# Mini-Lesson 1: Command Line, Quarto, Vim and GitHub

Alex Xi Lan

## Review of commands

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

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

These are the most frequently used terminal commands, and I will do a demonstration in the Mac Terminal: I will first create a sub-folder named minilesson_example under the minilesson_1 folder in repo. We need to change the current working directory in the terminal to where the minilesson folder's located. (pwd to show the current folder and then cd to /Users/alexlan/Documents/GitHub/student30538-w26/minilessons/minilesson_1) We are currently in the target directory, Let's see what are already here: (list the files in minilesson_1 using ls) To create the minilesson_example subfolder, (mkdir minilesson_example), to confirm it's created (ls) Let's make an empty text file in this subfolder! Still first change the current working directory (cd minilesson_example) (touch demo.txt) We can add some content using the echo demand (echo "hello world" > demo.txt) Now to check what's inside of demo.txt, (cat demo.txt) Ok we are done with miilesson_example folder, let's go back to minilesson_1 (cd ..)

## 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`

I have a pre-made python file called demo.py under minilesson_1 (show content), let's run it in terminal. (python3 demo.py) We can also compile a qmd in terminal, for example with our minilesson slides (quarto render minilesson_1.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** is a text editor with modes: Insert mode for typing, and Normal mode for issuing editing commands.
- 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 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
#
~
~
~
~
~
~
~
~
~
"~/Documents/GitHub/SolarTech/.git/COMMIT_EDITMSG" 12L, 346B
```

### 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:**

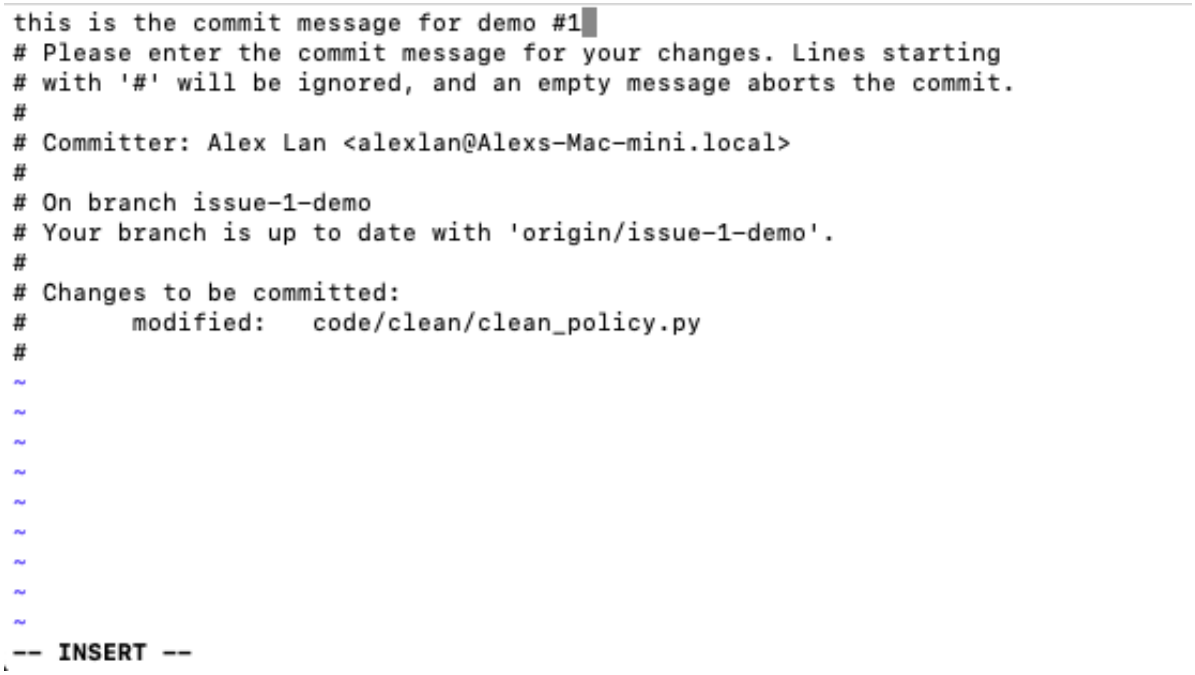
  - Press `Esc` → back to Normal mode

- **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

the colon (:) tells Vim that you're issuing a command. w stands for write, q stands for quit. the exclamation mark (!) means force the action.

### 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
#
~
~
~
~
~
~
~
~
~
:wq
```

## 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)
- If you want to add notes or annotations to the slides, do it outside the repo. Editing the slide files in the repo can cause conflicts when you pull updates.

(show ppl how to clone student30538-w26)