GIT COMMANDS

Command: git init -> initializes the new git repository which is empty.

A screenshot of a computer program

AI-generated content may be incorrect.

**ls -a**

shows the hidden files.

A screenshot of a computer program

AI-generated content may be incorrect.

**ls -al**

to show the hidden files in long format.

A computer screen shot of a program

AI-generated content may be incorrect.

Under the. git folder all the information regarding the git folder would be there.

A screen shot of a computer

AI-generated content may be incorrect.

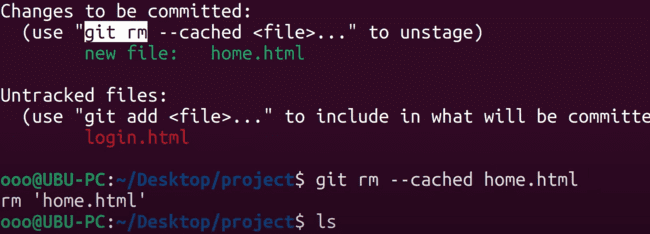
There are three stages in git:

1. Untracked
2. Staged
3. Commit

A yellow circle with text on it

AI-generated content may be incorrect.

Moving back to the untracked to staged, we need to use the following command:



git commit without giving “-m”, then editor would be open and then we need to provide the comment to save the file.

Adding the committing at the same time:

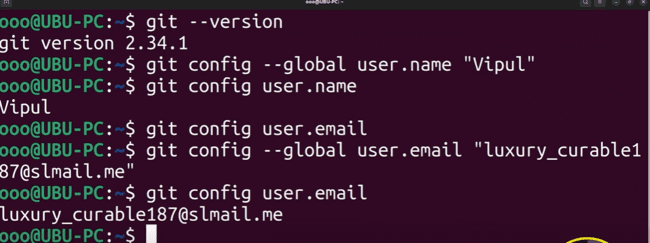
**git commit -am “<COMMENT>”**

GIT CONFIG: Following are the commands to config the name and the email address for the github.

A black background with white text

AI-generated content may be incorrect.

Configuring the email and the name of the user and checking that, user and the email address is configured as:



**GIT CHECKOUT:**

The command touch is used to create the file as shown below:

A screenshot of a computer

AI-generated content may be incorrect.

To create a new branch, we need to use the command “git branch <BRNACH\_NAME>” and to view the branches we need to use the command as “git branch” as shown below:

A screen shot of a computer

AI-generated content may be incorrect.

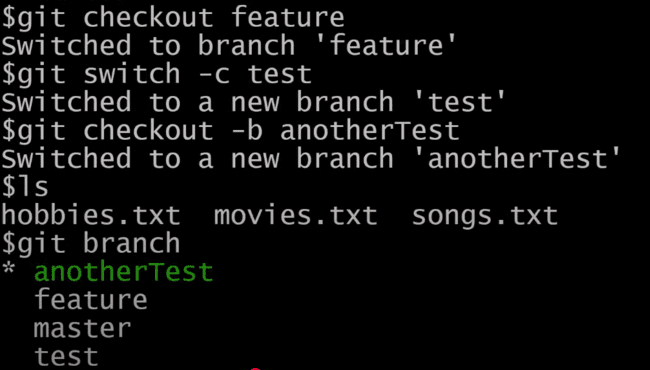
To switch to the feature branch, we need to use the “git switch <BRANCH\_NAME>” and to go back to the previous branch, we need to use the command “git switch -”:

A black screen with white text

AI-generated content may be incorrect.

Command “git switch -c test” would switch the branch to test branch and create the test branch.

Also “git checkout -b anotherBranch” would switch the branch to anotherBranch and create that branch.



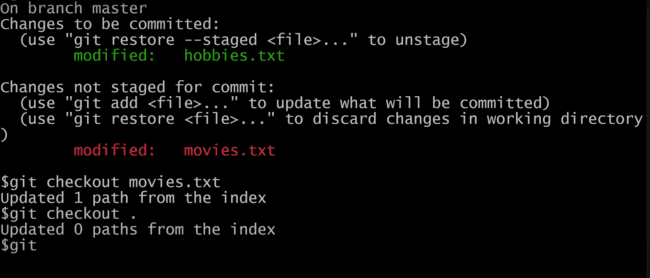
If you want to discard the changes, you need to use the command “git checkout <FILE\_NAME>” and that changes would be discarded and cannot be undo as well as shown below:

Like in the following example, there is one untracked file hobbies.txt and checkout that file would clear that. The files which are not in staging that would be discarded using the checkout command.

A screen shot of a computer

AI-generated content may be incorrect.

In the following two files are there, one is untracked, and one is in staging.

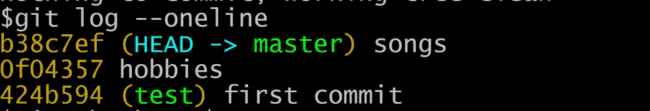


Command “git checkout” would discard the untacked file only, but staging file would be there.

A screen shot of a computer

AI-generated content may be incorrect.

Command “git –oneline” is used to check the HEAD version of the branch in which it currently is.



DETACHED HEAD:

In this case head is pointing to the commit, instead of pointing to any of the branch or any other commit. Pointing the head to some commit as shown below using the ref name:

A computer screen with white text

AI-generated content may be incorrect.

For the detached head, we can check that it is pointing to some commit:

A black background with white text

AI-generated content may be incorrect.

Cat HEAD is pointing to some reference, not to a folder as shown.

But if we are in the branch, head would be pointing to some branch.

HEAD 🡪 BRANCH 🡪 pointing to latest commit.

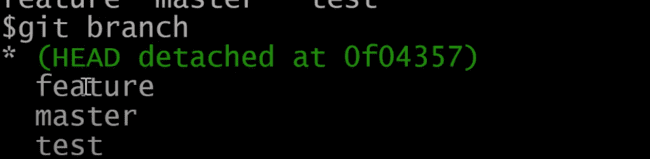
But in the below case, head to pointing to commit as shown:

HEAD 🡪 LATEST COMMIT

A screenshot of a computer screen

AI-generated content may be incorrect.

And “git branch” would do the following:



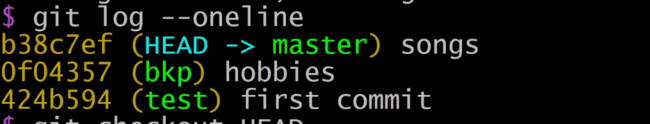
And switching to master:

A black screen with white text

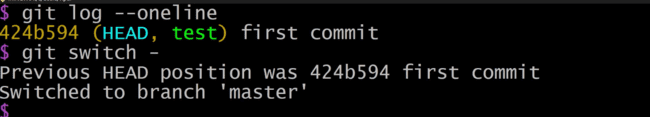
AI-generated content may be incorrect.

If you want to switch to the HEAD to the previous heads, then we need to use the wildcard “~”:

Command “git checkout HEAD~2”:



After “git checkout HEAD~2”:



There is one UI available for the GIT which is GitKraken Client as:

A screenshot of a computer

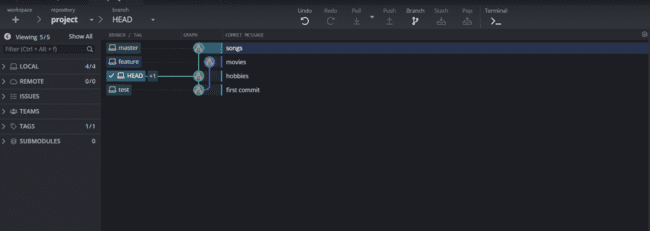
AI-generated content may be incorrect.

Importing the project to the GitKraken Client:

A screenshot of a computer

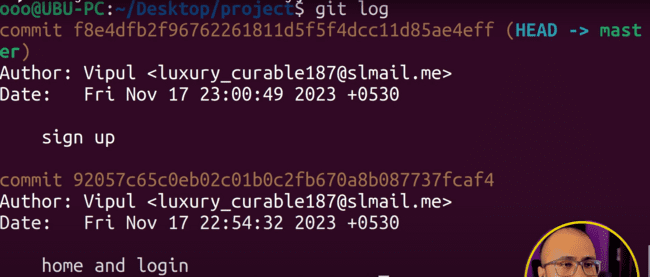
AI-generated content may be incorrect.

Now if execute “git checkout HEAD~1”, then UI would look like this:



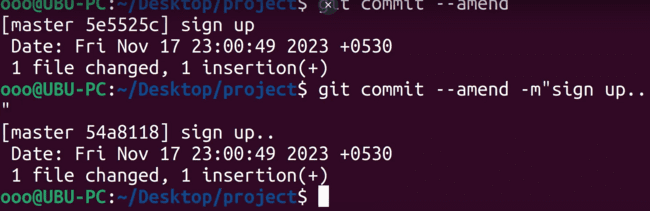
After “git status -”, that has been switched to the same branch.

Command “git log” displayed the messages and committed changes:



After changing in one file, if we need to add the same file to the “sign up” commit, we need to use the amend command.

git commit –amend -m“sign up..” as shown below:

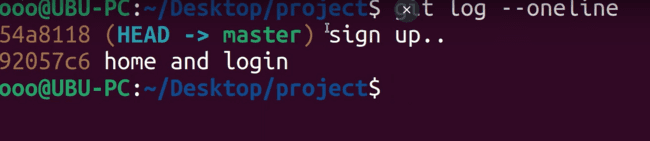


“git log” will display the updated commit message as shown below:

A computer screen shot of a program

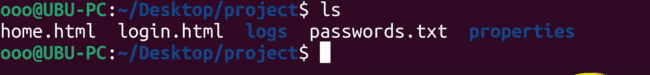
AI-generated content may be incorrect.

“git log --oneLine” would display the one line message for the commit as shown below:



“.gitignore” If we place any code in this file, that files or folders would be ignored. For example: we have added the following files and also added the following code in the “.gitignore” file as:

These are the files:

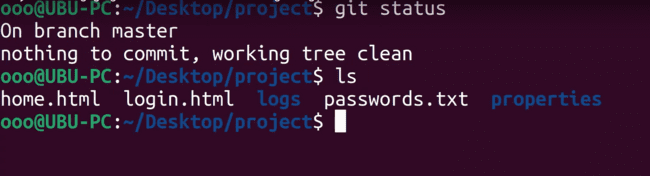


“cat .gitignore” contains the following code:

A computer screen with text on it

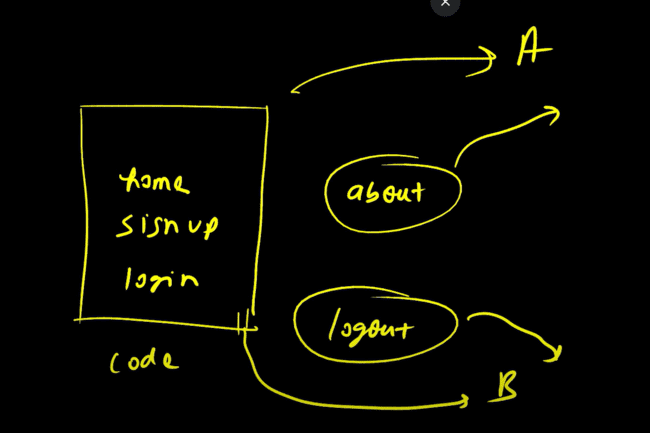
AI-generated content may be incorrect.

“git status” would be clean when code is added in the “.gitignore” file.

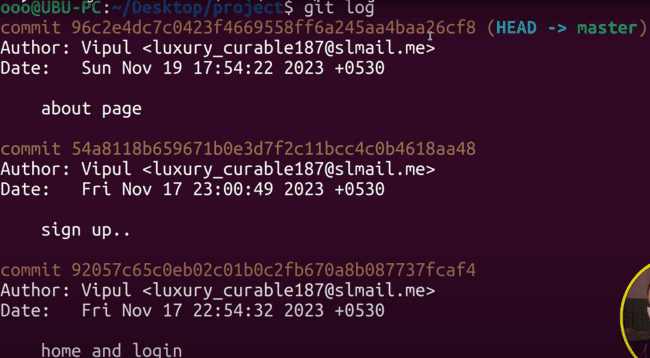


BRANCHES in GIT:

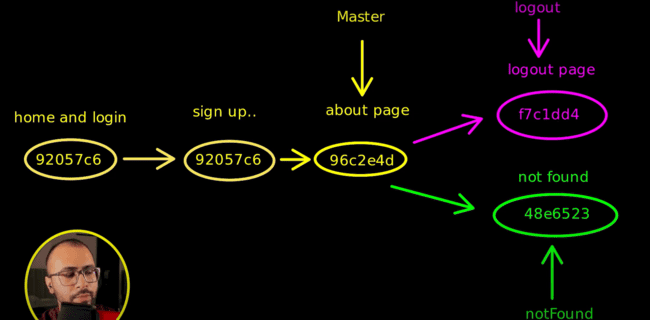
We have fetched the branches from the base code and for one branch we have added the “about.html” file and for another branch, we have added the “logout.html” file, and we can test them differently and then merge the same in the base branch.



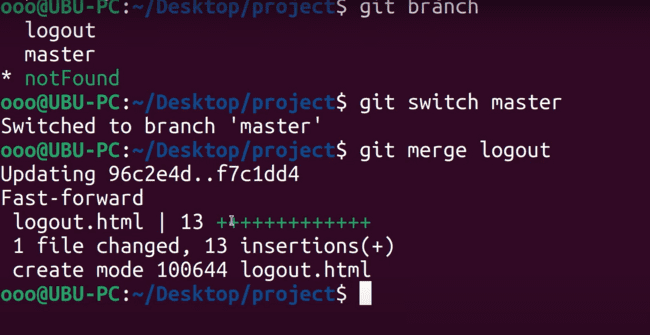
HEAD always points to the latest branch in the master when we do the commit, it moves to above again to the latest branch.



Merge the branches to the main master branch:



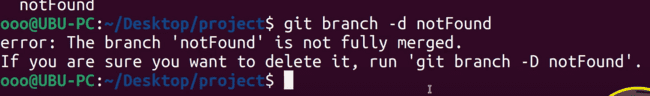
Now if we want to merge the logout code to the master, we can do that also, switching to the master and then merge the “logout” branch:



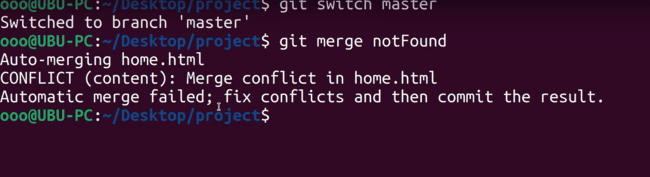
To delete the branch, we need to use the following command:



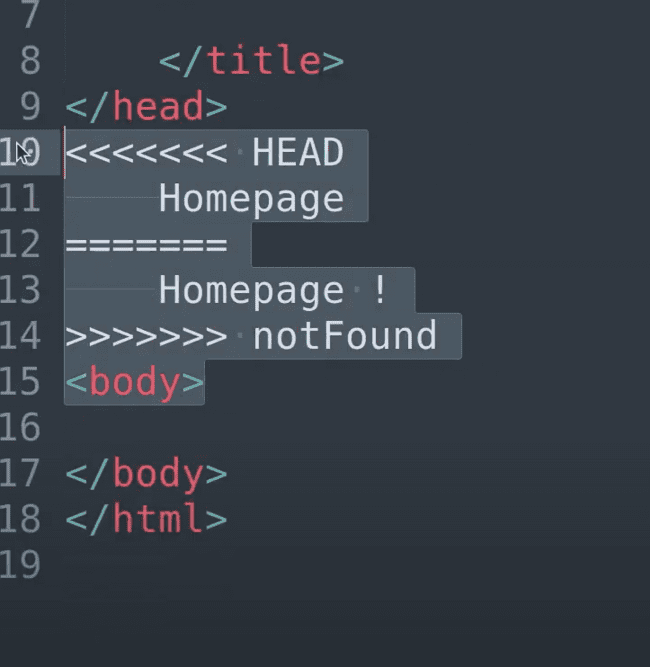
If the branch is not merged, then we are unable to delete that, and if we still want to delete that, we can do that using the “-D” which is force delete.



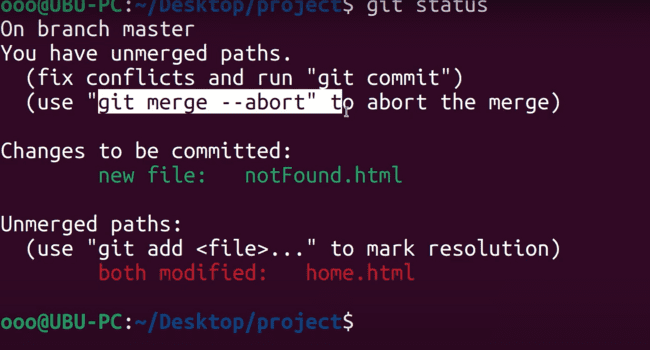
Now if we have done the changes in both the branches on the same line of the html page, then there would be the conflict. So, issue would be there.



If you open the home.html page, then we’ll be seeing the following message on the screen for the master branch and then after that for the other branch.

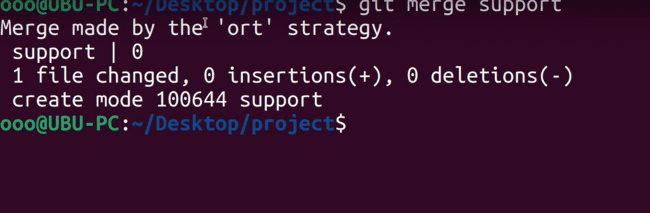


After resolving the conflicts, we can use the following to merge:



Other approach:

If we create the branch from master and add a page to that branch and then again from master, we add another branch and add a new html page, then a new merge would be created as:



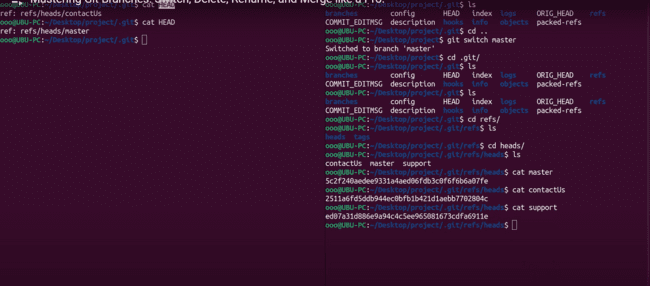
Previously, it was fast forward one, now it is ‘ort’ strategy. What happens is, a new commit is created for the same as:

A diagram of a flowchart

AI-generated content may be incorrect.

Need of HEAD:

It makes the branch switching easier, as head points to the reference of the current branch and contains the hash information which has been shown below:



To check the difference between the files, we can use the “diff” command as:

“git diff” which shows the difference between all the files.

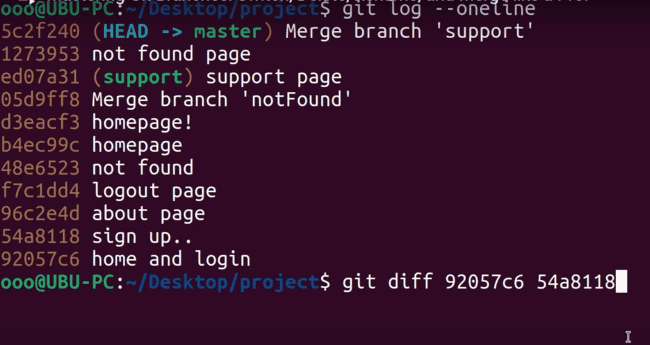
“git diff <FILE\_NAME>” which shows the difference for the specific file

“git diff --staged” shows the difference between the files that are in staging.

We can also check the difference between two commits as well:

“git diff <OLD\_ONE>..<NEW\_ONE>”

“git diff 9288..9000”



To check the difference for every files even that would be untracked or that would be staged, then we need to use the following command, which will compare to the last commit:

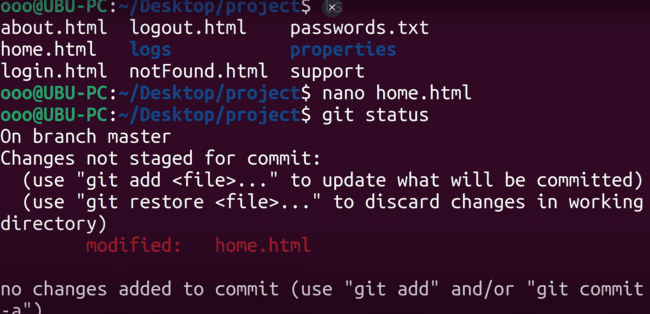
A screenshot of a computer

AI-generated content may be incorrect.

STASH:

Example: if we change the branch and we want that local changes would remain to the previous branch and would be saved and after that when we come back, we can get those changes, that can be done using the stash command.

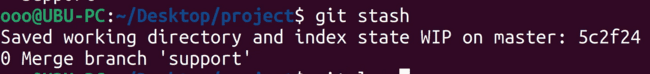
Changed in the home.html and that is untacked



Now, if you want to switch to another branch and want the changes would not be there in the other branch then:

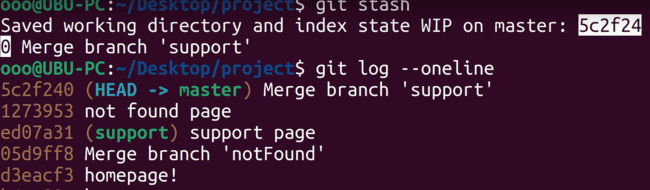
A screen shot of a computer

AI-generated content may be incorrect.

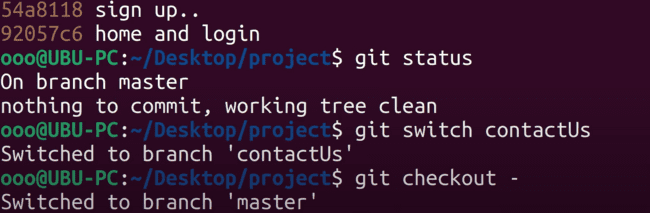


“git stash” would put the files in the main memory and working tree would be clean.

And saved in the last commit:



Stash the changes, switching the branch and then come back to the same branch.



“git stash list” to list the stash changes:

A computer screen shot of a code

AI-generated content may be incorrect.

To get back the stash changes: “git stash pop”

A screenshot of a computer program

AI-generated content may be incorrect.

After this if we run the command “git stash list”, then that would be empty.

A screen shot of a computer

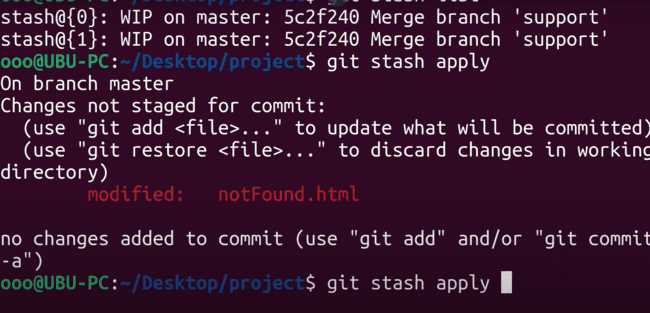
AI-generated content may be incorrect.

If you want that list won’t be empty, then you need to use, then add a new file and stash again:



In this scenario, if you want a particular change, which would either be the first one or the last one, then you need to use the following:

“git stash apply” will pick the last change and execute the pop command on that as well.



To delete something from the stash list, then you need to do the following:

A screen shot of a computer

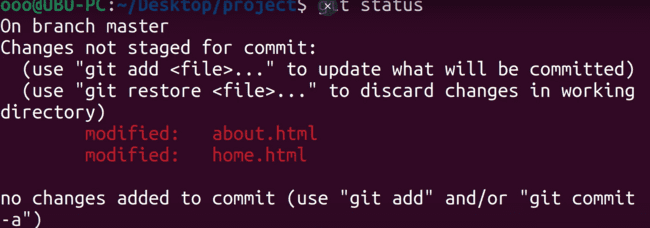
AI-generated content may be incorrect.

Also “git stash clear” would clear the whole stash list.

A screenshot of a computer program

AI-generated content may be incorrect.

If you want to stash a particular file, then you need to use the following command:



A screen shot of a computer program

AI-generated content may be incorrect.

We can also use the message while stashing the git changes.



A computer screen with blue text

AI-generated content may be incorrect.

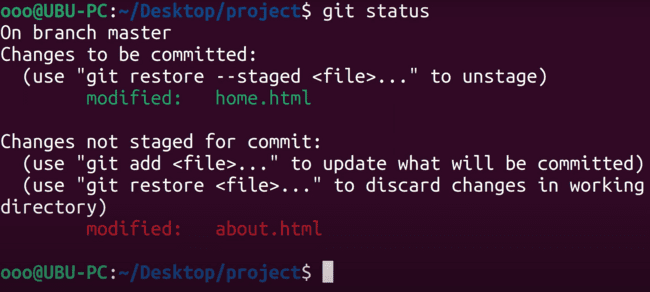
Command to show the changes:

A screen shot of a computer

AI-generated content may be incorrect.

**To see the exact changes, we need to use “git stash show -p stash@{0}”.**

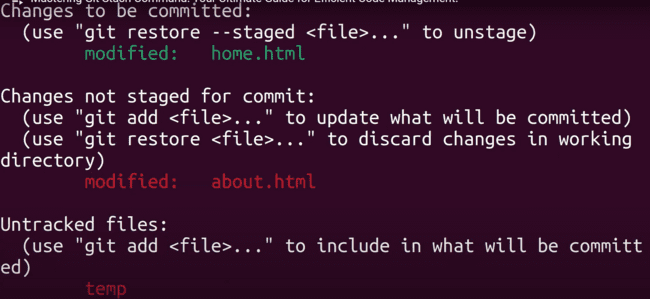
Home.html in staged and about.html is untracked:



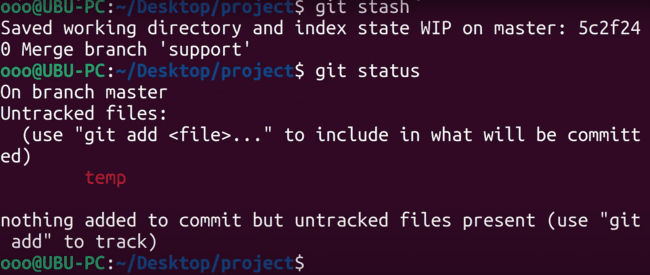
To stash only the untacked changes, not the staged one, then we need to use the following: A screenshot of a computer program

AI-generated content may be incorrect.

There is one more concept, if we have just created a new file, then it would be untracked, like that file is not used in the staged or even tracked and newly created.

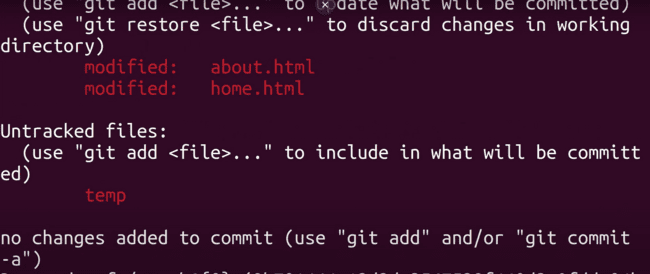


Doing git stash and then git status, temp file would also be there as well.



*Untracked file is not stashed, to make this file as stash we need to use the following command:*

First, you need to provide the command “git stash pop” and after that following command needs to be executed:





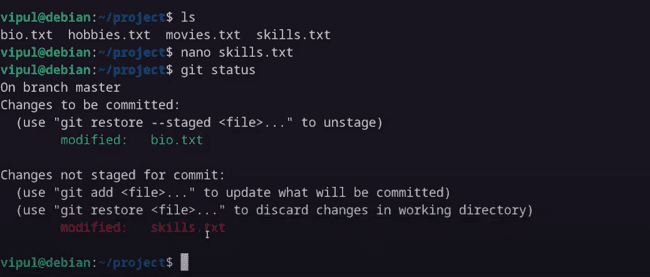
That temp untracked file would also be included, and we can see that using the “git status” command.

A screenshot of a computer

AI-generated content may be incorrect.

**GIT RESTORE**

Two files are there:



To make the file unstaged, you need to use the “restore” command:

A screen shot of a computer

AI-generated content may be incorrect.

Using the command “git status” we can see that file is in unstaged:

A screen shot of a computer program

AI-generated content may be incorrect.

To discard from the unstaged, following command needs to be used:

A screen shot of a computer

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

That file would be gone now:

A screen shot of a computer code

AI-generated content may be incorrect.

To remove both the files, we need to use the following command “git restore –staged --worktree”:

A screenshot of a computer program

AI-generated content may be incorrect.

“git restore .” also do the same:

A screen shot of a computer

AI-generated content may be incorrect.

If we want to check the files of the branch “skills”, what is the content of the file movies.txt, without changing the HEAD, then following command needs to be executed:

A screen shot of a computer program

AI-generated content may be incorrect.

Which is empty:

A screenshot of a computer

AI-generated content may be incorrect.

“git restore .” will come back to the master again. And, we can see the contents of the file movies.txt.

A screenshot of a computer

AI-generated content may be incorrect.

To check all the files on that source, we need to use the “git restore –source=244hv .”

A screenshot of a computer

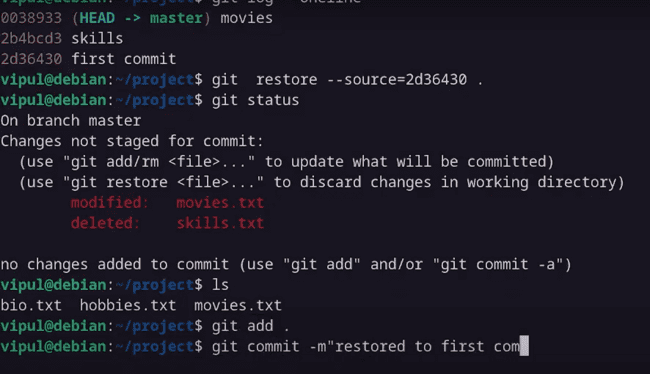
AI-generated content may be incorrect.

To go to the “first commit” source, we need to use:

A screen shot of a computer program

AI-generated content may be incorrect.

If we commit the files using some source, then we can use the following:



A new commit would be created as well:

A screen shot of a computer program

AI-generated content may be incorrect.

To go to the HEAD, then we need to use the following as HEAD in source, otherwise, need to use, HEAD~1, or HEAD~2:

A screen shot of a computer program

AI-generated content may be incorrect.

GIT RESET: is also known as UNDO -> moving head to the specific state.

With the GIT RESTORE, we can check the photo of previous commit and also moves the files to the specific commit and can make a new commit as well, but if you want to delete the commit history for a particular branch, then we need to use the GIT RESET.

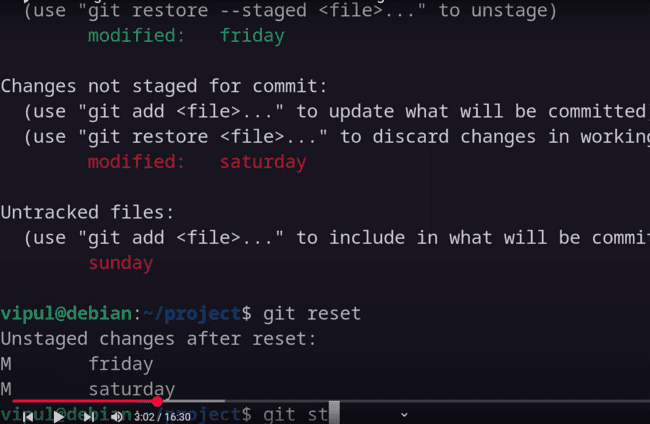
A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

You have added the file to staged and want to get that back to the unstaged one, then we can use the following:



After reset, the files would be:

A computer screen with white text

AI-generated content may be incorrect.

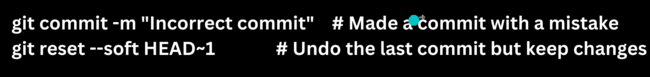
Second use of the RESET:

A black background with white text

AI-generated content may be incorrect.

**A black background with white text

AI-generated content may be incorrect.**

****

HEAD~1 -> goes back to the previous commit to HEAD.

--soft or –hard can be used in the command.

OTHER:



A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

--soft sends out the changes in the staging area and undo the commit.

But –mixed, undo the commit and also sends out the changes in the unstaged area.

A black background with white text

AI-generated content may be incorrect.

--hard discards the changes and undo the last commit.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

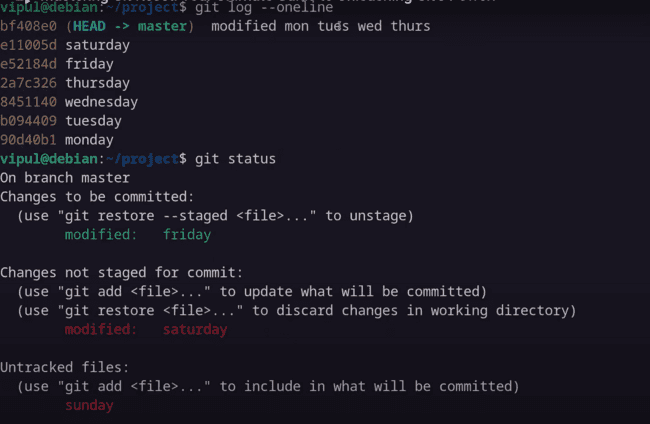
A black background with white text

AI-generated content may be incorrect.

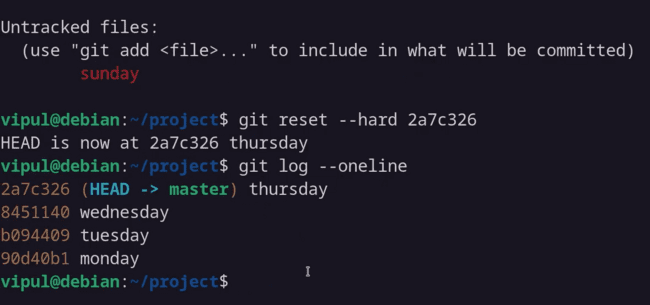
A black background with white text

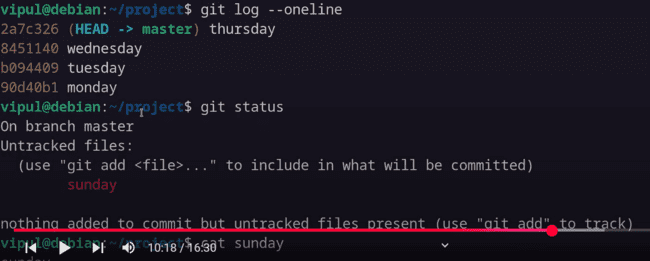
AI-generated content may be incorrect.

Example:

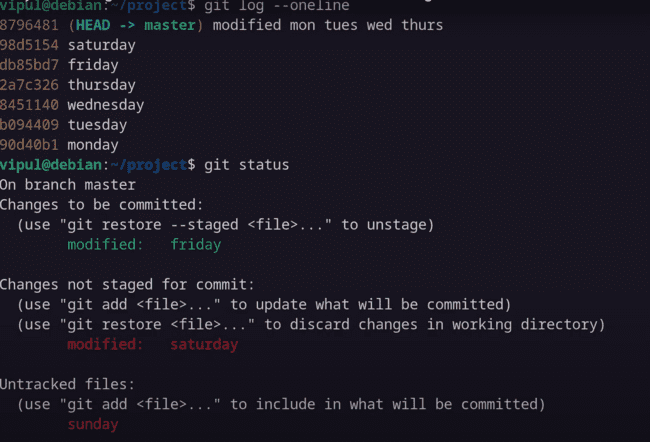


--hard goes to previous commit and all other commits are deleted and files as well instead of untracked files only





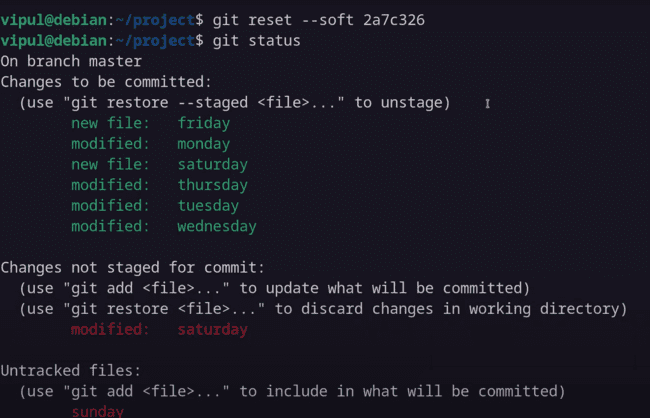
Again we have added the same files to check the other restore command.



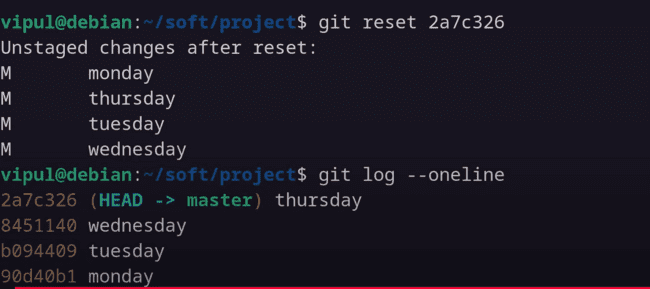
--soft will move all the staged changes from that commit to the previous commit and move the head to that previous commit, unlike –hard, which would discard all edits.

A screen shot of a computer

AI-generated content may be incorrect.



For –mixed, we don’t need to specify the mixed in the command and HEAD is reset to the previous commit and changes has been moved to the unstaged files and unstaged and staged would move to the the untrack one.



A screen shot of a computer program

AI-generated content may be incorrect.

GIT REVERT: not changing the commit history and want to undo the changes, then revert need to use which would create a new commit.

A computer screen shot of a black screen

AI-generated content may be incorrect.

*Now if someone is working in the master branch and we delete that using reset and move to the other branch “games”, then it would create issues while merging the code which would create the conflict.*

*So we can use ‘git revert’*

A computer screen with text

AI-generated content may be incorrect.

Which branch we need to undo, for that hashcode we need to give in the revert.

A black screen with white text and green text

AI-generated content may be incorrect.

This command did not delete the history, but added a new commit as:

A black screen with white text

AI-generated content may be incorrect.

CREATING SHORTCUT FOR THE GIT COMMANDS:  
Like for ‘git status’ we can use the ‘git st’ using alias.

Go to the home directory and need to open the .gitconfig file.

A screen shot of a computer screen

AI-generated content may be incorrect.

* nano .gitconfig

A computer screen with white text

AI-generated content may be incorrect.

And save the file.

Now ‘git st’ gives the same output as ‘git status’:

A computer screen with white text

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

For commit:

A computer screen with white text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Adding the alias using the command line: adding checkout

A screenshot of a computer program

AI-generated content may be incorrect.

To write “git status” as ‘gst’, then you need to do the following:

A screenshot of a computer program

AI-generated content may be incorrect.

Opening the file .bashrc, and at the end add the command:

A computer screen with white and green text

AI-generated content may be incorrect.

To restart the script, use the following command:

* source .bashrc

A screenshot of a computer program

AI-generated content may be incorrect.

GIT REBASE: