A Brief Introduction To Linux

Table Of Contents

Disclaimer

This document is not intended as a comprehensive guide to either Linux in general, or its specific implementation at UWE. It is a quick guide for people who know nothing about Linux, just to get you started. Once you know everything contained in this document do not attempt to use it as any sort of definitive reference, you should refer to the manuals.

Answers to common questions relating to the Linux systems can be found in the other help sheets.

Notes

Note that:

- Linux is case sensitive - most commands and options must be typed in lower case. In particular, if your password contains mixed case characters they must each be typed in the correct case in order for it to work.
- Linux commands cannot be abbreviated. They are mostly as short as possible anyway. Long command lines can however, be given an alias (see Basic Linux commands below).
- In this guide:

```
this is text that the computer puts on the screen
this is text that you type in
this is text that needs to be replaced with something
else
```

For example, where the guide

says:

`host% man the_command_name`

`host%` is the command prompt after which you might actually type:

```
man ls
```

Logging On and Logging Off

At the login screen, type your username. In response to the `Password:` prompt, type your password.

Once you are logged on to a machine you can log on to any other Linux machine in the University network (for which you have a valid account) using the ssh command:

`host% ssh remote_machine_name`

When you have finished your session on the computer, select ‘Log Out’ from the System menu.

Pathnames

In Linux you specify files using paths. An absolute path consists of a `/` (to indicate the root directory) followed by a series of directory names (separated by `/` 's) and finally, the file name, e.g. `/usr/users/fred/foo.bar` specifies the file `foo.bar` in the directory `fred`, which is a sub-directory of `users`, which is, in turn a sub-directory of `usr`.

It is also possible to specify paths relatively, using the following symbols:

`.`	specifies the current directory, i.e. either the directory that you are currently working in, or the directory specified by the path so far.
`..`	specifics the directory immediately above the current one, i.e. either the directory above the one that you are currently working in, or the one above the directory specified by the path so far.
`~`	specifies your home directory, i.e. the one that you get put into whenever you log on.

For example, for user fred (with home directory `/usr/users/fred`) currently working in directory `/usr/users/fred/src/myprog`

`./data`	specifies the file (or directory) `/usr/users/fred/src/myprog/data` (in most circumstances this would normally be specified by just typing `data`).
`../myprog2`	specifies the file (or directory) `/usr/users/fred/src/myprog2`
`~/bin`	specifies the file (or directory) `/usr/users/fred/bin`

Wildcards which can be used in path names are:

`*`	matches any number of occurences of any character.
`?`	matches a single occurrence of any character.
`{a,ab,xyz}`	matches any of the strings `a` , `ab` , or `xyz` .

Note: A command which expects only one path will not work if a wildcard matches more than one path, e.g. if you have two

subdirectories `mydir` and `mydir2`, then **`cd my*`** will not work but **`cd *2`** will.

Basic Linux Commands

The following are just a small selection of Linux commands to get you started, for more information on a particular command type **`man`** ***`the_command_name`*** to look at that command's on-line manual page. To find out what commands there are associated with a particular topic type **`man -k`** ***`some_keyword`***, or **`apropos`** ***`some_keyword.`*** To find out more about using the `man` command type **`man man`** .

`ls` ***`path`***	**`ls`**ts the contents of directory ***`path`*** , or the details of file ***`path`*** .
`cd` ***`path`***	**`c`**hanges **`d`**irectory to ***`path`*** .
`pwd`	**`p`**rint (display) your current **`w`**orking **`d`**irectory.
`mv` ***`path1 path2`***	**`m`**o**`v`**e whatever is at ***`path1`*** to ***`path2`*** .
`cp` ***`file1 file2`***	**`c`**o**`p`**y ***`file1`*** to ***`file2`***.
`rm` ***`file1 file2 ...`***	**`r`**e**`m`**ove files ***`file1 file2 ...`***
`rmdir` ***`dir1 dir2 ...`***	**`r`**e**`m`**ove **`dir`**ectories ***`dir1 dir2`***. . . (**Note:** The directories have to be empty).
`file` ***`file1 file2 ...`***	show what sort of **file** the files ***`file1 file2 ...`*** are for example text, data or binary etc. It is particularly advisable to check that a file is not binary before you try to look at it, or print it.
`cat` ***`file1 file2 ...`***	show the contents of the files ***`file1 file2 ...`*** this derives from the rather archaic word "**`cat`**enate".
`more` ***`file1 file2 ...`***	display the files ***`file1 file2 ...`*** one screen at a time; so called because it asks you if you want to see **more** at the end of each screen.
`head` ***`file1 file2 ...`***	display the **head** of the files ***`file1 file2 ...`*** i.e. the first few lines.
`tail` ***`file1 file2 ...`***	display the **tail** of the files ***`file1 file2 ...`*** i.e. the last few lines.

grep ***string file1 file2 ...***	show all the lines in the files ***file1 file2 ...*** which contain the string ***string*** - this derives from an old editor command "**g**lobal **r**egular **e**xpression **p**rint".
lp -d ***printer file1 file2 ...***	print the files ***file1 file2 ...*** on the printer called ***printer*** - the **l** is a relic from the days of **l**ine **p**rinters. The **-d** is for **d**estination.
lpstat -o ***printer***	show the **stat**us of the (**l**ine)**p**rinter output requests for the printer ***printer*** .
cancel ***print_job_id***	**cancel** the specified print job.
alias ***name command***	make ***name*** an **alias** for the command ***command*** - this is usually used to shorten long winded or difficult to remember commands.
w	display **w**ho is logged on, including the total number of people logged on, and how busy the machine is.
date	display the **date** and time.

An easy way to create files to practice the file handling commands with is to use the **touch** command - **touch** ***file*** creates an empty file with the name ***file*** .

Printers In CSCT and EDM

There are several printers available for use from the CSCT/EDM Linux system.
Other printers exist for individuals and workgroups around the faculty.

For further information on printers and printing options, please refer to the series of printing help sheets on the System Support web site.

Example, to print a copy of your .cshrc file on the `BWPrinting` printer device you would type:

```
host%  lp -d BWPrinting .cshrc
```

The PATH Environment Variable

Whenever you type a command in Linux it is necessary for the program which performs that command to be located. In order to do this, all the directories listed in the `PATH` environment variable are searched for the program. The first occurrence of the program is always used, therefore if there are two different versions of a program on the system you will always get the one which resides in the directory which comes earliest in the list in `PATH`. The only way to run another version is to specify its full path. For example, if you have /bin:/usr/ucb in your `PATH`, then typing **`ls`** will run the System V Linux version of `ls` (`/bin/ls`). In order to run the BSD version you would need to type `/usr/ucb/ls` .

The `which` command tells you which version of a command would be executed were you to type it, e.g.:

```
host% which ls
/bin/ls
```

indicates that `/bin/ls` would be run if you just typed **`ls`**

```
host% which xroids
 no xroids in  /usr/openwin/bin /usr/lang /usr/ucb /usr/etc
/bin  /usr/bin /usr/local/bin /usr/openwin/demo
```

indicates that the program associated with the command `xroids` is not to be found in any of the directories listed in `PATH` .

Note that the search is not actually done when you type the command, this would take too long. What happens is that, when you log on, a table of all the commands in all the directories in your `PATH` is compiled. So, if you have just created a new executable program, you may not be able to run it without giving its full path name.

Pipes and Input/Output redirection

Most Linux commands allow you to specify that their input (or output) should come from (or go to) somewhere other than the terminal. This is accomplished using the symbols `<`, `>`, `>>` and `|` :

command < file	means that ***command*** gets its input from the file ***file***
command > file	means that ***command*** sends its output to the file ***file***
command >> file	means that ***command*** appends its output to the contents of the file ***file***
command1 | command2	means that ***command1*** sends its output to the input of ***command2***

For example:

```
host% cat data.txt | grep oranges | grep –v lemons >>
result.txt
```

takes the contents of the file `data.txt` and puts all the lines which contain the word "`oranges`" but not the word "`lemons`" at the end of the

file `result.txt`
For more information on pipes and redirection, look at the manual page for `csh` or `bash`, depending on the shell you are using.

Process control

Background jobs which you leave running when you log out, anything which is `fork`'d off by a program, and a few other bits and pieces, will carry on running as detached processes. To find out what processes you have running (and to find out what their process identification numbers, PID's, are) use the `ps` command with the `-fu` *`username`* option, for example:

```
host% ps -fu g-marsh
   UID   PID  PPID  C    STIME TTY  TIME CMD
g-marsh   527   484  6   Oct 04 ?   110:07 /usr/openwin/bin/Xsun :0 -nobanner -
terminate
g-marsh   575   574  0   Oct 04 ?     0:00 /bin/ksh /usr/dt/bin/Xsession
g-marsh   589     1  0   Oct 04 ?     0:00 /usr/openwin/bin/speckeysd
g-marsh   659   651  0   Oct 04 ?    15:09 dtwm
g-marsh 23154 23153  0   Nov 10 ?     0:00 /usr/dt/bin/dtcm
g-marsh  4051  4050  0   Nov 07 ?     9:00 /usr/dt/bin/dtmail
g-marsh 16288 16287  0   Nov 01 ??    0:00 /usr/dt/bin/dtterm
g-marsh 29609 29608  0   Nov 11 ??    0:00 /usr/dt/bin/dtterm
g-marsh  5955  5954  0   Oct 04 ?     0:11 sdtperfmeter -f -H -t cpu -t disk -s 1
g-marsh 15447 15441  0 09:06:59 ?     1:11 /usr/local/thunderbird/thunderbird-bin
g-marsh 16348 16342  0 12:35:20 ?     2:29 /usr/local/firefox107/firefox-bin
g-marsh 16342 16308  0 12:35:20 ?     0:00 /bin/sh /usr/local/firefox107/run-
mozilla.sh
g-marsh 15420 15419  0 09:06:59 ?     0:00 /bin/sh /usr/local/bin/thunderbird
g-marsh 17018 16945  0 15:20:06 ?     0:00 vi users
g-marsh 16402 16401  0   Nov 01 ?     4:59
/Solaris/opt/staroffice7/program/soffice.bin
```

This shows the result for a typical X Windows session, with the window manager (`dtwm`), a few X applications (two `dtterm` terminal windows, a `dtcm` calendar manager etc), a copy of `firefox` and an editing session using `vi` .

Process control commands are:

`stop` ***`n`***	suspend the process with PID *`n`*, but do not kill it.
`kill` ***`n`***	kill the process with PID *`n`*.
`kill -9` ***`n`***	definitely kill the process with PID *`n`*.

If you run a program which forks off lots of processes or your session ends in an inelegant manner (e.g. the window manager crashes or hangs) please look to see what processes you may have left running, and `kill` them off.

File permissions

There are three basic levels of permission associated with a file. It may be **r**eadable, **w**ritable and/or e**x**ecutable. There are also three types of user who may access a file: the **u**ser who owns the file, a user in the same **g**roup as the file owner, and any **o**ther user.

To see the permissions associated with a file use the `-l` option to `ls`, e.g:

```
host% ls -l .cshrc
-rw-r-----   1 g-marsh  cots   6300 Oct  7 14:10 .cshrc
```

The first column indicates the type of file (`-` is an ordinary file, `d` is a directory, l is a link, etc).
The next three columns give the permissions assigned to the user who owns the file (in this case read and write).
The next three columns give the permissions assigned to users in the same group as the file owner (in this case read only).
The next three columns give the permissions assigned to all other users (in this case none).
The `1` indicates that there is only one instance of this file (i.e there are no links to it from other parts of the file system).
`g-marsh` is the user name of the owner of the file.
`cots` is the name of the group that the file belongs to (this will usually be the user group that the owner is in).
The file is `6300` bytes in size.
The file was last modified at ten past two on the afternoon of the 7th of October this year.

Use `chmod` to **ch**ange the permissions (**mod**e) of a file. There are two forms of this command, the first allows permissions to be set relative to what they currently are, the second specifics an absolute set of values.

With the relative method, use `u`, `g`, `o` to indicate the **u**ser who owns the file, the **g**roup, or **o**thers, or a for **a**ll of the above. Follow this with `+`, `-` or `=` depending on whether you want to add the level of access, remove it, or set it to a specific value. Finally specify `r`, `w`, or `x` (or a combination of the three, to specify **r**ead, **w**rite, or e**x**ecute permissions. For example:

```
host% chmod g-rx file
```

would remove read and execute permissions from users on the same group as ***file***, but would leave all the other permissions as they were previously.

With the absolute method, the permissions are represented by octal (base 8) numbers, with the places of the numbers relating to the types of user, in the order used by `ls -l` . Read permission has a value of `4`, write is `2`, and execute is `1` . So, for example:

```
host% chmod 644 file
```

would give read and write access to the owner of the file (`r + w = 4 + 2 = 6`), read access to other users in the group, and read access to all other users.

Note: the execute permission when applied to a directory has a special function. If a directory has permissions `700` then other users will not be able to access any files in it, even if they have read access to them. If the directory's permissions are changed to `711` then they will be able to read files that they have permission to read, but they will not be able to list the files in the directory - this requires `755` permissions on the directory.

For more information on file permissions, see the **File Permissions** help sheet.