Scientific Computing Course

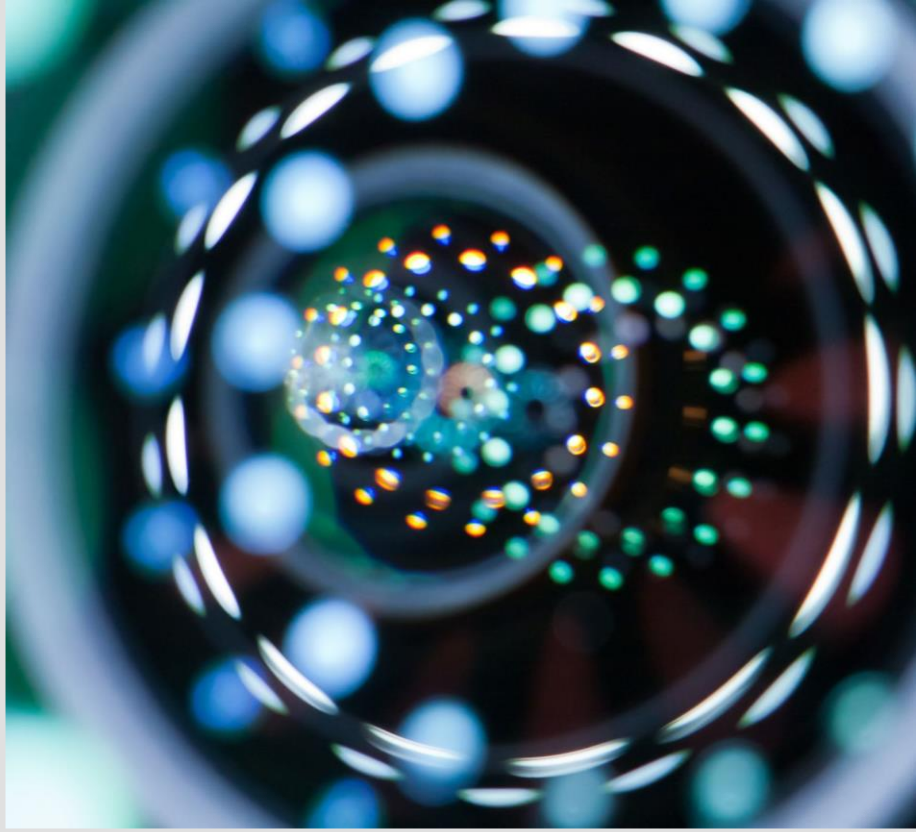
Introduction to Computing, Linux & the Bash Shell

Dr. John Pelan – Head of Scientific Computing @ SWC
< J.Pelan@ucl.ac.uk > – May 2023

Little bit about me

- BSc Applied Physics with Electronics (FYP: Multilayer Perceptrons)
- MSc Computational Science (BLAS implementation in i860 assembly)
- PhD Computational Atomic Physics (electron excitation of atoms & ions)
- LLM International Intellectual Property Law

- Commercial software - safe working envelope of lorry mounted cranes
- PDRA to an Atomic Physics consortium
- Recruited by Geoff Hinton to help start Gatsby Unit's IT infrastructure
- Early advocate for Linux & open-source in academic settings



Sainsbury Wellcome Centre

Course Overview

- Aims
  - ➢ Basic computer architecture & concepts.
  - ➢ Why Linux / Unix and what is POSIX ?
  - ➢ What is a ‘shell’ and how to use one.
  - ➢ Pointers to other learning resources

Timing

- 09:00 - 09:50 Introduction to Computing & Linux
- 10:00 - 10:50 Bash Shell & Command Line Basics
- 11:00 - 11:20 Unstructured section for practice & questions

Computer Architecture

Elements of a Computer

- Processor - CPU / GPU – something that takes instructions & data and acts on them
- RAM – Very fast but (usually) volatile memory which stores instructions & data
- Input / Output (I/O), e.g.
  - Human Interface Devices (HID) video output, keyboard, mouse
  - Data storage (non-volatile) HDD, NVMe, Flash drives
  - Data acquisition & control – A/D, D/A, control signals
  - Other I/O – USB, network etc.

- Buffers, caches – sitting between devices of different latency & bandwidths.
  - ➢ Caching is critical to efficient computer operation – optimization / 'tuning'

Bandwidth & Latency

- Data needs to travel over physical connections thus physical limitations
- Can be in serial or parallel

- Latency : transfer set-up time
- Bandwidth : bits/bytes per second (B - bytes, b – bits , NB powers of 2 or 10)

- CPU - Intel Core i7 – 13th Generation
  - External memory bandwidth ~90 GB/sec
  - Internal L2 cache – 24MB –

- GPU – Nvidia A100
  - External memory bandwidth ~1 TB/sec

- PCIe 4.0 ~2 GB/sec per lane (usually 16 lane max per PCIe card)
- PCIe 5.0 ~4 GB/sec per lane

- Audio sampling – e.g 44.1 KHz, 16-bit – 88.2 KB/sec (x2 for stereo)
- Neural recording electrodes in similar ball-park

‘Bare Metal’

- What do we want to do ? Run our program !
  - ➢ Take data in from somewhere (outside world / other computer)
  - ➢ Process it
  - ➢ Push data out to somewhere (affect outside world / other computer)
  - ➢ Repeat !

- How do we implement this ?
  - ➢ Can write a single, monolithic program that handles everything – ‘bare metal’
  - ➢ Typically embedded system / microcontroller – usually dedicated function,

    e.g. washing machine
  - ➢ Provides fine-grained control but can be difficult & complex to develop
  - ➢ What about multiple programs at the same time or sharing with someone else?

Operating System

Have a computer platform where:

- Multiple programs can run at the same time – "processes"
- Share resources – like RAM & CPU – priority / ownership
- Isolated from each other but also possible to intercommunicate
- Multiple people – need to introduce concept of 'user'
- Network stack – to communicate over Ethernet (wired/wireless)
- Drivers – to access hardware (could be vendor supplied & proprietary)
- Possibly Real-Time (RT) – can provide guarantees where timing is critical

- OS - Low level, underpinning program(s), enables fundamental functions.
- An environment in which a process can run and can mutually access resources, files etc.
- Comprised of a kernel, libraries, drivers and possibly supporting / supervising processes.

Operating System Examples

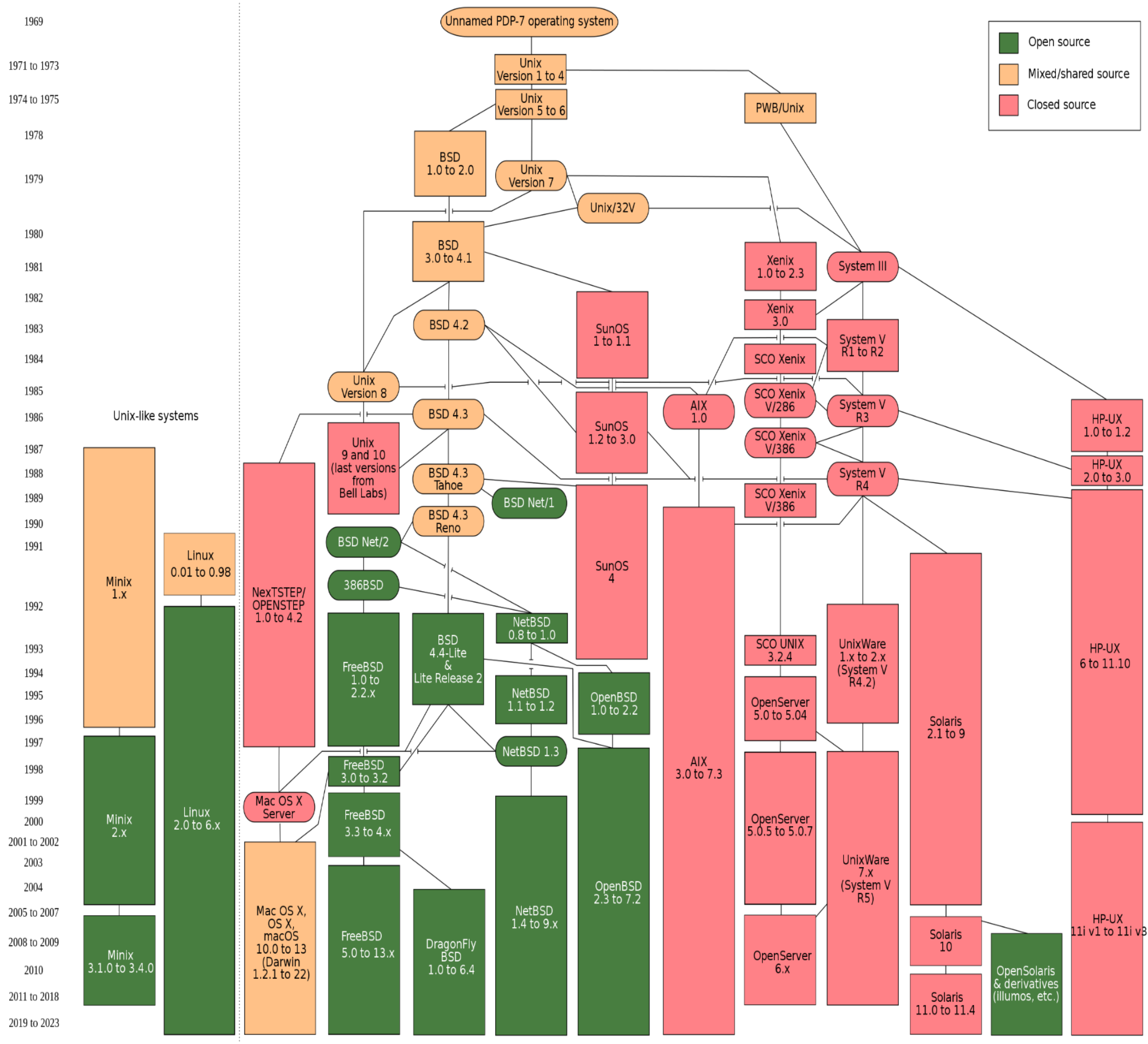
- CP/M or DOS - 1970s to 1980s
- VAX VMS - VAX systems from 1970s through to today (now called Open VMS)
- Unix / POSIX (many flavours – see next slide)
- MS Windows (evolution 3.x / NT / 95, XP, Windows 10 etc.)

- So why has Linux become de-facto OS for high-performance computing ?
  - Based on Unix – long, established history in HPC
  - Some flavour of Unix normally came with a 'big computer' (Cray, IBM etc.)
  - Feature set – fully networked, multiuser, commonly understood
  - Availability of compilers and full set of (often free) development tools
  - Open source (easier to add new features, lower cost)

Unix History

- Ken Thompson writes basis of the first Unix OS in an attempt to port a game to a PDP-7 computer
- Spawned shared and proprietary flavours
- POSIX – set of international standards covering many aspects of Unix-like systems (ISO/IEC 9945).
- Unix does not require a graphical interface, but a graphics layer can be added – typically X11.

- Linux has been independently developed.
- Strictly speaking Linux is just the kernel but in common usage Linux means the entire core software distribution.
- The bulk of the fundamental core software on Linux comes from the GNU Foundation.



https://en.wikipedia.org/wiki/History_of_Unix

POSIX Standards & Features

- Open Group Base Specifications Issue 7, 2018 edition
- Everything you need to know about the Unix environment fundamentals
- Linux not formally certified but compliant

1. General Concepts
2. File Format Notation
3. Character Set
4. Locale
5. Environment Variables
6. Regular Expressions
7. Directory Structure and Devices
8. General Terminal Interface
9. Utility Conventions
10. Headers

Unix Philosophy

Summarized by Peter H. Salus in “A Quarter-Century of Unix” (1994):

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

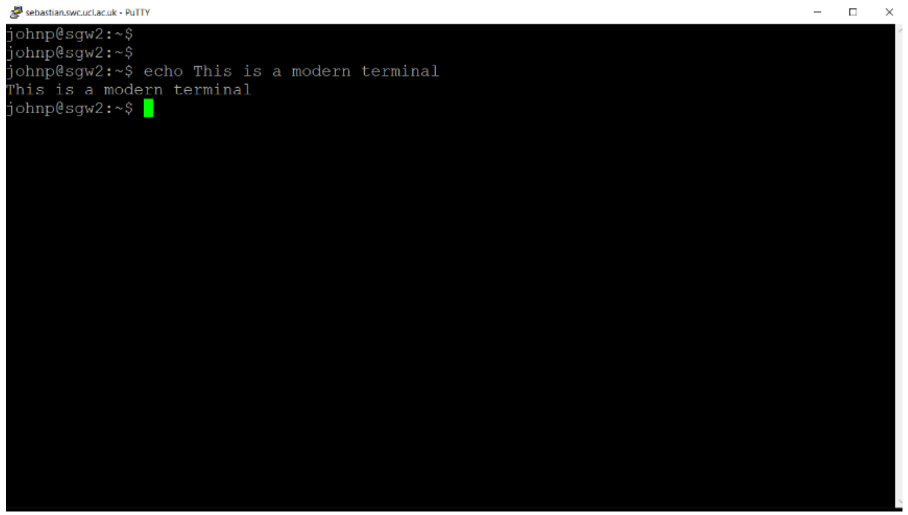
Dennis Ritchie and Ken Thompson in 1974, include the following design considerations:

- Make it easy to write, test, and run programs.
- Self-supporting system: all Unix software is maintained under Unix.

https://en.wikipedia.org/wiki/Unix_philosophy

What is a Shell ?

- "A program that interprets sequences of text input as commands. It may operate on an input stream or it may interactively prompt and read commands from a **terminal.**"



```
johnp@sgw2:~$
johnp@sgw2:~$
johnp@sgw2:~$ echo This is a modern terminal
This is a modern terminal
johnp@sgw2:~$
```

- "a stream is a file access object that allows access to an ordered sequence of characters"
- Means that commands can be provided by a script

Shell – core functions

- Means to interact with a computer & its OS
  - Transfer, examine & manipulate your files
  - Launch programs (“processes” or “jobs”), e.g.:
    - Develop software
    - Run computational code
    - Start a browser
  - Examine & manipulate running processes (e.g. stop or pause)
  - Both interactive & script based – i.e. programmable / repeatable
  - May have ‘built-in’ commands to replace normal system utilities (useful optimization)
- There are many shells available – e.g. sh, Bash, tcsh, zsh

Bash – GNU Bash

- Bash is a popular, but one of many, ‘shell’ programs – a command line interpreter
- Also the language you program it in, a so-called ‘scripting’ language
- Complies with POSIX 1003.1 standard
- Scripts can chain sequences of programs together, react to inputs, command failure etc.
- Bash often used for job submission on HPC clusters
- Notable features:
    - unlimited size command history,
    - job control,
    - shell functions and aliases,
    - indexed arrays of unlimited size,
    - integer arithmetic in any base from two to sixty-four.

`https://www.gnu.org/software/bash`

Bash – Syntax & Operation

When the shell reads input, it proceeds through a sequence of operations.

- If the input indicates the beginning of a comment, the shell ignores the comment symbol (‘#’), and the rest of that line.

- Otherwise, roughly speaking, the shell reads its input and divides the input into words and operators, employing the **quoting rules** to select which meanings to assign various words and characters.

- The shell then parses these tokens into commands and other constructs, removes the special meaning of certain words or characters, expands others, redirects input and output as needed, executes the specified command, waits for the command’s exit status, and makes that exit status available for further inspection or processing.

Commands & Parameters

- Many programs take some form of command-line parameters, e.g.
  - location of a configuration file
  - the enablement of an option or feature.
  - to adjust the functionality, perhaps performing a dry-run
- These are variously described as:
  - "arguments" , "options" or "parameters"
  - "switches" or "flags" – arguments that usually don't take additional parameters (Boolean)
- Parameters are normally indicated with a dash (short-form) or double-dash (long-form)

```
command --help
command –h
command --config /path/to/config-file
command --version
```

System Manuals & Help

- Most programs have some local documentation
- On the system and/or embedded in the binary.
- Manual page accessed using ‘`man`’ command

```
man some-command

man man
```

- Built-in help `--help` (sometimes short form `-h`)

```
some-command --help

man --help
```

Environment Variables

- Variables held by a shell (**NB only available to that instance**)
- Indicated with a dollar-sign prefix($)
- Can be read and written to by programs launched by that shell
- Some are standardized in POSIX, with specific definitions & purpose, e.g.:
  - `$PATH  - indicates to the shell where to look for commands (sequence of directory paths)`
  - `$HOME – your home directory on the system`
  - `$SHELL – the shell you are currently using`

Environment Variables (BASH)

- `variable=value`   # NB no spaces, ideally lowercase
- `env`   # Displays all environment variables & their values
- `echo $variable`   # Print value of $variable
- `read variable`   # Prompts user to enter value

Variables have "scope" which means their visibility is limited to the instance of the shell / script that created them – but you can make variables available to commands or scripts called from that shell instance, using `export`.

- `export variable`
- `some-command`

Files & Directory Tree (POSIX)

- On a Posix system, all files are structured in a single tree hierarchy
- The root of that tree is '`/`' (which is also the path separator – NB different direction to Windows)
- Directories are thus: `/some/series/of/directories/`
- Files are thus: `/some/series/of/directories/filename`
- Names are always case sensitive
- **NB Avoid using space or special characters in filenames**
- Files could be on multiple storage containers (HDD, USB stick etc.) or on remote systems
- But they are layered into the one tree – e.g. `/nfs/`, `/ceph/`, `/mnt/`
- Files & directories belong to an owner and a group – with commensurate access permissions

- Special directory names
  - '.' (dot) – the current directory: ./filename
  - '..' (double dot) – the next directory 'above' in the tree: ../filename

Filesystem Hierarchy Standard (FHS)

- /bin – where essential, core executables are
- /tmp – area for short-term files, not persistent across reboots
- /usr – (read-only) user applications & libraries
- /var – (variable files) log files, cache etc.
- /etc – system config files
- /opt – optional packages

https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

Navigating Directory (shell agnostic)

Every shell has a ‘working directory’ , i.e. the directory that is assumed if you don’t specify one. This is normally your home directory by default but can be changed on-the-fly.

```
pwd      # print working directory
cd       # change directory
ls       # list the contents (many options: ls -lF)
df –H    # show available storage  (‘Human readable’)
quota –v # show filesystem quota
```

Files & Directory Permissions (POSIX)

- Two ownership types – user & group
- Three permission classes - user / group / others
- Permission components
    - ❑ read (r)
    - ❑ write (w)
    - ❑ execute (x) - in the context of a directory means searchable
    - ❑ special bits (setuid, setgid, sticky) – also context dependent
- ACLs – Access Control Lists for complex permissions (try to avoid)

Files & Directory Permissions

```
> chmod u+rw,g+r,o-rw  filename

# You need to be a member of a group to change a file
# to that group ownership.

> groups   # shows you what groups you belong to
> chgrp groupname filename
```

Redirection

- Powerful feature of Unix shells – redirection / streams (as per POSIX)
  - Redirect output of one program into another program (pipe)
  - Redirect the output to a file
  - Direct the input from a file

  - Standard streams called:
    - STDIN - input
    - STDOUT - output
    - STDERR – error messages

- **NB** redirection syntax may differ slightly between the Unix shells.

Redirection (Bash)

```
prompt> command < filename
prompt> command > filename
prompt> command >> filename    # append if existing

prompt> command &> filename    # redirect STDOUT & STDERR
prompt> command &>> filename   # append STDOUT & STDERR

prompt> command1 | command2    # pipe output

prompt> command1 | command2 | command3 >> final-output
```

File Creation / Editing

```
> touch datafile      # changes modification date
                      # creates empty file if it didn’t exist

# Text Editors (examples, lots of choice)

prompt> nano
prompt> emacs
prompt> vim          # advanced vi

# Graphical text editors are available
# Some editors can be context aware, e.g. C, HTML, JSON etc.
# Called “syntax highlighting” - useful for checking syntax etc.
```

File Manipulation

```
prompt> mv oldname newname          # think of moving as renaming
prompt> mv filename ..              # move up a directory
prompt> mv filename /tmp/newname    # move to /tmp with new name

prompt> cp oldname newname          # make a copy of a file
prompt> cp filename ..              # create a copy one directory up
prompt> cp filename /tmp/newname    # make a copy in /tmp with new name

prompt> rm filename                 # delete
prompt> rm -i filename             # delete with 'are you sure'

prompt> mkdir newdirectory          # make directory
prompt> mkdir /tmp/myscratch        #
prompt> rmdir /tmp/myscratch        # remove *empty* directory
```

Scripts

- Just a text file containing a sequence of commands
- Comments indicated with ‘#’ (hash or cross-hatch)
- Can have special preamble on first line to indicate what shell to run it with - #! (called “hashbang” or “shebang”)
- Usual program logic & structures: if-then, loops etc.
- Can test for various states – if file exists, if command failed
- Variables (environment variables)
- Some environment variables are special / pre-defined
- Can pass in arguments to script on command line
- Script needs to be set as executable as appropriate – `chmod u+x scriptname`

Bash Scripts – Special Variables

$0	# The name of the script
$1 - $9	# The first 9 arguments to the script.
$#	# Number of arguments passed to the script.
$@	# All the arguments supplied to the Bash script.
$?	# The exit status of the most recently run process.
$$	# The process ID of the current script.

- NB in Bash, all variables are treated as strings by default

Example Bash Script (1)

```
#!/bin/bash

printf "This script is called: %s\n" $0

# loop through the variables
for param in "$@"; do
  echo "$param"
done

if [[ -v myvar ]]
 then
     echo $myvar
 else
     echo “You forgot to export myvar!”
fi
```

Example Bash Script (2)

```
# We want to record number of manual runs and limit it to 'maxrun' runs
runlog=myrunlogfile
declare -i maxrun=5
declare -i count=0
if [[ -f $runlog ]]   # Do we have a runlog ?
then
     count=`wc -l < $runlog`
     if (( $count >= $maxrun ))
     then
             echo Maximum run count reached: $count
             cat $runlog
             exit
     fi
     date >> $runlog
else
     date >  $runlog
fi
count=$(( count + 1 ))
echo Process round: $count of $maxrun
```

Start-up Scripts

- Can customize the Bash environment (every instance)
- Called ‘rc’ scripts - **`$HOME/.bashrc`**
- Set aliases

```
alias ls="ls -lF"
```

ssh – secure shell

- Not a shell !
- Secure access to a remote shell over a network.
- Name of application (on Unix) but also name of protocol.
- Encrypted connection, public-private key

- Implementations
  - Putty on Windows (needs to be installed)
  - ‘ssh’ on Linux, Mac and ‘Windows Subsystem for Linux’ (WSL2)

```
ssh ssh.swc.ucl.ac.uk { -l userid }
```

ssh – secure shell (2)

Has other features, e.g.:

- Can be used for secure file transfer (scp & rsync)
- Can redirect X11 graphics to remote client.
- Can act as a secure tunnel for specific network traffic.
- Can use passwordless login (not advised), was basis for major HPC hacking activity in 2020 so this feature is less likely to be enabled at remote site.

John’s Computing Maxim (1)

- ‘ **If you are working too hard at the computer, you are probably doing it wrong!** ’

  - People get fixated on particular methods rather than particular outcomes.
  - Remember that computers are there to make our lives easier.
  - Step back and consider alternative ways of achieving the same outcome.
  - “Inherited wisdom” tends towards sub-optimal over time.
  - Invest your time
    - Learn how to exploit pre-existing tools & functions.
    - Almost always worth automating processes that are time-consuming and/or error-prone.

John’s Computing Maxim (2)

- ‘ Don’t do anything you can’t undo ’
  - Mistakes happen and you may need to get back to earlier state.
  - Always have an reverse gear!
  - Use a revision control system like ‘git’ to track file changes
  - Caution using commands that recurse through directories (errors at scale)
  - Caution using wildcard matching (unintended actions, inadvertent space character)
  - Try a dry-run before large-scale changes
  - Reverting complex permissions is hard to do so keep them simple

Additional things you should understand

- Shell quoting rules
- Regular expressions
- rsync & scp
- ssh tunneling

More Resources

- Bash Reference Manual (from the people that developed Bash!)
  `https://www.gnu.org/software/bash/manual/bash.html`
- LinkedIn Learning – all UCL staff & students
  `https://www.ucl.ac.uk/isd/linkedin-learning`
- YouTube
- Books !
  - O'Reilly Series
  - **"Unix in A Nutshell"** `– excellent reference`
  - **"Learning the Bash Shell"**