Git and GitHub

Created	@October 29, 2025 6:00 PM
Class	Tech Notes & Research
Type	Research
Materials	https://git-scm.com/cheat-sheet https://education.github.com/git-cheat-sheet-education.pdf
Reviewed	☐

What is Git and GitHub?

- Git is a VCS (Version control system) which is a software, whereas GitHub is a website.
- Git is tool used to collaborate along multiple people for the entire documentation and history of the code. It helps track changes in code.
- GitHub is a website that uses Git internally. It allows developers to store and manage their code using Git.

Git

- It is installed locally
- First released in 2005
- Maintained by the Linux Foundation
- Focused on version control and code sharing
- Primarily a command-line tool
- Provides a desktop interface called Git Gui

- No user management features
- Minimal external tool configuration features
- Competes with IBM, Mercurial, Subversion, Rational Team Concert and ClearCase
- It is Open source licensed

GitHub

- Hosted in the cloud
- Company launched in 2008 and purchased by Microsoft in 2018
- Focused on centralized source code hosting
- Administered through the web
- Desktop interface named GitHub Desktop
- Built-in user management
- Active marketplace for tool integration
- Competes with Atlassian Bitbucket and GitLab
- Includes a free tier and pay-for-use tiers

What is a Repository?

- It is technically where you store your projects.
- You have to option to make the repository public (accessible to all) or private (accessible to only you).
- We can initialize the repository with a README file - it is a file where you can give the entire detail of your project like the name of project, what the project is, the use of it, why you made the project, what all features are there in the

project, etc. If you know basic HTML, then you can add more details to your README file.

- Changes are known as commit.
- Initial commit - means the first change you do after you create a repository.
- There are two ways to make changes in GitHub - first you add it and then commit it.
- On every commit - the reason for the commit will be asked (documentation and code history).

Setting up Git

1. Install VS Code
2. Install Git Bash (for windows) or terminal (for mac)

Configuring Git

Configuring Git means tell it from which account shall be making changes - whose name is to be associated with the changes and whose email is to associated.

There are two types of configuration - Global (changes to the complete system) and Local (changes to specific repo using another account)

~ indicates that you are in the root directory (main folder) of your system

1. git config --global user.name “My Name”
2. get config --global user.email “someone@email.com”
3. git config --list

Git Commands

- **Clone** - Cloning a repository on our local machine. There are two types - local (laptop/pc) and remote (GitHub). It is used to copy a repository from GitHub to our laptop

To copy the link - got to your GitHub repo, Click on the code button and copy the HTTPS link.

Open VS Code and the folder you want to clone the repo into. Go to the terminal and type the command

git clone←some link→

- **cd** - basic command that runs on Linux, windows and mac. It means 'change directory'. Directory means a folder.

  cd ←folder name→

  cd .. (to move out of a directory)

  mkdir ←directory name→ (to make a new directory)

- **ls** - means 'list files'. If you want to see all the files in a particular folder, you use this command. There are some files that are hidden and can be accessed only through the terminal.

  ls

  ls -a

- **Status** - It displays the status of the code. It shows whether you have modified the file and also if the changes made to the file has been committed.

There will be 4 types of status shown - Untracked (new files that git doesn't yet track), Modified (changed), Staged (File is ready to be committed) and Unmodified (unchanged).

git status

- **Add** - It adds new or changed files in your working directory to the Git staging area.

  git add ←file name→

  git add . (if you have multiple files to add, can add all with this)

- **Commit** - It is the record of change

  git commit -m "some message"

- **Push** - Upload local (laptop) repo content to remote (GitHub) repo

git push origin main

- **Init** - Used to create a new git repo, i.e., if you created a new repo and you have to use this command to initialize Git to the said repo.

  git init

  git remote add origin ←link→

  git remote -v (to verify remote)

  git branch (to check branch)

  git branch -M main (to rename branch)

  git push origin main

- **Branch** - Isolating work in branches, changing context, and integrating changes

  git branch (to check branch)

  git branch -M main (to rename branch)

  git checkout ←branch name→ (to navigate)

  git checkout -b ←new branch name→ (to create a new branch)

  git branch -d ←branch name→ (to delete a branch)

- **Merge** - for merging the code.

  Way 1:

  git diff ←branch name→ (to compare commits, branches, files & more)

  git merge ←branch name→ (to merge 2 branches)

  Way 2:

  Create a PR (Pull Request) - It lets you tell others about changes you've pushed to a branch in a repository on GitHub

- **Pull** - Used to fetch and download content from a remote repo and immediately update the local repo to match that content.

  git pull origin main

- **Undoing Changes** - Used to undo the commits or changes made

Case 1: Stages Changes:

git reset< - file name- >

git reset

Case 2: Committed changes (for one commits)

git reset HEAD~1

Case 3: Committed Changes (for many changes)

git reset ←commit hast→

git reset --hard←commit hast→

- **Temporary Commits** - Temporarily store modified, tracked files in order to change branches

git stash (Save modified and staged changes)

git stash list (list stack-order of stashed file changes)

git stash pop (write working from top of stash stack)

git stash drop (discard the changes from top of stash stack)

- **Inspect & Compare** - Examining logs, diffs and object information

git log (show the commit history for the currently active branch)

git log branchB..branchA (show the commits on branchA that are not on branchB)

git log --follow ←file→ (show the commits that changed file, even across renames)

git diff branchB...BranchA (show the diff of what is in branchA that is not in branchB)

- Tracking Path Changes - Versioning file removes and path changes

git rm ←file name→ (delete the file from project)

get mv ←existing path→←new path→ (change existing file path)

Resolving Merge Conflicts

An event that takes place when Git is unable to automatically resolve differences in code between two commits.

If we have made a change in the code in one branch and made another change int he same code in another branch, then this conflict arises.

Then we have to manually accept which change you want.

Fork

It is a new repository that shares code and visibility settings with the original “upstream” repository.

It is a rough copy

Workflow

GitHub Repository → Clone It → Make Changes → Add It → Commit It → Push It

Cheat Sheet Link

https://education.github.com/git-cheat-sheet-education.pdf

https://git-scm.com/cheat-sheet

Git and GitHub - Roadmap.sh

What is Version Control?

- VCS is a tool that helps developers to manage changes to their code over time. It allows multiple versions of a project to exist simultaneously - makes it easier to collaborate with other developers & maintains a record of all modifications.
- It records changes to files, such as code, documents, or configuration files, and stores them in a repository.

- Examples of VSC - Git, Mercurial, Subversion, Perforce, CVA, etc.

What is a Repository?

- A repository is a storage location for your projects code, documentation and other files. It is the highest unit of storage.
- It serves as a central hub for collaboration, version control, and code management. It allows multiple people to work on the same project without overwriting each other's work.
- The `git init` command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.
- The `git config` command is a convenience function that is used to set Git configuration values on a global or local project level.
- Local configuration: Run `git config --local [key] [value]` to set a local configuration setting for the current repository.
- Global configuration: Use `git config --global [key] [value]` to set a global configuration setting that applies to all repositories on your system.
- GitHub offers both private and public repositories.
    - Public repositories are visible to everyone on the internet, making them ideal for open-source projects, collaboration, and showcasing work to a wider audience. They encourage community contributions and can help developers build their portfolios.
    - Private repositories, on the other hand, are only accessible to the repository owner and designated collaborators. These are suitable for proprietary code, sensitive projects, or work that's not ready for public consumption.

What is Working Directory?

- A working directory in Git is the local environment where files are stored and modified as part of a project.

- It reflects the current state of the project's files, allowing developers to edit, add, or delete files.
- The working directory is connected to the Git repository, and it helps manage the differences between the committed history and the current state of the files.
- It plays a central role in tracking changes, testing, and developing new features.

What is Staging Area?

A staging area serves as an intermediate step between your local repository changes and the actual commit.

- Temporary storage: The staging area holds changes that are intended to be part of the next commit.
- Previewing changes: It allows you to preview your changes before committing them.

GitHub Essentials

These are the core features and functionalities that form the foundation of GitHub's version control and collaboration platform.

- **Branch** - They separate streams of works. Isolating related changes to a specific branch allows you to control and introduce those changes independently. All repositories contain a default branch (typically named "main").
- **Fork** - It is a copy of a main GitHub repository. But it's special in the sense that GitHub maintains a connection back to the main/parent repository.
- **Articles subdirectory -** You can typically find a main articles directory off the root of the repository. The `articles` directory contains a set of subdirectories Articles in the subdirectories are formatted as Markdown files that use an *.md* extension.
- **Media subdirectory -** Each article directory contains a `/media` subdirectory for corresponding media files. Media files contain images used by articles that

have image references.

- **Includes subdirectory -** Whenever we have reusable content that is shared across two or more articles, it is placed in an `/includes` subdirectory off the main `articles` directory.
- **Origin** - This term is the name assigned to the connection between your local repository and the repository from which it was cloned. In the Microsoft Learn workflow, the origin represents the connection to your fork.
- **Pull Request** - It is a request for a content owner to pull your changes into the official source. It is a proposal to merge a set of changes from one branch into another. Here, collaborators can review and discuss the proposed set of changes before they integrate the changes into the main codebase.
- **Remote** - It is a named connection to a remote repository. In the Microsoft Learn workflow, a remote is always a GitHub repository.
- **Upstream** - It represents the connection between your local repository and the main repository from which your fork was created.
- **Commit** - It's a way to record changes you made. Each commit has a unique ID and a note about what was changed.

Branching Basics

Branch is a unit of storage that contains the files and folders that make up a projects' content sets.

Collaboration

Git provides tools to facilitate collaboration through forking repositories using `git clone` or `git fork`, cloning them locally with git clone, managing pull requests with `git request-pull`, and resolving merge conflicts.

Creating a Branch

- It allows you to work on different features or fixes without affecting the main codebase. It represents an independent line of development.
- You can create branches either through terminal or GitHub interface.

- The `git branch` command lets you create, list, rename, and delete branches

Renaming a Branch

It means changing the name of the branch to something different while preserving its history and the commits it contains

Deleting a Branch

It means removing a line of development from your Git repository. A branch in Git is essentially a pointer to a specific commit. When you delete a branch, you're removing this pointer, making that line of development no longer accessible through the branch name.

Checkout from a Branch

It means to switch your working directory to that branch, making it the active branch. This updates your files to match the state of that branch and allows you to work on it.

Merging a Branch

It is the process of combining changes from one branch into another. When you want to integrate updates from one branch (the source) into another branch (the target), you need to perform a merge. This involves resolving conflicts between the two branches, if any exist.

Git Remote

- It is a reference to a repository that exists on another server or system. It allow you to access and interact with a copy of your repository that is stored elsewhere, making it possible to collaborate with others, share your work, and maintain multiple copies of your repository for backup and disaster recovery purposes.
- When you add a remote to your local repository, Git creates a reference to the remote repository, enabling you to push changes from your local repository to the remote one, pull changes from the remote to your local one, or fetch changes from the remote without updating your local copy.

Fetch without Merge

- Running `git fetch` retrieves changes from a remote repository into your local clone, but does not automatically merge any of these changes into your local working directory. This is different from `git pull`, which both fetches and merges remote changes.
- By using fetch without merge, you can ensure that your local clone is up-to-date with the latest information from the remote repository, while leaving your working directory unchanged.

Pushing/Pulling Changes

- Pull operation updates your local branch with the latest changes from the remote branch
- Push operation sends your local commit to a remote repository

Managing Remote

A remote repository refers to a copy of a project's source code stored on a server or other machine.

- Adding remotes: Use `git remote add [name] [url]` to add a new remote repository. This allows you to track changes and push/pull updates from the remote.
- Listing remotes: Run `git remote -v` to list all configured remotes with their URLs.
- Renaming remotes: Update the name of an existing remote using `git remote rename [old-name] [new-name]`.
- Deleting remotes: Remove a remote repository with `git remote remove [name]`.

Cloning Repository

It involves creating local copy of a remote repository on your computer. This allows you to work on the project locally, commit changes, and later push those changes back to the remote repository.

Cloning a repository means creating a local copy of a repository that exists on a remote server. When you fork a repository, you create a copy of someone else's repository in your own account.

Issues

It is a way to track and report bugs, feature requests, or other problems with a repository. Here are some key aspects of issues:

- Creating issues: Users can create new issues by submitting a form on the repository's Issues page.
- Issue titles and descriptions: Each issue has a title and body (description), which provide context for the problem or request.
- Assignees: Issues can be assigned to specific users, who are then responsible for addressing the issue.
- Labels: Labels are used to categorize issues by topic, priority, or other criteria. This helps filter and organize issues within a repository.
- States: Issues have states that reflect their status, such as "Open", "Closed", or "Pending".
- Comments: Users can comment on existing issues to discuss or provide additional context.
- Labels and milestones: Issues can be associated with labels (topics) and milestones (deadlines), which help filter and prioritize them.

Project Readme

- It is a crucial document that serves as the front page of a repository, providing essential information about the project.
- It typically includes a brief description of the project's purpose, installation instructions, usage guidelines, and contribution procedures.
- It often contains badges indicating build status, code coverage, and other metrics, as well as links to documentation, issue trackers, and community channels.

Commit Messages

It is a brief explanation of the changes introduced in a particular commit. It helps others (and your future self) understand the purpose of the changes and the context behind them.

Code Reviews

The purpose of a code review in software development is to help ensure that the code meets the organization's standards and requirements, is of high quality, and is maintainable. In addition to identifying errors and bugs, code reviews also promote a culture of learning and collaboration among the development team.

- Increase code quality by identifying defects in the code
- Ensure compliance with organizational standards, regulations, and the team's code style.
- Save time and money by detecting issues earlier
- Boost collaboration, communication, and knowledge sharing among developers

Collaboration on GitHub

It is a powerful way for multiple people to work together on the same project, using Git as the version control system.

- Collaborators and Members refer to individuals who contribute to or have access to your repository.
- Collaborators are users who have been granted permission to contribute code, make changes, and push updates to your repository.
- Members are the owners of a repository, including organization owners who have full control over their team's repositories.

GitHub Actions

- It is a powerful automation and continuous integration/continuous deployment (CI/CD) platform provided by GitHub.

- It allows developers to create custom workflows that automatically build, test, and deploy their code directly from their GitHub repositories.
- These workflows are triggered by specific events, such as push requests, pull requests, or scheduled tasks.
- GitHub Actions enables teams to streamline their development processes, improve code quality, and accelerate software delivery by automating repetitive tasks and integrating various tools and services seamlessly within their development pipeline.
- Workflow Triggers - They are events that initiate a GitHub Actions workflow. They can be scheduled, triggered by code changes, or manually initiated. This allows for automation of tasks based on specific conditions.

YAML Syntax

YAML (YAML Ain't Markup Language) is a human-readable data serialization standard for all programming languages. It is designed to be easily readable by humans while also being machine-parsable. Key features of YAML include:

1. **Simplicity:** YAML uses a minimalist syntax with significant whitespace and indentation.
2. **Versatility:** It can represent various data types, including scalars, lists, and associative arrays.
3. **Readability:** Its clear, concise format makes it easy for both humans and machines to understand.
4. **Language-independent:** YAML can be used with any programming language that has a YAML parser.

YAML is commonly used for:

- **Configuration files:** Many applications and tools use YAML for their configuration settings.
- **Data exchange:** It serves as a lightweight alternative to XML or JSON for data transfer between systems.

- **Data storage:** YAML can be used to store structured data in a human-readable format.
- **DevOps and CI/CD:** It's widely used in tools like Docker, Kubernetes, and various CI/CD platforms for defining workflows and configurations.

Committing changes in Git is a crucial part of version control, allowing you to save your progress and record a snapshot of your project's current state.

Ignored files are tracked in a special file named `.gitignore` that is checked in at the root of your repository. There is no explicit git ignore command: instead the `.gitignore` file must be edited and committed by hand when you have new files that you wish to ignore.

Viewing commit history is a crucial aspect of Git, allows users to examine the chronological record of repository changes. This is essential for understanding project evolution, tracking modifications, and facilitating effective team collaboration. Git provides various commands like `git log` and its options (e.g., `--oneline`, `--graph`, `--patch`, `--stat`) to display commit history in different formats. Users can filter commits by author, date range, and other criteria.

GitHub provides features to securely store and manage sensitive data, such as secrets and environment variables.

- Secrets: These are sensitive values that should not be committed to a repository, like API keys or database credentials.
- Environment Variables: They can be used to set values for your workflow or application, making it easier to manage dependencies.