Introduction to the UNIX Command Line

Lorne Whiteway
lorne.whiteway.13@ucl.ac.uk

Astrophysics Group
Department of Physics and Astronomy
University College London

30 January 2025

Where to find this presentation

Find the presentation at https://tinyurl.com/ytt3kdm3.

Goals of presentation

- How to access UCL UNIX systems
- How to use the UNIX command line

Information on the Web

Astrophysics Wiki

`https://liveuclac.sharepoint.com/sites/PhysAstAstPhysGrp`
This Wiki is freely viewable and editable by all members of the department. Please use it to record information that you think will be useful to others (including your future self). Be bold!

UCL Research Computing Platforms

`https://www.rc.ucl.ac.uk/`

Stack Overflow

`http://stackoverflow.com/` (But will it survive ChatGPT?)

Computing Environment for Astrophysics

- Large datasets requiring substantial processing followed by sophisticated statistical analysis
- Calculations often done on specialised *high-performance computing* (HPC) machines having large filesystems and large RAM; calculations are often broken into pieces that can be run simultaneously ('in parallel') across many processors.
- Much useful software is made freely available within the community. Software quality is usually high; documentation quality is more variable.
- Many users write their own software.

Local Computing Environment

You will have your own local machine, which might be:

- PC (Windows)
- Mac
- Linux

Also there are shared Linux machines:

- General purpose Astrophysics servers available from outside UCL: `zuserver1` and `zuserver2`
- UCL Cosmology HPC clusters: `splinter` and `hypatia`
- Other UCL clusters: `Myriad` and `Kathleen`
- National clusters: `DiRAC` (UK) and `NERSC` (US)

Work patterns

Several work patterns are possible:

- Write and test a program on your local machine; use the local machine to remotely connect to a server; upload the program to the server and run it there;
- Or do all your work locally (requires small data sets);
- Or use the local machine to remotely connect to a server and do all your work there.

Accessing remote machines

Credentials

- You will need a *username* and *password* for any machine that you want to access.
- Contact Edd Edmondson (e.edmondson@ucl.ac.uk) or John Deacon (j.deacon@ucl.ac.uk) to get these credentials.

The full names of the Astro servers are:

- `zuserver1.star.ucl.ac.uk`
- `zuserver2.star.ucl.ac.uk`
- `splinter-login.star.ucl.ac.uk`
- `hypatia-login.hpc.phys.ucl.ac.uk`

Software for connecting

How to connect to a shared machine

- Windows PC: use `PuTTY` (or `MobaXterm`, which uses `PuTTY`);
- Mac: go to the Terminal window and use `ssh`;
- Linux machine: go to the Terminal window and use `ssh`.

Visibility

- `zuserver*` ($\equiv$ `zuserver1` or `zuserver2`) can be seen from anywhere.
- If you are on the UCL network then you can see any Astro or UCL HPC server.
- If you are not on the UCL network then to connect to Astro or UCL HPC servers (except `zuserver*`) you must set up a *Virtual Private Environment*. For details see `https://www.ucl.ac.uk/isd/services/get-connected/ucl-virtual-private-network-vpn`. An alternative is to logon to `zuserver*` and from there logon to the server.

Using PuTTY for remote connections from Windows

- If you don't have PuTTY you can download it from `http://www.putty.org/`.
- On the 'Connection/SSH/X11' tab, click on 'enable X11 forwarding' and set 'X display location' to 'localhost:0' - this is necessary for handling graphical output.
- On the Session tab, set the Host Name as appropriate e.g. `zuserver1.star.ucl.ac.uk`.

Using ssh for remote connections from Mac and Linux

- Syntax: `ssh -YC username@servername`
- The 'Y' option is necessary for handling graphical output.

X-Windows client

- If the remote program that you are running produces graphical output, then you must have a program (an 'X-Windows client') running on your local machine to display this graphical output.
- On Windows you can use XMing (`https://sourceforge.net/projects/xming/`) or Exceed (available on the UCL Desktop).
- On Mac you can use XQuartz.
- On Linux you don't need to do anything special - the graphical interface is already an X-server.

Unix

- Unix and Unix-like computer operating systems have become the industry standard for scientific research.
- Unix was started in 1969 by Ken Thompson, Dennis Ritchie, and others.
- Unix is multi-tasking and multi-user, with a modular design: the operating system provides various single-purpose tools that may be linked together for more complicated tasks.
- Unix has a reputation for efficiency, robustness, and security.

Linux

- Linux is a popular 'Unix-like' operating system.
- Linux was created by Linus Torvalds, and was first released in 1991. His motivation was to avoid restrictive software licenses.
- `splinter` and `hypatia` use a version of Linux called 'CentOS'.
- Free and open source.

Command shell

- In Linux you will use a 'command shell'.
- This is a text-based environment in which you type commands and receive text output.
- Not GUI! Reflects the hardware limitations current when Unix was created. Low-tech and reliable e.g. for remote access.
- Various command shell programs are used: `bash`, `csh`, `tcsh`, `zsh`, etc. To see which one you are using, call `echo $0`.
- This presentation assumes `bash`! Other shells may use different names for some of the commands discussed.

Directory structure

- Everything is organised around files (which may be data files or program files i.e. instructions to be executed).
- Files live in directories. There is a hierarchical tree structure of directories.
- The *root* directory (the base of the directory tree) is called `/`.
- Sample file name: `/home/ucapwhi/foo.txt`
- Note use of slash '/', not backslash '\' as in Windows.
- Case sensitivity: 'Foo' and 'foo' are different strings.

Working directory

- The shell is always pointed at one particular directory, known as the *working directory*.
- Use `pwd` ('print working directory') to see the current working directory.
- Refer to files in the working directory simply via the file name (example `foo.txt`) and refer to files in other directories by directory name plus file name (example `/home/ucapwhi/foo.txt`).

Abbreviations for directories

- Full stop `.` is an abbreviation for the working directory.
- Two full stops `..` is an abbreviation for the parent of the working directory.
- Hyphen `-` is an abbreviation for the most-recent previous working directory – useful if you need to flip back and forth between directories!
- Tilde `~` is an abbreviation of the user's *home* directory e.g. `/home/ucapwhi/`. Many configuration files are located here by default.

Navigating the directory tree

- Use `cd` to change working directory. Example: `cd ../data/des/` .
- Use `ls` to list the files in the current working directory; use `ls <dir>` to list the files in another directory.

Keyboard speedups

- Linux has several speedups that make it efficient to use the keyboard to type commands.
- Keyboard interfaces predate more modern graphical interfaces; they are less friendly for new users, but are low-tech, robust, and very efficient for experienced users.

Keyboard shortcuts: tab

- Use the `tab` key to autocomplete commands, directory names and filenames.
- If what you have typed so far doesn't have a unique autocompletion, then it will complete up to the first ambiguous character.
- This will influence your naming conventions for directories and files!

Keyboard shortcuts: up and down arrows

- Use `up arrow` to scroll backwards through previous commands, and then `down arrow` to scroll forwards again.
- Follow with `Enter` to execute an old command that you have scrolled back to, or `ctrl+c` to cancel.

Keyboard shortcuts: ctrl+r

- Use `ctrl+r` to search backwards through previous commands (*reverse-i-search*).
- Type a substring (not necessarily initial) of the sought-for command.
- Example: `ctrl+r push` will bring up the most recent command that included the substring `push`.
- Follow with `Enter` to execute, `ctrl+c` to cancel, or `ctrl+r` to search further back.

Keyboard shortcuts: alias

- Use `alias` to create your own abbreviations for long commands.
- Example:
  `alias s='cd /share/ucapwhi/almanac_project/'`
- It's boring to retype all your aliases at the start of your session. So instead put them in a batch file that autoexecutes at login - this will be named `~/.bashrc` or something similar.

Environment variables (1)

- The operating system maintains a global namespace of 'environment variables' to store configuration information.
- Use `set` to see all environment variables;
- Use `echo $<variable_name>` to see the value of one environment variable (e.g. `echo $PATH`);
- Use `export $FOO='my_string'` to set an environment variable `FOO`.

Environment variables (2)

- Variables `PATH` and `PYTHONPATH` are used frequently (to maintain lists of directories in which to search for executable programs and Python modules, respectively).
- Linux has no equivalent of the Windows Registry; configuration is done via the directory structure and the environment variables.

Structure of commands

Structure

```
[command] -[option(s)] [argument]
```

Examples

```
ls -la
```

```
mkdir my_experiments
```

```
cp hello.cpp new_hello.cpp
```

Command reference

- For help with `<command>` (in increasing order of verbosity):
  `<command> -h` or `<command> --help`

  `man <command>`

  `info <command>`
- `http://www.computerhope.com/unix.htm` is a useful reference for Linux commands.

File management

- Use `mkdir <dir>` to make a new directory
- Use `rm -rf <dir>` to delete a directory and its contents (including subdirectories). This is irreversible!
- Use `cp <source> <destination>` to copy a file and `mv <source> <destination>` to move a file.
- Use `scp <source> <destination>` to copy a file between servers. The syntax for the remote server is `<username>@<servername>:<filename>`.

File contents

- Use `cat <file>` to show the contents of a file as text and `xxd <file>` to show the contents of a file as bytes.
- Use `head <file>` or `tail <file>` to show the first or last few lines in a file - helpful if the file is large!
- Use `grep` to search for text with a file or files.

Controlling processes

- Use `top` to see all the processes running on a server (not just your own); type `q` to exit `top`. Helpful if someone seems to be hogging the processor!
- Use `kill` to stop one of your own processes.
- Use `watch` to run a command repeatedly.
- Use `nohup` or `screen` to let a command continue to run even after you exit your session.
- Use `&` at the end of a command to have it run in the background; control is then immediately returned to you. Use `jobs` to see what is running in the background, and `fg` to move a job from background to foreground.

Long command lines

- Concatenate several commands onto one long command using semicolon e.g. `cd ~;cat .bashrc;cd -`.
- Conversely use `\` to split one long command over two lines.

Wildcards

- Some commands take *sets* of filenames as an argument. For example `rm <set of filenames>` will remove multiple files.
- Such sets can be given as a space-separated list (example `rm foo1.txt foo2.txt`), or else using *wildcards* (example `rm foo*.txt`).
- The wildcards are
  `*` (matches zero or more characters) and
  `?` (matches preceisely one character).

vim

- `vim` (an improved version of `vi`) is a text editor that is found on almost every UNIX machine.
- Some familiarity with `vim` is therefore useful, as it's often the fastest way to make small edits to text files.
- Bad news: the key commands for `vim` are *very* different from those that have become standard on personal computers. Good news: `vim` can be useful even if you know only a few commands.
- The minimum you need to know is how to exit `vim` if you get into it by accident: `esc : q! enter`.

Redirection

- Use `>` to *redirect* the text output from a program to go to a file instead of the screen. Example: `ls > listing.txt`.
- Conversely use `<` to allow text input to a program to be taken from a file rather than the keyboard.
- Use the pipe symbol `|` to allow output from one program to be used an input for another. Example: `ifconfig | grep UP`.

bash scripts

- bash comes with a Turing-complete scripting language in which you can write programs.
- Such scripts are usually used for control of processing (rather than for e.g. actual processing of data).
- The language has some idiosyncrasies (e.g. `fi` instead of `end_if`).
- It is common to use the extension `.sh` for such files.

Example bash script

```
#!/usr/bin/env bash
source /mnt/lustre/tursafs1/home/dp327/dp327/share
cd /mnt/lustre/tursafs1/home/dp327/dp327/shared/lf
rm -f /mnt/lustre/tursafs1/home/dp327/dp327/sharec
/mnt/lustre/tursafs1/home/dp327/dp327/shared/lfi_p
/mnt/lustre/tursafs1/home/dp327/dp327/shared/lfi_p
/mnt/lustre/tursafs1/home/dp327/dp327/shared/lfi_p
echo stop > /mnt/lustre/tursafs1/home/dp327/dp327/
python3 /mnt/lustre/tursafs1/home/dp327/dp327/shar
mkdir -v ./fof/
mv -v *.fofstats* ./fof/
chmod g+w -vR ./fof/
cd /mnt/lustre/tursafs1/home/dp327/dp327/shared/lf
tar -czvf run017.fof.tar.gz ./run017/fof/
test -f ./run017.fof.tar.gz && rm -rfv ./run017/fc
tar -czvf run017.tar.gz ./run017/
test -f ./run017.tar.gz && rm -v ./run017/run*
```

How to run a bash script

```
bash myscript.sh
```

Alternatively make the script executable:

- Add `#!/usr/bin/env bash` at the top of the file
- Then `chmod +x myscript.sh`

Then just need to call `./myscript.sh`

Scripts and environment variables

- It's natural to write a script that sets environment variables.
- But this doesn't work! The script gets a *copy* of the environment; this copy gets amended by the script, but is then discarded when the script stops.
- So instead use `source` e.g.

```
source my_set_environ_vars.sh
```

HPC environments

- `splinter` and `hypatia` are *high-performance computing* (HPC) clusters.
- Each has many linked computers (called *nodes*). Most nodes are *compute* nodes; these are available for processing. The *login* node (which you login to) is special. Use it for control, for editing and for compiling, but not for heavy processing!
- `srun --pty bash` will give you a session on one of the compute nodes.
- Alternatively you can use `sbatch` to launch an unattended job on one or several compute nodes.