THE UNIVERSITY OF WESTERN AUSTRALIA

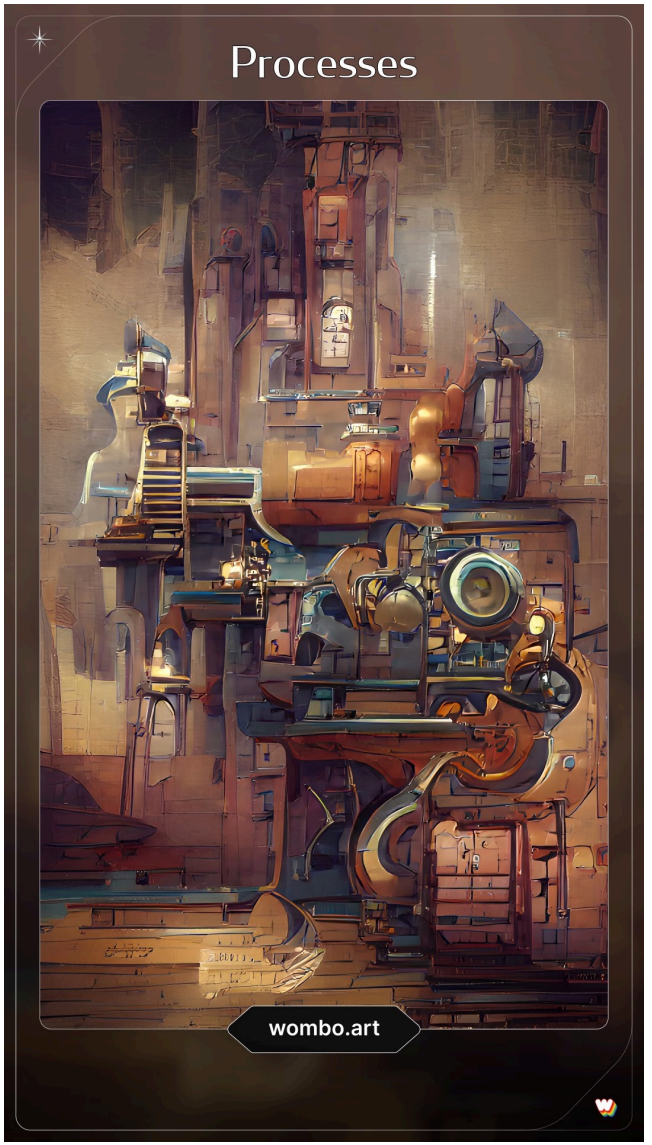
Processes

Lecture 4

Michael J. Wise

Processes

- A process is program in execution
- Each call to bash or opening a new window creates a separate processes.
- Windows can be created by the window manager (via menu)
- The process that controls the window is called the foreground process. That is, it can take input from the keyboard (if the window is active) and write to the window.



Processes

- You can run more than one process in a window by making the other processes background processes. (simply append `&` to end of command).
- Note: Background processes can neither accept keyboard input nor write to a window, so if either is required it must come from/go to a file.

E.g. `% gzip large_file.txt > trace 2> errs &`

Which then allows you to get on with other things will the gzip file compression is happening the background.

BTW. Unix `gzip` compresses files (`gunzip` extracts the compressed files)

Some Process Oriented Commands

`ps` *<options>*

- `ps` prints out information about processes you have running. If no options are specified, only the processes for the currently active window are reported.
- `-l` Provides a long, listing containing more information.
- `-f` provides more information about the command (which options were used)
- The most important pieces of information are the names of the processes and their process IDs or simply PIDs, e.g.

```
3638 ttys002    0:00.06 python
8808 ttys003    0:00.03 bash
```

Some Process Oriented Commands

- `top`

Lists a range of statistics about all the processes, and then lists the processes one by one in order of their CPU use. Useful for finding that process that has gone off the rails.

  - *lots of processes are asleep*

- `kill` *<options> <PIDs>*

A signal is sent to each of the listed processes. The options - generally not required - specify which signal is to be sent; `kill -9` is stronger medicine

Interlude: More Useful Commands - cut

- There is a HUGE number of commands/analyses available for Unix system. Lots to choose from. This is selection from those I most frequently use.

`cut` *[* `-d` *<delim>]* `-f` *<list> <files>*

Given an ASCII file structured as a number of fields separated by some delimiter, e.g. <TAB>, cut returns the fields indicated.

`-f` The comma separated list following `-f` specifies the fields that are to remain

`-d` Specifies an alternate delimiter

For example:

```
cut -d , -f 1,5-7 datafile.csv
```

Interlude: More Useful Commands - paste

`paste` *[`-d` <sep>]* *<files>*

Steps through one or more files in parallel. The output is a line containing all the first lines (separated by <*sep*>, say the Tab char), followed by a line containing all the second lines, and so on.

```
% cat file1
Twinkle Twinkle
How I wonder
Up above the world
Like a tea
```

```
% cat file2
little bat
what you're at!
you fly
tray in the sky
```

What does `paste -d" " file1 file2` do?

Interlude: More Useful Commands - tr

`tr` *<options> <string1> <string2>*

- Translate characters in the first string (say lower case letters) to characters in the second string (UC letters). Input from stdin.
- Most useful options are:

`-C` Complement characters in the first string

`-s` Squeeze multiple contiguous occurrences of string1 characters

```
tr '[A-Z]' '[a-z]' < textfile
tr -s ' ' '#' < textfile
tr '\015' '\012' < windows_textfile
```

The mapping of characters to codes: `man ascii`

Interlude: More Useful Commands - comm

`comm` *[<output options>] <file1> <file2>*

The two **sorted** files are compared, and three column output is produced. Column 1 contains those lines only found in <file1>, Column 2 contains lines only in <file2>, Column 3 contains lines common to both files.

The output options suppress the named columns,e.g.

`-1` Suppress column unique to <file1>

`-23` Suppress column unique to <file2> and column of common entries

Interlude: More Useful Commands - uniq

`uniq` *<file>*

Compresses adjacent lines that are repeated in a file to a single copy (i.e. duplicates removed). The

`-c` option outputs the single lines together with the counts of the number of copies of those line.

Interlude: More Useful Commands - sort

`sort` *[options] <files>*

This powerful sort utility takes the named files (or stdin) and sends the sorted output to stdout. It has MANY options. Some are:

-u Removes duplicated entries following the sort (i.e. as if uniq had been applied)

-t The character following the -t flag is used as the field separator (the default separator is <TAB>.

Sort

`-k` *<start>[<type>][,<end>[<type>]*

Instead of sorting from the first field, sort from the <start> field. If <end> is specified, only the fields from <start> to <end> are sorted.

For example:

```
ps| sort -k 4 # list processes alphabetically
```

Demo L0 (again)

- In this revisiting of the L0 demo, I want to be able to have reported the list of the top N words (in order of descending order of occurrence). In other words, the user specifies both the file and the number of top hits to be reported.

Demo

- Real example. In a bit of a version control problem, I found I had two directories with over-lapping sets of files; some with the same name, some different. I needed to write a small script, `dir_diffs`, which, given two directories, lists the files that are unique to the first directory and the files that are unique to the second directory.