Unix/Linux Practical 2
Advanced Topics

This session is designed to introduce you to the underlying UNIX based filesystem and computing infrastructure used within the School of GeoSciences.

1. Applications And Processes
- How to control applications from the command line. Background processes.

2. Complex And Linked Commands
- Linking together commands you already know, to do real, useful work.

3. Managing Printing in Linux
- How to check on print job progress and to cancel unwanted jobs.

4. Linux Text Editors: GNU Emacs; Controlling the Default Editor
- Introducing the GNU Emacs editor – for authoring content, for data processing, for programming.

This practical will allow you to familiarise yourself with the basics of working with UNIX (Linux) in the School of GeoSciences. The practical workbook is designed for you to work through during the practical session in the lab, when there are people around to help. **It is NOT a definitive reference!** Much more information is available on the web, including that given at:

`http://www.geos.ed.ac.uk/~gisteac/wkzero`

There is a set of attainment targets on the next page. Make sure you can do everything in the list. Many students **will** be using UNIX throughout the year. It is therefore enormously important that you speak to your lecturer/demonstrators if you are confused or have been unable to complete the tasks.

Making sense of this Workbook

Throughout this practical workbook any commands you have to type, will be given in **`bold courier font`**, and prompts or responses from the computer will be in `plain courier font`, for example:

```
[snnnnnnn@server ~]$ pwd
[snnnnnnn@server ~]$ cd /
[snnnnnnn@server /]$ ls -al
```

Other text to note, or buttons to click or be aware of, will be introduced in bold paragraph **body** text.

s*nnnnnnn* represents your unique login (e.g. matriculation number preceded by an **s**.)

NB For staff this may be a first initial immediately proceeded by surname up to 8 characters (e.g. jjohnson, jjohnso1, etc.) and for visitors with an initial v*n*fsurna (e.g. v1jjohns).

~ (known as **tilde**) represents your home directory. In the command prompt above note that this will change to reflect whichever directory you are currently working in.

Note that UNIX is **case sensitive** - all commands must be typed **exactly** as they are shown on this sheet or they will not work!

Tasks for you to do will have a letter in the left hand margin - a), b), c) etc.

Targets for this Session

You should be able to:

- Do everything you did in the previous practical
- Start a **process** in the **background**
- Move an active process to the background
- List the **processes** you have running
- **Kill** a process you no longer need
- Use **pipes** to link commands together
- **Redirect output** from a command to a file
- **Manage print jobs** from the command line
- Edit simple text files using **Emacs**

And you should still know:

- When to switch off the machine and go outside…

1 Applications And Processes

Earlier you started **emacs** from the command line. What happened to the command line? It locked – you could write a bestseller, but couldn't type any further *commands* until you closed **emacs**. The way to get round this (short of opening multiple **Terminal** windows which can be hard to manage) is to use Linux/UNIX's ability to run programs as separate **processes** and to run specific applications either in the **foreground** or the **background**. The simplest way to show this is with an example. Today we will use the fun old graphical UNIX program **xeyes** here, however the principles are the same for any software. So:

→ Log on to the Linux server using **xrdp** as before – in a lab simply entering the host **xrdp.geos.ed.ac.uk** should suffice in order to allow you to login.

Now run **xeyes** at the UNIX command line:

→ `[snnnnnnn@server ~]$ xeyes`

A new window should pop up and the command line should "freeze" – nothing you type will make any difference. You need to get control back. So…

→ Press **CONTROL ([Ctrl])** and **Z** at the same time on the keyboard. (Click on the **Terminal** window to re-activate it if necessary.)

a) What happens? (If nothing happens you may need to click on the **Terminal** window first.) What messages do you get? What does **xeyes** do when you click on it? Do the eyes still follow your mouse pointer?

……

You have **stopped** the **xeyes** process – but that's not much use either, is it? So, type:

→ `[snnnnnnn@server ~]$ bg`

This puts **xeyes** in the **background** – you can work with it but you can also work with the command line. **xeyes** is now running as a background **process**. You can bring **xeyes** back to the foreground by typing:

→ `[snnnnnnn@server ~]$ fg`

→ Now close the program by pressing **Ctrl-C** (or you can click the **Close** gadget). An easier way to achieve all of this would have been to type:

→ `[snnnnnnn@server ~]$ xeyes &`

in the first place – this means that **xeyes** starts as a background process, freeing the command line to do something else, or allowing you to safely log out and go home!

xeyes is (are!) quite happily sitting in the background – but what happens if something goes wrong? What if **xeyes** (or a really important application) stops working? How do you control it if none of the buttons respond? This can be a problem with scientific or data analysis software if you are using large datasets.

There are two commands that are very important for this: **ps** and **kill**.

→
```
[snnnnnnn@server ~]$ ps
  PID TTY          TIME CMD
18888 pts/175  00:00:00 xeyes
18890 pts/175  00:00:00 ps
19274 pts/175  00:00:00 bash
```

ps tells you the processes you have running on a machine. The command above is very useful – it uses a flag so that you can find **all** the **processes belonging to a user**. Try typing **ps -u snnnnnnn** where s*nnnnnnn* is, here, the username of your neighbour.

b) How many processes are they running and what are they?

……

……

The PID column tells you the process ID of a specific task. In the example above, **xeyes** has a **PID** of **18888**. We can close down this process from the command line with the command **kill**. Try shutting down **xeyes**. Replace the PID of the example below with your own PID from the previous **ps** command.

→
```
[snnnnnnn@server ~]$ kill -9 18888
```

xeyes should now disappear. What does the **-9** do? Stops it dead, no argument, no messing about. But, if a process has **children** (e.g. secondary processes or other sub-windows), this may not kill them – you will have to do this manually.

NB Before moving on you should check that you do not have any background **xeyes** processes still running! If so, you can safely **kill** these in the same way.

It is a good idea to check you have no processes running in the background before you log off! One user was found to be hogging 25% of the processor time of a UNIX machine, even though they had not sat at it for two weeks, because of unkilled background processes!

2 Complex and Linked Commands

Pipes

Pipes allow you to link commands together so that the output from one command is used as the input to another.

→ `[snnnnnnn@server ~]$ ps -ef | grep snnnnnnn`

where s*nnnnnnn* is the username of your neighbour.

c) What does it do? Clue: **`ps -ef`** lists all the processes running on the machine and gives a long description of them.

……..

……..

Try this one:

→ `[snnnnnnn@server ~]$ finger | grep 's1'`

d) What does it do (remember to look at **`man`** for help on what a command does.) Ask if you're not sure.

……..

……..

Long Command Lines and Multiple Commands on one line

To split a command across more than one line you use the **`\`** character (see the next page for an example involving the **`grep`** command). This can be very useful if you have long arguments.

You can also put more than one command on a line if you split them up with the **`;`** character. For example:

→ `[snnnnnnn@server ~]$ cd /; pwd; cd ~; pwd`

(Note that / signifies the root or top(!)-level directory on the UNIX system – an *inverted tree*!)

e) What do you think this does? What output do you get?

……..

……..

Redirecting Output

Nearly all the commands you have used so far have returned some information to the screen. Most of the time, this is fine, but you may want to store this information for future reference. You can do this by **redirecting output** from a command to a file, using the **>** character. Try the following:

→ [s*nnnnnnn*@server ~]$ **cd wkzero**
→ [s*nnnnnnn*@server wkzero]$ **who > users.txt**

→ What has happened? Nothing is returned to the screen. The information has been stored in a file called **users.txt**. Look at this file using **more** or **less** or another viewer, in Windows if you wish, and check it contains the correct information. Note this file is simply stored in the same directory (i.e. the current directory). If we wished to store the file elsewhere we could provide a fuller path in our redirection command, e.g.

EXAMPLE

```
[snnnnnnn@server wkzero]$ who > ~/userlist.txt
[snnnnnnn@server wkzero]$ more userlist.txt
userlist.txt: No such file or directory
[snnnnnnn@server wkzero]$ more ~/userlist.txt
.
.
.
Etc.
```

Compound Commands using \ – An example with grep

All of the examples above have been very simple. You are now going to use a combination of these to do some useful work. In **/geos/netdata/wkzero/** there is a file called **nation_data.txt**. The first few lines are shown below.

→ [s*nnnnnnn*@server wkzero]$ **head /geos/netdata/wkzero/nation_data.txt**

```
Nation  Continent       Area (km2)
Greenland       Europe  4501027.684
Canada  North America   472653.2342
Canada  North America   98308.28442
Norway  Europe  38767.3129
Norway  Europe  87746.89587
Canada  North America   27865.78227
Canada  North America   11483.96442
Canada  North America   8817.206272
Russia  Asia    26852385.1
```

This file was exported from Microsoft Excel. It is a tab-delimited file (i.e. the columns of data are separated by tabs). The first line shows the titles of the columns and the following lines record the nation, continent and area in square kilometres of all the polygons (islands and land masses) in a map of the world. The file is 275 lines long (measured with **wc**).

What you are going to do is **find all the land masses (polygons) in North America with an area greater than or equal to 100000 square kilometres and store their nation name, continent, and area in a new file**.

You could do this by opening the file in a text editor and searching through it, but there is a much faster and more elegant way… Type the command below (hitting **<return>** at the end of each line) – remember you can use **<tab>** to auto-complete paths e.g. **/geos**… etc.

DO NOT type the > brackets marked on the left. These will appear in place of the usual bash$ command prompt when you continue a long command on a new line.

```
→ [snnnnnn@server wkzero]$ grep 'North America' /geos/netdata/wkzero/nation_data.txt | \
> grep '[1-9][0-9][0-9][0-9][0-9][0-9]\.' > \
> nation_summary.txt
```

Now have a look at the new file **nation_summary.txt**. Does it answer the question above? The command looks very complicated so the following explanation is line by line:

```
[snnnnnn@server wkzero]$ grep 'North America' /geos/netdata/wkzero/nation_data.txt | \
```

This line searches the file **/geos/netdata/wkzero/nation_data.txt**, looking for all the lines in it (i.e. representing each distinct polygon, island, or land mass) that have the text **North America** in them (listed under the heading *continent* that is – which almost explains why Russia is amongst them!). The pipe **|** symbol towards the end of the line instructs UNIX to take the output from this command and send it to another command, rather than the screen – in this case the next command is on the following line. The **** symbol tells UNIX the whole *compound* command continues on another line…

```
> grep '[1-9][0-9][0-9][0-9][0-9][0-9]\.' > \
```

Note the **>** symbol at the *start* of this line. This is **NOT** a redirect symbol; it is merely a marker showing that the command is a long one split across two or more lines. It is **NOT** part of the command and should **NOT** be typed!

This line searches the output from the first **grep** command. This time it looks for lines that include a **decimal point preceded by six or more numbers**. This is an example of using a **regular expression** to search. Each set of square brackets and numbers **[0-9]** means "any character between **0** and **9**" i.e. **0 1 2 3 4 5 6 7 8 or 9**. **NB!** The first is, of course, **[1-9]** since we wish to find land masses greater than or equal to **1**00000 sq km. The full stop or period **.** has a special meaning when it is in a regular expression (see **http://unixhelp.ed.ac.uk/**) so to search for it you need to put a **** before it. This means "do not interpret the next character as a special character, just as a normal character". (This is in fact also what the end of line backslash does – it tells UNIX to ignore the newline entered when your press **<Return>** or **<Enter>**.) The sets of square brackets and full stop together mean six numbers followed by a decimal point, so this command line searches and finds all the lines of text that have a number in them greater than or equal to 100000. The **>** symbol at the *end* of the line is this time an actual user-specified instruction which **redirects** the output into a file, shown on the final line:

```
> nation_summary.txt
```

3 Managing Printing with Unix

Monitoring Printer Queues

In the University cloud printing the following may be of limited direct use however the commands are worth knowing should you work on another Linux/UNIX facility and have need to control printing or perhaps to use print commands programmatically.

You can monitor the progress of jobs using the **lpq** command. This shows the list of pending print jobs for your default printer. For example:

→
```
[snnnnnnn@server wkzero]$ lpq
Cloud-Mono is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  snnnnnn 234253  nation_summary.txt              1024 bytes
```

Or if you try it when there are no jobs pending:

→
```
[snnnnnnn@server wkzero]$ lpq
Cloud-Mono is ready
no entries
```

You can also monitor printers other than your default using the **-P** option to specify the relevant printer. For example:

```
[snnnnnnn@server wkzero]$ lpq -Pprintername
printername is ready
```

Removing Print Jobs from the Queue

To remove a job from the queue (perhaps if you can't afford to wait if the printer is busy and you wish to try submitting to another printer) you should note the job number. You can then use the **lprm** command, specifying the job number, to delete the job from that queue.

(→) For **EXAMPLE**:

```
[snnnnnnn@server wkzero]$ lpq
Cloud-Mono is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  snnnnnn 234253  nation_summary.txt              1024 bytes
[snnnnnnn@server wkzero]$ lprm 234253
```

NB With the current version of **lprm** no feedback is given unless the job has completed before you issue the command. This scenario is most likely with the fast new Cloud *virtual* printers! In this case you will receive a message such as:

```
lprm: Job #420021 is already completed - can't cancel.
```

The above operations can also be controlled (*in theory* – permissions permitting and may currently not be allowed!) through the GeoSciences CUPS web interface: **http://cups.geos.ed.ac.uk:631**. There are, however, some further UNIX command line tools which may be useful…

Advanced UNIX Printing

Printing text to laser printers – e.g. program code listings

NB Don't worry about printing to Cloud printers – as with printing from Windows you are only billed should you choose to physically print from one of the multi-function devices!

When printing computer code listings you must format all code in a non-proportional font such as **Courier** to retain the *fixed-width layout* used. This is essential for accurate reading and analysis of the code. A good text editor (e.g. **`emacs`**, or **PSPad/NotePad++** on Windows) will allow this, however if you are laying out code as part of, say, a **Word** or **OpenOffice** document, you will have to *manually format* the listings with the correct font.

It is also possible to print such listings from directly in Linux/UNIX. This can be useful if there are problems printing from Windows or where you may wish to program a print as part of a series of automated tasks (this is the power of a command-line environment.) You can simply print to a postscript print file using **`lpr`** (or **`lp`**) and use the relevant Linux/UNIX command to submit the file to the printer.

You can also use the printing commands either interactively, typing a series of lines and then **[`Ctrl-D`]** to finish and submit to the printer, or by specifying a text file to print. E.g.

→
```
[snnnnnnn@server wkzero]$ lp nation_summary.txt
request id is Cloud-Mono-234253 (1 file(s))
```

`lp` and **`lpr`** use different letters for specifying some printing options, but are virtually identical. **`lp`** though does provide useful feedback when a job is submitted unless run with the silent flag **`-s`** whereas **`lpr`** runs 'silently' at all times.

Sophisticated printing is easily achieved using the various options available to us when specifying one of the printing commands. For instance to print two pages to an A4 side we can issue a command like:

EXAMPLE

```
[snnnnnnn@server wkzero]$ lpr <file> -Pprinter -o sides=two-sided-long-edge
```

The **`a2ps`** utility is also useful for compressing long code for document production since it prints a reasonably attractive listing in two columns with the filename as a heading.

EXAMPLE

→
```
[snnnnnnn@server wkzero]$ a2ps nation_summary.txt
[nation_summary.txt (plain): 1 page on 1 sheet]
request id is Cloud-Mono-234260 (1 file(s))
[Total: 1 page on 1 sheet] sent to the default printer
[snnnnnnn@server wkzero]$
```

4 Linux Text Editors: GNU Emacs; Controlling the Default Editor

Various editors are available for use on the School's Linux machines. These consist of at least **`vim`**, **Emacs**/**`emacs`** (GNU Emacs – will work with X display or in a terminal window), **`nano`** and **`pico`**! You can see many of these from the menus when logged into the xrdp environment.

The *default* editor, sometimes invoked by other programs e.g .SQL Plus (**`sqlplus`**) as required, is the incredibly simple text-only editor **`pico`**. While this can be invaluable when working command line only (i.e. with no access to mouse-driven menus) since it provides basic instructions at all times, we will show you how you can change the default editor, should you wish to do so.

We recommend that you use **Emacs** on the School Linux systems as it is straight-forward to use yet offers a sophisticated graphical system when using **xrdp**. We will use **Emacs** in most MSc computing teaching and will look at this now, however we have also included a brief **`pico`** reference as an appendix to this document in case it proves a life saver!

Emacs has an incredible heritage, offers incredible power yet is easy to get started with – particularly via xrdp. However if you are wondering why we can't use a lean modern editor or simply a Linux version of some popular Windows/Mac text editors then you might find the following informative.
https://www.quora.com/Why-do-I-need-to-use-Vim-or-Emacs-in-2018
http://emacs-fu.blogspot.com/

A First Look at GNU Emacs

Start **Emacs** from either of the **Applications > Accessories** or **Applications > Programming** menus.

Creating a new text file (document)
Click on the left-most *icon* (button) to create a new file. You will be asked for a location and filename. You can browse the filesystem to a suitable location (you will start off browsing in your home directory) and then specify a filename. The file is only created there once you add some text content and go to **File > Save** or click the **Save** icon. Instead of using the icons (hover over each to find out what they do) you can use the menus entirely – e.g. **File > Visit New File** to do the same as we have just done above, or use keyboard short-cuts.

Opening and Saving Files
A simpler alternative is to make a change to an existing file which you can do from the menus via **File > Open File**. To save any changes to this, or equally to save a newly created file that you have added content to, you can go to **File > Save**.

Note that you can **Save As** from an unused empty file which will simply effectively rename the file so long as no content has been added.

The best option is just to play with the program to get used to the necessary basics which shouldn't take too long! There is however a built-in tutorial available from the **Help** menu.

Handy Keyboard Shortcuts (Refer back to these when required!)
No mouse? Command line access? Remote working and no **xrdp**? **Emacs** menus = **F10**!!

As well as using the menus you can make use of a number of quick keyboard shortcuts which are worth remembering. (**NB** These are especially worth learning if you make **Emacs** your default editor should you ever wish to connect with a text only Terminal! However, if you remember nothing else **F10** will get you to the menus where you can see the shortcut combinations. None of this is an issue if you use xrdp and control the screen size/colour depth if bandwidth is an issue, but it is worth mentioning here so you can refer back.)

Most shortcuts consist of pairs of key combinations usually involving one or more **control**-***key*** pairs perhaps followed by one further ***key*** press – e.g. to **Save** hold down **control** (**ctrl**) and press **x** (while still holding **ctrl**) then release both and immediately press and hold ctrl once more whilst pressing **s**. The most useful combinations are outlined below.

NB In **Emacs**' menus **Ctrl** (**control**) is shown by **C**. You may also see **M** meaning the **Escape** (**ESC**) key. A full tutorial can be found within the program from **Help > Tutorial** (or **C-h t**, i.e. **Ctrl+h** then press **t**).

Save	Ctrl+x then Ctrl+s (C-x C-s)
Save As	Ctrl+x Ctrl+w (C-x C-w)
Quit	Ctrl+x Ctrl+c (C-x C-c)
Visit New File	Ctrl+x Ctrl+f (C-x C-f)
Open Directory	Ctrl+x then d (C-x d) NB Only one use of Ctrl here!

There is no **Open File** shortcut since if no Window system were available the program could be run specifying a file to edit (either new or existing). The **Visit New File** function will either create and open a new file for editing or open an existing file in the same way as **Open File**.

And from the **Edit** menu:

Cut C-w	C-w
Copy M-w	M-w (**NB** M is the Escape or ESC key; in **Emacs** also **Alt**)
Paste C-y	C-y
Undo	C-x u
Select All	C-x h

Plus the following are invaluable – but not mentioned on the menus!!

Kill (a single line)	C-k
Set Mark (block editing)	C-SPC (SPC = Space) – use cursor keys then cut/block cut
Block Cut	C-x r k (Copy using cut, paste, then move to paste again)
Block Paste	C-x r y (y means yank as in yank out in **Emacs** speak!!)

A Task for You

Time to get you active again and give the brain a workout!

Try editing (correcting!) the file **`jefferson.txt`** that you downloaded in the previous session via FTP. (Alternatively you can find the file **`if.txt`** on netdata should you need to.)

If you struggle to find the file remember you can use the **`tree`** command to see if you have the folders we are expecting, or alternatively to examine any folder or directory structure. If you don't have the **`jefferson.txt`** file don't worry just copy the **`if.txt`** file from **netdata** instead.

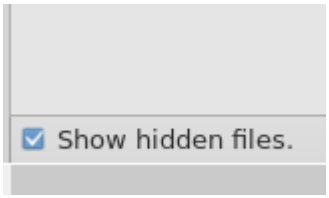
```
[snnnnnnn@server wkzero]$ cd ftpdata
[snnnnnnn@server ftpdata]$ emacs jefferson.txt
```

Alternatively start **Emacs** from the menus and use **File > Open File**.

Setting the default editor in Linux

Should you wish you can upgrade your default editor from **pico** to **emacs**/**Emacs**. To do this you need to edit a special hidden file (remember – that's one that starts with a dot – see the earlier practical if unsure).

You also need to make these files visible in **Emacs** by ticking the **Show hidden files** box to the bottom left of the **Find File:** dialog box.



The file to be edited is called:

`.bash_profile`

Simply add the following line at the very end of the file (be careful not to change anything else!)

```
export EDITOR=emacs
```

`EDITOR` must be in capitals and is a Linux/UNIX environment variable.

Once saved emacs will become the default editor in every *new* Terminal session launched.

That will suffice for this session – hopefully you will find this material of use in weeks to come!

APPENDIX: PICO (Default Minimalist Editor – For Reference)

Text can simply be typed on the screen and the cursor and **<backspace>** keys used to delete as normal. (Some editors *only* offer their own unique key combinations for navigating and editing files which while, very effective, have an extra initial learning curve.)

PICO functions involve holding down the [Ctrl] key (denoted in the PICO Help as the ^ symbol) while pressing another key. Several of the more popular functions are shown at the bottom of the editor window. This bottom-menu is context-sensitive, that is it changes depending on what you are doing.

→ `[snnnnnnn@server wkzero]$` **`pico`**

→ …starts PICO with no file (blank screen). Hold down **`Ctrl`** and press **`x`** to Exit (**`[Ctrl-x]`**). Now let's move to a suitable directory and look at one of our files from earlier.

→ `[snnnnnnn@server wkzero]$` **`cd ftpdata`**
→ `[snnnnnnn@server ftpdata]$` **`pico jefferson.txt`**

…starts PICO with an existing file or creates a new file if the given filename is not found. In this example the file **`jefferson.txt`**, which you should have from the previous session, will be opened for editing. You can **`Save and Exit`** with **`[Ctrl-x]`** – filenames etc. will be prompted for (hit **`<Return>`** if editing an existing file).

You can get help with **`[Ctrl-g]`**.

Using **`[Ctrl-k]`** cuts lines into a buffer and **`[Ctrl-u]`** retrieves (or un-cuts) all of the cut lines. To copy a line or several lines, you need to cut them, paste them back where they were and paste them again where you want the copy. This is a slightly clumsy way of copying-and-pasting, so you can also set a mark (**`[Ctrl][Shift]6`**), move up or down with the cursor keys, such that a block of text is highlighted, and then cut that whole block **`[Ctrl-k]`**.

Use these operations to remove the first line in *jefferson.txt* and fix the errors which have crept into the text. Try not to simply use the cursor keys and delete, instead use cut where more appropriate. For even more practice you can use the file **`if.txt`** in your **`wkzero`** folder.

PICO QUICK COMMAND REFERENCE

Function	Command
General Commands	
Get help	[Ctrl] g
Write the editor contents to a file	[Ctrl] o
Save the file and exit pico	[Ctrl] x
Spell check	[Ctrl] t
Justify the text	[Ctrl] j
Moving around in your file	
Move one character to the right	[Ctrl] f or the right arrow key
Move one character to the left	[Ctrl] b or the left arrow key
Move up one line	[Ctrl] p or the up arrow key
Move down one line	[Ctrl] n or the down arrow key
Move to the beginning of a line	[Ctrl] a
Move to the end of a line	[Ctrl] e
Scroll up one page	[Ctrl] y
Scroll down one page	[Ctrl] v
Get the position of your cursor	[Ctrl] c
Searching for text in your file	
Find [pattern]	[Ctrl] w
Cutting, pasting and deleting text	
Append another file at current cursor position	[Ctrl] r
Cut a line of text	[Ctrl] k
Paste a line of text	[Ctrl] u
Mark a block of text	[Ctrl] ^ (shift 6), then move cursor to the end of the text to be marked