# **Git and GitHub tutorial**

* Distributed Version Control System
* It helps in keeping track of the different versions of files
* Can easily find who introduced an issue and when
* Can be **rolled back** to the previous working state
* Easy file recovery

## **History of Version Control System**

* **Local VCS**

Databases to track files

Pros: Can track files and roll back

Cons: If we lose hard disk/ computer, everything is lost

* **Centralized VCS**

Programmer 1

Computer 1

Files pulled

Files pushed

Centralized server with files

Programmer 2

Computer 2

.

.

Programmer n

Computer n

.

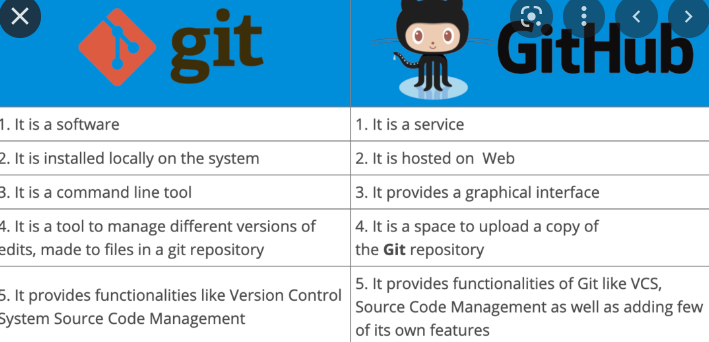
Programmers can pull files from the server and perform operations and push back the new versions of files to the server. But the issue is that if the central server is lost, file recovery is an issue.

* **Distributed VCS**

It has a complete history of files in every system. Even if the server loses the files, we can set up the server and add the backup.

**GIT is a software**

**GITHUB is a website that hosts GIT repositories, a cloud based hosting service**



**GIT features**

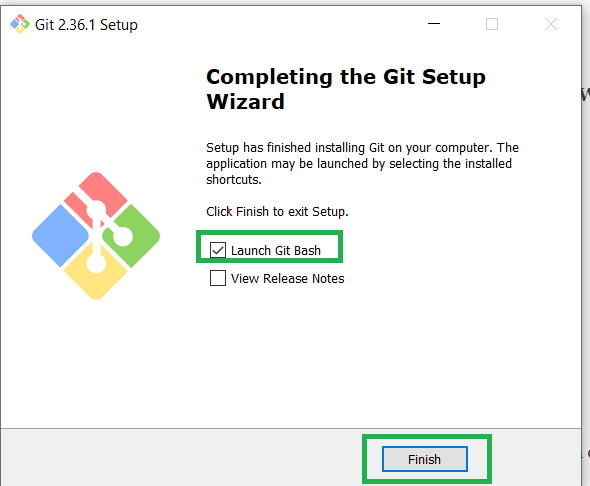
* Captures snapshots, not differences
* Almost every operation is local (Remote means related to the internet, something on the server, etc. not local)
* Git has integrity means it verifies the checksum of the file (to check if the file is exactly the same as sent by the programmer to download)
* It generally only adds data

## **GIT Install**

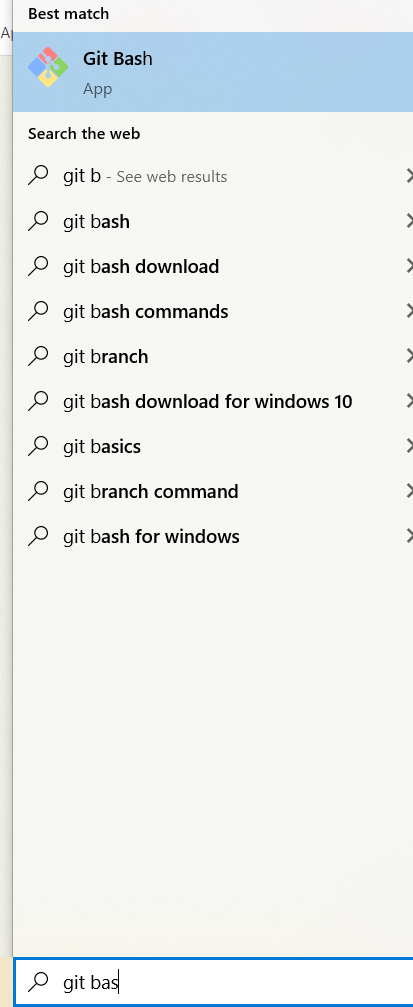
1. Search for **Git install** in any search engine.
2. Click to download the latest version.

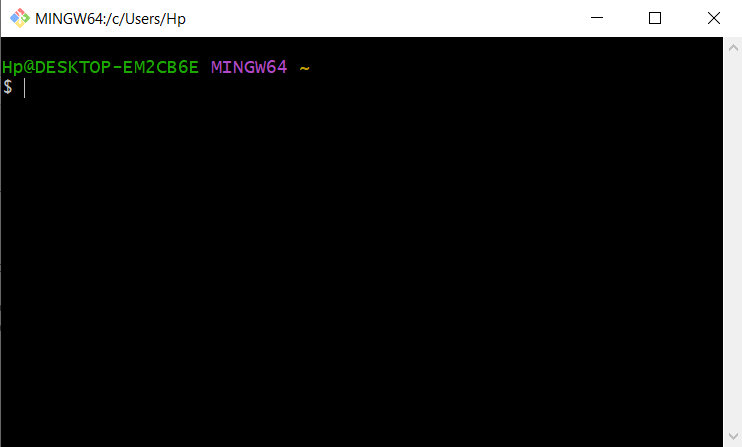


1. Download and set it up. In set up, check launch Git bash terminal and click finish.



1. Now, you can find the Git terminal when searching Git bash



1. Open Git bash terminal program. 

**Note:** Ctrl+ ‘+’ – zoom in

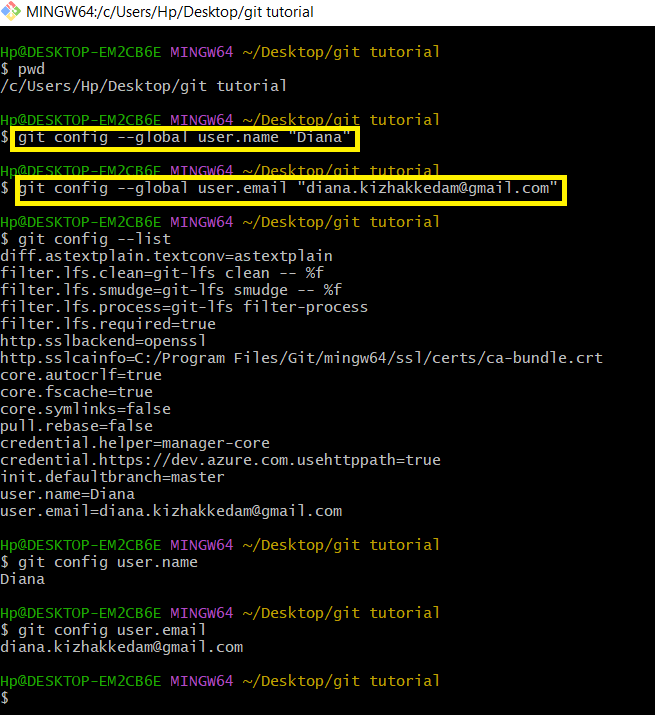
Ctrl+ ‘-’ - zoom out

1. Type git in the terminal and click enter.



1. User Configuration with name and email

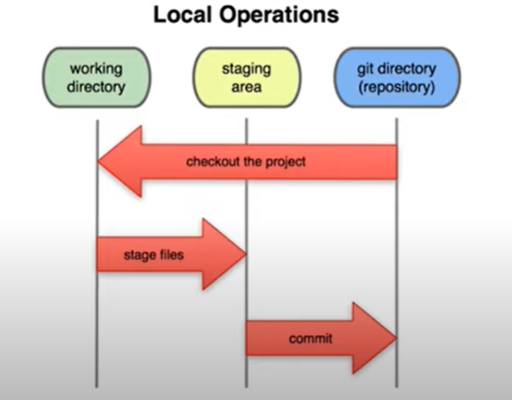
System Git config controls settings for all users and all repositories on your computer. **Global Git config controls settings for the currently logged in user and all his repositories.** **Local Git config controls settings for a specific repository**.



## **Git – Three Stage Architecture**

Commit- Snapshot

Stage- The files that need to be sent for next snapshot/commit

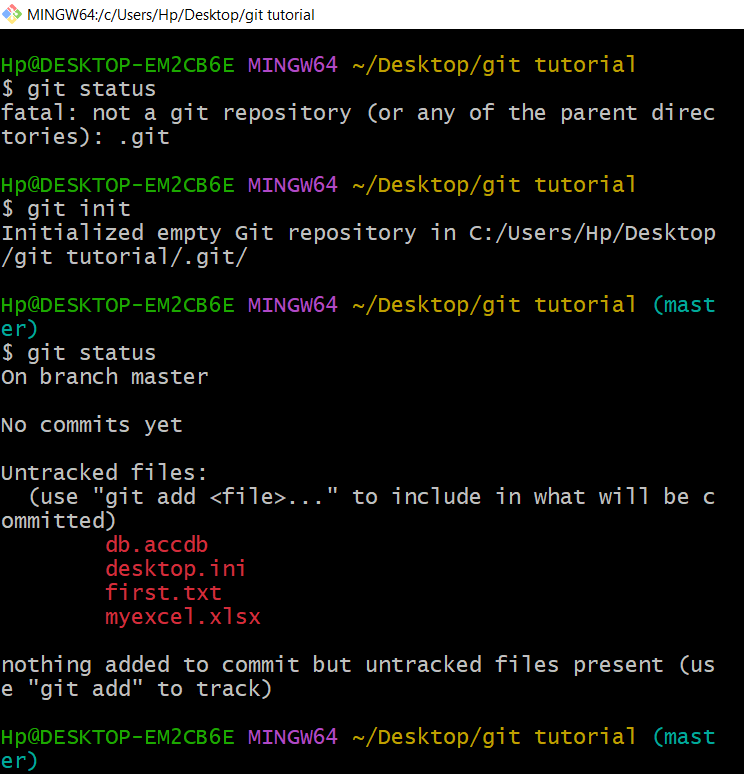


## **Sample Git Project**

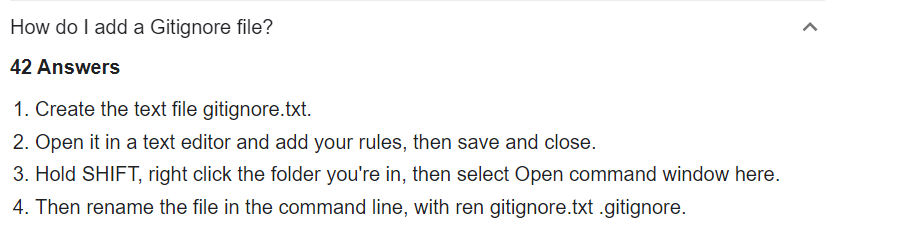
First, initialize git using the git init command

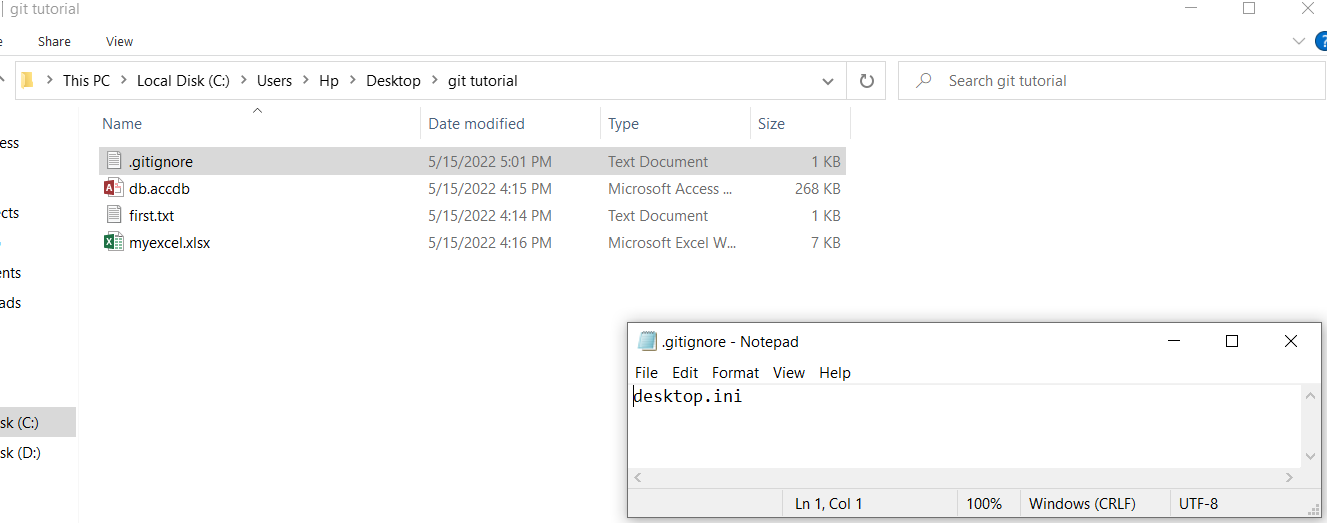
>> git init

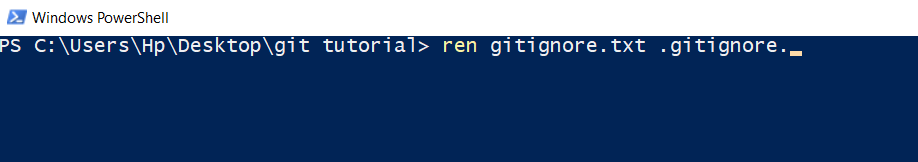
>> git status



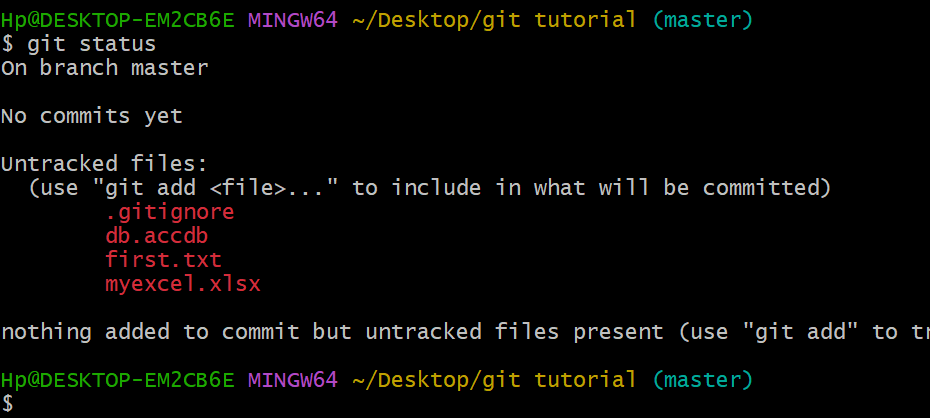
We have desktop.ini files present. The file desktop.ini is a system file generated in Windows. It appears in git status because git considers all files within its directory as potential repo files. What you want to do is add “desktop.ini” to a .gitignore file at the project root. This will make sure that git ignores any file matching that pattern. Once you add that to .gitignore and commit the .gitignore file, git will stop showing desktop.ini in the list of the item when you run git status.







In step 2, add the name of file to be added to .gitignore.txt file and save it, here it is desktop.ini.

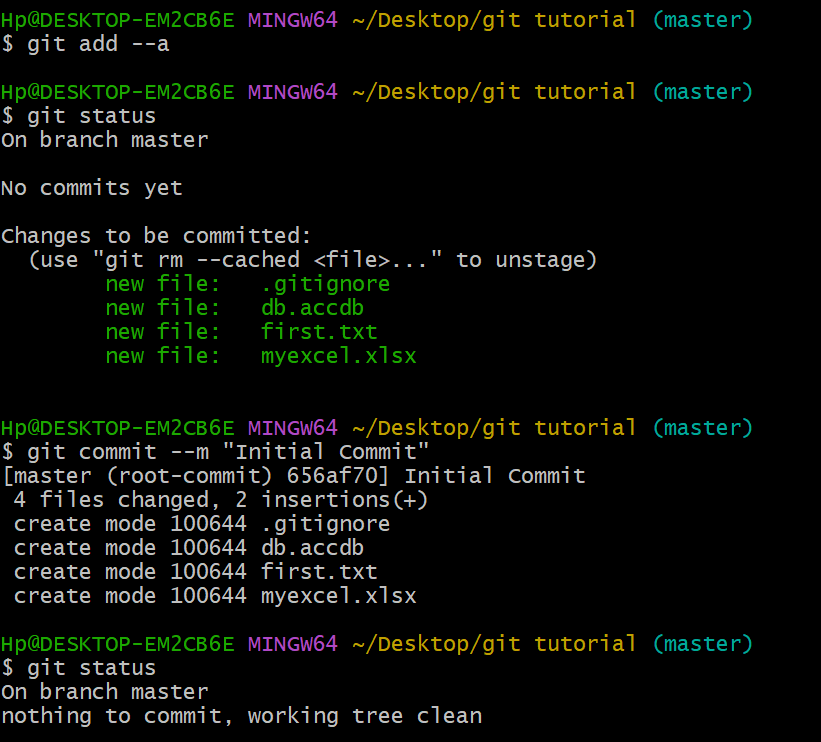


>> git add –a 🡪 to stage changes

>>git status

>>git commit –m “Initial Commit”

>>git status



Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git log

commit 656af7053d4ed58ef17315bb28cf475937647da6 (HEAD -> master)

Author: Diana <diana.kizhakkedam@gmail.com>

Date: Sun May 15 17:17:00 2022 +0530

Initial Commit

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes not staged for commit:

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

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

modified: first.txt

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

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes not staged for commit:

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

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

modified: first.txt

modified: myexcel.xlsx

Untracked files:

(use "git add <file>..." to include in what will be committed)

~$myexcel.xlsx

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

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git add first.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: first.txt

Changes not staged for commit:

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

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

modified: myexcel.xlsx

Untracked files:

(use "git add <file>..." to include in what will be committed)

~$myexcel.xlsx

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git commit -m "Commit done for first.txt"

[master 5ef9f86] Commit done for first.txt

1 file changed, 2 insertions(+), 1 deletion(-)

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes not staged for commit:

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

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

modified: myexcel.xlsx

Untracked files:

(use "git add <file>..." to include in what will be committed)

~$myexcel.xlsx

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

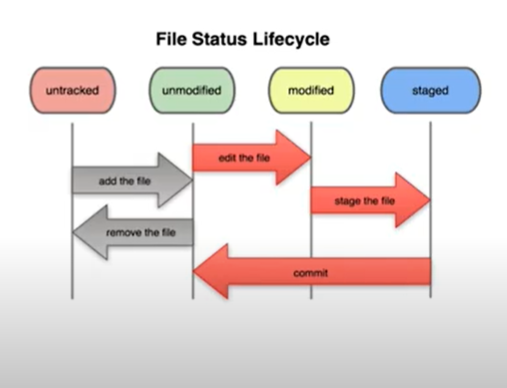
Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

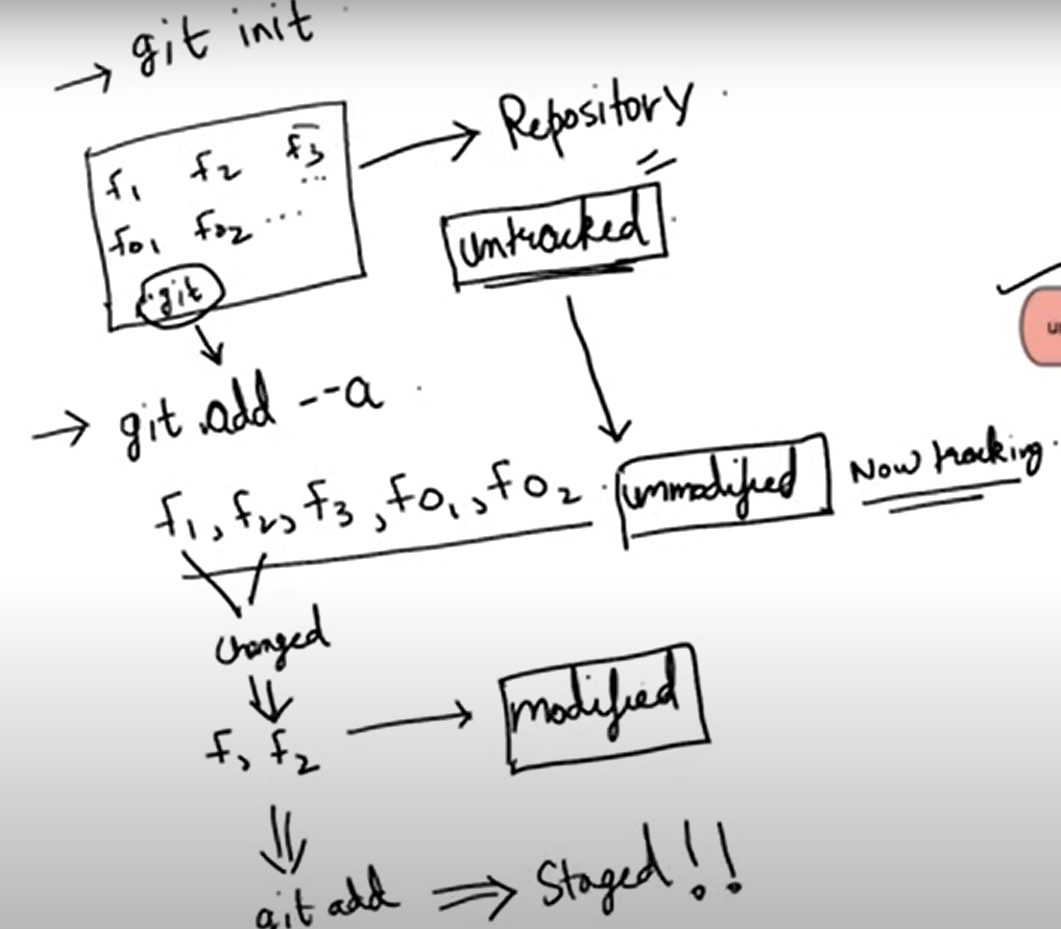
>> $ ls 🡪 to view files in a directory

>> rm –rf .git 🡪 Deletes our .git repository/ deletes the contents of a folder/ Remove git tracking

**Removes the file only from the Git repository, but not from the file system**. By default, the git rm command deletes files both from the Git repository as well as the file system. Using the --cached flag, the actual file on disk will not be deleted.

**The repository contains the folders and files. It has a hidden folder, .git which contains all the versions of files.**





**New file command**: >> touch filename

**To keep unwanted files, create a .gitignore file and keep inside**: >> touch .gitignore

\*\*\*If changes are added to the files listed in the. gitignore file, they will not be visible in the git status command i.e., it will be left ignored.

\*.log inside the .gitignore file means all the files with .log will be ignored.

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: static/third.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

$ git add --a

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: static/third.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: static/third.txt

Changes not staged for commit:

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

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

modified: static/third.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

$ git diff

**diff --git a/static/third.txt b/static/third.txt**

**index 03bd0d1..c49c6b3 100644**

**--- a/static/third.txt**

**+++ b/static/third.txt**

@@ -1 +1 @@

-asddedsd

\ No newline at end of file

+Halo Wie geht's

\ No newline at end of file

**Git diff** command to find the difference between the data added in the file in the working directory and the staged phase.

**(Modified – Staged(git add --a))**

**Git diff –staged** command to find the difference between the data added in the file in the previously committed working directory and the staged phase.

**(Commit - Staged)**

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git diff --staged

**diff --git a/static/third.txt b/static/third.txt**

**new file mode 100644**

**index 0000000..c49c6b3**

**--- /dev/null**

**+++ b/static/third.txt**

@@ -0,0 +1 @@

+Halo Wie geht's

\ No newline at end of file

**Direct commit skipping staging area:** >> $ git commit -a -m "type message"

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes not staged for commit:

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

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

modified: first.txt

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

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes not staged for commit:

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

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

modified: first.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

second.txt

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

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git commit -a -m "Direct commit skipping staging area"

[master 7f533c5] Direct commit skipping staging area

1 file changed, 1 insertion(+)

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

second.txt

nothing added to commit but untracked files are present (use "git add" to track)

**Remove a file:** >> git rm filename

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git rm third.txt

rm 'third.txt'

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: third.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git commit -m "removed third file"

[master fed40cb] removed third file

1 file changed, 0 insertions (+), 0 deletions (-)

delete mode 100644 third.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

nothing to commit, working tree clean

**Renaming a file:** >> git mv <filename> <new filename>

File is renamed and added to staging stage

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git mv first.txt first\_1.txt

Hp@DESKTOP-EM2CB6E MINGW64 ~/Desktop/git tutorial (master)

>> $ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

renamed: first.txt -> first\_1.txt

**Untrack a tracked file from .gitignore:** git rm --cached < filename> and commit it

## **Viewing and changing commits in git**

**>> git log**

>> **git log –p** ---complete status

>> **git log –p –n (n=1,2,3…)**

>> **git log --stat** --- Brief status

>> **git log --pretty=oneline** ---Brief status in one line

ddb315f6d3155a3b2ead4da4fed8567b359db85d TYP: resolve ignored mypy errors in core/describe.py (#46928)

5a599c19ec6ca97c6628853ee80cac2f65154119 ENH: allow non-nano in DatetimeArray, TimedeltaArray.\_simple\_new (#46901)

44dcef439664e8f095b9ca626bd4394fe00b8cd8 BUG: fix rolling with centering and axis=1 (#46135) (#46265)

78ee5552971ac1fed337a386040a20c8d8592d61 CLN: Remove special case for rank in groupby.ops (#46953)

d42a148cd83e06b5e5ef1fb6424e337d5b5efaa5 TYP: overload maybe\_downcast\_numeric and maybe\_downcast\_to\_dtype (#46929)

d4ba130e925aef6526ec061e1df4578abfde79b1 CI/DOC: Fix to\_hdf docstring validation (#46949)

422e92ab29ea279c95d212124d9ffe5988c34ab6 ENH: fields.get\_start\_end\_field support non-nano (#46902)

11462d67cd7b857d943e6ce1e27b29ea581eb1fa DEPR: numeric\_only default in DataFrame methods with None/True (#46906)

d5cf2b8674acb8a3b3eef469d26539a313517ca8 DOC: added index, dropna description for HDF methods #45030 (#46931)

009c4c622c739941ae3dfde4bc4117eb25172500 BUG: to\_json incorrectly localizes tz-naive datetimes to UTC (#46730)

8429441876b195edde66f8e4f495c8ba5f07b8d8 REF: stronger typing in \_box\_func (#46917)

>> **git log --pretty=full** ---Detailed status in one line

commit 101d4432cc92e8d382446c8ed37994f1c3a7aeeb

Author: jbrockmendel <jbrockmendel@gmail.com>

Commit: GitHub <noreply@github.com>

REF: simplify tzconversion (#47019)

commit 6f33c460e64a14e72333a6bf3b01fc60c6a75c9b

Author: auderson <48577571+auderson@users.noreply.github.com>

Commit: GitHub <noreply@github.com>

EHN: add same value count for skew & kurt (#46717)

\* add same value count for skew & kurt

\* add same value count for skew & kurt

\* update doc

\* merged

\* restore old comments

Co-authored-by: auderson <liao.renjie@techfin.ai>

commit e8308ceb3cf1652b34657a8b3a29b8dbfad33431

>> **git log --since=2.months / git log --since=1.days / git log --since=3.years --- Status filters**

$ git log --since=1.days

commit 9222cb0c7d8bacd9e6e96fc996c6ad6906b8e8f8 (HEAD -> main, origin/main, origin/HEAD)

Author: Torsten Wörtwein <twoertwein@users.noreply.github.com>

Date: Sun May 15 11:33:17 2022 -0400

TYP/CI: bump mypy&pyright (#46905)

commit 0e9a110b44cebb092a6e2a8453569b36b26315d5

Author: Matthew Roeschke <emailformattr@gmail.com>

Date: Sun May 15 06:49:48 2022 -0700

CI: Ensure no-use-pep517 with no-build-isolation with new pip (#47015)

Co-authored-by: Simon Hawkins <simonjayhawkins@gmail.com>

>> **git log --pretty=format:"%h -- %an"**

4a072fa0d0 -- Richard Shadrach

d5fcb40cf1 -- JHM Darbyshire

9f24918759 -- JosephParampathu

94280f021b -- arnaudlegout

>> **git log --pretty=format:"%H -- %ae"**

4a072fa0d0d34e83a0d80b1080846bf708bd7177 -- 45562402+rhshadrach@users.noreply.github.com

d5fcb40cf12487a2e28f2ca879bddba8ed232775 -- 24256554+attack68@users.noreply.github.com

9f2491875945d4b515f66491dbe507632060d6bf -- 88171669+JosephParampathu@users.noreply.github.com

94280f021bda7fb23e26db6e485e5367f2de8963 -- arnaud.legout@inria.fr

>> **git commit –amend – changes/additional messages are added to the file.**

Click ‘i’ to insert the message and esc and :wq to exit from the file

## **Unstaging and unmodifying the file in git**

**>> If the files are in staged phase/ before**

1. **git restore --staged <filename>**

git restore --staged first.txt

1. **git checkout [<options>] [<branch>] -- <file>...**

$ git checkout -- first.txt

To open/ view the file: $ vi third.txt

**Removes the file from the Staging Area, but leaves its actual modifications untouched.**

By default, the git restore command will discard any local, uncommitted changes in the corresponding files and thereby restore their last committed state. With the --staged option, however, the file will only be removed from the Staging Area - but its actual modifications will remain untouched.

1. **Git reset .**

**>> If the files are in commit stage**

**$ git checkout -f second.txt** --- If the files are in commit stage

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git checkout -f second.txt

Updated 1 path from the index

## Usage Examples

To only unstage a certain file and thereby undo a previous git add, you need to provide the --staged flag:

$ git restore --staged index.html

You can of course also remove multiple files at once from the Staging Area:

$ git restore --staged \*.css

If you want to discard uncommitted local changes in a file, simply omit the --staged flag. Keep in mind, however, that you cannot undo this!

$ git restore index.html

Another interesting use case is to restore a specific historic revision of a file:

$ git restore --source 7173808e index.html

$ git restore --source master~2 index.html

The first example will restore the file as it was in commit #7173808e, while the second one will restore it as it was "two commits before the current tip of the master branch".

$ git reset --soft HEAD~1

Using the “–soft” argument, **changes are kept in your working directory and index.**

## **Working with remote repositories**

git remote add origin https://github.com/DianaKizhakkedam/Git-Demo.git

git branch -M master

git push -u origin master

>> You can also use the commands in main branch using -- git branch -M main

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git remote -v

origin https://github.com/DianaKizhakkedam/Git-Demo.git (fetch)

origin https://github.com/DianaKizhakkedam/Git-Demo.git (push)

Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub without supplying your username and personal access token at each visit.

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git push -u origin master

Enumerating objects: 15, done.

Counting objects: 100% (15/15), done.

Delta compression using up to 4 threads

Compressing objects: 100% (14/14), done.

Writing objects: 100% (15/15), 1.47 KiB | 150.00 KiB/s, done.

Total 15 (delta 1), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (1/1), done.

To https://github.com/DianaKizhakkedam/Git-Demo.git

\* [new branch] master -> master

branch 'master' set up to track 'origin/master'.

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ ssh-keygen -t rsa -b 4096 -C "diana.kizhakkedam@gmail.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/Hp/.ssh/id\_rsa):

Created directory '/c/Users/Hp/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/Hp/.ssh/id\_rsa

Your public key has been saved in /c/Users/Hp/.ssh/id\_rsa.pub

The key fingerprint is:

SHA256:iyeRJsH5NTyGkVfZzi6e1e5Ldupq3PjcDhrvLHpzXjA diana.kizhakkedam@gmail.com

The key's randomart image is:

+---[RSA 4096]----+

| .. ..o |

| . ..+. . . |

| + ..\* o |

| o + o o |

| . = S . .E |

| o o .. o .o |

| o o. =.++ o|

| o o BO+\* |

| .+o@%oo|

+----[SHA256]-----+

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

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

Agent pid 221

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ ssh-add ~/.ssh/id\_rsa

Identity added: /c/Users/Hp/.ssh/id\_rsa (diana.kizhakkedam@gmail.com)

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ tail ~/.ssh/id\_rsa.pub

ssh-rsa  diana.kizhakkedam@gmail.com

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git push -u origin master

Everything up-to-date

branch 'master' set up to track 'origin/master'.



## **Setting Alias in Git**

**>>$ git config --global alias.<aliasname> <original command keyword>**

Ex: $ git config --global alias.st status

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git st

git: 'st' is not a git command. See 'git --help'.

The most similar commands are

status

reset

stage

stash

svn

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git config --global alias.st status

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git st

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git st

On branch master

Your branch is up to date with 'origin/master'.

Changes not staged for commit:

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

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

modified: third.txt

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

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git add .

warning: LF will be replaced by CRLF in third.txt.

The file will have its original line endings in your working directory

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git st

On branch master

Your branch is up to date with 'origin/master'.

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: third.txt

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git config --global alias.unstage 'restore --staged --'

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git unstage

fatal: you must specify path(s) to restore

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git unstage third.txt

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (master)

$ git status

On branch master

Your branch is up to date with 'origin/master'.

Changes not staged for commit:

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

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

modified: third.txt

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

## **Creating and Switching branches in Git**

**>> $ git checkout -b develop -- to develop a new branch**

**>> $ git checkout master -- Switching to master branch**

**>> $ git checkout develop -- Switching to develop branch**

Hp@DESKTOP-EM2CB6E MINGW64 /d/Users/diana/git tutorial (develop)

**$ git branch**

\* develop

Master

## **Branching and Merging a production grade project**

>> git merge <branch name>

## **Errors encountered and solutions**

* Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ cd /d/Documents/Application tutorials/Git and Github/Sample\_project/

**bash: cd: too many arguments**

**Solution: Use double quotes for path - cd “<path>”**

**Reason:** The most obvious however is that you have a space in your directory name. For example, if you have a directory called directory with space, and you do cd directory with space, the bash will interpret this as the command cd, with three arguments:

* directory
* with
* space

Which is not what you want. You can force the bash to treat it as one single argument by quoting it: cd "directory with space"

Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ cd "/d/Documents/Application tutorials/Git and Github/Sample\_project/"

* **! [rejected] master -> master (fetch first)'  
  Solution: Use:  $ git pull  
  then:  $ git push  
  Alternate solution:  $ git push --force origin master**but this will result in removal of files in your github repository so better prefer the first one it will pull the file first to your local repository and then you can push your work.
* Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ git remote add origin https://github.com/DianaKizhakkedam/Git-and-GitHub-Tutorial.git

**error: remote origin already exists.**

Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ git remote rm origin

Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ git remote add origin https://github.com/DianaKizhakkedam/Git-and-GitHub-Tutorial.git

Hp@DESKTOP-EM2CB6E MINGW64 /d/Documents/Application tutorials/Git and Github (master)

$ git push -u origin master

git remote rm **removes all references to the remote repository**. It does not remove the repository from the remote server. To verify that the remote was successfully removed, use the git remote command to list the remote connections: git remote -v.

## **Git Commands One-liner**

* git
* pwd
* git config --global user.name “<User name>”
* git config --global user.email “<User email>”
* git config --list
* git config user.name
* git config user.email
* git status
* git init
* git status
* git add --a / git add .
* git status
* ls
* git commit -m “<Message>”
* git commit -a -m “<Message>” --- Directly from untracked/modified to commit
* git rm <filename>
* git mv <New name of the file> <old name of the file>
* git restore --staged <file name>
* git restore --source CommitNo <file name>
* git restore --source master~2 <file name>
* git commit --amend --changes
* git log
* git log-p
* git log -p -n
* git log --stat
* git log --pretty
* git log --pretty=oneline
* git log --pretty=full
* git log --pretty= format: “%h -- %an”
* git log --pretty= format: “%H -- %ae”
* git remote add origin “<URL>”
* git branch -M master
* git remote -v
* git push -u origin master
* git status
* git pull origin master
* git config --global alias.<aliasname> <original command keyword>
* git checkout -b <New BranchName>
* git checkout master
* git checkout <New BranchName>