High-Performance Computing: Linux

Adrian F. Clark: alien@essex.ac.uk

2016–17

What is Linux?

Linux is a version of Unix, an operating system that traces its roots back to Bell Labs in the 1970s, developed by Ken Thompson, Dennis Ritchie (who also invented the C language) and others

Linux itself was developed by Linus Torvalds and it became popular principally because Linus encouraged other people to contribute to it

Although initially developed for PC-class hardware, Linux now runs on a wide range of hardware, from supercomputer-class machines down to mobile 'phones and cameras

There are several Linux variants, and the one used in CSEE is Ubuntu — you should be able to re-boot any machine in our PC labs into Linux

The Unix philosophy

PC-class computers running Linux are used in much the same way as those running Windows, though the detail is somewhat different

Although there are Linux equivalents of the kinds of tools you will have encountered on Windows — Word, web browsers, development environments, and so on — these are not the traditional way of working on Unix

The Unix philosophy can be thought of *Do one thing; do it well* and this results in a fairly large number of fairly task-specific programs — often with fairly obscure names, alas

These tools are best used from the command line, so we shall focus on this

When logged into Ubuntu Linux, the fastest way to bring up a terminal window is to type the key combination `Ctrl-Alt-t`

The shell

When you bring up a terminal window, you will see a prompt appear in it

The program that produced the prompt is called the *shell* — there are several possible shells that you can use but the default one is called `bash`

When you type a command, `bash` looks for a program whose name matches the command and runs it

This means it is possible to add new commands to a Unix system

The shell also has built-in commands for things like loops and conditionals, so it's fairly easy to write complete programs ("shell scripts')

Directories and files

You are undoubtedly familiar with files and folders from Windows — though what you call *folders* on Windows are termed *directories* on Unix

Unix has the concept of a *working directory*, meaning that commands will normally work on files in that directory only, and you can find what that is by typing the command `pwd` ("print working directory") followed by the return key

When you log into a Unix machine, you will initially be in your "home" directory

Files can have *any* name; a fairly small number of programs (*e.g.*, compilers) require specific file extensions (*e.g.*, `.cc` for C++)

It is best to avoid using spaces in file and directory names

The Unix filesystem

On Windows, you will be familiar with the various parts of a file's name being separated by the backslash character — on Unix, the parts are similar but separated by a *forward* slash /

On Windows, there is a separate 'tree' of directories on each device but on Unix, *everything* appears in a single tree

Devices can be 'mounted' at any position in the tree, and the command `df` lists them

Listing your files

To find out what files reside in your working directory, type the `ls` ("list") command

```
01-introduction.md    03-answers.md    04-performance.md
02-linux.md           03-clusters.md
```

This tells you nothing about the size of the file, but the command `ls -l` ("long list") gives you output like

```
-rw-r--r--  1 alien  staff  7800 10 Oct 08:57 01-introduction.md
-rw-r--r--  1 alien  staff  2125 13 Oct 12:42 02-linux.md
-rw-r--r--  1 alien  staff  3668 13 Oct 12:16 03-answers.md
-rw-r--r--  1 alien  staff  5724 13 Oct 12:13 03-clusters.md
-rw-r--r--  1 alien  staff   558  5 Oct 15:30 04-performance.md
```

Looking at files

To view the contents of a file, you can use the `cat` ("concatenate") command — it will become clear why it is called this later

```
cat 02-linux.md
```

To read the file a screenful at a time, use

```
less 02-linux.md
```

The space bar moves forward through the file by one screenful, and `b` backwards by the same amount

Command editing

You can re-run the previous command by typing the *up-arrow* key and then the return key — quite a few previous commands are retained

The `history` command lists out previous commands

It's common to make a typing mistake when entering commands, and the easiest way to correct one is to type the *up-arrow* key, and then the *left-* and *right-arrow* keys to find the mistake, then correct it and hit return

Filename completion

Typing long filenames quickly becomes tiresome, so there are shortcuts to make your life easier

`!$` is the last space-separated word on the previous command, so you might end up doing something like

```
cat 02-linux.md
less !$
```

If you type `less 01` and then hit the `TAB` key, then the shell will fill in the remainder of the file (if it is specified uniquely), or as much as possible if it isn't, prompting you with the filenames matched

Wildcards in filenames

You often want to manipulate groups of files so, rather than having to process each one with a separate command, the shell lets you specify them all using *wildcards*, as in

```
ls *.md
```

The two you will use most often are * (match any number of characters) and ? (match a single character)

Experienced Unix users tend to name files in a consistent way, so that it will be easy to use wildcards

Creating files and directories

To create a directory in the current one, use a command like

```
mkdir reports
```

To delete the directory `reports`, type

```
rmdir reports
```

Having created a directory, you can make it your working directory with the `cd` ("change directory") command

```
cd reports
```

To create a file call `try.txt`, type the command

```
touch try
```

To change the name of a file, use the `mv` (“move”) comamnd

```
mv try try.text
```

To delete a file, use the `rm` (“remove”) command

```
rm try.text
```

We shall look at more useful ways of creating files shortly

Special directories

Your current directory is always called `.`

The directory above your current directory in the hierarchy is always called `..`

In the shell, your login directory is called `~`

In the shell, user `joe`'s login directory is called `~joe` — note how this is different from `~/joe` which would be a file or directory named `joe` in your login directory

Re-direction

You can save the output of any command to a file by *command re-direction*:

```
ls > myfiles.txt
```

By default, the shell does not let you overwrite files using command re-direction

There is a similar syntax for *reading* from a file:

```
cat < myfiles.txt
```

These work by manipulating the *standard input* and *standard output* channels found in most programming languages

Pipes

You can also make the output of one program become the input to another with a *pipe*, represented as a vertical bar on the command line

For example, the following idiom is a common way to count the number of files in a directory

```
ls -l | wc -l
```

The `ls -l` command generates a single line of output for each file, and `wc -l` counts the number of lines in its input

Finding out more

This has provided only a brief introduction to the Unix shell and its commands. There are many good places to find out more, including:

- Linux tutorial for beginners
- Introduction to Unix (a entire course!)
- Learning the shell
- My notes on program development under Linux

Linux on networks

To make the subsequent examples a little more concrete, imagine you are:

- logged in on a Linux machine called `vulcan`
- interested in working with a remote machine called `mars.essex.ac.uk`

Important networking commands

To find if `mars` is on the Internet, use the command

```
ping mars.essex.ac.uk
```

To find out the route that packets take to reach `mars`, type

```
traceroute mars.essex.ac.uk
```

To find the DNS record for `mars`, use

```
nslookup mars.essex.ac.uk
```

If `vulcan` is connected to the Essex network, you can omit the `.essex.ac.uk`

Logging onto a remote machine

If you have an account on `mars`, the normal way to log onto it is using the *secure shell*, `ssh`:

```
ssh mars.essex.ac.uk
```

To login on `mars` as the user `joe`, use

```
ssh joe@mars.essex.ac.uk
```

You will normally need to enter the password *on the remote machine* to login

Copying files around the network

To copy the file `info.txt` in your current directory to the directory `info-files` on `mars`, use

```
scp info.txt mars.essex.ac.uk:info-files/
```

To copy the file `useful.py` from folder `python` in user `joe`'s account on `mars` to your current directory, type

```
scp joe@mars.essex.ac.uk:python/useful.py .
```

You will be prompted for `joe`'s password on `mars`

The r-tools

ssh and scp encrypt all the traffic they send over the network — this is good for general Internet use but overkill for a cluster on a private network

For private networks (**only**), it is easier to use the so-called r-tools, rlogin, rsh, and rcp, which perform similar tasks to ssh and scp but without encryption

If you create a file called .rhosts on a machine which contains the names of *trusted* hostnames, one per line, you will not be prompted for a password when logging in from one of those machines — this is obviously a glaring security hole in general but really useful on a cluster lying on a dedicated network

Running remote commands

Using `ssh`

```
ssh mars ls -l info-files
```

Using `rsh`

```
rsh cluster01 'nohup ./myjob &'
```

Note the use of:

- `&` at the end of the command to unhook it from `rsh`
- `nohup` to keep `myjob` running when `rsh` finishes