GitHub Tutorial

James Derrod
jderrod@oxy.edu
Occidental College

1 Introduction

This document serves as a tutorial for using the GitHub version control system through the command line and the GitHub Desktop app.

This tutorial will cover the following concepts:

- Repositories:
  - Initializing repositories.
  - Cloning repositories.
  - Adding to a repositories.
  - Committing to repositories.
  - Pushing to repositories.
  - Pulling from repositories.
  - Stashes.
- Merging:
  - Working with Branches
  - Merge conflicts & resolving them.

By the end of this tutorial you should have a concrete understanding of the fundamentals of managing projects with GitHub.

2 Repositories

Repositories are essentially containers for a project's files. These files may contain code, documentation, other resources and a revision history of each file. GitHub's version control system utilizes both local and remote repositories, and facilitates the process of moving files between them, allowing for ease of collaboration.

2.1 Initializing a Repository

Initializing a repository means creating a new Git repository from an existing directory or project that wasn't under version control before.

- Command Line:
  - Navigate to your project directory and execute 'git init' to initialize a new Git repository.
  - This creates a '.git' directory in your project, containing all necessary metadata for version control.
- GitHub Desktop
  1. GitHub Desktop does not directly support initializing a repository within the app. You'll need to initialize the repository using the command line or another Git interface, then add it to GitHub Desktop by going to 'File' - 'Add Local Repository'.

Initializing a repository is the foundational step to start tracking your project with Git, turning a directory into a workspace that records changes and history. See Git documentation here. [5][12]

2.2 Cloning a Repository

Cloning is the process of creating a local copy of a remote repository. It includes all branches and commits, enabling offline work and exploration of the repository's history.

- Command Line:
  - Use 'git clone [url]' to clone a repository.
  - You can specify a directory name to clone into with 'git clone [url] [directoryName]'.
- GitHub Desktop
  1. Open GitHub Desktop and log in to your GitHub account.
  2. Go to 'File' - 'Clone Repository'
  3. Choose the repository you wish to clone from the list or specify the URL.
  4. Choose the local path where you want to clone the repository.
  5. Click 'Clone'

Cloning is the first step to contribute to a project, allowing you to work with the repository's files locally. See Git documentation here. [3][10]

2.3 Adding to a Repository

Adding changes, also known as staging, prepares your changes for a commit by marking modifications to be included in the next commit snapshot. This step is crucial for granular control over what is committed.

- Command Line:
  - Use 'git add [file]' to stage a specific file
  - Or use 'git add .' to stage all changes in the directory. This command does not alter the repository until a commit is made.
- GitHub Desktop
  1. Changes are automatically detected and listed.
  2. To stage changes, select the checkbox next to each file you wish to include in your next commit.

Adding changes allows you to curate what goes into a commit, ensuring only desired modifications are included, enabling a neat commit history and facilitating precise project tracking. See Git documentation here. [1][8]

2.4 Committing to a Repository

Committing is the process of saving staged changes to the local repository. Each commit has a message that describes the changes, creating a transparent history of your work.

- Command Line:
  - Execute 'git commit -m "Commit message"' to commit staged changes with a descriptive message.
  - For multi-line messages, use 'git commit' without '-m' and an editor will open.
  - This editor is determined by your Git configuration, often defaulting to Vim or your system's default text editor. Here, you can write a multi-line commit message, providing the opportunity to include a detailed description of the changes made, why they were made, and any other relevant information.
- GitHub Desktop
  1. After staging changes, enter a commit message in the summary field.
  2. Click "Commit to [branch]" to save your changes locally

Committing encapsulates your progress, allowing you to document each step of development clearly and methodically. See Git documentation here. [4][11]

2.5 Pushing to a Repository

Pushing is the act of sending your committed changes to a remote repository. This updates the remote repository with your local commits, sharing your work with the team.

- Command Line:
  - Execute 'git push' to upload local branch commits to the remote repository.
  - If you're pushing a new branch, use 'git push -u origin [branchName]' to set the upstream branch.
- GitHub Desktop
  1. Commit your changes locally.
  2. Click "Push origin" to send your commits to the remote repository.

Pushing is essential for collaborative development, allowing others to see and work with your contributions. See Git documentation here. [7][14]

2.6 Pulling from a Repository

Pulling updates your local branch with changes from its remote counterpart, merging any new commits from the remote repository into your local working directory.

- Command Line:
  - Use 'git pull' to fetch changes from the remote and merge them into your current branch. This command combines 'git fetch' followed by 'git merge'
- GitHub Desktop
  1. Navigate to 'Repository' in the menu bar.
  2. Click 'Pull' to fetch and merge remote changes into your local branch.

Pulling ensures your repository stays up-to-date with others' work, facilitating collaboration and minimizing merge conflicts. See Git documentation here. [6][13]

2.7 Stashes

Stashing temporarily shelves (or stashes) changes so you can work on a different task. Your changes are saved on a stack, and you can reapply them later.

- Command Line:
  - To stash changes: 'git stash' saves your working directory and index state.
  - To apply stashed changes: 'git stash pop' reapplies the most recently stashed changes and removes them from the stash stack.
- GitHub Desktop
  - GitHub Desktop does not directly support stashing through the GUI. Stash management needs to be done via the command line or other Git tools.

Stashes are useful for switching contexts quickly without committing incomplete work. See Git documentation here. [6]

2.8 Working with Branches

Branches in Git enable you to diverge from the main line of development and work independently without affecting the main line.

- Command Line:
  - To create a new branch: 'git branch [branchName]'
  - To switch to a branch: 'git checkout [branchName]'
  - 'git merge [branchName]' - executed from the branch you want to merge into.
- GitHub Desktop
  1. Click the current branch name in the top menu to open the branch menu.
  2. Click "New Branch" to create and switch to a new branch.
  3. To merge, first switch to the branch you want to merge into. Then go to Branch in the menu, and select 'Merge into current branch'.

Branches are essential for managing features, fixes, and experiments in a segregated manner without disrupting the main codebase. See Git documentation here. [2][9]

3 Merge Conflicts and Resolving Them

Merge conflicts occur when Git can't automatically reconcile differences in code between two commits. They often happen when two branches have made edits to the same line in a file, or when one branch deletes a file while the other branch was modifying it.

- Command Line:
  1. Use 'git status' to identify conflicted files.
  2. Open conflicted files and look for conflicts indicated by ASCII markers in the text editor.
  3. Manually edit the file to resolve conflicts.
  4. After resolving conflicts, use 'git add [file]' to mark as resolved and commit the changes.
- GitHub Desktop
  - GitHub Desktop notifies you of conflicts during a merge but does not provide tools to resolve them within the app. You'll need to resolve conflicts in a text editor or IDE, then mark them as resolved through the app.

Resolving merge conflicts is crucial for maintaining a clean and efficient development process, ensuring that all changes are integrated correctly.

References

[1] Git. *Git Add Documentation*. 2024. URL: https://git-scm.com/docs/git-add.

[2] Git. *Git Branch Documentation*. 2024. URL: https://git-scm.com/docs/git-branch.

[3] Git. *Git Clone Documentation*. 2024. URL: https://git-scm.com/docs/git-clone.

[4] Git. *Git Commit Documentation*. 2024. URL: https://git-scm.com/docs/git-commit.

[5] Git. *Git Init Documentation*. 2024. URL: https://git-scm.com/docs/git-init.

[6] Git. *Git Pull Documentation*. 2024. URL: https://git-scm.com/docs/git-pull.

[7] Git. *Git Push Documentation*. 2024. URL: https://git-scm.com/docs/git-push.

[8] GitHub Open Source. *Git add*. 2022. URL: https://github.com/git-guides/git-add.

[9] GitHub Open Source. *Git branches merges*. 2022. URL: https://github.com/git-guides/git-Remote.

[10] GitHub Open Source. *Git clone*. 2022. URL: https://github.com/git-guides/git-clone.

[11] GitHub Open Source. *Git commit*. 2022. URL: https://github.com/git-guides/git-commit.

[12] GitHub Open Source. *Git init*. 2022. URL: https://github.com/git-guides/git-init.

[13] GitHub Open Source. *Git pull*. 2022. URL: https://github.com/git-guides/git-Pull.

[14] GitHub Open Source. *Git push*. 2022. URL: https://github.com/git-guides/git-push.