Version Control and Git

What is version control?

Simply, a version control tool tracks change to code. This also includes creation and changes to directories and their contents, that are located within some top level directory. Additionally, metadata is stored to easily identify when a change was made, who made it, and what comments the change was pushed with.

Why do we need it?

Most importantly for this class, version control saves all past versions of your code, so if something goes wrong you can move back to a previous version. Furthermore, Pintos is a very large codebase, so it's necessary that related files are saved together in the repository. You'll have to edit multiple files at a time, so being able to push changes across the repository at once will come in handy.

On a different note, version control is extremely useful for group work. Once a repository is created and shared, every group member will be able to see its contents. This eliminates the need for saving the entire project on one person's machine and the code being inaccessible to the other people. (I speak from my personal experience from Huffman and the 314 partner projects, that situation is simply not it.) In addition to viewing the repository, all group members can "commit" changes. Version control tools also track who committed each part, so it's easy to identify what each person worked on.

What is Git? (And GitHub? Or GitLab?)

Plainly, Git is a version control system, and GitHub, GitLab, and similar systems are cloud-based services that allow you to manage Git repositories. Git is very widely used, as is a good skill to have. I picked it up when I took OS, and it has already been helpful in other places, like internships.

Sign in to your UTCS GitLab account with your CS login and password.

Creating a Git Repository:

Once you've created your GitLab account, you can create new repositories. (We'll be going over how to do so from the website, but you can also create repositories from the terminal completely.)

- Click the "+" button in the top middle and select "New Project"
- For all school projects create a PRIVATE repo!
    - You don't want other students to be able to see your projects
- After naming and creating the repo, follow the commands to "Push an existing folder"
    - Make sure you're in the top most directory of your project before running the commands!
- Add the CS439 instructors with "Reporter" permission. You will find the information you need to share the repository on the Resources page.

Git Checklist:

Everytime you start working on a project, refer to this checklist!

- Run "git status" or "git fetch" when starting
    - If not up to date, "git pull"
- Write code!
- Run "git status" again
    - Make sure the correct files have been edited
- Run "git add ." to commit all files
    - Can also do "git add [filename]" for individual files
    - Can also run "git add -u" to update files that are known to git (will not add files that were created since the last commit).
- Run "git commit -m "[message about what was worked on]" "
- Run "git push"

What is a merge conflict, and should I be worried?

Short answer, no, If anyone has heard about Git, I'm sure you've heard about merge conflicts. However, this shouldn't deter you from using version control!

A merge conflict occurs when two people make changes to something, and Git can't figure out which change is the "correct" one. Thus, Git notifies you of the merge failure, and it's then your

job to see which changes your group would like to keep. For example, say a line of your program contains “hello world”. You change the line to say “hello world!!” and your partner changes it to say “Hello World”. Then, you both try to push your changes. Git has no way of knowing which version you want to keep, so it’s up to the developers to figure it out.

The easiest way to avoid merge conflicts is elementary, but effective. If your entire group works together, rather than having people edit code at the same time, it is impossible to create a merge conflict. As long as only one person pulls the code, everyone works on it together (using the Live Share extension from VS Code would be a good option), and then the code is pushed, no merge conflicts will occur. Essentially, you pull the repository, and no one else pushes before you once again push the code.

Of course, you may want to have people pushing/pulling on their own, so refer to these guides if you run into this issue:
Dealing With Merge Conflicts | Learn Version Control with Git
Git merge conflicts

For more information:

MIT Missing Semester
Oh, Sh*t, Git!?!