**First Run**

C:\Users\Mayank>cd c:\miniconda3\Git dir

c:\miniconda3\Git dir>git init #Initialized empty Git repository in c:/miniconda3/Git dir/.git/

c:\miniconda3\Git dir>git status

On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

**Add a file in staging area**

c:\miniconda3\Git dir>git add gittest.docx

c:\miniconda3\Git dir>git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: gittest.docx

**git rm –r –cached <foldername/>** to unstage to the files in foldername. –r stands for recursively.

Also you can use **git reset HEAD <file>** to unstage

You can use `**rm file name**` to permanently delete the fle. But if you do this after ‘**git add file**’, the file will be deleted but it’d still show up in ‘tracked list’. To overcome this, use above mentioned ‘**git rm -–cached file**’ to unstage.

**-------------------------------------------------**

**An experience –** By accident, a file with weird name #git docx# got created in Notes folder. Being unaware of this, I ran `**git add -f Notes**` followed gy `**git status**` I got the following response –

c:\miniconda3\notebooks\My Notebooks>git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: ML - Clustering - K-Means, DBSCAN and GMM.ipynb

renamed: Neural Networks and MLP.ipynb -> ML - Neural Networks.ipynb

modified: ML - Support Vector Machines.ipynb

modified: ML-Notes.ipynb

new file: Notes/#git docx#

The last line made me check the Notes directory and I found that there was indeed such a file there, I then came back to command prompt shell and ran ‘**git add .**’ followed by ‘**git status**’.

c:\miniconda3\notebooks\My Notebooks>git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: ML-Notes.ipynb

new file: Notes/#git docx#

modified: Notes/ml notes.docx

I deleted that file, then ran `git status` which gave me following response –

c:\miniconda3\notebooks\My Notebooks>git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: ML-Notes.ipynb

new file: Notes/#git docx#

modified: Notes/ml notes.docx

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

deleted: Notes/#git docx#

Then I tried following

c:\miniconda3\notebooks\My Notebooks>git rm --cached Notes/#git docx#

fatal: pathspec 'Notes/#git' did not match any files

c:\miniconda3\notebooks\My Notebooks>git rm Notes/#git docx#

fatal: pathspec 'Notes/#git' did not match any files

c:\miniconda3\notebooks\My Notebooks>git reset HEAD Notes/#git docx#

Unstaged changes after reset:

D Notes/#git docx#

Finally I tried this, which worked –

c:\miniconda3\notebooks\My Notebooks>git rm Notes/\\*#

rm 'Notes/#git docx#'

----------------------------

git add ‘file name’

git status #detailed

git status –s #short status

git diff #to see what we changed but not staged

git diff –-staged #diff between staged changes and last commit

git diff –cached

git commit

git commit –a #skipping the staging area

git log

git log –p

git log -1 #summary of latest commit. If ‘git log -2’ used, it would show summary of

last 2 commits.

untracked->staged->committed

Use `**git add .`** to stage all the modified files as shown by `git status`.

**A Note about git diff -** The git diff command is used when you want to see differences between any two trees. This could be the difference between your working environment and your staging area (git diff by itself), between your staging area and your last commit (git diff --staged), or between two commits (git diff master branchB).

**-----------------------------------------------------------------------------------**

**Moving Files**

Git as such doesn’t track directly renaming of files but it indirectly figures out that you have renamed a file.

You can rename a file by using `git mv` command like this -

**git mv file\_from file\_to**

**--------------------------------------------------------------------------------------------**

**Pushing repository**

**git remote add origin *url* #set the new remote**

**git remote –v #verify the url**

**git push origin master**

You may encounter these errors when trying to add a remote - ***Remote name already exists***

This error means you've tried to add a remote with a name that already exists in your local repository.

**A Note about** - `**git push origin master**’

That master is the <src> part of a [refspec](http://git-scm.com/book/ch9-5.html).

This means that your local master branch will be pushed to the master branch of the remote origin (orgin/master).

If you would have specified

git push origin master:my\_work

then you would have pushed your local master to origin/my\_work. If you don't use the :my\_work part, then the destination defaults to the same branch as given as source.

Just specifying

git push origin

will push every local branch that has a matching remote branch to that branch per default. **Not just the current branch**. This is the same as using git push origin :.

You can change this default with git config remote.origin.push HEAD, which would push the current branch to a remote branch with the same name.

­­­­------------------------------------------------

creating .gitignore file on Windows is tricky. One way to create .gitignore file is to first create a gitignore.txt file. Then open command prompt in the same directory by holding SHIFT key and select the command prompt by right clicking. Then run following command

**ren gitignore.txt .gitignore**

Secondly, while running git status, all files mentioned in .gitignore are ignored for tracking but git doesn't ignore the .gitignore itself. You can work around that by mentioning the .gitignore file in itself.

.gitignore doesn’t ignore the files which are already being tracked. Use git –r –rm cached <filename> to remove a file from ‘tracked list’ to ‘untracked list’

I created a .gitignore file with content copied from Github for Python. It started ignoring all .pynb and .txt files, not sure which rule is making it ignore those files. Anyway, I added following line at the end of .gitignore file –

!\*.pynb

This (! part) tells it to track all (\* part ) files ending with .ipynb files.

Use **git add –f <file>** to force add if that file is being ignored by .gitignore but you want to include that file.

-------------

In actuality, after I pushed local repository to Github, I added a README file to my repo directly on Github. This means, my local repository got ‘behind’ the Github repository. Then I made some local changes to repository and committed them. When I tried to push them I was greeted with the error message which implied that because there are changes on the remote branch that you don't have yet locally.

To sync both repo, I used following command - `**git pull origin master**`. This commands synced both repo. Following this I ran `**git push origin master**` which pushed the changes I made earlier to my Github repo.

**Working with a branch –**

**Git branch – list all branch and also shows active branch**

**git branch <branch>** – to create a new branch

**git merge <branch>** - merge the branch(before doing that, first checkout to master)

**git branch –d <branch>** -to delete the branch

**git checkout <branch>** - move to that branch

**git checkout –b <branch>** - create a new branch and move to it.

Create a branch. Check out that branch. Work on some issue and commit. Move to master branch. Then run **git merge branch**.