

**GITHUB: AI-POWERED DEVELOPER PLATFORM**  
**(Effective from the Academic Year 2023 - 2024)**  
**IV SEMESTER**

Course Code	CS42297CA	CIA Marks	50
Number of Contact Hours/Week (L: T: P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Exam Hours	03

## Activity Manual

### Activity-0 (Git Fundamentals)

#### 1. Introduction to Git

##### Q. What is Version Control system?

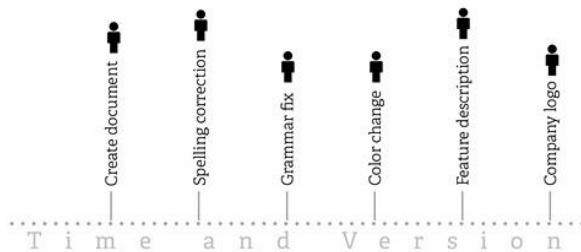
Daily we do tasks and it follow the cycle

1. Create 2. Save 3. Edit 4 save again. (Edit and save is ever ending process)

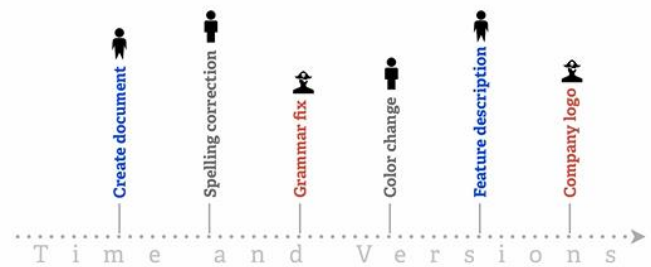
Saving the things again and again is where we need to **maintain version** (when you edited/created, why you did, and what is the change)

**Version control system records the changes to a files or set of files over times so that they can recall any specific version later.**

#### History Tracking



#### Collaborative History Tracking



(Who changed it, why they change, when they changed, what is the change)

**Note: Versions are store in place called "repository":- 1. Local 2. Centralized 3. Distributed.**

**1. Local VCS**→ all the task and its changes are locally tracked. (Individual developer work)

**2. Centralized VCS**→ all the task and its changes are tracked in centralized system. (Multiple developer work)

**3. Distributed VCS**→ all developer will have their own local copy in their system which they can collaborate with others by merging them and pushing them to remote repository.

**Q. What is the advantage of version control system?**

Collaboration, manage versions, rollback, analyse the project.

**Q. what is repository**

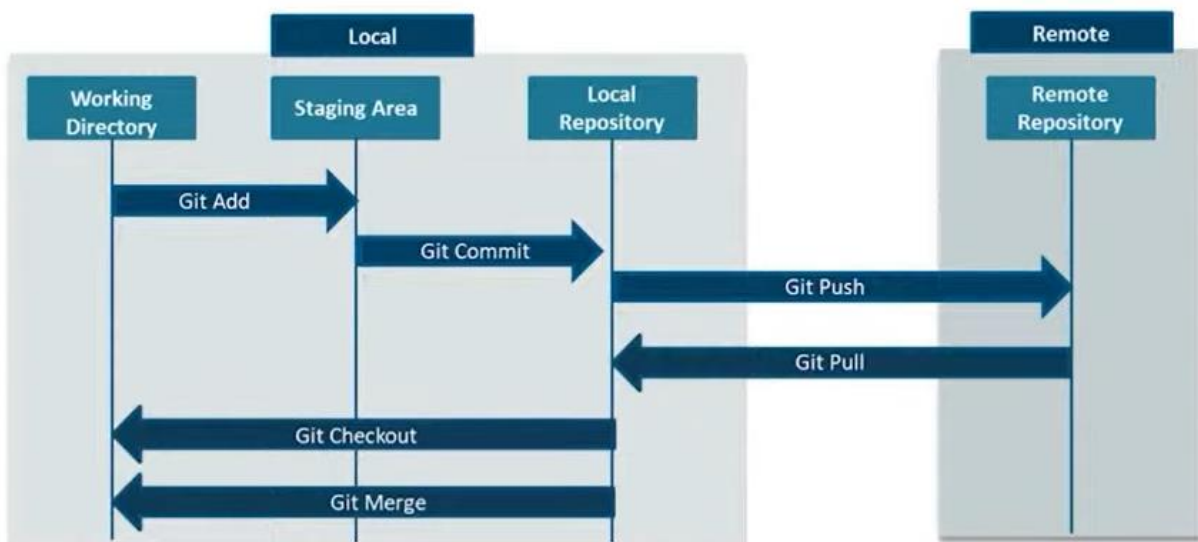
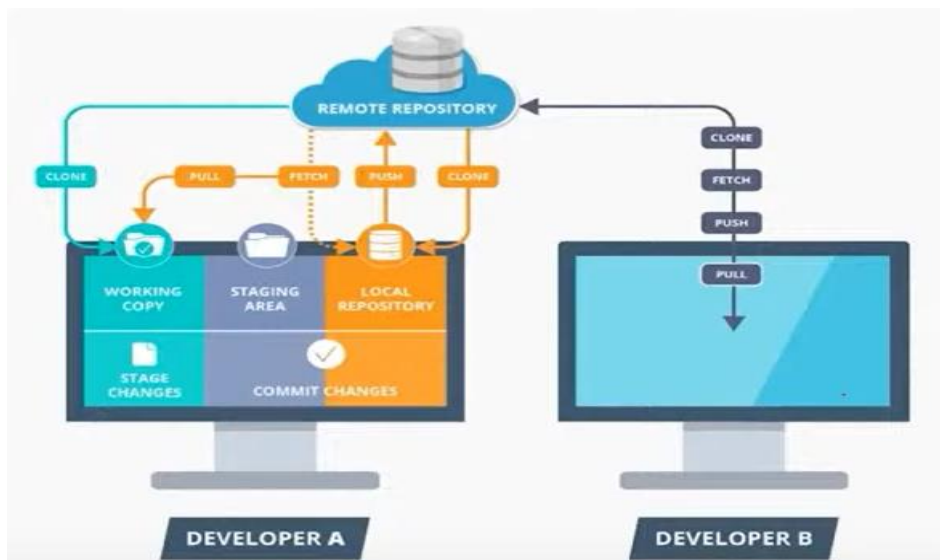
A Repository or repo is a directory or storage space where we keep all our project.

Example: local system or github

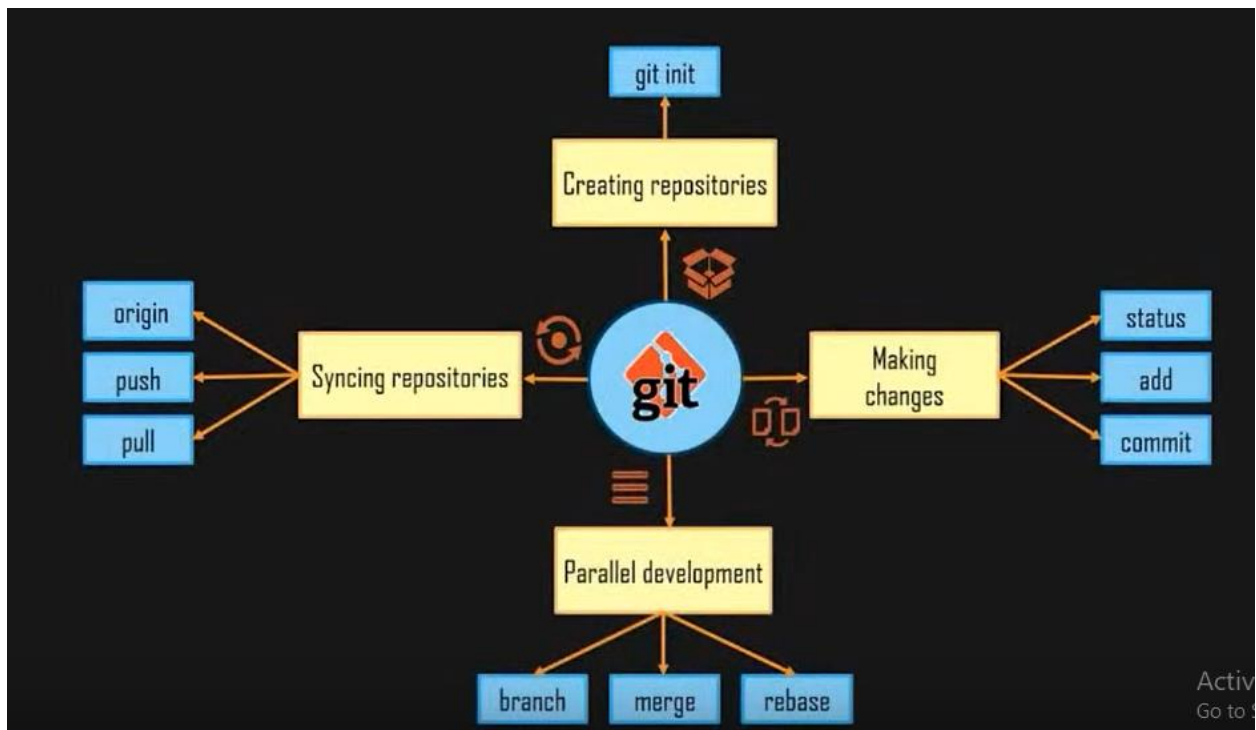
**Q what is Git**

Git is distributed version control system, which provide you version control for your project (code, document, animation, images etc)

Example:- hands-on



## Q. Basic Git operations



## Activity-1 (Setting Up and Basic Commands)

Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

### Q. Git installation and setup

Step-1: Install GitBash for windows

<https://git-scm.com/downloads>

Note :- for unix user

```
sudo apt update
```

```
sudo apt install git
```

Step-2: To check the version of git installed in your computer use below command

```
>git -v
```

Step-3 create the folder on any location in your computer

```
>mkdir sri( sri is your folder name)
```

Step-4 go inside folder using command

```
>cd sri
```

Step-5 make the test folder as your local repository using command

```
>git init
```

Step-6 check status of the git using command

```
>git status
```

```
C:\sri>git status
On branch master

No commits yet

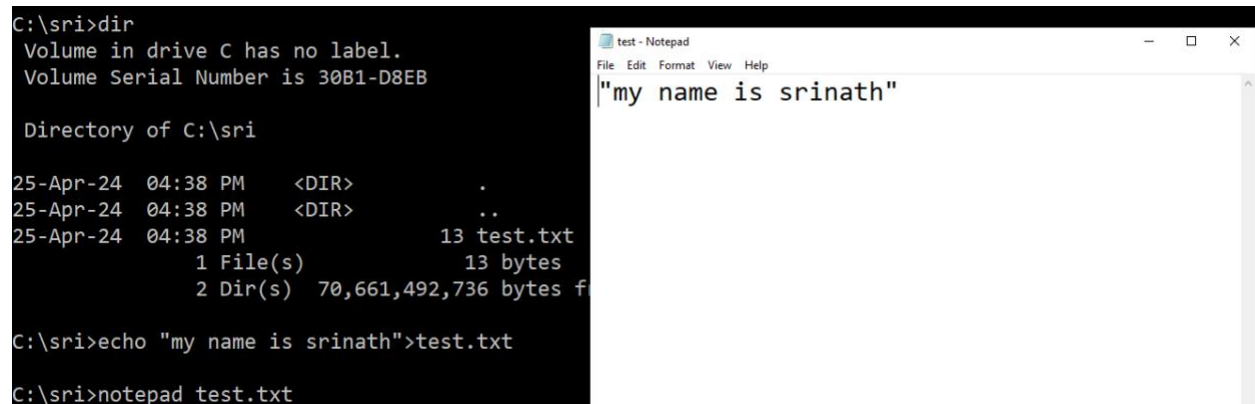
nothing to commit (create/copy files and use "git add" to track)
```

Step -7 create a text file "test.txt"

```
>echo>test.txt
```

Step-8 write something in a text file using command and open it in notepad and check the content.

```
>echo "my name is srinath">test.txt
```



Step-9:

Now check the status of git using command

```
>git status
```

```
C:\sri>git status
On branch master

No commits yet

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

        test.txt

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

Step-10:

Put your files from working area to staging area using command

>git add test.txt

Note: you can use command >git add . to add multiple files from working directory to staging area ( . indicate all the file )

Note: check the status of the file using >git status and you can see the file will be in green which means the file is now ready to add in local repository.

```
C:\sri>git add test.txt

C:\sri>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   test.txt
```

Step-11:

To add the file from working directory to local repository use the following command

>git commit -m "this is my first commit"

```
C:\sri>git commit -m "this is my first commit"
[master (root-commit) 8bebf5a] this is my first commit
1 file changed, 1 insertion(+)
create mode 100644 test.txt
```

Note: - 8bebf5a is the commit reference value used for maintain versions

## Activity-2 (Creating and Managing Branches)

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

Step-1 Create a new branch named "feature-branch" and switch to it using command

```
>git checkout -b feature-branch
```

Step-2 Make your changes and commit them to the "feature-branch".

Step-3 To Switch back to the "master" branch use command

```
> git checkout master
```

Step-3 To Merge the changes from "feature-branch" into "master" use command

```
> git merge feature-branch
```

## Activity-3 (Creating and Managing Branches)

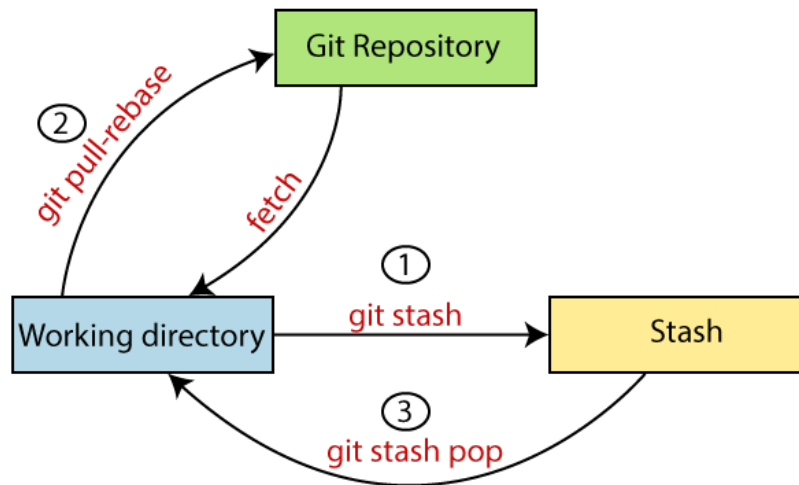
Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Q what is stash in git ?? What are its uses ??

"Sometime you don't want the changes made on file/code to be committed or discard before switching from current branch then you can keep it in stash later when you come back to that branch you can pull it back from stash"

In Git, a stash is a way to temporarily store changes that you've made to your working directory so that you can switch to another branch or perform other operations without committing those changes. It's particularly useful when you're in the middle of working on something but need to switch to a different task or branch.

When you stash changes, Git saves the modified tracked files and staged changes away for later use, leaving you with a clean working directory. Stashing is like putting your changes aside on a shelf temporarily.



Here's how stashing works in Git:

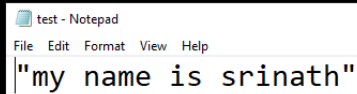
1. **Stash changes:** You use the `git stash` command to stash your changes. This command takes the modified tracked files (those that are already being tracked by Git) and staged changes and saves them away.
2. **Perform other operations:** After stashing your changes, you can perform other operations, such as switching branches or pulling changes from a remote repository, without worrying about committing incomplete or experimental changes.
3. **Apply or pop stashed changes:** When you're ready to continue working on the changes you stashed, you can either apply or pop the stashed changes back into your working directory.
  - o `>git stash apply`: This command applies the most recent stash to your working directory but leaves the stash intact. You can apply a specific stash by providing its reference.
  - o `>git stash pop`: This command applies the most recent stash and removes it from the stash list. Like `apply`, you can also specify a specific stash to pop.
4. **Clear or drop stashed changes:** If you no longer need the stashed changes, you can remove them from the stash list using the `>git stash drop` command. If you want to remove all stashed changes, you can use `>git stash clear`.

Step-1: Go to the working directory and open the file and see the changes using command

> notepad test.txt

```
C:\sri>notepad test.txt
```

```
C:\sri>
```



test - Notepad  
File Edit Format View Help  
"my name is srinath"

Step-2: check the git status using command

```
C:\sri>git status  
On branch master  
nothing to commit, working tree clean
```

Note:- if your working tree is clean then do not commit or else use git commit command and commit it.

Step-3: Make some changes in the file and stash the changes using command

> git stash

Note : after this command it will give random stash name/reference  
(eg:8bebf5a)

```
C:\sri>git stash  
Saved working directory and index state WIP on master: 8bebf5a this is my first commit
```

Note: if you want to give perfect name/reference then use following command  
>git stash save "add your stash name"

Step 4: To get the stash log use command

> git stash list

Step 5: to see what is there in the saved stash before applying it back use command

>git stash show

or to see full detail use command

>git stash show -p or (>git stash show stash{2} -p)

Step 6: to apply any particular stash and keep the stash entry as it is without deleting it then use command

> git stash apply (recent stash is applied)

Or

>git stash apply stash@{2}

Note:- use command >git stash pop to take recent stash back to file and delete the saved stash automatically from stash list.



Note: After applying the stashed changes, you might want to delete the stash entry if you no longer need it then use command  
> git stash drop

Note: if you want to clear entire stash list then use command  
> git stash clear

Note: if you want to create new branch with a stash then use command  
>git stash branch new\_feature stash@{0}

## Activity-4 (Collaboration and Remote Repositories)

Q. Clone a remote Git repository to your local machine.

Step-1 To clone a remote Git repository to your local machine, you would typically use the git clone command followed by the URL of the repository you want to clone.

```
> git clone <repository_url>
```

Example:-git clone https://github.com/example/repository.git

This command will create a new directory on your local machine with the name of the repository and will download all of its contents, including its entire version history.

Note:- If you need to clone a repository from a specific branch, you can specify the branch using the -b option followed by the branch name:

```
> git clone -b <branch_name> <repository_url>
```

Example:git clone -b development https://github.com/example/repository.git

*Note: - follow my class instruction*

## Activity-5 (Collaboration and Remote Repositories)

Q. Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch, you can follow these steps:

Step-1: First, ensure you're on the branch you want to update:

```
>git checkout your_local_branch
```

Step-2: Replace your\_local\_branch with the name of your local branch.

Step-3: Fetch the latest changes from the remote repository:

```
>git fetch origin
```

This command fetches the latest changes from the remote repository named "origin" (you can replace "origin" with the name of your remote if it's different).

Step-4: Rebase your local branch onto the updated remote branch:

```
git rebase origin/your_remote_branch
```

Replace your\_remote\_branch with the name of the branch in the remote repository you want to rebase onto.

Note:- If there are conflicts during the rebase process, resolve them as instructed by Git.

Once the rebase is complete, push your changes to the remote repository if necessary:

```
> git push origin your_local_branch --force
```

Use the --force option only if you have rebased commits that have already been pushed to the remote repository. Be cautious when using --force as it overwrites history on the remote repository.

## **Activity-6 (Collaboration and Remote Repositories)**

Q. Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge

To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git.

Step-1: Make sure you are in the "master" branch by switching to it:

```
$ git checkout master
```

Step-2. Create a new branch named "feature-branch" and switch to it:

```
$ git checkout -b feature-branch
```

This command will create a new branch called "feature-branch" and switch to it.

Step-3. Make your changes in the "feature-branch" by adding, modifying, or deleting files as

needed.

Step-4. Stage and commit your changes in the "feature-branch":

```
$ git add .
```

```
$ git commit -m "Your commit message for feature-branch"
```

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."

Step-5. Switch back to the "master" branch:

```
$ git checkout master
```

Step-6. Merge the "feature-branch" into the "master" branch:

```
$ git merge feature-branch
```

This command will incorporate the changes from the "feature-branch" into the "master" branch.

Now, your changes from the "feature-branch" have been merged into the "master" branch.

Your project's history will reflect the changes made in both branches

## Activity-7 (Git Tags and Releases)

Q. Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Git Tags is used to **label** the commits by creating a tag (reference) or it is a pointer for git history (bunch of commit in your repository).

Note : Tags are used to mark the version of the software such as ( v1, v1.1.0 v2, v2.1, v2.2.2 etc)

There are two types of tags

1. Light weighted tags: - they are just name/label, it is just a pointer.
2. Annotated tags:- it store extra meta data including author name and email, date and tagging message

Steps-1: create working directory/folder "Tagdemo" using command

```
>mkdir TagDemo
```

```
C:\Users\Aptra>mkdir TagDemo  
  
C:\Users\Aptra>
```

Step-2: go inside the working directory and initialize this directory as git repository using command

```
>cd TagDemo
```

```
>git init
```

```
C:\Users\Aptra>cd TagDemo  
  
C:\Users\Aptra\TagDemo>git init  
Initialized empty Git repository in C:/Users/Aptra/TagDemo/.git/  
  
C:\Users\Aptra\TagDemo>
```

Step-2: create file "test.txt" and put it to staging and then commit it using following command

```
>echo>test.txt
```

```
>git add test.txt
```

```
>git commit -m "first commit: test.txt created"
```

```
C:\Users\Aptra\TagDemo>echo>test.txt  
  
C:\Users\Aptra\TagDemo>git add test.txt  
  
C:\Users\Aptra\TagDemo>git commit -m "first commit: test.txt created"  
[master (root-commit) 7b2b3d0] first commit: test.txt created  
1 file changed, 1 insertion(+)  
create mode 100644 test.txt  
  
C:\Users\Aptra\TagDemo>
```

Step-3: now create the tag and check whether the tags are created using command

```
>git tag v1.0.0
```

```
C:\Users\Aptra\TagDemo>git tag v1.0.0

C:\Users\Aptra\TagDemo>git tag
v1.0.0

C:\Users\Aptra\TagDemo>
```

Step-4 add one line ("my name is srinath") inside test.txt file and make 2<sup>nd</sup> commit and then add tag (v2.0.0)

```
C:\Users\Aptra\TagDemo>echo "my name is srinath">test.txt

C:\Users\Aptra\TagDemo>type test.txt
"my name is srinath"

C:\Users\Aptra\TagDemo>git add test.txt

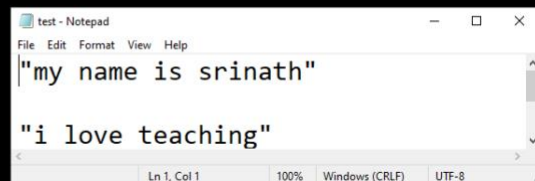
C:\Users\Aptra\TagDemo>git commit -m "second commit: line added inside file"
[master 2891386] second commit: line added inside file
1 file changed, 1 insertion(+), 1 deletion(-)
```

Step-5: add 2<sup>nd</sup> line (I love teaching) in the test.txt file and make 3<sup>rd</sup> commit and then add tag (v3.0.0)

```
C:\Users\Aptra\TagDemo>echo. >>test.txt

C:\Users\Aptra\TagDemo>echo "i love teaching">>test.txt

C:\Users\Aptra\TagDemo>
```



```
C:\Users\Aptra\TagDemo>git add test.txt

C:\Users\Aptra\TagDemo>git commit -m "third commit: 2nd line is added"
[master dea6ff9] third commit: 2nd line is added
1 file changed, 2 insertions(+)
```

```
C:\Users\Aptra\TagDemo>git tag v3.0.0
```

Step-6: check list of all tag available using command

>git tag

```
C:\Users\Aptra\TagDemo>git tag
v1.0.0
v2.0.0
v3.0.0
```

Step-7: to see any particular tag and its detail use command

>git show "tag name"

example :

>git show v2.0.0

```
C:\Users\Aptra\TagDemo>git show v2.0.0
commit 2891386a482e6fcfd0de3b1f0b4d40bdea965561 (tag: v2.0.0)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date: Tue May 14 18:33:51 2024 +0530

    second commit: line added inside file

diff --git a/test.txt b/test.txt
index 4d6a3b4..81d0b62 100644
--- a/test.txt
+++ b/test.txt
@@ -1,1 @@
-ECHO is on.
+"my name is srinath"

C:\Users\Aptra\TagDemo>
```

Step-8: To see the difference between two versions (i.e v1.0.0 and v3.0.0) use below command

>git diff v1.0.0 v3.0.0

```

C:\Users\Aptra\TagDemo>git diff v1.0.0 v3.0.0
diff --git a/test.txt b/test.txt
index 4d6a3b4..d601d29 100644
--- a/test.txt
+++ b/test.txt
@@ -1,3 @@
-ECHO is on.
+"my name is srinath"
+
+"i love teaching"

C:\Users\Aptra\TagDemo>

```

Note:- if you want to go to any particular tag use command

>git checkout "tag name"

```

C:\Users\Aptra\TagDemo>git log
commit dea6ff9876908c08647fd72965a9f04e4a7c5b88 (HEAD -> master, tag: v3.0.0)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date:   Wed May 15 15:09:17 2024 +0530

    third commit: 2nd line is added

commit 2891386a482e6fcfd0de3b1f0b4d40bdea965561 (tag: v2.0.0)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date:   Tue May 14 18:33:51 2024 +0530

    second commit: line added inside file

commit 7b2b3d0cff88854e08953b09f1fccdd678f1182e (tag: v1.0.0)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date:   Tue May 14 18:21:05 2024 +0530

    first commit: test.txt created

```

You can see from above figure, right now we are in master tag

```
C:\Users\Aptra\TagDemo>git checkout v2.0.0
Note: checking out 'v2.0.0'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

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

```
HEAD is now at 2891386 second commit: line added inside file
```

You can we are now checked out to v2.0.0 version.

Note :- if you want to delete any tag then use command

```
>git tag -d v3.0.0      (d indicate delete) (v3.0.0 is the version number)
```

## Activity-8 (Advanced Git Operations)

Q. Write the command to cherry-pick a range of commits from "source-branch" to the current branch. ("pick a commit from one branch and place in another branch" we use cherry-pick)

Cherry pick is used if you want to apply particular commit from one branch into another branch

Note: if you don't want to merge whole branch and you want some of the commits then cherry pick is helpful.

Cherry pick is same as rebase

Note : it is advised not to use cherry pick always, because it will cause duplicate commits

Steps-1: create new folder/project to demonstrate cherry pick

```
> mkdir cherrypick
```

```
C:\Users\Aptra> mkdir cherrypick
```

```
C:\Users\Aptra>
```



Step-2: go inside the folder cherrypick and initialize the git repository using the command

```
> cd cherrypick
```

```
> git init
```

```
C:\Users\Aptra>cd cherrypick

C:\Users\Aptra\cherrypick>git init
Initialized empty Git repository in C:/Users/Aptra/cherrypick/.git/

C:\Users\Aptra\cherrypick>
```

Step-3 : create a file "sri.txt" and add it to staging area then commit it using command

```
>echo > sri.txt
```

```
>git add sri.txt
```

```
>git commit -m "sri.txt is added"
```

```
C:\Users\Aptra\cherrypick>echo >sri.txt

C:\Users\Aptra\cherrypick>dir
Volume in drive C has no label.
Volume Serial Number is 30B1-D8EB

Directory of C:\Users\Aptra\cherrypick

13-May-24  10:37 AM    <DIR>          .
13-May-24  10:37 AM    <DIR>          ..
13-May-24  10:37 AM                13 sri.txt
               1 File(s)                13 bytes
               2 Dir(s)  68,797,263,872 bytes free

C:\Users\Aptra\cherrypick>git add sri.txt
```

```
C:\Users\Aptra\cherrypick>git commit -m "sri.txt is added"
[master (root-commit) 7a75ae5] sri.txt is added
1 file changed, 1 insertion(+)
create mode 100644 sri.txt
```

Step-4 : check the git log for the commit msg using command

> git log

```
C:\Users\Aptra\cherrypick>git log
commit 7a75ae5ee6fa98c20748a73165888557f204327 (HEAD -> master)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date:   Mon May 13 10:38:33 2024 +0530

    sri.txt is added

C:\Users\Aptra\cherrypick>
```

Or you can use shorthand command

>git log --oneline

```
C:\Users\Aptra\cherrypick>git log --oneline
7a75ae5 (HEAD -> master) sri.txt is added

C:\Users\Aptra\cherrypick>
```

Step-5 create three versions of the project that is three branches called v1, v2 and v3 and check the branches create or not using command

>git branch v1

>git branch v2

> git branch v3

Check

>git branch

```
C:\Users\Aptra\cherrypick>git branch v1  
C:\Users\Aptra\cherrypick>git branch v2  
C:\Users\Aptra\cherrypick>git branch v3  
C:\Users\Aptra\cherrypick>git branch  
* master  
  v1  
  v2  
  v3  
C:\Users\Aptra\cherrypick>
```

Step-6 go to v3 branch and create a file "test.txt" and add it to staging and then commit it using command  
>git checkout v3

```
C:\Users\Aptra\cherrypick>git checkout v3  
Switched to branch 'v3'
```

```
C:\Users\Aptra\cherrypick>echo>test.txt  
C:\Users\Aptra\cherrypick>git add test.txt  
C:\Users\Aptra\cherrypick>git commit -m "test.txt file is created in v3"  
[v3 e3d6d37] test.txt file is created in v3  
1 file changed, 1 insertion(+)  
create mode 100644 test.txt
```

Note: if there is a bug in the V3 then we need to fix the bug, so we simulate this bugfix by adding bugfix.txt file in this branch (create bugfix.txt file, add to staging and then committing it) using command

```

C:\Users\Aptra\cherrypick>echo>bugfix.txt

C:\Users\Aptra\cherrypick>git add bugfix.txt

C:\Users\Aptra\cherrypick>git commit -m "bugfix.txt is create to fix bug"
[v3 c057a7c] bugfix.txt is create to fix bug
1 file changed, 1 insertion(+)
create mode 100644 bugfix.txt

C:\Users\Aptra\cherrypick>

```

From below figure you can see bugfix is made only in v3 branch and hence and now we need apply this commit in v1 and v2 as well but don't want to merge this v2 bugfix in v1 and v2 because we are still working on it and not completed the fix fully. Hence in this scenario we go for cherry pick to copy the current entry into v2 and v1.

```

C:\Users\Aptra\cherrypick>git log --oneline
c057a7c (HEAD -> v3) bugfix.txt is create to fix bug
e3d6d37 test.txt file is created in v3
7a75ae5 (v2, v1, master) sri.txt is added

```

So to perform this cherry pick follow the below steps

Step-1 copy the bug fix hash ( i.e c057a7c see below figure ) using Ctrl C command

```

C:\Users\Aptra\cherrypick>git log --oneline
c057a7c (HEAD -> v3) bugfix.txt is create to fix bug
e3d6d37 test.txt file is created in v3
7a75ae5 (v2, v1, master) sri.txt is added

```

Step-2: checkout from V3 branch to V1 and then use cherry pick command to apply the commit as show below

> git checkout v1

>dir

Note : you see only one file (i.e sri.txt is available before cherry pick command is applied)

```

C:\Users\Aptra\cherrypick>git checkout v1
Switched to branch 'v1'

C:\Users\Aptra\cherrypick>dir
Volume in drive C has no label.
Volume Serial Number is 30B1-D8EB

Directory of C:\Users\Aptra\cherrypick

13-May-24  02:32 PM    <DIR>          .
13-May-24  02:32 PM    <DIR>          ..
13-May-24  10:37 AM                  13 sri.txt
                1 File(s)                13 bytes
                2 Dir(s)  69,257,932,800 bytes free

```

After applying cherry-pick command you see both files (i.e sri.txt and bugfix.txt)

>git cherry-pick c057a7c ( note : c057a7c is the hash of bugfix file)

```

C:\Users\Aptra\cherrypick>git cherry-pick c057a7c
[v1 96117fd] bugfix.txt is create to fix bug
Date: Mon May 13 14:04:59 2024 +0530
1 file changed, 1 insertion(+)
create mode 100644 bugfix.txt

C:\Users\Aptra\cherrypick>dir
Volume in drive C has no label.
Volume Serial Number is 30B1-D8EB

Directory of C:\Users\Aptra\cherrypick

13-May-24  02:34 PM    <DIR>          .
13-May-24  02:34 PM    <DIR>          ..
13-May-24  02:34 PM                  13 bugfix.txt
13-May-24  10:37 AM                  13 sri.txt
                2 File(s)                26 bytes

```

Step-3: Repeat above step2 to apply the changes in v2 branch also

## Activity-9 (Analysing and Changing Git History)

Q. Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

Step -1 : use git branch command to see which branch you are in

> git branch

```
C:\Users\Aptra\cherrypick>git branch
master
* v1
v2
v3
```

Step-2 check the log to get hash and commit message using following command

>git log --oneline

```
C:\Users\Aptra\cherrypick>git log --oneline
96117fd (HEAD -> v1) bugfix.txt is create to fix bug
7a75ae5 (v2, master) sri.txt is added
```

Step-3: To view the details of a specific commit, including the author, date, and commit message, you can use the git show command

>git show <commit\_id>

Example : >git show 96117fd

```
C:\Users\Aptra\cherrypick>git show 96117fd
commit 96117fd820e9be20931df8478be2d29a09167087 (HEAD -> v1)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date: Mon May 13 14:04:59 2024 +0530

    bugfix.txt is create to fix bug

diff --git a/bugfix.txt b/bugfix.txt
new file mode 100644
index 0000000..4d6a3b4
--- /dev/null
+++ b/bugfix.txt
@@ -0,0 +1 @@
+ECHO is on.
```

Note : if you want to see only author, date and commit message use the following command

\$ git log -n 1 <commit-ID>

Example : git log -n 1 96117fd

```
C:\Users\Aptra\cherrypick>git log -n 1 96117fd
commit 96117fd820e9be20931df8478be2d29a09167087 (HEAD -> v1)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date: Mon May 13 14:04:59 2024 +0530

    bugfix.txt is create to fix bug

C:\Users\Aptra\cherrypick>
```

Note: if you want to see only commit message the use command

> git log --format=%B -n 1 96117fd

```
C:\Users\Aptra\cherrypick>git log --format=%B -n 1 96117fd
bugfix.txt is create to fix bug

C:\Users\Aptra\cherrypick>
```

## Activity-10 (Analysing and Changing Git History)

Q. Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

Step-1 : To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31," you can use the git log command with the --author and --since and --until option

```
> git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"
C:\Users\Aptra\cherrypick>git log --author="SolanSrinath" --since="2024-01-01" --until="2024-05-30"
commit 96117fd820e9be20931df8478be2d29a09167087 (HEAD -> v1)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date: Mon May 13 14:04:59 2024 +0530

    bugfix.txt is create to fix bug

commit 7a75ae5ee6fa98c20748a73165888557f204327 (v2, master)
Author: SolanSrinath <50046113+SolanSrinath@users.noreply.github.com>
Date: Mon May 13 10:38:33 2024 +0530

    sri.txt is added

C:\Users\Aptra\cherrypick>
```

Activate Windows  
Go to Settings to activate Windows.

## Activity-11 (Analysing and Changing Git History)

Q. Write the command to display the last five commits in the repository's history.

To display the last five commits in the repository's history, you can use the git log command with the -n option to limit the number of commits displayed. Here's the command:



```
> git log -n 5
```

Note:- If you want a more condensed view, you can use the --oneline option to display each commit on a single line:

```
> git log -n 5 --oneline
```

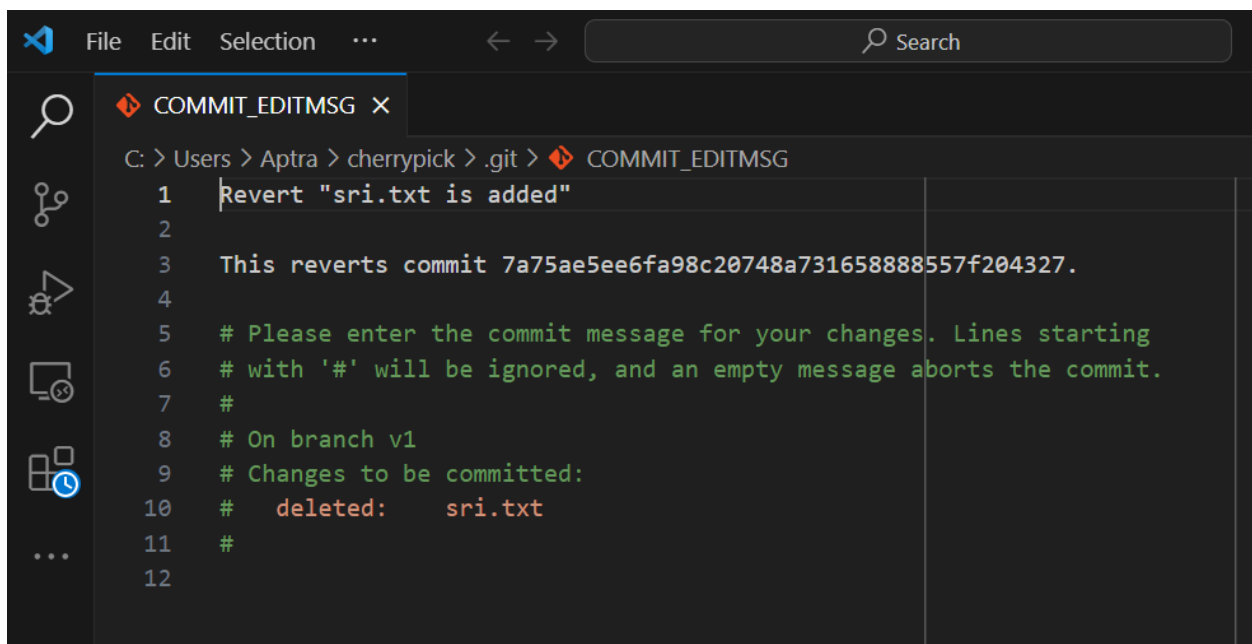
## Activity-12 (Analysing and Changing Git History)

Q. Write the command to undo the changes introduced by the commit with the ID "abc123".

To undo the changes introduced by a specific commit with the ID "abc123", you can use the "git revert" command. This command creates a new commit that undoes the changes made by the specified commit.

```
> git revert abc123
```

Sometimes It shows message waiting editor to close



The screenshot shows a code editor window titled "COMMIT\_EDITMSG" with a search bar at the top. The editor displays the following text:

```
C: > Users > Aprta > cherrypick > .git > COMMIT_EDITMSG
1  Revert "sri.txt is added"
2
3  This reverts commit 7a75ae5ee6fa98c20748a731658888557f204327.
4
5  # Please enter the commit message for your changes. Lines starting
6  # with '#' will be ignored, and an empty message aborts the commit.
7  #
8  # On branch v1
9  # Changes to be committed:
10 #   deleted:   sri.txt
11 #
12
```

```
C:\Users\Aptra\cherrypick>git log -n 5 --oneline
96117fd (HEAD -> v1) bugfix.txt is create to fix bug
7a75ae5 (v2, master) sri.txt is added

C:\Users\Aptra\cherrypick>git revert 7a75ae5
[v1 aa3ab35] Revert "sri.txt is added"
1 file changed, 1 deletion(-)
delete mode 100644 sri.txt
```

Note : If you want to revert multiple commits, you can specify a range of commits. For example, to revert all commits from "abc123" to "def456", you can use:

```
> git revert abc123..def456
```

Note: Remember that git revert does not remove the original commit from the history; instead, it creates a new commit that undoes the changes introduced by the specified commit. If you want to completely remove a commit from the history, you can use "git rebase" or "git reset", but these commands should be used with caution