

```
4096 Jun 6 19:15 bin
4096 Jun 15 11:38 boot
4096 Jun 6 19:05 cdrom
4020 Aug 5 01:46 dev
12288 Jun 15 11:37 etc
4096 Jun 6 19:06 home
 33 Jun 15 11:36 initrd.img -> boot/initrd.im
 33 Jun 15 11:36 initrd.img.old -> boot/initr
4096 Jun 6 19:08 lib
4096 Apr 26 19:18 lib64
16384 Jun 6 19:01 lost+found
4096 Apr 26 19:18 media
4096 Apr 26 19:18 mnt
4096 Apr 26 19:18 opt
 0 Aug 5 01:46 proc
4096 Apr 26 19:28 root
748 Aug 5 01:46 run
 5 Jun 6 19:15 sbin
```



The DoCSoc Guide To: Linux & The Command Line

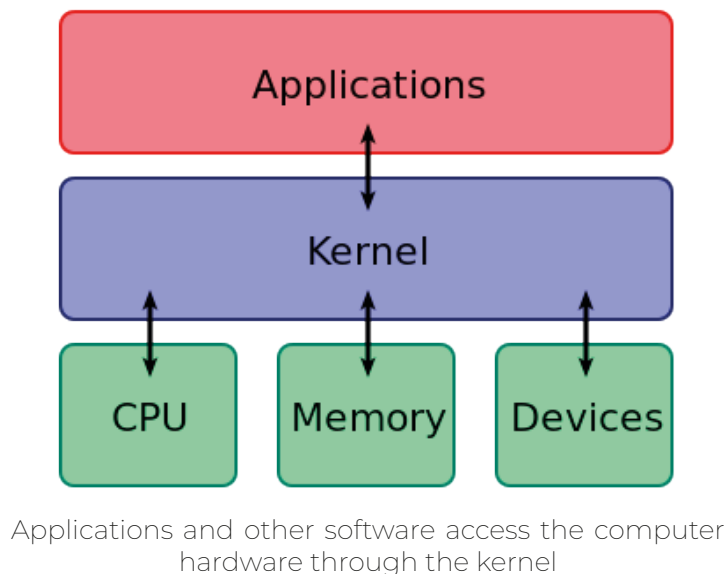
Table of contents

What is Linux?	4
The command line	5
More commands	12
Advanced command line concepts	22
Package managers	24
Text editors	25

What is Linux?

The word 'Linux' is actually used to describe two separate things: a family of open-source operating systems, and what is called the **kernel** that underlies those operating systems. The kernel is the core part of an operating system, and provides the interface between software and a computer's hardware. In this guide we'll just refer to the family of operating systems.

A Linux operating system typically includes the Linux **kernel**, plus a **package management system** (see the Package Managers section for more info), some libraries provided by GNU, and some other tools & utilities such as display managers. For that reason, Linux-based operating systems are sometimes referred to as 'GNU/Linux' to distinguish it from just the Linux kernel.



The Linux family is a subset of a larger family called Unix, which encompasses other operating systems such as macOS, BSD and Solaris. Windows is not part of the Unix family.

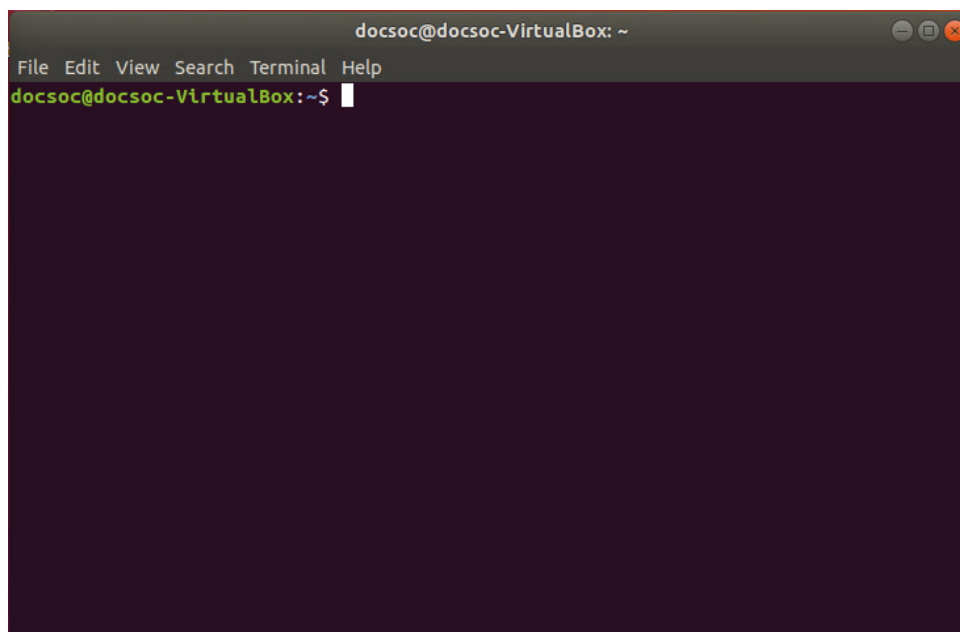
What is a distribution?

Each operating system within the Linux family is called a **distribution**. Each distribution provides a graphical user interface and package management system, whilst still using the Linux kernel and several other GNU libraries. Over 300 flavours of Linux now exist, so picking a single one to use can be hard! Some popular distributions include Ubuntu (installed on lab machines), Debian, Fedora, and Linux Mint.

The command line

In your applications folder there's a program called 'Terminal'. The Linux terminal provides a **command line interface** between you and your computer, which is named as such since the only thing you're allowed to do is enter commands on-screen. The terminal comes with a huge list of preset commands, and you can install even more commands with package managers. Almost all programming on Linux involves good knowledge of how to use the command line interface.

For now, we'll focus on some basic commands that allow you to interact with your computer. Start by opening the Terminal app; you will be greeted with a command line interface like so:



Don't worry about the left hand side too much yet - this reads as your user-name '@' the hostname of your computer

TIP: You can use Ctrl+ and Ctrl- to make the text bigger and smaller.

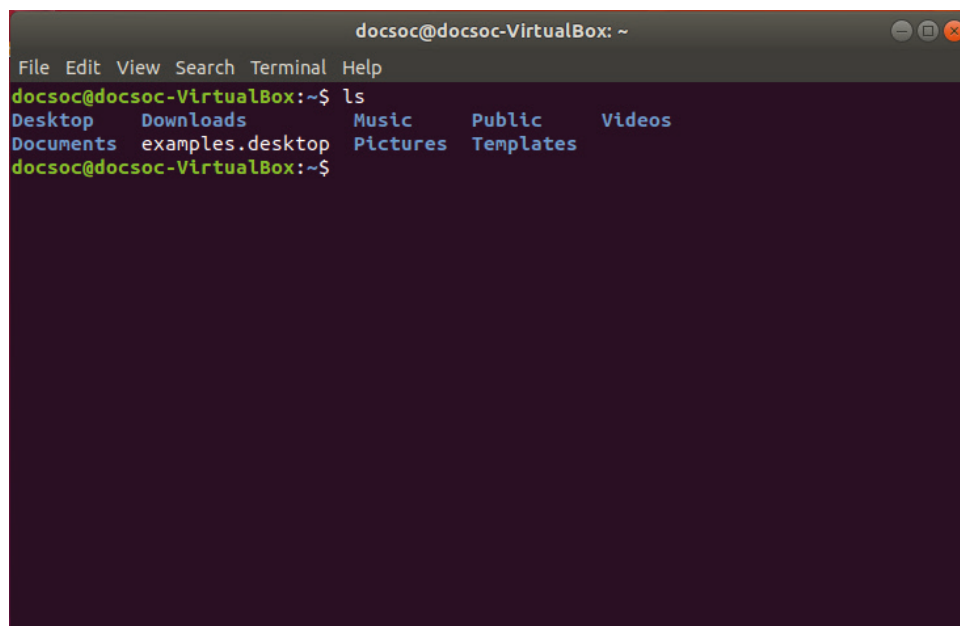
Before we begin, we'll have to explain some terms. Don't worry if these don't make sense at first, you'll pick up concepts quickly when experimenting with the **ls** and **cd** commands:

- Every command you execute in the terminal takes place from the perspective of a certain directory, called your **working directory**.

- The **root directory** is the highest directory in the Linux hierarchy, and contains all other files and directories. It is denoted by `/`. Folders in the root directory cannot be modified without **root privileges** (see the **sudo** command for more details). NOTE: do not try and modify any of these folders unless you know what you're doing.
- Your **home directory** is the default location for all your personal files, and is located at `/home/<your-username>/`. Your home directory is the default working directory when you open the terminal for the first time.

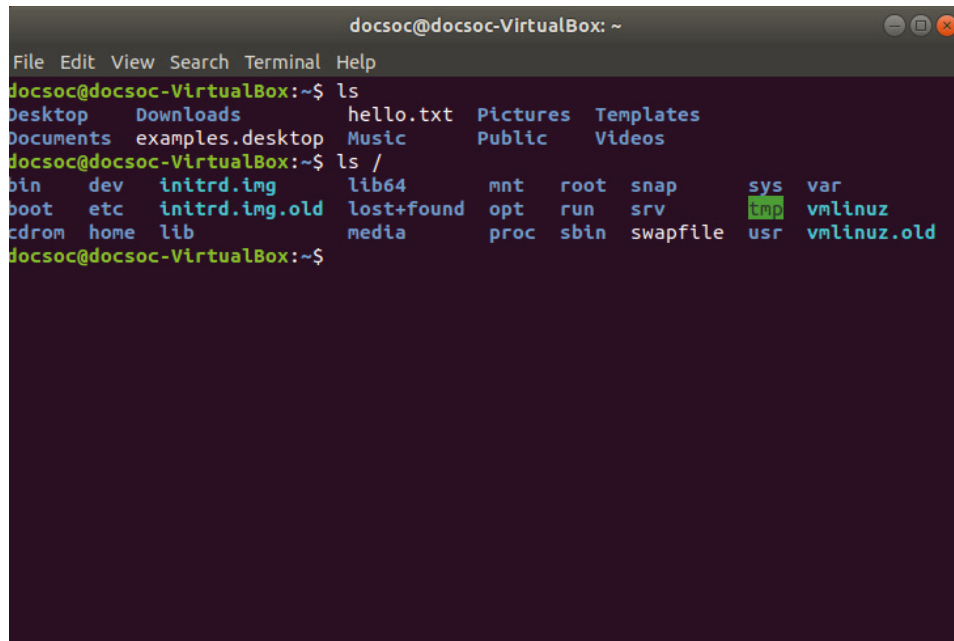
The `ls` command

The first command is the easiest - it simply lists all the files and directories in your working directory! Type in `ls` to see all the files in your current directory! (Remember by default, each Terminal window opens into your home directory).



```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop    Downloads  Music      Public     Videos  
Documents  examples.desktop  Pictures   Templates  
docsoc@docsoc-VirtualBox:~$
```

If we type `ls <directory>`, we can list all the files in that directory. As an example, from your home directory type `ls /`. You should be able to see all the files in the root directory!



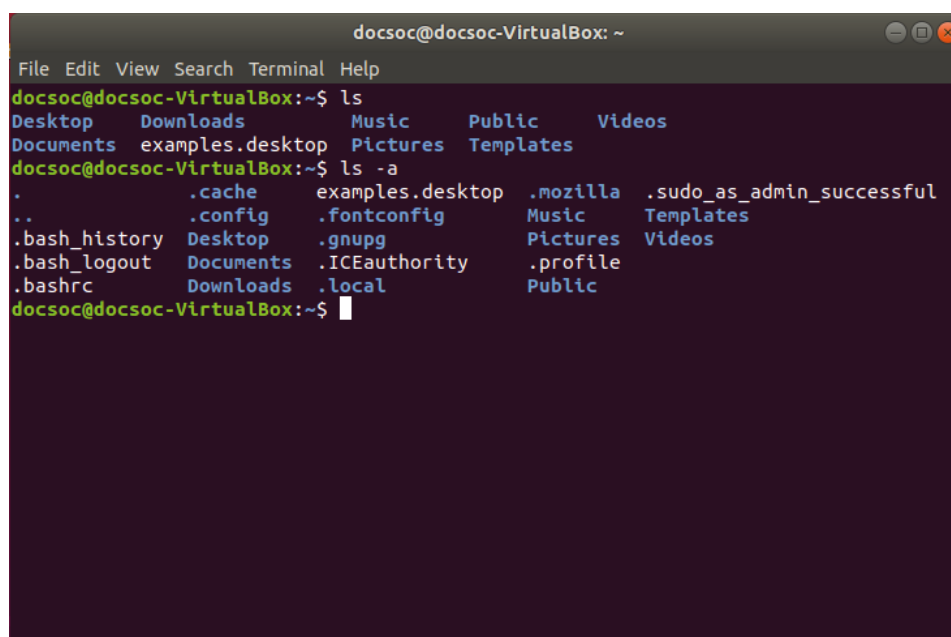
```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop    Downloads  hello.txt  Pictures  Templates  
Documents  examples.desktop  Music      Public    Videos  
docsoc@docsoc-VirtualBox:~$ ls /  
bin      dev  initrd.img  lib64      mnt  root  snap  sys  var  
boot    etc  initrd.img.old  lost+found  opt  run  srv   tmp  vmlinuz  
cdrom   home lib          media      proc  sbin swapfile  usr  vmlinuz.old  
docsoc@docsoc-VirtualBox:~$
```

You don't need to worry about the contents of the root directory just yet - this is just an example

From your home directory, you can type things like **ls Desktop** to view the contents of your Desktop, and so on. Try it out!

We can add several optional flags to **ls** to change its output.

Adding **-a** to the command will print all files and directories, **including any hidden files**, in the working directory. (For more info on hidden files see the Hidden Files section later on). See that **.** (a single dot) and **..** (double dots) are also printed; these are shortcuts that refer to the working directory and its parent directory respectively.

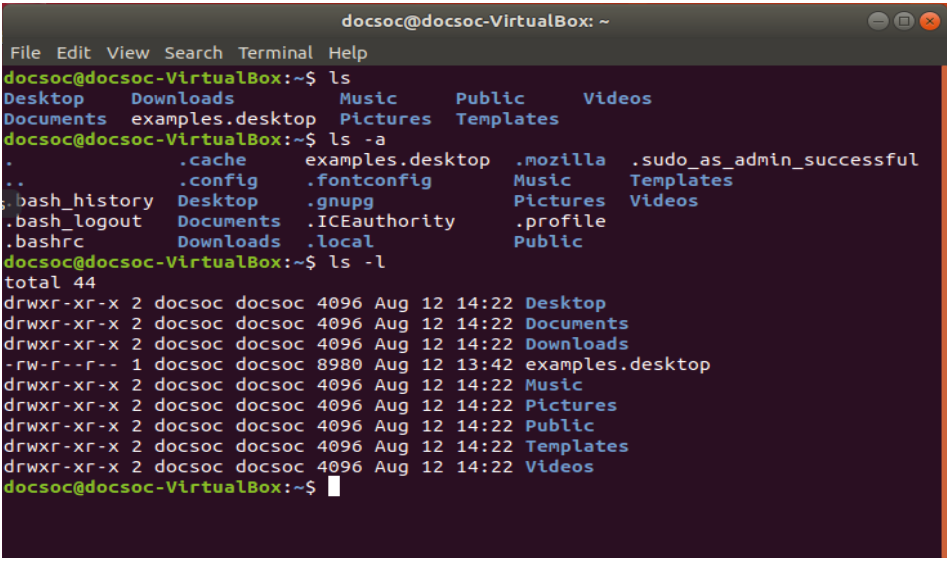


```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop    Downloads  Music      Public    Videos  
Documents  examples.desktop  Pictures  Templates  
docsoc@docsoc-VirtualBox:~$ ls -a  
.  
..  
.bash_history  Desktop  .gnupg      Music      Templates  
.bash_logout  Documents .ICEauthority .profile   Videos  
.bashrc       Downloads .local      Public  
docsoc@docsoc-VirtualBox:~$
```

Adding **-l** prints the list of files/directories in so-called long form. This prints information such as:

- file permissions
- owner name
- file size (in bytes)
- date last modified

We explain how to read file permissions (e.g. **drwxr-xr-x**) in the File Permissions section later on.



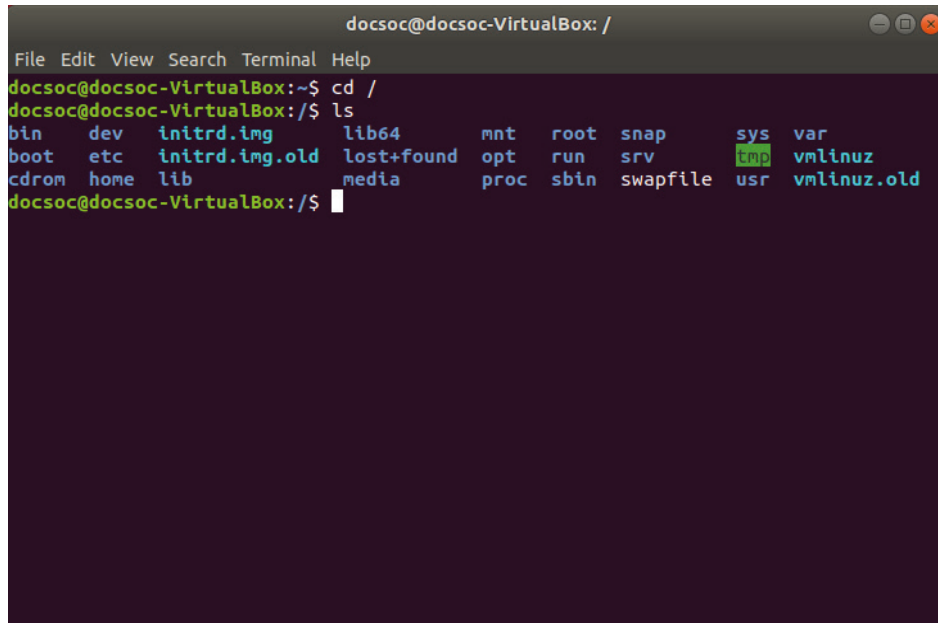
```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads Music Public Videos  
Documents examples.desktop Pictures Templates  
docsoc@docsoc-VirtualBox:~$ ls -a  
.  
..  
.cache examples.desktop .mozilla .sudo_as_admin_successful  
.config .fontconfig Music Templates  
.bash_history Desktop .gnupg Pictures Videos  
.bash_logout Documents .ICEauthority .profile  
.bashrc Downloads .local Public  
docsoc@docsoc-VirtualBox:~$ ls -l  
total 44  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Desktop  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Documents  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Downloads  
-rw-r--r-- 1 docsoc docsoc 8980 Aug 12 13:42 examples.desktop  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Music  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Pictures  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Public  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Templates  
drwxr-xr-x 2 docsoc docsoc 4096 Aug 12 14:22 Videos  
docsoc@docsoc-VirtualBox:~$
```

TIP: the up and down arrow keys can be used to scroll through previously used commands. This can save a lot of time when you eventually start using longer commands.

The **cd** command

The **cd** command is used to change your current working directory. Type **cd** **<directory>** to switch to that directory.

For example, **cd /** changes our current working directory to the root directory. Now typing **ls** has the same effect as typing **ls /** from any other directory.



```
docsoc@docsoc-VirtualBox: /
File Edit View Search Terminal Help
docsoc@docsoc-VirtualBox:~$ cd /
docsoc@docsoc-VirtualBox:/$ ls
bin    dev    initrd.img    lib64    mnt    root    snap    sys    var
boot   etc    initrd.img.old  lost+found  opt    run    srv    tmp    vmlinuz
cdrom  home  lib           media    proc   sbin   swapfile  usr    vmlinuz.old
docsoc@docsoc-VirtualBox:/$
```

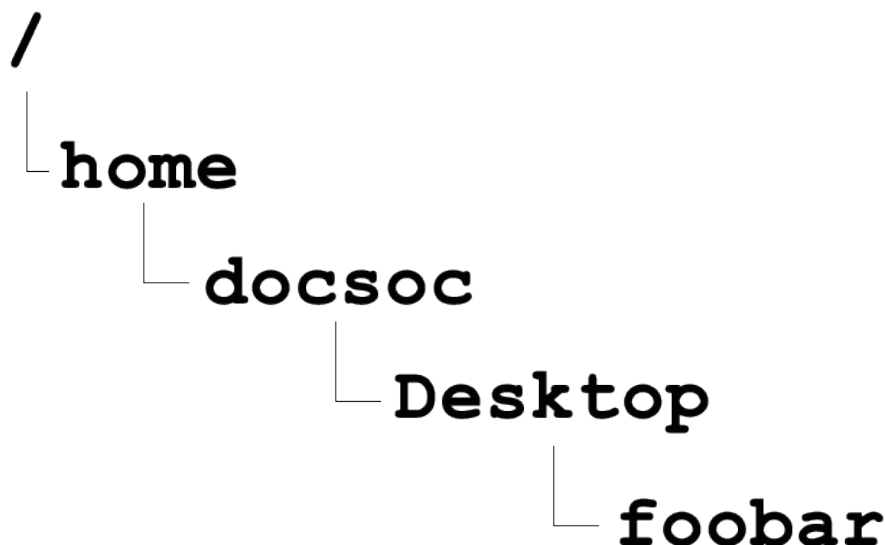
From your home directory, you can type things like **cd Desktop** to switch to your Desktop, and so on and so forth. That's it! You've learnt two of the most important Linux terminal commands.

Relative and absolute paths

In the terminal, a directory path (or file path) can be specified in two different ways:

An **absolute path** is given with reference to the root directory, and always begins with a **/**. For example, **/usr/local/bin/** is an absolute path. In contrast, a **relative path** is given with reference to the current working directory, and doesn't begin with a **/**.

For example, say we have a folder **foobar** located in our **Desktop**, and our user is called **docsoc**.



Our example directory structure

Let's run through some scenarios:

- The absolute path to **foobar** is **/home/docsoc/Desktop/foobar**.
- If our current working directory is **/**, then the relative path to **foobar** would be **home/docsoc/Desktop/foobar**.
- If our current working directory is **/home/**, then the relative path to **foobar** would be **docsoc/Desktop/foobar**.
- If our current working directory is **/home/docsoc/** (remember this is the default working directory when you open terminal for the first time) then the relative path to **foobar** would be **Desktop/foobar**.
- If our current working directory is **/home/docsoc/Desktop/**, then the relative path to **foobar** would simply be **foobar**.

Try experimenting more with the **ls** and **cd** commands, this time using both absolute and relative paths to specify directories.

Terminal shortcuts

Terminal has some shortcuts that can be used instead of typing out long directory names:

- **~/** refers to your own home directory
- **.** (a single dot) refers to the current working directory
- **..** (double dots) refers to its parent directory (i.e. one level higher up in the hierarchy)

These can be used with the **ls** and **cd** commands, and in general with any terminal command where a directory name is required.

Hidden files

Files or directories that begin with **.** are hidden, and by default aren't viewable to an ordinary user. Most hidden files and folders across the Linux filesystem are configuration files, e.g. the bash config file **.bashrc** and the **.git** folders inside git repositories. Hidden files can be viewed by using **ls -a**.

File permissions

ls -l prints file permissions in a single 10-character string, for example **drwxr-xrw-**. Here we'll decipher what this means.

The first character will be **d** if the object in question is a directory, otherwise it will be **-** for a file.

The next 9 characters are all either **r**, **w** or **x**:

- **r** means the file/directory is readable
- **w** means the file/directory is writable
- **x** means the file/directory is executable
- if any of these are not present, a **-** will be present instead.

The first 3 characters describe permissions for the **owner** of the file.

The next 3 characters describe permissions for the **owner's user group**.

The last 3 characters describe permissions for **everyone else**.

For example, let's decipher **drwxr-xrw-**.

The first character is **d**, meaning this is a directory.

The next group of 3 characters is **rw****x**. This means the owner has read, write and execute permissions.

The next group of 3 characters is **r****-****x**. This means everyone in the owner's user group can read and execute, but not write to the file.

The next group of 3 characters is **rw****-**. This means everyone else can read and write, but not execute the file.

For information on how to change a file's permissions, see the **chmod** command.

More commands

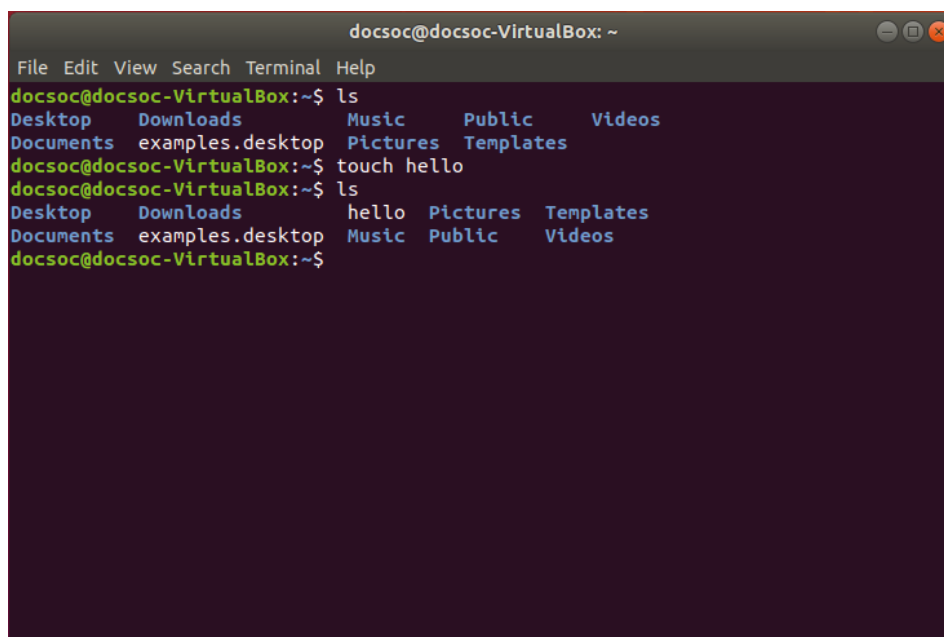
You've already learnt about **ls** and **cd**. Now, we'll teach you some more useful (nay, essential) Linux commands.

Don't worry if the following list of commands seems very long at first. After lots of experience with using the Linux terminal you'll gradually start to memorise each command.

Remember, whenever a command needs a file/directory path you can give it as either an absolute path or relative to your working directory.

touch

This is a really easy command. **touch <file>** simply creates an empty file at the specified location! Here it is demonstrated:

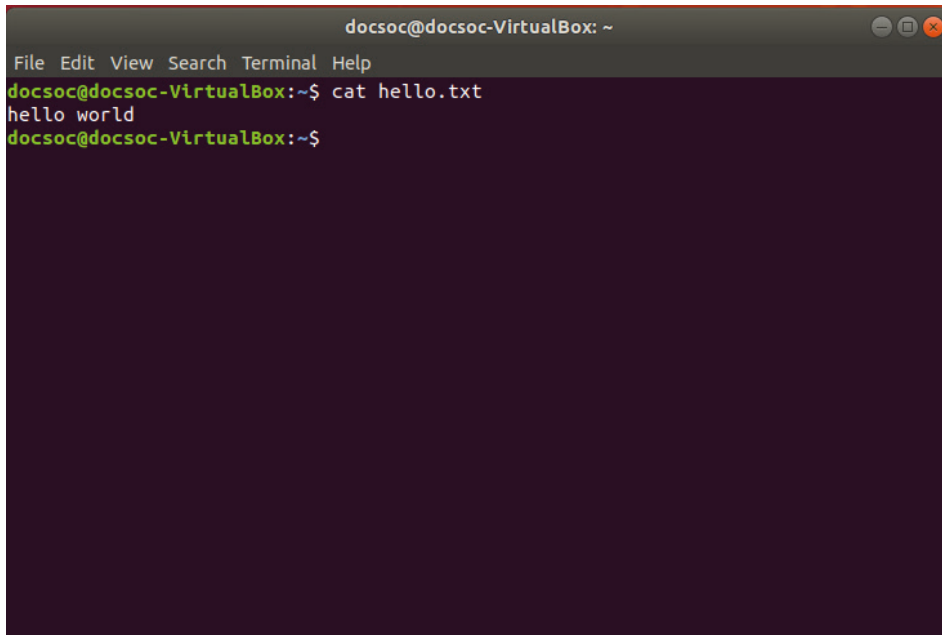


```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads Music Public Videos  
Documents examples.desktop Pictures Templates  
docsoc@docsoc-VirtualBox:~$ touch hello  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads hello Pictures Templates  
Documents examples.desktop Music Public Videos  
docsoc@docsoc-VirtualBox:~$
```

NOTE: files don't have to end with an extension like **.txt**. They can be called whatever you want! Files only have extensions as hints so that users and programs know what to do with that type of file.

cat

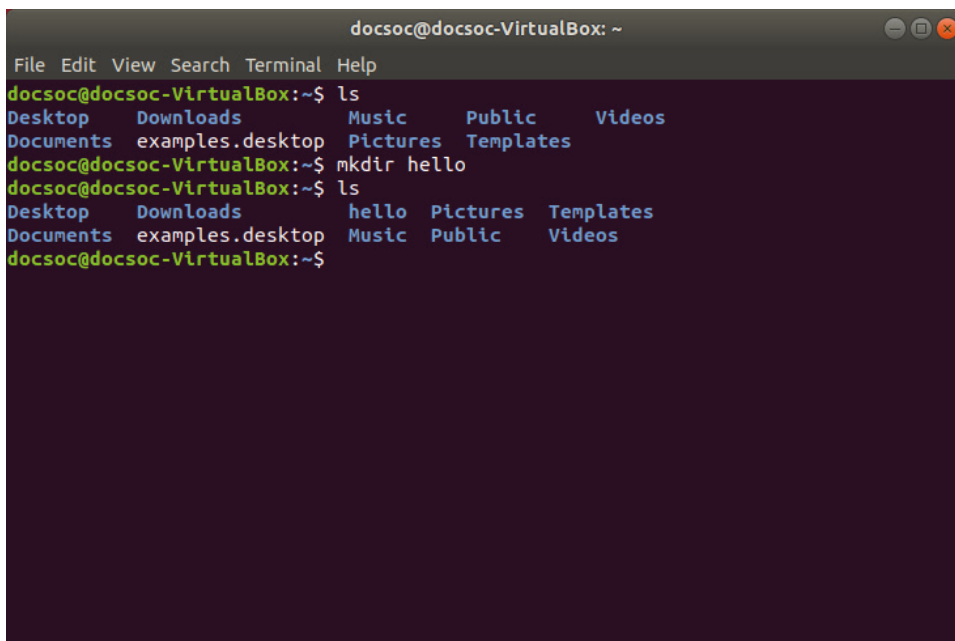
cat <file> prints the contents of that file to your screen. In the following example we've already created a file called **hello.txt** in our home folder, which contains the text **hello world**.

A terminal window titled 'docsoc@docsoc-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'docsoc@docsoc-VirtualBox:~\$'. The command 'cat hello.txt' has been entered, and the output 'hello world' is displayed on the next line. The prompt is now 'docsoc@docsoc-VirtualBox:~\$' again.

```
docsoc@docsoc-VirtualBox: ~
File Edit View Search Terminal Help
docsoc@docsoc-VirtualBox:~$ cat hello.txt
hello world
docsoc@docsoc-VirtualBox:~$
```

mkdir

mkdir <directory> creates an empty directory at the specified path.

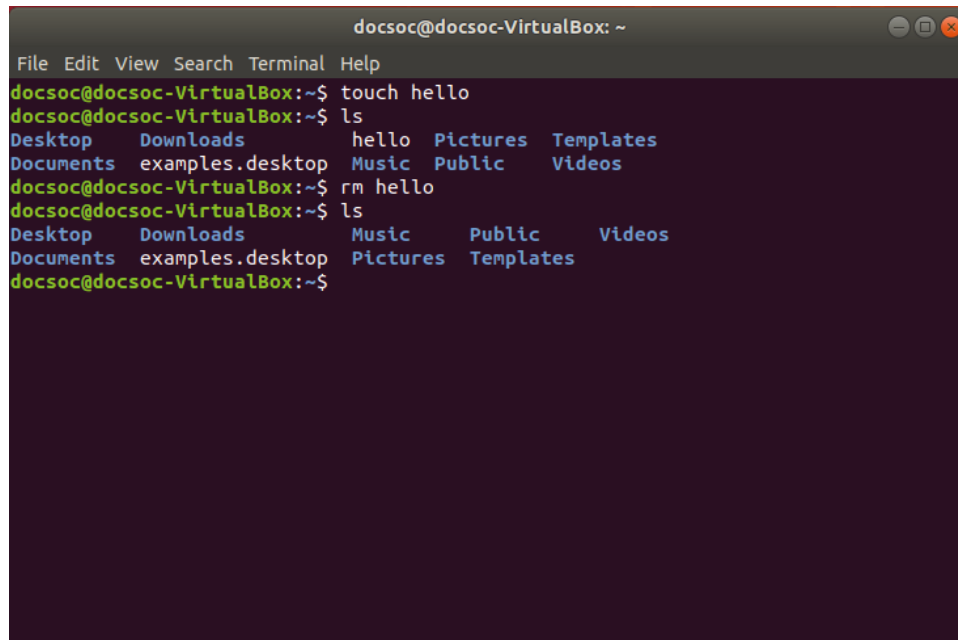
A terminal window titled 'docsoc@docsoc-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'docsoc@docsoc-VirtualBox:~\$'. The command 'ls' is entered, showing a list of directories: Desktop, Downloads, Music, Public, Videos, Documents, examples.desktop, Pictures, and Templates. Then, the command 'mkdir hello' is entered. Finally, 'ls' is entered again, and the output now includes 'hello' in the list of directories. The prompt is 'docsoc@docsoc-VirtualBox:~\$' throughout.

```
docsoc@docsoc-VirtualBox: ~
File Edit View Search Terminal Help
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  Music     Public    Videos
Documents examples.desktop Pictures Templates
docsoc@docsoc-VirtualBox:~$ mkdir hello
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  hello     Pictures  Templates
Documents examples.desktop Music     Public    Videos
docsoc@docsoc-VirtualBox:~$
```

NOTE: files appear in white but directories appear in blue! The terminal colours may vary depending on what flavour of Linux you are running.

rm

rm <file> removes that file **permanently** from your computer. Terminal commands don't use the Recycle Bin! Be very careful when using **rm** to remove files. In the example below we use **touch** to create an empty file and **rm** to remove it.

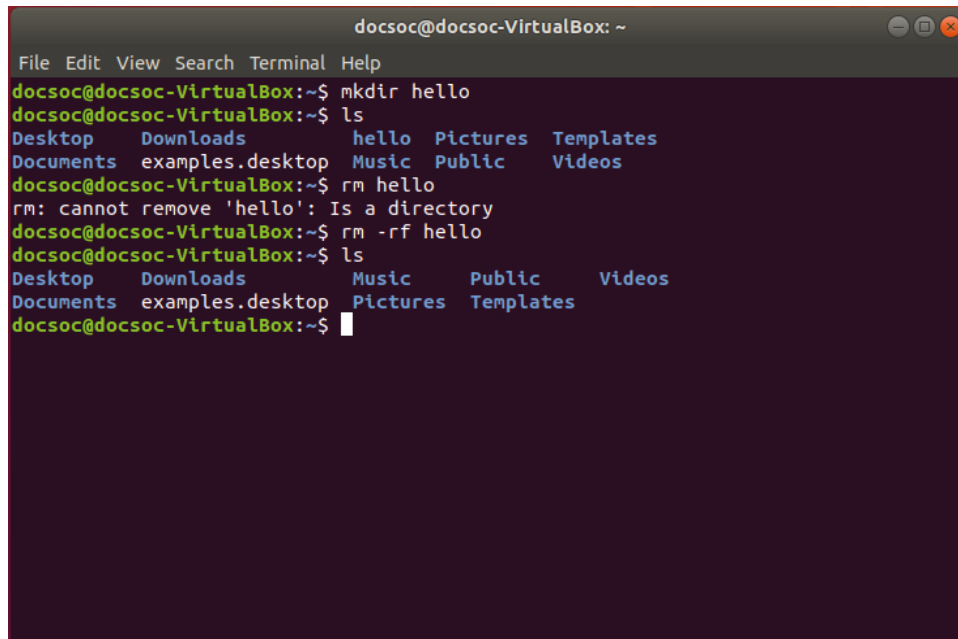
A terminal window titled 'docsoc@docsoc-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
docsoc@docsoc-VirtualBox:~$ touch hello
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  hello  Pictures  Templates
Documents examples.desktop Music  Public  Videos
docsoc@docsoc-VirtualBox:~$ rm hello
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  Music  Public  Videos
Documents examples.desktop Pictures  Templates
docsoc@docsoc-VirtualBox:~$
```

There's one catch: if you try to use **rm** on a directory, an error is printed! To remove a directory (and everything inside it) we must add the **-r** and **-f** flags.

- **-r** tells the computer to remove the directory recursively (i.e. to remove all subdirectories, subsubdirectories, etc.)
- **-f** tells the computer to erase all files/directories without asking for confirmation, no matter what the file permissions are

We can stick these flags together into one string, so typing **rm -rf <directory>** will remove that directory from the computer. **Remember this is a permanent action, there is no Recycle Bin!** In this example we create a directory with **mkdir** and remove it with **rm -rf**.



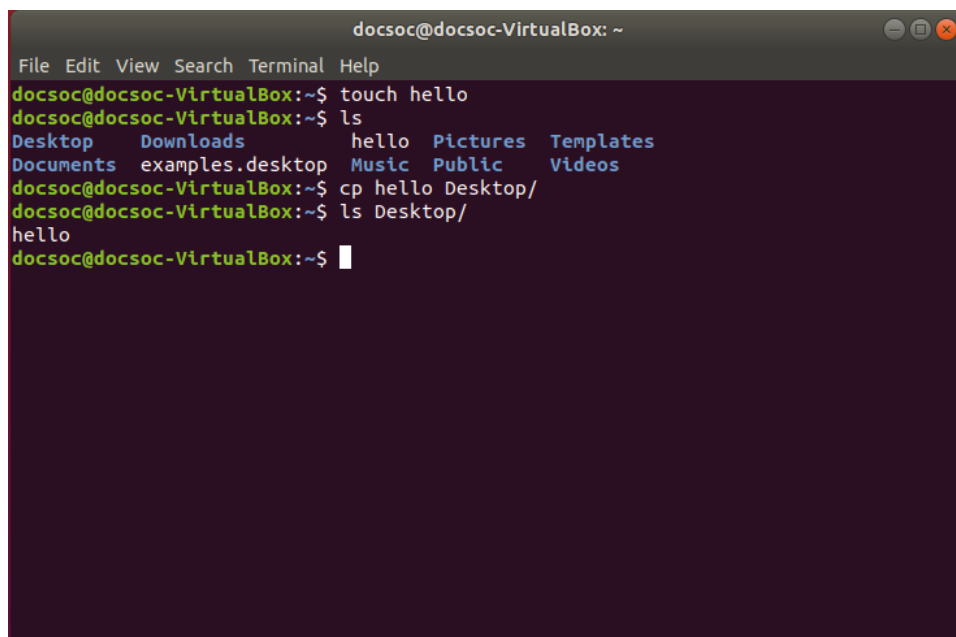
```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ mkdir hello  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads hello Pictures Templates  
Documents examples.desktop Music Public Videos  
docsoc@docsoc-VirtualBox:~$ rm hello  
rm: cannot remove 'hello': Is a directory  
docsoc@docsoc-VirtualBox:~$ rm -rf hello  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads Music Public Videos  
Documents examples.desktop Pictures Templates  
docsoc@docsoc-VirtualBox:~$
```

`rm` doesn't work, but `rm -rf` does

An added bonus: you can attach any number of files/directories to the end of your `rm` command, and `rm` will remove all of them! e.g. `rm file1 file2 file3` will remove the files `file1`, `file2` and `file3` from your working directory.

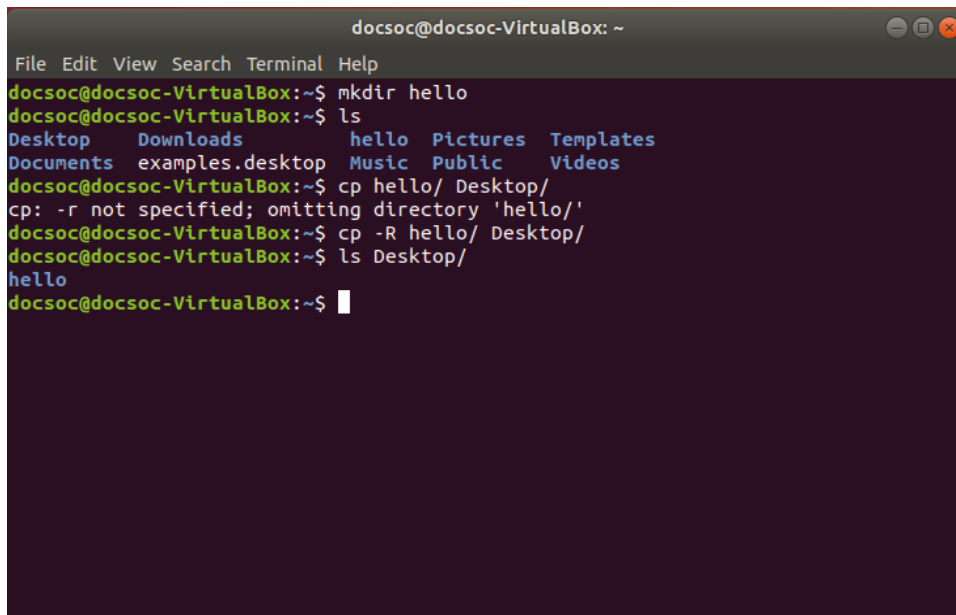
cp

`cp <file> <directory>` copies the file into the specified directory. The original is left intact. Here it is in action:



```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ touch hello  
docsoc@docsoc-VirtualBox:~$ ls  
Desktop Downloads hello Pictures Templates  
Documents examples.desktop Music Public Videos  
docsoc@docsoc-VirtualBox:~$ cp hello Desktop/  
docsoc@docsoc-VirtualBox:~$ ls Desktop/  
hello  
docsoc@docsoc-VirtualBox:~$
```

Similarly to **rm**, **cp** doesn't immediately work on copying directories. We must add the **-R** flag (R = recursive) to copy entire directories. Therefore **cp -R <directory1> <directory2>** copies the whole of directory1 into directory2.

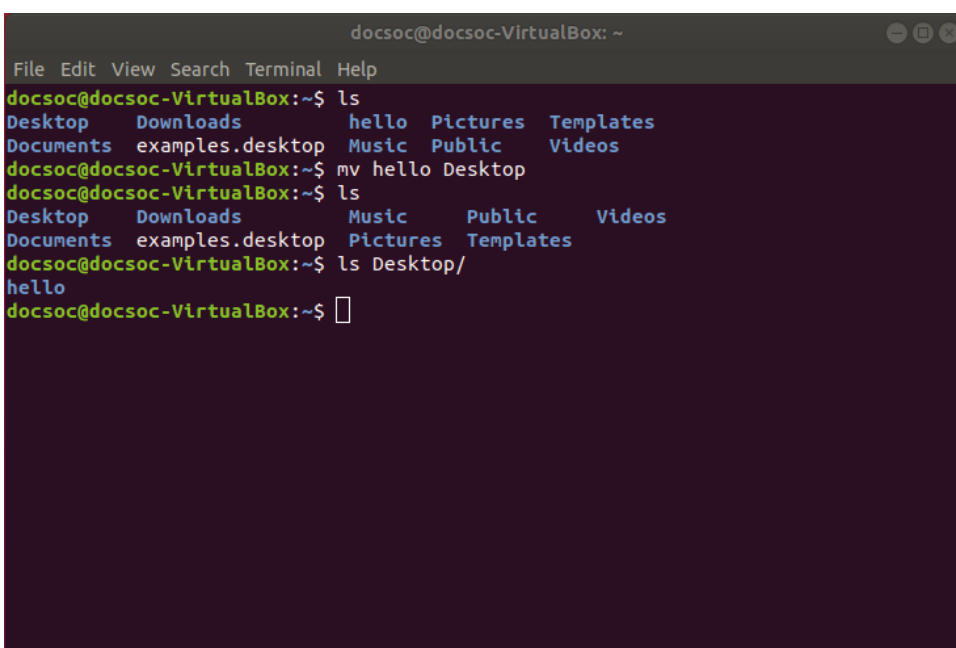
A terminal window titled 'docsoc@docsoc-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
docsoc@docsoc-VirtualBox:~$ mkdir hello
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  hello  Pictures  Templates
Documents examples.desktop Music  Public  Videos
docsoc@docsoc-VirtualBox:~$ cp hello/ Desktop/
cp: -r not specified; omitting directory 'hello/'
docsoc@docsoc-VirtualBox:~$ cp -R hello/ Desktop/
docsoc@docsoc-VirtualBox:~$ ls Desktop/
hello
docsoc@docsoc-VirtualBox:~$
```

cp doesn't work, but **cp -R** does

mv

mv <file/directory> <directory> moves the file/directory given by the first argument into the directory given by the second argument. Unlike **cp**, **mv** doesn't need a special flag for dealing with directories; it can handle them in the same way it handles files. In the example below we move a directory from our home folder to **Desktop**.

A terminal window titled 'docsoc@docsoc-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  hello  Pictures  Templates
Documents examples.desktop Music  Public  Videos
docsoc@docsoc-VirtualBox:~$ mv hello Desktop
docsoc@docsoc-VirtualBox:~$ ls
Desktop  Downloads  Music  Public  Videos
Documents examples.desktop Pictures  Templates
docsoc@docsoc-VirtualBox:~$ ls Desktop/
hello
docsoc@docsoc-VirtualBox:~$
```

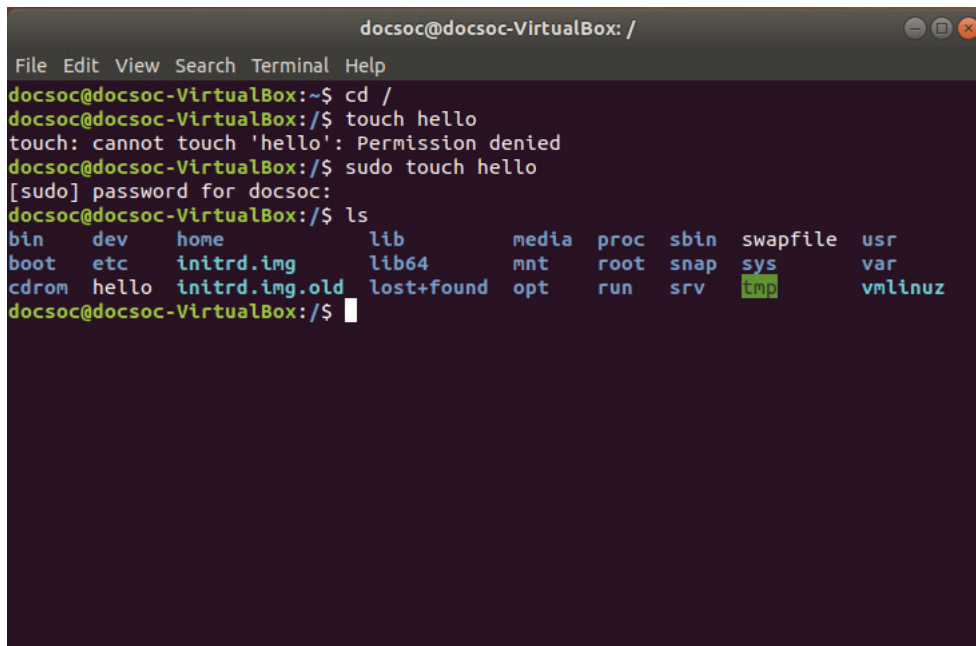

sudo

sudo is probably one of the most important commands in the Linux terminal, however we need to give you some background information before explaining what it does.

All Linux computers have a secret user called **root**, which is allowed to read, write and execute every single file on the computer. However, there are many critical system files littered around which could cause serious damage if modified accidentally. Therefore, your personal user (and all other normal users) are only allowed to read and write to certain directories on the computer, and if you try to modify a sensitive directory (like the directories in **/**) you're presented with an error.

The way around this is to prepend the word **sudo** to your command. This executes the command as the **root** user, allowing you to do pretty much whatever you want, and Terminal will ask for your password before executing the command. However you **MUST** make sure what you're doing is safe, otherwise you risk permanently breaking your Linux system. Be warned!

In the following example, we write an empty file to the root directory.

A terminal window titled 'docsoc@docsoc-VirtualBox: /' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
docsoc@docsoc-VirtualBox:~$ cd /
docsoc@docsoc-VirtualBox:/$ touch hello
touch: cannot touch 'hello': Permission denied
docsoc@docsoc-VirtualBox:/$ sudo touch hello
[sudo] password for docsoc:
docsoc@docsoc-VirtualBox:/$ ls
bin      dev      home     lib       media    proc     sbin     swapfile  usr
boot     etc      initrd.img lib64      mnt      root     snap     sys       var
cdrom    hello   initrd.img.old lost+found opt       run      srv       tmp       vmlinuz
docsoc@docsoc-VirtualBox:/$
```

NOTE: in the terminal, whenever you are asked to input your password, the characters you type aren't printed to the screen (although they are registered by the computer). That's why the bit after **[sudo] password for docsoc:** appears blank, even though we did actually enter our password.

A small note: students actually aren't allowed to use **sudo** on Huxley lab computers! Presumably the Department has had enough of students messing around with critical system files...if there's something which you think you need **sudo** for, contact the Computing Support Group (CSG).

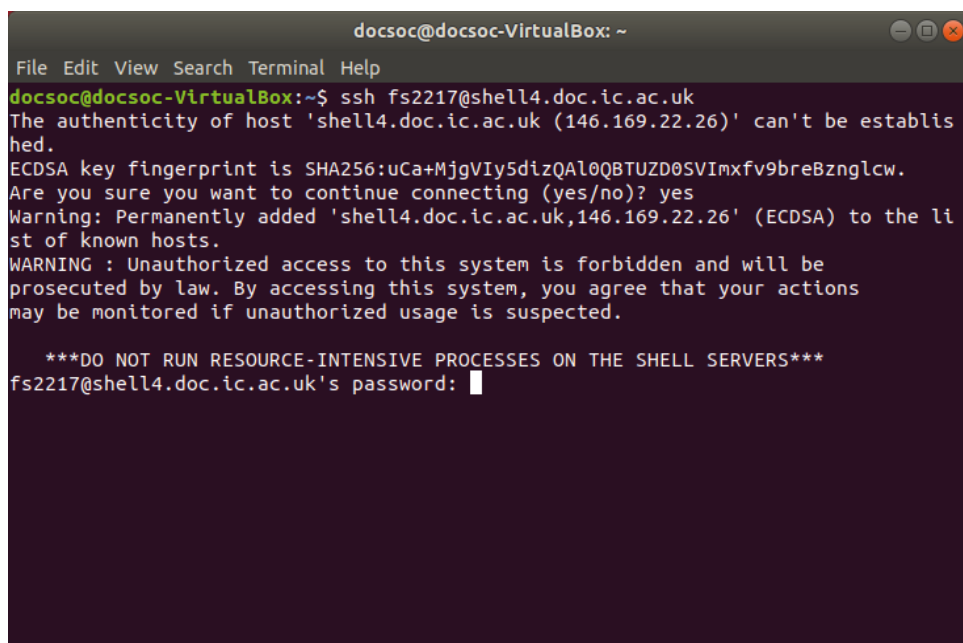
ssh (and how to remotely log in to lab computers)

ssh is a very useful Linux tool, since it allows you to access any other public Linux computer in the world. The only prerequisites are you need to know the hostname (or IP address) of the computer, and also the username and password for an account on that computer. The format for the **ssh** command is **ssh <username>@<computer>**, where **<computer>** can be either the hostname or the IP address of that computer. After executing this command, you will be prompted for the user's password before being able to remotely access the computer's terminal.

The Department of Computing provides a way for you to **ssh** into your personal account on the Huxley lab computers. There are 5 Linux servers which can be accessed via **ssh**, named as follows:

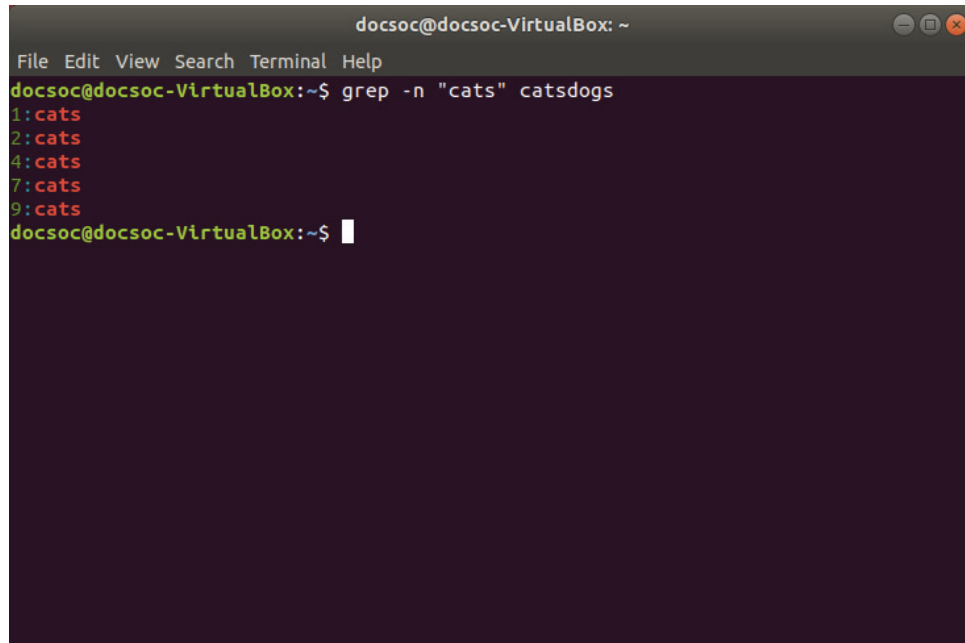
- **shell11.doc.ic.ac.uk**
- **shell12.doc.ic.ac.uk**
- **shell13.doc.ic.ac.uk**
- **shell14.doc.ic.ac.uk**
- **shell15.doc.ic.ac.uk**

Logging in to any one of them with your Imperial College credentials will allow you to access your personal account. (When the terminal asks **Are you sure you want to continue connecting?**, answer **yes**.)

A screenshot of a terminal window titled 'docsoc@docsoc-VirtualBox: ~'. The terminal shows the command 'ssh fs2217@shell4.doc.ic.ac.uk' being executed. It displays the host's fingerprint, asks for confirmation to continue connecting (which is answered 'yes'), and shows a warning about unauthorized access. Finally, it prompts for the password of user 'fs2217' on 'shell4.doc.ic.ac.uk'.

```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
docsoc@docsoc-VirtualBox:~$ ssh fs2217@shell4.doc.ic.ac.uk  
The authenticity of host 'shell4.doc.ic.ac.uk (146.169.22.26)' can't be established.  
ECDSA key fingerprint is SHA256:uCa+MjgVIy5dizQA10QBtUZD0SVImxfv9breBznglcw.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'shell4.doc.ic.ac.uk,146.169.22.26' (ECDSA) to the list of known hosts.  
WARNING : Unauthorized access to this system is forbidden and will be prosecuted by law. By accessing this system, you agree that your actions may be monitored if unauthorized usage is suspected.  
  
***DO NOT RUN RESOURCE-INTENSIVE PROCESSES ON THE SHELL SERVERS***  
fs2217@shell4.doc.ic.ac.uk's password: █
```

Here we're logging in as user **fs2217**

A screenshot of a terminal window titled 'docsoc@docsoc-VirtualBox: ~'. The terminal shows the command 'grep -n "cats" catsdogs' being executed. The output is as follows:
1: cats
2: cats
4: cats
7: cats
9: cats
The prompt 'docsoc@docsoc-VirtualBox:~\$' is visible at the bottom of the terminal.

grep can also be used to search for a phrase within a directory, if you add the **-r** flag. The format is **grep -rn "string" <directory>**.

chmod

chmod allows you to change the permissions for a file or directory. The format is as follows: **chmod <mode> <file/directory>**.

The **<mode>** argument must be given as a 3-digit number, with each digit in the range 0-7. The first digit sets permissions for your own user, the second digit sets permissions for your user group and the third digit sets permissions for everyone.

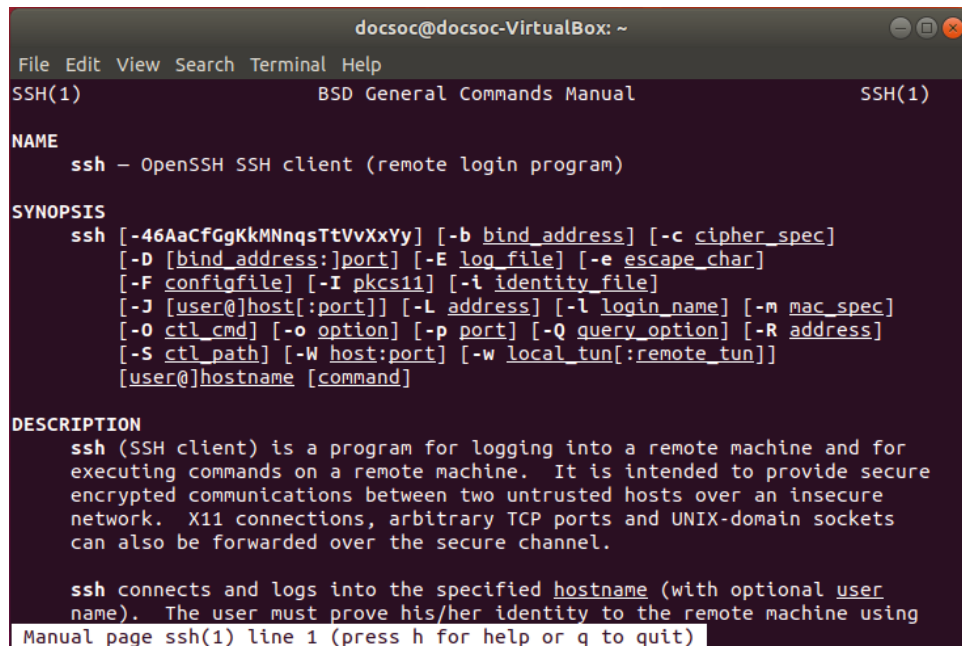
Each digit sets permissions as follows:

Digit	Shortcode	Permissions
7	rwX	read, write and execute
6	rw-	read and write
5	r-X	read and execute
4	r--	only read
3	-wX	write and execute
2	-w-	only write
1	--X	only execute
0	---	none

man

We've saved the most useful command until last. The **man** command gives you help on how to use any other Linux command! Type **man <commandname>** to see the built-in Linux manual entry for that command. All the information on commands we have described in this booklet can also be found using **man**. This can be especially useful for first year Lexis Tests, since you aren't allowed Internet access during the test, so this is your point of reference for any Linux commands.

For example, here's the beginning of the manual entry for **ssh**:



```
docsoc@docsoc-VirtualBox: ~  
File Edit View Search Terminal Help  
SSH(1) BSD General Commands Manual SSH(1)  
  
NAME  
    ssh - OpenSSH SSH client (remote login program)  
  
SYNOPSIS  
    ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]  
        [-D [bind_address:]port] [-E log_file] [-e escape_char]  
        [-F configfile] [-I pkcs11] [-i identity_file]  
        [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]  
        [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]  
        [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]  
        [user@]hostname [command]  
  
DESCRIPTION  
    ssh (SSH client) is a program for logging into a remote machine and for  
    executing commands on a remote machine. It is intended to provide secure  
    encrypted communications between two untrusted hosts over an insecure  
    network. X11 connections, arbitrary TCP ports and UNIX-domain sockets  
    can also be forwarded over the secure channel.  
  
    ssh connects and logs into the specified hostname (with optional user  
    name). The user must prove his/her identity to the remote machine using  
    Manual page ssh(1) line 1 (press h for help or q to quit)
```

You can scroll through the manual entry using the arrow keys.

Advanced command line concepts

These concepts are a little more complicated than your standard commands, so make sure you've experimented enough with commands you already know before moving on to these.

Outputting to a file

So far we've looked at a few commands which print output to the terminal screen (e.g. **cat**, **grep**). In your forays through the Linux world you will encounter many more commands which also print useful information to the screen. But wouldn't it be nice to be able to send all of this output to a file instead, for example if the output is very long?

If you append **> <file>** (that's '>', followed by the name of a file) to the end of your command, output will be written to **file** instead of being printed to the screen, so you have a permanent copy of the command's output. Nifty!

For example, **cat file1.txt > file2.txt** will read from **file1.txt** and write the contents to **file2.txt**.

Note that this will overwrite the destination file. If you want to append to the end of a file, use **>>**.

Streams and piping

In general, every terminal command takes in some text input and eventually spits out some text output. These pieces of data are sent between the command and the Terminal using virtual input and output channels in your computer, called **streams** (also known as standard streams). There is a single input stream, called **stdin**, and there are two output streams, called **stdout** and **stderr**.

Stream	Purpose
stdin	Handling input for a command
stdout	Handling output for a command
stderr	Handling any error messages printed by a command

When executing a program as normal, all streams are connected to the Terminal. This means that any data to be input into a command must be typed into the Terminal, and the output of any command (along with any error messages) is printed to the Terminal.

A **pipeline** allows you to connect the **stdout** of one command to the **stdin** of another (called “piping”). In other words, you can tell the computer to run a second command using the output of the first one. This is done using the format **<first-command> | <second-command>** (‘|’ is the ‘pipe’ symbol).

For example, let's say we want to find all the files/directories in our working directory that contain the string “cat”. We know a command which can list all the filenames in our working directory:

```
ls
```

And we know a command that can search for the string “cat” in a bunch of text:

```
grep -n "cat" <file>
```

We can pipe the output of **ls** into the input of **grep** like so:

```
ls | grep -n "cat"
```

This will print the names of all files/directories containing the string “cat”. Note that the **<file>** argument is not needed for **grep** in this case; the Terminal is smart enough to realise input will come from the piped command instead.

This doesn't just end with two commands, you can chain together as many commands as you want. Feel free to experiment!

Package managers

Every Linux distro comes with its own package management system, which is a program that allows you to download other programs off the Internet through the command line. If you need to install a program on Linux, for example OpenOffice, vim or git, 99% of the time it will be available on the default package manager. All software is verified securely before being put on any package management systems, which is one of the reasons why Linux is such a secure operating system in general.

The default package manager for most Debian-based distros (Ubuntu included) is **apt-get**. For Fedora-based distros (for example CentOS) the default package manager is **yum**.

Installing a program with either of these is easy. Type:

```
sudo apt-get install <package>
```

or

```
sudo yum install <package>
```

to install the necessary package.

For more commands, including how to update and delete packages, see **man apt-get** or **man yum**.

You can install other package managers on your system, and often there are package managers for modules written in specific languages. For example **npm** is a package manager for Javascript modules, and **pip** is a package manager for Python modules.

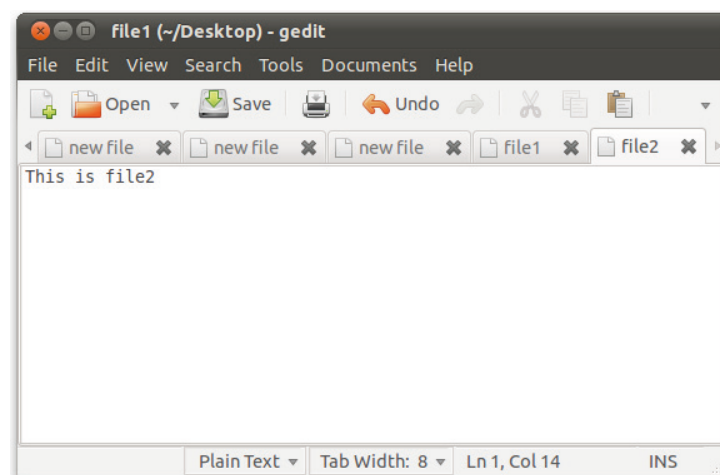
Text editors

Choosing the right text editor can be a difficult choice, one which has been the source of many a flame war on the Internet. Here we'll run through some popular text editor choices, so you can pick what feels best for you.

gedit

This is the bog-standard text editor that ships with Ubuntu. It works in almost exactly the way you would expect. There isn't too much to customise, and editing multiple files at the same time can be fairly slow. Any version control (e.g. with git) must be performed manually.

Customisable?	Version control integration?	Learning curve
No	No	Easy

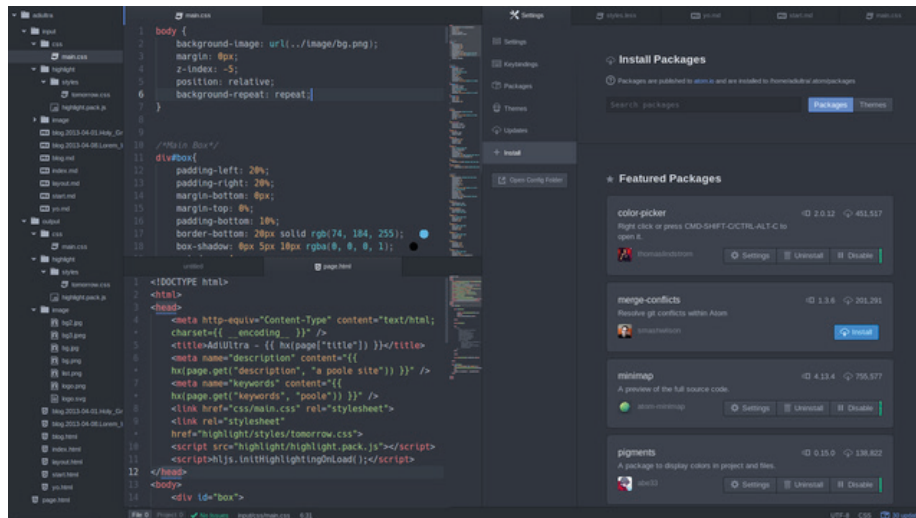


A typical gedit interface

Atom

This is a piece of software made by GitHub, advertised as being a “hackable text editor for the 21st century”. There are many plugins which you can install to speed up your workflow, and editing multiple files is a lot easier. Atom is incredibly customisable; you can control keyboard shortcuts, themes, plugins and even create a script to run when the app starts up. Version control with git is built in.

Customisable?	Version control integration?	Learning curve
Yes	Yes	Easy

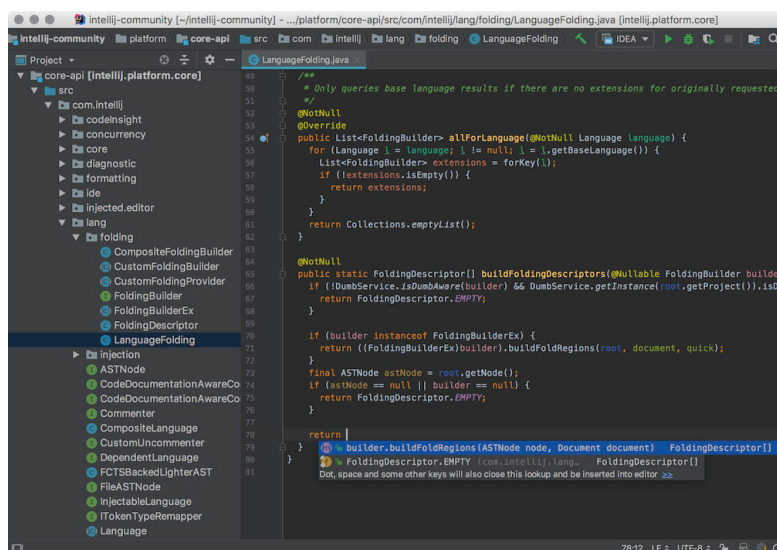


A typical Atom interface

Jetbrains IDEs

Jetbrains are a company that make a series of IDEs (integrated development environments) for various languages. Their flagship product, IntelliJ IDEA, is the recommended IDE to use when programming in Java in 1st/2nd term of 1st year, however they also have products such as PyCharm for Python programming and CLion for C programming (unfortunately there is no Haskell IDE!). You can customise the look of the IDE and can also configure many different plugins (e.g. vim keyboard shortcuts), and all products come with version control integration. Students can get all of Jetbrains' products for free.

Customisable?	Version control integration?	Learning curve
Yes	Yes	Medium



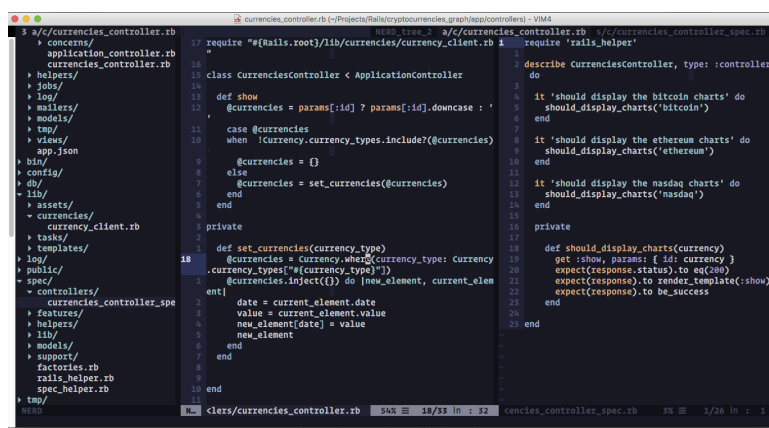
A typical IntelliJ IDEA interface

Vim

vim has a significantly steeper learning curve than other editors such as Atom, as it is what is called a 'modal' editor. However many rewards can be reaped in terms of workflow speed. There are a large number of built-in keyboard shortcuts, all of which can be remapped if you don't like them, and an almost uncountable number of plugins for vim are available online. Since vim is run within a command line interface it is very fast. Vim also has a very rich ecosystem of plugins which can be used for many purposes, for example to integrate with version control systems such as Git.

For a good tutorial to vim, check out [Vim Adventures](#).

Customisable?	Version control integration?	Learning curve
Yes, very	Yes, through plugins	Steep

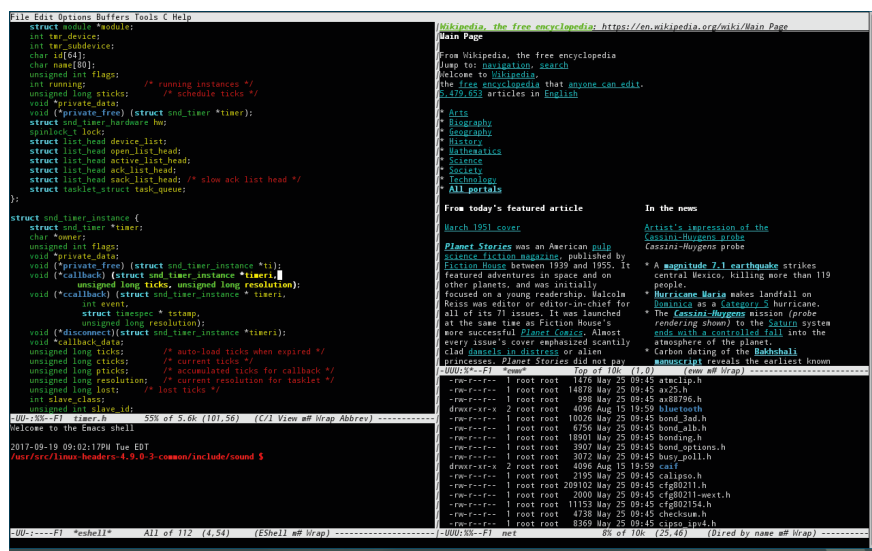


A typical (although heavily customised) vim interface

emacs

Similarly to vim, emacs is a text editor which is run from the command line, making it a very fast competitor. emacs has room for a huge number of themes and plugins, and includes several unique features such as a built-in terminal emulator which can be positioned around with normal panes. emacs also has its own unique set of keyboard shortcuts which are completely different to vim's. There is no built-in integration with version control.

Customisable?	Version control integration?	Learning curve
Yes, very	Yes, through plugins	Steep



A typical (although heavily customised) emacs interface

**We hope you learnt
something from this guide!**

The Linux command line is an incredibly useful tool, and we have barely skimmed the surface of its capabilities. If you have any suggestions for this guide, feel free to email us at docsoc@Imperial.ac.uk



Produced by Fawaz Shah

© DoCSoc 2018