unix intro Documentation

Release 1

Scott Wales

February 21, 2013

CONTENTS

1 Logging On **2**
- 1.1 Users & Groups . . . 2
- 1.2 Getting Help . . . 3

2 Directory Structure **4**
- 2.1 Home Directory . . . 4
- 2.2 Listing contents . . . 4
- 2.3 Moving around . . . 5
- 2.4 USB drives & CDs . . . 5
- 2.5 Creating directories . . . 5
- 2.6 Root . . . 6
- 2.7 Disk space . . . 6

3 Files **7**
- 3.1 Editors . . . 7
- 3.2 Deleting . . . 7
- 3.3 Copying & Moving . . . 7

4 Searching **9**
- 4.1 Searching for text . . . 9
- 4.2 Searching for files . . . 10
- 4.3 Regular expressions . . . 10
- 4.4 Find and replace . . . 10
- 4.5 Working with data in columns . . . 11

5 Piping Output **12**
- 5.1 Sending output to a file . . . 12
- 5.2 Sending output to a program . . . 12
- 5.3 Filtering output . . . 12

6 Background Programs **14**
- 6.1 List running programs . . . 14
- 6.2 Stopping a program . . . 14
- 6.3 Running in the background . . . 14

7 Remote Computers **15**
- 7.1 Connecting to another computer . . . 15
- 7.2 Remote Displays . . . 15
- 7.3 Remote Copying . . . 15
- 7.4 Being Nice . . . 16

7.5 Networked Machines 16

8 Working with multiple files **17**
8.1 Globbing 17
8.2 For loops 17

Contents:

CHAPTER

ONE

LOGGING ON

To be able to access the computer you will first need to log in using your department username and password. If you do not already have an account you will need to contact IT (support@earthsci.unimelb.edu.au).

While lots of tools have graphical interfaces a large amount of data processing is best done through the command line. Start a terminal by:

- Ubuntu: Run the *terminal* command, under accessories in the application menu
- OSX: Press command & space together, type in *terminal* and then enter
- Windows: Select *cygwin* in the start menu

The terminal runs a shell program, which is what allows you to see directories and run other programs. There are two shell variants in common use, *bash* and *cshell*. In ordinary use they are mostly the same, however programming in them uses different syntax.

Once you've started the terminal you will be presented with a //prompt//, where you can give commands to the computer. This documentation presents the prompt as `$`, although on your computer it might be a different character, or some representation of where you currently are on the system, e.g.:

```
swales@walesnix:~ $
```

1.1 Users & Groups

Unix systems are designed to operate with multiple users all running programs at the same time. Everyone has a username, and is in at least one group. Groups are mostly for handling access permissions - who is allowed to open or write to a file. Groups become more important on systems like NCI, where they are used for accounting how much supercomputer time you've used.

If you ever need to see your username you can use the command:

```
$ whoami
swales
```

To see what groups a user is in:

```
$ id swales
uid=1000(swales) gid=1000(swales) groups=1000(swales),4(adm),27(sudo),109(lpadmin)
```

uid and *gid* are the default user and group names, they affect default file permissions.

To find information about a user:

```
$ finger scott
Login: swales                           Name: Scott Wales
Directory: /home/swales                 Shell: /bin/bash
On since Thu Jan 10 12:43 (EST) on tty7    26 days 3 hours idle
On since Mon Jan 21 11:22 (EST) on pts/1 from :0
   23 minutes 20 seconds idle
On since Tue Feb  5 15:46 (EST) on pts/3 from :0
Last login Wed Jan 16 15:20 (EST) on pts/4 from vayu2.nci.org.au
No mail.
No Plan.
```

Help: man whoami; man id; man finger

1.2 Getting Help

Almost every command in Unix has documentation available in a //manpage//, Unix's online documentation system. To get help on a command run `man COMMAND`. This will print documnentation on the console, move around with the arrow keys & press `q` to exit.

If you don't know the name of a command you can use `apropos` to search for relevant commands:

```
$ apropos user
ExtUtils::testlib(3pm)   - add blib/* directories to @INC
File::chdir(3pm)         - a more sensible way to change directories
GetFileInfo(1), /usr/bin/GetFileInfo(1) - get attributes of files and directories
Path::Class::Dir(3pm)    - Objects representing directories
SVK::Command::Add(3pm)   - Put files and directories under version control
SetFile(1), /usr/bin/SetFile(1) - set attributes of files and directories
mkdir(1)                 - make directories
open(1)                  - open files and directories
removefile(3), removefile_state_alloc(3), removefile_state_free(3), removefile_state_get(3), removefi
rmdir(1)                 - remove directories
srm(1)                   - securely remove files or directories
sticky(8)                - sticky text and append-only directories
FcCacheNumSubdir(3)      - Return the number of subdirectories in cache
FcConfigGetCacheDirs(3)  - return the list of directories searched for cache files
FcConfigGetConfigDirs(3) - Get config directories
FcConfigGetFontDirs(3)   - Get font directories
fc-scan(1)               - scan font files or directories
```

Note that each command here has a number after it, these are different sections. Commands in section 1 can be run from the command line, commands in section 3 are in programming libraries, and available in C or Fortran programs.

In this documentation you can find links to related man pages at the end of each section. Note that these point to Linux man pages, they will be correct for Ubuntu and Cygwin. BSD based systems like OSX have slightly different options for historical reasons (options talked about here will work on all systems).

Help: man man; man apropos

CHAPTER

TWO

DIRECTORY STRUCTURE

Files in the system are arranged in a directory tree. This works pretty much identically to URLs on the internet, path names are separated by forward slashes. The major difference is that file paths start with **/** instead of **http://**.

2.1 Home Directory

When you first log in to the system you will be in your home directory. This is where your local files can be stored, you are free to create files there (although there is a size limit).

To see the full pathname of the directory you're in use the command:

```
$ pwd
/home/swales
```

Home directories are stored under **/home**, arranged by username. As a shortcut you can refer to your home directory using a tilde, **~**.

2.2 Listing contents

To see the contents of a directory use `ls`. On its own it will list the contents of the current directory you're in, or given a directory name it will list the contents of that directory.

```
$ ls
access
configs
Desktop
doc
Downloads
foo
intel
isus
projects
syntax

$ ls projects
fieldsfile
framework
previz
sawlibf
unixintro
```

Files and directories are usually distinguished by text colour, although this depends on the computer's settings.

Help: man ls

2.3 Moving around

There are two types of pathnames that you can use. Absolute paths start with **/**, and give the entire path from the root directory, e.g. **/home/swales/projects**. Relative paths on the other hand start with a directory name, e.g. **projects**, and are relative to the current working directory (what is printed when you run `pwd`).

To change the working directory use the command `cd`, giving it the name of the directory you want to switch to:

```
$ cd projects
$ ls
fieldsfile
framework
previz
sawlibf
unixintro
```

To go back to your home directory you can use the shortcut `cd ~`, or just `cd` on its own will also take you to your home directory. To go back to the previous working directory use `cd -`.

There are a couple other directory shortcuts that can come in handy - A single dot `.` means the current directory and two dots `..` means one level higher. `/home/swales` and `/home/swales/projects/..` both refer to the same directory.

Help: man cd

2.4 USB drives & CDs

Different systems load external media to different locations. Usually all you have to do is plug in a USB drive and it will appear in the file browser, to get there from the command line check the directories **/mnt**, **/media** or **/Volumes**.

For example on my Ubuntu machine my backup USB drive is located under **/media**:

```
$ ls /media
walesnix-2tb
```

2.5 Creating directories

You can create directories from the command line using the command `mkdir`. If you want to create a heirachy of directories all at once then add the flag `-p` (for parents). The `-p` flag also prevents `mkdir` from giving an error message if the directory already exists:

```
$ mkdir foo
$ ls
foo
$ mkdir foo/bar/baz
mkdir: foo/bar: No such file or directory
$ mkdir -p foo/bar/baz
$ ls foo/bar
baz
```

Help: man mkdir

2.6 Root

If you keep going upwards in the directory tree eventually you'll get to the root directory, **/** itself. This contains everything on the computer - programs, user data, configuration files etc. Unix systems don't share Window's drive letters for different hard drives, instead they are mounted as folders.

2.7 Disk space

To make sure no-one fills the entire disk everyone has a quota of how much space they can use. To see this use the `quota` command:

```
$ quota -s
Disk quotas for user swales (uid 126117):
Filesystem   space   quota   limit   grace   files   quota   limit   grace
/dev/sda3  21180K    489M    499M             656       0       0
```

To see how much space a file is using use `du -h`. If you give this a directory instead of a file it will print how much disk space everything under that directory is using:

```
$ du -h ~/projects/unixintro
408K        /home/swales/projects/unixintro/_build/html
236K        /home/swales/projects/unixintro/_build/latex
132K        /home/swales/projects/unixintro/_build/doctrees
780K        /home/swales/projects/unixintro/_build
4.0K        /home/swales/projects/unixintro/.git/objects/pack
8.0K        /home/swales/projects/unixintro/.git/objects/27
4.0K        /home/swales/projects/unixintro/.git/objects/info
44K         /home/swales/projects/unixintro/.git/objects
4.0K        /home/swales/projects/unixintro/.git/branches
4.0K        /home/swales/projects/unixintro/.git/refs/tags
4.0K        /home/swales/projects/unixintro/.git/refs/heads
12K         /home/swales/projects/unixintro/.git/refs
40K         /home/swales/projects/unixintro/.git/hooks
8.0K        /home/swales/projects/unixintro/.git/info
128K        /home/swales/projects/unixintro/.git
8.0K        /home/swales/projects/unixintro/_templates
4.0K        /home/swales/projects/unixintro/_static
1016K       /home/swales/projects/unixintro
```

To just get a summary add `--max-depth=0` to the command:

```
$ du -h --max-depth=0 ~/projects/unixintro
1020K       /home/swales/projects/unixintro/
```

Note that due to how hard drives work files have a minimum size of around 4 Kb.

Help: man df; man du

CHAPTER

THREE

FILES

3.1 Editors

You can use either graphical text editors or command-line based ones to edit files. Both vim and emacs are very powerful text editors which are programmable to support working with different file types, however they have steep learning curves. There are many other text editors available, which you use is personal preference, although it is helpful to at least know how to use command-line based editors if you do work on a remote machine.

An easy editor to start with is nano. Start it up with `nano`, it lists available commands at the bottom of the screen. ^ is a shortcut for Ctrl, so Ctrl-O saves and Ctrl-X exits.

Help: man nano

3.2 Deleting

To delete a file use the `rm` command. By itself this won't delete directories, to do this add `-r` (recursive):

```
$ touch foo
$ rm foo
$ mkdir bar
$ rm -r bar
```

(`touch` is a command used to update file timestamps, it will also create an empty file if it doesn't exist).

Help: man rm; man touch

3.3 Copying & Moving

The `cp` command copies files. You give it two paths, the first one is the source and the second the target. If the target is a directory it will copy the source file into that directory, otherwise it will use the target as a filename and copy the source to that file:

```
$ touch foo
$ mkdir bar

$ cp foo bar
$ ls bar
foo

$ cp foo foo1
```

```
$ ls
bar
foo
foo1
```

You can copy directories as well, just add the recursive flag `-r`.

Moving works the same as copying, using the `mv` command. The original file gets deleted after the move has happened.

Help: man cp; man mv

CHAPTER

FOUR

SEARCHING

4.1 Searching for text

The `grep` command searches text files for a given string, like the find command in an editor. The benefit of grep is that it is able to search across entire directories at once.

The most basic use of grep is `grep 'STRING' FILE`, which prints all the lines in FILE that contain STRING:

```
$ grep user logging_on.rst
department username and password. If you do not already have an account you
Unix systems are designed to operate with multiple users all running programs
at the same time. Everyone has a username, and is in at least one group. Groups
If you ever need to see your username you can use the command::
To see what groups a user is in::
*uid* and *gid* are the default user and group names, they affect default file
To find information about a user::
    $ apropos user
```

By default grep is case-sensitive, so lines contatining 'USER' or 'User' won't be found. To make grep case-insensitive add the `-i` flag.

If you give grep a directory instead of a file and add the `-r` flag it will search through every file in that directory, printing the filename for each match:

```
$ grep -r list .
conf.py:# for a list of supported languages.
conf.py:# A list of ignored prefixes for module index sorting.
conf.py:# a list of builtin themes.
conf.py:# further.  For a list of options available for each theme, see the
directories.rst:To see the contents of a directory use ``ls``. On its own it will list the
directories.rst:list the contents of that directory.
directories.rst:To see a listing of how much space is available on the system use the command
files.rst:An easy editor to start with is nano. Start it up with ``nano``, it lists
logging_on.rst:    FcConfigGetCacheDirs(3)  - return the list of directories searched for cache files
searching.rst:    $ grep -r list .
shortcuts.rst:Most Unix commands will accept a list of files when that makes sense, you don't
whatis_unix.rst:list of files, while the `sort` command sorts a list. To produce a sorted list
```

Help: man grep

4.2 Searching for files

To search by file name instead of contents use the `find` command:

```
$ find _build -name '*.js'
_build/html/_static/doctools.js
_build/html/_static/jquery.js
_build/html/_static/searchtools.js
_build/html/_static/sidebar.js
_build/html/_static/underscore.js
_build/html/_static/websupport.js
_build/html/searchindex.js
```

The basic format of find is `find DIR COMMANDS`. DIR is the base directory to search in, COMMANDS act as filters to select files with.

Command	Meaning
-name NAME	Case-sensitive name
-iname NAME	Case-insensitive name
-type d	Directories
-type f	Files

Help: man find

4.3 Regular expressions

Regular expressions are a powerful tool for searching text. They allow you to specify a pattern to match to in a shorthand notation, and are accepted by grep and sed, among other tools. Regular expressions are usually identified by surrounding them with slashes, e.g. `/foo.*/`.

Pattern	Matches
`foo`	The string "foo"
`.`	Any character
`[a-c]`	Either of a b or c
`[^a-c]`	Not any of a b or c
`\(foo|bar\)`	Either the string "foo" or the string "bar"
`\s`	A whitespace character
`\S`	A non-whitespace character
`*`	0 or more of the previous match
`\+`	1 or more of the previous match
`\?`	0 or 1 of the previous match
`^`, `$`	The start, end of the line
`\<`, `\>`	The start, end of a word

The pattern will try to match as much of a line as possible, the expression `.*` will match an entire line (any number of any characters).

4.4 Find and replace

To do a find & replace on a file with sed use the command `sed 's/FROM/TO/g' FILE`, with the sections in the replace command separated by slashes.

The replacement text is allowed to depend on the search text through the use of groups. If you surround part of the search with `\( \)` then you can refer to each group with `\1`, `\2` etc. As an example the expression `s/integer\s*\(::\)\?\s*\(\S\+\)/\2 = 0/i` will change a fortran variable definition to an assignment.

Section	Explaination
`s/`	Start a replace
`integer`	The string "integer"
`\s*`	Any number of spaces
`\(::\)\?`	An optional string "::", set as group 1
`\s*`	Any number of spaces
`\(\S\+\)`	1 or more non-whitespace characters, set as group 2
`/`	Separator
`\2 = 0`	Replace with whatever group 2 matched and the string " = 0"
`/i`	Matches are case-insensitive

Sed also has the ability to work on multiple lines at once, or only affect certain parts of a file.

Help: man sed

4.5 Working with data in columns

Awk is a simple programming language for doing calculations on columns of data. Awk commands look like:

```
$ awk FILE '{print $1, $2*$3}'
```

The actual program needs to be enclosed in quotes and braces, then you can do operations on the columns by referring to them as `$1`, `$2` etc.

You can also specify someting to run at the end, e.g. to find the mean of column 2:

```
$ awk FILE '{sum += $2} END {print sum/NR}'
```

This adds each entry in column 2 to a variable called sum, then at the end prints sum divided by NR, the number of rows in the file.

Help: man awk

CHAPTER

FIVE

PIPING OUTPUT

Unix programs have 2 types of output they can print: normal output, termed *stdout*, and error messages, termed *stderr*. Each of these by default outputs to the screen, but it is possible to redirect this output either to a file or to another program.

5.1 Sending output to a file

To redirect output to a file use the > operator. This will create a new file, overwriting anything already inside of it:

```
$ echo "hello" > hello
```

If you want to append the output instead use >>:

```
$ echo "goodbye" >> hello
$ cat hello
hello
goodbye
```

If you want to save error messages in a file, perhaps for later analysis, use `2>`. Combine stderr and stdout messages with `2>&1`.

5.2 Sending output to a program

Most programs will accept input from a *pipe* if you don't give it a file to work on. A pipe is denoted with the pipe character `|`, and sends the output of one program to the input of another:

```
$ cat hello | grep g
goodbye
```

5.3 Filtering output

Pipes allow you to filter the output of programs, selecting only the data you need. Grep is one useful tool for this, a couple more are `head` and `tail`, which select the first or last few lines of the input respectively. You can specify how many lines to take with the `-N` flag:

```
$ cat hello | head -N 1
hello
$ cat hello | tail -N 1
goodbye
```

`sort` is a command for sorting files. By default it does a textual sort (i.e. "10" sorts before "2" because of the first character), use the `-n` flag for a numeric sort. You can specify the column to sort on using the `-k` flag.

`uniq` removes duplicate lines from a file, giving you a count of how many lines there were if you add the `-c` (count) flag. Combined with sort this can give a simple histogram with the filter `sort -n | uniq -c`.

CHAPTER

SIX

BACKGROUND PROGRAMS

6.1 List running programs

To list all of the programs running in your current session use `ps`:

```
$ ps
  PID TTY           TIME CMD
10684 ttys000    0:00.12 -bash
10841 ttys000    0:34.15 vim directories.rst
11801 ttys000    0:00.00 /bin/bash -c (ps) >/var/folders/Dm/DmAfutF5FUyrDzp5ahAjs++++TI/-Tmp-/v633302
```

You'll see your shell (in this case bash) and the ps command itself, alongside any other programs running in the current shell. To see all running programs (for instance if you have started programs in the GUI) add the `-x` flag.

6.2 Stopping a program

If a program has hung, or if you don't care to wait for it to finish, you can use **ctrl-c** to quit it. You can also use the `kill` command, giving it the PID value shown by ps.

6.3 Running in the background

To suspend a program and get back to the shell use **ctrl-z**. This will put the program in a 'sleep' state where it isn't running but can be resumed. To resume it either use `fg` to start it back up and give it control of the terminal or `bg` to start it running in the background so that you can still use the shell. Start a program in the background by putting a `&` at the end of the command.

CHAPTER

SEVEN

REMOTE COMPUTERS

Often it is convenient to be able to access another computer remotely, for instance if you need to access data or use more computing power than you have at your desktop.

7.1 Connecting to another computer

To connect to another computer use the `ssh` command:

```
$ ssh swales@abyss
```

You give ssh a username and machine to connect to, formatted like an email address.

The local machines use your university account for authentication, you'll need to add `STUDENT\\` in front of your username:

```
$ ssh STUDENT\\swales@abyss
```

7.2 Remote Displays

You're not limited to the console in a remote connection. If you add the `-X` flag to ssh you can also use graphical programs over the network. Usually this will be a bit slower than if you were in front of the machine, but if it's a local connection you should be fine:

```
$ ssh -X STUDENT\\swales@abyss
```

7.3 Remote Copying

There are two commands that are useful for copying files between machines- `ssh` and `rsync`.

`ssh` works just like `cp`, to target a file on a remote machine add the login address and a `:` in front of the path:

```
$ scp saw562@vayu.nci.org.au:~/kpp.tar.gz .
```

`rsync` is used to synchronise files across two machines. It is similar to `ssh`, however it will only copy across files that have changed. This is helpful if there are only minor differences between directories:

```
$ rsync configs swales@walesnix:configs
```

7.4 Being Nice

When you're running in a shared environment alongside other people you should make sure to not overuse the system so that others can do their work. If you're running a program that will take a long time, like a simulation or data processing, use the `nice` command to say that the program shouldn't hog all of the resources:

```
nice real.exe
```

If you're running jobs on a supercomputer the job queue there serves a similar purpose.

7.5 Networked Machines

The following machines on the Earth Sciences network are available for general use, your supervisor may give you access to other machines:

```
* cove
* gulf
* gyre
* tide
* wave
* vislab01
* vislab02
* vislab03
* vislab04
```

CHAPTER

EIGHT

WORKING WITH MULTIPLE FILES

8.1 Globbing

Many Unix programs can work just as well on multiple files as they can with one. You can either specify files as a list on the command line, or if they match a pattern you can give the program a pattern to match. These are a simpler version of regular expressions.

Pattern	Matches
`abc`	The string "abc"
`*`	Any number of any characters
`?`	A single character
`[abc]`	Either of a, b or c
`{10,20,30}`	Either of 10, 20 or 30

The difference between square and curly brackets is that the former only works with single characters, while the latter works with any number.

Some examples:

Pattern	Matches
`*.nc`	Any file with extension .nc
`201201??.out`	Any of 20120101.out, 20120110.out 20120115.out
`foo{,.bak}`	Both foo and foo.bak

The last example can be useful for creating backups, for instance you can use `cp foo{,.bak}` to create a backup file.

8.2 For loops

If a program only accepts one file at a time you can create a loop that processes each of your files. Loops have different syntax depending on what shell you're using. Say you want to run `echo` once on each filename.

In bash the syntax is:

```
$ for file in *; do echo $file; done
```

In csh the syntax is (separate lines are needed):

```
% foreach file (*)
> echo $file
> end
```