RED & WHITE®

Multimedia Education

Shaping "skills" for "scaling" higher...!!!

WELCOME, PROGRAMMERS

01.

What is Git?

WHAT IS GIT?

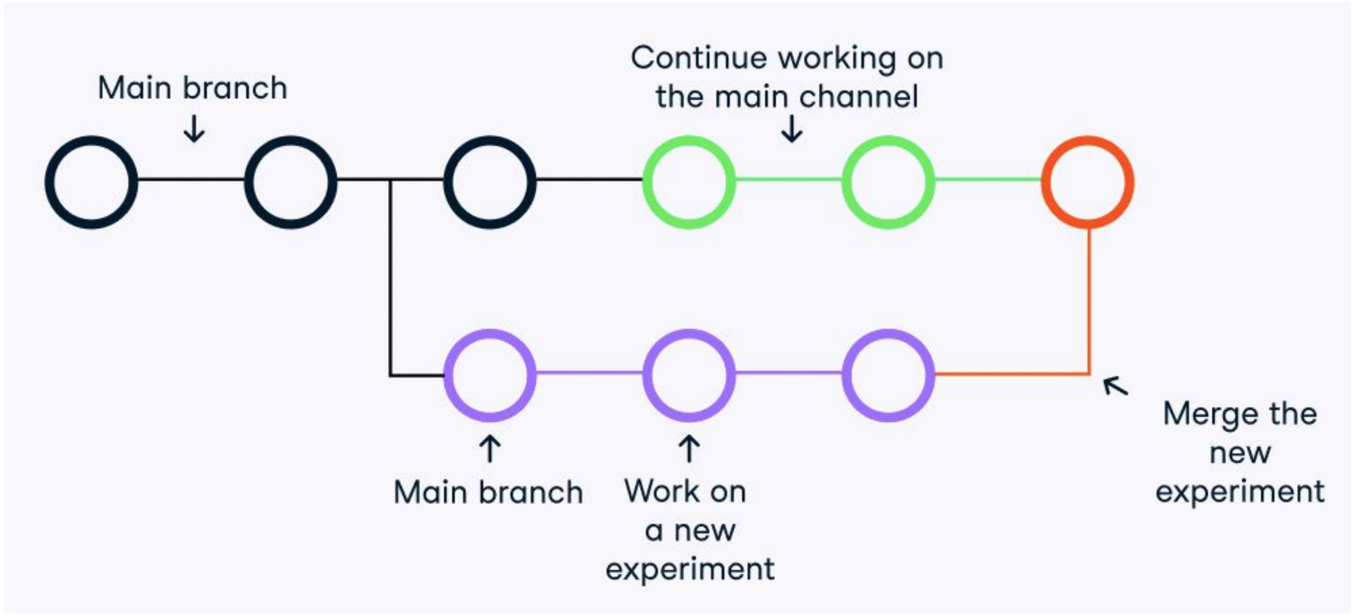
RED & WHITE®
Multimedia Education
Shaping "skills" for "scaling" higher...!!!

GIT

Git is a free and open-source **distributed version control system** designed to handle everything from small to very large projects with speed and efficiency.

Every clone of a **Git repository** is a complete repository with a full copy of the **project's history**. This makes it easy to **collaborate on projects**, as each developer can work on their own branch and then **merge their changes** back into the main repository.

VERSION CONTROLLING



GIT COMMANDS

1. Configurations
2. Starting a repository
3. Staging files
4. Committing to a repository
5. Pulling and pushing from and to repositories
6. Branching

01

CONFIGURATIONS COMMANDS

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email-address"
$ git config --list
```

CONFIGURATIONS COMMANDS

```
git config --global user.name "FirstName LastName"
```

This command is used to **set the author name** that will be used f**or all commits on the current system**.

This information is associated with your Git commits and is helpful for identifying who made a particular commit.

--global: This flag indicates that the configuration should be set globally, affecting all repositories on your system.

If you omit --global, the configuration will be specific to the current repository.

CONFIGURATIONS COMMANDS

```
git config --global user.email "your-email-address"
```

This command is used to **set the email address** associated with your Git commits.

This information is used to identify the author of the commits.

--global: This flag indicates that the configuration should be set globally, affecting all repositories on your system.

If you omit --global, the configuration will be specific to the current repository.

CONFIGURATIONS COMMANDS

```
git config --list
```

This command is used to **display the configuration settings for Git** on your system.

When you run this command, Git will output a list of all the configuration settings along with their current values.

02

STARTING REPOSITORY COMMANDS

```
$ git init
$ git status
```

RED & WHITE®
Multimedia Education
Shaping "skills" for "scaling" higher...!!!

STARTING REPOSITORY COMMANDS

```
git init
```

This command is used to **initialize a new Git repository**. It sets up all the necessary files and data structures required for version control using Git.

Here's a basic breakdown of what git init does:
Creates a Hidden Subfolder: It creates a hidden subfolder within the existing directory that houses the internal data structure required for version control. This subfolder is usually named **.git**.

Initializes Configuration Files: Git config files are created within the .git subfolder. These files will store your configuration options and project-specific settings.

Sets Up the Initial Branches: The command sets up the default branches (usually master) and initializes them to be empty.

Ready for Commits: After running git init, your project is ready to start tracking changes. You can now use other Git commands, like git add to stage files, and git commit to save a snapshot of the project.

STARTING REPOSITORY COMMANDS

```
git status
```

This command is used to **show the status of changes as untracked, modified, or staged in your working directory**. It provides information about the state of your repository and helps you understand what is happening with your files.

Here's a breakdown of what git status does:
Untracked Files: It shows a list of files in your working directory that are not currently tracked by Git. These files have not been added to the staging area.

Changes to be Committed: It shows a list of changes that are staged and ready to be committed. These are changes that you have added to the staging area using git add.

Changes not staged for Commit: It shows modifications in your working directory that haven't yet been staged. These are changes that you made to files but haven't yet added to the staging area.

Branch Information: If you're on a specific branch, git status will indicate which branch you are on and whether your branch is ahead, behind, or has diverged from a remote branch.

03

STAGING FILES COMMANDS

```
$ git add <file-name>
$ git add <file-name> <another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

STAGING FILES COMMANDS

```
git add <file-name>
```

This command is used to **stage changes in a specific file for the next commit**.

It tells Git that you want to include updates to that file in the next snapshot of the repository.

STAGING FILES COMMANDS

```
git add <file-name> <another-file-name>
```

This command is used to **stage changes in multiple specific files for the next commit**.

It allows you to add more than one file to the staging area in a single command.

STAGING FILES COMMANDS

```
git add .
```

This command is used to **stage all changes in the current directory and its subdirectories for the next commit**.

This command is a shorthand way to add all modifications and new files to the staging area.

STAGING FILES COMMANDS

```
git add --all
```

This command is used to **stage all changes, including modifications, deletions, and untracked files, in the entire working directory for the next commit**.

This command stages all changes, regardless of their current status.

STAGING FILES COMMANDS

```
git add -A
```

This command is another way to **stage all changes in the working directory for the next commit**.

It is equivalent to **git add --all** and stages all modifications, deletions, and untracked files.

This command stages all changes, regardless of their current status.

STAGING FILES COMMANDS

```
git rm --cached <file-name>
```

This command is used to **untrack a file in Git without deleting it from your working directory**.

This command removes a file from the staging area (index) but keeps it in your local file system.

STAGING FILES COMMANDS

```
git reset <file-name>
```

This command is used to **unstage changes for a specific file**.

It removes the changes from the staging area while keeping the modifications in your working directory unchanged.

04

COMMITTING REPOSITORY COMMANDS

```
$ git commit -m "Add three files"
$ git commit --amend -m < enter your message>
```

COMMITTING REPOSITORY COMMANDS

```
git commit -m "Add msg"
```

This command is used to **save the staged changes in your Git repository with a commit message**.

It creates a snapshot of the changes that you've previously staged using git add.

COMMITTING REPOSITORY COMMANDS

```
git commit --amend -m <updated commit message>
```

This command is used to **amend the last commit in Git**.

This means you can make changes to the last commit message or add changes that you forgot to include in the last commit.

Here's how the command works:
git commit --amend: This command allows you to modify the last commit. It opens the default text editor for you to modify the commit message. If you only want to modify the commit message without changing the content, you can skip the -m flag.

-m <enter your message>: This option is used to provide a new commit message directly on the command line without opening the text editor. Replace <enter your message> with the new commit message.

05

PULLING AND PUSHING COMMANDS

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

RED & WHITE®
Multimedia Education
Shaping "skills" for "scaling" higher...!!!

PULLING AND PUSHING COMMANDS

```
git remote add origin <link>
```

This command is used to **add a remote repository named "origin" to your local Git repository**.

The term "origin" is a conventionally used name, but you could choose a different name if you prefer.

Here's how the command works:
git remote add: This command is used to add a new remote repository.

origin: This is the name assigned to the remote repository. "Origin" is a common convention, but you could choose another name if you prefer.

<link>: Replace <link> with the URL of the remote repository. This is the link to the repository you want to connect to.

PULLING AND PUSHING COMMANDS

```
git push -u origin master
```

This command is used to **push the local changes from the "master" branch to the remote repository named "origin."**

This command is often used for the initial push to set up a tracking relationship between the local and remote branches.

PULLING AND PUSHING COMMANDS

```
git clone <repository URL>
```

This command is used to **create a copy of a remote Git repository on your local machine**.

This command clones a repository, creating a local working copy with the entire history of the project, including all branches.

PULLING AND PUSHING COMMANDS

```
git pull
```

This command is used to **fetch changes from a remote repository and integrate them into the current working branch**.

It's a combination of two separate Git commands:

git fetch: This command retrieves the changes from the remote repository, including new branches or changes to existing branches.

git merge: After fetching changes, git pull automatically merges those changes into the current branch.

06

BRANCHING COMMANDS

```
$ git branch
$ git branch <branch-name>
$ git checkout <branch-name>
$ git merge <branch-name>
$ git checkout -b <branch-name>
```

BRANCHING COMMANDS

```
git branch
```

This command is used to **list, create, or delete branches in a Git repository**.

When used without any additional arguments, this command lists all the branches in the repository.

The currently active branch is typically indicated with an asterisk (*).

BRANCHING COMMANDS

```
git branch <branch-name>
```

This command is used to **create a new branch in a Git repository**.

This command creates a new branch with the specified name but does not switch to that branch; you remain on the current branch.

BRANCHING COMMANDS

```
git checkout <branch-name>
```

This command is used to **switch to a different branch in a Git repository**.

It allows you to navigate between branches and work on the code associated with a specific branch.

BRANCHING COMMANDS

```
git merge <branch-name>
```

This command is used to **integrate changes from one branch into another**.

This command combines the changes made on a specified branch (called the source branch) into the current branch (called the target branch).

BRANCHING COMMANDS

```
git checkout -b <branch-name>
```

This command is a convenient and commonly used shortcut in Git.

It is equivalent to running two separate commands:
git branch <branch-name> to create a new branch and **git checkout <branch-name>** to switch to that branch.

02.

What is GitHub?

WHAT IS GITHUB?

GITHUB

GitHub is a **platform** and **cloud-based service** for software development and version control, allowing developers to store and manage their code.