Basic Commands

Commands: -

- **ls: -** The **ls** command, short for "**list**," is a Linux command that's used to **list the contents of the current working directory.** It's only work in **git bash terminal**.
- **cd: -**The **cd** command, also known as **Change directory**, is a **command-line shell** command used to **change the current working directory in various operating systems**. The **cd** command can be used to change **into a subdirectory**, **move back into the parent directory**, **move all the way back to the root directory**, and **Move to any given directory.**
- **cd .. : -** The (**cd ..**) command in **cmd** is used to **change the current directory to the parent directory** **(**going back to the previous directory**)**. The **parent directory is the directory that contains the current directory**.
- **mkdir**: - The **mkdir command**, also commonly typed as **md**, is used in **Command Prompt** (**cmd**) on **Windows** to **create new directories** (folders).
- **rm**: -In **Windows PowerShell** the **rm** command is a **Linux command** that is **used to delete files and folders**.
- **pwd**: - The **pwd** command is short for "**print working directory**". It is a **command-line utility that prints the full path of the current working directory**. But it's not work in **command prompt(cmd)** it works in **windows power shell** cmd.
- **cd .**: -The **cd . cmd** used to **open the vscode** in selected directory.

Git

Git is a **distributed version control system** widely used in software development. It allows developers to **track changes** in their code, **collaborate** with others, and manage different **versions** of their projects.

Here are some key features of Git:

→ **Distributed Version Control**: Full repository **copies** for each user, enabling offline work and fast operations.
→ **Branching and Merging**: Easy creation and **combination** of separate development lines.
→ **Staging Area**: Allows selective committing of changes before finalizing.
→ **Commit History**: Maintains a detailed log of all changes, enabling easy **tracking** and **reverting**.

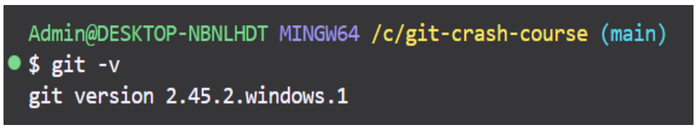
We can download **Git** from its **official website**, and once downloaded, it will be available in **VS Code**. Additionally, we can open the **Git Bash terminal** in **VS Code** and start interacting with it. We can try every command that we used in the **Windows PowerShell terminal**. This way, we will see which commands we used in the **Git Bash terminal**.

Basic Commands of Git Bash terminal: -

- **touch:** - In **Git Bash**, the **touch** command is used to **create a new file** or **multiple files** at a **time**. It does not work in **Command Prompt**; for that, we can use **Git Bash** or the **Windows PowerShell terminal**.
- **mv:** -The **git mv** command is used to **move** or **rename** files within a **Git repository.**
- **ls: -** The **ls** command, short for "**list**," is a **Linux** command that's used to **list the contents of the current working directory.**
- **cd: -**The **cd** command, also known as **Change directory**, is a **command-line shell** command used to **change the current working directory in various operating systems**. The **cd** command can be used to Change **into a subdirectory**, **move back into the parent directory**, **move all the way back to the root directory**, and **Move to any given directory.**
- **cd .. : -** The (**cd ..)** command in **cmd** is used to **change the current directory to the parent directory** (going back to the previous directory**)**. The **parent directory is the directory that contains the current directory**.
- **mkdir** - The **mkdir command** is used in **Command Prompt** (**cmd**) on **Windows** to **create new directories** (folders).
- **pwd:** - The **pwd** command is short for "**print working directory**". It is a **command-line utility that prints the full path of the current working directory**. But it's not work in **command prompt(cmd)** it works in **windows power shell** cmd.
- **Clear:** used to clear the gitbash Teminal.

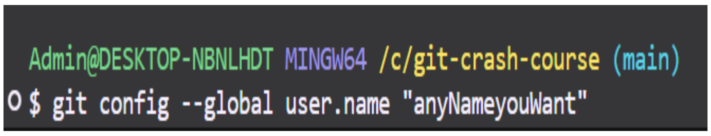
<u>After installing Git, we need to configure it and command which we used in git: -</u>

→ **Git -v:** It's used to check the **version** of **Git**.



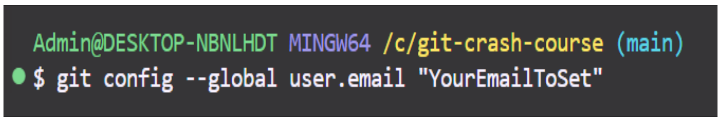
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git -v
git version 2.45.2.windows.1
```

→ **git config --global user.name "Your username": -** The command (**git config --global user.name "username")** is used to set our **Git username globally**. This means that the **username** we specify will be used for all **Git repositories on your computer**.



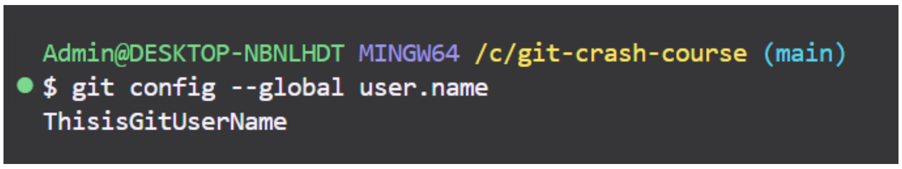
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --global user.name "anyNameyouWant"
```

→ **git config --global user.email "Your Email": -** This command sets our **email address** in **Git's global configuration**. This identifies us as the **author** of our **commits** across all **repositories** on our system, allowing others to see who made changes and potentially contact us. It's a crucial step in setting up **Git** for the first time on a machine.

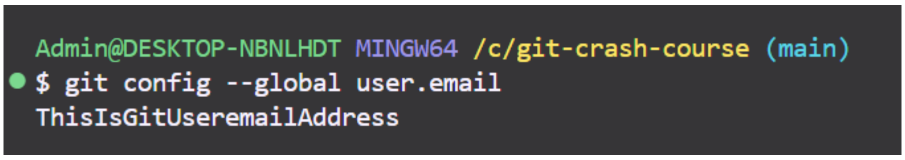


```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --global user.email "YourEmailToSet"
```

→ **To see only the username and email, we need to type:**

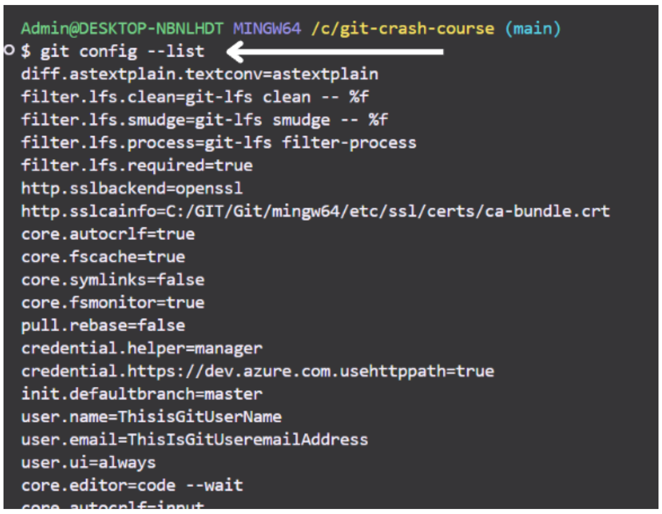


```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --global user.name
ThisisGitUserName
```



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --global user.email
ThisIsGitUseremailAddress
```

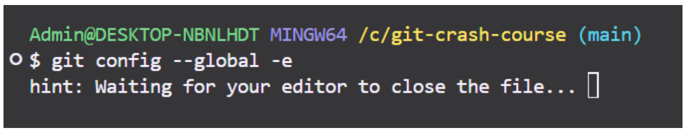
→ **To see all configuration details, we have to type the following command:**



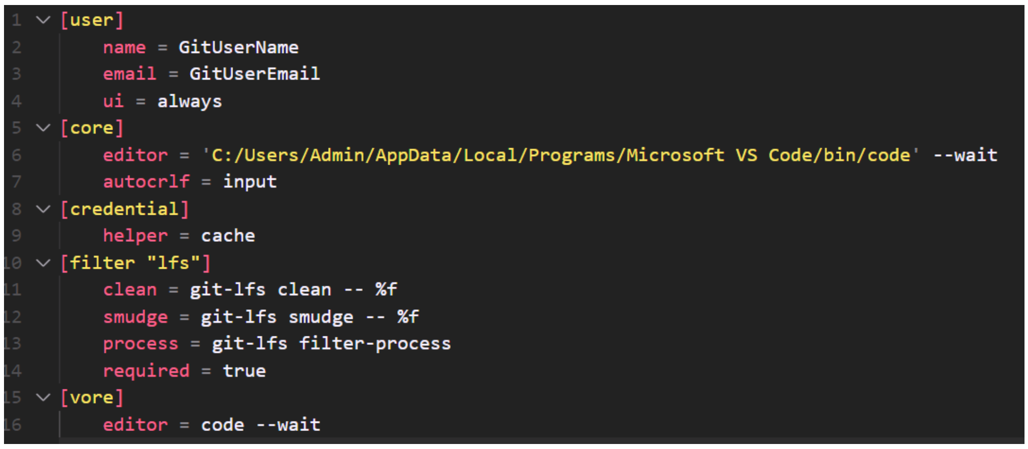
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/GIT/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.fsmonitor=true
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=ThisisGitUserName
user.email=ThisIsGitUseremailAddress
user.ui=always
core.editor=code --wait
core.autocrlf=input
```

→ **git config --global core.editor "code --wait": -** The command **git config --global core.editor "code --wait"** is used to **configure Git** to use **Visual Studio Code** as the **default text editor** for certain **Git operations** that require user input or editing, such as writing **commit messages** or **resolving merge conflicts**.

→ **git config --global -e: -** The git **config --global -e** command opens our **global Git configuration file** in a **text editor**. This allows us to directly **view** and **modify** all our **global Git settings** in one place.



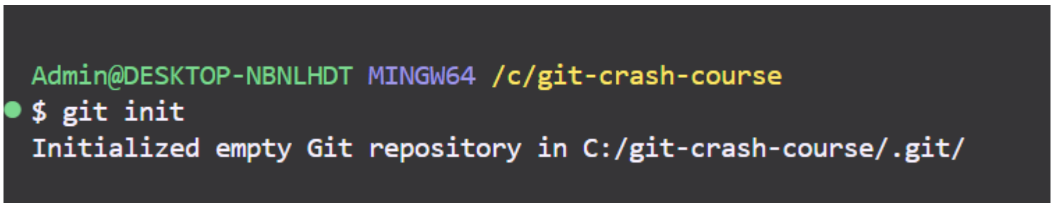
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (main)
$ git config --global -e
hint: Waiting for your editor to close the file...
```



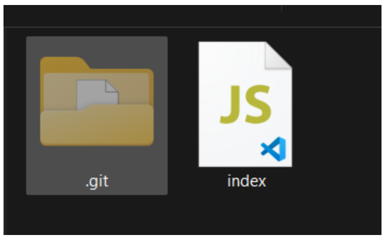
```
[user]
    name = GitUserName
    email = GitUserEmail
    ui = always
[core]
    editor = 'C:/Users/Admin/AppData/Local/Programs/Microsoft VS Code/bin/code' --wait
    autocrlf = input
[credential]
    helper = cache
[filter "lfs"]
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
    process = git-lfs filter-process
    required = true
[vore]
    editor = code --wait
```

→ **Git Init(initialization): -** The **git init command** is used to **create an empty Git repository**. This command is typically **used to create a new repository for a project**, but it can also be used to **reinitialize an existing repository**.

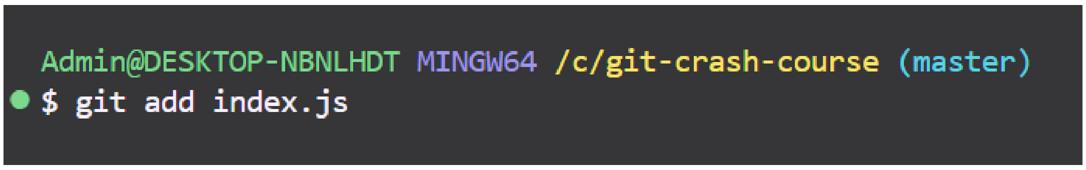
- When we run the **git init** command, Git creates a **hidden directory** called (**.git)** in the **current working directory**. This directory contains all of the **metadata** and **files** that **Git** needs to track the changes to our project.



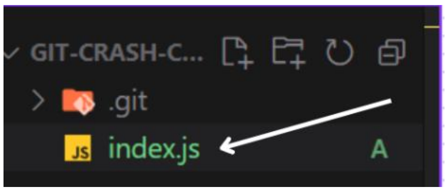
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course
$ git init
Initialized empty Git repository in C:/git-crash-course/.git/
```



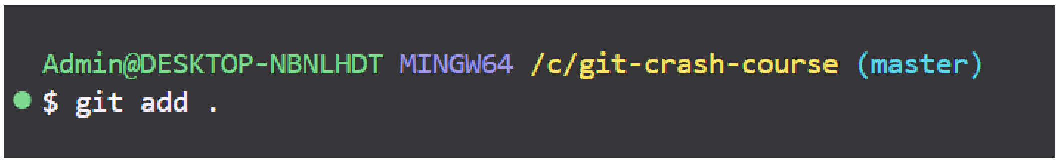
→ **Git add: - Git add** is a **command** in **Git** that **adds files to the staging area**. The **staging area** is a **temporary area** where we can store files that we want to include in our next **commit**. Once we have added files to the **staging area**, we can **commit** them to our **repository**.



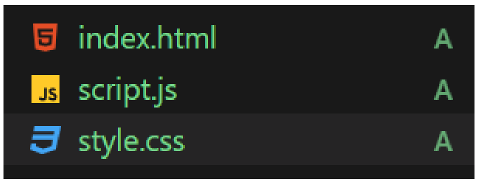
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git add index.js
```



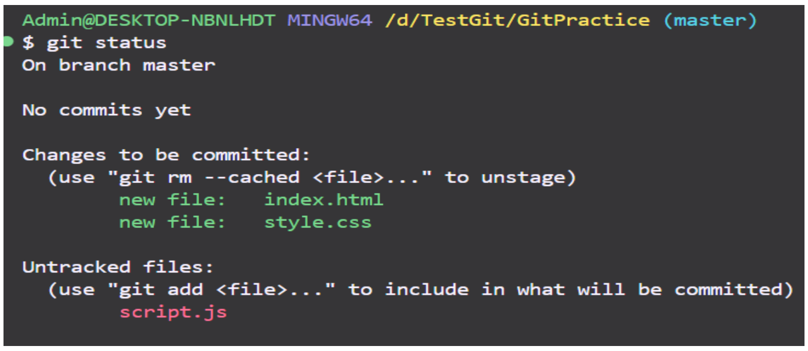
This will add the **specified file** to the **staging area**. we can also **add all files** in the **current directory** to the **staging area** by using the following command:



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git add .
```

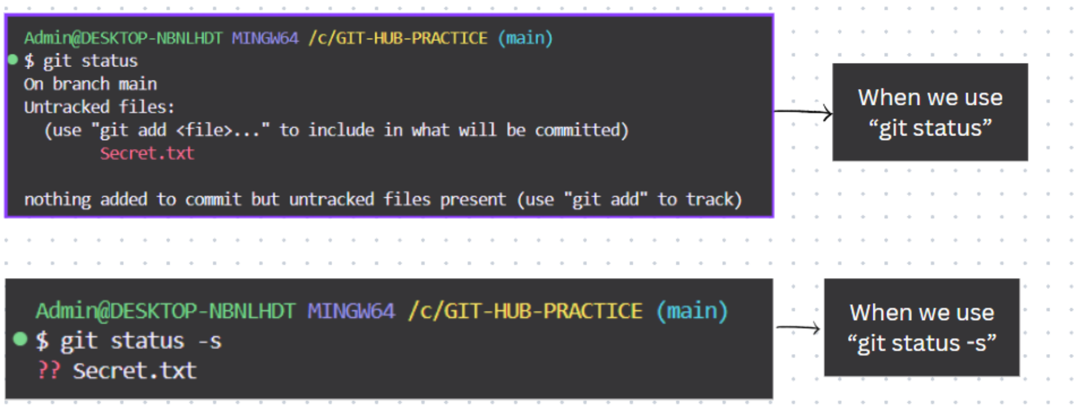


→ **git status: -** The **git status** command is used to **check the status of our local Git repository**. It shows us which files have been changed, which files have been staged, and which files are not being tracked by Git.



```
Admin@DESKTOP-NBNLHDT MINGW64 /d/TestGit/GitPractice (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
        new file:   style.css

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        script.js
```

→ **git status -s: -** The **git status -s** command provides a **concise overview of the current state of our working directory and staging area**.

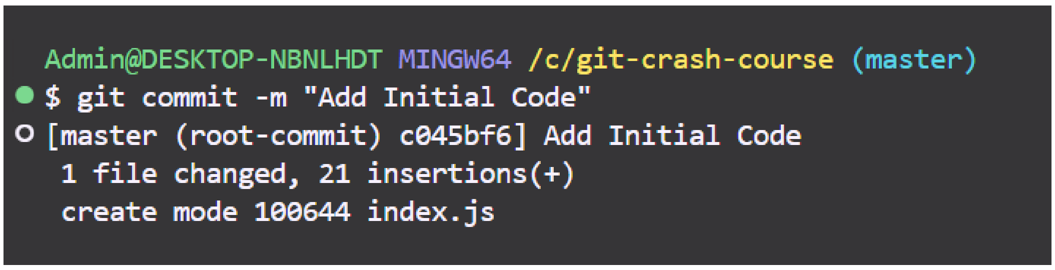


```
Admin@DESKTOP-NBNLHDT MINGW64 /c/GIT-HUB-PRACTICE (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Secret.txt

nothing added to commit but untracked files present (use "git add" to track)
```

```
Admin@DESKTOP-NBNLHDT MINGW64 /c/GIT-HUB-PRACTICE (main)
$ git status -s
?? Secret.txt
```

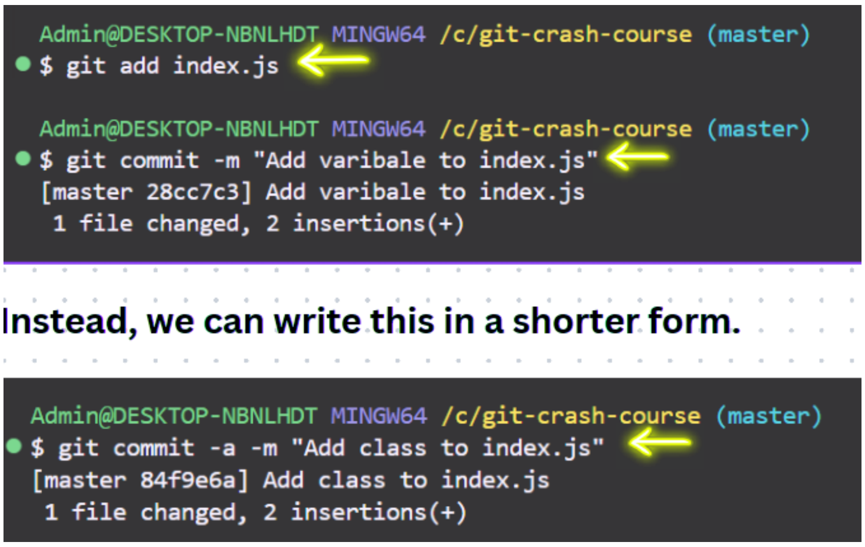
→ **git commit: -** The **git commit** command is used to record the changes we have made to our project. **It takes a snapshot of the current state of our project and stores it in the local repository.**

- To use the **git commit command**, we first need to add the changes we want to commit to the **staging area**. we can do this using the **git add command**. Once we have added the changes to the **staging area,** we can then run the **git commit command.**
- The **git commit command** takes a number of options, but the most important one is the **-m option**. This option allows us to specify a **commit message**. The commit message should be a brief description of the changes we have made.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git commit -m "Add Initial Code"
[master (root-commit) c045bf6] Add Initial Code
 1 file changed, 21 insertions(+)
 create mode 100644 index.js
```

→ **git commit -a -m "commit message": -** The command `**git commit -a -m "commit message"**` automatically **stages** all changes to **tracked files** and **commits** them with the message "**committed message**" It combines **staging** and **committing** into one step, saving time and effort.

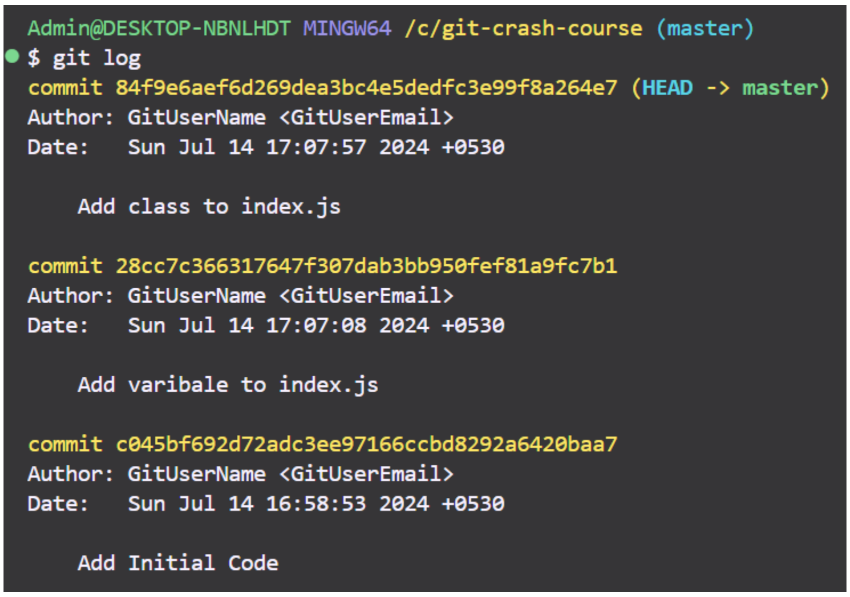


```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git add index.js

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git commit -m "Add varibale to index.js"
[master 28cc7c3] Add varibale to index.js
 1 file changed, 2 insertions(+)
```

Instead, we can write this in a shorter form.

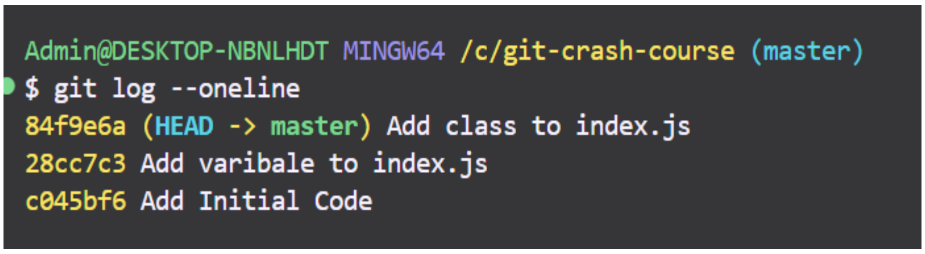
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git commit -a -m "Add class to index.js"
[master 84f9e6a] Add class to index.js
 1 file changed, 2 insertions(+)
```

→ **Git log:** git **log** shows us **all the commits that have been made to our repository**, in reverse chronological order.



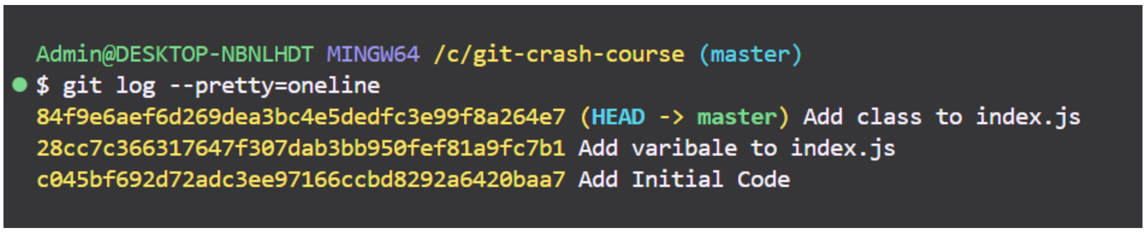
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git log
commit 84f9e6aef6d269dea3bc4e5dedfc3e99f8a264e7 (HEAD -> master)
Author: GitUserName <GitUserEmail>
Date:   Sun Jul 14 17:07:57 2024 +0530

    Add class to index.js

commit 28cc7c366317647f307dab3bb950fef81a9fc7b1
Author: GitUserName <GitUserEmail>
Date:   Sun Jul 14 17:07:08 2024 +0530

    Add varibale to index.js

commit c045bf692d72adc3ee97166ccbd8292a6420baa7
Author: GitUserName <GitUserEmail>
Date:   Sun Jul 14 16:58:53 2024 +0530

    Add Initial Code
```

→ **Git log --oneline: -** The `**git log --oneline**` command shows the **commit history** in a **simple**, **one-line format per commit**, displaying the **commit ID** and **message**. This makes it easy to quickly see the list of commits.



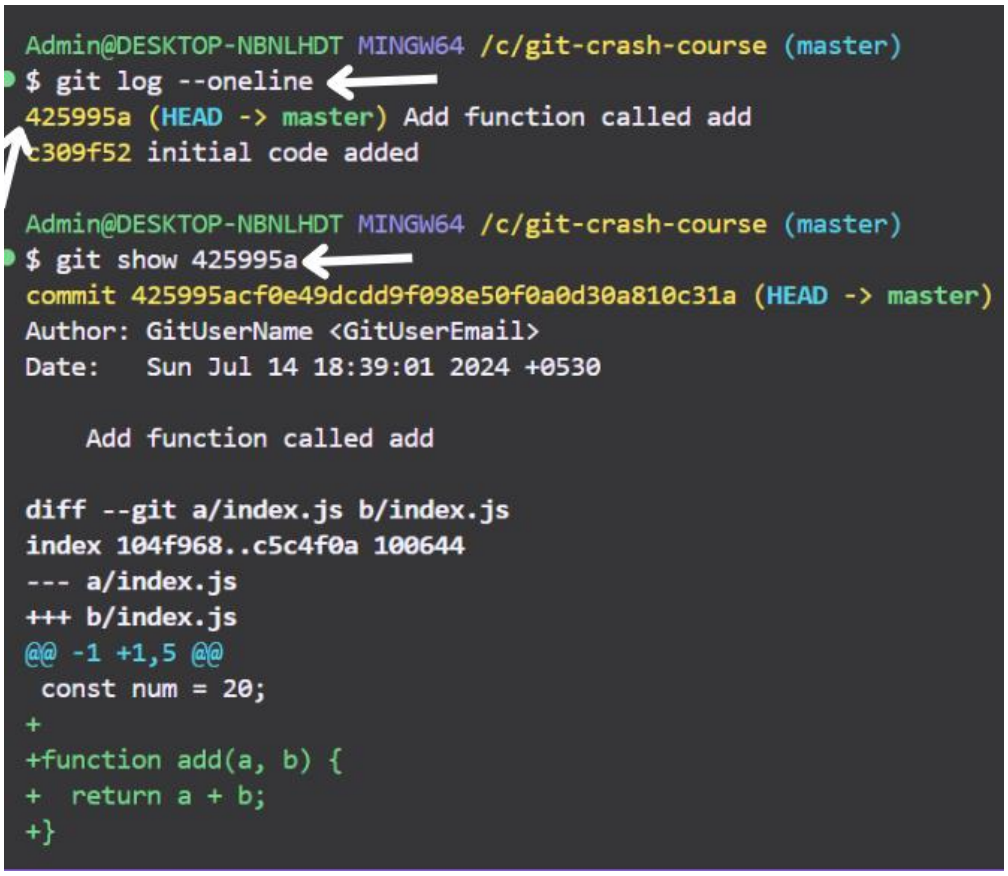
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git log --oneline
84f9e6a (HEAD -> master) Add class to index.js
28cc7c3 Add varibale to index.js
c045bf6 Add Initial Code
```

→ **Git log --pretty=oneline:** The **git log --pretty=oneline** command is similar to **git log --oneline**, but it provides more flexibility and control over the formatting of the **commit** log output.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git log --pretty=oneline
84f9e6aef6d269dea3bc4e5dedfc3e99f8a264e7 (HEAD -> master) Add class to index.js
28cc7c366317647f307dab3bb950fef81a9fc7b1 Add varibale to index.js
c045bf692d72adc3ee97166ccbd8292a6420baa7 Add Initial Code
```

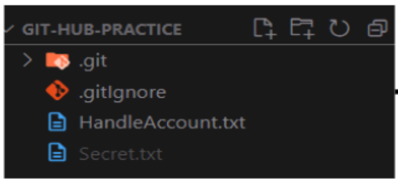
→ **Git show: -** The `**git show**` command displays detailed information about a specific **commit**, including the **message**, **author**, **date**, and **changes** made. It often requires a **commit ID** to specify which **commit** to examine, defaulting to the most recent commit if no ID is provided. This is useful for understanding the exact changes in a commit.



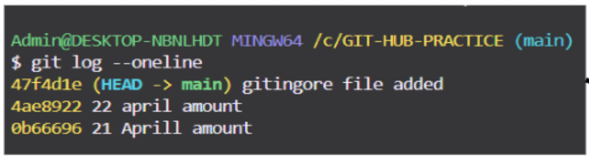
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git log --oneline
425995a (HEAD -> master) Add function called add
c309f52 initial code added

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git show 425995a
commit 425995acf0e49dcdd9f098e50f0a0d30a810c31a (HEAD -> master)
Author: GitUserName <GitUserEmail>
Date:   Sun Jul 14 18:39:01 2024 +0530

    Add function called add

diff --git a/index.js b/index.js
index 104f968..c5c4f0a 100644
--- a/index.js
+++ b/index.js
@@ -1 +1,5 @@
 const num = 20;
+
+function add(a, b) {
+  return a + b;
+}
```

→ **git reset --hard HEAD~1**: -The command **git reset --hard HEAD~1** is used to **undo the most recent commit** and **discard all changes associated with it.**

- **git reset**: This command is used to **reset** our **current HEAD** to a **specified state**.
- **--hard**: This option tells **Git** to **reset** the **staging area** (index) and the **working directory** to match the specified commit. All changes in the working directory are lost.
- **HEAD~1**: This refers to the commit just before the **current HEAD**. Essentially, it means "**one commit before HEAD**."

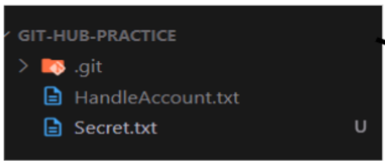


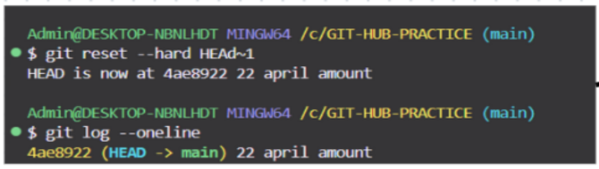
We have .gitignore, HandleAccount.txt and Secret.txt



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/GIT-HUB-PRACTICE (main)
$ git log --oneline
47f4d1e (HEAD -> main) gitingore file added
4ae8922 22 april amount
0b66696 21 Aprill amount
```

Our last commit is “git ignore file added,” but if we want to revert or go back to the last commit, we can use the “git reset --hard HEAD~1" command.

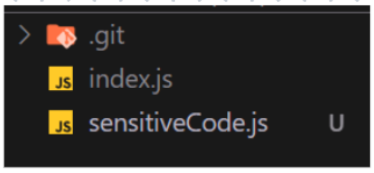




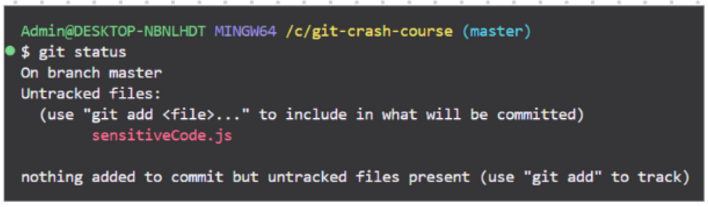
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/GIT-HUB-PRACTICE (main)
$ git reset --hard HEAd~1
HEAD is now at 4ae8922 22 april amount

Admin@DESKTOP-NBNLHDT MINGW64 /c/GIT-HUB-PRACTICE (main)
$ git log --oneline
4ae8922 (HEAD -> main) 22 april amount
```

When we use the “git reset --hard HEAD~1” command, it goes back to the previous commit, the HEAD is pointing to the previous commit, and the gitignopre file is deleted.

→ **. gitignore: -** It is needed in **Git** to specify which files and directories should be ignored by **Git**. to prevent **unwanted files** from **cluttering** our **Git repository** and to avoid accidentally **sharing** sensitive information.

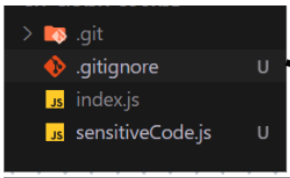


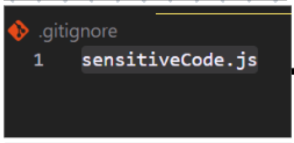
We created a file called "sensitiveCode.js" that contains some crucial information.



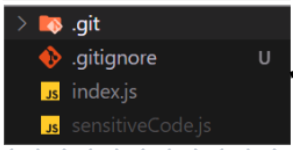
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sensitiveCode.js

nothing added to commit but untracked files present (use "git add" to track)
```

If we check the Git status. Git status showing that sensitiveCode.js is visible to version control.To prevent Git from tracking this sensitive file, We need to add it to the .gitignore file.

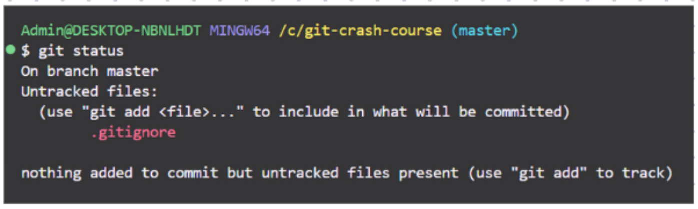




```
.gitignore
1    sensitiveCode.js
```



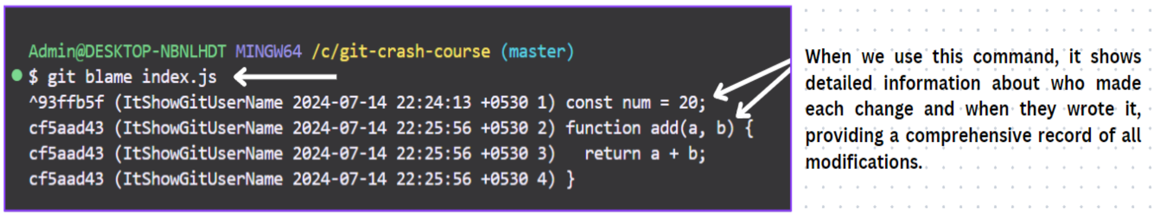
We created a .gitignore file after adding sensitiveCode.js to our .gitignore file, Git now ignores this file. In the file explorer, it appears light grey, indicating its ignored status. The 'U' (untracked) symbol previously associated with the file is no longer present.we can add n number file to ignore.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

When I run 'git status' again, it shows that the .gitignore file itself is untracked, but as expected, it no longer tracking the sensitiveCode.js.

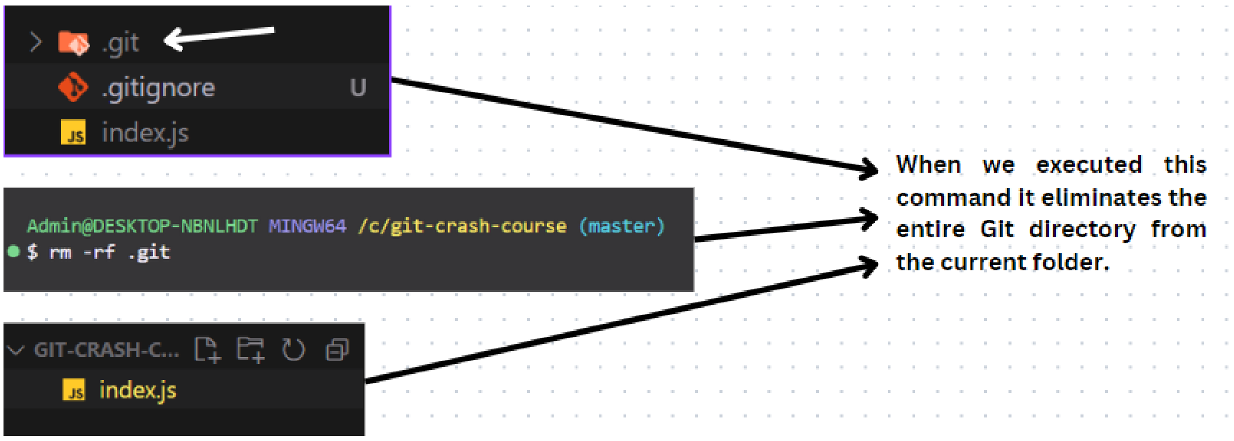
→ **git blame: -**git blame is a **command** in **Git** that shows the **last modification for each line in a file**. This command helps identify who made **changes** to the code, when they made them, and what **changes** were made. Essentially, it annotates each line of a file with information about the **commit** and the **author** responsible for that line.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git blame index.js
^93ffb5f (ItShowGitUserName 2024-07-14 22:24:13 +0530 1) const num = 20;
cf5aad43 (ItShowGitUserName 2024-07-14 22:25:56 +0530 2) function add(a, b) {
cf5aad43 (ItShowGitUserName 2024-07-14 22:25:56 +0530 3)    return a + b;
cf5aad43 (ItShowGitUserName 2024-07-14 22:25:56 +0530 4) }
```

When we use this command, it shows detailed information about who made each change and when they wrote it, providing a comprehensive record of all modifications.

→ **rm -rf .git: -** The command `rm -rf .git` forcefully removes the **entire git directory**, erasing all Git **version control information** from a project. This includes **commit history**, **branches**, and **remote repository data**. It's used to start fresh or **remove** Git from a directory, but should be used with extreme caution as it's irreversible and can lead to data loss if misused.

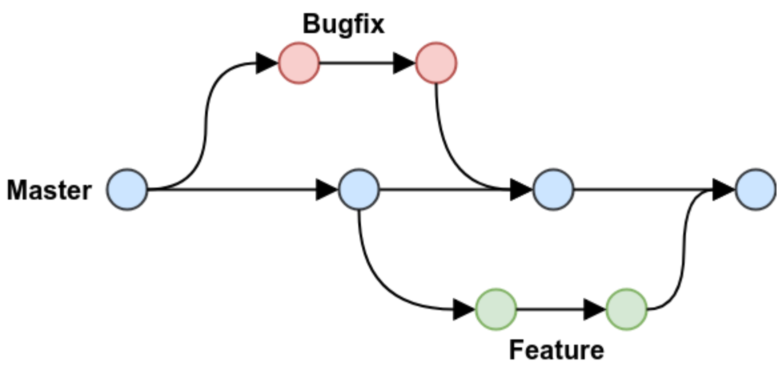


```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ rm -rf .git
```

When we executed this command it eliminates the entire Git directory from the current folder.

Branches in git

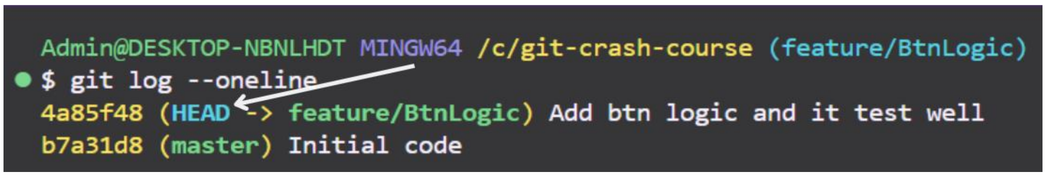
Branches are a way to work on different **versions** of a project at the same time. They allow us to create a separate line of **development** that can be worked on independently of the **main branch**. This can be useful when we want to make changes to a project without affecting the **main branch** or when we want to work on a **new feature** or **bug fix**.



Some developers can work on **Header**, some can work on **Footer**, some can work on **Content**, and some can work on **Layout**. This is a good example of how **branches** can be used in **git**.

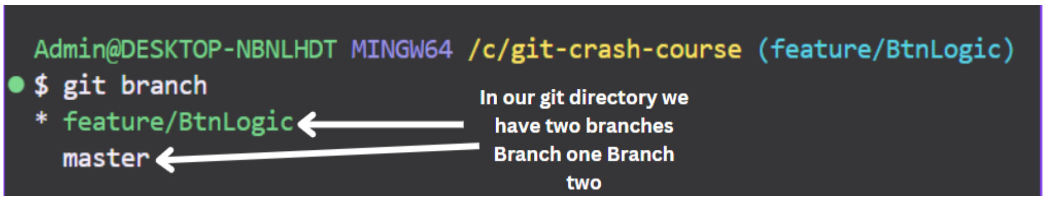
→ **HEAD in git: -** The **HEAD** is a **pointer** to the **current branch** that we are working on. It points to the **latest commit** in the **current branch**. When we create a **new branch**, it is automatically set as the **HEAD** of that **branch**.

× **Note: -**the default branch used to be **master**, but it is now called **main**. There is nothing special about main, it is just a convention



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (feature/BtnLogic)
$ git log --oneline
4a85f48 (HEAD -> feature/BtnLogic) Add btn logic and it test well
b7a31d8 (master) Initial code
```

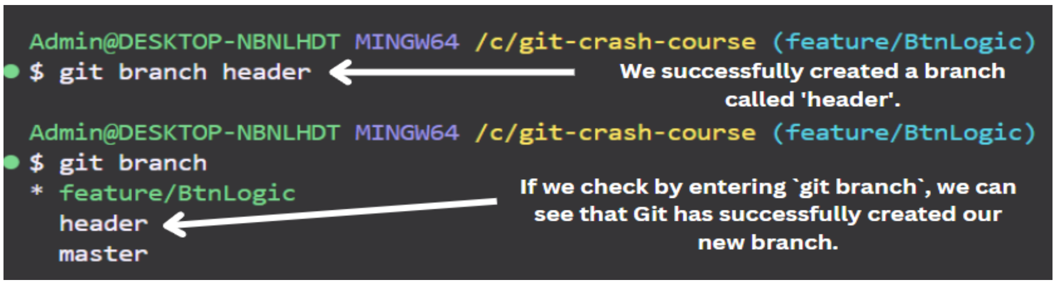
→ **git branch: -** This command **lists all the branches in the current repository**.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (feature/BtnLogic)
$ git branch
* feature/BtnLogic
  master
```

In our git directory we have two branches Branch one Branch two

→ **git branch branchName: -** This command creates a **new branch** with any name you choose.



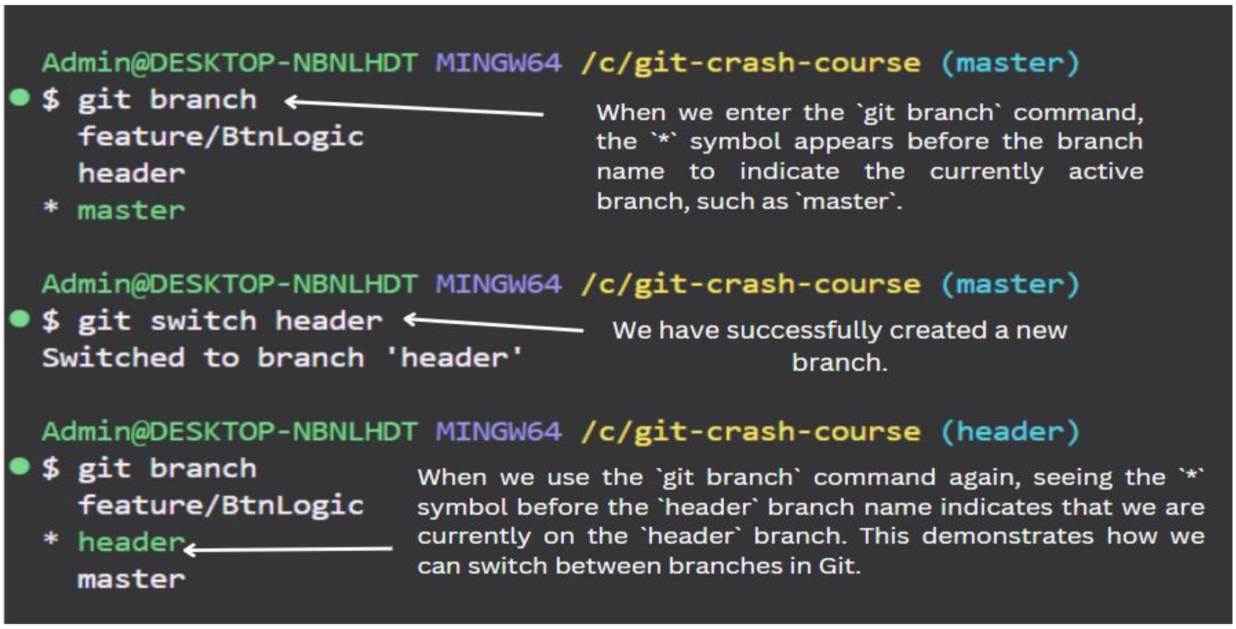
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (feature/BtnLogic)
$ git branch header
```

We successfully created a branch called 'header'.

```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (feature/BtnLogic)
$ git branch
* feature/BtnLogic
  header
  master
```

If we check by entering `git branch`, we can see that Git has successfully created our new branch.

→ **git switch branchName: -** The **git switch** command is used to **change branches** in a **Git repository,**



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch
  feature/BtnLogic
  header
* master
```

When we enter the `git branch` command, the `*` symbol appears before the branch name to indicate the currently active branch, such as `master`.

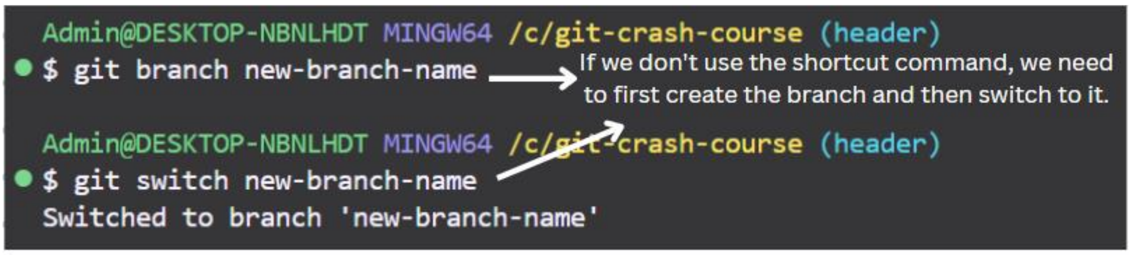
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git switch header
Switched to branch 'header'
```

We have successfully created a new branch.

```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (header)
$ git branch
  feature/BtnLogic
* header
  master
```

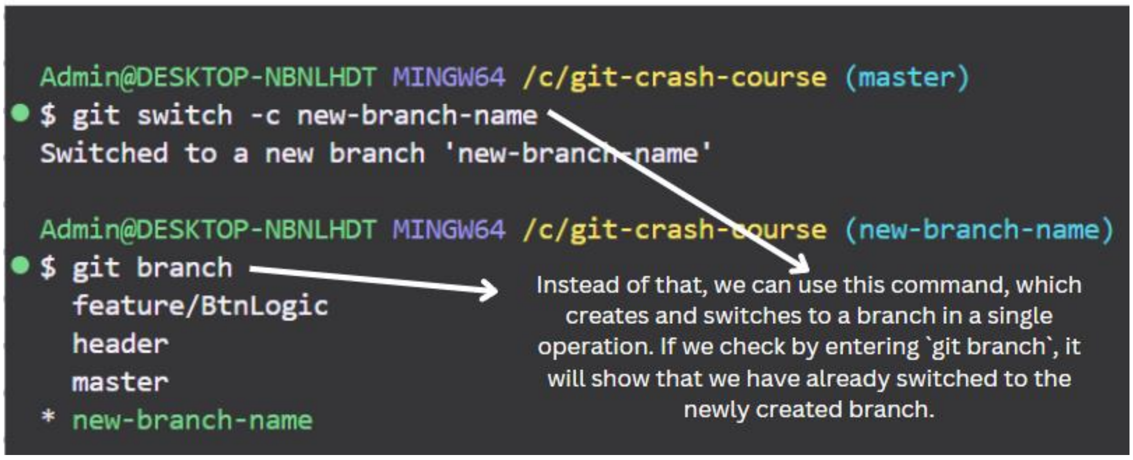
When we use the `git branch` command again, seeing the `*` symbol before the `header` branch name indicates that we are currently on the `header` branch. This demonstrates how we can switch between branches in Git.

→ **git switch -c new-branch-name: -** The `**git switch -c**` command is used to **create a new branch** and **switch to it in a single operation**. It simplifies the workflow by eliminating the need for separate commands to **create** and **switch branches**. For instance, `**git switch -c new-branch-name**` creates a **branch** named `**new-branch-name**` and sets it as the **active branch** for ongoing development.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (header)
$ git branch new-branch-name

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (header)
$ git switch new-branch-name
Switched to branch 'new-branch-name'
```

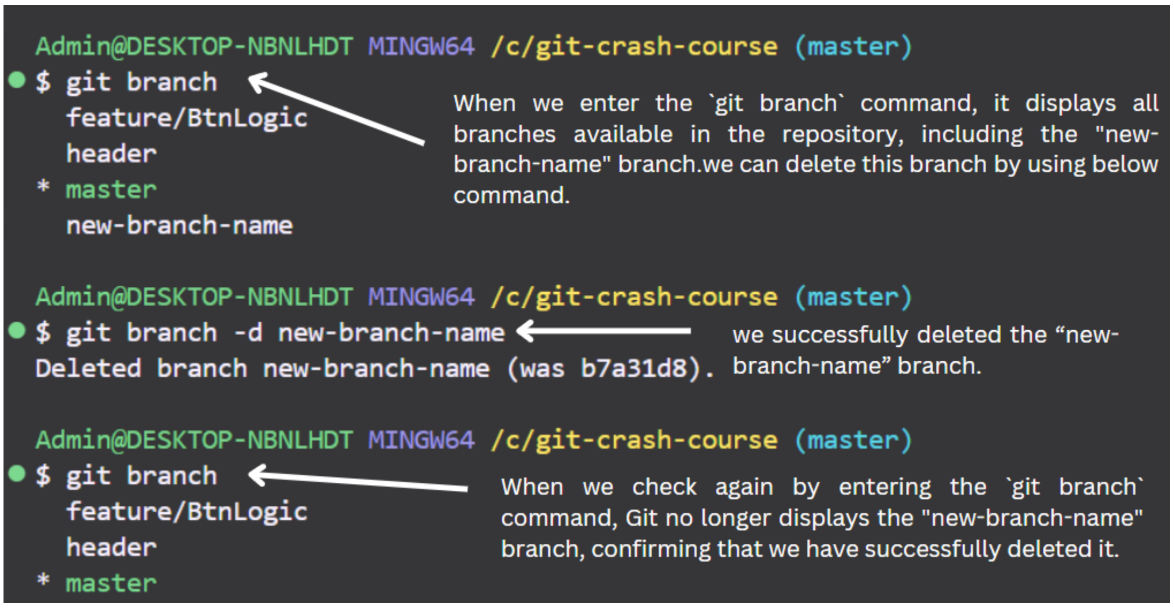
If we don't use the shortcut command, we need to first create the branch and then switch to it.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git switch -c new-branch-name
Switched to a new branch 'new-branch-name'

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (new-branch-name)
$ git branch
  feature/BtnLogic
  header
  master
* new-branch-name
```

Instead of that, we can use this command, which creates and switches to a branch in a single operation. If we check by entering `git branch`, it will show that we have already switched to the newly created branch.

→ **git branch -d branchName: -**The `**git branch -d branchName**` command is used to **delete** a **branch** that has already been **merged** into another **branch**, helping to keep the repository tidy and organized.



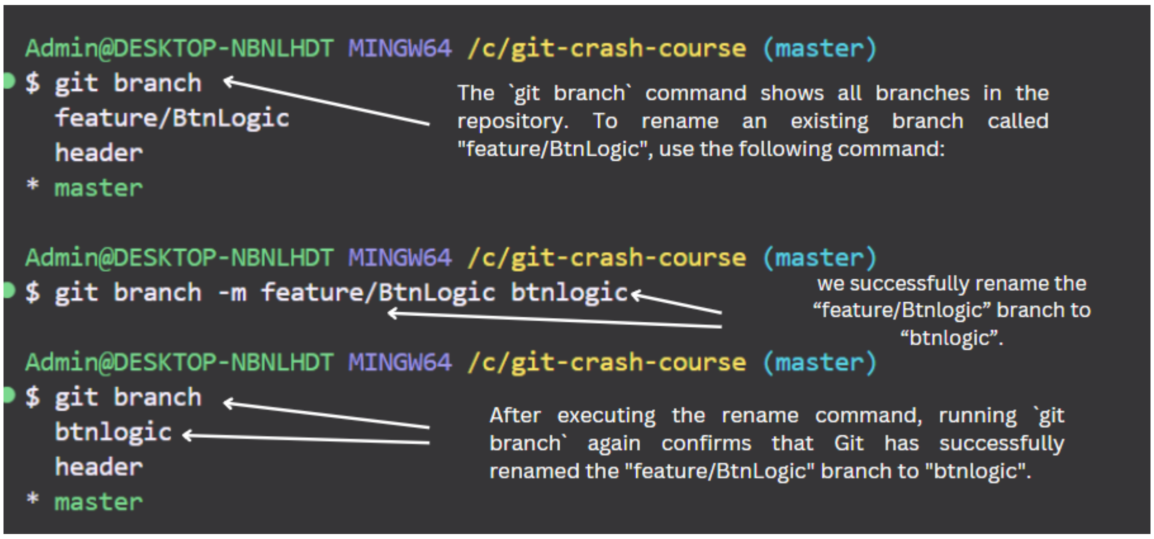
```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch
  feature/BtnLogic
  header
* master
  new-branch-name

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch -d new-branch-name
Deleted branch new-branch-name (was b7a31d8).

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch
  feature/BtnLogic
  header
* master
```

When we enter the `git branch` command, it displays all branches available in the repository, including the "new-branch-name" branch.we can delete this branch by using below command.

we successfully deleted the "new-branch-name" branch.

When we check again by entering the `git branch` command, Git no longer displays the "new-branch-name" branch, confirming that we have successfully deleted it.

→ **git branch -m old-branch-name new-branch-name: -** The `**git branch -m <old-branch-name> <new-branch-name>**` command is used to **rename** a **Git branch**. It changes the name of an existing branch from `<**old-branch-name**>` to `<**new-branch-name**>.



```
Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch
  feature/BtnLogic
  header
* master

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch -m feature/BtnLogic btnlogic

Admin@DESKTOP-NBNLHDT MINGW64 /c/git-crash-course (master)
$ git branch
  btnlogic
  header
* master
```

The `git branch` command shows all branches in the repository. To rename an existing branch called "feature/BtnLogic", use the following command:

we successfully rename the "feature/Btnlogic" branch to "btnlogic".

After executing the rename command, running `git branch` again confirms that Git has successfully renamed the "feature/BtnLogic" branch to "btnlogic".

git commit --amend