**Steps, in short, using Git command**

* Go to original repo. Forked a copy into your github account
* Go to your github account to download the forked repo to your local machine
* Open Jupyter Notebook / VS Code etc to work on it
* Commit changes to your local Git repo. This records your changes in your local version control.
* upload your commits from your local repository to the forked repository on GitHub. This synchronizes your local changes with your remote fork on GitHub.
* Once you're ready to submit your work, you create a pull request from your forked repository to the original repository. This is essentially a request for others to review your code
* Other collaborators will receive a notification about your pull request. They can then review your work

**Steps, in short, without Git command**

* Forked repo
* Download forked repo using ‘download as zip’
* Work locally
* Upload file using ‘add file > upload files’
* Pull request
  + Go to original repo, go to ‘pull request’ tab

A screenshot of a computer

Description automatically generated

* + Click ‘new pull request’
  + Choose fork and branch with the changes
  + Create pull request following github instruction

Please note that while this method works, it does not utilize the full power of Git for version control and collaboration. It's a manual process and may not be efficient for larger changes or for when you need to keep your local copy up to date with the repository. It's also error-prone and might lead to conflicts if not managed carefully.

**Demonstration**

* User X : owner of original repo
* User Y : is invited for collaboration
* Invite and accept invite

A screenshot of a web page

Description automatically generated

* Go to cmd, type ‘git clone https://github.com/name12t11/test1.git’ A screenshot of a computer

  Description automatically generated
* Lets say the cmd command is ‘C:\Users\test>git clone <https://github.com/name12t11/test1.git>’, then the local clone folder will be located at ‘C:\Users\test’

* A black screen with white text

  Description automatically generated
* To check whats the github account that you have signed in to on cmd

A black and white sign with white text

Description automatically generated

* To change the account on cmd

git config --global user.name "NewUsername"

git config --global user.email [newemail@example.com](mailto:newemail@example.com)

Regarding signing in on cmd

* Cloning Public Repositories:
  + You do not need to sign in. Anyone can clone a public repository without authentication.
* Cloning Private Repositories:
  + You need to authenticate with GitHub to prove that you have permission to access the repository. This could be through HTTPS using your GitHub username and a personal access token or via SSH using an SSH key that you've added to your GitHub account.
* Pushing Changes:
  + Whether the repository is public or private, you need to authenticate to push changes as this action requires write permission.
* Make your changes to the local clone
* Open cmd, go to folder of the clone repo that contain .git file.

C:\Users\test\cd C:\Users\test\test1

* To commit the changes

git commit -m "Commit message"

This command commits the staged changes to the local repository. Each commit is a snapshot of your work that you can revert to if needed. The **-m** flag allows you to add a commit message, which is a brief description of the changes included in the commit.

A screen shot of a computer

Description automatically generated

* git add test.py

This command stages all your current directory changes for the next commit. Staging is a step before the actual commit where you prepare and review the changes that will go into your next commit.

A black screen with white text

Description automatically generated

* A black background with white text

  Description automatically generated

indicates that the branch you're trying to push to, named master, doesn't exist in your local repository

* Check current branch name

A screen shot of a computer program

Description automatically generated

* If you are on ‘main branch’, you should push to main

A computer screen shot of a computer program

Description automatically generated

* Go to your github account, forked repo, to check that the changes has been made
* The system will look at the folder where my .git is located. The .git tell the system which remote repo this folder cloned. Then the system will update the changes to the remote repo if i push origin main
* Create new pull request

A screenshot of a computer

Description automatically generated

Red (-): Code that has been deleted or the old version before changes.

Green (+): New code that has been added or the new version after changes.

From collaborator’s perspective

A screenshot of a computer

Description automatically generated

From original repo owner’s perspective

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Any comments added will trigger an email notification to you

* Review in codespace

A white text on a blue background

Description automatically generated

**How to check what repo do I have collaboration**

* <https://github.com/settings/repositories>

A screenshot of a computer

Description automatically generated

**Cloning vs forking**

* Cloning is to make a local copy of the remote GitHub repo
* Forking is to copy someone else’s repo into your own repo

**Key aspects of collaborative software development**

It encompasses practices and tools that allow multiple developers to work together on software projects

* Version Control Systems (VCS): Tools like Git allow developers to track changes, revert to previous states, and work on different features simultaneously without interfering with each other's work.
* Hosting Services: Platforms like GitHub, GitLab, and Bitbucket provide remote repositories that can be accessed by multiple developers. They offer features like forking, branches, and pull requests to facilitate collaboration.
* Branching and Merging: Developers create branches to work on new features or bug fixes in isolation. ~~Later, these branches can be merged back into the main codebase~~.
* Code Review: ~~Before merging changes~~, other team members can review the code in pull requests, suggest improvements, and discuss implementation details.
* ~~Continuous Integration/Continuous Deployment (CI/CD): Automated pipelines run tests, build software, and deploy it to production, ensuring that changes don't break the software.~~
* ~~Issue Tracking: Tools to track bugs, feature requests, and tasks help coordinate the team's efforts.~~

Collaborative software development is a key practice in both open-source projects and in the industry, enabling teams to build complex systems efficiently.

**Process for a collaborator with no write access to original repo**

* Branching: You create a branch in your fork to work on new features, enhancements, or bug fixes.
* Committing: You make and commit changes to your branch, encapsulating your work in manageable chunks with descriptive commit messages.
* Pushing: You push the commits from your local branch to the corresponding branch in your fork on GitHub.
* Pull Requesting: Once you are ready for your changes to be reviewed, you open a pull request from your branch to the original repository. This is your proposal for integrating your changes.
* Code Review: Maintainers of the original repository will review your pull request. They may discuss it with you, request changes, ~~or approve it~~.
* ~~Merging: If your pull request is approved, a maintainer of the original repository will merge your changes into the main codebase. This step is typically not performed by you unless you have been given the necessary permissions.~~

This process allows for a structured review and integration of your contributions to the project while ensuring that the main codebase remains stable.

**Who can see your pull request?**

* Typically only the original private repository owner can view and interact with pull requests in a private repository. Other collaborators with access to the original private repository can see pull requests only if they have been granted the permissions to do so.
* In a private repository, there are different access levels:
  + Read: Can view the repository but cannot contribute directly. Users with read access cannot see pull requests.
  + Write: Can view and contribute to the repository directly, including viewing and commenting on pull requests.
  + Admin: Full access to the repository, including accepting pull requests. Only admin can accept pull requests.

**What is accepting pull requests?**

* When a pull request is accepted and merged by someone with ‘admin’ level access, the changes from your branch are integrated into the target branch of the original repository. If admin makes additional changes during the merge process (like resolving conflicts or making small tweaks), those changes will also be reflected in the original repository alongside your contributions.

**How to add collaborator to your repo?**

* In the "Access" section of the sidebar, click Collaborators. Click Add people. In the search field, start typing the name of person you want to invite, then click a name in the list of matches. Click Add NAME to REPOSITORY.

A screenshot of a computer

Description automatically generated

**How to accept collaboration?**

* Primary email address will receive a email to collaborate. Click accept.

A screenshot of a web page

Description automatically generated

**What is forking a repo and creating your own branch?**

**Forking a repo**

* Forking today copies all files up till today, forking tomorrow copies all files up till tomorrow. If changes are made between today and tomorrow, then you have 2 different forked repo for 2 different time periods
  + However, it's important to note that having multiple forks of the same repository can be confusing and is generally not a common practice. Typically, if you need to update your fork with the latest changes from the original repository, you would fetch and
* Forking a repository and creating your own branch are two fundamental practices in collaborative software development, particularly when using version control systems like Git, hosted on platforms like GitHub.
* Forking a repository means creating a personal copy of someone else's project on your own GitHub account. This allows you to freely experiment with changes without affecting the original project (also known as the "upstream" repository). When you fork a repo:
  + You get a snapshot of the project at the time of forking
  + You can make modifications in your fork without impacting the original repository.
  + You can later propose these changes to the original repository via pull requests, allowing the maintainers to review and possibly merge your updates

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

copy master branch only

* selecting this option
  + forking only master branch of original repo
* unselecting this option
  + fork entire repo, including branches

A branch in Git is essentially a pointer to a snapshot of your changes. When you start a new branch, you're creating a separate line of development that diverges from the main code base, which is typically referred to as the master or main branch. This allows you to work on your own version of the project without affecting the main branch.

**What are other branches (in case of whether to tick the box ‘copy main branch only’)?**

* In a repository, there may be multiple branches aside from the main branch. These could be for developing new features, fixing bugs, or experimenting with new ideas in isolation from the main code.
* Each branch can be developed independently, and when the work on a branch is complete, it can be merged back into the main branch to integrate the changes with the main code base.

Summary:

Main branch: The primary development branch, often called master or main.

Your branch: A personal branch where you make changes related to your assignment. Target branch is the branch in the original repository that you are proposing to merge your changes into with a pull request. It is where your contributions will go if the pull request is accepted.

Other branches: Any additional branches in the repository, which you probably don't need to worry about for a take-home assignment.

* These ‘other branches’ are not just pull requests from collaborators like you. It is just branches extending out of the main branch that may be added to the main branch

If you check "Copy the master branch only":

* You're forking only the master (or main) branch. Any pull requests you make would typically target this master branch in the original repository unless instructed otherwise.

If you don't check the option:

* You're forking all branches. You could then potentially target any of those branches with your pull request, depending on which branch your changes are relevant for.

**For the case of home-take assignments**

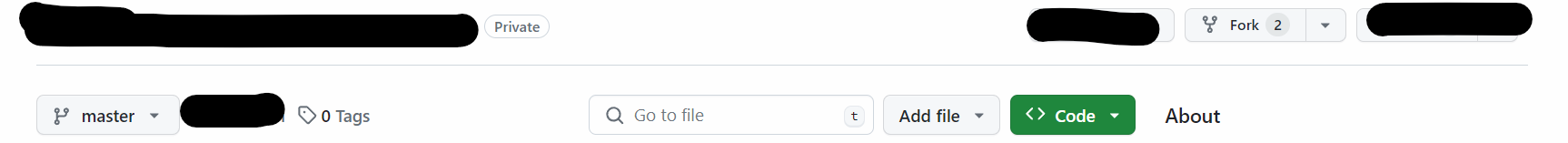
* The primary intention of forking the repository and creating branches is to work on your assignment and submit it for review via a pull request
* Typically to tick ‘copy main branch only’ as you don’t need to interact with other branches (in fact, there is likely no other branches)

**Creating your own branch**

* Creating a branch in Git allows you to work on new features or fixes in isolation, without disturbing the main codebase (usually the master or main branch). Here's what happens when you create a branch:
  + You create a parallel environment where you can make changes independently.
  + Each branch is like a new working directory, staging area, and project history that diverges from the main line of development.
  + After making changes in your branch, you can merge these changes back into the main branch of your fork or propose them to the original repository using a pull request.

**How to show your work to other collaborators?**

* Pushing Their Research:
  + Typically, contributors do not directly push to the original repository unless they have been given explicit collaborator access. Instead, they work on their own forks and branches and then create pull requests to submit their work. This is a common workflow in open-source projects and collaborative environments.
* Notify other collaborators:
  + Once the pull request is created, you can notify other collaborators by tagging them in the pull request description using @username (replace username with their GitHub username) or by sending them the link to the pull request via email or any other preferred communication method.
* Collaboration and Review:
  + other collaborators can review your pull request, provide feedback, and request changes if necessary.
  + You can make additional changes based on feedback, commit them, and push them to the same branch, which will automatically update the pull request.
* Collaboration and Privacy:
  + Collaboration in public repositories is transparent, where everyone can see the branches and pull requests.
  + In a scenario where the repository is private or where contributors want to keep their work confidential until a certain stage, they may work on private forks or branches and only share their work when ready.



This workflow is advantageous because it keeps all related work in one place, maintains a clear history of your research and changes, and facilitates collaborative review and discussion. It is generally preferred over sending files via email for projects managed with version control systems.

**How to set the forked repo as private in your personal GitHub account?**

* By default, forks inherit the visibility of the source repository (i.e., if the source repo is public, the fork is also public).
* If the original repo is public
  + Fork Publicly, Then Duplicate to a New Private Repository:
    - First, fork the repository normally by clicking "Create fork".
    - Once forked, create a new private repository in your GitHub account.
    - Clone the forked public repository to your local machine.
    - Push the local clone to your new private repository.
* If the original repo is private and you fork it
  + Your forked repo will be private
  + The owner of the original repo (and other collaborators that have access to the original private repo) will NOT have access to your forked private repo
  + You will need to invite collaborator to your forked private repo if desired

**Git and GitHub**

Git and GitHub are closely related but distinct concepts in software development:

Git:

* What It Is: Git is a distributed version control system. It helps manage and track changes in code, allowing multiple people to work on the same project without conflicting with each other.
* Key Features:
  + Branching and Merging: Git allows developers to create branches to work on new features or fixes and then merge these back into the main codebase.
  + Local and Remote Repositories: With Git, you have a local copy of your code repository, and you can sync it with a remote repository for collaboration.
  + Snapshot-Based Tracking: Git takes snapshots of changes and uses these to manage versions of a project.

GitHub:

* What It Is: GitHub is an internet-based hosting service for Git repositories. It provides a web-based graphical interface and additional features for project management.

Key Features:

* Pull Requests and Code Review: GitHub introduces the concept of 'pull requests' for proposing changes from one branch to another and facilitates code reviews.
* Issue Tracking: It offers tools for tracking bugs and managing tasks.
* Forking and Collaborating: GitHub allows you to fork (make a personal copy of) repositories, make changes, and then propose these changes back to the original repository via pull requests.

In summary, Git is the system that manages the versions of your code, while GitHub is a platform that hosts your Git repositories and provides tools for collaboration and project management. They work hand-in-hand to facilitate modern software development, especially in collaborative environments.