GIT COMMAND LINE BASICS

[Advantages of GIT Version Control 1](#_Toc190724922)

[Setting Up GIT 3](#_Toc190724923)

[Initialize a new repository 3](#_Toc190724924)

[Clone an existing repository 3](#_Toc190724925)

[Basic Workflow 3](#_Toc190724926)

[Check the status of your repository 3](#_Toc190724927)

[Add files to the staging area 3](#_Toc190724928)

[To add all changes 3](#_Toc190724929)

[Commit changes 3](#_Toc190724930)

[Branching and Merging 4](#_Toc190724931)

[Create a new branch 4](#_Toc190724932)

[Switch to a branch 4](#_Toc190724933)

[Create and switch to a new branch 4](#_Toc190724934)

[List all branches 4](#_Toc190724935)

[Merge a branch into the current branch 4](#_Toc190724936)

[Delete a branch 4](#_Toc190724937)

[Remote Repositories 5](#_Toc190724938)

[Add a remote repository 5](#_Toc190724939)

[View remote repositories 5](#_Toc190724940)

[Push changes to a remote repository 5](#_Toc190724941)

[Pull changes from a remote repository 5](#_Toc190724942)

[Viewing History 5](#_Toc190724943)

[View commit history 5](#_Toc190724944)

[View a simplified commit history 5](#_Toc190724945)

[View changes in a specific commit 5](#_Toc190724946)

[Undoing Changes 6](#_Toc190724947)

[Unstage a file 6](#_Toc190724948)

[Revert changes in a file 6](#_Toc190724949)

[Amend the last commit 6](#_Toc190724950)

[Revert a commit 6](#_Toc190724951)

[Stashing 6](#_Toc190724952)

[Stash changes 6](#_Toc190724953)

[Apply stashed changes: 6](#_Toc190724954)

[List all stashes 6](#_Toc190724955)

[Tags 6](#_Toc190724956)

[Create a tag 6](#_Toc190724957)

[Push tags to remote 6](#_Toc190724958)

[Fetching and Pulling 7](#_Toc190724959)

[Fetch changes from remote 7](#_Toc190724960)

[Pull changes and merge 7](#_Toc190724961)

[Ignoring Files 7](#_Toc190724962)

[Sample Workflow 7](#_Toc190724963)

Advantages of GIT Version Control

**1. Version Control and History**

* **Track Changes**: Git keeps a complete history of changes, allowing you to see who made changes, what was changed, and when.
* **Revert to Previous States**: If something goes wrong, you can easily revert to a previous version of your project.
* **Branching and Merging**: Git allows you to create branches to work on new features or experiments without affecting the main codebase. You can later merge these branches back into the main branch.

**2. Collaboration**

* **Multiple Contributors**: Git enables multiple developers to work on the same project simultaneously without overwriting each other's work.
* **Conflict Resolution**: Git provides tools to resolve conflicts when changes from different contributors overlap.
* **Remote Repositories**: Platforms like GitHub, GitLab, and Bitbucket allow teams to collaborate on projects hosted in remote repositories.

**3. Backup and Redundancy**

* **Distributed System**: Every developer's working copy of the code is a full backup of the repository, reducing the risk of data loss.
* **Remote Repositories**: Remote repositories act as additional backups, ensuring that your code is safe even if local copies are lost.

**4. Branching and Experimentation**

* **Feature Branches**: You can create separate branches for new features, bug fixes, or experiments, keeping the main branch stable.
* **Easy Merging**: Git makes it easy to merge branches back into the main codebase once the work is complete and tested.

**5. Efficiency and Performance**

* **Fast Operations**: Git is designed to be fast, even for large projects with extensive histories.
* **Offline Work**: You can commit changes, create branches, and view history without an internet connection. You only need to connect to sync with remote repositories.

**6. Open Source and Community Support**

* **Wide Adoption**: Git is widely used in the software development community, meaning there is extensive documentation, tutorials, and community support available.
* **Integration**: Git integrates with many development tools, including IDEs, continuous integration/continuous deployment (CI/CD) pipelines, and project management tools.

**7. Flexibility and Workflow Customization**

* **Custom Workflows**: Git supports various workflows (e.g., Git Flow, GitHub Flow) that can be tailored to your team's needs.
* **Hooks and Automation**: Git allows you to set up hooks to automate tasks like running tests or deploying code when certain actions are performed.

**8. Auditability and Accountability**

* **Detailed Logs**: Git provides detailed logs of who made changes and when, which is useful for auditing and accountability.
* **Code Reviews**: Platforms like GitHub and GitLab facilitate code reviews, ensuring that changes are reviewed and approved before being merged.

**9. Scalability**

* **Handles Large Projects**: Git is designed to handle projects of all sizes, from small personal projects to large enterprise-level applications.
* **Efficient Storage**: Git uses compression and delta encoding to store changes efficiently, reducing the storage footprint.

**10. Cross-Platform Compatibility**

* **Works on Multiple OS**: Git is compatible with Windows, macOS, and Linux, making it versatile for teams using different operating systems.

**11. Open Source**

* **Free to Use**: Git is open-source software, meaning it's free to use and can be modified to suit specific needs.

**12. Community and Ecosystem**

* **Rich Ecosystem**: Git has a rich ecosystem of tools and extensions that enhance its functionality, such as GUI clients, CI/CD integrations, and more.

**13. Security**

* **Data Integrity**: Git uses SHA-1 hashing to ensure the integrity of your data, making it difficult for changes to go unnoticed.
* **Access Control**: Remote repositories often provide access control mechanisms to restrict who can view or modify the code.

**14. Documentation and Metadata**

* **Commit Messages**: Commit messages provide a way to document changes, making it easier to understand the history and rationale behind changes.
* **Tags and Releases**: Git allows you to tag specific points in history as important (e.g., releases), making it easy to reference and deploy specific versions.

**Conclusion**

Git is a powerful and versatile tool that enhances collaboration, efficiency, and reliability in software development. Its distributed nature, robust branching model, and extensive ecosystem make it an essential tool for developers and teams of all sizes.

Setting Up GIT

Initialize a new repository

git init

Example:

git init my-project

# This will create a new folder called my-project and initialize git

Clone an existing repository

git clone <repository-url>

Example:

git clone https://github.com/user/repo.git

Basic Workflow

Check the status of your repository

git status

Add files to the staging area

git add <file-name>

Example:

git add index.html

To add all changes

git add .

Commit changes

git commit -m "Your commit message"

Example:

git commit -m "Added index.html file"

Branching and Merging

Create a new branch

git branch <branch-name>

Example:

git branch feature-login

Switch to a branch

git checkout <branch-name>

Example:

git checkout feature-login

Create and switch to a new branch

git checkout -b <branch-name>

Example:

git checkout -b feature-signup

List all branches

git branch

Merge a branch into the current branch

git merge <branch-name>

Example:

git merge feature-login

Delete a branch

git branch -d <branch-name>

Example:

git branch -d feature-login

Remote Repositories

Add a remote repository

git remote add <name> <repository-url>

Example:

git remote add origin https://github.com/user/repo.git

View remote repositories

git remote -v

Push changes to a remote repository

git push <remote-name> <branch-name>

Example:

git push origin main

Pull changes from a remote repository

git pull <remote-name> <branch-name>

Example:

git pull origin main

Viewing History

View commit history

git log

View a simplified commit history

git log --oneline

View changes in a specific commit

git show <commit-hash>

Example:

git show abc1234

Undoing Changes

Unstage a file

git reset <file-name>

Example:

git reset index.html

Revert changes in a file

git checkout -- <file-name>

Example:

git checkout -- index.html

Amend the last commit

git commit --amend

Revert a commit

git revert <commit-hash>

Example:

git revert abc1234

Stashing

Stash changes

git stash

Apply stashed changes:

git stash apply

List all stashes

git stash list

Tags

Create a tag

git tag <tag-name>

Example:

git tag v1.0.0

Push tags to remote

git push --tags

Fetching and Pulling

Fetch changes from remote

git fetch <remote-name>

Example:

git fetch origin

Pull changes and merge

git pull <remote-name> <branch-name>

Example:

git pull origin main

Ignoring Files

Create a .gitignore file to exclude files/folders from being tracked:

# Example .gitignore file

node\_modules/

.env

\*.log

Sample Workflow

Clone a repository

git clone https://github.com/user/repo.git

Create a new branch

git checkout -b feature-branch

Make changes and stage them

git add .

Commit changes

git commit -m "Added new feature"

Push changes to remote

git push origin feature-branch

Merge changes into main

git checkout main

git pull origin main

git merge feature-branch

git push origin main