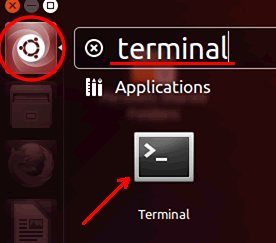
# Section 1: Terminal windows and text consoles

To launch a terminal window in Ubuntu Linux:  
  
Select the Terminal icon from the launcher sidebar.

Or click in the top left corner on the search icon and then type ‘terminal’.

You can launch multiple terminals if you want. Each runs an independent command line interpreter (“shell”).  
  
To launch a text console in PWF Linux, select Applications→ Unix Shell→ Gnome Terminal. This can be done as often as you want for as many text consoles as you want. Each runs an independent command line interpreter (“shell”).

Alternatively you can dispense with the graphical environment entirely and go back to the “good old days” of text-mode Unix. Press [Ctrl]+[Alt]+[F2] to get a pure text login console. If we enter our PWF login and password again we can log in here too. Note that we can be logged in both through the graphical interface and the text interface(s) simultaneously. In fact, we could be logged in to one interface and our neighbour could log in through another. Unix is a fully multi-user operating system. Identical interfaces are available by using [F3], [F4], [F5], or [F6] in place of [F2]. [F1] has a text console that tends to be more colourful.

|  |  |
| --- | --- |
| **!** | When you switch between consoles there is often a several second black screen as the console switches over. Please be patient and let the switch complete before you start switching back again. If you try to switch while the consoles are in mid-switch you can jam things badly. |

Using [Ctrl]+[Alt]+[F7] returns you to the graphical interface.

You will be faced with a set of text that looks like this in the graphical terminal window or the plain text console:

pcphxtr01:~$

The text at the start of the line is called the “prompt” and its purpose is to prompt you to enter some commands.

The prompt can be changed (see below) but the default prompts on PWF Linux have these components:

|  |  |
| --- | --- |
| pcphxtr01 | Your computer |
| : | separator |
| ~ | The directory your session is “in”, also known as the “current working directory”. “~” is shell short hand for “your home directory”. |
| $ | Final separator. |

To issue a command at the prompt simply type the name of the command and press the Return key, [↵]. For example, the ls command lists the files in the current working directory:

pcphxtr01:~$ ls  
Appscfg.PWF Desktop Library My Music My Pictures My Video Unix Intro

pcphxtr01:~$

Note that the prompt is repeated after the ls command has completed.

### Logging out

Once we are finished with a terminal or a terminal window we need to quit. We will illustrate three ways to do this.

### Close the window

In the graphical environment the terminal window is just another window. At its top right corner are the three buttons for minimising, maximising and closing. If you click in the [×] button the window is closed and the session cleanly ended.

### The exit command

In either a terminal window or a text console you can issue the command “exit”; this will end the session. In the graphical environment ending the session running in a window closes the window too[[1]](#footnote-2). In a text console the console is typically cleared and a fresh login prompt presented.

### [Ctrl]+[D]

Recall that “[Ctrl]+[D]” means to press down the [Ctrl] key at the same time as the [D] key. In practice we press the [Ctrl] key down, press and release the [D] key, and then release the [Ctrl] key.

On a Unix system [Ctrl]+[D] means “end of input”. We will meet it later when we are entering data nto a command and want to signal that we have finished. Here it signals to the shell that we have no more input for it so it might as well quit. And quit it does.

|  |  |  |
| --- | --- | --- |
| ****  [5 minutes] | 1. Log in to the graphical interface if you are not already logged in. 2. Start up two terminal windows. (Recall: Applications→ Unix Shell→ Gnome Terminal) 3. Switch to the [Ctrl]+[Alt]+[F2] console and log in there too. 4. Repeat for the [Ctrl]+[Alt]+[F3] console. 5. Run the command “w”. This shows who is logged in and where. You should get something like this:   pcphxtr01:~$ w  10:20:00 up 16 min, 5 users, load average: 0.01, 0.08, 0.16  USER TTY LOGIN@ IDLE JCPU PCPU WHAT  y250 tty2 10:10 10:00 0.05s 0.05s -bash  y250 tty3 10:10 0.00s 0.07s 0.00s w  y250 tty7 10:06 10:07 3.39s 0.47s /usr/bin/gnome-session  y250 pts/0 10:09 10:27 0.02s 0.02s bash  y250 pts/1 10:09 10:20 0.04s 0.04s bash  pcphxtr01:~$  The user column will show your ID rather than y250, of course.  Log out of one of the text consoles when you are done. | |
| **!** | | It is easy to forget to log out of sessions that aren't right in front of your eyes. Logging out of the graphical interface will not log you out of any of the text interfaces. |

### Just for interest

The letters “tty” in w's output stand for “teletype”. This was the name of the old, clunky paper printing terminals that used to be plugged into the backs of the old mainframe computers. Today we don't have that set up but instead we have multiple teletypes all provided through the same console and switched between using the [Ctrl]+[Alt]+[Fn] key combinations.

The teletype “tty2” corresponds to [Ctrl]+[Alt]+[F2], and “tty3” to [Ctrl]+[Alt]+[F3]. The seventh teletype (tty7) is dedicated to the graphical interface managed by a program called gnome-session. This is why you use [Ctrl]+[Alt]+[F7] to return to the graphical environment.

The graphical terminal windows are not “real” teletypes. These are managed by a “pseudo-terminal service”, or “pts” for short. The first two terminal windows you create are assigned pts/0 and pts/1 respectively.

I would strip out this section and only use the graphical terminal

|  |  |
| --- | --- |
| ****  [5 minutes] | The w command summarizes various bits of data which can also be seen individually. Try the “who” and “uptime” commands as well. |

---------------------- all the other sections ---------------------------  
  
Bioinf facility puts the examples in ‘course materials’ within the home so all the paths had to be updated  
  
The home isn’t the PWF shared one so references to my documents / my pictures etc had to be removed.

# Section 4: Remote access to other Linux systems

Mark generates the accounts for this ection and we use an Open Stack VM, we don’t use the remote system ‘soup’ specified in the PWF exercise. This whole section is different.

Today’s course will be taught using the training workstations and these have Ubuntu Linux installed on them. Unless you have made the decision to run Linux on the desktop I would expect that your need for learning Linux commands is because you need to work on remote systems i.e. HPC clusters or departmental servers.  
  
Many Linux systems have an SSH server installed on them to allow a remote log in, provided you have an account on the remote system.   
  
The command for this is “ssh” (“secure shell”) and as the name “shell” suggests it gives a command line on the remote system.

The first part of ssh's security is to check that you are connecting to the system you think you are. In the ssh world every machine has a “fingerprint”. The idea is that you can check from a workstation which has never made contact with it before.

## Remote login between cooperating systems

Let say we connect from our local machine “workstation” to the server “unix-training.hpc.private.cam.ac.uk” and our remote username is “y250”   
  
We would use the command “ssh y250@unix-training.hpc.private.cam.ac.uk”.   
  
If we haven’t connected before we will see a “not in list of known hosts” warning.

Workstaton:~$

RSA host key for IP address '172.24.47.21' not in list of known hosts.

y250@unix-training.hpc.private.cam.ac.uk’s password:

We enter the password for our remote user account. The password will not be repeated on the screen.

workstation:~$ **ssh y250@unix-training.hpc.private.cam.ac.uk**

y250@unix-training.hpc.private.cam.ac.uk's password:

=============================

Welcome to the Introduction for Unix shell scripting test environment.

Information on the course may be found at https://www.training.cam.ac.uk/event/2496994.

If you require assistance please feel free to ask the trainers.

=============================

Last login: Sun May 20 10:28:45 2018 from 131.111.56.111

Notice how both user name and machine name in the prompt has changed.

## Remote login to a new system

If you connect to a system which doesn't have fingerprint arrangements sorted out in advance you will get a challenge like this:

workstation:~$ ssh unix-training.hpc.private.cam.ac.uk  
The authenticity of host '172.24.47.21 (172.24.47.21)' can't be established.

ECDSA key fingerprint is SHA256:65jYNEzGK1SL0nJfX5xD3Cp8DjCLgnfB+Jeh/LSVBKc.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '172.24.47.21' (ECDSA) to the list of known hosts.

y250@unix-training.hpc.private.cam.ac.uk's password:

It is common for UIS to publish fingerprints online so that you can compare them to the fingerprint seen in the terminal. It is best practice to check a finger print before you accept, if you don’t trust the remote host answer ‘**no**’.

Once we have accepted a fingerprint on a system we will not be challenged again. If the remote system changes its fingerprint (which it should only do if it has hacked or had its operating system re-installed) then our login attempts will be rejected as insecure. The ssh command believes that the connection has been hijacked by another computer claiming to be the one you wanted and had been to before.

The ssh command assumes we have an account on the remote system with the same name as the account on the workstation. If this is not the case we can give an account name by preceding the machine name with “username@hostname”.

## Exercise 7: SSH for remote login (5 minutes)

Use ssh to login to the remote server unix-training.hpc.private.cam.ac.uk

1. On your local workstation, open a terminal
2. **ssh y250@unix-training.hpc.private.cam.ac.uk**

Hint: Replace y250 with the username you have been given for the remote server

1. Say **yes** to accept the remote key
2. Enter your password
3. type **ls**
4. Type **exit** to disconnect

## File transfer

In addition to logging on to remote systems we may also want to transfer files to or from them. In addition to ssh (“secure shell”) there is a related program “scp” (“secure copy”). This behaves in exactly the same way as cp except that one of the target or destination is actually a reference to a remote system.   
  
There is also “rsync” a program that can remotely synchronise files between computers (in either direction) and “sftp” which is an interactive file transfer program that lets you navigate a the far end.

### Fetching files and directories

If there was a file on the machine unix-training.hpc.private.cam.ac.uk called “/home/y250/LinuxIntro.tar”. To fetch it into the current working directory we could run the following command:

workstation:~$ **scp y250@unix-training.hpc.private.cam.ac.uk:/home/y250/LinuxIntro.tar LinuxIntro.tar**

LinuxIntro.tar 100% 4604KB 4.5MB/s 00:00

workstation:~$ **ls -lah LinuxIntro.tar**

-rw-rw-r-- 1 y250 y250 4.5M May 19 23:22 LinuxIntro.tar

workstation:~$

Not that we define a file on a remote computer: machine\_name:file\_path with a colon separating the two components.

Just as with ssh, scp assumes you have the same login id on the remote system as on the local one. If you have a different name on the remote system (in my example I do) then you specify “user@” before the machine\_name:file\_path element.   
  
If we are happy for the file name to remain the same there is a trick to save on the typing. We can say “copy it into this directory” in which case the copy will leave it with the same file name:

workstation:~$ scp y250@unix-training.hpc.private.cam.ac.uk:/tmp/LinuxIntro.tar .

Recall that “.” means “the current directory”.   
We can rename the file as we copy it simply by giving a different name as the second argument:

workstation:~$ scp y250@unix-training.hpc.private.cam.ac.uk:/tmp/LinuxIntro.tar New**LinuxIntro.tar**

To fetch a directory and everything in it, we must specify a recursive copy. Unfortunately, the scp program hasn't moved to the modern upper case “‑R” option for recursion so we have to use the lower case “‑r”:

localworkstation:~$ scp y250@unix-training.hpc.private.cam.ac.uk:/tmp/fetchable .

### Sending files and directories

To send data rather than to fetch it we simply use the same syntax for specifying a remote file but on the second argument rather than the first.

To copy a file from the current working directory to your home older on the remote location we specify it like this:

localworkstation:~$ scp newdata.txt y250@unix-training.hpc.private.cam.ac.uk:/home/y250/my\_data.txt

### Interactive file transfer

So far we have been able to send or fetch files but in both cases we need to know the remote location and we have had no opportunity to send some files and fetch others. The second file transfer program, “sftp”, will allow us to do that but stops us transferring directories recursively (for no readily apparent reason).

The sftp program is interactive, so rather than issue a single command, as with scp, you launch the sftp program to connect to a remote system and then issue a series of instructions within the sftp program telling it to change directories at either end, to fetch or send files, to list directories at either end etc.

We launch sftp by simply identifying the remote computer. Notice that the prompt changes to indicate that we are now inside the sftp program rather than the shell. To draw out a particular point we will move into the Linux Intro directory before launching the program.

workstation:~$ cd Linux\ Intro

workstation:Linux Intro$ **sftp y250@unix-training.hpc.private.cam.ac.uk**

The authenticity of host 'unix-training.hpc.private.cam.ac.uk (172.24.47.21)' can't be established.

ECDSA key fingerprint is SHA256:65jYNEzGK1SL0nJfX5xD3Cp8DjCLgnfB+Jeh/LSVBKc.

Are you sure you want to continue connecting (yes/no)? **yes**

sftp>

The examples assume that our local user is different to our remote user which is why we are putting the remote username in the command.

workstation:Linux Intro$ sftp y250@unix-training.hpc.private.cam.ac.uk

Notice how we do not specify a location. This is the first difference from scp. We always start in our home directory at the far end.

sftp> pwd   
Remote working directory: /home/y250

sftp>

The Linux command pwd in the sftp program gives information about the remote end. To get the local working directory, use the sftp-only command “lpwd” (“local pwd”):

sftp> lpwd   
Local working directory: /home/y250/Linux Intro

sftp>

This pattern is repeated for several Linux commands. The original Linux command inside sftp works on the remote system and the same command prefixed with an “l” (for “local”) works on the local system. For example to change directory at the remote end we use “cd” and to change directory locally we use “lcd”:

sftp> cd /tmp

sftp> lcd /home/y250

sftp> pwd   
Remote working directory: /tmp

sftp> lpwd   
Local working directory: /home/y250

sftp>

The ls and lls commands list files at either end and support the ‑l and ‑a options.

To actually transfer files we use two commands within sftp: “get” and “put”.

sftp> lls   
Desktop My Music My Pictures My Video newdata.txt Linux Intro

sftp> put newdata.txt y250\_example.txt   
Uploading newdata.txt to /tmp/y250\_example.txt   
newdata.txt 100% 86 0.0KB/s 00:00

sftp> get LinuxIntro.tar  
Fetching /tmp/LinuxIntro.tar   
/tmp/LinuxIntro.tar 100% 86 0.1KB/s 00:00

sftp>

To exit the sftp program, either enter [Ctrl]+[D] or the command “quit”. Notice how you return to the shell as you left it. The internal lcd command only affected the session within sftp.

sftp> quit

workstation:Linux Intro$

A full set of sftp commands is given by the sftp command “help”. A set of the most useful ones is given in the appendices to these notes.

sftp> help   
Available commands:   
cd path Change remote directory to 'path'   
lcd path Change local directory to 'path'   
…  
? Synonym for help

sftp>

## Exercise 8: SFTP for remote file transfer (10 minutes)

**Make sure you’ve completed exercise 7 first!**  
Use sftp transfer a file to the remote server unix-training.hpc.private.cam.ac.uk

1. On your local workstation, open a new terminal
2. Start an sftp session:   
   **sftp y250@unix-training.hpc.private.cam.ac.uk**

Hint: Replace y250 with the username you have been given for the remote server

1. Now transfer a file:   
   **put** LinuxIntro.tar
2. Check the file has transferred by listing the remote directory:  
   ls -lah   
   -rw-rw-r-- 1 y250 y250 4.5M May 19 23:22 LinuxIntro.tar
3. We have now transferred our file so we can leave the *sftp* session.
4. We now need to unzip the file on the remote system. The file has been compressed using ‘tar’ so we will use a tar command to uncompress the files.
5. Start an SSH session: **ssh y250@unix-training.hpc.private.cam.ac.uk**
6. Check the file is in your home by listing directory:  
   ls –lah
7. Now we will use the tar command to unzip the files.  
   **tar -xvf** LinuxIntro.tar
8. Check the file has unzipped by listing the remote directory  
   ls  
   Linux Intro LinuxIntro.tar
9. You can delete the zip file:  
   rm LinuxIntro.tar
10. Type **exit** to disconnect  
      
    Hint: Check the file **LinuxIntro.tar** is in your local directory before you try to transfer it

# Launching graphical applications from the command line

|  |  |
| --- | --- |
| **!** | This section requires that you be running in a graphical terminal window rather than in a text console. Please remember to be patient when switching between consoles.  If you are already running a web browser, please kill it off before starting this section. |

Now that we have a command line we can use it to launch more useful commands than just ls. For example we can launch the Chromium web browser.   
  
To get this exercise to work properly and we had to use chromium instead of firefox

### Background commands

For these examples, please quit any currently running browsers you may have.

We give the command “chromium”, with a lower case “c”.

pcphxtr01:~$ chromium↵

(For reasons that will become clear soon we are explicitly showing when the Return key, [↵], is being pressed.)

The Chromium web browser launches but in the terminal window the prompt has not been returned. We type “ls” (and press [↵]) but nothing happens (yet).

pcphxtr01:~$ chromium↵

ls↵

Now we quit the browser from the browser's menus (File→Quit) and we see that the prompt comes back, the ls command is repeated at it, and then run:

pcphxtr01:~$ chromium↵

ls↵

pcphxtr01:~$ ls  
Appscfg.PWF Desktop Library My Music My Pictures My Video Unix Intro

pcphxtr01:~$

This means that each application launched is going to tie up a terminal window and any future commands typed in that terminal window will have to wait for the current command to finish before they are run. We can do better than that by running commands “in the background”. To do this we follow the command with an ampersand, “&”:

pcphxtr01:~$ chromium &  
[1] 7941

pcphxtr01:~$

This time the prompt did come back immediately; the shell did not wait for the command to complete before asking for further instructions. We could now run ls again if we wanted for immediate results.

The “[1]” means that this is the first job we have in the background for this session. The number, 7941 in the example above, is a numerical identifier (the “process id”) for this backgrounded command. We don't need to know about it, but yours will almost certainly be a different number.

|  |  |
| --- | --- |
| **!** | During the course of this chapter you may see some warning messages appear, for example:  pcphxtr01:~$ chromium & [1] 7941  pcphxtr01:~$ \*\*\* nss-shared-helper: Shared database disabled (set NSS\_USE\_SHARED\_DB to enable).  NPP\_GetValue()  NPP\_GetValue()  Don't worry about these. Graphical commands are quite noisy like this because their authors know that you don't get to see the messages if you launch the application graphically.  Note that because the command is running in the background these messages arrive “asynchronously”. This means thet they arrive whenever they want and not in response to you doing anything. If they cause you to lose track of your prompt, just press [↵] to get another. |

There is one more feature to observe. We started the Chromium application from this shell and this shell gets informed when it finishes. If we close down Chromium from its menus (File→Quit) then we get a notification the next time a prompt is produced by the shell:

pcphxtr01:~$

[1]+ Done chromium

pcphxtr01:~$

The message “Done” indicates that the program terminated normally. You may get other messages if the program crashes.

|  |  |
| --- | --- |
| ****  [5 minutes] | 1. Run the xeyes command in the background. It should get a “[1]”. 2. Run Chromium too. It should get a “[2]”. |

### Job control

What can you do if you have already launched a graphical application but forgot to add the ampersand?

There are facilities for taking a running job and moving it into the background. The process comes in two stages: first we stop the running program (“stop” as in “pause” rather than “finish”) and then we restart it in the background.

We will use a second instance of xeyes as an example. First we start it in the foreground (i.e. without the ampersand):

pcphxtr01:~$ xeyes

To stop the job we press [Ctrl]+[Z]:

pcphxtr01:~$ xeyes  
^Z   
[3]+ Stopped xeyes

pcphxtr01:~$

The “[3]” means that this is the third command we have either backgrounded or stopped (if the two commands from the exercise are still running). The “Stopped” says that it's stopped, obviously. This is followed by the command itself.

At this point we have the prompt back but the xeyes program isn't active; it's stopped. Try moving a terminal window over the xeyes window. You will see that the xeyes program doesn't track the pointer any more or even redraw itself properly; it's completely inactive.

Now we restart it in the background. To do this we issue the command “bg” (for “background”).

pcphxtr01:~$ bg   
[3]+ xeyes &

Again we get a response indicating what has happened. The “[3]” matches the identifier we saw when we stopped it. We also get the command repeated but this time with a trailing ampersand to indicate that it's running in the background and indeed we do have xeyes running in the background as if we had started it with an ampersand in the first place. If you move the terminal window over the xeyes window you will see it redraw itself correctly.

If we had stopped the command with [Ctrl]+[Z] and changed our mind we can always restart the command in the foreground with the “fg” (“foreground”) command.

If you want to know what jobs you currently have running in the background, issue the “jobs” command:

pcphxtr01:~$ jobs   
[1] Running xeyes &   
[2]- Running chromium &   
[3]+ Running xeyes &

pcphxtr01:~$

The number in square brackets is called the “job number” and we can use it to identify a particular process. If we had thee three jobs running in the background and wanted to foreground the second of them (chromium) then we could do that with “fg %2” where the number after the percentage sign is the job number of the job being brought to the foreground. If we wanted to background it again we could do that with [Ctrl]+[Z] and bg again. It would still be job number 2. This sort of pushing and pulling of jobs into the background tends to be a minority interest. Typically you will start a job in the background with the ampersand or you won't want it in the background at all.

### Killing background jobs

Job control is also tied to another useful facility: killing rogue processes. Suppose a command had gone mad and was refusing to quit as you desperately clicked on the quit button. If a process doesn't die when you click its [×] button in the title bar then usually the graphical environment will wait sixty seconds or thereabouts and prompt you for whether or not you want the process killed more emphatically (while warning you that this will lose any unsaved work). Alternatively, we can use the command line.

We can start with the job numbers in square brackets. At the moment, if you have been following the notes, we have two instances of xeyes and one instance of chromium running. Note that our first instance of xeyes has job number 1, i.e. is labelled “[1]” in the jobs output. If we run the command “kill %1” then the process corresponding to “[1]” is killed, in our case one of our xeyes. A message will appear that the job has been killed the next time you get a prompt.

pcphxtr01:~$ kill %1

pcphxtr01:~$   
[1]- Terminated xeyes

pcphxtr01:~$ jobs   
[2]- Running chromium &   
[3]+ Running xeyes &

If a command gets really stuck then there is a stronger version of the “kill” command. By default kill politely requests that a command should wind up its business and terminate. If this fails, there is an option “kill ‑KILL” which causes the process to be abruptly killed by the operating system. Note that you can only kill processes which “belong” to you.

pcphxtr01:~$ jobs   
[2]- Running chromium &   
[3]+ Running xeyes &

pcphxtr01:~$ kill -KILL %3

pcphxtr01:~$   
[3]+ Killed xeyes

pcphxtr01:~$ jobs  
[2]+ Running chromium &

|  |  |
| --- | --- |
| ****  [5 minutes] | 1. In the Chromium that you are running in the background navigate to another page. 2. Kill the Chromium instance with the “kill ‑KILL” command and its job number. 3. Start Chromium again. You should get a warning window: “Your last Chromium session closed unexpectedly. You can restore the tabs and windows from your previous session, or start a new session if you think the problem was related to a page you were viewing.” 4. Select the option to restore the previous session. You should appear back at the last page you were looking at. |

### Why would you want job control?

What’s the point of job control? After all, you can always launch another terminal window.

Backgrounding can be used for much more than just graphical applications however. Any job that is going to take time to complete can be backgrounded. These jobs can be run in a purely text environment where you can’t just open another terminal. Alternatively, as we will see later, you may be running the program (graphical or otherwise) on a remote system where you only have one connection established. Backgrounding jobs is often a lot less hassle than establishing another connection.

### What would the GUI do?

We can ask the windowing system to open a file with “whatever application it would have used if we had double clicked on the icon in the graphical interface”. The command to do this is called “xdg-open”. (The style of window interface we use on PWF Linux is called “GNOME”.)

pcphxtr01:Unix Intro$ pwd  
/home/y250/Unix Intro

pcphxtr01:Unix Intro$ ls  
Play story.txt Treasure Island Work

**pcphxtr01:Unix Intro$ xdg-open story.txt**

**pcphxtr01:Unix Intro$ xdg-open Treasure\ Island/hispaniola.png**

pcphxtr01:Unix Intro$

This launches gedit (the GNOME editor) for the text file story.txt and eog (“eye of gnome”, the default GNOME picture viewer) for the graphical file hispaniola.png.

Because the command is only used for graphical applications it automatically “detaches” the applications (gedit, eog, etc.) from the terminal so it does not need to be backgrounded.

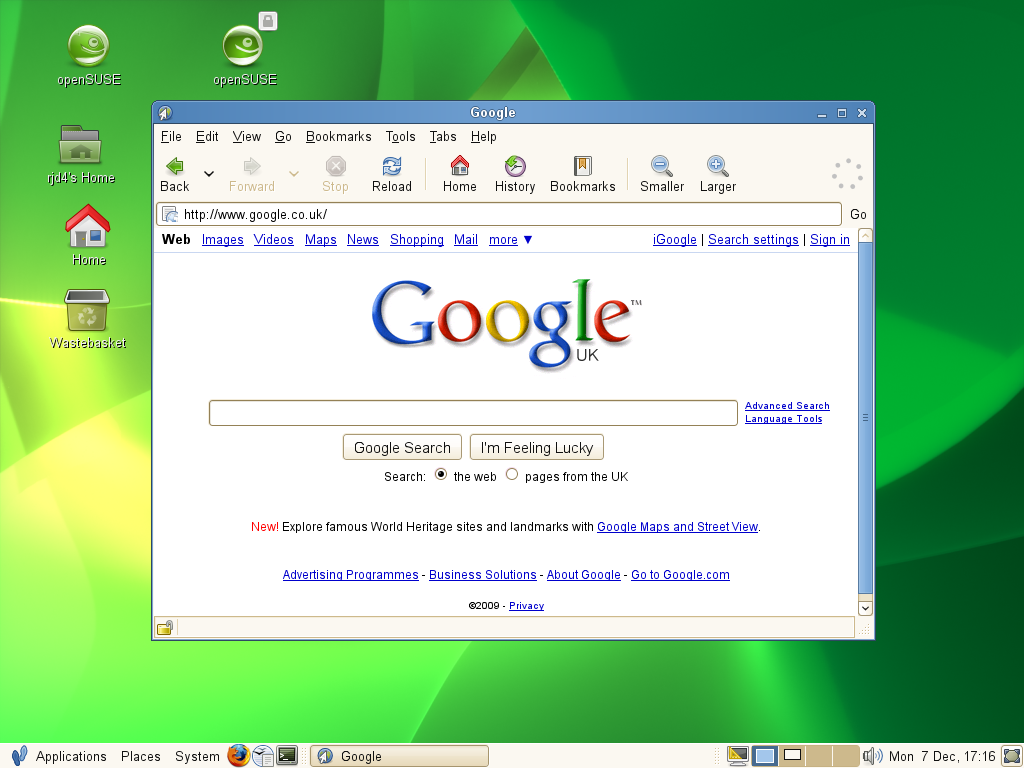
|  |  |
| --- | --- |
| **!** | The application cannot usefully open non-existent files. It will not launch the application suitable for a file of that name as a quick way to start an editor with an empty file, for example. |

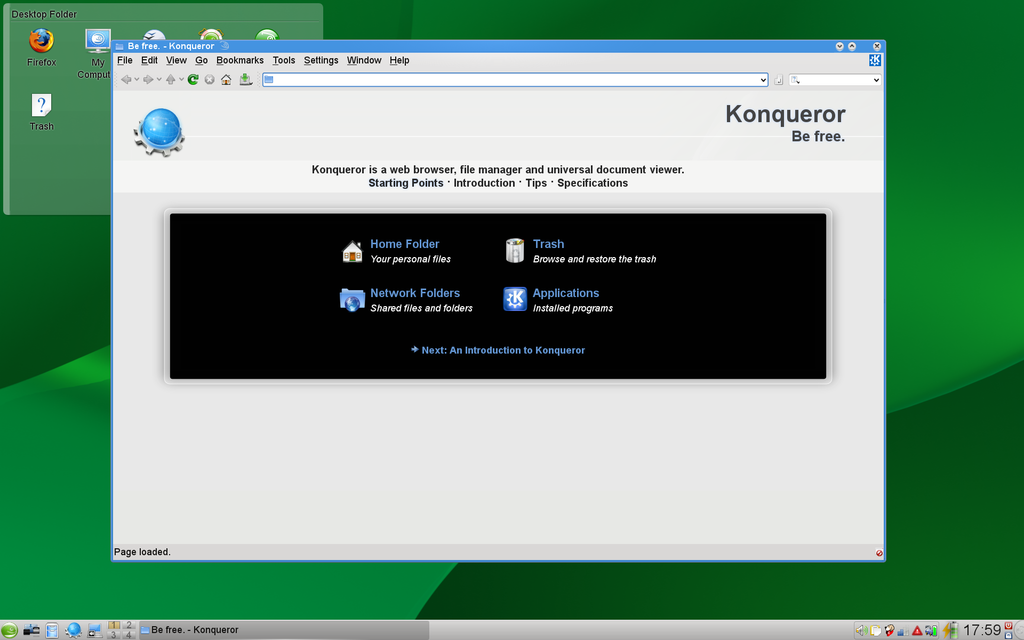
To create an empty file, fubar.txt say, use the “touch” command: “touch fubar.txt”.

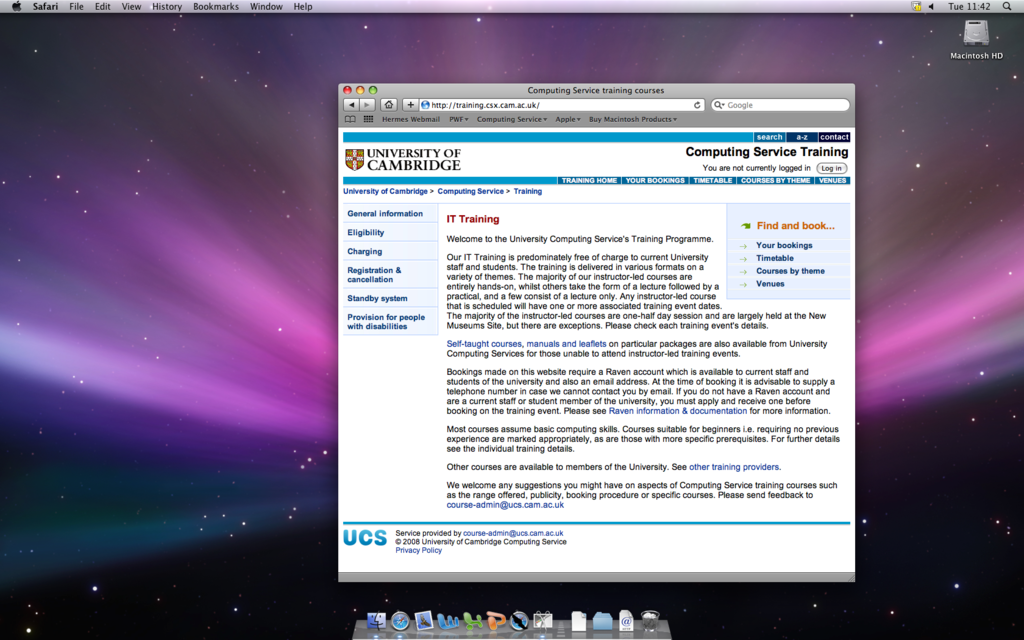
|  |  |
| --- | --- |
| ****  [5 minutes] | 1. Run gnome-open on each of the files in Work/Project Epsilon. (It will only have this name if you have done the previous exercises. It was originally called Project Alpha.) 2. Close down the applications and then try to work out the direct commands to use to run the same applications as gnome‑open did. (e.g. gedit story.txt is the equivalent of gnome‑open story.txt) Don't forget to background them. (Hint: Help→About in an application typically identifies the application.) |

### Just for interest

Linux has other windowing environments such as “KDE” and “GNOME”. The command we have used is “xdg-open”, on KDE you can use “kde-open” and on GNOME “gnome-oppen”. On MacOS X (a different flavour of Linux) the command is called simply “open” as there is no choice of windowing environment.

 GNOME

 KDE

 MAC OS X

During the date exercise we cover how the date can be used in a script

## Using the date in a shell scripting

It is quite common to write shell scripts that make use of the date command. Often we manipulate the date to help control the script or to create some kind of output such as a log file. The example below is a backup script, it logs the start and end date as well as using the short date for making folder names.



1. This is the case on PWF Linux and most traditional Unixes. On MacOS X the terminal window is left open and the window needs to be closed explicitly. [↑](#footnote-ref-2)