Collaboration and Version Control with Git

“If that doesn’t fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of “It’s really pretty simple, just think of branches as…” and eventually you’ll learn the commands that will fix everything.”

Source: xkcd.com/1597

Scenario 1

You're working on a CS50 problem set (say Finance). You're able to get some portions of the problem set working perfectly (`register`, `quote`, and `buy`) but are confused as to how to complete `index` and display a table of stocks that the user owns.

While attempting to make more progress, you accidentally mess up the working code that you already had, so all you see now are internal server errors everywhere, no matter what you try to do.

In hindsight, what could you have done to prevent this?

Scenario 2

You decide to team up with a friend for your CS50 final project and build a web app. The two of you are trying to decide how to split up tasks: you really want to design the home page using HTML/CSS, including the fonts, styles, colors, text, etc., while your friend really wants to implement the sign up and login workflow using a JavaScript frontend (also on the home page) and Flask backend.

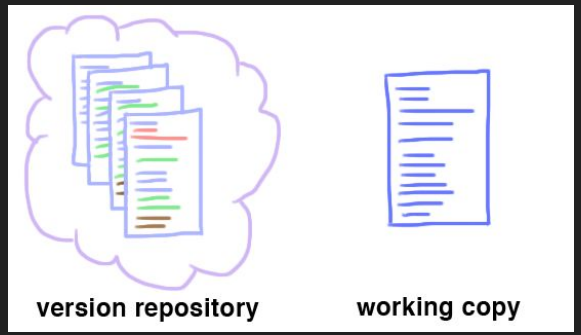
Given that you both want to edit the same file but in different ways, what are some ways to deal with this?

What is Git?

- Git is a command-line version control system.
- Projects are stored as *repositories*.
- Each saved version of the repository is called a *commit*.
- A local repository (on your device) can be associated with other repositories hosted elsewhere (like on GitHub, GitLab, etc.), called *remotes.*

What is Git?

- Each commit stores the changes made since the previous commit, and is identified through a *commit hash*.
- The normal directory structure on your local system along with the files it contains (in contrast to previous versions) is called the *working copy*.
- The set of tracked changes that will be saved in the next commit is called the *staging area*.



Source: https://cs61.seas.harvard.edu/site/ref/git/

Installation

- Windows: https://git-scm.com/download/win
- MacOS: Run `git --version` (should already be installed; if not, will prompt an installation)

Note that you don’t need to install anything if you’re working on Codespaces; this is only necessary if you want to develop locally.

Getting started

`git clone <URL>`

- Creates a local copy of an existing remote repository. Used when you want to view/run/contribute to an existing codebase.

`git init`

- Makes the current directory a Git repository. Used when you’re starting a new codebase.

Saving changes

Simple 3 step process.

`git add <files>`

- Tracks the specified file(s) by adding them to the staging area. The next commit will save changes that have been specified by `git add`.

`git commit -m “Commit message goes here”`

- Saves tracked changes in the form of a commit.

`git push`

- Pushes local commits to the remote repo.

Undoing changes

`git revert <commit>`

- Undo a commit by creating a new commit that does exactly the opposite of what the incorrect commit did
- Useful when you want the incorrect changes stored in history (say if you made some design changes to your website that you didn't like, but you might want to come back and use that code later)
- The commit can be specified either using a commit hash, or by using the `HEAD~n` syntax (`HEAD~n` represents the nth last commit, so `HEAD~1` is the last commit, etc.)

Other useful commands

`git status`

- Displays state of the repo and staging area.

`git log`

- Displays log of previous commits in the currently checked-out branch.

`git diff`

- Displays changes made.

Demo!

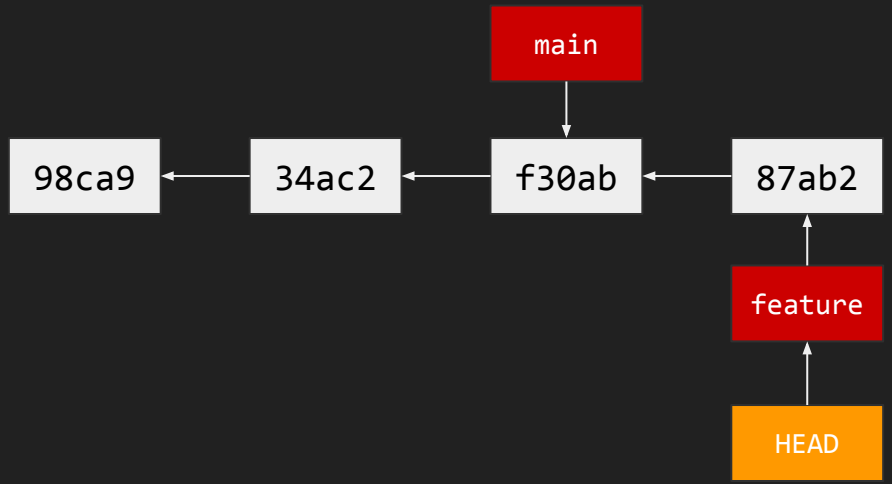
Pulling

```
git pull
```

- This downloads changes/commits that have been pushed by others to the remote and merges them into your own local repository
- Useful when collaborating with others since you won’t be the only one pushing code

Branches

- It’s really pretty simple, just think of branches as…
  - ...pointers to commits!
- Useful for working on new experimental features, fixing bugs, etc. without affecting main working branch, especially when collaborating with others
- Once the feature is complete, can merge into main branch



Branches

- Creating a branch:
  - `git branch <branch-name>`
    - (does not switch to the new branch)
  - `git checkout -b <branch-name>`
    - (switches to the new branch)
- Switching to a branch:
  - `git checkout <branch-name>`
- Pushing a new branch for the first time
  - `git push -u origin <branch-name>`

Pull Requests (PRs)

- Pull requests are *not* a git feature - they are specific to GitHub but very useful in collaborative repositories
  - Other remote providers like GitLab have analogous features with possibly different names (“merge request,” for example)
- A common workflow when working in a group is to always commit to a feature branch (and never to main), push the branch to the remote, open a PR, have someone else on the team review the PR, and, if approved, merge it into main

Merge Conflicts

- Often, git will be able to pull without any issues (if the changes that different people have been making are in different parts of the codebase)
- But if, say, two people both modify the same lines in a file but in different ways, git won't know how to merge them: this is a *merge conflict*

```
int main() {
    void *ptrs[10];
    for (int i = 0; i < 10; ++i)
<<<<<<< HEAD
        ptrs[i] = malloc(i | 1);
=======
        ptrs[i] = malloc(i + 1);
>>>>>>> origin/main
    for (int i = 0; i < 5; ++i)
        free(ptrs[i]);
}
```

Merge Conflicts

- The current version of the code is what's between the `<<<<<<<` and `=======`, while the incoming version is what's between the `=======` and `>>>>>>>`
- Can resolve conflicts manually either by choosing one of the two versions and removing the other, or by combining the two in whatever way makes sense
  - Editors like VS Code have convenient graphical UI elements that make this even easier
- Once resolved, just commit your resolved changes using `git commit`

Demo!

`submit50` uses git too!

https://github.com/cs50/lib50/blob/main/lib50/_api.py#L391