Git and Github

Ling 250/450: Data Science for Linguistics

C.M. Downey

Spring 2025

What even is Git?

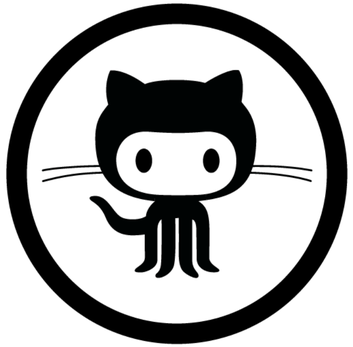
What even is Git?

- Git
  - At its core, a **system for tracking and saving changes** to files
  - Generally called a **Version Control System**
  - Very complex once you go beyond the basics (which we won't)

What even is Git?

- Git
  - At its core, a **system for tracking and saving changes** to files
  - Generally called a **Version Control System**
  - Very complex once you go beyond the basics (which we won't)

- Github
  - A website for **hosting Git projects online** (called **repositories**)
  - **Not strictly necessary** for using Git (though it almost always is)
  - Recently: acquired by **Microsoft** (Github, but not Git)



Main ideas of Git

Main ideas of Git

- Each change to each file must be **deliberately committed** (saved) to the master copy of the project
  - Unlike e.g. Google Docs, where it just saves as you go

Main ideas of Git

- Each change to each file must be **deliberately committed** (saved) to the master copy of the project
  - Unlike e.g. Google Docs, where it just saves as you go
- All changes are **tracked** and added to a **running history** of changes

Main ideas of Git

- Each change to each file must be **deliberately committed** (saved) to the master copy of the project
  - Unlike e.g. Google Docs, where it just saves as you go
- All changes are **tracked** and added to a **running history** of changes
- **Different versions** of the same project can **branch off**, and then later be **merged** back together
  - This helps team members work independently **without interfering with each other**

Why use Git?

Why use Git?

- In 2025, why not use something like Google docs (**synchronous editing**)?
  - With code, individual changes can have **unintended consequences** (e.g. you might introduce a bug that isn't discovered until much later)
  - With synchronous editing, it is **hard to identify the source** of a problem
  - Systems like Google Docs **don't support branching versions**

Why use Git?

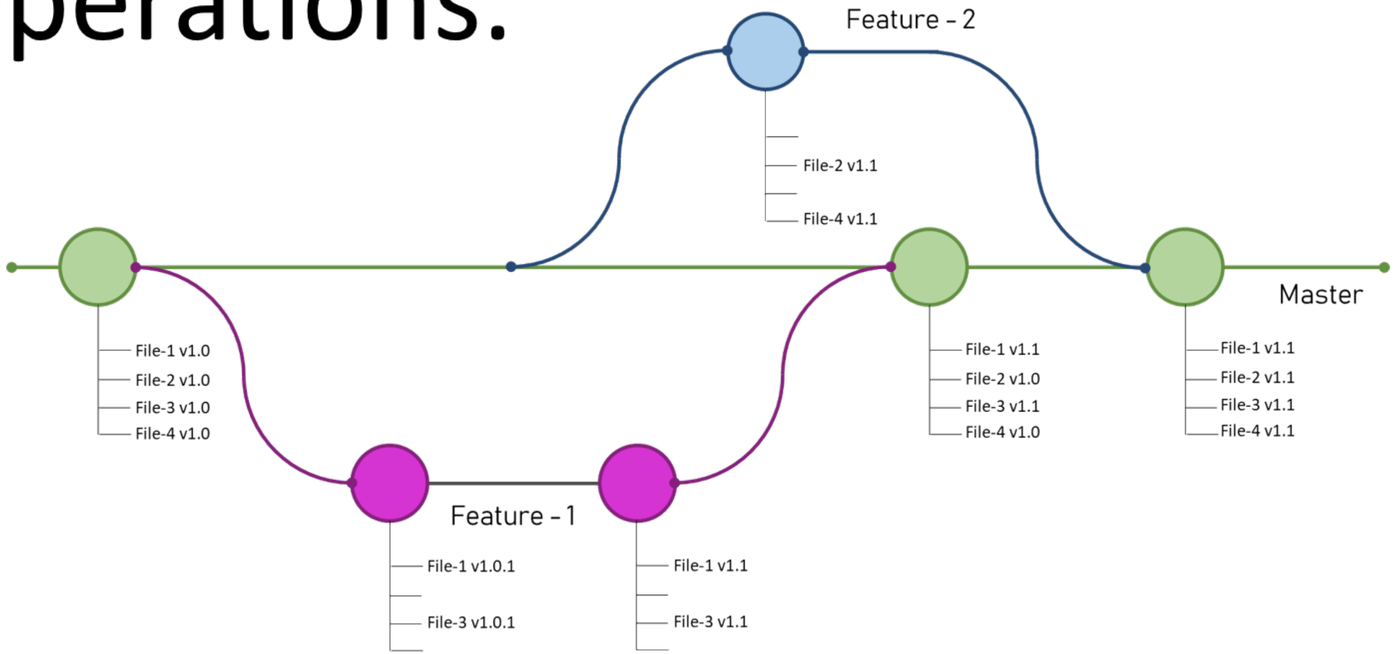
- In 2025, why not use something like Google docs (**synchronous editing**)?
  - With code, individual changes can have **unintended consequences** (e.g. you might introduce a bug that isn't discovered until much later)
  - With synchronous editing, it is **hard to identify the source** of a problem
  - Systems like Google Docs **don't support branching versions**
- Git gives you a **structured history** of the project
  - If you encounter a problem, it is easy to **rewind the project** to a certain point
  - It is also easy to **revert some changes but not others**

Why use Git?

- In 2025, why not use something like Google docs (**synchronous editing**)?
  - With code, individual changes can have **unintended consequences** (e.g. you might introduce a bug that isn't discovered until much later)
  - With synchronous editing, it is **hard to identify the source** of a problem
  - Systems like Google Docs **don't support branching versions**
- Git gives you a **structured history** of the project
  - If you encounter a problem, it is easy to **rewind the project** to a certain point
  - It is also easy to **revert some changes but not others**
- It is the **de-facto standard** for working with code (across industry and academia)

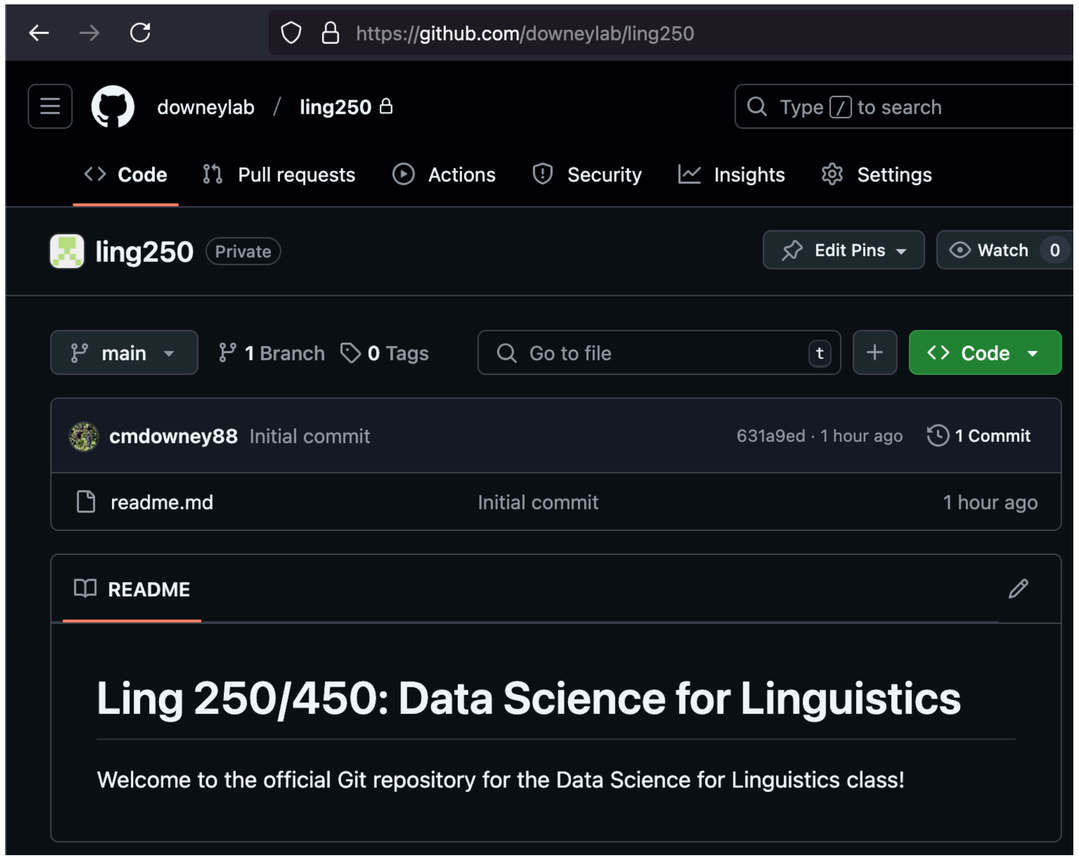
Example branching structure

GIT Branch and its Operations.



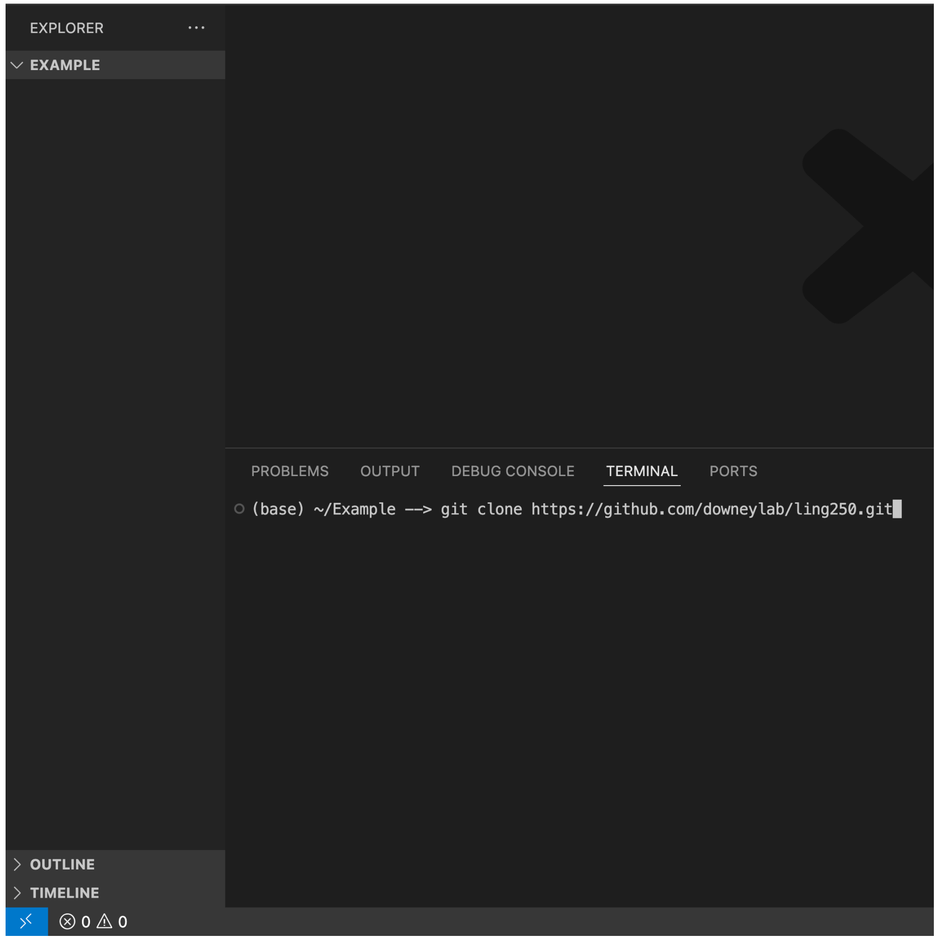
Repositories

- A repository is simply a **folder of files** that are **tracked by Git**
- A repository can be thought of as a **project**
- Repositories are often made **accessible on Github**
  - Github often serves as the **"central copy"** of the repository
- This class has an **example repository**



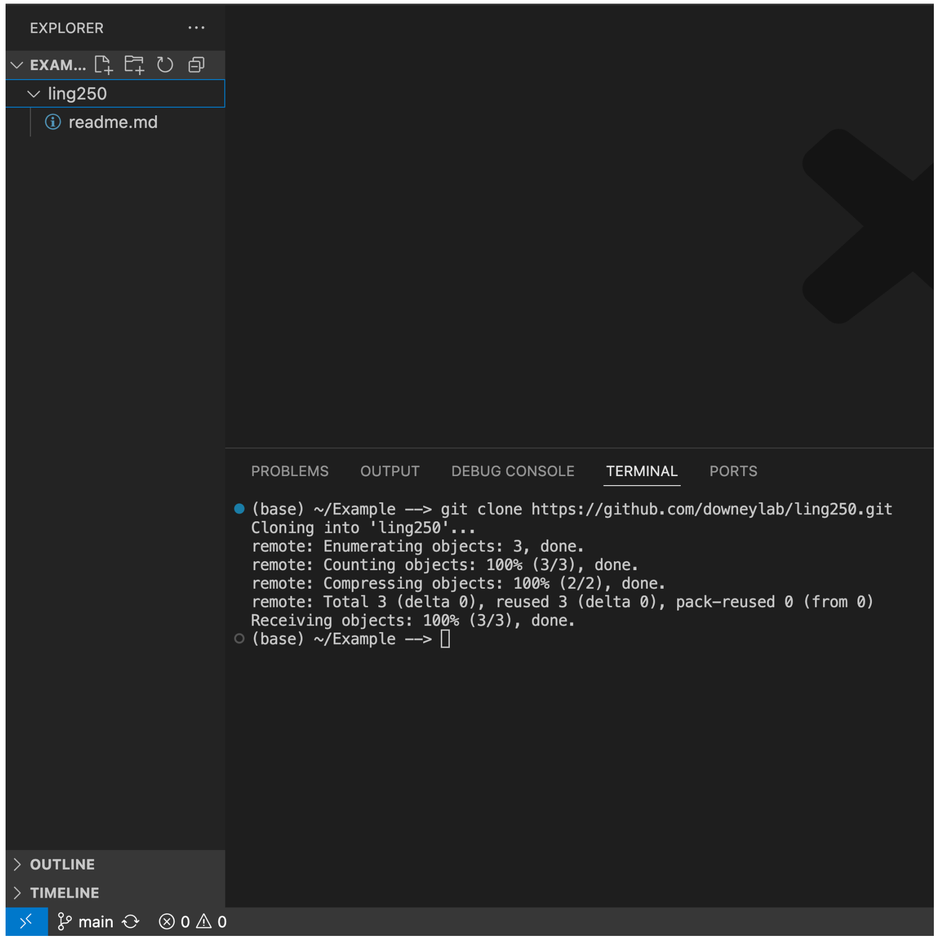
Cloning a repository

- If you have access, you can **clone** a repository to your computer
  - Done with `git clone <repo_url>`
  - Cloning just means **downloading a copy** to work on
- The cloned copy has the **full history** of the repository



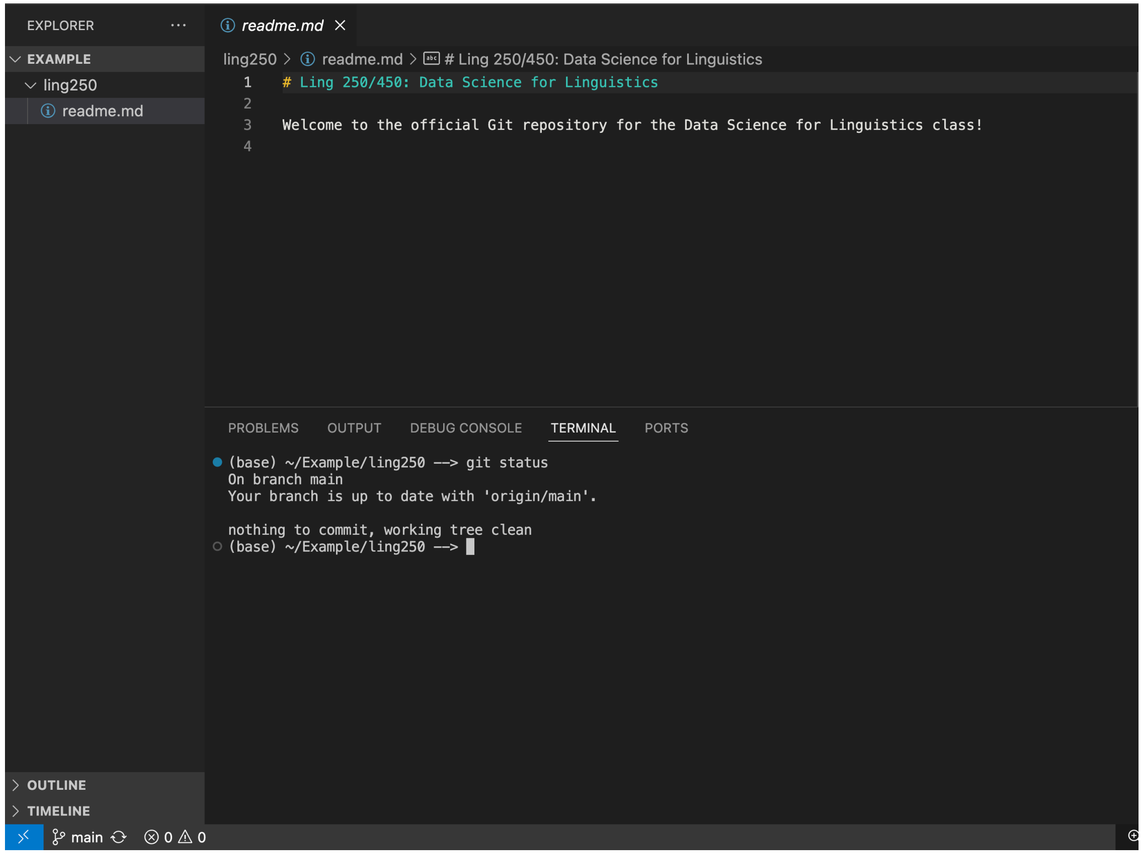
Cloning a repository

- If you have access, you can **clone** a repository to your computer
  - Done with `git clone <repo_url>`
  - Cloning just means **downloading a copy** to work on
- The cloned copy has the **full history** of the repository



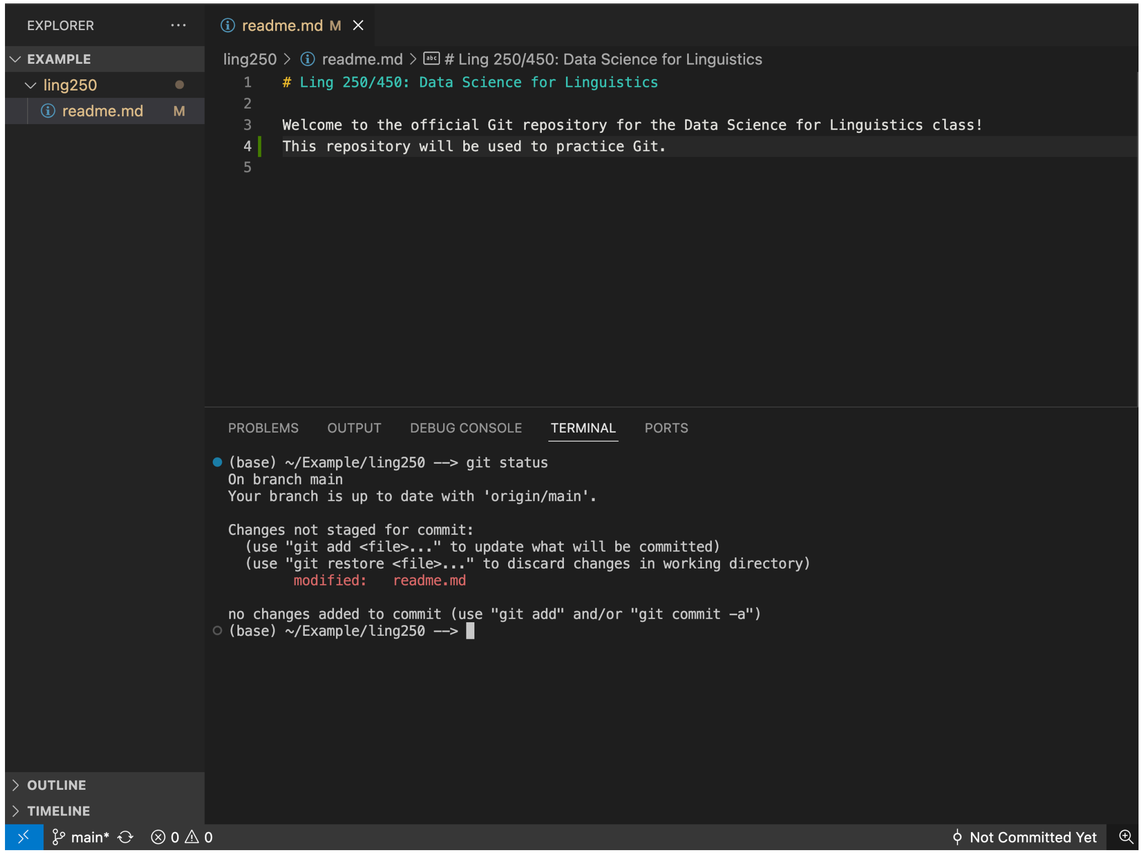
Editing and checking status

- `git status` can be used to **check the status** of the repository
  - Before you change anything, it will say **"working tree clean"**
- You can **freely edit files** (and even press the save button), but it is **not yet saved with Git**
  - Git status will call these **"changes not added to commit"**

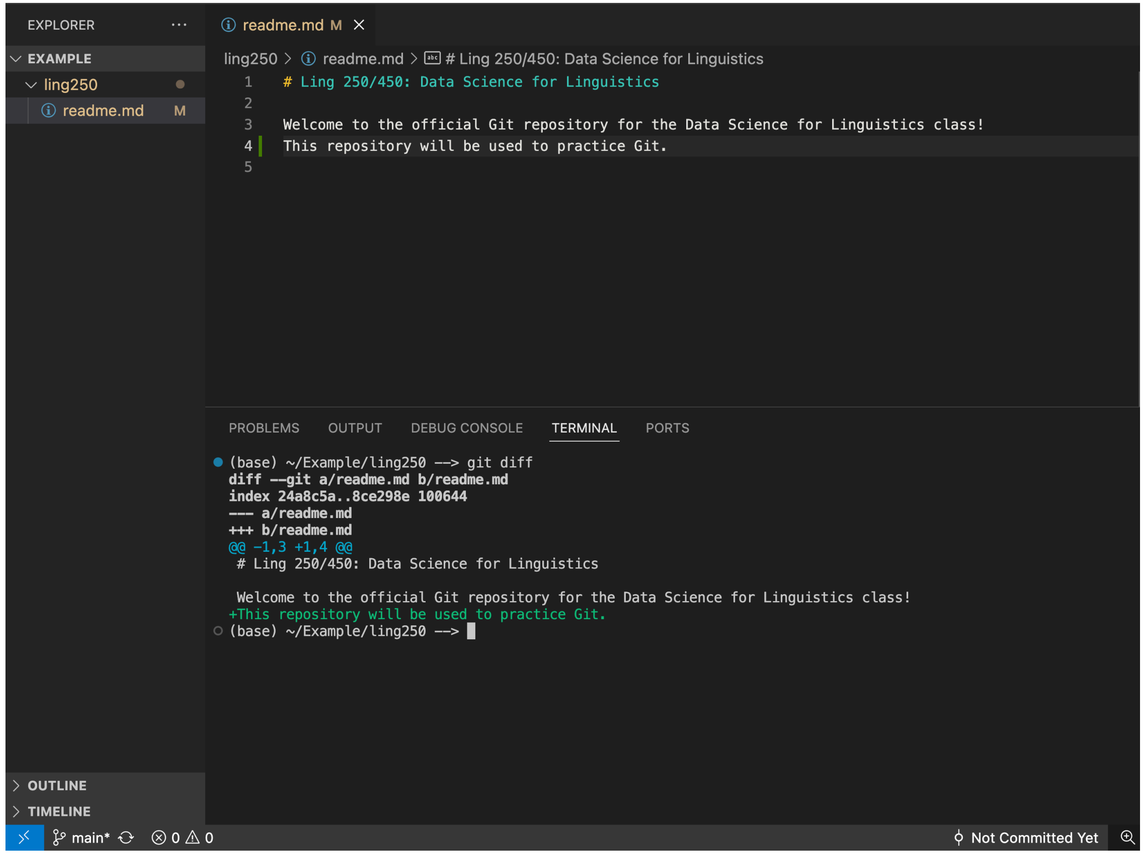


Editing and checking status

- `git status` can be used to **check the status** of the repository
  - Before you change anything, it will say **"working tree clean"**
- You can **freely edit files** (and even press the save button), but it is **not yet saved with Git**
  - Git status will call these **"changes not added to commit"**

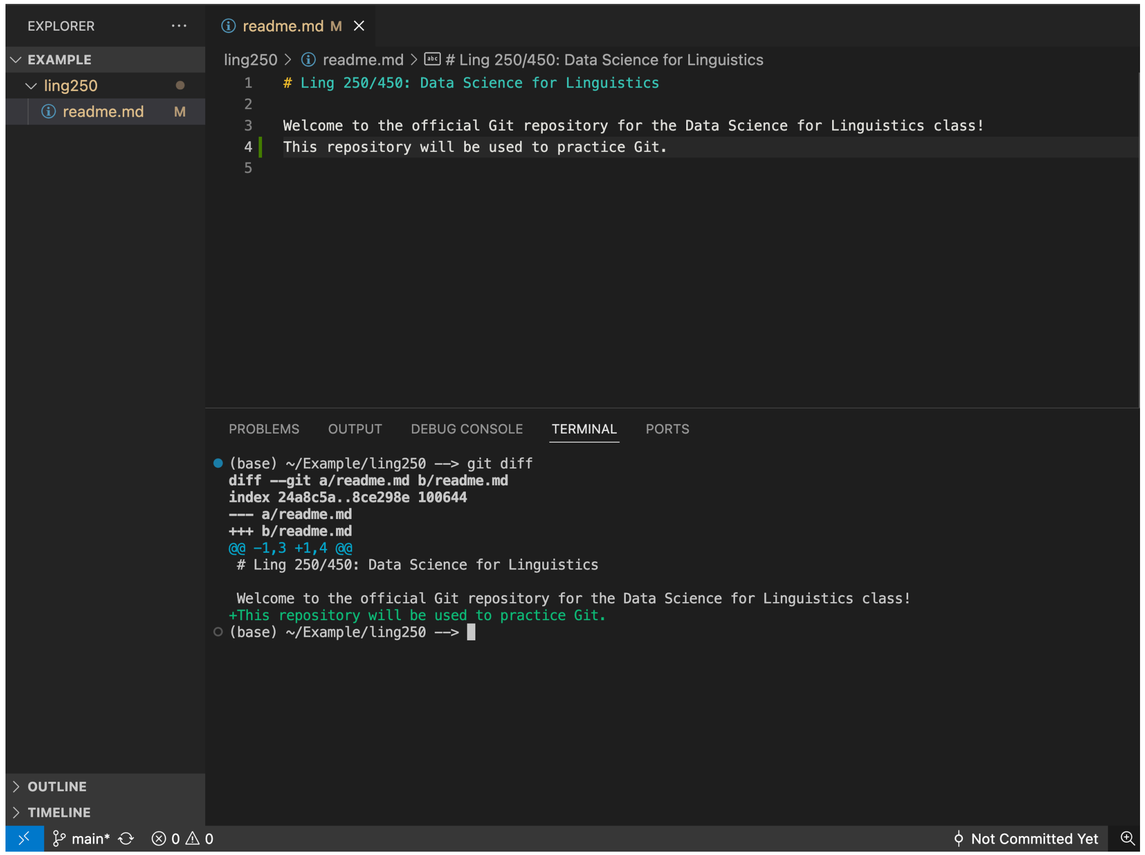


git add


EXPLORER
readme.md M
EXAMPLE
ling250
readme.md M
ling250 > readme.md > # Ling 250/450: Data Science for Linguistics
1 # Ling 250/450: Data Science for Linguistics
2
3 Welcome to the official Git repository for the Data Science for Linguistics class!
4 This repository will be used to practice Git.
5
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) ~/Example/ling250 --> git diff
diff --git a/readme.md b/readme.md
index 24a8c5a..8ce298e 100644
--- a/readme.md
+++ b/readme.md
@@ -1,3 +1,4 @@
# Ling 250/450: Data Science for Linguistics
Welcome to the official Git repository for the Data Science for Linguistics class!
+This repository will be used to practice Git.
(base) ~/Example/ling250 -->
OUTLINE
TIMELINE
main*
0 0
Not Committed Yet

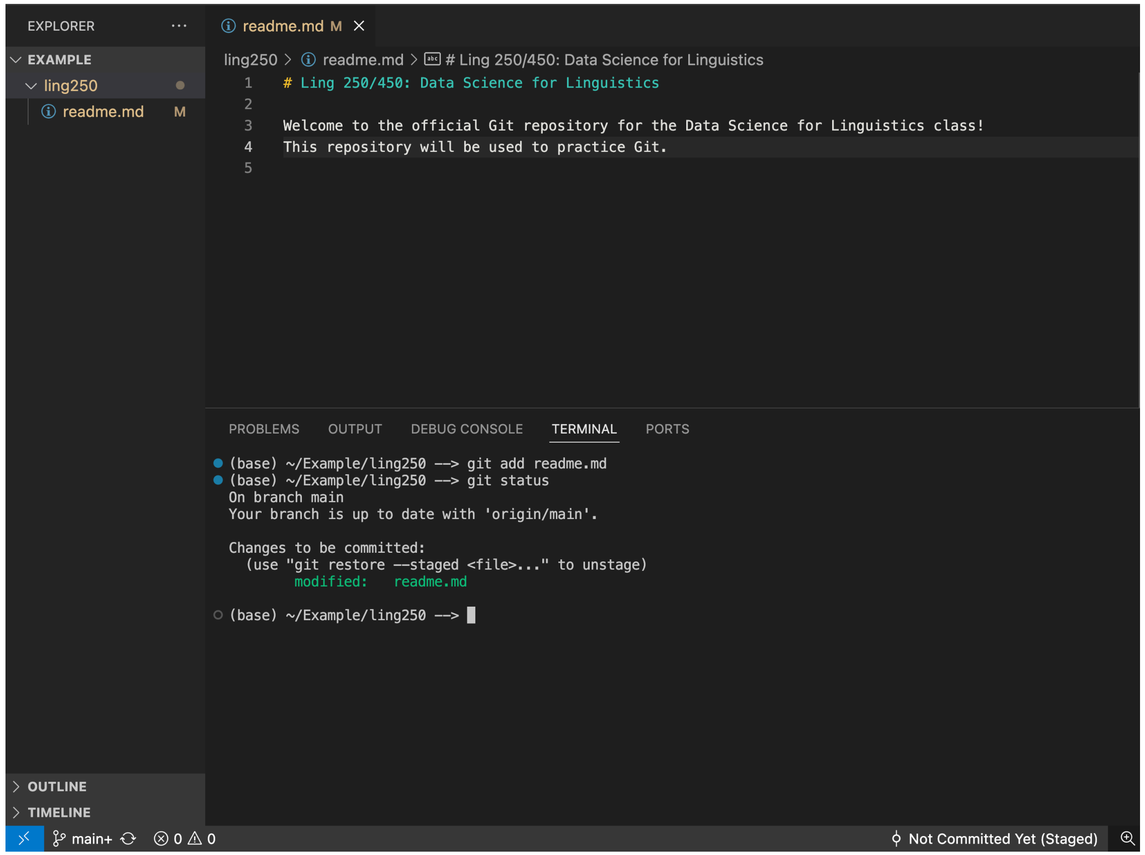
git add

- `git diff` can be used to view **specific changes** from the **last saved version** (called the last **commit**)



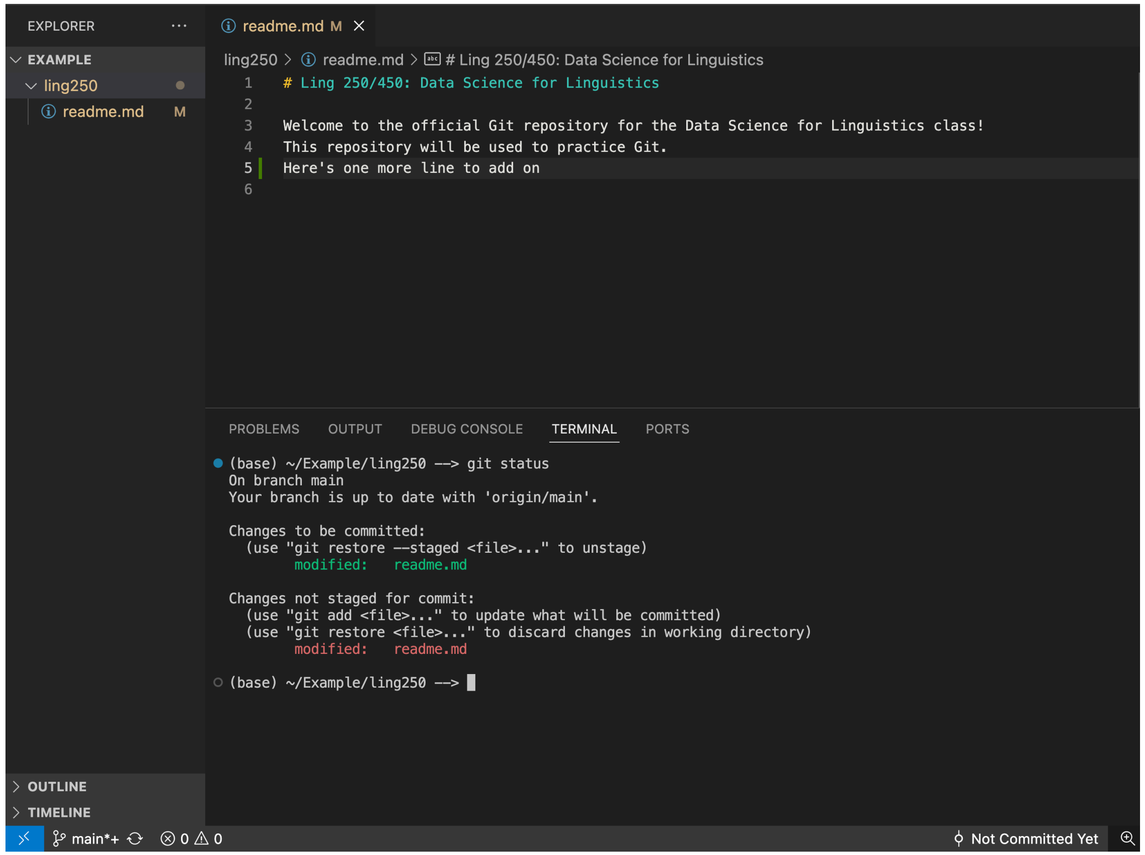
git add

- `git diff` can be used to view **specific changes** from the **last saved version** (called the last **commit**)
- If you're happy with the changes, `git add` will **"stage" the file** for committing
  - This essentially **"marks" the current version** of the file to be saved

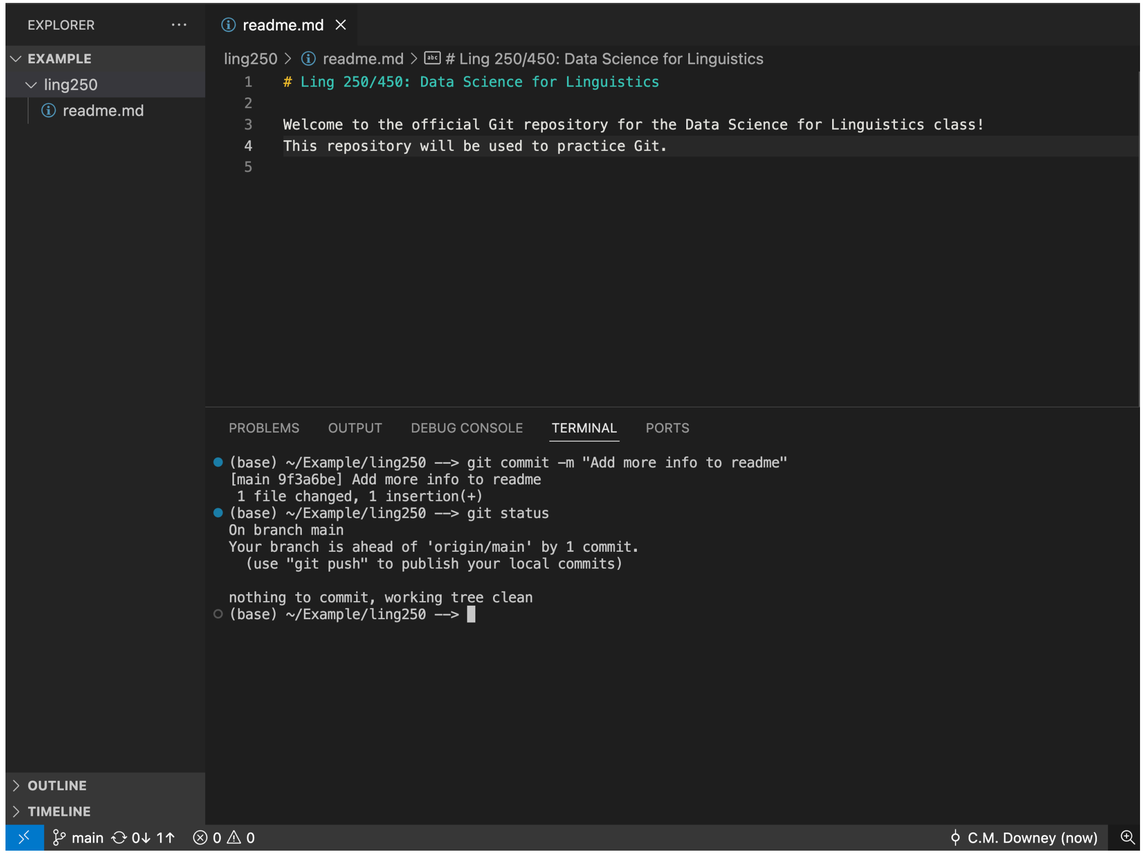


git add

- `git diff` can be used to view **specific changes** from the **last saved version** (called the last **commit**)
- If you're happy with the changes, `git add` will **"stage" the file** for committing
  - This essentially **"marks" the current version** of the file to be saved
- Changes made **after** `git add` are **not** added automatically!

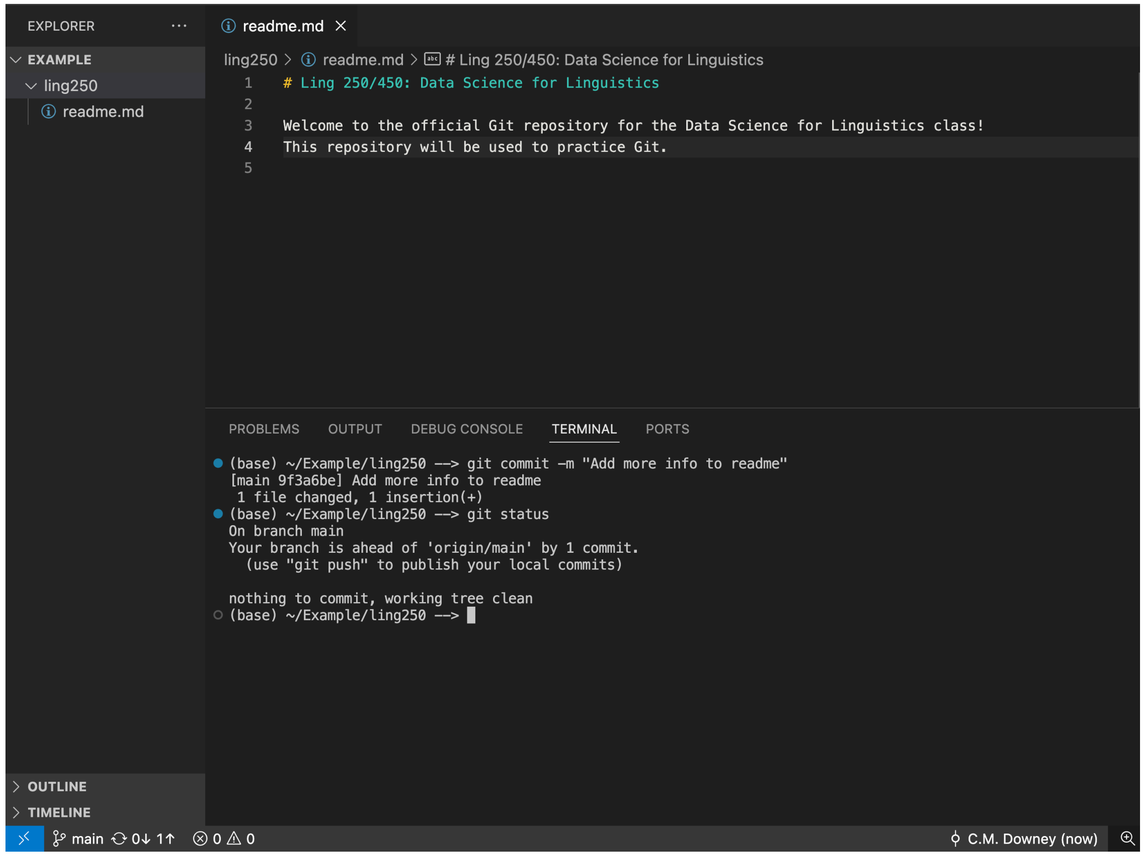


git commit


EXPLORER
EXAMPLE
ling250
readme.md
readme.md
ling250 > readme.md > # Ling 250/450: Data Science for Linguistics
1 # Ling 250/450: Data Science for Linguistics
2
3 Welcome to the official Git repository for the Data Science for Linguistics class!
4 This repository will be used to practice Git.
5
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
(base) ~/Example/ling250 --> git commit -m "Add more info to readme"
[main 9f3a6be] Add more info to readme
1 file changed, 1 insertion(+)
(base) ~/Example/ling250 --> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
nothing to commit, working tree clean
(base) ~/Example/ling250 -->
OUTLINE
TIMELINE
main 0↓ 1↑
0 0
C.M. Downey (now)

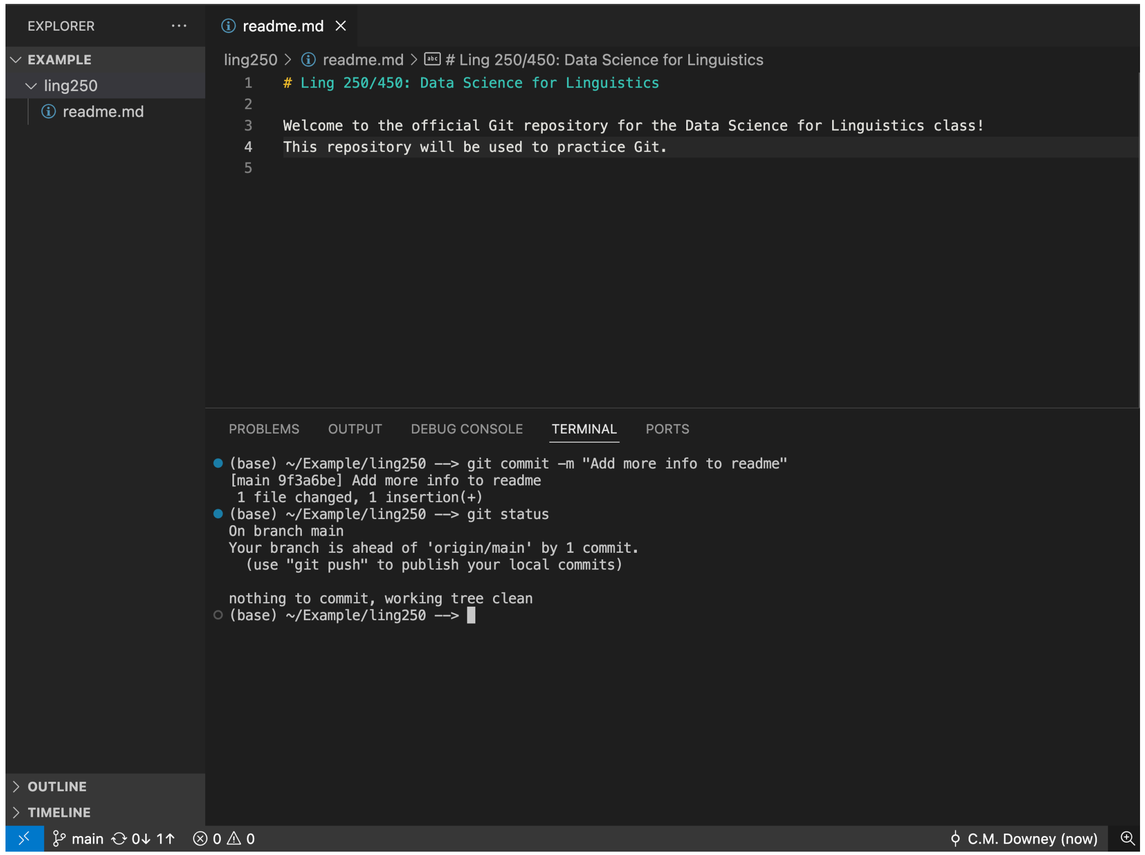
git commit

- git commit **saves** your staged changes as a **reference-able point in history**



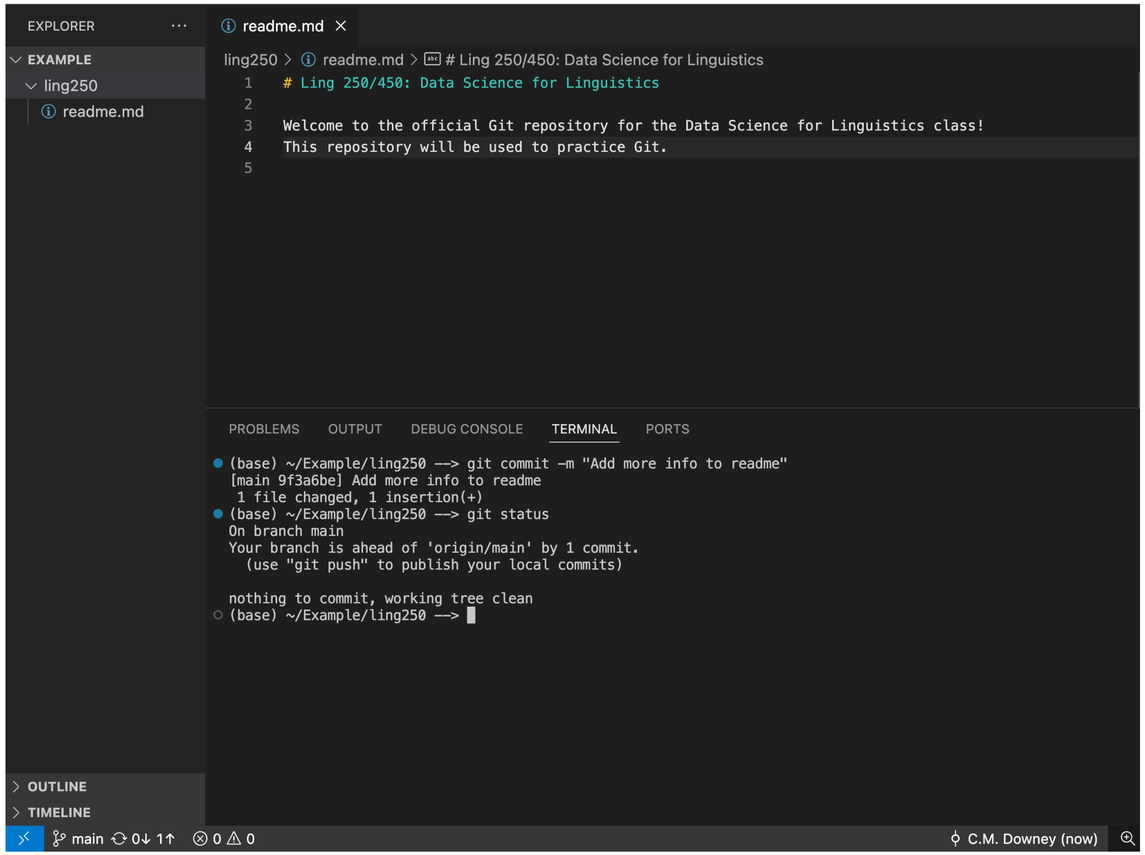
git commit

- git commit **saves** your staged changes as a **reference-able point in history**
- I recommend using `git commit -m` to **add a description** of the changes
  - If you don't use -m, it will **take to you a text editor called Vim** to write a message anyway. To **escape Vim**, type :cq!



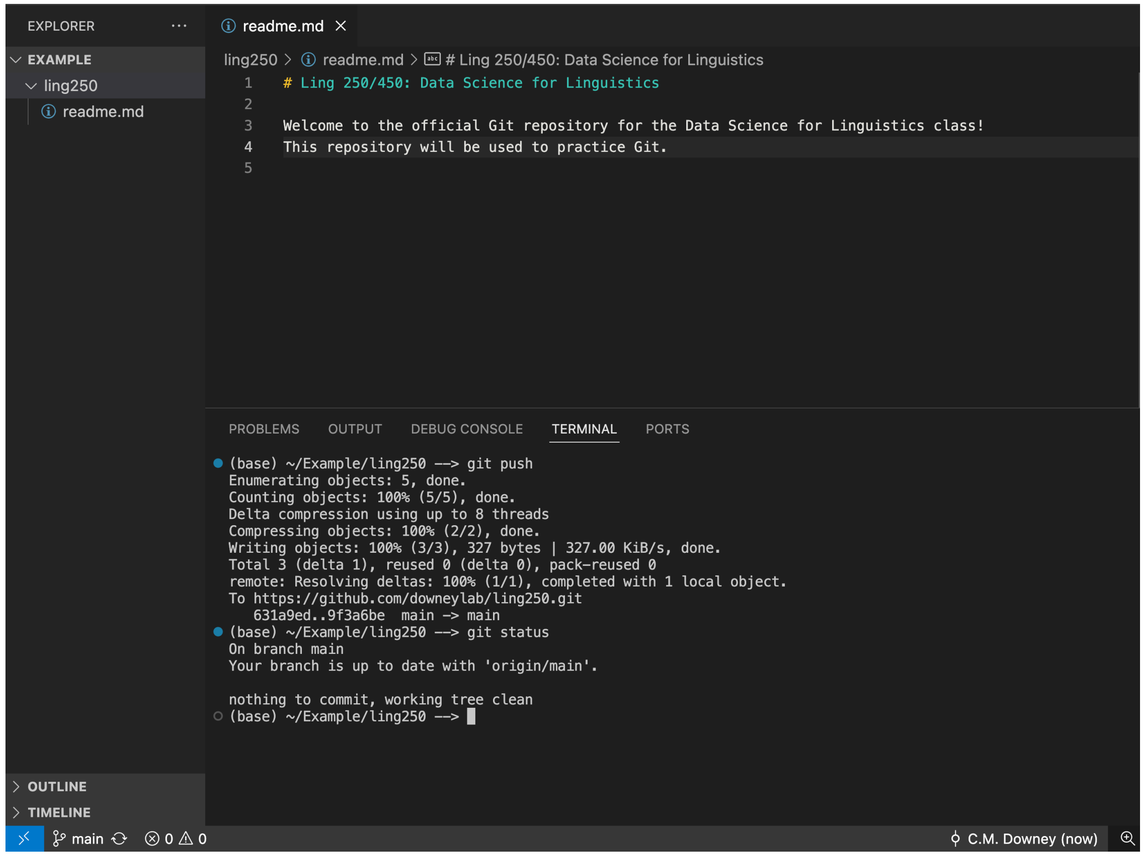
git commit

- git commit **saves** your staged changes as a **reference-able point in history**
- I recommend using `git commit -m` to **add a description** of the changes
  - If you don't use -m, it will **take to you a text editor called Vim** to write a message anyway. To **escape Vim**, type :cq!
- Notice it now says our branch is **"ahead by 1 commit"**. This is compared to the **online (Github) version**



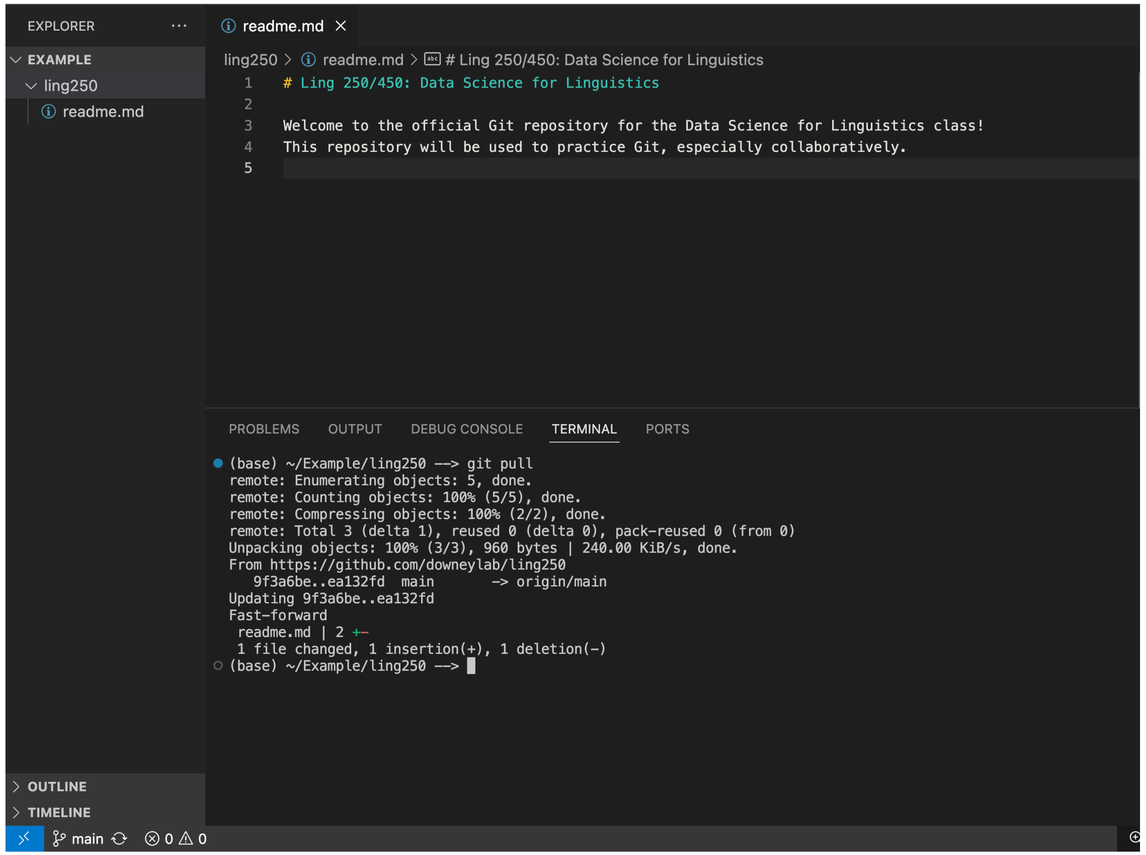
git push

- `git push` **publishes your commits** to the online repository
  - i.e. it makes them available for others
  - The online repository is often called the **"upstream"** version or `origin`
- If the upstream repository has **someone else's commits**, it will **block you** from pushing until you merge those changes with yours (more later)

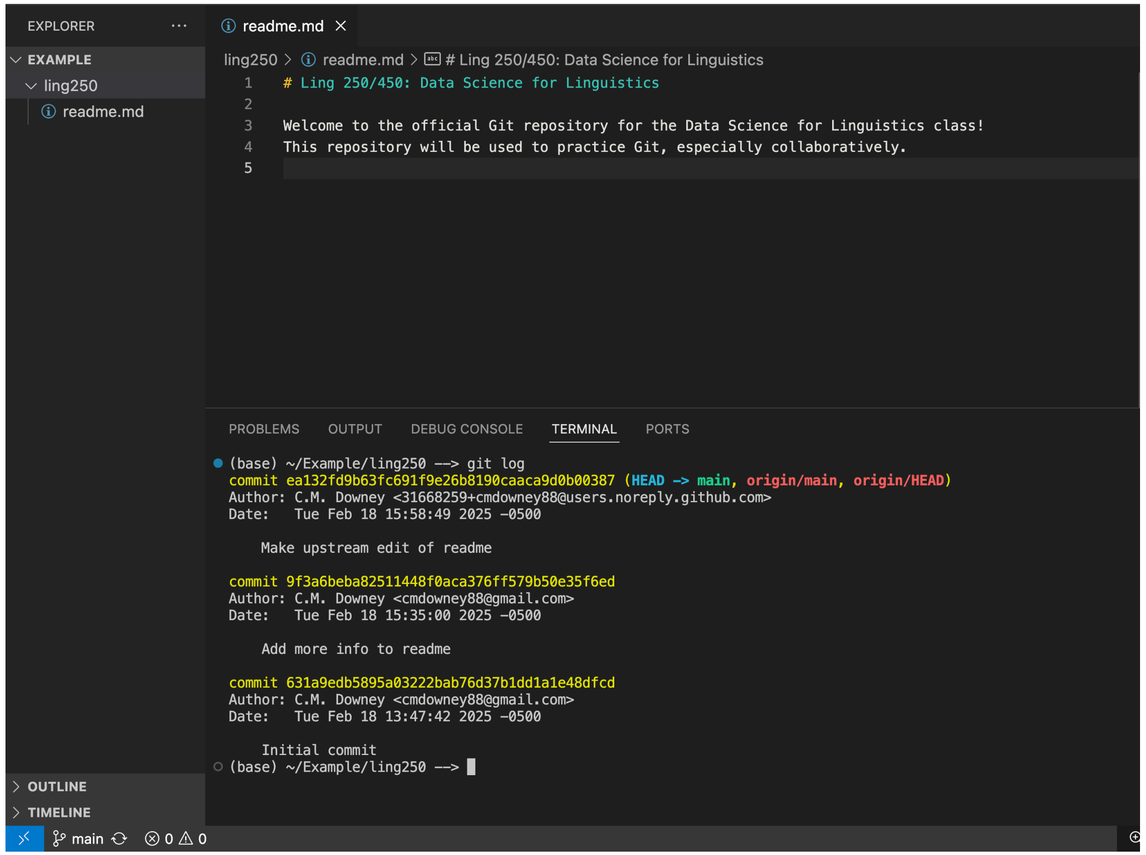


git pull

- `git pull` **downloads new commits** from the upstream repository
- Good practice: always **pull before you start new work!**
  - It's easier to work on top of the **most current version** rather than writing **potentially conflicting commits**
  - A **rule of thumb** could be to pull whenever `git status` says **"working tree clean"**

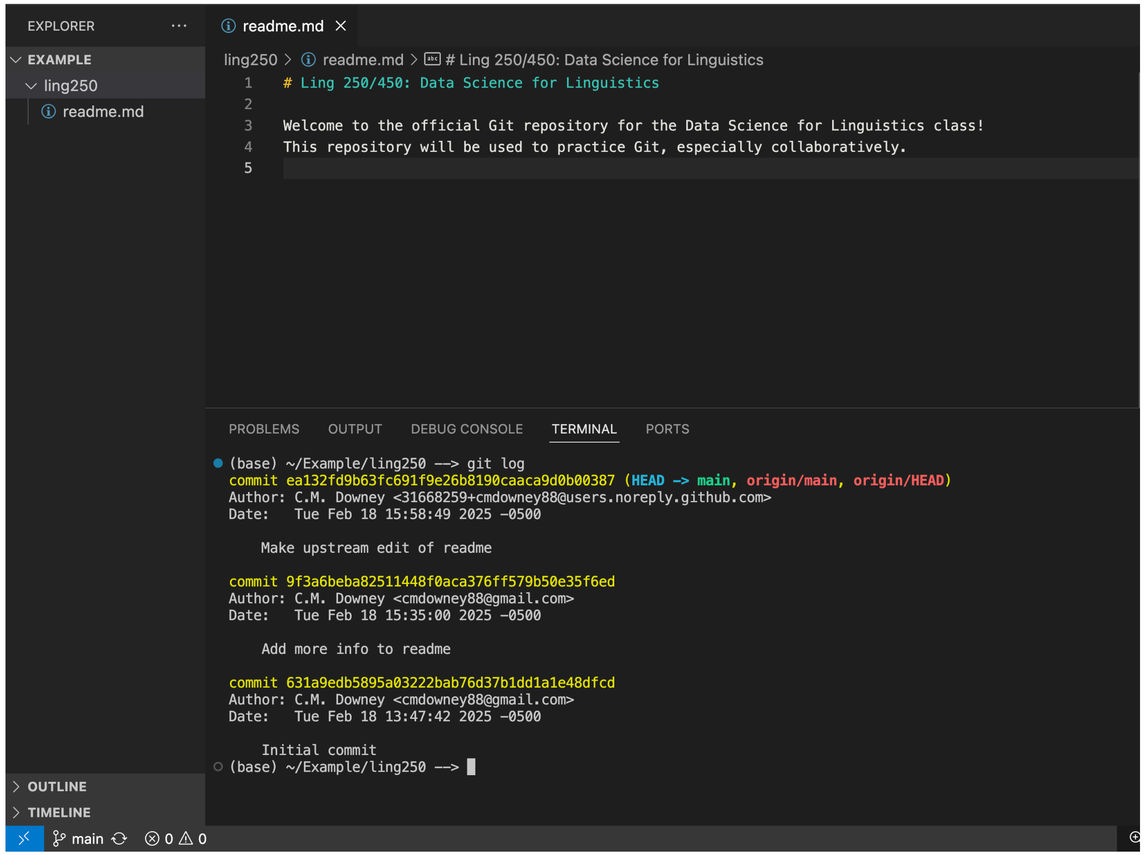


Viewing commit history



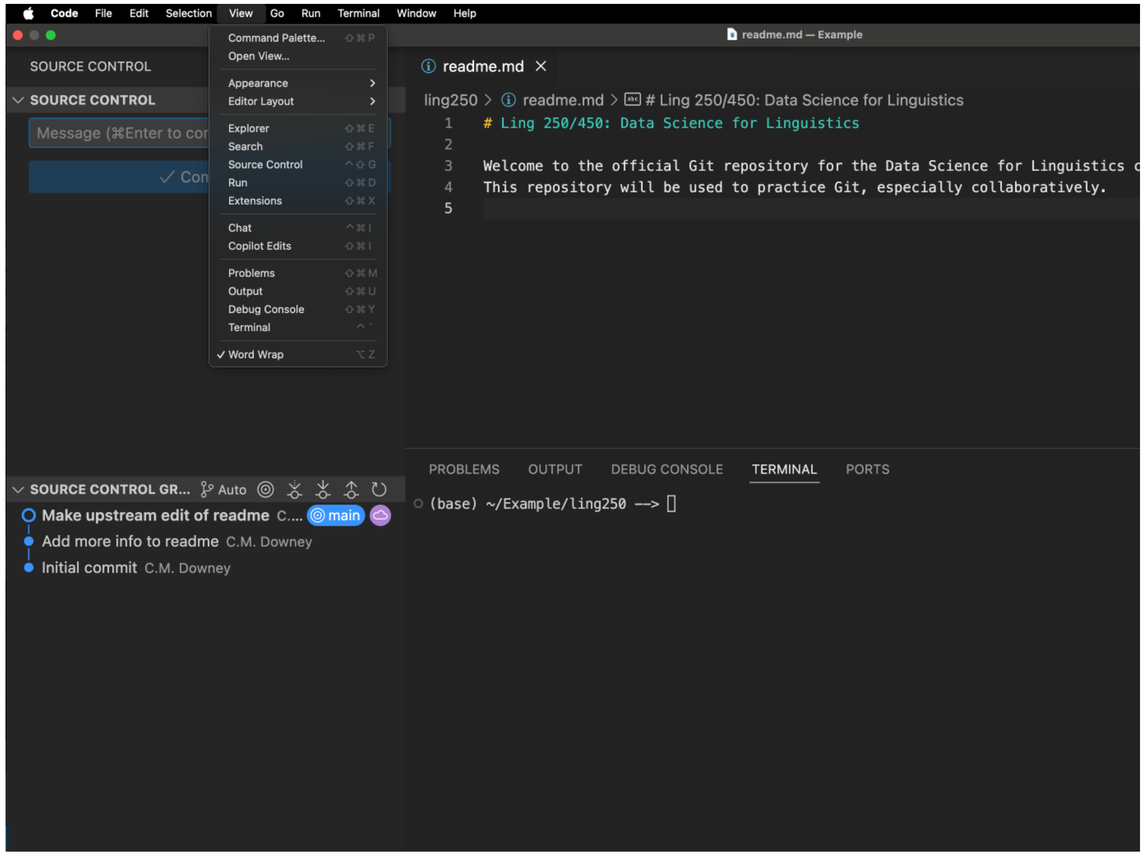
Viewing commit history

- `git log` will list the **commit history** in the **command line**



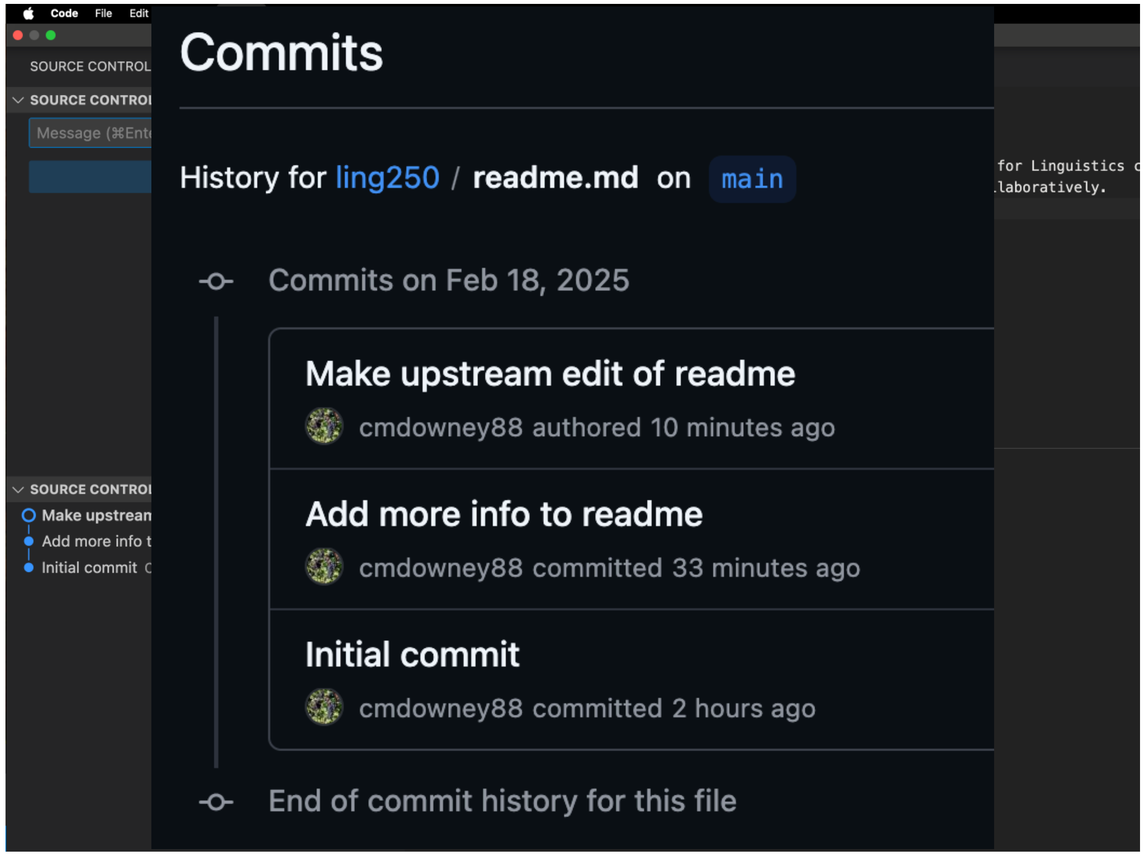
Viewing commit history

- `git log` will list the **commit history** in the **command line**
- Clicking View > Source Control in VSCode will open a **timeline view**
  - **Warning**: this will also open some **graphical interface buttons** for Git, which I **recommend against** when you're first learning Git



Viewing commit history

- `git log` will list the **commit history** in the **command line**
- Clicking View > Source Control in VSCode will open a **timeline view**
  - **Warning**: this will also open some **graphical interface buttons** for Git, which I **recommend against** when you're first learning Git
- Commit history can also be **viewed on Github**



General tips for committing

General tips for committing

- **Commit often!!**
  - "Commit" sounds serious, but think of it as **saving your work**
  - A **common pitfall** is to try to make every commit refined
  - Committing often allows Git to **reverse buggy changes** without losing other work

General tips for committing

- **Commit often!!**
  - "Commit" sounds serious, but think of it as **saving your work**
  - A **common pitfall** is to try to make every commit refined
  - Committing often allows Git to **reverse buggy changes** without losing other work
- Break changes to **separate files** into **separate commits**
  - (unless the changes are **closely tied**)
  - You can use `git add` to only add one file to be committed at a time

Commit messages

Commit messages

- Make your commit messages **informative!!**
  - Uninformative messages are a **bad habit**. Some common examples: "debugged", "fixes", "changes", "latest version", "update"
  - The message should **concisely** but **concretely** indicate **what was changed**
  - This is another reason to prefer **smaller, frequent commits**

Commit messages

- Make your commit messages **informative!!**
  - Uninformative messages are a **bad habit**. Some common examples: "debugged", "fixes", "changes", "latest version", "update"
  - The message should **concisely** but **concretely** indicate **what was changed**
  - This is another reason to prefer **smaller, frequent commits**
- Possible exception: solving **mysterious bugs**
  - I.e. "I have no idea what the problem was but this solved it"
  - The message should still indicate **what part** of the code was debugged