Top 20 Git Commands With Examples

by Sahiti Kappagantula MVB · Jan. 22, 20 · DevOps Zone · Tutorial

In the previous blog, you got an understanding of what git is. In this blog, I will talk about the Top 20 Git Commands that you will be using frequently while you are working with Git.

Here are the Git commands which are being covered:

- **git config**
- **git init**
- **git clone**
- **git add**
- **git commit**
- **git diff**
- **git reset**
- **git status**
- **git rm**
- **git log**
- **git show**
- **git tag**
- **git branch**
- **git checkout**
- **git merge**

- **git remote**
- **git push**
- **git pull**
- **git stash**

So, let's get started!

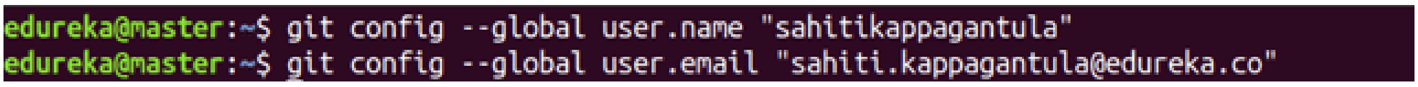
Git Commands

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.

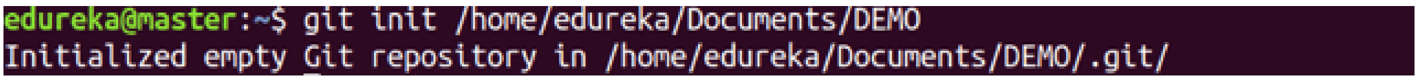


```
edureka@master:~$ git config --global user.name "sahitikappagantula"
edureka@master:~$ git config --global user.email "sahiti.kappagantula@edureka.co"
```

git init

Usage: `git init [repository name]`

This command is used to start a new repository.

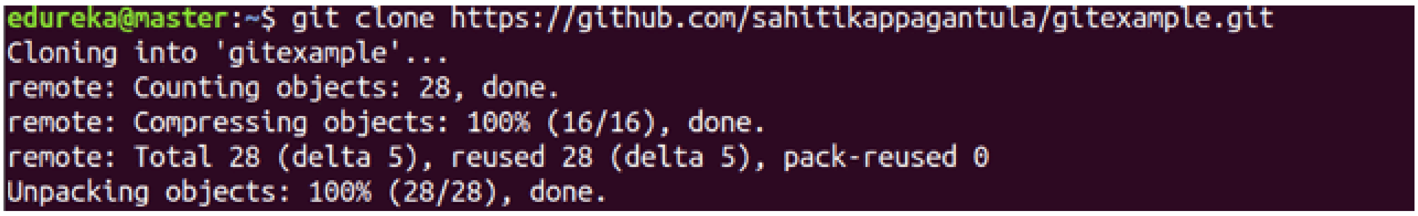


```
edureka@master:~$ git init /home/edureka/Documents/DEMO
Initialized empty Git repository in /home/edureka/Documents/DEMO/.git/
```

git clone

Usage: `git clone [url]`

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



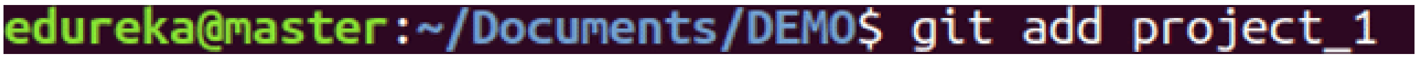
```
edureka@master:~$ git clone https://github.com/sahitikappagantula/gitexample.git
Cloning into 'gitexample'...
remote: Counting objects: 28, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 28 (delta 5), reused 28 (delta 5), pack-reused 0
Unpacking objects: 100% (28/28), done.
```

git add

Usage: `git add [file]`

This command adds a file to the staging area

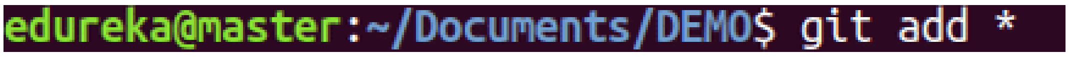
This command adds a file to the staging area.



```
edureka@master:~/Documents/DEMO$ git add project_1
```

Usage: `git add *`

This command adds one or more to the staging area.

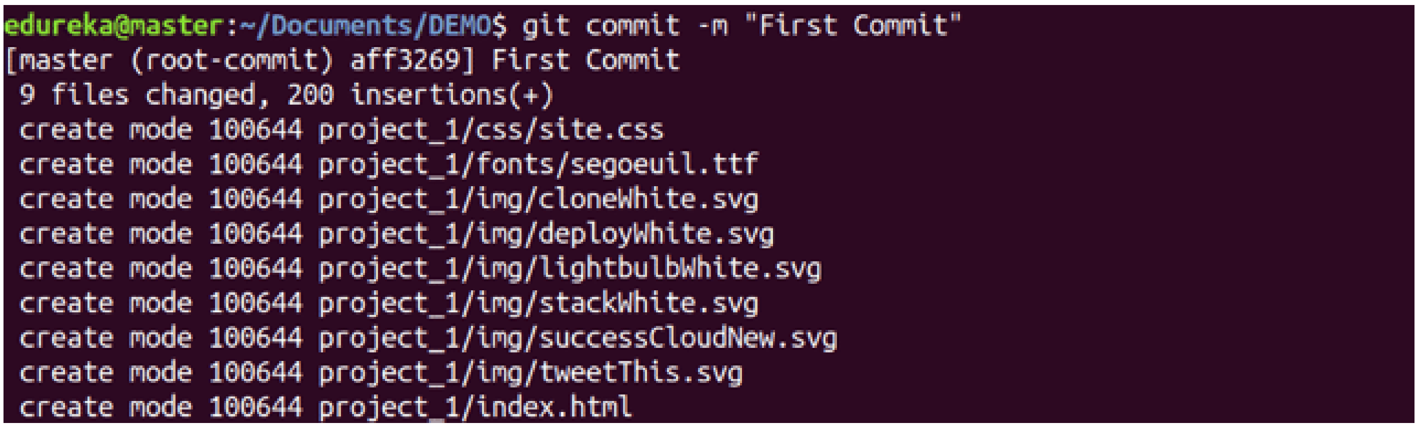


```
edureka@master:~/Documents/DEMO$ git add *
```

git commit

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

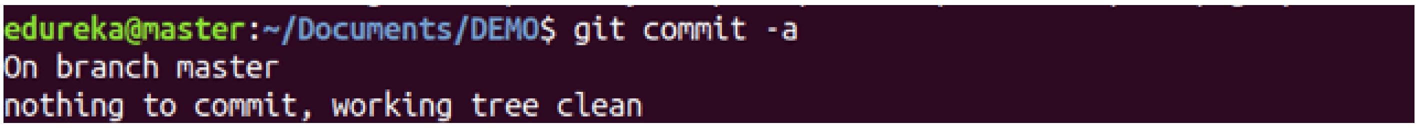
This command records or snapshots the file permanently in the version history.



```
edureka@master:~/Documents/DEMO$ git commit -m "First Commit"
[master (root-commit) aff3269] First Commit
 9 files changed, 200 insertions(+)
 create mode 100644 project_1/css/site.css
 create mode 100644 project_1/fonts/segoeuil.ttf
 create mode 100644 project_1/img/cloneWhite.svg
 create mode 100644 project_1/img/deployWhite.svg
 create mode 100644 project_1/img/lightbulbWhite.svg
 create mode 100644 project_1/img/stackWhite.svg
 create mode 100644 project_1/img/successCloudNew.svg
 create mode 100644 project_1/img/tweetThis.svg
 create mode 100644 project_1/index.html
```

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.

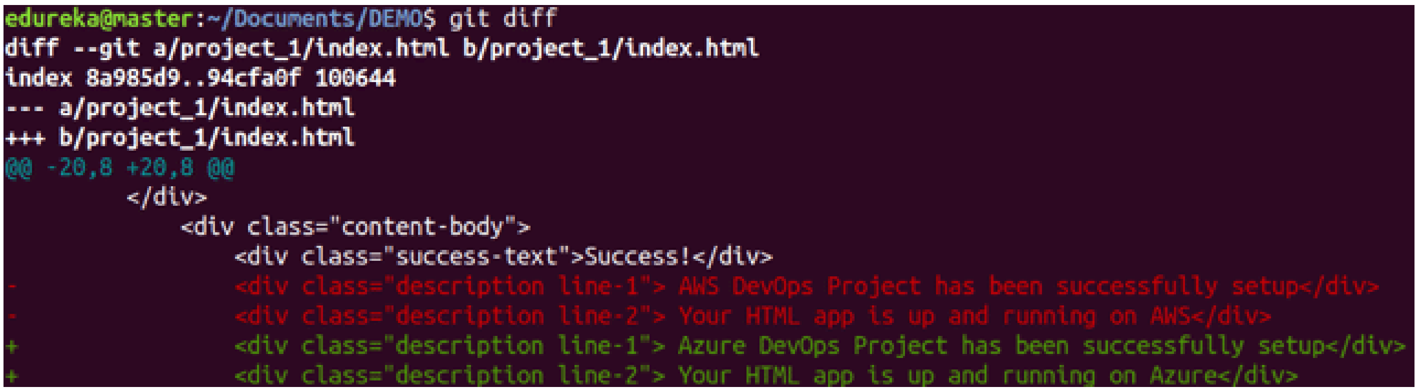


```
edureka@master:~/Documents/DEMO$ git commit -a
On branch master
nothing to commit, working tree clean
```

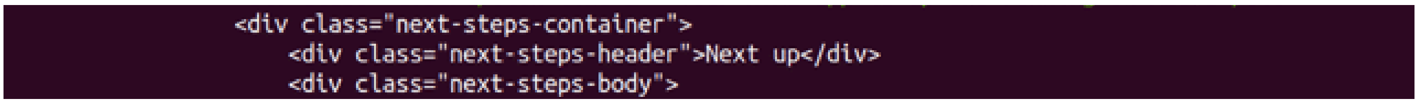
git diff

Usage: `git diff`

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



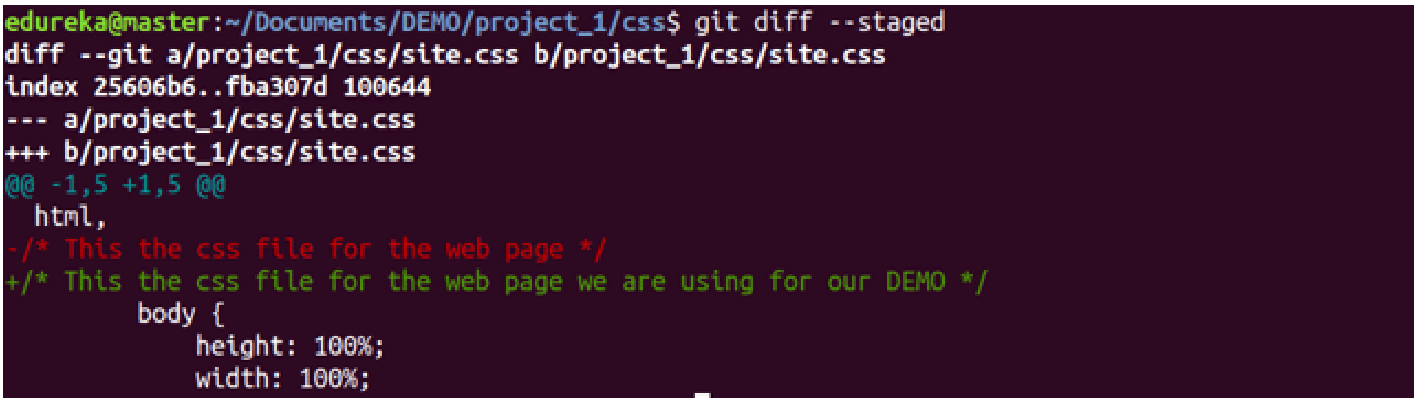
```
edureka@master:~/Documents/DEMO$ git diff
diff --git a/project_1/index.html b/project_1/index.html
index 8a985d9..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -20,8 +20,8 @@
         </div>
             <div class="content-body">
                 <div class="success-text">Success!</div>
-                <div class="description line-1"> AWS DevOps Project has been successfully setup</div>
-                <div class="description line-2"> Your HTML app is up and running on AWS</div>
+                <div class="description line-1"> Azure DevOps Project has been successfully setup</div>
+                <div class="description line-2"> Your HTML app is up and running on Azure</div>
```



```
                <div class="next-steps-container">
                    <div class="next-steps-header">Next up</div>
                    <div class="next-steps-body">
```

Usage: `git diff –staged`

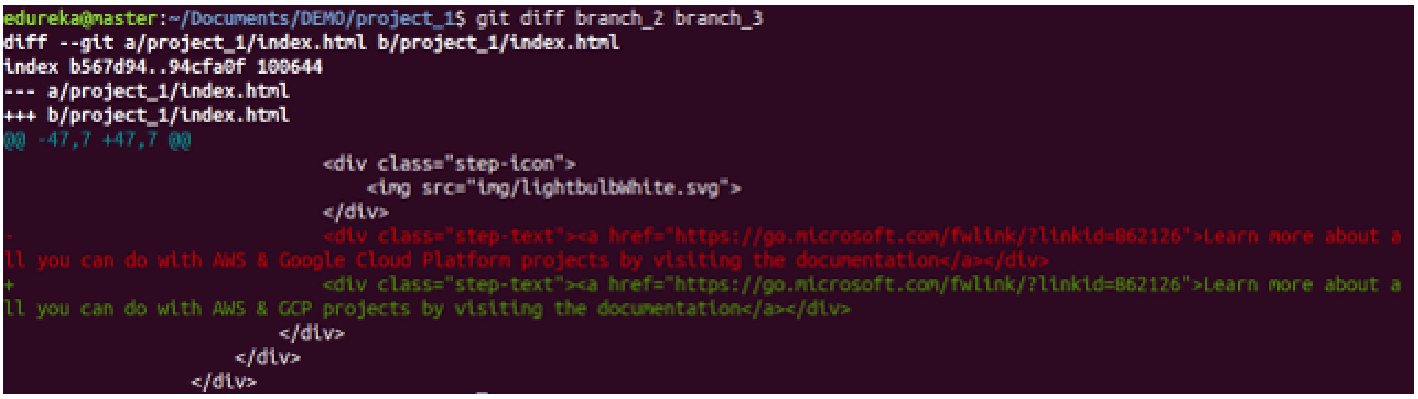
This command shows the differences between the files in the staging area and the latest version present.



```
edureka@master:~/Documents/DEMO/project_1/css$ git diff --staged
diff --git a/project_1/css/site.css b/project_1/css/site.css
index 25606b6..fba307d 100644
--- a/project_1/css/site.css
+++ b/project_1/css/site.css
@@ -1,5 +1,5 @@
  html,
-/* This the css file for the web page */
+/* This the css file for the web page we are using for our DEMO */
         body {
             height: 100%;
             width: 100%;
```

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

This command shows the differences between the two branches mentioned.

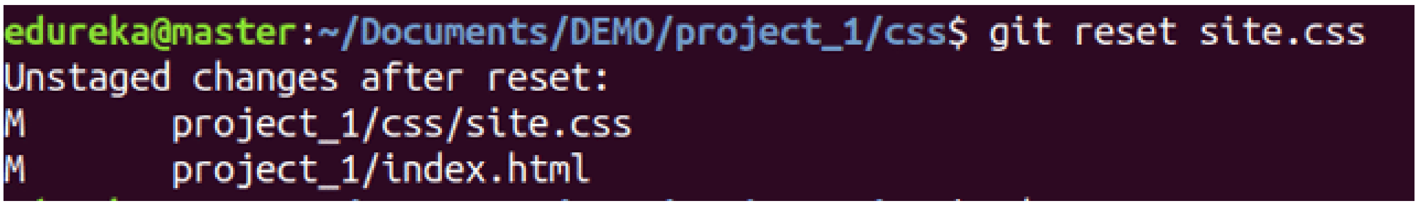


```
edureka@master:~/Documents/DEMO/project_1$ git diff branch_2 branch_3
diff --git a/project_1/index.html b/project_1/index.html
index b567d94..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -47,7 +47,7 @@
                                <div class="step-icon">
                                    <img src="img/lightbulbWhite.svg">
                                </div>
-                               <div class="step-text"><a href="https://go.microsoft.com/fwlink/?linkid=862126">Learn more about a
ll you can do with AWS & Google Cloud Platform projects by visiting the documentation</a></div>
+                               <div class="step-text"><a href="https://go.microsoft.com/fwlink/?linkid=862126">Learn more about a
ll you can do with AWS & GCP projects by visiting the documentation</a></div>
                            </div>
                        </div>
                    </div>
```

git reset

Usage: `git reset [file]`

This command unstages the file, but it preserves the file contents.

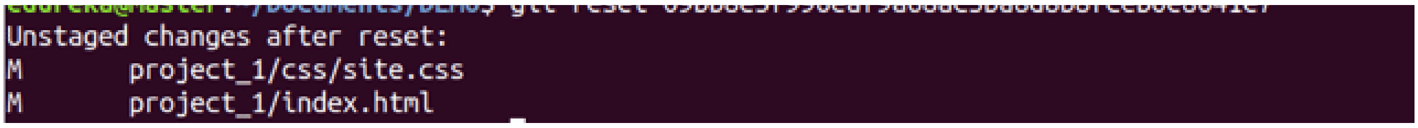


```
edureka@master:~/Documents/DEMO/project_1/css$ git reset site.css
Unstaged changes after reset:
M       project_1/css/site.css
M       project_1/index.html
```

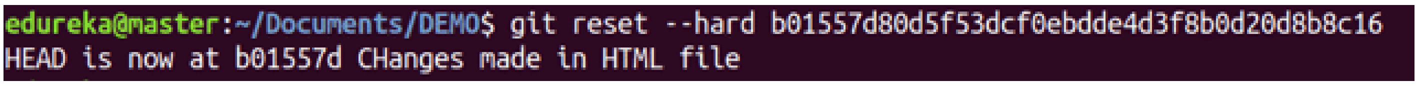
Usage: `git reset [commit]`

This command undoes all the commits after the specified commit and preserves the changes locally.

```
edureka@master:~/Documents/DEMO$ git reset 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7
```


```
Unstaged changes after reset:
M       project_1/css/site.css
M       project_1/index.html
```

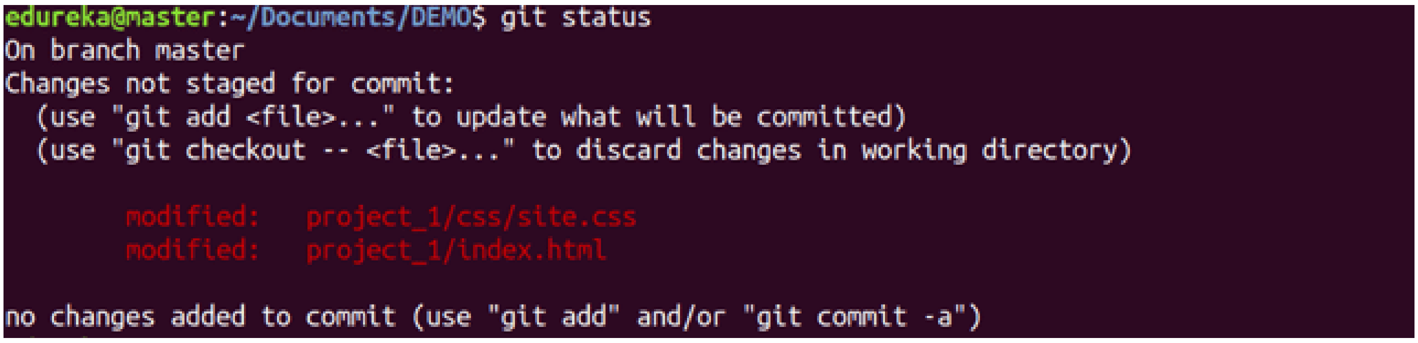
Usage: `git reset -hard [commit]` This command discards all history and goes back to the specified commit.


```
edureka@master:~/Documents/DEMO$ git reset --hard b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
HEAD is now at b01557d CHanges made in HTML file
```

git status

Usage: `git status`

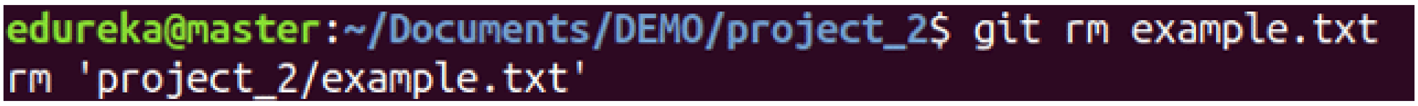
This command lists all the files that have to be committed.


```
edureka@master:~/Documents/DEMO$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   project_1/css/site.css
        modified:   project_1/index.html

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

git rm

Usage: `git rm [file]`

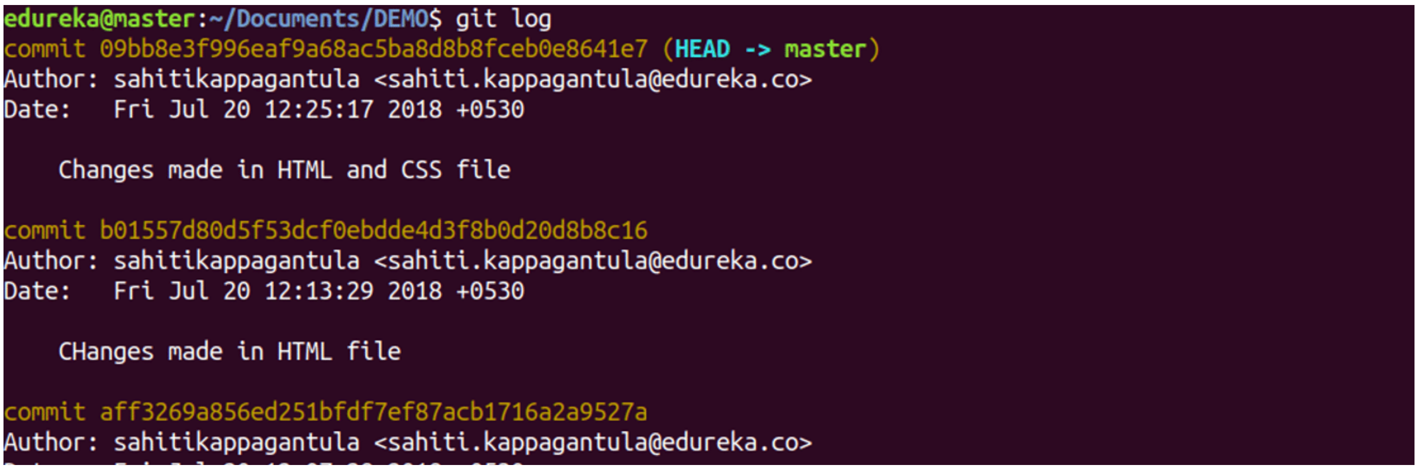
This command deletes the file from your working directory and stages the deletion.


```
edureka@master:~/Documents/DEMO/project_2$ git rm example.txt
rm 'project_2/example.txt'
```

git log

Usage: `git log`

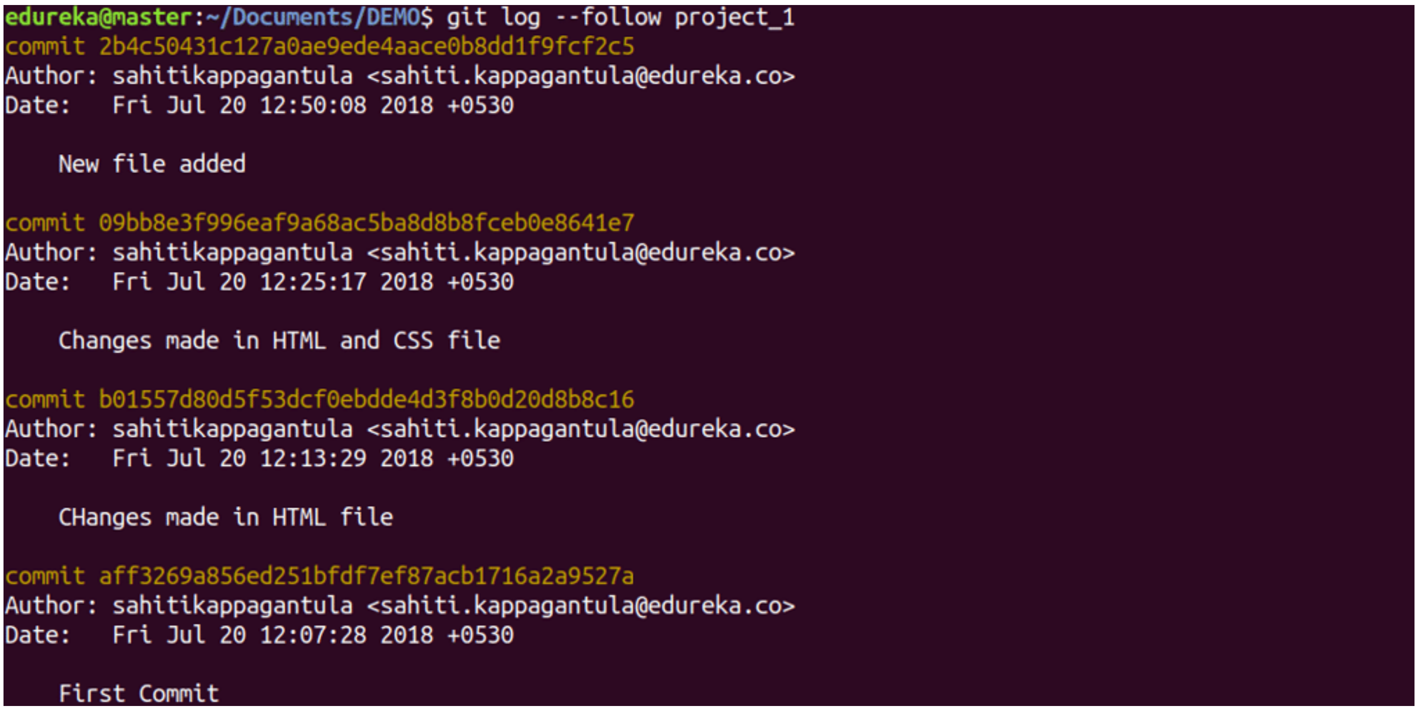
This command is used to list the version history for the current branch.


```
edureka@master:~/Documents/DEMO$ git log
commit 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7 (HEAD -> master)
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:25:17 2018 +0530

    Changes made in HTML and CSS file

commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

commit aff3269a856ed251bfdf7ef87acb1716a2a9527a
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
```

```
Date:   Fri Jul 20 12:07:28 2018 +0530

    First Commit
```

Usage: `git log –follow[file]`

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

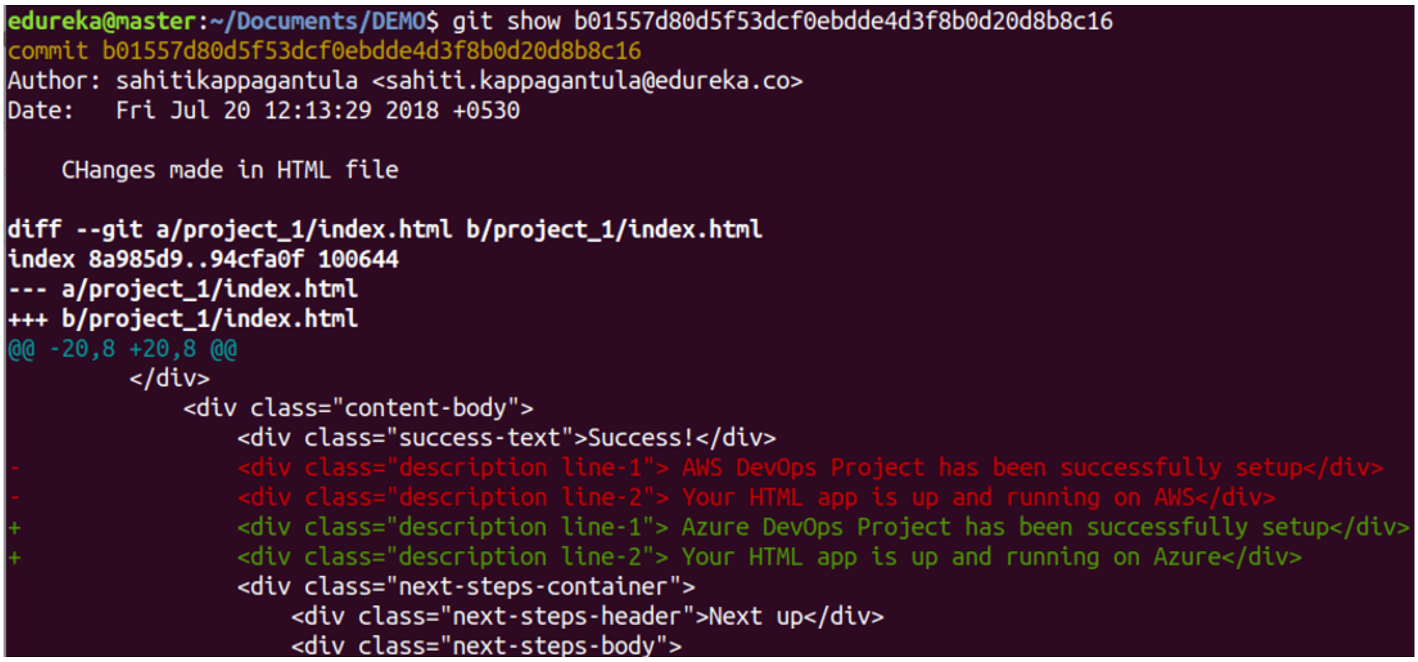


```
edureka@master:~/Documents/DEMO$ git log --follow project_1
commit 2b4c50431c127a0ae9ede4aace0b8dd1f9fcf2c5
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:50:08 2018 +0530

    New file added

commit 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:25:17 2018 +0530

    Changes made in HTML and CSS file

commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

commit aff3269a856ed251bfdf7ef87acb1716a2a9527a
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:07:28 2018 +0530

    First Commit
```

git show

Usage: `git show [commit]`

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

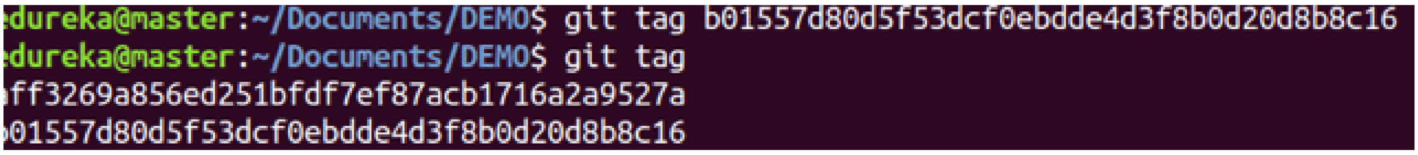


```
edureka@master:~/Documents/DEMO$ git show b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

diff --git a/project_1/index.html b/project_1/index.html
index 8a985d9..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -20,8 +20,8 @@
         </div>
             <div class="content-body">
                 <div class="success-text">Success!</div>
-                <div class="description line-1"> AWS DevOps Project has been successfully setup</div>
-                <div class="description line-2"> Your HTML app is up and running on AWS</div>
+                <div class="description line-1"> Azure DevOps Project has been successfully setup</div>
+                <div class="description line-2"> Your HTML app is up and running on Azure</div>
                 <div class="next-steps-container">
                     <div class="next-steps-header">Next up</div>
                     <div class="next-steps-body">
```

git tag

Usage: `git tag [commitID]`

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

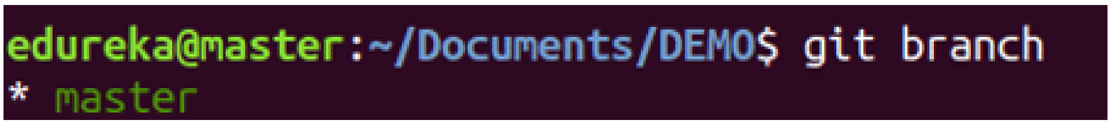


```
edureka@master:~/Documents/DEMO$ git tag b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
edureka@master:~/Documents/DEMO$ git tag
ff3269a856ed251bfdf7ef87acb1716a2a9527a
01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
```

git branch

Usage: `git branch`

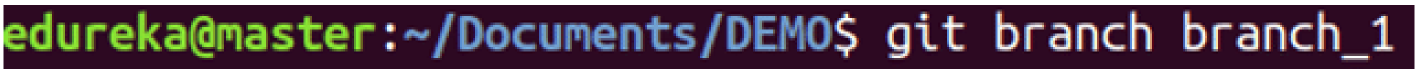
This command lists all the local branches in the current repository.



```
edureka@master:~/Documents/DEMO$ git branch
* master
```

Usage: `git branch [branch name]`

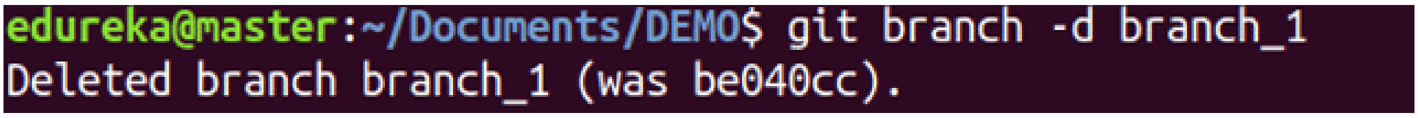
This command creates a new branch.



```
edureka@master:~/Documents/DEMO$ git branch branch_1
```

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

This command deletes the feature branch.

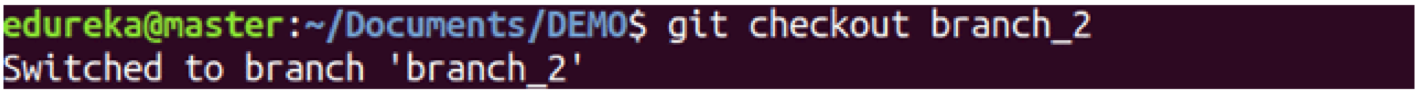


```
edureka@master:~/Documents/DEMO$ git branch -d branch_1
Deleted branch branch_1 (was be040cc).
```

git checkout

Usage: `git checkout [branch name]`

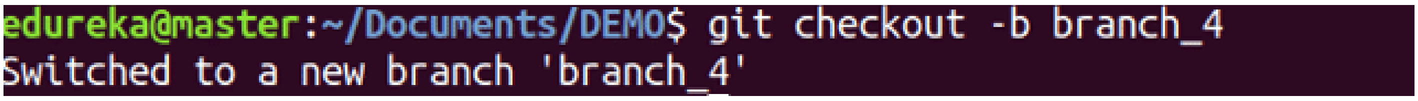
This command is used to switch from one branch to another.



```
edureka@master:~/Documents/DEMO$ git checkout branch_2
Switched to branch 'branch_2'
```

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

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

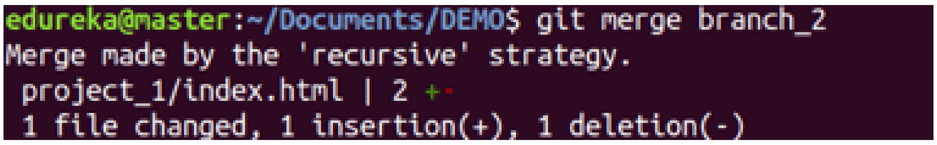


```
edureka@master:~/Documents/DEMO$ git checkout -b branch_4
Switched to a new branch 'branch_4'
```

git merge

Usage: `git merge [branch name]`

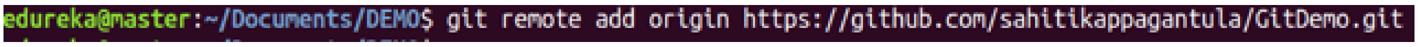
This command merges the specified branch’s history into the current branch.


```
edureka@master:~/Documents/DEMO$ git merge branch_2
Merge made by the 'recursive' strategy.
 project_1/index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

git remote

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

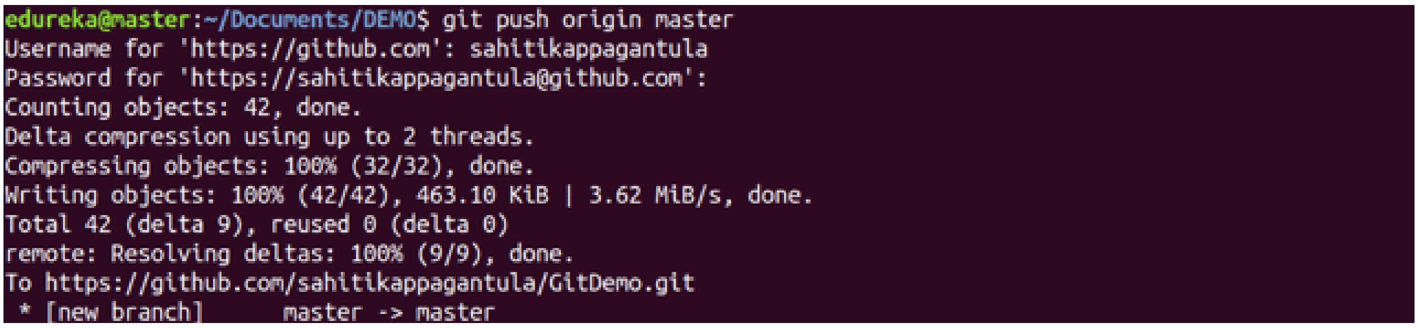
This command is used to connect your local repository to the remote server.


```
edureka@master:~/Documents/DEMO$ git remote add origin https://github.com/sahitikappagantula/GitDemo.git
```

git push

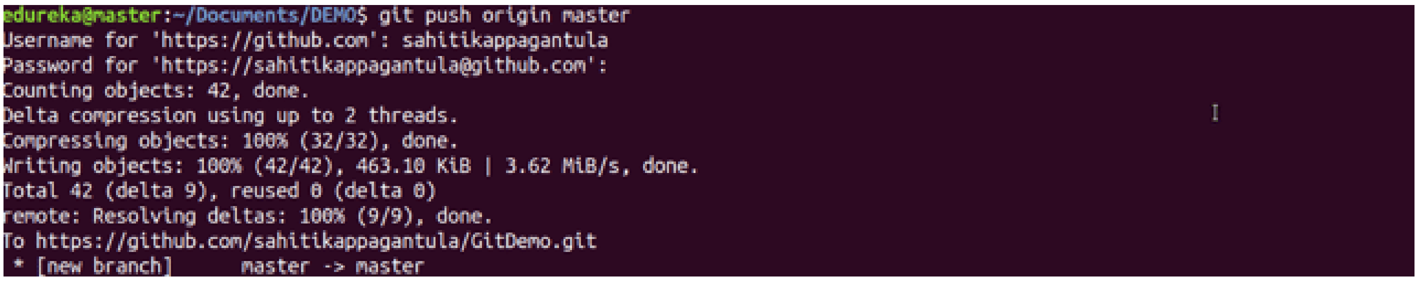
Usage: `git push [variable name] master`

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


```
edureka@master:~/Documents/DEMO$ git push origin master
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Counting objects: 42, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (42/42), 463.10 KiB | 3.62 MiB/s, done.
Total 42 (delta 9), reused 0 (delta 0)
remote: Resolving deltas: 100% (9/9), done.
To https://github.com/sahitikappagantula/GitDemo.git
 * [new branch]      master -> master
```

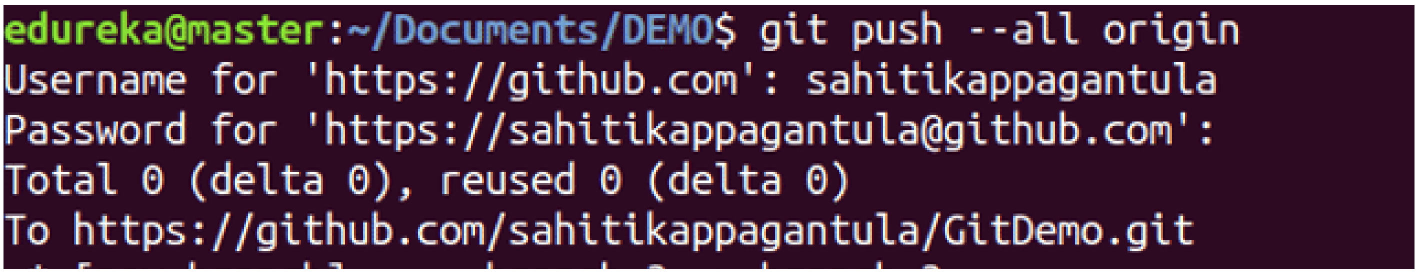
Usage: `git push [variable name] [branch]`

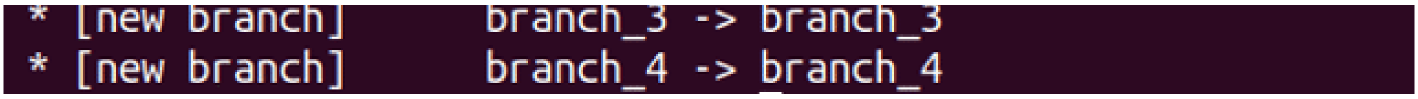
This command sends the branch commits to your remote repository.


```
edureka@master:~/Documents/DEMO$ git push origin master
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Counting objects: 42, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (42/42), 463.10 KiB | 3.62 MiB/s, done.
Total 42 (delta 9), reused 0 (delta 0)
remote: Resolving deltas: 100% (9/9), done.
To https://github.com/sahitikappagantula/GitDemo.git
 * [new branch]      master -> master
```

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

This command pushes all branches to your remote repository.

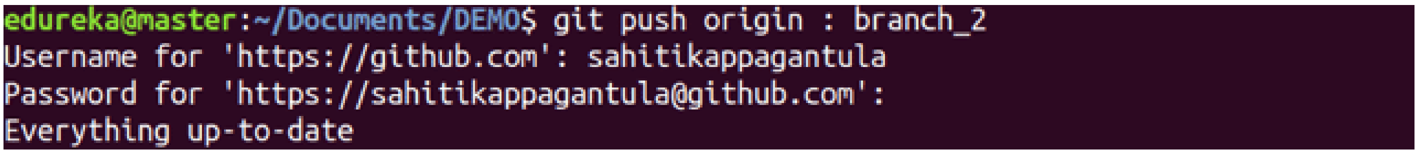

```
edureka@master:~/Documents/DEMO$ git push --all origin
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/sahitikappagantula/GitDemo.git
```



```
 * [new branch]      branch_3 -> branch_3
 * [new branch]      branch_4 -> branch_4
```

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

This command deletes a branch on your remote repository.

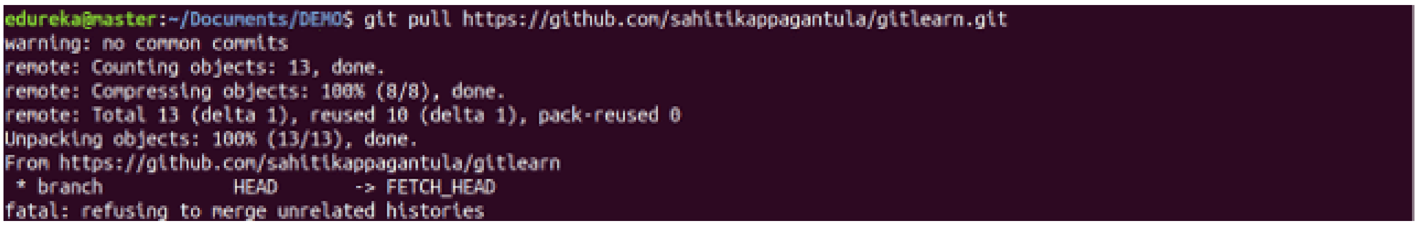


```
edureka@master:~/Documents/DEMO$ git push origin : branch_2
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Everything up-to-date
```

git pull

Usage: `git pull [Repository Link]`

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

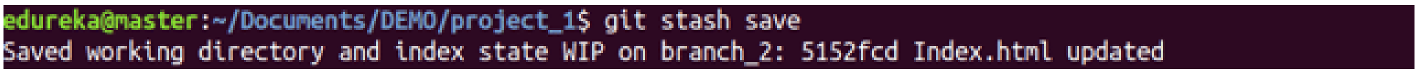


```
edureka@master:~/Documents/DEMO$ git pull https://github.com/sahitikappagantula/gitlearn.git
warning: no common commits
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 1), reused 10 (delta 1), pack-reused 0
Unpacking objects: 100% (13/13), done.
From https://github.com/sahitikappagantula/gitlearn
 * branch            HEAD       -> FETCH_HEAD
fatal: refusing to merge unrelated histories
```

git stash

Usage: `git stash save`

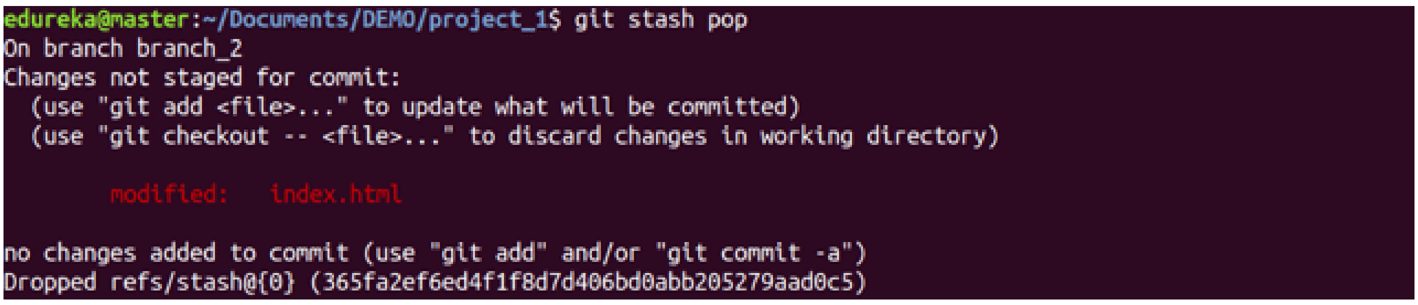
This command temporarily stores all the modified tracked files.



```
edureka@master:~/Documents/DEMO/project_1$ git stash save
Saved working directory and index state WIP on branch_2: 5152fcd Index.html updated
```

Usage: `git stash pop`

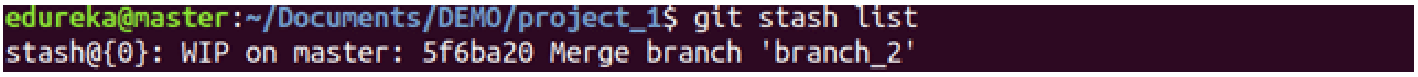
This command restores the most recently stashed files.



```
edureka@master:~/Documents/DEMO/project_1$ git stash pop
On branch branch_2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (365fa2ef6ed4f1f8d7d406bd0abb205279aad0c5)
```

Usage: `git stash list`

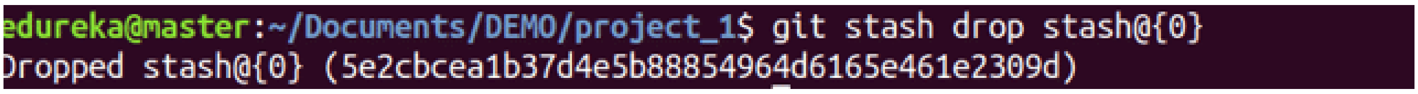
This command lists all stashed changesets.



```
edureka@master:~/Documents/DEMO/project_1$ git stash list
stash@{0}: WIP on master: 5f6ba20 Merge branch 'branch_2'
```

Usage: `git stash drop`

This command discards the most recently stashed changeset.



```
edureka@master:~/Documents/DEMO/project_1$ git stash drop stash@{0}
Dropped stash@{0} (5e2cbcea1b37d4e5b88854964d6165e461e2309d)
```

Want to learn more about git commands? Here is a Git Tutorial to get you started. Alternatively, you can take a top-down approach and start with this DevOps Tutorial.

Like This Article? Read More From DZone

How to git squash in 6 Steps

Git Commands Tutorial - Part 2

Git Commands Tutorial - Part 1

Free DZone Refcard
Getting Started With Feature Flags

Topics: OPEN SOURCE , GIT , GIT COMMANDS , COMMAND EXAMPLES

DZone > DevOps Zone > 9 Reasons Why You Need A Software Test Management Tool

9 Reasons Why You Need A Software Test Management Tool

by Angela Sabeion · Apr 20, 20 · DevOps Zone · Opinion by Angela Sabejon · Apr 20, 20 · DevOps Zone · Opinion

The dawn of new technology opened a number opportunities for everyone. It helped us accomplish tasks more efficiently. The use of automation and robotics made it possible to execute complicated work in a shorter time frame.

For a long time, software development companies took several weeks to months before they could complete the dry run, bug identification, and troubleshooting of new software. The introduction of more modern approaches such as Agile and DevOps has changed this, and now, companies are required to release faster. It is precisely where software test management tools can help. With the help of a test management tool, companies can quickly come up with a conclusion as to whether or not their software works and what are the areas that need fixing.

To provide you a thorough explanation of why you need a software test management tool, here are nine reasons that you should know.

1. Upgrade Software Quality

All software developers have the same goal, which is to develop and offer software that could satisfy the customer's requirements. These management tools are a vital support for testing to ensure that your software has high quality. According to Forbes, software testing is essential to pinpoint possible errors and check features that may not work as planned. For example, a software test management tool can help identify where you can find the bug that is causing errors.

2. Adaptable Environment

Similar to other innovations, software development needs constant adaptation and enhancement of functions based on client needs. For it to be possible, all software must have updates in the backend for a satisfying client experience. Software test management tools often offer various options, including the number of test cases, test runs, and project inclusions, which makes it possible for software developers to create an adaptive environment for their software.

3. Handles Users and Access Permissions

Just like any other company, software developers also have various departments that get involved in forming a single software. It may include designers, programmers, content creators, coders, testers, and many more. There are specific areas in the development process that do not need to be accessed by particular departments. It is where a powerful software test management tool becomes more useful. A test management tool can handle and manage access permissions that allow the project manager to choose which users have access to significant areas of software development.

4. Safeguard the Testing Data

With the use of a software test management tool, you can rest knowing that all data processed in the software will be secure. Through proper management of users and access permissions, not all can get information from the cloud, especially the sensitive stuff. This feature is essential as online scams are rampant today. Some tools also have certifications that can indicate their security standards. And, it is recommended that you'll ask a potential vendor if they hold such accreditation.

5. Improves Team Productivity

Team productivity is vital for every company. It is because it has a considerable impact on the output and sales of a company. A productive team also creates a positive outlook on each of its members. To improve your team productivity, each one should be working in sync with the other. If this scenario happens, the team can complete tasks efficiently and effectively.

6. Lowering the Risk for Repetitive Work

A task that is done repeatedly without results can be taxing. More often than not, it occurs when one person does not know that another is doing the same. A test management tool can help to lower the risk of people doing repetitive work. For example, when there is a bug in the software, the software test management tool directs the task to the person whose role is specified to troubleshoot errors. Still, it will provide updates to parties involved, but it will prevent duplication of work, which can save time and effort.

7. Compatibility with Testing Platforms

Software test management tools should be able to adapt and incorporate well with testing platforms. It is to provide more accurate test results, run trackers, assist in bug detection, and many more. With the recent updates on test management tools and several testing platforms in the market such as Jira and Slack, you can do integration without much effort.

8. Real-time Testing

Software development is not a one-time process. It needs a considerable amount of time and effort in testing, repetitive assessment, plus continuous upgrade and troubleshooting when an error arises. In such cases, developers need real-time test reports to accurately assess, analyze, and act on whatever error they may encounter. Delayed intervention can cause more damage. Therefore, a software test management tool that can provide real-time testing and generate results right after is a necessity.

9. Role and Task Organization

Each department in a testing team that has involvement in software building and development needs proper organization and role assignment. It is to manage all information and tasks that the manager needs to disseminate for every department. Most software test management tools have a dashboard that permits software developers to put essential data into one place. A panel helps everyone to have access to critical updates, task assignments, and which projects are currently ongoing. It assists project managers in managing people and monitor work performances.

Takeaway

From improving software quality to monitoring the developer team's work performance, a software test management tool is the best decision that you will ever have this year. It is one of your ultimate project management tools that can help you to assess, analyze, and evaluate your software.

Having a testing tool such as this will be a big help to your team as it can check errors, which means you can fix them even before your software is out in the market. It helps you upgrade your software quality and safeguard your testing data. An excellent test management tool can also improve your team's productivity as it effectively supports the allocation and execution of tasks, which decreases the risk for repetitive work.

Topics: SOFTWARE, TEST MANAGEMENT TOOL, SOFTWARE TESTING TOOLS

Topics: SOFTWARE, TEST MANAGEMENT TOOL, SOFTWARE TESTING TOOLS,
PROJECT MANAGEMENT TOOL, TEST MANAGEMENT, QA TESTING TOOLS