THE UNIVERSITY OF WESTERN AUSTRALIA

Files

Lecture 3

Michael J. Wise

Files Live on Paths

- A file can be found by listing the *path* from the root (/) to the file, for example

`/staff/michaelw/a.out`

Or

`~michaelw/a.out`

Or if I'm already in ~michaelw

`./a.out`

- Emphasis on shared resources, c.f. Android, IOS, though both based on Unix

File Permissions

- Unix controls access to files using 3 sorts of access permissions:
  - *r (read access)*
  - *w (write/modify access)*
  - *x (execute permission)*

granted to 3 different sets of users.

  - *u (user, i.e. file owner)*
  - *g (group)*
  - *o (other, i.e. everyone else)*

- This can be represented a string of 9 letters, .e.g.

  `rw-|---|---` where the first 3 represent (rwx) user/owner, the next 3 (rwx) group and the last 3 (rwx) other

File Permissions - Directories

- Directory preceded by a `d`, e.g. `drwxr-xr-x`
- `d` added by `mkdir`
- Permissions have slightly different meaning for directories:
  - *r user/group/other can read list of files in the directory*
  - *w user/group/other can write/update a file in the directory*
  - *x user/group/other can pass through the directory (to get to a file or subdirectory if name already known).*

Setting File Permissions

- `chmod` *<octal mode> <files>*
- `chmod` *<symbolic mode> <files>*

chmod is used to change the permissions of one or more files that you own. There are two ways of specifying the mode of the file(s): as an octal number or symbolically.

- Octal Number:

Each group of three permissions (owner, group, others) viewed as three bits of octal number:

100 Read permission set
010 Write permission set
001 Execute permission set

E.g `chmod 644 Alice_in_Wonderland.txt`

Setting File Permissions

- `chmod` <*symbolic mode*> <*files*>

The file permissions can also be specified symbolically

E.g. `chmod go+r Alice_in_Wonderland.txt`

May need multiple chmod calls:

```
chmod ugo+rwx myapp
chmod go-rw myapp
```

That is, the user can read, write or execute the program, but anyone else can just execute it.

Input Output

- UNIX was revolutionary in its day because it treats every file as a file of characters; it's up to the programs that use a file to make sense of it.
- By default, input to a command is typically via *Standard Input* (file descriptor 0). You read from *stdin*. Stdin is by default the keyboard
- Output from a command is by default via *Standard output, stdout* (file descriptor 1) which goes to the screen.
  - *(programs can also write directly to files!)*
- Standard Error output (*stderr*) (file descriptor 2) also by default goes to the screen. Kept separate for error messages rather than expected output

Redirection

- Each of stdin, stdout, stderr can be redirected from/to a file.

< Redirect standard input

> Redirect standard output

>> Redirect and append standard output to the named file

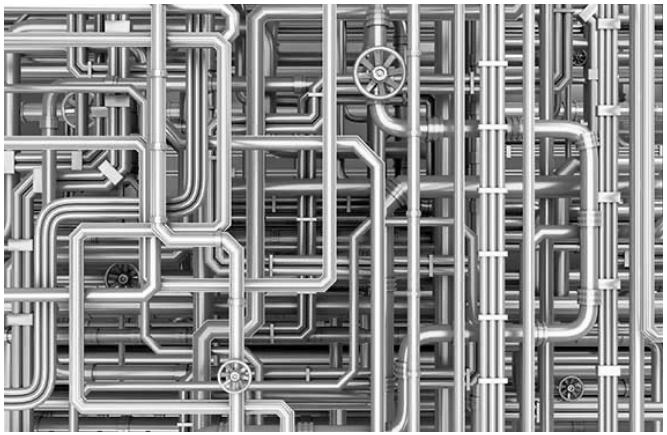
2> Redirect standard error output

For example:

```
date > _a    # current date and time
wc < _a # count number of lines,words,letters
wc _a
```

Piping

- Redirection takes data from a file to a process (stdin) or from a process to a file (stdout)
- If you want to connect stdout from one process to stdin of another process you can use a pipe `|` (bar typically found above \ )



vistaprojects.com

- For example, `date | wc`
- Unfortunately, | only applies to stdout, not stderr
- But, `my_prog 2> errors | analyse`

DIY scripts

- Executing single commands from the command-line is fine for small things, but requires repeated typing (and getting the typing right 😡)
- To save time (and precious sanity) better to create an executable file containing a script, i.e. a shell program,
- You need to use a **plain text** editor (i.e. it writes simple ASCII text without markups, unlike, e.g. textedit (RTF). Most common one from within Unix is `vi`. (See Resources for `vi` tutorial; otherwise `man vi`.)

DIY scripts

- First line of the file says how the file is to be interpreted:

```
#!/usr/bin/env bash
```

Or

```
#!/bin/bash
```

- After saving (don't forget!), the file has to be made executable:

`chmod u+x` *<file>*

- Commands then go one after the other.
  - *If more than one command on a line, separated by a ;*

Demo

- Among other things, the Unix command date prints out the date and time right now

```
% date
```

Thu Jan 27 10:54:09 AWST 2022

`TimeNow`

- What if I just want the time, or better still, the hours and minutes (who really needs the number of seconds?), so in this case:

```
% TimeNow
```

10:54:09

or

10:54

There are several ways to do this. Perhaps use `cut`?

Processing Command-line Arguments

- Arguments are the items in the command line, `ls -l`, has two
- Inside the command (if it's in Bash), arguments numbered from 0 (the command itself), preceded by `$`, e.g. `$0`, `$1`, etc, so `$0` is `ls`, `$1` is `-l`
- The number of arguments (excluding the program name) is reported as `$#`

Demo L0 revisited

Revisiting the problem of listing words in a text file in order of descending occurrence, the existing single liner requires typing out each time. Need to create a script count_occurences which can be passed any text file.