Kharagpur Open Source Society presents

GIT AND GITHUB CONCEPTS

Always code as if the guy ends up maintaining your code will be a violent psychopath who knows where you live.

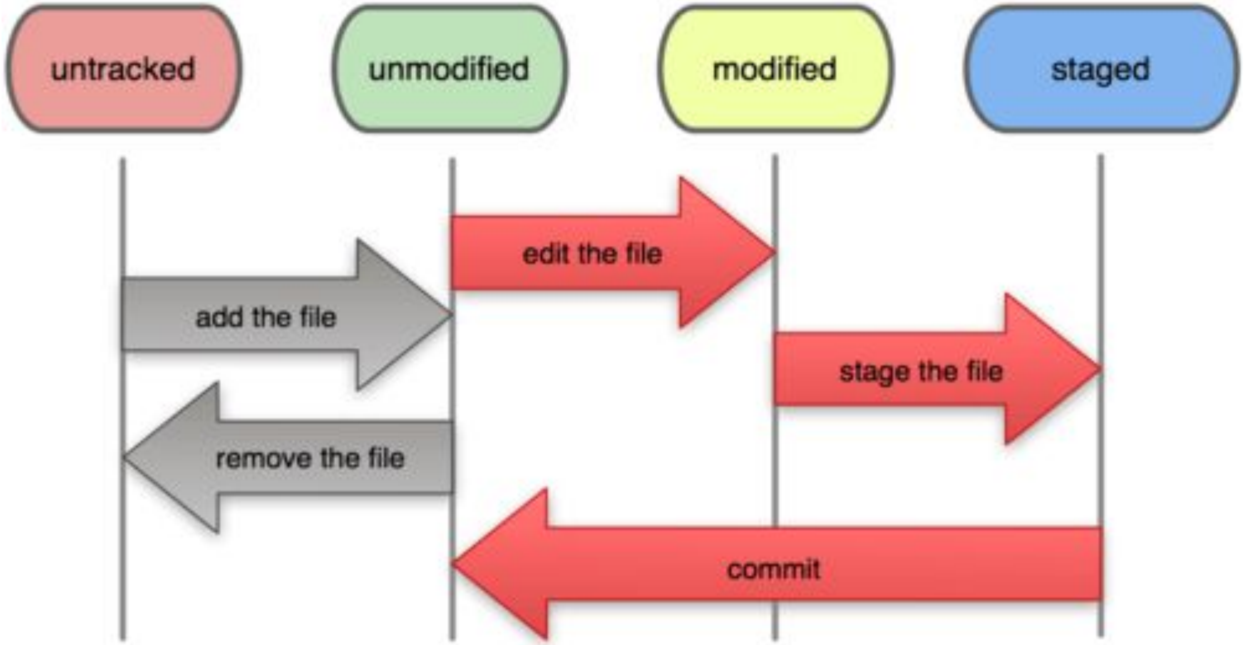
- Martin Golding

Table of Contents :

Current status

Git Operations	Incident	🟥	API Requests	Incident	🟥
Webhooks	Incident	🟥	Issues	Incident	🟥
Pull Requests	Incident	🟥	Actions	Incident	🟥
Packages	Normal	✓	Pages	Degraded	🟨
Codespaces	Incident	🟥	Copilot	Normal	✓

File Status Lifecycle



BASIC GIT COMMANDS :

QUICK QUIZ ?

1. What tab do you use to request that someone reviews and approves your changes to a project before they become final?
2. Which git command allows you to see the state of your working directory and the staged snapshot of your changes?
3. Under which tab on GitHub will you find all the source files for a repository?
4. Which Git command transfers changes from your local repository to the remote repository?

Helper Notes

ADVANCED GIT CONCEPTS :

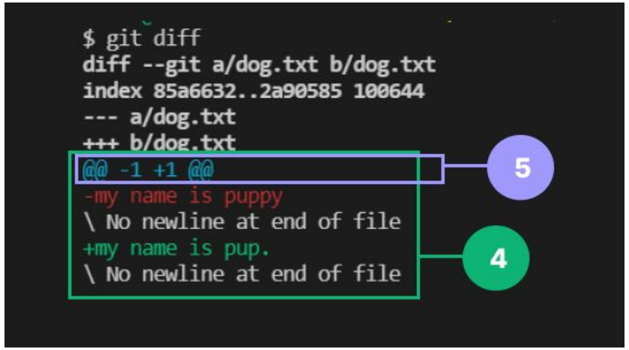
1)GIT DIFF

- *Synopsis*

  Diffing is a function that takes two input data sets and outputs the changes between them.The data sources can be commits,branches,working tree,etc.

- *Example*

  Suppose we have created a file **“dog.txt”**. We added it to the staging area with content **‘my name is puppy’** in it. Now we apply git diff command but it shows nothing.That is because the command compares the working directory with the staging area file.But now we modify **‘puppy’** with **‘pup’** .If we run command git diff, the outcome is **weird**.
  In the first line, Git has named old version of file as **‘A’** & newer as **‘B’.**
  In third line,git assigned a minus sign **‘-’** to **‘A’** & **‘+’** to **‘B’**. In 4th line,Git shows chunk of lines with red being modified to green lines.
  The fifth corresponds to a chunk header.



```
$ git diff
diff --git a/dog.txt b/dog.txt
index 85a6632..2a90585 100644
--- a/dog.txt
+++ b/dog.txt
@@ -1 +1 @@
-my name is puppy
\ No newline at end of file
+my name is pup.
\ No newline at end of file
```

What does the second line signify???
Can you see -1 & +1 ??? \ No newline at end of file ???

- *Other Comparisons*

1. **git diff HEAD** − Command to compare the both staged and unstaged changes with your last commit.

2. **git diff <branch_name1> <branch_name2>** – Command to compare the changes from the first branch with changes from the second branch.Order does matter when you're comparing branches. So diff's outcome will change depending on the order. Branch comparison considers the commits only. It doesn't look for staged and unstaged changes.

3. **git diff <commit_hash> <commit_hash>** – Command to compare the changes between two commits.

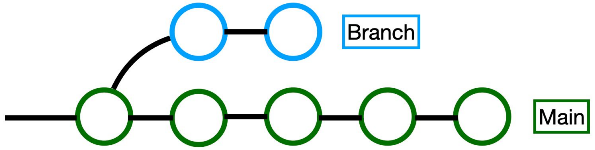
4. **git diff --staged <file_name>** – Command to compare the staging area file with committed file.

5. **git diff --cached <file_name>** – **Can you guess?**

How can you display a list of files added or modified in a specific commit?
git diff-tree <commit_hash>

2)GIT SWITCH

- *Synopsis & Options*



1. **git switch <branch_name>** – It simply specifies the local branch you want to switch.This will make given branch the new HEAD branch.

2. **git switch -c <branch_name>** – You can create and switch to a new branch in one go.

3. **git switch –** – Using it ,you can switch back to the previously checkout branch.

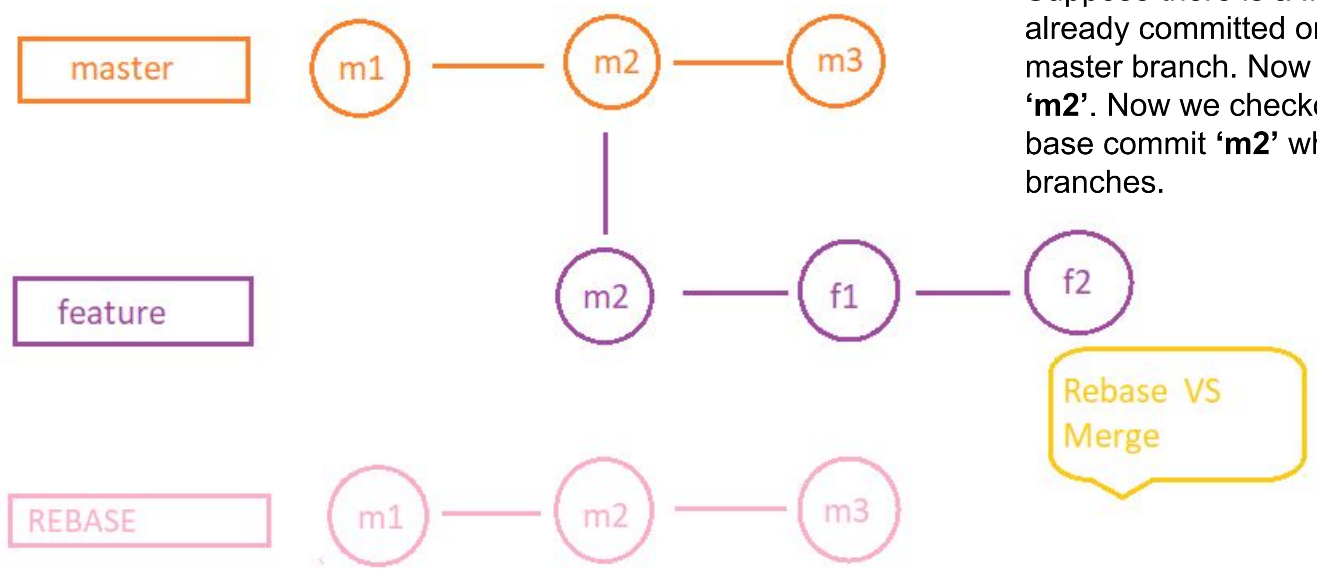
4. **git switch remote branch** – If you want to check out a remote branch (that doesn't yet exist as a local branch in your local repository), you can simply provide the remote branch's name. When Git cannot find the specified name as a local branch, it will assume you want to check out the respective remote branch of that name.This will not only create a local branch, but also set up a "tracking relationship" between the two branches, making sure that pulling and pushing will be as easy as "git pull" and "git push".

What is a remote branch ? How is it different from a local branch?

3)GIT REBASE

- *Synopsis & Examples*

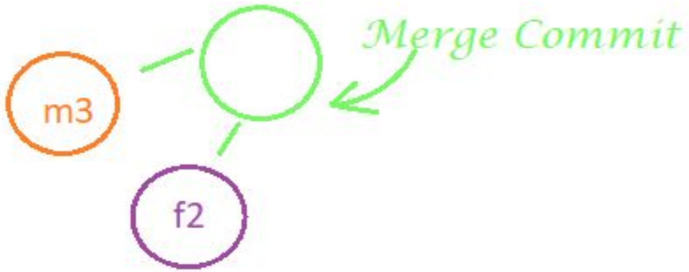
Rebase is another way to integrate changes from one branch to another. Rebase compresses all the changes into a single “patch”. Then it integrates the patch onto the target branch.

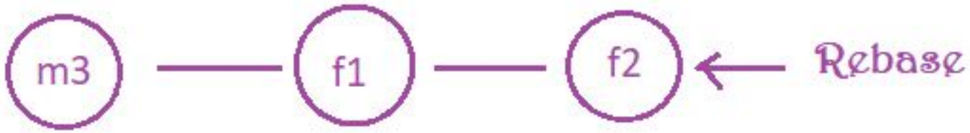


Suppose there is a file in working directory .It is already committed once with message **‘m1’** on master branch. Now it is committed second time as **‘m2’**. Now we checkout a branch **‘feature’** on initial base commit **‘m2’** which is common in both branches.

Now we are working in **‘feature’** branch and file is added and committed once with message **‘f1’**. In meantime,our colleagues are working in the master branch and have committed it third time as **‘m3’**.And we have again committed the file in feature as **‘f2’**.
Now we want to integrate the changes made in both the branches. There are two ways :

1. **git merge** – The command simply merges the branch to the master along with all commits being merged .It may be beneficial but we have an additional commit just for merge with HEAD pointing to master. Moreover ,it is tedious to understand and locate a commit.





2. **git rebase** – We apply command ‘git rebase feature’.Now git compares the last commit of both branches.Here the last commit is ‘m3’.So git rebase command changes the base of feature branch with the last commit , here,i.e., **‘m2’** to **‘m3’**.Thus **’f1’** is based on **‘m3’**
   And git proceeds feature branch ahead with this base and aligns feature branch with master branch putting every next commit of feature branch on top of **‘m3’** commit.

Now if we rebase the master branch ,then git checks changes in feature branch and apply it to master branch and also update the master branch to latest commit.

Rebases are how changes should pass from the top of the hierarchy downwards, and merges are how they flow back upwards.

To have a clear and simple branching, git rebase <branch_name> is advisable to use due to traceability and understandable history commit.

What is purpose of command **‘git rebase -i’** ?

What does the command **‘git rebase -i HEAD~10’** do?

What is **‘merge conflict’**?

How many **ways** are present in Git to integrate changes from one branch into another?

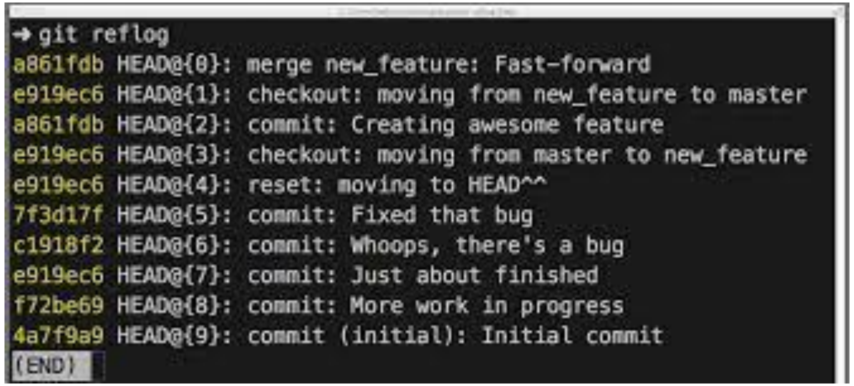
What is the main issue with using **git rebase** when working with multiple developers?

.

4)GIT REFLOG

- *Synopsis & Example*

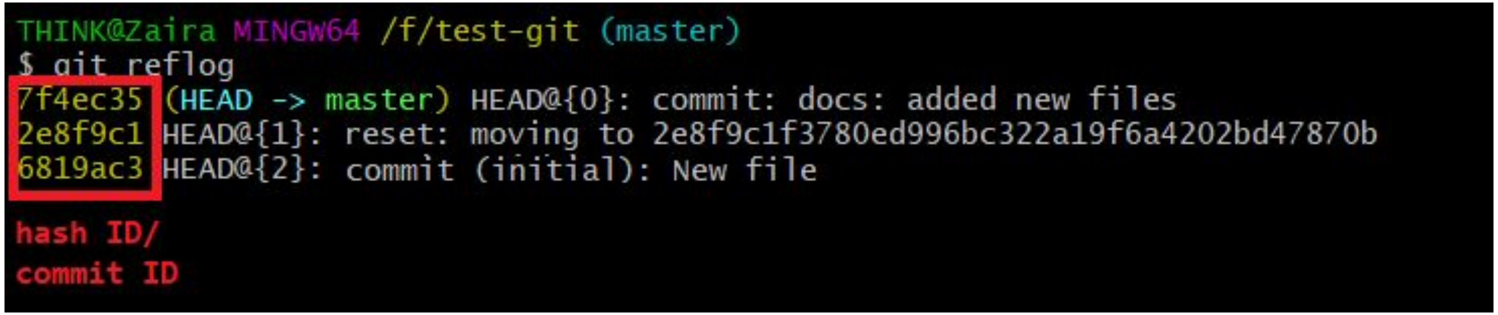
‘Reflogs’ are short form of reference logs.Git keeps a record of tips of branches and other references that were updated in the repository.
The git reflog accepts subcommands like show,expire,delete and exists.
Git can show logs history of at most last 90 days.
The ‘git reflog show’ command will show the log of a specific reference (it defaults to HEAD).To view logs of tips of a particular branch,we can apply **‘git reflog show <branch_name>’**.



```
➜ git reflog
a861fdb HEAD@{0}: merge new_feature: Fast-forward
e919ec6 HEAD@{1}: checkout: moving from new_feature to master
a861fdb HEAD@{2}: commit: Creating awesome feature
e919ec6 HEAD@{3}: checkout: moving from master to new_feature
e919ec6 HEAD@{4}: reset: moving to HEAD^^
7f3d17f HEAD@{5}: commit: Fixed that bug
c1918f2 HEAD@{6}: commit: Whoops, there's a bug
e919ec6 HEAD@{7}: commit: Just about finished
f72be69 HEAD@{8}: commit: More work in progress
4a7f9a9 HEAD@{9}: commit (initial): Initial commit
(END)
```

Let us clear concept using image beside.The last line in image corresponds to the initial commit.The characters shown in yellow text together are called as a hash of commit and checkout.The hashes and HEAD@{ } are references and are helpful to move from one place to another.The hash and head@ are not fixed.As we apply commands further,they change. But the recent command is always allocated as HEAD@{0}.Since particular branch is not specified, it follows the head pointer.

Thus,reflog is a logging mechanism and keeps a track of all the changes against their unique hash-id. Picking the commit ID using ‘git reflog <hash-id>’ helps us to revert a commit.



```
THINK@Zaira MINGW64 /f/test-git (master)
$ git reflog
7f4ec35 (HEAD -> master) HEAD@{0}: commit: docs: added new files
2e8f9c1 HEAD@{1}: reset: moving to 2e8f9c1f3780ed996bc322a19f6a4202bd47870b
6819ac3 HEAD@{2}: commit (initial): New file
```

What information do Git reflogs (reference logs) store?

After accidentally deleting a branch in your local repository, how can you recover it? Find the hash of the branch with the reflog command, then execute git checkout -b <branchname> <hash>.

5)GIT CHERRY-PICK

- *Synopsis*

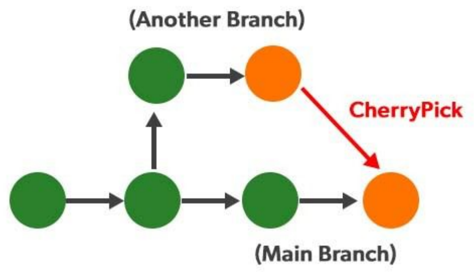
Cherry-picking in git means choosing a commit from one branch and applying it to another branch. This is in contrast with other ways such as merge and rebases which normally apply many commits into another branch.

Cherry-picking is just like rebasing It is mainly used if you don't want to merge the whole branch and you want some of the commits.It is mainly used for bug fixes where you want to place that bug fix commit in all the version branches.It is also used when we accidentally made a commit in wrong branch.
But it can cause lot of duplicate commits.

- *Example as a **DEMO VIDEO** & QUIZ*

We use command **git cherry-pick <commit_id>** for selective commits.

Practical Implication by me



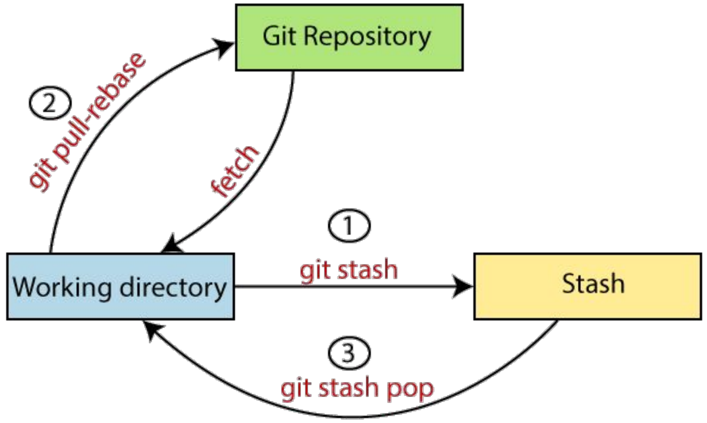
What does the following command describe?

```
git checkout feature-user-location
git cherry-pick kj2342134sdf090093f0sdgasdf99sdfo992mmmf9921231
```

6)GIT STASH

- *Synopsis*

git stash command saves the previously written code and then returns to the last commit for a fresh start. Now you can add the new feature without disturbing the old one as it is saved locally. After committing to the new feature you can go on with working on the old feature which was incomplete and uncommitted.It allows us to switch branches without committing the current branch. It helps us to store them in hidden files.



- *Example as a* ***DEMO VIDEO***

Normally before switching branch,we first commit our work.But suppose we don't wish to commit our undone work.

If we switch without committing, two things may happen:

1. Switches to branch carrying the changes
2. Git will not allow to switch the branch and asks to commit or stash the changes.

Git stash command demo video

- *Some of the git stash commands to know :*
  - **Git stash save “message”**
  - **Git stash list**
  - **Git stash apply**
  - **Git stash changes**
  - **Git stash pop**
  - **Git stash drop**
  - **Git stash clear**
  - **Git stash branch**

While modifying a file, you're unexpectedly assigned an urgent bug fix on another branch. How can you temporarily save your local work without committing?

What do the **git stash drop** and **git stash branch <branch_name>** command do?

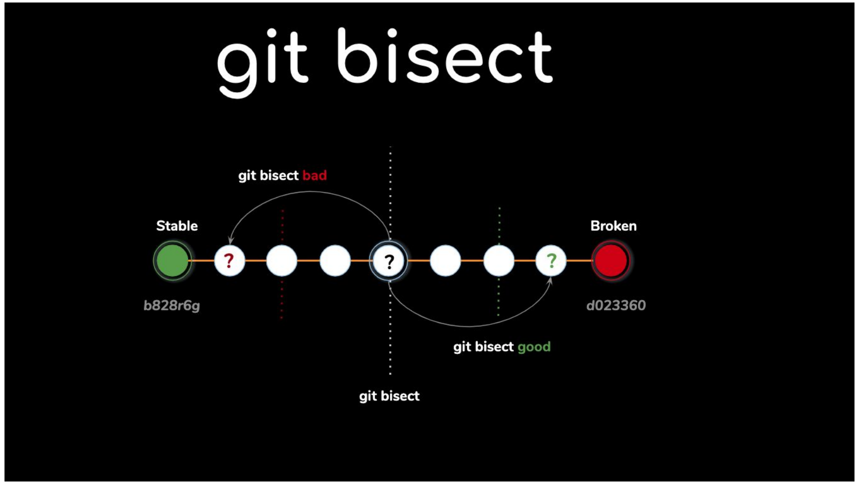
7)GIT BISECT

- *Synopsis*

This command uses the binary search algorithm to find out the commit that introduced a bug.

How it works?

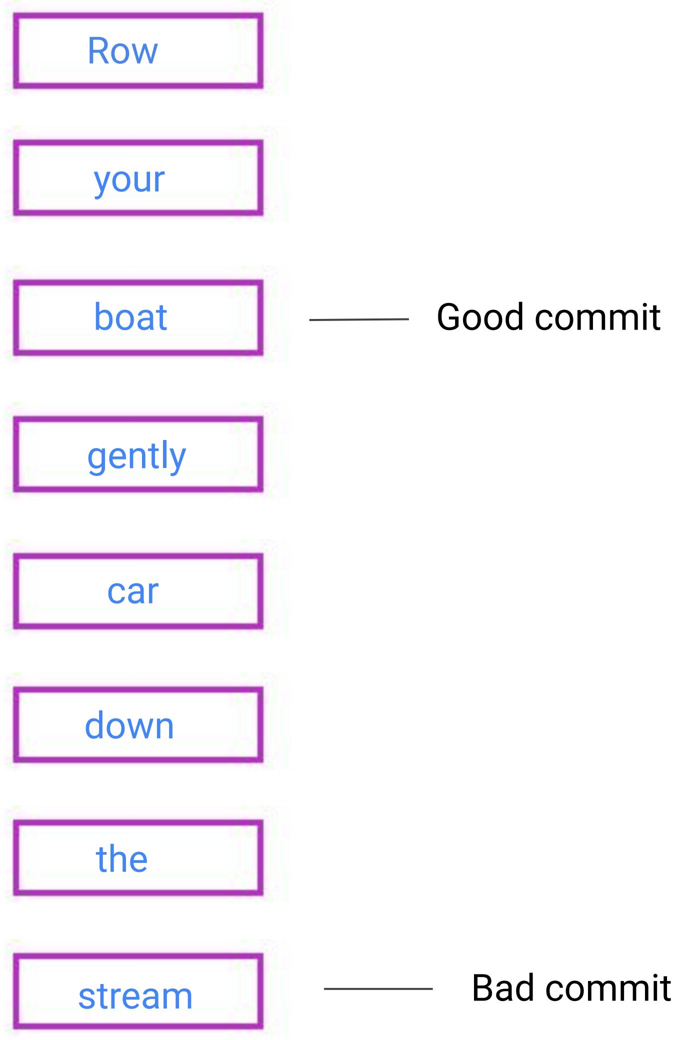
- **Git bisect start** : It starts up a git bisect wizard.
- **Git bisect bad "version"** : It lets the git bisect wizard of a bad commit.
- **Git bisect good "version"** : It lets the git bisect wizard of a bad commit.
- **Git bisect reset** : It ends git bisect wizard.



- *Quick quiz ???*

What is the operation doing given the Git commands below?

git bisect **start**
git bisect **bad5d41402abc4b2a76b9719d911017c592**
git bisect **good 69faab6268350295550de7d587bc323d**



- *Example*

Suppose we have created a file ‘text.txt’ which spells out a poem .We have written a verse *‘Row your boat gently down the stream’*.But by mistake,we have introduced a bug since word *boat* is replaced with *car* in further commits.The following given snapboxes are commits in master branch.The file is committed after every single word.So we will use **git bisect** to find the commit which introduced the bug.

After starting wizard, we will declare a good commit using git log and checking our output.Then we may use the most recent commit as a bad commit as our final result is not satisfying.
When we typed **git bisect bad** in this step, git bisect checked out an old commit for us – the commit halfway between the latest *bad* commit and the known *good* commit. This is how bisect works – it cuts the commit history down in halves until it finds the original bad commit.
Based on our result for the half ,we will declare the commit pointed as good or bad. Here,on dividing,the word *car* has appeared. Declaring bad commit, git bisects the half again.

Finally, we are able to find that particular commit .
The command **git bisect reset** is used to exit the wizard.

CONCLUSION :

Git is a very powerful tool and helps to collaborate professionally,save our code,presentation and learn from others’ coding related projects.

Take a tiny second to celebrate our cognitive effort**.**Because we have already gotten our hands dirty with git commands’ outcome.

THANKS
FOR THE
OPPORTUNITY

MODI MEET -9913239263
modimeet20@gmail.com