03. Shell Commands, Git, Demo

CPSC 120: Introduction to Programming
Pratishtha Soni~ CSU Fullerton

Agenda

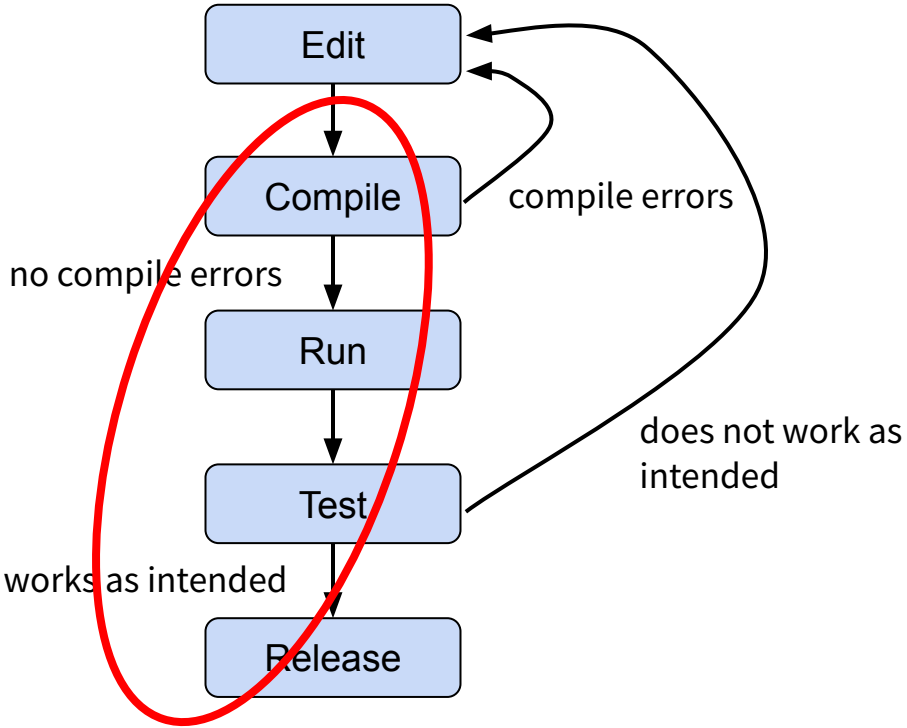
Shell Commands

Git

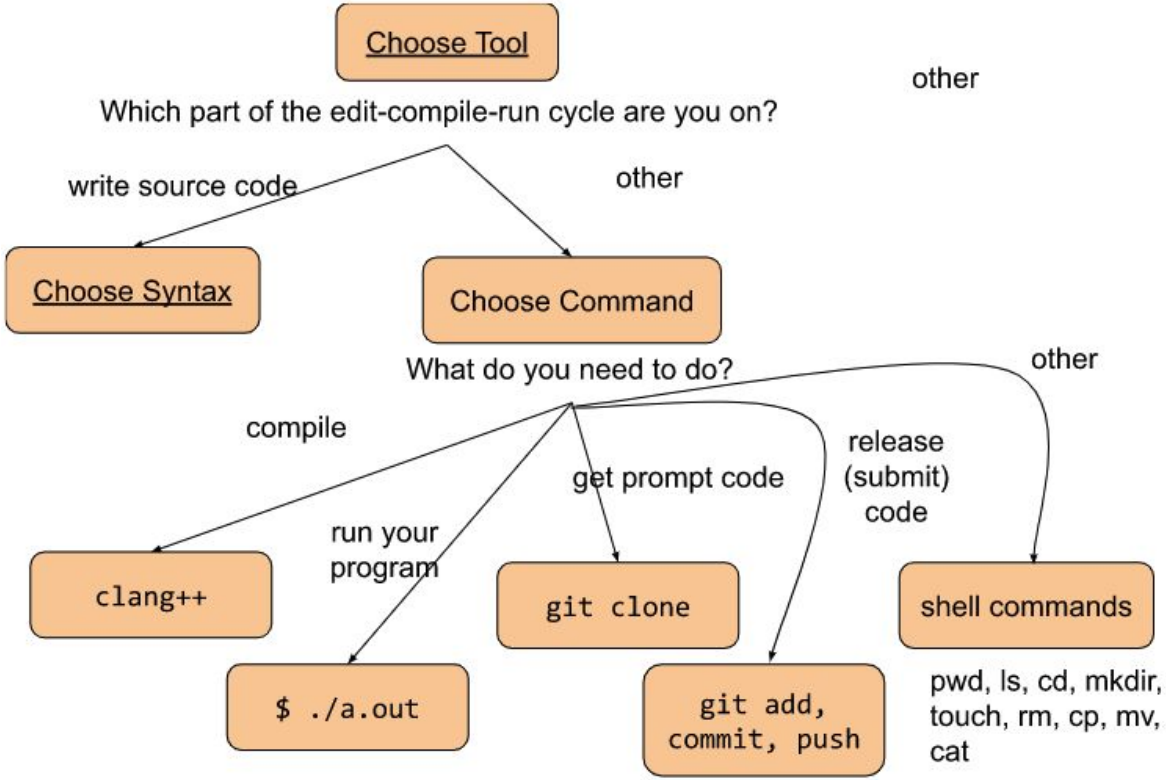
Demo

1. Shell Commands

The Edit-Compile-Run 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

Demo: cd, ls, pwd

mkdir

```
$ mkdir PATH
```

- mkdir: **M**a**K**e **DIR**ectory
- Creates a new empty directory in PATH
- Remember that you can use absolute or relative paths

rmdir

```
$ rmdir PATH
```

- rmdir: **R**e**M**ove **DIR**ectory
- PATH must be a path to an **empty** directory
- (safety mechanism to prevent you from deleting files by mistake)
- Two patterns to note
  - Naming pattern: mk = make, rm = remove
  - Safety interlocks: tools are picky to prevent harmful oversights

rm

```
$ rm [OPTION...] FILE...
```

- rm: **R**e**M**ove regular file
- each FILE must be a path to a regular file (not a directory)
- Important OPTIONs:
  - -R: recursive; if FILE is a directory, remove it and all its children
  - -f: force, “that’s an order!”; ignore safety interlocks
- The -R and -f arguments are used in other commands

echo $?

```
$ echo $?
```

- Prints the **exit code** of the previous command
- 0 means success
- Positive or negative number means failure

Demo: Editing files, mkdir, rmdir, rm, exit code

clang++

```
$ clang++ [OPTION...] SOURCE
```

- clang++: open source C++ compiler
- SOURCE must be a path to a C++ source file, usually ending in .cc
- Tries to compile SOURCE
  - On failure, clang++ prints **messages** describing **syntax errors** or **warnings** (problems)
  - On success, creates an executable object code program named **a.out**
- (There are many OPTIONs to clang++ that we'll cover later.)

Running your program; `./a.out`

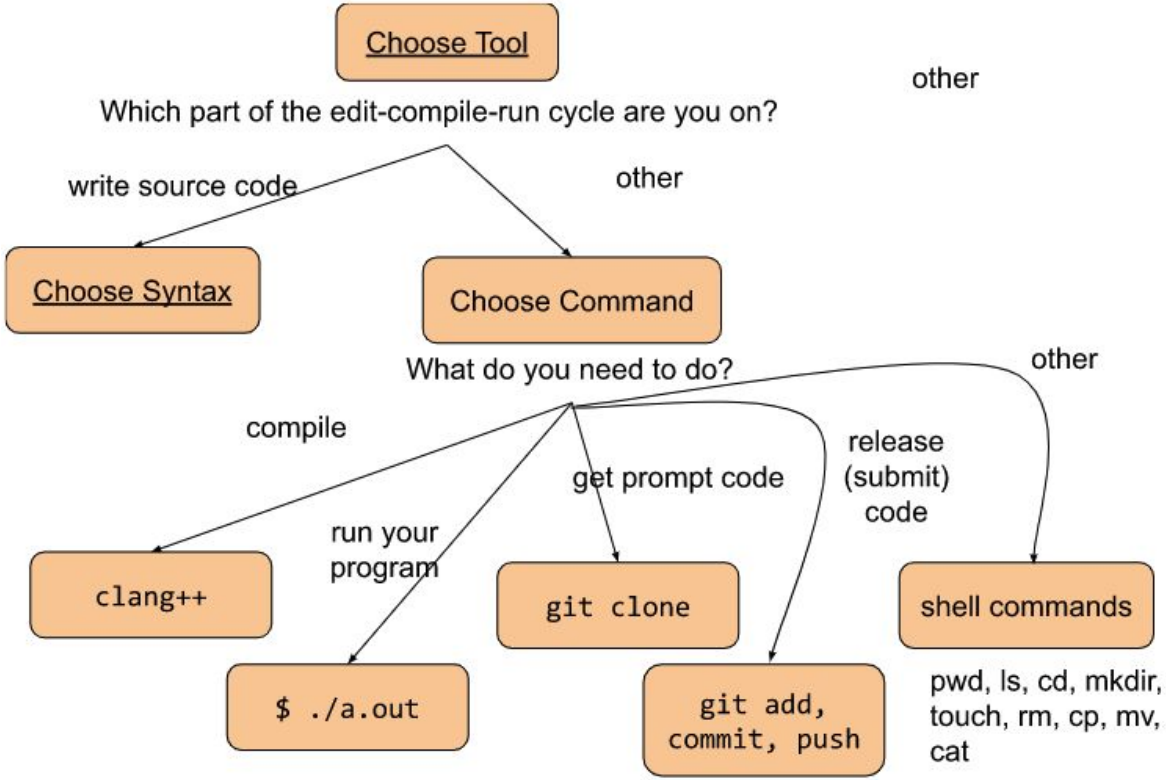
```
$ ./PROG

$ ./a.out
```

- tldr; the command is **`./a.out`**
- To run a program
  - The command name is the path of the executable object code program file
  - For technical reasons, this must begin with a directory name
  - To run program **PROG** in the current directory, run **`./PROG`**
- The clang++ command on the previous slide creates program **a.out**
- To run it, use command **`./a.out`**

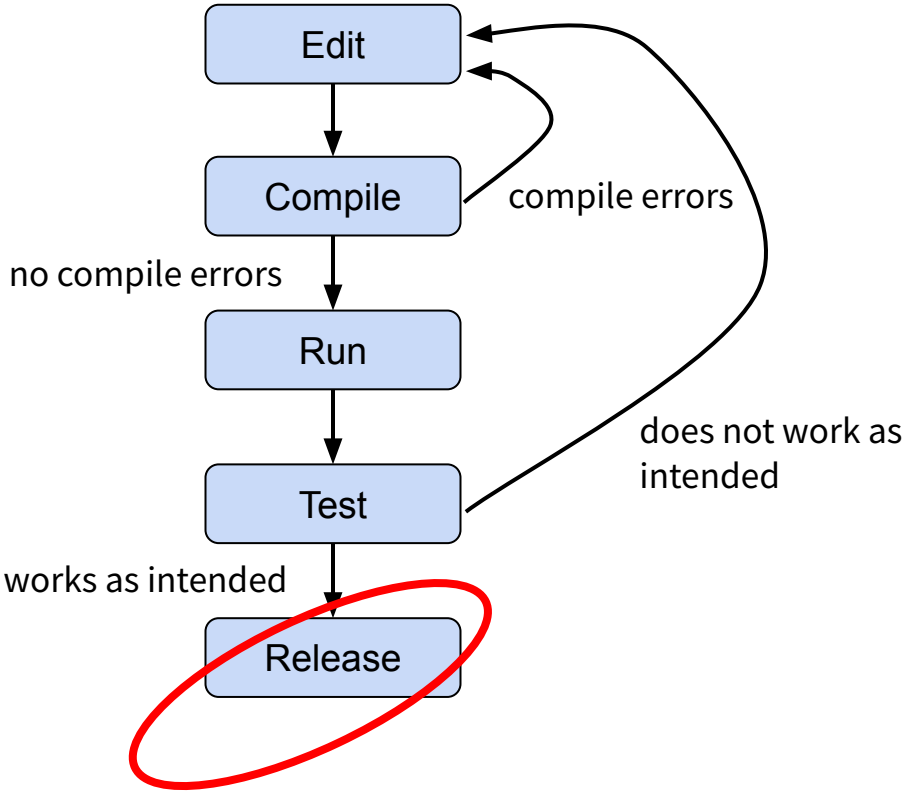
Demo: Saving, Compiling, and Running

Choose Tool Flowchart



2. Git

The Edit-Compile-Run Cycle



Git and GitHub

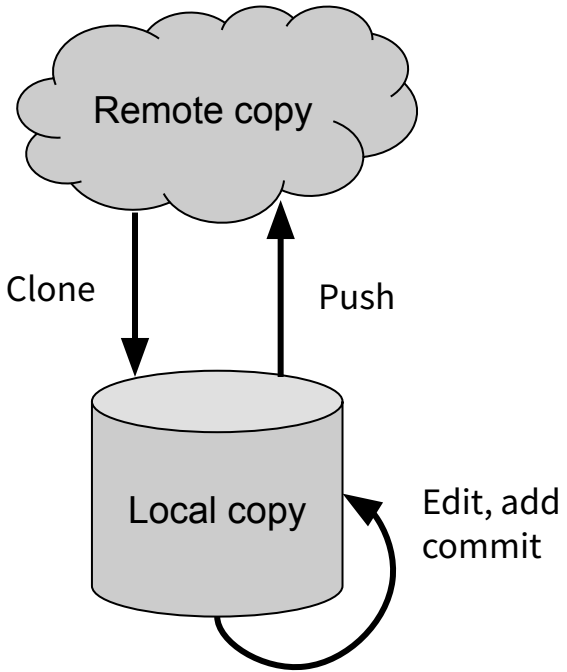
- **Source code control**: tool for programmers to share, track source code
- **git**: popular source code control shell program
- **GitHub**: cloud git service
  - facilitates sharing code with others around the world
- **Repository** (“**repo**”): holds a project
- Example: chromium, chrome_history_client.cc
- Our labs

GitHub Workflow

git understands that a repo can be copied into multiple places at the same time

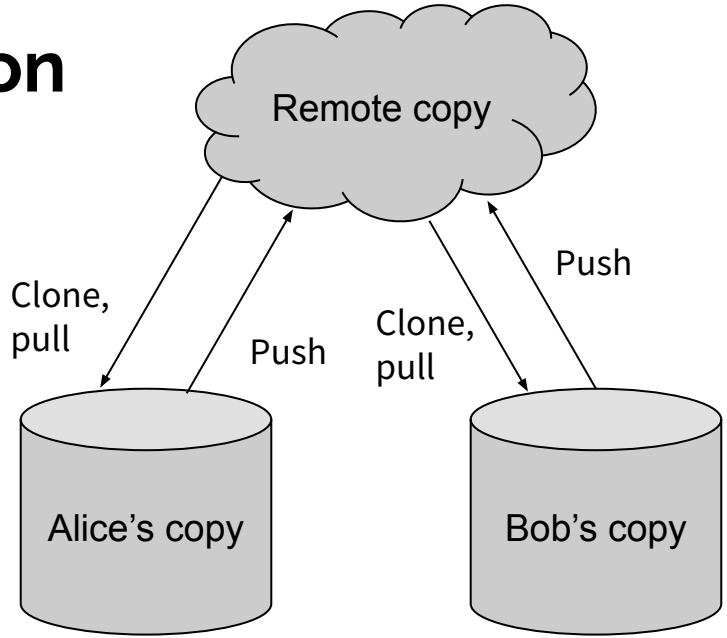
single-developer workflow:

1. Create a **remote copy** repo (lives on github.com)
2. **Clone** a **local copy** onto your computer
3. Edit, save files inside local copy
4. Create **commit(s)** summarizing changes
5. **Push** commits to **remote copy**



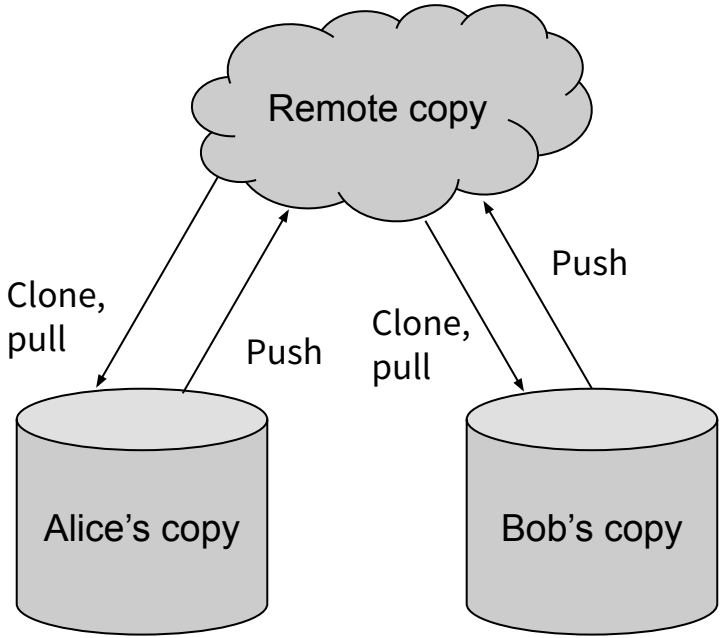
Multi-Developer Synchronization

- Git is intended for large teams
- **Synchronization problem:** what if...
- Alice, Bob both clone their own copies
- Alice changes main.cc
- Bob changes main.cc differently
- Alice pushes
- Bob pushes
- **Which version of main.cpp wins, Alice's or Bob's?**
  - A human must decide



Pull and Merge Conflicts

- Git rule: if your repo is behind, you have to...
- ...**pull** remote changes
- ...resolve conflicts
- ...**push**
- Suggestion: **avoid this** at first
- Use one account and computer at a time



Releasing Work to GitHub: Theory

- Git records edits, who made them, when, why
  - see chrome_history_client.cc history
- Working backwards...
- **git push**: transmits every **commit** in your local repo to GitHub.com
  - First, a commit needs to exist
- **git commit**: logs a commit action
  - Applies to all currently-**staged** files
  - Commit **message**: human-readable text describing what you did
  - First, at least one file needs to be staged
- **git add**
  - **Stages** a file = “this file is part of the next commit”

Releasing Work to GitHub: Practice

Working forwards…

1. Edit, save work in VS Code
2. Compile (`clang++`), run (`./a.out`), test
3. **Add**: for each FILE you changed,
   ```
   $ git add FILE
   ```
4. **Commit**: once,
   ```
   $ git commit -m “MESSAGE”
   ```
5. **Push**: once,
   ```
   $ git push
   ```
6. Check: confirm changes are visible on github.com

git clone

```
$ git clone REPO
```

- REPO comes from the “Clone or download” button and ends in .git
  - For https://github.com/cpsc-pilot-fall-2022/hello-world
  - REPO-URL = https://github.com/cpsc-pilot-fall-2022/hello-world.git
- Download the contents of REPO into a directory on the local computer
- Prints status to stdout, even on success
- May ask for your GitHub username/password

git status

```
$ git status
```

- Must be run **inside a git repo**
- Prints out
  - List of files that have been modified, but not committed yet
  - List of all commits that haven't been pushed yet
- Quick way to check for un-pushed work

git add

```
$ git add FILE...
```

- Must be run **inside a git repo**
- Each of FILE… is hereby “**staged for commit**”
- (the next `git commit` will apply to them)

git commit

```
$ git commit -m “MESSAGE”
```

- Must be run **inside a git repo**
- Creates a **commit** that applies to all currently-staged files
- MESSAGE should be a human-readable description what the commit represents
  - E.g. “fixed the crash bug”, “finished lab 2”
- Note: **quotes** around MESSAGE !
- If you forget -m “MESSAGE” then git will open a venerable editor “nano” and force you to write a message that way

git push

```
$ git push
```

- Must be run **inside a git repo**
- Upload all local commits to the remote repo
- Synchronization check
  - git checks for commits that were pushed since you last cloned/pulled
  - If you are out of sync, git push fails
  - Have to git pull first
- Should make all local changes visible on github.com
- Best practice: after a git push, look at your repo in a browser, confirm it is up to date

git pull

```
$ git pull
```

- Check remote copy for commits
- If any exist, download them to local computer
- git will tell you if there are conflicts
- If so, it will mark up the conflicting files, you need to use VS Code to clean them up
- Then you can push

Demo: Git clone, Edit, add, commit, push