An Introduction to UNIX

Andrew Hazel

Department of Mathematics,
University of Manchester
Outline of course

• 11:00–12:00: First Lecture
  • General Introduction and basic concepts

• 12:00–13:00: Group 1: Lunch
  Group 2: Practical Session 1

• 13:00–14:00: Group 1: Practical Session 1
  Group 2: Lunch

• 14:00–14:45: Second Lecture
  • The UNIX filesystem and shells

• 14:45–15:00: Break

• 15:00–15:30: Final Lecture
  • Using remote machines and job control

• 15:30–16:30: Final Practical Session
What is UNIX?

- Multi-tasking, multi-user operating system
- “Standard” for big computing
- UNIX underpins the internet and many of the services it provides
- Departmental fileserver and most powerful computers run UNIX
What is UNIX?

- Multi-tasking, multi-user operating system
- “Standard” for big computing
- UNIX underpins the internet and many of the services it provides
- Departmental fileserver and most powerful computers run UNIX

Advantages
- True multi-tasking
- Flexible
- Portable
- “Nice” programming environment
What is UNIX?

- Multi-tasking, multi-user operating system
- “Standard” for big computing
- UNIX underpins the internet and many of the services it provides
- Departmental fileserver and most powerful computers run UNIX

Advantages
- True multi-tasking
- Flexible
- Portable
- “Nice” programming environment

Disadvantages
- Not that user friendly
- Steep learning curve
- Not 100% compatible with the microsoft world
What can UNIX do for you?

- Departmental E-mail
- World Wide Web
- Scientific Programming
  - FORTRAN
  - C/C++
  - Matlab
  - Maple
- Text Processing
  - \[ \text{\LaTeX} \ 2 \epsilon \]
- Plotting packages
  - Gnuplot
  - tecplot
**UNIX philosophy**

- Small specialised programs ... not complete integration
- Everything is a file
- Power is more important than style
- Many users can work at once
- To use UNIX effectively you need to master a number of different programs and couple them together

- ... but most people survive on a small handful of commands
**UNIX commands**

- UNIX commands take the generic form:

  \[
  \text{command} \ [\text{options}] \ [\text{expression}] \ [\text{files}]
  \]

  \[
  \text{ls} \ -l \ *.c
  \text{emacs} \ \text{junk.txt}
  \text{lpq} \ -l
  \]

- Error messages are often cryptic
- Commands are entered on the command line and interpreted by the shell
- Choice of shell can affect your working environment
Getting help

• Use the **man** and **info** commands for online documentation

• **man -k** or **apropos** can be used to search for commands associated with keywords

• Try **man man** and **info info** to find out how to use these commands !

• Ask others in the computer rooms

• Look at the departmental web pages for documentation

• Look at books (lots of linux/unix books available)

• Send e-mail to support@ma.man.ac.uk
Electronic (E-)mail

- E-mail is a way of sending text between users on computer systems
- An e-mail address is of the form `username@hostname`
- You should have a university e-mail account that can be accessed at `https://webmail.manchster.ac.uk`
- You will need to register on a university PC in order to obtain your University user name and password, which are **NOT** the same as your Departmental user name and password.

- E-mail will be used to circulate important information, so make sure that you can use it!
Editing files

- Most of your time will be spent editing files.
- There are a number of text editors available under UNIX.
- emacs is one of the easiest to learn initially.
- The control key (CTL) and the Escape key (Esc) are used to access specialised editing commands.

- The only way to learn an editor is to use it, so practice.
The basic printing commands are \texttt{lpr}, \texttt{lpq} and \texttt{lprm}

\begin{itemize}
  \item \texttt{lpr} -P \textit{printer} \textit{file} \quad \text{Sends \textit{file} to the printer \textit{printer}}
  \item \texttt{lpq} -l -P \textit{printer} \quad \text{Lists the print queue for \textit{printer}}
  \item \texttt{lprm} -P \textit{printer} \textit{n} \quad \text{Removes job number \textit{n} from \textit{printer}’s queue}
\end{itemize}
Printing

- The basic printing commands are lpr, lpq and lprm

- **lpr** `-P` *printer* *file* Sends *file* to the printer *printer*
- **lpq** `-l` `-P` *printer* Lists the print queue for *printer*
- **lprm** `-P` *printer* `n` Removes job number `n` from *printer*’s queue

- You can set your default printer by using the `printer` command
  [This is a custom command and will only work in the department]
- Generally files must be converted into Postscript before being sent to the printer
- **a2ps** `file.txt` will convert the text file `file.txt` to postscript and send it to the default printer
- Postscript files may be previewed onscreen using the `gv` command
The UNIX filesystem

- Tree-like structure
- / is the root directory (the top of the tree)
- Every other directory is a subdirectory of /
- Every subdirectory has two special directories
  - . is the directory itself
  - .. is the parent directory
- cd (or change directory) is used to move between directories
- If you get lost pwd (or print working directory) will show you the complete path from the root / to your current directory
- mkdir creates (or makes) new directories and rmdir deletes (or removes) directories
- cd with no arguments changes to home directory
- ~ refers to your home directory
Copying and moving files

```
cp file1 file2  copies file1 to file2

cp file1 dir    copies file1 into the directory dir
                 Its path-name will be dir/file1

cp file1 file2 &c dir  copies multiple files into the directory dir
```
Copy the `copying and moving files`

```
cp file1 file2               copies file1 to file2
cp file1 dir                copies file1 into the directory dir
                           Its path-name will be dir/file1
cp file1 file2 &c dir        copies multiple files into the directory dir

mv oldfile newfile          renames oldfile to newfile
mv file1 dir                moves file1 into the directory dir
                           Its path-name will be dir/file1
mv file1 file2 &c dir        moves multiple files into the directory dir
```

`cp` and `mv` with `-i` will warn you before overwriting an existing file.
Deleting files

- `rm file1` deletes the file `file1`
- `rm file1 file2 &c` deletes multiple files
- `rm -i file1` warns you before deleting files
- `rm -r dir` removes all files in the directory `dir`

Be very careful using `rm -r`
Examine files

- `ls` lists the files and directories in the current directory.
- `ls -l` gives a long listing of the files.
- `ls -a` lists hidden files (those starting with a period).
- `ls -t` lists files in order of creation time.
- `ls -R` lists contents of directories.
## File permissions

- `ls -l` shows the file permissions:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Link</th>
<th>Owner</th>
<th>Group</th>
<th>File size</th>
<th>Timestamp</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rw-r--r--</td>
<td>1</td>
<td>andrew</td>
<td>users</td>
<td>0</td>
<td>Sep 12 17:58</td>
<td>file1</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>2</td>
<td>andrew</td>
<td>users</td>
<td>1024</td>
<td>Sep 12 17:59</td>
<td>dir1/</td>
</tr>
</tbody>
</table>
File permissions

- `ls -l` shows the file permissions:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Link</th>
<th>Owner</th>
<th>Group</th>
<th>File size</th>
<th>Timestamp</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rw-r–r–</td>
<td>1</td>
<td>andrew</td>
<td>users</td>
<td>0</td>
<td>Sep 12 17:58</td>
<td>file1</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>2</td>
<td>andrew</td>
<td>users</td>
<td>1024</td>
<td>Sep 12 17:59</td>
<td>dir1/</td>
</tr>
</tbody>
</table>

- First entry in permissions is file type i.e. `d` for directory
File permissions

- `ls -l` shows the file permissions:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Link</th>
<th>Owner</th>
<th>Group</th>
<th>File size</th>
<th>Timestamp</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rw-r–r–</td>
<td>1</td>
<td>andrew</td>
<td>users</td>
<td>0</td>
<td>Sep 12 17:58</td>
<td>file1</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>2</td>
<td>andrew</td>
<td>users</td>
<td>1024</td>
<td>Sep 12 17:59</td>
<td>dir1/</td>
</tr>
</tbody>
</table>

- First entry in permissions is file type i.e. `d` for directory
- Access permissions are in three groups of three
  - `u`, User permissions
  - `g`, Group permissions
  - `o`, Other permissions
  - `r` is read permission
  - `w` is write permission
  - `x` is execute permission
Changing file permissions

`chmod n1n2n3 file` changes file permissions

- Permissions can be changed using numbers or symbolic codes
- e.g. `chmod 741 file` gives the permissions `- rwx r– –x file`
- e.g. `chmod a=r file` gives the permissions `- r– r– r– file`

- Full details may be found on the info pages
A shell is a command interpreter. It translates the commands you type into instructions to the main operating system.

Advantage of a shell is that it can make life a lot easier with wildcards, filename completion and history mechanisms.

The shell can also be used as a programming language to write *shell scripts*.

Many different shells:
- Bourne
- C
- Korn
- Bash

Only going to cover the tcsh (default) shell. Use the `ypchsh` command to change your default shell.
Wildcards

- Wildcards are special characters that can be used to make life easier
- The most used is * which replaces any string
  - `ls *` lists all files
  - `ls *.txt` lists all files ending in .txt
  - `rm a*.txt` deletes all files starting with a and ending in .txt e.g. andrew.text, awfully_boring.txt, another_long_file.txt
Wildcards

- Wildcards are special characters that can be used to make life easier
- The most used is * which replaces any string
  - `ls *` lists all files
  - `ls *.txt` lists all files ending in .txt
  - `rm a*.txt` deletes all files starting with a and ending in .txt e.g. andrew.text, awfully_boring.txt, another_long_file.txt
- `?` will replace only a single character
  - `rm a?.txt` will only delete files of the form ab.txt or ac.txt, **not** andrew.txt
Wildcards

• Wildcards are special characters that can be used to make life easier

• The most used is * which replaces any string
  • ls * lists all files
  • ls *.txt lists all files ending in .txt
  • rm a*.txt deletes all files starting with a and ending in .txt e.g. andrew.txt, awfully_boring.txt, another_long_file.txt

• [ ] encloses a choice of values
  • ls [ab]*.txt will list files that start with a or b and end in .txt
  • The hyphen denotes a range e.g. rm [a-z].txt will remove any of the files a.txt, b.txt, c.txt, etc

• Warning the only sensible ranges are 0-9, a-z, A-Z or subsets thereof (unless you happen to be conversant in the numerical order of ASCII symbols)
Wildcards

- Wildcards are special characters that can be used to make life easier.
- The most used is * which replaces any string:
  - `ls *` lists all files.
  - `ls *.txt` lists all files ending in .txt.
  - `rm a*.txt` deletes all files starting with a and ending in .txt e.g. andrew.text, awfully_boring.txt, another_long_file.txt.
- A caret ^ as the first character in square brackets acts as a logical not:
  - `ls[^e]*` lists all files that don’t start with e.
Quoting

• What do you do if you have a filename containing a wildcard?
• Quoting (or escaping) removes the special nature of wildcard characters
• Backslash, \\', single quotes, ’, or double quotes, “, are used to quote characters
• For example to delete a file with a space in it
  • `rm “strange file”`   `rm ’strange file’`   `rm strange\ file`
• Or with a star
  • `rm “file*”`   `rm ’file*’`   `rm file\`*
• There are subtle differences between quotes, safest are single or strong quotes
• May need to quote when passing arguments to commands
You’ve probably been editing the command line without realising it!

Many emacs (or vi) editing commands are available on the command line

Arrow keys move left, right, CTL-t transposes two letters, CTL-a moves to start of line, etc
Editing the command line (Shell dependent)

- You’ve probably been editing the command line without realising it!
- Many emacs (or vi) editing commands are available on the command line
- Arrow keys move left, right, CTL-t transposes two letters, CTL-a moves to start of line, etc
- **history** will display a numbered list of previous commands
- Up and down arrows move through the history list
Editing the command line (Shell dependent)

- You’ve probably been editing the command line without realising it!
- Many emacs (or vi) editing commands are available on the command line
- Arrow keys move left, right, CTL-t transposes two letters, CTL-a moves to start of line, etc
- history will display a numbered list of previous commands
- Up and down arrows move through the history list

!n
Repeat command n

!!
Repeat last command

!string
Repeat last command starting with string

!?string[?]
Repeat last command containing string

!n:m
mth “word” of nth command

!$
last word of last command

!n:s/old/new/
Repeat nth command substituting old for new
Completion

• The TAB key will attempt to fill in commands a filenames for you

(In tcsh need to type set autolist first)

• It works on the command line, in emacs and various other programs

• For example, if there are three files: file1, file2 & file3 typing ls f and then TAB will expand the f to file and then beep. Pressing TAB again should present a list of possible further choices

• It also works with commands e.g. type l and then hit TAB twice. You should see a list of all commands that start with l
I/O Redirection

- By default input comes from the keyboard and output goes to the screen
- Standard input is terminated by CTL-d
I/O Redirection

- By default input comes from the keyboard and output goes to the screen.
- Standard input is terminated by CTL-d.
- Standard input can be redirected using `<`
  - `cat < file` will list the contents of `file` on the screen.
I/O Redirection

- By default input comes from the keyboard and output goes to the screen
- Standard input is terminated by CTL-d
- Standard input can be redirected using `<
  - `cat < file` will list the contents of `file` on the screen.
- Standard output can be redirected using `>
  - `ls > ls.txt` will send a file listing to the file `ls.txt`
I/O Redirection

• By default input comes from the keyboard and output goes to the screen

• Standard input is terminated by CTL-d

• Standard input can be redirected using <
  • cat < file will list the contents of file on the screen.

• Standard output can be redirected using >
  • ls > ls.txt will send a file listing to the file ls.txt

• cat file1 file2 > file3 will concatenate “add” the contents of file1 and file2 and put the result in file3

• cat file4 >> file3 will append file4 to the end of file 3
Pipes

• A Pipe | passes the output of one command to the input of the next

• For example consider the two stage process
  • `ls -l > ls.txt`
  • `more ls.txt`

• `ls -l | more` sends the output of the directory listing to the more command (which stops the listing flying off the screen)

• Pipes can be stacked together (called a pipeline)

• `ls -l | sort | more` sorts the line entries alphabetically before passing the output to more
Processes

- A process is (basically) any running program
- Every process is assigned a unique(ish) number or PID
- To suspend the current process use CTL-z
- To quit the current process use CTL-c (Remember this if nothing else)
- `ps` will list currently running processes
- `kill PID` will kill the process PID
- `kill -9 PID` means really kill the process!
Job control

- Programs (or jobs) may be run in the foreground or background.
- Running in the foreground prevents you from doing anything (in the shell from which you launch the program) until the program is finished.
- Any foreground processes will be killed if you logout (quit the shell).
- Background jobs will continue to run after you logout and will return control of the shell to you.
- To run a job in the background simply type `command &`.
- For safety you should also use the `nohup` command if you are planning to logout:
  - `nohup command &`
Job control II

• If you have a foreground job that you want to move to the background type CTL-z (to stop the job) and then bg which moves the job into the background

• The fg command can be used to move background jobs back into the foreground

• Use the jobs command to check on currently running jobs

• If you want to kill a background job you must find the PID using ps and use the kill command
Job priority

- Each job is assigned a priority number.
- If you are planning on running a long job (longer than 5 minutes) you must use the `nice` command to alter the priority.
- Thus the whole command for a job running overnight would be:

  ```bash
  nohup nice +19 program &
  ```
Remote machines

• The easiest way to access other public machines is to use the remote machines menu

• Standard networking commands are

  ssh machine gives a remote shell on another machine (Encrypted)

  sftp machine allows remote file transfer

  firefox Web browser
Final thoughts

Basic knowledge

- Login and logout
- Change your password
- Send and receive e-mail
- Edit and print files
Final thoughts

Basic knowledge

- Login and logout
- Change your password
- Send and receive e-mail
- Edit and print files

Average knowledge

- Use directories to organise your files
- Move about the filesystem confidently
- Use a few simple wildcards and history
- Monitor running processes