02. Pair Programming, Environments, Shell Commands

CPSC 120: Introduction to Programming

Kevin A. Wortman ~ CSU Fullerton

Agenda

0. Sign-in sheet
1. Q&A
2. Pair Programming
3. Development Environments
4. Shell Commands
5. (section 03) Supplemental Instruction Introduction

1. Q&A

Q&A

Let’s hear your questions about...

- This week’s Lab
- Linux
- Any other issues

Reminder: write these questions in your notebook during lab

2. Pair Programming

Why Pair Programming?

According to research, pair programming improves...

- **Quality** of the work
- Amount of **time** taken
- **Enjoyment** of the process
- **Collaboration** and **communication** skills
- **Peer networks**
- **Retention**: number of students who pass course, remain in major

Forming Pairs

- Pairings are created randomly
  - Supports goals on previous slide
  - NCSU study: 93% satisfaction w/ random partners
- New partner each lab
- Odd # students: one group of 3
- To keep working outside lab class
  - Need to schedule yourself
  - Be flexible and professional
  - ECS Open Lab, room CS-200: http://www.fullerton.edu/ecs/cs/resources/labs.php

Grading

- Make one submission (one GitHub repo) per pair
- Both partners will get the same grade
- Later: confidential survey on your partner’s cooperation
- Participation is a part of your lab grade

Roles

- Pair shares one PC
- *Driver:* controls keyboard and mouse
- *Navigator:* observes, asks questions, suggests solutions, longer-term strategies, tracks flowchart
  - Ex. “remember to save before compiling”
  - Group of 3: two navigators
- Switch every **30 minutes**: TA’s phone timer or verbal announcement

Dealing with Differences

- Expect mismatch of preparation, hard skills, soft skills
- Partner not participating properly:
  - First bring it up directly to them
  - Can ask TA/ILA for help/clarification during lab
  - Can contact TA/instructor outside of class

3. Development Environments

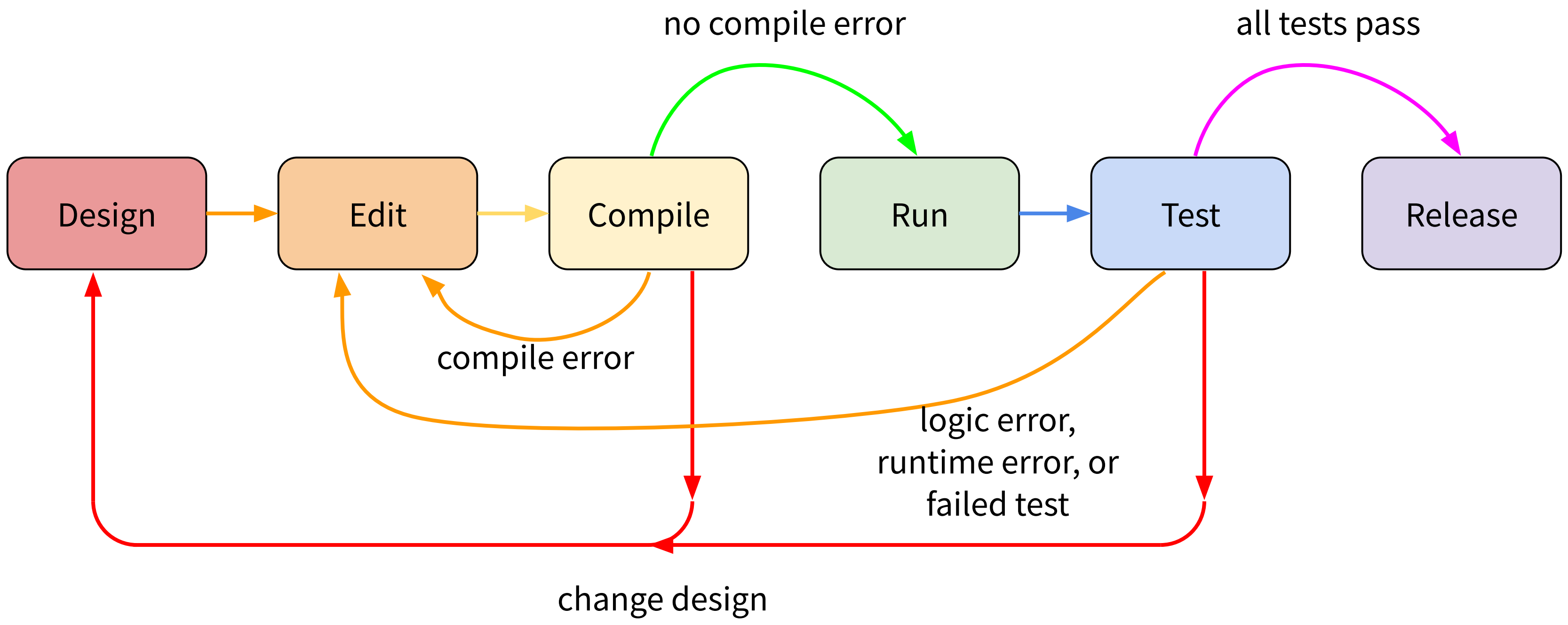
Development Environments

- *Development Environment:* suite of software tools for programming
    - Edit, compile, run, test, debug, release
- *Integrated Development Environment (IDE)*
    - Microsoft Visual Studio, Apple Xcode, Eclipse, …
    - Graphical native app for all tasks
    - Intimidating, confusing for beginners
- *Command-Line Interface (CLI)*
    - Separate shell command for each task
    - Old-school (nothing wrong with that)
    - Learn in pieces
    - Exposes what’s happening
    - What we are doing

Consumer versus professional workspaces

Consumer experience	**Professional workspace**
movie theater	movie set
restaurant dining room	commercial kitchen
(polished experience of final product)	(productive, safe, creative workshop)
Windows, macOS, Android, iOS, XBox, ...	Linux, Xcode, Visual Studio
inadequate for creation	supports creation

The Development Cycle



Keyboard-First Principle

- Humans can type faster than they can click
- Excessive mouse moving causes Repetitive Stress Injury
  - RSI, Carpal Tunnel Syndrome
- **Keyboard-First Principle**: using keyboard is better than mouse

Mise-en-Place Principle

- Mise-en-Place: putting tools, components in place for ergonomics

Programming Mise-en-Place

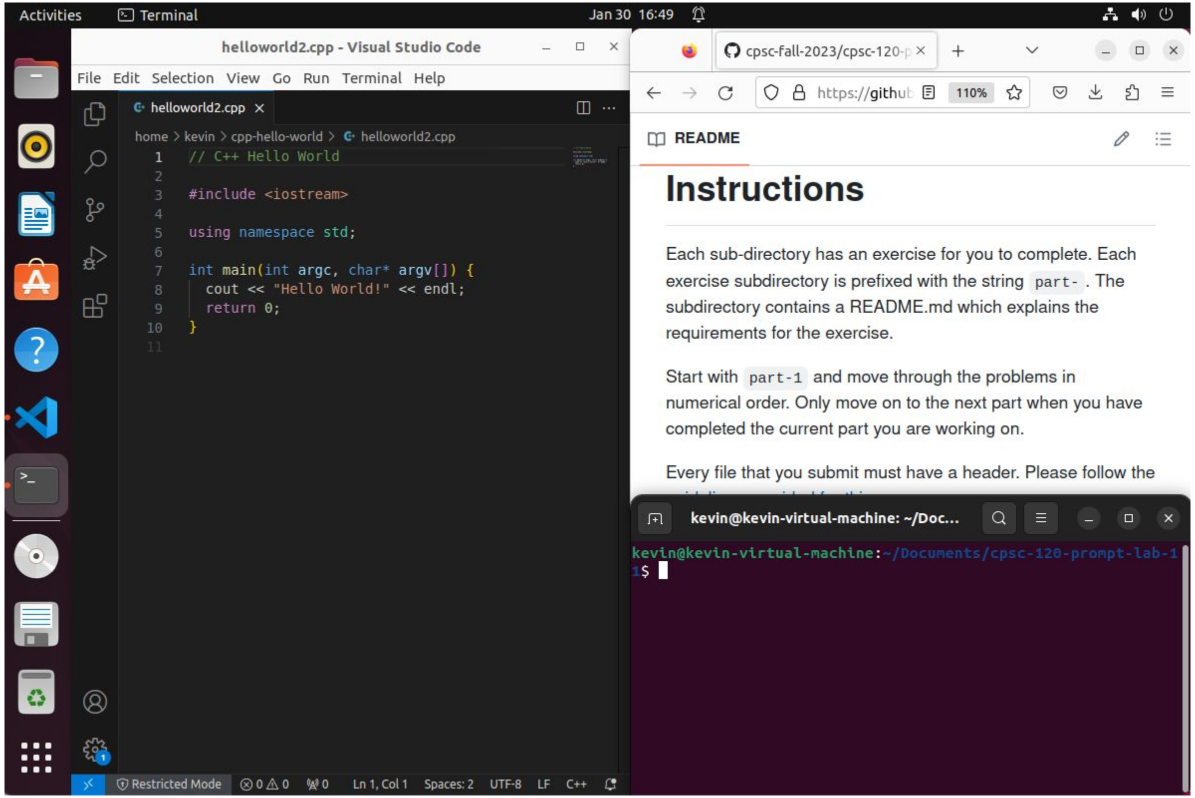
Gotta-haves while programming:

1. Editor (VS Code)
2. Shell
3. Documentation (browser, lab instructions, cppreference.com, etc.)

Best Practice:

- (keyboard-first, mise-en-place)
- Arrange windows so you can see 1, 2, 3 at the same time
- ALT-TAB to switch between the windows (don’t click)

Window Mise-en-Place


Activities
Terminal
Jan 30 16:49
helloworld2.cpp - Visual Studio Code
File Edit Selection View Go Run Terminal Help
helloworld2.cpp
home > kevin > cpp-hello-world > helloworld2.cpp
Restricted Mode
Ln 1, Col 1 Spaces: 2 UTF-8 LF C++
cpsc-fall-2023/cpsc-120-p
https://githut
110%
README
kevin@kevin-virtual-machine: ~/Doc...

```
// C++ Hello World

#include <iostream>

using namespace std;

int main(int argc, char* argv[]) {
  cout << "Hello World!" << endl;
  return 0;
}
```

Instructions

Each sub-directory has an exercise for you to complete. Each exercise subdirectory is prefixed with the string `part-`. The subdirectory contains a README.md which explains the requirements for the exercise.

Start with `part-1` and move through the problems in numerical order. Only move on to the next part when you have completed the current part you are working on.

Every file that you submit must have a header. Please follow the

```
kevin@kevin-virtual-machine:~/Documents/cpsc-120-prompt-lab-1
1$ 
```

Unix, Linux, Ubuntu

- **Unix**: widely-used framework for operating systems
  - All modern platforms except Microsoft
  - macOS, iOS, Linux, Android, Chrome OS, PlayStation, cloud servers, ...
  - WSL: Unix inside Windows
- **Linux**: a popular, free, version of Unix
  - Created by Linus Torvalds
  - Rhymes with his Finnish name: “linnukks”
- **Ubuntu**: Linux distribution (version)
  - Popular
  - Good installation support
  - What we are using
- See *CPSC 351 Operating Systems*

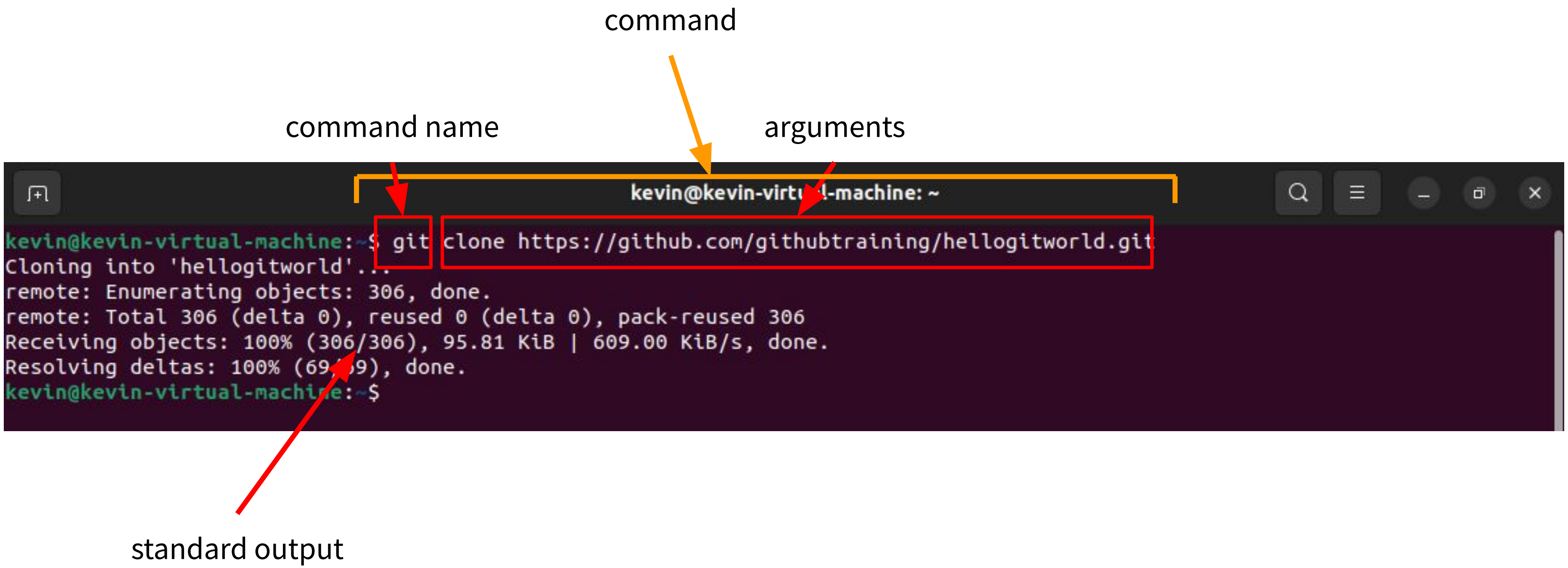
Files, Text Files, Editors

- **Text file**: a file that contains human-readable text
- Types of text files
  - .txt: text for human consumption, e.g. LICENSE.txt
  - .cc: C++ source code
  - .md: Markdown, text for human consumption with formatting, e.g. README.md
- **Text editor** (aka “editor”): program for opening, editing, saving text files
  - Core programmer’s tool
- We use an editor called **VS Code**

Shell and Terminal

- **Shell**: a special Unix program that allows a user (you) to run and interact with other programs
- **Terminal**: a thing that lets you see shell input/output
  - Physical terminal: monitor, keyboard, connection to real computer
  - Terminal emulator: program that simulates a physical terminal
- **Prompt**: when the shell is waiting for a command,
  It prints a “prompt” ending in $ (dollar sign)
- You type a **command**, then the Enter key to run the command
- Unix programs are **concise**: if everything worked, there is no stdout output

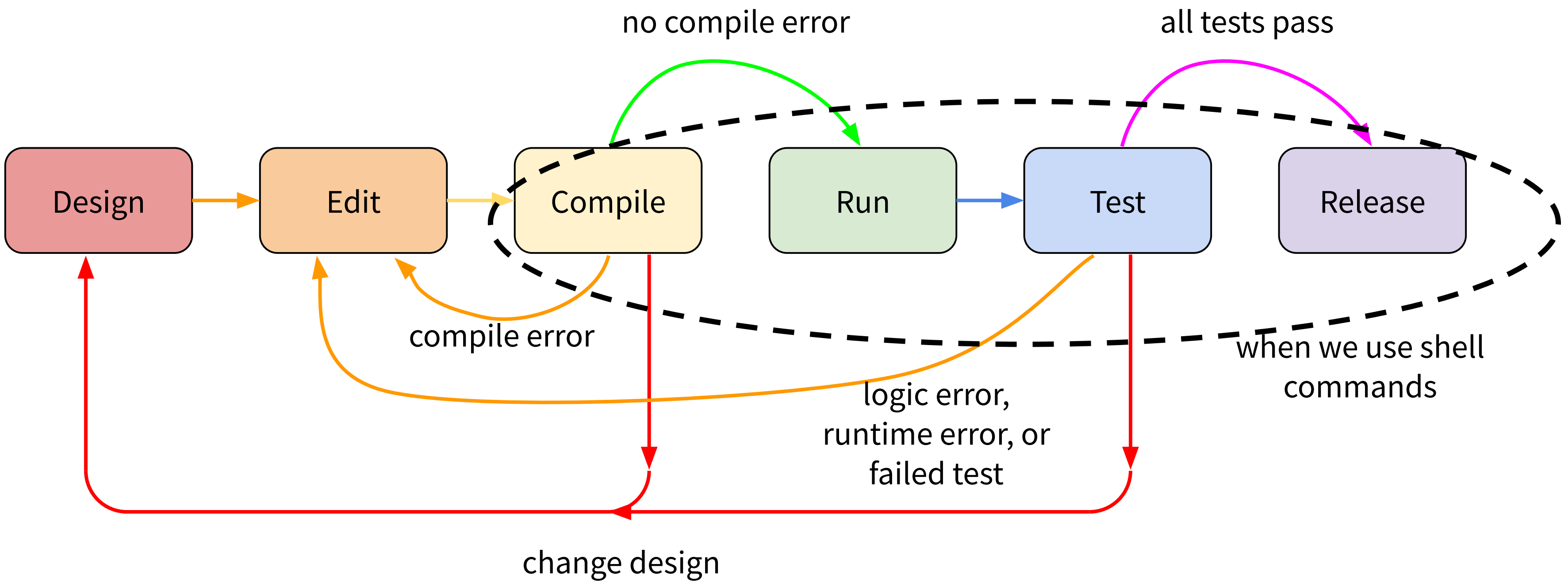
Running a Shell Program



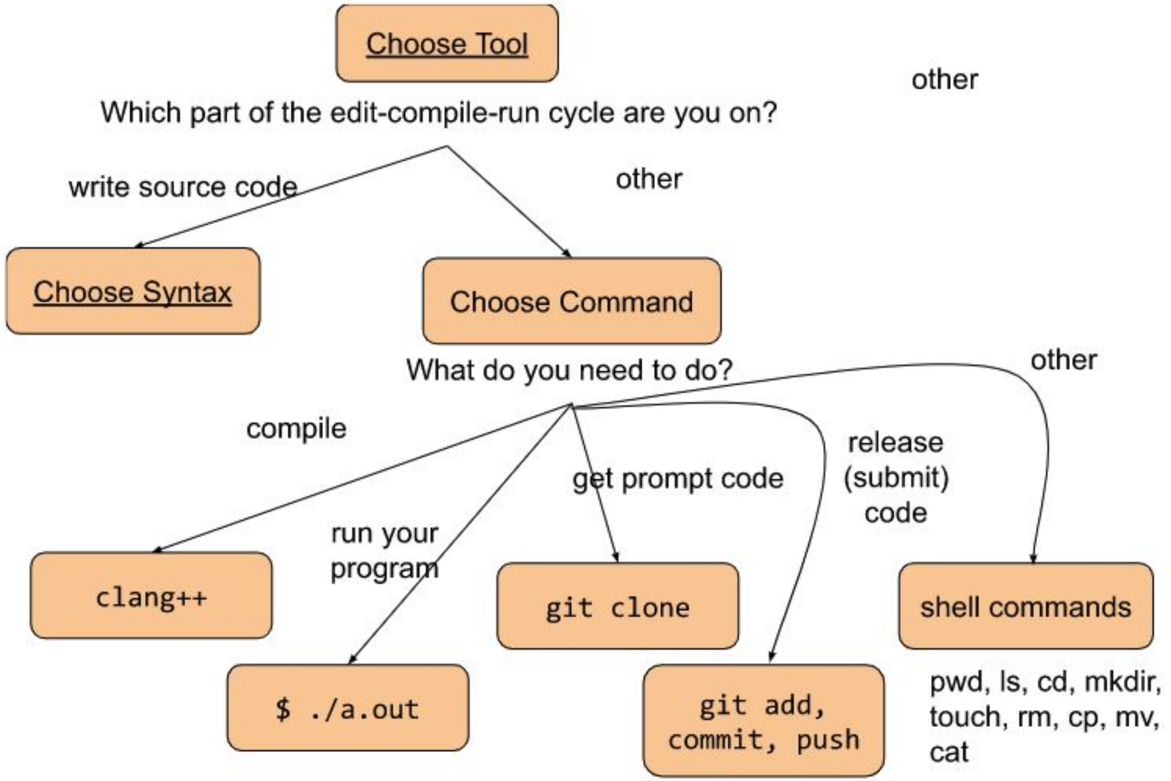
```
kevin@kevin-virtual-machine:~$ git clone https://github.com/githubtraining/hellogitworld.git
Cloning into 'hellogitworld'...
remote: Enumerating objects: 306, done.
remote: Total 306 (delta 0), reused 0 (delta 0), pack-reused 306
Receiving objects: 100% (306/306), 95.81 KiB | 609.00 KiB/s, done.
Resolving deltas: 100% (69/69), done.
kevin@kevin-virtual-machine:~$
```

4. Shell Commands

The Development Cycle



Choose Tool Flowchart



Filesystem

- Unix organizes storage into a **filesystem**
- A **file** holds data and has a **filename** (e.g. README.txt)
- A **directory** holds files or other directories
    - *Family tree* analogy: the “**parent”** directory holds “**child”** files/directories
- The **root** directory, written / (forward-slash), is the parent of everything else
- A **path** is the location of a file
- **Absolute path**: directions starting from /, with / separating each directory/file name
    - Ex: /usr/share/dict/words
    - The initial / means “start from the root”

Current Directory

- **current directory** = location where a program "is"
  - a.k.a. **working directory**
- **State:** current configuration, subject to change
- Keep current directory in mind
  - Unlike search-based apps

Relative Paths

Special path names:

- **Current directory**
- **Home directory**: user student has a “home directory” at /home/student
- **Aliases** (abbreviations) for these:
  - Current directory = . (dot/period)
  - Parent directory = .. (two dots/periods)
  - Home directory = ~ (tilde; look above the TAB key)
- **Relative path**: path relative to . or .. or ~
  - Ex.: if you are in ~, then ~/Documents and ./Documents are relative paths to /home/student/Documents
  - Relative paths do not start with /

Pattern: Shell Command

```
$ COMMAND [ARGUMENT...]
```

- Cues that this is a shell command
  - Dollar sign
  - Fixed-width font
- You type everything **after the $**, then press Enter key
- ALL-CAPS are fill-in-the blank
- [BRACKETS] means optional
- ELLIPSIS... means you may repeat

cd

```
$ cd [DIRECTORY]
```

- cd: **change directory**
- [DIRECTORY] provided: change current directory to DIRECTORY
- Otherwise (omitted): change to ~ (home)

ls

```
$ ls [OPTION...]
```

- ls: **LiSt** files
- prints files in the current directory
- Good habit: ls after entering a directory, to check that you are where you think you are

pwd

```
$ pwd
```

- pwd: **P**rint **W**orking **D**irectory
- Prints the current directory as an absolute path
- If you’re confused about where you are, pwd to get your bearings