6. Git & GitHub

Overview of Git and GitHub

- **Git:**
  - A distributed **version control** system
  - Allows users to **track changes** in source code and revert to **previous** versions and review project **history**
- **GitHub:**
  - A popular **web-hosted** service for **Git repositories**
  - Facilitates easier collaboration and project management

Basic Terminology

- **SSH:**
  - A method for secure remote login from one computer to another
- **Repository:**
  - Contains project folders that are set up for version control
- **Fork:**
  - A copy of a repository into your GitHub account
- **Pull request:**
  - How you request that someone review and approve your changes before they become final
- **Working directory:**
  - Contains the files and subdirectories on your computer that are associated with a Git repository
- **Commit:**
  - Snapshot of project's current state along with a description of the changes made
- **Branch:**
  - A separate line of development that allows you to work on features or fixes independently
- **Merging:**
  - Combines changes from one branch into another, typically merging a feature branch into the main branch
- **Cloning:**
  - A local copy of the remote Git repository on the computer
- **Liscense:**
  - To exepress how people can use your code

GitHub Branches

- All **files** in GitHub are stored on a **branch**
- The **main** branch stores the **deployable** version of your code
  - It is created by **default**, however, you can use any branch as the main
- When you plan to change things, create a **new** branch and give it a descriptive name
  - The new branch starts as an **exact copy** of the original branch
  - Developers commit changes to their branches, writing meaningful comments for the changes
  - A **pull request** makes the proposed committed **changes** available for others to **review** before merging them into the main branch
- GitHub automatically makes a pull request if you make a change on a branch that you do **not** own
- When all changes for a branch are complete, that branch is considered obsolete and should be deleted

Overview Of Git Commands

mkdir

To create a new directory

- `mkdir my-repository`

cd

To neviaget to a directory

- `cd my-directory`

git init

To initializes a new Git repository

- `mkdir my_project`
- `cd my_project`
- `git init`

git add

Adds any file that you created from your working directory to the staging area

The staging area is a temporary storage space where you collect the selected files before asking git to save them in the local repository

- `git add <filename.txt>`
  - To add a specific file
- `git add .`
  - To add all the files that are new or changed in the current directory

git status

To view the status of all the changes so far in your current branch

git commit

To save changes with a descriptive message

- `git commit -m “your-message”`

git log

Displays the commit history for the current branch you are working on

git branch

Lists, creates, or deletes branches in a repository

- `git branch`
  - To list branches
- `git branch <new-branch>`
  - To create a new branch
- `git branch -d <branch-name>`
  - To delete the branch, first check out the branch using `git checkout` and then run the command to delete the branch locally

git checkout

To switch between branches

- `git checkout <branch-name>`

git merge

To merge changes back into the main branch

- `git checkout main-branch` → Nevigate to the main branch
- `git merge <branch-name>`
  - Name of the branch you want to merge with the main branch

git clone

It copies a repository from a remote source to your local machine

- Use terminal and switch to target directory where you want to clone the repository
  - `git clone <repository URL>`

git diff

It shows changes between commits

- `git diff`
  - Shows the difference between the working directory and the last commit
- `git diff HEAD~1 HEAD`
  - Shows the difference between the last and second-last commits
- `git diff <branch-1> <branch-2>`
  - Compares the specified branches

git pull

Fetches changes from a remote repository and merges them into your current branch

First, switch to the branch that you want to merge changes into by running the `git checkout`

- `git pull origin main`
  - Fetches the changes from the main branch of the origin remote repository and merge them into your current branch

git fetch

Fetches changes from the remote repository but does not merge them with your current branch

- `git fetch origin`

Example

- Imagine you’re working on a feature branch (`feature/login`) of a shared project, and your teammate has been making changes to the `main` branch. This command will update your view of the `origin/main` branch (the remote tracking branch) with any new commits that your teammate has pushed to `main`. However, it won’t change anything in your local branches yet. After fetching, you can look at what has changed in `origin/main` and see if any new updates could affect your work

git push

It uploads local repository content to a remote repository

Make sure you are on the branch that you want to push by running the `git checkout` command first

- `git push origin <branch-name>`
  - This command pushes `<branch-name>` to the remote repository named `origin`
  - It does not set the upstream tracking relationship, meaning that if you try to push or pull in the future without specifying the branch, Git won't know which remote branch to use
- `git push -u origin <branch-name>`
  - `-u` sets the upstream tracking relationship
  - This means that in future pushes or pulls, you can simply use `git push` or `git pull` without specifying the branch, as Git will remember that `main` is linked to `origin/main`

git reset

Undoes the changes
Alters the commit history by removing commits from history, which can cause issues if the branch has already been shared with others

- `git reset --hard HEAD`
    - Removes last commit and it’s changes

git revert

Undoes the changes made by a specific commit
Adds a new commit, preserving the history and making it clear that a reversal was done. Safer for collaborative environments where others might be working on the same branch

- `git revert HEAD --no-edit`
    - `HEAD`: Refers to the latest commit in the current branch
    - `--no-edit`: Revert without opening the default text editor to edit the commit message
        - If open, press the Control (or Ctrl) key simultaneously with X to quit

git version

It displays the current Git version installed on your system

- `git version`

More Git Commands

Creating A New Local Repository

- Create a `myrepo` directory
    - `mkdir myrepo`
- Go into the `myrepo` directory
    - `cd myrepo`
- Initiate the `myrepo` directory as a git repository
    - `git init`
- A local git repository is now initiated with a `.git` folder containing all the git files, which you can verify by doing a directory listing by running the following command
    - `ls -la .git`
        - `.` prefix will make the git directory hidden
        - `-la`: renders a long list, including the access permission, time of creation and other details for all the files in the hidden git directory

- Now create an empty file named `newfile`
  - `touch newfile`
- Add this file to the repo
  - `git add newfile`
- Before you commit the changes, you need to tell Git who you are
  - `git config --global user.email " you@example.com"`
  - `git config --global user.name "Your Name"`
- You can now commit your changes
  - `git commit -m "added newfile"`
- Create a new branch in your local repostitory & Switch to the newly created branch
  - `git branch my1stbranch`
  - `git checkout my1stbranch`
  - *Shortcut: Creates the branch and makes it active*
    - *`git checkout -b my1stbranch`*

- Make some changes to your new branch, add some text to `newfile` by running the following command that will append the string "Here is some text in my newfile." into the file
  - `echo 'Here is some text in my newfile.' >> newfile`
- Verify the text has been added
  - `cat newfile`
- A shortcut to adding all modifications and additions
  - `git add .`
- Now that your changes are ready, save them to the branch
  - `git commit -m "added readme.md modified newfile"`
- Get a history of recent commit
  - `git log`
- Merge the contents of the `my1stbranch` into the `main` branch
  - `git checkout main`
  - `git merge my1stbranch`
- Now that changes have been merged into master branch, the `my1stbranch` can be deleted
  - `git branch -d my1stbranch`

Cloning A Remote Repository

- Cloning creates a **copy** of the project's code, and its complete **version history** from the remote repository on your **local** computer
- The **connection** established during cloning enables you to **push** code **changes** to the remote repository and also **pull** any **changes** from the remote repository to your local repository
- Scenario: Working on a project for E-Commerce website and you're assigned to implement the product recommendation feature
  - There are multiple ways to code the feature. A good practice is to create a **branch** from the main branch, and add the feature without interfering with the main code base
  - After developing the feature, select the changed files and move them to a staging area
  - Commit the files to the newly created branch
- Pushing changes to the remote repository
  - After committing the changes, the code needs to be **reviewed** before it is merged into the **main** branch in the **remote** repository
  - Create a pull request to review the changes in your branch

Example

- Use terminal to go the target directory where you want to clone the remote repository.
  - Run `git clone <repository-URL>`
  - Then use `cd` to change to the cloned repository
- Start by listing branches with the `git branch` command
- Create the child-branch branch using the `git branch child-branch` command
- Switch to the new branch using `git checkout child-branch` command
- Make changes to files (add, update, delete)
- Use `git status` command to check the status of all the changes made
- Stage the changes using the `git add` command
- Commit these changes using `commit -m “your-message”` command
- Merge child-branch with the main branch using `git checkout main` and `git merge child-branch`
- Push the changes to the main branch of the remote repository using `git push -u origin main`

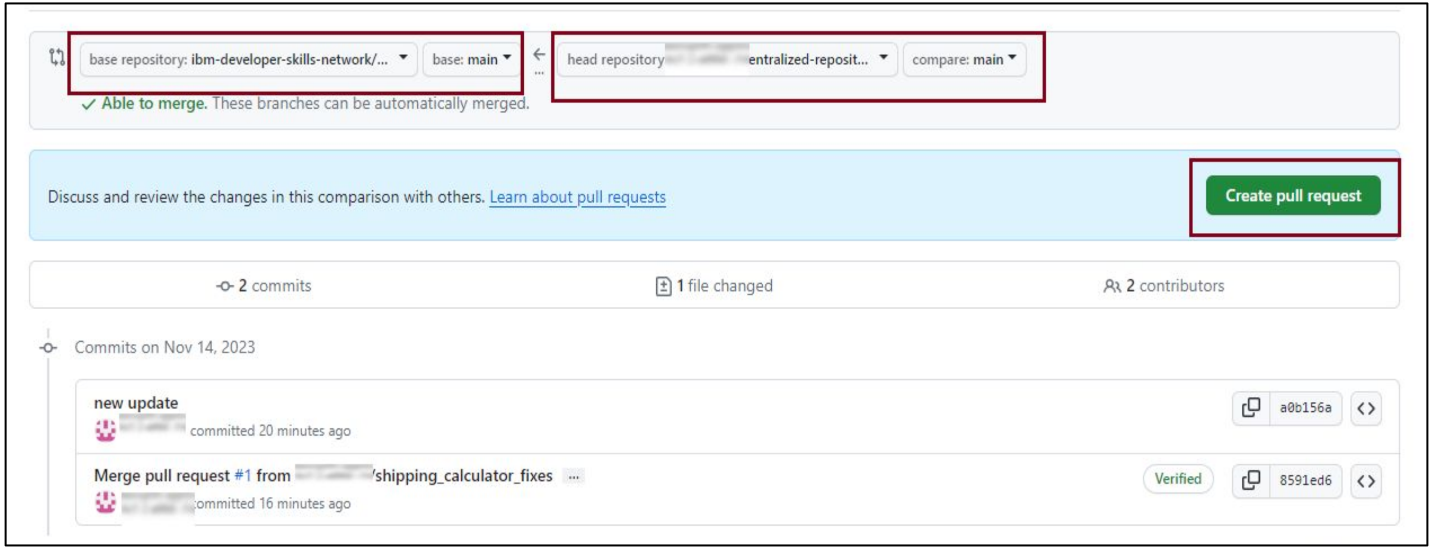
Forking A Repository

- Forking a Git repository is a way to create your <u>own copy</u> of <u>someone else’s</u> repository on your <u>GitHub</u> account
- This allows you to freely experiment with changes without affecting the original project, making it a popular approach for open-source contributions and collaboration

Steps

1. Navigate to the Repository: Go to the original repository’s page on GitHub
2. Click the “Fork” button at the top right corner of the page
   - GitHub will create a copy of that repository under your account
3. Clone Your Fork Locally: The forked copy of the repo becomes the <u>origin</u>
   - `git clone https://github.com/your-username/repository-name.git`
4. Make Changes and Commit
   - `git checkout -b new-feature`
   - `git add .`
   - `git commit -m "Add new feature"`

5. **Push Changes to Your Fork**
    - Once you're satisfied with your changes, push them to your fork on GitHub
        - `git push origin new-feature`
6. **Open a Pull Request**
    - If you want your changes to be considered in the original repository, you can open a pull request. This asks the original repository's maintainers to review and merge your changes
        - Go to your fork on GitHub
        - Navigate to the branch with your changes
        - Click on "Compare & pull request."
        - Describe your changes and submit the pull request


base repository: ibm-developer-skills-network/... base: main ← head repository: [redacted] entralized-reposit... compare: main

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

Commits on Nov 14, 2023

new update
[redacted] committed 20 minutes ago — a0b156a

Merge pull request #1 from [redacted]/shipping_calculator_fixes ...
[redacted] committed 16 minutes ago — Verified — 8591ed6

Terms

- `Origin` - refers to <u>your</u> work
- `Upstream` - refers to <u>original</u> work
- `head repository` and the `compare` - refers to the repository from where you want to <u>initiate</u> the pull request. In this instance, it is <u>your</u> GitHub repository
- `base repository` and the `base` - refers to the <u>upstream</u> repository and branch where you intend to submit a pull request

Keeping Your Fork Up-to-Date

Since the original repository may continue to change, you'll often want to keep your fork in sync with it

- Add the Original Repository as a Remote: This allows you to pull updates from the original repository
  - `git remote add upstream https://github.com/original-owner/repository-name.git`
- Two main options to incorporate changes from the upstream repository into your fork:

  Option A: Merge

  - `git checkout main`
  - Merge the changes from the **upstream** `main` branch into your **local** `main` branch
    - `git merge upstream/main`
  - If there are **no** conflicts, Git will merge the changes, and your local main branch will be up-to-date

  Option B: Rebase

  - `git checkout main`
  - Rebase the changes from the upstream main branch
    - `git rebase upstream/main`
  - **Resolve** any conflicts if they arise, and continue the rebase with `git rebase --continue`
- Push the Updates to <u>Your</u> Fork
  - `git push origin main`

Cloning & Forking Repository - Lab

GitHub Copilot

- An **AI-powered** tool designed to assist developers in **writing code** more efficiently and effectively
- Features
  - Offers code **auto-completion**, suggesting entire lines or blocks of code based on the context of your current work across various programming languages, including Python, JavaScript, Java, and C++
- Typical Workflow with GitHub Copilot
  - To use GitHub Copilot, you first need to **install** the **extension** in your Visual Studio Code environment and activate it by signing in with your GitHub account
  - As you write code, GitHub Copilot generates suggestions for function bodies and variable names, which you can accept using the tab key or by selecting the suggestion
  - After generating or editing code, it's crucial to review it for alignment with project requirements and best coding practices, as GitHub Copilot does not assist with debugging

SSH Protocol

- Enables secure **communication** between **two** networked devices
- It operates on port **22** and allows
  - Secure remote login
  - Command execution
  - File transfer
- SSH supports multiple **authentication** methods, the most common being:
  - **Password-based**
    - Uses a <u>username</u> and <u>password</u>
  - **Public key-based**
    - Involves generating a pair of cryptographic <u>keys</u> (public and private)

SSH Key

- An SSH key is a cryptographic **pair** of keys used for SSH **authentication**
    - **Private Key:** Stored securely on the client device and should not be shared
    - **Public Key:** Shared with the server (e.g., GitHub) and is used to verify the client
- Using SSH keys with **Git** and **GitHub**
    - It allows you to securely **authenticate** and **access** your repositories without repeatedly entering a username and password

Generating an SSH Key

- Launch a terminal. If you are using Windows, launch Git Bash
- Replacing <your email address> with the email address that is linked to your Github account
  - `ssh-keygen -t rsa -b 4096 -C "<your email address>"`
- You will be prompted to enter a directory to **save** the key
  - You can simply press Enter to accept the default location, which is an .ssh folder in the home directory. This means you will be able to locate the key in
    - `~/.ssh/id_rsa`
- You will be prompted to choose a passphrase
  - To skip the passphrase, press Enter twice to confirm that the passphrase is empty
- You now need to **add** the SSH key to the `ssh-agent`, which helps with the authentication process
  - **Start** the ssh-agent
    - `eval "$(ssh-agent -s)"`
  - **Add** the key to the agent
    - `ssh-add ~/.ssh/id_rsa`

Adding SSH To A Repo

- To check if you already have an SSH key
  - `ls -al ~/.ssh`
- **Copy** your public key to the clipboard
  - `pbcopy < ~/.ssh/id_rsa.pub`
- Go to GitHub SSH keys settings
  - Click on "New SSH key"
  - Give it a title (e.g., "MacBook SSH Key") and paste the SSH key from your clipboard
  - Click "Add SSH key"
- To clone the repo - you need the SSH URL of the repo
  - `git clone <your repo ssh url>`

Divergent Branches (`git pull origin main`)

Divergent branches

- Your local branch and the remote main branch have different histories. This might happen if:
  - You've made changes locally that haven’t been pushed.
  - The remote main branch has new commits that you haven’t pulled yet.
- Git doesn't know how to automatically combine these changes, so it prompts you to specify how to reconcile them.

What git pull origin main Does

- **Fetches changes** from the main branch of the origin remote repository
- Attempts to merge the changes into your current branch
- However, if the branches have **diverged**, Git will ask on how you want to handle these differences

Resolving Differences

Merge

-
  ```
  git config pull.rebase false
  git pull origin main
  ```
- This configures Git to merge the changes
- Merging creates a new commit that combines your local changes with the remote changes
- Shortcut: `git pull --no-rebase`

Rebase

-
  ```
  git config pull.rebase true
  git pull origin main
  ```
- This configures Git to reapply your local commits on top of the fetched commits
- Rebasing rewrites commit history to create a linear sequence of commits
- Shortcut: `git pull --rebase`

Fast-forward only

-
  ```
  git config pull.ff only
  git pull origin main
  ```
- This allows pulling only if your local branch can be fast-forwarded (no diverging commits)
- If the branches have diverged, Git will refuse to pull
- Shortcut: `git pull --ff-only`

`git config --global`

- You can set one of the above options globally for **all** repositories
- For example: `git config --global pull.rebase false`

`git fetch origin main`

- To resolve the conflicts manually **without** merging or rebasing