GIT

# Stages:

Modified

Staged

Committed

# 3 sections of Git project

**Working Directory:**

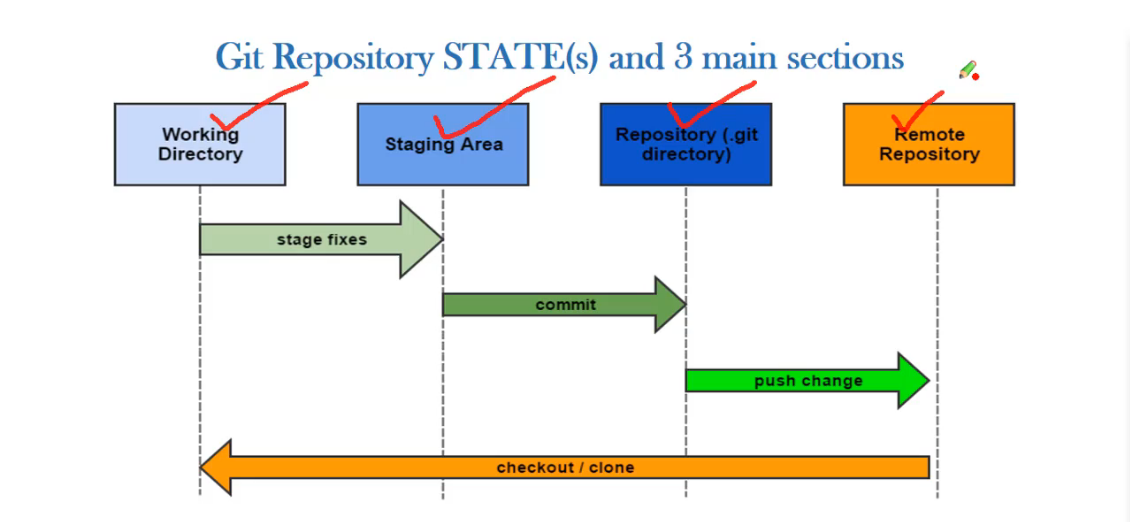
This is the root directory of your Git Project.

**Staging Area:**

Also called as index, this is where all related changes are built up. (When you execute git add command, artifact comes in this area.

**Commit Area:**

This is where all artifacts are stacked safely in Git Database.



Installing Git:

apt-get install git

Ways of creating a Git repository:

**From Scratch:**

Creating a repository from blank state.

**Existing Project:**

Converting an existing un versioned project to git repository. ( bringing existing project and version control).

**Copying:**

Copying and existing git repository from GitHUB.

**Git help assistance:**

git help

git help -a ( list available sub commands)

git help -g ( some concept guides)

git help <command name>

**Setting up Git User:**

git config --global –list ( To check if there already existing configuration )

**Git new author and email configuration:**

git config --global user.name “nitin8147”

git config --global user.email [nitin8147@gmail.com](mailto:nitin8147@gmail.com)

**Creating a Git repository:**

git init

**What is Git Hub:**

Web based git repository

Launched in 2008

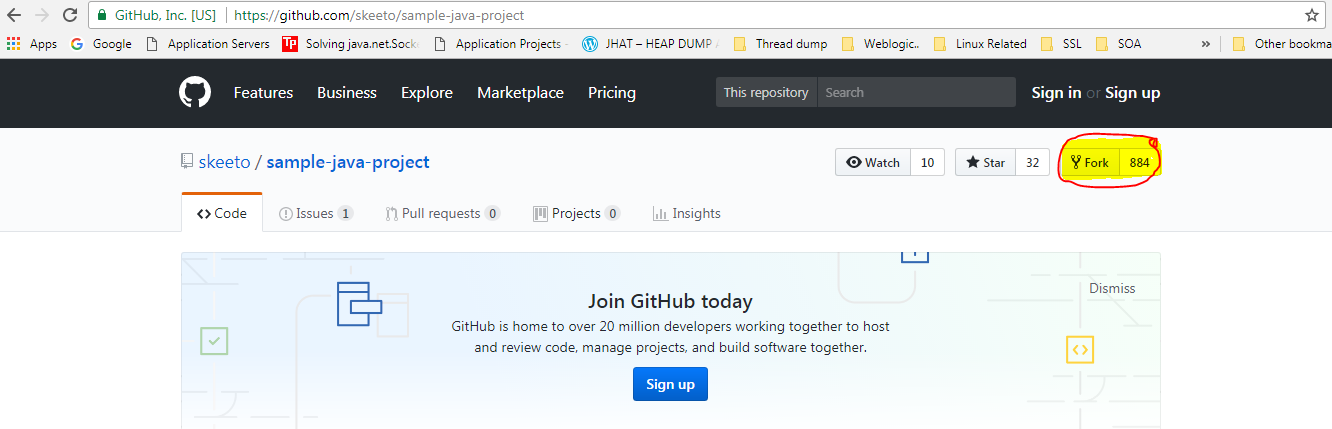
Git hub has based both free and paid plans

Need to create user account

Documentation and bug tracking

Largest hosted Git Repository

**Forking an existing project from Git Hub:**



**Copying a Git Hub repository:**

git clone < git repository url>

eg :- **git clone** <https://github.com/nitin8147/sample-java-project.git>

Once you update any artifact that is the part of the git repository, you have to mark it for update via add command.

Changing an artifact of a repository 🡪 **Modified Stage**

Marking the changed artifact 🡪 **Staged Stage**

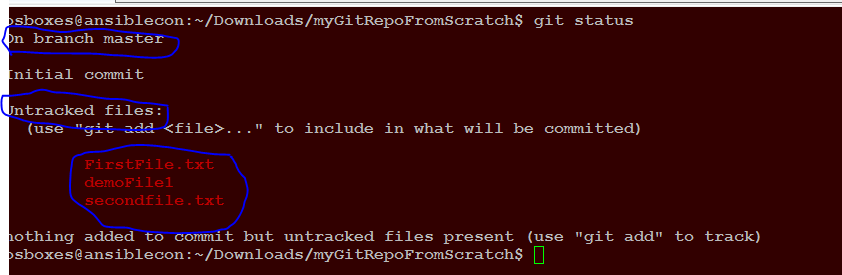


Figure 1.0

**git status command:**

Figure 1.0 shows the output of git status command. Once you are inside the git repository and you can check the artifacts that have been changed but not marked for tracking (Staging) or committed.

**git status**

**Note three things from Figure 1.0:**

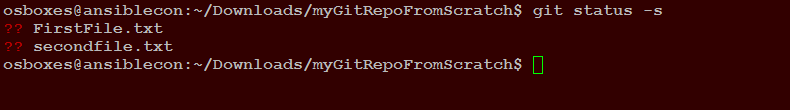
First line indicating that you are on master branch

Line showing that certain files are untracked

Line showing the names of the file that are untracked

One handy “git status option is -s “. Simply check only the names of tracked and untracked files without extra details.

Eg a. git status -s ( Below figure shows 2 untracked files)

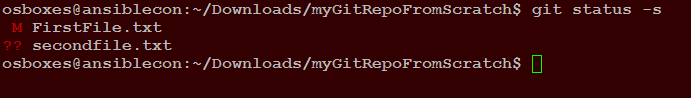


Eg b. Below output of “git status -s” shows that one file is tracked and another is untracked



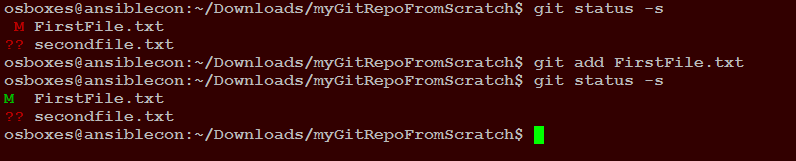
Eg c. Below out is displayed by git status command when you have made changes to earlier committed file but has not marked for staging

M- Modified



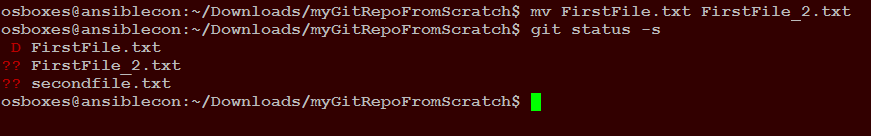
Eg c . Below output when you have staged the modified file

M- Colour changes to green here



Eg d- Below output when a file is renamed.

D- status for the older file which stands for deleted. Since I renamed FirstFile.txt so its status changes to D and the renamed file appears with ?? ( New and unstaged)



**git add command:**

This command is used to mark the artifact for staging.

**git add <file name>**

once the artifact has been marked for staging you can check its status by typing the git status command again. See Figure 1.2 , here **demoFile1** was marked for staging using the git add command .

Note: to add everything from the currently working directory

**git add .**

**Note:** notice the dot at the end of above command

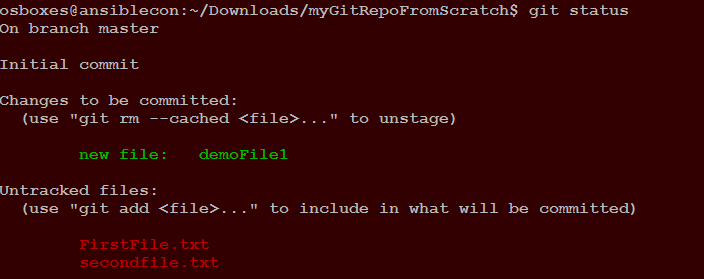


Figure 1.2

Note: To unmark a file that has been marked for staging use

**git rm --cached <File name that has been marked>**

**git commit command:**

After the artifact that has been marked staged using the “git add” command, we can commit the changes using the below command:

**git commit -m "< message that will identify the committed changes>"**

**git express commit: (Committing without the need to add for staging)**

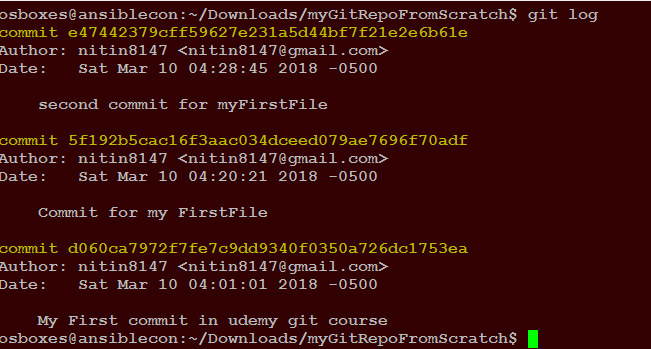
git commit -am “< message> ”

**git log:**

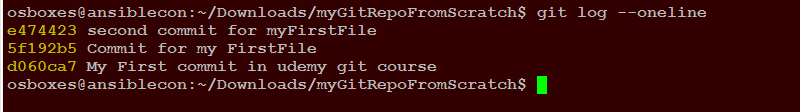
This command is used to view the git commit history

Git log Usage options with screenshot:

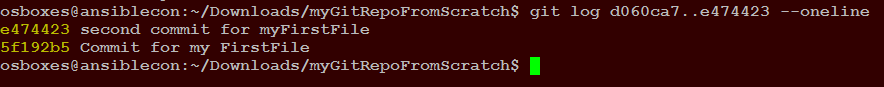
**git log**



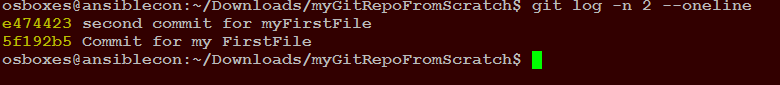
**git log --oneline**



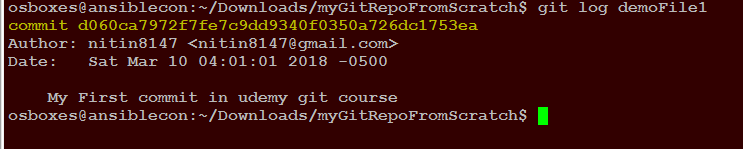
**git log since..until (Commit log between 2 git identifiers)**



**git log -n 2 –oneline ( get log of the last 2 commits)**



**git log <File name > ( git logs for a particular file)**



**Check the branches of a repository:**

**git branch**



**Check both local and remote branch:**

**git branch -a**



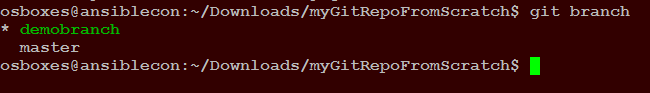
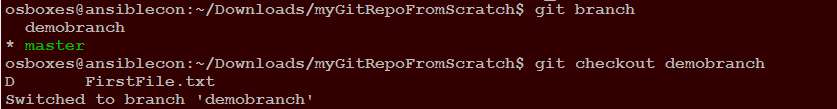
**Creating a new branch:**

**git branch <branch name>**



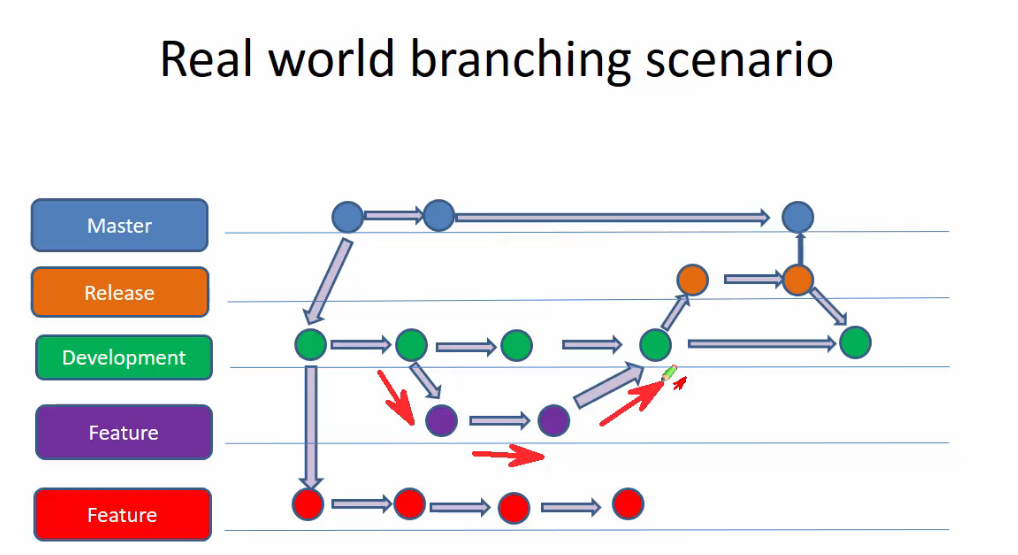
**Switching to the created branch:**

**git checkout <branch name>**



**Note:** When you have switched to your own branch using “git checkout” from the master branch, you can check which branch is currently by symbol \* and the green color of the branch name

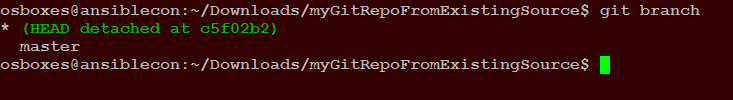
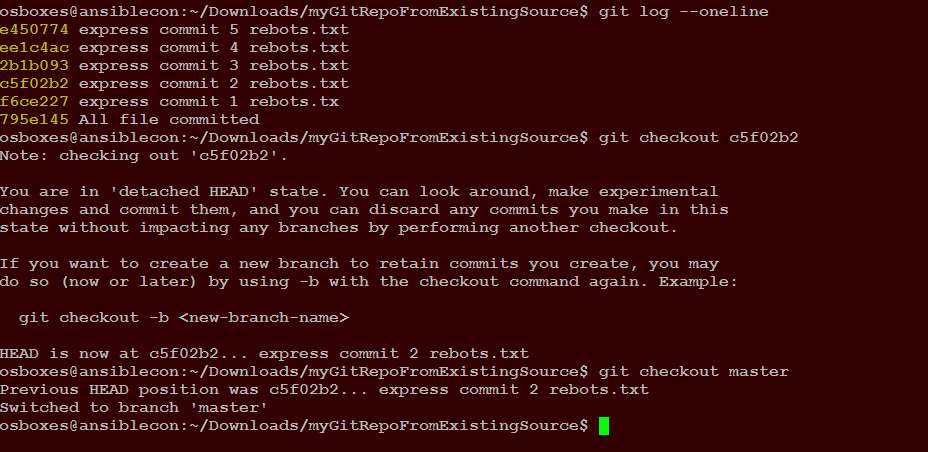
**Note:** To create and switch the branch use “git branch -b <branch name>”



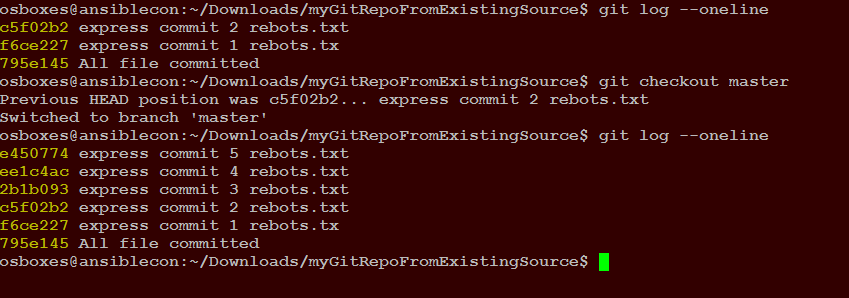
**Git check out options:**

In the normal situations the latest commit id points to the branch master but when we try to checkout any older commit with its specific commit id, the head position changes and then that specific checked out id points to the master branch. But this is only for reading and experimenting with the selected commit state of the selected commit id but branch still contains artifacts in its very recent commit state.

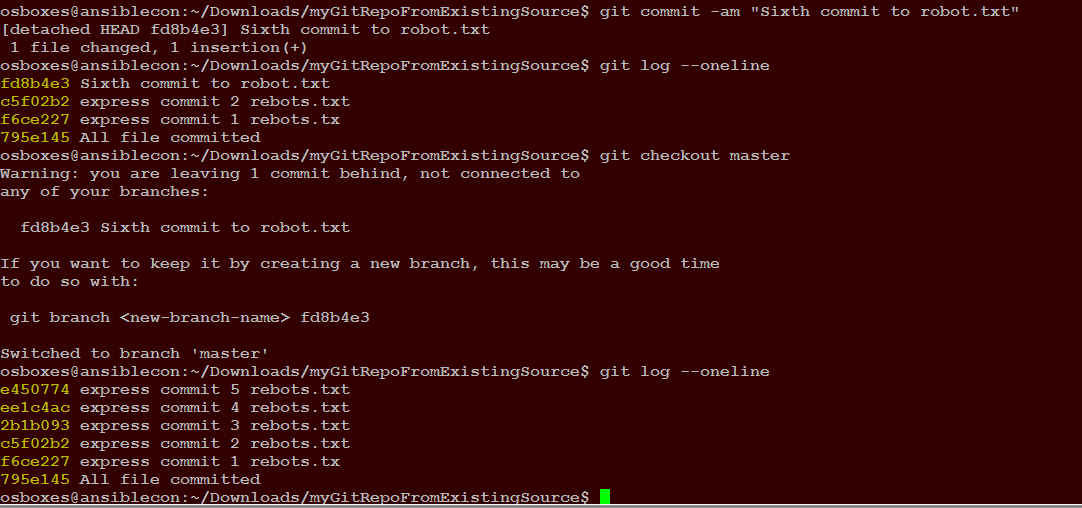
But if you want to retain the commit of that specific commit id, you will have to create a new branch and commit it to the master



Check out the head pointing in detached head state and in normail state



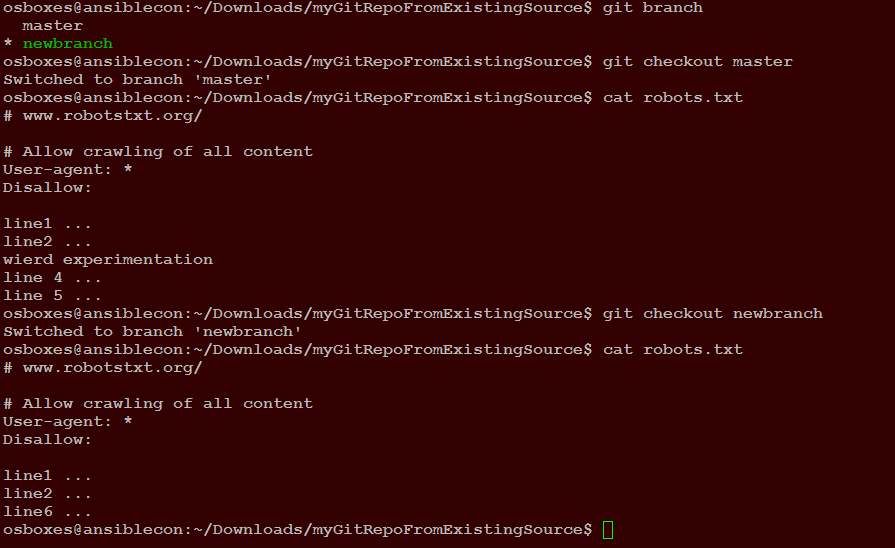
Below Screen shot showing that when you make changes in detached head brach and commit it they are remain in the same detached head branch only . Check difference git log output in detached head branch and master branch .



Now if you switch to master branch and come back to head branch, you will notic that the changes you commited in detached head branch artifacts are no longer viisble , hence prooved that changes in this state not retained.

Now if you want to retain those changes you will have create a new branch out of detached head and you will those changes are now retained in the new branch

**git checkout -b newbranch ( Retains the commit to this new branch that you create , -b option is important)**



**git checkout for files:**

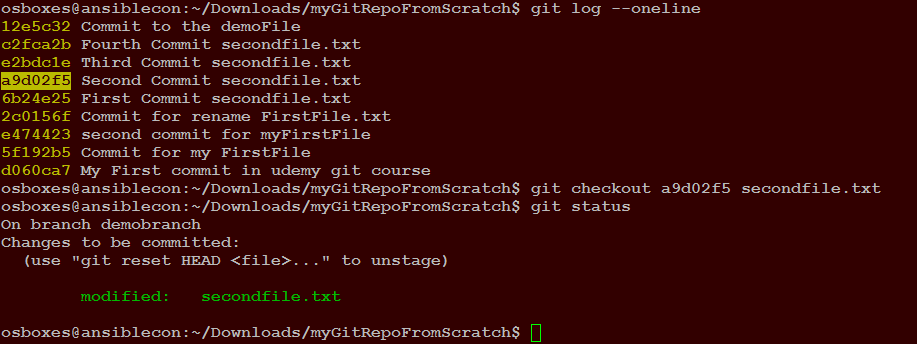
**git checkout <commitid> filename**

Note: Remember when we don’t specify the name of the file like the last time we create a headless state and anything done in this headless state does not make the changes permanent and has no effect on master branch .

But when we specify file name along with the commit id , we are referrring only to the state of the file in that repository and changes made will take an effect for this file after those have been commited.

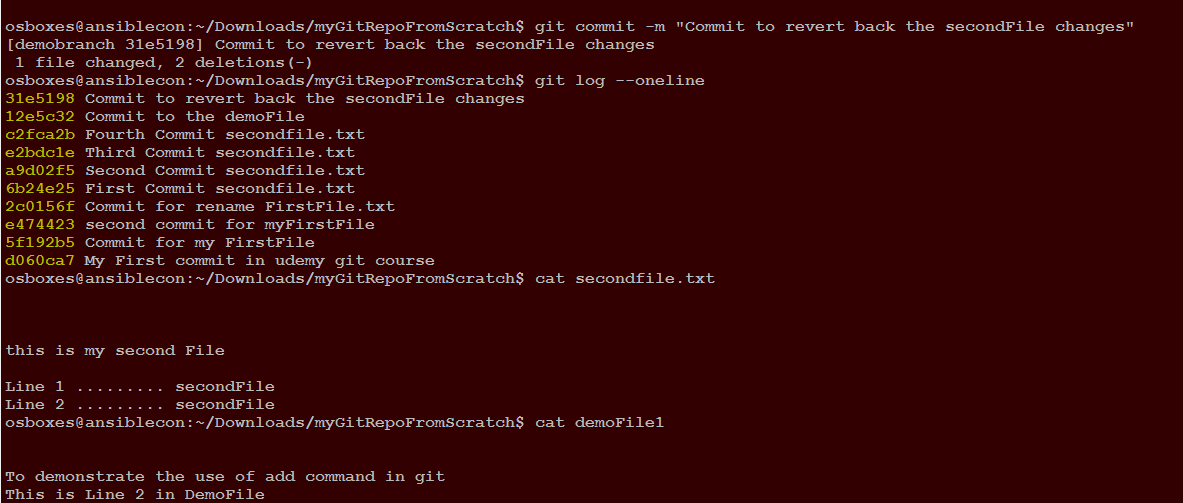
I below figure I have changed the second file 4 times and performed commit each time. I also have changed the demoFile at the end and performed the commit.

When I executed check for secondfile selecting a particular id, secondfile directly moves to modified state.



**Note:** If you want to revert the changed that you have staged with this check out command, you can use the below command to fallback and un stage.

**git checkout HEAD <filename that was staged>**



Notice after performing the commit to retrieve the particular state of second File, changes in “demoFile” does not get reverted. Changes take effect only for the file that we select with git checkout <commitid> <filename>.

**Deleting a git branch:-**

For Local branch:

**git branch -d < branch name>**

For remote branch:

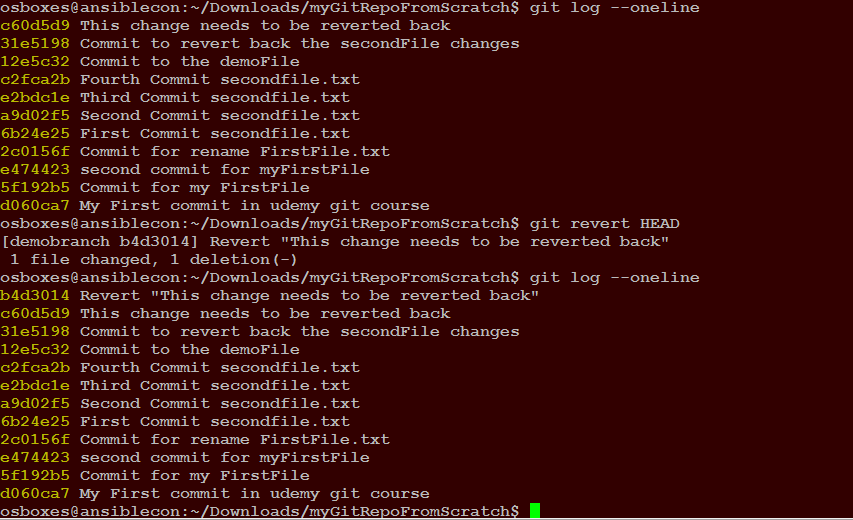
**git push <remote name> --delete <branch name>**

**Renaming a branch:**

**git branch -m <old branch name> <new name of the branch>**

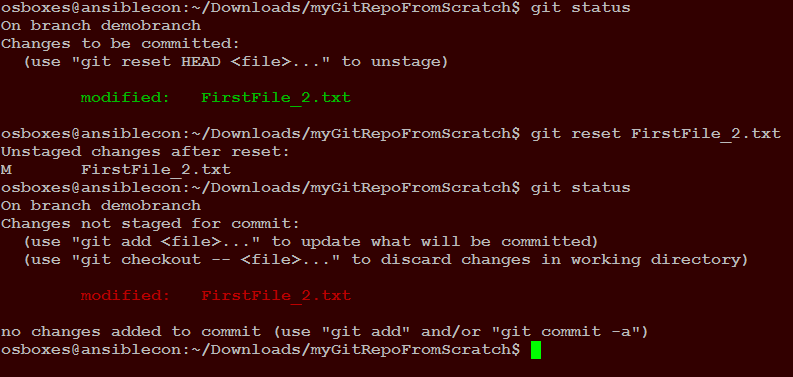
**git revert HEAD :**

This commit is used to revert the changes directly to the last commit where HEAD is pointing and also records the revert.



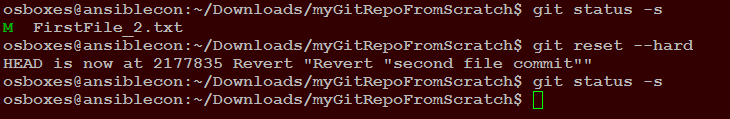
**git reset command: ( git reset <filename> or git reset )**

Moves the files in staged state, back to Modified.



**git reset –hard:**

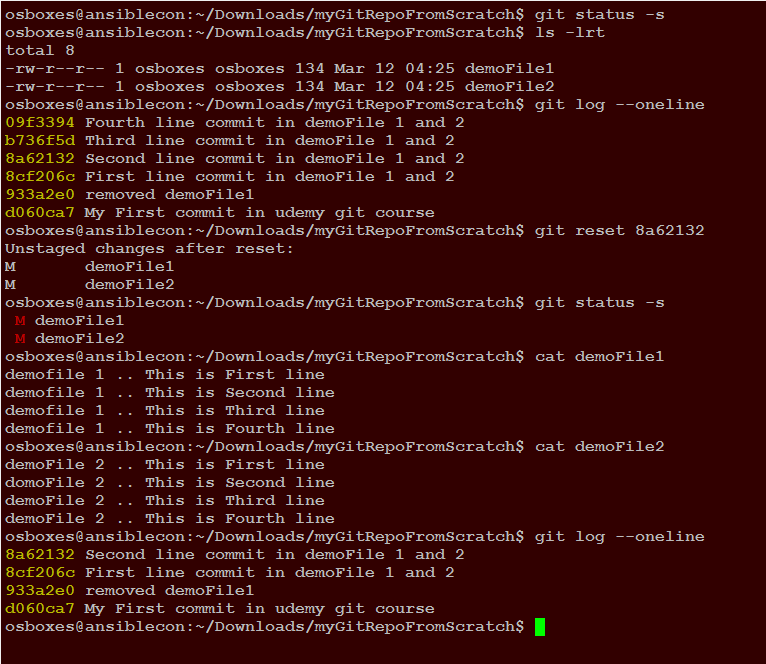
This command resets or undoes all the changes in staging directory as well as working directory or modified state back to its last commit state.



**git reset with a commit id:**

**git commit <commit id>**

This command is used when you want to remove all the comment history set the commit id of your selection. In such case the artifacts of the commit states move to modified state. You can either commit them one by one to generate commit history for each of them or commit them all together to generate a single commit history. As you can see in below screenshot artifacts are not reverted back, all log comments and remove and head points to commit id of your selection.

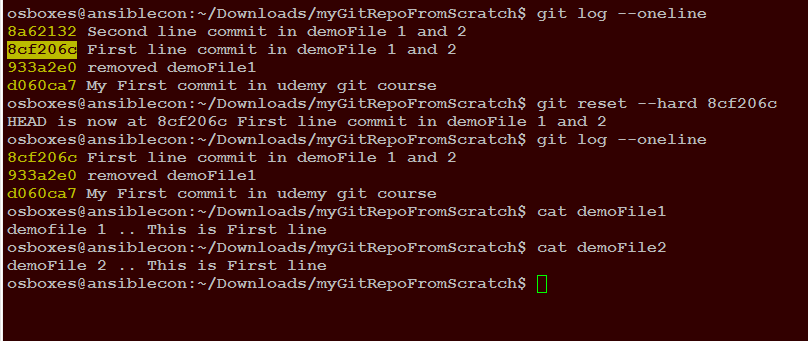


**git reset hard command:**

**git reset –hard <commit id>**

This comments not only removes the other commit above the selected commit id but it also restores the artifacts to the selected commit id state.

It is very destructive command and should be used very carefully



**git clean:**

This command deletes the untracked files. (Files in modified state)

**git clean -n :**

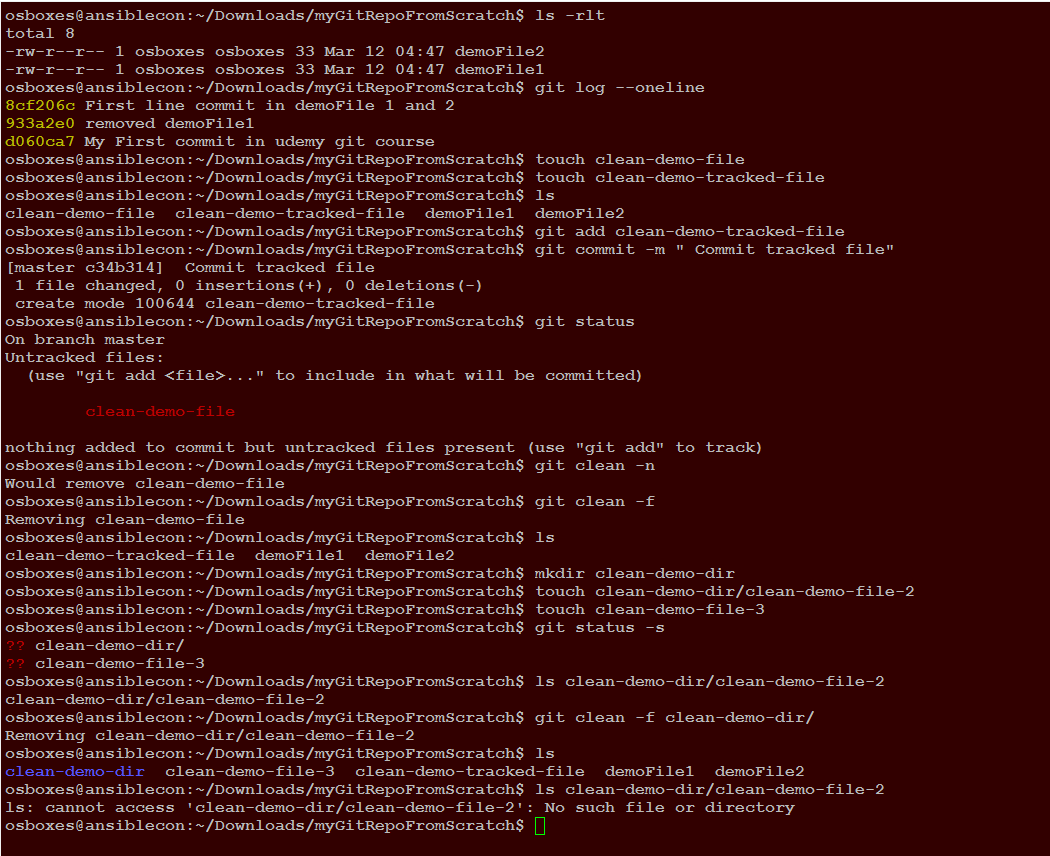
This command runs a dry run to update you about the files that will be deleted by “git clean” command.

**git clean -f :**

This command deletes the untracked files from the current directory.

**git clean -f <path>:**

This command deletes the untracked files in that certain directory only.

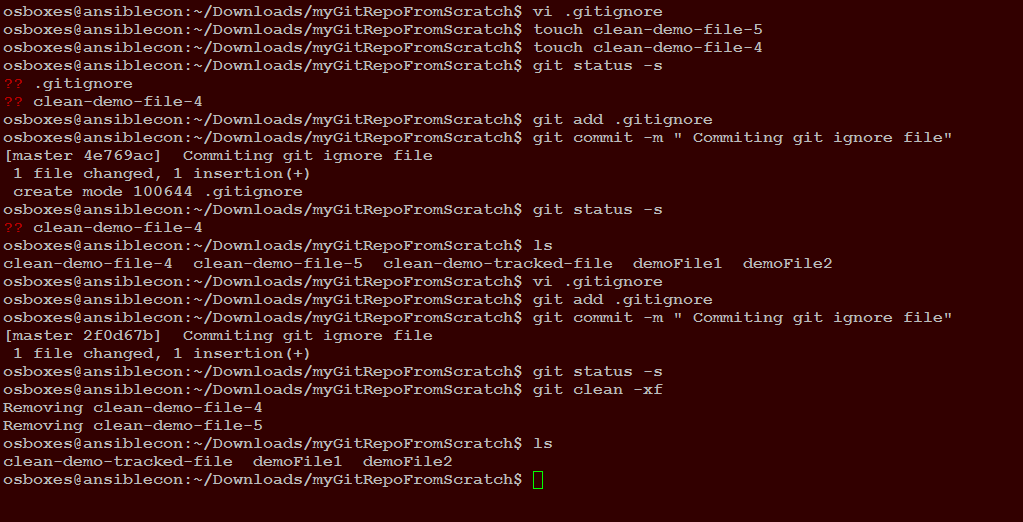


**git clean df:**

Deletes both untracked files and directories.

**.gitignore file and git clean -xf :**

**.gitignore** file is the file which contains the names of files that you want to ignore for normal operations like clean. But with **“git clean -xf”** it will remove all the untracked files even if they are mentioned in **.gitignore** file.



**Pulling from Git hub repository:**

**git pull origin master**

This command is used to sync the local repository with the remote repository. When clone a repository with “**git clone**” command it automatically creats a remote connection called **origin** pointing back to the remote repository. **master**  is the name of the master branch.

**Pushing to remote repository:**

**git push origin master**

git push only works if the repository has been first clone from the same remote repository. Also an important thing is if somebody else pushes to this repository first then your push will be rejected and you will have first sync your repository with **“git pull”** and then try to push again.

**Setting up SSH connection to remote repositories:**

Generate public/private key pair

**ssh-keygen -t rsa -b 2048 -C "key pair generation demo"**

Start the ssh agent in back ground

**eval "$(ssh-agent -s)"**

Add public key to ssh agent

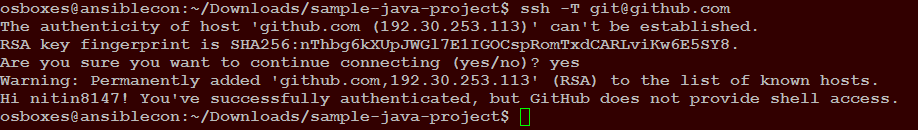
**ssh-add ~/.ssh/id\_rsa.pub**

Add the generated public to the git hub repository

**On git hub under your profile > settings > ssh keys > add new key and paste the public key**

Test your ssh connection

ssh -T [git@github.com](mailto:git@github.com)



**git remote -v:**

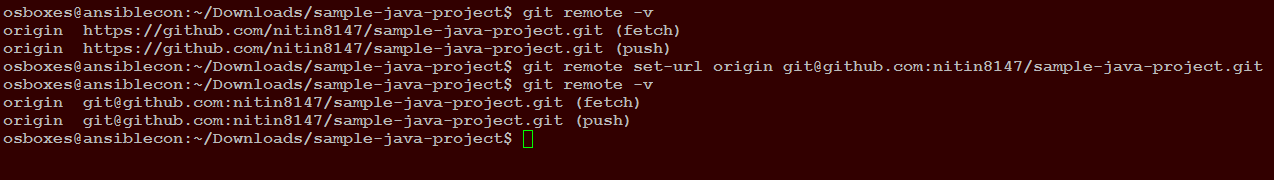
This command is used to know the communication protocol that ie being used to communicate with remote repository.

**Setting the remote repository or origin:**

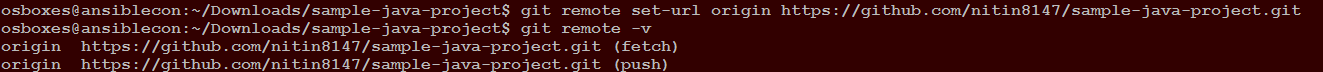
**git remote add origin <git hub repository url>**

**Change the communication protocol to ssh from https:**

**git remote set-url origin git@github.com:<username>/<repositoryname>.git**



**Change the communication protocol to https from ssh:**



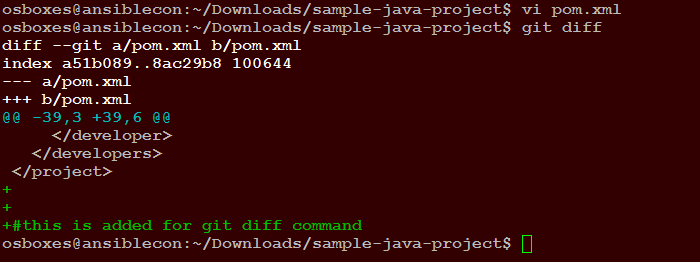
**Setting password for your ssh connection:**

**ssh-keygen -p**

enter the password that you want to set.

**git diff:**

This command will update us about the artificate that has been modifed and what has been modified.



**Note:** git diff is a very bigger concept. This may include differences in working directory and staging area, staging area and commit area, commit area and remote repository

**Git p4merge and diff operations:**

**Installing and Configuring p4merge in windows**

**🡪** Go to <http://perforce.com> and download p4merge tool ( **Helix visual merge tool**) for windows

🡪 Run the downloaded exe file and perform the installer. While installing there will be several components. Select only “Visual Merge Tool” and deselect others while installation.

-🡪 git config –global diff.tool p4merge (Configure p4merge as a diff tool)

🡪 git config –global difftool.p4merge.path <path to p4merge.exe with exe file in path> (Configure p4merge path)

🡪 git config –global difftool.prompt false ( will stop the continues prompt whenever you launch p4merge)

**Now configure merge tool in the same manner**

-🡪 git config –global merge.tool p4merge (Configure p4merge as a merge tool)

🡪 git config –global mergetool.p4merge.path <path to p4merge.exe with exe file in path> (Configure p4merge path)

🡪 git config –global mergetool.prompt false ( will stop the continues prompt whenever you launch p4merge)

**Installing and Configuring p4merge in Linux**

**🡪** Go to <http://perforce.com> and download p4merge tool ( **Helix visual merge tool**) for Linux

🡪 You will get a tar.gz file. Copy it to a folder and unzip and untar it.

🡪 Create a directory /opt/p4merge and move bin and lib directory from the extraction to this newly create directory.

🡪 ln -s /opt/p4merge/bin/p4merge /usr/local/bin/p4merge (Create a sym link…above 2 commands will require root access.

🡪 Test installation by running command p4merge

-🡪 git config –global diff.tool p4merge

🡪 git config –global difftool.p4merge.path /usr/local/bin/p4merge

🡪 git config –global difftool.prompt false

🡪 git config –global merge.tool p4merge

🡪 git config –global mergetool.p4merge.path /usr/local/bin/p4merge

🡪 git config –global mergetool.prompt false

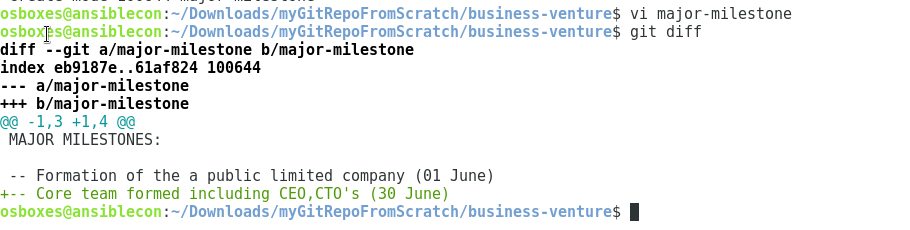
🡪 git config –global –list (Check the updated config)

**git diff:**

**git diff**

**Tells about differences in the contents of the artifacts in different Working directory and the Staging area.**

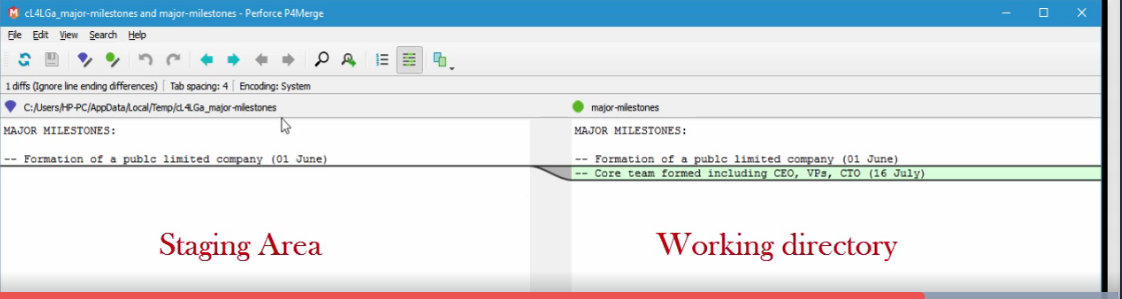
**Like in below screenshot , it tells about the differences in contents of the artifact (major-milestone file) in working directory and staging area.**



**git diff tool command:**

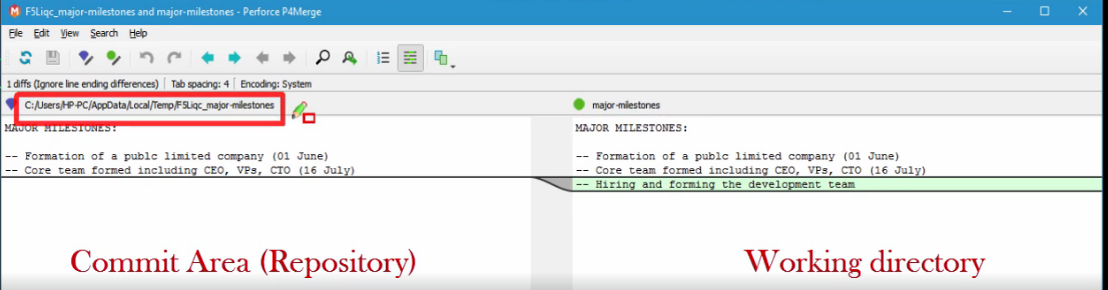
**git difftool**

**equivalent of git diff command but it displays the difference in gui format.**



**git diff HEAD and git difftool HEAD:**

**Tells about differences in the contents of the artifacts in different Working directory and the Commit area.**



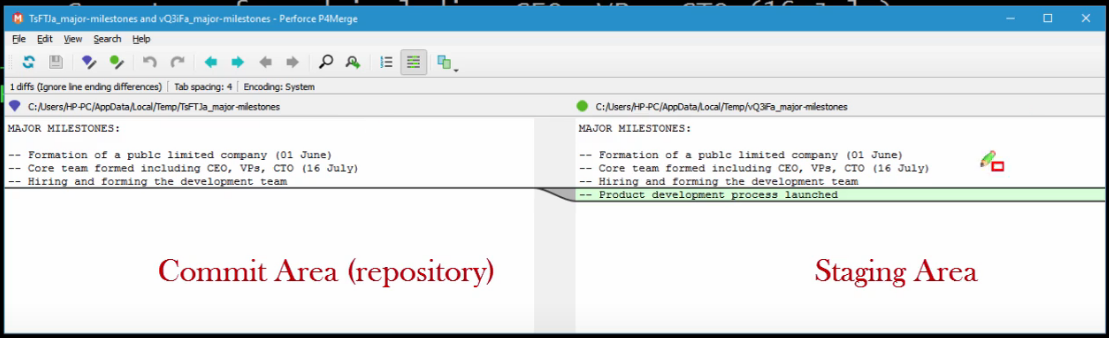
**git diff –staged HEAD or git difftool –staged HEAD:**

**same as *git diff –cached HEAD or git difftool –cached HEAD***

**which is same as git diff –cached or git difftool –cached**

**or git diff –staged or git difftool –staged**

**Tells about differences in the contents of the artifacts in different Staging area and the Commit area**



**Other git diff commands**

**Staging area and commit aread**

**git diff <particular commit id > HEAD**

**git diff <particular commit id > HEAD**

**git diff HEAD^ HEAD : same as git diff HEAD~1 EAD**

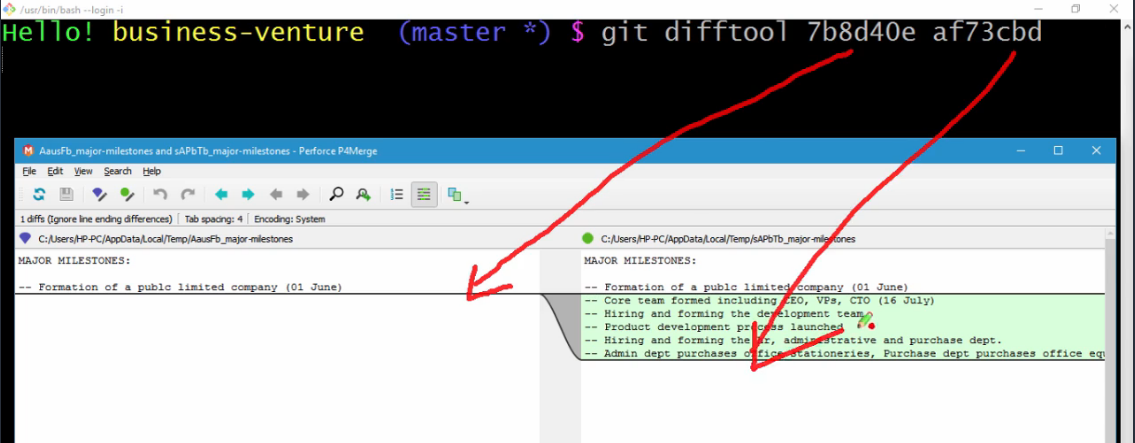
**^HEAD is for commit id prior to HEAD**

**HEAD~1 is for commit id prior to HEAD**

**Git diff between the 2-arbitrary commit:**

**git difftool <commit id 1 > commit id 2> or git diff <commit id 1> <commit id 2>**

If there are differences in more than one doc p4merge will launch another file automatically on closing the first one



**Viewing and comparing the diff between a single file:**

**git difftool -- <filename>**

**Viewing and comparing the difference between the remote and local file:**

**git diff master origin/master**

**or**

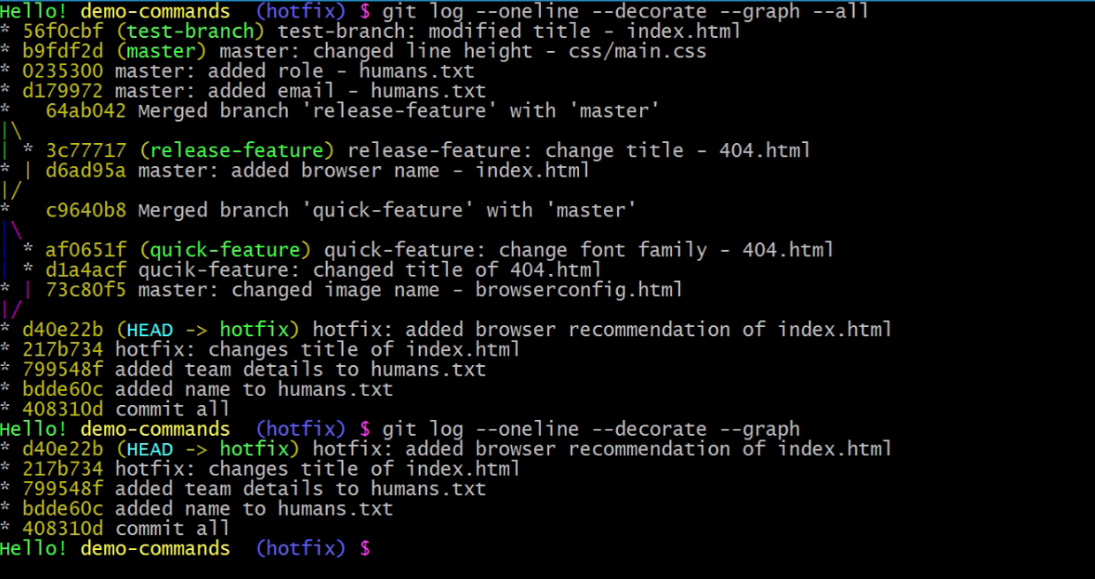
**git diff <branch name> origin/master**

**Viewing and comparing the difference between 2 branches**

**git diff <created branchname> master**

**git log command details:**

**git log –oneline –decorate –graph –all (displays commit of all branches in decorative and graph form)**



**git log –stat** (displays the files added and deleted in various commits)

**git log –p** (displays detailed diff of each and every commit)

**git log <directory or file name>** (commits info of a particular file or directory)

**git log --oneline --grep="<string to be searched>"** ( searches all commits which contains this string in the commit message)