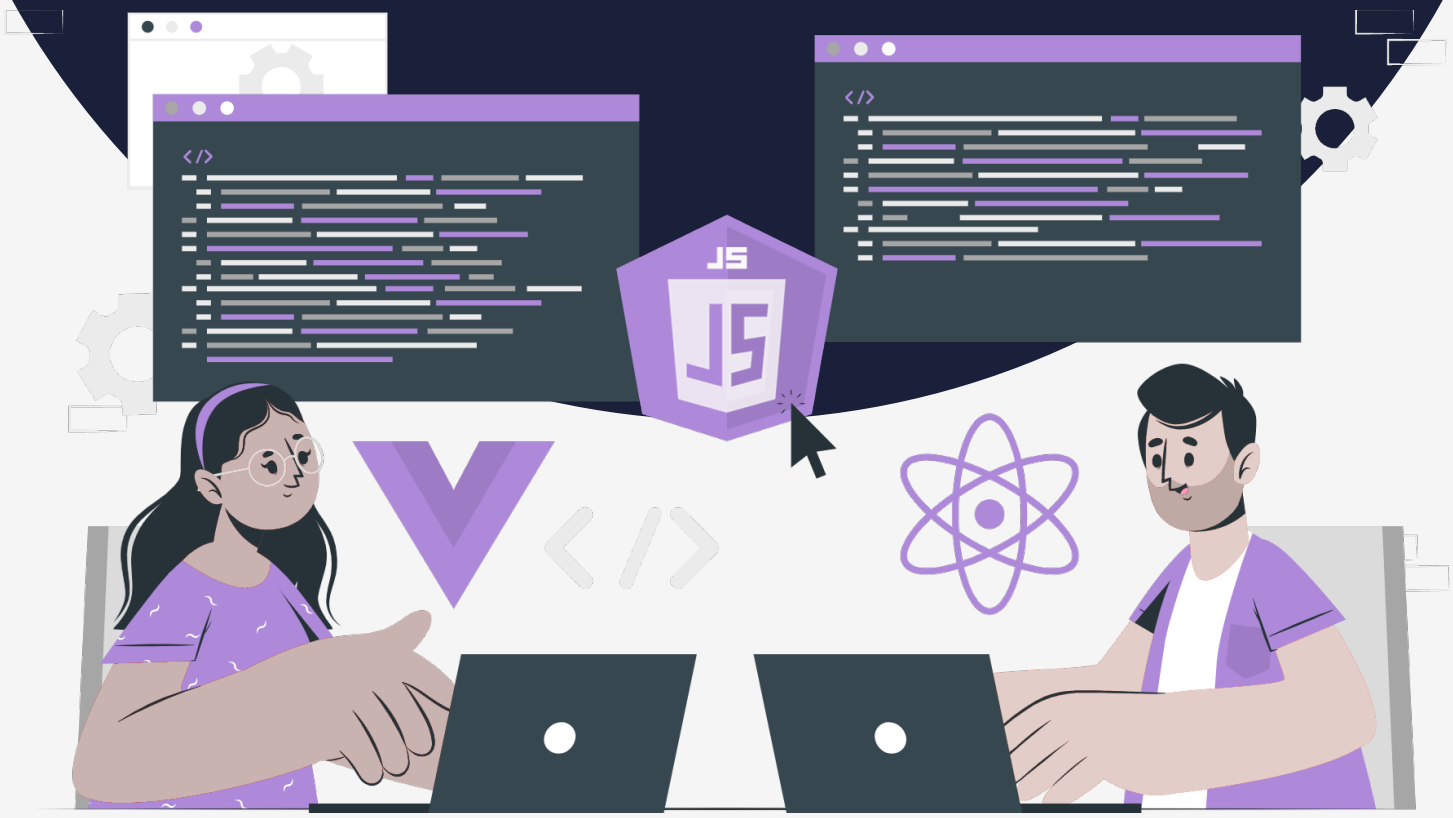
Lesson:

Git Initialization and First Commit



PW SKILLS

Topics Covered:

- git init
- git add
- Difference between git add . and git add *
- git commit

git init

Git is a powerful version control system that allows you to keep track of changes to your code over time. Initializing a Git repository is the first step in using Git to manage your project. Here's a more detailed explanation of each step involved in initializing a Git repository using the **git init** command.

Step 1: Open a Terminal or Command Prompt

To begin, you'll need to open a terminal or command prompt on your computer. This is the interface where you will be running the Git commands. On macOS or Linux, you can use the Terminal application. On Windows, you can use the Command Prompt or PowerShell.

Step 2: Navigate to the Project Directory

Once you have your terminal open, you'll need to navigate to the directory where you want to create your Git repository. You can do this using the cd command, which stands for "change directory". For example, if you want to create your repository in a folder called "my-project" on your desktop, you can type

```
cd ~/Desktop/my-project
```

The tilde character (~) is a shortcut that represents your home directory. You can replace it with the actual path to your desktop if necessary.

Step 3: Initialize the Repository

Now that you're in the project directory, you can initialize the Git repository using the **git init** command:

```
git init
```

This command creates a new **.git** folder in your project directory that contains all the necessary Git files and directories. The **.git** folder is hidden by default, so you won't see it in your file browser unless you show hidden files.

git add

The **git add** command is a fundamental command in Git that tells Git to start tracking changes to a file or files. Here's a more detailed explanation of the **git add** command and how to use it effectively.

Syntax:
The basic syntax of the git add command is:

```
git add <file>
```

This command tells Git to start tracking changes to the specified file. You can also specify multiple files by separating them with spaces, like this:

```
git add <file1> <dile2> <file3>
```

Alternatively, you can use a wildcard character (*) to add all files in the current directory:

```
git add *
```

And also, you can use **git add .** to add files in the current directory:

```
git add .
```

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

Staging area:

Before you can make a commit, you need to add your changes to the staging area. The staging area is a temporary storage area where Git keeps track of the changes that you want to include in your next commit. When you add files using **git add**, Git moves them from the working directory to the staging area.

Adding changes
When you make changes to a file, Git doesn't automatically track those changes. You need to use the git add command to start tracking them. For example, if you make changes to a file called example.txt, you can use the following command to add the changes to the staging area:

```
git add example.txt
```

This tells Git to start tracking changes to the **example.txt** file. You can then use the **git status** command to see the changes that have been added to the staging area.

Removing changes

If you want to remove changes from the staging area, you can use the **git reset** command. For example, if you added a file to the staging area by mistake, you can use the following command to remove it:

```
git reset example.txt
```

This will remove the **example.txt** file from the staging area and move it back to the working directory.

Overall, The **git add** command is a fundamental command in Git that allows you to start tracking changes to a file or files. By adding changes to the staging area, you can prepare them for your next commit and keep track of the changes that you make to your project over time. Git provides a powerful set of tools for managing changes to your codebase, and the **git add** command is one of the most important commands in your Git toolkit.

Difference between git add . and git add *

Both **git add .** and **git add *** are used to add files to the staging area in Git. However, there is a subtle difference between the two commands.

git add . stages all changes in the current directory and its subdirectories, including new files, modified files, and deleted files. It does not include files that are explicitly ignored by Git (e.g., files listed in .gitignore). This command is commonly used to stage all changes before committing them.

On the other hand, the **git add *** command will add all new, modified, and deleted files in the current directory and its subdirectories to the staging area. This means that any files that have been deleted from the working directory will also be removed from the staging area.

For example, let's say you have three files in your project directory: **file1.txt, file2.txt**, and **file3.txt**. If you modify **file1.txt**, delete **file2.txt**, and create a new file called **file4.txt**, here's what would happen if you run each command:

```
git add .
```

This command would add the changes to **file1.txt** to the staging area, but it would not remove **file2.txt** from the staging area. It would also add **file4.txt** to the staging area.

```
git add *
```

This command would add the changes to **file1.txt** to the staging area and remove **file2.txt** from the staging area. It would also add **file4.txt** to the staging area.

In general, it's a good idea to use **git add .** when you only want to add new and modified files to the staging area, and use **git add *** when you want to add all changes, including deleted files. However, it's important to be careful when using **git add *** as it can remove files from the staging area that you didn't intend to remove.

git commit

The git commit command is used to create a new commit in Git, which records changes to the repository. Here's a more detailed explanation of the git commit command and how to use it effectively.

Syntax:
The basic syntax of the git commit command is:

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

This command creates a new commit with the changes that have been added to the staging area. The commit message should be a brief summary of the changes that have been made in this commit. The **-m** flag indicates that the commit message will be specified in the command line.

For example, if you made changes to a file called **example.txt** and added it to the staging area using the **git add** command, you can create a new commit with the following command:

```
git commit -m "Updated example.txt file with new content"
```

This will create a new commit with the changes made to example.txt and a commit message that describes the changes.

Amending a Commit:

If you need to make changes to a commit after it has been created, you can use the **--amend** flag with the **git commit** command. This allows you to modify the last commit you made. For example, if you forgot to include a file in your last commit, you can add it to the staging area and then amend the last commit with the following command:

```
git add <new-file>
git commit --amend -m "Updated commit message"
```

This will add the new file to the last commit and modify the commit message.

Overall, The **git commit** command is used to create a new commit in Git, which records changes to the repository. By creating commits with descriptive commit messages, you can keep track of the changes you make to your project over time and collaborate effectively with others. Git provides a powerful set of tools for managing changes to your codebase, and the **git commit** command is one of the most important commands in your Git toolkit.