Linux Practical 2
Advanced Topics

This session is designed to introduce you to more sophisticated usage of the underlying UNIX-based Linux file system and computing infrastructure used within the School of GeoSciences.

1. Applications and Processes
- How to control applications from the command line. Foreground versus background processes.

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

3. Editing Text Files: Nano; Text/Code for Reports; UNIX vs. DOS
- Introducing `nano`, a simple editor also used by apps; Text into report docs; Unix vs Dos format.

4. Managing File and Folder Permissions; Groupwork Management
- Managing file permissions in Linux with `chmod` for multi-user access or security settings.

APPENDIX A: Web Pages in GeoSciences; Making Web Folders Appear!
- A useful quick guide for those likely to use the GeoSciences File-Based Web Server (e.g. TIGIS)

APPENDIX B: Managing Printing in Linux [REFERENCE ONLY]
- How to check on print job progress and to cancel unwanted jobs.

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

www.geos.ed.ac.uk/~gisteac/wkzero

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:

```
s1234567@baltic10:~$ pwd
s1234567@baltic10:~$ cd /
s1234567@baltic10:/$ ls -al
```

→ Will indicate tasks to be completed or key learning points

(→) May be used to indicate useful but optional tasks or related info

Specific questions you can answer will have a letter in the left hand margin - a), b), c) etc.

You can record answers or related notes in the spaces provided (if you choose to print these notes), or in e.g. a text file or document see the box below on making no

 This symbol indicates text of high importance or direct usefulness during the practical.

 This symbol indicates further information it would or might be useful to know later.

s1234567 represents your unique student login (matriculation number preceded by an **s**.)

Note: For staff this may be 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 Linux is **case sensitive** - all commands must be typed **exactly** as they are shown on this sheet or they will not work!

How to make notes/record answers during practicals in an online age

While you may print out the practical notes and annotate them, or use the spaces provided to answer any questions, you may wish to work efficiently wholly on-screen. You can arrange multiple windows side by side (2x2 or even up to 4x4) by holding down the Windows key, then using the cursor keys to arrange the windows. This often also works for/on remote systems so you may first need to click **Restore Down**  on the remote window *blue bar* to be able to move it this way. **Note:** Use the lower horizontal *scrollbar* on the remote Linux system to access the whole desktop, or **Maximise** as required. If you need to you can also drag window sides to resize in other configurations (e.g. 3x3).

Better however is to make notes in a text file *directly* on the remote Linux desktop and arrange your program windows side by side entirely *within Linux* – very effective for working! You can use **LibreOffice Writer** from **Applications ► Office** (and can save in Word *.docx* format if preferred) or use another text program. Save to **Home** or **Home ► Documents** to save within your GeoSciences home directory space (M: under Windows).

There follows a set of attainment targets listed below. Make sure you can do everything in the list. Many students **will** be using Linux throughout the year. It is therefore important that you speak to one of the demonstrators if you are confused or have been unable to complete the tasks.

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 text files in the Linux environment
- Know how to change your default editor for command line use by 3rd party apps

And you should still know:

- When to switch off the machine and go outside…

1 Applications and Processes

Earlier you started **xcalc** from the command line. What happened to the command line? It locked – you could compute endless sums, but couldn't type any further *commands* until you closed **xcalc**. While with XRDP we can simply open more **Terminal** windows, too many windows quickly becomes hard to manage and is no solution for automated processes which may need to open a child (or sub)process without preventing their own progress.

The correct way to get round this is to use Linux'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 – see the previous Intro to Linux pratical or **https://blogs.ed.ac.uk/geosciences-it-help/remote-desktop/** if you need a reminder how.

→ Now let's run **xeyes** at the command line. Open a **Terminal** from the menu bar, then type:

→ `s1234567@baltic10:~$ xeyes`

→ A new window should pop up with a pair of eyes – try moving your mouse pointer over and around them. The command line meanwhile should "freeze" – nothing you type will make any difference. At some point some you will need to get control back (or be aware that closing the Terminal will kill any sub-process launched from that Terminal window!) So…

→ If necessary, first click on the **Terminal** window to re-activate, then press and while still holding **Ctrl** (Control) press **Z** at the same time. We say press **Ctrl+Z** (or **Ctrl-Z**) at the command line. On a Mac use the **Cmd** (Command) key instead of Control (Ctrl).

a) What happens? (If nothing, you may need to click the **Terminal** window first.) 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:

→ `s1234567@baltic10:~$ 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:

→ `s1234567@baltic10:~$ fg`

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

→ `s1234567@baltic10:~$ 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**.

→
```
s1234567@baltic10:~$ ps
    PID TTY          TIME CMD
  21830 pts/1    00:00:00 bash
  79677 pts/1    00:00:00 xeyes
  79695 pts/1    00:00:00 ps
```

→ **ps** tells you the processes you have running on a machine. This command is very useful – it can also use a flag so that you can find **all** the **processes belonging to a user**.

→ Try typing **ps -u s1234567** where s1234567 is, here, the username of a classmate (you can use command **top** to obtain a list of the most resource-intensive running processes and who is running them if required – this updates constantly so press **q** to quit.). The form **ps -u** will also give you a complete list of your own processes, revealing any you have forgotten!

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.

→ s1234567@baltic10:~$ **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 still not kill them – you will have to do this manually.

→ **Note:** You should only use the **-9** option *if you really have to* – do try the softer option first!

Note: 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 in the School was found to be hogging 25% of the processor time of a UNIX server machine, even though they had not connected to it for two weeks, just 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. This allows a huge amount of data processing power in very few lines.

→ `s1234567@baltic10:~$ ps -ef | grep root`

→ where root is the username of the file system owner and thus bound to return some results!

c) What does it do? Clue: **`ps -ef`** lists all the processes running on the machine and gives a long description of them. Try running this first before adding the **`| grep root`** part to see.

……..

……..

→ Try this one:

d) `s1234567@baltic10:~$ ls /home | grep 's2'`

→ What do you think it does? Try breaking it down into smaller steps either side of the pipe.

……..

……..

Long Command Lines and Multiple Commands on one line

→ To split a command across more than one line you use the **** backslash (see the next page for an example involving the **`grep`** command). This is 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 (Note that a forward slash ***/*** signifies the root or top(!)-level directory on UNIX/Linux systems – an *inverted tree*!):

→ `s1234567@baltic10:~$ cd /; pwd; cd ~; pwd`

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:

→ `s1234567@baltic10:~$ cd my_wkzero`
→ `s1234567@baltic10:~/my_wkzero$ ls > filelist.txt`

→ What has happened? Nothing is returned to the screen. The information has been stored in a file called **`filelist.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.

```
s1234567@baltic10:~/my_wkzero$ ls > ~/my_wkzero_list.txt
s1234567@baltic10:~/my_wkzero$ more my_wkzero_list.txt
more: cannot open my_wkzero_list.txt: No such file or directory
s1234567@baltic10:~/my_wkzero$ more ~/my_wkzero_list.txt
```

Creating *Index* pages for web folders (for aesthetic or security reasons)
Redirecting ls output to a file is useful if creating ***index.html*** pages for directories on a web server. Presence of such a file prevents the directory being listed across the web. Simply redirect the ls output to an index.html file then add a little HTML as required to turn the file and folder names listed into links. Some taught MSc students will be provided web folders on the School file-based web server. Others can request this from IT.

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.

→ `s1234567@baltic10:~/my_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.

→
```
s1234567@baltic10:~/my_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:

```
s1234567@baltic10:~/my_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 Linux 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 Linux 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**. **Note:** 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 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 Linux 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 Editing Text Files: Nano; Text/Code for Reports; UNIX vs. DOS

Nano: Simple and Quick Editing, e.g. code scripts/SQL queries

→ The editor nano is recommended for general use as it is incredibly simple yet highly capable. It is also set within the School to be the editor used by some other applications, e.g. Oracle.

→ Text can simply be typed on the screen and the cursor and **Backspace/Delete** keys used to edit text 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.)

→ Nano functions involve holding down the **Ctrl** key (denoted on-screen or in help as the ^ symbol) while pressing another key (similar to process-control at the Linux prompt). 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.

→ `s1234567@baltic10:~$ nano`

→ …starts the program with no file (a blank 'screen' or window). Hold down **Ctrl** and **x** to exit (**Ctrl-x**). Now let's move to a suitable directory and look at a file from earlier.

→ `[snnnnnnn@server wkzero]$ cd data`
→ `[snnnnnnn@server data]$ nano jefferson.txt`

→ This starts the editor 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.

→ (*If need be you can re-download and unzip this from the main wkzero practicals page at **www.geos.ed.ac.uk/~gisteac/wkzero/practicals/index.html**).

→ For simple edits using the cursor (arrow) keys and backspace (delete backwards) or delete (delete forwards) will suffice but there are some handy key shortcuts worth knowing. You can get help with **[Ctrl-g]**.

→ To delete a whole line you can use **Ctrl-k**. You can also use this to copy a line by cutting, then immediately pasting again using **Ctrl-u** for "uncut"! You can then uncut (or paste in normal language) as many times as you like or wherever you like until you cut something else in its place. To copy and paste a block of text you can set a mark with **Ctrl-Shift-6** then move up or down with the cursor keys as required to highlight a block of text, before cutting that block with the usual **Ctrl-k**. You can paste the block as usual with **Ctrl-u**.

→ You can **Exit** with **[Ctrl-x]**, you will then be asked whether you wish to save your edits or not (n for No, or y for Yes) – very easy! You can also save without exiting using **Ctrl-o**. You will be prompted for the name of the file, simply hit enter if you wish to overwrite the existing file, or change the filename as appropriate. You can hit **Ctrl-c** to cancel saving.

→ Use these operations to remove the first line in *jefferson.txt* and fix the errors found in 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.

Incorporating Code Listings/Text Files in Written Reports/Docs

→ When printing computer code listings you should format all code in a non-proportional font such as **Courier** to maintain code readability (showing that it clearly is *code* rather than prose text) and to retain the *fixed-width layout* used in a code context. This is essential for accurate reading and analysis of the code and ensures that any indenting of text (allowing text to be arranged in blocks depending on function – e.g. in Python code) is preserved. A text editor will preserve the column alignment due to its use of fixed-width fonts where each character (letter, number or symbol) takes up the same space horizontally.

→ When pasting text into a written report in a word processor (e.g. Word, Libre Office Writer) you may need to ensure such indenting (see the if statement in the below example) is correctly shown. This may involve adjusting the formatting or justification (left, fully-justified etc.) of the text, or manually changing the font used/applied.

Example Python code:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

→ **Inserting a Text File**
As well as copy and paste, you can directly insert a whole text file into a report (e.g. in Word by going to **Insert ► Text ► Object (drop-down arrow to right) ► Text from File**.

UNIX vs. DOS Text Formats: End of Line (EOL) Markers

→ Something you should be aware of is that Windows and Linux handle judging where the end of one line of text ends and another begins differently. Each operating system uses one or more hidden special control characters (a, b, tab, newline etc. – all are represented by different numeric codes or character encodings).

→ Linux uses UNIX format which marks ends-of-lines (EOL) with a single newline character (linefeed or LF), while Windows uses DOS (its command line) format which involves two characters (linefeed LF *and* carriage return – think old typewriters or printers – CR).

→ If you are running a process on Linux that involves data in plain (human-readable) text format and/or text format control or specification files and things are not working as they should, try and see if the various text files are in UNIX format or DOS format.

→ You should be able to convert in a good text edtor, including nano. To do this you can toggle between formats at the choosing a filename stage: use **Esc-D** (shown as M-D where M is short for Meta(key) – generally meaning the Escape key) to do this. Note: nano will automatically convert DOS format files back to UNIX format upon loading. Try the following to see (you will need to copy these to your own space if you wish to save):

→ [s*nnnnnnn*@server wkzero]$ **nano /geos/netdata/wkzero/dostxt.txt**
→ [s*nnnnnnn*@server wkzero]$ **nano /geos/netdatawkzero/unixtxt.txt**

→ Finally, note the **dos2unix** and **unix2dos** commands available upon the School servers.

4 Managing File and Folder Permissions; Groupwork Management

Managing File/Folder Permissions with chmod

→ UNIX and Linux file systems offer a relatively easy yet highly sophisticated means of controlling various file permissions – the ability for the computer to read a file, write to or edit/delete a fie, or execute or run a file (i.e. to process or follow the contents as a set of instructions for action rather than merely data or textual information to be stored.)

→ We looked briefly at these in the first practical however we will quickly introduce the **chmod** (*change file mode bits*) command used to control the file or folder permissions.

→ Each file has a set of permissions for its owner, i.e. the owner can ask for the file to be read, written to (or deleted), or run (executed as a set of computer instructions). Additionally each file/folder belongs to a UNIX group (usually this is set up as one group per owner for security reasons but can be set to be one of any number of groups – useful for collaborative working, or teaching, e.g. group netdata. Finally each file/folder can have a set of permissions for all other users on the system (global other settings – effectively world in a web context).

→ Changing these may be required on occasion – e.g. if working on data processing projects as part of a team or group, or if working on web development (web scripts are not allowed group write access for instance which must be removed for the script to run on the web server). This can be achieved using the chmod command.

→ The **chmod** command can take a range of numbers (a binary digit for each of read, write and execute for each of owner, group and other. This is relatively easy to understand but hard to explain (r,w,x are represented as 4,2,1 to give an overall octal – base 7 number from 0 to 7 for each of user, group, other, e.g. **chmod 644 <file>** sets user read and write (4 and 2), but others read only, **755** means user all permissions (read, write, execute), with all others (group, other) having read and execute permission only (i.e. they can't edit or delete the file).

→ Look at **/geos/netdata/wkzero/644example** and **755example**. Use **ls -l** to see.

→ Much easier however is to use the textual form of e.g.:

→ **chmod g-w <file/folder>** (group minus write), or **u+x <file>** (user plus execute – needed to make a new code, e.g. Python, script runnable) etc. See the chmod man pages for more detailed information.

→ Try creating some files in **nano** for practice (and possibly a folder using mkdir) in your own space and experiment with the `chmod` command – don't be scared!

Managing Group Project Work

On some systems you may need to issue **chmod** commands if working collaboratively with complex data formats, e.g. GIS data. With modern online systems and configuration tools such challenges may be less but it is still worth knowing the tool. For instance, if working with ESRI Grid format (still out there!) it might be necessary to issue a **chmod** command recursively (**-r**) on all files (*****), or at least on the contents of both the grid folder and the shared info folder used to index all grids and related info. Or use a raster format such as ***tif***!

APPENDIX A: Web Pages in GeoSciences; Making Web Folders Appear!

GeoSciences File-Bassed Web Server

Space on the GeoSciences Linux File-Based Web Server (a simple apahce web server) is available to members of the School; see more at **https://blogs.ed.ac.uk/geosciences-it-help/personal-web-space/file-based-web-service/**. This is also used on some courses involving web development or basic coding for the web.

There is one complexity with web folders in that for reasons of security or resource, or both, they will only appear in a File Browser (Caja) or by typing ls /web/ upon having been first directly accessed.

What this means:

*In a file browser (e.g. **Caja**):*

To access your web browsers you can navigate to FileSystem (the root or / location), then click on the web folder. If you see a folder with your username (s1234567) great you can click on this to enter. This may happen if you have previously accessed the folder elsewhere, or if someone has browsed one of your web pages!

If no folder with your username is visible press **Ctrl-l** (i.e. Ctrl-L) to bring up a location bar then edit the location to be /web/s1234567 (with your *own* username as appropriate!!)

*In a **Terminal** window:*

Simply type

```
ls /web/s1234567
```

Or to move (change) into the directory type

```
cd /web/s1234567
```

Web Pages *Served* across the Web vs. Edited/Retrieved as a File

The public_html folder in each user's folder (i.e. s1234567) is a special folder where you should put all your web pages. It is only referenced over the file system; not over the web (which is why you may have never seen it in any web URL!). E.g. (**EXAMPLE ONLY**):

```
/web/s1234567/public_html/mypage.html
```

could be accessed in a web browser at:

www.geos.ed.ac.uk/~s1234567/mypage.html

As mentioned very briefly earlier you can suppress web users from listing your directory in a simple manner by employing an index.html page in any directory you wish not to be listable over the web! There are further options and further directories for different purposes. Full training will be given on relevant courses, or see the IT pages for more information/guidance.

APPENDIX B: Managing Printing in Linux [REFERENCE ONLY]

Monitoring Printer Queues

In the modern University cloud printing environment the following may be of very limited direct use however the commands are included should you have reason to 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:

```
s1234567@baltic10:~/my_wkzero$ lpq
EdPrintPull 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:

```
s1234567@baltic10:~/my_wkzero$ lpq
EdPrintPull is ready
no entries
```

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

```
s1234567@baltic10:~/my_wkzero$ lpq -Pprintername
printername is ready
```

Removing Print Jobs from the Queue [IF PERMITTED!]

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**:

```
s1234567@baltic10:~/my_wkzero$ lpq
Cloud-Mono is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  snnnnnn 234253  nation_summary.txt              1024 bytes
s1234567@baltic10:~/my_wkzero$ lprm 234253
```

Note: With modern network and printer speeds you are unlikely to be fast enough to cancel a print job and you can simply choose not to print it (or delete) it at one of the network copier-printers located around the University estate. The lprm command however is silent if successful, otherwise you'll get a message along the lines of:

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

Advanced Linux Printing – Code/Data Tables

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

Note: 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 should format code in a non-proportional or fixed width/monospace font such as Courier (or Consolas) to maintain code readability and column alignment. A good editor will allow this however manual formatting in word processors will often be required.

It is also still possible to print such listings from directly in Linux (or other UNIX variant). 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 – remember you can always print to a virtual printer file!) You can simply print to a postscript print file using **`lpr`** (or **`lp`**) and use the relevant Linux command to submit the file to the printer. You may need to consult with IT should you wish to do this for any reason.

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.

→
```
s1234567@baltic10:~ my_wkzero$ lp nation_summary.txt
request id is EdPrintPull-27630 (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

```
s1234567@baltic10:~/my_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

→
```
s1234567@baltic10:~ my_wkzero$ a2ps nation_summary.txt
[nation_summary.txt (plain): 1 page on 1 sheet]
request id is EdPrintPull-234260 (1 file(s))
[Total: 1 page on 1 sheet] sent to the default printer
s1234567@baltic10:~ wkzero$
```