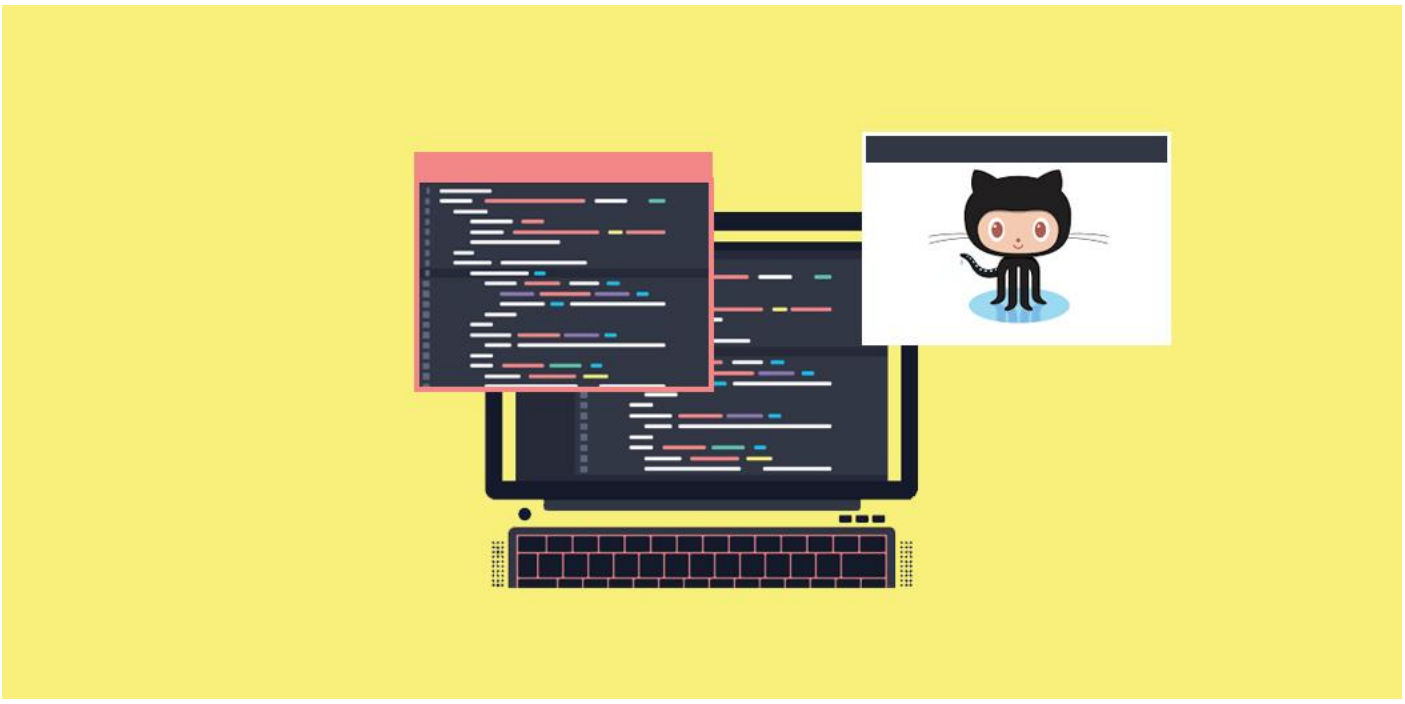
Created By: Dhruv Singhal | LinkedIn

What Is GitHub?

GitHub is a web-based platform used for version control and collaborative software development. It's built around Git, a distributed version control system, which helps developers track changes to their codebase and collaborate with others efficiently. GitHub offers a variety of features and tools that enhance software development, making it an essential platform for developers worldwide.

GitHub



Key Features of GitHub:

1. Version Control with Git

- **Repository (Repo):** A storage space for your project, which can contain files, folders, and the history of changes.

- **Branching:** Developers can create branches to work on different features or bug fixes independently.
- **Pull Requests:** A process to propose changes to the codebase, enabling review, discussion, and approval before merging.
- **Commit:** A snapshot of your project, capturing the changes made at a specific point.

2. Collaboration Tools

- **Code Review:** Team members can review and comment on each other's code to ensure quality.
- **Issues & Discussions:** Users can report bugs, suggest features, or discuss ideas.
- **Wiki:** A space for documentation related to the project.

3. Automation and CI/CD

- **GitHub Actions:** Automate workflows like testing, deployment, and more.
- **Continuous Integration/Continuous Deployment (CI/CD):** Automate building, testing, and deploying code.

4. Project Management

- **Projects:** Kanban-style boards for task management.
- **Milestones:** Group issues and pull requests under specific goals.

5. Security

- **Dependency Scanning:** Identifies vulnerabilities in dependencies.
- **Code Scanning:** Detects vulnerabilities in your code.
- **Secrets Management:** Helps store sensitive information securely.

6. Community and Open Source

- Many open-source projects are hosted on GitHub, allowing anyone to contribute.
- **Forking:** Users can copy a repository to make changes without affecting the original project.
- **Stars:** Developers can star repositories to show interest or keep track of them.

7. Integration and Extensibility

- **APIs:** GitHub provides APIs for automation and integration with other tools.
- **Third-party Apps and Plugins:** Extend GitHub functionality.

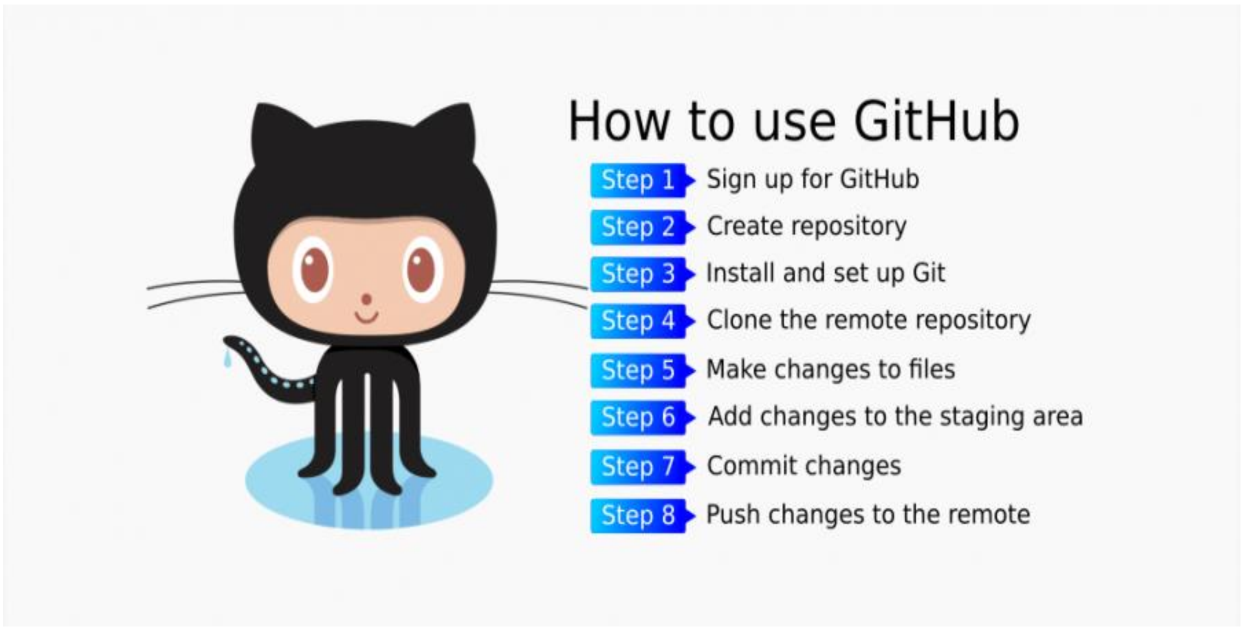
Use Cases for GitHub:

1. **Individual Projects:** Version control and personal portfolio.
2. **Team Collaboration:** Software development teams can work on projects together.
3. **Open Source Contribution:** Users can contribute to open-source projects globally.
4. **Project Management:** Manage tasks and milestones effectively.
5. **Learning and Education:** Many developers use GitHub to share learning resources and projects.

GitHub Terminology:

- **Fork:** A personal copy of another user's repository.
- **Clone:** A local copy of a repository stored on your machine.

- **Merge:** Combining changes from one branch into another.
- **Remote:** A version of your project hosted on GitHub or another Git repository.


How to use GitHub
Step 1 Sign up for GitHub
Step 2 Create repository
Step 3 Install and set up Git
Step 4 Clone the remote repository
Step 5 Make changes to files
Step 6 Add changes to the staging area
Step 7 Commit changes
Step 8 Push changes to the remote

Core Concepts of GitHub

1. Repositories (Repos)

A **repository** is the core component of GitHub. It serves as a project workspace, containing:

- **Code files** (source code, documentation).
- **Version history** of all changes made to the project.
- **Metadata** like issues, pull requests, and discussions.

Repositories can be:

- **Public:** Open for anyone to view and contribute.
- **Private:** Restricted access, typically used for proprietary projects.

2. Branches

Branches allow parallel development without affecting the main codebase.

- **Main Branch (default):** Represents the stable version of your project.
- **Feature Branches:** Used for developing specific features or fixing bugs.

Key Commands:

- `git branch <branch_name>` – Create a new branch.

- `git checkout <branch_name>` – Switch to a branch.

3. Commits

A **commit** is a snapshot of changes made to the codebase. Each commit is:

- **Tracked:** With a unique SHA identifier.
- **Reversible:** You can revert changes to any previous commit.

Key Components of a Commit:

- **Commit Message:** Describes what changes were made.
- **Commit Author:** The person who made the changes.

4. Pull Requests (PRs)

A **Pull Request** (PR) is a method to propose changes in one branch and merge them into another (e.g., from a feature branch to the main branch).

PR Workflow:

1. Developer creates a PR.
2. Reviewers comment, suggest changes, or approve.
3. Changes are merged after approval.

5. GitHub Actions

GitHub Actions enable **automation of workflows**. You can set up CI/CD pipelines, automate testing, or even deploy applications.

Examples:

- Automatically run tests on every PR.
- Deploy code to production after merging.

Key Concepts:

- **Workflow Files:** Define automation steps in YAML format.
- **Triggers:** Events like `push`, `pull_request`, or `schedule`.

6. Issues and Project Management

GitHub provides tools for tracking tasks, bugs, and feature requests:

- **Issues:** For bug reports, feature requests, or general discussion.
- **Projects:** Kanban-style boards for visual task management.
- **Milestones:** Group related issues and PRs under a specific goal.

7. Security Features

GitHub enhances project security through various tools:

- **Dependabot Alerts:** Automatically scans dependencies for known vulnerabilities.
- **Secret Scanning:** Detects accidentally committed sensitive data (e.g., API keys).
- **Code Scanning:** Analyzes code for vulnerabilities and potential security risks.

8. Collaborative Features

GitHub is designed for team collaboration:

- **Forks:** Users can create a personal copy of a repository to make changes without affecting the original.
- **Code Review:** Team members can review and comment on each other's code.
- **Discussions:** Open-ended threads for brainstorming or general project discussion.

GitHub Ecosystem

1. GitHub Desktop

A GUI tool for managing repositories without using the command line. It simplifies:

- Cloning repositories.
- Managing branches.
- Committing and pushing changes.

2. GitHub CLI (Command Line Interface)

A tool to interact with GitHub from the terminal. Examples:

- `gh repo create` – Create a new GitHub repository.
- `gh pr create` – Create a pull request.

3. Integrations and APIs

GitHub integrates with various tools and services:

- **Slack, Jira, Trello:** For project communication and task management.
- **GitHub API:** Automate and extend GitHub's functionality programmatically.

GitHub Workflows

Basic Workflow

1. **Clone a Repository**: `git clone <repository_url>`

2. **Create a Branch**: `git checkout -b <branch_name>`

3. **Make Changes and Commit**:
```
git add .
git commit -m "Descriptive commit message"
```

4. **Push Changes**: `git push origin <branch_name>`

Forking Workflow (for open-source contribution)

1. **Fork the Repository**:
   - Create your own copy of the original project.

2. **Clone the Fork**: `git clone <fork_url>`
3. **Add Remote to Original Repository**: `git remote add upstream <original_repo_url>`
4. **Make Changes and Create a Pull Request**.

Feature Branch Workflow

Useful for working on multiple features in parallel.

1. Start with the latest code: `git pull origin main`
2. Create a new branch for each feature: `git checkout -b feature/<feature-name>`
3. After completing the feature, create a pull request.

Open Source and Community

GitHub is home to millions of open-source projects:

- **Contribution Guide:** Each project typically has a `CONTRIBUTING.md` file with guidelines.
- **Licenses:** Projects specify licenses (e.g., MIT, GPL) in a `LICENSE` file.

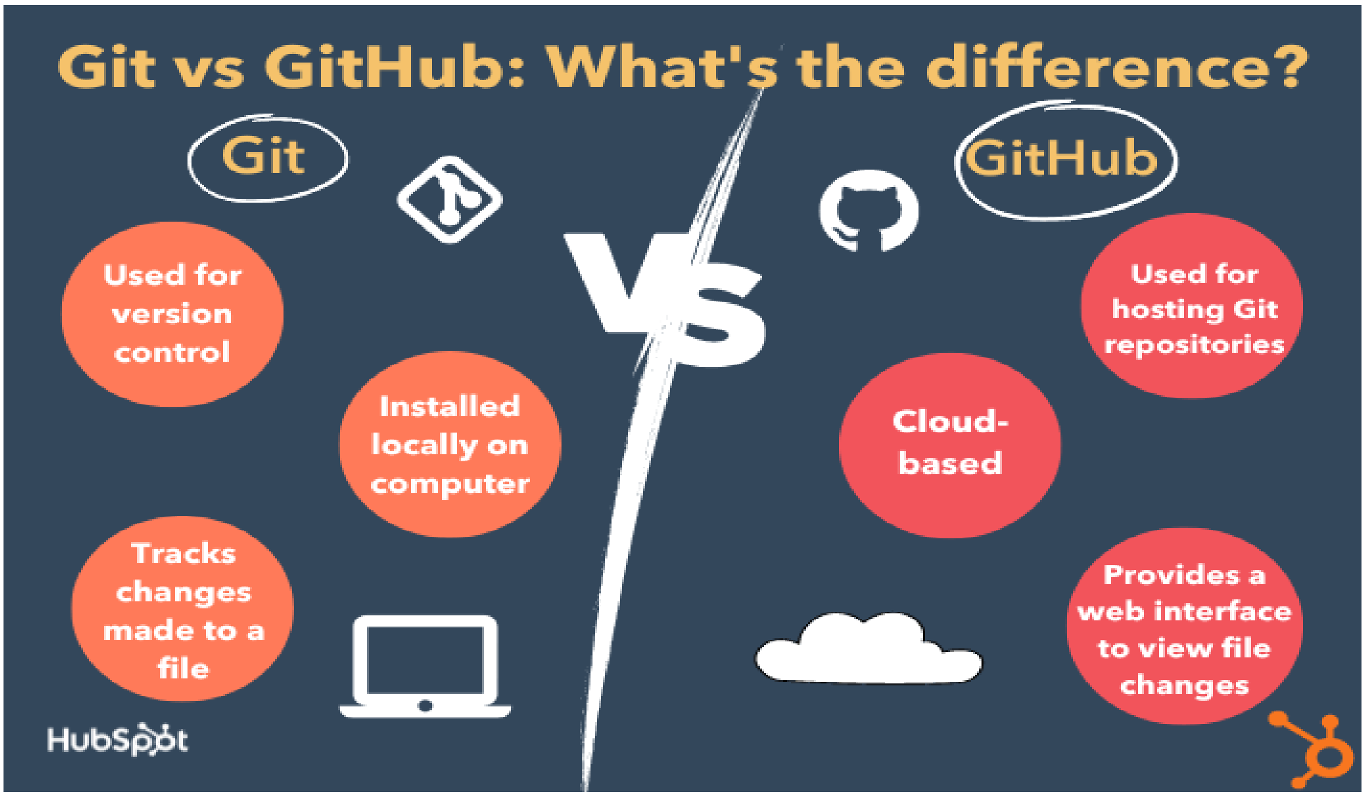
GitHub Plans:

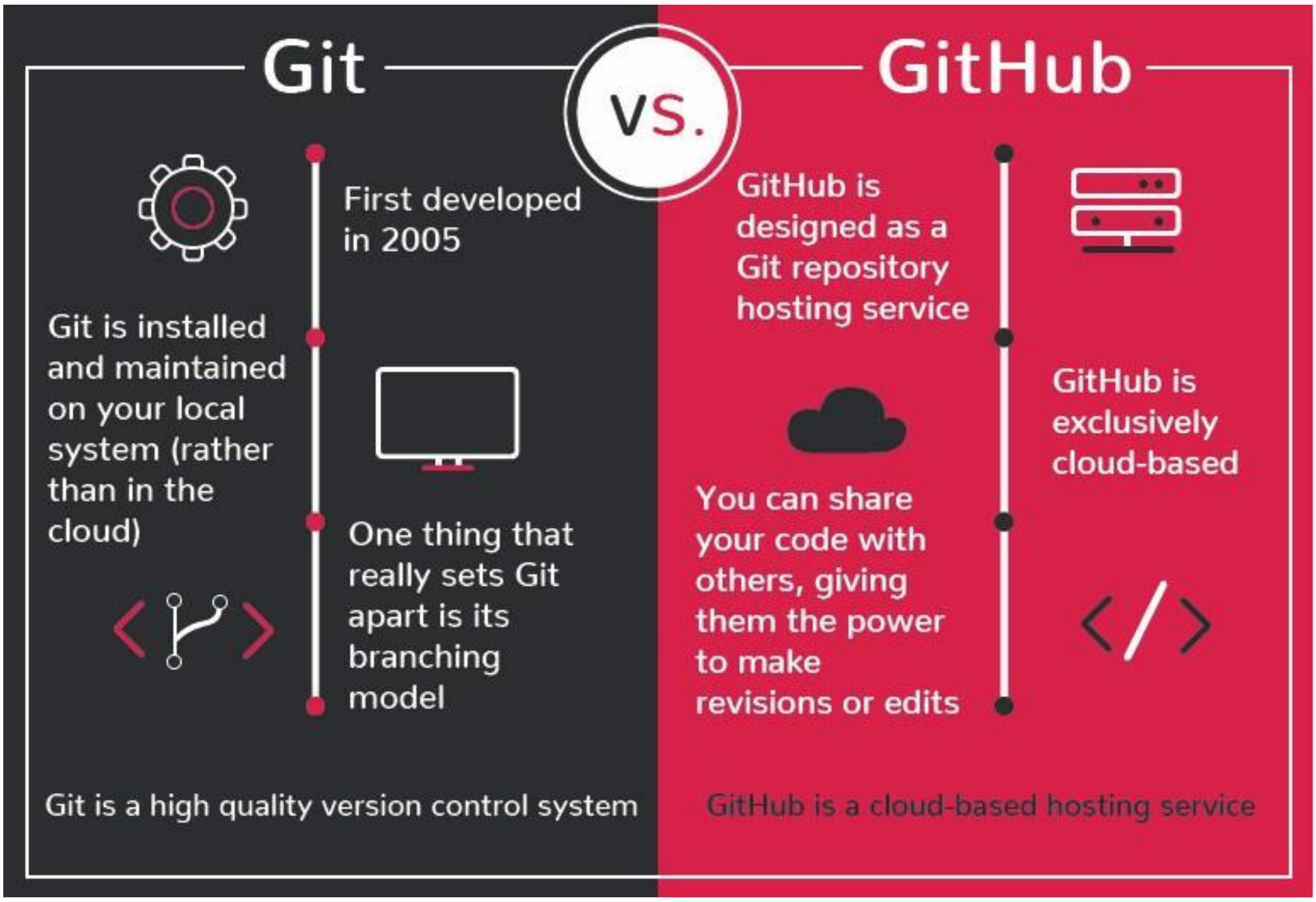
1. **Free Plan**:
   - Unlimited public and private repositories.
   - Limited CI/CD minutes.

2. **Pro Plan** (For Individuals):
   - Advanced collaboration features.
   - Additional CI/CD minutes.

3. **Team and Enterprise Plans**:
   - Best for organizations needing more control, security, and support.

Git vs GitHub


Git vs GitHub: What's the difference?
Git
Used for version control
Installed locally on computer
Tracks changes made to a file
VS
GitHub
Used for hosting Git repositories
Cloud-based
Provides a web interface to view file changes
HubSpot


Git
VS.
GitHub
First developed in 2005
Git is installed and maintained on your local system (rather than in the cloud)
One thing that really sets Git apart is its branching model
Git is a high quality version control system
GitHub is designed as a Git repository hosting service
GitHub is exclusively cloud-based
You can share your code with others, giving them the power to make revisions or edits
GitHub is a cloud-based hosting service

Created By: Dhruv Singhal | LinkedIn

1. What is Git?

Git is a **distributed version control system (VCS)** developed by Linus Torvalds in 2005. It's a tool used to track changes in source code during software development. Git helps manage code history and facilitates collaboration by allowing multiple developers to work on a project simultaneously.

Key Features of Git:

- **Version Control**: Tracks changes, maintains history, and allows rollback to previous versions.
- **Branching and Merging**: Developers can work on isolated branches and merge them later.
- **Distributed System**: Every developer has a complete copy of the project, including its full history.
- **Offline Work**: Most Git operations (committing, branching, etc.) can be performed without an internet connection.

Common Git Commands:

- `git init`: Initialize a new repository.
- `git add`: Stage changes.
- `git commit`: Save staged changes.
- `git checkout`: Switch branches.
- `git merge`: Merge changes from one branch to another.

2. What is GitHub?

GitHub is a **web-based platform** that provides a hosting service for Git repositories. It adds collaboration features on top of Git, such as issue tracking, code review, and integration with other tools.

Key Features of GitHub:

- **Repository Hosting**: Centralized location for storing Git repositories.
- **Pull Requests**: Mechanism to propose, discuss, and review changes.
- **Issues and Discussions**: Tools for tracking tasks, bugs, and feature requests.
- **CI/CD (GitHub Actions)**: Automate workflows for testing, deployment, and more.
- **Social Features**: Users can follow repositories, star projects, and contribute to open-source projects.

Web Interface:

GitHub provides a user-friendly web interface for managing repositories, viewing commit history, reviewing pull requests, and more.

Key Differences Between Git and GitHub

Feature	Git	GitHub
Purpose	Version control system for tracking code changes.	Platform for hosting Git repositories and collaboration.

Feature	Git	GitHub
Type	Command-line tool.	Web-based service with a GUI.
Functionality	Focuses on version control and local repository.	Adds collaboration, issue tracking, and automation.
Repository Location	Local (on your machine).	Hosted on the cloud (GitHub servers).
Collaboration	Limited to direct sharing (e.g., email patches).	Enables team collaboration via pull requests and reviews.
Work Environment	Works offline.	Requires internet for accessing hosted repos.
Automation	No built-in automation.	Supports CI/CD pipelines with GitHub Actions.
Community	No social or community features.	Large community with social features (stars, forks).
Pricing	Free and open-source.	Free for public repos, paid plans for advanced features.

How Git and GitHub Work Together

1. **Local Repository with Git**:
   - Developers use Git locally to manage their codebase, track changes, and experiment on different branches.
2. **Remote Repository on GitHub**:
   - GitHub serves as a remote backup and collaboration platform for Git repositories.

Example Workflow:

1. **Clone a GitHub Repository**:
   git clone https://github.com/username/repo.git
2. **Work Locally with Git:**
   Make changes, commit, and test locally.
3. **Push Changes to GitHub:**
   git push origin main
4. **Collaborate Using GitHub:**

   1) Open a pull request for code review.
   2) Discuss changes with team members.
   3) Merge approved changes into the main branch.

Created By: Dhruv Singhal | LinkedIn

Which One Should You Use?

- **Git** is essential for any software project, even if you're working alone, as it provides powerful version control features.
- **GitHub** is ideal for collaboration, project management, and sharing code, especially for open-source projects or teams.

Essential Git Commands for GitHub

To interact with GitHub, you primarily use **Git commands**. Below is a comprehensive list of commonly used Git commands, categorized for easy reference.

1. Configuration and Setup

- `git config --global user.name "Your Name"`
  Sets your Git username globally.

- `git config --global user.email "your.email@example.com"`
  Sets your Git email globally.

- `git config --list`
  Displays all configuration settings.

2. Repository Management

- `git init`
  Initializes a new Git repository in your local directory.

- `git clone <repository_url>`
  Creates a local copy of a remote repository.

- `git remote add origin <repository_url>`
  Links your local repository to a remote repository.

- `git remote -v`
  Lists the remote repositories linked to your local repository.

3. Basic File Operations

- `git add <file>`
  Adds specific files to the staging area.

- `git add .`
  Adds all files (modified, new, or deleted) to the staging area.

- `git rm <file>`
  Removes a file from the working directory and staging area.

4. Committing Changes

- `git commit -m "Commit message"`
  Commits staged changes with a message.

- `git commit -a -m "Commit message"`
  Skips the staging area and commits all changes to tracked files.

- `git commit --amend`
  Amends the previous commit (e.g., to fix a message or add missed files).

5. Branching and Merging

- `git branch`
  Lists all branches in the repository.

- `git branch <branch_name>`
  Creates a new branch.
- `git checkout <branch_name>`
  Switches to the specified branch.

- `git checkout -b <branch_name>`
  Creates and switches to a new branch.

- `git merge <branch_name>`
  Merges the specified branch into the current branch.

- `git branch -d <branch_name>`
  Deletes the specified branch locally.

6. Synchronizing with Remote Repositories

- `git push origin <branch_name>`
  Pushes changes from your local branch to the corresponding branch on GitHub.

- `git push -u origin <branch_name>`
  Pushes and sets the upstream branch for future pushes.

- `git fetch`
  Downloads changes from the remote repository but doesn’t apply them.

- `git pull`
  Fetches changes from the remote repository and merges them into your local branch.

- `git push`
  Pushes changes from all local branches to their corresponding remote branches.

7. Status and Logs

- `git status`
  Shows the current state of the working directory and staging area.
- `git log`
  Displays a history of commits.
- `git log --oneline`
  Shows a concise one-line summary of each commit.
- `git diff`
  Shows differences between files in the working directory and the staging area.
- `git diff <commit1> <commit2>`
  Compares changes between two commits.

8. Stashing and Cleaning

- `git stash`
  Saves changes in a temporary area and reverts the working directory to the last commit.
- `git stash pop`
  Applies the most recent stash and removes it from the stash list.
- `git stash list`
  Displays a list of stashed changes.
- `git stash apply`
  Applies a specific stash but keeps it in the stash list.
- `git clean -f`
  Removes untracked files from the working directory.

9. Rewriting History

- `git reset <file>`
  Unstages a file from the staging area but keeps changes in the working directory.
- `git reset --hard`
  Resets the working directory and staging area to the last commit.
- `git reset --soft <commit>`
  Resets to a specific commit but retains changes in the staging area.
- `git revert <commit>`
  Creates a new commit that undoes the changes made in a previous commit.

10. Tagging

- `git tag <tag_name>`
  Creates a lightweight tag for the current commit.

- `git tag -a <tag_name> -m "Tag message"`
  Creates an annotated tag with a message.

- `git push origin <tag_name>`
  Pushes the tag to the remote repository.

- `git tag -d <tag_name>`
  Deletes a tag locally.

11. Collaboration and Review

- `git blame <file>`
  Shows who last modified each line of a file.

- `git shortlog`
  Summarizes commit history by author.

12. Advanced Commands

- `git cherry-pick <commit>`
  Applies a specific commit to the current branch.

- `git rebase <branch>`
  Re-applies commits from the current branch onto another branch.

- `git bisect`
  Helps to find the commit that introduced a bug by binary searching through the commit history.

13. GitHub-Specific Commands (GitHub CLI)

If you use the **GitHub CLI (gh)**, here are some additional commands:

- `gh repo clone <repository>`
  Clone a GitHub repository.

- `gh repo create`
  Create a new repository on GitHub.

- `gh issue list`
  List issues in a repository.

- `gh pr create`
  Create a pull request.

- `gh pr merge`
  Merge a pull request.

For more information, see following:

https://dev.to/nopenoshishi/understanding-git-through-images-4an1

https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

https://learn.microsoft.com/en-us/contribute/content/git-github-fundamentals

https://github.com/topics/documentation

https://git-scm.com/doc