SESSION 1

Q1. What is a “version control System”?
Version control systems are a category of software tools that helps record changes to files by keeping a track of modifications done to the code.

- A repository: It can be thought as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- Copy of Work (sometimes called as checkout): It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.

Q2. What are the types Version Control Systems:

Types of Version Control Systems:

- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository and each user gets their own working copy. You need to commit to reflecting your changes in the repository. It is possible for others to see your changes by updating.
Two things are required to make your changes visible to others which are:

- You commit
- They update

The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.
It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven’t been kept? You lose absolutely everything.
Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get other’s changes unless you have first pulled those changes into your repository.
To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, Mercurial. They help us overcome the problem of single point of failure.

Q3. What is the Purpose of Version Control system

- Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
- It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- It integrates the work that is done simultaneously by different members of the team. In some rare case, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
- Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

Q4. Mention the various Git repository hosting functions.

Github

Gitlab

Bitbucket

SourceForge

GitEnterprise

Q5. What is the use of SVN?

Subversion is an open-source versioning system. It keeps the repository of every change made to code files or any other files in a system. This system also manages the additions or deletions made. Use of this system is that it provides all the details of a person who made changes to source code. Useful in projects where multiple people work on the same code base.

Q6. Explain the version control tool git.

Git is one of the best version control tools that is available in the present market.

Features

- Provides strong support for non-linear development.
- Distributed repository model.
- Compatible with existing systems and protocols like HTTP, FTP, ssh.
- Capable of efficiently handling small to large sized projects.
- Cryptographic authentication of history.
- Pluggable merge strategies.
- Toolkit-based design.
- Periodic explicit object packing.
- Garbage accumulates until collected.

Pros

- Super-fast and efficient performance.
- Cross-platform
- Code changes can be very easily and clearly tracked.
- Easily maintainable and robust.
- Offers an amazing command line utility known as git bash.
- Also offers GIT GUI where you can very quickly re-scan, state change, sign off, commit & push the code quickly with just a few clicks.

Cons

- Complex and bigger history log become difficult to understand.
- Does not support keyword expansion and timestamp preservation.

Open Source: Yes
Cost: Free

Q7. Explain the version control tool CVS.

It is yet another most popular revision control system. CVS has been the tool of choice for a long time.

Features

- Client-server repository model.
- Multiple developers might work on the same project parallelly.
- CVS client will keep the working copy of the file up-to-date and requires manual intervention only when an edit conflict occurs
- Keeps a historical snapshot of the project.
- Anonymous read access.
- 'Update' command to keep local copies up to date.
- Can uphold different branches of a project.
- Excludes symbolic links to avoid a security risk.
- Uses delta compression technique for efficient storage.

Pros

- Excellent cross-platform support.
- Robust and fully-featured command-line client permits powerful scripting
- Helpful support from vast CVS community
- allows good web browsing of the source code repository
- It's a very old, well known & understood tool.
- Suits the collaborative nature of the open-source world splendidly.

Cons
- No integrity checking for source code repository.
- Does not support atomic check-outs and commits.
- Poor support for distributed source control.
- Does not support signed revisions and merge tracking.

Open Source: Yes
Cost: Free

Q8. Explain the version control tool RCS.

Revision Control system (RCS), developed by Thien-Thi Nguyen works on the local repository model and supports Unix-like platforms. RCS is a very old tool and was first released in 1982. It is an early version of VCS(Version Control System).

Features:
- Was originally intended for programs, but, is also helpful for text documents or config files that often get revised.
- RCS can be considered as a set of Unix Commands that permits various users to build and maintain program code or documents.
- Allows revision of documents, committing changes and merging docs together.
- Store revisions in a tree structure.

Pros
- Simple architecture
- Easy to work with
- It has local repository model, so the saving of revisions is independent of the central repository.

Cons
- Less security, version history is editable.
- At a time, only one user can work on the same file.

Open Source: Yes
Cost: Free

Q9. Explain the version control tool TFS.

TFS, an acronym for team foundation server is a version control product by Microsoft. It is based on client-server, distributed repository model and has a proprietary license. It provides Windows, cross-platform OS support through Visual Studio Team Services (VSTS).

Features
- Provides entire application lifecycle support including source code management, project management, reporting, automated builds, testing, release management and requirement management.
- Empowers DevOps capabilities.
- Can be used as a backend for several IDEs.
- Available in two different forms (on-premises and online (known as VSTS)).

Pros

- Easy administration. Familiar interfaces and tight integration with other Microsoft products.
- Allows continuous integration, the team builds and unit test integration.
- Great support for branching and merging operations.
- Custom check-in policies to aid in implementing a steady & stable codebase in your source control.

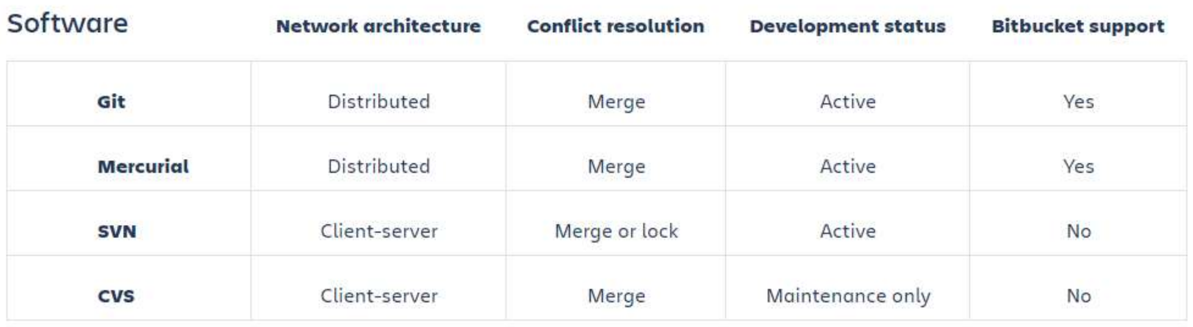
Cons

- Frequent merge conflicts.
- Connection to the central repository is always required.
- Quite slow in performing a pull, check-in, and branching operations.

Open Source: No

Cost: Free of cost for up to 5 users in the VSTS or for open source projects via codeplex.com; else paid and licensed through MSDN subscription or direct buy.

Q10.Difference between git, mercurial, SVN, CVS version control tool.



Software	Network architecture	Conflict resolution	Development status	Bitbucket support
Git	Distributed	Merge	Active	Yes
Mercurial	Distributed	Merge	Active	Yes
SVN	Client-server	Merge or lock	Active	No
CVS	Client-server	Merge	Maintenance only	No

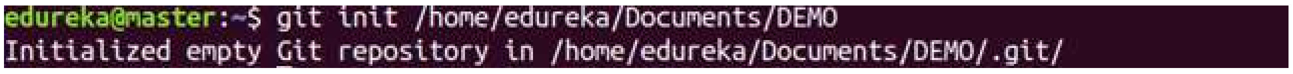
SESSION 2

DEMONSTRATE THE FOLLOWING STAGES OF GIT

1.git init

Usage: git init [repository name]

This command is used to start a new repository.

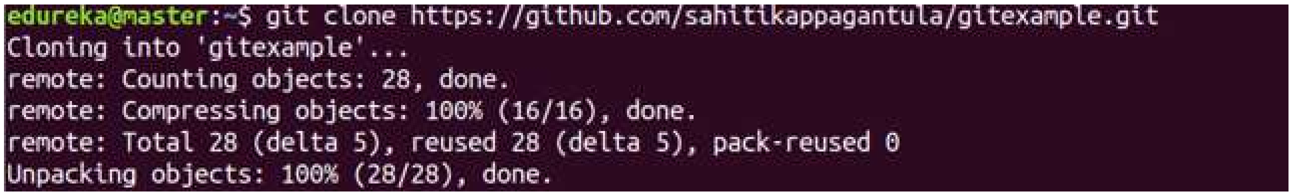


```
edureka@master:~$ git init /home/edureka/Documents/DEMO
Initialized empty Git repository in /home/edureka/Documents/DEMO/.git/
```

2. git clone

Usage: git clone [url]

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

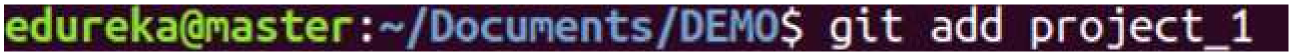


```
edureka@master:~$ git clone https://github.com/sahitikappagantula/gitexample.git
Cloning into 'gitexample'...
remote: Counting objects: 28, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 28 (delta 5), reused 28 (delta 5), pack-reused 0
Unpacking objects: 100% (28/28), done.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

3.git add

Usage: git add [file]

This command adds a file to the staging area.

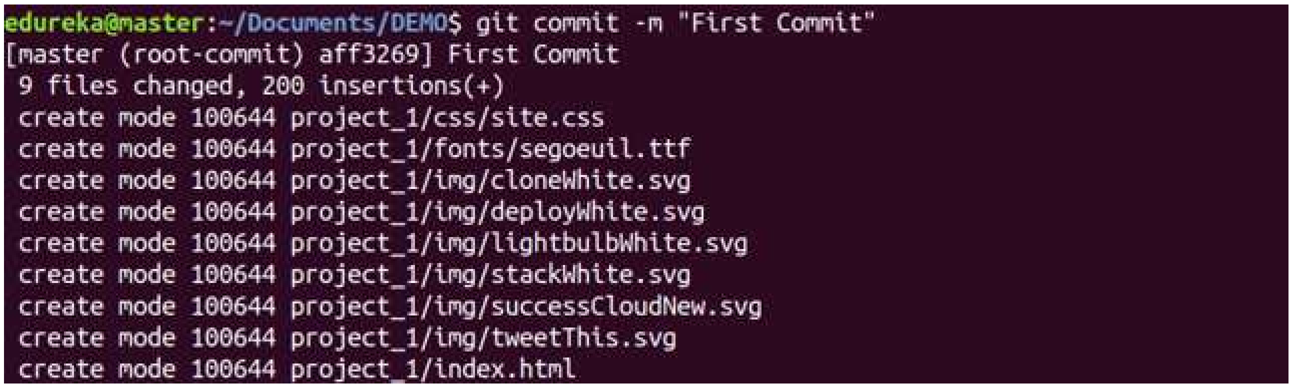


```
edureka@master:~/Documents/DEMO$ git add project_1
```

4.git commit

Usage: git commit -m "[Type in the commit message]"

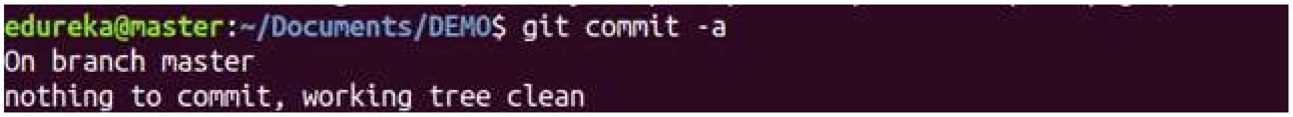
This command records or snapshots the file permanently in the version history.



```
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
```

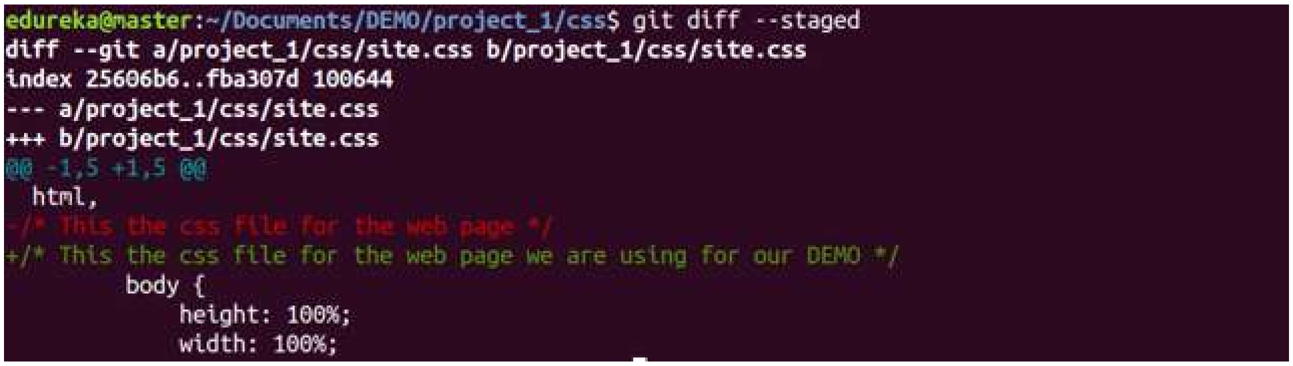
Usage: git commit -a

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
```

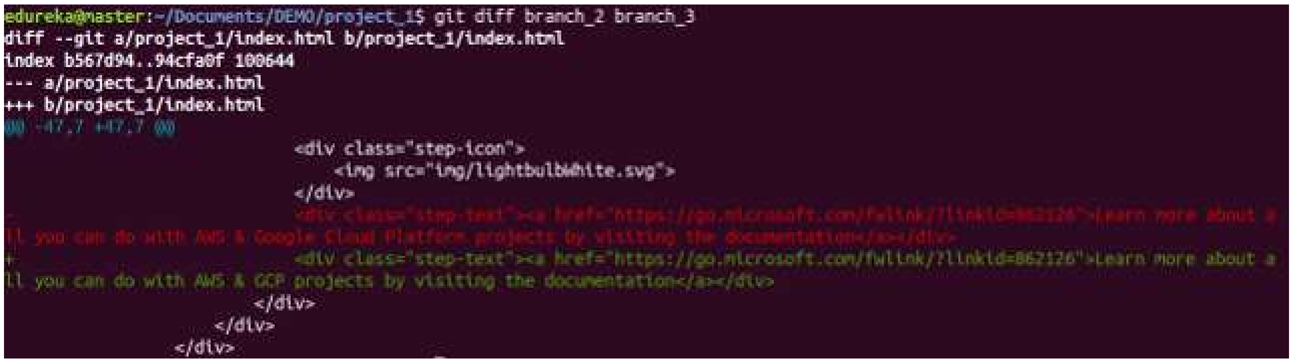
5.git diff



```
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.

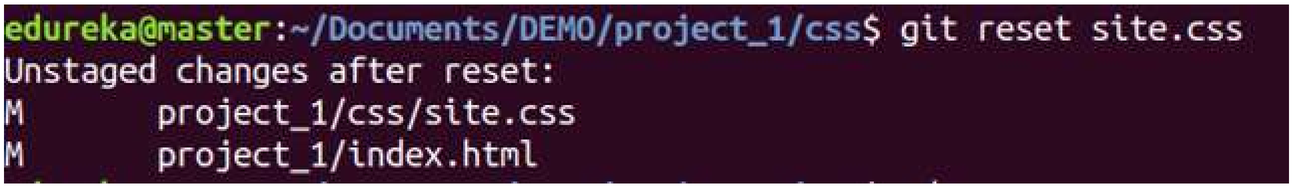


```
edureka@master:~/Documents/DEMO/project_1$ git diff branch_2 branch_3
diff --git a/project_1/index.html b/project_1/index.html
index b567d94..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -47,7 +47,7 @@
                        <div class="step-icon">
                            <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>
```

6. git reset

Usage: git reset [file]

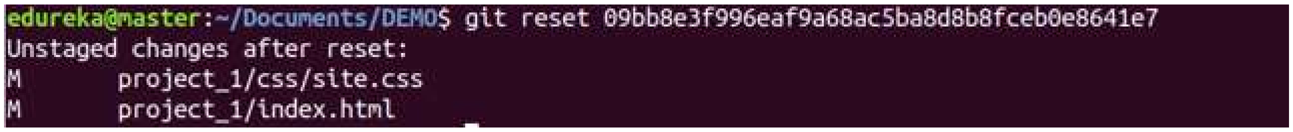
This command unstages the file, but it preserves the file contents.



```
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
```

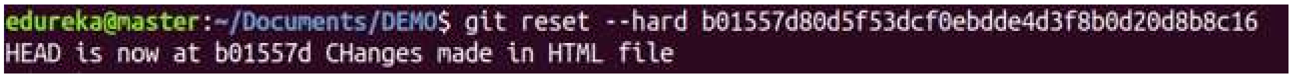
Usage: git reset [commit]

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.

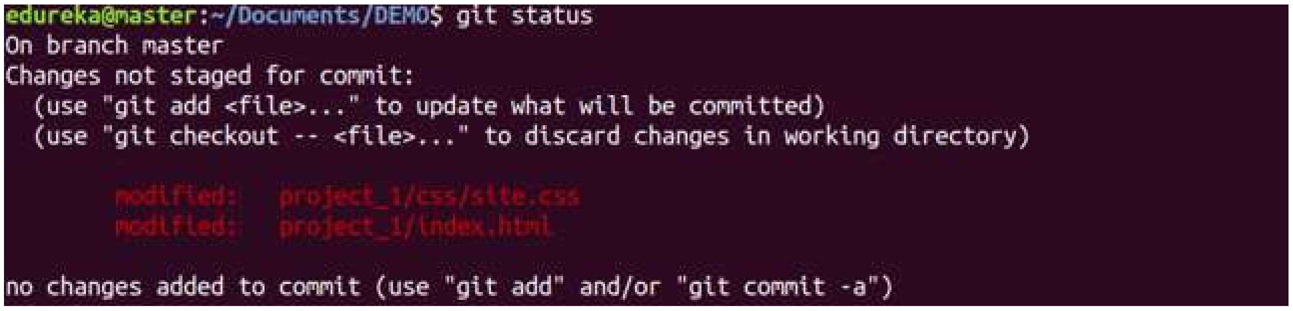


```
edureka@master:~/Documents/DEMO$ git reset --hard b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
HEAD is now at b01557d CHanges made in HTML file
```

7. git status

Usage: git status

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

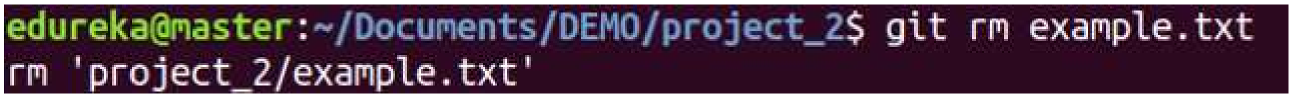


```
edureka@master:~/Documents/DEMO$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

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

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

8. git rm

Usage: git rm [file]

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

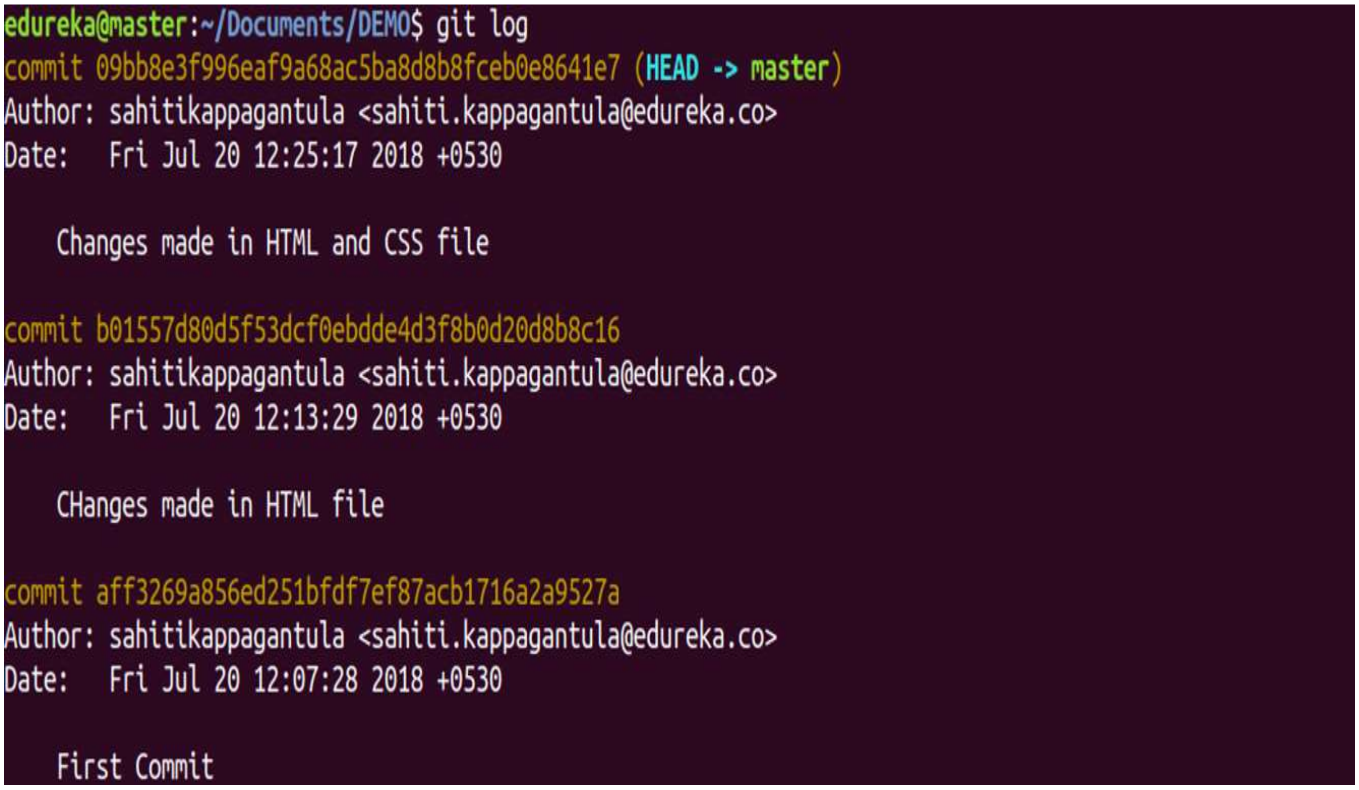


```
edureka@master:~/Documents/DEMO/project_2$ git rm example.txt
rm 'project_2/example.txt'
```

9. git log

Usage: git log

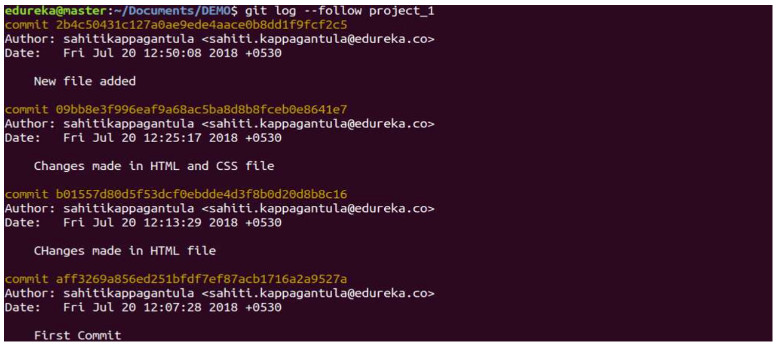
This command is used to list the version history for the current branch.



```
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.

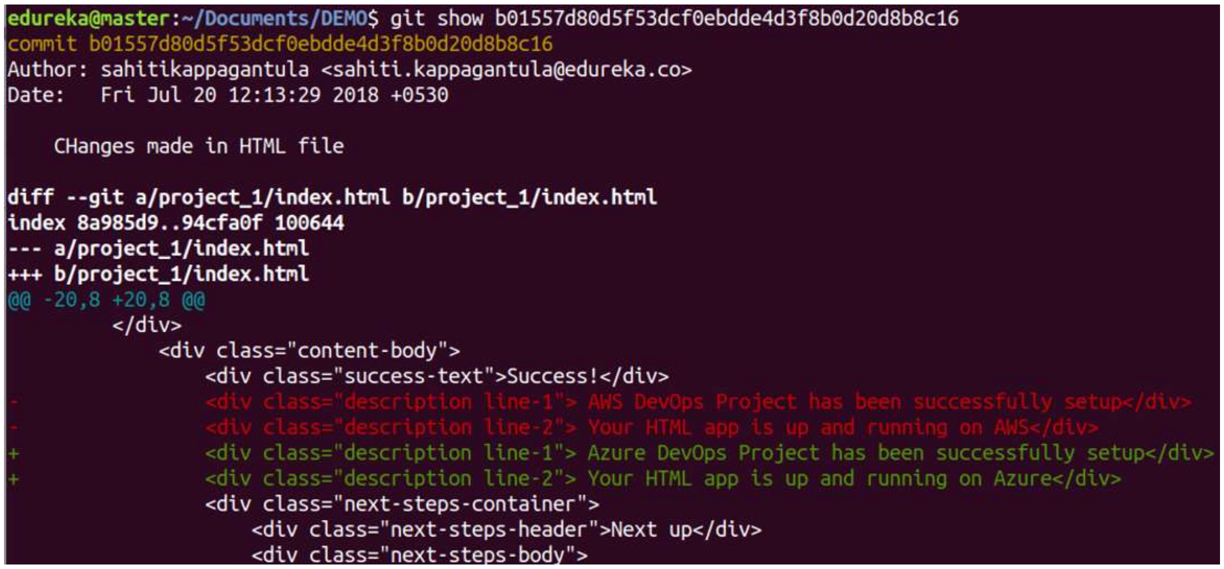


```
edureka@master:~/Documents/DEMO$ git log --follow project_1
commit 2b4c50431c127a0ae9ede4aace0b8dd1f9fcf2c5
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:50:08 2018 +0530

    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
```

10. git show

Usage: git show [commit]

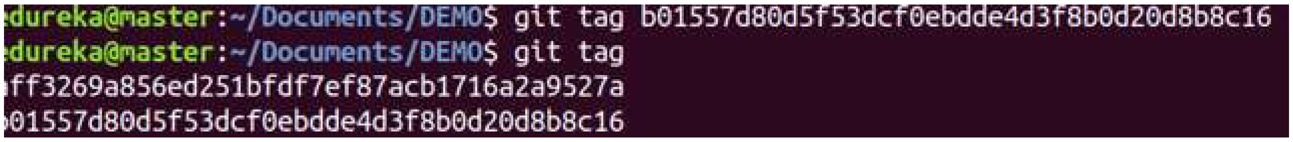
This command shows the metadata and content changes of the specified commit.


```
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">
```

11.git tag

Usage: git tag [commitID]

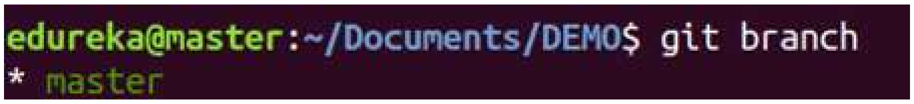
This command is used to give tags to the specified commit.


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

12. git branch

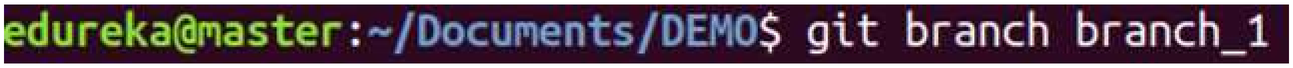
Usage: git branch

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


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

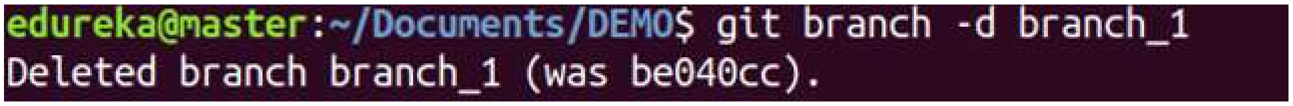
Usage: git branch [branch name]

This command creates a new branch.


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

Usage: git branch -d [branch name]

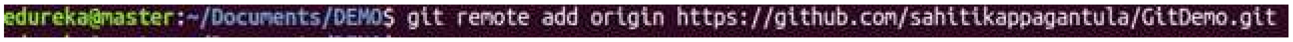
This command deletes the feature branch.


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

13. git remote

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

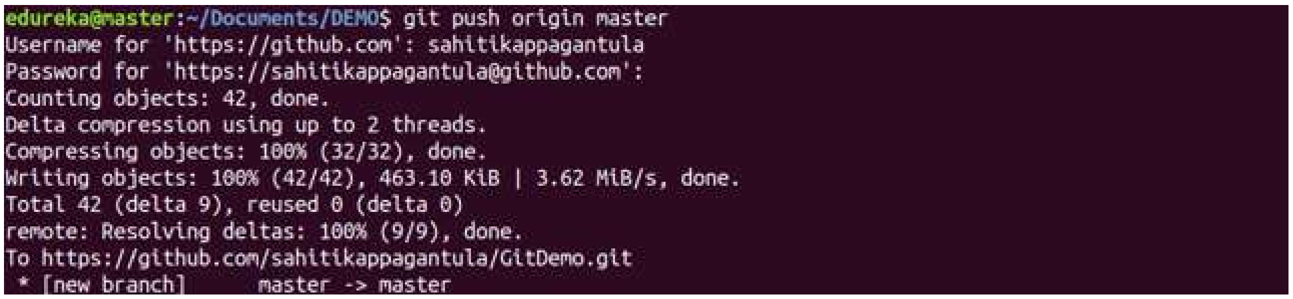
This command is used to connect your local repository to the remote server.


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

14. git push

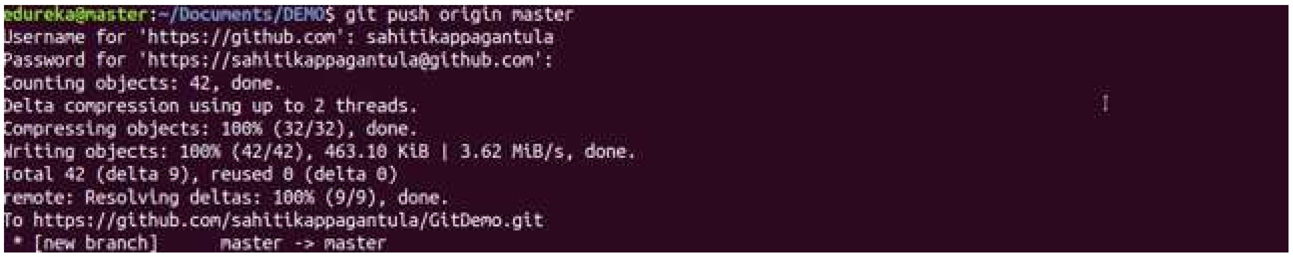
Usage: git push [variable name] master

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
```

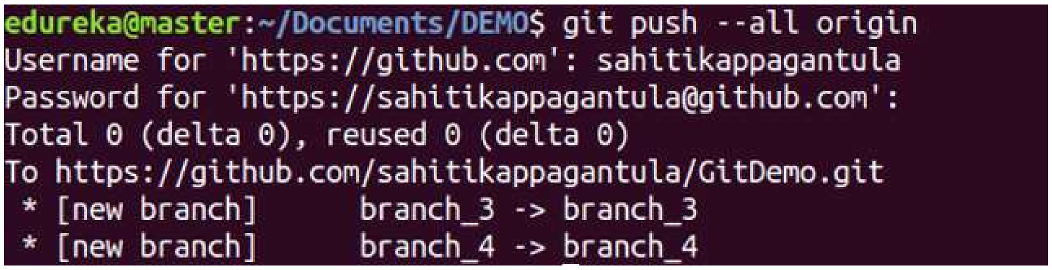
Usage: git push [variable name] [branch]

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]

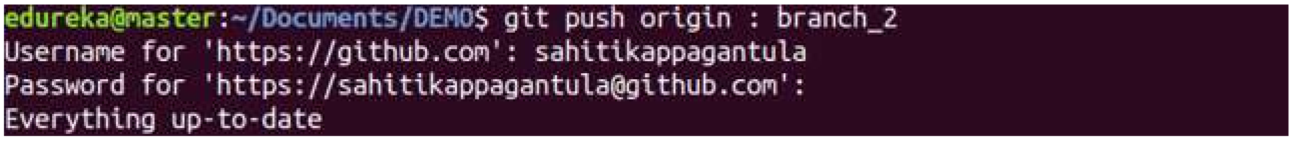
This command pushes all branches to your remote repository.



```
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.

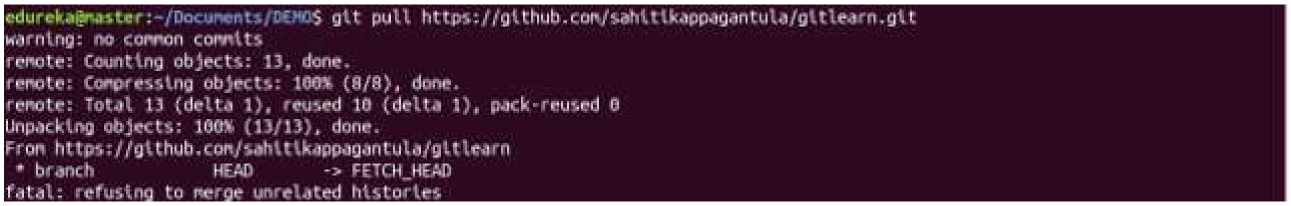


```
edureka@master:~/Documents/DEMO$ git push origin : branch_2
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Everything up-to-date
```

15. git pull

Usage: git pull [Repository Link]

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

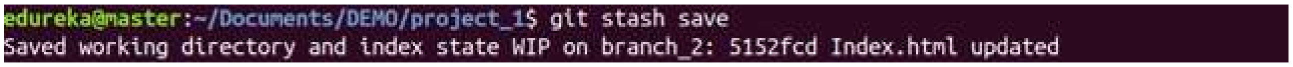


```
edureka@master:~/Documents/DEMO$ git pull https://github.com/sahitikappagantula/gitlearn.git
warning: no common commits
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 1), reused 10 (delta 1), pack-reused 0
Unpacking objects: 100% (13/13),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

16. git stash

Usage: git stash save

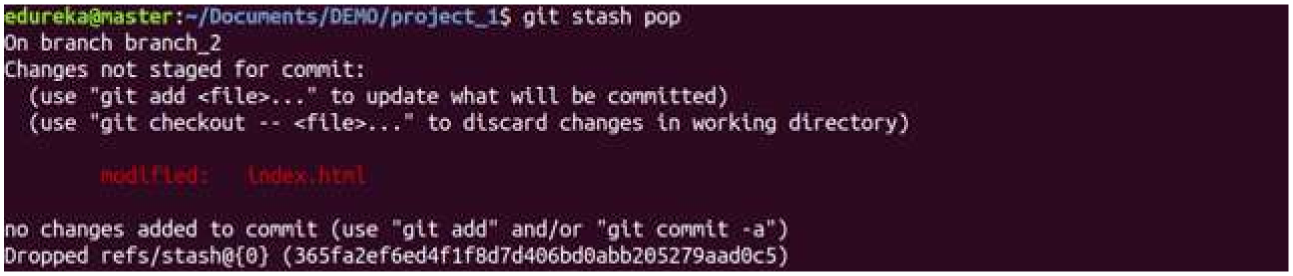
This command temporarily stores all the modified tracked files.



```
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

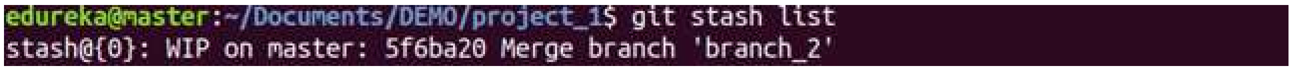
This command restores the most recently stashed files.



```
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

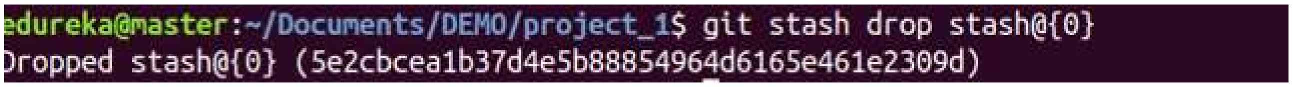
This command lists all stashed changesets.



```
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)
```

SESSION 3

Q1. MAKE DIRECTORY

1. From the command line, move to the directory you want to contain your Git repository.
2. Type the following command to configure your Git username, where <your name> will be your GitHub username.

git config --global user.name "<your name>"

3. After entering the above command, you should be returned to the command prompt. Next, enter your e-mail address by typing the following command, where <your e-mail> is your e-mail address.

Q2. How to connect to a remote repository

git config --global user.email "<your e-mail>"

4. Once the above steps have been completed, you'll be ready to connect to a remote repository. To find the repository address, go to a repository on GitHub and click the Clone or download repository link to get the address. For example, we've created a

repository called "example" at https://github.com/Computerhope/example.git address. Copy the address to your clipboard.

5. Once copied go back to the command line and type the following command, where <URL> is the address you copied. To paste that address into the command line right-click in the command line window and click paste.

Q3. How to a new directory in your current directory
git clone <URL>

6. Once the Git repository is created, you'll have a new directory in your current directory with the name of the Git repository.
7. Once the Git remote repository is cloned to your local repository, you should have a new folder in the current directory with the name of the Git repository. For example, in our "example" Git we would have a new directory called "example". Use the cd command to change into the new directory.
8. Once in the new directory, type the following command to list the remote repositories.

Q4. How to check name of master git branch
git remote

9. If successful, you should see "origin" that is the name of your master Git branch. To see the aliases (URL or path), type the following command.

git remote -v
Working in your local repository and pushing files

After following the above steps and cloning a remote repository, you can work on the files as you normally would. You can create new files or edit existing files from the command line or your favorite text editor. Below we will go through the steps in creating a new file and pushing that new file as well as editing an existing file and pushing the update.

Q5. Creating a new file and pushing to remote repository

1. Create a new file in the Git directory by typing the following command from either the Bash or Windows command line. The following command opens and creates a file called example.txt in Notepad. In Git Bash, you could also use the touch command to create a blank new file and then type "start <name of file>" to open the file in your favorite text editor.

start notepad example.txt

2. In the text editor, enter some text into the file and save and exit the file.
3. Back at the command line type the following command to get the current status of your branch and untracked files.

git status

4. Git displays a window similar to the example shown below, showing that the file we created is new and untracked by Git.

5. As mentioned in the notes and seen in the picture, we'll now want to add this file to Git to be tracked by typing the following command. If your file is not named "example.txt," you'd want to change the text to the name of your file.

```
git add example.txt
```

6. After entering the above command, the file is added as a new file also known as staging. Typing git status again shows you in green that the file is a new file that is ready to be committed.
7. Next, type the following command to commit the changes made in the local workspace to the local repository. In the example below, our notes "First example" should be notes that will make sense to you and anyone else who may be working with you on your project.

```
git commit -m "First example"
```

8. Finally, now that changes have been moved from your workspace into your local repository it is ready to be pushed to the remote repository. Type the following command to push all changes to the remote repository.

```
    git push
```

Q6. Modifying a file and pushing to remote repository

1. Edit and modify one or more files in your Git.
2. Type git status to get see the status of all the files that have not yet been committed from the workspace to the local repository.
3. Type the following command to add *all* files. The single period indicates that you want all files to be added to the local repository. Some people may also use git add -A to add all.

```
git add .
```

4. Once the files have been added, type the following command to commit. Change the notes to apply to your commit.

```
git commit -m "Second update"
```

Finally, type git push to push the commit to the remote repository.

If it's been awhile since you've committed any work, perform the git pull command to get the latest updates from the remote repository and merge them into your local repository.

To get all changes without merging, run the git fetch command to grab all of the latest updates from the remote repository without merging any of the new changes.

```
git add .
git commit -m "Fixed the merge conflict"
git push origin master
```

Q7. To create a branch in your local repository, follow the steps below.

1. In the Git master directory, type the following command, where "<New Branch>" is where you would put the name of the new branch name. For example, we could call the branch "examplebranch".

```
git branch <New Branch>
```

2. Next, type the following command to switch to the branch.

```
git checkout <Branch Name>
```

3. After entering the above command, the prompt (in Git Bash) changes from "master" to the branch name to indicate you're working in a branch and not the master.
4. From this point, you can continue to use Git and modify the files how you have in the past.
5. To see all available branches, you can use the git branch command. For example, typing git branch displays your local repository branches. Typing git branch -a displays all local and remote repositories.
6. If you need to push the branch to the remote repository, you can run the following command.

```
git push --set-upstream origin <Branch Name>
```

7. Finally, if you need to switch back to the master branch, you can type the following command.

```
git checkout master
```

Q8.How to merge a branch back into the master

After you've completed your work in a branch, you'll want to merge it back into the master or another branch by following the steps below.

1. Move into the branch you want to merge into. For example, if you wanted to merge back into the master, type the following command.

```
git checkout master
```

2. Once in the master, type the following command to merge the branch.

```
git merge <Branch Name>
```

3. Once the merge is performed, add the files.
4. Next, commit the changes.
5. Once merged and committed, push the merge by typing the following command. If you get conflicts during the merge, see our how to deal with merge conflicts section.

```
git push
```

Q9. How to delete a local and remote branch

If after merging a branch you no longer want to keep the local or remote branch, you can perform the following commands.

To delete the local branch, type the following command.

```
git branch -d <Branch Name>
```

To delete the remote branch, type the following command.

```
git push origin --delete <Branch Name>
```