Command Line Interface

CS 246

Objectives

- Students will be able to:
  - explain the differences between GUIs and CLIs
  - perform basic tasks using CLIs

Overview

- There are two categories of user interfaces:
  - a **command-line interface** (CLI) is strictly text-based and is controlled by a keyboard
  - a **graphical user interface** (GUI) presents a visual environment that the user can interact with using a mouse, keyboard, touchscreen, stylus, etc., etc.
  - a CLI is typically provided by a program called a shell, running in a terminal

Advantages of Using CLI Tools

- Many developer tools have both a CLI and GUI version
- The former:
    - use fewer system resources
    - may be your only choice on servers
    - can be wired together to great effect
    - may be automated using scripting tools
    - can be easier to use (no hunting for a button/menu item)

Shell Types

- macOS (accessible via Terminal):
  - bash
  - zsh
- Windows (also accessible via Terminal):
  - PowerShell
  - cmd.exe
  - Git Bash (installed with Git)
  - WSL

CLI: A Quick Start

1. Launch Terminal (on either macOS or Windows)
2. **pwd** (prints the working directory, where you are)
3. **mkdir** Programs (makes a new directory called Programs)
4. **cd** Programs (changes directory to Programs)
5. **nano** hello.c

```
name = input("What is your name? ")
print("Hello, ", name)
```

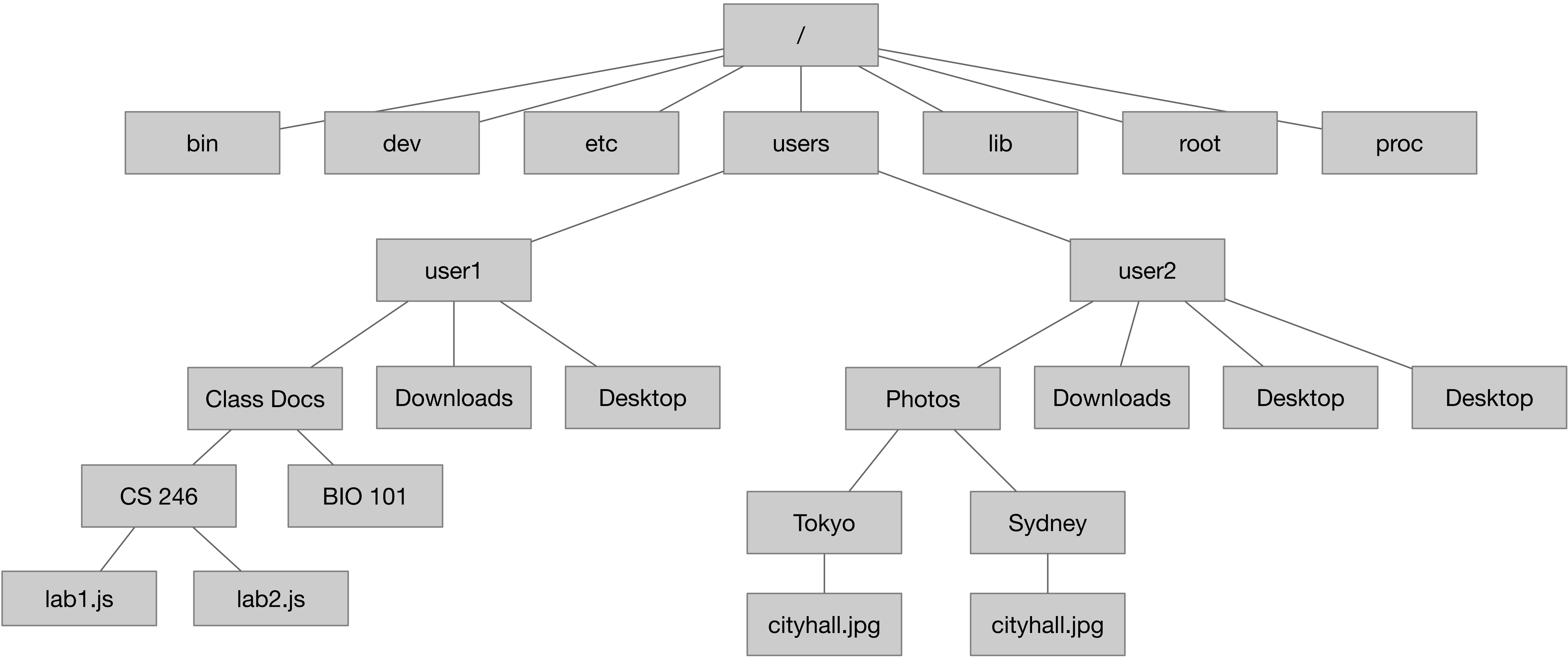
6. **ls** (to see what files you have created)
7. **ls -l** (to see the details of what you have created: -l provides a long listing)
8. **cd** (changes directory to your home directory)
9. **man ls** (to display the manual page for the ls command)

Shell Commands

- Commands in the **shell** (the program that is running when you are at the command line) generally have three parts:
  - **command options file**
  - options are preceded by a -
  - multiple options can be specified separately or combined
- Example:

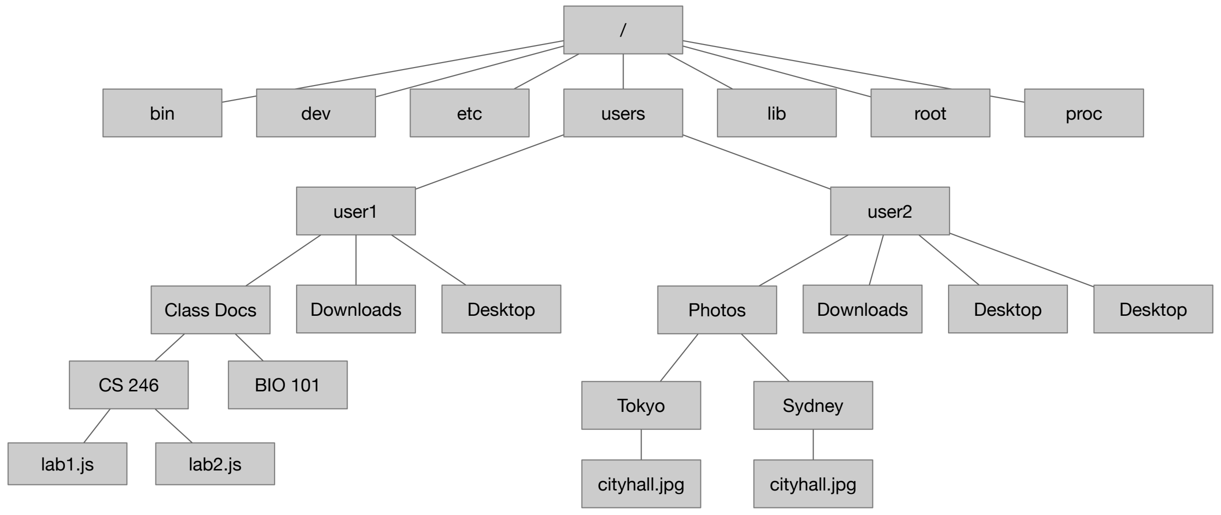
```
ls -a -l /home/pi   # a long listing, including invisible files
ls -al /home/pi     # does the same thing (-la would also work)
```

A Typical File System



Absolute vs. Relative Paths

- Paths that start with a **/** are absolute paths, e.g.,
  - e.g., /users/user1/Class Docs CS246/lab1.js
- Paths that do **not** start with a **/** are relative paths, relative to your working directory*
  - e.g., if your current directory is user2, Photos/Tokyo/cityhall.jpg references city hall.jpg, and Photos/Sydney/cityhall.jpg references a different file
- Relative paths are only valid if they can be "seen" from your current location.
- e.g., if you were in /users, then Photos/Tokyo/cityhall.jpg would be invalid, as Photos is not a directory in /users.



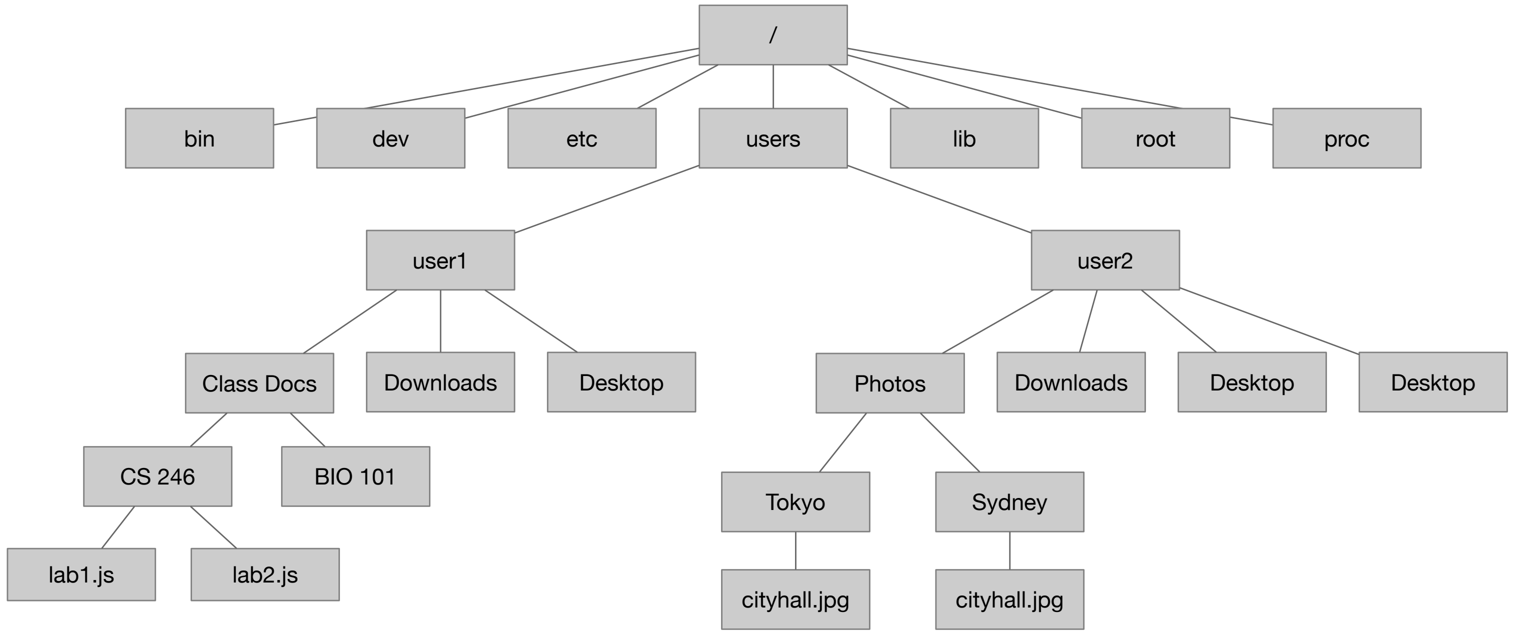
*You are always *somewhere* in the file system. That location is your *working* directory. When you first log in, it is your home directory, i.e., /students/you

Shortcuts

. = current directory

.. = parent directory

~ = your home directory



e.g., if you are in /users/user1/Class Docs

. = /users/user1/Class Docs

.. = /users/user1

~ = /users/user1 # really good to know ~

../.. = /users

How would you get to Tokyo?

Where Am I?

- **pwd** - print working directory

Seeing What's Out There: ls

- **ls** # lists contents of the current directory
- **ls** *directory* # lists contents of *directory*
  - e.g., ls /home/pi/Documents, or ls
- **ls -l** *directory* # long listing (date, file size) of *directory*
  - e.g., ls -l   # since there is no directory, this is a long listing of the current directory
- **ls -a** *directory* # lists *all* contents of *directory,* including the dot files (ones that start with a .)
- **ls -F** *directory* # lists contents of *directory*, labelling directories with a slash, programs with *
- **ls -lt** *directory* # a long listing of the contents of *directory,* sorted by time last modified

Some flags can be combined, e.g., ls -la is a long listing of all files

directory is optional - if omitted, it lists contents of the current directory

directory may be a relative path or an absolute path

Is a command taking too long? Use control-C to stop execution.

Seeing What's Out There: ls

- **ls -ltr** # reverse sort by time
- **ls -S** # sorts by size
- **ls -rS** # sorts by reverse size

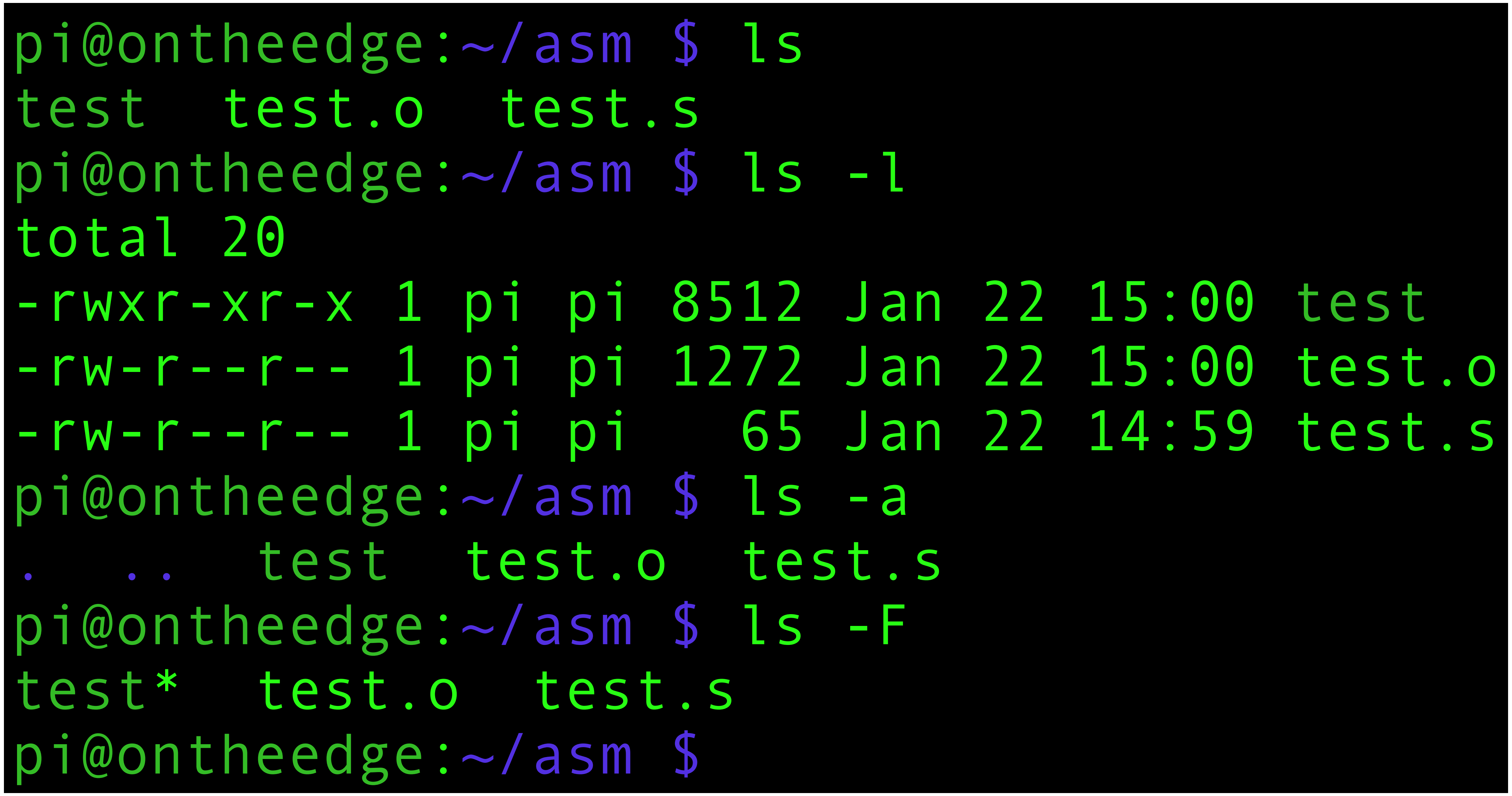
Some flags can be combined, e.g., ls -la is a long listing of all files

directory is optional - if omitted, it lists contents of the current directory

directory may be a relative path or an absolute path

Is a command taking too long? Use control-C to stop execution.

Example: ls



```
pi@ontheedge:~/asm $ ls
test  test.o  test.s
pi@ontheedge:~/asm $ ls -l
total 20
-rwxr-xr-x 1 pi pi 8512 Jan 22 15:00 test
-rw-r--r-- 1 pi pi 1272 Jan 22 15:00 test.o
-rw-r--r-- 1 pi pi   65 Jan 22 14:59 test.s
pi@ontheedge:~/asm $ ls -a
.  ..  test  test.o  test.s
pi@ontheedge:~/asm $ ls -F
test*  test.o  test.s
pi@ontheedge:~/asm $
```

File Permissions

- Every file can be **r**ead, **w**ritten to, or e**x**ecuted, depending on its permissions. These permissions can be specified separately for the **u**ser who owns the file; the file's **g**roup; and **o**ther users.
- For example, if a file has permissions rwx r-x ---, it means that the user can read write and execute it; anyone in the group can read and execute it (but not write to it), and others can do nothing at all to the file
- **chmod** can be used to change a file's permissions
- e.g., chmod u=rwx,g=rx,o= hello.bin # permissions are rwx r-x ---
- For details, read the docs.

Directory Permissions

- r (read): you can list the directory’s contents
- w (write): you can create, delete, or rename files in the directory
- x (execute): you can enter the directory (cd into it) and access files inside it
- All three are typically needed to work with a directory.
- chmod can be used to change a file’s or directory’s permissions (although on macOS, chmod u=x *directory* will not work

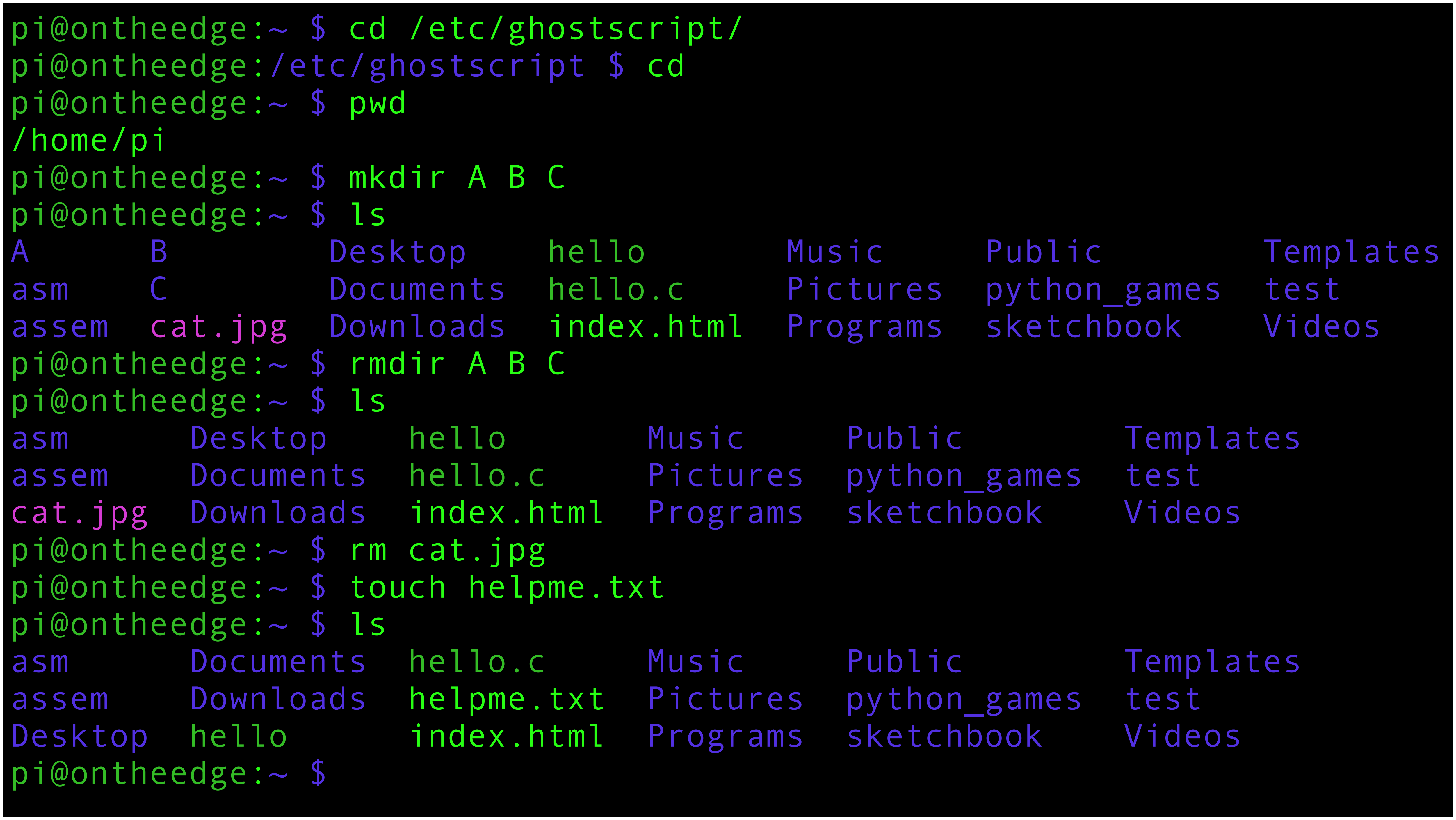
Moving About the File System: cd

- **cd** *directory* - changes to specified directory (home if it's omitted)
  - cd /etc/ghostscript # changes to the /etc/ghostscript directory
  - cd # wherever you were, now you are home!
    # Call it the ruby slippers command 😀
  - cd ~ # also brings you home
  - cd .. # one directory above where you were
  - cd ../.. # two directories above where you were
  - cd CS246 # into the CS246 directory from Class Docs
  - cd / # root level of the file system

Altering the File System: rm, rmdir and touch

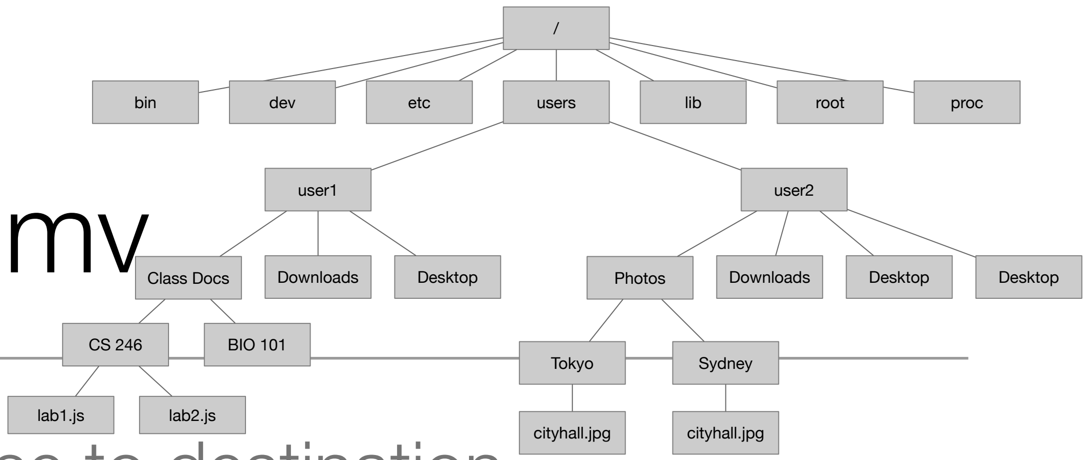
- **mkdir** *directory* - makes a directory (or directories)
  - cd ~ ; mkdir Programs # changes to home, then makes the Programs directory
  - mkdir wins losses ties # makes 3 directories, wins, losses, ties, in the current directory
- **rmdir** *directory* - removes specified directory (or directories)
  - rmdir wins
  - rmdir losses ties
- **rm** *filename* - removes a specified file
  - rm *.jpg   # removed all files ending in .jpg in your current directory. Use it wisely, as there is **no** undo command 😱
  - rm ~/Pictures/London/*.jpg.   # removed all .jpg files in ~/Pictures/London
- **touch** *filename* - creates an empty file (or files) in the current working directory
  - touch hello.c goodbye.c see_ya_later.c # creates 3 files
  - touch file-{1.10}.txt   # creates file-1.txt, file-2.txt, ..., file-10.txt 😀 -- pretty cool, eh? The power of wildcards...

Moving About and Altering the File System



```
pi@ontheedge:~ $ cd /etc/ghostscript/
pi@ontheedge:/etc/ghostscript $ cd
pi@ontheedge:~ $ pwd
/home/pi
pi@ontheedge:~ $ mkdir A B C
pi@ontheedge:~ $ ls
A      B        Desktop    hello       Music     Public        Templates
asm    C        Documents  hello.c     Pictures  python_games  test
assem  cat.jpg  Downloads  index.html  Programs  sketchbook    Videos
pi@ontheedge:~ $ rmdir A B C
pi@ontheedge:~ $ ls
asm      Desktop    hello       Music     Public        Templates
assem    Documents  hello.c     Pictures  python_games  test
cat.jpg  Downloads  index.html  Programs  sketchbook    Videos
pi@ontheedge:~ $ rm cat.jpg
pi@ontheedge:~ $ touch helpme.txt
pi@ontheedge:~ $ ls
asm      Documents  hello.c     Music     Public        Templates
assem    Downloads  helpme.txt  Pictures  python_games  test
Desktop  hello      index.html  Programs  sketchbook    Videos
pi@ontheedge:~ $
```

Copying, and Moving Files: cp, mv



- **cp** *source destination* # copies a file from source to destination
  - cp cat.jpg Pictures # copies cat.jpg into the Pictures directory
  - ls -l Pictures # verifies that cat.jpg is there
  - cp cat.jpg .. # copies cat.jpg into parent directory (e.g., if you are in Pictures .. = /home/pi)
- **mv** *source destination* # moves a file from one location to another
  - mv cat.jpg fluffy.jpg # renames cat.jpg as fluffy.jpg
  - mv fluffy.jpg .. # .. means the directory above your current directory
  - mv Photos/Sydney Desktop # moves Photos/Sydney directory to Desktop

Viewing Files: cat, head, tail, nano (!)

- **cat** *file* # lists the content of a file or files
- **head** *file* # lists the first 10 lines of a file
- **tail** *file* # lists the last 10 lines of a file
- **nano** *file* # opens file for editing

Copying, Moving, Creating and Viewing Files

```
pi@ontheedge:~/asm $ ls
test  test.o  test.s
pi@ontheedge:~/asm $ cp test.s helpme.s
pi@ontheedge:~/asm $ ls
helpme.s  test  test.o  test.s
pi@ontheedge:~/asm $ mkdir bin
pi@ontheedge:~/asm $ mv test.o bin # now it's in the bin directory
pi@ontheedge:~/asm $ ls
bin  helpme.s  test  test.s
pi@ontheedge:~/asm $ ls bin # lists contents of bin directory
test.o
pi@ontheedge:~/asm $ touch diary.txt
pi@ontheedge:~/asm $ ls
bin  diary.txt  helpme.s  test  test.s
pi@ontheedge:~/asm $ cat test.s
.global main
.func main
.text

main:
  mov r0, #10
        bx lr
```

Tab Completion and Up/Down Arrow Keys

- Tab completion allows you to just write a few letters and press tab to insert the name of a file
- Every command becomes part of the terminal window's history
- Use ⬆️ and ⬇️ arrow keys to get to a previously used statement and then press enter, rather than tediously retype the entire command
- You can also edit the command before pressing enter, if necessary

Editors

- vim, emacs (powerful, definitely geeky)
- nano (less difficult, less geeky)
- VS Code's editor
- VS Code's editor with vscodevim

Search

- **find** *directory pattern* # searches a directory/subdirectory for patterns
  - find . -name "*.jpg" # searches current directory and those below
    # for files that end with jpg
- **grep** *pattern file(s)* # search file(s) *contents* for *regular expressions*
  - grep "for*" ~/Programs/* # searches all files in Programs for
    # words starting with "for"
  - grep "f[aeiou]r" ~ # searches for far, fer, ..., fur in the home directory
- **whereis** # finds the location of a given command
  - whereis grep # /usr/bin/grep

Miscellaneous Commands and Operators

- **unzip** # unzips a zipped file
- **tar** # store or extract files from a tape archive
- **pipe (|)** - sends output from one command as input into another
  - e.g., curl --help | less # takes the help and pipes it into less, so you can scroll back and forth
- **&** # runs a command in the background
  - e.g., curl http://www.google.com & echo "hello raspian"
- **wget** # downloads a file from the web
- **curl** # downloading/uploading files from a web server
- **man** # manual pages
- **df** # shows file system usage (e.g., df / shows file system usage for root); use -h for human readable" values

Pipe in Action

```
# Count how many files are in a directory
ls | wc -l

# Find all .js files containing the word “fetch”
grep -r "fetch" . | grep "\.js"

# Show the top 10 largest files
du -h * | sort -h | tail -10

# Sort a list of words and remove duplicates
cat words.txt | sort | uniq

# See what commands you run the most
history | awk '{print $2}' | sort | uniq -c | sort -nr | head

# Show only the first column of a CSV
cat data.csv | cut -d, -f1

# Find processes using the most memory
ps aux | sort -nrk 4 | head

# Continuously watch the number of running processes
watch "ps | wc -l"

# Show only directories (no files)
ls -l | grep "^d"

# Extract all URLs from an HTML file
cat page.html | grep -o 'http[^"]*'
```

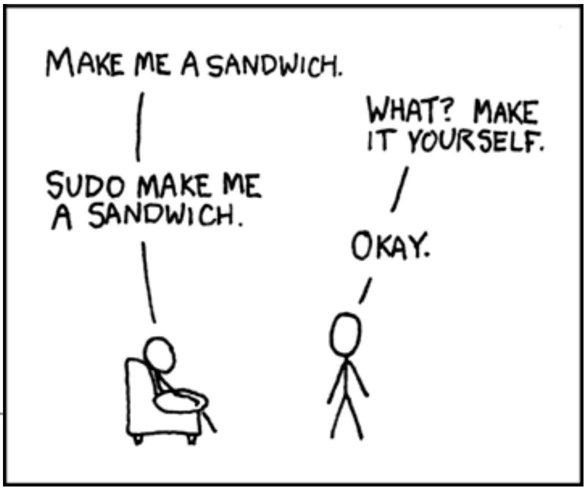
Miscellaneous Commands and Operators

```
pi@ontheedge:~/asm $ cat test.s | wc # takes output of test,pipes it into wc
      8      11      65
pi@ontheedge:~/asm $ wget -nv http://cbc.ca & #fetched web page in background
```

You May Need These Someday: Networking

- **ping** #to see if a host is available
- **map** # network exploration and scanning tool
- **hostname** # name of the host
- **ifconfig** # determine network info

Sudoers



- Linux is a multiuser operating system: multiple people can log in and run programs simultaneously
- Ordinary users can only work in their own home directories, and are limited in what applications they can run
- A superuser, known as root, can access any files on the system and do virtually anything
- To issue a command with the same authority as root, preface that command with the term **sudo**
  - e.g., sudo apt-get install
- Only certain users -- those listed in /etc/sudoers -- can use sudo
- While you are not in this file on this system, you could on your own Linux box -- we recommend Raspberry Pis.

Installing New Software :-)

- **apt** - advanced package tool - is used to easily install/upgrade/remove software from Debian
- If the software works with the ARM architecture, and is in a Debian package, it should work
- apt keeps a list of repositories at /etc/apt/sources.list. To keep it up-to-date, run:
    - sudo apt-get update

apt

- To install a package:
    - sudo apt-get install *thingtoinstall*
- To remove a package:
    - sudo apt-get remove *thingtoremove*
- To update a package:
    1. sudo apt-get update *thingtoupdate*
    2. sudo apt-get upgrade *thingtoupdate*
- To search for software:
    - apt-cache search *termtosearchfor*
- To get detailed information on a particular package
    - apt-cache show *package*

Resources

- Fundamental Linux Usage - https://www.raspberrypi.org/documentation/linux/

- The Linux Command Line - http://linuxcommand.org/index.php

- The Debian Wiki = https://wiki.debian.org/FrontPage

- File System Details - https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard