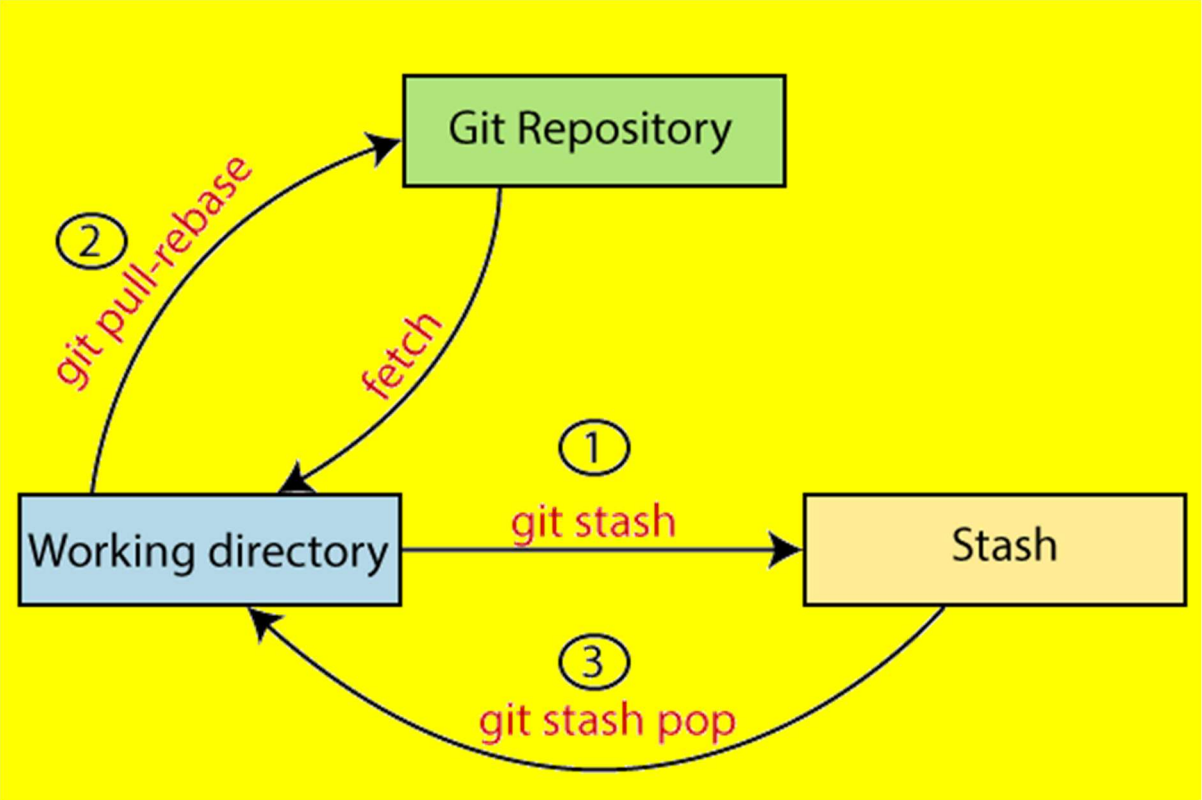
Git Stash

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branches without committing the current branch.

The below figure demonstrates the properties and role of stashing concerning repository and working directory.



Generally, the stash's meaning is "**store something safely in a hidden place**." The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

Stashing takes the messy state of your working directory, and temporarily save it for further use. Many options are available with git stash. Some useful options are given below:

- **Git stash**
- **Git stash save**
- **Git stash list**
- **Git stash apply**

- **Git stash changes**
- **Git stash pop**
- **Git stash drop**
- **Git stash clear**
- **Git stash branch**

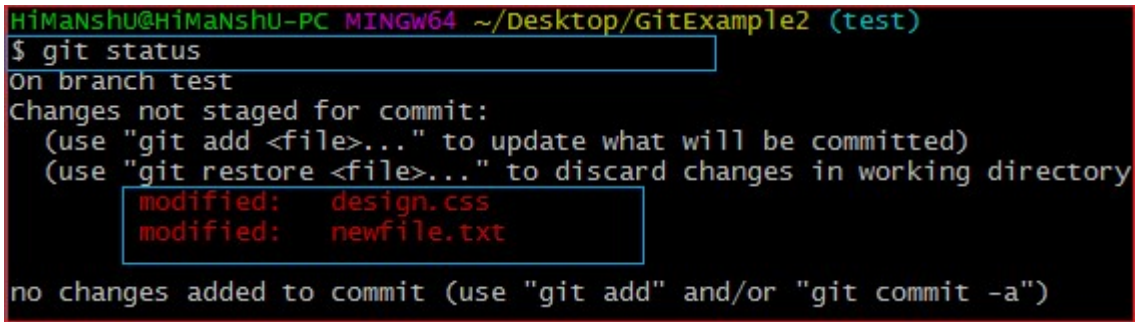
Stashing Work

Let's understand it with a real-time scenario. I have made changes to my project GitExample2 in two files from two distinct branches. I am in a messy state, and I have not entirely edited any file yet. So I want to save it temporarily for future use. We can stash it to save as its current status. To stash, let's have a look at the repository's current status. To check the current status of the repository, run the git status command. The git status command is used as:

Syntax:

1. $ git status

Output:

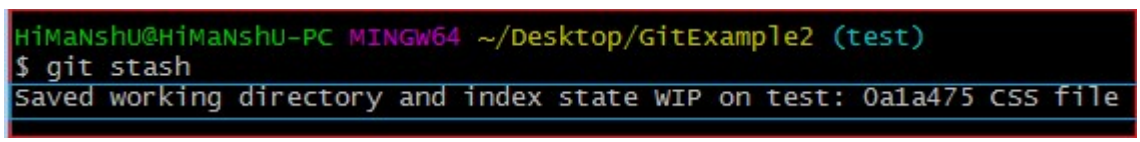


From the above output, you can see the status that there are two untracked file **design.css** and **newfile.txt** available in the repository. To save it temporarily, we can use the git stash command. The git stash command is used as:

Syntax:

1. $ git stash

Output:



In the given output, the work is saved with git stash command. We can check the status of the repository.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git status
On branch test
nothing to commit, working tree clean
```

As you can see, my work is just stashed in its current position. Now, the directory is cleaned. At this point, you can switch between branches and work on them.

Git Stash Save (Saving Stashes with the message):

In Git, the changes can be stashed with a message. To stash a change with a message, run the below command:

Syntax:

1. $ git stash save "**<Stashing** Message**>**"

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git stash save "Edited both files"
Saved working directory and index state On test: Edited both files
```


The above stash will be saved with a message

Git Stash List (Check the Stored Stashes)

To check the stored stashes, run the below command:

Syntax:

1. $ git stash list

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git stash list
stash@{0}: WIP on test: 0a1a475 CSS file

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$
```

In the above case, I have made one stash, which is displayed as "**stash@{0}: WIP on the test: 0a1a475 CSS file**".

If we have more than one stash, then It will display all the stashes respectively with different stash id. Consider the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git stash list
stash@{0}: On test: Edited both files
stash@{1}: WIP on test: 0a1a475 CSS file
```

It will show all the stashes with indexing as **stash@{0}: stash@{1}:** and so on.

Git Stash Apply

You can re-apply the changes that you just stashed by using the git stash command. To apply the commit, use the git stash command, followed by the apply option. It is used as:

Syntax:

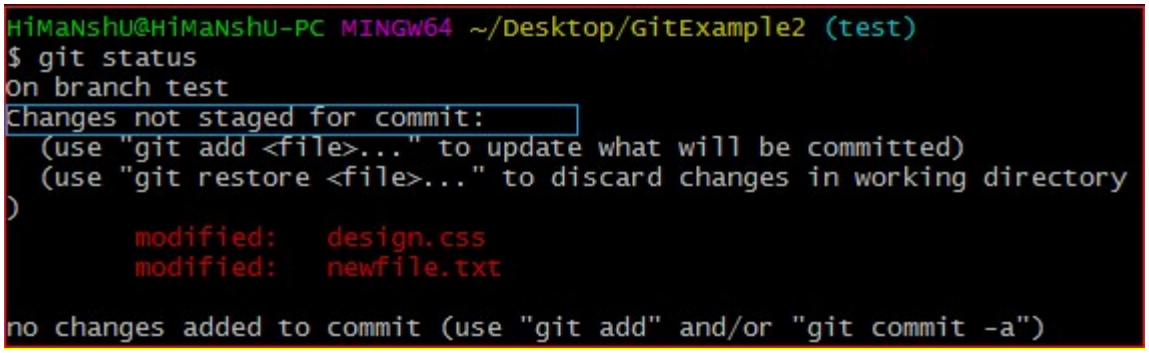
1. $ git stash apply

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git stash apply
On branch test
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:   design.css
        modified:   newfile.txt

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

The above output restores the last stash. Now, if you will check the status of the repository, it will show the changes that are made on the file. Consider the below **output:**



From the above output, you can see that the repository is restored to its previous state before stash. It is showing output as "**Changes not staged for commit**."

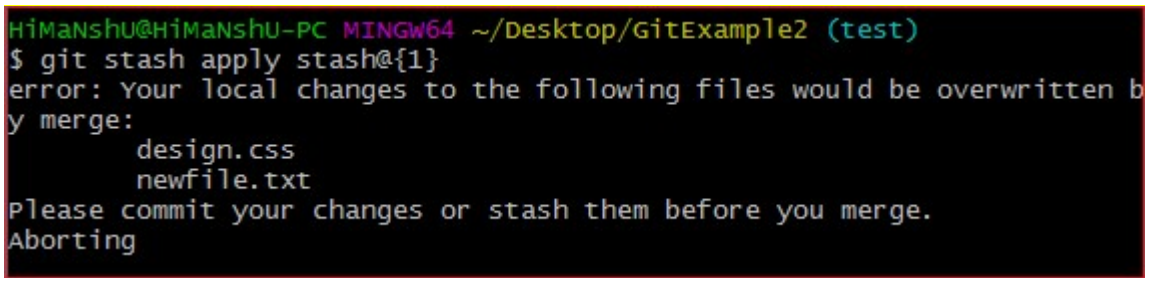
In case of more than one stash, you can use "git stash apply" command followed by stash index id to apply the particular commit. It is used as:

Syntax:

1. $ git stash apply **<stash** id**>**

Consider the below output:

Output:



If we don't specify a stash, Git takes the most recent stash and tries to apply it.

Git Stash Changes

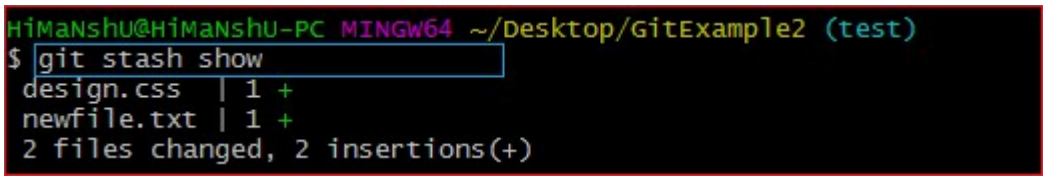
We can track the stashes and their changes. To see the changes in the file before stash and after stash operation, run the below command:

Syntax:

1. $ git stash show

The above command will show the file that is stashed and changes made on them. Consider the below output:

Output:



The above output illustrates that there are two files that are stashed, and two insertions performed on them.

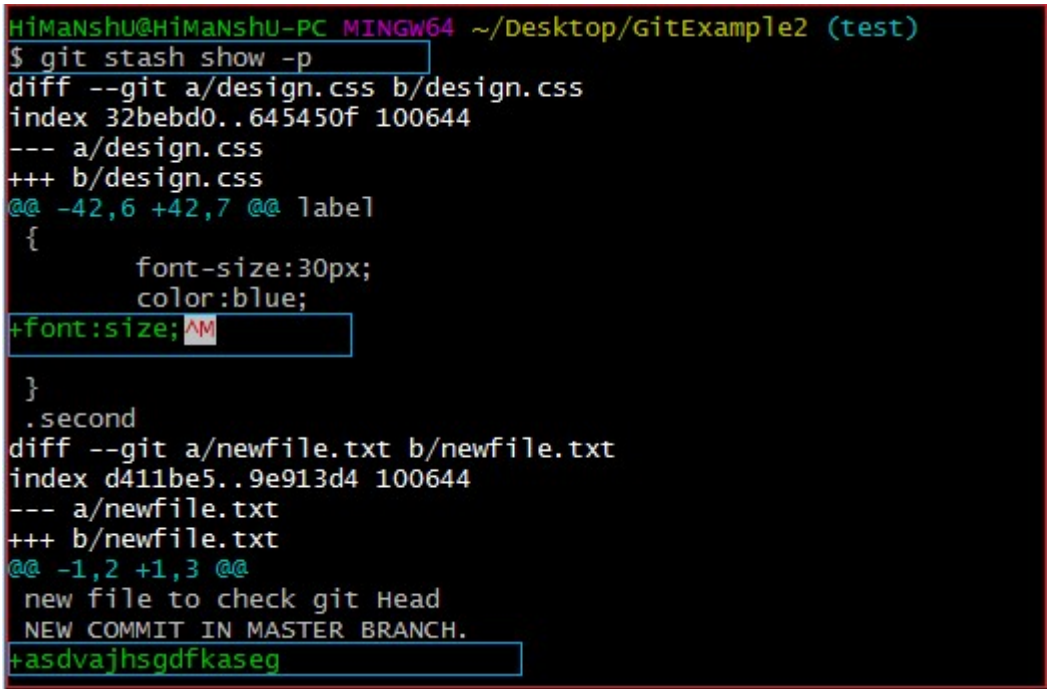
We can exactly track what changes are made on the file. To display the changed content of the file, perform the below command:

Syntax:

1. $ git stash show -p

Here, -p stands for the partial stash. The given command will show the edited files and content, consider the below output:

Output:



The above output is showing the file name with changed content. It acts the same as git diff command. The **git diff** command will also show the exact output.

Git Stash Pop (Reapplying Stashed Changes)

Git allows the user to re-apply the previous commits by using git stash pop command. The popping option removes the changes from stash and applies them to your working file.

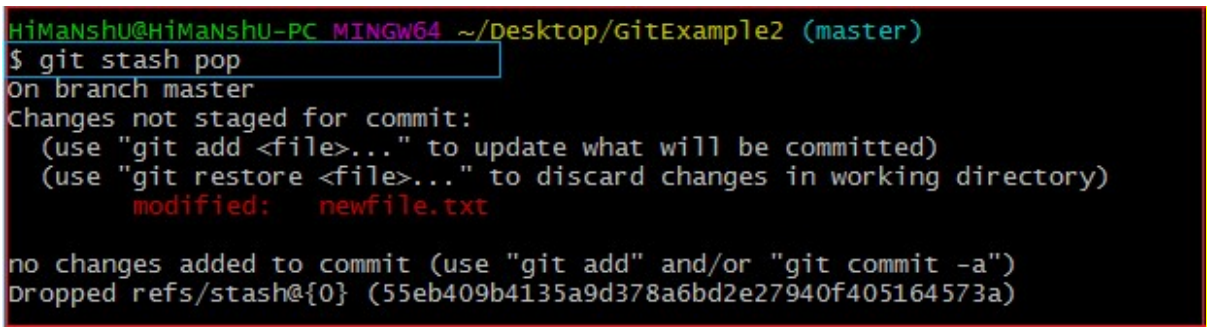
The git stash pop command is quite similar to git stash apply. The main difference between both of these commands is stash pop command that deletes the stash from the stack after it is applied.

Syntax:

1. $ git stash pop

The above command will re-apply the previous commits to the repository. Consider the below output.

Output:



Git Stash Drop (Unstash)

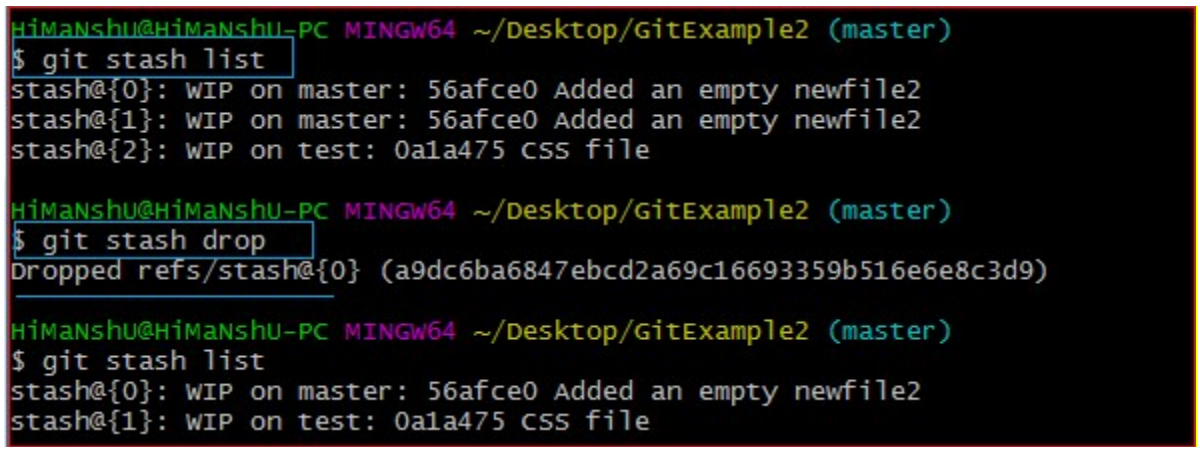
The **git stash drop** command is used to delete a stash from the queue. Generally, it deletes the most recent stash. Caution should be taken before using stash drop command, as it is difficult to undo if once applied.

The only way to revert it is if you do not close the terminal after deleting the stash. The stash drop command will be used as:

Syntax:

1. $ git stash drop

Output:



In the above output, the most recent stash **(stash@{0})** has been dropped from given three stashes. The stash list command lists all the available stashes in the queue.

We can also delete a particular stash from the queue. To delete a particular stash from the available stashes, pass the stash id in stash drop command. It will be processed as:

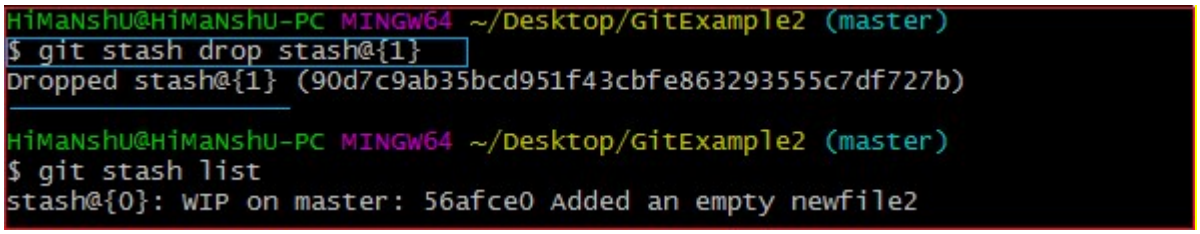
Syntax:

1. $ git stash drop **<stash** id**>**

Assume that I have two stashes available in my queue, and I don't want to drop my most recent stash, but I want to delete the older one. Then, it will be operated as:

1. $ git stash drop stash@{1}

Consider the below output:



In the above output, the commit **stash@{1}** has been deleted from the queue.

Git Stash Clear

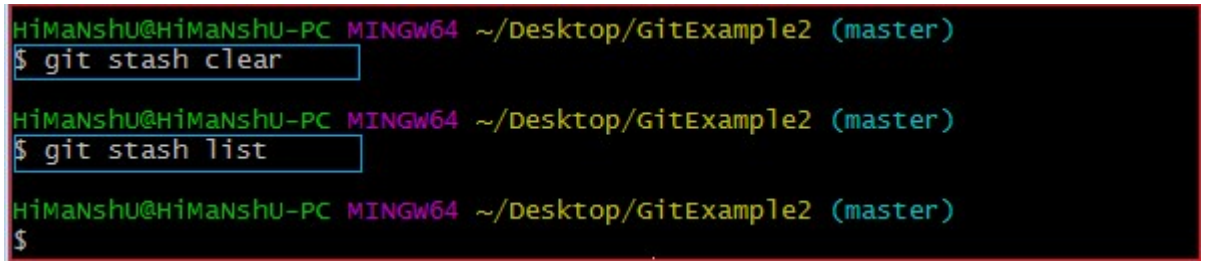
The **git stash clear** command allows deleting all the available stashes at once. To delete all the available stashes, operate below command:

Syntax:

1. $ git stash clear

it will delete all the stashes that exist in the repository.

Output:



All the stashes are deleted in the above output. The git stash list command is blank because there are no stashes available in the repository.

Git Stash Branch

If you stashed some work on a particular branch and continue working on that branch. Then, it may create a conflict during merging. So, it is good to stash work on a separate branch.

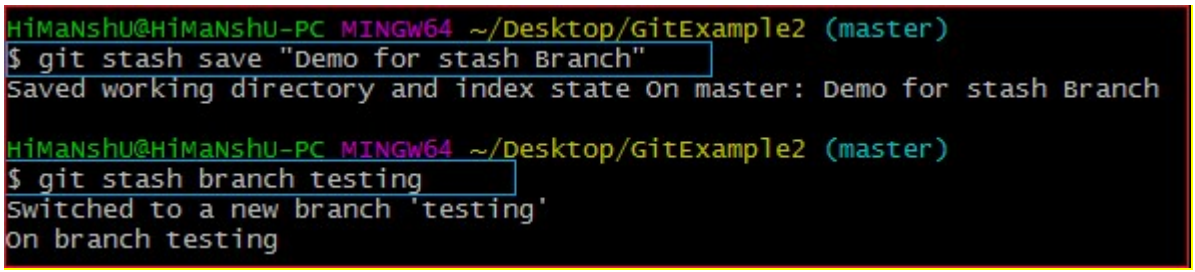
The git stash branch command allows the user to stash work on a separate branch to avoid conflicts. The syntax for this branch is as follows:

Syntax:

1. $ git stash branch **<Branch** Name**>**

The above command will create a new branch and transfer the stashed work on that. Consider the below output:

Output:



In the above output, the stashed work is transferred to a newly created branch testing. It will avoid the merge conflict on the master branch.