What Is Git? – Explore A Distributed Version Control Tool

Reshma Ahmed, Last updated on May 22,2019

Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005 to develop Linux Kernel. Git has the functionality, performance, security and flexibility that most teams and individual developers need. It also serves as an important distributed version-control **DevOps tool**. This 'What Is Git' blog is the first blog of my Git Tutorial series. I hope you will enjoy it. :-)

In this 'What is Git' blog, you will learn:

- Why Git came into existence?
- What is Git?
- Features of Git
- How Git plays a vital role in DevOps?
- How Microsoft and other companies are using Git

What is Git – Why Git Came Into Existence?

We all know "Necessity is the mother of all inventions". And similarly, Git was also invented to fulfill certain necessities that the developers faced before Git. So, let us take a step back to learn all about Version Control Systems (VCS) and how Git came into existence.

Version Control is the management of changes to documents, computer programs, large websites and other collection of information.

There are two types of VCS:

- Centralized Version Control System (CVCS)
- Distributed Version Control System (DVCS)

Centralized VCS

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

Please refer to the diagram below to get a better idea of CVCS:

Centralized version control system
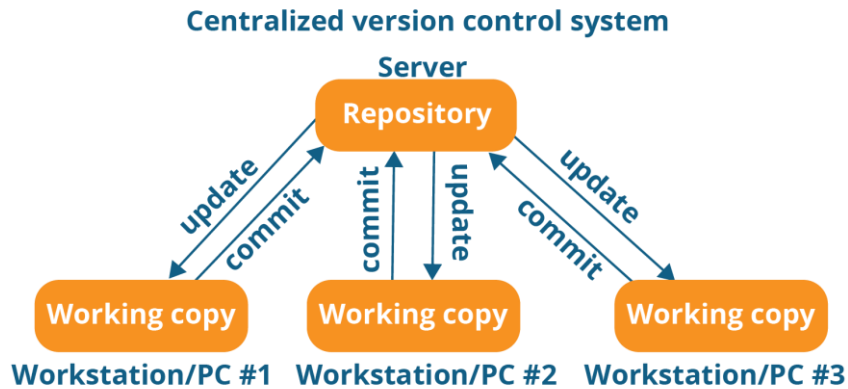
The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.

Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.

Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

- It is not locally available; meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.
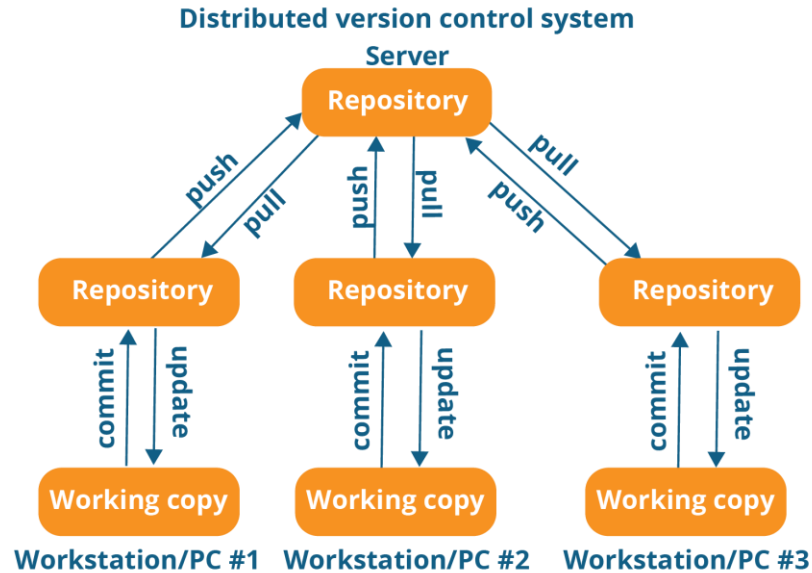
This is when Distributed VCS comes to the rescue.

Distributed VCS

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

You will understand it better by referring to the diagram below:

Distributed version control system

Server

Repository

push / pull

push / pull

pull

push

Repository   **Repository**   **Repository**

commit / update   commit / update   commit / update

Working copy   **Working copy**   **Working copy**

Workstation/PC #1   Workstation/PC #2   Workstation/PC #3

As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called "**pull**" and affect changes to the main repository by an operation called "**push**" from their local repository.

The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an Internet connection.
- Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.
- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
- If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.

After knowing Distributed VCS, its time we take a dive into what is Git.

What Is Git?

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software. Before you go ahead, check out this video on GIT which will give you better in-sight.

Git provides with all the Distributed VCS facilities to the user that was mentioned earlier. Git repositories are very easy to find and access. You will know how flexible and compatible Git is with your system when you go through the features mentioned below:

What is Git – Features of Git

Free and open source:
Git is released under GPL's (General Public License) open source license. You don't need to purchase Git. It is absolutely free. And since it is open source, you can modify the source code as per your requirement.

Speed:
Since you do not have to connect to any network for performing all operations, it completes all the tasks really fast. Performance tests done by Mozilla showed it was an order of magnitude faster than other version control systems. Fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server. The core part of Git is written in C, which avoids runtime overheads associated with other high level languages.

Scalable:
Git is very scalable. So, if in future, the number of collaborators increase Git can easily handle this change. Though Git represents an entire repository, the data stored on the client's side is very small as Git compresses all the huge data through a lossless compression technique.

Reliable:
Since every contributor has its own local repository, on the events of a system crash, the lost data can be recovered from any of the local repositories. You will always have a backup of all your files.

Secure:
Git uses the **SHA1** (Secure Hash Function) to name and identify objects within its repository. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed.

Economical:
In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

Supports non-linear development:
Git supports rapid branching and merging and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers. Branches in Git are very lightweight. A branch in Git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.

Easy Branching:
Branch management with Git is very simple. It takes only few seconds to create, delete, and merge branches. Feature branches provide an isolated environment for every change to your codebase. When a developer wants to start working on something, no matter how big or small, they create a new branch. This ensures that the master branch always contains production-quality code.

Distributed development:
Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches and can be merged in the same way as a locally developed branch.

Compatibility with existing systems or protocol
Repositories can be published via http, ftp or a Git protocol over either a plain socket, or ssh. Git also has a Concurrent Version Systems (CVS) server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Apache SubVersion (SVN) and SVK repositories can be used directly with Git-SVN.

What is Git – Role of Git in DevOps?

Now that you know what is Git, you should know Git is an integral part of DevOps.

DevOps is the practice of bringing agility to the process of development and operations. It's an entirely new ideology that has swept IT organizations worldwide, boosting project life-cycles and in turn increasing profits. DevOps promotes communication between development engineers and operations, participating together in the entire service life-cycle, from design through the development process to production support.

The diagram below depicts the Devops life cycle and displays how Git fits in Devops.
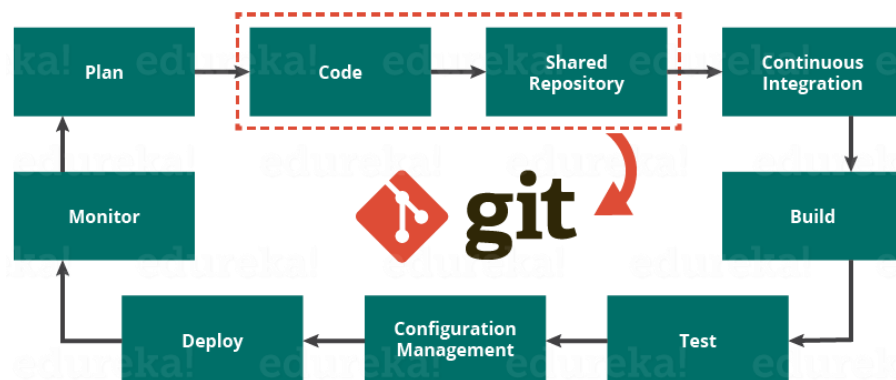

The diagram above shows the entire life cycle of Devops starting from planning the project to its deployment and monitoring. Git plays a vital role when it comes to managing the code that the collaborators contribute to the shared repository. This code is then extracted for performing continuous integration to create a build and test it on the test server and eventually deploy it on the production.

Tools like Git enable communication between the development and the operations team. When you are developing a large project with a huge number of collaborators, it is very important to have

communication between the collaborators while making changes in the project. Commit messages in Git play a very important role in communicating among the team. The bits and pieces that we all deploy lies in the Version Control system like Git. To succeed in DevOps, you need to have all of the communication in Version Control. Hence, Git plays a vital role in succeeding at DevOps.

Companies Using Git

Git has earned way more popularity compared to other version control tools available in the market like Apache Subversion(SVN), Concurrent Version Systems(CVS), Mercurial etc. You can compare the interest of Git by time with other version control tools with the graph collected from *Google Trends* below:
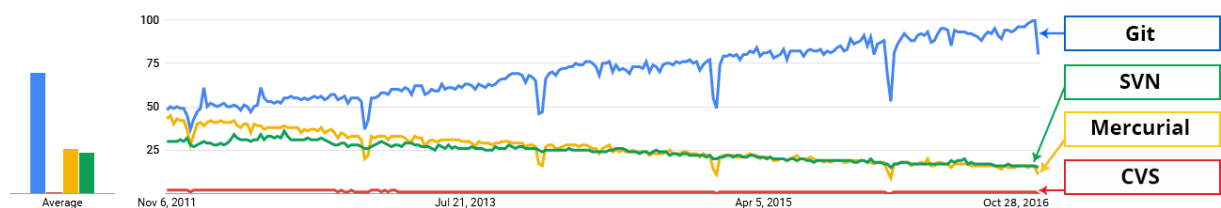
In large companies, products are generally developed by developers located all around the world. To enable communication among them, Git is the solution.

Some companies that use Git for version control are: Facebook, Yahoo, Zynga, Quora, Twitter, eBay, Salesforce, Microsoft and many more.

Lately, all of Microsoft's new development work has been in Git features. Microsoft is migrating .NET and many of its open source projects on GitHub which are managed by Git.
One of such projects is the LightGBM. It is a fast, distributed, high performance gradient boosting framework based on decision tree algorithms which is used for ranking, classification and many other machine learning tasks.

Here, Git plays an important role in managing this distributed version of LightGBM by providing speed and accuracy.

Top 20 Git Commands with Examples

Now that you (presumably) know what Git is and how it works, take a look at examples of how to use the top 20 Git commands.

Sahiti Kappagantula, Jul. 24, 18

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.

```
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
b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
```

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.