Git Basics

Version control for your project, from Zed

Why version control

When you write code over several days, you keep changing files. Without a record, you cannot tell what changed, when, or why, and you cannot go back to a version that worked. **Version control** is a tool that saves snapshots of your project so you always have a history you can return to. In this course we use **git**, the standard tool for this, and **GitHub**, a website that stores a copy of your project online so a partner can see it too.

You do not need to understand how git works inside. You need five simple actions, and you can do all of them from **Zed**, the editor you set up before Session X. Zed has a **git panel** (a side panel that shows your changes) and, like everything in Zed, git actions are available from the **command palette** (`Cmd+Shift+P` on macOS, `Ctrl+Shift+P` on Windows/Linux). This page uses the command-palette names: they always work, whatever the panel looks like. We show the matching **terminal command** next to each action as well, because the words are the same everywhere, and you will see those commands in videos, tutorials, and error messages. Learn the word once and it works in both places.

> **i** Note
>
> This page assumes you already have **uv** and **Zed** installed from the Session IX homework. If not, do Installing Python first.

The five operations

Below are the only five git operations you need for this course. Each one has a plain definition, the way to do it in Zed, the matching terminal command, and when you actually need it.

Repository

A **repository** (or "repo") is a folder that git watches. Everything inside it, and its whole history, is tracked together.

You almost never create one by hand in this course. When you start a project with `uv init`, **uv also turns that folder into a git repository for you**, as long as git is already installed (which step 1 of the one-time setup below handles first). So after this command you already have a repo:

```
uv init my-project
```

The only time you make a repo yourself is if you have an existing folder that is *not* yet under git. In Zed you run **`git: init`** from the command palette. The terminal equivalent is:

```
git init
```

When you need it: almost never on its own. `uv init` covers the normal case. Use `git init` only to add git to an old folder that does not have it yet, or if you ran `uv init` *before* installing git, in which case run `git init` once inside that folder to add the repo it skipped.

Commit

A **commit** is one saved snapshot of your project, with a short message describing what you changed. Think of it as a labeled save point you can always come back to.

A commit has two small steps: first you **stage** the changes you want to include (mark them as "part of this snapshot"), then you commit them with a message.

In Zed you run **`git: stage all`** from the command palette, type a message, and commit with **`git: commit`** (`Cmd+Enter` on macOS, `Ctrl+Enter` on Windows/Linux). The terminal equivalent is two commands:

```
git add .
git commit -m "Add price calculation"
```

`git add .` stages every changed file; `git commit -m "..."` saves the snapshot with your message.

When you need it: every time you finish a meaningful piece of work. Commit often, with short honest messages.

Push

Push sends your commits from your computer up to GitHub, so the online copy matches your local one.

In Zed you run **`git: push`** from the command palette.

> ⚠ Warning
>
> **The very first push of a brand-new project is different.** Run this exact terminal command once, from inside your project folder:
>
> ```
> git push -u origin HEAD
> ```
>
> The `-u` connects your local project to the GitHub copy, and `HEAD` means "push whatever branch I am on" (so it works no matter what your branch is called). After this one-time command, every later push can be the normal **`git: push`** action in Zed. If you *cloned* a partner's repo instead of starting your own, you do not need this step, cloning already sets up the connection.

The everyday terminal equivalent, after that first push, is:

```
git push
```

When you need it: whenever you stop working, so your partner (and a safe online backup) has your latest commits.

Pull

Pull brings commits *down* from GitHub into your computer, so you get whatever your partner pushed.

In Zed you run **`git: pull`** from the command palette. The terminal equivalent is:

```
git pull
```

When you need it: every time *before* you start working. This is the single most important habit when you share a repo. Pull first, and you almost never run into trouble.

Clone

Clone makes a full copy of a repository from GitHub onto your computer, including all its history. This is how you get onto your partner's project the first time.

In Zed, open the command palette and run **`git: clone`**, then paste the repository's web address (its HTTPS URL, which looks like `https://github.com/name/project.git`). The terminal equivalent is:

```
git clone https://github.com/name/project.git
```

When you need it: once, when you join a project someone else created. After cloning you have the repo locally and can commit, push, and pull like normal.

GitHub setup and authentication

Before any push or pull can reach GitHub, your computer has to prove it is allowed to (this is **authentication**). Signing in to the Zed app itself does **not** do this, it only unlocks Zed's own collaboration and AI features. Git needs its own separate setup, and the reliable way to do it is the **GitHub CLI**, a small terminal tool called `gh`. You do this **once per computer**.

> ⚠ Warning
>
> Signing in to Zed is not the same as authenticating git. Even fully signed in to Zed, a push will fail until you complete the `gh` steps below. If a push ever asks for a password or reports a permission or authentication error, the fix is to redo these steps, not to sign in to Zed again.

One-time setup

1. **Install git itself.** Check with `git --version` in a terminal.
   - macOS: if git is missing, that command offers to install it. Accept.
   - Windows: `winget install Git.Git` (or the installer from git-scm.com).

   Then re-run `git --version` to confirm you see a version number.
2. **Install `gh`** if you do not have it. Find the installer for your system at cli.github.com. Installing `gh` does **not** install git itself. That was step 1.
   - macOS: `brew install gh`
   - Windows: `winget install GitHub.cli`
3. **Log in.** Run this in a terminal:

   ```
   gh auth login
   ```

   Answer the prompts: choose **GitHub.com**, choose **HTTPS** as the protocol, and authenticate through the **browser** (it shows you a short code to type into a GitHub page). When it asks *"Authenticate Git with your GitHub credentials?"*, answer **yes**. This runs `gh auth setup-git` for you, which is the step that lets git push and pull. (If you answered no by mistake, you can run it yourself afterwards:)

   ```
   gh auth setup-git
   ```
4. **Check it worked:**

   ```
   gh auth status
   ```

   It should report that you are logged in to github.com and that git operations use the https protocol.
5. **Tell git how to combine work when you pull:**

```
git config --global pull.rebase false
```

This tells git to simply combine your partner's work with yours when you pull. You only set this once. Without it, git may refuse a pull with a confusing message about "divergent branches".

> **i** Note
>
> Signing in to the Zed app (`client: sign in` in the command palette) is optional and only for Zed's collaboration and AI features. It has nothing to do with git push, so do not treat it as part of git setup.

Connecting a new project to GitHub

When *you* create the project (with `uv init`), GitHub does not know about it yet. Three small steps connect them:

1. On github.com, create a new **empty** repository (do not add a README, so it stays empty).
2. In Zed, run **`git: create remote`** from the command palette and paste the repository's **HTTPS URL** (`https://github.com/<you>/<repo>.git`). Use the HTTPS URL, not the SSH one, the login you set up above only works over HTTPS. If Zed asks for a remote name, use `origin`. (The terminal equivalent that guarantees the name is `git remote add origin <HTTPS url>`.)
3. Do the one-time **first push** from the terminal:

```
git push -u origin HEAD
```

After this, everyday pushing is just **`git: push`** in Zed.

Working as a pair

In this course you work in pairs, and you share one GitHub repository. There are two clear roles.

The repo owner creates the project and invites the partner:

1. Create the project (`uv init`), connect it to GitHub, and do the first push (the three steps above).
2. On GitHub, open the repository's **Settings** → **Collaborators** and invite your partner by their GitHub username. This gives them permission to push.

The partner joins the existing project by cloning it:

1. Accept the invitation (GitHub sends it by email or shows it on the site).
2. In Zed, run **`git: clone`** with the repository's HTTPS URL (terminal: `git clone <url>`).

From then on **both of you** follow the same two habits every session:

- **Pull before you start.** Run `git: pull` so you have your partner's latest work.
- **Push when you stop.** Commit, then `git: push`, so your partner gets your work.

What a conflict looks like

If you and your partner both change **the same lines** of the same file, git cannot decide which version wins. This is a **conflict**. Git marks the spot inside the file with strange-looking lines (`<<<<<<<`, `=======`, `>>>>>>>`) showing both versions.

> ⚠ Warning
>
> **Do not try to fix a conflict alone in week one.** Bring it to class or office hours, we will resolve it together. The one habit that prevents almost all conflicts: **pull before you start working**, so you build on your partner's latest version instead of an old one.

Troubleshooting

Each entry gives the **symptom** first, then the **fix**.

Symptom: a push is rejected with `! [rejected]`, "fetch first", or "the remote contains work that you do not have locally". Fix: your partner pushed commits that you do not have yet. This is completely **normal in a pair** and is *not* an authentication problem. Pull first (`git: pull` or `git pull`), then push again. If the pull complains about "divergent branches", run the one-time command from the setup section (`git config --global pull.rebase false`), then pull again. If the pull instead reports a **conflict**, see What a conflict looks like. That pull is exactly when one can appear. Never force anything: no `--force`, no "force push".

Symptom: you committed, but your partner still does not see your changes. Fix: a commit only saves the snapshot **on your computer**. It reaches GitHub (and your partner) when you push. Run `git: push`.

Symptom: a push asks for a username/password, or reports a permission or authentication error. Fix: your machine's git login is not set up (or expired). Redo the GitHub setup above: run `gh auth login` again and answer **yes** to "Authenticate Git with your GitHub credentials?". Do **not** sign in to the Zed app to fix this, that is a different thing.

Symptom: a git command says `fatal: not a git repository`. Fix: usually you are running git in a folder that is not a repo (or above your project folder). Move into your project folder first. In a terminal, `cd my-project`. In Zed, make sure the folder you opened is the project folder that contains your code. The other cause: you created the folder with `uv init` *before* git was installed, so uv could not make the repo. Fix that by running `git init` once inside the project folder.

Symptom: you committed the wrong thing (bad message, or forgot a file). Fix: never rewrite history that others may already have. Two safe options:

- **Only if you have not pushed that commit yet** (it is still just on your computer, so nobody has seen it), you can fix the last commit in place. Stage the correction and amend:

```
git add .
git commit --amend -m "Better message"
```

- If you already pushed it (or you are unsure), simplest of all: just make a **new commit** that corrects the mistake. A tidy extra commit is completely fine in this course, and it never puts your shared history at risk.

Cheatsheet

Operation	What it does	In Zed	Terminal	When you need it
Create repo	Makes a folder that git tracks	`uv init` (auto), or `git: init`	`uv init <name>` / `git init`	Start of a new project (uv does it for you)
Stage + commit	Saves a labeled snapshot	`git: stage all`, then `git: commit`	`git add .` then `git commit -m "msg"`	Every time you finish a piece of work
Push	Sends commits up to GitHub	`git: push`	`git push`	When you stop working
Pull	Brings partner's commits down	`git: pull`	`git pull`	**Before** you start working
Clone	Copies a repo from GitHub	`git: clone`	`git clone <url>`	Once, when you join a partner's project
Status / history	Shows changes and past commits	git panel shows changes	`git status` / `git log --oneline`	Any time, to see where you are

The very first push of a project you created yourself is the one exception: run `git push -u origin HEAD` once in the terminal (see Push).

Recap

You can now:

1. Explain what a repository, a commit, push, pull, and clone are.

2. Set up your machine once: GitHub authentication with `gh auth login`, plus the one-time pull setting.
3. Connect a new `uv init` project to GitHub and do the first push.
4. Work in a pair: pull before you start, commit as you go, push when you stop.
5. Recognize a conflict and know the course rule: bring it to class instead of fixing it alone.