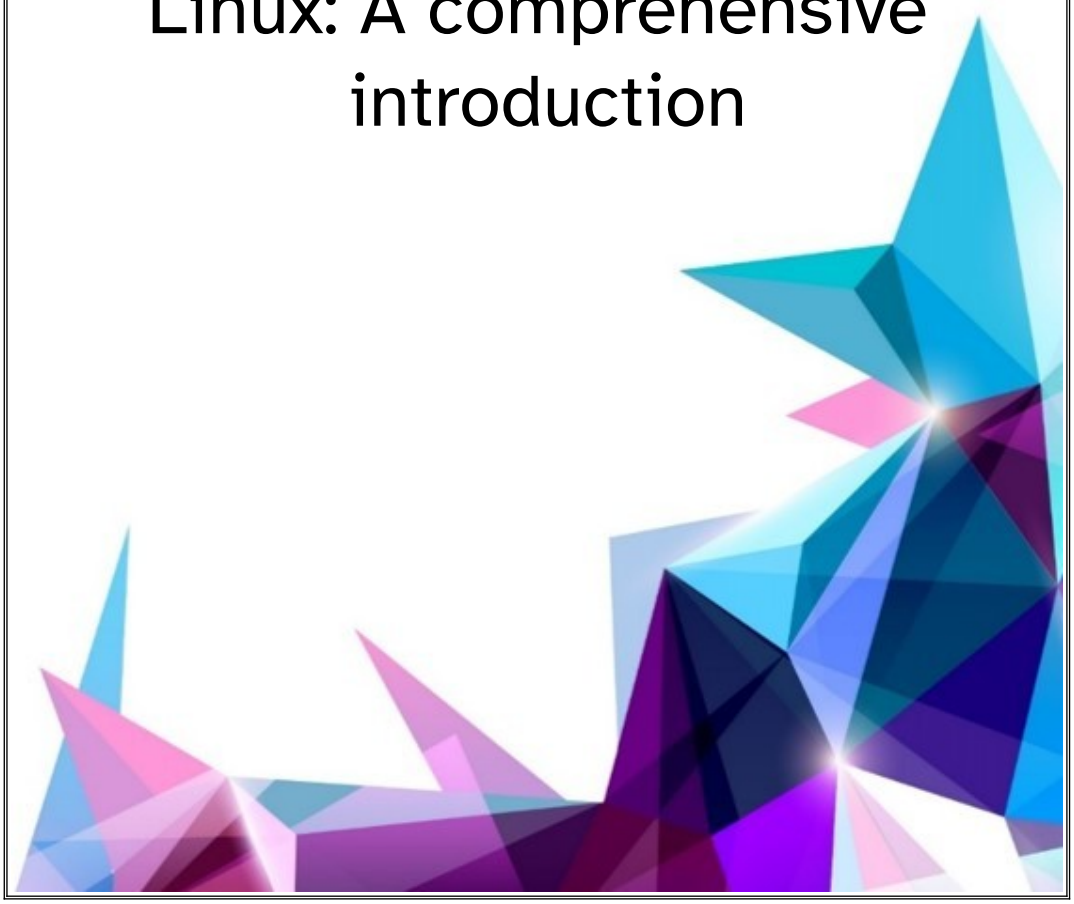
Linux: A comprehensive introduction



iT Centre Learning

iT services

UNIVERSITY OF OXFORD

The small print

Prerequisites

Time in the workshop is precious – it is an opportunity for you to interact with the workshop leader and other participants through questions and discussions and to share your experiences and concerns. To make the most of this time we sometimes ask you to carry out learning activities ahead of the workshop so that everyone comes into the class with the same basic knowledge. We keep this prior learning to a minimum and often make use of online videos. Online videos provided through LinkedIn Learning can be accessed free of charge by University members anytime, anywhere, through a browser or app.

Your course booking will tell you if any prior learning activity is required. If you don't have an environment where you can do this learning, you can come along to one of our LinkedIn Learning sessions. These are a quiet space where you can work through videos or other workshop resources.

If you arrive for a workshop without having done the prior learning, the workshop leader may suggest that you come back on another session.

Copyright

About the workshop designer

Graham Addis started his first technology role in 1978 and has gathered decades of practical experience in industry. He has always been passionate about passing on his knowledge and undertook his first formal teaching position as a Customer Training Specialist for Intel back in 1984.

Since that time his career has combined extensive real world experience with teaching and mentoring. In 2017 he joined the academic world at the University of Oxford and currently specialises in teaching spreadsheets, databases and programming.

Revision history

Version	Date	Author	Comments
5.1	Nov 2023	Graham Addis	Added slide for creating bootable USBs
5.0	Aug 2023	Graham Addis	Updated for Ubuntu 22.04.03 (Jammy Jellyfish)
	Oct 2019	Susan Hutchinson	Updated links for Remote Access section
	Apr 2019	Susan Hutchinson	Minor corrections TT 2019
	Jan 2019	Susan Hutchinson	Updated footer for HT 2019
	Oct 2018	Susan Hutchinson	Typos fixed immediately after MT2018 presentation
	Sep 2017	Susan Hutchinson	First section rewritten for Bionic Beaver, 18.04
	Apr 2017	Susan Hutchinson	Corrections for 2017-2018
	Jan 2017	Susan Hutchinson	Added branded front page
	Apr 2016	Susan Hutchinson	Major changes to Section 4: replaced software installation with remote access to linux.ox.ac.uk.
	Nov 2015	Susan Hutchinson	Minor corrections and checking. Regular expressions.
	Jan 2015	Susan Hutchinson	Minor corrections
	Sep 2014	Susan Hutchinson	Updated for Ubuntu 14.04 (Trusty Tahr)
	May 2014	Susan Hutchinson	Minor updates – replace Usenet with StackExchange

Acknowledgements

This course was originally developed by Jon Lockley and Susan Hutchinson.

About this workshop

This workshop introduces you to the Linux operating system. We will cover using a desktop linux installation, using Ubuntu. As well as covering the use of the graphical desktop, the workshop also covers an introduction to command line access and basic shell scripting. Further to working on a desktop environment we will also cover the basics of accessing remote Linux computers.

We will include pointers to other courses and resources that will help you further familiarise yourself with Linux.

What you need to know

The ideas and techniques covered in this workshop will apply to a range of Linux installations. We will demonstrate using Ubuntu which is widely available and free to download, install and use. However, the concepts will be the same or similar, whatever Linux distribution you decide to use.

I will assume that you are reasonably confident in using graphical user interfaces, for example MS Windows, MacOS, or even a Linux desktop.

- Use of a mouse to navigate to menu options
- Copying, deleting and pasting using shortcut keys.
- Understanding of the concept of files and folders.

If you need to review these activities, LinkedIn Learning is a great place to get guidance. There is an activity with relevant videos in the IT Learning Portfolio: visit skills.it.ox.ac.uk/it-learning-portfolio and search for “[long documents] activity”.

The resources you need

Sample documents that you can use to experiment with will be made available, but you may like to bring along your own.

The resources for most workshops, including any pre-course activity, are in the IT Learning Portfolio: visit skills.it.ox.ac.uk/it-learning-portfolio and search for “Linux: A comprehensive introduction”.

Unless you have been told otherwise, in classroom workshops there will be a computer available for you to use with Ubuntu Linux available.

How to Use this User Guide

This handbook accompanies the taught sessions for the course. Each section contains a brief overview of a topic for your reference and then one or more exercises.

Exercises are arranged as follows:

- A title and brief overview of the tasks to be carried out;
- A numbered set of tasks, together with a brief description of each;
- A numbered set of detailed steps that will achieve each task.

Some exercises, particularly those within the same section, assume that you have completed earlier exercises. Your teacher will direct you to the location of files that are needed for the exercises. If you have any problems with the text or the exercises, please ask the teacher or one of the demonstrators for help.

This book includes plenty of exercise activities – more than can usually be completed during the hands-on sessions of the course. You should select some to try during the course, while the teacher and demonstrator(s) are around to guide you. Other exercises are for you to try on your own, as a reminder or an extension of the work done during the course.

Text Conventions

A number of conventions are used to help you to be clear about what you need to do in each step of a task.

- In general, the word **press** indicates you need to press a key on the keyboard. **Click**, **choose** or **select** refer to using the mouse and clicking on items on the screen. If you have more than one mouse button, click usually refers to the left button unless stated otherwise.
- Names of keys on the keyboard, for example, the Enter (or Return) key are shown like this Enter.
- Multiple key names linked by a - (for example, Ctrl-z) indicate that the first key should be held down while the remaining keys are pressed; all keys can then be released together.
- Words and commands typed in by the user are shown **`like this.`**
- Labels and titles on the screen are shown **`like this`**.
- Drop-down menu options are indicated by the name of the options separated by a vertical bar, for example **File|Print**. In this example you need to select the option **Print** from the **File** menu or tab. To do this, click when the mouse pointer is on the **File** menu or tab name; move the pointer to **Print**; when **Print** is highlighted, click the mouse button again.
- A button to be clicked will look **like this**.
- The names of software packages are identified *like this*, and the names of files to be used **`like this`**.

Contents

Exercises

1 Introduction

Today's course is divided into four parts each of which consists of a presentation followed by exercises.

- Getting started: the desktop, office applications, the terminal.
- Using the command line.
- Editors, regular expressions, shell scripts.
- Working on remote computers.

1.1. Aims for Today

The course is designed to help you become a confident Linux user. The topics covered are described in the Course Outline below.

1.2. Course Outline

1.2.1. Session One

This session will cover:

- A brief history of Linux.
- What is Open Source software?
- Who uses Linux.
- Office applications in Linux.
- What is a shell?
- Some simple commands.

The exercises will look at:

- Exploring and configuring the Linux desktop.
- Exploring LibreOffice applications.
- Simple use of the command line.

1.2.2. Session Two

As well as a powerful and useful desktop, Linux also has a command line interface. This session starts to explore the command line. We will look at

- Viewing files, pathnames, getting help.
- Pattern matching
- Pipes or how to build your own commands

The exercises will cover:

- Using commands to manipulate files.
- Getting help.
- Using patterns for searching for files.
- Building your own commands.

1.2.3. Session Three

This session builds on the previous session's introduction to the command line. We will use the command line to do more complicated searches. We will also look at shell scripting.

- Regular expressions.

- Text editors.
- Writing shell scripts.

The exercises cover:

- Using regular expressions patterns for searching.
- Building your own commands.
- Using an editor to develop shell scripts

1.2.4. Session Four

The final session looks at ways of using remote computers. It will also briefly discuss managing your own computer.

- Remote access commands.
- Using package managers to find and install software.

The exercises will cover

- Using ssh to access a remote computer
- Using scp to move files and directories between computers
- Using screen to leave a program running on a remote computer after disconnection

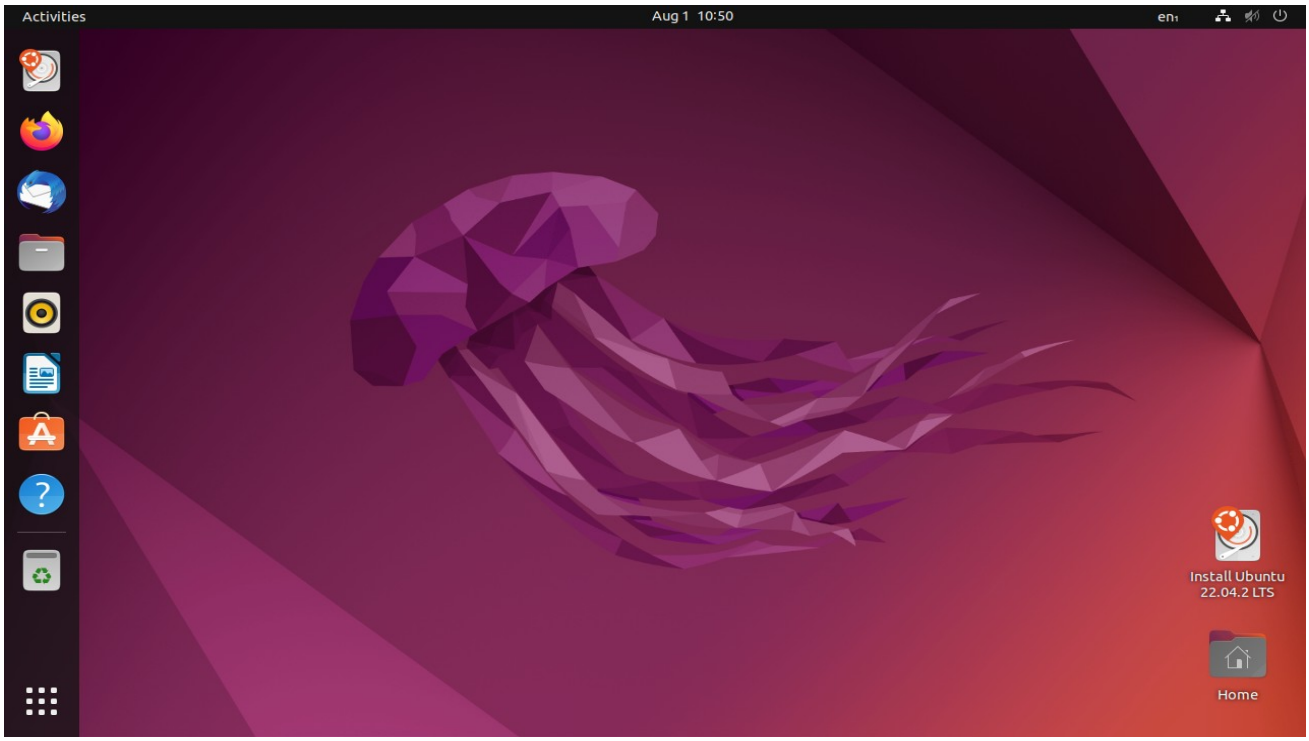
Session One.

2 Getting started

Here are some things to try to help you get to know Ubuntu Linux.

Exercise 1 **A first look**

This is the standard Live Ubuntu start-up screen.



Let's look at the screen more carefully. The bar along the top has three components:

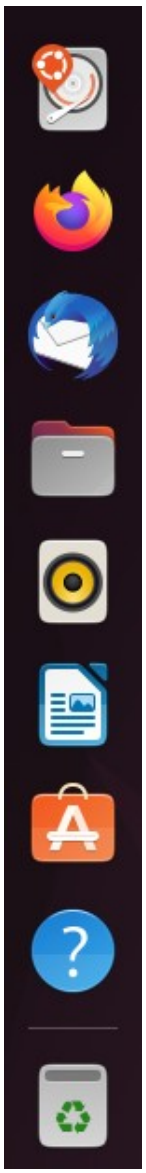
Activities	Show current applications and search for others
Aug 1 10:50	Time – can be configured to include date
en	Various settings: • Language and keyboard settings • Network settings • Volume control • Systems settings and logout

On the left side of the Desktop, below the Activities launcher is the Dock.

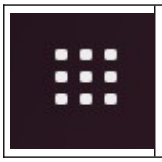
Under the Dock you should see a vertical list of popular applications. If you hold your mouse over any of these icons then their name pops up.

The Install icon can be used to install Ubuntu on your PC, but **please don't do this!**

Dock icon titles:

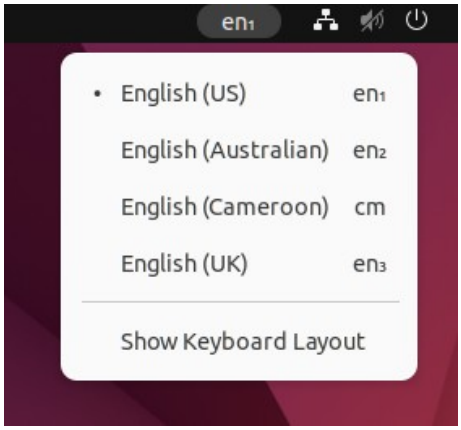
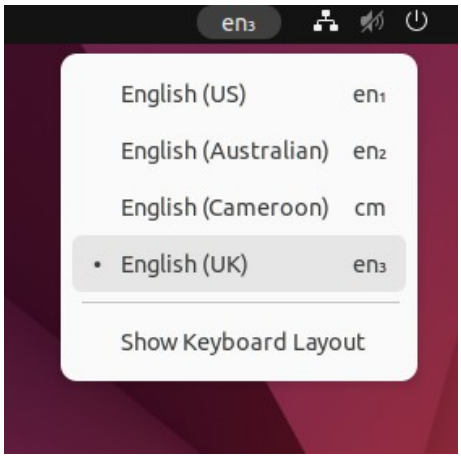
	Install Ubuntu	Install Ubuntu on your PC. **Don't use this now!**
	Firefox Web Browser	Browse the web
	Thunderbird Mail	Read email
	Files	Find files and folders and explore the system
	Rhythmbox	Music player
	LibreOffice Writer	Create documents, similar to Microsoft Word
	Ubuntu Software	Find, add, update and remove applications
	Help	Get help
	Trash	Recycle bin

At the bottom of the Dock is

	Show Applications	

Exercise 2 **Changing the keyboard layout**

The keyboard is currently set up with the US-style layout. To set it to a UK layout:

i. Right click on **en₁** on the top right bar to display the keyboard language selection menu	
ii. Select English (UK) **en3**	

You should now be using a UK Keyboard layout.

Exercise 3 **The home area**

When you log onto a Linux system you are located in your home area. This is where any files that are created during this session are saved. See if you can find a quick way to open a window which shows the contents of your home area.

The answers are at the end of the exercises in 6 Answers.

Exercise 4 **Getting to know the desktop**

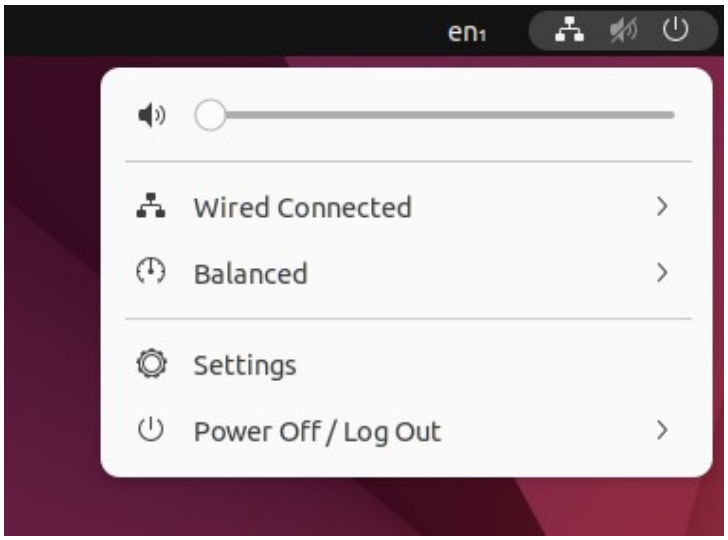
I. Finding applications

From the Dock find

- The Firefox Web Browser
- LibreOffice Writer
- Help

From the top panel find

- System Settings [Hint: Look for the crossed wrench/screwdriver icon under the arrow on the top right]



II. Window operations

We're going to experiment with actions on windows. Start Firefox. Make sure you can do the following to this window

i. Maximise – make full screen size.
ii. Minimise – close the window and send to the Dock.
iii. Restore – bring back a minimised window from the Dock.
iv. Close – exit from a window permanently.
v. Resize – make the window smaller or larger.
vi. Display options for a window. [Hint: right click on the dark grey bar at the top of a window.]

Did you notice that when you have started an application there are small orange dots to the left and right of the icon on the Dock?

What happens if you click on a different application?

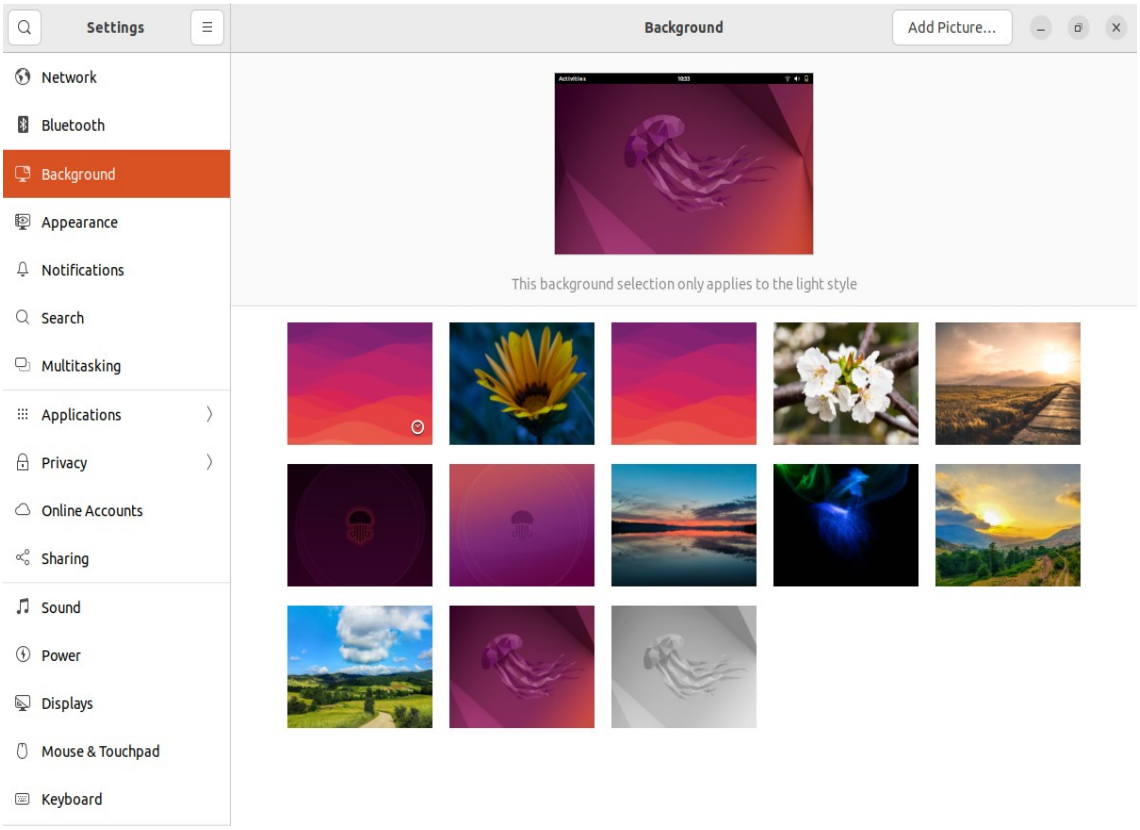
Did you notice that the appearance on the Dock of the active application changes? What happens if you open a new copy of the application so that there are two (or more) instances running?

The answers are at the end of the exercises in 6 Answers.

III. Configuring your desktop

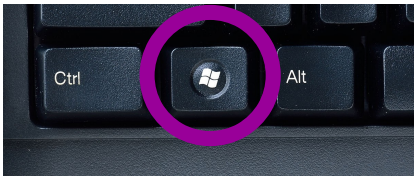
System Settings allows you to configure your desktop environment and many other features.

i. Open the **System Settings** menu and click on **Background**.

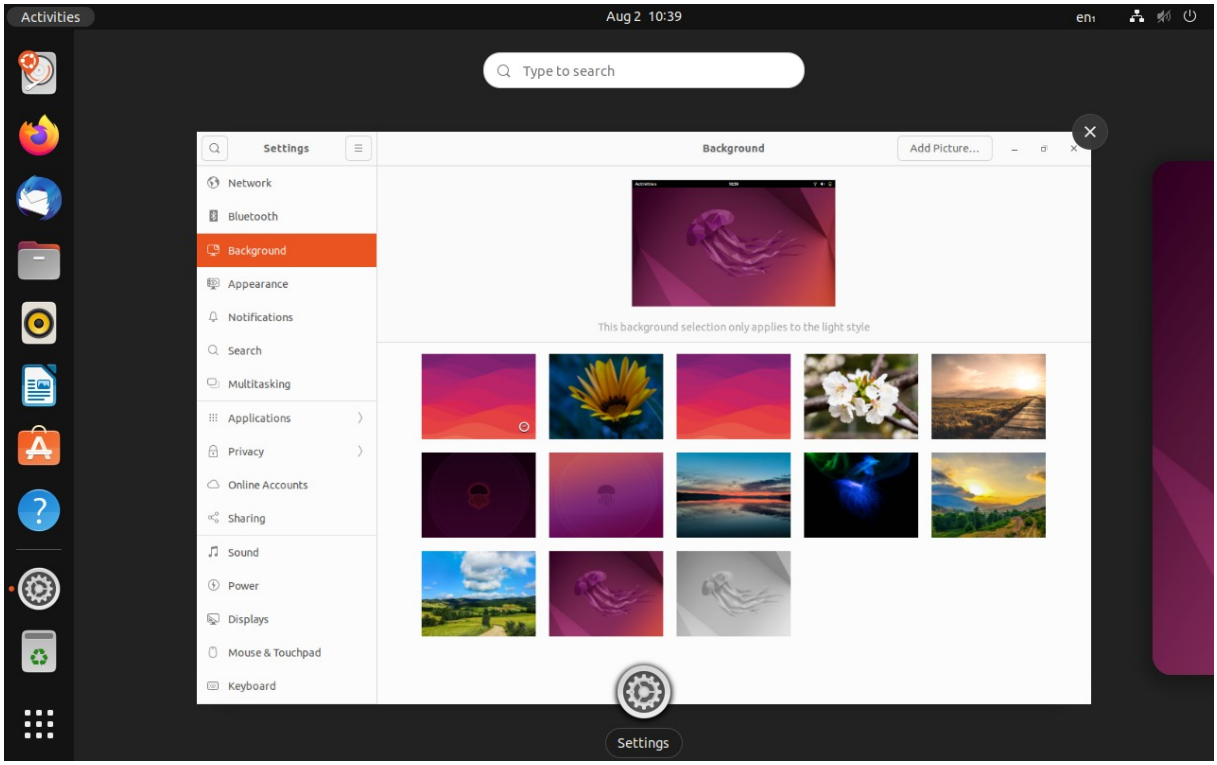


ii. Change your wallpaper. Ubuntu Live has a limited range of wallpapers; a full install will have more.

iii. Linux supports the use of multiple workspaces or desktops. This allows you to organise work more efficiently and reduce clutter. Press the Super (or Windows) key on the bottom left of the keyboard:



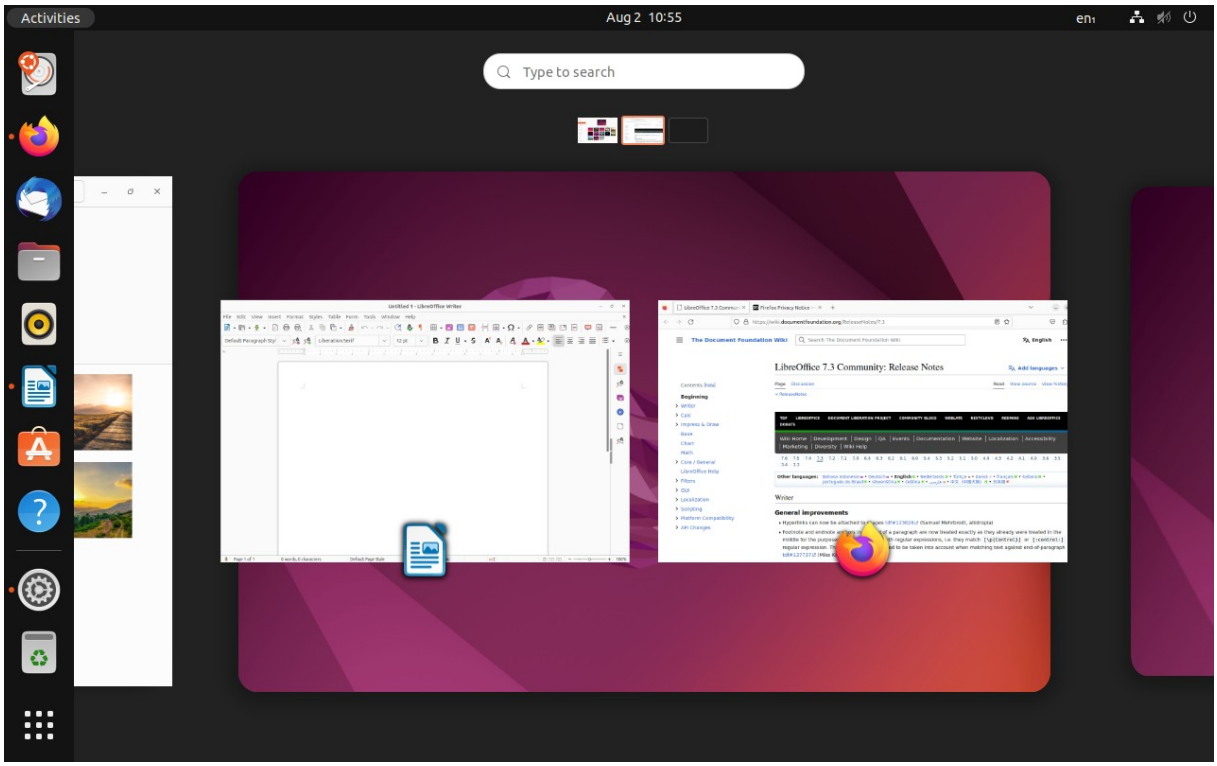
to display additional desktops.



You should see something like this:

You can then use the mouse to move between workspaces, and move applications between these workspaces.

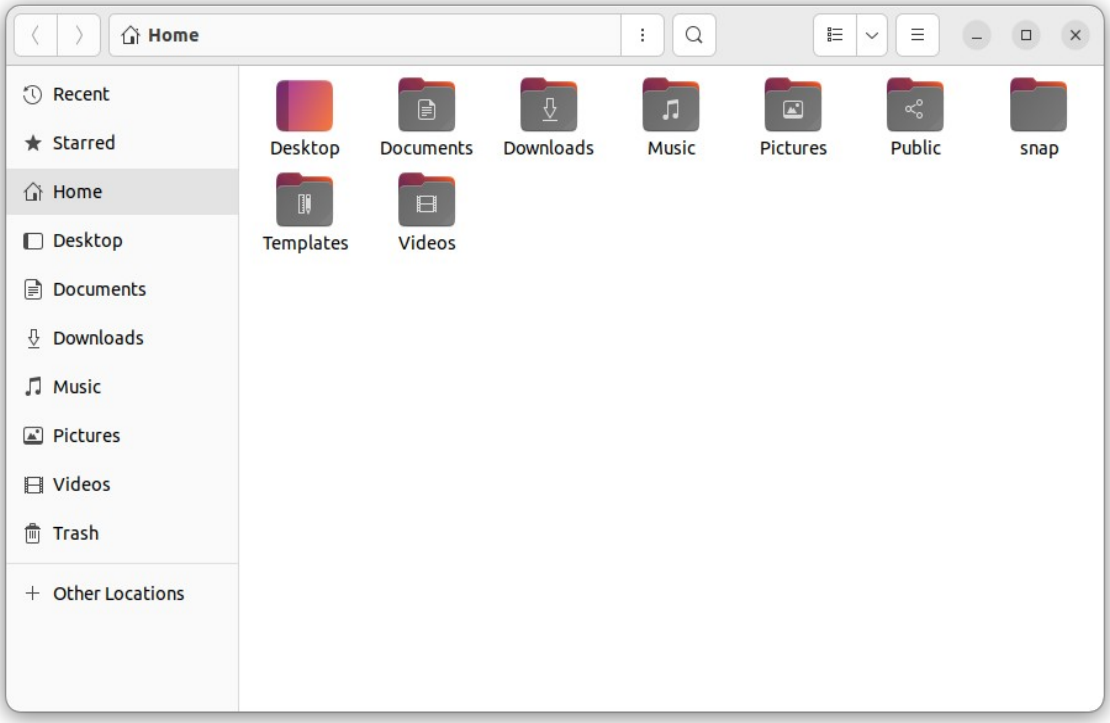
In this screenshot there are applications open in two workspaces.



iv. *More difficult:* Change some keyboard shortcuts. Keyboard shortcuts use the keyboard rather than the mouse to perform actions. Go to **Keyboard** in the System Settings) and use this to set up a quick way to switch between workspaces. Scroll through this list of shortcuts, looking for **Switch to workspace 1** and replace **Super+Home** by, for example, **Ctrl-1** [Hold the Ctrl key and 1 down at the same time]. **Switch to workspace 2** to **Ctrl-2** and so on. Do this for 4 workspaces. You should now be able to switch between workspaces using these keystrokes. Some keyboard shortcuts are pre-configured.
The answers are at the end of the exercises in 6 Answers.

Exercise 5 Finding your way around

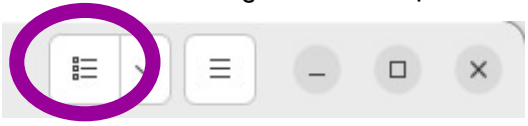
Go to the **Dock** and open **Files**.



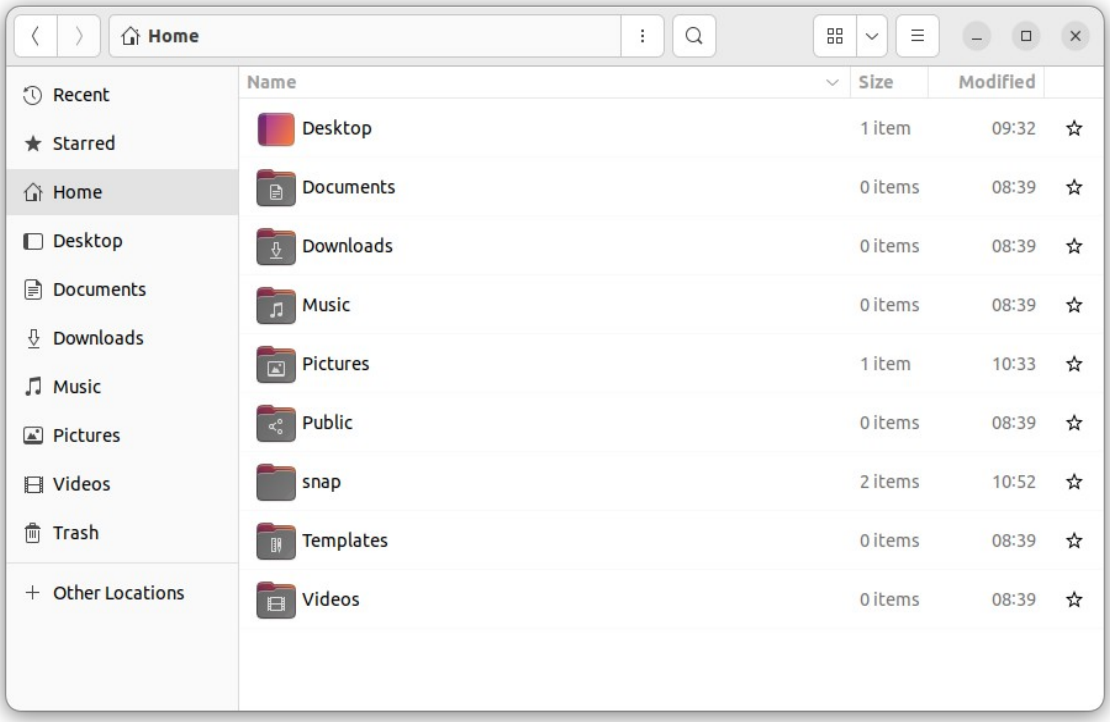
You can use **Files** to browse files and directories (the Linux equivalent of folders).

You should be in an area called **Home**. When you start the file browser, it opens in your home directory. This is the place where your files will be stored by default – that is if you don't specify another location.

Click on the small grid on the top bar

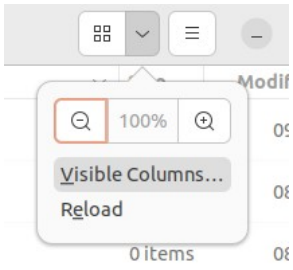


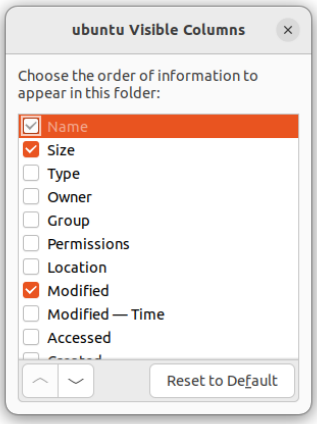
The view in **Files** should change



You can change the properties you see for each file and directory by selecting the **View Options** icon.

Followed by the **Visible Columns...** menu:



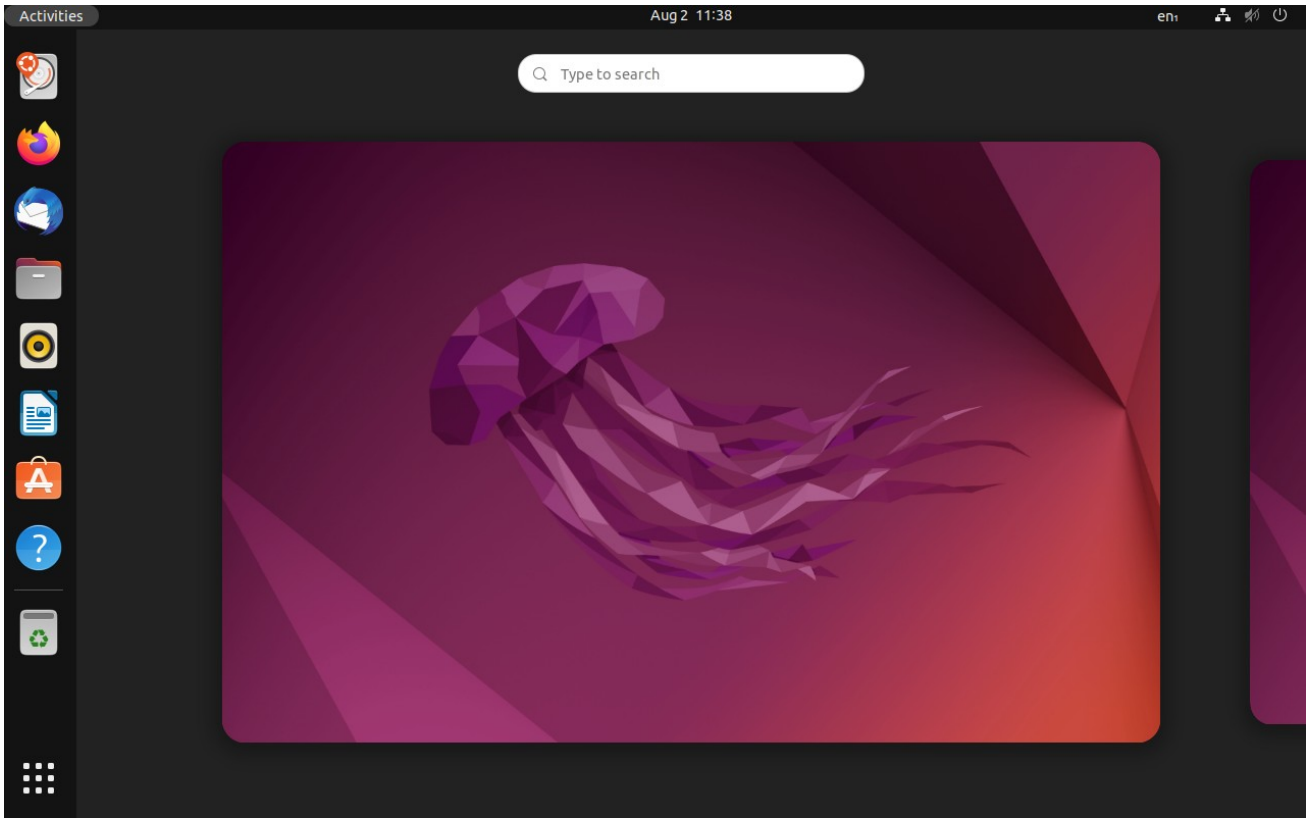


Change this list so that you can see the **Type, Owner, Permissions and Modified** – Time.

Finally, see if you can find a directory that you can't see inside. We will talk about how privileged accounts are used in the final session.

Exercise 6 **Finding applications not on the Dock**

There are only a limited number of applications on the Dock. What if you want to use something that isn't there? Clicking on the Activities icon displays a window like this:



To find an application you can either browse for it by category or search in the search box. In later sessions we will be needing a terminal window. Enter terminal in the search box and see what happens. You may not need to type in the whole word to get what you want.

Once you have found what you want, a single click on the icon will open the application. As well as opening the application, an icon should also appear on the Dock. If you right click on this icon, you can keep the application there by selecting **Add to Favorites**, even after you have closed it. Test this with the terminal window.

The following two exercises look at the LibreOffice equivalents of Microsoft Word and Microsoft PowerPoint. As time is short, choose the one you are most likely to find useful.

Exercise 7 **Working with files**

Use your Firefox browser to visit

https://skills.it.ox.ac.uk/files/CP001_Files_Linux_Intro_Demos.tgz

By default the file will be saved in the "Downloads" folder. Use the File Manager to move the file to the Home directory, then Right-Click on the file and select **'Extract here…'**.

The following files will appear in the directory "CP001_Files_Intro_Demos":

- **Simpsons**
- **Usingmsc.doc**
- **my_date.sh**
- **my_wordcount**

- **small.sh**

Move these files into the Home directory, where they will be used in subsequent exercises.

Exercise 8 **Word Processing**

Start **LibreOffice Writer** from the Dock.

Write a short document (perhaps a covering letter for a job application for example). Don't spend too long on this but see if you can find out (from your experience with MS Word or similar) how to insert things like tables, do indentation and so on. If there's something you'd like to see but you're not sure where it is, ask for help.

Now save what you've done. By default the file is saved in ODF (Open Document Format). Once you done this, see if you can find out how to save a document in MS Word format instead. You can also save the file in PDF format. Again see if you can find out how to do this (it's not in quite the same place).

Now let's close this document by clicking on **File** and **Close**.

In your home directory you should see MS Word file, **Usingmsc.doc** which was downloaded in Exercise 7 Working with files.

In the word processor open this file. It should work perfectly even though it was created with MS Office.

Although most MS Office files work with LibreOffice, a few don't convert very well so be warned that it's not 100% compatible. (Maybe 98% is a fair number).

Now close Writer and go to the next exercise.

Exercise 9 **Slide shows**

Use LibreOffice Impress to make your slide show; it is an application similar to MS PowerPoint. You will need to use **Activities** to search for this.

You don't have time to make a large presentation today so we suggest as an exercise that you make one or more of:

- A three page presentation about your course/research
- A poster for an event/party
- A flier for Ubuntu Linux

Use the Firefox browser in Desktop 1 to search for and to download images and insert them into your presentations. Ask your demonstrators if you can't find the features you want to use.

Save your work as both in *Open Document* format and *MS PowerPoint* format.

Now close the application.

Exercise 10 **StackExchange**

We will now look at another source of information and help that is available. StackExchange provides a gateway to communities of expertise. A particular benefit of the sites is the lack of additional chat – in general there is a strong focus on direct answers to questions. Browse to **http://stackexchange.com/about** to find out more.

Browse to **http://www.stackexchange.com** and click on 'Explore our sites!' to see the huge range of topics covered. Let's investigate the 'Ask Ubuntu' site. On my browser the link to this site appears in a large orange box to the right of the large 'Mathematics' box which is near the top on the

left hand side. If you can't find the link then use the browser search feature (usually Ctrl-f) and enter Ubuntu.

When you click on the '**Ask Ubuntu**' box you should see a '**Visit Site**' button. Click on this.

Now click on 'Take the 2-minute tour' to see a brief guide.

If, at the end, you have time have a look at other useful sites such as 'Unix & Linux'.

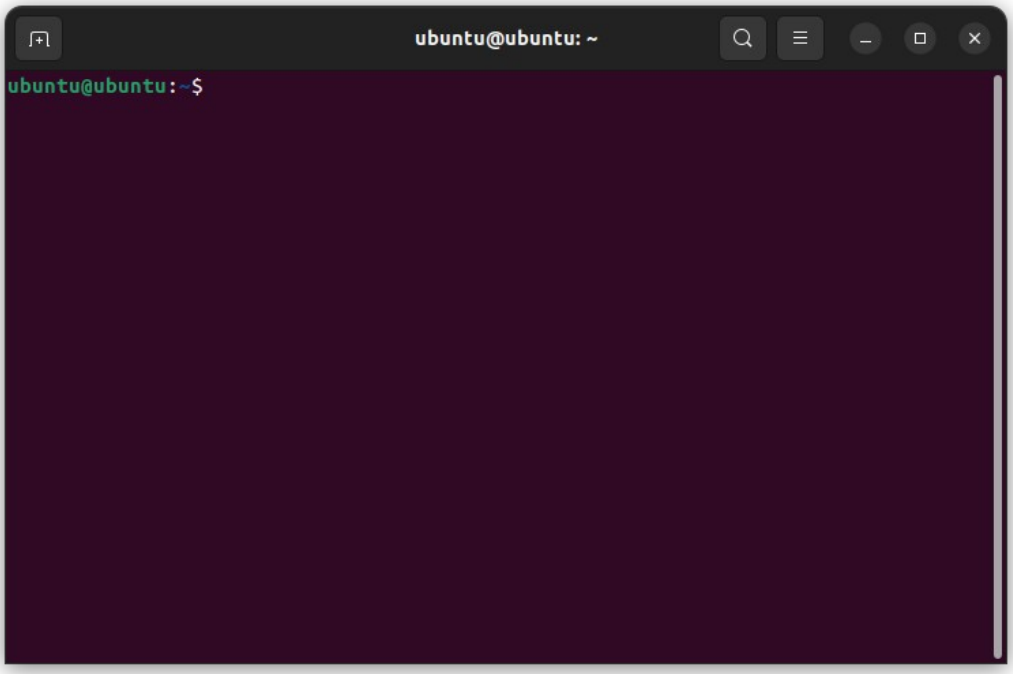
Exercise 11 **Starting a terminal window**

Now that you are more familiar with the Ubuntu desktop, let's start looking at the command line interface.

The first thing we need is a shell prompt/command line. When using a graphical desktop there is usually a terminal application which gives you access to the command line.

Click on 'Activities' and enter Terminal in the search box. When you have found the terminal open it. If you want to keep it on the Dock, right click on the icon on the left hand side of the screen and select '**Add to Favourites**'.

You should see a Window open that looks like this:



Note that it's perfectly possible to have a command line with no graphical desktop at all. This is often the case with server systems which are not used interactively and need all the processor power and memory they can get for computation.

We will be finding out more using remote server systems in the final section.

Exercise 12 **Where am I? What's all this?**

Let's start to look at navigation of the Linux file system. The following commands are introduced:

Command	Purpose
pwd	Print working directory. In other words, “where am I?”
ls [options] directory	List files. If used on its own, it lists everything in the “current working directory“, (where you are currently “located”).
file filename	Tells you what sort of file the file called filename (for example) is.
cd	Change directory. In other words, “please change my current location”.
man command	Command manual pages.

Note that all commands are typed in lower case. There are very few Linux commands which have any uppercase (CAPITAL) letters. We will look at case-sensitivity and file names in the next session.

Right away we can see how *quiet* Linux commands are by default. Try typing in

```
$ cd
```

at the **ubuntu@ubuntu**:~$ prompt and you will get no output at all. This does not mean that anything has gone wrong. For many commands, no output means successful completion.

A digression on prompts. You can customise your prompt to look however you like. We won't do that now, but you will notice that it changes as you move around the file system.

Not all commands are silent. Try

```
$ pwd
```

You should get a response like: `/home/ubuntu`. Now try

```
$ ls
```

You should now see a listing of all the files in the directory `/home/ubuntu`. Let's try finding out what sort of files each is. Take the file “Desktop”.

```
$ file Desktop
```

The shell tells you that this isn't a regular file, it's a directory. In other words it's a special file which acts as a holder for yet more files (like a folder in Windows). If you now try

```
$ file Desktop/*
```

you should see a description of the two files in that directory.

TIME SAVING INFORMATION

If you haven't already tried this then you should now. A lot of typing can be avoided by several useful shortcuts. The <tab> key can be used to complete commands file names and the arrow keys to recall previous commands and perhaps change them.

Exercise 13 **File and directory manipulation**

Now you are going to create a directory and put some files there. The commands you need are

Command	**Purpose**
cd	Change directory. In other words, “please change my current location”.
mkdir *directoryname*	Create a directory called *directoryname*
touch *file1 file2*	Create one or more empty file(s) called *file1*, *file2*
cp *file1 file2*	Copy *file1* to *file2*. Can also be used to copy whole directories.
ls	List files. If used on its own, it lists everything in the “current working directory“ (where you are currently “located”).
rm *file1*	Remove (or delete) a file called file1. Can also be used to remove whole directories.

```
$ cd
$ mkdir directory1
$ cd directory1
$ touch file1 file2 file3 file4
```

Remember that words in italic should be replaced by names that you have chosen. Experiment to see what happens if you are not in your home directory. What happens if you try to create a directory in **/usr/bin**? Is there anywhere outside your home directory where you are allowed to create directories? [Hint: look at the top level directory **/** - you should be able to create a files and directories in one of those. The name of the directory might also be a clue.

The answers are at the end of the exercises in 6 Answers.

Use the **cp** command to copy one file to another and then use **ls** to check that you have done what you want. Then delete a file using

```
$ rm file1
```

Now we are going to copy one directory to another. The commands you need are

```
$ cd
$ cp -r directory1 directory2
```

Use ls to make sure you have done what you want. The new directory should contain exactly the same files as the old one. Note use of the *-r* option. This makes **cp** copy the contents of a directory – this is known as a recursive copy.

Finally remove the new directory with

```
$ rm -rf directory2
```

Note that this is a dangerous command and should be used with care!

Use ls to check that this has worked. You should now be familiar with these simple file manipulation commands. Remember that in Linux the **rm** command really does delete files. There is no Recycle Bin to retrieve files that were deleted by mistake.

This completes the exercises for Session One

Session Two.

3 Command line exercises

Exercise 14 **Viewing files**

We're going to download some files and directories which will be used during these exercises. Although it is possible to use a browser to download this file you can also do this from the command line. Use

```
$ cd
$ wget https://skills.it.ox.ac.uk/files/CP001_Files_Linux_Intro_LinuxFiles.tgz
```

to download the files and then

```
$ tar -xvzf CP001_Files_Linux_Intro_LinuxFiles.tgz
```

to unpack them.

Command	Purpose
cat *file*	Show the whole contents of a file called *file*
more *file*	Display the contents of *file* a screenful at a time
less *file*	Display the contents of *file* a screenful at a time but with more options. For example, after starting less you can enter G and go straight to the end of a file and then move backwards.

Use the following commands to look at the contents of the file **google.txt.**

```
$ cd Files
$ cat google.txt
```

This is not very useful if the file is more than a screenful.

```
$ more google.txt
```

Note that <space> takes you to the next page and q will quit before the end of the file. Now try

```
$ less google.txt
```

See if you can go straight to the end of the file. Then use q to exit.

Exercise 15 **Absolute and relative pathnames**

We're now going to make use of two things, the **ls** command and the knowledge that the file called `/usr/share/info` is a directory, to illustrate the concepts of *absolute* and *relative pathnames*.

```
$ cd
$ cd ../../usr/share
$ ls info
```

and you will get a listing of the contents of the directory.

```
$ ls /usr/share/info
```

and you should get the *same* list of files.

The absolute (i.e. complete) location of the **info** directory is **/usr/share/info**. We have just asked to see what is kept inside it in two different ways. The first is a *relative* pathname while the second is the *full* or *absolute* name.

Imagine the **info** directory is a particular house, say 42, High Street, Abingdon and I ask you to deliver a letter there. I could tell you to deliver the letter to "42 High Street, Abingdon": the full/absolute address. No matter where you are in the UK, that's enough information. However, if you were already in Abingdon I could tell you to deliver the letter to the relative address of "42, High Street" or even better, if you were standing on the high street just "number 42" would be enough.

The "**ls** *info*" command above worked because you were already in the **/usr/share/directory**. It wouldn't work from somewhere else. The command "**ls** */usr/share/info*" command will work from anywhere (although it's more long winded). Let's prove it by changing our current location using the **cd** command to go back to the home directory

```
$ cd
$ cd Desktop
$ pwd
```

you should get "/home/ubuntu/Desktop" i.e. you have moved into the Desktop directory.

```
$ ls /home/ubuntu/Desktop
```

should give you the list of files in that directory. In fact you could use "**ls**" on its own without the name of the directory because you have already moved there with **cd**. Let's see what happens when we deliberately do something wrong:

```
$ ls Desktop
```

should give you an error saying there is "No such file or directory" which is correct. The command fails because Desktop on its own is a *relative* name and you've started from the wrong place.

Let's expand the idea of relative and absolute path names using the **cd** command. Make sure you are still in the Desktop directory before you start (check with **pwd**).

```
$ pwd
$ cd ..
$ pwd
$ cd ..
$ pwd
```

And so on until you can't go any further (you won't see an error, you just stop going anywhere). "**..**" is a special location which means "up one level". All directories contain a "**..**" so you can go up a level. The exception is called "**/**" or "the root" or just "slash". You can't go higher than **/** so "**..**" doesn't take you anywhere. Note that there is another special directory called "**.**" which means "current location". All directories contain a "**.**" directory and we'll see why it is needed later.

During the above task you went up the directories one level at a time. Now reverse the process and go back to the Desktop directory one level at a time. You should be in "**/**".

*Note that you don't have to do the pwd(s) but it may help you visualize what is going on. You can also use **ls** to have a look around each level if you have time.*

```
$ cd home
$ pwd
$ cd ubuntu
$ pwd
$ cd Desktop
$ pwd
```

Try to answer/do the following:

Were you just using absolute or relative paths?

1. Now try to get back to the root (or "**/**") directory with one command only using an absolute path.
2. Now get back to the **/home/ubuntu/Desktop** directory using one command only.
3. What are the contents of the "**/**" directory? From your home directory use one command only to find out.

Exercise 16 **Help commands**

Command	**Purpose**
man *command*	Read the manual page for a *command*. So **man** *ls* gives you details about **ls** and **man** *less* would give you details about *less*.
apropos *word*	Search manual pages for names and descriptions. So **apropos copy** would list all the commands that have the word copy in the description.
which *command*	Displays the location of the command you are using.
whatis *command*	Gives a brief description of a command

If you know what command you need, you can use the man command to find out the details of that command. Try it with a few of the commands you have used already. Not all commands have as many options as **ls**!

```
$ man ls
```

to find out details of the **ls** command.

1. What option is used to display modification time?
2. What option is used to display the size of a file?

3. How can you reverse the order of the sort so that the largest/most recently changed file is at the bottom of the list?
4. Check that they do what you expect.

Sometimes you might not be sure exactly what the command is. In that case you can use the apropos command which finds all command descriptions which match a given word. So to find out what commands there are to manipulate files are available use

```
$ apropos file
```

Note that the output from this command is very long. We will look at a neat way round this in the next session. Here is a real life example. I needed to know which command in this Ubuntu distribution was the equivalent of Microsoft Draw so I ran

```
$ apropos draw
```

See if you can see the name of the command I used.

The answers are at the end of the exercises in 6 Answers.

Sometimes you need to know where Linux stores command. Use **which** to display the location of the file. Try it with **less, more, cp, apropos**:

```
$ which apropos
$ which cp
$ which more
$ which less
```

Did you notice that **more** and **less** are stored in different places? The directory **/bin** conventionally is used to store a few necessary commands that can be used if all else fails. Finally you may have seen a command and want to know briefly what it does. Use the **whatis** command to find out. Try this on some commands.

Exercise 17 **File and directory names**

These are the commands that will be used:

Command	**Purpose**
cd	Change directory. In other words, “please change my current location”.
cat *file*	Show the whole contents of a file called *file*

We’re going to look at some of the problems you can encounter with files and directories. You should have a directory called **Files**. Now do this

```
$ cd
$ cd Files
$ cd TestDir
```

First look in the **Cases** directory.

```
$ cd Cases
```

Make sure that you can read all three files there. Now look in the **OpenThis** directory.

```
$ cd ..
$ cd OpenThis
```

1. Now see if you can read the files **star, astar** and ***star**. What happens when you try to read the file called ***star**? The ***** character has caused confusion because it is a wildcard – it matches any character. We will be looking at wildcards and pattern matching in a later exercise. See if you can read this file without reading the other ones. Using google is not cheating!
   The answers are at the end of the exercises in 6 Answers.
2. Now change into the directory **Open This.** Double quotes or backslashes help.
3. Now try to read the file called **-ReadMe**. Using quotations round the filename or the escape character (\) won't work this time. In each case the command (one of cat, more or less) is interpreting the leading – (hyphen) as an option. Again, don't be afraid to google to see if you can find the answer to this. Try deleting the file called **-DeleteMe**. To help you with this, **man** *less* says:
   -- A command line argument of "**--**" marks the end of option arguments Any arguments following this are interpreted as file-names. This can be useful when viewing a file whose name begins with a "**-**" or "**+**".
   The answers are at the end of the exercises in 6 Answers..

Exercise 18 **Looking at files**

These are the commands that will be used:

Command	**Purpose**
cat *file*	Show the entire contents of a file called *file*.
head ***file***	Display the first 10 lines of a *file*.
less *file*	Display the contents of *file* a screenful at a time but with more options. For example, after starting less you can enter *G* and go straight to the end of a file and move backwards.
man *command*	Read the manual pages to read all about a command. So **man** *head* would describe how to use the man command.
more *file*	Display the contents of *file* a screenful at a time.
tail *file*	Display the last 10 lines of a *file*.
wc *file*	Counts the number of characters, words and lines in a *file*.

Using

```
$ cd Advanced
```

```
$ cat longfile.txt
```

you see all of **longfile.txt;** using

```
$ less longfile.txt
```

gives you a screenful at a time. Use the space-bar to move on a screenful, **G** to go to the end of the file and **q** to exit. Now use

```
$ head longfile.txt
```

and

```
$ tail longfile.txt
```

to look at the first and last 10 lines of the file.

Now use the **wc** command to find out the number of lines, words and characters there are in **longfile.txt**.

Exercise 19 Using wildcards to match filenames

File globbing or wildcard expansion allows you to use special characters to match more than one file or directory name.

Character	What it matches
***	The ***** (asterisk or star) matches any number of characters or none.
?	The *?* (question mark) matches exactly one occurrence of any character.
[ ]	Matches any characters in a given range.

Change back to your home directory (if you are not already there) and then change to the directory called **WildCards**.

```
$ cd
$ cd WildCards
```

Now experiment with wild card characters. What do the following match?

```
$ ls foo?
$ ls foo2*
$ ls foo[1-2]
```

What command would you need to match just foo20 and foo2bar? [Hint: you might need to use more than one wild card character.]

The answers are at the end of the exercises in 6 Answers.

Now use

```
$ wc -l *
```

to find the length of each file. Note that the **-l** is a hyphen followed by the lowercase letter l. You should see output like this:

```
1 foo
```

```
 2 foo1
 2 foo10
 1 foo2
 1 foo20
 3 foo2bar
10 total
```

Now see if you can create a match so that **`wc -l`** just shows the files with a 1 (the number one) in their name. Again a possible answer is at the end.

Exercise 20 **Searching and sorting**

In this exercise we will use commands to search for patterns within files and sort the contents of files.

Command	**Description**
grep *pattern file*	Search for the characters in *pattern* within *file*.
sort *file*	Sorts the contents of one or more files.

Now change into the **`Searching`** directory.

```
$ cd
$ cd Searching
```

I suggest you use the cat command to check the contents of the two files, fruit and veg, so that you can see what they contain. Now use

```
$ grep melon fruit
```

Can you see what has happened? Only the matching lines in the file are shown. Try

```
$ grep green fruit veg
```

What do you think has happened here? Did you notice that the file names were included in the output. This is because more than one file (both fruit and veg) has been searched and so the output informs us where the matches were found.

Using

```
$ man grep
```

see if you can find the option that causes **grep** to ignore case so that both Melon and melon would be found.

Finally use the **sort** command to order the files. Note that you can sort more than one file which merges the output of all the files.

```
$ sort fruit veg
```

What is the option that reverses the order – so that the fruit and vegetables are sorted in reverse alphabetical order?

The answers are at the end of the exercises in 6 Answers.

Exercise 21 **Pipes and redirection**

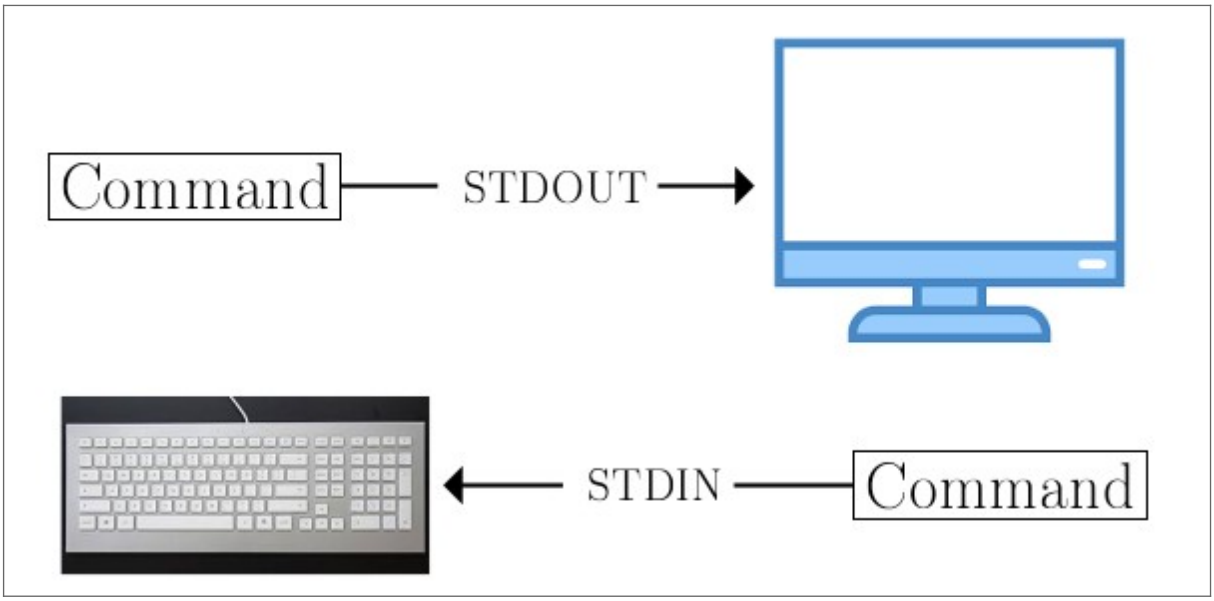
In this exercise we are going to explore two very powerful command-line features which increase the flexibility and range enormously.

Command	**Description**
`du -sk`	Displays sizes in Kilobytes of all files in a directory.
`grep`	Search for the characters in *`pattern`* within *`file`*.
`sort`	Sorts the contents of one or more files.
`tail`	View the last few lines of a file
`wc`	Counts the number of characters, words and lines in a file.

and we will be using the following characters. If you haven't already remapped your keyboard you should do that now. See Exercise 2.

Character	**Purpose**
>	Sends the output from a command to the named file. If the file already exists the previous contents will be lost. If the file doesn't exist it will be created.
>>	Appends the output from a command to the named file. If the file doesn't exist it will be created.
<	Reads input from the named file. NB This option is rarely used.
|	Uses the output from one command as the input to the next.

Almost all Unix/Linux commands use standard input for receiving instructions and standard output for displaying the results.



Pipes make use of commands to join STDOUT to STDIN.

So we could run

```
$ grep green fruit veg > output
```

which would store the output of the *grep* command in a file called output. Check this with

```
$ cat output
```

If you run the command again, this time using

```
$ grep green fruit veg >> output
```

you should see that there are now two copies of the output. It is also possible to use < redirect the input from the keyboard to a file. For example

```
$ cat <output
```

also works.

In the previous exercise we used grep to look for occurrences of the word ***green*** in two files. Obviously in this short example we can count how many times green appears, but when there are many matches it would be useful to use wc to find out. We can redirect the output from the command into a file.

Note that in all cases the **-l** option to the **wc** command is a minus sign followed by the lower case letter **l**.

```
$ grep green fruit veg >output
```

and then run wc on the file

```
$ wc -l output
```

but it would be much more efficient to join the two commands together with a pipe. Use

```
$ grep green fruit veg | wc -l
```

Here the two commands **grep green fruit veg** and **wc -l** are joined together by a special symbol called a pipe.

Exercise 22 **Finding the largest file**

Now we're going build a longer command which will find the 5 largest files in a directory. When building pipes of commands it often helps if you make sure each link in the pipe works before adding the next.

First display the size in Kilobytes of all the files in the **/usr/bin** directory.

```
$ du -sk /usr/bin/*
```

Now sort this output by size, so that the largest are first.

```
$ du -sk /usr/bin/* | sort -nr
```

Now display the final 5 lines which will be the 5 largest files.

```
$ du -sk /usr/bin/* | sort -nr | head -5
```

Now how you would you find the 5 largest files in the **/usr/bin** directory beginning with the letter '**s**'?

Exercise 23 **Merge information from different files**

Command	Description
awk	A pattern scanning and text matching program.
paste	Merge lines of a file

This is a rather contrived example but demonstrates some very useful Unix/Linux commands. Make sure you are still in the Searching directory.

```
$ cd
$ cd Searching
```

Now we are going to use the **awk** command to print out only the first column of the file called creatures. Have a look at the file before you run this command.

```
$ awk '{print $1}' creatures
```

What do you think would appear if you replaced *$1* by *$2*? Make sure you copy the command exactly. In particular you need the single quote character and curly brackets. Now change this to

```
$ awk '{print "A", $1, "eats"}' creatures
```

As well as a list of animals you should also see that the words "**A**" and "**eats**" appears either side of each animal. Now we are going to use the paste command to include the output from the fruit file.

```
$ awk '{print "A", $1, "eats"}' creatures | paste -d" " - fruit
```

Again, make sure that you have entered the commands exactly as they appear here. To explain what the paste command is doing: the **-d" "** option makes a space rather than a tab appear before the name of each fruit, the – before **fruit** means that the output from **awk**... is included.

Now see if you can make the fish eat the vegetables!

The answers are at the end of the exercises in 6 Answers.

Exercise 24 **Unpacking a longer example**

In this exercise you will look at a longer example and repeatedly remove commands to unpack a complex command. Enter these commands.

```
$ cd
$ cd Advanced
$ cat my_wordcount
```

We will be using these commands

Command	Description
sed	Allows you to change the contents of a file; frequently used in pipes.
sort	Sorts the contents of one or more files.
tail	View the last few lines of a file
tr	Translate either groups of characters or single characters.
uniq	Report or remove duplicate lines.

When you run the commands

```
$ chmod +x  my_wordcount
$ ./my_wordcount longfile.txt
```

you should see output like this:

```
      6 microsoft
      6 more
      6 mr
      6 said
      6 that
      7 has
      7 yahoo
      8 engine
      9 and
      9 on
     10 pages
     10 web
     11 in
     12 of
     13 its
     14 a
```

```
15 search
16 google
20 to
29 the
```

This is a list of the 20 most common words used in the file **longfile.txt**. Now, from the command line, we're going to run the command repeatedly, each time removing the final element, to unpack how it works.

Using the mouse, copy the contents of my_wordcount and paste into the command line. You should have a very long command like this:

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:] | sed -e
   '/^$/d' | sort | uniq -c | sort -n | tail -20
```

This looks very complicated!

Now lets remove the final pipe which is

```
| tail -20
```

so you have

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:] | sed -e
   '/^$/d' | sort | uniq -c | sort -n
```

Remember that you can use the up arrow and backspace keys to do this. There is no need to type anything in. This time the output shows all the words in the file, not just the 20 most frequent. Now let's remove

```
| sort -n
```

and see what happens

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:] | sed -e
   '/^$/d' | sort | uniq -c
```

Now the list is sorted alphabetically rather than numerically. Again, remove the final pipe,

```
| uniq -c
```

and see what happens.

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:] | sed -e
   '/^$/d' | sort
```

Now you should see an alphabetical list of all the words, but with multiple occurrences of many words.

Remember you can always add

```
| less
```

on to each version of the command so that you can look at the output a screenful at a time.

The **uniq** command (which is very useful) removes duplicates, and with the **-c** option, adds a count of the number of times it has found a match. Now remove the final pipe,

```
| sort
```

again so that you have

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:] | sed -e
   '/^$/d'
```

This time you have all the words in the file with one word on each line but in the order they appeared in the original file, not sorted alphabetically. You can read the contents of **longfile.txt** if you want to check this. Now again remove the last pipe again

```
| sed -e '/^$/d'
```

and see what the output looks like.

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012' | tr [:upper:] [:lower:]
```

The file no longer contains any blank lines. If you haven't already add **| less** on to this version and the previous version to compare. Again, remove the final pipe,

```
| tr [:upper:] [:lower:]
```

and see what happens.

Did you notice that some uppercase letters appeared? The **tr** command allows you to translate either classes of characters (such as all uppercase characters) or individual characters to something else. In this case we are translating all uppercase characters to their lowercase equivalent. Now remove the final pipe again.

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt | tr ' ' '\012
```

Now you should see the file with many words on each line. In this case we have used **tr** to translate all space characters to the newline character (which is represented by **\012**).. The first command – which is very complicated - removes some punctuation from **longfile.txt**.

```
$ sed -e 's/\.//g' -e 's/,//g' -e 's/\!//g' -e "s/\'//g" -e "s/\"//g" -e
   's/-//g' longfile.txt
```

This is a complicated command! We are using the **sed** editor to match and remove several separate things. Each match begins with **-e** and involves replacing various punctuation characters with nothing at all. In some cases the punctuation character is preceded by "\" because it has special meaning in the **sed** editor.

This completes the exercises for Session Two

Session Three.

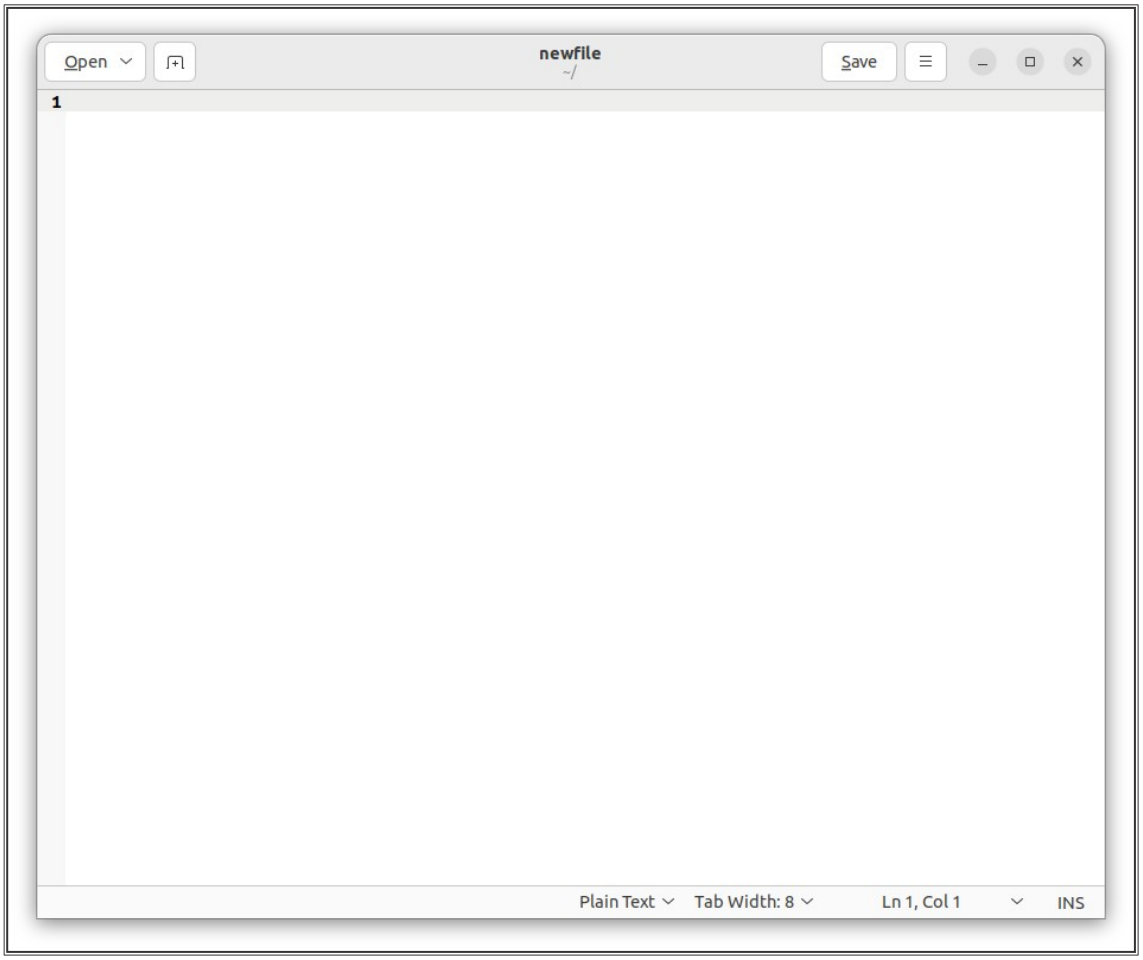
4 Editors, regular expressions, and shell scripts

Exercise 25 **Using gedit**

First open a terminal window so that you can use the command line. Using this window type in

```
$ gedit newfile &
```

A window like this should appear. This command starts the *gedit* editor, creates a file called **newfile** and returns you to the shell prompt.



The new gedit window you have opened consists of

- a menu bar at the top with **Open**, **New Document** and **Save** buttons. Note that an expanded menu appears on the bar at the top of the screen.
- a main window for entering text

If other files are open then you can have a series of tabs in the main window.

You can now enter text in the main window. You can move around the file with the Arrow keys or the Page Up and Page Down keys.

Try them! Now see if you can enter some text into the file, save the text and quit gedit. Now start the editor up again with the same file, change the text, save it and quit again.

A small digression. When you started gedit you entered **`gedit newfile &.`** The **`&`** character has a special function. It allows you to start a command and continue using the terminal window to enter further commands. Technically it is called 'running a program in the background'. See what happens if you don´t include the & - just type in **`gedit newfile`**. You will need to exit gedit or enter Ctrl-c [i.e. press the Ctrl key and c at the same time] to get back control of the command line.

A brief summary of some frequently used command line control sequences are

Shortcut	**Description**
Ctrl-c	interrupt a running program
Ctrl-d	send an end of file, ending text input for most Linux/Unix programs
Ctrl-s	freezes the screen. Unfreeze with Ctrl-q
Ctrl-q	unfreezes the screen if Ctrl-s has been used to freeze it.
Ctrl-z	suspends a running program. Use **`bg`** to continue running the program in the background; or **`fg`** to continue running the program as it was.

Exercise 26 **Simple regular expressions**

Or finding needles in haystacks! We are going to use the following commands and shorthand characters..

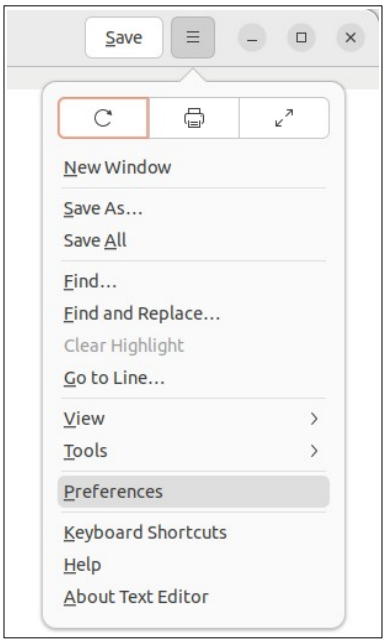
Command	**Description**
`gedit`	A simple text editor
`grep`	Print lines matching a pattern. Options used: **`-n`** display line numbers **`-E`** use extended regular expressions

Regular expression shorthand character	Description
^	Match the beginning of the line
$	Match the end of the line

Have a look at the file **HAYSTACK**. It's going to be easier to look at it in gedit so enter

```
$ cd
$ gedit HAYSTACK &
```

from a terminal window. If the line numbers are not displayed it is helpful to show them. To do this, on the menu bar, select the **menu** icon:



Then select Preferences. On the **View** tab select '**Display line numbers**' which should be the first item.

We'll spend more time looking at gedit in the next session, but for now we're going to use it to show the contents of a file.

Have a look at the file. There are many lines of text with mostly the word 'hay' none or more times. There are some needles hidden in there, as well as some nedles, neeedles and noodles.

A simple search

```
$ grep -n -E needles HAYSTACK
```

will show all the occurrences of needles in the file.

```
$ grep -n -E ^needles HAYSTACK
```

will show the lines that contain needles right at the beginning of the line. The search

```
$ grep -n -E needles$ HAYSTACK
```

will show only the lines that have needles at the end of the line.

How would you search for a line which contains only one occurrence of *needle*s and nothing else?

The answers are at the end of the exercises in 6 Answers.

Exercise 27 **Regular expressions with spaces**

Command	**Description**
grep	Print lines matching a pattern

Regular expression shorthand character	**Description**
\s	match white space
\b	match a word boundary
\B	match a non-word boundary
+	matches one or more of the previous character

Sometimes words at the beginning of a line are preceded by white space (either a tab or a space). Regular expressions have many character shorthands which allow us to make general searches. In this example we use '\s' to match a single white space character.

```
$ grep -n -E '^\sneedles' HAYSTACK
```

to find all the lines that begin with a single white space and then needles.

What if there are multiple spaces? We can use the '+' character which will match one or more instances:

```
$ grep -n -E '^\s+needles' HAYSTACK
```

How many lines end with one or more white spaces?

```
$ grep -n -E '\s+$' HAYSTACK | wc -l
```

Note that this time, we haven't restricted the search just to lines ending in needles.

In some places hay and needles have been run together. To find only the places where needles is separate from hay use

```
$ grep -n -E '\bneedles\b' HAYSTACK
```

This is a particularly useful search because it means we don't have to worry about matches for beginning and ends of lines. Incidentally we can reverse this by using the

```
$ grep -n -E '\Bneedles' HAYSTACK
```

and

```
$ grep -n -E 'needles\B' HAYSTACK
```

which will find instances where needles are joined to hay either at the beginning or end of the word.

What happens if you use

```
$ grep -n -E '\Bneedles\B' HAYSTACK
```

The answers are at the end of the exercises in 6 Answers.

Exercise 28 **Finding noodles**

Command	**Description**
grep	Print lines matching a pattern

Regular expression shorthand character	**Description**
|	give alternative matches
.	matches one or more characters
{ }	specifies a number of matches

Looking at the contents of **HAYSTACK**, we can see that there are words similar to *needles* - *noodles*, *nedles* and *neeedles*. How can we find them all?

We could use the alternation character: '|' to find needles or noodles.

```
$ grep -n -E 'needles|noodles' HAYSTACK
```

This is a little restricted as we may need to specify many alternatives. Try this:

```
$ grep -n -E 'n.{0,2}dles' HAYSTACK
```

This will match all strings that

- start with '*n*'
- have 0, 1 or 2 characters before '*dles*'

Exercise 29 **Changing what you've found.**

This is slightly more challenging

Command	Description
grep	print lines matching a pattern
sed	a stream editor; it allows you to change the contents of a file and is often used in piped commands. Options used: **-r** Use extended regular expressions in a search **-e** Add the script to commands to be executed **s/*find*/*replace*/g** Change the string specified by ***find***, by string specified by ***replace***. The **g** at the end stands for global – on each line, all occurrences of the find string are replaced. Without this option only the first occurrence would be changed.
cat	print files on the standard output
diff	compare files line by line, displaying the differences

Use the above to change all occurrences of *nedles*, *needles*, and *neeedles* to *noodles*.

Check that

```
$ grep -n -E 'ne{0,3}dles' HAYSTACK
```

finds all occurrences of *nedles*, *needles* and *neeedles* in **HAYSTACK**. Then enter

```
$ sed -r -e 's/ne{0,3}dles/noodles/g' HAYSTACK | cat -n
```

The '**| cat -n**' isn't necessary, but adds line numbers to the output to show which lines have been changed.

Note that the regular expression used by **sed** is the same as the one used by **grep**. This is a particularly useful feature of regular expressions.

Finally, send the output to a new file, **NEW_HAYSTACK**.

```
$ sed -r -e 's/ne{0,3}dles/noodles/g' HAYSTACK > NEW_HAYSTACK
```

All your *nedles*, *needles* and *neeedles* should now be *noodles*! You can compare the two files with

```
$ diff HAYSTACK NEW_HAYSTACK
```

This completes the exercises on regular expressions. This is a very brief introduction: they are a powerful and flexible tool.

SHELL SCRIPT EXERCISES

Exercise 30 An example shell script

In your home directory you should see the file, **small.sh** which was downloaded in Exercise 7 Working with files.

You may need to make it executable with

```
$ chmod +x small.sh
```

Now enter

```
$ ./small.sh
```

and see what output appears. Now start editing the file with

```
$ gedit small.sh &
```

and change "Hello" to "Goodbye", save the file and quit. Then run the file again to see what happens.

Exercise 31 Developing your first shell script

You are now going to create your own version of the hello script. In the terminal window type in the following command and an editor window should appear:

```
$ gedit hello &
```

- Type in the first version of the script – you need the two lines in the box above.

```
#!/bin/bash
echo "Hello world"
```

- Save the script - the file will be called "hello"
- In the terminal window enter the command

```
$ chmod +x hello
```

Remember how files have properties associated with them? One such property is whether a file is **executable**. If a file is executable, you can run it (like a '.exe' file in Windows).

```
$ chmod +x file
```

makes file executable by all users (hence a **+x**).

Now let's run the script. It is located in your current directory called '.' so we type:

```
$ ./hello
```

You should see the output

```
Hello world
```

When you have a working version of the script copy your file "**hello**" to a new file "**hello1**" so that if a later version stops working you can revert to a previously working version.

```
$ cp hello hello1
```

Version 2 of the script

```
#!/bin/bash
echo "Hello $1"
```

New concept: you can choose who to greet using a command line argument. This is interpreted by the script with $1.

- Now edit the file so that it looks like the version 2 of the script
- Run the script with a command line argument and check that it works

```
$ ./hello Marge
```

You should see the output

```
Hello Marge
```

Again copy the “hello” file to a new file “hello2” to save a known working version.

Now repeat these steps for the four further examples:

Version 3 of the script

```
#!/bin/bash
for name in $*
do
    echo "Hello $name"
done
```

New concepts:

- **$*** matches all the arguments entered on the command line
- The **for name in list** construct is particularly useful for operating on command line arguments. The variable $name is assigned to each argument in turn and then evaluated when used with **echo**.

After editing the file to make the above changes, run the command

```
$ ./hello Bart Lisa Maggie
```

This should produce the output

```
Hello Bart
Hello Lisa
Hello Maggie
```

Version 4 of the script

```
#!/bin/bash
if [ $# -eq 0 ] ; then
    echo "No one to say hello to"
    exit 1
else
    for name in $*
    do
        echo "Hello $name"
    done
fi
```

New concepts:

- **$#** contains the number of arguments supplied. You can check this by adding the line

  `$ echo $#`

  above the line beginning **if [$# eq 0]**...
- Control flow – the idea that you can choose what happens next based on a condition being met – is introduced using if expression then else fi.
- **exit 1** causes the script to stop

After editing the file to make the above changes, run the command

`$ ./hello`

You should see the output:

```
No one to say hello to
```

Now run the command **for name in list**

`$ ./hello Homer Marge Bart`

which should produce the output

```
Hello Homer
Hello Marge
Hello Bart
```

as in version 3.

Version 5 of the script

```
#!/bin/bash
if [ $# -eq 0 ] ; then
    echo "No one to say hello to"
    echo "Please enter people to greet"
    read people
    echo $people
else
    people=$*
fi
for name in $people
do
    echo "Hello $name"
done
```

New concepts:

- Prompting for input from the script. Instead of stopping when no command line arguments are supplied the script now prompts for input. The name or names supplied are stored in the variable people. At this point the names are stored in one of two places so the else clause is used to move the contents of the command line argument to the variable people.
- The for name in loop now uses people rather than **$*** for this list of names.

Make the above changes and run the command

```
$ ./hello
```

now produces the output

```
No one to say hello to
Please enter people to greet
```

enter the names:

```
Homer Marge Bart
```

and hit return.

you should see

```
Hello Homer
Hello Marge
Hello Bart
```

Running

```
$ ./hello Homer Marge Bart
```

will also produce

```
Hello Homer
Hello Marge
Hello Bart
```

Now for you to try a couple of things.

- Create a test within the for loop to check for a particular name – say Homer – and change the message for this name. Hint: the test might look something like this.

  if [$name == "Homer"]

- Create a small file called Names containing a few names. Now run the following command:

  $ **cat Names | xargs ./hello**

Note how easy it is to change where the input comes from.

In your home directory you should see MS Word file, **Simpsons** which was downloaded in Exercise 7 Working with files, which has an even longer list of names.

Version 6 of the script

The final version of this script introduces a slightly more complicated concept.

```
#!/bin/bash
if [ $# -eq 0 ] ; then
    echo "No one to say hello to"
    echo "Please enter people to greet"
    read people
else
    people=$*
fi
for name in $people
do
    echo "Hello `echo $name | rev`"
done
```

New concept:

The line beginning **echo "Hello** ... has been changed. Looking at this in more detail shows that the there is `(*backquote*) character before echo $name and after rev. This means that echo $name | rev is executed before being displayed on the screen. The effect of this is to pipe the output from echo through the command **rev**, which simply reverses the output.

The *backquote* is found at the top **left hand** side of the keyboard next to 1.

Running the command:

$ **./hello Homer Marge Bart**

will produce the output

Hello remoH

Hello egraM

Hello traB

that is the names have been reversed.

Exercise 32 File manipulation scripts

We are now going to look at some scripts to create and copy files. This script will create several files with similar names.

```
#!/bin/bash
first=$1
last=$2
root=$3

while [ $first -le $last ]; do
    touch $root$first
    let first=$first+1
done#
```

Save this in a file called **create**, give the file execute permission and then use

```
$ ./create 1 4 newfile
```

which would create four files called

```
newfile1 newfile2 newfile3 newfile4
```

A while loop is used to repeatedly create files until the number specified is reached. The line let first=$first+1 adds one to first each time the loop is executed.

Change this script to check that the right number of command line arguments have been given. Print out a message and stop if there aren't three arguments.

The answers are at the end of the exercises in 6 Answers.

Now we will look at a script to copy all files with the same root to another name.

```
#!/bin/bash
let n=1
for file in $1*; do
    cp $1$n $2$n
    let n=$n+1
done
```

So assuming the script is in a file called **mycopy**

```
$ ./mycopy newfile test
```

All files called **newfile1** … **newfilen** will be copied to **test1** … **testn**.

Try it. What does the **$1*** in the line beginning for file … do? Change the script so that the new file name has a dot between the name and the number. [This is very easy!]

[More difficult.] Change the script so that the new file name has a dot after the number and then the date of the form 03Mar13 so that the files look like **test1.27Feb13**

You will need to look at the date command – use **date --help** or **man date** and Version 6 of the hello script.

The answers are at the end of the exercises in 6 Answers.

Note that there is a very useful command rename which will rename one or more files based on a regular expression. It may not be installed by default. If you want to investigate this enter

```
$ sudo apt-get install rename
$ rename --help
```

apt-get install is the command line tool for installing extra command or packages. There is a brief discussion of **sudo** in the next session.

Concluding Remarks:

These exercises can only scratch the surface of what can be done with the command line and shell scripts. Don't expect everything you've seen today to sink in. Working with command lines takes some getting used to and the best way to learn is by lots of doing. For now, simply take away the ideas of what this environment is capable of.

This completes the exercises for Session Three

Session Four.

5 Using remote computers

Sometimes you will need to access a remote computer. This may be from home to a Department system, or from a desktop computer to a more powerful computer.

Exercise 33 **Setting up access to IT Services Linux system**

First you need to set up access to the IT services Linux system. An overview of this service is available here: https://help.it.ox.ac.uk/use-linux-service

A direct link to activate your Linux shell account is here: https://register.it.ox.ac.uk/accman/shell

Click on this link and you should now see the screen to activate the account.

If you are asked to choose a shell, make sure you choose **/bin/bash**.

Exercise 34 **Using ssh**

Now that the account has been enabled, you can log on.

Command	Description
`ssh`	ssh connects to a different Linux/Unix system.

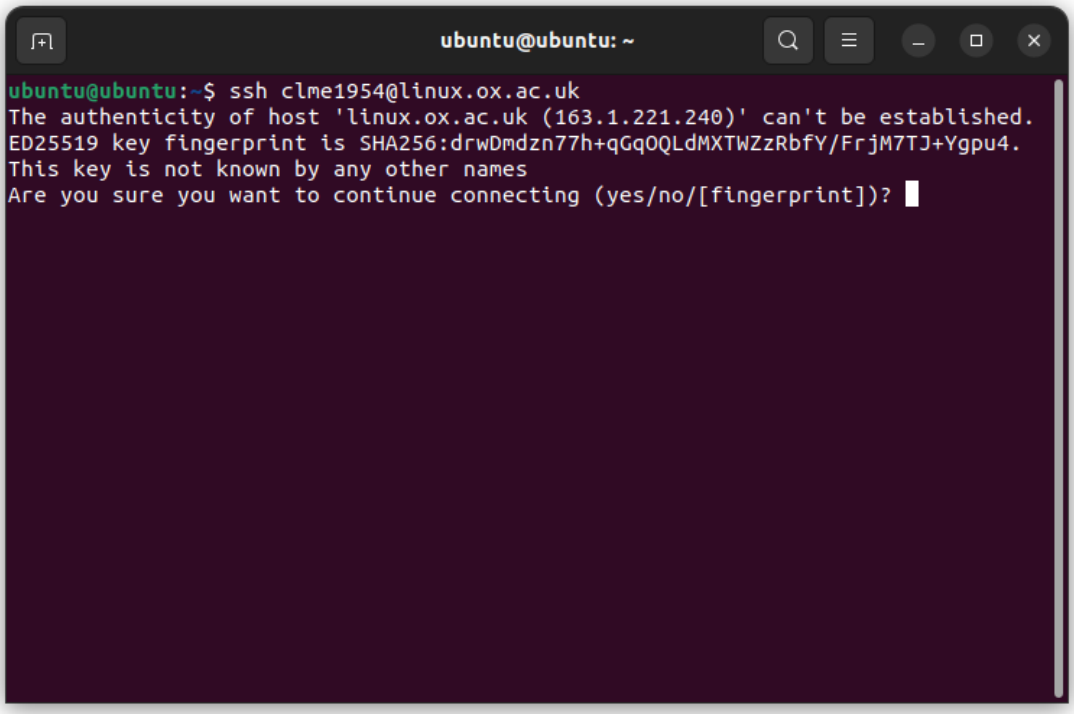
Use

```
$ ssh SSO@linux.ox.ac.uk
```

at the login prompt enter your SSO and password. So if your SSO is **coll1234** you would type in

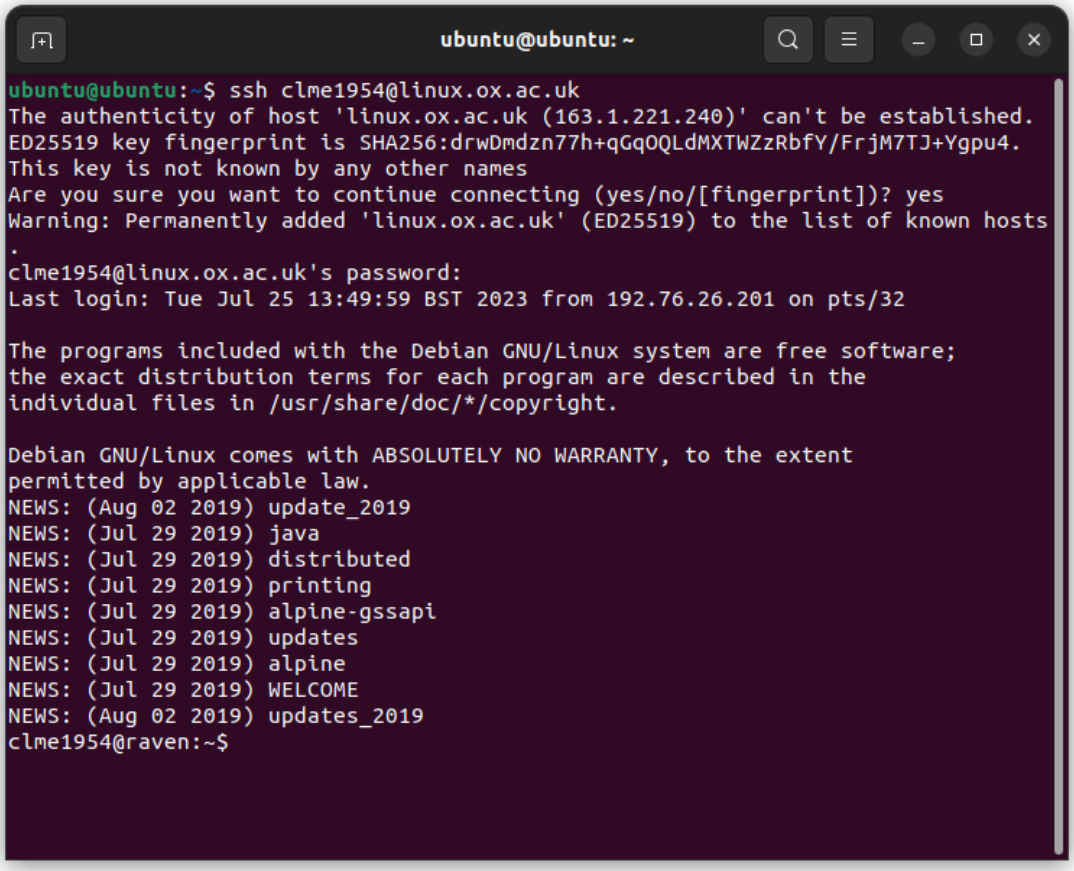
```
$ ssh coll1234@linux.ox.ac.uk
```

You should see a screen like this:



Accept the fingerprint by typing ‘yes’.

After this you will be prompted for your password and eventually you will get a command prompt:



The commands that you have used on the live Ubuntu, will also work on this system, although it is running a different Linux distribution. It is running Debian.

Use the **cd**, **wget** and **tar** commands from Exercise 14 to download the files to this server.

```
$ cd
$ wget https://skills.it.ox.ac.uk/files/CP001_Files_Linux_Intro_LinuxFiles.tgz
$ tar -xvzf CP001_Files_Linux_Intro_LinuxFiles.tgz
```

Make sure that commands are behaving in the same way by using examples from previous exercises.

Command	Description
ssh	ssh connects to a different Linux/Unix system. Options used: **-X** Allow X11 forwarding.

It is possible – and sometimes necessary – to open a graphical application on the remote system. To do this logout using

```
$ exit
```

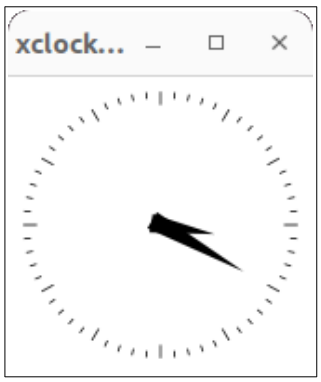
and log back in again from your local Ubuntu desktop with

```
$ ssh -X SSO@linux.ox.ac.uk
```

On *linux.ox.ac.uk* enter

```
$ xclock &
```

You should see a window like this appear:



Exercise 35 **Copying files between systems**

As well as logging on to remote systems, it is often necessary to copy files and directories from one system to another.

Command	Description
scp	Copy files and directories between systems.

You will need two terminal windows open.

- The local live Ubuntu
- A remote session on *linux.ox.ac.uk*

On *linux.ox.ac.uk*, create a new directory to store the files and directories you will be copying.

```
$ cd
$ mkdir Copies
```

Use **ls Copies** to make sure it is empty.

Now in the local live Ubuntu terminal type in

```
$ scp HAYSTACK SSO@linux.ox.ac.uk:~/Copies/.
```

Make sure you type the command exactly as it appears.

The command is quite complex so each component is explained in detail here:

scp command	Description
scp	Copy files and directories between systems.
HAYSTACK	Name of file to be copied.
SSO	Username on remote system. Replace by your single sign-on.
@	Location indicator.
linux.ox.ac.uk	Name of remote system
:	Separator between remote system name and location of file.
~	A short cut which refers to your home directory. Useful if you don't know the path to your home directory.
/Copies/	The directory where the file will be stored
.	"." indicates that the file will be given the same name as the original.

Now in the *linux.ox.ac.uk* terminal use

```
$ ls Copies
```

to make sure the file has been copied.

See if you can copy HAYSTACK from the live Ubuntu session to the same location but with a different name.

See if you can use wild cards to copy all the files beginning foo2 from the Wildcards directory to the same location and with the same name.

Use **ls Copies** on *linux.ox.ac.uk* to make sure the files have copied successfully.

The answers are at the end of the exercises in 6 Answers.

Finally, remember that **wget** will download datasets available from the web.

Exercise 36 **Copying directories between systems**

scp command	**Description**
scp	Copy files and directories between systems. Options used: **-r** recursively copy the contents of a directory
Files	Name of directory to be copied

On the local live Ubuntu terminal enter

```
$ cd
$ scp -r Files SSO@linux.ox.ac.uk:~/Copies/.
```

On the *linux.ox.ac.uk* terminal check that the directory and contents have been copied.

Exercise 37 **Managing sessions on remote systems**

Command	**Description**
screen	Connect and disconnect from a session, possibly from multiple locations, and allow long-running processes to persist without an active shell session. Options used: **-r** reattach a screen session **-list** display all screen sessions

In the *linux.ox.ac.uk* terminal use

```
$ screen
```

to start the screen session. You should see a window like this:

Press return and you should see the standard prompt. Now use wget to get the script you will be using

```
$ cd
$ wget https://skills.it.ox.ac.uk/files/CP001_Files_Linux_Intro_Demos.tgz
```

to download the files and then

```
$ tar -xvzf CP001_Files_Linux_Intro_Demos.tgz
```

Permanently change the permissions, so that the script can be executed.

```
$ chmod +x ./my_date.sh
```

Have a look at the **my_date.sh** script to see what it does. Don't worry if you can't understand every line.

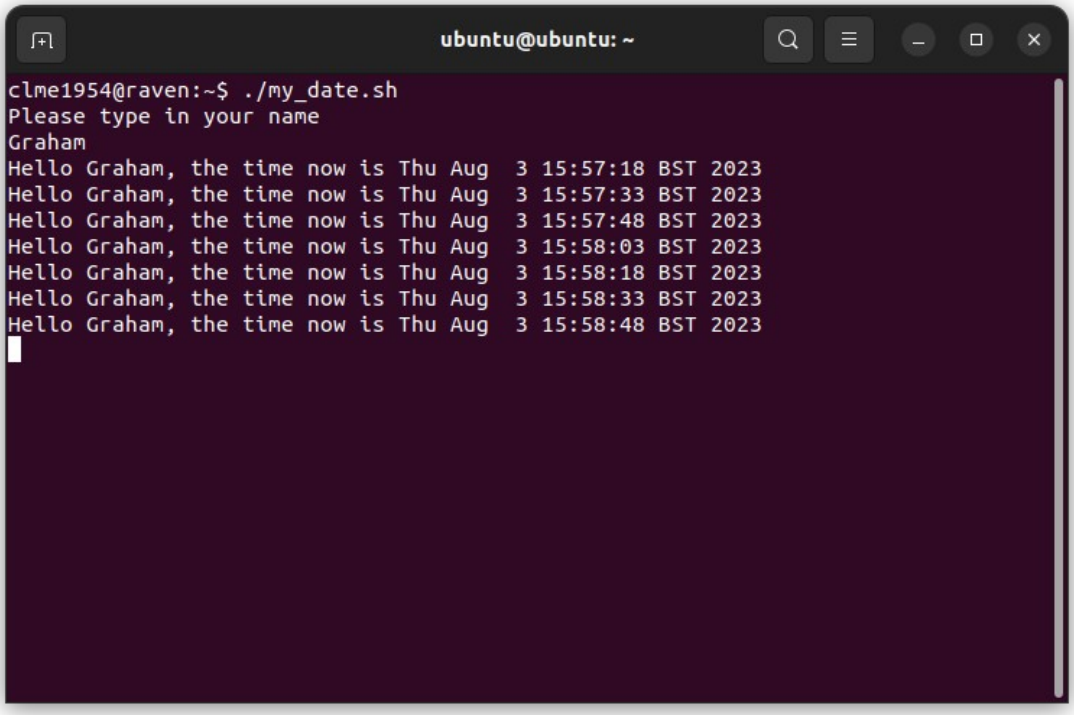
```
$ cat my_date.sh
```

```
#!/bin/bash
# Ask for name.
# Repeatedly output the name and date, waiting 15 seconds.
# Terminate after 15 minutes.
echo "Please type in your name"
read name
start=`date +%s`
now=$start
while (( (( $start + 900 )) > $now )) ;
do
     echo "Hello $name, the time now is `date`"
     sleep 15
     (( now += 15 ))
done
```

The script displays the date and time every 15 seconds for 15 minutes. Run the script interactively to watch what happens.

```
$ ./my_date.sh
```

Watch the output appearing for a minute or so. You should see something like this:



Now we are going to detach from the screen session so that the script will continue running but we will logout of linux.ox.ac.uk. To detach from the screen session use

Ctrl-a d

That is, hold down Ctrl and a, then press D. You should see a message

[detached from XXXX.pts-XX.raven]

where each X is a digit.

You can check what screens you have with

```
$ screen -list
```

and should see something like this:

```
There is a screen on:
6144.pts-30.raven (01/23/17 17:29:58) (Detached)
1 Socket in /var/run/screen/S-hutchins.
```

Logout from *linux.ox.ac.uk*

```
$ exit
```

Perhaps from another desktop (swap with your neighbour!), log on again to *linux.ox.ac.uk*

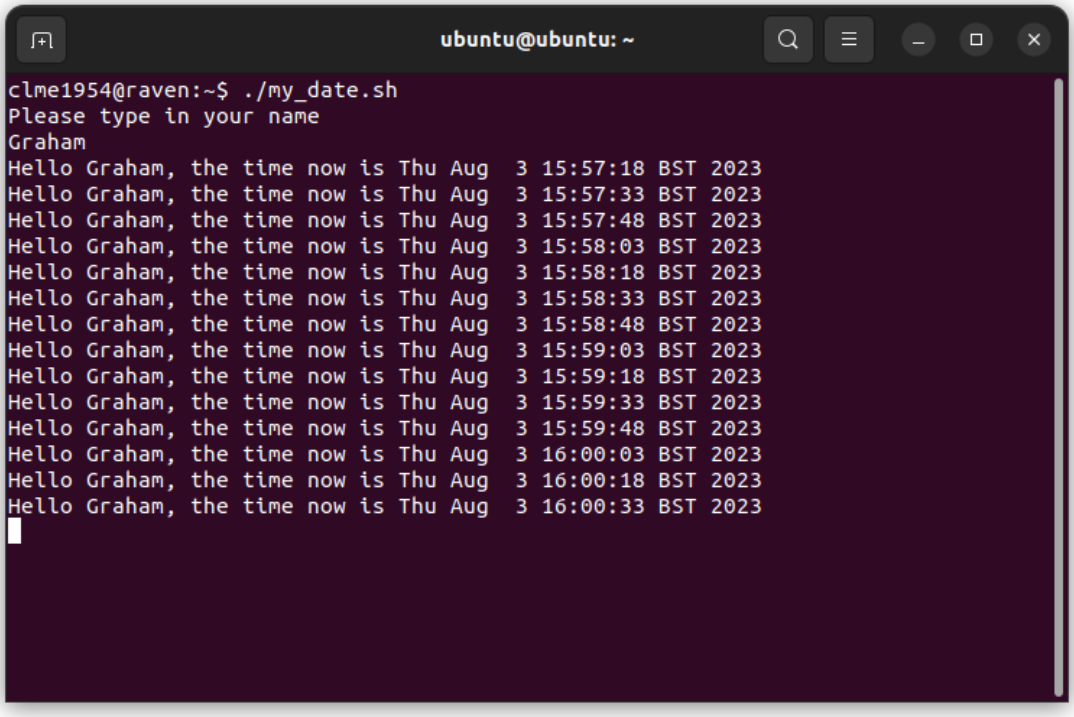
```
$ ssh SSO@linux.ox.ac.uk
```

replacing SSO by your single sign-on.

Reattach the screen

```
$ screen -r
```

and you should see the list of times with no interruption.



This is a slightly contrived example. In general output from long running jobs is best saved in a file rather than displayed on the screen, as there limits to the amount of data that a terminal displays. However, the technique for detaching the screen and logging out is the same.

Concluding Remarks:

Accessing remote computers is a common requirement for Linux users. On Windows, a small application called **`PuTTY`** provides a secure means for logging on to a Linux system. On Macs, use the terminal window.

This completes the exercises for Session Four

6 Answers

2 Getting started

Exercise 3 The home area

Try Dock | Files

Exercise 4 Getting to know the desktop

The active application is highlighted in a lighter coloured square. For each open application an orange dot appears.

Exercise 4 Getting to know the desktop Keyboard shortcuts

To create keyboard shortcuts to switch between desktops

1. Open **System Settings | Keyboard**
2. Click on **Keyboard Shortcuts | View and Customize Shortcuts**
3. Click on **Navigation**
4. Scroll down until you find **Switch to workspace 1** and select it
5. Hold down the **Ctrl** key and press **F1** (on the top left hand side of the keyboard)
6. Now right click on **Switch to workspace 2**
7. Set this to **Ctrl** and **F2**
8. Do this for workspace 3 and workspace 4...
9. Click on **X** to save the changes.

If you now hold down the Ctrl key and press F2 you should switch from Desktop 1 to Desktop 2

Exercise 13 File and directory manipulation You can also create files in `/tmp`.

3 Command line exercises

Exercise 16 Help commands Finding a drawing application

This is straightforward:

```
$ apropros draw
```

lists several possible matches. The most useful is

```
lodraw (1) - OpenOffice.org office suite
```

If you enter **lodraw &** on the command line the application will start.

Exercise 17 File and directory names Reading a file called *star

```
$ cat \*star
$ cat "*star"
```

are both possibilities. A \ (backslash) is an escape character which stops the star being interpreted as a wildcard. Surrounding file names that contain special characters with quotation marks often work. There may well be other ways to do this.

Changing to a directory called Open This.

```
$ cd ..
$ cd Open\ This
```

but again there will be alternatives.

Reading a file called `-ReadMe`. Again there are lots of possibilities

```
$ cat ./-ReadMe
$ cat -- -ReadMe
```

both work. In the first suggestion (**`cat ../-ReadMe`**) using a relative path overcomes the problem of the leading dash; in the second (**`cat -- -ReadMe`**) the cat command is given an extra option of two dashes which alerts it to the fact that the filename starts with a dash. Both these options work with the more and less commands too. Similarly to delete the file use

```
$ rm -- -DeleteMe
$ rm ./-DeleteMe
```

Exercise 19 Using wildcards to match filenames Wildcards

ls foo? matches

```
foo1  foo2
```

ls foo2* matches

```
foo2  foo20  foo2bar
```

ls foo[1-2] matches

```
foo1  foo2
```

To match just the files foo20 and foo2bar use

```
$ ls foo2?*
```

To use wc -l just to display the files with a 1 in their name.

```
$ wc -l *1*
```

Exercise 20 Searching and sorting

Use **grep -i melon *file*** to find all occurrences of melon whatever the case.

Use **sort -r fruit veg** to reverse the order of the sort.

Exercise 22 Finding the largest file

Use **du -sk /usr/bin/s* | sort -n | tail -5** to find the 5 largest files beginning with 's'.

Exercise 23 Merge information from different files

```
$ awk '{print "A", $2, "eats"}' creatures | paste -d" " - veg
```

4 Editors, regular expressions, and shell scripts

Exercise 26 Simple regular expressions

```
$ grep -n -E ^needles$ HAYSTACK
```

will find all lines with just one occurrence of the word needles with no white space either before or after. We'll look at how to match white space in the next set of exercises.

Exercise 27 Regular expressions with spaces

No matches are found, because this search is looking for matches where needles has characters both before and after it - *hayneedleshay* for example - and there are no instances of this in HAYSTACK.

Exercise 32 File manipulation scripts

```
$ cp $1$n $2$n.`date +%d%b%y`
```

5 Using remote computers

Exercise 35 Copying files between systems

```
$ scp HAYSTACK sso@linux.ox.ac.uk:~/Copies/HAYSTACK2
$ scp foo2* sso@linux.ox.ac.uk:~/Copies/.
```

Further information

Getting extra help

Course Clinics

The IT Learning Centre offers bookable clinics where you can get pre- or post-course advice. Contact us using courses@it.ox.ac.uk.

Study Videos from LinkedIn Learning

On our website, you will find our collection of self-service courses and resources. This includes providing LinkedIn Learning video-based courses free to all members of the University. Visit skills.it.ox.ac.uk/linkedin-learning and sign in with your Single Sign-On (SSO) credentials.

Some courses recommend pre- and/or post-course activities to support your learning. You can watch the online videos anywhere, anytime, and even download them onto a tablet or smartphone for off line viewing.

About the IT Learning Portfolio online

Many of the resources used in the IT Learning Centre courses and workshops are made available as Open Educational Resources (OER) via our Portfolio website at skills.it.ox.ac.uk/it-learning-portfolio.

About the IT Learning Centre

The IT Learning Centre delivers over 100 IT-related teacher-led courses, which are provided in our teaching rooms and online, and we give you access to thousands of on-line self-service courses through powered by LinkedIn Learning.

Our team of teachers have backgrounds in academia, research, business and education and are supported by other experts from around the University and beyond.

Our courses are open to all members of the University at a small charge. Where resources allow, we can deliver closed courses to departments and colleges, which can be more cost-effective than signing up individually. We can also customize courses to suit your needs.

Our fully equipped suite of seven teaching and training rooms are usually available for hire for your own events and courses.

For more information, contact us at courses@it.ox.ac.uk.

About IT Customer Services

The IT Learning Centre is part of the Customer Services Group. The group provides the main user support services for the department, assisting all staff and students within the University as well as retired staff and other users of University IT services. It supports all the services offered by IT Services plus general IT support queries from any user, working in collaboration with local IT support units.

The Customer Services Group also offers a data back-up service; an online shop; and a computer maintenance scheme. Customer Services is further responsible for desktop computing services – for staff and in public/shared areas – throughout UAS and the Bodleian Libraries.

Computer platforms: Linux: A comprehensive introduction

Graham Addis

graham.addis@it.ox.ac.uk

IT Services Linux Courses

Today's course is divided into four parts each consisting of:

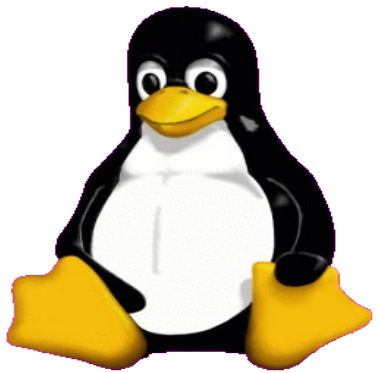
- A short presentation.
- Exercises.

The four parts are

- Introducing the Linux desktop.
- Simple use of the command line.
- Further use of the command line and shell scripting.
- Using remote computers and managing your own computer.

Session 1

Introducing the Linux desktop



Session 1

- What is Linux?
  - Open source and why it is important.
  - Where it came from and how it is made.
- What is Linux used for?
- How to get Linux.
- Linux office applications.
- Your questions and a practical session.

How Linux changed things

- Linux does not come from a single large corporation.
- It offers an alternative approach.
  - Freedom of choice.
  - Freedom to understand and change.
  - Software written with quality rather than profit as a goal.
- Competition and alternative approaches benefit users and consumers.
  - Software developments: firefox for example.
  - Increased awareness of open standards by Microsoft.
  - Easier to work using different systems.

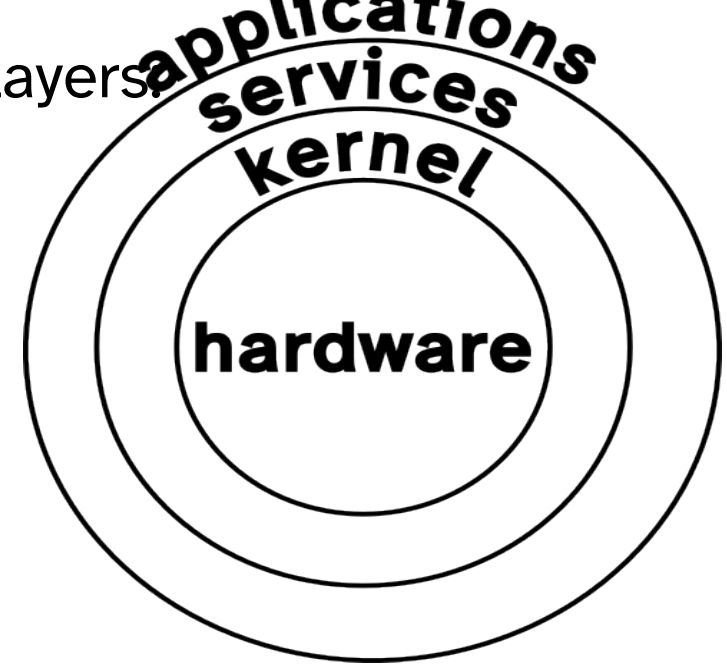
What is Linux?

Linux is an operating system designed by Linus Torvalds in the early 1990s.

An operating system protects the user from the hardware and vice versa.

An operating system has several parts/layers:

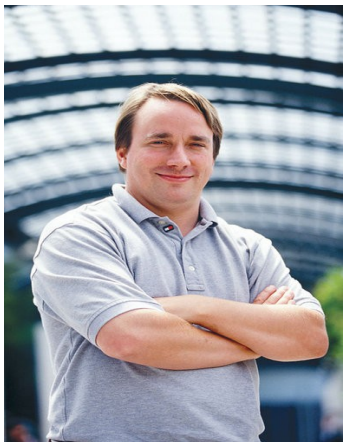
- kernel
- system services
- applications



Other Operating Systems include

- Windows (7, Vista, XP, 2000, 95)
- MacOS
- Unix

Who makes Linux?



- Linus Torvalds created the first Linux kernel in August 1991:

"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 brewingsince april, and is starting to get ready."

- Text of original email from Linus to the comp.os.unix newsgroup sent on 25 August 1991.

How is Linux developed?

Linux is maintained and developed both by businesses and private individuals.

- Hundreds are involved in work on the Linux kernel.
- Thousands work on the applications which come with Linux distributions.

It is known as open source software.

Linux or GNU/Linux?

- Linux is the kernel.
- GNU is the project to create a Unix-like operating system.

GNU stands for GNU's Not Unix (terrible Unix joke).

What is Linux used for?

Linux was originally designed for desktop use but this did not make a significant impact on the market.

Linux did quickly gain a large following for

- Servers – for example, web servers
- Internet services (DNS, routers etc)
- Programming
- High performance computing

Significantly improved user interface and economic advantages mean that Linux is gaining popularity on the desktop.

Getting Linux

The easiest way to get Linux is to download or to buy a “distro” or distribution.

A distribution = a kernel + applications + installation software+ support + documentation.

Typical applications are

- Office software
- Databases
- Programming languages and tools
- Web browsers
- Email readers
- Internet services (www, DNS, NIS, firewalls)
- Games

Linux distros

The two big distros are RedHat/Fedora and Suse:

Debian

Mint

Mepis

Slackware

Mageia

arch

Ubuntu

For this course we will be using the Ubuntu distro

This version of Ubuntu is a special kind of Linux known as a “Live” system and has two significant features:

- It runs from USB.
- The hard drive is not touched.

It's safe to run this on a Windows system without installing or changing anything.

Ubuntu is a Debian-based version of Linux, providing a simple desktop and straightforward installation.
We are using Ubuntu 22.04 LTS (Long Term Support, Jammy Jellyfish)

Creating a bootable USB

- Step 1:
  - Download suitable bootable image
    https://ubuntu.com/download/desktop
- Step 2:
  - Download tool for creating bootable USBs from a disk image (.iso)
  - Lots of options available using web searches
  - We use Ventoy
    https://www.ventoy.net/
    which provides the option of storing multiple boot images on a single USB to be selected at boot time

Linux and Windows

- You can install Linux on your PC without removing Windows (or any other operating system).
  - At boot, choose which to use.
  - Could use emulation software (VMWare, wine, WSL).
- Can share devices using samba.
- Linux will read most Windows file systems.
- Windows won't support Linux file systems.

Office applications

Desktop Environments

- Desktop systems in Linux come with a variety of desktop environments.
- Provide an easy to use graphical use interface or GUI based on the X window system.
- Choice: KDE or Gnome are the most widely available.
- Generally they provide very similar functionality.
- Competition has produced two very usable solutions.

LibreOffice and Microsoft #1

- LibreOffice runs on Windows just as well as on Linux and Unix.
- LibreOffice supports all Microsoft data formats.
- LibreOffice offers almost all the functionality that Microsoft office does (a few fancy bits might be missing).
- LibreOffice was far more standardised than Microsoft Office
  but the latest version of Microsoft Office does allow you to save files in an open standard (XML) format.

LibreOffice and Microsoft #2

- Microsoft Office products and their LibreOffice equivalents:

Microsoft		LibreOffice
Word	<=>	Writer
Excel	<=>	Calc
PowerPoint	<=>	Impress
Paint	<=>	Draw

Getting help

Getting help with Linux questions can be difficult so it helps to know where to look.
The sites I use often are:

- StackExchange – a network of community managed sites providing expert answers to questions.
- Google – particularly when I have an error message I don't understand.

The command line

Why use the command line?

- How does the command line work?
- What is a shell?
- Some simple commands.
- Files and directories.
- Hints and tips for file names.
- Getting help.

The command line

- A graphical user interface (GUI) is available in both Windows and Linux.
- The command line is often unfamiliar to Windows users.
- Compare Windows and Linux when reading a PDF file or starting a browser.
- Either double click on the icon or enter the command

```
evince file.pdf &
```

```
firefox &
```

How commands work

A command at its simplest is of the form:

`command`

Often commands require an argument such as a file name

`command file`

Some examples are

`date` [to display the time and date]

`gedit newfile` [start the gedit editor to change the file newfile]

Speeding things up

Surely all that typing can't be right!
How can we speed things up?

Filename and command completion:

- <tab> key completes commands and filenames

What else can we do to save typing?

- arrow keys allow us to:
  - recall previous commands
  - change previous commands

Upper and lower case

Case sensitivity

- Linux commands and filenames are case sensitive:
- BIG is different from big.
- Almost all commands use lower case.
- In Windows case is not significant. If you save a file as big.doc, type in BIG.DOC and Word will find it.
- However, if you are working with files in both Linux and Windows you need to take care: Windows will see BIG.DOC and big.doc but can only use BIG.DOC.

Guidelines for file names: don't include: spaces, *, ?, -, / or \

Two things Linux does better

… in my opinion!

Multiple desktops

- reduce clutter
- organise work logically

Flexibility

- better organised desktop
- powerful command line
- choice of many applications

Exercises for Session 1

The aims of these exercises are:

- To look at the desktop.
- To explore some components of LibreOffice.
- To take a first look at the command line.
- Please feel free to:
  - Ignore desktop exercises which are not relevant your work.
  - Try to do the tasks in LibreOffice that you do with Microsoft Office.

Session 2

Using the command line

Session 2

We're now going to look in more detail at the shell and the command line

- What is the shell?
- Navigation – how to move around.
- More advanced use of commands.
- Pipes or how to build your own commands.

Shells

A shell sits between the user and the kernel.

There are several ways of using shells:

- Graphical desktops (indirectly).
- Running applications (indirectly).
- Command line interface (directly).
- Scripts (directly).

- There are different shells available for Linux
- The default shell in Linux is called BASH.

Unix/Linux commands

We can interact directly with a shell with a shell prompt aka the Command Line Interface

The UNIX Philosophy: commands should

- do only one small task;
- do it quickly;
- do it quietly and don't ask for confirmation.

If a new requirement arises start again - don't add features to an existing command.

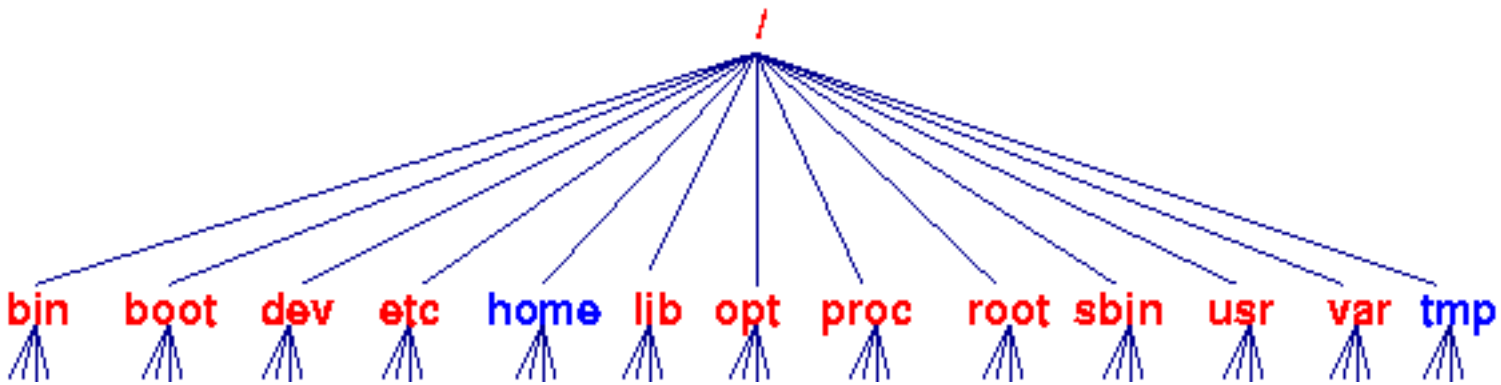
Advantages and disadvantages with this approach:

- Many commands so each is small and simple.
- A lot to remember.

Files and directories

We can use commands to explore the way Linux locates and stores files.

A Linux system looks a bit like a tree. Here's and example of the top (or bottom) layer:



A digression on paths

Paths describe the location of a file in the filesystem. There are three sorts of paths:

- simple a file or directory name

  `newfile`

- relative a reference to a file or directory from the current directory

  `../tmp/newfile`

- absolute a reference that will work anywhere

  `/home/ubuntu/Desktop/tmp/newfile`

Commands and navigation

The file browser moves you around the system and manipulates your files.

Can use commands instead

cd = change directory

ls = list files

cp = copy

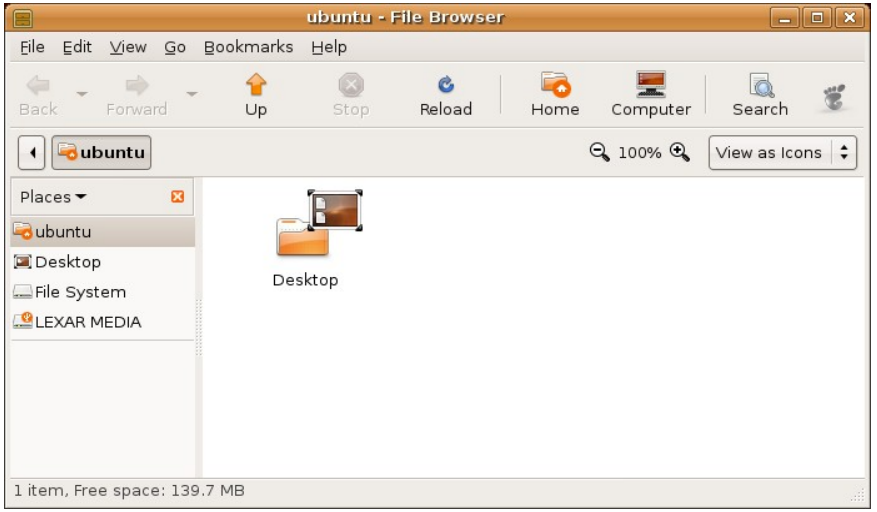
mv = move

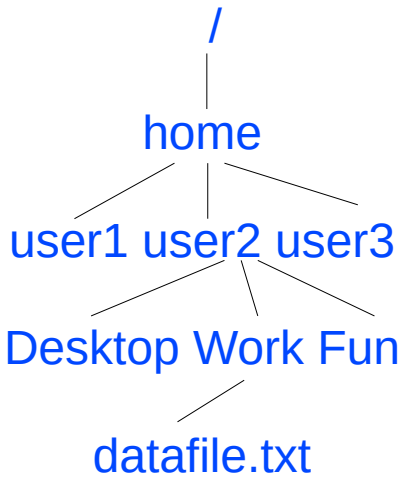
pwd = where am I?

mkdir = make new directory

rm = remove file

rmdir = remove directory





Learn to navigate up and down directories

Finally ... some commands

Orientation and navigation or where am I and where do I want to go?

- pwd [print working directory]
- cd directory [change directory]
- cd [go to your home directory]
- cd .. [go up one directory]

… and more …

File and directory manipulation

- **`ls`** list files in a directory
  - **`ls -a`** list all files in a directory
  - **`ls -l`** display a long listing
  - **`ls -la`** display a long listing of all files
- **`mkdir directory`** create a directory
- **`rmdir directory`** delete a directory
- **`cp file1 file2`** copy file1 to file2
- **`rm file`** delete a file
- **`mv file1 file2`** move file1 to file2

... and more

There are several ways of looking at the contents of a file:

`cat file`	look at the contents of one or more files.
`more file`	look at the contents of a file at the end of each screen full.
`less file`	like more only with more options.

Some other commands

`du -sk file`	find out the size of a file
`sort`	sort the contents of a file, line by line

Other commonly used commands

`man` get help with a command

`whatis` short description of a command

`lp` print a file

Some more powerful commands

`find` find files matching pattern

`grep` search for a pattern in a file

`wc` report the number of characters/words/lines

```
find . -mtime -5 -exec grep -i statistics {} \; -ls
```

We will be looking at more complicated commands next session

What is a file?

In Windows, files:

- Usually include formatting and application information.
- Are used by only one application.
- Have a suffix (part of the name after the .) that determines what sort of file it is.

In Linux, files:

- Are often plain text.
- May be manipulated by many applications.
- Don't need a suffix to determine their type.

Wild cards and globbing

This slide could be called file name expansion!

The most commonly used special characters to represent parts of a filename are:

`*`	matches none or more characters
`?`	matches a single character
`[ ]`	matches any characters in a given range
`[! ]`	matches any characters not in a given range

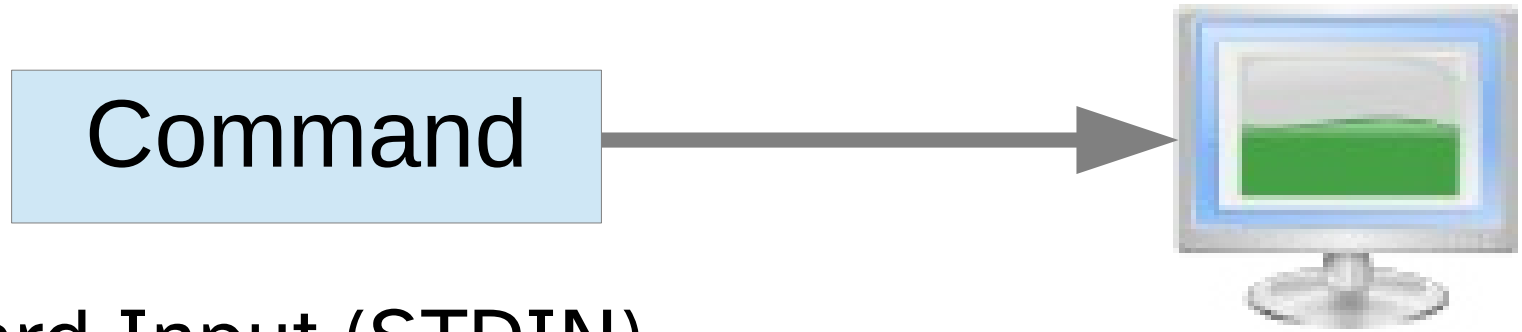
So to list all files whose name ends with .txt use

```
ls *.txt
```

Standard input and output

Linux commands make use of the concepts of

Standard Output (STDOUT)

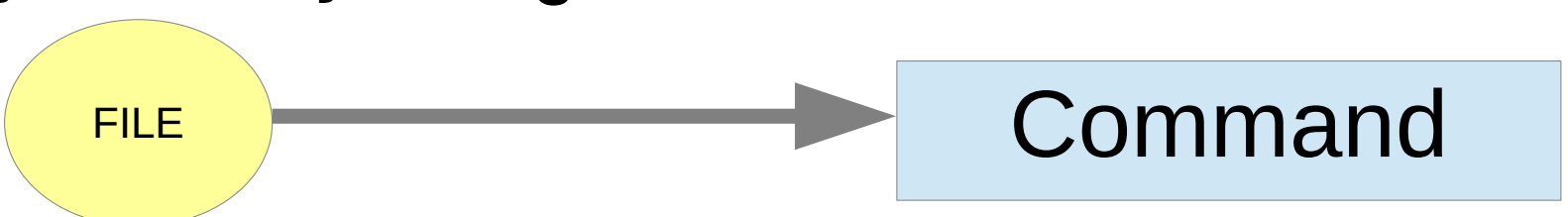


Standard Input (STDIN)

Redirection

- Linux can redirect standard output to targets other than the display such as files by using the > character.

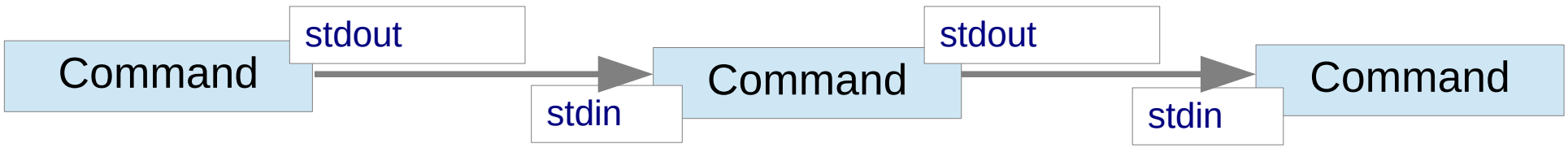
Or take input from sources other than the keyboard by using the < character.



Pipes

If commands use STDIN and STDOUT, is it possible to connect the STDOUT from one command to the STDIN of another?

YES – use pipes: **|**



A simple example

```
du -sk /usr/bin/* | sort -n | tail -5
```

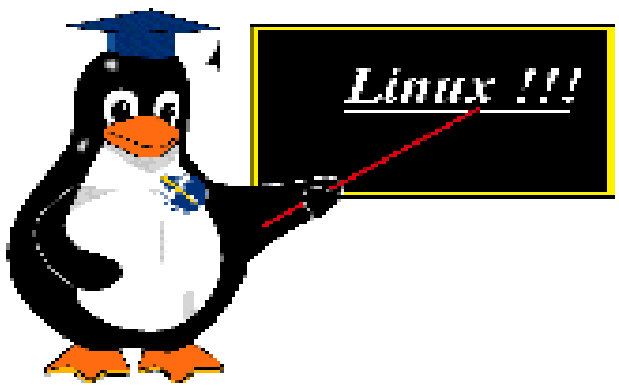
displays the five largest files in the /usr/bin directory

Exercises for Session 2

- Using the command line to find your way around.
- Using commands to manipulate files.
- Getting help.
- Using wildcards to match files and directories.
- Searching and sorting.
- Pipes and redirection.
- Building your own commands.

Session 3

Editors,
regular expressions,
shell scripts



Text Editors

Many files in Linux are in plain text format:

- Configuration files.
- Shell programs.
- Manual pages (although sometimes stored in compressed format).

So we need a utility which to let us change text files.

- emacs, xemacs
- vi, vim
- gedit, pico, nano

Common to run an editor in the background [demo]

Controlling the command line

- Use **&** to run commands in the background.
- What happens if you forget?
- There are control-key commands which allow you to to manipulate the command line:

CTRL-C interrupt a running program.

CTRL-D send an end of file, ending text input for most Linux/Unix programs.

CTRL-Z suspend a running program.

Some editors

`vi` – visual editor, now often known as vim (vi improved).

- Very fast, supports regular expressions and a relatively small but powerful command set.
- Almost all beginners find it difficult to learn.

GNU **`emacs`** - “extensible, customizable, self-documenting real-time display editor”.

- It is an “ integrated environment”: so can do much else apart from editing.
- It is aware of what sort of file you are editing, so particularly useful for editing programs and scripts.

gedit

GNOME Editor is widely available in Linux

Simple text editor for the GNOME desktop.

- Intuitive – similar to notepad.
- Supports tabs so several files can be edited at once.
- Context aware – highlights syntax.

But if you are familiar with another editor do use that.

Regular expressions

What is a regular expression?

- Searches for string patterns within text files.
- Uses metacharacters to extend a pattern.

Examples:

```
grep green fruit
```

- to search for green anywhere in the file, fruit.

```
grep ^green fruit
```

- to search for green at the beginning of lines only.

Extended regular expressions

Extended regular expressions allow a much richer pattern to be specified.

Many powerful features

- Match white space [space, tab]
- Match word boundaries
- Digits only
- Ranges

and many more.

Some limitations

Extended regular expressions can be used with many commands.

However, the way these are formed varies with different applications.

The most common are variations are

- Command line tools (grep, sed and so on)
- Perl
- C#
- PCRE – Perl Compatible Regular Expressions

Taking control of the shell

So far we have looked at single commands:

- For example we have run simple commands to do one thing, e.g. cd, mkdir, grep and so on.

More typically:

- Jobs are more complex than the ones we've seen.
- Need recipes of multiple commands.

So we need to control the flow of :

- Commands.
- Data.

Scripts

- Rather than typing commands on the command line it can be more convenient to put the commands into a single file. This is called a script.
- A script can be run more simply than a series of commands which you have to get exactly right every time.

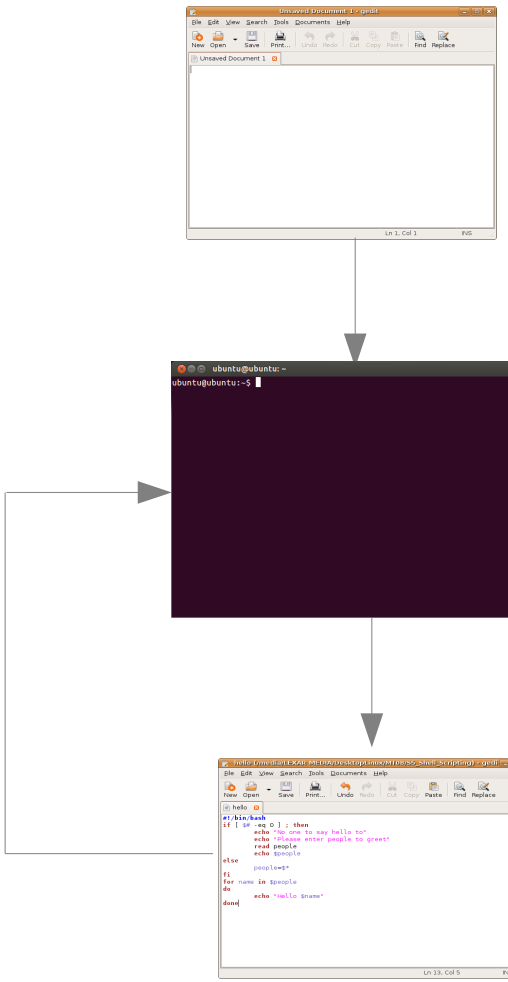
Writing scripts

- It's usual to write some of the script, try it out, save a working version and add to it. This cycle is repeated.
- Rather than closing your editor every time you want to try the script, run it in the background with “&”
- For example

```
gedit &

emacs &
```

- You keep control of the command line



When to script

- A lot of what you want to do is covered by one or more Linux commands anyway.
- The problem is conceptually simple.
- It doesn't have to be blindingly fast.
- It's something you have to do a lot.
- It's something you want to be automated.

When not to script

- Your problem can't be broken down into existing UNIX commands.
- You need lots of data/program flow control.
- It has to be fast.
- It's very numerical.
- It's a one off job.

Executables

- In Windows **`.exe`** files are executable – in Linux the conventions are much looser.
- In Linux if you write a script you have to manually give people permission to run it.
- The **`chmod`** command changes permissions on a file e.g.:
  `chmod +x file`
  allows your script to be executed.

Exercises for Session 3

- Regular expressions
- Use the gedit text editor
- Write some shell scripts

Session 4
Using computers remotely

Session 4

- Using computers remotely.
- We will also look briefly at package management and system administration.
- Exercises.

Using remote systems

- Often the computer we are sitting at is not the only one we need to use.
  - More powerful
  - Shared resources
  - Additional software
- Set up your Oxford Single Sign-On (SSO) to access the IT services linux system.

Accessing systems and copying data

- Use

  `ssh` to access remote computers.

  `scp` to transfer data.

Managing jobs on remote systems

- Why?
- Jobs may last longer than a few minutes.
- Don’t want to keep a session on a remote system: lose connection, lose job.
- Use
  `screen`
  to detach from a session and logout.

Managing systems

Managing systems

- The following slides discuss briefly techniques for
- Managing Ubuntu systems
  - Using a privileged account
  - Maintaining a system: updates
- Installing new software: different methods
  - Using the Software Center
  - Using the command line
  - The difficult way!

Why manage packages?

- Complexity of current systems.
- Dependencies – new applications may affect existing software.
- Each system may be configured slightly differently to suit its own environment.
- Enormous range of hardware that must be supported.
- We don't want – if possible – to concern ourselves with these details when we need to install some new packages.

Additionally, as with Windows a mechanism is needed to keep the system up-to-date.

Ubuntu Software Updater

- Provides graphical access to available packages.
- Can also be used to update the system.
- Unfortunately not working correctly on this Live Ubuntu release.
- Will work fine on installed version.

Other ways of installing software

The quick way:

- Using the command line

  `apt-get install package`

The hard way:

- Software is not always bundled in a format that APT or RPM can use.
- It will come in a single .tgz or .tar.gz or .bz2 file.
- Also known as a tarball.
- Requires rather more knowledge of the system.

Managing your system

- Until now we have used the the system to change our files but not made any system changes.
- Unlike Windows, Linux has a clear and rigid distinction between using a system and making changes to a system, such as installing new software.
- So to make changes a special account is needed.

The root account

- In most Linux systems you will have your own username and password.
- Standard user accounts are set up so that you can change your own home files but not any system files.
- To install new software we need an account that can change files anywhere on the system.
- This account is known as the root account.

Using sudo

- Some systems use a different approach to administering system.
- Use **`sudo`** + command when carrying out privileged actions e.g.:
  `sudo apt-get install bluefish`
  will install the bluefish editor.
- Normally you will be prompted for a password; this will be remembered by the system for about 15 minutes.

Exercises for session 4

- We will use the IT services linux service to:

- Log in remotely.
- Copy files between systems.
- Demonstrate a technique to manage jobs on other computers.

graham.addis@it.ox.ac.uk