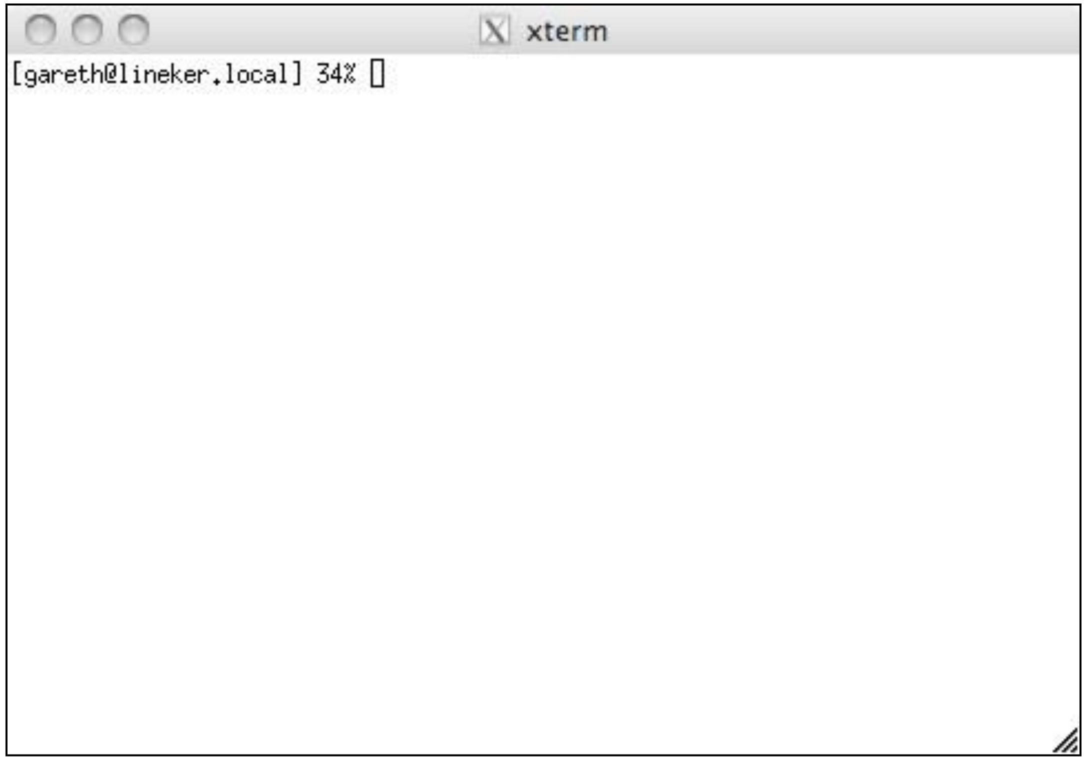
GEO 242: Numerical methods and modeling in the geosciences



```
[gareth@lineker.local] 34% 
```

Lecture 2: The command line and scripting

Outline

Unix concepts

The command line and shell

Help commands

File system commands

(Very) simple scripts

Unix concepts (1): the shell

The shell is a text command line user interface, that allows the running of individual commands, or groups of commands (scripts)

There are two ‘standard’ families of shells, that are found on most Unix systems:

- – the Bourne shell (sh), and descendants (e.g. bash)
- – the C shell (csh), and descendants (e.g. tcsh)

The two shells do most things similarly, but differ in how they implement scripting (some differences in syntax)

bash is the default shell in most operating systems

Unix concepts (2): X-windows

Since the mid-1980s, most Unix operating systems have employed a window-based graphical interface, known as X-windows, X11 or, simply, X

X runs as a command on top of the Unix kernel, and is responsible for drawing and moving windows, and relating them to inputs (i.e. keyboard and mouse)

The windows can contain individual shells, or the outputs of other programs

X is implemented as a client-server system. The client need not be run on the same machine as the server (i.e. you can get graphical output from programs running on one machine, remotely)

Unix concepts (2): X-windows

Mac OS X, somewhat confusingly, does not use X for its graphical management, using a system called ‘Quartz’ instead.

However, Unix-style X-windows can be used in parallel if the optional ‘XQuartz’ program is installed.

Similar systems exist for Windows (‘VcXsrv’, ‘cygwin’), although Windows is not based on Unix, unlike Mac OS X, so Unix features are emulated, not native.

Unix concepts (3): Ownership

Running programs (‘processes’) in Unix are all assigned to users – in most cases, the person that ran them

For system processes (e.g. the X server), the user is ‘root’, also known as the ‘super user’

In most cases, the person that owns the program is the only one, other than root, who is permitted to stop it

Most sensitive system files are also owned by root, which means that they cannot be edited by other users

This system works well for security, as you do not log in to the root account; usually only the system administrator has the root password

Unix concepts (4): Permissions

Permissions in Unix state what a particular user or group of users is allowed to do with a file

Choices are:

- readable
- writeable (deletable)
- executable

And the different classes of user:

- user (the individual that owns the file)
- group (a set of people identified within the operating system)
- all (everyone with access to the system)

Accessing the command line

The program xterm is a common X-windows client program found in many BSD Unixes

On the Mac, it is run automatically when you open X11/XQuartz

A terminal can also be opened by running the ‘Terminal’ program, also found in Applications/Utilities

In Ubuntu, the terminal can be found by searching in the ‘Dash’ for xterm

In Windows, if you have installed the Ubuntu terminal, it can be found in the start menu

Unix help commands

`hostname` – displays computer name

`pwd` – 'present working directory' (gives current directory name)

`whoami` – displays user name

for help with running commands:

`man` – display manual page

in many cases, running a command with no arguments will bring up information on that command

Unix file system commands

cd – change directory

cp – copy file

ls – list files in current directory

mkdir – make a new directory

mv – move file

rm – remove file

rmdir – remove directory

tar – bundle files together into a single file (‘tape archive’)

cd

change directory

```
cd <directory>
```

change to the named directory (can be a local directory – i.e. within the current directory, or a full path)

```
cd ..
```

move back one directory

```
cd
```

move to your homespace

Wildcards

Ways of referring to multiple files or directories:

`*` – everything

`?` – a single ‘wild’ character

`[a-z]` or `[A-Z]` or `[0-9]` – a defined alphabetic or numeric range of characters

These can be combined, e.g.

`[a-r]*.txt` – all .txt files starting with lower case ‘a’ to ‘r’

`????????.dat` – all 8 character .dat files

ls

list files and directories in current directory

```
ls <optional-wildcards>
```

display a list of files and directories in the current directory, ordered in numerical order, then alphabetical order of capitals, then alphabetical order of lower case. Wildcards can be used to narrow down the list

```
ls -l
```

long list, including permissions, file sizes and times

```
ls -t
```

list ordered by time (most recent first)

ls

…continued

```
ls -r <optional-wildcards>
```

list the files in the reverse of their normal order

```
ls -a <optional-wildcards>
```

includes hidden files in the directory listing

Options can be combined, for example:

```
ls -ltra
```

long list, including hidden files, in reverse time order (i.e. most recent last)

cp

copy (or duplicate) file

```
cp <source> <destination>
```

copy the named source file(s) to the named destination; if there is only one file to copy and no directory with the name ‘destination’ exists, the file is duplicated

```
cp <file(s)> .
```

copy the named file(s) to the current directory

```
cp -r <source-dir> <destination-dir>
```

copy source directory and all its files to the destination (‘-r’ means ‘recursive’)

mv

move (or rename) file

```
mv <source(s)> <destination>
```

move the named source file(s) to the named destination; if there is only one file to copy and no directory with the name ‘destination’ exists, the file is renamed

```
mv <file(s)> .
```

move the named file(s) to the current directory

No ‘recursive’ option is required, but if there are multiple sources and the named destination is not a directory, you will get an error

mkdir

make new directory/ies

```
mkdir <directory-1> <directory-2> …
```

make one or more new directories at names/locations specified

rmdir

remove empty directory/ies

```
rmdir <directory-1> <directory-2> …
```

remove one or more directories at the names/locations specified. The directories must be empty.

rm

remove files and/or directories

```
rm <file(s)1> <file(s)2> …
```

remove the named file(s)

```
rm -r <file/directory-1> <file/directory-2> …
```

remove the named files/directories. If the directories contain files or other directories, these are removed too.

```
rm -i <files>
```

the ‘-i’ essentially adds an ‘are you sure?’ to the delete process – rm will ask you if you want to delete each file

tar

bundle or unbundle files, keeping directory structures intact

```
tar cv <files-to-bundle> > <tarfile.tar>
```

takes the input files and bundles them into a single file

```
tar xvf <tarfile.tar> <location>
```

unbundles the files stored in the tarfile to the specified location (if not specified, files are unbundled in the current directory)

```
tar cvz <files-to-bundle> > <tarfile.tar.gz>
```

```
tar xvfz <tarfile.tar.gz> <location>
```

adding a ‘z’ means that tar (un)does compression on the tarfile

Unix concepts (5): Standard input and output

Unix commands typically write any output they have to 'standard output'

Unless specified otherwise, standard output is written to the terminal screen

However, output can be redirected, e.g. to a file, using the '>' symbol, i.e.

```
ls > list.txt
```

which makes a text file containing the standard directory listing

Standard input and output

Standard input is taken from the keyboard, primarily, but can also be taken from files, using the ‘< ‘ symbol, e.g.

```
cat < list.txt
```

which will concatenate the contents of the file fed from standard input and display them to standard output

You can move the output of one command directly into the input of another by using the ‘|’ symbol, i.e.

```
ls | cat
```

which does exactly the same thing as the two steps above.

```
ls | sort -r | tail
```

takes the listing, sorts it in reverse order and outputs the last ten entries to standard output

Scripts

In its simplest form, a Unix script is a set of Unix commands saved together in a text file

If the script file is made executable, or is executed through one of the shell commands (e.g. `sh` or `csh`), those commands will be executed in turn

The only way they will not all be executed is if one of the commands returns an error message, or the processes are canceled somehow

A simple (and useless) script

What will this script do, do you think?

```
#!/bin/sh

echo hello world
```

To execute it, make a text file containing that text, go to the directory containing it and type:

```
    sh <scriptfilename>
```

We could also make it executable…

Text editors

These are some of the most useful add-on programs in Unix/Linux

Two types of text editor exist

– ‘within terminal’ editors that allow editing within the terminal window itself (e.g. vi, emacs, nano)

– ‘X windows’ editors that open a new window containing the editor (e.g. xemacs, gedit, nedit)

(I currently use nano and gedit)

Changing file permisssions

To make a file executable, we need to change the permissions of the file – we use the command `chmod`

```
chmod +x <scriptfile>
```

will change the user permissions to executable

```
chmod a+x <scriptfile>
```

will change the user permissions to executable for everyone

And then you can execute the script by typing its name…

Unix concepts (6): Path

Commands in Unix are themselves executable files – you can find them by executing

```
which <command>
```

When you enter such a command, Unix has a list of places where it will look for executable files – this is called the *path*

You can check it by executing

```
echo $PATH
```

If the desired command is in one of the directories named, it will execute (and if it isn’t, it won’t)

Unix concepts (7): Processes

Each executed program or command in Unix has a process number. You can see these by running

```
ps -e
```

Depending on your operating system, you may either see a lot of processes (Linux, MacOS) or very few (WSL). This is because WSL is emulated, whereas all programs on Macs or Linux boxes are native commands.

An interactive list of processes can be obtained with

```
top
```

Killing processes

If you have a process (job) that is running out of control, you can use its process id (PID) number to kill it

```
kill <PID>
```

Sometimes a polite request to kill a job is not heeded by the shell and you have to be a bit more forceful, in which case,

```
kill -9 <PID>
```

should do the job.

Being nice

If you are on a shared machine, then it is possible to be nice and deprioritize your jobs

```
nice -19 <PID>
```

will set your job to be the lowest priority possible, so it does not completely dominate the machine

Practice!

Download ‘files.tar’ from Canvas

Using the command line:

1) Make a directory on your computer with an appropriate name for this class
2) Make a directory in that directory for lecture1
3) Move the tar file to that directory
4) Unpack the tar file
5) Make three more directories (‘carnivore’, ‘herbivore’ and ‘omnivore’) and move the appropriate animals into those directories