Table of Contents

`bash`

Notes based on https://missing.csail.mit.edu/2020/course-shell/

Bourne again shell

- `man baz`: give manual page for `baz`
- `-h`, `--help`, Windows: `/?`: giving these as arguments typically gives some text on usage and options
- `foo\ baz`: the backslash is used to escape single characters
  - Quoting reference
  - use of `'foo'` inhibits interpretation of a sequence of characters
  - use of `"foo"` inhibits most of the interpretation of a sequence of characters
- **`ctrl`+`l`**: clear terminal and go to the top
- `./`: execute file

Paths and permissions

- `echo $PATH` lists the path environment variable, a list of paths in which the shell looks for a binary you are trying to execute
- `which foo` indicates the absolute path of `foo` that would be executed if you try to run it
- `pwd` gives you the path
- `.` the current directory
- `..` the parent directory
- `ls` lists files in current directory
- ~ expands to the home directory
- `cd -` changes to the previous directory you were in

Long listing

- `ls -l`: long listing of files in directory

As an example:

```
sinkers@DESKTOP-HQ8VENU:~$ ls -l
total 16
-rw-r--r-- 1 root    root      65 Feb  5 17:27 hello_world.c
-rwxrwxrwx 1 sinkers sinkers 8304 Feb  5 17:28 hello_world.out
```

- column 1: file type and permissions
  - first letter: file type
    - `d`: directory
    - `l`: symlink/soft link; pointer to another location in the file system
    - `-`: file
  - following letters gives user permissions:
    - first 3: owner
    - next 3: group
    - last 3: anyone else
  - permission values on a file
    - `r`: read from the file
    - `w`: write to the file
    - `x`: execute the file
    - `-`: do not have permission
  - permission values on a directory
    - `r`: list contents of directory
    - `w`: rename, create, remove files from directory
    - `x`: “search”; are you allowed to enter this directory? to enter a directory you need to have execute permission on all parent directories and directory itself
- column 2: owner
- column 3: group
- column 4: file size [/bytes]
- column 5: last modified date-time
- column 6: file name

Move, copy, create

- `mv old_path new_path`: move file at `old_path` to `new_path`

- `cp`: copy
- `rm`: remove
  - `rm -r`: recursive remove
  - `rmdir`: remove directory if empty
- `mkdir`: make directory

Streams and Redirection

(Simplifying) Programs have two main streams: * input: this is fed to the program; by default it is what you type in the terminal * output: this is what comes out of the program; by default it is printed to the terminal

Shell gives you ways to redirect those streams: * `< foo`: rewire input to preceding program to be contents of `foo` * `> baz`: rewire output of preceding program to baz

- `cat`: prints contents of file
- `cat < hello.txt > hello2.txt`: the effect is to copy hello.txt to hello2.txt. But cat is unaware of the redirection, the shell handles the redirect.
- >>: append
- |: pipe; take output of program to left as input of program to right
- `tail`: prints last n lines of input
  - `tail -n1`: print last line
- `$ ls -l / | tail -n1`: prints last line of long file listing; programs again unaware of redirection

`root` user

- superuser, gets to do whatever they want
- `sudo`: do as su; do as superuser
- #: pound means “run this as root”
- `$` means you are not root!
- `sudo su`: gets you a su shell

In `/sys` is a bunch of parameters for devices on the computer. Say we were in backlight directory:

```
$ echo 500 > brightness
bash: brightness: Permission denied
```

But would if we use sudo ```bash $ sudo echo 500 > brightness bash: brightness: Permission denied ``` Heresudoapplies to the echo command, so when it gets redirected brightness is not executed with the shell running as root *tee: takes input, and writes it to a file, but also to the standard output ```bash $ echo 500 | sudo tee brightness 500 ``` Now **this** works: here output fromechogets piped as input **for**tee, which is run withsudo‘, so it has permission to write to brightness file

Exercises

1. Create a new directory called `missing` under `/tmp`.
2. Look up the `touch` program. The `man` program is your friend.
3. Use `touch` to create a new file called `semester` in `missing`.
4. Write the following into that file, one line at a time: `#!/bin/sh curl --head --silent https://missing.csail.mit.edu` The first line might be tricky to get working. It's helpful to know that `#` starts a comment in Bash, and `!` has a special meaning even within double-quoted (") strings. Bash treats single-quoted strings (') differently: they will do the trick in this case. See the Bash quoting manual page for more information.
5. Try to execute the file. Investigate why it doesn't work with `ls`.
6. Look up the `chmod` program.
7. Use `chmod` to make it possible to run the command `./semester`.
8. Use `|` and `>` to write the "last modified" date output by `semester` into a file called `last-modified.txt` in your home directory.
9. Write a command that reads out your laptop battery's power level or your desktop machine's CPU temperature from `/sys`. Note: if you're a macOS user, your OS doesn't have sysfs, so you can skip this exercise.

Solutions

1. `$ mkdir /tmp/missing`
2. `$ man touch`. Changes file timestamps to current time
3. `touch semester`
4. `bash $ echo '#!/bin/sh'> semester $ echo "curl --head --silent https://csail.mit.edu">> semester`
5. `bash $ ./semester -bash: ./semester: Permission denied $ ls -l total 0 -rw-rw-rw- 1 sinkers sinkers 96 Feb 7 01:06 semester` There are no execute

permissions set for anyone.

6. `$ man chmod`: `chmod` changes file mode bits chmod calculator
7. `bash $ chmod u+x semester $ ls -l total 0 -rwxrw-rw- 1 sinkers sinkers 96 Feb 7 01:06 semester` Now `semester` can be executed
8. `bash $ ./semester | grep Date > ~/last-modified.txt`
9. `bash $ cat /sys/class/power_supply/battery/capacity 98`