What is a shell?

- `login` is a program that logs users in to a computer.
- When it logs you in, `login` checks `/etc/passwd` for your shell.
- After it authenticates you, it runs whatever your shell happens to be.

What is a shell?

- `login` is a program that logs users in to a computer.
- When it logs you in, `login` checks `/etc/passwd` for your shell.
- After it authenticates you, it runs whatever your shell happens to be.
- Shells give you a way to run programs and view their output.
- They also usually include some built-in commands.
- Shells use variables to track information about commands and the system environment.

What is a shell?

- `login` is a program that logs users in to a computer.
- When it logs you in, `login` checks `/etc/passwd` for your shell.
- After it authenticates you, it runs whatever your shell happens to be.
- Shells give you a way to run programs and view their output.
- They also usually include some built-in commands.
- Shells use variables to track information about commands and the system environment.
- The standard interactive shell is `bash`.
- There are others, though! `zsh` and `fish` are both popular.

Navigating the filesystem

- `ls` **L**i**s**t files. You can give it a directory to list.
  - `-l` Display the output in a detailed **l**ist, one line per file.
  - `-h` Display file sizes in a **h**uman-readable format.
  - `-a` Display **a**ll files, including hidden ones.

Navigating the filesystem

- `ls` **L**i**s**t files. You can give it a directory to list.
  - `-l` Display the output in a detailed **l**ist, one line per file.
  - `-h` Display file sizes in a **h**uman-readable format.
  - `-a` Display **a**ll files, including hidden ones.
- `pwd` **P**rint **w**orking **d**irectory.

Navigating the filesystem

- `ls` **L**ist files. You can give it a directory to list.
  - `-l` Display the output in a detailed **l**ist, one line per file.
  - `-h` Display file sizes in a **h**uman-readable format.
  - `-a` Display **a**ll files, including hidden ones.
- `pwd` **P**rint **w**orking **d**irectory.
- `cd DIRECTORY` **C**hange **d**irectory.
  - `cd` without a directory takes you `$HOME`.
  - `cd -` takes you to the previous directory you were in.

Rearranging files

- `mv SOURCE DESTINATION` **M**o**v**e (or rename) files.
  - `-i` **I**nteractively ask you before overwriting files.
  - `-n` **N**ever overwrite files.

Rearranging files

- `mv SOURCE DESTINATION` **Mov**e (or rename) files.
  - `-i` **I**nteractively ask you before overwriting files.
  - `-n` **N**ever overwrite files.
- `cp SOURCE DESTINATION` **Cop**y files.
  - `-r` **R**ecursively copy directories, which is what you want to do.

Rearranging files

- `mv SOURCE DESTINATION` **Mov**e (or rename) files.
  - `-i` **I**nteractively ask you before overwriting files.
  - `-n` **N**ever overwrite files.
- `cp SOURCE DESTINATION` **Cop**y files.
  - `-r` **R**ecursively copy directories, which is what you want to do.
- `rm FILE` **Rem**ove one or more files.
  - `-f` **F**orcibly remove nonexistent files.
- `mkdir DIRECTORY` **Mak**es a **dir**ectory.
  - `-p` Makes every missing directory in the given **p**ath

Looking at files

- `cat [FILE]` Print out file contents.

Looking at files

- `cat [FILE]` Print out file contents.
- `less [FILE]` Paginate files or STDIN.

Looking at files

- `cat [FILE]` Print out file contents.
- `less [FILE]` Paginate files or STDIN.
- `head [FILE]` Print lines from the top of a file or STDIN.
- `tail [FILE]` Print lines from the end of a file or STDIN.
  - `-n LINES` Print LINES lines instead of 10.
  - `-f` Print new lines as they are appended (`tail` only).

Looking at files

- `cat [FILE]` Print out file contents.
- `less [FILE]` Paginate files or STDIN.
- `head [FILE]` Print lines from the top of a file or STDIN.
- `tail [FILE]` Print lines from the end of a file or STDIN.
  - `-n LINES` Print LINES lines instead of 10.
  - `-f` Print new lines as they are appended (`tail` only).
- `sort [FILE]` **Sort**s files or STDIN.
  - `-u` Only prints one of each matching line (**u**nique).
  - Often paired with `uniq` for similar effect.

Looking at files

- `cat [FILE]` Print out file contents.
- `less [FILE]` Paginate files or STDIN.
- `head [FILE]` Print lines from the top of a file or STDIN.
- `tail [FILE]` Print lines from the end of a file or STDIN.
  - `-n LINES` Print LINES lines instead of 10.
  - `-f` Print new lines as they are appended (`tail` only).
- `sort [FILE]` **Sort**s files or STDIN.
  - `-u` Only prints one of each matching line (**u**nique).
  - Often paired with `uniq` for similar effect.
- `diff FILE1 FILE2` Shows **diff**erences between files.
  - **a/d/c** Added/Deleted/Changed.

Redirecting IO

- Each program has three default IO streams:
  - STDIN: input, by default from the keyboard (`cin`).
  - STDOUT: output, by default to the screen (`cout`).
  - STDERR: output, by default to the screen (`cerr`).

Redirecting IO

- Each program has three default IO streams:
  - STDIN: input, by default from the keyboard (`cin`).
  - STDOUT: output, by default to the screen (`cout`).
  - STDERR: output, by default to the screen (`cerr`).
- We can redirect IO to or from files or other programs.
- `cmd1 | cmd2` Pipe STDOUT from `cmd1` into STDIN for `cmd2` .

Redirecting IO

- Each program has three default IO streams:
  - STDIN: input, by default from the keyboard (`cin`).
  - STDOUT: output, by default to the screen (`cout`).
  - STDERR: output, by default to the screen (`cerr`).
- We can redirect IO to or from files or other programs.
- `cmd1 | cmd2` Pipe STDOUT from `cmd1` into STDIN for `cmd2` .
- `cmd < input.txt` Funnel data from `input.txt` to STDIN for `cmd` .
- `cmd > output.txt` Funnel STDOUT from `cmd` into `output.txt` .

STDERR redirection tricks

- `bash` uses `1` and `2` to refer to STDOUT and STDERR.

STDERR redirection tricks

- `bash` uses `1` and `2` to refer to STDOUT and STDERR.
- `cmd 2> err.txt` Funnel STDERR from `cmd` into `err.txt`.
- `cmd 2>&1` Funnel STDERR from `cmd` into STDOUT.

STDERR redirection tricks

- `bash` uses `1` and `2` to refer to STDOUT and STDERR.
- `cmd 2> err.txt` Funnel STDERR from `cmd` into `err.txt`.
- `cmd 2>&1` Funnel STDERR from `cmd` into STDOUT.
- `cmd &> all-output.txt` Funnel all output from `cmd` into `all-output.txt`
- Common usage: `cmd &> /dev/null` dumps all output to the bit bucket.

Environment Variables

- Shells keep track of a lot of information in variables.
- `set` shows all the environment variables set in your shell.
- `env` shows **exported** environment variables (variables that are also set in the environment of programs launched from this shell).

Environment Variables

- Shells keep track of a lot of information in variables.
- `set` shows all the environment variables set in your shell.
- `env` shows **exported** environment variables (variables that are also set in the environment of programs launched from this shell).
- `VAR="value"` sets the value of `$VAR`. (No spaces around the `=`!)
- `echo $VAR` prints the value of a variable in the shell.

Environment Variables

- Shells keep track of a lot of information in variables.
- `set` shows all the environment variables set in your shell.
- `env` shows **exported** environment variables (variables that are also set in the environment of programs launched from this shell).
- `VAR="value"` sets the value of `$VAR`. (No spaces around the `=`!)
- `echo $VAR` prints the value of a variable in the shell.
- You can get environment variable values in C++ with getenv()

Useful variables

- `$PATH` Colon-delimited list of directories to look for programs in.

Useful variables

- `$PATH` Colon-delimited list of directories to look for programs in.
- `$EDITOR` Tells which editor you would prefer programs to launch for you.

Useful variables

- `$PATH` Colon-delimited list of directories to look for programs in.
- `$EDITOR` Tells which editor you would prefer programs to launch for you.
- `$PS1` Customize your shell prompt!

Useful variables

- `$PATH` Colon-delimited list of directories to look for programs in.
- `$EDITOR` Tells which editor you would prefer programs to launch for you.
- `$PS1` Customize your shell prompt!
- `~/.bashrc` runs every time you start `bash`, so you can `export` customizations there.

Neat bash tricks

- Tab completion works for files and commands!

Neat bash tricks

- Tab completion works for files and commands!
- History:
  - ↑ / ↓ scroll through history.
  - Ctrl + r searches backwards through history.

Neat bash tricks

- Tab completion works for files and commands!
- History:
  - ↑ / ↓ scroll through history.
  - Ctrl + r searches backwards through history.
- `!!` holds the last command executed.
- `!$` holds the last argument to the last command.

Neat bash tricks

- Tab completion works for files and commands!
- History:
  - ↑ / ↓ scroll through history.
  - Ctrl + r searches backwards through history.
- `!!` holds the last command executed.
- `!$` holds the last argument to the last command.
- `alias sl=ls` runs `ls` when you type `sl`.

Processes

- `ps` **P**rocess li**s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.

Processes

- `ps` **P**rocess li**s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.
- `top` and `htop` give an interactive process listing.

Processes

- `ps` **P**rocess **s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.
- `top` and `htop` give an interactive process listing.
- Job Control:
  - You can start processes in the background by doing `command &`.
  - If you have a command running in the foreground, you can stop it with Ctrl + z.

Processes

- `ps` **P**rocess **l**i**s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.
- `top` and `htop` give an interactive process listing.
- Job Control:
  - You can start processes in the background by doing `command &`.
  - If you have a command running in the foreground, you can stop it with Ctrl + z.
  - `fg` starts the last process in the foreground.
  - `bg` starts the last process in the background.

Processes

- `ps` **P**rocess **s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.
- `top` and `htop` give an interactive process listing.
- Job Control:
  - You can start processes in the background by doing `command &`.
  - If you have a command running in the foreground, you can stop it with Ctrl + z.
  - `fg` starts the last process in the foreground.
  - `bg` starts the last process in the background.
  - `jobs` shows your running jobs.
  - `fg %2` starts job 2 in the foreground.

Processes

- `ps` **P**rocess **s**t.
  - `aux` / `-ef` show lots of information about all processes.
  - `ps` has crazy whack options.
- `top` and `htop` give an interactive process listing.
- Job Control:
  - You can start processes in the background by doing `command &`.
  - If you have a command running in the foreground, you can stop it with Ctrl + z.
  - `fg` starts the last process in the foreground.
  - `bg` starts the last process in the background.
  - `jobs` shows your running jobs.
  - `fg %2` starts job 2 in the foreground.
  - `kill PID` Kills a process. (You can do `kill %1`!)
  - `killall command` Kills every process running `command`.

Last but not least: The Manual!

- `man COMMAND` opens a manual listing for that command.

Last but not least: The Manual!

- `man COMMAND` opens a manual listing for that command.
- `q` quits the manual.

Last but not least: The Manual!

- `man COMMAND` opens a manual listing for that command.
- q quits the manual.
- j / k scroll up and down a line.
- Space scrolls down one page.

Last but not least: The Manual!

- `man COMMAND` opens a manual listing for that command.
- `q` quits the manual.
- `j`/`k` scroll up and down a line.
- `Space` scrolls down one page.
- `/thing` searches for things.
- `n`/`N` go to next/previous search result.

Last but not least: The Manual!

- `man COMMAND` opens a manual listing for that command.
- `q` quits the manual.
- `j`/`k` scroll up and down a line.
- `Space` scrolls down one page.
- `/thing` searches for things.
- `n`/`N` go to next/previous search result.
- `man man` gives you the manual for the manual!