GIT Up and Go

A quick introduction to GIT

What is GIT?

GIT is a version control system that is used for software development and other version control tasks. It was created by Linus Torvalds in 2005 to manage the development of the Linux kernel. GIT allows developers to keep track of changes made to their code, collaborate with others on a project, and revert to previous versions if necessary.

GIT works by creating a repository (or "repo") that contains all the files and folders associated with a project. Developers can make changes to their code and then "commit" those changes to the repository. Each commit is like a snapshot of the code at a particular point in time, with information about who made the change and why.

GIT also supports branching and merging, which allows developers to work on multiple versions of the code at the same time. Each branch represents a separate line of development, and changes made to one branch can be merged into another when they are ready.

In addition to its use in software development, GIT can be used for a variety of version control tasks, including managing configuration files and tracking changes to documents. GIT is open source software, which means that it is free to use and can be modified by anyone.

Why GIT? #1

Git is a popular version control system that allows developers to efficiently manage their code changes and collaborate with others on a project. Here are some of the key benefits of using Git:

- Track changes: Git allows you to keep track of every change you make to your code, including who made the change, when it was made, and why it was made. This makes it easy to track down bugs and revert to previous versions if necessary.
- Collaboration: Git makes it easy for developers to collaborate on a project, even if they are working in different locations. Developers can work on their own branches of code and merge their changes together when they are ready.
- Branching and merging: Git allows developers to create multiple branches of their code, which can be worked on independently of each other. When a feature is ready, it can be merged back into the main branch.

Why GIT? #2

- Code reviews: Git makes it easy to review code changes before they are merged into the main branch. This ensures that code changes meet the project's standards and that bugs are caught early on.
- Backup and restore: Git is an excellent backup tool that can help you restore your code to a previous state if something goes wrong.

Overall, Git is a powerful tool that can help developers work more efficiently and collaborate more effectively.

Install GIT

1. Install Git: If you haven’t already done so, download and install Git on your computer. You can find the download link at **https://git-scm.com/**.
2. Open the terminal: To configure Git, you’ll need to use the terminal (or command prompt) on your computer. Open the terminal by searching for "Terminal" or "Command Prompt" in your operating system’s search bar.
3. Configure your username: The first thing you’ll need to do is configure your username in Git. This is the name that will be associated with the changes you make to your code. Type the following command in the terminal, replacing "Your Name" with your actual name:

```
git config --global user.name "Your Name"
```

If you don’t want to use your real name, you can use a nickname or pseudonym instead.

Configure GIT

1. Configure your email address: Next, you'll need to configure your email address in Git. This is the email address that will be associated with the changes you make to your code. Type the following command in the terminal, replacing "your.email@example.com" with your actual email address

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

2. Configure your default text editor: Git uses a text editor to display and edit text files, such as commit messages. You can configure your preferred text editor by typing the following command in the terminal, replacing "code" with the name of your preferred text editor:

```
git config --global core.editor code
```

3. Check your configuration: To check that your configuration has been set correctly, type the following command in the terminal:

```
git config --list # or git config --global --list
```

This will display a list of your Git configuration settings, including your name, email address, and text editor.

Initializing Git

1. Open the terminal: To initialize Git, you'll need to use the terminal (or command prompt) on your computer. Open the terminal by searching for "Terminal" or "Command Prompt" in your operating system's search bar.
2. Navigate to your repository: Use the "cd" command to navigate to the folder that you chose as your Git repository. For example, if your repository is located in the "SourceCode" folder, you would type "cd SourceCode" and press Enter.
3. Initialize Git: Type the following command to initialize Git in your repository:

```
git init
```

This will create a new Git repository in the folder you have navigated to. It creates an hidden folder to keep track of changes.

Adding files to a repository

Check the status of your repository: Type the following command to check the status of your Git repository:

```
git status
```

This will display a list of all the files in your repository and their status.

1. Stage or add files: To stage or add files to your Git repository, use the "git add" command followed by the name of the file or folder that you want to add. For example, if you want to add a file named "index.html", you would type:

```
git add index.html
```

You can also use the "git add" command with a wildcard (*) to add all files in a folder:

```
git add .
```

This will add all files in the current directory to the staging area.

Committing changes

1. Check the status of your repository again: After adding files to your repository, use the "git status" command again to check the status of your repository. You should see that the files you added are now listed as "changes to be committed."
2. Commit changes: Once you have staged or added files to your repository, you'll need to commit your changes. Use the "git commit" command followed by a message describing the changes you made. For example:

```
git commit -m "Add index.html file" # commit with message
```

This will create a new commit with the changes you made to the repository.

Making commits

```
git commit -m "my commit message"
```

Commit without staging you don’t need to stage the files with the command "git add ." but the files will be commit direcly.

```
git commit -a -m "my commit message"
```

Viewing commit history

Displays a list of commit history for a repository

```
git log # list all commits
```

Displays a condensed version of the commit history in a single line per commit.

```
git log --online # condensed version
```

Working with branches

Creating a new branch

```
git branch '<name>' # create a new branch <name>
```

Listing all available branches

```
git branch # list all available branches
```

Switching to another branch

```
git branch checkout '<name>' # switch to the branch <name>
```

Creating a new branch and switching to it

```
git checkout -b '<name>' # create a new branch <name> and switch to it
```

Deleting a branch

```
git branch -d '<name>' # delete the branch <name>
```

Merging two branches

```
git merge '<name>' # merge the branch <name> into the current branch
```

Working with remotes

Adding a remote repository

```
git remote add '<name>' '<url>'
```

Listing all available remote repositories

```
git remote # list all available remote repositories
```

Pushing changes to a remote repository

```
git push '<remote>' '<branch>' # push the current branch to the remote repository
```

Pulling changes from a remote repository

```
git pull '<remote>' '<branch>' # pull the changes from the remote repository into the current branch
```

Fetching changes from a remote repository

```
git fetch '<remote>' '<branch>' # fetch the changes from the remote repository into the current branch
```

Cloning a remote repository

```
git clone '<url>' # clone the remote repository into a new directory
```

Glossary #1

- Git: A distributed version control system that allows you to track changes to files and collaborate with others on projects.
- Repository: A place where Git stores all the files and their history for a particular project.
- Commit: A saved snapshot of changes to one or more files in the repository.
- Branch: A separate line of development that diverges from the main codebase, allowing you to make changes without affecting the main codebase.
- Merge: Combining changes from one branch into another.
- Pull: Fetching changes from a remote repository and merging them into a local branch.
- Push: Sending changes from a local branch to a remote repository.
- Remote: A repository that is hosted on a different server than the local repository.

Glossary #2

- Fork: A copy of a repository that allows you to make changes without affecting the original repository.
- Clone: Creating a copy of a remote repository on your local machine.
- Pull Request: A request to merge changes from one branch or fork into another branch, typically used in open source projects to facilitate collaboration.
- Rebase: Changing the base of a branch to a different commit, typically used to incorporate changes from another branch.
- Tag: A named reference to a specific commit in the repository history, often used to mark important versions or releases.
- Conflict: A situation where Git is unable to automatically merge changes between two branches, requiring manual intervention to resolve the differences.

More GIT commands

Awesome Cheat Sheet

Emanuele Bartolesi

Microsoft MVP & GitHub Star

- @kasuken
- github.com/kasuken
- linkedin.com/in/bartolesiemanuele
- emanuelebartolesi.com