<https://git-scm.com/docs/gitrepository-layout>

git credential : dmishra4/deepakkumar-hub

Password: Ainesh\*008

Step 1 : git status

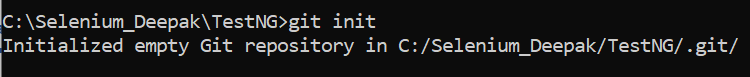
fatal: not a git repository (or any of the parent directories): .git



Step 2 : Create or initialize empty local Git repository using below command

Command: git init

Output: Initialized empty Git repository in C:/Selenium\_Deepak/TestNG/.git/



**Step 3:** Create a file in target directory and execute below command to add created file in Staging area.

Create file command: echo <file-content> **>** <file-name>

Edit command : echo <file-content> **>>** <file-name>

Example:: echo hello > m3.txt

**Note**: Whenever you create any file in your target directory, it will be called as untracked file as git is not aware about it. If you want that git should track this file then you have to add newly created file in staging area using below git commands.

There are many ways to add file/s in staging area

* git add <file\_name> ( To add one file in Statging area)
* git add <file\_name1> <file\_name2> ( To add
* git add .
* git add -A

Example: git add hello.txt

Git add .

Link :: <https://mincong.io/2018/04/28/git-index/#:~:text=The%20index%20is%20a%20binary,Git%3A%20they%20are%20used%20interchangeably.>

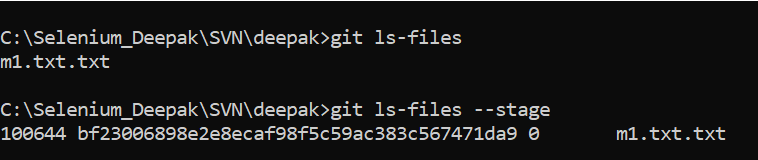
**Note:** 1. Whenever we add any new object in git using above command just after creation of .git repository, one new binary file named index under .git folder.

This index file having multiple name including staging area, index, etc.

The index contains all the information necessary to generate a single (uniquely determined) tree object. Hence, when we perform commit operation index file contents are getting considered to generate a unique tree for commit.

We can see the index file content using below command

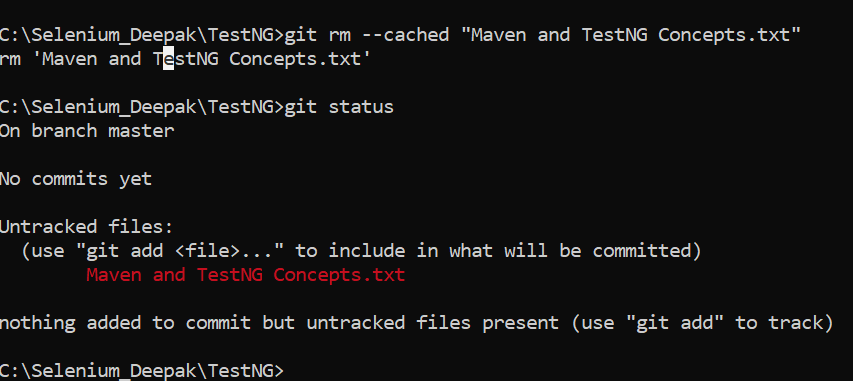
* git ls-files
* git ls-files --stage



2. unstage ( sending file from staging area to target area ) any file , execute below command. Double quote is not required if there is no space in file name.

Command: git rm --cached <filename> or git rm -c <filename>

Example: git rm –cached “Maven and TestNG Concepts.txt”



**Step 4:** After adding the file in staging area, we can commit the file in git repository using below command in order to save changes in local repository.

Command : git commit -m “Commit Msg” -m “Commit description”

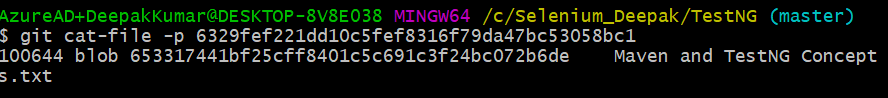
First -m stand for message and second -m is for description.

Example : git commit -m “file1 added” -m “ Module 1 file”

Once the successful commit will be executed , two additional objects will be created under .git/objects folder.

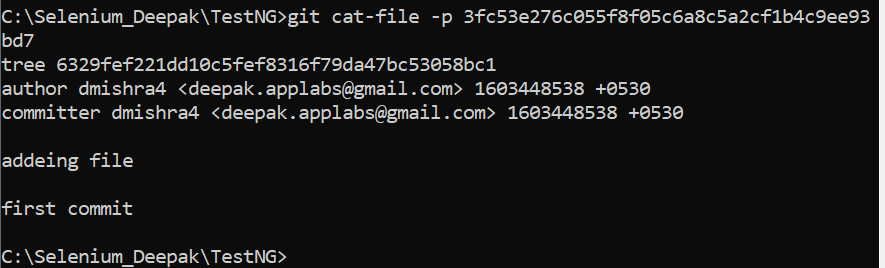
One object will be tree type will keep the information about list of file available during commit.

git cat-file -p 6329fef221dd10c5fef8316f79da47bc53058bc1



Second object will be commit type, which will keep the information about tree SHA code, author details, committer details, comment and description

git cat-file -p 3fc53e276c055f8f05c6a8c5a2cf1b4c9ee93bd7



git log --pretty="format:%h %ar %s"

**Step 5:** Again create one more file in working directory and add this in staging area and then do the commit.

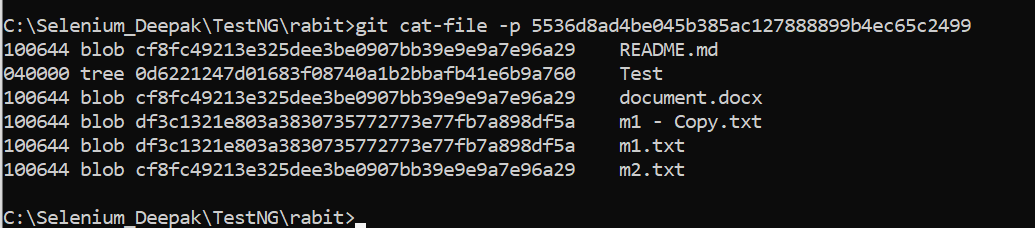
As said earlier, whenever either you add new file or modify existing file, new object will be created.

However, object name will be same if content of the two different type is same. In addition, even if two files having different same but same content, in that case also both file will share the same object.

Hence, new object creation will happen only if content of the either new or existing file is different. In other words, git object creation is depends on file content not the file name or type.

In addition, if you create a subfolder, then tree type object will be created for that folder in tree object.

In the below scrrenshot, Test is a folder and content of README.md, document.docx and m2.txt are same so their reference name is same i.e. cf8fc49213e325dee3be0907bb39e9e9a7e96a29



**Step 6:** if you want to **revert** any early commit, that we can do using below command.

Command : git revert <SHA1 for last commit>

Example : git revert 83ad862fb476248982ea7932afe62fed6df64d27

New reverted commit will be added after executing above command.

In this case, only commit object will be created, tree object will not be created for git revert command as earlier created tree object will be used.

However, you can also execute above command using below code.

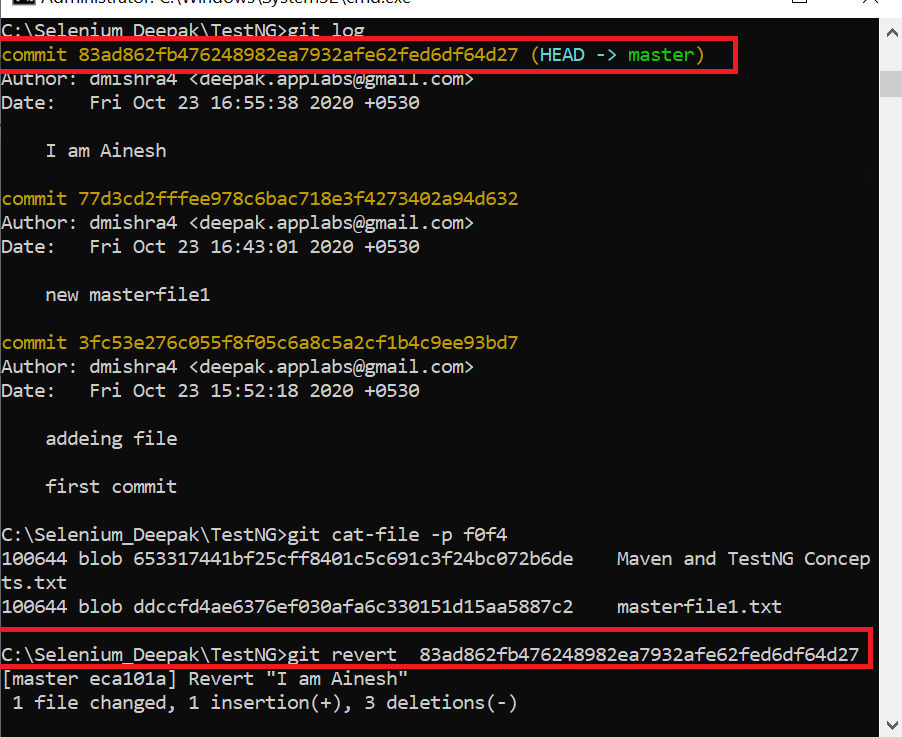
**Command**: git revert -n <SHA1 commit value>

In this case, first code will be reflected in staging area then we have to perform commit explicitly.

In addition, To cancel the revert operation, use below command

Command : git revert –abort

The file mode; one of 100644 for file (blob), 100755 for executable (blob), 040000 for subdirectory (tree), 160000 for submodule (commit), or 120000 for a blob that specifies the path of a symlink.



To cancel the revert operation, use below command

Command : git revert –abort

Step 7 : We can also add tag to any existing commit

There are two types of tags. Light weight tag and annonated tag.

Light weight tag will be connected to existing commit. This merely function as a pointer to a specific commit. We can’t add any message in that. Moreover, git is also not creating any additional object for the leight weight tag in git repository.

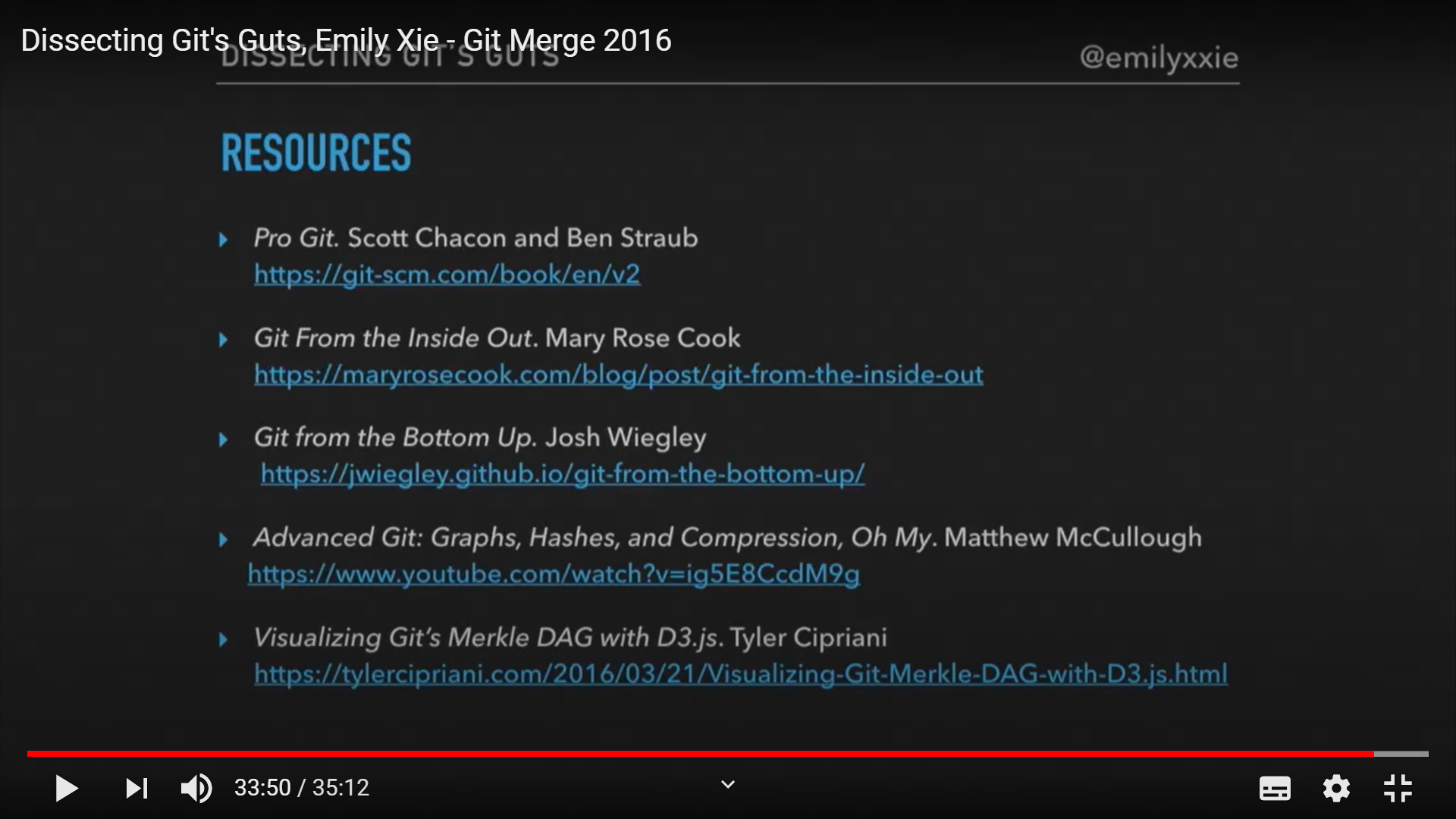
Command to create light weight tag is : git tag <tag\_name>

Whereas, annotated tag will also connected to an existing commit but with their own hash value. Annotated tag creates an additional object which store information associated with that tag. Annotated tag also provide the distails of tagger, tagged date and time, commit hashcode etc. ( See below scrrenshot )

Common : git tag -a <tag\_name> -m <any message>

Example : git tag -a release1 -m “ Release 1 final code”





############################# GIT Commands ############################

Git log command :

* git log ( display list of all commit on current branch)
* git log feature ( Will display commit on the feature branch)
* git log --oneline ( display all commit information in one line )
* git log --graph –oneline –decorate --all

Git diff command:

* git diff : ( Display the difference between working directory and staging area )
* git diff –cached ( Display the difference stages changes from last commit )
* git diff –staged ( --cached and –staged are synonyms )

Branch commands:

* git branch feature
* git checkout feature
* git checkout -b feature
* git switch -c feature ( we can use after git 2.23 version to create branch and switch together)
* git switch feature
* git branch (display all branch on local git repository)
* git branch -a (display all the branch available on both local and remote git repository)
* git branch -r (display only branch at remote repository)
* git branch --merged master ( list all branch name merged with master branch)
* git branch --no-merged master ( list of all branch name not merged with master branch)
* git branch -v ( To see last commit on each branch)
* git branch --move <old-branch-name> <new branch-name> ( To change branch name )
* git branch -d <branch name> ( Soft branch delete, will not allow to delete if branch is not merged)
* git branch -D <branch-name> ( Hard branch , will allow to delete without always)

**Git Undoing Command:**

* Git commit –amend ( To make change in last commit like changing coming message )

Below two commands will get executed to add new file in existing commit.

* git add m4.txt  
  git commit --amend
* git reset head --unstage (To unstage the file from staging area) or
* git restore –stage <filename> ( alternate of reset, Git version 2.25.0 introduced a new command: git restore. It’s basically an alternative to git reset)
* git restrore <file\_name> ( To discard changes in working directory )

**Note**: Its important to understand that git restore <file> is a **dangerous** command. Any

local changes you made to that file are gone — Git just replaced that file with the

most recently committed version. Don’t ever use this command unless you

absolutely know that you don’t want those unsaved local changes.

**Working with Remote:**

* git clone <URL>
* git remote: it will display the list of remote repository.
* git remote -v : it will display list of remote repository with URL of remote repository’s nickname.
* git remote add <nickname> <URL>

**Note**: git clone command implicitly add origin remote but to add a new remote repository, we used above command.

* Git fetch <fetch\_name> : it commands only download data into local repository, it does not merge . We have to manually merge after fetch command.
* git pull : it will automatically fetch and then merge that remote repository into the current branch.
* Git push <remote> <branch> ( Example : git push origin master, if you want to push master branch to your origin server )
* Git remote show < remote\_name> ( it will display more information about a particular remote)
* Git remore show
* Git remote rename <oldname> <newname> (rename remote repository)
* Git remote remove <remote repo name> (remove remote repository )

Merger command: