisible for other devices on the network (it could be pictured below the **Link** layer) and is rare in 2019.

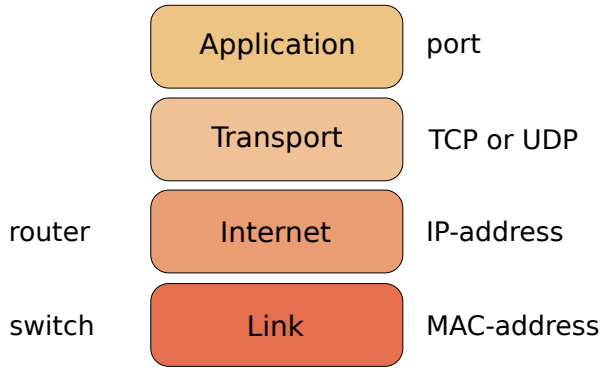
63.3.2 switch

A **switch** is an intelligent device that can make decisions based on the **MAC-address** in the network **frames**. A **switch** is a device in the **Link** layer.

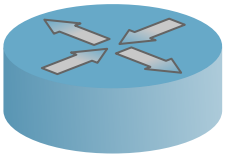


63.3.3 router

A **router** is an intelligent device that can make decisions based on the **IP-address**, so it belongs in the **Internet** layer. Some switches can also do this, so the line between routers and switches is blurring.



A router is often depicted like this in a network diagram.



63.4 unicast - broadcast

A **broadcast** message is a message to everyone (in a certain group). A **unicast** message is part of a one-to-one communication, a **unicast** message is sent to one member of a group.

More information on unicast, broadcast and others, and nice pictures, can be found here:

```
https://en.wikipedia.org/wiki/Routing#Delivery_schemes
```

63.5 LAN - WAN

A **LAN** is a **Local Area Network**, in a **LAN** the computers are close to each other. There can be few computers in a **LAN** and there can be many, for example a large building with hundreds of computers can be a **LAN**. A **LAN** often uses **Ethernet**.

A **WAN** or **Wide Area Network** is a network where the computers are far apart. For example a computer in Beijing and a computer in Paris can be connected via a **WAN**. There are several **WAN** protocols like ATM, Frame Relay, X.25 and so on.

A lot more information can be found here:

```
https://en.wikipedia.org/wiki/Computer_network
```

63.6 Cheat sheet

Table 63.1: Introduction to networks

term	**explanation**
TCP/IP	A common name for the Internet protocol stack.
RFC	Request For Comment, usually a TCP/IP standard.
the Internet	A worldwide TCP/IP network.
an intranet	A local (organisation-wide) TCP/IP network.
IP address	A unique address for a computer on a TCP/IP network.
MAC address	A unique address for a computer on a local network.
TCP	A reliable connection protocol.
UDP	An unreliable connectionless fast protocol.
port	A unique identifier for an application.
hub	A layer 1 device.
switch	A layer 2 device (though the distinction with a router is blurring).
router	A layer 3 device (it makes decisions based on IP address).
unicast	A one-to-one message.
broadcast	A message to everyone (of a certain group).
LAN	A Local Area Network (computers close to each other, often Ethernet).
WAN	A Wide Area Network (computers far apart).

63.7 Practice

1. Name the website were you can find all the RFC's.
2. Name the largest TCP/IP network.
3. Name the four TCP/IP layers from top to bottom.
4. In which layer do we find IP-addresses?
5. In which layer do we find a switch?
6. Which device makes decision based on IP-address?
7. How are applications defined in TCP/IP?
8. A message to all members of a group is a . . . ?
9. What are two key differences between a LAN and a WAN?

63.8 Solution

1. Name the website were you can find all the RFC's.

```
www.rfc-editor.org
```

2. Name the largest TCP/IP network.

```
The Internet
```

3. Name the four TCP/IP layers from top to bottom.

```
Application
Transport
Internet
Link
```

4. In which layer do we find IP-addresses?

```
Internet
```

5. In which layer do we find a switch?

```
Link
```

6. Which device makes decision based on IP-address?

```
a router
```

7. How are applications defined in TCP/IP?

```
by a port number
```

8. A message to all members of a group is a . . . ?

```
broadcast
```

9. What are two key differences between a LAN and a WAN?

```
distance between computers
protocol being used
```

Chapter 64

Managing tcp/ip

64.1 network adapters

Every server has one or more network adapters. You can see a list of network adapters in **/sys/class/net**. The files in there are symbolic links to **/sys/devices/**.

```
paul@debian10:~$ ls /sys/class/net/
enp0s3  enp0s8  lo
paul@debian10:~$
```

64.2 ip a

The first tool of this chapter is the **ip** tool. Most common use is maybe **ip a** to get IP-address, subnet and MAC information from all network adapters. This includes the loopback adapter named **lo**.

Our **debian10** server of this screenshot has IP-address 192.168.56.101 on interface enp0s3 and IP-address 10.0.2.101 on interface enp0s8 (Look behind **inet**). After the **inet6** keyword are the IPv6 addresses for this server.

```
paul@debian10:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group  ↩
    default qlen 1000
    link/ether 08:00:27:3f:27:a6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s3
       valid_lft 1157sec preferred_lft 1157sec
    inet6 fe80::a00:27ff:fe3f:27a6/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group  ↩
    default qlen 1000
    link/ether 08:00:27:ff:0c:3c brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.101/24 brd 10.0.2.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:c3c/64 scope link
       valid_lft forever preferred_lft forever
paul@debian10:~$
```

You can show information about just one adapter by typing **ip a s dev** followed by the adapter name. The **a** is short for **address**, and the **s** is short for **show**.

```
paul@debian10:~$ ip a s dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group  ↩
    default qlen 1000
    link/ether 08:00:27:3f:27:a6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s3
       valid_lft 948sec preferred_lft 948sec
    inet6 fe80::a00:27ff:fe3f:27a6/64 scope link
       valid_lft forever preferred_lft forever
paul@debian10:~$
```

64.3 net-tools

Wait, what about the **ifconfig** command, isn't that the standard on Debian Linux? The answer is **not anymore**. The **ifconfig** command is part of the **net-tools** package, which is no longer installed by default. But it still works, as this screenshot shows.

```
paul@debian10:~$ /sbin/ifconfig enp0s3
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.101  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe3f:27a6  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:3f:27:a6  txqueuelen 1000  (Ethernet)
        RX packets 1091  bytes 93631 (91.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 645  bytes 108797 (106.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

paul@debian10:~$
```

64.4 ip n

Next is the **ip n** command to show the **ARP table** of the server. Whenever your server has contact with another device on the network, then it's MAC-address will be kept in the **ARP cache** (for a while). You can see this **ARP cache** with the **ip n** command (n is short for **neighbour**).

```
paul@debian10:~$ ip n
192.168.56.102 dev enp0s3 lladdr 08:00:27:f5:54:03 REACHABLE
10.0.2.1 dev enp0s8 lladdr 52:54:00:12:35:00 STALE
192.168.56.100 dev enp0s3 lladdr 08:00:27:b4:90:ab STALE
192.168.56.1 dev enp0s3 lladdr 0a:00:27:00:00:00 REACHABLE
paul@debian10:~$
```

The **net-tools** equivalent of this command is the **arp** command, which in my humble opinion gives a much more readable output than **ip n**.

```
paul@debian10:~$ /sbin/arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.56.102           ether   08:00:27:f5:54:03   C                     enp0s3
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s8
192.168.56.100           ether   08:00:27:b4:90:ab   C                     enp0s3
192.168.56.1             ether   0a:00:27:00:00:00   C                     enp0s3
paul@debian10:~$
```

64.5 ip r

Type **ip r** to see the **routing table** of your server. The screenshot shows that 10.0.2.1 is the **default gateway** and the server is connected to two subnets (192.168.56.0/24 and 10.0.2.0/24).

```
paul@debian10:~$ ip r
default via 10.0.2.1 dev enp0s8 onlink
10.0.2.0/24 dev enp0s8 proto kernel scope link src 10.0.2.101
169.254.0.0/16 dev enp0s8 scope link metric 1000
192.168.56.0/24 dev enp0s3 proto kernel scope link src 192.168.56.101
paul@debian10:~$
```

Again I personally find the output of the net-tools **route** command much more readable. This is identical to typing **netstat -nr** (which is also from net-tools).

```
paul@debian10:~$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    0      0        0 enp0s8
```

```
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 enp0s8
link-local      0.0.0.0         255.255.0.0     U     1000   0        0 enp0s8
192.168.56.0    0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
paul@debian10:~$
```

64.6 /etc/network/interfaces

Configuration of network interfaces is done in the **/etc/network/interfaces** file. The first interface defined in this file is often the **loopback** interface. You probably never have to change this.

```
# The loopback network interface
auto lo
iface lo inet loopback
```

Next is the primary interface, which is usually configured when installing the Debian server. In this example DHCP is used (even though the server gets a fixed IP-address from DHCP). The name of the network adapter **enp0s3** can be different on your server.

```
# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet dhcp
```

Our server has a fixed IP-address on the second network card. The fixed IP is 10.0.2.101 and the subnet mask is 255.255.255.0 (or /24). This interface also has a default gateway.

```
allow-hotplug enp0s8
iface enp0s8 inet static
address 10.0.2.101
netmask 255.255.255.0
gateway 10.0.2.1
```

For DHCPv6 (to obtain an IPv6) address add the following line to the interface configuration.

```
iface enp0s8 inet6 auto
```

64.7 ifup & ifdown

You can turn off an interface with the **ifdown** command. And you can turn it back on with **ifup**. These commands will use the information in **/etc/network/interfaces** . Nothing much seems to happen in this screenshot because **enp0s8** has a fixed IP-address.

Tip
Do not do an **ifdown** of the interface over which you are connected.

```
root@debian10:~# ifdown enp0s8
root@debian10:~# ifup enp0s8
root@debian10:~#
```

Using **ifdown** and **ifup** on the **enp0s3** interface provides a lot of output because this interface is configured to use DHCP. The **ifdown** command will release the IP-address from the DHCP server.

```
root@debian10:~# ifdown enp0s3
Killed old client process
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
```

```
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:3f:27:a6
Sending on   LPF/enp0s3/08:00:27:3f:27:a6
Sending on   Socket/fallback
DHCPRELEASE of 192.168.56.101 on enp0s3 to 192.168.56.100 port 67
root@debian10:~#
```

Using **ifup** will again contact the DHCP server to get an IP-address. Depending on the configuration of the DHCP server this can be the same IP-address or a new one. Setting up a DHCP server is a whole new chapter later in this book.

```
root@debian10:~# ifup enp0s3
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:3f:27:a6
Sending on   LPF/enp0s3/08:00:27:3f:27:a6
Sending on   Socket/fallback
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 8
DHCPOFFER of 192.168.56.101 from 192.168.56.100
DHCPREQUEST for 192.168.56.101 on enp0s3 to 255.255.255.255 port 67
DHCPACK of 192.168.56.101 from 192.168.56.100
bound to 192.168.56.101 -- renewal in 558 seconds.
root@debian10:~#
```

64.8 /etc/resolv.conf

Usually when using DHCP then the **/etc/resolv.conf** file will be automatically configured to contain the correct DNS server. If not, then you can manually add a DNS server in this file. In the example below we use **8.8.8.8** as a DNS server.

```
root@debian10:~# cat /etc/resolv.conf
domain lan
search lan
nameserver 8.8.8.8
root@debian10:~#
```

64.9 /etc/services

Debian Linux comes with an extensive list of ports for TCP and UDP in the **/etc/services** file. Here you (and applications) can find applications related to a certain port.

```
root@debian10:~# head -30 /etc/services | tail -6
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                          # SSH Remote Login Protocol
root@debian10:~#
```

64.10 /etc/protocols

The **/etc/protocols** file contains protocols like TCP, UDP and ICMP associated with a number to be used in the IP-datagram. You probably never need this file, when talking about protocols people actually mean protocols in **/etc/services**.

```
root@debian10:~# head -20 /etc/protocols | tail -5
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway (Cisco)
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
root@debian10:~#
```

64.11 ping

One of the most basic tools for testing a **TCP/IP** connection is **ping**. You can literally **ping** a local or remote server. This screenshot first **pings** a local server on the same subnet, and then does a **ping** to 8.8.8.8 which is a Google server on the Internet.

```
paul@debian10:~$ ping -c1 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.538 ms

--- 10.0.2.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.538/0.538/0.538/0.000 ms
paul@debian10:~$ ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=54 time=25.3 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.268/25.268/25.268/0.000 ms
paul@debian10:~$
```

The **ping** to the local server happens in less than a millisecond, where the **ping** tot the Google server takes 25 milliseconds. The **-c1** option performs just one ping, otherwise use **Ctrl-c** to interrupt the command.

64.12 traceroute

The **traceroute** tool can be used to trace how many routers there are between your server and another server. Each router is called a **hop**. The 8.8.8.8 server is eight **hops** away from our **debian10** server in this screenshot.

```
root@debian10~# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  172.31.1.1 (172.31.1.1)  0.115 ms  0.062 ms  0.057 ms
 2  10298.your-cloud.host (88.99.159.100)  0.167 ms  0.117 ms  0.094 ms
 3  * * *
 4  spine1.cloud1.nbg1.hetzner.com (85.10.237.193)  0.751 ms  1.116 ms spine2.cloud1.nbg1. ↩
    hetzner.com (85.10.237.197)  0.837 ms
 5  213-239-208-221.clients.your-server.de (213.239.208.221)  0.356 ms  0.369 ms core11. ↩
    nbg1.hetzner.com (85.10.250.209)  0.339 ms
 6  core4.fra.hetzner.com (213.239.245.33)  3.592 ms core0.fra.hetzner.com (213.239.252.21) ↩
      3.517 ms core4.fra.hetzner.com (213.239.245.245)  3.438 ms
 7  72.14.218.94 (72.14.218.94)  3.726 ms  3.698 ms  3.669 ms
 8  * 108.170.251.129 (108.170.251.129)  3.754 ms 108.170.251.193 (108.170.251.193)  3.358  ↩
    ms
```

```
 9  64.233.175.171 (64.233.175.171)  3.674 ms dns.google (8.8.8.8)  3.529 ms  ↩
    108.170.235.249 (108.170.235.249)  3.448 ms
root@debian10~#
```

64.13 ss

The **ss** tool is the modern version of the legacy **netstat** tool. It has much the same functionality and most of the same options. In this screenshot we use both tools to look at open TCP ports on our server. The **t** option is for TCP, replace with **u** for UDP. Port 22 is used for SSH connections.

```
root@debian10:~# ss -napt | cut -b-84
State     Recv-Q    Send-Q        Local Address:Port       Peer Address:Port
LISTEN    0         128                 0.0.0.0:22             0.0.0.0:*        use
LISTEN    0         20                127.0.0.1:25             0.0.0.0:*        use
ESTAB     0         0             192.168.56.101:22         192.168.56.1:56934    use
LISTEN    0         128                   [::]:22                [::]:*        use
LISTEN    0         20                   [::1]:25                [::]:*        use
root@debian10:~# netstat -napt | cut -b-84
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      500/
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      738/
tcp        0      0 192.168.56.101:22       192.168.56.1:56934      ESTABLISHED 1716
tcp6       0      0 :::22                   :::*                    LISTEN      500/
tcp6       0      0 ::1:25                  :::*                    LISTEN      738/
root@debian10:~#
```

64.14 dhclient

On servers with a fixed IP-address it can happen that the DHCP client named **dhclient** is running to obtain this fixed IP-address. Do not kill this client, as your server will lose its IP-address after a while.

```
root@debian10:~# ps fax | grep dhclient
 1896 pts/0    S+     0:00                    \_ grep dhclient
  945?        Ss     0:00 /sbin/dhclient -4 -v -i -pf /run/dhclient.enp0s3.pid -lf /var/lib ↩
      /dhcp/dhclient.enp0s3.leases -I -df /var/lib/dhcp/dhclient6.enp0s3.leases enp0s3
root@debian10:~#
```

64.15 Cheat sheet

Table 64.1: Managing TCP/IP

command	explanation
/sys/class/net	Contains names of network adapters on this Debian computer.
ip a	Display TCP/IP configuration for network adapters.
ifconfig	Legacy tool to display TCP/IP configuration.
ip n	Display ARP cache.
arp	Legacy tool to display ARP cache.
ip r	Display routing information.
route	Legacy tool to display routing information.
/etc/network/interfaces	Contains the TCP/IP configuration of network adapters.
ifup foo	Bring the interface **foo** up (with its configuration).
ifdown foo	Bring the interface **foo** down.
/etc/resolv.conf	Contains DNS server configuration.
/etc/services	Lists port to application configuration.
/etc/protocols	Lists layer 4 protocol configuration.
ping	A tool to test an IP connection.
traceroute	A tool to display routes between computers.
ss	A tool to display open ports and their applications.
netstat	A legacy tool to display open ports and their applications.
dhclient	A daemon that maintains an IP configuration received from a DHCP server.

64.16 Practice

1. List the names of the network adapters on your server.
2. List the IP and MAC address of each of your network adapters.
3. List the cache of MAC-IP addresses of your server.
4. List the routing table of your server.
5. Display the configuration of the network adapters.
6. Which DNS server is being used on your server?
7. Display the list of TCP and UDP protocols.
8. Test your connection to 8.8.8.8.
9. Display the hops between you and 8.8.8.8.
10. List the open TCP ports on your server.

64.17 Solution

1. List the names of the network adapters on your server.

```
ls /sys/class/net/
```

2. List the IP and MAC address of each of your network adapters.

```
ip a
/sbin/ifconfig
```

3. List the cache of MAC-IP addresses of your server.

```
ip n
/sbin/arp
```

4. List the routing table of your server.

```
ip r
/sbin/route
```

5. Display the configuration of the network adapters.

```
more /etc/network/interfaces
```

6. Which DNS server is being used on your server?

```
cat /etc/resolv.conf
```

7. Display the list of TCP and UDP protocols.

```
more /etc/services
```

8. Test your connection to 8.8.8.8.

```
ping 8.8.8.8
```

9. Display the hops between you and 8.8.8.8.

```
traceroute 8.8.8.8
```

10. List the open TCP ports on your server.

```
ss -napt
```

Chapter 65

Sniffing the network

65.1 tcpdump

The tools to use for sniffing the network are **tcpdump** on the command line, and **wireshark** if you have (and prefer) a graphical interface. We will first take a look at **tcpdump**.

```
root@debian10:~# apt-get install tcpdump
<output truncated>
root@debian10:~# tcpdump --version
tcpdump version 4.9.2
libpcap version 1.8.1
OpenSSL 1.1.1c  28 May 2019
root@debian10:~#
```

65.1.1 Listing all interfaces

Type **tcpdump -D** to obtain a list of interfaces on which **tcpdump** can listen. You should see a list of all your network adapters followed by an **any** adapter and **lo** for the loopback device. The **nflog** and **nfqueue** devices are part of the netfilter firewall, these are out of the scope of this chapter.

```
root@debian10:~# tcpdump -D
1.enp0s3 [Up, Running]
2.enp0s8 [Up, Running]
3.any (Pseudo-device that captures on all interfaces) [Up, Running]
4.lo [Up, Running, Loopback]
5.nflog (Linux netfilter log (NFLOG) interface)
6.nfqueue (Linux netfilter queue (NFQUEUE) interface)
7.usbmon1 (USB bus number 1)
root@debian10:~#
```

65.1.2 Choosing an interface

Using the **-i** option of **tcpdump** you can specify an interface to listen on. The amount of traffic passing over an interface can be overwhelming. Especially if you select the interface of your **ssh** connection. In this screenshot we select the second network adapter on our server, there is currently no traffic on this adapter.

```
root@debian10:~# tcpdump -i enp0s8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Note
We will also use the **-n** option of **tcpdump** to prevent DNS lookups of the hosts.

65.1.3 Limiting the number of packets

Use the **-c** option of **tcpdump** to display a count of packages. This is useful to limit the output of the command, since there can be many hundreds of packets per second. To obtain the screenshot below we started a **ping** command on **server2** to this second adapter on the server named **debian10**. We capture only four packets.

```
root@debian10:~# tcpdump -i enp0s8 -c 4 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
17:31:11.490797 IP 10.0.2.102 > 10.0.2.101: ICMP echo request, id 1087, seq 1, length 64
17:31:11.490830 IP 10.0.2.101 > 10.0.2.102: ICMP echo reply, id 1087, seq 1, length 64
17:31:12.563476 IP 10.0.2.102 > 10.0.2.101: ICMP echo request, id 1087, seq 2, length 64
```

```
17:31:12.563510 IP 10.0.2.101 > 10.0.2.102: ICMP echo reply, id 1087, seq 2, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
root@debian10:~#
```

Note
Notice that **ping** uses the ICMP protocol (from /**etc**/**protocols**).

65.1.4 writing to a file

Using **tcpdump -w** will write the packets to a file, in this case to **dump.pcap**. I put the command in background here so I could generate some traffic with **wget** of an **index.html** file.

```
root@debian10:~# tcpdump -i any -n -w dump.pcap &
[1] 761
root@debian10:~# tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture  ↩
    size 262144 bytes

root@debian10:~# wget http://linux-training.be
<output truncated>
2019-09-18 12:57:29 (149 MB/s) - index.html saved [4375]
root@debian10:~# fg
tcpdump -i any -n -w dump.pcap
^C142 packets captured
145 packets received by filter
0 packets dropped by kernel
root@debian10:~#
```

Tip
You can download this dump.pcap file with **wget http://linux-training.be/dump.pcap** .

65.1.5 sniffing for the DNS port

We can now use **tcpdump** or **wireshark** on the saved **.pcap** file. In this screenshot we are sniffing for DNS traffic by limiting the sniff to **port 53**. Notice how there is a query for an A record and for a quad-A (AAAA) record. We will see DNS in detail later in this book.

```
root@debian10:~# tcpdump -n -r dump.pcap port 53
reading from file dump.pcap, link-type EN10MB (Ethernet)
16:02:07.880301 IP 10.0.2.101.41759 > 8.8.8.8.53: 49036+ A? linux-training.be. (35)
16:02:07.880315 IP 10.0.2.101.41759 > 8.8.8.8.53: 7570+ AAAA? linux-training.be. (35)
16:02:07.899846 IP 8.8.8.8.53 > 10.0.2.101.41759: 7570 0/1/0 (96)
16:02:07.920145 IP 8.8.8.8.53 > 10.0.2.101.41759: 49036 1/0/0 A 88.151.243.8 (51)
root@debian10:~#
```

65.1.6 sniffing for a host

We received **88.151.243.8** as the IP-address of the web server in the DNS query above. In this screenshot we sniff for the first three packets in relation to this host. What we see is here is TCP triple handshake; there is a SYN **[S]**, then a SYN,ACK **[S.]** and then an ACK **[.]** .

```
root@debian10:~# tcpdump -nt -r dump.pcap host 88.151.243.8 | head -3 | cut -b-84
reading from file dump.pcap, link-type EN10MB (Ethernet)
IP 10.0.2.101.53728 > 88.151.243.8.80: Flags [S], seq 1072223233, win 29200, options
IP 88.151.243.8.80 > 10.0.2.101.53728: Flags [S.], seq 6552, ack 1072223234, win 327
IP 10.0.2.101.53728 > 88.151.243.8.80: Flags [.], ack 1, win 29200, length 0
root@debian10:~#
```

65.1.7 Displaying HEX and ASCII

Using the **-X** option of **tcpdump** you can display the packet contents in hexadecimal and ASCII format. We sniff for TCP port 80 (=http) and for destination (=dst) IP 88.151.243.8. The output of this screenshot is heavily truncated.

```
root@debian10:~# tcpdump -nt -r dump.pcap tcp port http and dst 88.151.243.8 -X
<output truncated>
IP 10.0.2.101.53728 > 88.151.243.8.80: Flags [P.], seq 0:144, ack 1, win 29200, length 144: ↩
    HTTP: GET / HTTP/1.1
        0x0000:  4500 00b8 a976 4000 4006 38c5 0a00 0265  E....v@.@.8....e
        0x0010:  5897 f308 d1e0 0050 3fe8 d402 0000 1999  X......P?.......
        0x0020:  5018 7210 58af 0000 4745 5420 2f20 4854  P.r.X...GET./.HT
        0x0030:  5450 2f31 2e31 0d0a 5573 6572 2d41 6765  TP/1.1..User-Age
        0x0040:  6e74 3a20 5767 6574 2f31 2e32 302e 3120  nt:.Wget/1.20.1.
        0x0050:  286c 696e 7578 2d67 6e75 290d 0a41 6363  (linux-gnu)..Acc
        0x0060:  6570 743a 202a 2f2a 0d0a 4163 6365 7074  ept:./*..Accept
        0x0070:  2d45 6e63 6f64 696e 673a 2069 6465 6e74  -Encoding:.ident
        0x0080:  6974 790d 0a48 6f73 743a 206c 696e 7578  ity..Host:.linux
        0x0090:  2d74 7261 696e 696e 672e 6265 0d0a 436f  -training.be..Co
        0x00a0:  6e6e 6563 7469 6f6e 3a20 4b65 6570 2d41  nnection:.Keep-A
        0x00b0:  6c69 7665 0d0a 0d0a                      live....
<output truncated>
```

What you see in the output in ASCII is an **HTTP GET** instruction, followed by the **User Agent** of the **wget** command. The web server will respond to this with an **index.html** page (or which ever is set on the web server).

65.1.8 Sniffing with just ASCII

This is the same as the previous example, but with the **-A** option instead of **-X**.

```
root@debian10:~# tcpdump -nt -r dump.pcap tcp port http and dst 88.151.243.8 -A
<output truncated>
IP 10.0.2.101.53728 > 88.151.243.8.80: Flags [P.], seq 0:144, ack 1, win 29200, length 144: ↩
    HTTP: GET / HTTP/1.1
E....v@.@.8.
..eX......P?.......P.r.X...GET / HTTP/1.1
User-Agent: Wget/1.20.1 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: linux-training.be
Connection: Keep-Alive
<output truncated>
```

65.1.9 Sniffing for ARP

Sniffing for **ARP** can be a bit of a challenge, because if there is already some communication going on, then the **ARP cache** will be used. In this screenshot the 10.0.2.101 server (which is our **debian10** server) asks for the **MAC address** of the 10.0.2.1 computer (= the gateway). And it gets a response.

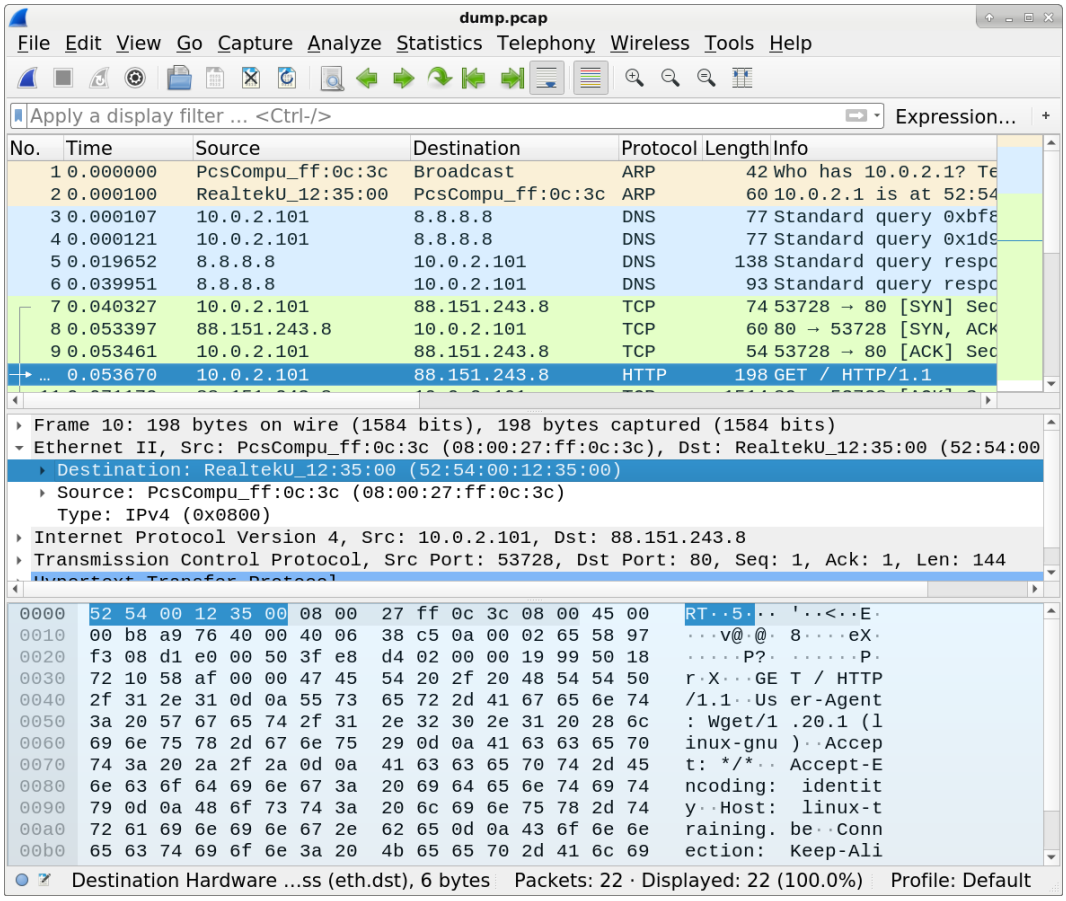
```
root@debian10:~# tcpdump -nt -r dump.pcap arp
reading from file dump.pcap, link-type EN10MB (Ethernet)
ARP, Request who-has 10.0.2.1 tell 10.0.2.101, length 28
ARP, Reply 10.0.2.1 is-at 52:54:00:12:35:00, length 46
root@debian10:~#
```

Note
It is common to see the reverse ARP from .1 asking for .101 right after this one, but in our case the gateway still had that information in its **ARP cache**.

65.2 Wireshark

Wireshark is a graphical application that can help you a lot in analysing network traffic, or in studying how protocols work. We opened our **dump.pcap** file, which we created with **tcpdump**, in **Wireshark** and it looks like this screenshot.

In the top pane you should recognise the first two **ARP** packets, followed by the four **DNS** packets, and then followed by the **TCP triple handshake** and the **HTTP GET** packet. We discussed these in the **tcpdump** section of this chapter.



In the Wireshark screenshot the 10th packet is selected in the top pane. Details of the 10th packet are visible in the middle pane and in the lower pane.

In the middle pane you can see the **layers** of the TCP/IP protocol stack. The **Ethernet** (or physical) layer, below this the **Internet protocol** (or Internet) layer and below that the **TCP** (or Transport) layer. In the lower pane you can see a hex dump and an ASCII dump of the selected packet.

You can also see that the **destination MAC address** is selected in the middle pane. This selection is visible as the first six bytes of the **Ethernet frame**. The next six bytes form the **source MAC-address**. Using different selections in the middle pane will inform you on the location of the selected item in the Ethernet frame.

65.2.1 Wireshark filters

You can use several filters in the "Apply a display filter..." box of **Wireshark**. Valid filters are for example **tcp**, **udp**, **dns**, **arp** and numerous other protocols. To filter on IP-address type for example **ip.addr==10.0.2.101**, and to filter for a source IP-address type for example **ip.src==88.151.243.8** .

Filters can be combined with **and** and **or** for example **ip.src==88.151.243.8 and http** or **ip.addr==10.0.2.101 or dns** .

65.3 Cheat sheet

Table 65.1: Sniffing the network

command	**explanation**
tcpdump -D	List all interfaces.
tcpdump -i foo	Listen on the **foo** interface.
tcpdump -c 4	Sniff only four packets.
tcpdump -n	Don't resolve to DNS names.
tcpdump -w foo.pcap	Save output to a file named **foo.pcap**.
tcpdump port 53	Listen only on port 53.
tcpdump host 8.8.8.8	Listen only for packets concerning host 8.8.8.8.
tcpdump -X	Display hexadecimal and ASCII in output.
tcpdump -A	Display ASCII packet contents.
tcpdump arp	Sniff only ARP packets.
wireshark	A graphical sniffer to learn about networks.

65.4 Practice

1. Install **tcpdump** and list all interfaces that can be monitored by **tcpdump**.
2. Start a ping to your server and capture the ICMP packets with **tcpdump**.
3. Start a ping to your server and capture only the ICMP packets with **tcpdump**.
4. Capture DNS traffic with **tcpdump** .
5. Capture only traffic to and from the 8.8.8.8 server.
6. Capture ARP traffic and show hardware MAC addresses in the output of **tcpdump**.

65.5 Solution

1. Install **tcpdump** and list all interfaces that can be monitored by **tcpdump**.

```
apt-get install tcpdump
tcpdump -D
```

2. Start a ping to your server and capture the ICMP packets with **tcpdump**.

```
tcpdump -n -i any
```

3. Start a ping to your server and capture only the ICMP packets with **tcpdump**.

```
tcpdump -i any icmp
```

4. Capture DNS traffic with **tcpdump** .

```
tcpdump -i any port 53
```

5. Capture only traffic to and from the 8.8.8.8 server.

```
tcpdump -i any host 8.8.8.8
```

6. Capture ARP traffic and show hardware MAC addresses in the output of **tcpdump**.

```
tcpdump -e -i any arp
```

Chapter 66

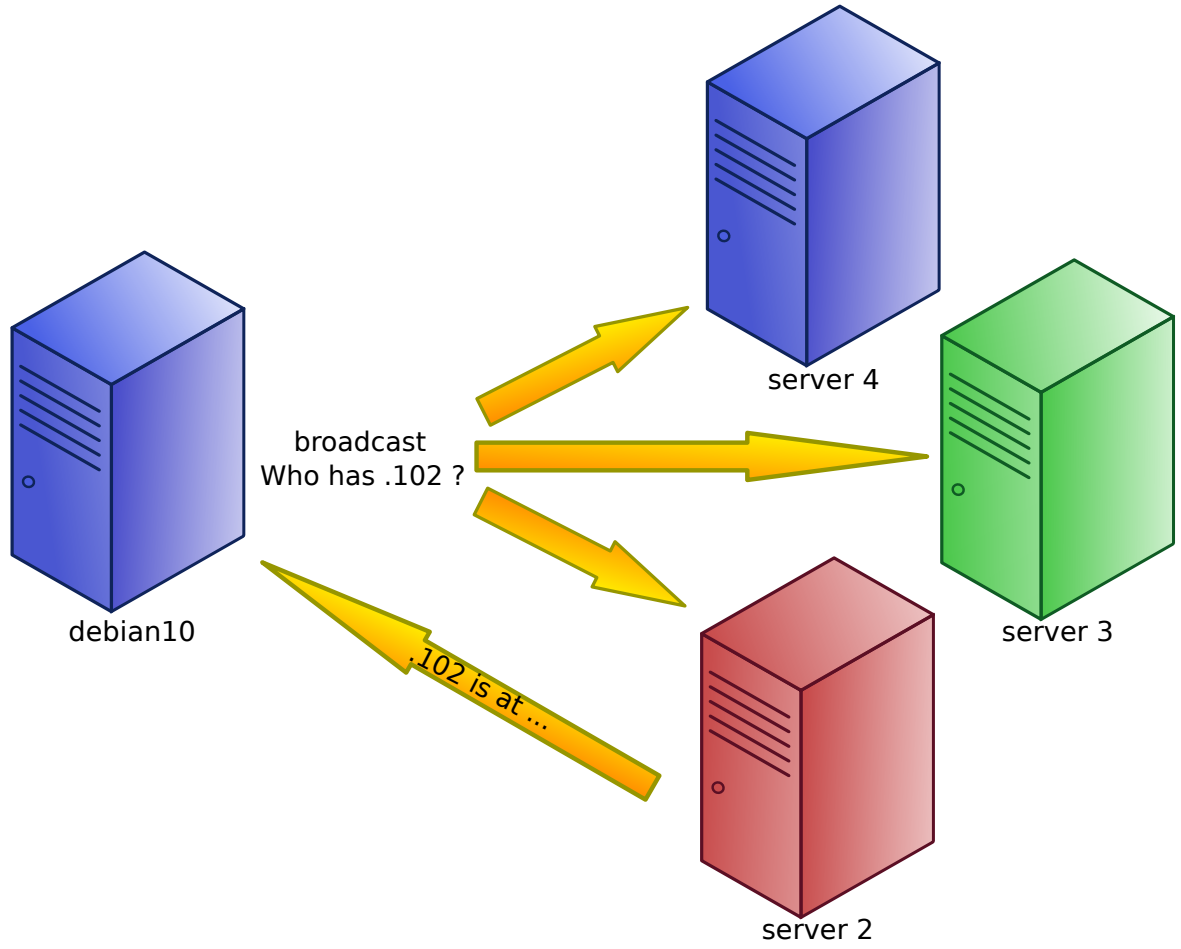
Introduction to protocols

You can skip this chapter if you don't wish to learn more about networking and TCP/IP.

66.1 ARP

We start with the **ARP** or **A**ddress **R**esolution **P**rotocol because this is often the first protocol on the network. The goal of this protocol is to find the **MAC-address** of a computer with a certain **IP-address**. For example when you type **ping 10.0.2.102** then your computer has a destination **IP-address** but no destination **MAC-address**. This destination **MAC-address** is required because it is the first six bytes of the Ethernet frame.

In this scenario the **ARP** protocol will construct an **Ethernet frame** with **ff:ff:ff:ff:ff:ff** as the destination MAC-address. This frame will be received by all computers on the local subnet. This frame has a simple question: **"Who has 10.0.2.102?"** and all computers will disregard this question, except the computer with this IP-address. This 10.0.2.102 computer will reply with an Ethernet frame containing its MAC-address.



You can try to use a sniffer like **tcpdump** or **Wireshark** to see this traffic, but this can be tricky to do because of the **ARP cache**. If there is a communication between two computers, then they will each have the IP-to-MAC-address translation of each other in the **ARP cache**.

Using this little script we managed to capture an **ARP request** and an **ARP reply**. We made sure there was no prior communication with the 10.0.2.102 server.

```
root@debian10:~# cat arp.sh
!/bin/bash

tcpdump -i -n enp0s8 -w dump2.pcap &
sleep 5
ping -c2 10.0.2.102
fg
root@debian10:~
```

To run it, we **source** the script, otherwise the **fg** command does not work. As soon as the **tcpdump -n -i enp0s8 -w dump2.pcap** appears we pressed Ctrl-c.

```
root@debian10:~# source arp.sh
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
PING 10.0.2.102 (10.0.2.102) 56(84) bytes of data.
64 bytes from 10.0.2.102: icmp_seq=1 ttl=64 time=1.11 ms
64 bytes from 10.0.2.102: icmp_seq=2 ttl=64 time=0.527 ms

--- 10.0.2.102 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 0.527/0.818/1.110/0.292 ms
tcpdump -n -i enp0s8 -w dump2.pcap
^C6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@debian10:~#
```

We now have a nice capture of the ARP protocol in action, right before the **ping** or **ICMP** packets.

```
root@debian10:~# tcpdump -nt -r dump2.pcap
reading from file dump2.pcap, link-type EN10MB (Ethernet)
ARP, Request who-has 10.0.2.102 tell 10.0.2.101, length 28
ARP, Reply 10.0.2.102 is-at 08:00:27:2f:13:ed, length 46
IP 10.0.2.101 > 10.0.2.102: ICMP echo request, id 1097, seq 1, length 64
IP 10.0.2.102 > 10.0.2.101: ICMP echo reply, id 1097, seq 1, length 64
IP 10.0.2.101 > 10.0.2.102: ICMP echo request, id 1097, seq 2, length 64
IP 10.0.2.102 > 10.0.2.101: ICMP echo reply, id 1097, seq 2, length 64
root@debian10:~#
```

The tcpdump **-e** option will show the MAC-addresses of this ARP request and reply. Notice the broadcast MAC-address ff:ff:ff:ff:ff:ff.

```
root@debian10:~# tcpdump -entr dump2.pcap arp
reading from file dump2.pcap, link-type EN10MB (Ethernet)
08:00:27:ff:0c:3c > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: Request who-has  ↩
    10.0.2.102 tell 10.0.2.101, length 28
08:00:27:2f:13:ed > 08:00:27:ff:0c:3c, ethertype ARP (0x0806), length 60: Reply 10.0.2.102  ↩
    is-at 08:00:27:2f:13:ed, length 46
root@debian10:~#
```

This dump can be downloaded with **wget http://linux-training.be/dump2.pcap** .

Note
The above story about **ARP** is true if the destination **IP-address** is on the same network (the same subnet) as the source computer. If the destination computer is on another network then the Ethernet frame will be sent to the **default gateway** (= a router).
And also if you ping to yourself, then there is no **ARP** request sent out.

66.2 UDP DNS

Next we will take a look at DNS or **D**omain **N**ame **S**ystem. DNS is an application layer protocol that works on top of the UDP protocol. The goal of DNS is to translate a name to an IP-address. Since people tend to remember names rather than IP-addresses, this is a very common action.

Setting up a DNS server is a couple of chapters later in this book. For now we are happy if we can capture a **DNS query** and a **DNS response** using tcpdump. We will use this little script, this time we **ping** to a name.

```
root@debian10:~# cat dns.sh
!/bin/bash

tcpdump -n -i enp0s8 -w dump3.pcap &
sleep 5
ping -c2 www.linux-training.be
fg
root@debian10:~
```

Again we source the script and we receive a whopping 14 packets this time. Maybe we caught something extra?

```
root@debian10:~# source dns.sh
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
PING www.linux-training.be (88.151.243.8) 56(84) bytes of data.
64 bytes from fosfor.openminds.be (88.151.243.8): icmp_seq=1 ttl=55 time=13.1 ms
64 bytes from fosfor.openminds.be (88.151.243.8): icmp_seq=2 ttl=55 time=13.3 ms

--- www.linux-training.be ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 3ms
rtt min/avg/max/mdev = 13.116/13.201/13.286/0.085 ms
tcpdump -n -i enp0s8 -w dump3.pcap
^C14 packets captured
14 packets received by filter
0 packets dropped by kernel
root@debian10:~#
```

Firstly, let us see if we caught any ARP packets this time. It seems we do, we caught the ARP request for the default gateway (10.0.2.1 in this case, yours will vary) and the ARP response.

```
root@debian10:~# tcpdump -tnr dump3.pcap arp
reading from file dump3.pcap, link-type EN10MB (Ethernet)
ARP, Request who-has 10.0.2.1 tell 10.0.2.101, length 28
ARP, Reply 10.0.2.1 is-at 52:54:00:12:35:00, length 46
root@debian10:~#
```

Secondly we take a look at the UDP packets in this capture. We caught eight UDP packets, of which the first two are not related to DNS, but to **NTP** (the **N**etwork **T**ime **P**rotocol). Our server wanted to know what time it is.

After the NTP packets are two DNS queries. One for the A record for **www.linux-training.be** and one for the quad A (AAAA) record for **www.linux-training.be**. Both queries go to the server with IP-address 8.8.8.8. The 8.8.8.8 server responds with an A record pointing to 88.151.243.8.

The last two lines in the screenshot are a query and a response for a PTR record, which we will discuss in the DNS chapters.

```
root@debian10:~# tcpdump -tnr dump3.pcap udp
reading from file dump3.pcap, link-type EN10MB (Ethernet)
IP 10.0.2.101.38390 > 85.88.55.5.123: NTPv4, Client, length 48
IP 85.88.55.5.123 > 10.0.2.101.38390: NTPv4, Server, length 48
IP 10.0.2.101.48961 > 8.8.8.8.53: 7889+ A? www.linux-training.be. (39)
IP 10.0.2.101.48961 > 8.8.8.8.53: 28383+ AAAA? www.linux-training.be. (39)
IP 8.8.8.8.53 > 10.0.2.101.48961: 28383 0/1/0 (100)
IP 8.8.8.8.53 > 10.0.2.101.48961: 7889 1/0/0 A 88.151.243.8 (55)
IP 10.0.2.101.45778 > 8.8.8.8.53: 21726+ PTR? 8.243.151.88.in-addr.arpa. (43)
IP 8.8.8.8.53 > 10.0.2.101.45778: 21726 1/0/0 PTR fosfor.openminds.be. (76)
root@debian10:~#
```

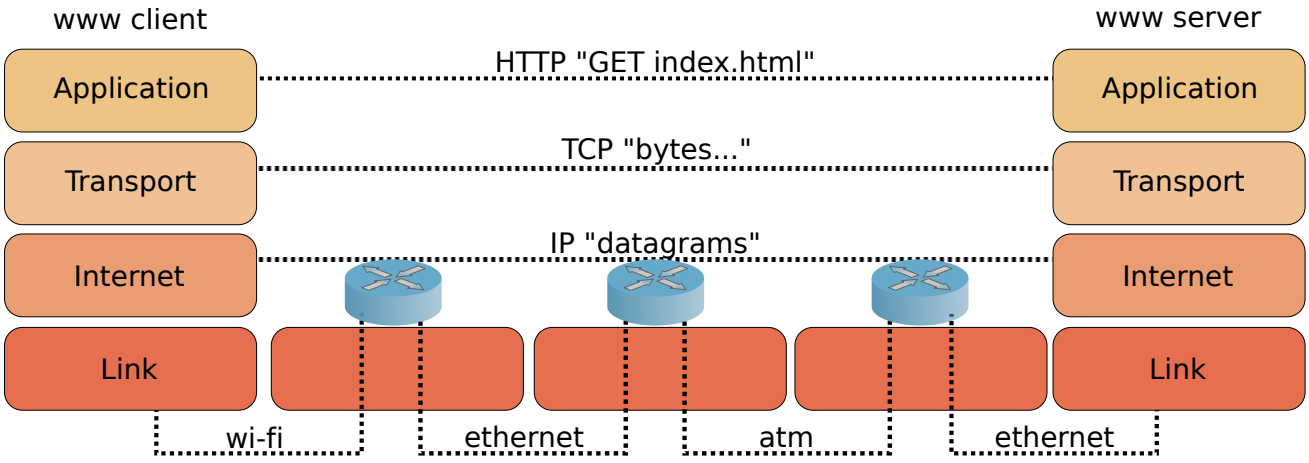
Thirdly the last four packets of this tcpdump are the ping echo requests and replies. They go from our server at 10.0.2.101 to the linux-training.be server at 88.151.243.8.

```
root@debian10:~# tcpdump -tnr dump3.pcap icmp
reading from file dump3.pcap, link-type EN10MB (Ethernet)
```

```
IP 10.0.2.101 > 88.151.243.8: ICMP echo request, id 768, seq 1, length 64
IP 88.151.243.8 > 10.0.2.101: ICMP echo reply, id 768, seq 1, length 64
IP 10.0.2.101 > 88.151.243.8: ICMP echo request, id 768, seq 2, length 64
IP 88.151.243.8 > 10.0.2.101: ICMP echo reply, id 768, seq 2, length 64
root@debian10:~#
```

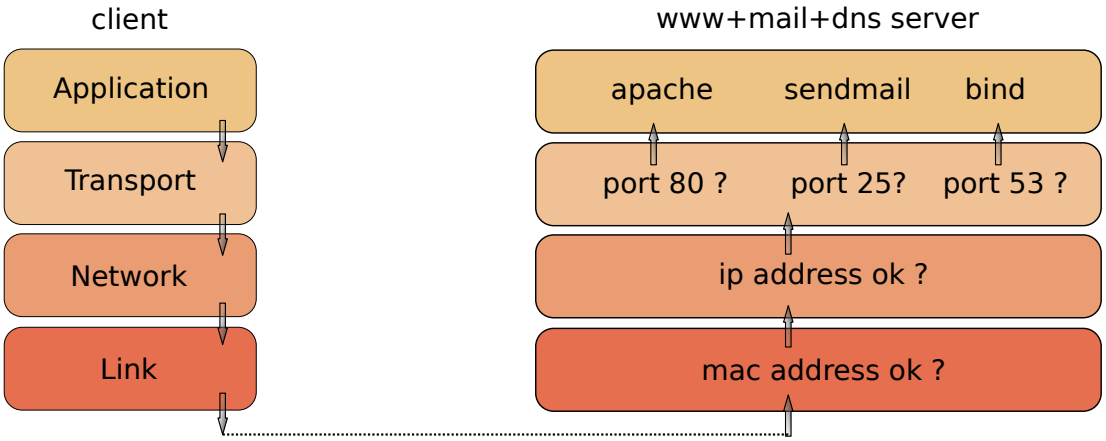
66.3 WWW example

When you surf with Firefox or Chrome to a website, then a connection is made between your browser (Firefox or Chrome) and the WWW server. This connection happens in the **Application** Layer. But applications don't know about *connections* or *networks* so the *connection* is made by TCP in the **Transport** layer. TCP however does not know about delivering packets on a network so the **Network** layer is used to deliver the datagram to the correct destination IP-address, probably passing over several **routers**. At the link layer the **Ethernet** protocol is responsible for LAN communication and for delivering the **Ethernet frame** to the next computer (often a router). And at the bottom of the layers there is hardware necessary that can transfer electrical signals or radio waves from one point to the next. This hardware does not know anything about HTTP, TCP or IP.



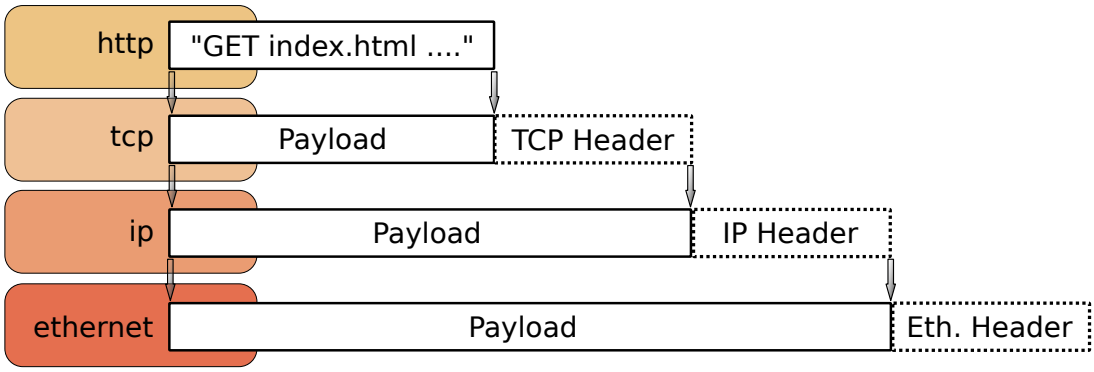
On the receiving computer, the WWW server, the electrical signals are received by the network card and are translated into bytes. Then the destination MAC-address will be verified, if this is correct then the Ethernet header will be removed and the datagram will be given to IP. IP will verify the IP-address, strip its header and give the packet to TCP. TCP will verify the port, 80 or 443 in this case, and will also strip its header and give the rest to the WWW server application (for example Apache).

Packets with other destination ports will be given to other applications.



66.4 Encapsulation

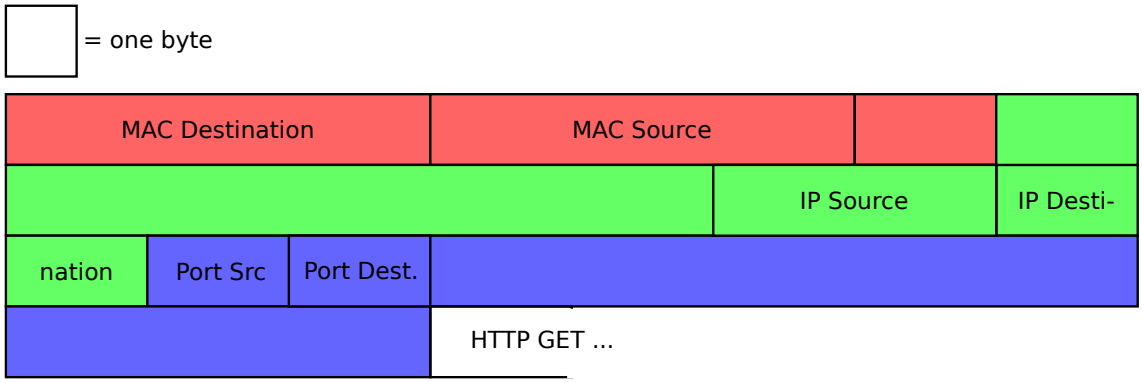
HTTP is the protocol used between the browser (for example Firefox) and the web server (often Apache). TCP adds a TCP header to this protocol, and has the Application layer protocol as **payload** or **data**. IP adds an IP header and has the TCP header and payload as its **payload**. Ethernet adds an Ethernet header and has the IP header and payload as **payload**, and so the frame goes to the hardware and ends up in electrical signals or radio waves (with their own header), to be received on the next computer.



Note
Note that the above picture is simplified, the headers are not really appended.

66.5 An Ethernet Frame

An **Ethernet frame** in the case of a browser's first contact with a web server, contains an Ethernet header, an IP header, a TCP header and an HTTP GET request. The image below shows the location of these bytes in an Ethernet frame.



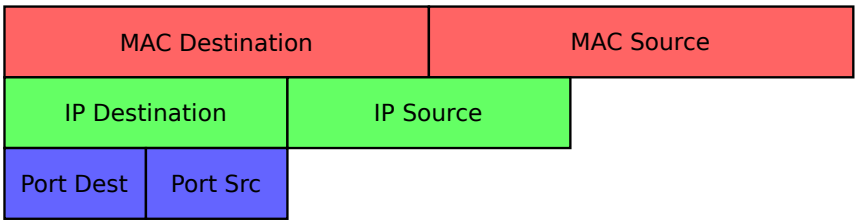
The **Ethernet** header is in red. The first six bytes of the frame is the **destination MAC-address**, the next six bytes is the **source MAC-address**.

The **IP** header is in green and contains at the end four bytes for the **destination IP-address** and before that four bytes for the **source IP-address**.

In blue is the **TCP** header which starts with two bytes for the **source port** and two bytes for the **destination port**. After the TCP header we find the HTTP protocol (in white), a typical Ethernet frame can be up to 1500 bytes in length.

66.6 Simplified packet

In this book we will use this simplified picture for an Ethernet frame. Note that we always put the destination before the source, this improves readability.


MAC Destination
MAC Source
IP Destination
IP Source
Port Dest
Port Src

IP Desti-

66.7 Cheat sheet

Table 66.1: Protocols

term	**explanation**
ARP	Address Resolution Protocol, resolves MAC-to-IP relations.
ff:ff:ff:ff:ff:ff	The broadcast MAC address.
DNS	Domain Name System, resolves name-to-IP relations (see the DNS chapters).
frame	A packet on the network.
datagram	A frame without the Ethernet header.
packet	A term used to describe frames, datagrams, and TCP/UDP packets.
wi-fi	A trademark for a wireless Ethernet.
Ethernet	A LAN protocol for connecting network adapters locally.

66.8 Practice

1. Imagine a network were there has been no traffic on yet, nothing. And you perform a **wget http://linux-training.be**. List all the packets you will see on your server's network interface in the correct order. Suppose the DNS server is on the network and provides a complete answer from its cache, suppose the gateway is not a DNS server.
2. Try to get as many of the packets as possible with tcpdump writing to a file. Open this file in Wireshark and verify with your answer from the previous question. Note that it may not be possible to clear the ARP/DNS cache on all network devices.

66.9 Solution

1. Imagine a network were there has been no traffic on yet, nothing. And you perform a **wget http://linux-training.be**. List all the packets you will see on your server's network interface in the correct order. Suppose the DNS server is on the network and provides a complete answer from its cache, suppose the gateway is not a DNS server.

```
ARP query for DNS MAC
ARP reply with DNS MAC
DNS query for A linux-training.be
DNS query for AAAA linux-training.be
DNS response for linux-training.be
ARP query for gateway MAC
ARP reply with gateway MAC
TCP SYN with linux-training.be
TCP SYN,ACK with linux-training.be
TCP ACK with linux-training.be
HTTP GET with linux-training.be
several TCP packets with web page
TCP FIN with linux-training.be
TCP ACK with linux-training.be
TCP FIN with linux-training.be
TCP ACK with linux-training.be
```

2. Try to get as many of the packets as possible with tcpdump writing to a file. Open this file in Wireshark and verify with your answer from the previous question. Note that it may not be possible to clear the ARP/DNS cache on all network devices.

Chapter 67

Binding and bonding

67.1 Binding IP-addresses

The idea behind **binding** is to enable multiple IP-addresses to a single network adapter. The reasoning can be that the adapter is connected to multiple subnets, or that you have applications that require a dedicated IP-address.

67.1.1 Binding with the ip command

The new way of adding IP-addresses, using the **ip** command is to add **up** and **down** lines in **/etc/network/interfaces**. In the screenshot below we added two IP-addresses to the **enp0s8** adapter (Look behind the **up** lines).

```
root@debian10:~# tail /etc/network/interfaces
auto enp0s8
allow-hotplug enp0s8
iface enp0s8 inet static
  address 10.0.2.101
  netmask 255.255.255.0
  gateway 10.0.2.1
  up ip addr add 10.0.2.201/24 dev enp0s8 label enp0s8:0
  down ip addr del 10.0.2.201/24 dev enp0s8 label enp0s8:0
  up ip addr add 10.0.2.202/24 dev enp0s8 label enp0s8:1
  down ip addr del 10.0.2.202/24 dev enp0s8 label enp0s8:1
root@debian10:~#
```

You can now use the **ifdown** and **ifup** commands on the **enp0s8** interface, or a **systemctl restart networking** command. The latter will briefly interrupt your **ssh** session though.

The configuration of **enp0s8** now looks like in this screenshot.

```
root@debian10:~# ip a show dev enp0s8
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group  ↩
    default qlen 1000
    link/ether 08:00:27:ff:0c:3c brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.101/24 brd 10.0.2.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet 10.0.2.201/24 scope global secondary enp0s8:0
       valid_lft forever preferred_lft forever
    inet 10.0.2.202/24 scope global secondary enp0s8:1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:c3c/64 scope link
       valid_lft forever preferred_lft forever
root@debian10:~#
```

The **ifconfig** command from **net-tools** cannot handle this new configuration method.

67.2 Bonding network adapters

The idea of **bonding** is the reverse of **binding**. Instead of adding multiple IP-addresses to a single adapter we assign multiple adapters to a single IP-address. This can double (or triple. . .) the network throughput and can also serve as a failover redundancy.

Bonding can be achieved with the **ifenslave** package or with **systemd**. We first look at the **ifenslave** method.

67.3 Bonding with ifenslave

First add two network adapters to the same network, the same subnet. Then for **bonding** with **ifenslave** to work we need to install the **ifenslave** package.

```
root@debian10:~# aptitude install ifenslave
```

Then remove the configuration of the individual adapters from **/etc/network/interfaces** and configure the **bond0** interface as in this screenshot.

```
root@debian10:~# tail -7 /etc/network/interfaces
auto bond0
iface bond0 inet static
  address 10.0.2.101
  netmask 255.255.255.0
  gateway 10.0.2.1
  slaves enp0s8 enp0s9
  bond-mode active-backup
root@debian10:~#
```

Next we restart the networking with **systemctl restart networking** and we see a new interface named **bond0** with our assigned IP-address.

```
root@debian10:~# ip a show dev bond0
5: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group  ↩
    default qlen 1000
    link/ether 08:00:27:ff:0c:3c brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.101/24 brd 10.0.2.255 scope global bond0
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feff:c3c/64 scope link
       valid_lft forever preferred_lft forever
root@debian10:~#
```

67.4 Bonding with systemd

No extra software is needed for **bonding** to work with **systemd**, but we do need to enalbe the **systemd-networkd** service.

```
root@debian10:~# systemctl enable systemd-networkd
```

First we need to create a file in **/etc/systemd/network/** that ends in **.netdev** and that contain a definition of our **bond0** device.

```
root@debian10:~# cat /etc/systemd/network/bond0.netdev
[NetDev]
Name=bond0
Description=Bonding interface
Kind=bond

[Bond]
Mode=active-backup
root@debian10:~#
```

Next we need to tell **systemd**, in a **.network** file, which devices should be used, using their **PCI** address. You can find these **PCI** addresses by executing **lspci -D | grep Ethernet** from within vi.

```
root@debian10:~# cat /etc/systemd/network/bond0.network
[Match]
Path=pci-0000:00:08.0
Path=pci-0000:00:09.0

[Network]
Bond=bond0
root@debian10:~#
```

And finally we need a **.network** file to configure our **bond0** device with an IP-address etcetera. Also remove any existing configuration on these adapters from **/etc/network/interfaces** .

```
root@debian10:~# cat /etc/systemd/network/bonding.network
[Match]
Name=bond0

[Network]
Address=10.0.2.101
Gateway=10.0.2.1
root@debian10:~#
```

Make sure the previous bonds are gone and unconfigured. Maybe run a **systemctl networking restart** to unconfigure them. Then run **systemctl restart systemd-networkd** . Then two **SLAVE** devices and one **MASTER** bond device should be there, as in this screenshot.

```
root@debian10:~# ip a | tail
3: enp0s8: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond0  ↩
    state UP group default qlen 1000
    link/ether 02:6b:96:52:9b:09 brd ff:ff:ff:ff:ff:ff
4: enp0s9: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond0  ↩
    state UP group default qlen 1000
    link/ether 02:6b:96:52:9b:09 brd ff:ff:ff:ff:ff:ff
5: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group  ↩
    default qlen 1000
    link/ether 02:6b:96:52:9b:09 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.101/8 brd 10.255.255.255 scope global bond0
       valid_lft forever preferred_lft forever
    inet6 fe80::6b:96ff:fe52:9b09/64 scope link
       valid_lft forever preferred_lft forever
root@debian10:~#
```

67.5 Cheat sheet

Table 67.1: Binding and bonding

command/file/term	**explanation**
binding	Putting multiple IP addresses on one network adapter.
/etc/network/interfaces	Location to configure **binding**.
ifup foo	Activate the network configuration of the **foo** adapter.
bonding	Putting multiple network adapters behind one IP address.
apt-get install ifenslave	Install the **bonding** software.
/etc/network/interfaces	Location to configure **bonding**.
systemctl restart networking	Activate the **bonding** configuration.
systemd bonding	An alternative way to configure **bonding**.
/etc/systemd/network/bond0.netdev	Location to configure **systemd bonding** interface name.
/etc/systemd/network/bond0.network	Location to configure **systemd bonding** interface members.
/etc/systemd/network/bonding.network	Location to configure **systemd bonding** interface IP-address.
lspci -D	Tool to obtain PCI addresses.

67.6 Practice

1. Bind two extra IP-addresses to an existing network adapter. Test that you can ping all three IP-addresses.
2. Bond two adapters behind one IP-address using the **ifenslave** method.
3. Bond two adapters behind one IP-address using the **systemd** method.

67.7 Solution

1. Bind two extra IP-addresses to an existing network adapter. Test that you can ping all three IP-addresses.

```
vi /etc/network/interfaces    # Configure like in the theory
systemctl restart networking
ip a

ping 10.0.2.101               # From another computer
ping 10.0.2.201               # From another computer
ping 10.0.2.202               # From another computer
```

2. Bond two adapters behind one IP-address using the **ifenslave** method.

```
apt-get install ifenslave
vi /etc/network/interfaces    # Configure like in the theory
systemctl restart networking
ip a show dev bond0
```

3. Bond two adapters behind one IP-address using the **systemd** method.

```
systemctl enable systemd-networkd
vi /etc/systemd/network/bond0.netdev
vi /etc/systemd/network/bond0.network
vi /etc/systemd/network/bonding.network
systemctl restart systemd-networkd
```

Chapter 68

Network monitoring

68.1 Sniffing the network

We already had a chapter on sniffing the network with **tcpdump** and **Wireshark**. These can serve as the ultimate tools for network troubleshooting.

68.2 netstat ss

68.3 ntop

68.4 iftop

```
root@server2:~# fuser -u ssh/tcp
ssh/tcp:              424(root)   433(root)   445(paul)
root@server2:~#
```

68.5 iptraf

68.6 nmon

68.7 nmap

```
```

===

```
```

===

```
```

68.8 Practice

68.9 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 69

Introduction to ssh

69.1 ssh

The legacy commands **telnet**, **rsh** and **rlogin** have all been replaced by **ssh**, which stands for **s**ecure **sh**ell. This application and protocol encrypts all traffic and eliminates eavesdropping, connection hijacking and other attacks. Debian 10 installs the **OpenSSH client** by default. The **ssh** command is one of the most used applications in the world.

OpenSSH is developed and maintained by the **OpenBSD Project** .

69.2 Using ssh

69.2.1 First time connection

The format to use ssh is to type **ssh user@IP-address** where you can omit the user if it is the same as your current user, and were **IP-address** can also be a **hostname** or DNS name.

When you connect to a Debian 10 server for the first time, then **ssh** will warn you that this is an unknown system and will show you a fingerprint of this system. If you answer **yes**, then this information will be stored, and will be **verified** each time you connect to that **IP-address**.

```
paul@debian10:~$ ssh paul@10.0.2.102
The authenticity of host 10.0.2.102 (10.0.2.102) can't be established.
ECDSA key fingerprint is SHA256:iuWk63GLd+kDvZO/b1wrtvhTyruHsMxgLxteHcuBgJo.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 10.0.2.102 (ECDSA) to the list of known hosts.
paul@10.0.2.102's password:
Linux server2 4.19.0-5-amd64 #1 SMP Debian 4.19.37-5+deb10u2 (2019-08-08) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Mon Sep 23 11:22:06 2019 from 192.168.56.1
paul@server2:~$
```

69.2.2 Manually verifying the fingerprint

You could physically log on to the server and verify this fingerprint with **ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key.pub**, as shown in this screenshot.

```
root@server2:~# ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key.pub
256 SHA256:iuWk63GLd+kDvZO/b1wrtvhTyruHsMxgLxteHcuBgJo root@server2 (ECDSA)
root@server2:~#
```

69.2.3 Automatic verifying of the fingerprint

All subsequent logons to this server (to this **IP-address**) will verify this fingerprint. If another server hijacks the **IP-address** then you get a warning like in this screenshot.

```
paul@debian10:~$ ssh paul@10.0.2.102
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:wUEbNVGpMVew1TruJOhtvv0dPjvjJVUIsbw9Vho4hUo.
Please contact your system administrator.
Add correct host key in /home/paul/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/paul/.ssh/known_hosts:3
  remove with:
  ssh-keygen -f "/home/paul/.ssh/known_hosts" -R "10.0.2.102"
ECDSA host key for 10.0.2.102 has changed and you have requested strict checking.
Host key verification failed.
paul@debian10:~$
```

If you somehow installed a new server with this **IP-address**, and without copying the **ssh** keys, then you can follow the instructions on the removal of this message.

69.2.4 /etc/ssh/ssh_config

The **OpenSSH client** is configured in the **/etc/ssh/ssh_config** file. The defaults in this file are usually okay. Note that the file configures **port 22**, which is the default port for **ssh**. And it uses **Protocol 2** only, since **Protocol 1** is no longer considered secure.

```
root@debian10:~# grep -e Prot -e Port /etc/ssh/ssh_config
#   Port 22
#   Protocol 2
root@debian10:~#
```

69.2.5 Showing a visual key

You can configure **ssh** to always show a visual key by setting the **VisualHostKey** to **yes** in the **/etc/ssh/ssh_config** file.

```
root@debian10:~# vi /etc/ssh/ssh_config
root@debian10:~# grep Visual /etc/ssh/ssh_config
VisualHostKey yes
root@debian10:~#
```

Connecting via SSH now shows the fingerprint as usual, but also includes a **sha256** hash of the public key of the server.

```
root@debian10:~# ssh paul@10.0.2.102
Host key fingerprint is SHA256:iuWk63GLd+kDvZO/b1wrtvhTyruHsMxgLxteHcuBgJo
+---[ECDSA 256]---+
|                 |
|       .         |
|      . .        |
|     o   . .     |
|    E o.S . o    |
|     *..+ .o +o  |
|    + +o.O.=+= . |
|     =.oOo=.X o  |
|   .+..o+*+**B   |
+----[SHA256]-----+
paul@10.0.2.102's password:
```

You can verify this on the server with **ssh-keygen** as shown in this screenshot.

```
root@server2:/etc/ssh# ssh-keygen -lv -E sha256 -f ssh_host_ecdsa_key.pub
256 SHA256:iuWk63GLd+kDvZO/b1wrtvhTyruHsMxgLxteHcuBgJo root@server2 (ECDSA)
+---[ECDSA 256]---+
|                 |
```

```
|        .        |
|       . .       |
|      o   . .    |
|     E o.S . o   |
|      *..+ .o +o |
|     + +o.O.=+= . |
|      =.oOo=.X o |
|   .+..o+*+**B   |
+----[SHA256]-----+
root@server2:/etc/ssh#
```

69.3 ssh server

With a client comes a server, so for this chapter we also install the **OpenSSH server** with **apt-get install openssh-server** (if you have not done this already at install time).

```
apt-get install openssh-server
```

69.4 /etc/ssh/sshd_config

The **OpenSSH server** is configured in **/etc/ssh/sshd_config**, note the extra **d** for **daemon** compared to the client configuration file.

69.4.1 Changing the ssh port

The default ssh server listens on **port 22**. This can be changed in the configuration file to any other port, for example **tcp port 8472** as in this screenshot.

```
root@debian10:~# vi /etc/ssh/sshd_config
root@debian10:~# grep Port /etc/ssh/sshd_config
#Port 22
Port 8472
#GatewayPorts no
root@debian10:~# systemctl restart sshd.service
root@debian10:~#
```

Note that adding the **Port 8472** line here will also disable **Port 22**, so now the server is only reachable via **port 8472**. You have to configure the client for this, or use the **-p** option.

```
ssh -p8472 192.168.56.101
```

69.4.2 PermitRootLogin

Debian 10 by default does not allow remote logins of the **root** account with a password. This is set in the **#PermitRootLogin prohibit-password** line. You can enable **root** logins by setting this value to **yes**.

If you have a server with an IP-address on the Internet then hackers will try to log in to this server using **root** and a password.

```
root@debian10:~# grep ^#PermitRoot /etc/ssh/sshd_config
#PermitRootLogin prohibit-password
root@debian10:~#
```

69.5 public private key

To improve on security you can set up **ssh** with a **key pair**, a **public key** and **a private key**. We will skip on the mathematics of large prime numbers behind this and go straight to setting this up.

69.5.1 generating a key pair

A public-private key pair can be generated for you using **ssh-keygen** as shown in this screenshot. This command will create the **~/.ssh** directory if not already present.

```
paul@debian10:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/paul/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/paul/.ssh/id_rsa.
Your public key has been saved in /home/paul/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:wo86IAGznHzTwZu6jjkP8jfXEfQdZDQFW6op7WNlYPg paul@debian10
The key's randomart image is:
---[RSA 2048]----
|    .      o*oo  |
|o    o  .. ..=   |
|o. . +...o.o.   |
|o o =  ...    |
| .. o o S.E o    |
|. ..   .o o     |
|.o .. ....      |
|.o+... .. .     |
| o=.+           |
----[SHA256]-----
paul@debian10:~$
```

69.5.2 the key pair

After this command you have two new files in **~/.ssh** . The first file is **id_rsa**, this is your private key, you guard it with your life. If a hacker can copy this key, and you gave an empty passphrase, then he can use the key just like you.

The second file is **id_rsa.pub**. This is your public key, this is the key that you give to others. This is the key that we will copy to a target server to logon without using a password.

```
paul@debian10:~$ ls -l .ssh/
total 12
-rw------- 1 paul paul 1823 Sep 23 13:02 id_rsa
-rw-r--r-- 1 paul paul  395 Sep 23 13:02 id_rsa.pub
-rw-r--r-- 1 paul paul  666 Sep 23 11:27 known_hosts
paul@debian10:~$
```

69.5.3 copying the key to a server

The goal of copying a **public key** to a server is to enable us to logon to that server without using a password. We prove our identity with the **private key** (that we alone have!).

```
paul@debian10:~$ ssh-copy-id paul@10.0.2.102
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/paul/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any ↩
    that are already installed
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it  ↩
    is to install the new keys
Host key fingerprint is SHA256:iuWk63GLd+kDvZO/b1wrtvhTyruHsMxgLxteHcuBgJo
---[ECDSA 256]---
|                 |
|       .         |
|      . .        |
|     o   . .     |
|    E o.S . o    |
|     ..+ .o o  |
|    + +o.O.== . |
|     =.oOo=.X o  |
|   ...o**B   |
+----[SHA256]-----
paul@10.0.2.102's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh paul@10.0.2.102"
and check to make sure that only the key(s) you wanted were added.

paul@debian10:~$
```

What this command does is copy the **id_rsa.pub** file and append it to the **.ssh/authorized_keys** file on the server (in that user's home directory). You can also do this manually.

69.5.4 passwordless ssh

You have now set up **ssh** without password to that user on that server. Typing **ssh paul@10.0.2.102** will automatically log me on, and there I can check the **authorized_keys** file.

```
paul@server2:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDXy1HKu6jX8HvZCD9Plat7I651X/ejsjPo5IUORgvGX635va\
NqOZ6zTGRJoUX8tkxd/3Rog9/pbtEYRxc+dj8y/Ok4YUB6/q7q/7ARTLANEY2vcqj4oC7P9HdUD4UDJPTi8BRp\
M3MZNtskDQA7H1NfSiPFAHWhzJL2RXJTflLv8LJu3OT9sPmzd7vIXPVisOXDsssdJX9Wmq19gq/DzUWX/VmmZs\
zB83cK6EKJf3pSjJhpHCG8Ps+TRnO7Z7yKAe8HQXt1+WxG+6rKa4tD++lrwZLmYSQRSMIJ9rL005GB35TMw8JJ\
x+gfcGpOS3eGD/5f7/6LYWVmBCR12Qv5V+pf paul@debian10
paul@server2:~$
```

69.5.5 ssh remote commands

I can now also remotely execute commands on **server2** without needing a password. You can even disable the password for the **paul** user. (I have disabled the **VirtualHostKey yes** for this screenshot.)

```
paul@debian10:~$ ssh 10.0.2.102 'cat .ssh/authorized_keys'
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDXy1HKu6jX8HvZCD9Plat7I651X/ejsjPo5IUORgvGX635va\
NqOZ6zTGRJoUX8tkxd/3Rog9/pbtEYRxc+dj8y/Ok4YUB6/q7q/7ARTLANEY2vcqj4oC7P9HdUD4UDJPTi8BRp\
M3MZNtskDQA7H1NfSiPFAHWhzJL2RXJTflLv8LJu3OT9sPmzd7vIXPVisOXDsssdJX9Wmq19gq/DzUWX/VmmZs\
zB83cK6EKJf3pSjJhpHCG8Ps+TRnO7Z7yKAe8HQXt1+WxG+6rKa4tD++lrwZLmYSQRSMIJ9rL005GB35TMw8JJ\
x+gfcGpOS3eGD/5f7/6LYWVmBCR12Qv5V+pf paul@debian10
paul@debian10:~$
```

69.6 scp

The **scp** tool (short for **s**ecure **co**p**y**) is useful to copy files from one server to another, and uses **ssh** to perform this copy. Since we set up **ssh** without password, this will also work without password.

```
paul@debian10:~$ scp dates.txt 10.0.2.102:/home/paul/
dates.txt                                     100% 1118   867.0KB/s   00:00
paul@debian10:~$
```

69.7 ssh -X

You can use any graphical application over **ssh** with the **X11forwarding** or using the **-X** option. So you can run **xeyes** or **firefox** on a server, while viewing the screen on your laptop.

In this screenshot I connect from my laptop to a server and run the **xeyes** program on the server. I did an **apt-get install x11-apps** to get a minimal GUI on the **debian10** server.

```
paul@MBDebian~/lt4/git$ ssh -X 192.168.56.101
paul@192.168.56.101's password:
Linux debian10 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2 (2019-08-28) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Mon Sep 23 19:43:29 2019 from 192.168.56.1
paul@debian10:~$ xeyes
```

69.8 ssh -D proxy

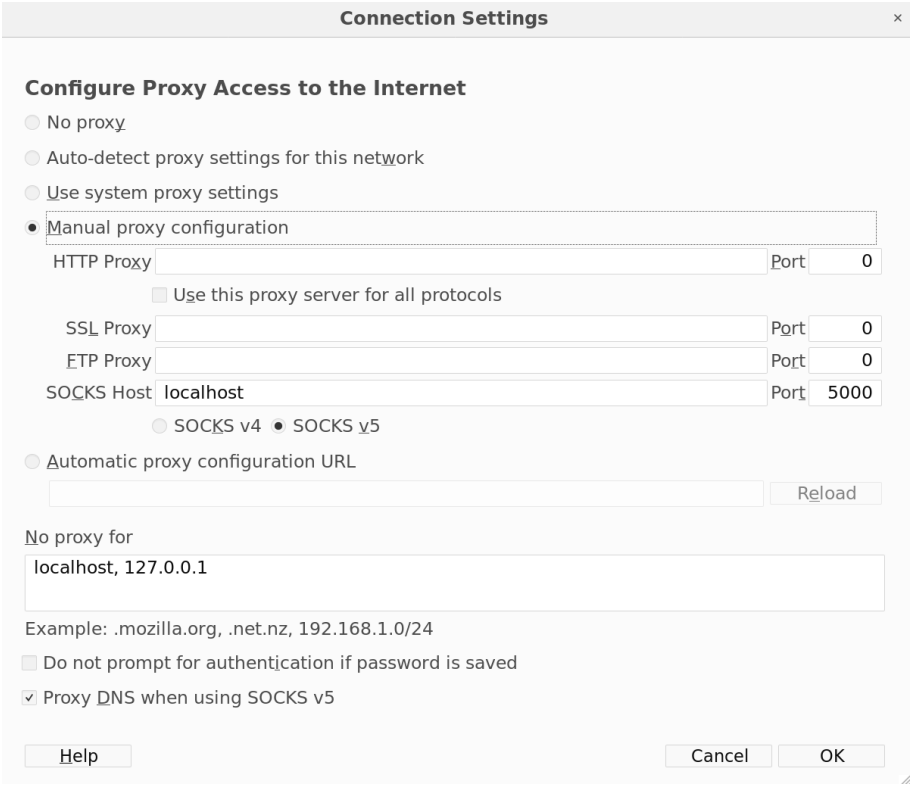
If you have ssh access to a server on the internet, then you can forward all your browser traffic over this connection, thus breaking out of restrictive firewalls.

To achieve this start an **ssh tunnel** with the **ssh -D 5000** command as in this screenshot. It will serve as a **socks proxy** on localhost:5000 for your browser.

```
paul@MBDebian~$ ssh -D 5000 -q -N paul@example.com
```

You may need to open port 443 on your server for ssh connections, since many firewalls block port 22 for outgoing connections.

This tunnel from a laptop to a server stays open until you press **Ctrl-c**, configure your browser as in this screenshot.


Connection Settings
Configure Proxy Access to the Internet
No proxy
Auto-detect proxy settings for this network
Use system proxy settings
Manual proxy configuration
HTTP Proxy
Port
0
Use this proxy server for all protocols
SSL Proxy
Port
0
FTP Proxy
Port
0
SOCKS Host
localhost
Port
5000
SOCKS v4
SOCKS v5
Automatic proxy configuration URL
Reload
No proxy for
localhost, 127.0.0.1
Example: .mozilla.org, .net.nz, 192.168.1.0/24
Do not prompt for authentication if password is saved
Proxy DNS when using SOCKS v5
Help
Cancel
OK

69.9 Cheat sheet

Table 69.1: ssh

command	explanation
ssh	**S**ecure **SH**ell which replaces telnet, rsh and rlogin.
ssh foo@bar	Connect with user **foo** to computer **bar**.
/etc/ssh/ssh_config	The **ssh** client configuration.
/etc/ssh/sshd_config	The **ssh** server configuration.
apt-get install openssh-server	Install the **ssh** server software.
ssh-keygen	Tool to generate and verify keys.
ssh-keygen -t rsa	Generate an **ssh** key pair.
ssh-copy-id	Copy a public key to another computer.
~/.ssh/authorized_keys	Store for remote public keys.
scp	Tool to copy files over **ssh** from one computer to another.
ssh -X	A way to pipe graphical applications over **ssh**.
ssh -D	A socks proxy over **ssh**.

69.10 Practice

1. Start two Debian servers and connect from one (the client) to the other (the server) with ssh.
2. Verify the fingerprint you receive from the server.
3. Add 8472 as a port for ssh connection on your server.
4. Connect to this new port on the server via ssh on the client.
5. Create a key pair on your client for passwordless ssh.
6. Copy the public key to the server.
7. Test the passwordless connection.
8. Copy a directory with **scp** to your server.

69.11 Solution

1. Start two Debian servers and connect from one (the client) to the other (the server) with ssh.

```
ssh paul@10.0.2.102      # Use your username and your server's IP-address
```

2. Verify the fingerprint you receive from the server.

```
ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key.pub      # On the server
```

3. Add 8472 as a port for ssh connection on your server.

```
root@debian10:~# vi Port /etc/ssh/sshd_config
root@debian10:~# grep Port /etc/ssh/sshd_config
Port 22
Port 8472
#GatewayPorts no
root@debian10:~# systemctl restart sshd
root@debian10:~#
```

4. Connect to this new port on the server via ssh on the client.

```
ssh -p8472 paul@10.0.2.102
```

5. Create a key pair on your client for passwordless ssh.

```
ssh-keygen -t rsa
ls -l ~/.ssh
```

6. Copy the public key to the server.

```
ssh-copy-id paul@10.0.2.102
```

7. Test the passwordless connection.

```
ssh paul@10.0.2.102
```

8. Copy a directory with **scp** to your server.

```
scp -r backup/ paul@10.0.2.102:~
```

Chapter 70

Network file system

70.1 NFS

An **NFS server** is a service on the network that shares directories (and the files in the directories). NFS is often used for Linux to Linux (or Unix to Unix) sharing. If you want to share directories for Microsoft computers then take a look at the Samba chapters.

NFS was developed by SUN in 1984 and released as an **open standard** in RFC 1094 in 1989 as NFSv2. NFSv3 and NFSv4 were also released as RFC's allowing anyone to implement these protocols. The last update was NFSv4.2 in November 2016. NFSv4 uses port 2049.

```
https://tools.ietf.org/rfc/rfc1094.txt
https://tools.ietf.org/rfc/rfc1813.txt
https://tools.ietf.org/rfc/rfc3010.txt
https://tools.ietf.org/rfc/rfc3530.txt
https://tools.ietf.org/rfc/rfc5661.txt
https://tools.ietf.org/rfc/rfc7530.txt
https://tools.ietf.org/rfc/rfc7862.txt
```

70.2 NFS server

70.2.1 Installing the NFS server

We will use our Debian 10 named **server2** as an NFS server. We install **nfs-kernel-server** and some of its dependencies here.

```
root@server2:~# apt-get install nfs-kernel-server
```

70.2.2 Exporting directories

NFS shares directories over the network, so we need to decide on a directory. In this screenshot we use **/srv/project42** and create a file in it. Then we add it to the **/etc/exports** file.

```
root@server2:~# mkdir /srv/project42
root@server2:~# chmod 777 /srv/project42/
root@server2:~# echo Welcome to this project > /srv/project42/hello.txt
root@server2:~# vi /etc/exports
root@server2:~# echo '/srv/project42  *(rw,sync,no_subtree_check)' >> /etc/exports
root@server2:~#
```

70.2.3 Restarting the service

The next step is to restart the service with **systemctl** and then hop over to the client for testing of this network share.

```
root@server2:~# systemctl restart nfs-kernel-server.service
root@server2:~# systemctl status nfs-kernel-server.service
* nfs-server.service - NFS server and services
   Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
   Active: active (exited) since Tue 2019-09-24 14:54:29 CEST; 55s ago
  Process: 19197 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
  Process: 19198 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited, status=0/SUCCESS)
 Main PID: 19198 (code=exited, status=0/SUCCESS)

Sep 24 14:54:29 server2 systemd[1]: Starting NFS server and services...
Sep 24 14:54:29 server2 systemd[1]: Started NFS server and services.
root@server2:~#
```

70.3 NFS client

70.3.1 Mounting the NFS share

On the **debian10** client computer we install **nfs-common** and create a directory to use as a **mount point**. The **mount** command is very similar to mounting a partition or block device.

```
root@debian10:~# apt-get install nfs-common
<output truncated>
root@debian10:~# mkdir /home/project42
root@debian10:~# mount 192.168.56.102:/srv/project42 /home/project42
root@debian10:~#
```

After the **mount** command you can use the mount point as if it was a normal directory.

70.3.2 Seeing the connection

The NFS connection over port 2049 is visible with **ss** (or netstat) and also shows up when using the **df** command.

```
root@debian10:~# ss -napt | grep 2049
ESTAB    0        0            192.168.56.101:893        192.168.56.102:2049
root@debian10:~# df -h | grep project
192.168.56.102:/srv/project42   15G  2.1G   12G  16% /home/project42
root@debian10:~#
```

70.3.3 Testing the NFS share

We test the NFS share by displaying the **hello.txt** file and by creating a **testfromroot.txtx** file. We notice that the ownership of the file is **nobody:nogroup** instead of **root:root**. That is because **root** is squashed to **nobody** by default.

```
root@debian10:~# cat /home/project42/hello.txt
Welcome to this project
root@debian10:~# echo Test > /home/project42/testfromroot.txt
root@debian10:~# cat /home/project42/testfromroot.txt
Test
root@debian10:~# ll /home/project42/
total 8.0K
-rw-r--r-- 1 root   root    24 Sep 24 14:47 hello.txt
-rw-r--r-- 1 nobody nogroup  5 Sep 24 15:38 testfromroot.txt
root@debian10:~#
```

We also test the connection as a normal user by creating a **testfrompaul.txt** file and here we see **paul:paul** as owners of the file.

```
root@debian10:~# logout
paul@debian10:~$ echo test > /home/project42/testfrompaul.txt
paul@debian10:~$ ls -l /home/project42/
total 12
-rw-r--r-- 1 root   root    24 Sep 24 14:47 hello.txt
-rw-r--r-- 1 paul   paul     5 Sep 24 15:58 testfrompaul.txt
-rw-r--r-- 1 nobody nogroup  5 Sep 24 15:38 testfromroot.txt
paul@debian10:~$
```

Important
If you have multiple client Linux computers connecting to the NFS share, then you have to match UIDs on all clients, the UID is used as file owner, not the name! UIDs can be centralised using NIS, Samba, Kerberos, MS Active Directory or another domain system.

70.4 /etc/exports

We added our first share to the **/etc/exports** file and then restarted the NFS service. Let's take a closer look at this file with some example exports.

```
root@server2:~# tail -12 /etc/exports
## access for all, but read only
/srv/allread    *(ro,no_subtree_check)

## access for 192.168.56.0/24 and all users can write and are nobody
/srv/netwrite    192.168.56.0/24(rw,all_squash,no_subtree_check)

## access for one computer
/srv/one       192.168.56.101(rw,no_subtree_check)

## access for root user as root
/srv/two       *(rw,no_root_squash,no_subtree_check)

root@server2:~#
```

Read **man exports** for a full list of options for this file. Also you can view all current exports on the server (after a **systemctl restart nfs-kernel-server**) with **exportfs -va**.

```
root@server2:~# exportfs -va
exporting 192.168.56.101:/srv/one
exporting 192.168.56.0/24:/srv/netwrite
exporting *:/srv/two
exporting *:/srv/allread
exporting *:/srv/project42
root@server2:~#
```

70.5 /etc/fstab

NFS mounts can be set in the **/etc/fstab** file on the client. Check the **man nfs** page if you need special options for this mount.

```
root@debian10:~# vi /etc/fstab
root@debian10:~# tail -1 /etc/fstab
192.168.56.102:/srv/project42   /home/project42  nfs   rw  0 0
root@debian10:~# umount /home/project42
root@debian10:~# mount /home/project42/
root@debian10:~#
```

70.6 Cheat sheet

Table 70.1: NFS

command	**explanation**
apt-get install nfs-kernel-server	Install the NFS server software.
/etc/exports	The NFS server configuration file.
systemctl restart nfs-kernel-server	(Re)start the NFS server.
exportfs -va	See all shared directories.
apt-get install nfs-common	Install the NFS client software.
mount server:/foo /bar	Mount the NFS share **server:/foo** on **/bar**.
ss -napt | grep 2049	Display the NFS connection.
/etc/fstab	Can contain NFS shares to mount at boot time.

70.7 Practice

1. Choose a computer to be NFS server and install **nfs-kernel-server** on it.
2. Create and export the **/srv/project42** directory, with read-write access for all computers.
3. Choose a client computer and install **nfs-common** on it. Then mount the share from the server on **/home/project42**.
4. Look at the mount with the **df -h** command and with the **mount command**.
5. Display the connection with **ss** or **netstat**.
6. Create a file as root on the network share, then create a file as a normal user. Then list the owners of the files.
7. Verify on the server that the files were really created there.
8. Create an NFS share that is accessible by only one IP-address, namely that of your client. Also create a share for the next valid IP-address. Test that it works.

70.8 Solution

1. Choose a computer to be NFS server and install **nfs-kernel-server** on it.

```
apt-get install nfs-kernel-server
```

2. Create and export the **/srv/project42** directory, with read-write access for all computers.

```
mkdir /srv/project42
echo '/srv/project42   *(rw)' >> /etc/exports
systemctl restart nfs-kernel-server
```

3. Choose a client computer and install **nfs-common** on it. Then mount the share from the server on **/home/project42**.

```
apt-get install nfs-common
mkdir /home/project42
mount 192.168.56.102:/srv/project42 /home/project42
```

4. Look at the mount with the **df -h** command and with the **mount command**.

```
df -h | grep project42
mount | grep project42
```

5. Display the connection with **ss** or **netstat**.

```
ss -napt | grep 2049
```

6. Create a file as root on the network share, then create a file as a normal user. Then list the owners of the files.

```
touch /home/project42/fromroot.txt
su - paul
touch /home/project42/frompaul.txt
ls -l /home/project42
```

7. Verify on the server that the files were really created there.

```
ls -l /srv/project42
```

8. Create an NFS share that is accessible by only one IP-address, namely that of your client. Also create a share for the next valid IP-address. Test that it works.

```
echo /srv/project42   192.168.56.101(rw) >> /etc/exports
echo /srv/project33   192.168.56.102(rw) >> /etc/exports
# Verify on the client that only the first one mounts.
```

Chapter 71

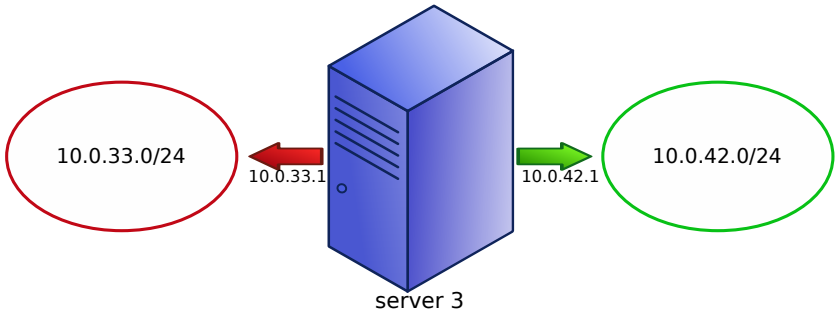
Introduction to routing

71.1 A standard router

In the networking chapter we saw that a **router** is a device that makes decisions based on **IP-addresses**. Obviously it can do a lot more than that.

71.1.1 Debian 10 as a router

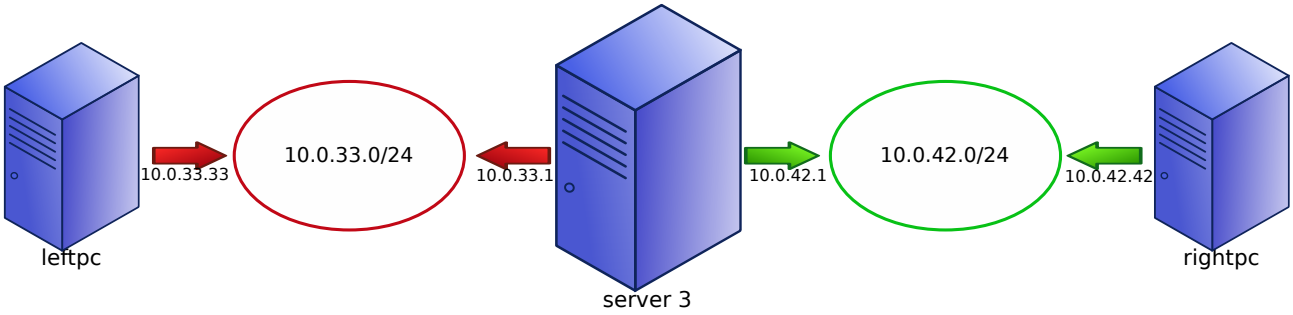
In this chapter we will use our Debian 10 named **server3** as a router. On one side of the router we will construct a network named **leftnet** with IP-range 10.0.33.0/24, on the other side we will construct **rightnet** with IP-range 10.0.42.0/24.



Note
Notice that the router has an **IP-address** ending in **.1** on both networks, which is common practice for routers.

Important
If you are simulating this on VirtualBox then you may need to use an IP-address ending in **.2** instead of **.1** since VirtualBox appears to use **.1** for its internal router.

On each network we will put one client. On **leftnet** we put a Debian 10 server named **leftpc** with IP-address 10.0.33.33 and on **rightnet** we put a Debian 10 server named **rightpc** with IP-address 10.0.42.42 .



71.1.2 a default gateway

If we log on to **leftpc** and issue a **ping** command to the IP-address of **rightpc** then we get a **Network is unreachable** error. The computer knows that 10.0.42.42 is on another network, but it has no gateway to that network.

```
paul@leftpc:~$ ping 10.0.42.42
connect: Network is unreachable
paul@leftpc:~$
```

So step 1 is to enter the IP-address of the router as a default gateway. This happens in **/etc/network/interfaces** by adding a **gateway 10.0.33.1** line.

```
root@leftpc:~# vi /etc/network/interfaces
root@leftpc:~# tail -5 /etc/network/interfaces
allow-hotplug enp0s8
iface enp0s8 inet static
address 10.0.33.33
netmask 255.255.255.0
gateway 10.0.33.1
root@leftpc:~#
```

Tip
The router has two IP-addresses, make sure you enter the IP-address of the router on **leftnet**.

71.1.3 Being a router

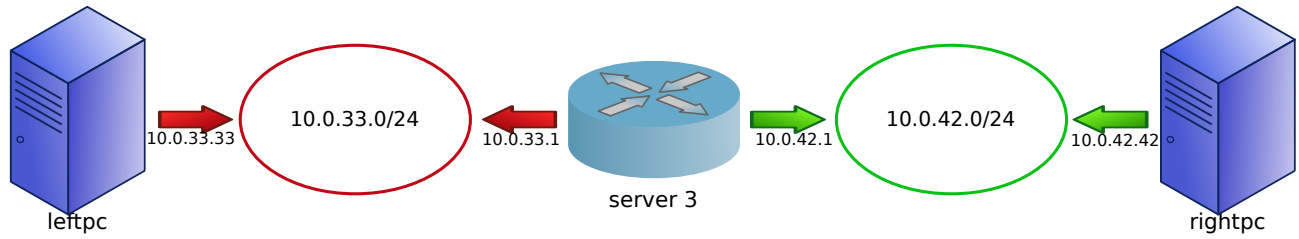
If we now try the **ping** to the IP-address of **rightpc** then we get no output until we press **Ctrl-c**. Then it says all packets were lost.

```
paul@leftpc:~$ ping 10.0.42.42
PING 10.0.42.42 (10.0.42.42) 56(84) bytes of data.
^C
--- 10.0.42.42 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 177ms

paul@leftpc:~$
```

The **leftpc** is now sending the packets to its gateway at 10.0.33.1, but this machine is a default Debian 10 Linux, so it is not a router. We make it a router by setting **/proc/sys/net/ipv4/ip_forward** to 1.

```
root@server3:~# cat /proc/sys/net/ipv4/ip_forward
0
root@server3:~# echo 1 > /proc/sys/net/ipv4/ip_forward
root@server3:~#
```

We can now change the icon of our **server 3** to the router icon.



After entering the **gateway** on **rightpc** there is a response to the ping which passes over our router. We now have a basic router that will forward all traffic between the two networks.

```
paul@leftpc:~$ ping 10.0.42.42
PING 10.0.42.42 (10.0.42.42) 56(84) bytes of data.
64 bytes from 10.0.42.42: icmp_seq=1 ttl=63 time=0.762 ms
64 bytes from 10.0.42.42: icmp_seq=2 ttl=63 time=1.46 ms
^C
--- 10.0.42.42 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 0.762/1.111/1.460/0.349 ms
paul@leftpc:~$
```

71.1.4 Being a permanent router

After a reboot this router function of **server3** is gone. To make it a permanent router you have to uncomment a line in **/etc/sysctl.conf**.

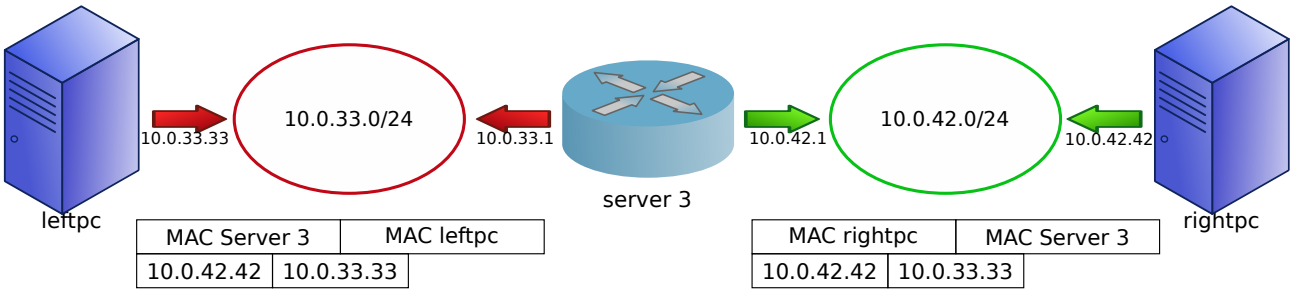
```
root@server3:~# cat /proc/sys/net/ipv4/ip_forward
0
root@server3:~# vi /etc/sysctl.conf
root@server3:~# grep ip_forward /etc/sysctl.conf
net.ipv4.ip_forward=1
root@server3:~#
```

71.1.5 Looking at the Ethernet frames

In this section we take a look at what the router is doing with the **Ethernet frame** that it receives from **leftpc** after a **ping 10.0.42.42** command.

The router is not touching the **IP-addresses** in the frame, which means that during the journey from leftpc to rightpc the **source IP-address** is always 10.0.33.33 and the **destination IP-address** is always 10.0.42.42.

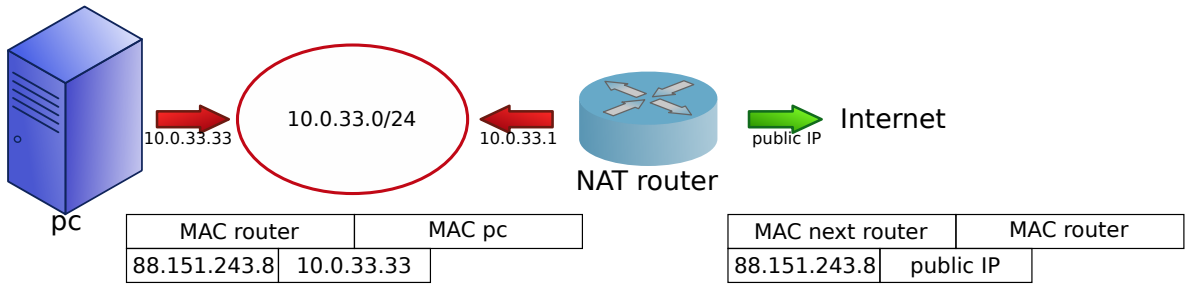
But the router is changing the **MAC-addresses** in the frame, because they are unique to each Ethernet network. In the picture below we added part of the Ethernet and IP header, note the changing MAC addresses between leftnet and rightnet. (Remember we alwyas put the destination before the source in our simplified frame.)



71.2 NAT

NAT is short for **Network Address Translation** and the **network address** in this case is an IP-address. So a NAT router **will** change IP-addresses in a packet.

NAT is often used to translate private IP-addresses from a private network into one public IP-address, as shown in this image. See the change in source IP-address, in addition to the change in MAC-addresses.



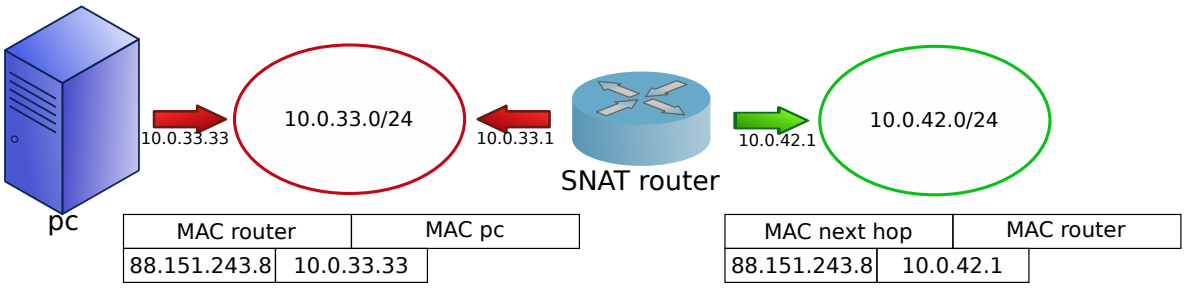
Note that the destination port in this scenario stays the same, while the source port can (probably will) differ. This enables the router to create a NAT table so that a reply from the Internet can be sent to the correct computer.

We will set up a NAT router with Debian 10 and **nftables** in the next chapter.

71.2.1 SNAT

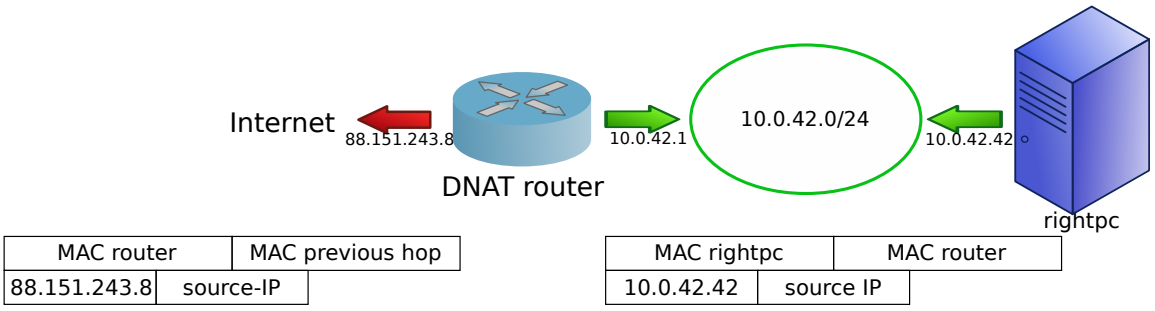
SNAT is short for **Source Network Address Translation**. This means that the source IP-address of a packet should be modified.

In the picture below we show a SNAT router that modifies to source address of the 10.0.33.33 computer to its own 10.42.0.1 address. In this drawing we go from a private address IP-range to another private address IP-range, but this is also common from a private IP-range to a public IP-range. (Very similar to the NAT situation from the previous section.)



71.2.2 DNAT

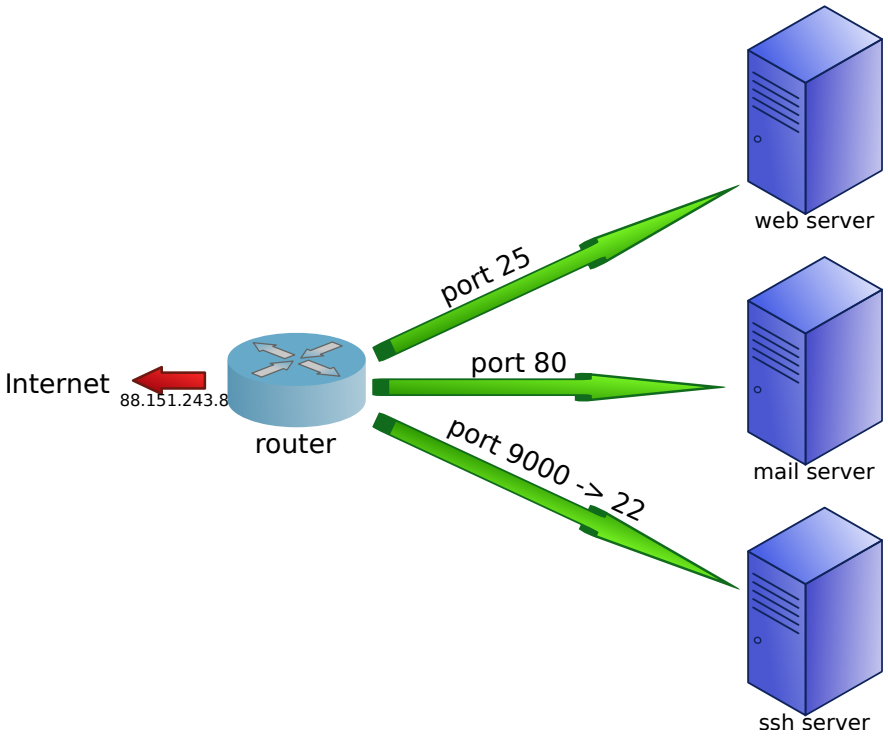
DNAT is short for **Destination Network Address Translation** which means that the destination IP-address of a packet should be modified. This is common for servers with public IP-addresses when not the actual server but rather a router or firewall holds the public IP-address.



In the image above the **rightpc** is the actual web server with a private IP-address, and the router holds the public IP-address by which this web server is known on the Internet.

71.3 port forwarding

Port forwarding can be similar to DNAT, but depends on the destination port. A router can decide to forward all destination port 80 traffic to a local HTTP server and all destination port 25 traffic to another local SMTP server. But it can also decide to forward destination port 9000 to destination port 22 on a local server. All this while also changing the destination IP-address.



71.4 PAT and Static NAT?

Many vendors use another definition of SNAT or will use PAT instead of port forwarding. In the next chapter we will use the Linux kernel firewall with the **nftables** tool and follow the terminology of the Linux kernel and **netfilter** documentation.

So always read and understand the documentation that comes with your device.

71.5 Cheat sheet

Table 71.1: Routing

file or term	**explanation**
/proc/sys/net/ipv4/ip_forward	Contains 1 if computer is a router.
/etc/sysctl.conf	Contains a permanent setting for being a router.
/etc/network/interfaces	Contains **default gateway** (=router) address.
NAT	Network (=IP) Address Translation.
SNAT	Source NAT.
DNAT	Destination NAT.
PAT	Port Address Translation.

71.6 Practice

71.7 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 72

nftables firewall

In Debian 10 buster it is advised to use **nftables** instead of the legacy **iptables**.

72.1 Practice

72.2 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 73

The Linux kernel

73.1 Linus Torvalds

The Linux kernel project was started in 1991 by Linus Torvalds and today in 2020 he is still the leader of the project. There are thousands of developers working on the Linux kernel, often paid by big companies like Intel, AMD, Red Hat, IBM, Huawei, Google just to name a few. The Linux kernel has 28 million lines of code.

Here is a small quote of the message on the **comp.os.minix** newsgroup from 25 August 1991 by Linus Torvalds.

```
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu)  ↩
    for 386(486) AT clones. This has been brewing since april, and is starting to get ready. ↩
     I'd like any feedback on things people like/dislike in minix, as my OS resembles it  ↩
    somewhat (same physical layout of the file-system (due to practical reasons) among other ↩
     things).
```

73.2 kernel.org

The official website of the kernel is **kernel.org**. There you can find all kernel versions and their source code. The Debian project gets its kernel from here and adds it to the distribution.

73.3 kernel versions

The Linux kernel is (as of this writing) at version 5.4. Long ago, with the 2.x series the even version numbers were stable kernels and the odd numbers were *testing*. But since kernel 3.0 this system was abandoned. After 3.0 there was 3.1 and then 3.2 and so on.

When Linus Torvalds thinks the number behind the dot is high enough, then he ups the first version number. There is no big difference between version 4.20 and 5.0, it is only a change in numbering.

73.4 uname -r

You can see the version of the kernel that is running on your system by typing **uname -r**. The Debian **stable** distribution is currently running kernel version 4.19 (on AMD64 architecture).

```
paul@debian10:~$ uname -r
4.19.0-6-amd64
paul@debian10:~$
```

73.5 /boot/vmlinuz

We know from the **booting Linux** chapter that the kernel is located in **/boot/vmlinuz**. There may be previous versions of the kernel in your **/boot** directory.

```
paul@debian10:~$ file /boot/vmlinuz-4.19.0-6-amd64
/boot/vmlinuz-4.19.0-6-amd64: Linux kernel x86 boot executable bzImage, version 4.19.0-6- ↩
    amd64 (debian-kernel@lists.debian.org) #1 SMP Debian 4.19.67-2+deb10u1 (2019-09-20), RO- ↩
    rootFS, swap_dev 0x5, Normal VGA
paul@debian10:~$
```

73.6 linux-image

You can also verify with **dpkg -l** which kernel versions are installed. On this computer there are two kernel versions installed.

```
paul@debian10:~$ dpkg -l | grep linux-image | cut -b-84
ii  linux-image-4.19.0-5-amd64     4.19.37-5+deb10u2                   amd64
ii  linux-image-4.19.0-6-amd64     4.19.67-2+deb10u1                   amd64
ii  linux-image-amd64              4.19+105+deb10u1                    amd64
paul@debian10:~$
```

Tip
You don't need to add the **cut** command to the previous screenshot, it is just there to limit the width of the output.

73.7 /proc/cmdline

The kernel is started by the **boot loader** with some parameters. You can see these parameters that were given to the kernel with **cat /proc/cmdline**.

```
root@debian10:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.19.0-6-amd64 root=UUID=7e43974a-31a1-4533-9ff7-1f73f9f75100 ro  ↩
    quiet
root@debian10:~#
```

The **root** argument is followed by the UUID of the root file system, then there is **ro** (it mounts the root file system as **r**ead **o**nly) and **quiet** as arguments to the kernel.

73.8 single user mode

When booting with **grub2** you can edit the kernel command line (by typing **e** in grub2) and add the **single** keyword. The kernel now boots in **single user mode**, only the root user can log on, and there is no network. This can be useful when troubleshooting a compromised or infected system.

```
root@debian10:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.19.0-6-amd64 root=UUID=7e43974a-31a1-4533-9ff7-1f73f9f75100  ↩
    single ro quiet
root@debian10:~#
```

73.9 init=/bin/bash

You can also give the kernel a new **init** program at boot. For example when you lost the root password, then adding **init=/bin/bash** to the kernel arguments can be a quick way to restore access to the system.

You will need to mount the **/** filesystem first with **rw** access.

```
root@(none):/# mount -o remount,rw /
root@(none):/# passwd
New password:
Retype new password:
passwd: password updated successfully
root@(none):/#
```

73.10 dmesg

The kernel does a lot of reporting (logging) when it is booting on a system. You can see this report with the **dmesg** command. Each line is preceded by the time since the kernel started.

```
root@debian10:~# dmesg | head
[    0.000000] Linux version 4.19.0-6-amd64 (debian-kernel@lists.debian.org) (gcc version  ↩
    8.3.0 (Debian 8.3.0-6)) #1 SMP Debian 4.19.67-2+deb10u1 (2019-09-20)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.19.0-6-amd64 root=UUID=7e43974a-31 ↩
    a1-4533-9ff7-1f73f9f75100 ro quiet
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: x87 floating point registers
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: SSE registers
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: AVX registers
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, using  ↩
    standard format.
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
root@debian10:~#
```

73.11 compiling a kernel

In previous versions of this book there was a section on compiling the kernel yourself. Since this is rare in 2020 these instructions are no longer included.

73.12 Cheat sheet

Table 73.1: Linux kernel

command	explanation
kernel.org	This is the official web page of the Linux kernel.
uname -r	Display the kernel version currently in use.
/boot/vmlinuz-*	This is the compressed kernel, started at boot time.
linux-image-*	The name of the package that contains a Linux kernel.
/proc/cmdline	This file contains the startup arguments given to the kernel.
init=/bin/bash	Argument given to the kernel to gain access as root without password.
dmesg	Display all kernel messages.

73.13 Practice

1. Visit **kernel.org** and note the latest kernel version.
2. Which kernel version is your Debian running now?
3. Which kernel versions are installed on your system?
4. Display the command line that was used to start your kernel.
5. Boot into single user mode.
6. Boot with **bash** as the **init** program. Then change the root password.
7. Boot normally and show the first ten lines of the kernel log.
8. Did the kernel recognise SCSI devices at boot time?

73.14 Solution

1. Visit **kernel.org** and note the latest kernel version.

```
Open Firefox or Chromium and surf to http://kernel.org
```

2. Which kernel version is your Debian running now?

```
uname -r
```

3. Which kernel versions are installed on your system?

```
ls -l /boot/vmlinuz*
```

or

```
dpkg -l | grep linux-image
```

4. Display the command line that was used to start your kernel.

```
cat /proc/cmdline
```

5. Boot into single user mode.

```
reboot
while in grub press e and add single to the kernel line
```

6. Boot with **bash** as the **init** program. Then change the root password.

```
while in grub press e and add init=/bin/bash to the kernel line
mount -o remount,rw /
passwd
```

7. Boot normally and show the first ten lines of the kernel log.

```
dmesg | head
```

8. Did the kernel recognise SCSI devices at boot time?

```
dmesg | grep -i scsi
```

Chapter 74

Kernel modules

74.1 modules

Modules are parts of the kernel that were compiled (usually) at the same time as the kernel. A module is linked to a kernel version and can expand the functionality of a running kernel (without reboot).

Modules are loaded on demand, this makes it possible to use a kernel with a much smaller footprint (as opposed to compiling all functionality in one huge kernel file).

74.2 /lib/modules

The kernel modules are stored in **/lib/modules** (/lib in Debian 10 Buster is a symbolic link to /usr/lib). Their filename end in **.ko** and there are thousands of them.

```
root@debian10:~# find /lib/modules/$(uname -r) -name '*.ko' | wc -l
3559
root@debian10:~#
```

Each kernel has its own directory of modules (since they are compiled for a certain kernel). In the example below two kernels are installed.

```
root@debian10:~# ls -l /lib/modules
total 8
drwxr-xr-x 3 root root 4096 Aug 11 10:56 4.19.0-5-amd64
drwxr-xr-x 3 root root 4096 Oct 14 10:37 4.19.0-6-amd64
root@debian10:~#
```

74.3 lsmod

You can type **lsmod** to see the list of currently loaded modules. The **lsmod** command also displays the size of the module and a list of modules using that module (modules can depend on other modules).

```
root@debian10:~# lsmod | tail
libahci                40960  1 ahci
ohci_hcd               61440  1 ohci_pci
psmouse               172032  0
ehci_hcd               94208  1 ehci_pci
usbcore               294912  4 ohci_hcd,ehci_pci,ehci_hcd,ohci_pci
libata                270336  4 ata_piix,libahci,ahci,ata_generic
usb_common             16384  1 usbcore
scsi_mod              249856  4 sd_mod,libata,sg,sr_mod
e1000                 155648  0
i2c_piix4              24576  0
root@debian10:~#
```

74.4 modinfo

You can use **modinfo** to obtain detailed information about a certain module. In this example we ask for information about the **e1000** module for our Intel network driver.

```
root@debian10:~# modinfo e1000 | head
filename:       /lib/modules/4.19.0-6-amd64/kernel/drivers/net/ethernet/intel/e1000/e1000. ↩
    ko
version:        7.3.21-k8-NAPI
license:        GPL
```

```
description:    Intel(R) PRO/1000 Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     42890A3B22DF8A972B7E072
alias:          pci:v00008086d00002E6Esv*sd*bc*sc*i*
alias:          pci:v00008086d000010B5sv*sd*bc*sc*i*
alias:          pci:v00008086d00001099sv*sd*bc*sc*i*
alias:          pci:v00008086d0000108Asv*sd*bc*sc*i*
root@debian10:~#
```

74.5 modprobe

Modules are normally automatically loaded when needed. If this is not the case then you can use **insmod** or **modprobe** to load a module. The **insmod** command will try to load (insert) a module, but it does not load dependencies automatically.

The **modprobe** tool will (try to) load a module and its dependencies. In this example we load the **vfat** module, which needs the **fat** module. The **fat** module is automatically loaded by **modprobe** as a dependency for **vfat**..

```
root@debian10:~# modprobe vfat
root@debian10:~# lsmod | grep vfat
vfat                   20480  0
fat                    86016  1 vfat
root@debian10:~#
```

74.6 modprobe -r

Modules can be removed with the **modprobe -r** command. This also removes the dependency modules, as shown in this screenshot.

```
root@debian10:~# modprobe -r vfat
root@debian10:~# lsmod | grep vfat
root@debian10:~#
```

74.7 modprobe --show-depends

indexterm:[]

You can list dependencies (and the corresponding **insmod** commands) with the **modprobe --show-depends** command, as shown in this screenshot.

```
root@debian10:~# modprobe --show-depends vfat
insmod /lib/modules/4.19.0-6-amd64/kernel/fs/fat/fat.ko
insmod /lib/modules/4.19.0-6-amd64/kernel/fs/fat/vfat.ko
root@debian10:~#
```

74.8 modules.dep

Module dependencies are stored in the **modules.dep** file. This file can be re-generated with the **depmod** command. This screenshot shows the **vfat** dependency.

```
root@debian10:~# grep vfat /lib/modules/4.19.0-6-amd64/modules.dep
kernel/fs/fat/vfat.ko: kernel/fs/fat/fat.ko
root@debian10:~#
```

74.9 /proc/modules

Technically the **lsmod** command gets its information from **/proc/modules**, which is the kernel interface to all loaded modules. The last column of this file contains the kernel memory location for the module.

```
root@debian10:~# grep vfat /proc/modules
root@debian10:~# modprobe vfat
root@debian10:~# grep vfat /proc/modules
vfat 20480 0 - Live 0xffffffffc04c4000
fat 86016 1 vfat, Live 0xffffffffc05f4000
root@debian10:~#
```

74.10 systemd

You can have modules automatically loaded by **systemd** by adding them in **/etc/modules-load.d/** with their proper configuration file, or added in **modules.conf** in that directory.

```
root@debian10:~# ls -l /etc/modules-load.d/
total 0
lrwxrwxrwx 1 root root 10 Aug 20 13:50 modules.conf -> ../modules
root@debian10:~# cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "" are ignored.

root@debian10:~
```

74.11 compiling a module

74.12 Cheat sheet

Table 74.1: Kernel modules

command	**explanation**
/lib/modules	This is the parent directory for kernel modules.
lsmod	List currently loaded kernel modules.
modinfo foo	Display information about kernel module **foo**.
modprobe foo	Load the kernel module named **foo**.
modprobe -r foo	Unload the kernel module named **foo**.
modprobe --show-depends foo	List dependencies for module **foo**.
/lib/modules/. . . /modules.dep	This file contains module dependencies.
/proc/modules	Contains a list of loaded modules and their location.
/etc/modules	This file can contain modules to load at boot time.
/etc/modules-load.d/	This directory can contain modules to load at boot time.

74.13 Practice

1. Count the number of modules that exist on your system for your running kernel.
2. List the currently loaded modules.
3. List information on the **vfat** module.
4. Load the **vfat** module and all its dependencies.
5. Unload the **vfat** module and all its dependencies.
6. Show the dependencies of the **vfat** module.
7. Does **systemd** load any modules when your system boots?

74.14 Solution

1. Count the number of modules that exist on your system for your running kernel.

```
find /lib/modules/$(uname -r) -name '*.ko' | wc -l
```

2. List the currently loaded modules.

```
lsmod
```

3. List information on the **vfat** module.

```
modinfo vfat
```

4. Load the **vfat** module and all its dependencies.

```
modprobe vfat
```

5. Unload the **vfat** module and all its dependencies.

```
modprobe -r vfat
```

6. Show the dependencies of the **vfat** module.

```
modprobe --show-depends vfat
```

7. Does **systemd** load any modules when your system boots?

```
ls -l /etc/modules-load.d/
cat /etc/modules
```

.

Chapter 75

Introduction to libraries

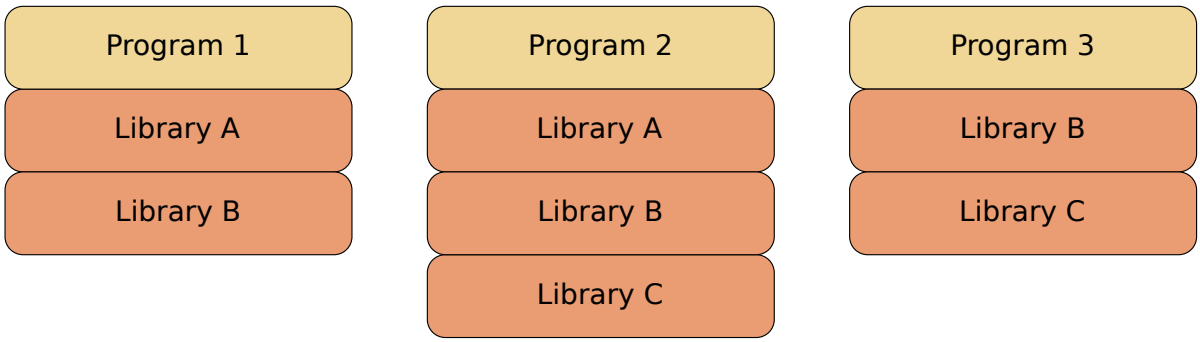
75.1 libraries

This chapter is an introduction to libraries for system administrators, so they can talk to, and understand, developers who have issues with libraries.

In Linux **libraries** are pieces of compiled source code that are called by a program and executed on behalf of the program. This is good for programmers because libraries provide reusable functions and data structures.

75.1.1 static linking of libraries

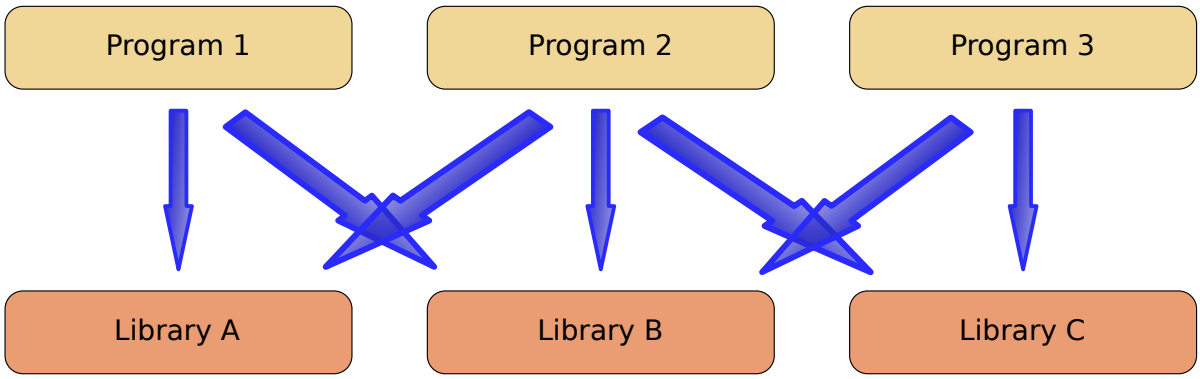
Libraries can be statically linked to programs. This picture shows what happens when libraries are statically linked. Programs contain the library as part of the program itself in this scenario.



As you can see **Program 1** and **Program 2** both have in their compiled program a lot of identical code (namely Library A and Library B). The same is true for **Program 2** with relation to **Program 3**, they both contain Library B and Library C. A lot of programs will be bigger in size with code that is identical for all those programs.

75.1.2 dynamic loading of libraries

Libraries that are dynamically linked and loaded are called by the program when they are needed, but can be shared between all programs. The dynamically linked library needs to exist only once.



75.1.3 the files

The name of dynamically linked and loaded libraries usually contains **.so** which is short for **s**hared **o**bject. The screenshot shows some libraries in the **/lib** directory.

```
root@debian10:~# ls /lib/*.so*
/lib/klibc-ae-2A4n9ZnfImcw9WIt7dF-3OvQ.so  /lib/libgsasl.so.7
/lib/libdiscover.so.2                      /lib/libgsasl.so.7.9.6
/lib/libdiscover.so.2.0.1
root@debian10:~#
```

A quick and dirty count finds over a thousand dynamically linked and loaded libraries on this Debian server. Because many of these are symbolic links to actual libraries the real number of libraries is much lower.

```
root@debian10:~# find /usr/lib -name '*.so*'  | wc -l
1335
root@debian10:~#
```

75.2 ldd

You can see the required libraries for a command using the **ldd** command. In this screenshot we list the libraries needed for the **file** command.

```
root@debian10:~# ldd $(which file)
        linux-vdso.so.1 (0x00007fff52590000)
        libmagic.so.1 => /lib/x86_64-linux-gnu/libmagic.so.1 (0x00007f287b777000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f287b559000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f287b398000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f287b7af000)
root@debian10:~#
```

75.3 ltrace

The **ltrace** command (aptitude install ltrace) is a **library call tracer**. It will run a command until it exits and record the dynamic library calls that were done by that command. The example below traces the calls to the **libmagic.so.1** library made by a simple **file dump.sh** command. The first line of the output is from the **file** command, the rest is **ltrace**.

```
root@debian10:~# ltrace -c -e @libmagic.so.1 file dump.sh
dump.sh: Bourne-Again shell script, ASCII text executable
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 34.15    0.031652          99       319 __ctype_b_loc
 27.29    0.025294         116       217 memcpy
  5.38    0.004984         101        49 free
  5.04    0.004667          97        48 iswprint
  4.71    0.004364          90        48 mbrtowc
  2.37    0.002199          99        22 uselocale
  2.15    0.001995          99        20 malloc
  1.41    0.001307         108        12 regexec
  1.36    0.001261        1261         1 magic_getpath
  1.34    0.001246         103        12 memset
<output truncated>
```

Note
Developers will want to expand their knowledge of this tool.

75.4 dpkg -S

You can verify which package installed a certain library with **dpkg -S** followed by the name of the library. This example shows this information for the **libmagic.so.1** library.

```
root@debian10:~# dpkg -S libmagic.so.1
libmagic1:amd64: /usr/lib/x86_64-linux-gnu/libmagic.so.1
libmagic1:amd64: /usr/lib/x86_64-linux-gnu/libmagic.so.1.0.0
root@debian10:~#
```

Using **debsums** (aptitude install debsums) then, you can verify the integrity of this library.

```
root@debian10:~# debsums libmagic1
/usr/lib/x86_64-linux-gnu/libmagic.so.1.0.0                                 OK
/usr/share/bug/libmagic1/control                                            OK
/usr/share/bug/libmagic1/presubj                                            OK
/usr/share/doc/libmagic1/changelog.Debian.gz                                OK
/usr/share/doc/libmagic1/changelog.gz                                       OK
/usr/share/doc/libmagic1/copyright                                          OK
/usr/share/man/man5/magic.5.gz                                              OK
root@debian10:~#
```

Should this library be corrupt, then it can be reinstalled with **aptitude** as seen in this screenshot.

```
root@debian10:~# aptitude reinstall libmagic1
The following packages will be REINSTALLED:
  libmagic1
0 packages upgraded, 0 newly installed, 1 reinstalled, 0 to remove and 0 not upgraded.
Need to get 117 kB of archives. After unpacking 0 B will be used.
Get: 1 http://deb.debian.org/debian buster/main amd64 libmagic1 amd64 1:5.35-4 [117 kB]
Fetched 117 kB in 0s (427 kB/s)
(Reading database ... 42595 files and directories currently installed.)
Preparing to unpack .../libmagic1_1%3a5.35-4_amd64.deb ...
Unpacking libmagic1:amd64 (1:5.35-4) over (1:5.35-4) ...
Setting up libmagic1:amd64 (1:5.35-4) ...
Processing triggers for libc-bin (2.28-10) ...
Processing triggers for man-db (2.8.5-2) ...

root@debian10:~#
```

75.5 ldconfig

Chances are, that as a system administrator, you never need to use **ldconfig** or its configuration files at **/etc/ld.so.conf** and the directory **/etc/ld.so.conf.d**. You could count the number of libraries on your system with **ldconfig -p**, since this number should go up if an in-house application installs more libraries.

```
root@debian10:~# ldconfig -p | head
288 libs found in cache /etc/ld.so.cache
        libzstd.so.1 (libc6,x86-64) => /lib/x86_64-linux-gnu/libzstd.so.1
        libz.so.1 (libc6,x86-64) => /lib/x86_64-linux-gnu/libz.so.1
        libxtables.so.12 (libc6,x86-64) => /lib/x86_64-linux-gnu/libxtables.so.12
        libxml2.so.2 (libc6,x86-64) => /lib/x86_64-linux-gnu/libxml2.so.2
        libxkbfile.so.1 (libc6,x86-64) => /lib/x86_64-linux-gnu/libxkbfile.so.1
        libxcb.so.1 (libc6,x86-64) => /lib/x86_64-linux-gnu/libxcb.so.1
        libxapian.so.30 (libc6,x86-64) => /lib/x86_64-linux-gnu/libxapian.so.30
        libwrap.so.0 (libc6,x86-64) => /lib/x86_64-linux-gnu/libwrap.so.0
        libuuid.so.1 (libc6,x86-64) => /lib/x86_64-linux-gnu/libuuid.so.1
root@debian10:~#
```

In case a developer gives you a new library path to add to the **ld.so cache** (or to add to the Debian Linux system), then you can add that path to a custom file in **/etc/ld.so.conf.d/** followed by running **ldconfig**.

```
root@debian10:~# vi /etc/ld.so.conf.d/inhouse.conf
root@debian10:~# ldconfig
root@debian10:~#
```

75.6 Cheat sheet

Table 75.1: Libraries

command	explanation
/lib	The parent location of shared libraries in Debian.
ldd /foo	List the libraries needed for the **/foo** command.
apt-get install ltrace	Install a library tracer software.
ltrace -c -e @bar foo	Trace the use of the **bar** library by the **foo** command.
dpkg -S foo	Discover the package that installed **foo**.
debsums foo	Verify the integrity of an installed package named **foo**.
aptitude reinstall foo	Reinstall a package named **foo**.
ldconfig	Tool to configure libraries.

75.7 Practice

1. Which paths on your computer contain libraries?
2. Which libraries are used by the **ls** command?
3. Which tool can trace library calls?
4. Which package provides **libc.so**?
5. Verify the integrity of the **libc.so** file.
6. Re-install the **libc.so** library.

75.8 Solution

1. Which paths on your computer contain libraries?

```
cat /etc/ld.so.conf
cat /etc/ld.so.conf.d/*
```

2. Which libraries are used by the **ls** command?

```
ldd $(which ls)
```

3. Which tool can trace library calls?

```
ltrace
```

4. Which package provides **libc.so**?

```
dpkg -S libc.so
```

5. Verify the integrity of the **libc.so** file.

```
root@debian10:~# debsums libc6-dev | grep libc.so
/usr/lib/x86_64-linux-gnu/libc.so                                       OK
root@debian10:~#
```

6. Re-install the **libc.so** library.

```
aptitude reinstall libc6-dev
```

Chapter 76

Simple backups

76.1 cp

The simplest form of a backup is a **cp** command from one location to another. The **cp** command will copy all the files it receives as arguments to a target directory. By default the **cp** command will not copy directories, as this screenshot shows.

```
paul@debian10:~$ mkdir backup
paul@debian10:~$ cp * backup/
cp: -r not specified; omitting directory 'backup'
cp: -r not specified; omitting directory 'cpp'
cp: -r not specified; omitting directory 'data'
cp: -r not specified; omitting directory 'globbing'
cp: -r not specified; omitting directory 'links'
cp: -r not specified; omitting directory 'music'
cp: -r not specified; omitting directory 'pipes'
paul@debian10:~$
```

To copy all directories and subdirectories with **cp** you can use **cp -r** for a **r**ecursive copy, as shown in this screenshot.

```
paul@debian10:~$ mkdir backup
paul@debian10:~$ cp -r * backup/
cp: cannot copy a directory, backup, into itself, backup/backup
paul@debian10:~$
```

Copying with **cp** to the same hard disk device is, of course, not very fault tolerant. If you have iSCSI or multipath set up, then copying from a local disk to one or more of those devices is advised.

Warning
A backup on the same hard disk as the data is not really a backup!

76.2 tar

The **tar** command, short for **t**ape **ar**chive, is very popular on Linux. It creates an archive of (many) files. This archive is not limited to tapes, it can be put anywhere.

76.2.1 tar cf

The **tar cf filename file1 file2** command runs **tar** with the **c**reate and **f**ile options. It creates a new tar archive named **filename** and puts the files named **file1** and **file2** in the archive. Consider the example in the screenshot.

```
paul@debian10:~$ tar cf backup.tar *
paul@debian10:~$ file backup.tar
backup.tar: POSIX tar archive (GNU)
paul@debian10:~$ ls -lh backup.tar
-rw-r--r-- 1 paul paul 23M Oct 23 11:35 backup.tar
paul@debian10:~$
```

76.2.2 tar czf

Adding the **z** option to the previous command will compress the archive. The rest of the syntax is the same as before.

```
paul@debian10:~$ tar czf backup.tgz *
paul@debian10:~$ file backup.tgz
backup.tgz: gzip compressed data, last modified: Wed Oct 23 09:37:48 2019, from Unix,  ↩
    original size 23787520
paul@debian10:~$ ls -lh backup.tgz
-rw-r--r-- 1 paul paul 20M Oct 23 11:37 backup.tgz
paul@debian10:~$
```

76.2.3 tar tvf

You can see the contents of an existing **tarball** with **tar tvf** (or with **tar tvzf** when compressed) as this example shows.

```
paul@debian10:~$ tar tvzf backup.tgz | head
-rw-r--r-- paul/paul    254130 2019-08-05 05:40 all.txt.bz2
-rw-r--r-- paul/paul        35 2019-08-13 22:19 artist
-rwxr-xr-x paul/paul        89 2019-08-09 13:15 ask.sh
-rwxr-xr-x paul/paul       343 2019-08-22 12:56 cgroupsdemo.sh
-rwxr-xr-x paul/paul       150 2019-08-09 13:00 choice.sh
-rw-r--r-- paul/paul        71 2019-08-04 15:21 cities
-rw-r--r-- paul/paul       198 2019-08-03 21:09 combi.txt
-rw-r--r-- paul/paul      7307 2019-08-05 07:51 copy.gz
-rwxr-xr-x paul/paul       200 2019-08-09 15:05 count.sh
-rw-r--r-- paul/paul        19 2019-07-27 23:42 count.txt
paul@debian10:~$
```

76.2.4 tar xf

Use **tar xf** to e**x**tract files from the tar archive. Add the **z** if it is a compressed tar archive. Adding a filename extracts only that one file.

```
paul@debian10:~$ tar czf backup.tgz *
paul@debian10:~$ rm file42
paul@debian10:~$ tar xzf backup.tgz file42
paul@debian10:~$ file file42
file42: ASCII text
paul@debian10:~$
```

76.3 rsync

The **rsync** tool can be installed with **aptitude install rsync**. It is a fast, versatile, remote and local file-copying tool.

The first example shows a simple copy from one directory (/home/paul/) to another directory on the same computer (/backup). Again, if this is on the same hard disk, then it is not really a backup.

```
root@debian10:~# mkdir /backup
root@debian10:~# rsync -rt /home/paul/ /backup/
skipping non-regular file "links/symlink.txt"
skipping non-regular file "pipes/pipe33a"
skipping non-regular file "pipes/pipe33b"
skipping non-regular file "pipes/pipe42a"
skipping non-regular file "pipes/pipe42b"
skipping non-regular file "pipes/pipe9000a"
skipping non-regular file "pipes/pipe9000b"
root@debian10:~# du -sh /backup/
43M     /backup/
root@debian10:~#
```

One of **rsync**'s strengths is that it can copy files from the local server to a remote server over **ssh**. This requires **rsync** to be installed on both computers.

```
root@debian10:~# rsync -rt /home/paul/ 192.168.56.102:/backup/debian10/
root@192.168.56.102's password:

skipping non-regular file "links/symlink.txt"
skipping non-regular file "pipes/pipe33a"
skipping non-regular file "pipes/pipe33b"
skipping non-regular file "pipes/pipe42a"
skipping non-regular file "pipes/pipe42b"
skipping non-regular file "pipes/pipe9000a"
skipping non-regular file "pipes/pipe9000b"
root@debian10:~#
root@debian10:~#
```

You probably noticed that by default **rsync** does not copy symbolic links and other non-regular files. This can be done with the **--archive** (or -a) option, as this screenshot shows.

```
root@debian10:~# rsync --archive /home/paul/ 192.168.56.102:/backup/debian10/
root@192.168.56.102's password:
root@debian10:~#
```

Both the symbolic link and the pipes are now copied, and nothing else because all the other files are the same as before. The **rsync** tool transfers only the differences between source and destination.

76.4 dump restore

The **dump** and **restore** combination allows for easy backup and restoration of a complete file system. They can be installed with **aptitude install dump**.

```
root@debian10:~# df -h /home/project42
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       2.0G  1.1G  766M  59% /home/project42
root@debian10:~# dump -0 -f project42.dump /home/project42/
  DUMP: Date of this level 0 dump: Wed Oct 23 13:30:31 2019
  DUMP: Dumping /dev/sdb1 (/home/project42) to project42.dump
  DUMP: Label: none
  DUMP: Writing 10 Kilobyte records
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 1107276 blocks.
  DUMP: Volume 1 started with block 1 at: Wed Oct 23 13:30:31 2019
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: Closing project42.dump
  DUMP: Volume 1 completed at: Wed Oct 23 13:30:47 2019
  DUMP: Volume 1 1102880 blocks (1077.03MB)
  DUMP: Volume 1 took 0:00:16
  DUMP: Volume 1 transfer rate: 68930 kB/s
  DUMP: 1102880 blocks (1077.03MB) on 1 volume(s)
  DUMP: finished in 15 seconds, throughput 73525 kBytes/sec
  DUMP: Date of this level 0 dump: Wed Oct 23 13:30:31 2019
  DUMP: Date this dump completed:  Wed Oct 23 13:30:47 2019
  DUMP: Average transfer rate: 68930 kB/s
  DUMP: DUMP IS DONE
root@debian10:~#
```

76.4.1 dump levels

A level 0 dump copies all the files. A level 1 dump copies all the files that were changed since the last level 0 dump. A level 2 dump copies all the files that were changed since the last lower level dump, and so on.

This system allows for example for a full backup (level 0) in the weekend and an incremental backup (level 1 2 3 4 5) on weekdays.

76.4.2 restore

A **restore** from a level 0 **dump** happens on a pristine (mkfs.ext4) file system. This screenshot shows how we enter the empty file system and start the **restore** command.

```
root@debian10:~# cd /home/project42/
root@debian10:/home/project42# ls -l
total 16
drwx------ 2 root root 16384 Oct 23 13:46 lost+found
root@debian10:/home/project42# restore -rf ~/project42.dump
restore: ./lost+found: File exists
root@debian10:/home/project42# du -sh
1.1G    .
root@debian10:/home/project42#
```

76.5 cpio

The **cpio** tool, short for **copy input output**, gets a list of files from **stdin** and copies those files to **stdout**. See this screenshot for a simple backup.

```
root@debian10:~# mkdir /backup
root@debian10:~# ls | cpio -o > /backup/backup.cpio
2207416 blocks
root@debian10:~# file /backup/backup.cpio
/backup/backup.cpio: cpio archive
root@debian10:~# ls -lh /backup/backup.cpio
-rw-r--r-- 1 root root 1.1G Oct 23 15:37 /backup/backup.cpio
root@debian10:~#
```

You can use **cpio -id** to restore the files to the same or in a different location. In this screenshot we restore the files in a **test** directory.

```
root@debian10:~# mkdir test
root@debian10:~# cd test/
root@debian10:~/test# cpio -id < /backup/backup.cpio
2207416 blocks
root@debian10:~/test#
```

You may notice while testing this that **cpio** copies the directories it received from the **ls** command, but not the contents of those directories. The two previous screenshots are pretty useless for full backups (but they explain how **cpio** works).

Most often **cpio** is run behind a **find** command, so it receives all the files that are needed in the backup. See how this example with **find .** copies more files than with **ls** .

```
root@debian10:~# ls | cpio -o > /backup/backup.cpio
2207416 blocks
root@debian10:~# find . | cpio -o > /backup/backup.cpio
2252048 blocks
root@debian10:~#
```

The previous pipe however has a problem when you have files with newlines or tabs in the filename. To make sure those files do not generate an error we can use the **-print0** option of the **find** command, together with the **-0** option of the **cpio** command.

```
root@debian10:~# find . -print0 | cpio -o0 > /backup/backup.cpio
2252048 blocks
root@debian10:~#
```

One last example uses **cpio** to copy a directory from one location to another using the **-p** (for pass-through) option of the command.

```
root@debian10:~# find . -print0 | cpio -p0 /backup/
2251924 blocks
root@debian10:~#
```

76.6 dd

We have already seen **dd** in the utilities chapter, but here are some backup examples. First we create a backup of the Debian netinst CD. You can see that **dd** defaults to mebibytes instead of megabytes.

```
root@debian10:~# mount /dev/cdrom /mnt
mount: /mnt: WARNING: device write-protected, mounted read-only.
root@debian10:~# du -sh /mnt
386M    /mnt
root@debian10:~# umount /mnt
root@debian10:~# dd if=/dev/cdrom of=debian-10.0.0-amd64-netinst.iso
684032+0 records in
684032+0 records out
350224384 bytes (350 MB, 334 MiB) copied, 2.12869 s, 165 MB/s
root@debian10:~#
```

A little know feature of **dd** is that you can get a progress report by adding **status=progress** to the **dd** command, this can be useful for larger volumes.

```
root@debian10:~# dd if=/dev/cdrom of=debian-10.0.0-amd64-dvd.iso status=progress
3692559360 bytes (3.7 GB, 3.4 GiB) copied, 22 s, 168 MB/s
7503872+0 records in
7503872+0 records out
3841982464 bytes (3.8 GB, 3.6 GiB) copied, 23 s, 167 MB/s
root@debian10:~#
```

But **dd** can also be used to make a complete backup of a file system. In this screenshot we also add gzip to compress the file system image. Note that **dd** copies the whole file system, including the empty space.

```
root@debian10:~# du -sh /home/project42/
1.1G    /home/project42/
root@debian10:~# mount | grep project42
/dev/sdb1 on /home/project42 type ext4 (rw,relatime)
root@debian10:~# dd if=/dev/sdb1 | gzip > project42drive.dd.gz
4194304+0 records in
4194304+0 records out
2147483648 bytes (2.1 GB, 2.0 GiB) copied, 55.3767 s, 38.8 MB/s
root@debian10:~#
```

Note
Use **dd** only on an off-line file system, otherwise it can result in an inconsistent backup.

76.7 network backups

Most organisations prefer to take backups to another site (as in another physical location on this planet). Luckily there is **ssh**, which can be inserted in most of the command lines of this chapter, to securely transfer data from one site to another.

We start of with an example of the **tar** command in combination with **ssh**. Note that we use **-** to denote **stdout** after the **f** option of the **tar** command. Likewise we use **-** to denote **stdin** for the **cat** command.

```
paul@debian10:~$ tar czf - * | ssh root@192.168.56.102 "cat - > /backup.tgz"
root@192.168.56.102's password:
paul@debian10:~$
```

Similarly we can tell **dump** to use **stdout** and pipe the dump through **ssh** to be received on the other end by a **cat** command that reads from **stdin**.

```
root@debian10:~# dump -0 -f - /home/project42/ | ssh root@192.168.56.102 "cat - > /pro42.d
ump"
  DUMP: Date of this level 0 dump: Wed Oct 23 18:36:18 2019
  DUMP: Dumping /dev/sdb1 (/home/project42) to standard output
  DUMP: Label: none
  DUMP: Writing 10 Kilobyte records
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 1109157 blocks.
  DUMP: Volume 1 started with block 1 at: Wed Oct 23 18:36:18 2019
  DUMP: dumping (Pass III) [directories]
root@192.168.56.102's password:
  DUMP: dumping (Pass IV) [regular files]
  DUMP: Volume 1 completed at: Wed Oct 23 18:36:38 2019
  DUMP: Volume 1 1104760 blocks (1078.87MB)
  DUMP: Volume 1 took 0:00:20
  DUMP: Volume 1 transfer rate: 55238 kB/s
  DUMP: 1104760 blocks (1078.87MB)
  DUMP: finished in 20 seconds, throughput 55238 kBytes/sec
  DUMP: Date of this level 0 dump: Wed Oct 23 18:36:18 2019
  DUMP: Date this dump completed:  Wed Oct 23 18:36:38 2019
  DUMP: Average transfer rate: 55238 kB/s
  DUMP: DUMP IS DONE
root@debian10:~#
```

Tip
You will need passwordless **ssh** if you want to schedule these jobs with **cron** .

76.8 Cheat sheet

Table 76.1: Simple backups

command	explanation
cp bar foo/	Copy the **bar** file to a directory named **foo**.
tar cf foo bar	Create a **tar** file named **foo** that contains the **bar** file.
tar tvf foo	List the contents of the **tar** file named **foo**.
tar xf foo	Extract all files from a **tar** file.
tar czf foo bar	Create a compressed **tar** file named **foo** that contains the **bar** file.
tar tzvf foo	List the contents of the compressed **tar** file named **foo**.
tar xzf foo	Extract all files from a compressed **tar** file.
rsync -rt foo bar	Copy all regular files and directories from **foo** to **bar**.
rsync -rt foo server2:bar	Copy all regular files and directories from **foo** to **bar** on another computer.
rsync -a	Also copies symbolic links and other special files.
dump -0 -f foo.dump /bar	A level 0 backup of the contents of **/bar** to **foo.dump**.
restore -rf foo.dump	Restore a dump.
find . . . |cpio -o > foo.cpio	Copy a list of files to a cpio archive named **foo.cpio**.
cpio -id < foo.cpio	Restore the **foo.cpio** archive in the current location.
dd if=foo of=bar	Copy input file **foo** to output file **bar**.
dd if=/dev/cdrom of=cd.iso	Create an .ISO file of a CD-ROM.
dd if=/dev/sda of=disk.dd	Copy a complete block device to a file named **disk.dd**.
foo | ssh user@server 'bar'	Execute the **foo** command on this computer and pipe the output to the **bar** command on another computer.

76.9 Practice

1. Use **tar** to create an archive of the **/etc** directory.
2. Use **tar** to create a compressed archive of the **/etc** directory.
3. Verify (display) the list of files in this last archive.
4. Use **rsync** to copy **/etc** to another server.
5. Dump the offline **/dev/sdb1** file system to a file name **/pro42.dump** .
6. Restore this dump on **/dev/sdb1** .
7. Use **cpio** to create an archive of **/etc** .
8. Restore this cpio archive in a **test** directory.
9. Insert a CD/DVD in your server (if you can) and create an **.ISO** file of it.
10. Use **tar** with compression to copy the **/etc** directory from one server to a backup directory on another server.
11. The same as the previous question, but with **ssh** compression instead of **tar** compression.

76.10 Solution

1. Use **tar** to create an archive of the **/etc** directory.

```
tar cf etc.tar /etc
```

2. Use **tar** to create a compressed archive of the **/etc** directory.

```
tar czf etc.tgz /etc
```

3. Verify (display) the list of files in this last archive.

```
tar tzvf etc.tgz | more
```

4. Use **rsync** to copy **/etc** to another server.

```
ssh root@192.168.56.102 "mkdir -p /backup/server1"
rsync -rt /etc 192.168.56.102:/backup/server1/
```

5. Dump the offline **/dev/sdb1** file system to a file name **/pro42.dump** .

```
dump -0 -f /pro42.dump /dev/sdb1
```

6. Restore this dump on **/dev/sdb1** .

```
mkfs.ext4 /dev/sdb1
mount /dev/sdb1 /mnt
cd /mnt
restore -rf /pro42.dump
ls
```

7. Use **cpio** to create an archive of **/etc** .

```
find /etc | cpio -o > /root/etc.cpio
```

8. Restore this cpio archive in a **test** directory.

```
mkdir test
cd test
cpio -id < /root/etc.cpio
```

9. Insert a CD/DVD in your server (if you can) and create an **.ISO** file of it.

```
dd if=/dev/cdrom of=file.iso
```

10. Use **tar** with compression to copy the **/etc** directory from one server to a backup directory on another server.

```
ssh root@192.168.56.102 "mkdir /backup"
tar czf - /etc | ssh root@192.168.56.102 "cd /backup ; tar xzf -"
```

11. The same as the previous question, but with **ssh** compression instead of **tar** compression.

```
tar cf - /etc | ssh -C root@192.168.56.102 "cd /backup ; tar xf -"
```

Chapter 77

Backup Tools

There are several tools that automate the backup process, often with graphical interfaces. We look at two of them in this chapter, in alphabetical order.

77.1 Amanda

The **Amanda Network Backup** solution enables setting up a backup server that can do network backup of many Linux, Unix, Mac OS X or Microsoft backup clients. Its official website is https://amanda.org . **Amanda** is by default available in the Debian stable repository.

```
root@server6:~# aptitude search amanda
p   amanda-client                       - Advanced Maryland Automatic Network Disk Archi
p   amanda-common                       - Advanced Maryland Automatic Network Disk Archi
p   amanda-server                       - Advanced Maryland Automatic Network Disk Archi
root@server6:~#
```

77.1.1 Amanda installation

In this chapter we will use the Debian 10 server named **server6** as an Amanda server, and we will use the **backup** user to manage backups with **Amanda**. The installation of Amanda is done with the following commands.

```
root@server6:~# aptitude install amanda-server amanda-client
<output truncated>
root@server6:~#
```

Typing **apropos amanda** now gives you a small heart attack; there is a lot of documentation on the **Amanda** tools.

77.1.2 Getting started

The https://amanda.org website has a wiki with a **Getting Started** section. We follow this guide, and add the necessary commands for it to work on Debian 10. The first task is to create a bunch of directories.

```
root@server6:~# mkdir -p /amanda/vtapes/slot{1,2,3,4}
root@server6:~# mkdir -p /amanda/holding
root@server6:~# mkdir -p /amanda/state/{curinfo,log,index}
root@server6:~# mkdir -p /etc/amanda/MyConfig
root@server6:~#
```

The next step is to populate the **amanda.conf** file in our custom **MyConfig** directory. The only change in comparison with the **amanda.org** wiki is the name of the **dumpuser**.

```
root@server6:~# vi /etc/amanda/MyConfig/amanda.conf
root@server6:~# cat /etc/amanda/MyConfig/amanda.conf
org "MyConfig"
infofile "/amanda/state/curinfo"
logdir "/amanda/state/log"
indexdir "/amanda/state/index"
dumpuser "backup"

tpchanger "chg-disk:/amanda/vtapes"
labelstr "MyData[0-9][0-9]"
autolabel "MyData%%" EMPTY VOLUME_ERROR
tapecycle 4
dumpcycle 3 days
amrecover_changer "changer"

tapetype "TEST-TAPE"
define tapetype TEST-TAPE {
  length 100 mbytes
  filemark 4 kbytes
}
```

```
define dumptype simple-gnutar-local {
    auth "local"
    compress none
    program "GNUTAR"
}

holdingdisk hd1 {
    directory "/amanda/holding"
    use 50 mbytes
    chunksize 1 mbyte
}
root@server6:~#
```

The **disklist** file is also mandatory for **Amanda** to work. Again we take the version from the wiki, which configures the **server6** to take a **tar** archive of the **/etc** directory.

```
root@server6:~# vi /etc/amanda/MyConfig/disklist
root@server6:~# cat /etc/amanda/MyConfig/disklist
localhost /etc simple-gnutar-local
root@server6:~#
```

77.1.3 backup user account

We need to set permissions for the **backup** user and give that user a shell so we can log on with it. Doing this modifies default **Debian 10** behaviour, another option would be to create a dedicated **amandabackup** user account.

```
root@server6:~# chown -R backup /amanda/
root@server6:~# chown -R backup /etc/amanda
root@server6:~# usermod -s /bin/bash backup
root@server6:~#
```

Tip

Both **backup** and **amandabackup** are easy to guess user accounts and are prone to remote login attempts. Improve security by using another name for the backup account.

77.1.4 Amanda security

Debian 10 puts the **/usr/bin** directory before the **/bin** directory in the $PATH variable, so we need to update the **/etc/amanda-security.conf** file to have the correct path for **tar**.

```
root@server6:~# vi /etc/amanda-security.conf
root@server6:~# grep ^[^#] /etc/amanda-security.conf
runtar:gnutar_path=/usr/bin/tar
amgtar:gnutar_path=/usr/bin/tar
tcp_port_range=512,1024
udp_port_range=840,860
root@server6:~#
```

77.1.5 amcheck

The configuration can be verified (by the **backup** user!) with the **amcheck** command.

```
root@server6:~# su - backup
backup@server6:~$ amcheck MyConfig
Amanda Tape Server Host Check
-----------------------------
NOTE: tapelist file does not exists
      it will be created on the next run
NOTE: Holding disk '/amanda/holding': 12808192 KB disk space available, using 51200 KB as  ↩
    requested
slot 1: contains an empty volume
Will write label 'MyData01' to new volume in slot 1.
NOTE: skipping tape-writable test
NOTE: host info dir '/amanda/state/curinfo/localhost' does not exist
      It will be created on the next run
NOTE: index dir '/amanda/state/index/localhost' does not exist
      it will be created on the next run
Server check took 0.216 seconds
Amanda Backup Client Hosts Check
--------------------------------
Client check: 1 host checked in 2.269 seconds.  0 problems found.
(brought to you by Amanda 3.5.1)
backup@server6:~$
```

77.1.6 amdump

It is time now for a test run of the backup of **/etc** with the **amdump MyConfig** command. This has to be run as the **backup** user.

```
backup@server6:~$ amdump MyConfig && echo $?
0
backup@server6:~$
```

77.1.7 amreport

A report on this dump can be displayed with **amreport MyConfig**.

```
backup@server6:~$ amreport MyConfig
Hostname: server6
Org     : MyConfig
Config  : MyConfig
Date    : October 28, 2019

These dumps were to tape MyData01.
The next tape Amanda expects to use is: 1 new tape.


STATISTICS:
                          Total       Full      Incr.   Level:#
                        --------   --------   --------  --------
Estimate Time (hrs:min)     0:00
Run Time (hrs:min)          0:00
Dump Time (hrs:min)         0:00       0:00       0:00
Output Size (meg)            2.3        2.3        0.0
Original Size (meg)          2.3        2.3        0.0
Avg Compressed Size (%)    100.0      100.0        --
DLEs Dumped                    1          1          0
Avg Dump Rate (k/s)       2024.0     2024.0        --

Tape Time (hrs:min)         0:00       0:00       0:00
Tape Size (meg)              2.3        2.3        0.0
```

```
Tape Used (%)                 2.3        2.3        0.0
DLEs Taped                      1          1          0
Parts Taped                     1          1          0
Avg Tp Write Rate (k/s)  23600.0    23600.0        --


USAGE BY TAPE:
  Label                 Time         Size      %  DLEs Parts
  MyData01              0:00        2360K    2.3     1     1


NOTES:
  planner: tapecycle (3) <= runspercycle (3)
  planner: Adding new disk localhost:/etc.
  taper: Slot 1 without label can be labeled
  taper: tape MyData01 kb 2360 fm 1 [OK]


DUMP SUMMARY:
                                                    DUMPER STATS   TAPER STATS
HOSTNAME     DISK        L ORIG-KB  OUT-KB  COMP%  MMM:SS   KB/s MMM:SS    KB/s
-------------------------- ---------------------- -------------- --------------
localhost    /etc        0    2360    2360    --     0:01 2023.3   0:00 23600.0

(brought to you by Amanda version 3.5.1)
backup@server6:~$
```

77.1.8 backup location

The backup is located in the **/amanda/vtapes/slot1/** directory. We use **/amanda** as an example directory, preferably this is a mount point (to a multipath device for example).

```
backup@server6:~$ ls -lh /amanda/vtapes/slot1/
total 2.4M
-rw------- 1 backup backup  32K Oct 28 19:54 00000.MyData01
-rw------- 1 backup backup 2.4M Oct 28 19:54 00001.localhost._etc.0
backup@server6:~$ file /amanda/vtapes/slot1/00001.localhost._etc.0
/amanda/vtapes/slot1/00001.localhost._etc.0: AMANDA
backup@server6:~$
```

Using **cron** the backup can be scheduled to run. Use **crontab -e** as the backup user to schedule backups with the **amdump** command.

77.2 Recovery with Amanda

Recovery is done with the **root** user. In this example we recover files on the backup server (you can use **scp** or something to get the files to the proper host). Recovery is done with the **amrecover** tool inside an empty temporary directory.

```
root@server6:~# mkdir /tmp/recovery
root@server6:~# cd /tmp/recovery/
root@server6:/tmp/recovery#
```

Provide access for the root user to the backups on this **server6**, from this **server6** known as **localhost**.

```
root@server6:/tmp/recovery# vi /var/backups/.amandahosts
root@server6:/tmp/recovery# tail -1 /var/backups/.amandahosts
localhost root amindexd amidxtaped
root@server6:/tmp/recovery#
```

Start **amrecover MyConfig** as the root user, make sure you are in the **/tmp/recovery** directory. You enter an interactive shell.

```
root@server6:/tmp/recovery# amrecover MyConfig
AMRECOVER Version 3.5.1. Contacting server on localhost ...
220 server6 AMANDA index server (3.5.1) ready.
Setting restore date to today (2019-10-28)
200 Working date set to 2019-10-28.
200 Config set to MyConfig.
501 Host server6 is not in your disklist.
Trying host server6 ...
501 Host server6 is not in your disklist.
Trying host server6 ...
501 Host server6 is not in your disklist.
Use the sethost command to choose a host to recover
amrecover>
```

At the **amrecover** prompt, set the hostname to localhost and the disk to recover to **/etc**. Then add the list of files you wish to recover (we only ask for the **passwd** file).

```
amrecover> sethost localhost
200 Dump host set to localhost.
amrecover> setdisk /etc
200 Disk set to /etc.
amrecover> add passwd
Added file /passwd
amrecover>
```

Then type **extract** to extract the files in your list to the temporary location on the server.

```
amrecover> extract

Extracting files using tape drive changer on host localhost.
The following tapes are needed: MyData01

Extracting files using tape drive changer on host localhost.
Load tape MyData01 now
Continue [?/Y/n/s/d]?
Restoring files into directory /tmp/recovery
All existing files in /tmp/recovery can be deleted
Continue [?/Y/n]?

./passwd
2360 kb
amrecover>
```

It is now time to exit the shell and to maybe copy the extracted file to its destination.

```
amrecover> exit
200 Good bye.
root@server6:/tmp/recovery# ls -l
total 4
-rw-r--r-- 1 root root 1530 Oct 28 19:51 passwd
root@server6:/tmp/recovery#
```

77.3 Bacula

Bacula is a complete backup solution that is available in the standard stable Debian 10 repository. We install it here with the **apt** tool.

```
root@server6:~# apt install bacula
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bacula-bscan bacula-client bacula-common bacula-common-pgsql bacula-console
  bacula-director bacula-director-pgsql bacula-fd bacula-sd bacula-server bsd-mailx
  dbconfig-common dbconfig-pgsql exim4-base exim4-config exim4-daemon-light
  libevent-2.1-6 libgnutls-dane0 libllvm7 liblockfile1 liblzo2-2 libpq5 libsensors-config
  libsensors5 libunbound8 libxslt1.1 mt-st mtx postgresql postgresql-11 postgresql-client
  postgresql-client-11 postgresql-client-common postgresql-common psmisc ssl-cert sysstat
Suggested packages:
  gdb bacula-doc gawk dds2tar scsitools sg3-utils exim4-doc-html | exim4-doc-info eximon4
  spf-tools-perl swaks dns-root-data lm-sensors postgresql-doc postgresql-doc-11
  libjson-perl openssl-blacklist isag
The following NEW packages will be installed:
  bacula bacula-bscan bacula-client bacula-common bacula-common-pgsql bacula-console
  bacula-director bacula-director-pgsql bacula-fd bacula-sd bacula-server bsd-mailx
  dbconfig-common dbconfig-pgsql exim4-base exim4-config exim4-daemon-light
  libevent-2.1-6 libgnutls-dane0 libllvm7 liblockfile1 liblzo2-2 libpq5 libsensors-config
  libsensors5 libunbound8 libxslt1.1 mt-st mtx postgresql postgresql-11 postgresql-client
  postgresql-client-11 postgresql-client-common postgresql-common psmisc ssl-cert sysstat
0 upgraded, 38 newly installed, 0 to remove and 0 not upgraded.
Need to get 35.9 MB of archives.
After this operation, 131 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

77.3.1 configuration

Change the location of backups to **/backup**, in real live this should be a remote mount point. The files to change are **/etc/bacula/bacula-dir.conf**, and **/etc/bacula/bacula-sd.conf**.

```
root@server6:/etc/bacula# mkdir /backup
root@server6:/etc/bacula# vi bacula-dir.conf
root@server6:/etc/bacula# grep '/backup' bacula-dir.conf
  Where = /backup/bacula-restores
    File = /backup        ## in the Exclude section
root@server6:/etc/bacula# vi bacula-sd.conf
root@server6:/etc/bacula# grep '/backup' bacula-sd.conf
  Archive Device = /backup
  Archive Device = /backup
  Archive Device = /backup
  Archive Device = /backup
root@server6:/etc/bacula#
```

Note
Remove all the /**nonexistant** paths from the configuration.

Also note that **Bacula** has four services in **systemd**. Restart the **bacula-sd.service** and the **bacula-dir.service** after changing the configuration. And make sure the other two services are started.

```
root@debian10:~# systemctl restart bacula-sd
root@debian10:~# systemctl restart bacula-dir
root@debian10:~# systemctl restart bacula-fd
root@debian10:~# systemctl restart bacula-director
root@debian10:~#
```

77.4 bconsole

Backups are managed via the **bconsole** application. Type **help** here to get a list of commands. The **show filesets** command in **bconsole** indicates that **/usr/sbin** is included and seven directories are excluded in our **Full Set**.

```
root@server6:/etc/bacula# bconsole
Connecting to Director localhost:9101
1000 OK: 103 server6-dir Version: 9.4.2 (04 February 2019)
Enter a period to cancel a command.
*show filesets
FileSet: name=Full Set IgnoreFileSetChanges=0
      O M
      N
      I /usr/sbin
      N
      E /var/lib/bacula
      E /proc
      E /tmp
      E /sys
      E /.journal
      E /.fsck
      E /backup
      N
FileSet: name=Catalog IgnoreFileSetChanges=0
      O M
      N
      I /var/lib/bacula/bacula.sql
      N
*
```

77.4.1 Bacula status

The **status dir** command in the **bconsole** will display scheduled, running and completed jobs. Also try to run a **status storage** in this console.

```
*status dir
server6-dir Version: 9.4.2 (04 February 2019) x86_64-pc-linux-gnu debian buster/sid
Daemon started 28-Oct-19 11:20, conf reloaded 28-Oct-2019 11:20:30
 Jobs: run=0, running=0 mode=0,0
 Heap: heap=282,624 smbytes=56,078 max_bytes=56,078 bufs=238 max_bufs=238
 Res: njobs=3 nclients=1 nstores=2 npools=3 ncats=1 nfsets=2 nscheds=2

Scheduled Jobs:
Level          Type     Pri  Scheduled          Job Name           Volume
===================================================================================
Incremental    Backup    10  28-Oct-19 23:05    BackupClient1      unknown
Full           Backup    11  28-Oct-19 23:10    BackupCatalog      unknown
====

Running Jobs:
Console connected at 28-Oct-19 11:20
No Jobs running.
====
No Terminated Jobs.
====
*
```

It is time to run our first job in **Bacula**, with the **run** command. An **incremental** job that is first run is automatically converted to a Full backup.

```
*run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
The defined Job resources are:
     1: BackupClient1
     2: BackupCatalog
     3: RestoreFiles
Select Job resource (1-3): 1
Run Backup job
JobName:  BackupClient1
Level:    Incremental
Client:   server6-fd
FileSet:  Full Set
Pool:     File (From Job resource)
Storage:  File1 (From Job resource)
When:     2019-10-28 11:25:20
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=1
You have messages.
*
```

It does not take long to take a backup of **/usr/sbin**, so you can run the **messages** command in the **bconsole** to display a report about the backup job.

```
*messages
28-Oct 11:25 server6-dir JobId 1: No prior Full backup Job record found.
28-Oct 11:25 server6-dir JobId 1: No prior or suitable Full backup found in catalog. Doing ↩
    FULL backup.
28-Oct 11:25 server6-dir JobId 1: Start Backup JobId 1, Job=BackupClient1.2019-10-28_11 ↩
    .25.55_03
28-Oct 11:25 server6-dir JobId 1: Created new Volume="Vol-0001", Pool="File", MediaType=" ↩
    File1" in catalog.
28-Oct 11:25 server6-dir JobId 1: Using Device "FileChgr1-Dev1" to write.
28-Oct 11:25 server6-sd JobId 1: Labeled new Volume "Vol-0001" on File device "FileChgr1- ↩
    Dev1" (/backup).
28-Oct 11:25 server6-sd JobId 1: Wrote label to prelabeled Volume "Vol-0001" on File device ↩
     "FileChgr1-Dev1" (/backup)
28-Oct 11:26 server6-sd JobId 1: Elapsed time=00:00:04, Transfer rate=5.701 M Bytes/second
28-Oct 11:26 server6-sd JobId 1: Sending spooled attrs to the Director. Despooling 62,870 ↩
    bytes ...
28-Oct 11:26 server6-dir JobId 1: Bacula server6-dir 9.4.2 (04Feb19):
  Build OS:               x86_64-pc-linux-gnu debian buster/sid
  JobId:                  1
  Job:                    BackupClient1.2019-10-28_11.25.55_03
  Backup Level:           Full (upgraded from Incremental)
  Client:                 "server6-fd" 9.4.2 (04Feb19) x86_64-pc-linux-gnu,debian,buster/ ↩
      sid
  FileSet:                "Full Set" 2019-10-28 11:25:55
  Pool:                   "File" (From Job resource)
  Catalog:                "MyCatalog" (From Client resource)
  Storage:                "File1" (From Job resource)
  Scheduled time:         28-Oct-2019 11:25:20
  Start time:             28-Oct-2019 11:25:57
  End time:               28-Oct-2019 11:26:01
  Elapsed time:           4 secs
  Priority:               10
  FD Files Written:       318
  SD Files Written:       318
  FD Bytes Written:       22,771,230 (22.77 MB)
```

```
  SD Bytes Written:       22,804,184 (22.80 MB)
  Rate:                   5692.8 KB/s
  Software Compression:   None
  Comm Line Compression:  42.7% 1.7:1
  Snapshot/VSS:           no
  Encryption:             no
  Accurate:               no
  Volume name(s):         Vol-0001
  Volume Session Id:      1
  Volume Session Time:    1572257421
  Last Volume Bytes:      22,830,076 (22.83 MB)
  Non-fatal FD errors:    0
  SD Errors:              0
  FD termination status:  OK
  SD termination status:  OK
  Termination:            Backup OK

28-Oct 11:26 server6-dir JobId 1: Begin pruning Jobs older than 6 months .
28-Oct 11:26 server6-dir JobId 1: No Jobs found to prune.
28-Oct 11:26 server6-dir JobId 1: Begin pruning Files.
28-Oct 11:26 server6-dir JobId 1: No Files found to prune.
28-Oct 11:26 server6-dir JobId 1: End auto prune.

*quit
root@server6:~#
```

77.5 Recovery with Bacula

In this example we recover all files from the backup, using the **restore all** command. In the practice of this chapter we want you to execute **restore** instead to restore a single file from the backup.

```
root@server6:~# bconsole
Connecting to Director localhost:9101
1000 OK: 103 server6-dir Version: 9.4.2 (04 February 2019)
Enter a period to cancel a command.
*restore all
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified Job date
    13: Cancel
Select item:  (1-13): 5
```

```
Automatically selected Client: server6-fd
Automatically selected FileSet: Full Set
+-------+-------+----------+-------------+---------------------+------------+
| jobid | level | jobfiles | jobbytes    | starttime           | volumename |
+-------+-------+----------+-------------+---------------------+------------+
|     1 | F     |      318 | 22,771,230 | 2019-10-28 11:25:57 | Vol-0001   |
+-------+-------+----------+-------------+---------------------+------------+
You have selected the following JobId: 1

Building directory tree for JobId(s) 1 ...  +
317 files inserted into the tree and marked for extraction.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$
```

Typing **done** will start the actual restore of all files in the **/backup/bacula-restores/** directory. You can then copy them manually back to their original location.

```
$ done
Bootstrap records written to /var/lib/bacula/server6-dir.restore.1.bsr

The Job will require the following (*=>InChanger):
   Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

    Vol-0001                  File1                     FileChgr1

Volumes marked with "*" are in the Autochanger.


318 files selected to be restored.

Using Catalog "MyCatalog"
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /var/lib/bacula/server6-dir.restore.1.bsr
Where:           /backup/bacula-restores
Replace:         Always
FileSet:         Full Set
Backup Client:   server6-fd
Restore Client:  server6-fd
Storage:         File1
When:            2019-10-28 20:42:55
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): yes
Job queued. JobId=2
*
*quit
root@server6:~# ls -l /backup/bacula-restores/
total 4
drwxr-xr-x 3 root root 4096 Oct 28 20:43 usr
root@server6:~# ls -l /backup/bacula-restores/usr/
total 16
drwxr-xr-x 2 root root 16384 Oct 28 10:56 sbin
root@server6:~#
```

77.6 Cheat sheet

Table 77.1: Complete backups

command	explanation
apt-get install amanda-server	Install the Amanda server software.
apt-get install amanda-client	Install the Amanda client software.
/etc/amanda	Location of Amanda configuration.
amcheck	Tool to verify Amanda configuration.
amdump	Tool to take a backup with Amanda.
amreport	Displays a report on the backups.
apt install bacula	Install the Bacula backup solution.
/etc/bacula	Location of Bacula configuration.
bconsole	Interactive tool to manage backups and restores.

77.7 Practice

1. Install and configure Amanda, and take a backup of **/etc**, as shown in the theory.
2. Populate **/home/project42** with some files (a copy of /etc or /sbin will do). Use **Amanda** to take a **full backup** of this directory.
3. Delete one of the files in **/home/project42**, then restore it from this backup.
4. Install and configure Bacula, as shown in the theory.
5. Populate **/home/project33** with some files (a copy of /etc or /sbin will do). Use **Bacula** to take a **full backup** of this directory.
6. Delete one file in **/home/project33/** and then restore it from this last backup.

77.8 Solution

1. Install and configure Amanda, and take a backup of **/etc**, as shown in the theory.

```
root@debian10:~# aptitude install amanda-server amanda-client
<output truncated>
root@debian10:~# mkdir -p /amanda/vtapes/slot{1,2,3,4}
root@debian10:~# mkdir -p /amanda/holding
root@debian10:~# mkdir -p /amanda/state/{curinfo,log,index}
root@debian10:~# mkdir -p /etc/amanda/MyConfig
root@debian10:~# vi /etc/amanda/MyConfig/amanda.conf
root@debian10:~# vi /etc/amanda/MyConfig/disklist
root@debian10:~# chown -R backup /amanda/
root@debian10:~# chown -R backup /etc/amanda/
root@debian10:~# usermod -s /bin/bash backup
root@debian10:~# vi /etc/amanda-security.conf
root@debian10:~# su - backup
backup@debian10:~$ amcheck MyConfig
<output truncated>
Client check: 1 host checked in 2.280 seconds.  0 problems found.
(brought to you by Amanda 3.5.1)
backup@debian10:~$ amdump MyConfig && echo $?
0
backup@debian10:~$ amreport MyConfig
<output truncated>
backup@debian10:~$
```

2. Populate **/home/project42** with some files (a copy of /etc or /sbin will do). Use **Amanda** to take a **full backup** of this directory.

```
root@debian10:~# find /etc -exec cp {} /home/project42/ \; 2>/dev/null
root@debian10:~# du -sh /home/project42/
41M     /home/project42/
root@debian10:~# vi /etc/amanda/MyConfig/disklist
root@debian10:~# cat /etc/amanda/MyConfig/disklist
localhost /etc simple-gnutar-local
localhost /home/project42 simple-gnutar-local
root@debian10:~# su - backup
backup@debian10:~$ amdump MyConfig && echo $?
0
backup@debian10:~$ amreport MyConfig
Hostname: debian10
Org     : MyConfig
Config  : MyConfig
Date    : January 13, 2020

These dumps were to tape MyData02.
The next tape Amanda expects to use is: 1 new tape.


STATISTICS:
                          Total       Full      Incr.   Level:#
                        --------   --------   --------  --------
Estimate Time (hrs:min)     0:00
Run Time (hrs:min)          0:00
Dump Time (hrs:min)         0:00       0:00       0:00
Output Size (meg)           38.4       38.2        0.2
Original Size (meg)         38.4       38.2        0.2
Avg Compressed Size (%)    100.0      100.0      100.0
DLEs Dumped                    2          1          1  1:1
Avg Dump Rate (k/s)      16960.3    32812.8      177.8
```

```
Tape Time (hrs:min)         0:00       0:00       0:00
Tape Size (meg)             38.4       38.2        0.2
Tape Used (%)               38.4       38.2        0.2
DLEs Taped                     2          1          1  1:1
Parts Taped                    2          1          1  1:1
Avg Tp Write Rate (k/s)   196400     390800     2000.0


USAGE BY TAPE:
  Label                 Time         Size      %  DLEs Parts
  MyData02              0:00       39280K   38.4     2     2


NOTES:
  planner: tapecycle (3) <= runspercycle (3)
  planner: Last full dump of localhost:/etc on tape MyData01 overwritten in 3 runs.
  planner: Adding new disk localhost:/home/project42.
  taper: Slot 2 without label can be labeled
  taper: tape MyData02 kb 39280 fm 2 [OK]


DUMP SUMMARY:
                                                        DUMPER STATS     TAPER STATS
HOSTNAME     DISK            L ORIG-KB  OUT-KB  COMP%  MMM:SS    KB/s MMM:SS     KB/s
-------------------------------- ---------------------- --------------- ---------------
localhost    /etc            1     200     200    --     0:01   177.7   0:00   2000.0
localhost    /home/project42 0   39080   39080    --     0:01 32788.0   0:00 390800.0

(brought to you by Amanda version 3.5.1)
backup@debian10:~$
```

3. Delete one of the files in **/home/project42**, then restore it from this backup.

```
root@debian10:~# ls -l /home/project42/hosts
-rw-r--r-- 1 root root 215 Jan 13 14:20 /home/project42/hosts
root@debian10:~# rm /home/project42/hosts
root@debian10:~# vi /var/backups/.amandahosts
root@debian10:~# cat /var/backups/.amandahosts
localhost backup
localhost root amindexd amidxtaped
root@debian10:~# cd /home/project42
root@debian10:/home/project42# amrecover MyConfig
AMRECOVER Version 3.5.1. Contacting server on localhost ...
220 debian10 AMANDA index server (3.5.1) ready.
Setting restore date to today (2020-01-13)
200 Working date set to 2020-01-13.
200 Config set to MyConfig.
501 Host debian10 is not in your disklist.
Trying host debian10.linux-training.be ...
501 Host debian10.linux-training.be is not in your disklist.
Trying host debian10 ...
501 Host debian10 is not in your disklist.
Trying host debian10.linux-training.be ...
501 Host debian10.linux-training.be is not in your disklist.
Use the sethost command to choose a host to recover
amrecover> sethost localhost
200 Dump host set to localhost.
amrecover> setdisk /home/project42
200 Disk set to /home/project42.
amrecover> add hosts
Added file /hosts
```

```
amrecover> extract

Extracting files using tape drive changer on host localhost.
The following tapes are needed: MyData02

Extracting files using tape drive changer on host localhost.
Load tape MyData02 now
Continue [?/Y/n/s/d]?
Restoring files into directory /home/project42
All existing files in /home/project42 can be deleted
Continue [?/Y/n]?

./hosts
39080 kb
amrecover> exit
200 Good bye.
root@debian10:/home/project42# !ls
ls -l /home/project42/hosts
-rw-r--r-- 1 root root 215 Jan 13 14:20 /home/project42/hosts
root@debian10:/home/project42#
```

4. Install and configure Bacula, as shown in the theory.

```
root@debian10:~# aptitude install bacula
<output truncated>
root@debian10:~# mkdir /backup
root@debian10:~# vi /etc/bacula/bacula-dir.conf
root@debian10:~# vi /etc/bacula/bacula-sd.conf
root@debian10:~#
```

5. Populate **/home/project33** with some files (a copy of /etc or /sbin will do). Use **Bacula** to take a **full backup** of this directory.

```
root@debian10:~# find /etc -exec cp {} /home/project33/ \; 2>/dev/null
root@debian10:~# du -sh /home/project33/
42M     /home/project33/
root@debian10:~# vi /etc/bacula/bacula-dir.conf
root@debian10:~# systemctl restart bacula-sd
root@debian10:~# systemctl restart bacula-dir
root@debian10:~# systemctl restart bacula-fd
root@debian10:~# systemctl restart bacula-director
root@debian10:~# bconsole
Connecting to Director localhost:9101
1000 OK: 103 debian10-dir Version: 9.4.2 (04 February 2019)
Enter a period to cancel a command.
*show filesets
FileSet: name=Full Set IgnoreFileSetChanges=0
      O M
      N
      I /home/project33
      N
      E /var/lib/bacula
      E /proc
      E /tmp
      E /sys
      E /.journal
      E /.fsck
      E /backup
      N
FileSet: name=Catalog IgnoreFileSetChanges=0
      O M
```

```
      N
      I /var/lib/bacula/bacula.sql
      N
*run
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
A job name must be specified.
The defined Job resources are:
     1: BackupClient1
     2: BackupCatalog
     3: RestoreFiles
Select Job resource (1-3): 1
Run Backup job
JobName:  BackupClient1
Level:    Incremental
Client:   debian10-fd
FileSet:  Full Set
Pool:     File (From Job resource)
Storage:  File1 (From Job resource)
When:     2020-01-13 18:56:01
Priority: 10
OK to run? (yes/mod/no): yes
Job queued. JobId=1
You have messages.
*quit
```

6. Delete one file in **/home/project33/** and then restore it from this last backup.

```
root@debian10:~# ls -l /home/project33/resolv.conf
-rw-r--r-- 1 root root 41 Jan 13 18:51 /home/project33/resolv.conf
root@debian10:~# rm /home/project33/resolv.conf

root@debian10:~# bconsole
Connecting to Director localhost:9101
1000 OK: 103 debian10-dir Version: 9.4.2 (04 February 2019)
Enter a period to cancel a command.
*restore
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Find the JobIds of the most recent backup for a client
    10: Find the JobIds for a backup for a client before a specified time
    11: Enter a list of directories to restore for found JobIds
    12: Select full restore to a specified Job date
    13: Cancel
Select item:  (1-13): 5
Automatically selected Client: debian10-fd
Automatically selected FileSet: Full Set
```

```
+-------+-------+----------+-----------+---------------------+------------+
| jobid | level | jobfiles | jobbytes  | starttime           | volumename |
+-------+-------+----------+-----------+---------------------+------------+
|     1 | F     |    1,347 | 39,876,214 | 2020-01-13 19:35:50 | Vol-0001   |
+-------+-------+----------+-----------+---------------------+------------+
You have selected the following JobId: 1

Building directory tree for JobId(s) 1 ...  +++
1,346 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$ cd /home/project33
cwd is: /home/project33/
$ mark resolv.conf
1 file marked.
$ done
Bootstrap records written to /var/lib/bacula/debian10-dir.restore.1.bsr

The Job will require the following (*=>InChanger):
   Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

    Vol-0001                  File1                     FileChgr1

Volumes marked with "*" are in the Autochanger.


1 file selected to be restored.

Using Catalog "MyCatalog"
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /var/lib/bacula/debian10-dir.restore.1.bsr
Where:           /backup/bacula-restores
Replace:         Always
FileSet:         Full Set
Backup Client:   debian10-fd
Restore Client:  debian10-fd
Storage:         File1
When:            2020-01-13 19:55:49
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): yes
Job queued. JobId=2
You have messages.
*quit
root@debian10:~# ls -l /backup/bacula-restores/home/project33/resolv.conf
-rw-r--r-- 1 root root 41 Jan 13 18:51 /backup/bacula-restores/home/project33/resolv. ↩
    conf
root@debian10:~#
```

Chapter 78

Time management

78.1 Time terminology

There is a lot to say about time, it is a curious thing in the universe. Managing the correct time on your Debian servers is important, so let us first agree on some terminology.

78.1.1 UTC

UTC or **coordinated universal time** is a scale maintained at **NIST** (the National Institute of Standards and Technology) by about ten atomic clocks. You can display UTC time with the **date -u** command. The output of this command is identical all over the world and also in the International Space Station.

```
paul@debian10:~$ date -u
Tue 29 Oct 2019 10:35:07 AM UTC
paul@debian10:~$
```

78.1.2 timezone

A timezone is a region on Earth with a uniform **standard time**. For example **UTC+01:00** is the identifier for the **Central European Time** and **West African Time**. Whereas **UTC-8:00** is the identifier for the **Pacific Time Zone** in USA and Canada (including parts of Mexico and some islands).

```
paul@debian10:~$ env TZ='America/New_York' date
Tue 29 Oct 2019 08:22:32 AM EDT
paul@debian10:~$ env TZ='Asia/Bangkok' date
Tue 29 Oct 2019 07:22:34 PM +07
paul@debian10:~$ env TZ='Europe/Madrid' date
Tue 29 Oct 2019 01:22:36 PM CET
paul@debian10:~$
```

The **TZ** variable needs to be correct, otherwise UTC is displayed. You can find all valid timezone strings in **/usr/share/zoneinfo/**. The **systemd** command **timedatectl** has its own list, type **timedatectl list-timezones** to see the **systemd** list.

```
paul@debian10:~$ timedatectl list-timezones | head -5
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
paul@debian10:~$
```

Use **date -R** to see the time according to your timezone, but with the offset to UTC mentioned. This offset is unambiguous compared to the (ambiguous) timezone information.

```
root@debian10:~# date
Tue Oct 29 14:59:17 CET 2019
root@debian10:~# date -R
Tue, 29 Oct 2019 14:59:18 +0100
root@debian10:~#
```

Another view of your time (zone) in relation to UTC can be displayed with the **systemd**'s **timedatectl** command by typing **timedatectl status** (or just **timedatectl**).

```
root@debian10:~# timedatectl status
               Local time: Tue 2019-10-29 15:55:32 CET
           Universal time: Tue 2019-10-29 14:55:32 UTC
                 RTC time: Tue 2019-10-29 14:55:33
                Time zone: Europe/Brussels (CET, +0100)
System clock synchronized: yes
```

```
              NTP service: active
          RTC in local TZ: no
root@debian10:~#
```

Setting the correct timezone is also done with the **timedatectl** command, as shown in this example.

```
root@debian10:~# timedatectl set-timezone 'Europe/Brussels'
root@debian10:~#
```

78.1.3 /etc/localtime

The local timezone is defined in **/etc/localtime** which is a link to the correct timezone file in **/usr/share/zoneinfo/**.

```
root@debian10:~# file /etc/localtime
/etc/localtime: symbolic link to /usr/share/zoneinfo/Europe/Brussels
root@debian10:~#
```

78.1.4 Daylight Saving Time

Some countries or areas change the clock twice a year to save energy (a disputed claim). For example Central European Time (CET in the above screenshot) is one hour ahead of UTC in winter and two hours ahead in summer.

Note that it is an offset to UTC that changes, UTC itself does not have DST. Below is a screenshot from **Summer Time** in Europe, note the two hour difference with UTC.

```
root@debian10:~# timedatectl
               Local time: Thu 2019-10-10 18:23:04 CEST
           Universal time: Thu 2019-10-10 16:23:04 UTC
                 RTC time: Thu 2019-10-10 16:23:04
                Time zone: Europe/Brussels (CEST, +0200)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
root@debian10:~#
```

78.2 NTP

The **Network Time Protocol** or **NTP** is a protocol that works on top of TCP/IP and that synchronises clocks on computers (both servers and clients). The most recent RFC's for NTP and SNTP are https://tools.ietf.org/rfc/rfc7822.txt and https://tools.ietf.org/-rfc/rfc4330.txt.

If the time on the server is more than 17 minutes apart from a time server (an NTP server), then no synchronisation will happen (because the time difference is above the **insane** threshold). You can use the deprecated **ntpdate** command to synchronise with a time server, this will set your server's clock to within a few milliseconds of the time server.

```
root@debian10:~# aptitude install ntpdate
<output truncated>
root@debian10:~# ntpdate pool.ntp.org
29 Oct 15:37:55 ntpdate[2808]: adjust time server 82.64.45.50 offset -0.033943 sec
root@debian10:~# ntpdate pool.ntp.org
29 Oct 15:38:37 ntpdate[2815]: adjust time server 82.64.45.50 offset 0.005487 sec
root@debian10:~# ntpdate pool.ntp.org
29 Oct 15:40:53 ntpdate[2818]: adjust time server 82.64.45.50 offset -0.022203 sec
root@debian10:~#
```

78.3 ntp.service?

The legacy **NTP Daemon** is deprecated in Debian 10 in favour of the default installation of **systemd-timesyncd.service** . You can still install **ntp** and then this **systemd** service will be disabled.

```
root@debian10:~# systemctl status systemd-timesyncd.service
* systemd-timesyncd.service - Network Time Synchronization
   Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; vendor prese
  Drop-In: /usr/lib/systemd/system/systemd-timesyncd.service.d
           `-disable-with-time-daemon.conf
   Active: active (running) since Tue 2019-10-29 19:13:10 CET; 2min 12s ago
     Docs: man:systemd-timesyncd.service(8)
 Main PID: 1570 (systemd-timesyn)
   Status: "Synchronized to time server for the first time 45.87.76.3:123 (0.debian.poo
    Tasks: 2 (limit: 1148)
   Memory: 1.4M
   CGroup: /system.slice/systemd-timesyncd.service
           `-1570 /lib/systemd/systemd-timesyncd

Oct 29 19:13:10 debian10 systemd[1]: Starting Network Time Synchronization...
Oct 29 19:13:10 debian10 systemd[1]: Started Network Time Synchronization.
Oct 29 19:14:01 debian10 systemd-timesyncd[1570]: Synchronized to time server for the f
root@debian10:~#
```

78.4 hwclock

78.5 ntp

78.6 rdate

78.7 dpkg-reconfigure tzdatas

===

78.8 Practice

78.9 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 79

git

79.1 about git

79.1.1 Linus Torvalds

When the Linux kernel could no longer use the version control system they had used for years, then Linus Torvalds himself decided to write his own tool named **git**. This was in 2005, and since then a huge number of projects have migrated to **git**, making it the most popular version control system.

79.1.2 revision control

Revision (or version) control is a very useful feature for developers, but also for many other people that work with regular updates in any kind of files. Revision control works best on text files, since you can attribute every single line of a text file in **git** to an author and a commit.

79.1.3 repository

With **git** every developer has a full copy of the repository (or version database). This means you can work on all files, even when the **git** server is unreachable. This is in contrast to legacy systems like CVS or Subversion.

79.1.4 for administrators?

Even though **git** is primarily aimed at developers, I personally think that administrators can benefit from using **git** for scripts and configuration files.

79.1.5 installation

We will use our trusty **debian10** server as a git server and **server2** as a git client. Install **git** on both machines. Client-server communication happens over ssh, so you may need to install **openssh-server** on the server.

```
root@debian10:~# aptitude install git
The following NEW packages will be installed:
  git git-man{a} liberror-perl{a} patch{a}
0 packages upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 7398 kB of archives. After unpacking 38.2 MB will be used.
Do you want to continue? [Y/n/?]
<output truncated>
```

```
root@server2:~# aptitude install git
<output truncated>
```

79.2 git server configuration

79.2.1 Creating a git user

In this screenshot a git user name **lego** is created with a special shell named **/usr/bin/git-shell** and with a password. Note that this password is not mandatory if you setup ssh keys.

```
root@debian10:~# useradd -m -d /home/lego -s /usr/bin/git-shell lego
root@debian10:~# passwd lego
New password:
Retype new password:
passwd: password updated successfully
root@debian10:~#
```

79.2.2 Creating two git directories

Two directories are initialised with **git init --bare**. One directory is for NXC programs (a C-like language to program the Lego NXT brick), the other is for LDD drawings (Lego Digital Designer).

```
root@debian10:~# mkdir /home/lego/nxc
root@debian10:~# mkdir /home/lego/ldd
root@debian10:~# cd /home/lego/nxc/
root@debian10:/home/lego/nxc# git init --bare
Initialized empty Git repository in /home/lego/nxc/
root@debian10:/home/lego/nxc# cd ../ldd/
root@debian10:/home/lego/ldd# git init --bare
Initialized empty Git repository in /home/lego/ldd/
```

Since we did all this as **root** we need to make sure all files belong to the **lego** user account.

```
root@debian10:/home/lego/ldd# cd
root@debian10:~# chown -R lego:lego /home/lego
root@debian10:~#
```

And that is it on the server. We now have two git repositories that we can access from a client.

79.3 git client setup

79.3.1 the two directories

The two directories where we want a git repository are **/home/paul/NXT** for the NXC programs and **/home/paul/Lego/LDD** for the LDD files. Both directories already have a small number of files.

```
paul@server2:~$ ls NXT/
arm.nxc  basis.nxc  menu.nxc
paul@server2:~$ ls Lego/LDD/
paul_arm_1dof.lxf  paul_arm_2dof.lxf  paul_basis.lxf
paul@server2:~$
```

79.3.2 global user name and email

We will use **server2** and the user **paul** as a client. First we install **git**, then we need to configure our username and email for **git**. This can be done with the **git config --global --edit** command, or with these two lines.

```
paul@server2:~$ git config --global user.name "Paul Cobbaut"
paul@server2:~$ git config --global user.email "paul@linux-training.be"
paul@server2:~$
```

79.3.3 initialising our first git directory

Next we go into the first directory that we want on our revision control server and initialise it with **git init**.

```
paul@server2:~$ cd NXT/
paul@server2:~/NXT$ ls -l
total 12
-rw-r--r-- 1 paul paul 487 Oct 29 20:31 arm.nxc
-rw-r--r-- 1 paul paul 853 Oct 29 20:31 basis.nxc
-rw-r--r-- 1 paul paul 983 Oct 29 20:31 menu.nxc
paul@server2:~/NXT$ git init
Initialized empty Git repository in /home/paul/NXT/.git/
paul@server2:~/NXT$
```

This **git init** command will create a hidden **.git** directory to contain all information about git. When you no longer want your project in **git** just remove this directory with **rm -rf .git**.

79.4 configuring the git directory

Next we need to configure the **git** directory with the coordinates of our **git** server. This happens in the **config** file in the hidden **.git** directory. The key is to get the **url =** line correct.

```
paul@server2:~/NXT$ vi .git/config
paul@server2:~/NXT$ cat .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = lego@192.168.56.101:nxc
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master

paul@server2:~/NXT$
```

79.4.1 staging two files in git

Next we tell the local **git** to stage two files in this local git repository. Then we add a **commit** message for these two files. Once committed these files are part of the local git repository.

```
paul@server2:~/NXT$ ls
arm.nxc  basis.nxc  menu.nxc
paul@server2:~/NXT$ git add arm.nxc
paul@server2:~/NXT$ git add basis.nxc
paul@server2:~/NXT$ git commit -m 'Starting a Lego project'
[master (root-commit) 71253d2] Starting a Lego project
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 arm.nxc
 create mode 100644 basis.nxc
paul@server2:~/NXT$
```

79.4.2 pushing to the server

With **git push** we synchronise the local git repository with the remote location. The transfer of data happens over **ssh**.

```
paul@server2:~/NXT$ git push
lego@192.168.56.101's password:
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 806 bytes | 806.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To 192.168.56.101:nxc
 * [new branch]      master -> master
paul@server2:~/NXT$
```

79.5 And the second directory

This is similar to the first directory, only some filenames and the git server location URL are different.

```
paul@server2:~$ cd Lego/LDD/
paul@server2:~/Lego/LDD$ ls
paul_arm_1dof.lxf  paul_arm_2dof.lxf  paul_basis.lxf
paul@server2:~/Lego/LDD$ git init
Initialized empty Git repository in /home/paul/Lego/LDD/.git/
paul@server2:~/Lego/LDD$ cp ~/NXT/.git/config .git/
paul@server2:~/Lego/LDD$ vi .git/config
paul@server2:~/Lego/LDD$ cat .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = lego@192.168.56.101:ldd
        fetch = refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master

paul@server2:~/Lego/LDD$ git add paul_arm_1dof.lxf
paul@server2:~/Lego/LDD$ git add paul_arm_2dof.lxf*
paul@server2:~/Lego/LDD$ git commit -m 'Building an arm'
[master (root-commit) 93853e2] Building an arm
 2 files changed, 0 insertions(), 0 deletions(-)
 create mode 100644 paul_arm_1dof.lxf
 create mode 100644 paul_arm_2dof.lxf
paul@server2:~/Lego/LDD$ git push
lego@192.168.56.101's password:
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 44.78 KiB | 22.39 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To 192.168.56.101:ldd
 * [new branch]      master -> master
paul@server2:~/Lego/LDD$
```

79.5.1 git status

The **git status** command will give you an update on untracked files, and on the number of committed files that are not yet pushed to the server.

```
paul@server2:~/Lego/LDD$ git status
On branch master
Your branch is up to date with origin/master.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        paul_basis.lxf

nothing added to commit but untracked files present (use "git add" to track)
paul@server2:~/Lego/LDD$
```

79.5.2 .gitignore

It is entirely possible that you want files in your **git** directory that you never want on the server. To prevent **git status** from displaying these files you can create a **.gitignore** file and list files to be ignored in it. The screenshot shows my personal **.gitignore** file when writing this book.

```
paul@MBDebian~/lt4/git$ cat .gitignore
.gitignore
full.adoc
full.xml
full.pdf
images/tux.png
chapters/later.txt
.vimrc
paul@MBDebian~/lt4/git$
```

79.5.3 git pull

When multiple users connect to the same git server, then you will want to perform a **git pull** regularly to be up-to-date with all the files on the server.

```
paul@server2:~/NXT$ git pull
Already up to date.
paul@server2:~/NXT$
```

Tip
We covered only the very basics in this chapter. There are numerous tutorials and video's on the internet that explain a more complete git workflow.

79.6 passwordless ssh

You can set up **ssh** with public/private keys and an empty passphrase (or better with ssh-agent) to prevent **git push** from asking a password.

79.6.1 on the client

On the client we use **scp** to copy the public key to the server. You cannot directly copy the key because the **lego** user has **git-shell** as a shell.

```
paul@server2:~$ scp .ssh/id_rsa.pub 192.168.56.101:~
paul@192.168.56.101's password:
id_rsa.pub                                    100%  394   569.7KB/s   00:00
paul@server2:~$
```

79.6.2 on the server

On the server we need to copy the public key from the **paul** account to the correct location in the **lego** account, and make sure that **lego** owns all the files with proper permissions.

```
root@debian10:~# mkdir /home/lego/.ssh
root@debian10:~# chmod 700 /home/lego/.ssh
root@debian10:~# cp /home/paul/id_rsa.pub /home/lego/.ssh/authorized_keys
root@debian10:~# chown -R lego:lego /home/lego/.ssh/
root@debian10:~# chmod 600 /home/lego/.ssh/authorized_keys
root@debian10:~#
```

79.7 Practice

1. Decide on a git server and install **openssh-server** and **git** on it.
2. Create a user account for **git** named **git**.
3. Create a directory to host the **git** repository, initialise it and set proper ownership.
4. Decide on a git client computer and install **git** on it. Also set a global **user.name** and **user.email** for git on this computer. (Please use your name and email, not mine.)
5. Decide on a git directory on this client computer and initialise it. Then configure the directory with the server information.
6. Now copy (or create) a file in this directory and add it to the staging area.
7. Commit this file and push it to the git server.
8. Create a file that ignores the main.txt file in this directory.

79.8 Solution

1. Decide on a git server and install **openssh-server** and **git** on it.

```
aptitude install git openssh-server
```

2. Create a user account for **git** named **git**.

```
useradd -m -s /usr/bin/git-shell git
```

3. Create a directory to host the **git** repository, initialise it and set proper ownership.

```
mkdir /home/git/repo
cd /home/git/repo
git init --bare
chown -R git:git /home/git
```

4. Decide on a git client computer and install **git** on it. Also set a global **user.name** and **user.email** for git on this computer. (Please use your name and email, not mine.)

```
root# aptitude install git
paul$ git config --global user.name "Paul Cobbaut"
paul$ git config --global user.email "paul@linux-training.be"
```

5. Decide on a git directory on this client computer and initialise it. Then configure the directory with the server information.

```
mkdir project42
cd project42
git init
vi .git/config
```

6. Now copy (or create) a file in this directory and add it to the staging area.

```
touch main.cpp
git add main.cpp
```

7. Commit this file and push it to the git server.

```
git commit -m 'Initial commit of main.cpp'
git push
```

8. Create a file that ignores the main.txt file in this directory.

```
vi .gitignore     ## type main.txt on line 1
```

Part VII

Linux Servers

Chapter 80

xen

80.1 About Xen

80.2 installation

Installation of **Xen** via **apt-get install xen-system** will add an entry in **grub2** to enable booting with or without the **xen hypervisor**.

```
root@MBDebian~# apt-get install xen-system xen-tools
```

===

80.3 Creating an Xen image

```
root@MBDebian~# xen-create-image --hostname xen1.xen.local --ip 192.168.1.101 --vcpus 1 -- ↩
    pygrub --dist buster --size=8gb --fs=ext4 --image=sparse --memory=512Mb --swap=512Mb
WARNING:  No gateway address specified!
WARNING:  No netmask address specified!

General Information
--------------------
Hostname       :  xen1.xen.local
Distribution   :  buster
Mirror         :  http://deb.debian.org/debian
Partitions     :  swap            512Mb (swap)
                  /               8gb   (ext4)
Image type     :  sparse
Memory size    :  512Mb
Bootloader     :  pygrub

Networking Information
----------------------
IP Address 1   : 192.168.1.101 [MAC: 00:16:3E:CB:F0:1A]

WARNING
-------
Loopback module not loaded and you're using loopback images
Run the following to load the module:

modprobe loop max_loop=255


Creating partition image: /home/xen/domains/xen1.xen.local/swap.img
Done

Creating swap on /home/xen/domains/xen1.xen.local/swap.img
Done

Creating partition image: /home/xen/domains/xen1.xen.local/disk.img
Done

Creating ext4 filesystem on /home/xen/domains/xen1.xen.local/disk.img
Done
```

```
Installation method: debootstrap
Done

Running hooks
Done

No role scripts were specified.  Skipping

Creating Xen configuration file
Done

No role scripts were specified.  Skipping
Setting up root password
Generating a password for the new guest.
All done


Logfile produced at:
         /var/log/xen-tools/xen1.xen.local.log

Installation Summary
--------------------
Hostname        :  xen1.xen.local
Distribution    :  buster
MAC Address     :  00:16:3E:CB:F0:1A
IP Address(es)  :  192.168.1.101
SSH Fingerprint :  SHA256:xvcu/0smpDj3+BExbxZZ0/8DaJEaVbCaqQwp60Fflx8 (DSA)
SSH Fingerprint :  SHA256:bJfySQD0tXxxfIAtCXC3Jv9fEKxTgl1bR4oUAFTeZ58 (ECDSA)
SSH Fingerprint :  SHA256:GiaGEpGViaEs8WVC1vBnDLh2HMe1hD0NwbndCXYz8yM (ED25519)
SSH Fingerprint :  SHA256:5xoZoVcbzL4O0ynTaRJJTyrIM6xmZNTwTTr0P3YxYXQ (RSA)
Root Password   :  VuSMEy3QmYcZwN7JBcxJ8Jk

root@MBDebian~#
```

This created the **/home/xen** directory that we configured in **/etc/xen-tools/xen-tools.conf**.

```
root@MBDebian~# ls /home/xen/domains/
xen1.xen.local
root@MBDebian~#
```

And this image can be quickly verified with **xen-list-images**.

```
root@MBDebian~# xen-list-images
Name: xen1.xen.local
Memory: 512 MB
IP: 192.168.1.101
Config: /etc/xen/xen1.xen.local.cfg
root@MBDebian~#
```

80.4 Practice

80.5 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 81

DHCP

81.1 Dynamic Host Configuration Protocol

81.2 DHCP Server

We will set up our trusty **debian10** server as a DHCP server. Then below we will configure **server2** as a DHCP client. Both computers are running Debian 10 Buster.

81.2.1 installation

The standard DHCP server on Debian 10 is **isc-dhcp-server** from the **I**nternet **S**ystems **C**onsortium. You can find more information here https://www.isc.org/dhcp/ .

```
root@debian10:~# aptitude install isc-dhcp-server
The following NEW packages will be installed:
  isc-dhcp-server libirs-export161{a} libisccfg-export163{a} policycoreutils{a}
  selinux-utils{a}
0 packages upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 1615 kB of archives. After unpacking 6539 kB will be used.
Do you want to continue? [Y/n/?]
<output truncated>
```

You will receive failed messages because the DHCP is not set up, but the post-install script decides to start it anyway.

81.2.2 /etc/default/isc-dhcp-server

The first thing to do is to configure the interface on which the DHCP server should function. You can list available interfaces with **netstat -i** or with **ip l**.

```
root@debian10:~# netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enp0s3    1500     1258      0      0 0           871      0      0      0 BMRU
enp0s8    1500     1470      0      0 0           577      0      0      0 BMRU
enp0s9    1500       21      0      0 0            26      0      0      0 BMRU
lo       65536        2      0      0 0             2      0      0      0 LRU
root@debian10:~#
```

This server has three network interfaces (not counting the loopback interface **lo**). We choose to run the DHCP server on the **enp0s3** interface. Make sure that the interface that you choose is connected to the correct network!

Configuration of this interface is done in **/etc/default/isc-dhcp-server**, there you can also choose the configuration files (we leave the defaults) and the ipv4 and/or ipv6 protocol.

```
root@debian10:~# vi /etc/default/isc-dhcp-server
root@debian10:~# grep enp0 /etc/default/isc-dhcp-server
INTERFACESv4="enp0s3"
root@debian10:~#
```

81.2.3 static IP

A DHCP server needs a static IP on the interface that is connected to the DHCP network. This configuration is done in **/etc/network/interfaces**.

```
root@debian10:~# vi /etc/network/interfaces
root@debian10:~# grep -A2 enp0s3 /etc/network/interfaces
auto enp0s3
allow-hotplug enp0s3
iface enp0s3 inet static
address 192.168.56.101/24

root@debian10:~#
```

After this change restart the networking with **systemctl restart networking** and then verify with **ip** that the configuration of your interface was without errors.

```
root@debian10:~# systemctl restart networking.service
root@debian10:~# ip a show dev enp0s3 | grep 192
    inet 192.168.56.101/24 brd 192.168.56.255 scope global enp0s3
root@debian10:~#
```

81.2.4 subnet and range

The DHCP server works in a certain subnet (192.168.56.0/24 in this example) and has a range of addresses available to hand out to clients (.201 till .210 in this example). In this screenshot we create a very minimalistic DHCP server configuration.

```
root@debian10:~# vi /etc/dhcp/dhcpd.conf
root@debian10:~# cat /etc/dhcp/dhcpd.conf
subnet 192.168.56.0 netmask 255.255.255.0 {
  range 192.168.56.201 192.168.56.210;
}
root@debian10:~#
```

81.2.5 restart the service

It is now time to restart the **isc-dhcp-server** service and to verify that it is running properly.

```
root@debian10:~# systemctl restart isc-dhcp-server.service
root@debian10:~# systemctl status isc-dhcp-server.service
* isc-dhcp-server.service - LSB: DHCP server
   Loaded: loaded (/etc/init.d/isc-dhcp-server; generated)
   Active: active (running) since Wed 2019-10-30 15:21:37 CET; 5s ago
<output truncated>
```

81.3 DHCP Client

Chances are you don't have to do anything on the client, if it is already using DHCP. Except maybe renew the DHCP lease.

```
root@server2:~# ifdown enp0s3 && ifup enp0s3
Killed old client process
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
```

```
Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPRELEASE of 192.168.56.102 on enp0s3 to 192.168.56.100 port 67
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 4
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 8
DHCPOFFER of 192.168.56.201 from 192.168.56.101
DHCPREQUEST for 192.168.56.201 on enp0s3 to 255.255.255.255 port 67
DHCPACK of 192.168.56.201 from 192.168.56.101
bound to 192.168.56.201 -- renewal in 16770 seconds.
root@server2:~#
```

Note
Typing **ifdown enp0s3** when connected over ssh on that interface, will break your connection (until that interface is up again with the same IP).

81.3.1 /etc/network/interfaces

For completeness, here is the configuration of the **enp0s3** adapter on the client in **/etc/network/interfaces**. This was the default configuration after installation of Debian 10.

```
root@server2:~# grep enp0s3 /etc/network/interfaces
allow-hotplug enp0s3
iface enp0s3 inet dhcp
root@server2:~#
```

81.4 IP address reservation

You can **reserve** IP-addresses for certain clients in the DHCP server. This is done by linking the MAC-address of a client to an IP-address.

```
root@debian10:~# cat /etc/dhcp/dhcpd.conf
subnet 192.168.56.0 netmask 255.255.255.0 {
  range 192.168.56.201 192.168.56.210;
}

host server2 {
  hardware ethernet 08:00:27:f5:54:03;
  fixed-address 192.168.56.42;
}

root@debian10:~#
```

This can be tested on the client by typing **ifdown enp0s3 && ifup enp0s3**. The client should then have 192.168.56.42 as IP-address.

```
root@server2:~# ifdown enp0s3 && ifup enp0s3
Killed old client process
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPRELEASE of 192.168.56.201 on enp0s3 to 192.168.56.101 port 67
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 7
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 8
DHCPOFFER of 192.168.56.42 from 192.168.56.101
DHCPREQUEST for 192.168.56.42 on enp0s3 to 255.255.255.255 port 67
DHCPACK of 192.168.56.42 from 192.168.56.101
bound to 192.168.56.42 -- renewal in 18688 seconds.
root@server2:~#
```

81.5 lease time

The DHCP server can force a certain lease time for the IP configuration that it hands out to clients. In this example we add two lines to **/etc/dhcp/dhcpd.conf** to force clients to renew their lease every ten minutes (600 seconds).

```
root@debian10:~# vi /etc/dhcp/dhcpd.conf
root@debian10:~# tail -3 /etc/dhcp/dhcpd.conf

default-lease-time 600;
max-lease-time 600;
root@debian10:~#
```

81.6 domain name

You can also set a **domain name** and **DNS server** option in **/etc/dhcp/dhcpd.conf**. We set them here as global variables, but they can also occur within a subnet or with a reservation.

```
root@debian10:~# vi /etc/dhcp/dhcpd.conf
root@debian10:~# cat /etc/dhcp/dhcpd.conf
subnet 192.168.56.0 netmask 255.255.255.0 {
  range 192.168.56.201 192.168.56.210;
}

host server2 {
  hardware ethernet 08:00:27:f5:54:03;
  fixed-address 192.168.56.42;
}

default-lease-time 600;
```

```
max-lease-time 600;

option domain-name "linux-training.lan";
option domain-name-servers 8.8.8.8;
root@debian10:~#
```

Don't forget to restart the service with **systemctl restart isc**-tabkey and verify the **/etc/resolv.conf** file on the client (after renewing the lease).

```
root@server2:~# ifdown enp0s3 && ifup enp0s3
Killed old client process
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPRELEASE of 192.168.56.42 on enp0s3 to 192.168.56.101 port 67
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/enp0s3/08:00:27:f5:54:03
Sending on   LPF/enp0s3/08:00:27:f5:54:03
Sending on   Socket/fallback
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 8
DHCPDISCOVER on enp0s3 to 255.255.255.255 port 67 interval 10
DHCPOFFER of 192.168.56.42 from 192.168.56.101
DHCPREQUEST for 192.168.56.42 on enp0s3 to 255.255.255.255 port 67
DHCPACK of 192.168.56.42 from 192.168.56.101
bound to 192.168.56.42 -- renewal in 251 seconds.
root@server2:~# cat /etc/resolv.conf
domain linux-training.lan
search linux-training.lan
nameserver 8.8.8.8
root@server2:~#
```

81.7 Practice

81.8 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 82

DNS

82.1 Explaining DNS

DNS resolves hostnames to IP-addresses.

82.1.1 host for an A record

Typing linux-training.be in a browser then e DNS query is sent to resolve linux-training.be to 88.151.243.8. The browser will set up a TCP connection with this IP-address and will use HTTP to obtain the website.

```
paul@MBDebian~$ host -t a linux-training.be
linux-training.be has address 88.151.243.8
paul@MBDebian~$
```

The **-t a** option queries for the A record in DNS. An A record is a record that associates a DNS name with an IP-address.

82.1.2 host for PTR record

Reverse lookup of PTR record.

```
paul@MBDebian~$ host 88.151.243.8
8.243.151.88.in-addr.arpa domain name pointer fosfor.openminds.be.
paul@MBDebian~$
```

82.1.3 nslookup for A record (first time)

The **nslookup** tool will tell you which DNS server it queries (the first two lines of the output) and then the answer to your query.

```
paul@MBDebian~$ nslookup linux-training.be
Server:         195.130.130.11
Address:        195.130.130.11#53

Non-authoritative answer:
Name:   linux-training.be
Address: 88.151.243.8

paul@MBDebian~$
```

82.1.4 dig for an A record

```
paul@MBDebian~$ dig linux-training.be

; <<>> DiG 9.11.5-P4-5.1-Debian <<>> linux-training.be
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18378
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;linux-training.be.             IN      A

;; ANSWER SECTION:
linux-training.be.      922     IN      A       88.151.243.8
```

```
;; Query time: 37 msec
;; SERVER: 195.130.130.11#53(195.130.130.11)
;; WHEN: Wed Nov 06 15:02:49 CET 2019
;; MSG SIZE  rcvd: 62

paul@MBDebian~$
```

82.1.5 nslookup for NS record

Looking for **name servers** (or NS records) with **nslookup**.

```
paul@MBDebian~$ nslookup
> set type=NS
> linux-training.be
Server:         195.130.130.11
Address:        195.130.130.11#53

Non-authoritative answer:
linux-training.be       nameserver = ns3.om-powered.net.
linux-training.be       nameserver = ns2.openminds.be.
linux-training.be       nameserver = ns1.openminds.be.

Authoritative answers can be found from:
>
```

82.1.6 nslookup for authoritative NS response

To get an authoritative response query the authoritative **name server** directly with **nslookup**.

```
> server ns1.openminds.be
Default server: ns1.openminds.be
Address: 195.47.215.14#53
Default server: ns1.openminds.be
Address: 2a02:d08:53::a1#53
> linux-training.be
Server:         ns1.openminds.be
Address:        195.47.215.14#53

linux-training.be       nameserver = ns1.openminds.be.
linux-training.be       nameserver = ns3.om-powered.net.
linux-training.be       nameserver = ns2.openminds.be.
>
```

82.1.7 host for NS record

So which are the **DNS servers** or **name servers** that are authoritative for linux-training.be? This can be answered by the **host -t ns** command.

```
paul@MBDebian~$ host -t ns linux-training.be
linux-training.be name server ns3.om-powered.net.
linux-training.be name server ns2.openminds.be.
linux-training.be name server ns1.openminds.be.
paul@MBDebian~$
```

82.1.8 host for MX record

When sending e-mail your local mail server will need to know where to send it. It obtains this information by querying for an MX record of the domain.

```
paul@MBDebian~$ host -t mx linux-training.be
linux-training.be mail is handled by 10 argon.openminds.be.
linux-training.be mail is handled by 20 hotel.openminds.be.
paul@MBDebian~$
```

82.1.9 dig for MX at 8.8.8.8

```
paul@MBDebian~$ dig @8.8.8.8 linux-training.be mx +short
20 hotel.openminds.be.
10 argon.openminds.be.
paul@MBDebian~$
```

82.1.10 dig +short alias

```
paul@MBDebian~$ alias dig=dig +short
paul@MBDebian~$ dig linux-training.be
88.151.243.8
paul@MBDebian~$
```

82.2 DNS software

82.2.1 BIND

82.3 Caching only DNS server

82.3.1 What it is.

82.3.2 Configuration

82.3.3 Testing

===

===

```
```

```
===
```

```
```

```
===
```

```
```

```
===
```

```
```

```
===
```

```
```

```
===
```

```
```

82.4 Practice

82.5 Solution

.
+

.
+

.
+

.
+

.
+

.
+

.
+

.
+

Chapter 83

Index

Colophon

Linux Fun

Published as a pdf file on linux-training.be website