Outline of lecture (next 26 or so slides)

- Shells – for context, in case you wonder
  - many shells exist, were developed with styles & strengths
  - Many can be installed & switched, suited to purpose
  - Choose bash as most commonly bundled & used for scripts,
- A maze of commands & qualifiers – flags & arguments
  - But inbuilt help : relatively boring but comprehensive text
    - Man – text, with vi editor like interface
    - Info – basic hyperlinked text, with emacs like interface
- Typing seems a pain, but endless click & drag is a bigger pain
  - Shells are programmable, save years of pain
- Tricks of the trade
  - Tab completion
  - Command history (up & down arrows) & line editing ^a, ^e
  - Wildcards * ?
- Tip : try it and you'll see – not as bad as it looks, give bash a lash!

Linux/Unix - Introduction & Review

- Began as a review for a less advanced class,
  - Summarising main useful issues
- Seemed good as an intro for a previous course.
  - To get up to speed, before going deeper
- Probably rather simple in places and slow in pace,
  - but will change as we progress
- Somewhat repetitive for
  - Progressively deeper passes through topics
  - Review & remind – ***to hammer it home!***
  - And a large slide pack to distill…
    - But need to start somewhere
  - Lots of inter-related introductory material
    - Commands, files & filesystems, processes
  - Repetition of basics, hope some will stick!
- **Don't worry, can't retain all this at once, but step-by-step it sticks!**
- **And much easier to do and see in the lab than previewed in a lecture!**

Linux/Unix - Introduction & Review

- Course initially specified as Unix & notes developed that way
  - in a hurry (few days notice) as someone took ill
- Subsequently migrated towards Linux, so drifting that way
- But while there are some differences
  - Licensing – free & open source vs. proprietary
  - Kernels, package managers and implementation details
- The bash commands names and functions are virtually identical
  - Especially the common ones
  - Except some of the less common ones have some variations in flags & configuration details
- So it's easier to cover both together, although we use Linux
  - Windows Subsystem for Linux (WSL) supports Linux, defaulting to Ubuntu, supports many
  - But note OS X uses Unix BSD underneath - but not all
    - commands are identical
    - Nor scripts portable from Linux to Unix…
- Why pay for Unix when Linux is free!?
- Companies may avoid open source as there is no comeback (who can they sue!?)
  - Although most software have flexible liability avoidance clauses
- Red Hat and Canonical
  - consolidated disparate Linux distros (basically Fedora & Debian)
  - and provide commercial enterprise support

Shells

- Shell is the interactive and programmable command line interface….
  - Interprets commands and runs associated programs & scripts
  - Not as fancy as GUI, but much nicer than OS calls
- Lots of shells, developed over the years, with +/- etc.
  - Most now a mashup of the extensions to 2 main originals
- Bourne family
  - Sh – the original Unix shell 70's – pipes to $PATH
  - Bourne shell – 79 – still retains .sh
  - Bash (rewritten Bourne again shell) most common & useful
- C shell – introduced history & editing etc.. common in BSD
  - Derivation : tcsh, which in turn influenced zsh
  - Which is default on OS X (& Kali – security testing!)
- Apparent backward compatibility on modern systems may be due to redirection to to more modern shells on system.. e.g. sh runs bash
  - All set up in configuration files… ignore more daft details...

More recent Shells

- Fish shell – friendly interactive shell – easy to use & install
  - Cross-platform… Linux, Unix, OS X, Win
- Dash – Debian (ash) Almquist shell –
  - small (space & libraries) & fast – 4 x bash
  - Smaller codebase => smaller attack surface
- Ion – faster and more secure than Dash
  - written in Rust, clean, concise design
- Then all sorts of implementation variants
  - Scsh – easier to program, if you're into Scheme, a (Lisp (List Processor) (AI language) variant),
  - Eshell – emacs (editor) facilities & Lisp support
  - Oh – written in Go but with Scheme dialect
  - Elvish – written in Go but with C dialect
  - Xonsh – uses Python 3.5+, cross platform
- NB Python has a largely OS-independent API for much OS scripting

But bash is best for now

- Most common …
  - Not just bundled
  - But standard on most
- So most
  - Code
  - Support
  - Online help - probably
- So best to do
- Sh may be more portable, but not as modern

For comparison :-

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

Linux/Unix – on other systems

- Best for now to run on college lab machines for
  - Compatibility -
  - Safety – no risk of
    - corrupting your filesytem & losing data
    - Bricking your device if system files corrupted
    - Rare, but can lose all your files… dept. has not time to fix!
  - Until you become a little more competent – but
- Avoid / ignore below for now, but for info. & later reference :
  - Mac OS X is a GUI on top of Unix BSD
    - So just use the Terminal app
      - Default shell may be zsh rather than bash
        - Can **ch**ange **sh**ell with   : chsh -s /bin/bash
        - And back to zsh with   : chsh -s /bin/zsh
  - Windows – Windows Subsystem for Linux (WSL)
    - Has bash and can install Ubuntu versions & others
      - (20/09/21) had malware reported, so avoid early versions
  - Chromebook Chrome OS – is built on a Linux variant
    - Supports add-on to run bash and Linux apps. (also Android)

Learning - What's important – is what you need..

- In education and in life you need the skills
  - to prune to bare essentials
  - & know where to get more info if and when needed.
  - The Basics – bare essentials – enough to get you around.
    - Overview – why, what & how…
    - Basic commands and facilities
  - The details – of where to get more info. When needed.
    - A more detailed overview without
      - overload & overwhelm & overstress
  - The details of where to get more info. / an answer
    - System : manual (man), apropos
    - Online : tutorials, examples, blogs, & online manual
      - Not the easiest to read, but comprehensive
        - https://www.gnu.org/software/bash/manual/
    - Books : basic – advanced – more later.

Learning - What's important – is what you need..

Give a man a fish, and you feed him for a day,

Teach him how to fish, and he feeds himself for life!

This course will not teach you all about Unix,

but it should give you all you need to know for now:

And know where to discover more

And follow most well specified procedures!

You should be able to read, understand and modify most

- installation and configuration instructions
- most scripts, except for complex inter process comms.
  - Where even the experts are exasperated!

What is Unix?

- A multi-user networked operating system
  - "Operating System"
    - Handles files, running other programs, input/output
    - Just like DOS or Windows
  - "Networked"
    - Designed for server use
    - Networking is an intrinsic part of the system
  - "Multi-user"
    - Every user has different settings and permissions
    - Multiple users can be logged in simultaneously
- Tons of fun!!! (Get a life!)
  - But it might get you a job or make it easier!

Basic Course Overview – enough to get going!

- Basic commands
  - Files
  - Directories
  - Access management - modes
  - Disk space management
  - Processes & process management
- Editors
  - Basic ubiquitous : (emacs isn't)
    - Ed /Sed / Vi(m) / pico/nano
  - GUI
  - IDE
- Tools & Utilities
  - Grep – basically to find strings, either names, contents, but flexible & programmable with regular expressions : a way of specifying string patterns (grep:- global regular expression print)
  - AWK (GNU open source gawk; modern : mawk) -
- Extras
  - bash scripting
  - ssh/VPN
- System Administration
  - System configuration
  - User administration
  - Software installation / compiles etc.

No point in reinventing the wheel,
So rather than developing a script,
Check commands and options,
And tools and options
Before attempting to write a script

Then write a script if unavoidable
Simplifying it by Using the powerful
Commands & options, and tools
covered above!

Simpler course structure : structures of

- commands
- To and from files
  - Redirection operators
- To and from other commands
  - (plumbing pipes & tees)
- Files – browsing around
- Help files
- filesystems
- Processes (if we get time!)

Handy shortcuts – to minimise t(r)yp(e)ing!?

- **Autocompletion…**
  - When entering a filename, give it a start, with a few initial characters and press TAB, and the system will autocomplete as far as possible:
    - If the start is unique, then it will complete the filename
    - If the start is not unique, then it will go as far as it can
      - Eg if in /users and type cd cs TAB

        then it will autocomplete to csdipact201

        and expect you to finish off with 2 or 3
- **History – saves retyping previous commands, handy to :-**
  - Fix an error, just run the cursor over (arrow keys not mouse) retype corrections
  - Re-run a complex command where
    - you are liable to make another error,
    - or can't be bothered rethinking and/or retyping it
  - Check history to see how you achieved such wonderful results!?
  - For another time? : can reference history by number and search

Fast moves – to minimise (re-)t(r)yp(e)ing!?

- These are based on emacs editor commands (default)
  - can also configure & use vi editor in cmd line - But for another time? : after looking at vi(m) etc.
- Fast Moves
  - Open Command line Terminal : Ctrl + Meta* + t
  - Line : to (start/end) of command line
    - Ctrl+(a/e) : Hints: a-start of alphabet; e - end
  - Words : to (previous/next) word (alphanumeric/not symbols):
    - Meta + (b/f) : back / forward
  - Characters :
    - to (previous/next) character :

      Ctrl + (b/f) : back / forward
  - Clear Screen : Ctrl + l (lowercase L)
  - Delete
    - current character : Ctrl +d
    - Delete previous character : backspace

* Meta is normally ALT but may need to be set in OS X, via

Terminal > Preferences > Settings > Keyboard, and enable 'Use option as meta key'

NB are referring to Meta in the context of a key on the keyboard,

not as a metacharacter acting as a control/modifier grouping char for shell or regex

Pronounced as in 'met a'

As I was going up the stairs, I met a man who wasn't there!
He wasn't there again today, I wish that man would go away

Fast cuts & pastes – aka kill and yank

- Cuts from cursor to
  - end of Line : Ctrl + k
    - To clear command line : Ctrl + A followed by Ctrl + k
  - end of Word : Meta +d
  - start of Word : Meta +DEL
  - Previous whitespace (excluding current) : Ctrl + w
- Pastes at current cursor position
  - Last cut : Ctl + y
  - Loop over past cuts and paste : Meta + y (only use after Ctrl + y)
    - Often referred to in manuals as rotate the kill-ring & Yank top

      - circular buffers not russian roulette…
  - Loop through and paste last argument of previous commands
    - e.g. when you want to apply a new command to a previous file

History : repeating the past!!!

Some say, If we don't learn from it,
we're doomed to repeat it!?
But others say, all we seem to learn from it is that we:
(don't learn from it and) are doomed to repeat it!

- But let's continue with CLI history which can be searched
  - Up and down arrows let us cycle through past commands
    - Which can be re-executed by hitting return
    - Or modified by backspacing and retyping, or fast moves
  - Ctrl + r *searchterm* to find a searchterm in history
    - Ctrl + r – repeatedly to find previous hits
    - Ctrl + j : to stop search and use the current hit
    - Ctrl + g : to get out of search and get back to original line

ALERT : May be confusing if you choose Vi(m) – so don't waste time on it!

Shell simpler Shortcuts – all u need 4 now

- Tab completion
  - Type part of a file/directory name, hit `<tab>`, and the shell will finish as much of the name as it can
  - Works if you're running `tcsh` or `bash`
- Command history
  - Don't re-type previous commands – use the up-arrow to access them
- Wildcards
  - Special character(s) which can be expanded to match other file/directory names
    - `*` Zero or more characters (like multiplication… A x 0 = 0!!!)
    - `?` Zero or one character (for the existentialists, unsure if they exist!)
  - Examples:
    - `ls *.txt`
    - `rm may-?-notes.txt`

Command structure

General command format

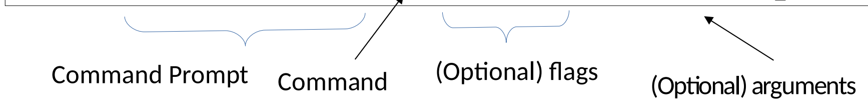
Learn to use the desktop too…
use the blue circle ?

Good help facility!

The Command Prompt

- Commands are the way to “do things” in Unix consisting of a command name with options called “flags” and arguments !!??
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

```
[prompt]$ <command> <flags> <args>
Cs1>:/abc123/~$ ls -l -a directory1
```



Command Prompt

Command

(Optional) flags

(Optional) arguments

Note: *(flags and arguments when bashing… football, politics…!?)*
1) In Unix, you're expected to know what you're doing.
2) Many commands will alert only if there was an error in it's execution; but could almost destroy your system without telling you! Careful!
3) Optional flags can be combined into a single contiguous continuous string preceded by a single dash '-' rather than being listed separately… So ls -l -a … can be replaced by ls -la … or ls -al … order is unimportant

Two Essential Info Commands

- The most useful commands you'll ever learn:
  - `man` (short for "*man*ual")
  - `Info` (somewhat hyperlinked… at least at the contents level)
- They help you find information about other commands
  - `man <cmd>` or `info <cmd>` retrieves detailed information about `<cmd>`
  - `man -k <keyword>` searches the man page summaries (faster, and will probably give more selective results)
  - `man -K <keyword>` searches the full text of the man pages

```
cs1:/abc123/dir1$ man -k password
passwd    (5)  - password file
xlock     (1)  - Locks the local X display
                until a password is entered
cs1:/abc123/dir1$ passwd
Lab system modified passwd cmd to work all OS'es,
so is not a normal Linux passwd cmd.
```

Two Essential Info Commands

- Info, as opposed to man, is category based
- Documents are hyperlinked
  - `info <cmd>` gives information on `<cmd>`
  - `info` by itself gives help on how to use it
  - Type `q` to quit.
  - Type `h` for help.

Online Alternatives

- Online
  - General GNU:- **G**NU’s **N**ot **U**nix
    - https://www.gnu.org/software/coreutils/manual/html_node/index.html#Top
  - system manpages should be for installed system – so may be most relevant
  - Online may be for a different system, but may be easier to process
    - but there are others for : Ubuntu
      - https://manpages.ubuntu.com/

Alternatives

- not all online; may need to install; installed outdated?
- Bropages – just get to the point – examples
- Cheat : cheatsheets at the command line
- But if not installed or no browser – no good
- TLDR – Too Long Didn’t Read
  - an interactive alternative too TLDR++
  - Can be installed,
  - BUT ALSO
    - Is Online
    - And has smartphone apps.

Scrolling around

Output from man etc. is sent through a page formatter with these commands

- space-bar or f or - forward to next page
- b - back a page
- p - to first page
  - (not to be confused with -P option
    - specifies the default pager
      - Which formats pages for display and is NOT an personal alerting system!
- Also arrow keys will scroll pages

Stopping a scroll roll & other things

Most modern systems are configured to output a page at a time,
so it doesn't all scroll by in a flash too fast to read,
but it is good to know the old handy standbys.

man

- Output
  - ^s - stop
  - ^w – write (continue)
- Input
  - ^d - end of file marker

Else pipe the command to a pager **less** or **more** e.g. **cmd | less**
Which needs a prompt (spacebar will do) to display next page.

Will see more in a few slides...

Similar commands to suspend/stop process

- Suspend a process
  - **^z** – puts it to sleep, catch some 'z's
    - Sends it to the background allowing interuption
    - Can resume with fg (foreground) or %x
- Cut a process off
  - **^c** – for Cut or kill a process
  - Another related command (for another time) is
    - **kill -flags #PID**

    but need to find PID first using **ps** command etc.
- And of course there are variations and catches, unless POSIX (Portable OS Info eXchange) compliant

Summary – take away for now

- Shells – for context, in case you wonder
  - many shells exist, were developed with styles & strengths
  - Many can be installed & switched, suited to purpose
  - Choose bash as most commonly bundled & used for scripts,
- A maze of commands & qualifiers – flags & arguments
  - But inbuilt help : relatively boring but comprehensive text
    - Man – text, with vi editor like interface
    - Info – basic hyperlinked text, with emacs like interface
- Typing seems a pain, but endless click & drag is a bigger pain
  - Shells are programmable, save years of pain
- Tricks of the trade
  - Tab completion
  - Command history (up & down arrows) & line editing ^a, ^e
  - Wildcards * ?
- Tip : try it and you'll see – not as bad as it looks, give bash a lash!

Files
– away, always, anyway
Perusal,
Redirection

Beauty of Unix - philanthropic files!?

Cuts out user having to manage (make & clear) temporary middle files!

- All data are handled as files, simply a byte stream includes devices : screen, keyboard, printer, and of course…disk, media streams and sensors
- Files can be
  - Redirected…using redirection operators ‘<‘, ‘>’ …

    ```
    cmd <inputfile  >outputfile
    ```

    - So screen output can be sent to a file instead
    - input taken from a file instead of keyboard & vice-versa too
  - Appended…>>append_outputfile
    - Added to, rather than overwriting, an existing file,
  - Piped from program to program : cmd1 | cmd2 | command3 …
    - Instead of the user having to create temporary files, with all the time, typing (errors), naming & reclaiming, the system does it all, by redirecting the output of one command into a temporary file buffer which is then used as input to the input of another, system reclaims space afterwards
    - Incredibly handy and powerful ‘one-liner’ commands composed from simpler commands, including own scripts.
    - Unix philosophy : Keep it simple, make it fast, do it well.
  - Teed – ‘tee’
    - Outputs from a program can be split : as in a T-junction

Files and directory perusal finding & searching done later...

- Any long display output (command, program or file) will race to the end of file uninterrupted so that only the end is seen, which is not convenient to browse file contents for whatever reason
- **head/tail -n filename**
  - Will display the first/last n lines of filename,
    - with n=10 as default.
      - But in Linux, lots of things, including defaults can be redefined
- **more** – the standard unix pager, displays a page at a time, until you hit
  - Return - next line
  - space_bar - next page
- **less** (is more in Linux, with extras… and is often adopted by Unix)
  - Allows arrows/page up/down to scroll up and down the page (& also f & b)
- Directories
  - Directory listing is just like a file display or output to the screen, so will race to the end, unless piped through more or less .. ls -l | less
  - Info from the manual is automatically piped and paged
    - Further information in about 10 slides...

Files and directory perusal finding & searching done later...

Elsewhere we see how to

- Create and edit
  - Files
  - Directories (albeit indirectly, via filesystem commands as you cannot directly edit directories – which are system files ….)
- Find specifics
  - Text In files
    - Strings : specified directly
    - String patterns : specified using regular expressions (regex)
  - Or files in directories
  - Or even text in files within directories...
    - recursively starting from the current directory.

But for now, assume directories and files exist.

Neat cat Tips – quick & handy for text display/input – edit later

- ***cat filename1 filename2 >filename3***
  - Concatenates or joins
    - the input files : *filename1 filename2*
      - *No designator need for the input file*
    - Into the output file *filename2*
      - *Designated by the redirection operator >*
- ***Cat file >filecopy***
  - Copies file to filecopy, but best to use copy command ***cp file filecopy***
- ***cat filename1***
  - displays the contents of filename1 on std. out – screen
    - No output file is designated, so standard output is used; the screen
- ***cat >filename1***
  - creates a new file named filename1 whose contents are whatever you type on the keyboard
    - No input file is designated, so standard input is used; the keyboard

```
~$cat >my_file
At last the cats …
… lost the match.
^d   (end-of-file character  CTRL+D pressed together, on new line)
~$
```

Re-direction operators

- <file read standard input from file
- >file write standard output to file

  (file will be created if not there, ***otherwise overwritten!***)
- >>fileappend (add to end) standard output

  to end of file – *will not overwrite existing file*

- << ? HERE documents - rarely used – as data is 'fixed/harcoded' in script

  Take data in for preceding command in a script

  directly from the following lines within the script,

  delineated by one or more characters after the angle brackets, in this case a '?'

```
<< ?
Dataline_1
Dataline_2
…
Dataline-n
?
```

Pipes for …

Pipes

- 'pipe' output from one command or process
  - Through intermediate files
    - Which the system automatically
      - Creates so they can be used, and saves user doing it
      - Clears up after use to save space
- To another command or process
- Linux/Unix treats all data streams like files; pipe/stream=temporary file
- And so on … e.g.
  - *cat myfile | grep key | sort | lpr*
- Or a normal shorter equivalent, with the first cat & pipe replaced
  - *grep key myfile | sort | lpr*
  - From the end: print a sorted list of lines containing key in myfile;
- More later: grep does global regular expression pattern matching

Typical use of pipes

These are only examples of pipes,
rather than a good illustration of how to use grep,
which can search through files within a directory, recursively, and will cover later.
The examples below only search the filenames for patterns, not the file contents, as the 'file' piped to grep, is merely the file listing from 'ls'

(and far better to use find command… designed for this specific task)

- ls -alR | grep searchterm
  - ls - alR will
    - List all filenames in long format Recursively from cwd
      - cwd = current working directory
        - aka : pwd=present working directory, also a command

Commands operated from within the cwd unless the command allows a specified path.

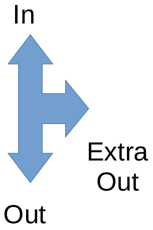
  - Piping the output to grep which
    - Will only output lines matching the searchterm
- e.g. ls -al | grep music
  - Will output all filenames from ls within the current directory which have music in their title
  - Saves waiting on scrolling and searching through huge output on screen

(and far better to use find command… designed for this specific task)

Brief note on grep & find

- **Find** – for filenames
  - finds ***filenames*** in a directory hierarchy, with many of the features of grep below
  - Has security issues in a multiuser environment
- **locate**
  - Runs faster when called since it
  - Searches a file index / database, already created, so
    - The system needs to build an index in advance
    - It may miss recently created (or deleted) files
  - is clearly less flexible than a DIY grep configured command
- **Grep** – for text (usually file contents, can do filenames a directory listing is piped to grep)
  - use regular expressions to specify ***text*** patterns primarily ***within file contents***
  - search, (recursively)
    - File contents
      - either directly by giving filenames, directories, etc
      - Or indirectly by piping through some listing command such as cat
    - File names, by piping just output from 'ls'
  - Runs much slower than **locate**, as it works through the filesystem when called
  - is more flexible than **locate**, which only works on pre-indexed filenames

Tee (join – like plumbing T-piece) e.g.to save & see



- Tee command - as in T-juntion - 2 ways
  - Copies std input to std output, and to named file(s)
  - lets you see the output on screen
  - And can pipe a copy to the next command
  - Functions like pipe, but placed differently
- Tee et_messagefile | mailx et_home
  - Takes standard input until ^d - just like cat – if there!!!
  - saves a copy in et_messagefile
  - and mails the message to et_home
- Ls -alR | grep music tee musicinfilename
  - Lists all files long format recursively,
  - Searching for filenames with ‘music’ in their name
  - Writing the output to the screen and to a file musicinfilename, creating it if it is not there, and overwriting it if it is
  - Use tee >>musicinfilename to append output to end of file

Beauty of Unix : do it your way!

- Can do it your way
  - Scripts : as a sequence of existing commands
  - If you can’t find a command then
    - Write a script (a program of existing commands) * to have the same effect
    - And if no script can be configured to do it, then **
      - write a program in C, Java, Python whatever,
      - And make that a command for yourself to use
      - And if it’s useful, others may use it too…
- And that’s how Unix grew…
  - Co-operative co-operation!
- Basis of Open Source

* Covered in this course

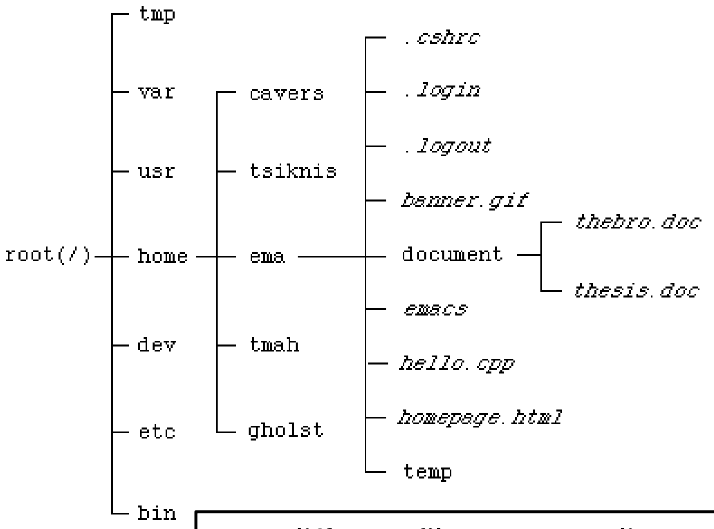
** Too advanced for this course

Filesystem :-
Files,Directories
Finding way around
Changing & (re)moving

Filesystem : Complexity management!

- Chaos -> order : classify & separate
- Or… ‘Divide and conquer”
  - E.g. fast sorting algorithm : quicksort
- Structure data in Files & filesystems according to
  - Type & function
  - Which implies
    - Filesystems :
      - function type : system, users, utilities
    - Files : data type : formats, headers
      - Text : programs, configurations, user credentials,
      - Binary : executables.
      - Special formats : databases, logs
      - Proprietary : optimised for a specific function,
        - (if only to lock a client into a product line!)

Unix Hierarchical Tree File Structure



Note different file types : endings are mostly for our convenience, all are just bytestreams to Unix!

File systems…

- What do we want?
  - To store data
  - in a structured manner
    - (in a file, in a filing system)
  - so we can
    - find it, structured organisation; index; search tools
    - use it, in programs, and command sequences
    - and change it if required…
- How do we do it? … a step at a time… this course being the first, of course!

Files in filesystems…

- in a structured manner
  - This implies some sort of structured filing system
    - Files : structure helps place and locate data within file
    - Filesystem : with a structure so files can be located
    - Code format : Unicode etc.
    - Header records : name, data type, file structure
      - Just know they exist and not be intimidated,
      - don't need to know details,
    - File / filesystem structure
      - Sequential … e.g. text.
      - Indexed … e.g. database

File systems…

- so we can
  - find it.
    - structured organisation; index; search tools
  - use it, … using application
    - commands
    - Command sequences
      - Scripts : generally imply interpreted & may be interactive
      - Programs : generally compiled & background
  - and change it if required…
    - By editors :
      - Manually and interactively
      - Some of which can be programmed
    - Or programs : such as
      - Dedicated applications
      - Programmed sequences of editing commands : Unix strongpoint!

File systems…

- To store data within a file
  - Requires some sort of encoding :
    - Usually just as text characters:
      - Text : ASCII – Roman alphabet,
      - Unicode : all alphabets
    - But can be in other formats
      - Binary : machine executables
      - Numeric format : for number heavy applications
      - Proprietary format : for efficiency for a given application
        - » (or the 'deadly embrace'…lock you in to their product!)
      - Encrypted for privacy
      - With error-corrrection coding …for error correction
      - Distributed for
        - » Speed : parallel access
        - » Fault tolerance : not having all your eggs in one basket

Bigger picture – other file types

- **Linked files**
  - not a copy, just a reference…saves space
  - Hard… links to inode, file stays until all hard links gone (inode is
  - Soft… links to name, link lost if name gone
- **Directories** – files which appear to store other files, but only store names & links

The following ones are generally not seen in basic use of Unix:-

- **Pipes** are powerful extensions for stringing simple commands to create powerful one-line programs, with a few versions.
  - Unamed pipes:
    - temporary files created by the system to 'pipe' output from one process into another; the process can be a program or command.
  - Named pipes
    - Extension to unamed pipes with the same purpose and behaviour, but exist as static non-volatile byte-stream
- **Sockets** are similar to pipes in function, but are data types are extended from byte-streams to datagram sequences used in networking protocols, offering compatibility with networked Inter Process Communication (IPC)

Files.

- Create
  - Use an editor :
    - Trad ubiquitous unix : ed / sed / Vi(m) / pico / nano /
    - Newer GUI text editors & IDE's
  - or the cat to concatenate a few files together
  - Or output from a program.
- Names, paths and Access modes
- Directories / subdirectories etc.
- Copy
- Move
- Remove
- Disk space management

Basic File commands

- **Create**
  - Using editors, program output, or the cat to concatenate a few files together
  - If no input file is specified and output redirected to an output file, then that output file is created within the current directory
  - **cat >newfilename**
- **Copy** – to copy a file

  retaining the source
  - **cp sourcefilename destinationfilename**

  Where source and destination are filenames…
- **Move** – to move a file to a new destination,

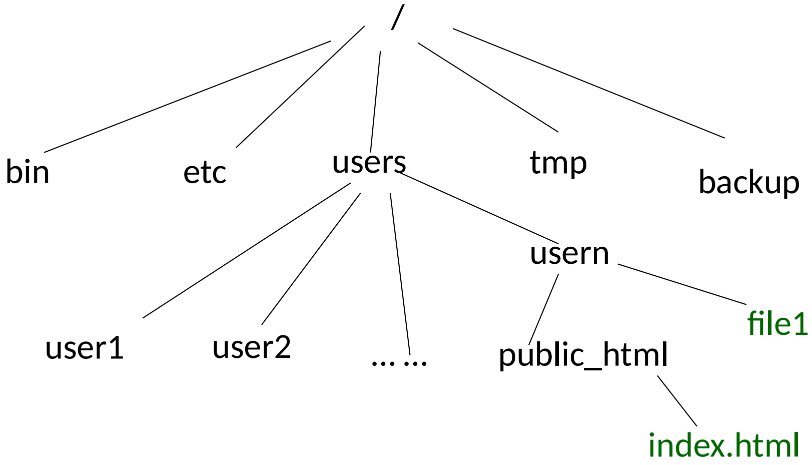
  removing the source : effectively renaming

  usually achieved by copying source to new location, with removal of old.
  - **mv sourcefilename destinationfilename**
- **Remove** – to remove a file : basically deleting it
  - **rm sourcefilename**
  - NB **rmdir** is recommended for directory removal, as it will only delete empty ones; as it cannot remove files within the directory.
  - But **rm** with
    - **-**d will attempt to delete directories;
    - –R options will attempt to delete everything Recursively, including directories.

What is a directory?

Directories can hold files and other directories

Imposes an organised filing structure on files!?



What is a Directory?

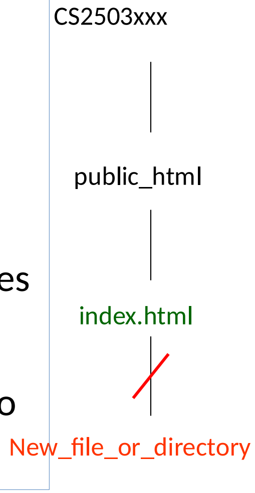
A file cannot hold a directory or a file!

A collection of files grouped together, either by the user or system, for ease of management.

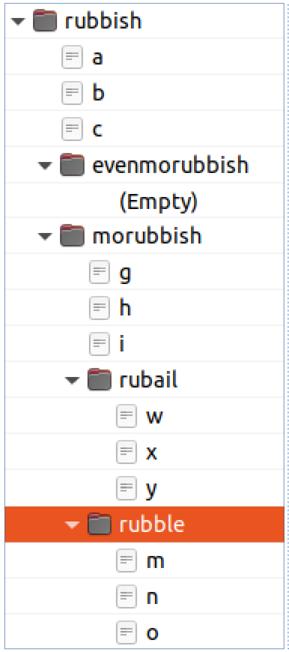
Your home directory typically contains a public_html directory.

Your public_html directory typically contains an “index.html” file, which indexes other dirs & files for data

But any file cannot contain another file, so index only points to!



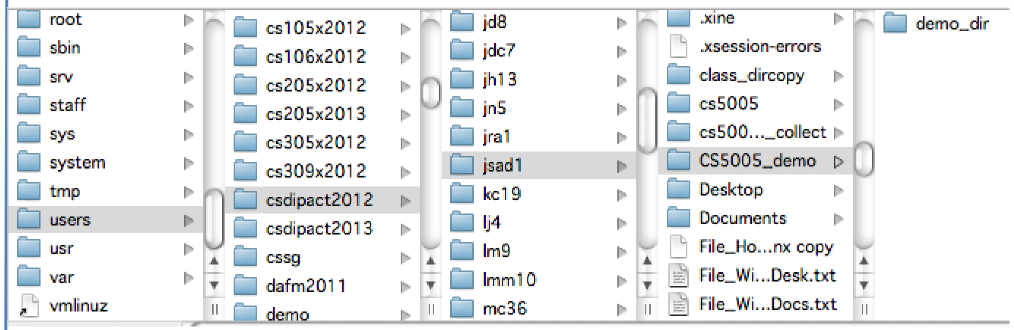
GUI (screenshot) vs CLI { ls -R } vs tree {may not be installed}



```
:~/OS1T$ ls -R
.:
rubbish

./rubbish:
a  b  c  evenmorubbish  morubbish

./rubbish/evenmorubbish:

./rubbish/morubbish:
g  h  i  rubail  rubble

./rubbish/morubbish/rubail:
w  x  y

./rubbish/morubbish/rubble:
m  n  o
```

```
/OS1T$ tree
.
└── rubbish
    ├── a
    ├── b
    ├── c
    ├── evenmorubbish
    └── morubbish
        ├── g
        ├── h
        ├── i
        ├── rubail
        │   ├── w
        │   ├── x
        │   └── y
        └── rubble
            ├── m
            ├── n
            └── o

5 directories, 12 files
```

Directory – holds files, incl. other directories
Files – cannot hold files, or directories

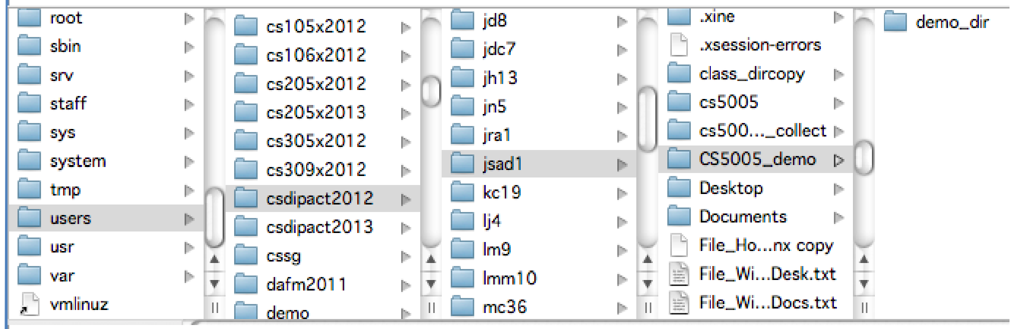


- Views
  - Treeview : as above, or something similar
  - Grid – icons on your desktop, size may vary
  - Coverflow – iTunes like flypast
  - List :

    the old reliable...and most flexible ... fields often customisable

    (indentation can convey tree structure in list!)
  - Hybrid : some mix of those above...

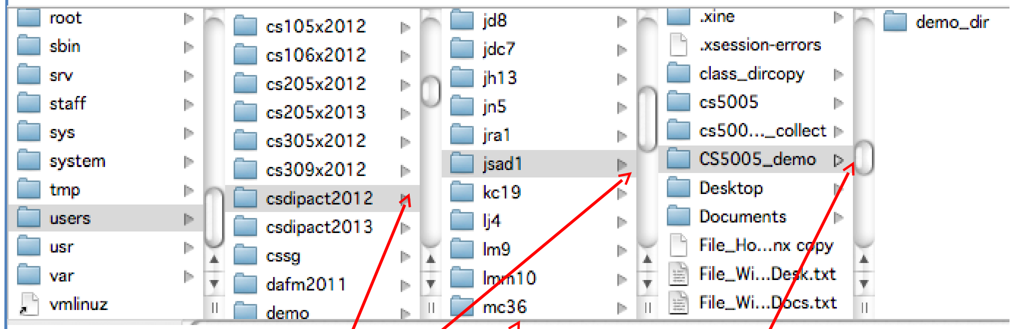
Directory – holds files, incl. other directories
Files – cannot hold files, or directories



NB

- The root directory of the filesystem is in the leftmost column in this tree view.
- The directory named root, within that column is:
  - the home directory of the root user account
  - And not the root of the filesystem tree.
- Most other directories at the root level are system related
  - Can ignore for now, and in most cases, forever!

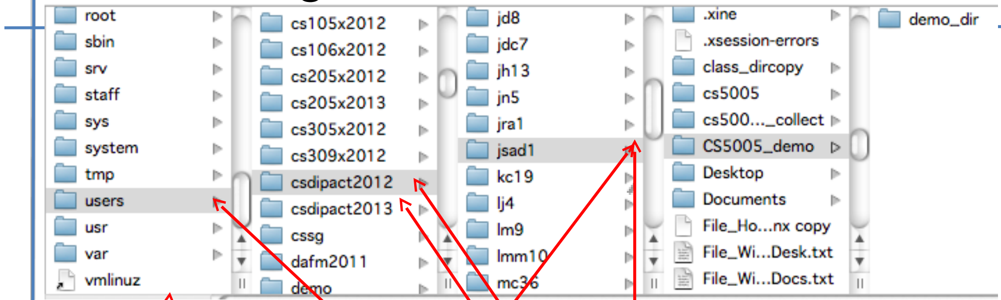
Check out the neighbourhood..!
GUI File Manager vs Terminal / Konsole



```
cs1> pwd
/users/csdipact2012/jsad1
cs1> cd ..
cs1> pwd
/users/csdipact2012
cs1> ls
am29  ... Jd8 ...  jsad1 ... mc3 ... Sx1
cs1> cd jsad1 (or cd ./jsad1)
```

```
cs1> pwd
/users/csdipact2012/jsad1
cs1> ls
class_dircopy   Music           Shopping
Cs5005              Pictures         sum
Cs5005....        _collect         Public
Templates
CS5005_demo  public_html     Videos
Desktop          rubb        Windows
```

Rooting about...from root to home!



```
cs1> cd /
cs1> pwd
/
cs1> cd users/csdipact2012/jsad1
cs1> pwd
/users/csdipact2012/jsad1
cs1> cd /
cs1> pwd
/
```

```
Or use relative links...
cs1> cd ~
cs1> cd ../../csdipact2013
cs1> pwd
/users/csdipact2013
```

```
cs1> cd ~
cs1> pwd
/users/csdipact2012/jsad1
cs1> cd ..
cs1> pwd
/users/csdipact2012
cs1> cd ../csdipact2013
cs1> pwd
/users/csdipact2013
```

Structured Filesystem

- Filesystem / Directory tree
  - Moving about : to get or put what & where
  - Adding/Removing your own (sub)directories
- Files ...
  - Cannot store other files or directories, just data
  - Create, edit and remove.
  - Edit
- Directories are 'special' files, giving 'directions' to others
  - Which merely list other files, including directories 'within' them,
    - but in reality just point to other files
      - which are logically gathered internal to the directory,
      - but physically external to the directory on other disk blocks..

(sub)directory directions : Names, paths and Access modes

- Files are stored in topically or logically related groups which are
- specified by a **pathname** or **name**
  - the path of directory names to access the file, separated by '/' .

  ...e.g. **/dir1/dir2/dir3/localfile** **/users/csdipact2012/jsad1/sum**
- In one of two ways
  - Either **full** or **absolute** to the root of the system directory tree
    - Signified and starting with:- **/full_pathname_from_root**
  - Or **relative** to
    - The current directory
      - Signified and starting with:- **localfilename**
    - The parent directory
      - Signified and starting with:- **../sibling_directory/nepotism**
    - The user's home directory
      - Signified and starting with:- **~/pathname_within_home_dir**

Directories

- In Unix, files are grouped together in other files called *directories;* analogous to *folders* in Windows
- Directory paths are separated by a forward slash: /
  - Example: /cs1/abc123/classes/cs1235
- The hierarchical/tree structure of directories begins at a special directory called the *root*, or /
  - *Absolute paths* start at /
    - Example: /cs1/abc123/classes/cs1235
  - *Relative paths* start in the current directory
    - Example: classes/cs1235 (if you're currently in /cs1/abc123)
- Your home directory is where your personal files are located, and where you start when you log in.
  - Example: /cs1/abc123

Handy moves!

- Pwd – (Print Working Directory) : absolute pathname of current working directory
- cd *[dir]* – change directory
  - "~" – tilde ~ refers to home dir...meandering home*..!?*
  - *Cd alone with no argument has the same effect.*
  - back to parent directory. "**..**" 2-dots for 2 parents is the relative pathname to the parent directory.
  - "." -stands for current (working) directory – one dot.

NB Directories cannot be directly edited by a user, but changed only by making and (re)moving files (incl. Directories) within them.

Otherwise like data files within the system.

But must be executable for user to peruse.

Directory Navigation - review

- Directory Paths
  - unlike MS-DOS, UNIX file systems use forward slashes
  - Eg. /user/csdipact2013/your_id/ (UNIX)

    \mydocuments\myword\ (MS)
- Change Directory: cd *directory_path*
  - Go up to parent directory: cd ..
  - Go to home directory: just cd or cd ~
- Useful Commands
  - pwd: returns "present/print working directory"
  - ls: "list contents of directory"
    - ls : tab delimited list
    - ls –l : one entry per line, long format with file attributes
    - ls –a : list all files, including hidden (system) ones..
    - ls –la : lists all files incl. hidden files + shows file attributes
    - ls –al : same as above, flag order irrlevant here… but not always!

Directories continued…

- To delete files
  - rm *file_name*
- To create files within a directory
  - For speed and ease, if it's simple text, use catenate

    which normally concatenates (jams) a filelist into a single file

    but here is used this way: cat >*new_file_name*
    - no input file is specified, it defaults to standard input (the keyboard)
    - This redirects standard input, the keyboard, to the new file ,
    - finish with CTRL+d denoting end of file; end of input.
    - CTRL+D is often used to end things in Unix.
  - Or use an editor…
    - Nano
    - Kate:KDE Advanced Text Editor
    - Vim : Vi improved; vi = visual ;-0
    - Emacs : highly programmable editor, powerful, flexible, complex
  - Or any text editor…

What's a directory?

- Files are grouped in the directory structure.
- The file-system is arranged like hierarchical tree (inverted)structure.
- The top of the tree is called "root" which usually contains several sub-directories.
- In UNIX "/"(forward slash) is used to present the "root", and subsequent levels in the tree.
- *Some variants have*
  - a directory called 'root' i.e. '/root'
  - at the root – i.e. '/'
    - not to be confused
- The directory 'root' - '/root', at the root of the tree, '/'
  - Is just the home directory of the root account in some implementations.

Directions…from a directory …

Remember, this was before GUI's, could not see the big picture!

Think or text output only, needed to know location, and options at each level.

- Directories are like
  - Fingerposts at each node in the directory tree
  - Tables of contents (abbrev. toc – common in CS)
  - Index
- Which show
  - Options and contents at each level
  - Effectively a path to a file.
- Filenames are effectively pathnames to a file!

  /users/my_group/my_homedir/my_subdir1/

  /users/csdipact2013/
- NB : backslash '\' in Microsoft (MS-DOS, Powershell)
- But going forward in Unix '/' (or forwardslash)

Directory navigation

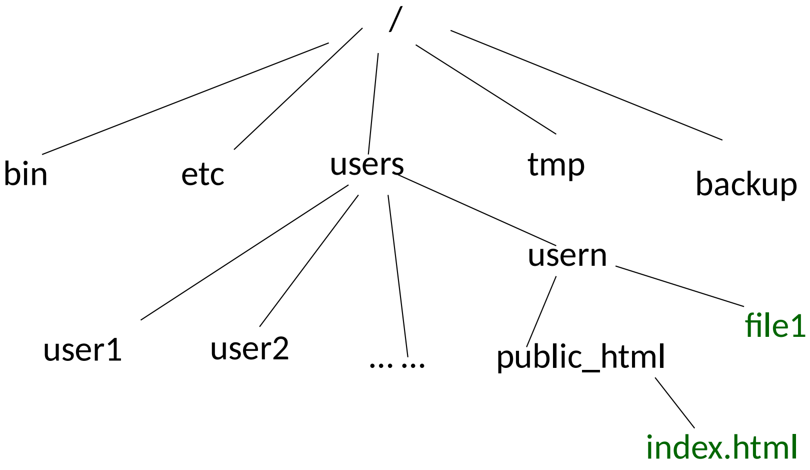
- **Where** am I? Print (present) working directory : **pwd**
  - System will display your current directory,
    that is : where you are in the filesystem tree
  - Note some command prompts (the bit at the start of a line, inviting a new command when finished the previous one) give partial information on this already.
- **What**'s here? List directory contents : **ls**
  - This will give a list of files in the current directory
    - Desktop Documents Downloads …etc

    Common qualifiers are listed in the next slide
- **Where** can I go? **How** to change directory? : **cd**
  - cd destination_directory

  the destination_directory is specified the same way as any other filename, as outlined on the previous slide, including full absolute and relative names: /users/csdipact20nn/urid/Desktop/snap
  ~/Desktop/snap… etc.

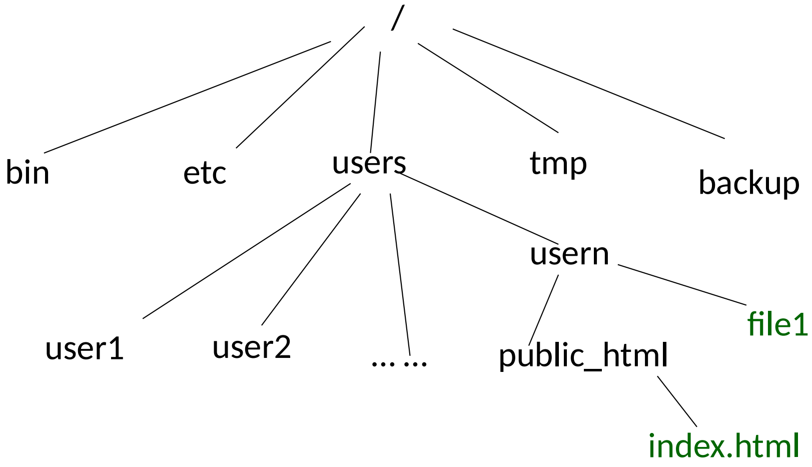
Specifying Paths (filename=path)

What is the **absolute path** to index.html?


/
bin
etc
users
tmp
backup
user1
user2
… …
usern
public_html
file1
index.html

Specifying Paths (filename=path)

What is the **relative path** to index.html (assuming that usern is your pwd)?


/
bin
etc
users
tmp
backup
user1
user2
… …
usern
public_html
file1
index.html

Pathnames=how to reach a file

- Absolute Pathnames
  - In the previous tree /users/usern/file1 is an absolute pathname.
- Relative pathnames
  - If you are already in the users directory, the relative pathname for file1 is usern/file1.
  - If you were already in another user directory, at the same level in the tree, the relative pathname could be written as
    ../usern/file1

Where are we (going!)?

- Where are we?
  - pwd - present/print working directory …

    /users/csdipact2012/jsad1

    by default you login to your home directory,
    although almost everything can be configured in Unix

  - Returns a pathname from root of directory tree
    - Although system configuration settings can be changed…to display a relative pathname (e.g. within user's home directory)

- What's there when we get there?
  - List the directory contents using - ls

How do we get there?

- cd – change directory…
  - Absolute - starting from the root …
    - cd /users/my_group/my_homedir/my_subdir1/
  - Relative
    - To home directory
      - If I'm at home : cd my_subdir1 … or … cd ./my_subdir1

        . Denotes the current directory, needs ./ or omit entirely
      - Or if I'm not : cd ~/my_subdir1

        ~ is a shortcut for your home directory!
      - Or if I'm in a directory with a common parent

        e.g. in csdipact2013 & want to go to csdipact2012

        cd ../csdipact2012

        .. Denotes a parent directory, in this case /users…

What's in a directory – list directory - ls

ls *[names]* – list files contained in a directory *name* or that match a file *name*.

If no *name*, it will list all files in current directory,

Wildcards useful **ls *.txt** - lists files ending in .txt

- *ls –a* list all files including hidden files
- *ls –l* list in long format (including details like permissions, owner, size, etc.), works very much like *dir*
- *ls –al* list all files (including hidden files) in long format
- *ls –dl dir_name* lists long information about the directory, "dir_name".

Other flags can give info on mod times etc.

What's in a directory – list directory - ls

Common qualifiers:-

ls just gives a tab separated list of filenames

```
Desktop     Documents     …etc
```

ls -l (letter 'el') lists 1 per line…**with a header line** giving total blocksize for dir.

```
cs1> ls -l
total 96
drwxr-xr-x  3 jsad1 csdipact2012 4096 2012-09-29 12:15 CS5005_demo
drwx------  2 jsad1 csdipact2012 4096 2011-09-29 09:45 Desktop
drwx------  9 jsad1 csdipact2012 4096 2012-09-29 13:53 Documents
```

ls -1 lists as 'one' per line **without the header line** indicating dir. size… handy for counting files, by piping to **word count** with a **linecount** flag **–l**

Ls -1 | wc –l

(using ls –l would give wrong filecount, as it includes top line 'total 96')

More on ls qualifiers..

ls -a - gives **all** files, including some hidden system ones beginning with a ' . '

```
.      ..     .adobe    .bash history etc.
```

Note dirs also have pointers to current (.) and parent (..) dirs in addition to internal files/dirs

ls -l - gives a **long** detailed listing, showing lots of file attributes,

```
total 92
drwx------      2 jsad1 csdipact2012 4096       2011-09-29 09:45 Desktop
drwx------      7 jsad1 csdipact2012 4096       2011-11-28 15:14 Documents
 -rw-------     1 jsad1 csdipact2012    72      2011-11-07 14:31 sum
```

ls -al - gives long detailed listing, including hidden system ones beginning with a ' . '

```
total 376
drwx--x--- 43 jsad1 www-data      8192 2011-12-07 13:06 .
drwxr-xr-x 38       root  csdipact2012   4096 2011-10-24 14:46 ..
drwx------  3   jsad1 csdipact2012   4096 2010-10-14 15:35 .adobe
-rw-------  1    jsad1 csdipact2012   8241 2011-12-19 22:23 .bash_history
```

Starting letters: b, c, d, l, s, p (just be aware others exist, normally only encounter: '-', 'd' & 'l')
indicate : **b**lock, **c**haracter (special), **d**irectory, **l**ink(soft symbolic), **s**ocket, **p**ipe (static named FIFO, as opposed to temporary ones generated during commone pipes ls -1 | wc –l etc..

Recursive…application to all subdirectories

- ls –R (UPPER CASE 'R')
  - Will recursively list all files within
    - The current directory
    - And any sub-directories
      - Recursively

But the output is still not very nice,

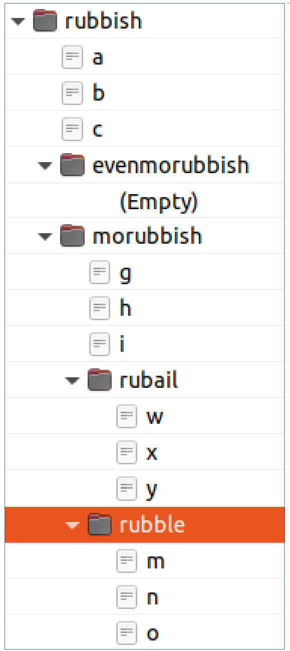
Better to use tree...if installed.

- ls -r (lower case 'r')
  - reverses the default display sort order
- ls -t displays according to modification time,
- ls -rt displays in reverse order of above

NB flags such as r are not case consistent across commands...

For some commands -r is recursive, others require R...!!! Chaos

GNOME Files vs CLI {: tree vs ls -R}



```
rubbish
    a
    b
    c
    evenmorubbish
        (Empty)
    morubbish
        g
        h
        i
        rubail
            w
            x
            y
        rubble
            m
            n
            o
```

```
/OS1T$ tree
.
└── rubbish
    ├── a
    ├── b
    ├── c
    ├── evenmorubbish
    └── morubbish
        ├── g
        ├── h
        ├── i
        ├── rubail
        │   ├── w
        │   ├── x
        │   └── y
        └── rubble
            ├── m
            ├── n
            └── o

5 directories, 12 files
```

```
:~/OS1T$ ls -R
.:
rubbish

./rubbish:
a  b  c  evenmorubbish  morubbish

./rubbish/evenmorubbish:

./rubbish/morubbish:
g  h  i  rubail  rubble

./rubbish/morubbish/rubail:
w  x  y

./rubbish/morubbish/rubble:
m  n  o
```

Basic Directory commands.

- To create a new directory:
  - mkdir *dir_name*
- To remove an empty directory:
  - rmdir *dir_name*
- To remove a directory that has files and subfolders:
  - rm –R *dir_name*

For safety :
use rmdir rather than rm
As it blocks deletion of
a non-empty directory

- *Hazard…delete all your files from your home directory*
  - go to home directory : cd ~ or just cd (default ~)
  - rm –R * remove recursively all files

  * is wildcard matching all text strings

NEVER RISK : rm – R , If you are root user or have sysadmin privileges, as you can wipe virtually the entire filesystem, if you are in the root directory – normally blocked on most systems.

Wildcard ‘ * ‘ : with risk of ‘wild’ behaviour!

- ‘ * ‘ is powerful and dangerous…always
  - Even for experienced (& absent minded) users.
- It has 2 context dependent meanings

1. And In the shell ‘ * ’ can mean any string, basically any number of any characters
2. But in Regular Expressions, used elsewhere, editors, grep (for text find), prog. Languages, it means any number of the previous character!
3. And to complicate things Regular expressions can be used within some shell commands!

- . GOOD ADVICE :
  - ALWAYS BEFORE USING :
  - ***rm file****, (and even without the –r recursive flag) check by issuing an ***ls file**** to see what files will be involved …

Shell Metacharacters (expanded by the shell)

- based on (but incompatible variant of) Regular Expressions

Characters with special meaning to the shell

*, ?, [...],

* matches any grouping of zero or more characters

   <u>- but in RegEx zero or more of preceding char only</u>

? matches any single character

[...] allows matching a range of characters

[0-9] matches any digit

[A-Z] matches any capital letter

[a-d] matches a, b, c, or d

[aeiou] matches any vowel

Metacharacter Examples

Suppose a directory has the following files:

```
Boy    toy    coy    cow    soy1   soy2    soy.1  soy1.1
bow    mow1   mow2   mow3   say1.1 say1.2  say1.3
hay    shay   tray   fray   flay   chow  slay   bay     buy
```

How do we select groups of these files using metacharacters?

Will see more later when we do regular expressions but basic idea is :-

*	any string of characters of any length
?	any single character *but in regex 0 or 1 of previous char*
[set]	any <u>one</u> of a set of characters, here s, e or t
[a-e]	one of the alphabetic sequence a-e, inclusive
$ ls *[oa][wy]	names ending in either a or o followed by w or y.
$ ls *.?	names ending in a full stop followed by one char i.e. the dot ones .1, .2, .3

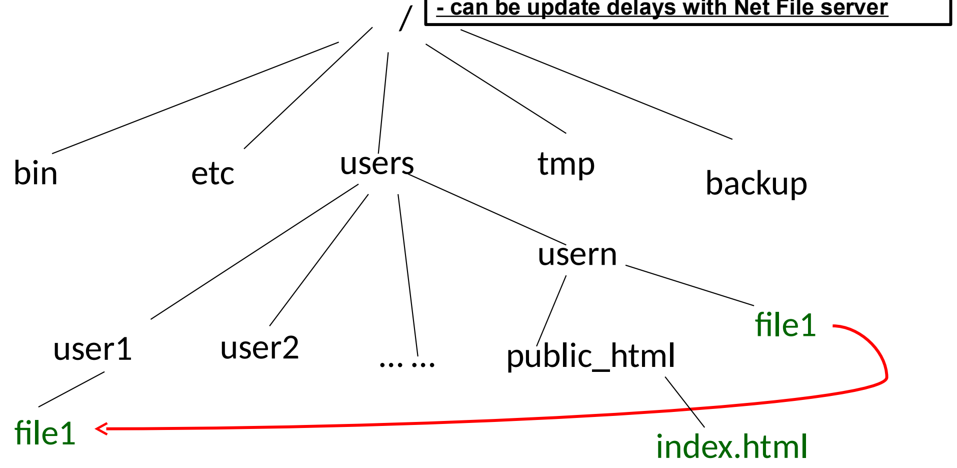
Difference But in shell regex a ‘.’ just denotes any character!

Move (mv) & Copy (cp) cmd /users/usern/file1 /users/user1/file1

If permitted both will overwrite (&lose) any destination files with same name

-i Interactive flag checks first…
Do you want to overwrite file...?

mv removes or renames original file but
mv dir1 dir2,
actually inserts dir1 into dir2
mv moves a directory into a target directory (inconsistent with mv on files, but avoids directory being overwritten…)
<u>- can be update delays with Net File server</u>



Basics - simple

1. **File to file** – no problem, but may overwrite existing file,
   - unless interactive flag is used or set (via command alias) to check with user first before overwriting
   - Or --no-clobber flag (new long format so prefix - -) which just ignores mv if clash occurs
2. Moving a **file to a directory**
   - *should normally* copy it into the directory,
   - and overwrite any file with same name there,
     - unless no-clobber or interactive mode checks.
3. Moving a **directory to a directory** *should normally* copy it into the target directory, irrespective of whether the names are identical.
4. Clearly can't move anything (file or directory) into a file, with such commands, but can
   - Overwrite file contents with new file (1 above)
   - Append / concatenate file contents (Cat >>appendage)
   - Can redirect directory listing into file for logging!?
   - And of course do whatever with an editor!?
5. Force -f flag, overrides interactive settings even if set as default ..
   - so rm -r might ask for confirmation… but rm -rf just wipes recursively with no warnings!

Moving files about

1. File to file
   - No problem, **but** will overwrite existing destination file unless
     - -i flag for interactive e.g. mv -i .. .i for interactive
     - --no-clobber is used to ignore if clash occurs
     - system/command is modified, via alias with either of above or another
2. File to directory
   - Since a file cannot hold other files like a directory, files are copied **into** destination directories
3. Directory to either a new or into another directory
   - Sure, **but** cannot put a directory into itself!
4. Directory to file – no way!
   - Impossible, since a file cannot act as a directory,
   - However, a directory listing can be stored as text in a file
   - ls > dirlisttext or append using ls >>dirlisttext
5. Odd exceptions and rules exist which **force** a **backup** when trying to copy a file into itself
   - cp -fb file file will create a backup of file; the backup will be named file~
   - -fb can be extended to more intelligible newer (with 2 dashes) --force – backup

copy/move files & dirs – Ubuntu16.04 on csgate (Aut 2020 – approx. behaviour – devil in the details)

source	dest	Overwrite alert/exception	destination exists	Destination Not there
file	file	-i :interactive, -f:force, --no-clobber for all	overwrite; Unless -i	Create (cp) Rename (mv)
file	dir		cp/mv insert; no overwrite	Create
dir	dir	cp needs flag -r recursive	Overwrite (cp) Refuses (mv)	create

* Aut 2020 – software app compatibility & stability

- Can't copy a directory into a file
  - But can redirect directory list as text into a file
- cp/mv will create/rename file if destination doesn't exist
- cp needs a recursive flag for a directory, mv does not.
- Overwrite alert: interactive: checks; no clobber: won't; f-forces. Can do lab exercises with examples, but might drive us daft!(er!)

Files: - Re/Move, copy

- **Copy** – to copy a file **retaining** the source
  - **cp sourcefilename destinationfilename**

  Where source and destination are filenames…
- **Move** – to move a file to a new destination, **removing** the source : effectively renaming by copying source to new location,

  with removal of old 'sourcefilename'
  - **mv sourcefilename destinationfilename**
- **ReMove/delete** – to remove a file
  - **rm sourcefilename**
  - **rmdir empty_directory (also rm -d)**
  - **rm -r directory_and_contents**

Files: Accidental deletion Hazard

- In both cases (cp & mv)…for files but not directories
  - destination file will be created if it does not already exist,
  - If the destination file already exists, it will be overwritten!
- **Hazard**
  - If a file already exists with the same destination filename,

    it will be overwritten and original contents lost

  Unless these flags are used (when destination file exists)
  - –no-clobber – will do nothing and move on;
  - –i (interactive flag) will check with user before overwriting,
  - unless suppressed with the –f (force) flag,
  - last specified overrules – a rule in theory, but not in practice!?

Seems odd to choose to ignore an alert, except when
- Sure : such as a script has already checked filesystem,
- to avoid interruptions, as in a non-interactive script

Avoid Accidental deletion Hazard

Check before wreck!

- Check the wildcard filelist before action
  - ls wildlist*
- list all files for the wildlist* pattern
- Before applying commands like
  - rm wildlist*

Files – move / copy **files to a directory**

- Clearly a data file is not a directory,
  - **Can't move or copy a data file to become a directory,**
    - Instead Files are moved into the target directory
  - **Definitely can't move or copy a directory to a file!**
    - (but can redirect a directory listing into a file as text!)
- Absolute pathnames can be used
  - Long but no confusion
    - cp /users/urgroup/urid/urfile /users/urgroup/urid/urdirectory
- Relative pathnames are handy – same effect as above
  - But easy to make a mistake or be in wrong directory!
  - E.g. if you are in your home directory, containing both :
    - cp urfile urdirectory
- cp copies the source to the destination.
- mv really just renames the file to the new pathname
    - Remember: filename=file location=file_pathname

Same effect
If both in same
(in this case home)
directory

Directory – Re/Move, copy

- Create
  - Make directory:- **mkdir**
- Remove – to remove an empty directory: basically deleting
  - **rmdir sourcefilename**
- Remove – to remove a non-empty directory: basically deleting
  - **rm -r sourcefilename**
  - the recursive flag(-r –R)* means that the entire filesystem tree within the directory is affected,
  - Case of r is inconsistent across commands –
    - eg -r => reverse order for likes of ls with s (size) t (time) flags
  - Recursive :
    - vital for filesystem work, all branches from location
    - runs-again within every subdirectory encountered,
  - Dangerous :
    - eg rm –r * will delete everything in the current directory
    - With root access, the entire filesystem can be deleted!
    - *** DON'T RISK IT *** (is obstructed/removed on many systems)

Directory – Re/Move, copy

- Move and copy work identically, except
  - copy retains original copy of source (file or directory)
  - Move removes original copy of source (file or directory)
  - mv does not need, but cp **must use the recursive option** to work for source directories, whether empty or not
    - Because all directories include two hidden entries to the current and parent directories . & ..
      - ls –a shows these hidden ones
    - Otherwise system will generate a message:

      $ cp: omitting directory 'source name'
- If the destination directory name
  - Exists
    - A copy of the source directory & contents will be inserted
  - Does not exist :
    - it will be created with the contents of the source directory

But the real deal is more complex

- Most filesystems use links…to save
  - Space : only one copy, but can pop up anywhere
  - And time : only one copy, so only one edit!
- BUT ALL OF THESE copy, (re)move commands have peculiar exceptions & rules with respect to links.
- This is further complicated, because there are different types of links with different behaviour.
- And complicated even further on our system, which has a central file server, therefore:
  - has update delays especially when loaded, so results display are delayed… unsure of state.
  - and does not handle some of the more obtuse aspects of Unix links properly.

Directory manipulation

Blinking links…
… every solution only makes more problems …
 - links to files save space… but not our time!!!
Will cover soon, but command link handling..!*&!

- Create
  - Make directory:-**mkdir**
- Copy – to copy a directory
  retaining the source
  - As for files :-   **cp sourcedirname destinationdirname**
  - And if desired, **recursively**
    - Use the –R option with the command : **cp –R source dest**
    - (all files within, including hard, but not soft links (unless –L is also used), are also copied)
- Move – to move a file to a new destination directory,
  (the destination directory must exist and not be a simple file, or it will fail!)
  removing the source : effectively renaming
  usually achieved by copying source to new location, with removal of old.
  - mv sourcefilename destinationfilename
  - Again if desired **recursively** (all files within are also copied)
    - Use the –R option with the command : **cp –R source dest**

(So the command will suceed, some commands support a -p flag, to automatically create missing parent directories in a specified destination path: /new_granny/new_par/new_file)

- Remove – to remove / delete a directory, but directory must be empty : safeguard
  - **rmdir sourcefilename**
- NB **rm** with **–d** or **–R** options will attempt to remove all files, including directories.

Directories

- In Unix, files are grouped together in other files called *directories*, which are analogous to *folders* in Windows
- Directory paths are separated by a forward slash: /
  - Example: /cs1/abc123/classes/cs1235
- The hierarchical structure of directories (the directory tree) begins at a special directory called the *root*, or /
  - *Absolute paths* start at /
    - Example: /cs1/abc123/classes/cs1235
  - *Relative paths* start in the current directory
    - Example: classes/cs1235 (if you're currently in /cs1/abc123)
- Your home directory is where your personal files are located, and where you start when you log in.
  - Example: /cs1/abc123

Directories (continued)

- Handy directories to know

  ~ Your home directory

  .. The parent directory

  . The current directory
- `ls`
  - *LiSts* the contents of the specified directories (or the current directory if no files are specified)
  - Syntax: `ls [<file> … ]`
  - Example: `ls backups`
- `pwd`
  - *Print Working Directory*

Directories (continued further)

- `cd`
  - Change Directory (or your home directory if unspecified)
  - Syntax: `cd <directory>`
  - Examples:
    - `cd backups/unix-tutorial`
    - `cd ../class-notes`
- `mkdir`
  - *MaKe DIRectory*
  - Syntax: `mkdir <directories>`
  - Example: `mkdir backups class-notes`
- `rmdir`
  - *ReMove DIRectory*, which *must be empty*
  - Syntax: `rmdir <directories>`
  - Example: `rmdir backups class-notes`

Files

- Unlike Windows, in Unix file types (e.g. "executable files, " "data files," "text files") are *not* determined by file extension (e.g. "foo.exe", "foo.dat", "foo.txt")
- Thus, the file-manipulation commands are few & simple … no file 'extensions' or 'extension' specific variants!
- Many commands only use 2 letters (keyboards : rare, tough, bash 'em!)
- `rm`
  - ReMoves a file,
    - ***without a possibility of "undelete!"***
    - ***unlike GUI to trash…***
      - ***Unless disk recovery tools are used, but no guarantee!***

  Servers run nightly (if not hourly) backups… but what about yours!?

  - Syntax: `rm [options] <file(s)>`
  - Example: `rm tutorial.txt backups/old.txt`
  - -r option: `recursive (delete directories)`
  - -f option: `force. Do no matter what`

Files (continued)

- `cp`
  - *CoPies* a file, preserving the original
  - Syntax: `cp [options] <sources> <destination>`
  - Example: `cp tutorial.txt tutorial.txt.bak`
  - -r option: `recursive (copies directories)`
- `mv`
  - *MoVes* (renames) a file or directory, destroying the original
  - Syntax: `mv [options] <sources> <destination>`
  - Examples:
    - `mv tutorial.txt tutorial.txt.bak`
    - `mv tutorial.txt tutorial-slides.ppt backups/`

Note: Both of these commands will over-write existing files without warning you!

Limits on freedom: permissions

Permissions for files

- Files are owned by both a user and a group
- You will either belong to ugrad_cs or year_name – but just false groups for grouping home folders – not real filesharing ones
- Each file has 3 sets of group permissions (**u, g, o, a**)
  - Permissions for the user who owns it (user permissions)
  - Permissions for group that owns it (group permissions)
  - Permissions for everyone else ('other' or 'world' permissions)
  - Permissions for all groups above
- There are 3 types of permissions
  - read -- controls ability to read a file
  - write -- controls ability to write or change a file
  - executable -- controls whether or not a file can be executed as a program
  - Directories must be executable to be browsed

Permissions for files (example)

To see the permissions on a file, do a 'ls –l'

```
attu4:/abc123/dir1$ ls –l
-r--r--r--        1 dir1  year_name  17375  Apr 26 2000      rgb.txt
-rw-r--r--        1 dir1  year_name  17375  Apr  5 02:57  set10.csv
drwxr-xr--   1 dir1  year_name   1024   Jan 19 19:39      tests
```

Changing Permissions

chmod (ugoa)(+-=)(rxw) <filename>

e.g.

chmod u+w rgb.txt	allow user to write (change)
chmod go-w rgb.txt	block group & others from writing
chmod a-x rgb.txt	block all from executing

File access modes / permissions

chmod – change modes/access permissions, else denied access

permits the file owner or root

- to set or clear : access permissions (or modes)
- for groups : using alphabetic (letter) or binary (bit) codes

Groups

a – all groups below
u – user
g – group
o – others

+	set
-	Clear
=	sets specified permissions within specified user groups and clears all others

Executable bit on directory allows it to be searched

Permissions

r – read
w – write
x - execute
Lots others

chmod ug+x myscript	d ? ? x ? ? x ? ? ?
chmod 751 myscripts	d r w x r – x - - x
chmod a-x badscripts	d ? ? - ? ? - ? ? -
chmod 644 badscript	- r w - r - - r - -
chmod a=r badscript	- r - - r - - r - -

? – unchanged from whatever it was

Ownership of files (example)

Changing file Ownership to another user.

```
chown <user>.<group> <filename>
```

e.g. `chown abc123.year_name rgb.txt`

- user abc123 now owns rgb.txt

Changing file Ownership to another group.

```
chgrp <user>.<group> <filename>
```

e.g. `chgrp abc123.year_name rgb.txt`

- year_name group now owns rgb.txt

Note: For security reasons, you cannot change who owns file, unless you are an administrator.
So this is for information only, as useless to us now!

Permissions on directories

- Directory: think of as a file that lists all the files it contains.
- They also belong to a user and a group
- They have the same 3 sets of permissions
- For directories, this is what the permissions mean
  - Read
    - You can read the list of files
    - (eg. You can use “ls” & “find” etc. with directory contents:)
  - Write
    - You can change the list of files
    - (eg. You can add or remove files from the directory)
  - Executable
    - the directory lets the operating system “find” the file. This means, you have access to the file.
    - All directories generally have execute permission.

Versions of Linux or more strictly bash may change behaviour.

e.g. bash v4 to 5,

Miscellaneous extras.

touch

- Update the access and modification times of a file
- Syntax: *touch [-c] [-f] filename ...*
  - *-c* - Do not create *filename* if it doesn't exist,
    (the default is to create *filename)*
  - *-f* - Attempt to force the update in spite of read/write permissions associated with *filename*
- Why would you want to do this?
  - This ~~is~~ was particularly useful in software development, e.g. to update file timestamp, fooling the system into thinking it is updated, thus ensuring it is included in latest build…etc,

conCATenate file(s) - cat

- Isn't it wonderful how much information we can ignore, unless needed!?
- Concatenate files to standard output
- Syntax: *cat [-benstuv] [filename...]*
  - *b* - number all lines except blanks
  - *e* - display non-printing characters and $ at end-of-line
  - *n* - number all lines
  - *s* - substitute a single blank line for multiple adjacent blank lines
  - *t* - display non-printing and tab characters
  - *u* – unbuffered – to ensure output is immediately available as to
    - Stdout – output seen so user knows program is running
    - Or in FIFO pipe so next process can proceed

      Buffering results from blocking for efficient transfers to disk/ network and is unspecified in POSIX for 'cat', so -u ensures unbuffered.

      Unix supports it, Linux ignores it as the default is unbuffered.

      (Main points of this : POSIX not comprehensive; buffering blocks I/O until block full)
  - *v* - display non printing characters as follows:
    - ^X for Control-X
    - M-X for non-ASCII characters with high bit set, where X is the corresponding ASCII character

Neat cat Tips

- *cat filename1 filename2 > filename3*
  - creates a new file, *filename3* that consists of
    - the contents of *filename1*
    - followed by the contents of *filename2*
- Cat filename1
  - displays the contents of filename1 on std. out - screen
- *cat > filename1*
  - creates a new file named filename1 whose contents are whatever you type on the keyboard

```
urmachine%cat > my_file
The cat in the hat
smiled back at me.
^d   (end-of-file character)
urmachine%
```

Head – old Unix versions

- Display the first few lines of a specified file
- Syntax: *head [-n] [filename...]*
  - *-n* - number of lines to display, default is 10
  - *filename...* - list of filenames to display
- When more than one filename is specified, the start of each files listing displays

==>filename<==

```
$ head a b c
==> a <==
==> b <==
==> c <==
```

(just the names as the files are empty !!)

Tail – old Unix version

- Displays the last part of a file
- Syntax: *tail +|-number [lbc] [f] [filename]*

  or: *tail +|-number [l] [rf] [filename]*
  - *+number* - begins copying at distance *number* from beginning of file, if *number* isn'y given, defaults to 10
  - *-number* - begins from end of file
  - *l* - *number* is in units of lines
  - *b* - units are blocks
  - *c* - units are characters
  - *r* - print in reverse order (not in Linux now)
  - f - if input is not a pipe, do not terminate after end of file has been copied but loop. This is useful to monitor a file being written by another process…
  - In general for buffers – which buffer different I/O speeds for read/write from files
  - (normally for pipes, but all man output is reading from a file to file (screen) buffering sorted)
    - data loss problems if writer overtakes reader – standard file buffer issue!
    - Possible answers – but foolproof answer involves process synchronisation
      - pause writer – can also lose data…
      - bigger buffer...faster reader

Head – wrecking options - Linux

```
NAME
      head - output the first part of files
SYNOPSIS
      head [OPTION]... [FILE]...
DESCRIPTION
      Print the first 10 lines of each FILE to standard output.  With more than one FILE, precede each with a
      header giving the file name.
      With no FILE, or when FILE is -, read standard input.
      Mandatory arguments to long options are mandatory for short options too.
      -c, --bytes=[-]NUM          print the first NUM bytes of each file;
                                  with the leading '-', print all but the last  NUM  bytes
            of each file
      -n, --lines=[-]NUM          print  the first NUM lines instead of the first 10;
                                  with the leading '-', print all but the last NUM lines of each file
      -q, --quiet, --silent       never print headers giving file names
      -v, --verbose               always print headers giving file names
      -z, --zero-terminated       line delimiter is NUL, not newline
      --help                      display this help and exit
      --version                   output version information and exit

      NUM may have a multiplier suffix: b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000,
      G 1024*1024*1024, and so on for T, P, E, Z, Y.  (Tera, Peta, Exa, Zetta, Yotta )
```

Disk space administration

These commands can be used to determine or identify files/directories with substantial disk use

- – either intrinsically with options etc.

  (which vary with bash Linux/BSD versions)
- – Or additionally with pipes using sort and grep,

- du – displays disk block usage statistics
- df – displays info on disk free..

Using grep and sort, simply with list directory (ls) can select files of a specific type and sort them by size / modification date, so you can decide which ones to archive or delete.

Check manual (man) for more information and usage examples.

ls -1 | grep 'whatever' | sort –(field no e.g. date or size)

To list the top 10 add | head -10 (& pipe to rm… but trickier/later…)

Backups – brief outline, more later.

Various Options

- Entire filesystems / partitions
  - dump/restore
  - dd – disk duplicate (bit by bit – including DRM !!!)
- Selective:
  - tar - tape archive, simple, stable, ubiquitous & updated :
    - ***Still used for file distribution, before internet, cloud & GIT,***
  - cpio – great but modifies file (create & access) times
  - rsync – (remote sync of files)
    - incremental copy (only the changes)
      - Saves transmission time on low bandwidth link
      - But wastes time checking update modification times since last copy

Tar backup – assuming previous setup

Archiving entire filesystem from root

```
$ cd /
$ sudo tar –cf /dev/st0
```

*//dev/st0 - refers to device **s**treaming **t**ape **0***

- The tar command then
  - creates an archive(**c**) (x for extraction)
    - actually appends it to existing one if present, creates otherwise
  - on the argument to file **f /dev/st0** (**f**) which could be any file, on the net.
- To (de)compress the archive, use **j** to call bzip2:

  ```
  $ sudo tar –cjf /dev/st0
  ```
- To create a backup of a directory mydir

  ```
  $ tar cvf /dev/st0 mydir
  ```
- To make a verbose archive of everything in current directory beginning with an a

  ```
  tar cvf /dev/st0 a*
  ```
- To **extract** just replace the c (create) with x (extract); otherwise identical.

Selective restoring with tar …

- Usually, the entire archive is restored to the original state; all files etc.
- But, any selected file can be restored alone, by giving it's original pathname as used in creating the archive.
- A toc (table of contents) can be recreated from the archive to help locate the original pathname

  ```
  tar tf /dev/st0 > tocfile
  ```
- Which you can browse
- Or grep to find the selected file's pathname.

  ```
  grep myfile tocfile
  ```
  - Or be really concise… (will only work on archival tape with a rewind option)

    ```
    tar xvf /dev/st0 `tar tf /dev/st0 | grep ’myfile'`
    ```
  - but give it time to locate the file, before proceeding with restoration

    ```
    tar xvf /dev/st0 `tar tf /dev/st0 | grep ’myfile' ; sleep 60`
    ```

NB the backticks ` ` actually cause the command within to run, returning the file, if found in the archive table of contents.

Processes – ps

Several *nix flavour variants BSD etc… but all basically the same,

use **man ps | less** for further info on specifics

- ps aux
  - (**a**ll, **u**ser, **x** (without) being tied to a terminal)
- ps axu | sort –rk 3 | head
  - can pipe to sort and sort by a key field
    - in this case k = 3, which for axu flags is the cpu use, so the list is sorted by cpu use
    - r flag with sort does a reverse sort, so descending cpu use, so that the cpu hogs are at the top of the list
  - then piped to head which outputs the ‘TOP 10’

Alternative : top command

- top (could be piped to **nice** command to modify priority.. and be nice to others!)
  - dynamically updated every few secs
  - displays processes in order of decreasing cpu use
  - terminated by ctrl+c (often indicated by ^C or occasionally ^+C)

Note that the stats on virtual systems are virtually misleading!

- running on one of many programs running on host machine

+ extra levels of indirection and abstraction, like any human admin system!?

If Ctrl+c doesn't interrupt – then kill the process

Hope to do much more later, but for now

- occasionally a process
  - Hangs - in an infinite loop, or blocked waiting for another
  - or was stopped and never restarted, even a simple listing
    - And you get a ‘stopped process’ alert on exit
- And needs to be stopped or killed
  - Occasionally even a terminal process hangs,
    - Log into another terminal process and kill it!
    - or shut window & go, leaving zombies & work for sys admin !

```
user@csgate@csgate:~$ ps
  PID TTY          TIME CMD
12816 pts/8    00:00:00 bash
12831 pts/8    00:00:00 ps
user@csgate@csgate:~$ bash
You have used  …
user@csgate@csgate:~$ ps
  PID TTY          TIME CMD
12816 pts/8    00:00:00 bash
12832 pts/8    00:00:00 bash
12845 pts/8    00:00:00 ps
```

Process killed by ID # here,
Can also be done by name pkill
More later

```
 user@csgate@csgate:~$ kill -9 12832
Killed
user@csgate@csgate:~$ ps
  PID TTY          TIME CMD
12816 pts/8    00:00:00 bash
12868 pts/8    00:00:00 ps
user@csgate@csgate:~$
```

If Ctrl+c doesn't interrupt – then kill the process

```
Ignore highlighted: just comments
Just after login check processes:
user1@csgate:~$ ps
  PID TTY          TIME CMD
18635 pts/2    00:00:00 bash
18655 pts/2    00:00:00 ps
Create a file man.text from
manual entry on itself!
user1@csgate:~$ man man
>man.text
Display using less
user1@csgate:~$ less man.text
Too long, didn't read: stop ^z
[1]+  Stopped                 less
man.text
Try to exit
user1@csgate:~$ exit
But get alert about stopped jobs
exit
There are stopped jobs.
So check for self and/or all!?
```

```
ps -S   …. shows state, T for stopped!
user1@csgate:~$ ps -S
  PID TTY      STAT   TIME COMMAND
17414 pts/9    Ss     0:00 -bash
18634 pts/9    S+     0:00 script procs
18635 pts/2    Ss     0:00 bash -i
18749 pts/2    T      0:00 less man.text
18764 pts/2    R+     0:00 ps -S
So pkill it by process name, not kill id
user1@csgate:~$ pkill -9 less
[1]+  Killed                  less man.text
user1@csgate:~$ ps -S
  PID TTY      STAT   TIME COMMAND
17414 pts/9    Ss     0:00 -bash
18634 pts/9    S+     0:00 script procs
18635 pts/2    Ss     0:00 bash -i
18805 pts/2    R+     0:00 ps -S
Now, no stopped process to block exit
user1@csgate:~$ exit
logout
Connection to csgate.ucc.ie closed.
```

Some super admin commands mostly for superuser.

Great for system monitoring for analysis & resolution of problems & bottlenecks – again, check for more info using man, books etc.

May not generally be installed or available…

General

- **vmstat**
  - although it seems specific to virtual memory, (not machine) it is a generally useful command which gives a general widely useful overview

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0     84 2737840 344060 4699420    0    0     3     3    3    0  1  1 98  0
```

- **mpstat** – multiprocessor version of vmstat

Virtual Memory (using disk as RAM memory extension):- **swapon, swapinfo**

Disk I/O :- **iostat**

Disk Performance :-

- **xdd** (examine disk devices)
- **sar** (system activity recorder/monitor) – similar to vmstat, but logs over time (default is 10 minute intervals since midnight, so providing historical data)

Review : basic file commands

• Where am I in the filesystem	$pwd	
• What's here	$ls	
• To move around the filesystem	$cd	
• Make my own pad / directory	$mkdir	
• And fill with a few files	$nano, vim	
– Even	$cat >newfile,	
– List the file	$cat newfile	
– Or just a bit of it	$head newfile $tail newfile	
• Rename or move them around	$mv	
• Even duplicate them	$cp	
• Or dump them	$rm, rmdir	
• Where can I find out more about		
– a command	$man	
– Any topic	$apropos	$man -k

Directory Navigation - review

- Directory Paths
  - unlike MS-DOS, UNIX file systems use forward slashes
  - Eg. /user/csdipact2013/your_id/   (UNIX)

    \mydocuments\myword\ (MS)
- Change Directory: cd *directory_path*
  - Go up to parent directory: cd ..
  - Go to home directory: just cd alone, or if you insist cd ~
- Useful Commands
  - pwd: returns "present/print working directory"
  - ls: "list contents of directory"
    - ls   : tab delimited list
    - ls –l   : one entry per line, long format with file attributes
    - ls –a   : list all files, including hidden (system) ones..
    - ls –la   : lists all files incl. hidden files + shows file attributes