Mindful Learning

git

GitHub

Git

- Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.
- It is used for:
  - Tracking code changes
  - Tracking who made changes
  - Coding collaboration

What does Git do?

- Manage projects with Repositories
- Clone a project to work on a local copy
- Control and track changes with Staging and Committing
- Branch and Merge to allow for work on different parts and versions of a project
- Pull the latest version of the project to a local copy
- Push local updates to the main project

Working with Git

- Initialize Git on a folder, making it a Repository
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered modified
- You select the modified files you want to Stage
- The Staged files are Committed, which prompts Git to store a permanent snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Advantages of using Git

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

Github

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
- In this tutorial, we will focus on using Git with GitHub.

Getting Started

- Install Git
  - https://www.git-scm.com/
- Check if Git is properly installed
  - `git --version`
- Configure Git, let Git know who you are
  - `git config --global user.name "test"`
  - `git config --global user.email "test@mindullearning.com"`
- Creating a Git folder
  - `mkdir myproject`
  - `cd myproject`
- Initializing Git
  - `git init`

Adding Files

- Add some files to the newly created Git repository
- Check Git status
  - `git status`
- One of the core functions of Git is the concepts of the Staging Environment, and the Commit.
- As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment. You can add multiple files
  - `git add helloworld.py`
  - `git status`

Adding Files

- Files in your Git repository folder can be in one of 2 states:
  - Tracked - files that Git knows about and are added to the repository
  - Untracked - files that are in your working directory, but not added to the repository

Staging and Commit

- Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.
- When we commit, we should always include a message.
- By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when
  - `git commit -m "First release of Hello World!“`
- To automatically notice any modified (but not new) files, add them to the index, and commit, all in one step.
  - `git commit -a`

Git Log and Help

- You can get the log of commits
  - `git log`
  - `git log --oneline`
- If you are having trouble remembering commands or options for commands, you can use Git help.
- There are a couple of different ways you can use the help command in command line:
  - `git <command> -help` - See all the available options for the specific command
  - `git help --all` - See all possible commands

Revert

- We revert the latest commit using git revert HEAD (revert the latest change, and then commit), adding the option --no-edit to skip the commit message editor (getting the default revert message)
  - `git revert HEAD --no-edit`
  - `git log –oneline`
- To revert to earlier commits, use `git revert HEAD~x` (x being a number. 1 going back one more, 2 going back two more, etc.)

Reset

- The reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.
- Step 1: Find the previous commit:
  - `git log --oneline`
  - Find the log ID
- Step 2: Reset
  - `git reset 9a9add8`
  - `git restore helloworld.py`
  - See the log once again

Branching

- In Git, a branch is a new/separate version of the main repository.
- Let's say you have a large project, and you need to update the design on it.
- With Git:
  - With a new branch called new-design, edit the code directly without impacting the main branch
  - EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
    - Create a new branch from the main project called small-error-fix
    - Fix the unrelated error and merge the small-error-fix branch with the main branch
    - You go back to the new-design branch, and finish the work there
    - Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branching with Git Advantages

- Branches allow you to work on different parts of a project without impacting the main branch.
- When the work is complete, a branch can be merged with the main project.
- You can even switch between branches and work on different projects without them interfering with each other.
- Branching in Git is very lightweight and fast!

Branching

- Creating a new branch
  - `git branch hello-world-2`
- Confirm you have created a new branch
  - `git branch`
- **checkout** is the command used to check out a branch. Moving us from the current branch, to the one specified at the end of the command
  - `git checkout hello-world-2`

Emergency Fixes

- Now imagine that we are not yet done with hello-world-1, but we need to fix an error on master. You don't want to mess with master directly, and you do not want to mess with hello-world-2, since it is not done yet.
- So we create a new branch from master to deal with the emergency:
  - `git checkout -b emergency-fix`
  - `git add index.html`
  - `git commit -m "updated index.html with emergency fix"`

Branch Merge

- We have the emergency fix ready, and so let's merge the master and emergency-fix branches.
  - `git checkout master`
  - `git merge emergency-fix`
- Since the emergency-fix branch came directly from master, and no other changes had been made to master while we were working, Git sees this as a continuation of master. So it can "Fast-forward", just pointing both master and emergency-fix to the same commit.
- As master and emergency-fix are essentially the same now, we can delete emergency-fix, as it is no longer needed:
  - `git branch -d emergency-fix`

Github

- Go to GitHub and sign up for an account
- Sign in, and create a new repository
- Push Local Repository to GitHub
  - `git remote add origin https://github.com/test/hello-world.git`
  - `git push -u origin master`

Github: Pull and Push

- You can pull all changes from a remote repository into the branch you are working on
  - `git pull origin`
- We can make changes in the local repository and push it back to github
  - `git push origin`

Summary

- `git init`
- `git commit -m "first commit"`
- `git branch -M main`
- `git remote add origin https://github.com/mindful-ai/test.git`
- `git push -u origin main`

Summary

- `git remote add origin https://github.com/mindful-ai/test.git`
- `git branch -M main`
- `git push -u origin main`

Git Clone from GitHub

- A clone is a full copy of a repository, including all logging and versions of files.
  - `git clone https://github.com/test/test.io.git`
  - `git log`

GitHub Fork

- At the heart of Git is collaboration. However, Git does not allow you to add code to someone else's repository without access rights.
- A fork is a copy of a repository. This is useful when you want to contribute to someone else's project or start your own project based on theirs.
- fork is not a command in Git, but something offered in GitHub and other repository hosts. Let's start by logging in to GitHub, and fork our repository

Push Changes to Our GitHub Fork

- We have made a lot of changes to our local Git.
- Now we push them to our GitHub fork, commit the changes
  - `git push origin`
- Go to GitHub, and we see that the repository has a new commit. And we can send a Pull Request to the original repository
- Now any member with access can see the Pull Request when they see the original repository and they can see proposed changes
- You can comment on the changes and merge.