Following are the Git commands which are being covered:

* [**git config**](https://www.edureka.co/blog/git-commands-with-example/#git%20config)
* [**git init**](https://www.edureka.co/blog/git-commands-with-example/#git%20init)
* [**git clone**](https://www.edureka.co/blog/git-commands-with-example/#git%20clone)
* [**git add**](https://www.edureka.co/blog/git-commands-with-example/#git%20add)
* [**git commit**](https://www.edureka.co/blog/git-commands-with-example/#git%20commit)
* [**git diff**](https://www.edureka.co/blog/git-commands-with-example/#git%20diff)
* [**git reset**](https://www.edureka.co/blog/git-commands-with-example/#git%20reset)
* [**git status**](https://www.edureka.co/blog/git-commands-with-example/#git%20status)
* [**git rm**](https://www.edureka.co/blog/git-commands-with-example/#git%20rm)
* [**git log**](https://www.edureka.co/blog/git-commands-with-example/#git%20log)
* [**git show**](https://www.edureka.co/blog/git-commands-with-example/#git%20show)
* [**git tag**](https://www.edureka.co/blog/git-commands-with-example/#git%20tag)
* [**git branch**](https://www.edureka.co/blog/git-commands-with-example/#git%20branch)
* [**git checkout**](https://www.edureka.co/blog/git-commands-with-example/#git%20checkout)
* [**git merge**](https://www.edureka.co/blog/git-commands-with-example/#git%20merge)
* [**git remote**](https://www.edureka.co/blog/git-commands-with-example/#git%20remote)
* [**git push**](https://www.edureka.co/blog/git-commands-with-example/#git%20push)
* [**git pull**](https://www.edureka.co/blog/git-commands-with-example/#git%20pull)
* [**git stash**](https://www.edureka.co/blog/git-commands-with-example/#git%20stash)

**Git Commands**

**git config**

**Usage: git config –global user.name “[name]”**

**Usage: git config –global user. email “[email address]”**

This command sets the author name and email address respectively to be used with your commits.

Git Config Command - Git Commands - Edureka

### ****git init****

**Usage: git init [repository name]**

This command is used to start a new repository.

GitInit Command - Git Commands - Edureka

### ****git clone****

**Usage: git clone [url]**

This command is used to obtain a repository from an existing URL.



### ****git add****

**Usage: git add [file]**

This command adds a file to the staging area.

Git Add Command - Git Commands - Edureka

**Usage: git add \***

This command adds one or more to the staging area.

Git Add Command - Git Commands - Edureka

### ****git commit****

**Usage: git commit -m “[Type in the commit message]”**

This command records or snapshots the file permanently in the version history.



**Usage: git commit -a**

This command commits any files you’ve added with the git add command and also commits any files you’ve changed since then.

Git Commit Command - Git Commands - Edureka

**git diff**

**Usage: git diff**

This command shows the file differences which are not yet staged.



**Usage: git diff –staged**

This command shows the differences between the files in the staging area and the latest version present.



**Usage: git diff [first branch] [second branch]**

This command shows the differences between the two branches mentioned.



## 

## ****[Git Commands with Examples](https://www.edureka.co/git-github-sp" \t "_blank)****

### ****[Git reset](https://www.edureka.co/git-github-sp" \t "_blank)****

**[Usage: git reset [file]](https://www.edureka.co/git-github-sp" \t "_blank)**

[This command unstages the file, but it preserves the file contents.](https://www.edureka.co/git-github-sp" \t "_blank)

[](https://www.edureka.co/git-github-sp" \t "_blank)

**[Usage: git reset [commit]](https://www.edureka.co/git-github-sp" \t "_blank)**

[This command undoes all the commits after the specified commit and preserves the changes locally.](https://www.edureka.co/git-github-sp" \t "_blank)

[Git Reset Command - Git Commands - Edureka](https://www.edureka.co/git-github-sp" \t "_blank)

**[Usage: git reset –hard [commit]](https://www.edureka.co/git-github-sp" \t "_blank)**

[This command discards all history and goes back to the specified commit.](https://www.edureka.co/git-github-sp" \t "_blank)

[Git Reset Command - Git Commands - Edureka](https://www.edureka.co/git-github-sp" \t "_blank)

### ****git status****

**Usage: git status**

This command lists all the files that have to be committed.



### ****git rm****

**Usage: git rm [file]**

This command deletes the file from your working directory and stages the deletion.

Git Rm Command - Git Commands - Edureka

### ****git log****

**Usage: git log**

This command is used to list the version history for the current branch.



**Usage: git log –follow [file]**

This command lists version history for a file, including the renaming of files also.



### ****git show****

**Usage: git show [commit]**

This command shows the metadata and content changes of the specified commit.



### ****git tag****

**Usage: git tag [commitID]**

This command is used to give tags to the specified commit.



### ****git branch****

**Usage: git branch**

This command lists all the local branches in the current repository.

Git Branch Command - Git Commands - Edureka

**Usage: git branch [branch name]**

This command creates a new branch.

Git Branch Command - Git Commands - Edureka

**Usage: git branch -d [branch name]**

This command deletes the feature branch.

Git Branch Command - Git Commands - Edureka

### ****git checkout****

**Usage: git checkout [branch name]**

This command is used to switch from one branch to another.

Git Checkout Command - Git Commands - Edureka

**Usage: git checkout -b [branch name]**

This command creates a new branch and also switches to it.

Git Checkout Command - Git Commands - Edureka

### ****git merge****

**Usage: git merge [branch name]**

This command merges the specified branch’s history into the current branch.

Git Merge Command - Git Commands - Edureka

### ****git remote****

**Usage: git remote add [variable name] [Remote Server Link]**

This command is used to connect your local repository to the remote server.

Git Remote Command - Git Commands - Edureka

### ****git push****

**Usage: git push [variable name] master**

This command sends the committed changes of master branch to your remote repository.



**Usage: git push [variable name] [branch]**

This command sends the branch commits to your remote repository.



**Usage: git push –all [variable name]**

This command pushes all branches to your remote repository.



**Usage: git push [variable name] :[branch name]**

This command deletes a branch on your remote repository.



### ****Git pull****

**Usage:  git pull [Repository Link]**

This command fetches and merges changes on the remote server to your working directory.



### ****git stash****

**Usage: git stash save**

This command temporarily stores all the modified tracked files.

Git Stash Command - Git Commands - Edureka

**Usage: git stash pop**

This command restores the most recently stashed files.

**Usage: git stash list**

This command lists all stashed change sets.

**Usage: git stash drop**

**Usage: git stash pop**

This command restores the most recently stashed files.



**Usage: git stash list**

This command lists all stashed changesets.

Git Stash Command - Git Commands - Edureka

**Usage: git stash drop**

This command discards the most recently stashed changeset.

Git Stash Command - Git Commands - Edureka

.

# 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.

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.

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:**

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:**

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:

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:**

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.

rm -f .git/index.lock(removing any lock by running command)

rm –rf .git(remove the git folder)