

Mini-Lesson 1:

Command Line, Quarto,

Vim and GitHub

Alex Xi Lan

Review of commands

Action	macOS (Terminal)	Windows (PowerShell)
Show current folder	<code>pwd</code>	<code>pwd</code>
List files in folder	<code>ls</code>	<code>ls</code>
Change into a folder	<code>cd <folder></code>	<code>cd <folder></code>
Go up one level	<code>cd ..</code>	<code>cd ..</code>
Create a new folder	<code>mkdir <folder></code>	<code>mkdir <folder></code>
Make an empty file	<code>touch <filename></code>	<code>New-Item <filename></code>
View a text file quickly	<code>cat <filename></code>	<code>Get-Content <filename></code>

- need to be inside of a git repo directory to use git commands

Run a script at the command line

- to run a python script in the command line
 - `python3 path/file.py`
 - `path` is the file location relative to your current working directory
- to compile a qmd in the command
 - `quarto render path/file.qmd`

Quarto and .qmd file

- A **.qmd file** is a **Quarto** document
 - It combines text, code, and output in a single file
 - Quarto supports multiple languages: R, Python, Julia
 - Think of **.qmd** as a cross-language version of R Markdown.
- A **.qmd** file can be rendered into:
 - HTML, PDF, Slides, Word documents
- In this course:
 - **All problem sets will be written in **.qmd** files**
 - You will receive pre-filled **.qmd** templates via GitHub Classroom
 - Your job is to edit the template and render it
- to compile a qmd in the command
 - `quarto render path/file.qmd`

Vim: How Vim shows up in GH

- In this class you should mostly see vim when you forget to add a commit message
 - e.g. `git commit` with changes staged for commit, `git merge` when there're divergent histories.
 - If you don't provide `-m`, Git automatically opens a **text editor Vim**.
 - The best practice is to do `git commit -m` every time!
 - In the following example, I have changes staged for commit, but I run `git commit` without supplying a message.

```
[alexlan@Alexs-Mac-mini SolarTech % git status
On branch issue-1-demo
Your branch is up to date with 'origin/issue-1-demo'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code/clean/clean_policy.py

no changes added to commit (use "git add" and/or "git commit -a")
[alexlan@Alexs-Mac-mini SolarTech % git add .
[alexlan@Alexs-Mac-mini SolarTech % git commit]
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commi
#
# Committer: Alex Lan <alexlan@Alexs-Mac-mini.local>
#
# On branch issue-1-demo
# Your branch is up to date with 'origin/issue-1-demo'.
#
# Changes to be committed:
#       modified:   code/clean/clean_policy.py
#
~  
~  
~  
~  
~  
~  
~  
~  
~
```

Basic Vim workflow

- Vim has modes:
 - **Insert mode:** type text
 - **Normal mode:** issue commands
- **Insert text:**
 - Press `i` → enter Insert mode (Vim starts in Normal mode) → Type your commit message
- **Return to command mode:**
 - Press `Esc` → back to Normal mode
- **Save and quit (the usual case):**
 - Press `Esc` (exit Normal mode) → Type `:wq` and press Enter
- **Quit WITHOUT saving (escape hatch):**
 - Press `Esc` → Type `:q!` and press Enter

Basic Vim workflow (example)

- In the `git commit` example, I can press `i` to enter insert mode for the message, or force quit by typing `:q!` and enter.
 - In the insert mode, type the commit message at the top of the window.
 - To save and quit, press `Esc`, and type `:wq` at the bottom of the window and then Enter

```
this is the commit message for demo #1
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit
#
# Committer: Alex Lan <alexlan@Alexs-Mac-mini.local>
#
# On branch issue-1-demo
# Your branch is up to date with 'origin/issue-1-demo'.
#
# Changes to be committed:
#       modified:   code/clean/clean_policy.py
#
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

GitHub: Forking and Cloning Repos

- **Forking (on GitHub)** Creates **your own copy** of someone else's repository
 - Lets you make changes **without touching** the original
 - Click **Fork** on GitHub. Your fork lives at: `github.com/<username>/<repo>`
- **Cloning (on your computer)** Downloads a GitHub repo to your **local machine**
 - Steps:
 1. On GitHub → **Code** → copy HTTPS URL
 2. In Terminal: `gh repo clone <username>/<repo>`
 3. Enter the folder: `cd <repo>`
- Key difference
 - Forking creates a new GitHub repo under your account (remote ownership).
 - Cloning creates a local copy on your computer (no new repo)