

St. Francis Institute of Technology  
Department of Computer Engineering  
Five Days Student Development Programme  
on  
DevOps  
(28th June to 2nd July 2021)

Assignment 01

**GIT & GITHUB TASK**

1. Create a GitHub Account (Use Existing account, if you already have one) (<https://github.com/join>)
2. Create a remote repository on GitHub and name it (**fullname-rollno**)
3. Perform all the git operations on the local and remote repository
4. Take screenshots of all the operations. (Screenshots must reflect your name in the GitHub repository and on the bash terminal)
5. Refer to the link given to perform various operations.  
(<https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>)
6. Take screenshots of all the steps and prepare a report for the same.  
Provide the Link to your public repository in the report

NAME: Saravana sundar Nadar

ROLL NO:72

PARTICIPANT ENROLLMENT ID: 192072 **(PID)**

CLASS: SE CMPN A

Link to my public repository :

<https://github.com/saravana-sn/Saravana-sundar-192072>

## **GIT & GITHUB**

# **Operations & Commands**

Some of the basic operations in Git are:

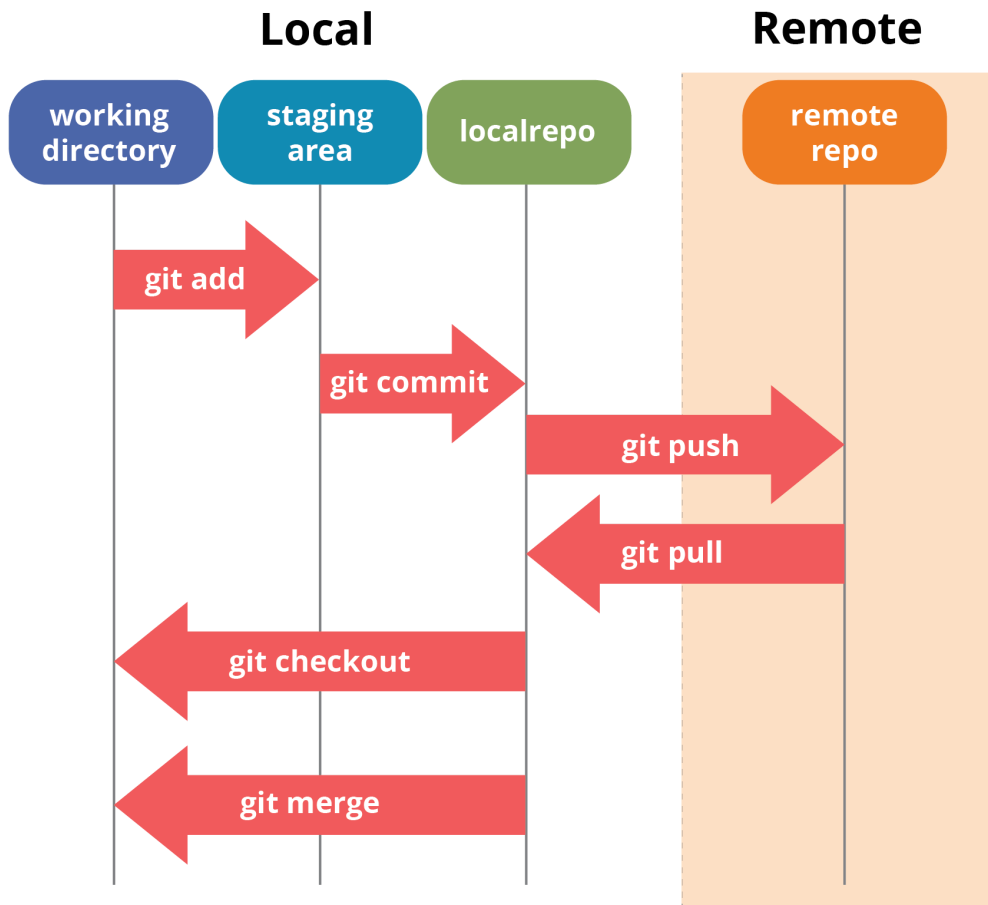
1. Initialize
2. Add
3. Commit
4. Pull
5. Push

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing

Let me first give you a brief idea about how these operations work within Git repositories.

Take a look at the architecture of Git below:



## Creating a Github account & a remote repository.


### 1. Creating a new repository.

- You need to create a new repository and click on the plus sign.
- Fill up all the required details, i.e., repository name, description and also make the repository public this time as it is free.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository [Import a repository](#).

Owner


 Olivia-Smithcoder100 ▾

Repository name \*


/ FaceDetection ✓

Great repository names are short and memorable. Need inspiration? How about **animated-octo-men**

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾



**Create repository**

## 2. Download Git Bash.

- Git Bash can be downloaded in the git website, and it is a shell used to interface with the operating system which follows the UNIX command.
- After Downloading check the git version with the command `git --version` in the cmd.

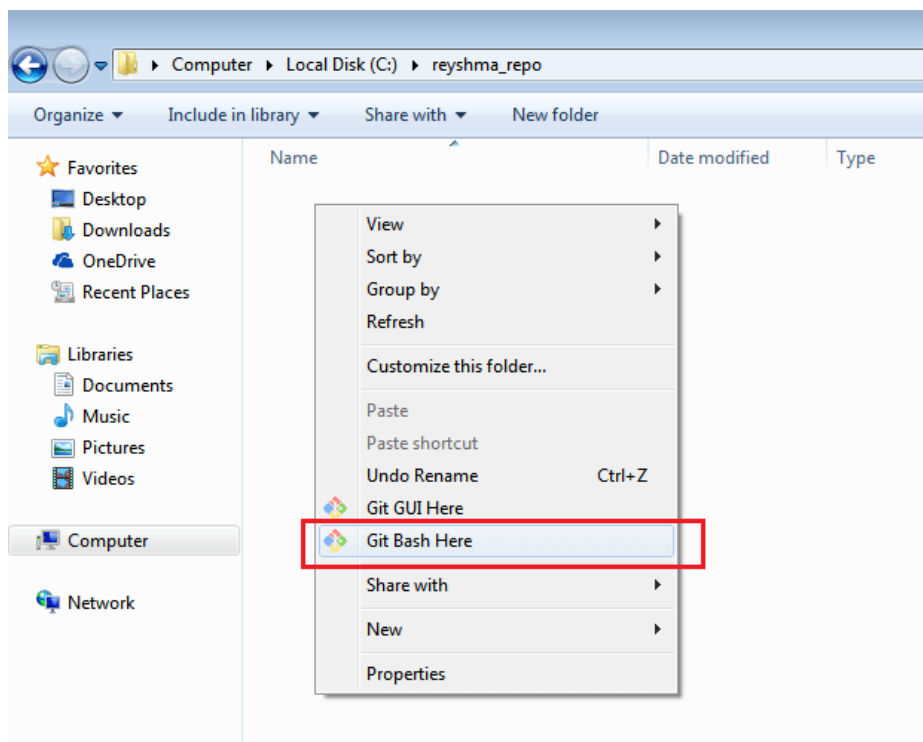
```
C:\Users\Saravana>git -version
unknown option: -version
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        [--super-prefix=<path>] [--config-env=<name>=<envvar>]
        <command> [<args>]

C:\Users\Saravana>git --version
git version 2.32.0.windows.1

C:\Users\Saravana>_
```

I will show you the commands and the operations using Git Bash. Git Bash is a text-only command-line interface for using Git on Windows and provides features to run automated scripts.

After installing Git in your Windows system, just open your folder/directory where you want to store all your project files; right-click and select '***Git Bash Here***'.



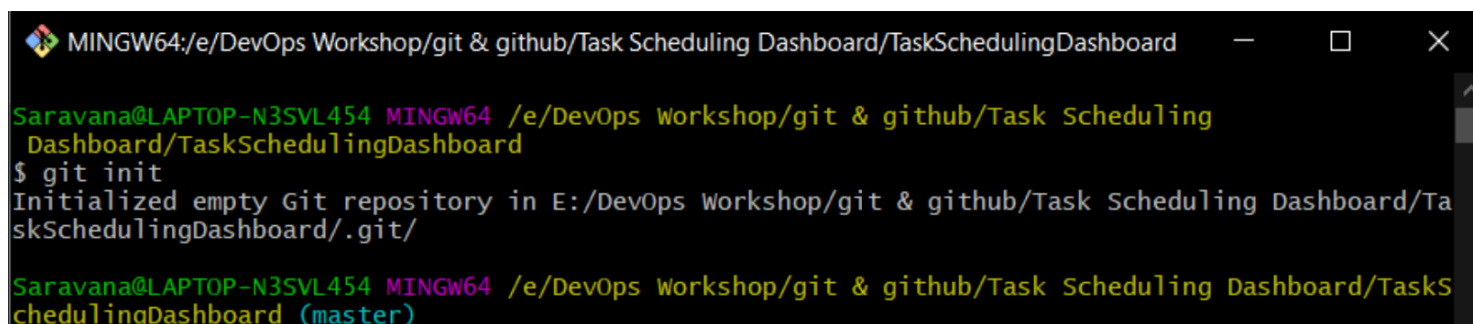
This will open up the Git Bash terminal where you can enter commands to perform various Git operations.

**\*\* I have not taken screenshots during installation of git and creating github account and repository so used from internet\*\***

**3. Push Your local repository to the created github repository .**

## Initialize

In order to do that, we use the command `git init`. Please refer to the following screenshot.

A screenshot of a Windows terminal window titled "MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard". The terminal shows the command prompt "Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard" followed by the command "\$ git init". The output is "Initialized empty Git repository in E:/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard/.git/". The prompt then changes to "Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)".

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard
$ git init
Initialized empty Git repository in E:/DevOps Workshop/git & github/Task Scheduling Dashboard/Ta
skSchedulingDashboard/.git/
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskS
chedulingDashboard (master)
```

*Git Initialize example*

**git init** creates an empty Git repository or re-initializes an existing one. It basically creates a `.git` directory with subdirectories and template files. Running a `git init` in an existing repository will not overwrite things that are already there. It rather picks up the newly added templates.

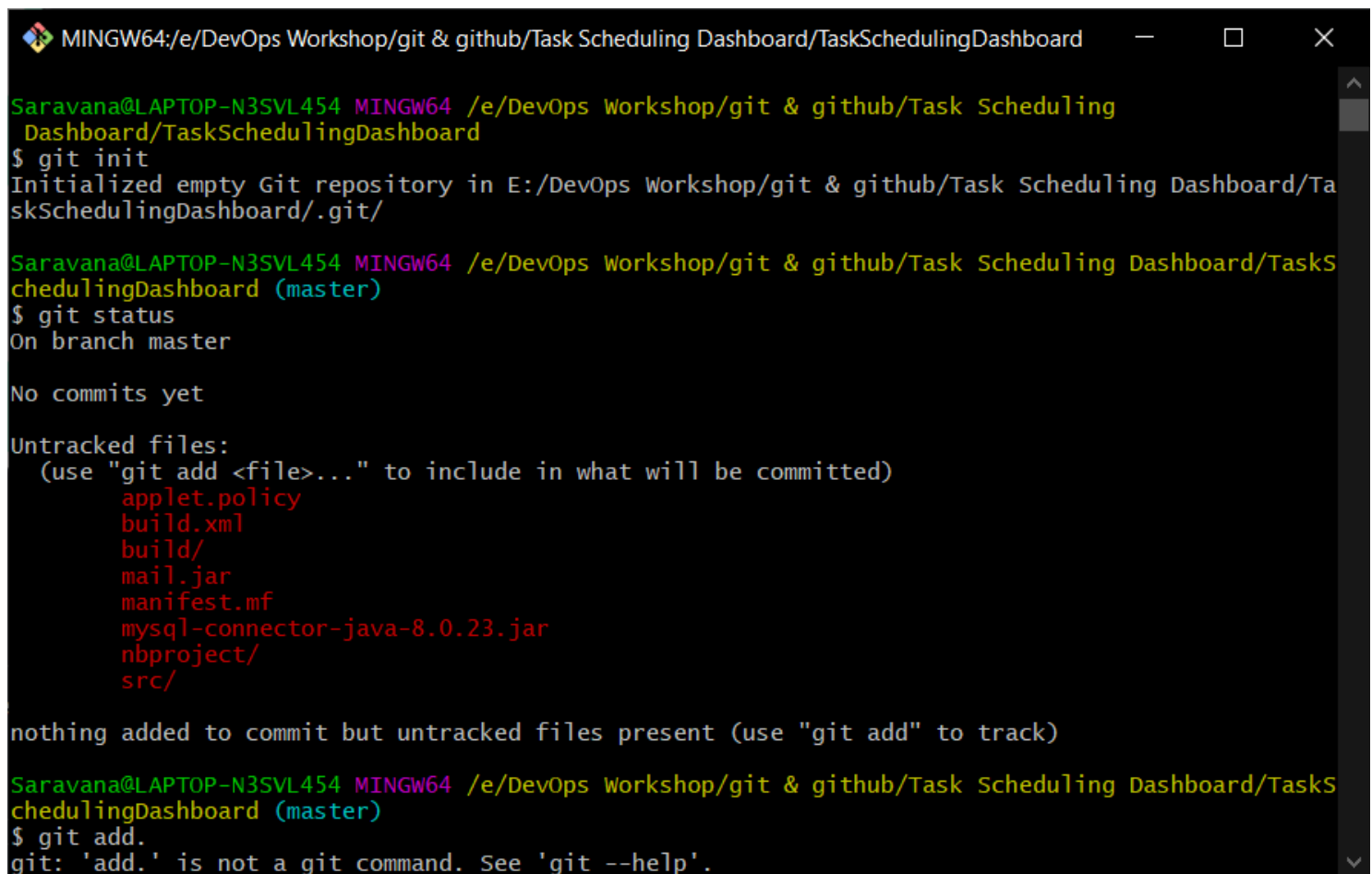
Now that my repository is initialized, let me create some files in the directory/repository. Let's see if these files are in my index or not using the command `git status`. The index holds a

snapshot of the content of the working tree/directory, and this snapshot is taken as the contents for the next change to be made in the local repository.

## Git Status

The `git status` command lists all the modified files which are ready to be added to the local repository.

Let us type in the command to see what happens:

A screenshot of a Windows Command Prompt window with a black background and white text. The title bar at the top reads 'MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard'. The prompt shows a user named 'Saravana@LAPTOP-N3SVL454' in a 'MINGW64' environment. The user has navigated to the directory '/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard'. They have executed '\$ git init', which initialized an empty Git repository. Then they executed '\$ git status', which reported 'On branch master' and 'No commits yet'. It listed untracked files: 'applet.policy', 'build.xml', 'build/', 'mail.jar', 'manifest.mf', 'mysql-connector-java-8.0.23.jar', 'nbproject/', and 'src/'. A message at the bottom says 'nothing added to commit but untracked files present (use "git add" to track)'. Finally, they typed '\$ git add.', which resulted in an error: 'git: 'add.' is not a git command. See 'git --help'.'

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard
$ git init
Initialized empty Git repository in E:/DevOps Workshop/git & github/Task Scheduling Dashboard/Ta
skSchedulingDashboard/.git/

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/Tasks
chedulingDashboard (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    applet.policy
    build.xml
    build/
    mail.jar
    manifest.mf
    mysql-connector-java-8.0.23.jar
    nbproject/
    src/

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

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/Tasks
chedulingDashboard (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.
```

*Git Status example*

This shows that I have two files that are not added to the index yet. This means I cannot commit changes with these files unless I have added them explicitly in the index.

## Add

This command updates the index using the current content found in the working tree and then prepares the content in the staging area for the next commit.

Thus, after making changes to the working tree, and before running the `commit` command, you must use the `add` command to add any new or modified files to the index. For that, use the commands below:

**`git add <directory>`**

or

**`git add <file>`**

Let me demonstrate the `git add` for you so that you can understand it better.

Let us add the files using the command `git add -A` or `git add .` This command will add all the files to the index, which are in the directory but not updated in the index yet.



```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
    add

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git add .
warning: LF will be replaced by CRLF in build/classes/Form/Bundle.properties.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople.form.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople1.form.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople1.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople10.form.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople10.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople11.form.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople11.java.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/AdminView/AddPeople12.form.
```

*Git Add . example*

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in src/userview/userprofile1.java.
The file will have its original line endings in your working directory

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git status
On branch master

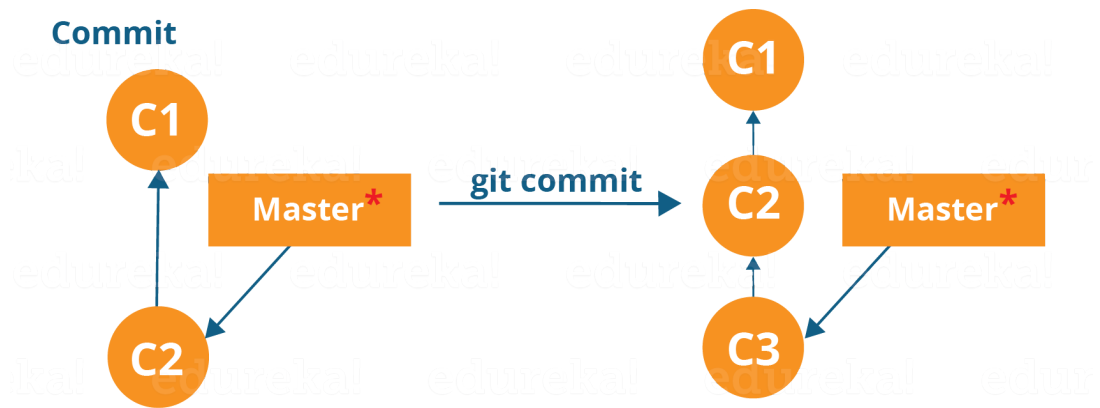
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   applet.policy
    new file:   build.xml
    new file:   build/built-jar.properties
    new file:   build/classes/AdminView/AddPeople$1.class
    new file:   build/classes/AdminView/AddPeople$2.class
    new file:   build/classes/AdminView/AddPeople.class
    new file:   build/classes/AdminView/AddPeople1$1.class
    new file:   build/classes/AdminView/AddPeople1$2.class
    new file:   build/classes/AdminView/AddPeople1.class
    new file:   build/classes/AdminView/AddPeople10$1.class
    new file:   build/classes/AdminView/AddPeople10$2.class
    new file:   build/classes/AdminView/AddPeople10.class
    new file:   build/classes/AdminView/AddPeople11$1.class
    new file:   build/classes/AdminView/AddPeople11$2.class
    new file:   build/classes/AdminView/AddPeople11.class
    new file:   build/classes/AdminView/AddPeople12$1.class
    new file:   build/classes/AdminView/AddPeople12$2.class
    new file:   build/classes/AdminView/AddPeople12.class
```

Now that the new files are added to the index, you are ready to commit them.

## Commit

It refers to recording snapshots of the repository at a given time. Committed snapshots will never change unless done explicitly. Let me explain how commit works with the diagram below :



*Git Command Workflow*

Here, C1 is the initial commit, i.e. the snapshot of the first change from which another snapshot is created with changes named C2. Note that the master points to the latest commit.

Now, when I commit again, another snapshot C3 is created and now the master points to C3 instead of C2.

Git aims to keep commits as lightweight as possible. So, it doesn't blindly copy the entire directory every time you commit; it includes commit as a set of changes, or "delta" from one version of the repository to the other. In easy words, it only copies the changes made in the repository.

You can commit by using the command below:

### **git commit**

This will commit the staged snapshot and will launch a text editor prompting you for a commit message.

Or you can use:

**git commit -m "<message>"**

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git config user.name "saravana"

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git config user.email "nadarsaravanasundar@gmail.com"

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git commit -m "First Commit"
[master (root-commit) 26da107] First Commit
604 files changed, 53210 insertions(+)
create mode 100644 applet.policy
create mode 100644 build.xml
create mode 100644 build/built-jar.properties
create mode 100644 build/classes/AdminView/AddPeople$1.class
create mode 100644 build/classes/AdminView/AddPeople$2.class
create mode 100644 build/classes/AdminView/AddPeople.class
create mode 100644 build/classes/AdminView/AddPeople1$1.class
create mode 100644 build/classes/AdminView/AddPeople1$2.class
create mode 100644 build/classes/AdminView/AddPeople1.class
create mode 100644 build/classes/AdminView/AddPeople10$1.class
create mode 100644 build/classes/AdminView/AddPeople10$2.class
create mode 100644 build/classes/AdminView/AddPeople10.class
create mode 100644 build/classes/AdminView/AddPeople11$1.class
create mode 100644 build/classes/AdminView/AddPeople11$2.class
create mode 100644 build/classes/AdminView/AddPeople11.class
create mode 100644 build/classes/AdminView/AddPeople12$1.class
create mode 100644 build/classes/AdminView/AddPeople12$2.class
```

```
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ git status
On branch master
nothing to commit, working tree clean

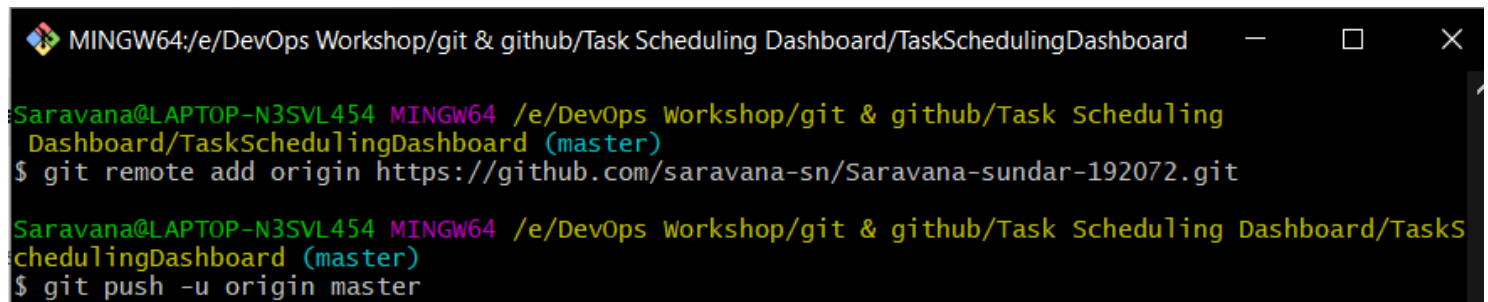
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)
$ |
```

## Copy your remote repository's URL from GitHub.

- The HTTPS or URL is copied from the given GitHub account, which is the place of the remote repository. [saravana-sn/Saravana-sundar-192072: DevOps Workshop Assignment - GIT & GITHUB TASK](https://github.com/saravana-sn/Saravana-sundar-192072: DevOps Workshop Assignment - GIT & GITHUB TASK)

Add the URL copied, which is your remote repository to where your local content from your repository is pushed.

- `git remote add origin 'your_url_name'`
- In the above code, The 'origin' is the remote name, and the remote URL is "[saravana-sn/Saravana-sundar-192072: DevOps Workshop Assignment - GIT & GITHUB TASK](https://github.com/saravana-sn/Saravana-sundar-192072: DevOps Workshop Assignment - GIT & GITHUB TASK)". You can see the remote as GitHub in this case, and GitHub provides the URL for adding to the remote repository.

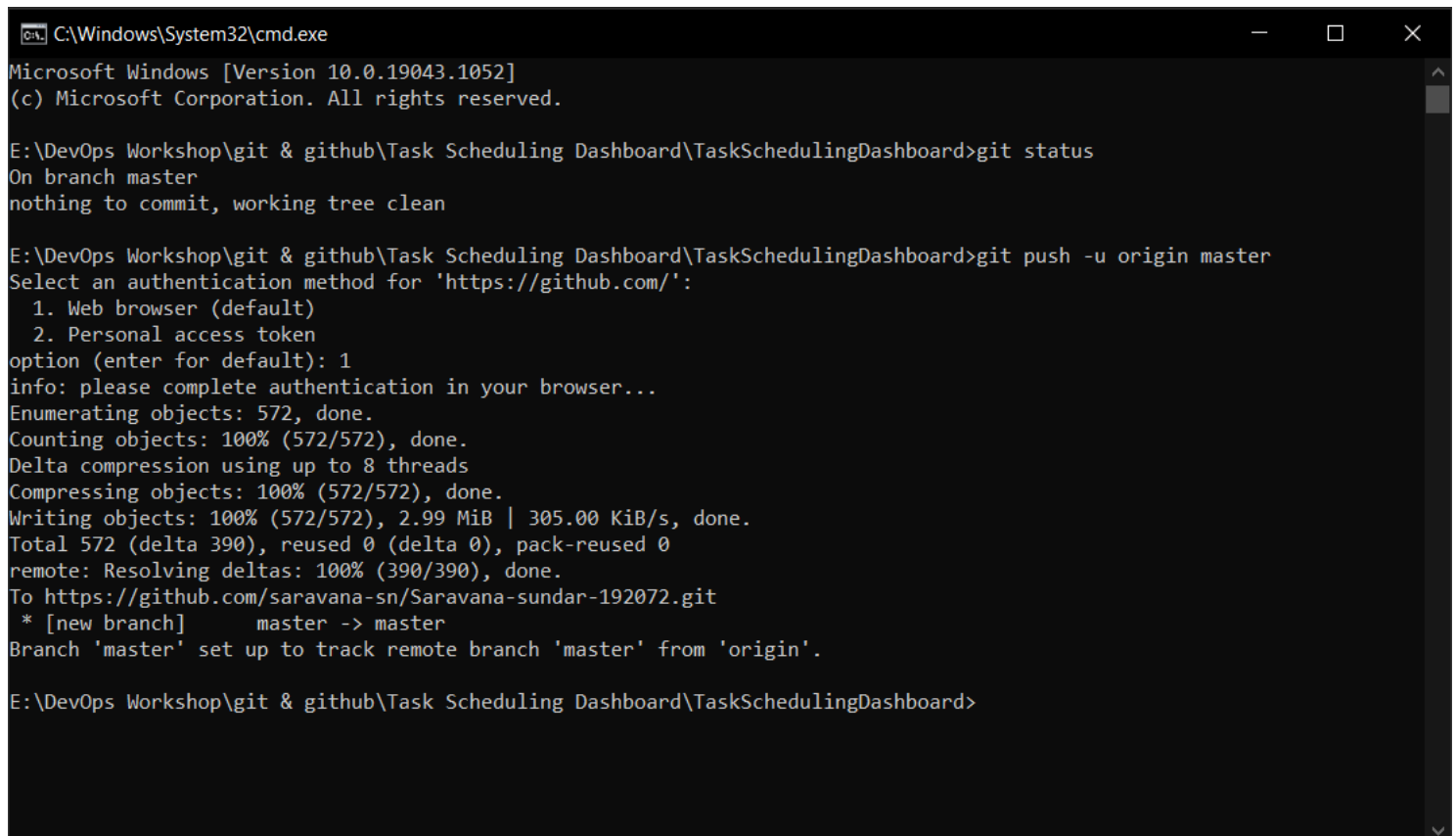
A screenshot of a Windows terminal window with a black background and green text. The window title is 'MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard'. The prompt is 'Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)'. The user enters the command '\$ git remote add origin https://github.com/saravana-sn/Saravana-sundar-192072.git'. The prompt changes to 'Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard (master)'. The user enters the command '\$ git push -u origin master'.

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard (master)
$ git remote add origin https://github.com/saravana-sn/Saravana-sundar-192072.git

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskS
chedulingDashboard (master)
$ git push -u origin master
```

## Push the code in your local repository to GitHub

- `git push -u origin master` is used for pushing local content to GitHub.
- In the code, the origin is your default remote repository name and '-u' flag is upstream, which is equivalent to '-set-upstream.' and the master is the branch, name.upstream is the repository that we have cloned the project.
- Fill in your GitHub username and password.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1052]
(c) Microsoft Corporation. All rights reserved.

E:\DevOps Workshop\git & github\Task Scheduling Dashboard\TaskSchedulingDashboard>git status
On branch master
nothing to commit, working tree clean

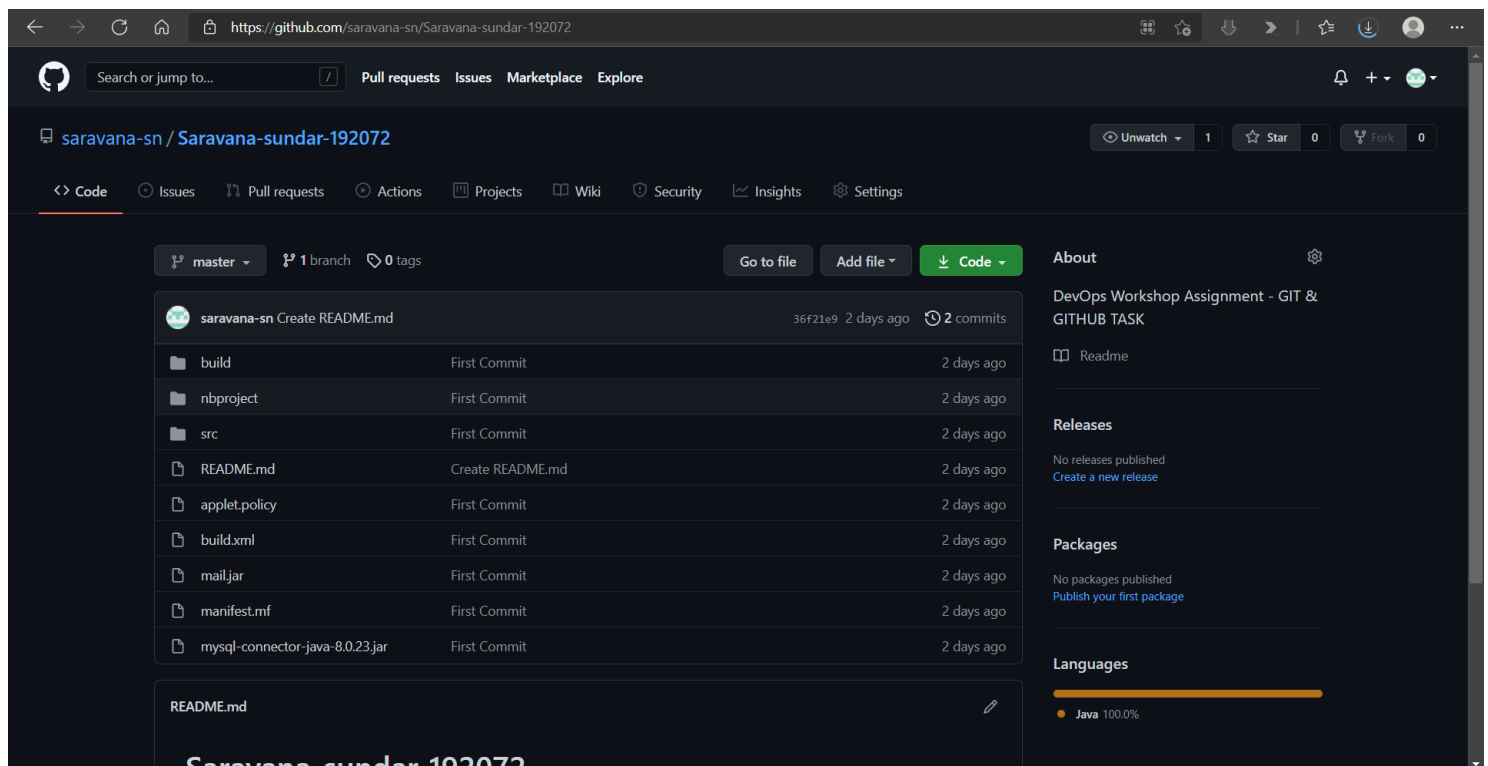
E:\DevOps Workshop\git & github\Task Scheduling Dashboard\TaskSchedulingDashboard>git push -u origin master
Select an authentication method for 'https://github.com/':
 1. Web browser (default)
 2. Personal access token
option (enter for default): 1
info: please complete authentication in your browser...
Enumerating objects: 572, done.
Counting objects: 100% (572/572), done.
Delta compression using up to 8 threads
Compressing objects: 100% (572/572), done.
Writing objects: 100% (572/572), 2.99 MiB | 305.00 KiB/s, done.
Total 572 (delta 390), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (390/390), done.
To https://github.com/saravana-sn/Saravana-sundar-192072.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

E:\DevOps Workshop\git & github\Task Scheduling Dashboard\TaskSchedulingDashboard>
```

\* git push was not responding in my git bash so I ran git push in the cmd.

View your files in your repository hosted on GitHub.

- You can finally see the file hosted on GitHub.



Link to my public repository :

<https://github.com/saravana-sn/Saravana-sundar-192072>

## Now , lets see some advanced git operations

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing

## Branching

Branches in Git are nothing but pointers to a specific commit. Git generally prefers to keep its branches as lightweight as possible.

There are basically two types of branches viz. *local branches* and *remote-tracking branches*.

A local branch is just another path of your working tree. On the other hand, remote-tracking branches have special purposes. Some of them are:

- They link your work from the local repository to the work on the central repository.
- They automatically detect which remote branches to get changes from when you use

`git pull`.

You can check what your current branch is by using the command:

**git branch**



The one mantra that you should always be chanting while branching is “branch early, and branch often”

To create a new branch, we use the following command:

**git branch <branch-name>**

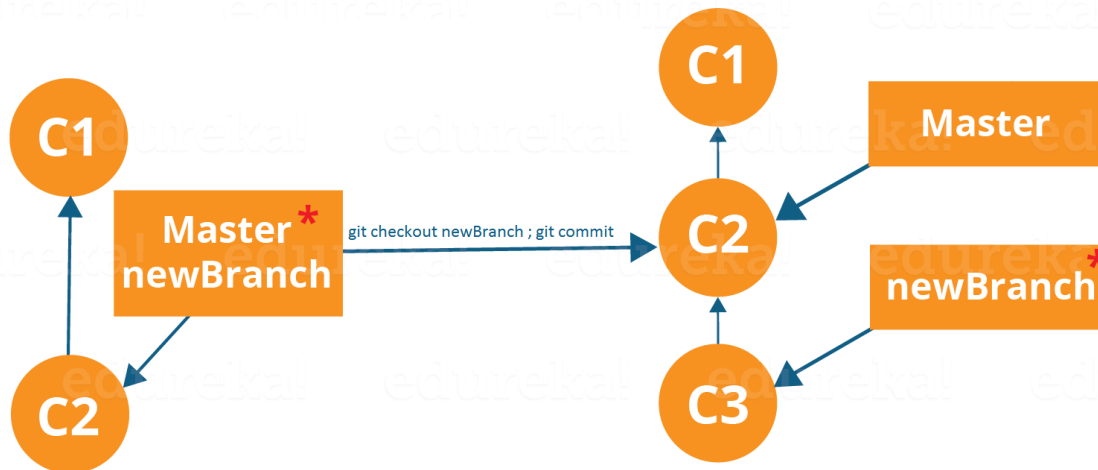


*Creating a Branch Workflow in Git*

The diagram above shows the workflow when a new branch is created. When we create a new branch, it originates from the master branch itself.

Since there is no storage/memory overhead with making many branches, it is easier to logically divide up your work rather than have big chunky branches.

Now, let us see how to commit using branches.



*Commit Using the Branches Workflow*

Branching includes the work of a particular commit along with all parent commits. As you can see in the diagram above, the `newBranch` has detached itself from the master and hence will create a different path.

Use the command below:

**git checkout <branch\_name>**

```
MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard (master)
$ git branch Branch1

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard (master)
$ git checkout Branch1
Switched to branch 'Branch1'

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling
Dashboard/TaskSchedulingDashboard (Branch1)
$
```

## Branching in Git

Here, I have created a new branch named “Branch1” and switched on to the new branch using the command `git checkout`.

One shortcut to the above commands is:

**`git checkout -b[ branch_name]`**

This command will create a new branch and check out the new branch at the same time.

Now while we are in the branch **Branch1**, add and commit the text file *edureka6.txt* using the following commands:

**`git add Branch.txt`**

**`git commit -m” adding Branch.txt in Branch1”`**

```
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (master)
$ git checkout Branch1
Switched to branch 'Branch1'

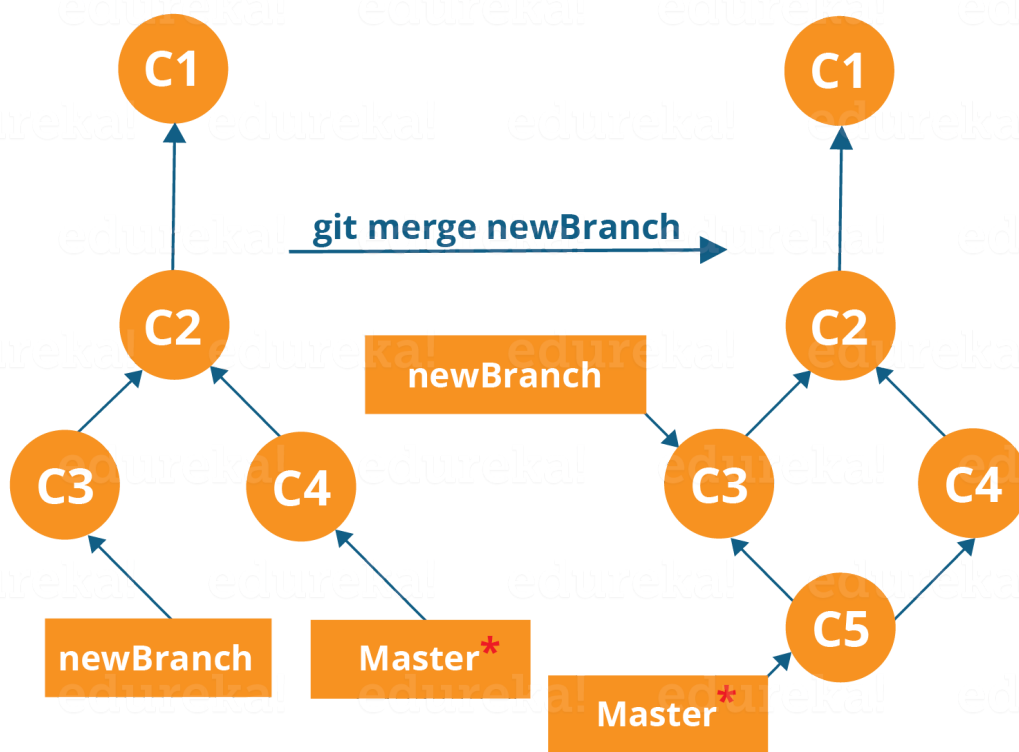
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (Branch1)
$ git add Branch.txt

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (Branch1)
$ git commit -m "Adding Branch.txt in Branch1"
[Branch1 ed7e8d7] Adding Branch.txt in Branch1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Branch.txt

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (Branch1)
$
```

## Merging

Merging is the way to combine the work of different branches together. This will allow us to branch off, develop a new feature, and then combine it back in.



*Merging Workflow in Git*

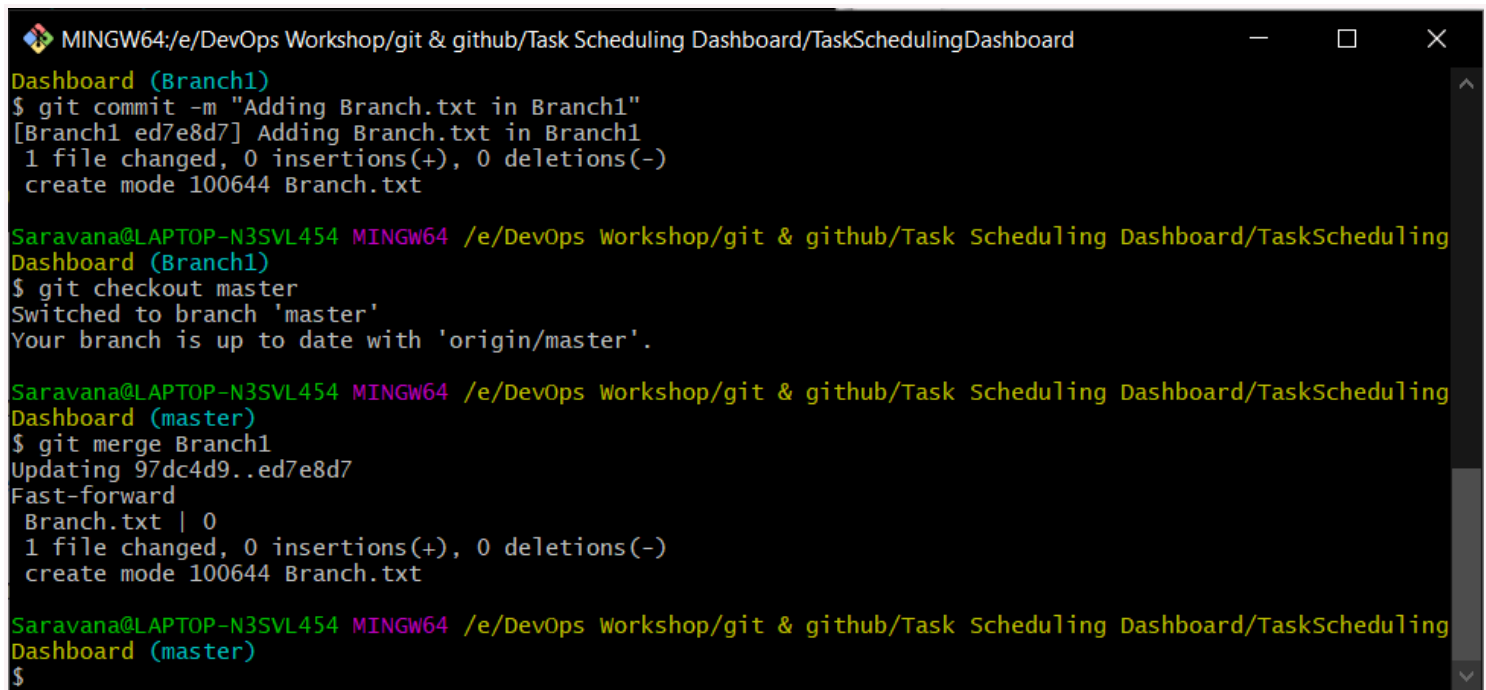
The diagram above shows us two different branches-> newBranch and master. Now, when we merge the work of newBranch into master, it creates a new commit that contains all the work of master and newBranch.

Now, let us merge the two branches with the command below:

**git merge <branch\_name>**

It is important to know that the branch name in the above command should be the branch you want to merge into the branch you are currently checking out. So, make sure that you are checked out in the destination branch.

Now, let us merge all of the work of the branch `EdurekaImages` into the master branch. For that, I will first check out the master branch with the command `git checkout master` and merge `Branch1` with the command `git merge Branch1`.

A terminal window titled 'MINGW64:/e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskSchedulingDashboard'. The terminal shows the following commands and output:  
1. `Dashboard (Branch1)`  
\$ git commit -m "Adding Branch.txt in Branch1"  
[Branch1 ed7e8d7] Adding Branch.txt in Branch1  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 Branch.txt  
2. `Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (Branch1)`  
\$ git checkout master  
Switched to branch 'master'  
Your branch is up to date with 'origin/master'.  
3. `Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (master)`  
\$ git merge Branch1  
Updating 97dc4d9..ed7e8d7  
Fast-forward  
Branch.txt | 0  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 Branch.txt  
4. `Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (master)`  
\$

### *Git Merge*

As you can see above, all the data from the branch name are merged to the master branch.

Now, the text file *Branch.txt* has been added to the master branch.

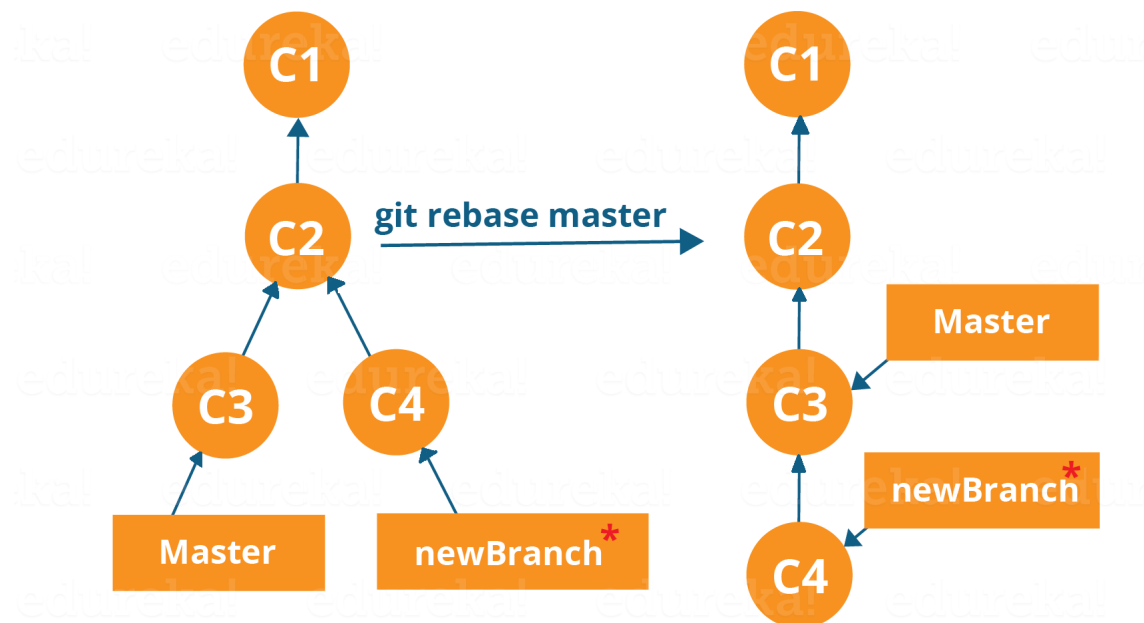
Merging in Git creates a special commit that has two unique parents.

## Rebasing

This is also a way of combining the work between different branches. Rebasing takes a set of commits, copies them, and stores them outside your repository.

The advantage of rebasing is that it can be used to make a linear sequence of commits. The commit log or history of the repository stays clean if rebasing is done.

Let us see how it happens.



### *Rebasing in Git*

Now, our work from `newBranch` is placed right after master and we have a nice linear sequence of commits.

**Note:** *Rebasing also prevents upstream merges, meaning you cannot place master right after newBranch.*

Now, to rebase master, type the command below in your Git Bash:

**git rebase master**

```
Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (master)
$ git rebase master
Current branch master is up to date.

Saravana@LAPTOP-N3SVL454 MINGW64 /e/DevOps Workshop/git & github/Task Scheduling Dashboard/TaskScheduling Dashboard (master)
$ |
```

### *Rebase in Git*

This command will move all our work from the current branch to the master. They look as if they are developed sequentially, but they are developed parallelly.