5.1 | Git - DVCS | Master Notes

Main Tool Name	Git and Github
Date Created	@November 4, 2022 6:35 PM
Progress?	Need to work with Anvith and redo the notes

Table of Contents

▼ 1. Git basics

▼ About Version Control

What is Version Control?

Version control, also known as source control, is the practice of **tracking** and **managing** changes to software code.

What is Version Control System?

Version control systems are **software tools** used to **track** and **manage** changes to software code.

Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

What are the benefits of VCS?

1. Saves from creating multiple backup files
2. Allows for multiple people work on the same set of files
3. track changes and author of those changes
4. easy to roll back to older version as and when required
5. Increases the productivity of software development cycle

What are types of Version control System?

Local version control system

Centralised version control system

Distributed version control system

What is Local version control system (LVCS)?

Local version control system is a type of VCS that **tracks and manages changes** to files **locally** in the form of patch sets (i.e difference between two files) and recreates what any file looked by adding up patches

What are Drawbacks of LVCS

Lack of collaboration when multiple users work on the same code.

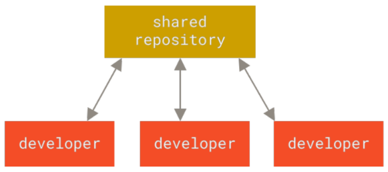
Database corruption

What are Popular LVCS?

RCS

What is Centralised version control system (CVCS)?

The CVCS is type of VCS that **tracks and manages changes** in a **shared repository** which is hosted on a **server**, where clients/users checkout from.



What are Popular CVCS?

CVS

Subversion (SVN)

Perforce

What are the advantages of CVCS?

Everyone knows what everyone else is doing on the project

Administer can control who can do what

Tracking changes in CVCS is far easier than LVCS

Strong tool for non linear development

What are the disadvantages of CVCS?

The main disadvantages is the **single point of failure**

If server goes down, teams cannot **collaborate or save versions.**

Database becoming corrupted or losing all data risks the project development cycle.

What is Distributed Version Control System (DVCS)?

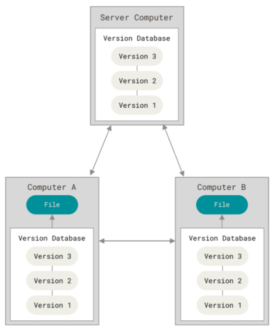


Figure 3. Distributed version control

A distributed version control system (DVCS) is a type of VCS that **tracks and manages changes** into a **repository** that is **stored both locally and in server**. Here, every user in the working group will have a **mirror image** of the repository and its full version history.

What are the advantages of DVCS?

Can work offline

Faster branching and merging

Everybody has a backup in case central repository fails

What are the disadvantages of DVCS?

Having large project leads to having large binary files

(NTU)

What are the popular DVCS?

Git

Mercurial

Bazaar

▼ First time Setup in Git

What is git config?

It is a tool used to define configuration variables that determines how git looks and operates

where does git config store its data?

etc/gitconfig file

.config file

How to view all of git config settings and the path they are stored?

`git config --list --show-orgin`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git config --list --show-origin
file:/home/sanjay/.gitconfig    user.name=sanjay
file:/home/sanjay/.gitconfig    user.email=emailsanjayrg@gmail.com
file:/home/sanjay/.gitconfig    init.deafultbranch=main
file:/home/sanjay/.gitconfig    core.editor=vim
file:/home/sanjay/.gitconfig    credential.helper=
file:/home/sanjay/.gitconfig    credential.helper=/usr/local/bin/git-credential-manager
file:/home/sanjay/.gitconfig    credential.credentialstore=secretservice
file:/home/sanjay/.gitconfig    credential.https://dev.azure.com.usehttppath=true
```

```
file:.git/config        core.repositoryformatversion=0
file:.git/config        core.filemode=true
file:.git/config        core.bare=false
file:.git/config        core.logallrefupdates=true
```

How to add user name (sanjay) to the git configuration?

`git config --global user.name "sanjay"`

How to add email id (emailsanjayrg@gmail.com) to the git configuration?

`git config --global user.email " emailsanjayrg@gmail.com”`

How to change the default text editor such as vim?

`git config --global core.editor vim`

What is the default branch git creates during new repository?

master

What is the default branch github creates during new repository?

main

How to set the default branch name as main

`git config --global init.defaultBranch main`

How to list all the configurations stored in git config?

`git config --list`

▼ Getting Help

How to get a concise help for operating commands in git? ex. for merge

`git merge -h`

How to get a detailed help for operating commands in git? ex. for merge (similar to man page)

`git merge --help`

▼ Getting a Git repository

What are the two ways to create a git repo?

cloning an existing repo from a remote repo &

Turning an existing local directory to git repo

How to initialise an existing directory to git repo?

we run `git init` at the desired path inside a terminal.

What does `git init` command do?

It creates a new sub-directory called "**.git**" that contains all of **necessary metadata required run git operations.**
At this point, nothing in your project is **tracked yet**.

How to make git track files inside the directory where `git init` was run?

we **add** the files to the staging area and do **commit** to make the git track files.

What is the command to clone an existing repo from Github? ex: url is https://github.com/githubtraining/hellogitworld.git

`git clone  https://github.com/githubtraining/hellogitworld.git`

What exactly does `git clone` do? ex: https://github.com/githubtraining/hellogitworld.git

git clone essentially does the following five processes:

1. **creates** a directory named "**hello world"**,
2. **initializes** a **.git sub-directory inside** the directory,
3. Creates a bookmark for the remote url, generally "origin"
4. it pulls down all the data from that remote repository into the directory created,
5. checks out a **working copy** of the latest version into git repository.

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ ls -l
total 12
-rw-rw-r-- 1 sanjay sanjay     0 Nov 30 19:00  file01.txt
-rwxrwxr-x 1 sanjay sanjay 10066 Nov 25 13:18 'hello world.odt'
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git clone https://github.com/githubtraining/hellogitwor
Cloning into 'hellogitworld'...
remote: Enumerating objects: 306, done.
remote: Total 306 (delta 0), reused 0 (delta 0), pack-reused 306
Receiving objects: 100% (306/306), 95.81 KiB | 131.00 KiB/s, done.
Resolving deltas: 100% (69/69), done.
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ ls -l
total 16
-rw-rw-r-- 1 sanjay sanjay     0 Nov 30 19:00  file01.txt
drwxrwxr-x 5 sanjay sanjay  4096 Dec  1 13:00  hellogitworld
-rwxrwxr-x 1 sanjay sanjay 10066 Nov 25 13:18 'hello world.odt'
```

How to clone a repo from github and name the working copy differently ex: instead of "hello world" rename it as "namaste" url: https://github.com/githubtraining/hellogitworld.git

Here we pass an extra argument:

`git clone  https://github.com/githubtraining/hellogitworld.git  namaste`

This is used to name the directory - differently not the remote name.

What types of protocol does git use to transfer data?

https

ssh

▼ Recording changes to the repository [status, add, commit, rm, mv, restore

▼ What are the different states of files in the working directory?

There are:

Tracked files

Untracked files

What are tracked files?

Tracked files are files that were **in the last commit**, as well **staging area**; they can be **unmodified, modified, or staged**.

In short, tracked files are files that **Git knows about.**

What are modified files? (refer to previous section)

What are untracked files?

Untracked files are any files the working directory that were not in last commit and are not in staging area

How to check the status of files (i.e to see which files are in which state)?

`git status`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git status
On branch master #which branch we are in
Your branch is up to date with 'origin/master'. #informs you that it has not diverged from the same branch on server

nothing to commit, working tree clean
```

How to track new files (i.e make git know about) ex: readme

`git add readme`

How to stage a new file (ex. readme) to the staging area (or index or preview before commit snapshot)?

`git add readme`

If you commit at this point, the ***version*** *of the file* ***at the time you ran git add*** *is what will be in the subsequent* ***historical snapshot***

How to stage all files in the working directory that were both new, modifed and/or delted or renamed in single instance?

`git add .`

what is the state of a tracked file when it is edited or changed?

modified

What is the command to get short status in git?

`git status -s`

What are the notation “M, MM, A, ??” mean - which appears during `git status -s` command?

M - Modified file

?? - untracked file

A - new files added to staging area

MM - a tracked (old) file was staged and then it was modified again.

What is the command to see the differences/changes made to files present in working directory wrt to files that was present in latest commit?

`git diff`

what is the command to see differences/changes made to files that is present in staging area wrt to files that was present in latest commit?

`git diff --staged`

What is the command to see the differences or changes after we have staged a file wrt to previous commit?

`git diff --staged`

What is the command to run diff with a difftool?

`git difftool` for `git diff`

`git difftool --staged` for `git diff --staged`

How to commit / take a snapshot of a file that is in staging area? with an option to add message (ex: “nice weather”) while committing?

`git commit -m “nice weather”`

How to commit every file that is already tracked, maybe modified but not staged? with a message “skip the staging area using the option -a”

`git commit -am "skip the staging area using the option -a"`

This will not work files that are new and untracked

What is the command to remove a file (ex. fix.txt) from working directory (not from staging area)?

`rm fix.txt`

The above changes need to be both staged and committed as fix.txt is a tracked file.

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ rm fix.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   fix.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  deleted:    fix.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git diff
diff --git a/fix.txt b/fix.txt
deleted file mode 100644
index 7152294..0000000
--- a/fix.txt
+++ /dev/null
@@ -1 +0,0 @@
-changed to check git diff
```

How to restore the file (ex. fix.txt) that was deleted using only `rm fix.txt`?

`git restore fix.txt`

It is mainly used for: "git restore <file>..." to discard changes in working directory (here even deletion is a change)

This brings the file that was present as is in latest commit

What is the command to remove a file (ex. fix.txt) both from working directory and from staging area?

`git rm fix.txt`

If you modified the file or had already added it to the staging area, you must force the removal with the `-f` option.

here, the changes made (i.e deletion) need not be staged as it is removed from staging area as well. However the changes need to be commited to get the latest snapshot

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  deleted:    fix.txt
```

How to restore the file (ex. fix.txt) that was deleted using only `git rm fix.txt`?

`git restore --staged fix.txt`

How to rename a file in git that makes changes both in working directory and staging area? (ex. build.gradle to **b.g)**

`git mv build.gradle b.g`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  renamed:    build.gradle -> b.g
```

The above changes still need to be committed.

How to rename a file in git that makes changes only in working directory and not staging area? (ex. build.gradle to **b.g)**

`mv build.gradle b.g`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  deleted:    build.gradle

no changes added to commit (use "git add" and/or "git commit -a")
```

The above changes need to be both staged and committed. Hence `git mv` is a better option

---

▼ View the commit history

What is the command to see the commit history?

`git log`

*By default, with no arguments, git log lists the commits made in that repository in **reverse** chronological order; that is, the most recent commits show up first.*

What is the command to see the commit history? but limit to only the latest 3.

`git log -3`

What is the command to see the commit history? and see the patches wrt each commit?

`git log -p`

- *p or --patch, which shows the difference (the patch output) introduced in each commit.*
- *This option displays the same information but with a **diff directly following each entry.***

What is the command to see the commit history? and see the patches wrt each commit? and limit it to latest two commits?

`git log -p -2`

What is the command to see the commit history? in single line consisting of commit hash, & commit message?

`git log --pretty=oneline`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git log --pretty=oneline
ef7bebf8bdb1919d947afe46ab4b2fb4278039b3 (HEAD -> master, origin/master, origin/HEAD) Fix groupId after package refactor
ebbbf773431ba07510251bb03f9525c7bab2b13a Update package name, directory
45a30ea9afa413e226ca8614179c011d545ca883 Update package name, directory
9805760644754c38d10a9f1522a54a4bdc00fa8a Fix YAML name-value pair missing space
435ffceb7ba576c937e922766e37d4f7abdcc122 Merge pull request #35 from githubtraining/travis-yml-docker
7c5ee6478d137417ae602140c615e33aed91887c Add special option flag for Travis Docker use case
```

What is the command to see the commit history? to see in a graphical style?

`git log --graph`

What is the command to see the commit history? to see in a graphical style and in single line?

`git log --graph --pretty=oneline`

What is the command to summarize the commit history for publishing?

`git shortlog`

▼ Undoing things

▼ Undoing a commit:

What is the command to replace the latest commit with a new commit? which is used to incorporate changes that was forgotten to stage during previous commit and/or add a new commit message “replace the old commit”?

`git commit --amend -m "replace the old commit"`

▼ git restore

*Git version 2.23.0 introduced a new command: **git restore**. It’s basically an alternative to **git reset***

What is the command to unstage a file (i.e bring back to working directory from staging area) (ex. foo.txt)?

`git restore --staged foo.txt`

What is the command used to discard changes in the working directory that was not staged yet? (ex. foo.txt)

`git restore foo.txt`

The above command used to discard any sort of changes in the working directory, it includes, modification, deletion, rename etc.,

▼ Working with remotes

Here, remote means remote repositories.

By *default when we clone a project from github, a remote named as "origin" is added.*

*Remote connections are **more like bookmarks** rather than direct links into other repositories. Instead of providing real-time access to another repository, they serve as **convenient names** that can be used to reference a **not-so-convenient URL.***

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git remote -v
origin  https://github.com/githubtraining/hellogitworld.git (fetch)
origin  https://github.com/githubtraining/hellogitworld.git (push)
```

ALWAYS has both fetch and push types for each repo

*Remote repositories are **versions** of our project that are hosted on the Internet.* You can have several of them, each of which generally is either read-only or read/write for you.

💡 One repository can have multiple remotes

Managing remote repositories includes knowing **how to add remote repositories**, **remove remotes** that are no longer valid, **manage various remote branches** and define them as being tracked or not, and more.

What is the command to see the remote repositories we have configured ?
`git remote`

what is the default remote repo created after we have cloned?

origin

What is the command to see the remote repositories we have configured ? along with its url?

`git remote -v`

What is the command to add a new remote with a short name "tvremote" and url: https://github.com/paulboone/ticgit?

`git remote add tvremote` `https://github.com/paulboone/ticgit`

> *Here, the shortname is like a bookmark to access the url added to it. Post this, we can fetch / pull / push by just saying* `git push tvremote master`

What is the command to fetch ALL the information from a remote repo (ex. for a remote shortname "tvremote"), but NOT download in the local repository?

`git fetch tvremote`

*The command goes out to that remote project and **pulls down all the data** from that remote project that **you don't have yet.** After you do this, you should have **references to all the branches** from that remote, which you can **merge in or inspect at any time.***

If you clone a repository, the command automatically adds that remote repository under the name "origin". So, git fetch origin fetches any new work that has been pushed to that server since you cloned (or last fetched from) it.

*the git clone command automatically sets up your **local master branch** to track the **remote master branch** (or whatever the default branch is called) on the server you cloned from*

What is the command if you want to merge by default when pulling? (fast-forward if possible, else create a merge commit)?

`git config --global pull.rebase "false"`

What is the command if you want to rebase by default when pulling?

`git config --global pull.rebase "true"`

What is the command to push the changes made in “master” branch to a remote repository having a shortname as “tvremote”?

`git push tvremote master`

> *it says push the changes to the remote (bookmark to a url) named as “tvremote” to the branch named as “master”*
>
> *This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime.*
>
> > *If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You’ll have to fetch their work first and incorporate it into yours before you’ll be allowed to push.*

What is the command to see detailed information of a remote “origin” and its branches, its push and pull configured links etc.,?

`git remote show origin`

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git remote show tvremote
* remote tvremote
  Fetch URL: https://github.com/paulboone/ticgit
  Push  URL: https://github.com/paulboone/ticgit
  HEAD branch: master
  Remote branches:
    master new (next fetch will store in remotes/tvremote)
    ticgit new (next fetch will store in remotes/tvremote)
  Local ref configured for 'git push':
    master pushes to master (local out of date)
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice/hellogitworld$ git remote show origin
* remote origin
  Fetch URL: https://github.com/githubtraining/hellogitworld.git
  Push  URL: https://github.com/githubtraining/hellogitworld.git
  HEAD branch: master
  Remote branches:
    bisect                       tracked
    fbaddfiles                   tracked
    fbmaster                     tracked
    feature_division             tracked
    feature_division_polished    tracked
    feature_image                tracked
    feature_subtraction          tracked
    feature_subtraction_polished tracked
    gh-pages                     tracked
    master                       tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

What is the command to rename an existing remote (tvremote) to another name (acremote)?

`git remote rename tvremote acremote`

▼ Tagging

What is the literal meaning of tag?

A tag is a **label** that works as an additional identifier.

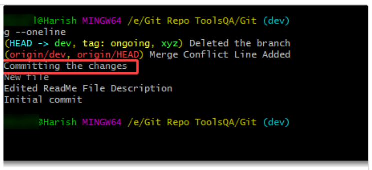
we often tag a weird name to a couple or a girl.

Explain the concept of tag in git?

What are Git Tags and How to create, remove, view and tagging in git?

"Attitude is like a price tag; it decides your value ". It is quite a typical quote and truly depicts the meaning of the word "tag ". Tag is not an alien word for any of us. A tag is a label that works as an additional identifier (or an identifier if there aren't any).

https://www.toolsqa.com/git/git-tags/



```
[redacted]l@Harish MINGW64 /e/Git Repo ToolsQA/Git (dev)
g --oneline
(HEAD -> dev, tag: ongoing, xyz) Deleted the branch
(origin/dev, origin/HEAD) Merge Conflict Line Added
Committing the changes
New file
Edited ReadMe File Description
Initial commit

[redacted]@Harish MINGW64 /e/Git Repo ToolsQA/Git (dev)
```

Define tagging in git in oneline?

Tagging in git is used to **label an object** (generally a commit) in the git history and is used to a **mark a version release.**

When is a commit tagged?

A commit is tagged, when it is the last commit before a version release. *For example, tagging a commit with release version 3.0 means that commit was the final commit before the launch of the 3.0 version of the software*

What is the command to list all the existing tags?

`git tag`

This command lists the tags in alphabetical order; the order in which they are displayed has no real importance.

What are the two types of tag in git?

Git supports two types of tags: lightweight and annotated.

What is a lightweight tag?

It is just a label to a specific object (generally a commit) in the project history.

Explain lightweight tags briefly?

Lightweight tags don't require a message or store other data.

It is the default git tag.

Git stores each lightweight tag **as a file** in the `.git/refs/tags` directory.

This is basically the commit checksum stored in a file — no other information is kept.

What is the command to tag the latest commit as "v1.5" ? (lightweight)

`git tag v1.5`

Here, no message because it is a lightweight tag

What is an annotated tag?

It is a label to a specific object (generally a commit) in the project history which also includes a **tag message, tagger name, email and date the tag** created.

> It's also worth mentioning that annotated tags **include security measures,** and can be signed and verified with GNU Privacy Guard (GPG).

This is marked as releases in Github

What is the command to tag (ex. v2.3) to the latest commit - with a message ("I am tagged, now I can be pointed out easily"), tagger name, email and date of tag?

`git tag -a v2.3 -m "I am tagged, now I can be pointed out easily"`

How to add a tag (ex. v9.8.7) to a commit that is not the latest commit, having commit cheksum value as "9fceb02" with a message (" tagging later")?

`git tag -a v9.8.7 -m "tagging later" 9fceb02`

How to add a single tag (ex. v1.2.0) to a remote repository?

`git push origin v1.2.0`

By default, the git push command doesn't transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them

How to add all the tags to a remote repository?

`git push origin --tags`

How to delete a tag (ex. v4.5)?

`git tag -d v4.5`

Note that this does not remove the tag from any remote servers

How to delete a tag (v87) in remote repo (with the shortname "pablo")?

`git push pablo --delete v87`

▼ Git Aliases

What is the literal meaning of alias?

a false or assumed identity. / aka / otherwise.

What is git aliases?

it is a tool to create a **new git command** to an **existing git command** which is used *frequently* or *long* or *difficult* to remember.

It is like creating a **shortcut** to a frequently used / long / difficult command.

What is the command to create a new git command for `git log --pretty=oneline --graph` as `git logp1g`?

`git config --global alias.logp1g "log --pretty=oneline --graph"`

💡 Here, do not use git as a suffix, do not use spaced words, symbols for new command.

▼ 2. Git branching

what is a branch in git?

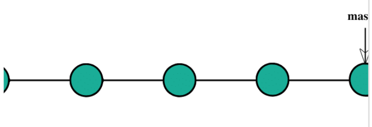
A branch is a stream where we commit our changes

More explanation:

What is a Branch in Git and the importance of Git Branches?

"Branch " is not an unfamiliar word. It is too common to hear the words such as the branch of a tree, branch of a bank, a branch of science, etc. We use them quite often in our lives. Today, you will link another meaning to the word "branches " in your already existing vocabulary.

https://www.toolsqa.com/git/branch-in-git/



Why do we create branches in git?

Branches create another line of development that is entirely different or isolated from the **main stable master branch.**

We create branches for the following:

1. When you're doing development work that is somewhat experimental in nature. [acts as a sandbox environment]
2. You want to solve a bug. Its better to solve it on other branch and merge it later
3. to keep the commits shorter in main branch (we do changes in branches and add as a single commit to master branch)

What is the command to create a new branch (ex. feature)?

`git branch feature`

What is a HEAD in git?

It is a pointer to the latest commit in the history of the branch.

What happens when you create a new branch?

It creates a new pointer (HEAD) to the commit history.

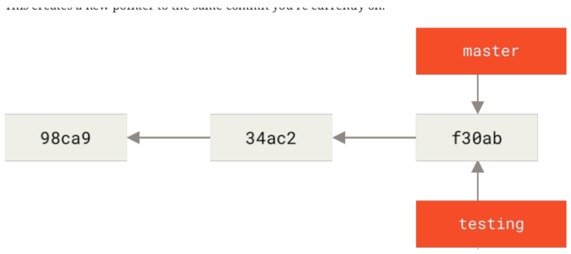


Figure 12. Two branches pointing into the same series of commits

Before creating a branch>>>

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git log1g
* e9cafd7a8b6c6a21c9d103ab0260249c28ef8f30 (HEAD -> reflogge) made to see git diff staged later
                #the above means, the HEAD is in reflogge branch

*   ba957091b0814f4a6b25f624467ce3bd32194552 (master, feat1) remove content from hello world to resolve conflict
|\
| * 4b4f48b0eb602d23ebbaf4c71b9f4f85f5715c81 added twinkle in hello world file in feat1 branch
```

After creating a branch named as feature >>>

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git log1g
* e9cafd7a8b6c6a21c9d103ab0260249c28ef8f30 (HEAD -> reflogge, feature) made to see git diff staged later
            # above means the HEAD is at both reflogge and feature branch
            # since we are in reflogge branch, it appears first. if we switch to feature branch
            #... that appears first.
*   ba957091b0814f4a6b25f624467ce3bd32194552 (master, feat1) remove content from hello world to resolve conflict
|\
| * 4b4f48b0eb602d23ebbaf4c71b9f4f85f5715c81 added twinkle in hello world file in feat1 branch
```

What is the command to switch to an existing branch (ex. “feature”) from current branch (old way)?

`git checkout feature`

What is the command to show log of a particular branch (ex. feature) other than current branch?

`git log feature`

Here we can use `git log --oneline --graph feature`

What is the command to show log of all branches?

`git log --all`

What is the command to create a new branch (ex. "feature") and switch to it at the same time? (old way)

`git checkout -b feature`

What is the command to switch to a branch (ex. leaf) from an existing branch? (NEW way)

`git switch leaf`

What is the command to create a new branch (ex. "feature") and switch to it at the same time? (NEW way)

`git switch -c feature`

The -c flag stands for create, you can also use the full flag: --create.

What is the command to rename a branch (old : "feature", new name "iss53")?

`git branch -m feature iss53`

but this change is only local for now. To let others see the corrected branch on the remote, we need push it

What is the command to let other see the renamed / corrected branch name (ex: "iss53"on the remote repo?

`git push --set-upstream origin iss53`

> 💡 *Notice that you're on the branch iss53 and it's available on the remote. However, the branch with the bad name is also still present there but you can delete it by executing the following command:* `git push origin --delete feature` *[in similar fashion as deleting a branch in remote]*

What is the command to delete a branch in the remote repo? (ex: "feature")

`git push origin remote --delete feature`

What is the command to delete a branch (ex. "hotfix")?

`git branch -d hotfix`

What is merge in git?

Merge in git is used to **integrate changes** from a one branch to another.

> 💡 This will create a new **merge commit** (unless it is fast forward merge)

What is the command to merge a branch (ex. hotfix) to master branch?

`git merge hotfix`

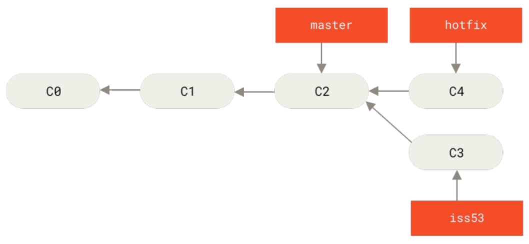
We need to first come to master branch and perform the above command.

▼ **What is a fast-forward merge? (NtU)**

This happens when a parent commit to the branch has no (TBU)……..

when you try to merge one commit with a commit that can be ***reached*** *by following the first commit's history, Git simplifies things by moving the pointer forward because there is no divergent work to merge together — this is called a "fast-forward."*

before:



after:

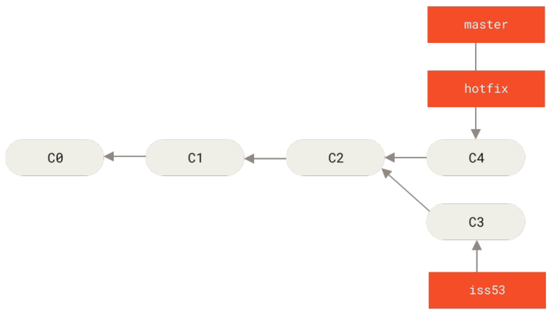


Figure 22. `master` *is fast-forwarded to* `hotfix`

▼ **What is a recursive merge? or when is a merge commit created? aka three-way merge!**

*If your development history in a branch has diverged from some older point. Because the commit on the branch you're on isn't a direct ancestor of the branch you're merging in, Git has to do some work. In this case, Git does a s**imple three-way merge**, using the two snapshots pointed to by the branch tips and the common ancestor of the two.*

Instead of just moving the branch pointer forward, Git creates a new snapshot that results from this three-way merge and automatically creates a new commit that points to it. This is referred to as a merge commit, and is special in that it has more than one parent.

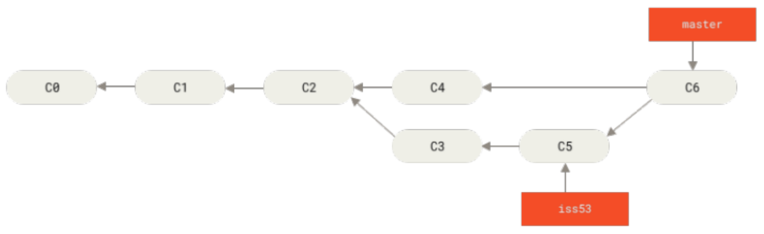
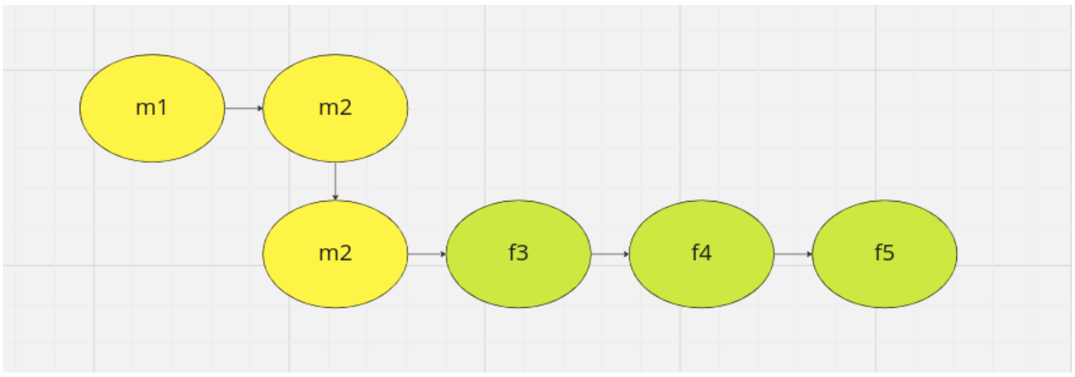


Figure 25. A merge commit

This happens when there is no merge conflict.

▼ My Merge Exercises

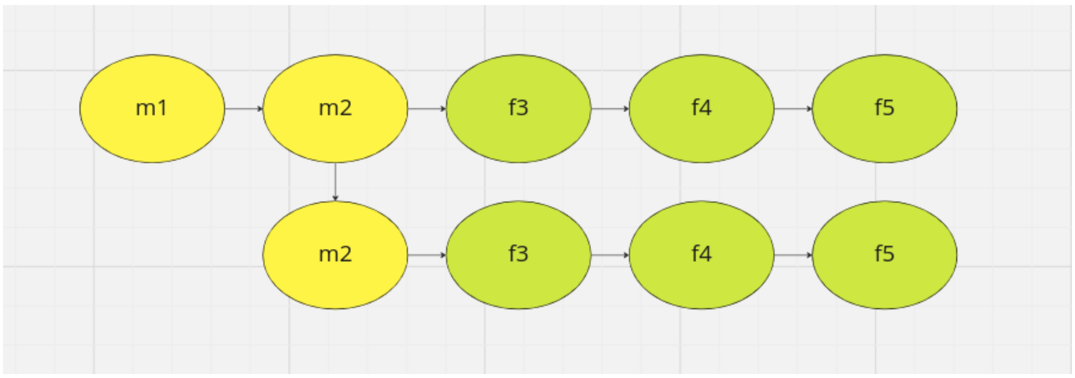
▼ Case1: Fast- Forward Merge



commits were created in this fashion before merging

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c0$ git log --oneline --graph --all
* c89fbfa (HEAD -> feature) f5
* 0dd5f75 f4
* d6a8ed8 f3
* ef4dbc6 (master) m2
* a41a445 m1
```

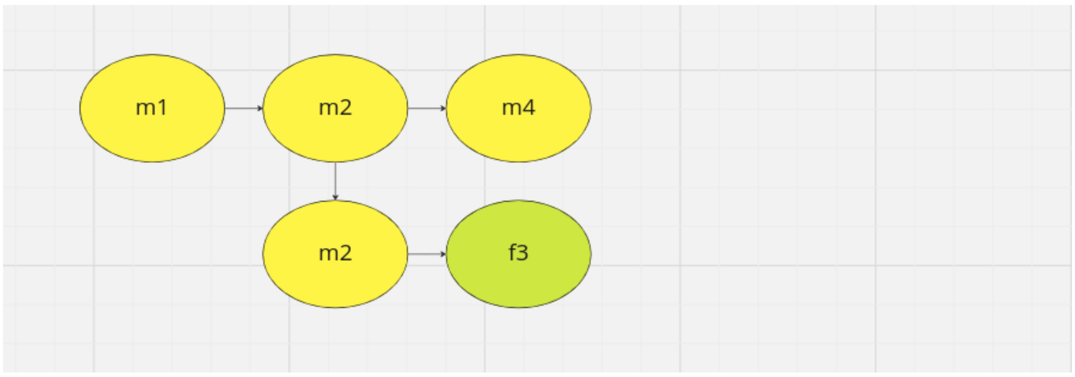
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c0$ git switch master
Switched to branch 'master'
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c0$ git merge feature
Updating ef4dbc6..c89fbfa
Fast-forward
 f.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 f.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c0$ git log --oneline --graph --all
* c89fbfa (HEAD -> master, feature) f5
* 0dd5f75 f4
* d6a8ed8 f3
* ef4dbc6 m2
* a41a445 m1
```



commits were created in this fashion after merging

💡 What I learnt?
Here, since there is no divergent changes in feature branch and the parent for both is same commit (m2) git copies and paste all the commits onto master branch

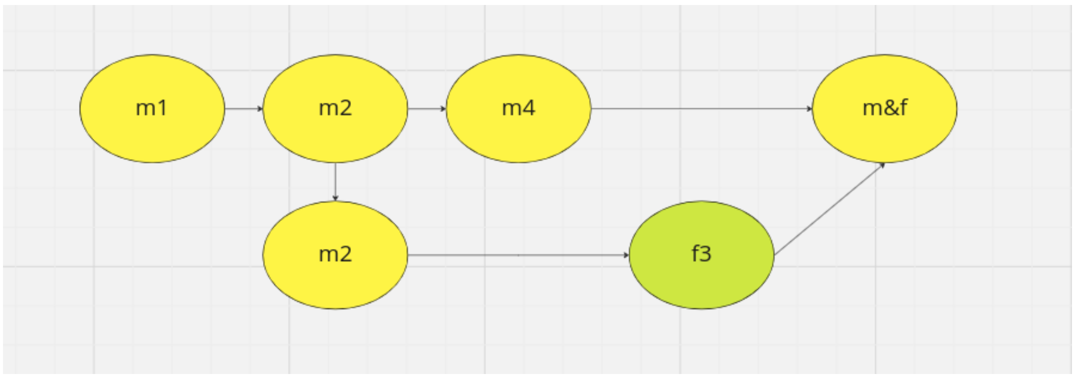
▼ Case2: Three-way Merge (both are at same level)



commits were created in this fashion before merging (to create divergent commit changes)

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c1$ git log --oneline --graph --all
* 2e93c7d (HEAD -> master) m4
| * 7f203e8 (feature) f3
|/
* 15b69d1 m2
* 9ea4d3c m1
```

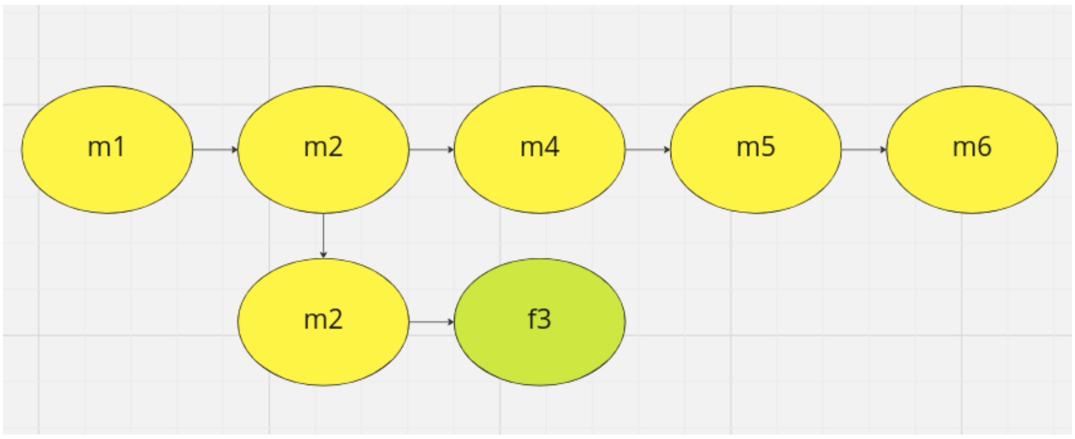
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c1$ git log --oneline --graph --all
*   28bf03b (HEAD -> master) merge master and feature
|\
| * 7f203e8 (feature) f3
* | 2e93c7d m4
|/
* 15b69d1 m2
* 9ea4d3c m1
```



commits were created in this fashion after merging >> this is also called as **three way merge**

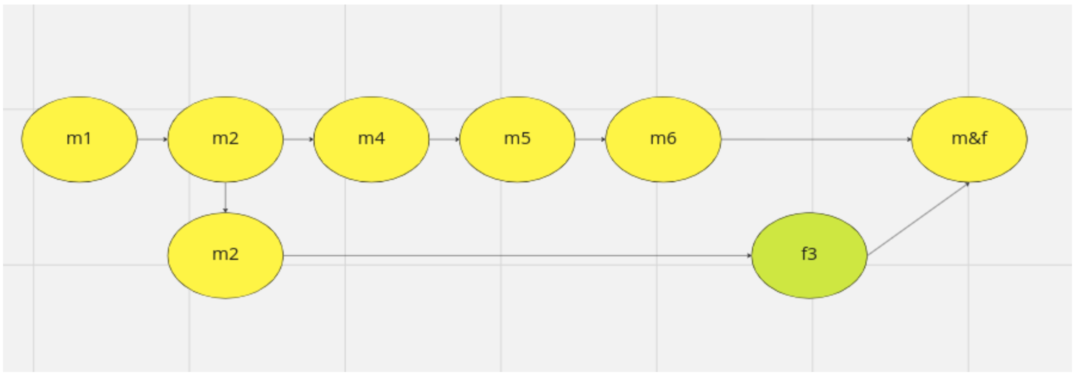
💡 What I learnt?
Git takes the f3 commit and pushes ahead of m4 in commit history and makes a merge commit as m&f on the master branch. (can see the same above)

▼ Case3: Three-way Merge (master’s HEAD is ahead of feature’s HEAD)



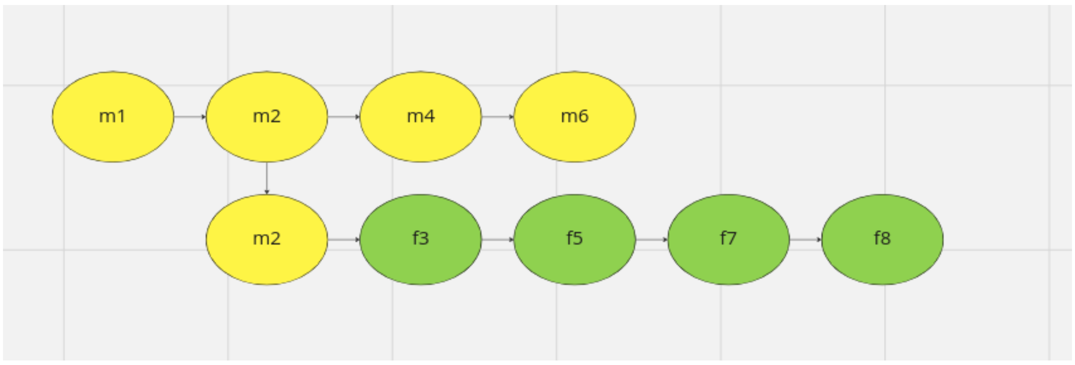
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c2$ git log --oneline --graph --all
* 24fd817 (HEAD -> master) m6
* 6e17e6a m5
* f05be00 m4
| * 54a0be1 (feature) f3
|/
* b22461f m2
* 30a7de9 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c2$ git log --oneline --graph --all
*   427788f (HEAD -> master)  m&f
|\
| * 54a0be1 (feature) f3
* | 24fd817 m6
* | 6e17e6a m5
* | f05be00 m4
|/
* b22461f m2
* 30a7de9 m1
```



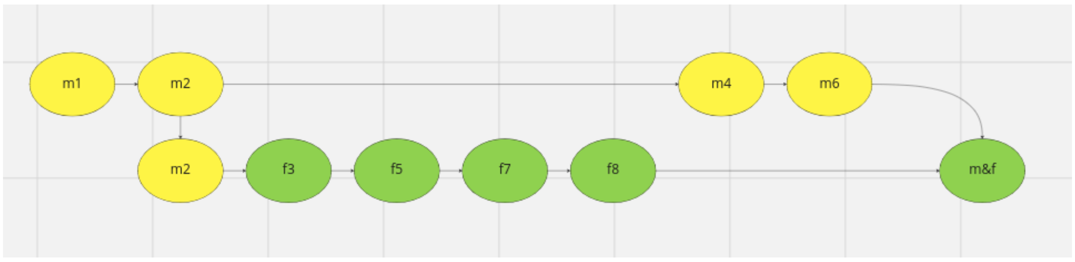
💡 What I learnt?
Here also f3 is pushed ahead of m6 in commit history and adding the merge commit as m&f on the master branch. (can see the same above)

▼ Case3(r): Three-way Merge (Feature’s HEAD is ahead of master’s HEAD)



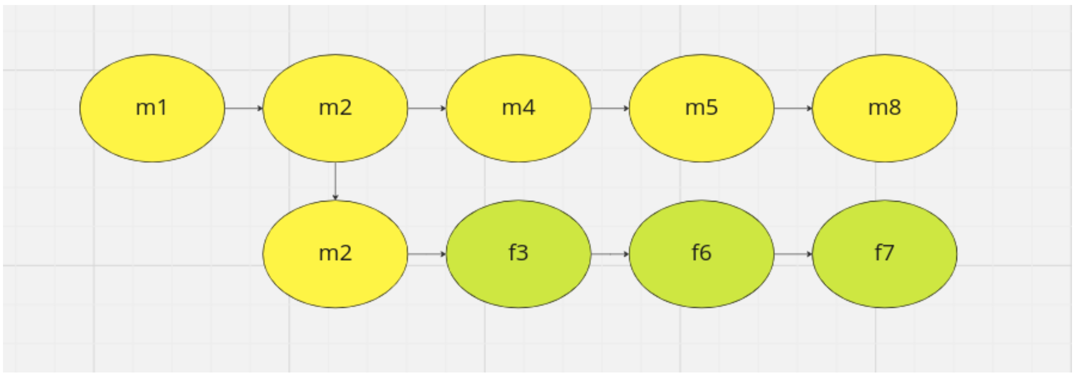
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c3(r)$ git branch
* feature
  master
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c3(r)$ git log --oneline --graph --all
* b16cc07 (HEAD -> feature) f8
* 85734d8 f7
* 161dfd4 f5
* 2bb164f f3
| * 992dc12 (master) m6
| * 2fbeb08 m4
|/
* 7beee23 m2
* e89df10 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c3(r)$ git merge master
Merge made by the 'ort' strategy.
 m.txt | 2 ++
 1 file changed, 2 insertions(+)
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c3(r)$ git log --oneline --graph --all
*   13c4430 (HEAD -> feature) m&f
|\
| * 992dc12 (master) m6
| * 2fbeb08 m4
* | b16cc07 f8
* | 85734d8 f7
* | 161dfd4 f5
* | 2bb164f f3
|/
* 7beee23 m2
* e89df10 m1
```



💡 What I learnt?
Here also m4 and m6 is pushed ahead of f8 in commit history and adding the merge commit as m&f on the feature branch. (can see the same above)

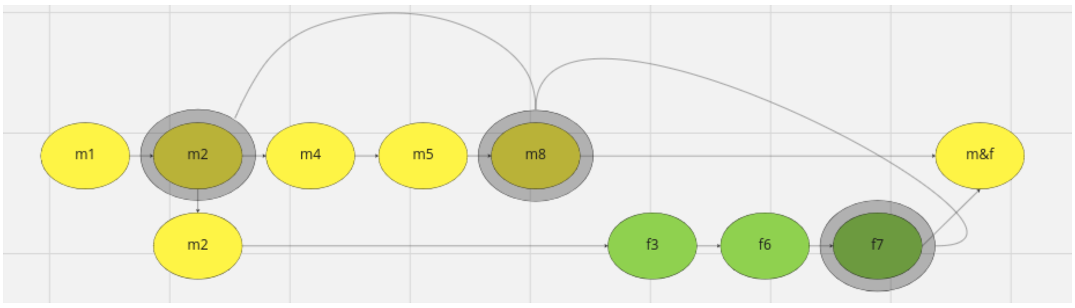
▼ Case4: Thre-way merge (both are equal but have long commits)



commits were created in this fashion before merging

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c4$ git log --oneline --graph --all
* 3a63bef (HEAD -> master) m8
* f0aa1ed m5
* 58d082e m4
| * deb4be4 (feature) f7
| * 3dcbf71 f6
| * a9ceb75 f3
|/
* c585691 m2
* ca251b9 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c4$ git branch
  feature
* master
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c4$ git merge feature
Merge made by the 'ort' strategy.
 f.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 f.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c4$ git branch
  feature
* master
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_merge_c4$ git log --oneline --graph --all
*   25552fa (HEAD -> master) m&f
|\
| * deb4be4 (feature) f7
| * 3dcbf71 f6
| * a9ceb75 f3
* | 3a63bef m8
* | f0aa1ed m5
* | 58d082e m4
|/
* c585691 m2
* ca251b9 m1
```



commits were created in this fashion after merging

💡 WiL?
I think, git pushes the commits in the feature branch ahead of all commits (even though created sequentially) in the master branch and combines all the commits present in features branch, looks at the changes between 3 commits (viz., parent commit, latest commit in master and latest commit in feature and creates a new “merge commit” that merges all the changes made in feature branch as one commit.

▼ How to resolve a merge conflict?

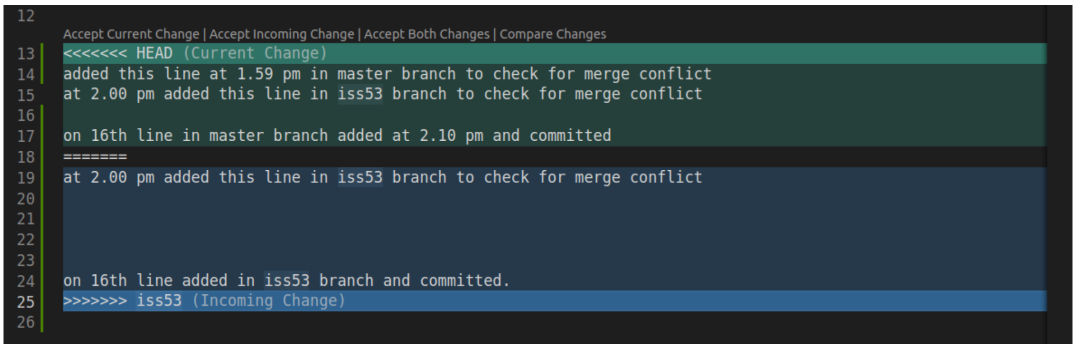
scenario>> *shows which file has merge conflict*

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git merge iss53
Auto-merging file01.txt
CONFLICT (content): Merge conflict in file01.txt
Automatic merge failed; fix conflicts and then commit the result.
```

git status>> *shows umerged paths*

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
  both modified:   file01.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
```

Git adds conflict resolution markers to the file


Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<<< HEAD (Current Change)
added this line at 1.59 pm in master branch to check for merge conflict
at 2.00 pm added this line in iss53 branch to check for merge conflict

on 16th line in master branch added at 2.10 pm and committed
=======
at 2.00 pm added this line in iss53 branch to check for merge conflict




on 16th line added in iss53 branch and committed.
>>>>>>> iss53 (Incoming Change)

```

options in git are present in the vscode.

once resolved, we need to **add** and **commit** in the master branch and it automatically merges.

git log after adding and committing

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_practice$ git log1g --all
*   cf9ba22c84d3358b6c8ecb329074002e5220580a (HEAD -> master) made changes in file01 to resolve the conflict
            ## here it created a new merge commit that has made changes to the file that maybe is different from bo
|\
| * f8a9978df4462662793d1f30cb115384b0adcd2d (iss53) added this line at 2.11 pm in iss53 branch to check for merge conf
* | e9c5ff647f976000d905654fdfbda4d5a1840a7b added this line at 2.10 pm in master branch to check for merge conflict
* | c6239cddcb0c141683d0f89038087c968938b56c made to resolve conflict
```

In which scenarios does a merge conflict can occur?

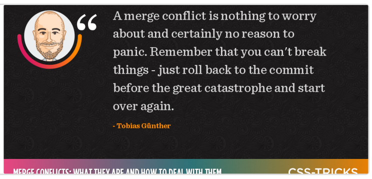
Followings cases:

1. when the exact **<u>same line of code</u>** was changed in two commits, on two different branches.
2. when a file has been deleted in one branch but edited in the other.

Merge Conflicts: What They Are and How to Deal with Them | CSS-Tricks

Merge conflicts... Nobody likes them. Some of us even fear them. But they are a fact of life when you're working with Git, especially when you're teaming up with other developers. In most cases, merge conflicts aren't as scary as you might think.

https://css-tricks.com/merge-conflicts-what-they-are-and-how-to-deal-with-them/



“A merge conflict is nothing to worry about and certainly no reason to panic. Remember that you can't break things - just roll back to the commit before the great catastrophe and start over again.

- Tobias Günther

Nice explanation of merge conflict in this link

What is the command to see the branches present in the local repository?

`git branch`

What is the command to see the differences between two branches (ex. master and feature)

`git diff master feature`

What is the command to see which branches have merged?

`git branch --merged`

💡 *Branches on this list without the * in front of them are generally fine to delete with git branch -d;*

What is the command to see which branches have not merged?

`git branch --no-merged`

What is Branching Workflows?

It is a form of best practices to follow while branching and merging in git.

Getting started with branching workflows, Git Flow and GitHub Flow

Become a Patreon and get source code access: https://www.patreon.com/nickchapsas Check out my courses: https://nickchapsas.com Hello everybody I'm Nick and in this video I am going to talk about the two most popular git branching workflows and those are Git Flow and GitHub flow.

https://www.youtube.com/watch?v=gW6dFpTMk8s&ab_channel=NickChapsas

Nicely explained what is Git flow and git flow here.

What is the command to fetch all changes from a remote (ex. “pablo”) and a branch (ex. “develop”)?

`git fetch pablo`

What is the command to pull all changes from a remote (ex. “pablo”) and a branch (ex. “develop”)?

`git pull pablo develop`

💡 *While the `git fetch` command will fetch all the changes on the server that you don't have yet, it will not modify your working directory at all. It will simply get the data for you and let you merge it yourself.*

However, there is a command called `git pull` which is essentially a `git fetch` immediately followed by a `git merge` in most cases.

If you have a tracking branch set up as demonstrated in the last section, either by explicitly setting it or by having it created for you by the clone or checkout commands, git pull will look up what server and branch your current branch is tracking, fetch from that server and then try to merge in that remote branch. Generally it's better to simply use the fetch and merge commands explicitly as the magic of git pull can often be confusing.

What is the command to delete a remote branch (ex. serverfix)?

`git push origin --delete serverfix`

💡 *Basically all this does is to remove the pointer from the server. The Git server will generally keep the data there for a while until a garbage collection runs, so if it was accidentally deleted, it's often easy to recover.*

▼ **What is literal meaning of rebase?**

rebase >> re - base

What is re? >> again

re - apply : apply again

re - do : do again

re - decorate : decorate again

What is base?

lowest position

base of a tree / base of building etc.,

💡 Rebase : to change the base again.
in git, i think the base is the parent commit. so when we rebase it, changes the base of a branch

What is rebase in git?

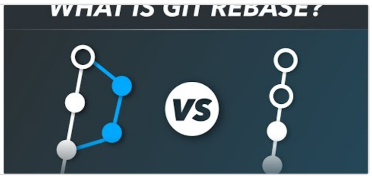
In Git, it is ***the process of combining _ a <u>sequence of commits</u> from one branch to a <u>new base commit</u> on another branch***

💡 rebasing ***<u>re-writes</u>*** the project history by creating brand new commits for each commit in the original branch. i.e it results in brand new commits.

Learn Git Rebase in 6 minutes // explained with live animations!

In this git rebase tutorial, I'll show you a different way of approaching git branching/merging with rebase. My team at Amazon adopted the workflow you'll see in the video and we love it.

https://www.youtube.com/watch?v=f1wnYdLEpgI&t=2s&ab_channel=TheModernCoder



Why do we use rebase in git?

The major benefit of rebasing is that you get a much **<u>cleaner project history.</u>**

First, it **eliminates the unnecessary merge commits** required by `git merge`

Second, it maintains a perfect **linear project history** . This makes it easier to navigate your project with commands like `git log`

What is the downside to rebasing in git?

There are two trade-offs for this : **safety and traceability**.

If you don't follow the Golden Rule of Rebasing, re-writing project history can be potentially catastrophic for your collaboration workflow. And, less importantly, rebasing loses the context provided by a merge commit—you can't see when upstream changes were incorporated into the feature.

💡 The golden rule of `git rebase` is to never use it on *public* branches.

Merging vs. Rebasing | Atlassian Git Tutorial

The git rebase command has a reputation for being magical Git voodoo that beginners should stay away from, but it can actually make life much easier for a development team when used with care. In this article, we'll compare git rebase with the related git merge command and identify all of the potential opportunities to incorporate rebasing into the typical Git workflow.

https://www.atlassian.com/git/tutorials/merging-vs-rebasing#the-golden-rule-of-rebasing

For more details ^.

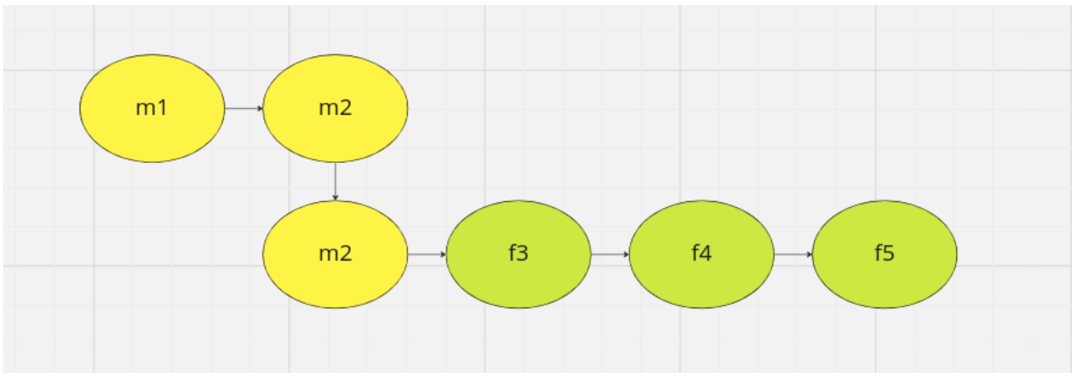
What is the command to rebase "feature" branch onto the master branch?

`git rebase feature`

This command is performed after checkout to master branch

▼ My Rebase Exercises

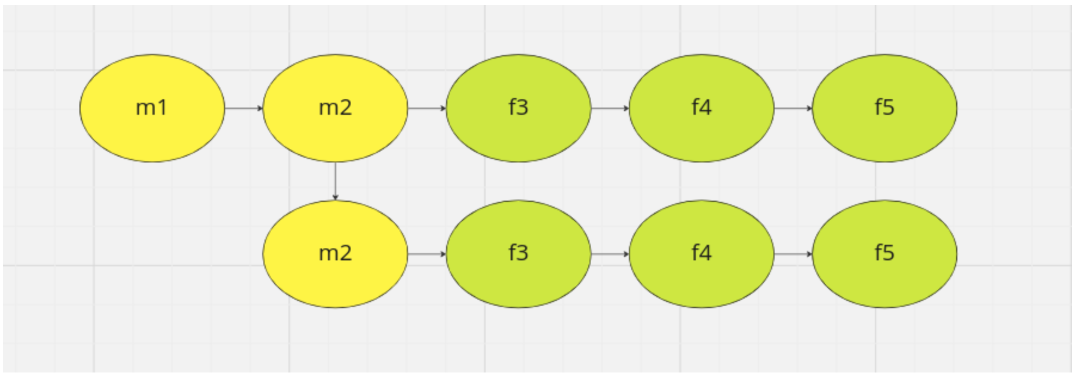
▼ Case1: feature is ahead and master has no additional commits since branching



commits were created in this fashion before rebasing

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c1$ git log --oneline --graph --all
* ae85066 (feature) f5
* b03f9c2 f4
* e420bd3 f3
* bf4cf0f (HEAD -> master) m2
* e829e3e m1
```

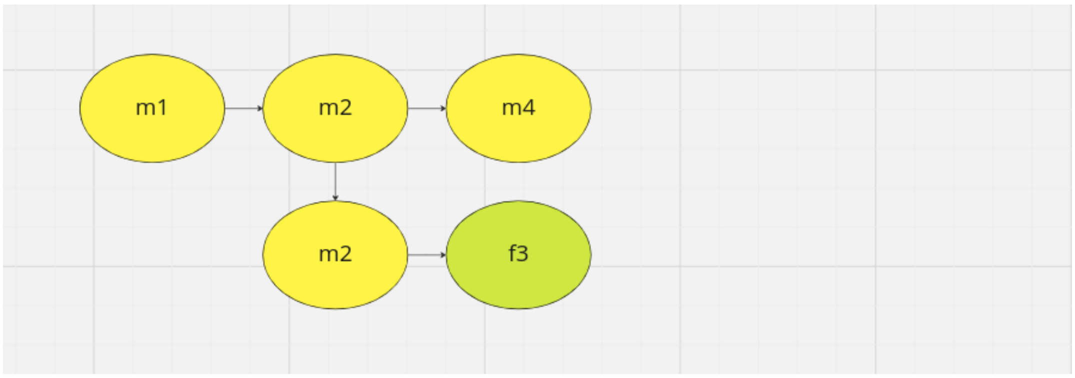
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c1$ git log --oneline --graph --all
* ae85066 (HEAD -> master, feature) f5
* b03f9c2 f4
* e420bd3 f3
* bf4cf0f m2
* e829e3e m1
```



commits were created in this fashion after rebasing

💡 WiL?
Here, it is similar to fast forward merge, as the parent commit is same and no divergence from master branch.

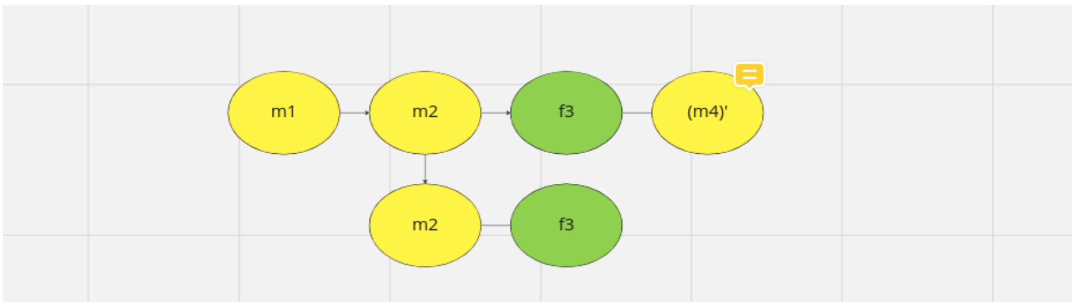
▼ Case2: (both are at same level)



commits were created in this fashion before rebasing (to create divergent commit changes)

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git log --oneline --graph --all
* 7779a61 (HEAD -> master) m4
| * bb4bebe (feature) f3
|/
* 52b2319 m2
* fe9ae21 m1
```

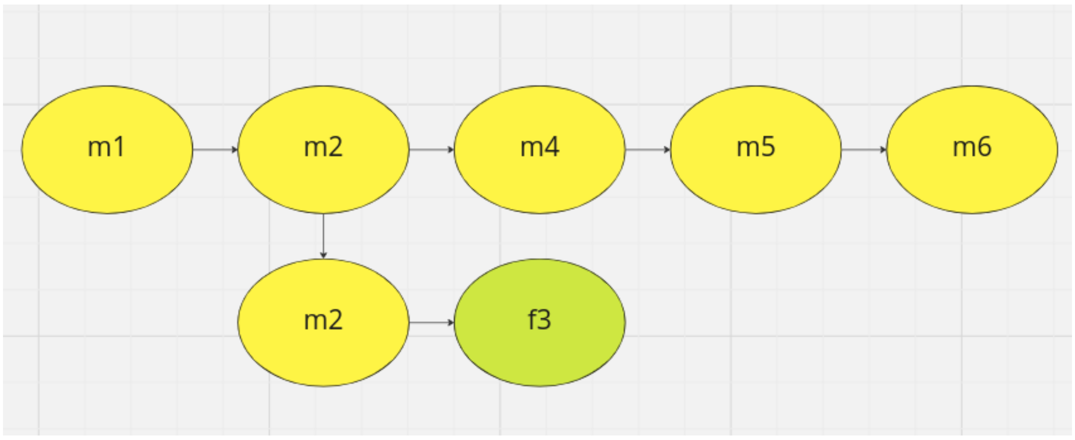
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git switch master
Already on 'master'

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git rebase feature
Successfully rebased and updated refs/heads/master.

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git log --oneline --graph --all
* 250db28 (HEAD -> master) m4   # commit hash has been changed to m4
* bb4bebe (feature) f3
* 52b2319 m2
* fe9ae21 m1

________________________

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git switch feature
Switched to branch 'feature'

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c2$ git log --oneline --graph --all
* 250db28 (master) m4
* bb4bebe (HEAD -> feature) f3  # this shows that branch is still present and the HEAD is behind master HEAD
* 52b2319 m2
* fe9ae21 m1
```



commits were created in this fashion after rebasing. here, the f3 commit is rebased on m2 on master branch (since it was created before m4) and m4 is pushed ahead of f3 (as it was created later)

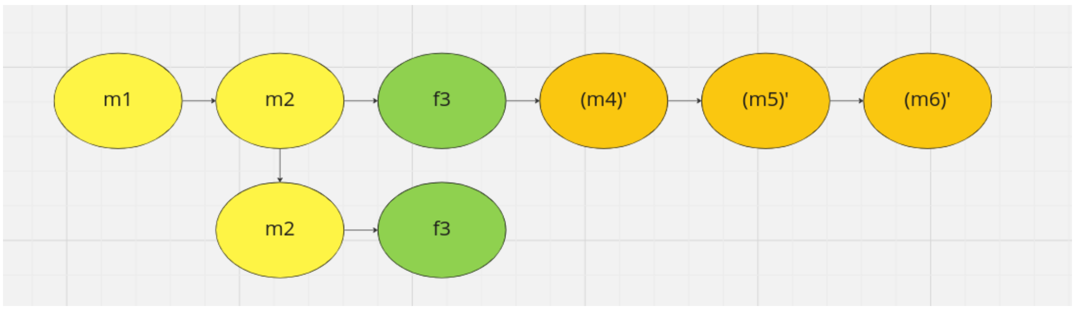
💡 What I learnt?
When we rebase with divergent changes, git rebases chronologically. here f3 was created prior to m4, hence f3 was rebased onto m2 and m4 was pushed and placed on top of m4 (by creating a new commit hash for it)

▼ Case3: master's HEAD is ahead of feature's



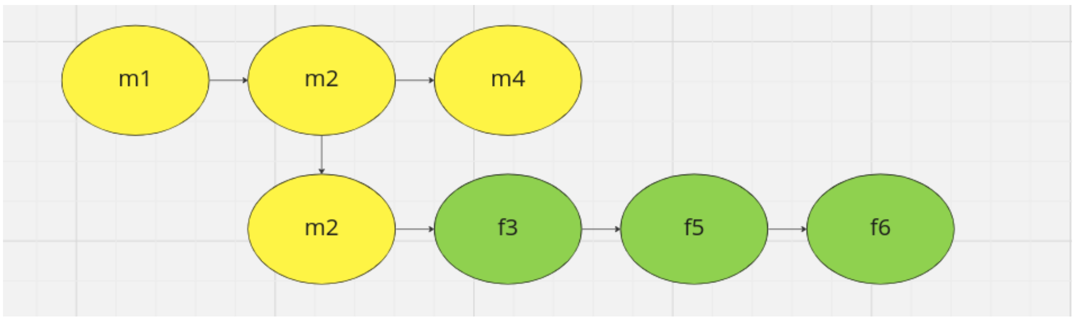
```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3$ git log --oneline --graph --all
* 0e32a54 (HEAD -> master) m6
* c9bc501 m5
* 94c02ab m4
| * 95d0052 (feature) f3
|/
* bfedb51 m2
* a2cbbd7 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3$ git rebase feature
Successfully rebased and updated refs/heads/master.

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3$ git log --oneline --graph --all
* edfc3de (HEAD -> master) m6
* 534dda4 m5
* 7101598 m4  # all the above commit hashes from m4 to m6 has been changed
* 95d0052 (feature) f3  # this commit hash has not been changed
* bfedb51 m2
* a2cbbd7 m1

_________________________

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3$ git checkout feature
Switched to branch 'feature'
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3$ git log --oneline --graph --all
* edfc3de (master) m6
* 534dda4 m5
* 7101598 m4
* 95d0052 (HEAD -> feature) f3  # here the HEAD still present
* bfedb51 m2
* a2cbbd7 m1
```



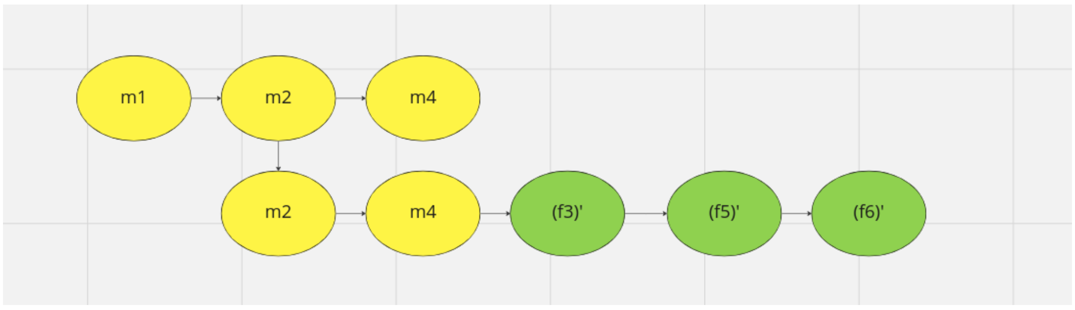
💡 What I learnt?
It is same as in case2, f3 is rebased on the parent commit as it was made before m4. I also learnt that all the above commits' hashes has been changed. as if it has recreated history of snapshots (appears as if f3 was part of master branch and m4, m5 and m6 were snapshots made after).

▼ Case3(r): Feature's HEAD is ahead of master's

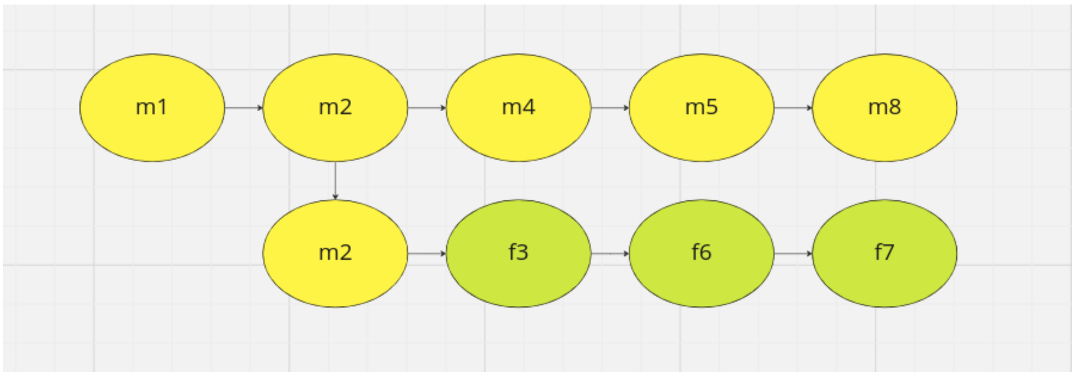


```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3(r)$ git log --oneline --graph --all
* d947291 (HEAD -> feature) f6
* 9b092f5 f5
* bb2e208 f3
| * eae5140 (master) m4
|/
* e639cc5 m2
* 1509f50 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3(r)$ git status
On branch feature
nothing to commit, working tree clean

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c3(r)$ git log --oneline --graph --all
* 20b27a2 (HEAD -> feature) f6
* c9a7351 f5
* 81ecffc f3  #here, the commit hashes for all the above has been changed
* eae5140 (master) m4  # the commit is same as it was in master
* e639cc5 m2
* 1509f50 m1
```



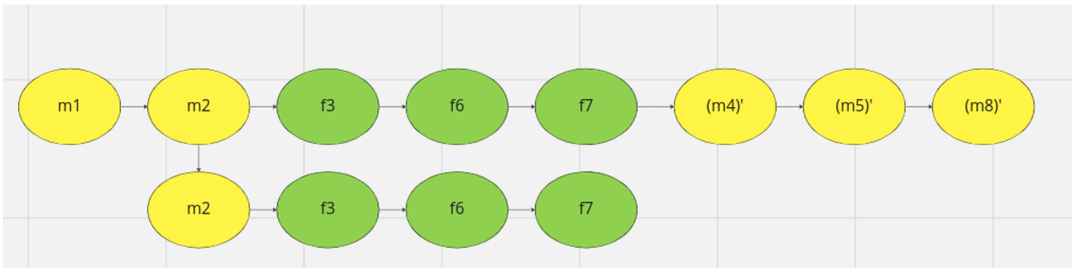
▼ Case4: both are equal but have long commits and are created in criss cross fashion



commits were created in this fashion before rebasing

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c4$ git log --oneline --graph --all
* bf4b540 (HEAD -> master) m8
* 3cdeb02 m5
* b01dd61 m4
| * 8592055 (feature) f7
| * 4a64e14 f6
| * 5b3da22 f3
|/
* 8e53ba4 m2
* 812ef36 m1
```

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c4$ git branch
  feature
* master

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c4$ git rebase feature
Successfully rebased and updated refs/heads/master.

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git_rebase_c4$ git log --oneline --graph --all
* 1da7352 (HEAD -> master) m8
* 283d0c8 m5
* 676907e m4  # m4, m5, m6 are pushed ahead of f7 and changed its hashes
* 8592055 (feature) f7
* 4a64e14 f6
* 5b3da22 f3  # f3, f6, f7 are rewriiten on main branch on top of m2 because it is the base
* 8e53ba4 m2
* 812ef36 m1
```



💡 WiL?
So it is clear that, when we rebase a feature branch on a master branch,
git will check for **common parent commit** for feature branch and master branch
git will rebase all the commits of feature branch on top of that same **common parent commit**
it will push all the commits on master branch ahead of the commits combined from feature branch and re-snapshot / re-commit them.

▼ 3. Git on the server

What is gitweb?

Gitweb provides a web interface to Git repositories. Its features include: Viewing multiple Git repositories with common root. Browsing every revision of the repository. Viewing the contents of files in the repository at any revision.

What is gitlab?

GitLab Tutorial For Beginners | What Is GitLab And How To Use It? | GitLab Tutorial | Simplilearn

https://www.youtube.com/watch?v=mX7lKoabmDw&ab_channel=Simplilearn

What is the difference between github & gitlab?

Same thing but each offers different functionalities.

Git lab **is a better tool for Devops approach**

GitHub vs GitLab | Difference between GitHub and GitLab | Which one is Best

In this video, we will see some key differences between GitHub and GitLab. #GitHub #GitLab #Git #GitHubvsGitLab If you have any questions or doubts you can ask in the comment section below. Do like and share this video if you found this video helpful.

https://www.youtube.com/watch?v=IFy7avS0ZxU&ab_channel=S3CloudHub

▼ 4. Github

What is github?

GitHub is a code **hosting platform** for **collaboration** and **version control.**

What is forking?

In software engineering, a project fork happens when developers **take a copy of source code** from one software package and start independent development on it, **creating a distinct and separate piece of software.**

What is forking in github?

A fork is done on github & it is used to ***copy another user's repository onto our account's repositories***
Forking a repository allows you to freely experiment with changes **without affecting the original project.**

> 💡 If you want to contribute to an existing project to which you **don't have push** access, you can "fork" the project.

The GitHub Flow

GitHub is designed around a particular collaboration workflow, **centered on Pull Requests**.

This flow works whether you're collaborating with a tightly-knit team in a single shared repository, or a globally-distributed company or network of strangers contributing to a project through dozens of forks.

It is centered on the **Topic Branches workflow** covered in Git Branching. Here's how it generally works:

1. Fork the project.
2. Create a topic branch from master.
3. Make some commits to improve the project.
4. Push this branch to your GitHub project.
5. Open a Pull Request on GitHub.
6. Discuss, and optionally continue committing.
7. The project owner merges or closes the Pull Request.
8. Sync the updated master back to your fork.

> 💡 ***Not Only Forks***
> It's important to note that you can also open a Pull Request between two branches in the same repository. If you're working on a feature with someone and you both have write access to the project, you can push a topic branch to the repository and open a Pull Request on it to the master branch of that same project to initiate the code review and discussion process. No forking necessary.

▼ 5. Git tools

what is git reflog?

It is a log of where the **HEAD and branch references** have been for the last few months.

It is like a temporary history

What is the command to see log of HEADS and Branch references?

`git reflog`

What is the literal meaning of stashing?

store (something) safely in a hidden or secret place.

What is stashing in git?

git stash **stores** the **changes** made in the **working directory** so we can work on **something else**, and then come back and **retrieve those changes back.**

What states of files does git stash store?

Git stash stores both **modified files and staged files** and applies stash.

Git stash will not stash untracked files, files that are ignored

▼ Exercise

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git init
Initialized empty Git repository in /home/sanjay/devops/progit/git stash/.git/

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ echo "m1" >> m.txt && git add m.txt && git commit -m "m1"
[master (root-commit) ce233b2] m1
 1 file changed, 1 insertion(+)
 create mode 100644 m.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ echo "stash now" >> m.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ echo "stage and stash" >> s.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git add s.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file:   s.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  modified:   m.txt

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git stash
Saved working directory and index state WIP on master: ce233b2 m1

sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git status
On branch master
nothing to commit, working tree clean
```

What is the use-case example for for using git stash?

After making some changes in a file or a deletion change etc., when we try to change the branch - we will be restrained from switching branches.

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ echo "make changes again" >> c.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git branch
* feature
  master
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git switch master
error: Your local changes to the following files would be overwritten by checkout:
  c.txt
Please commit your changes or stash them before you switch branches.
Aborting
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$
```

What is the command to see which stashes we have stored?

`git stash list`

▼ My exercises

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stash$ git stash list
stash@{0}: WIP on master: e23eec8 f3
stash@{1}: WIP on feature: 39427e6 error solved to switch branches after modifying
stash@{2}: WIP on master: ce233b2 m1
```

Breakdown:

stash@{2}: This is the earliest stash. stash@{0}: is the latest stash. Hence, numbering of stash is in reverse order, latest gets zero, and is the previous stash is kept on pushing on the positive number line

WIP on master: ce233b2 m1 : this is short has of the latest commit.

What is the command to *reapply stashed changes or un-stash* the changes back? ex. for latest stash.

`git stash apply`

Should be careful while doing this. We need to keep our working directory clean to avoid any merge conflict that appears during `git stash apply`

What is the command to stash even untracked files in git?

`git stash -u`

What is the command to *reapply stashed changes or un-stash* the changes back? ex. for stash number 2 (meaning last third stash).

`git stash apply stash@{2}`

Does git delete the stashed changes on the stack after `git stash apply` ?

No, it stays on as a recorded stash in the stack, until its deleted.

There are two files in two states, One file was in staged state and another file was modified state. Both were stashed. After entering git stash apply, what will be the state of files? same or changed?

No, both will be back to modified file state

> 💡 In order to stash any files, at least one commit should be present and the file should be tracker i.e should have been in at-least in one snapshot / commit

What is the command to *reapply stashed changes or un-stash* the changes made back to exactly previous state? ex. for latest stash.

`git stash apply --index`

What is the command to see what changes are stashed? ex. stash no.2

`git stash show stash@{2}`

What is the command to stash few changes with a message (ex. “I will save now commit later”)?

`git stash save "I will save now commit later”`

▼ my terminal

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ echo "fff" >> m.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ git stash save "added ff in m.txt"
Saved working directory and index state On master: added ff in m.txt
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ git stash list
stash@{0}: On master: added ff in m.txt
stash@{1}: WIP on master: a01193f m6
```

What is the command to remove a stash after `git stash apply`? or to remove the stashed changes from the stack directly? ex. for latest stash

`git stash drop`

▼ My exercise

```
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ git stash list
stash@{0}: WIP on master: a01193f m6
stash@{1}: WIP on master: a01193f m6
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ git stash drop
Dropped refs/stash@{0} (cc51713d65eaefb466cc414bf0cc820fab78778c)
sanjay@sanjay-HP-Pavilion-Notebook:~/devops/progit/git stashc1$ git stash list
stash@{0}: WIP on master: a01193f m6
```

What is the command to remove a stash after `git stash apply`? or to remove the stashed changes from the stack directly? ex. for stash number 1

`git stash drop stash@{1}`

What is the command to apply the stash and then immediately drop it from the stash stack?

`git stash pop`

> 💡 *`git stash pop`* = *`git stash apply`* + *`git stash drop`*

What is the command to remove all un-tracked files from working tree?
it should also clean forcefully & recursively for directories & also delete files that maybe configured to .gitignore

`git clean -fdx`

What is the command to modify your last commit message? and open an editor with previous commit message?

`git commit --amend`

What is the command to modify your last commit message? without opening an editor and add the new message (ex. "I amend to change the commit")

`git commit --amend -m "I amend to change the commit"`

What is the command to rewrite the latest commit? i.e to change the actual content of the latest commit by adding, removing and modifying files and re-commit? with a message "I changed but I amended instead of making another commit :P"

`git commit --amend "I changed but I amended instead of making another commit :P"`

What is the command for getting the actual directory listing and SHA-1 checksums for each file in the HEAD snapshot?

`git cat-file -p HEAD`

> 💡 Complex commands, can be skipped

What is the command that shows you what your index currently looks like?

`git ls-files -s`

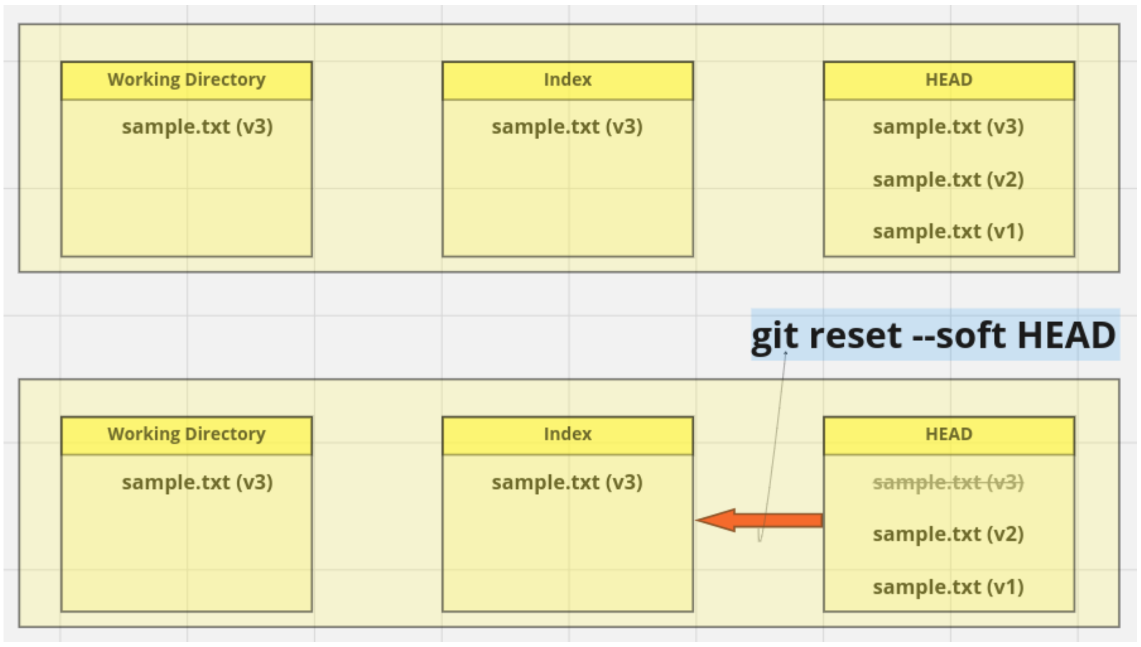
> 💡 Complex commands, can be skipped

What is the command to undo the latest commit and bring the files back to staging area?

`git reset --soft HEAD`

▼ **What is git reset soft actually do?**

When you run git commit, Git creates a new commit and moves the branch that HEAD points to up to it. When you reset back to HEAD~ (the parent of HEAD), you are moving the branch back to where it was, without changing the index or working directory. You could now update the index and run git commit again to accomplish what git commit --amend would have done

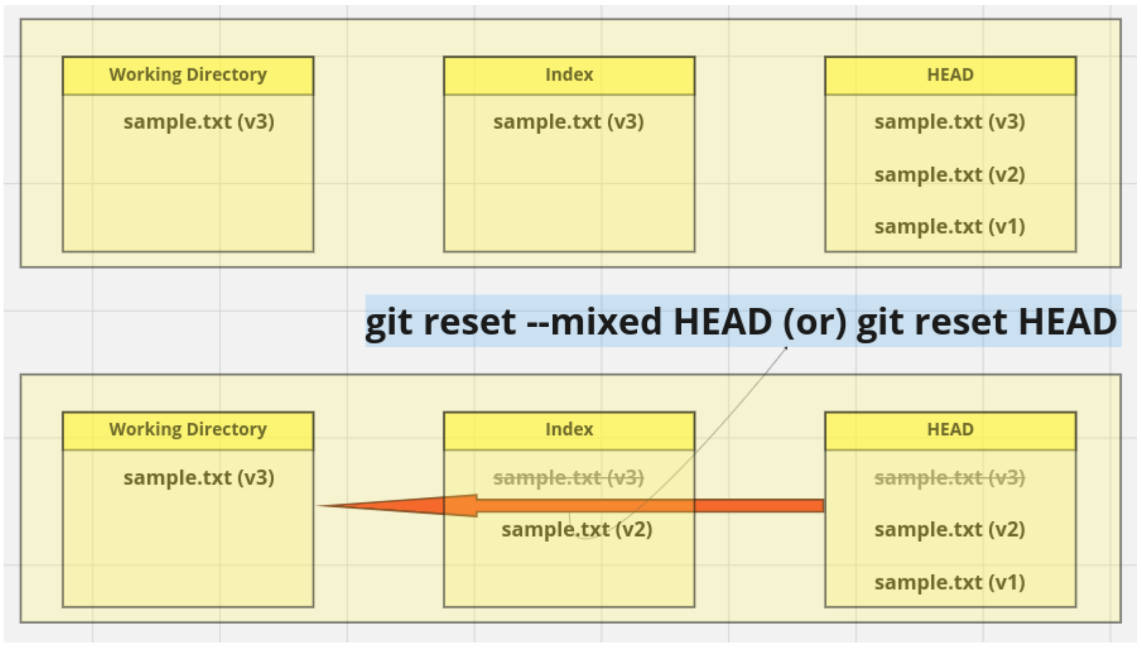


What is the command to undo the latest commit and un-stage the files and bring the files back to working directory?

`git reset --mixed HEAD`

▼ **What does git reset mixed actually do?**

it undid your last commit, but also unstaged everything. You rolled back to before you ran all your git add and git commit commands.



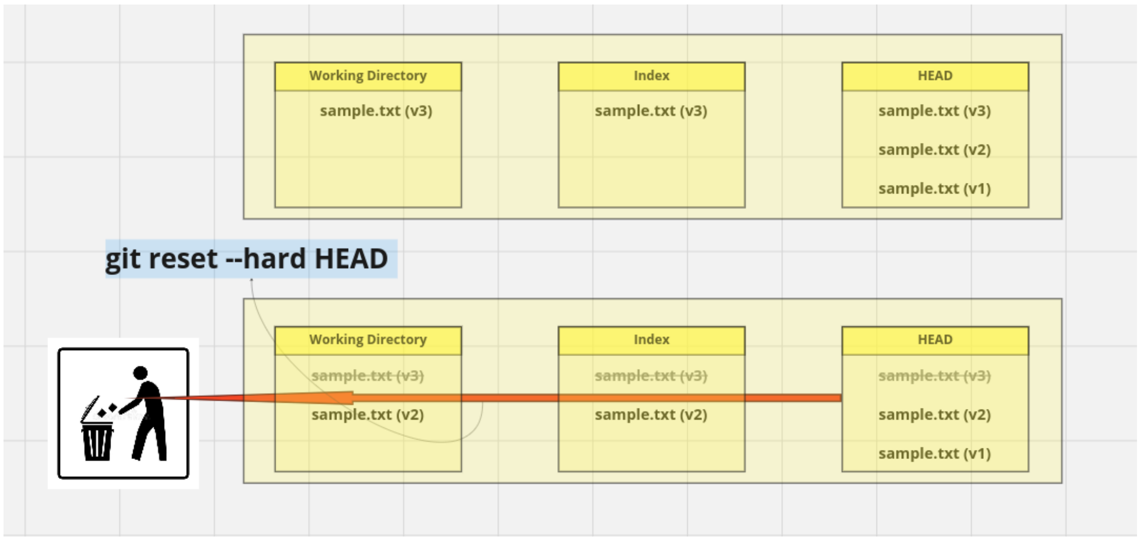
What is the command to undo the latest commit and un-stage the files and and undo the changes in the working directory and bring the files as it was it previous commit?

`git reset --hard HEAD`

▼ **What does git reset hard actually do?**

You undid your last commit, the git add and git commit commands, and all the work you did in your working directory.

It's important to note that this flag (--hard) is the only way to make the reset command dangerous, and one of the very few cases where Git will actually destroy data. Any other invocation of reset can be pretty easily undone, but the --hard option cannot, since it forcibly overwrites files in the working directory. In this particular case, we still have the v3 version of our file in a commit in our Git DB, and we could get it **back by looking at our reflog,** but if we had not committed it, Git still would have **overwritten the file and it would be unrecoverable.**



What is the command for the case - you have made changes to a file (ex. fix.txt) and decide all the changes are wrong and want to revert to the earlier version of the file as it was in the previous commit?

`git reset HEAD fix.txt`

▼ 6. Get the Git

▼ History of Git

In 2002, the Linux kernel project began using a proprietary DVCS called **BitKeeper**.
In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked.

This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool called as Git.

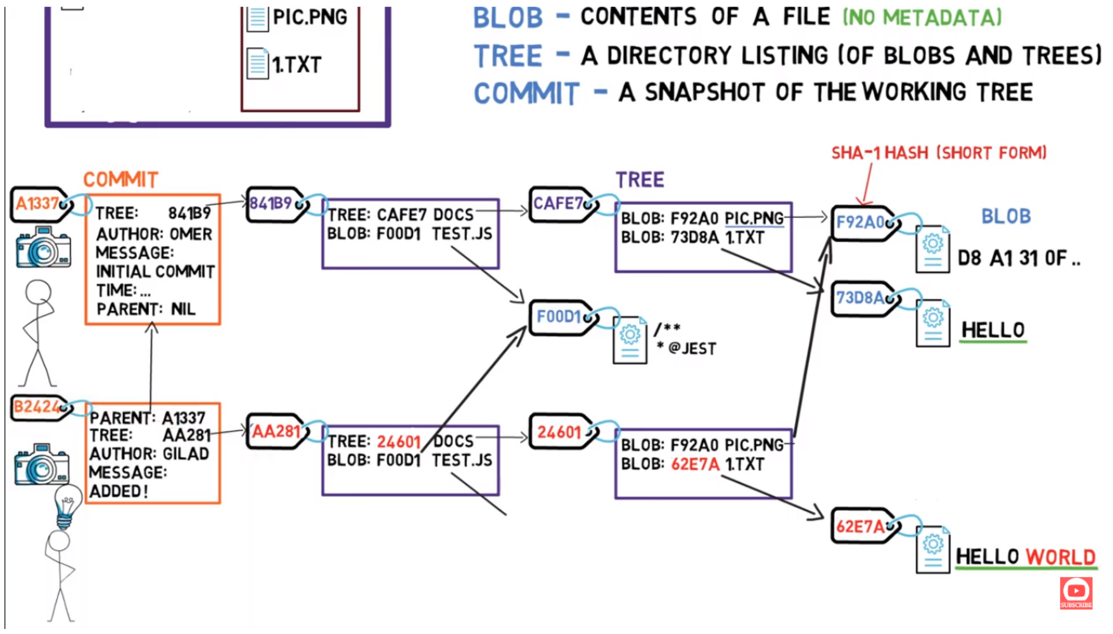
▼ What is Git?

Definition: Git is free and open source DVCS that is used to **track** and **manage** changes to software code.

How does Git think of its data?

Git thinks of its data more like a series of **snapshots** of a filesystem.

With Git, every time you commit (**saves the state of your project**), Git basically takes a picture of what all your files look like at that moment and stores a **reference** to that snapshot.



Git Internals - Git Objects

https://www.youtube.com/watch?v=MyvyqdQ3OjI&t=298s&ab_channel=Brief

▼ How does Git maintain its integrity wrt to data?

Everything in Git is **checksummed** before it is stored and is then **referred** to by that checksum.

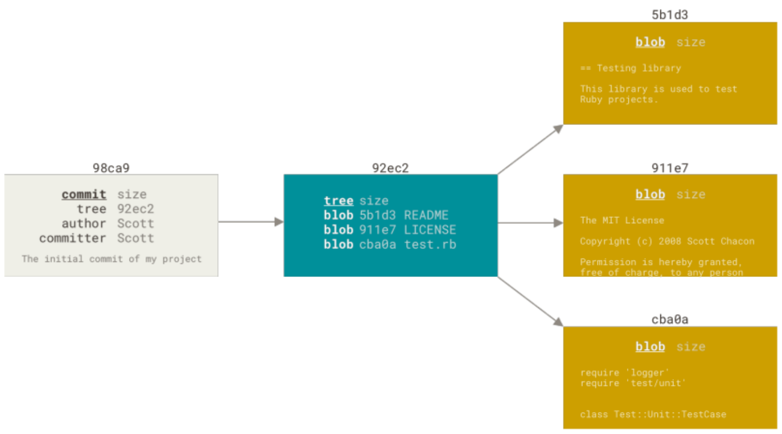


Figure 9. A commit and its tree

9. How GIT stores the data. Explore the git SHA1 hash objects in repository with cat-file command.

https://www.youtube.com/watch?v=Z1kFUUzxqgU&ab_channel=LeelaWebDev

What is a checksum?

A checksum is ***a simple number that's created by taking data and feeding it into a mathematical algorithm***

What mechanism does git use to checksum the data?

The mechanism that Git uses for this checksumming is called as **SHA-1 hash.**

What is SHA- 1 hash?

This is a **40-character string** composed of hexadecimal characters (i.e 16 characters : 0–9 and a–f) and calculated based on the contents of a file or directory structure in Git.

Ex: `24b9da6552252987aa493b52f8696cd6d3b00373`

How does SHA-1 work? we can try online

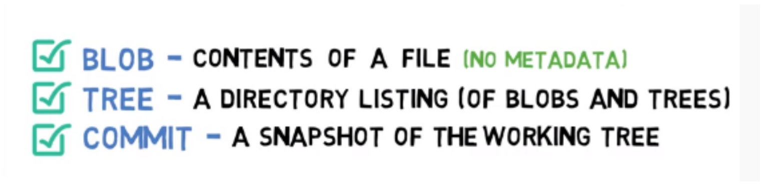
SHA1

SHA1 online hash function

https://emn178.github.io/online-tools/sha1.html

▼ What are git internals or git objects?

There are three : blob, tree, and commits


BLOB - CONTENTS OF A FILE (NO METADATA)
TREE - A DIRECTORY LISTING (OF BLOBS AND TREES)
COMMIT - A SNAPSHOT OF THE WORKING TREE

Git Internals - Git Objects

https://www.youtube.com/watch?v=MyvyqdQ3OjI&ab_channel=Brief

▼ What are the three states of a file in Git?

Git has three main states that your files can reside in:

modified,

staged, and

committed

What is **modified** state of a file?

When we edit a file that was in last commit, git treats that file as **modified**, because we have made changes i.e modified wrt to its previous state.

What is **staged** state of a file?

Staged means that you have **marked a modified file** to go into your next commit snapshot.

What is **committed** state of a file?

Committed means that the data is safely stored in your local repository.

▼ What are the sections of a git project?

the three main sections of a Git project:

the **working directory,**

the **staging area**, and

the **Git directory or local repository**

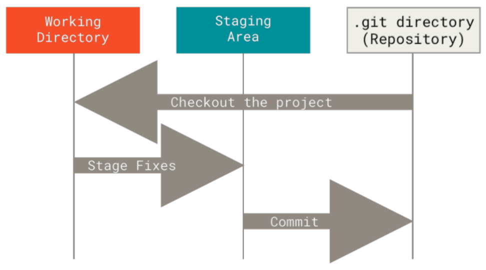


Figure 6. Working tree, staging area, and Git directory

What is Working directory section in git?

The Working Directory in Git is ***a directory (a folder)*** *on our* ***local file system*** *used to* ***make changes*** *to the files whose changes are tracked by git and is linked with git repository.*

What is Staging area section in git? (aka index - indicator)

The staging area is a **preview of files** that is **ready** to go in to our next commit.

What is Git directory or local Repository section in git?

The Git directory is where Git **stores the metadata and object database** for the project