Session 03-04

GIT

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.

Version Control System

- [ ] Software designed to record changes made to a file or set of files over time.
- [ ] Ability to revert back to a previous file version or project version.
- [ ] Compare changes made to files from one version to another version.

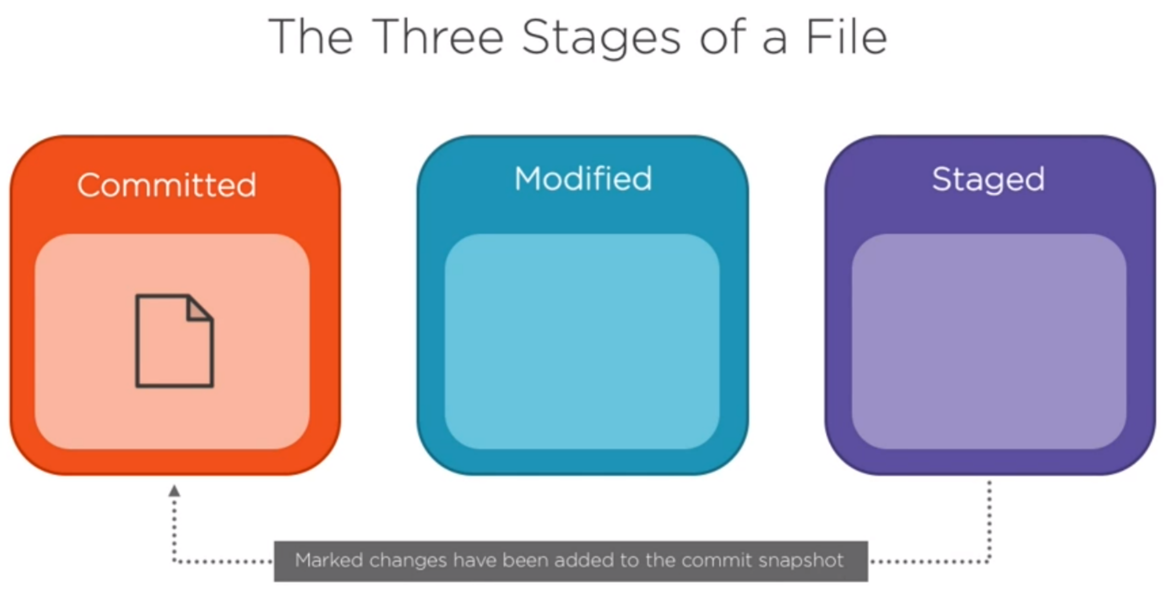
Centralized Version Control

- In CVS, a client need to get local copy of source from server, do the changes and commit those changes to centeral source on server.
- Working on branches in difficult in CVS. Developer often faces merge conflicts.
- CVS system do not provide offline access.
- CVS is slower as every command need to communicate with server.
- If CVS Server is down, developers cannot work.

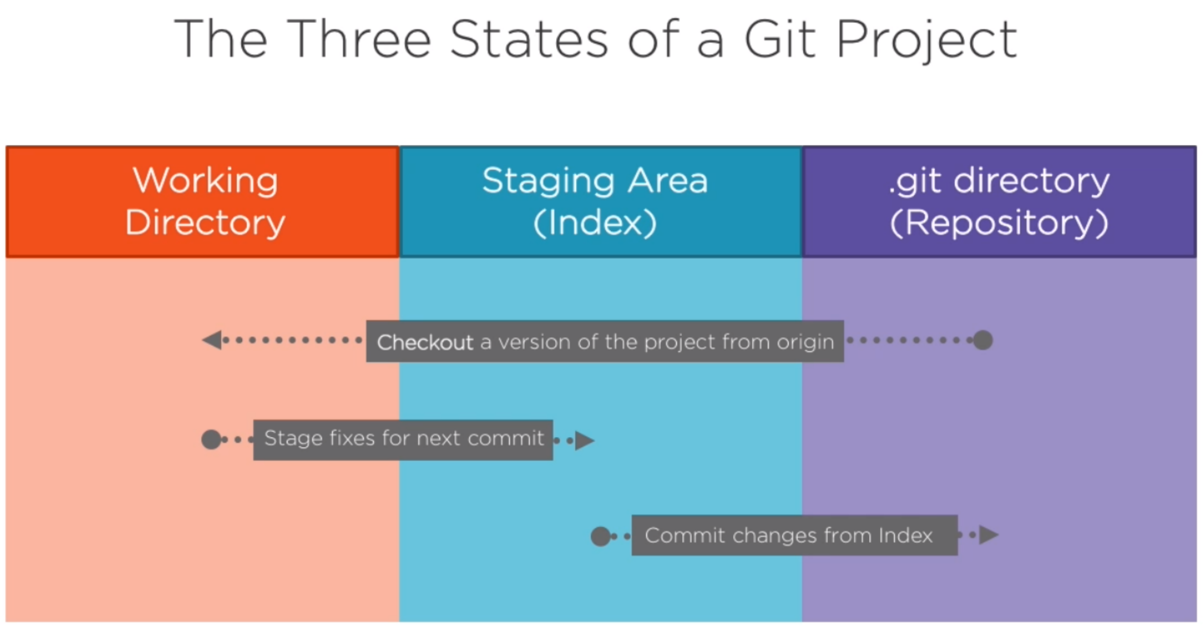
3 Stages of a File

Distributed Version Control

- In DVS, each client can have a local branch as well and have a complete history on it. Client need to push the changes to branch which will then be pushed to server repository.
- Working on branches in easier in DVS. Developer faces lesser conflicts.
- DVD systems are workable offline as a client copies the entire repository on their local machine.
- DVS is faster as mostly user deals with local copy without hitting server everytime.
- If DVS server is down, developer can work using their local copies.


The Three Stages of a File
Committed
Modified
Staged
Marked changes have been added to the commit snapshot

3 Stages of a git Project


The Three States of a Git Project
Working Directory
Staging Area (Index)
.git directory (Repository)
Checkout a version of the project from origin
Stage fixes for next commit
Commit changes from Index

Configuring GIT

```
git config --global user.name "Swagat Swain"
git config --global user.email "ssswagatss@gmail.com"
git config --list

git config user.name
```

Init

```
//Any project with git initialized, is called a git repo.
```

Git Remote Repo.

☐ Why do we need a remote repo.

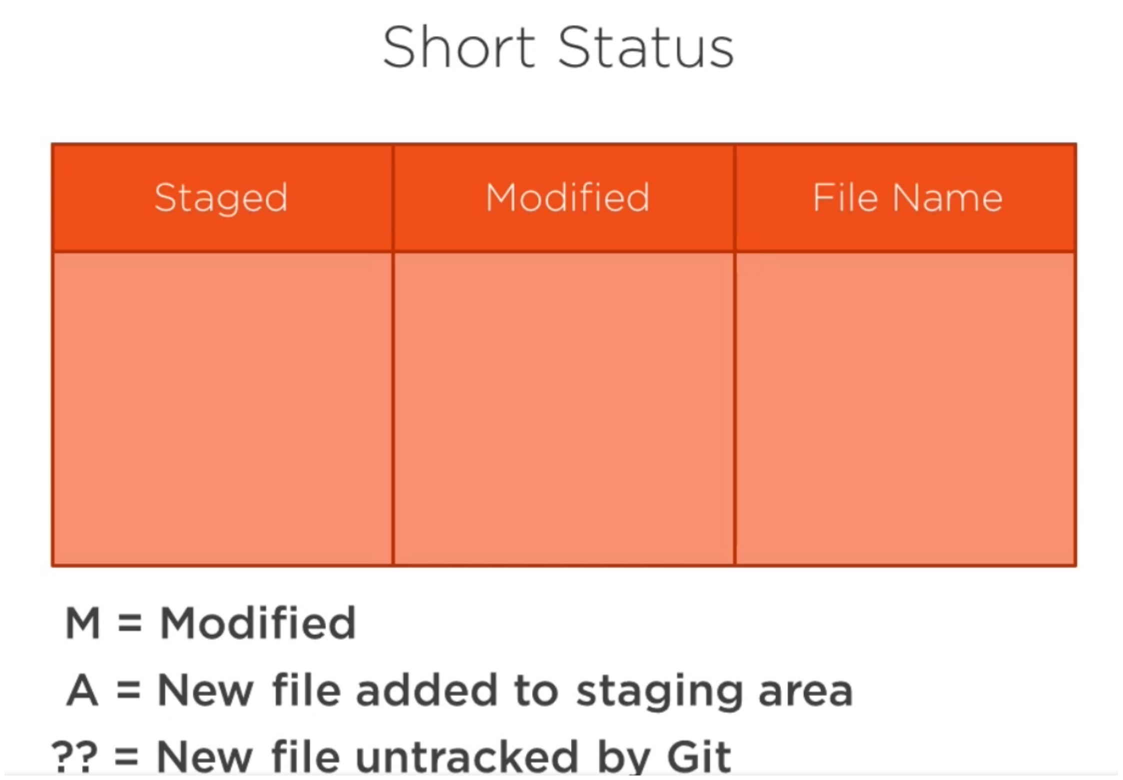
☐ Setting up a new project

☐ Adding files and setting up remote.

Basic Commands for Everyday use.

```
git status
  - Returns back the name of the current branch. (Branching is a core concept, we will cover that later.
  - Branch is up to date with origin/master
  - Working tree is clean.
```

☐ Show a file in both staged and modified



Short Status

Staged	Modified	File Name

M = Modified

A = New file added to staging area

?? = New file untracked by Git

GIT Diff

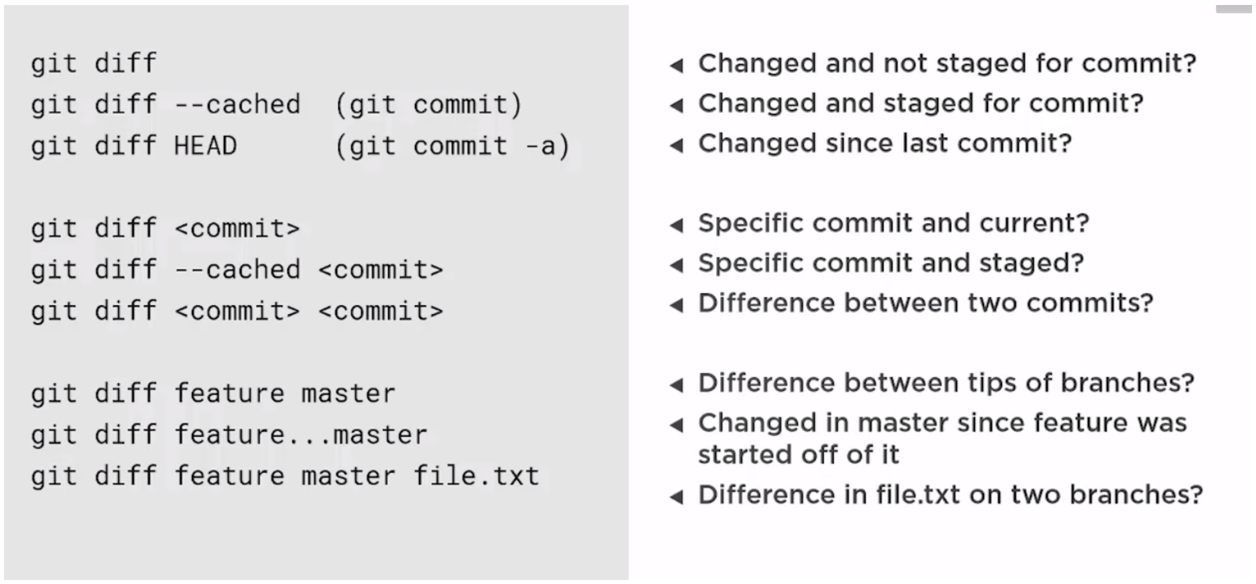
☐ What changes have I made that are ready to be committed.

- [ ]

```
git diff --staged
   git diff --staged --no-rename


   git diff
   This only shows the changes at  the workking directory.
```

- [ ] What changes have I made, that are not yet staged.

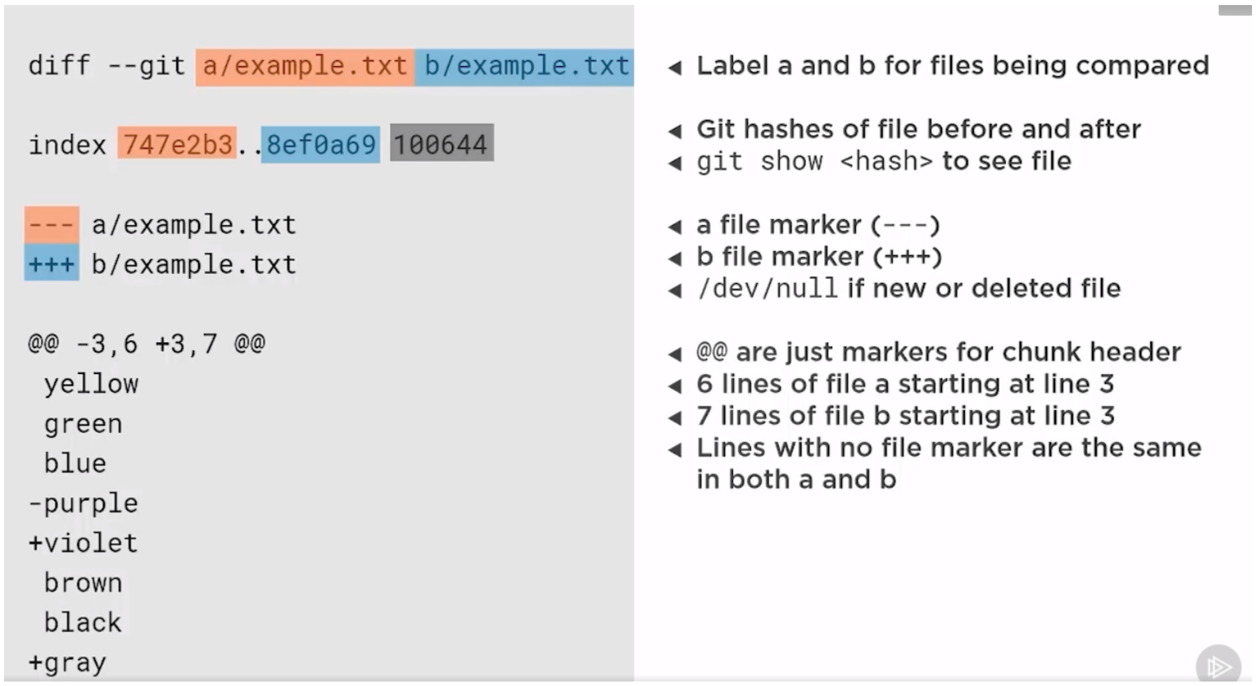


```
git diff
git diff --cached  (git commit)
git diff HEAD      (git commit -a)

git diff <commit>
git diff --cached <commit>
git diff <commit> <commit>

git diff feature master
git diff feature...master
git diff feature master file.txt
```

- ◀ Changed and not staged for commit?
- ◀ Changed and staged for commit?
- ◀ Changed since last commit?
- ◀ Specific commit and current?
- ◀ Specific commit and staged?
- ◀ Difference between two commits?
- ◀ Difference between tips of branches?
- ◀ Changed in master since feature was started off of it
- ◀ Difference in file.txt on two branches?



```
diff --git a/example.txt b/example.txt

index 747e2b3..8ef0a69 100644

--- a/example.txt
+++ b/example.txt

@@ -3,6 +3,7 @@
 yellow
 green
 blue
-purple
+violet
 brown
 black
+gray
```

- Label a and b for files being compared
- Git hashes of file before and after
- `git show <hash>` to see file
- a file marker (---)
- b file marker (+++)
- `/dev/null` if new or deleted file
- @@ are just markers for chunk header
- 6 lines of file a starting at line 3
- 7 lines of file b starting at line 3
- Lines with no file marker are the same in both a and b

SHA1

```
Check the SHA1
echo "Apple Pie" | git hash-object --stdin


Write the COntent to git
echo "Apple Pie 4" | git hash-object --stdin -w

Explain the sub directory

If you want to check the content
git cat-file 5484825f540d9c50cc8c82512eee6f46bda4e3f7 -t
git cat-file 5484825f540d9c50cc8c82512eee6f46bda4e3f7 -p
```

More fundamental commands

```
git log

--Shows the recent commits.
- Commit Hash
-Author and Date Time Stamp
-Commit Messange as well

git log -2
--Limits the log to 2 commits.

git log --oneline
--Shows just one line of the log


git log --stat
--Gives back a more detailed status of commit history
```

```
git log --patch
--Gives even more details with differences.
```

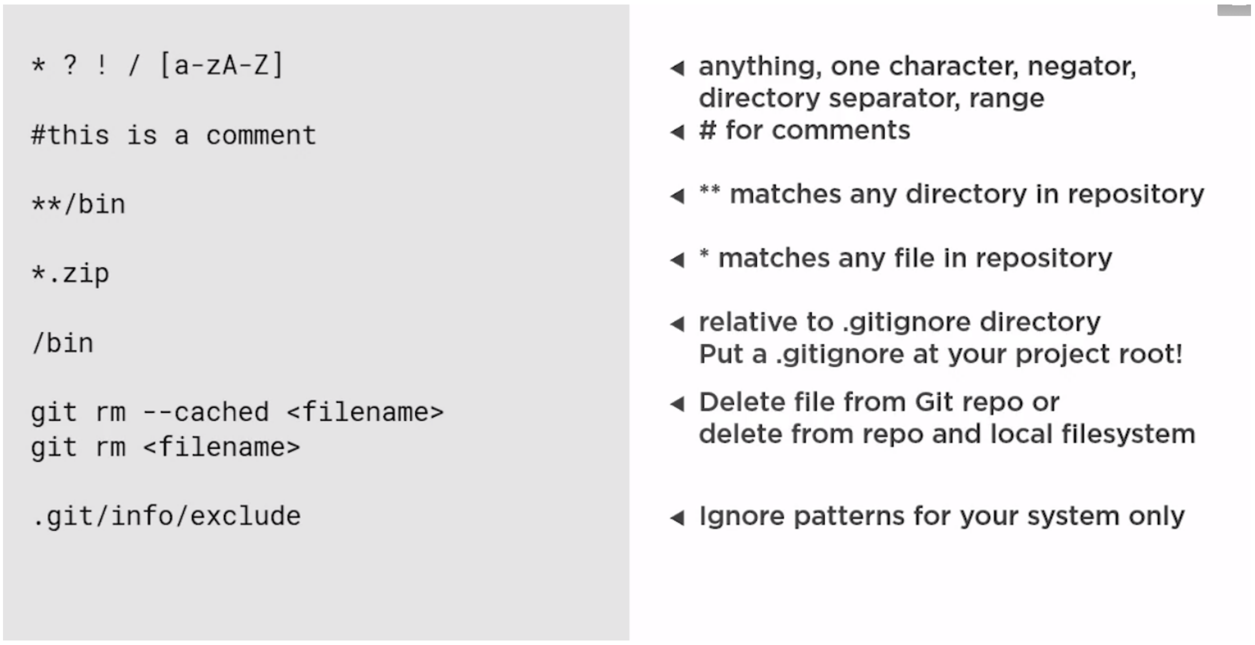
```
--Try renaming a file to show how git manages it.
--It will delete the file and create the new file.

-- To rename a file run the following command
git mv student.txt students2.txt

git rm school.txt
--Deletes a file from file system as well as git

in case you want to remove the file from git, and still want them to be a part of your file system
git rm --cached filename.txt

Then show an example of .gitignore file
```



```
* ? ! / [a-zA-Z]

#this is a comment

**/bin

*.zip

/bin

git rm --cached <filename>
git rm <filename>

.git/info/exclude
```

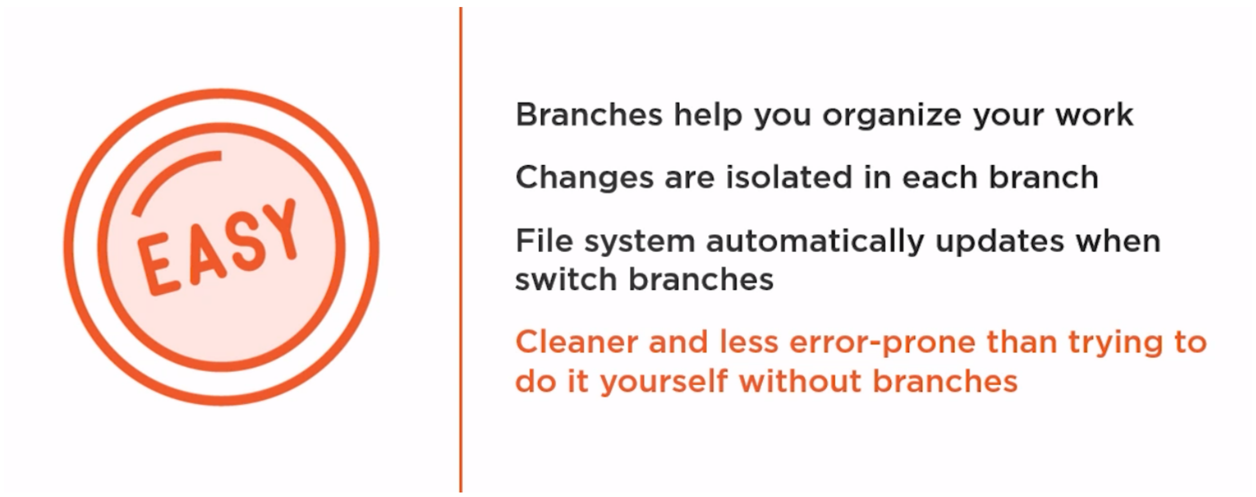
◄ anything, one character, negator, directory separator, range
◄ # for comments
◄ ** matches any directory in repository
◄ * matches any file in repository
◄ relative to .gitignore directory
Put a .gitignore at your project root!
◄ Delete file from Git repo or
delete from repo and local filesystem
◄ Ignore patterns for your system only

Branch

A Branch is a light weight pointer to a commit made in the project.

```
git branch

git checkout branch_name - Checks out.
```



Branches help you organize your work

Changes are isolated in each branch

File system automatically updates when switch branches

Cleaner and less error-prone than trying to do it yourself without branches

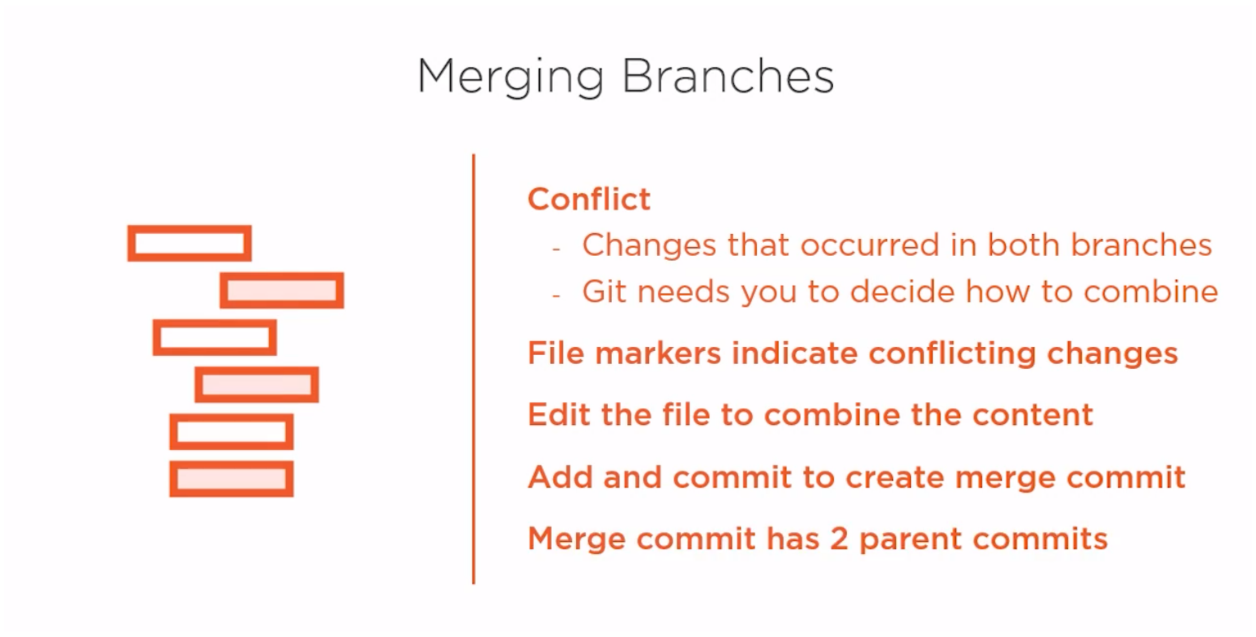
```
git branch -m quick_fix long_fix
git branch -d long_fix

Git wont let you delete a branch, if there are un committed changes. If you still want to force git, use -D instead of -d
```

Merging

```
git checkout target
git merge source

git diff branch1 branch2
```



Merging Branches

Conflict
- Changes that occurred in both branches
- Git needs you to decide how to combine

File markers indicate conflicting changes

Edit the file to combine the content

Add and commit to create merge commit

Merge commit has 2 parent commits

```
<<<<<<< HEAD
yellow
green
light purple
=======
bright yellow
green
dark purple
>>>>>>> master
```

```
bright yellow
green
light purple
```

◀ This is the information in your branch

◀ Separator for other file content

◀ This is the information in the file from the branch you are merging in

◀ Resolve conflicts, save and commit file
◀ Be sure to remove file markers