Introduction to Bash video lecture

03 Issuing commands

Commands

- Bash comes together with tons of small programs intended to be used in the command line
- Such small programs are commonly referred to as *commands*
- Commands can accept *arguments* and *flags/options*
- Often a command doesn't make sense without arguments or flags

```
$ cp
cp: missing file operand
Try 'cp --help' for more information.
```

- Here are some basic commands:

```
echo ls cd cp mv pwd head tail cat file
```

Options/flags

- You can instruct most commands *how to* do their job
- Options/flags typically start with a dash (short options):
  `ls -a`
- or two dashes (long options):
  `ls --all`
- The above tells ls to list all files, to include even files whose filename start with a dot (hidden by default)
- `ls -l` (long listing) tells ls how to format its output (to include all sorts of information)
- `ls -1` (one column) tills ls how to format its output - one file per line

Arguments

- You can instruct many commands *what* to do using arguments
- Arguments come after the command: `ls /tmp` (list the `/tmp` directory)
- Many commands don’t make sense without an argument and require arguments in order to work

```
cp ../file.txt Documents/

cd Documents/

ls /tmp/

mv old_filname new_filename

mpg321 Warlords.mp3
```

Commands that require arguments

- Often it is very intuitive when a command needs arguments
- Much like our natural language
  - put!
  - fetch!
  - give!
  - take!
  - punch!
  - kill!
- For instance the copy command, `cp`, needs to know what to copy and where (same with the move command, `mv`)

Running a command occupies the shell

- When you run a command, it is by default run *in the foreground*
- That means you cannot use the shell until the command has finished
- Most commands are very quick so you won't think about this
- Other commands are meant to run for a long time (a browser, an editor, an image processing application, a web server)
- If you want to start a command *in the background* you put an ampersand at the end of the command line:

```
$ gedit welcome.sh  &                # start the editor in bg
[1] 18663                            # its process id was 18663
$                                    # its job number was 1
```

Basic job control

- Start process in background: put & at the end of command line
- Bring process to foreground:
  `fg or fg %N` (where N is the job number)
- Bring process to background
  `bg or bg %N`
- Stop (pause, not end) foreground job:
  Ctrl-Z
- List current jobs:
  `jobs`
- Ctrl-C terminates (abruptly) a foreground job

Jobs

- A job is a group of one or more processes
- A process is a program running and how the operating system handles running programs
- Having a job with a group of commands allows you to control the group as if it were one process (group of commands is typically a pipeline that we will learn more about later on)
- When the shell terminates (e.g. you logout or close the terminal), any jobs still running will be terminated
- Often you are warned about this when trying to logout, e.g.
  “There are unfinished jobs”

A common use for Ctrl-Z, fg

- Some applications occupy the whole terminal window (like editors etc)
- Often, you want to go back to the shell to do some work, but don’t want to stop the editor
- Then you pause the editor using Ctrl-Z which drops you back to the shell
- when you want to continue editing, you type fg to put the editor in the foreground again (and you will continue exactly where you were)
- Imagine that you are writing a script and want to try how well it is doing
- You pause your editor and run the script and discover a bug
- You fg the editor, fix the bug, then Ctrl-Z again to run the script etc
- This is a very common workflow for developers working in Bash

Process

- When you start a program, the operating system loads the program instructions to RAM and schedules it for execution
- The OS creates a *process* for each running program
- A process has the code and some contextual information (who started the program and when, what is the current directory - where was it started - etc)
- Each process has an ID called pid
- You can list processes with the `ps` command
- Processes means that you can run a program in more than one instance
- You can start most programs more than once in parallel (e.g. two Chrome browsers running side-by-side, two terminals etc)

Examples of ps

```
$ ps
  PID TTY          TIME CMD
17200 pts/33   00:00:01 bash
19271 pts/33   00:00:00 ps
$ gedit welcome.sh  &
[1] 19288
$ ps
  PID TTY          TIME CMD
17200 pts/33   00:00:01 bash
19288 pts/33   00:00:00 gedit
19300 pts/33   00:00:00 ps
```

Summary

- You run commands from the command line (the shell)
- Commands often need extra information
  - flags/options tell them *how* to do their job
  - arguments tell them *what* to do
- You run commands (and programs) in the foreground, occupying the shell until the command exits or is stopped (paused)
- You can pause a program using Ctrl-Z
- You can start a program in the background using &
- Running programs are called *jobs* (a job can actually be a group)
  - To the operating system, each program is run in a *process*
- You can control jobs with fg and bg