An introduction to Unix* and the† shell

(*) unix-like operating systems
(†) actually, *a* shell

Overview

This brief course will give you two things:

1. An introduction to Unix

2. An introduction to using the shell

…both of which will help you if you plan to attend the cluster training course or the bioinformatics programming courses.

This course has a practical component, you will need a 'virtual machine' on your laptop.

Session I

1   Introduction
2   Files and directories
3   Creating things

Session II

4   Pipes and filters
5   Finding things

Session III

6   Transferring files
7   *Loops*
8   *Shell scripts*

Bad reasons to be here

~~The shell is intutive and easy to use.~~

    We'll let you judge…

~~Shell tools let us process all kinds of data.~~

    Only if the data is suitably 'retro'.

~~The shell is a good programming language.~~

    The shell pre-dates 40 years of important advances in software engineering.

Good reasons to be here

Unix-like operating systems are everywhere, and you can control them through the shell.

The shell allows you to automate workflows and eliminate repetetive tasks.

The shell is the natural route to other power tools like C, perl, R, & Java.

The shell is your gateway to the world's supercomputers.

Places you will find unix

Apple computers

servers

Android mobile devices
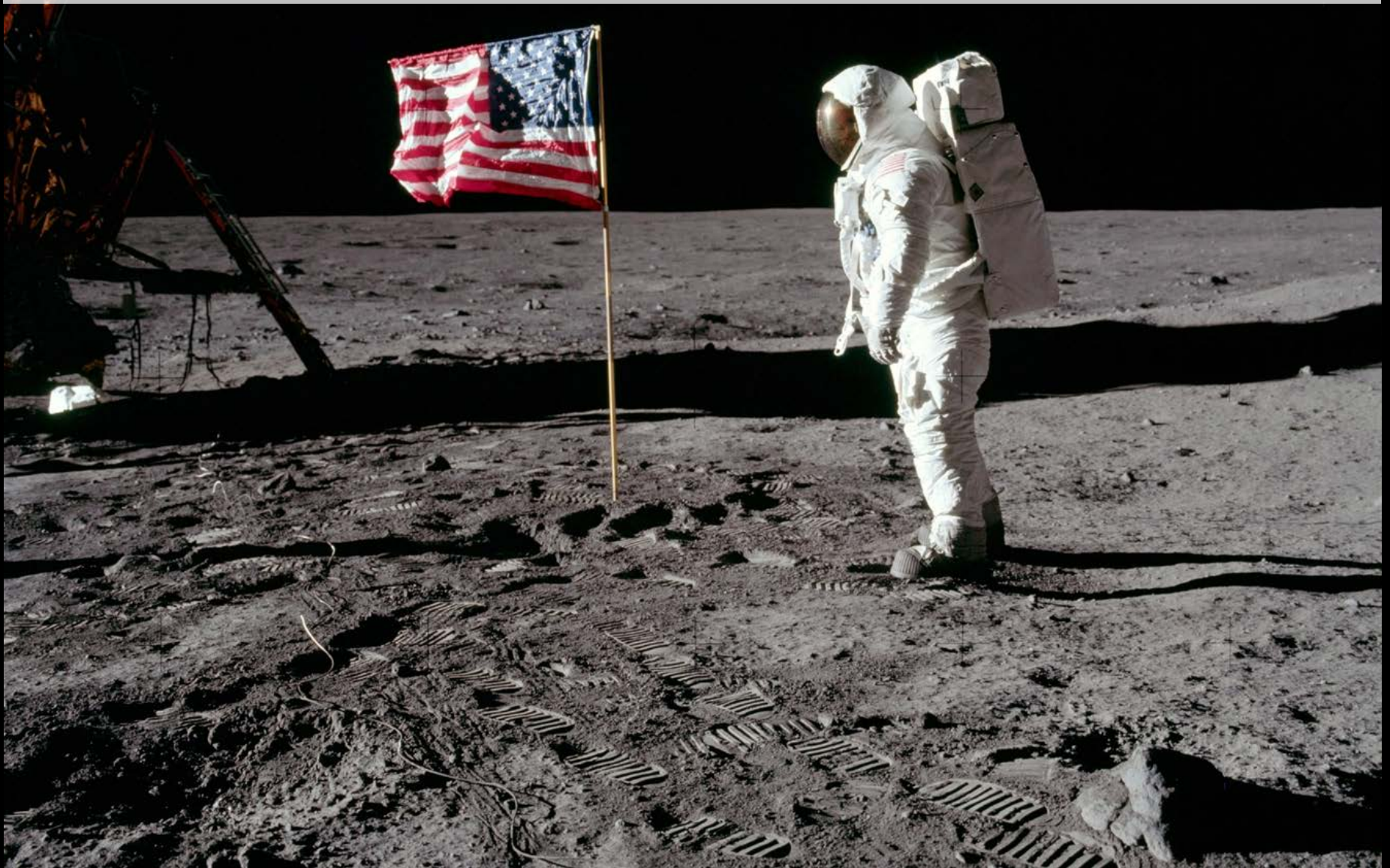

cars

things*

(As in 'internet-of-things')

But first, a warning.
It's always 1969, and we are all American.

'unix time'

Time stamps are encoded as the number of seconds since 00:00 on Thursday 01 January 1970.

Unix systems administrators have big parties every 1,000,000,000 seconds.

retro files

In 1970, most 'files' were lists of typewriter commands. Many unix commands still assume this to be true.

```
cccc ccc ccccc\n
cccccc\n
cccc cccc ccc\n
[EOF]
```

terminals

Unix and the shell pre-date windows and mice so everything works fine on an old terminal.

Text is entered and printed left to right, top to bottom.

'Advanced' software had moving flashing cursors and paging.

```cpp
void
PTY::OpenPTY(string const& terminal)
{
    char* shell = getenv("SHELL");
    if(!shell)
        shell = (char*)"/bin/sh";
    else
        shell = &shell[5];


    pid_t pid;
    if((pid = forkpty(&fd, NULL, NULL, NULL)) < 0)
        abort();
    else if(!pid)
    {
        // child
```

50,14-17   23%

Warning – non-SI units!

Computers use binary internally, not base10, so powers of two have a special status.

2^{10} bytes = 1,024 B

When that was a lot of data, it was loosely termed a 'kilobyte' (KB).
An SI kilobyte would be 1,000 bytes (kB).

So what is a MB?

1MB = 1,000 x 1,024 KB ?
1MB = 1,024 x 1,024 KB ?

And so on for GB, TB, PB. Definitions of the value of a petabyte vary by ~125 TB!

caveat emptor!

ASCII

One standard *was* adopted – the "American Standard Code for Information Interchange".

This defines 128 characters, based on US English typewriter keyboards and teletype commands – whitespace, carriage returns, beeps.

…no European accents, no Kanji, no Traditional Chinese.

Punctuation and special characters (, ; $ * ?) were the only 'spares' to use as special commands. Interesting, strange or very bad things can happen if you have these in your file names.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Operating Systems and Processes

'Unix' or 'linux' (or 'UNIX') is our *operating system* – the program that controls the processes and their access to the network, screen, etc.

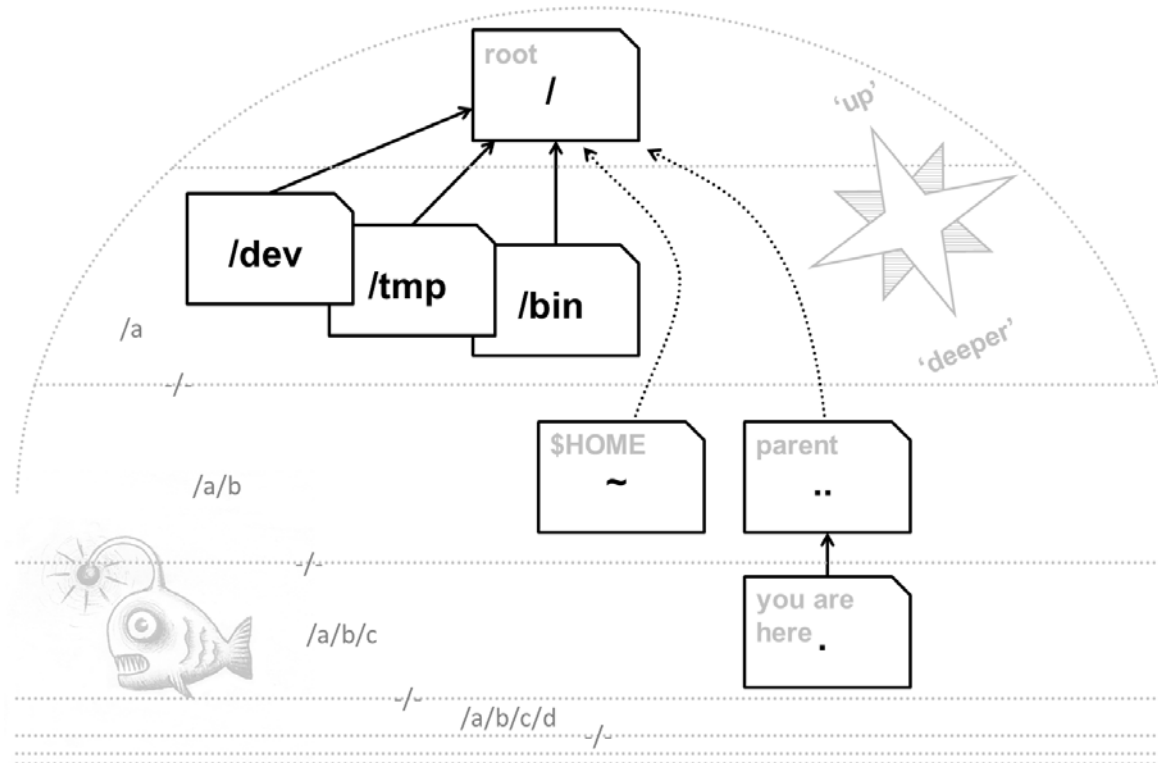The shell is a *process* – it happens to be one that can see its own OS, which is one of the reasons it's so useful.
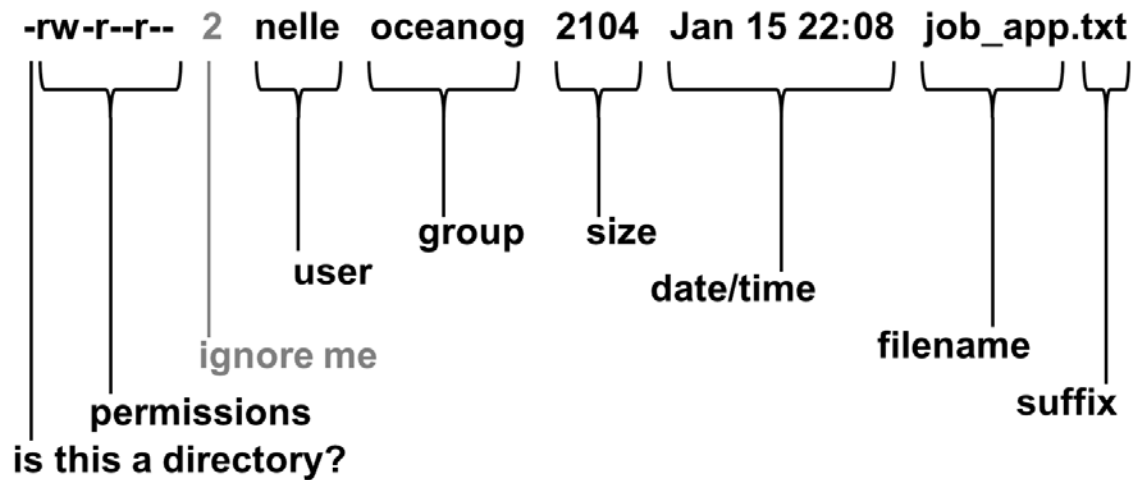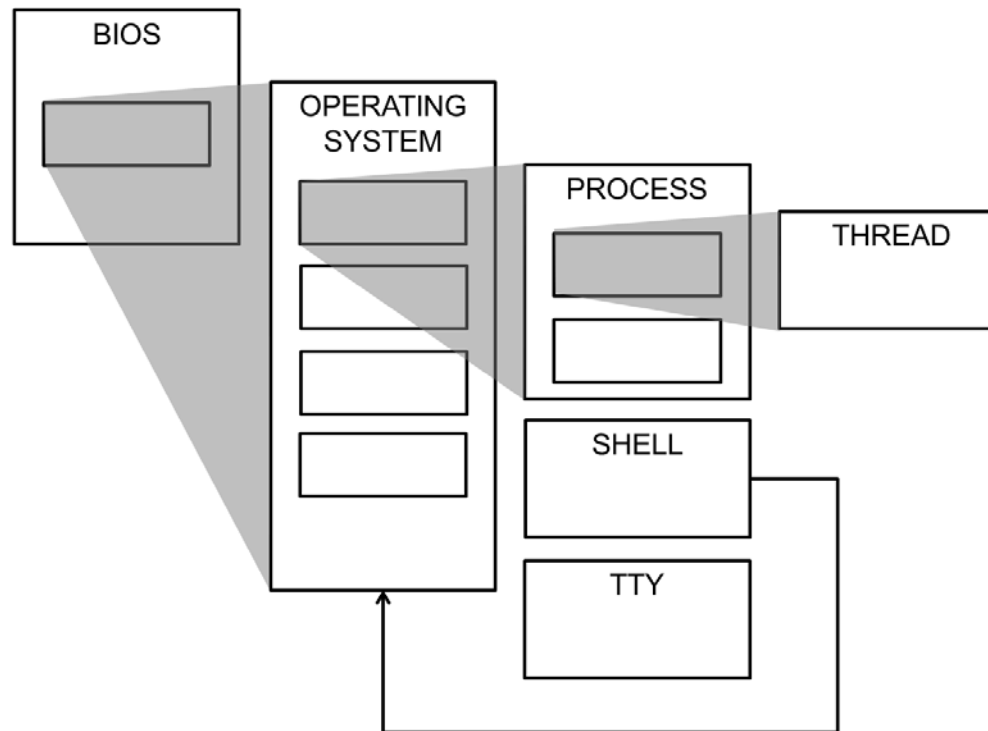
Navigation concepts

You need to be able to navigate without a GUI.
Fortunately some things are always in the same place.
Unix file systems are trees, *with the roots at the top.*

Directories and files

This is the output from the command '`ls -l`'.

It shows how unix likes to think about files and directories.


```
-rw-r--r--  2  nelle  oceanog  2104  Jan 15 22:08  job_app.txt
```

is this a directory?
permissions
ignore me
user
group
size
date/time
filename
suffix

Files and directories

You'll learn how to navigate a file system, see some of the sights, and get HOME when you are lost.

Creating things

You'll make some files and directories of your own – without using your mouse once! – and learn how to clean up after yourself.
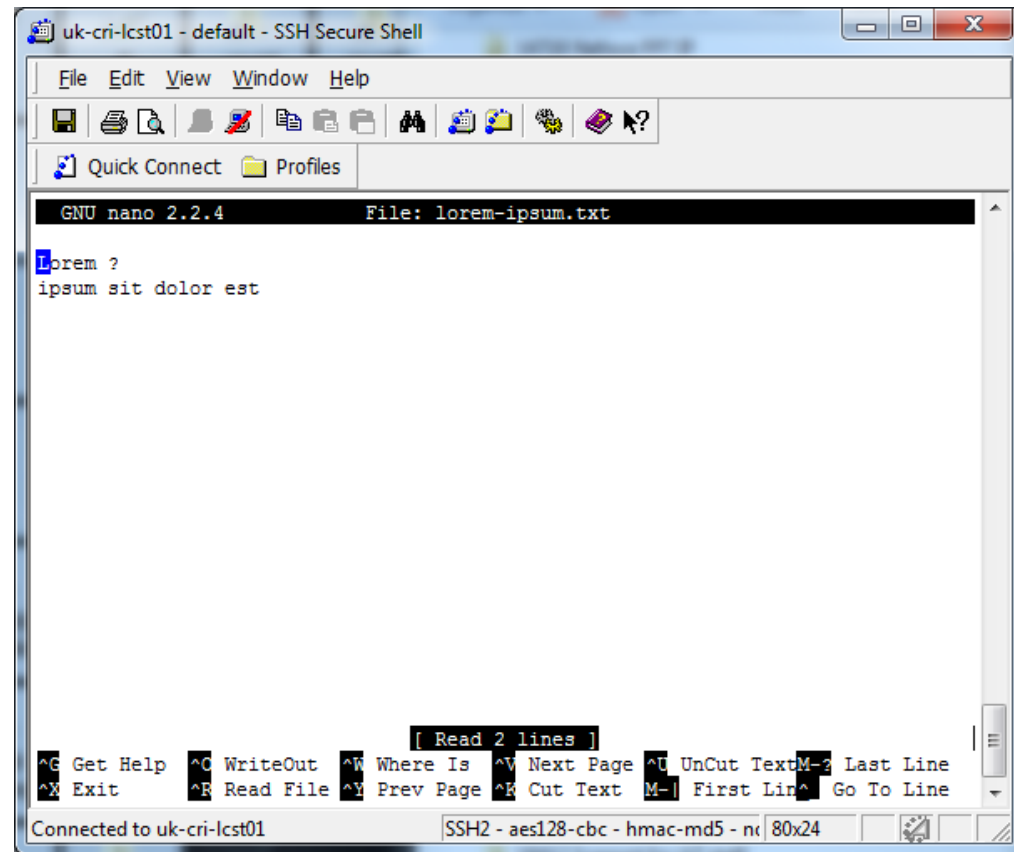
Editors…

Ah, yes, editing. We're sorry. Editing before the invention of windows wasn't pretty.

nano

The course uses 'nano' as its text editor. Those **^X** characters mean "Ctrl+X", they are often used in unixland to get at the "missing" ASCII characters not on the keyboard.

It's easy to use, but sadly it's not standard and you might not find it everywhere.

vi

You will find vi everywhere.

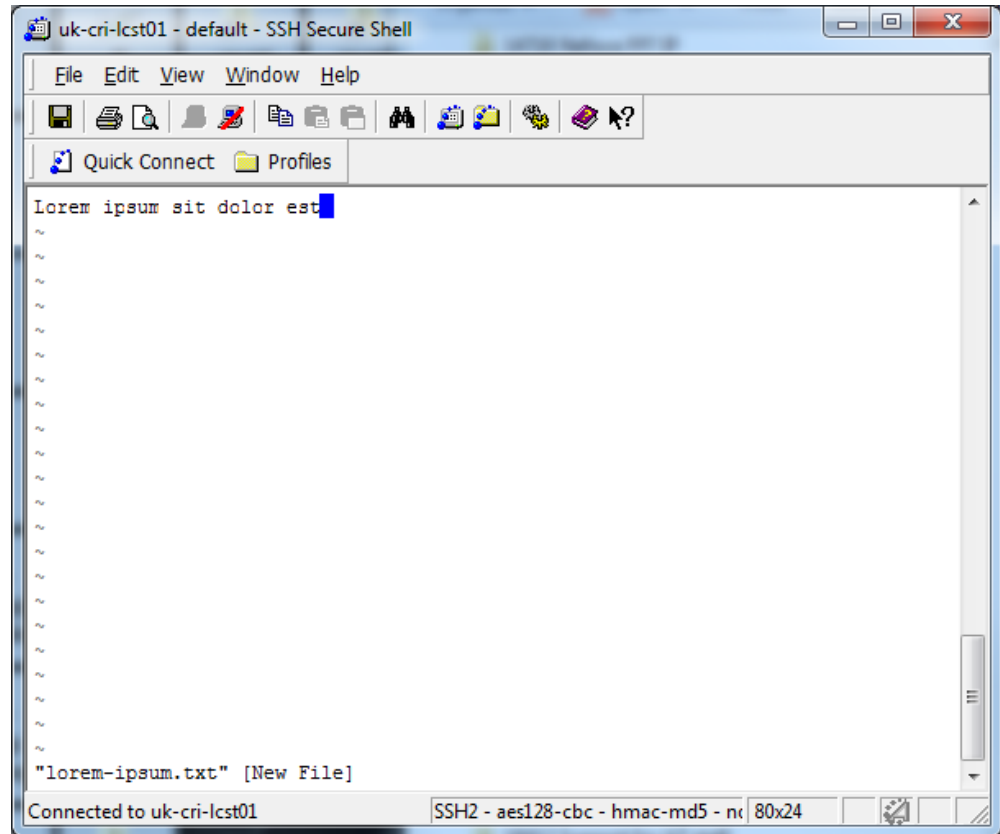Unfortunately, you need to memorise the commands to use it –

i – enter insert mode
a – enter insert mode
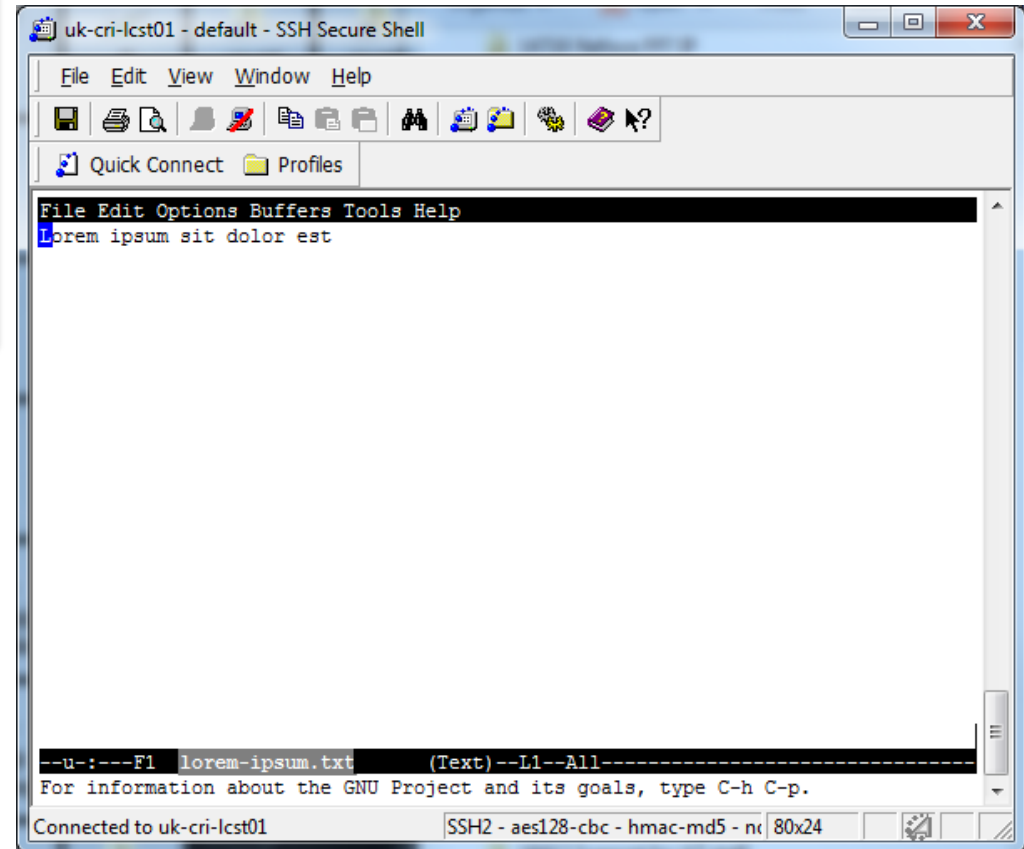**ESC, :, w, NL** – write
ESC, :, q, !, NL – quit without save
…

emacs

emacs was as good as editors got before windows GUI editors arrived. It allows you to open multiple files, has online help, and powerful search and replace.

pwd

'print working directory'

This tells you where you are in the file system, and how deep.

cd

'change directory'

The command that moves you from place to place.

whoami

'who am I (logged in as)?'

Not as stupid as it sounds – it tells you which username you are logged in with. *No spaces!*

ls, ls -F

'list'

Shows the content of the current directory.

mkdir

'make directory'

Creates a new directory, in the current directory.

vi

'visual editor'

Some day you will need to learn vi. Not today.

touch

Literally, 'touch'

Updates the timestamp on a file, or creates it if it doesn't exist.

rmdir

'remove directory'

It removes a directory – but is relatively forgiving.

nano

'an editor called nano'

A simple text editor for use in a terminal window.

rm, rm -r

'remove'

Removes a file, or (with flags) a whole branch. `rm` does not forgive. There is no wastebasket.

Practical Session I

Our practical sessions come courtesy of software-carpentry.org

- Get your virtual machine working

- Go to:

    http://tiny.cc/crukUnix

(Which is http://bioinformatics-core-shared-training.github.io/shell-novice/)

- Work through 1.1, 1.2, and 1.3

- Q & A session

Session II

Pipes and filters

Finding things

Practical session
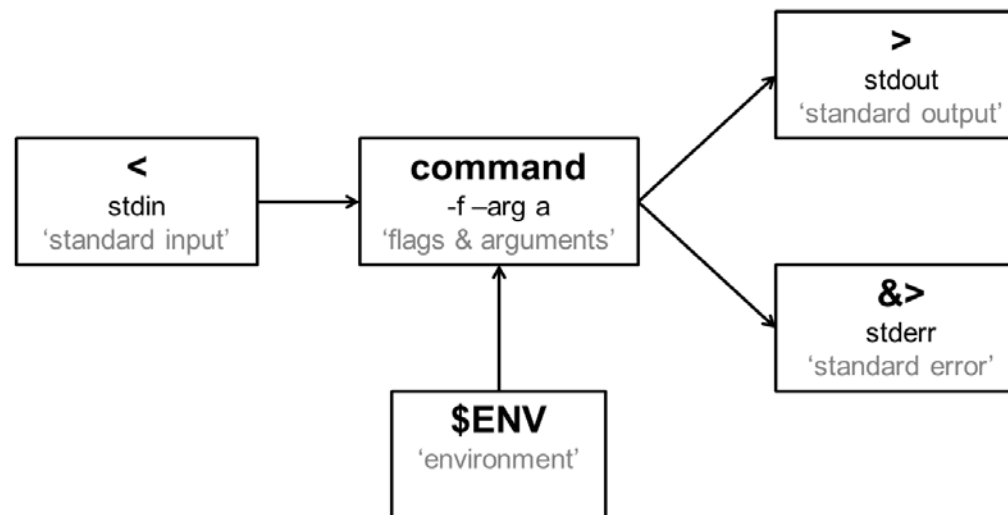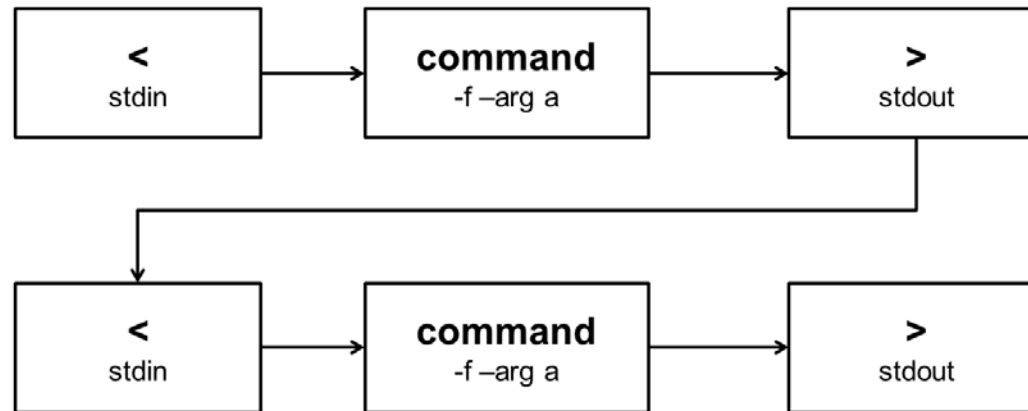
The anatomy of a unix command.

Unix processes have some standard ways of handling input and output.

The "environment" is the list of properties the process picks up from its parent. Your processes will all have the shell as their parent.
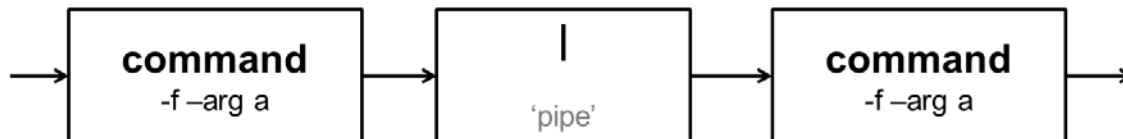
Pipes

Laziness is seen as a virtue among computer programmers. Rather than carry out this pattern over and over:

```
┌──────────┐      ┌──────────────┐      ┌──────────┐
│    <     │ ───► │   command    │ ───► │    >     │
│  stdin   │      │  -f –arg a   │      │  stdout  │
└──────────┘      └──────────────┘      └──────────┘
                                              │
        ┌─────────────────────────────────────┘
        ▼
┌──────────┐      ┌──────────────┐      ┌──────────┐
│    <     │ ───► │   command    │ ───► │    >     │
│  stdin   │      │  -f –arg a   │      │  stdout  │
└──────────┘      └──────────────┘      └──────────┘
```

…you can short-circuit the stdout/stdin using a 'pipe'.

```
        ┌──────────────┐      ┌──────────┐      ┌──────────────┐
  ───►  │   command    │ ───► │    |     │ ───► │   command    │ ───►
        │  -f –arg a   │      │  'pipe'  │      │  -f –arg a   │
        └──────────────┘      └──────────┘      └──────────────┘
```

We've lost <, >, and | from our keyboard – time to lose some more.

Globbing

In the shell, * ? and […] are treated as wildcards:

```
*.txt  – any text file
bob*.txt  – matches bob.txt  and  bobcat.txt
bo?.txt  – matches bob.txt  not  bobcat.txt
bo[bg].txt  – matches bob.txt  and bog.txt
```

Regular expressions

There are more complex patterns called regular expressions which add even more complex rules (with slightly different syntax):

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$
```

If you like regular expressions, you'll love Perl…

Pipes and filters

You'll learn how to use pipes, and then you'll connect together a pipeline of commands to do some actual data filtering.

Finding things

This exercise shows you how to look for patterns in text files, and some of the ways to find individual files in a large file system.

wc
'word count'

Counts the number of characters, words and lines in a text file.

head, head -N
'head'

Prints the first few lines of a file, you can choose how many.

cat
'concatenate'

Prints a single file or list of files to the screen.

tail, tail -N
'tail'

Prints the last few lines of a file.

sort, sort -n
Yes, 'sort'

Sorts the lines of a text file alphabetically or by number.

grep

'global regular expression print'

Search for lines in a file containing a pattern.

man command

'manual page'

Prints the manual page for a unix command.
Very useful for flags and parameters.

find, find -name

'find'

Search for files whose name (or other
properties) match the search parameters.

Practical Session II

- Go to:

  ## http://tiny.cc/crukUnix

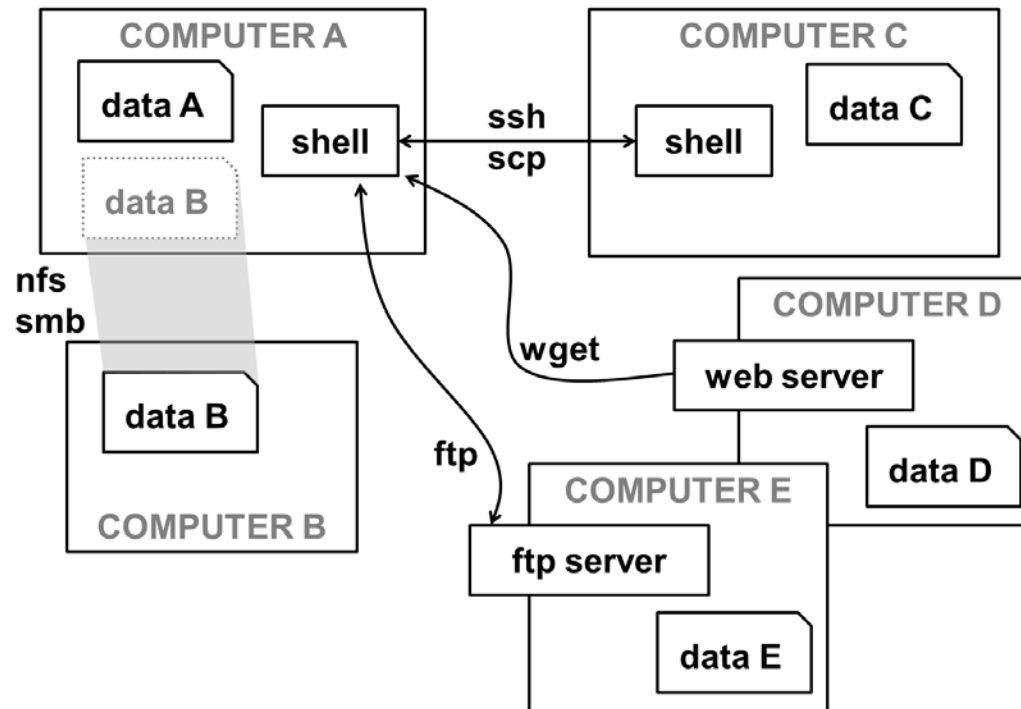 (Which is http://bioinformatics-core-shared-training.github.io/shell-novice/)

- Work through II.4 and II.5

- Q & A session

Moving data, or yourself

Most of the ways of moving data around the internet were developed for Unix first. You also have the option of going to where the data is, with a remote shell.

Shell programming

The shell gets to each line you type before it is passed to a unix command. So not everything you type is a unix command, some are instructions to the shell's own language.

Beware! There are many different shells, and each is programmed in a slightly different way. We're using `bash`

In the next exercises you will use

```
history
```

…giving access to the shell's memory of what commands you have typed, and loops which let you repeat operations:

```
for x in a b c
do
    echo $x
done
```

Transferring files

In this exerise you will experience telepresence, 1970-style, and rescue some files from other computers across the world.

Loops and shell scripts

The final exercises will introduce you to the basics of shell programming and scripting.

ssh

'secure shell'

Opens up a shell session on a remote machine, over an encrypted channel.

scp

'secure copy'

Carries out a copy between two machines, using the ssh machinery.

wget

'web get'

Lets you grab a file using a url, without all that messing around with web browsers.

ftp

'file transfer protocol'

Creates another shell-like environment (with a different command set), from which you can connect to other machines and push or retrieve files.

Other shells

Just for completeness, be aware that there are different shells…

bash

'Bourne again shell'

The most widely used 'user-friendly' shell.

csh

'the Berkeley UNIX C shell'

A hard-core sys admin's shell.

sh

'shell'

The original shell – sometimes the default.

Practical Session III

http://tiny.cc/crukUnix

(Which is http://bioinformatics-core-shared-training.github.io/shell-novice/)

- Work through III.6

- Q & A session

- There are two more sections III.7, III.8 – these introduce some programming skills, and are a bit much for a half day introduction.

Feedback please: http://tiny.cc/unix-june22

Credits:

Cancer Research UK Cambridge Institute
www.cruk.cam.ac.uk

    Simon Bell, computing
    Mark Dunning, bioinformatics
    Peter Maccallum, computing
    Marc O'Brien, computing
    Anne Pajon, bioinformatics

The Software Carpentry Foundation
www.software-carpentry.org

UNIVERSITY OF CAMBRIDGE