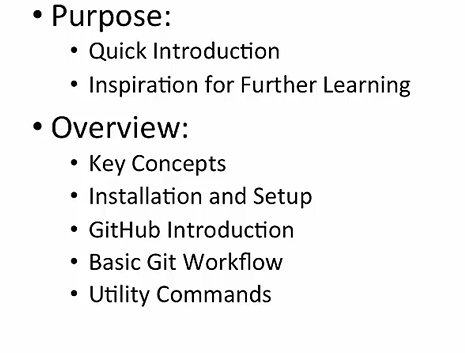
git & git hub



Documentation: <https://git-scm.com/book/en/v2>

Latest Version: (**2.37.3**) **64-bit** version

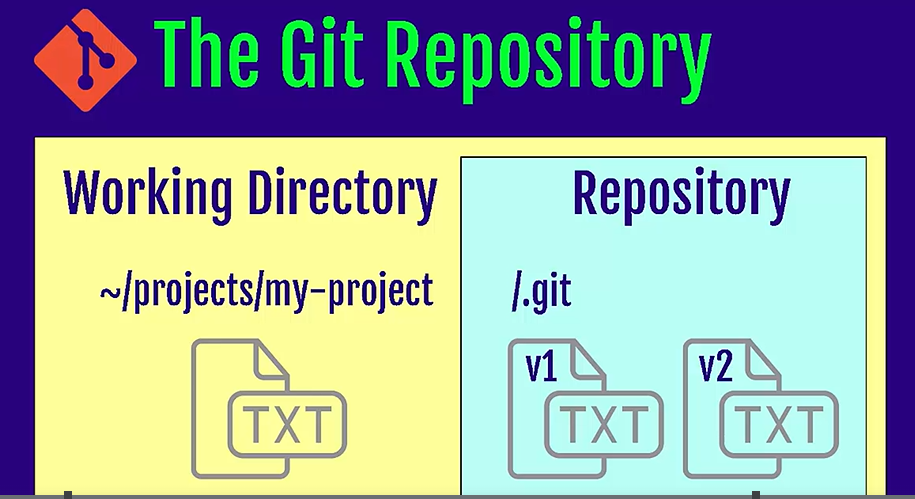


**What is GIT?**

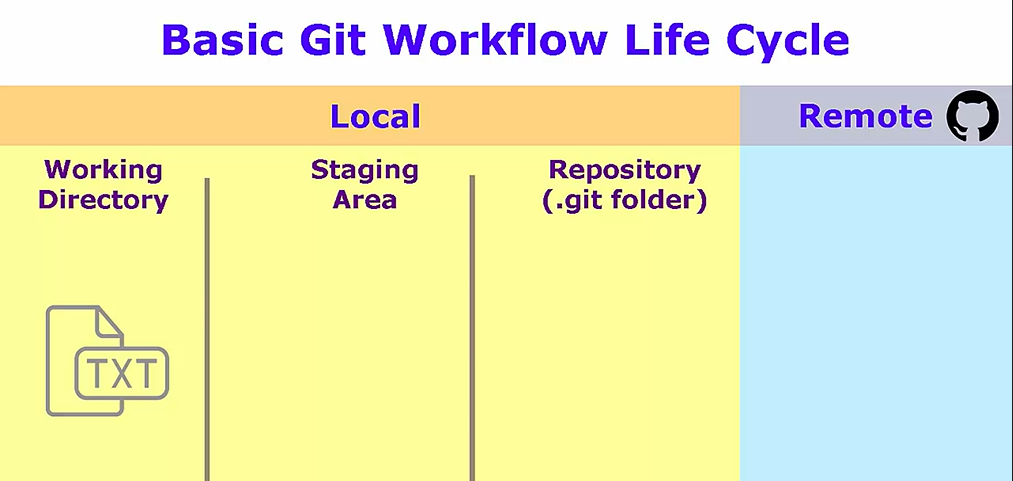
1. Distributed source control systems
2. Massively scales
3. Open Source
4. Developed for Linux project requirements
5. Most operations are local
6. Very fast
7. Active community

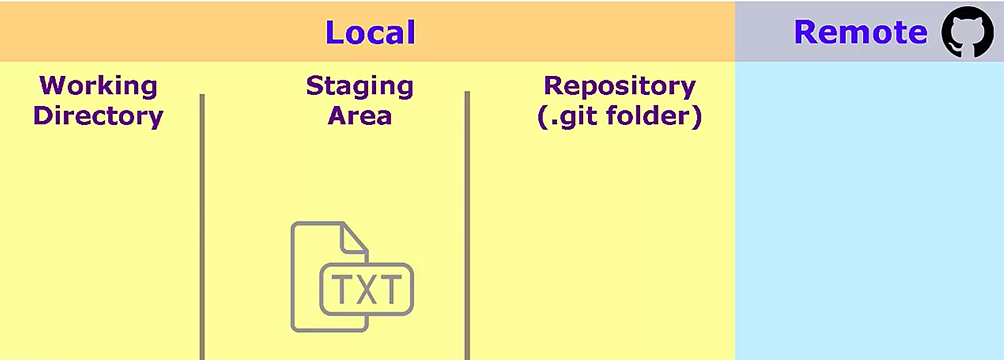
**Key Concepts:**

1. Repository contains files, history, config managed by git

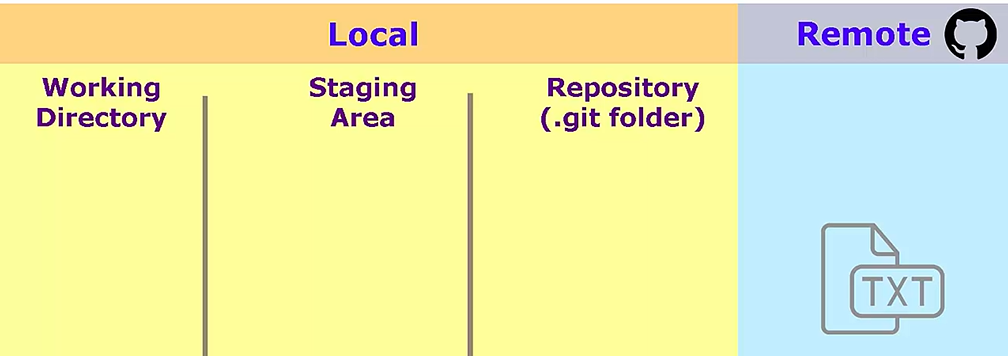


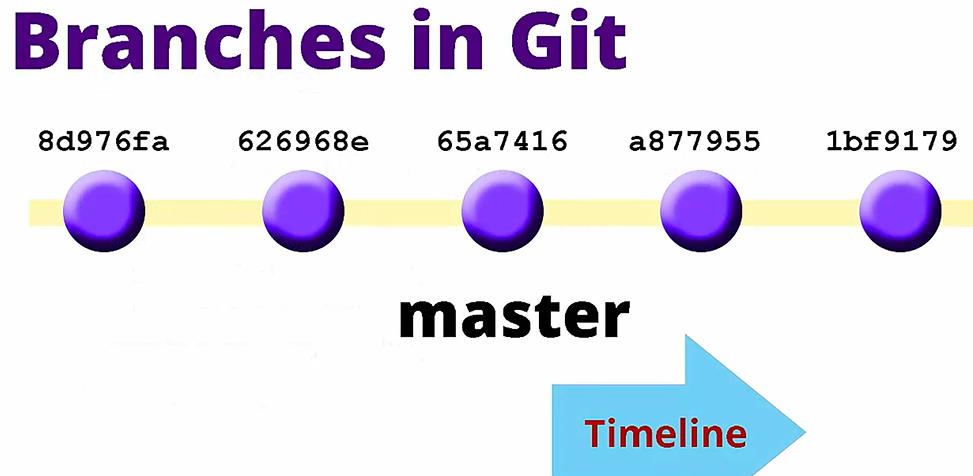
1. Three states of Git   
    a. Working directory   
    b. Staging area – pre-commit holding area   
    c. Commit – Git repository history





3. Remote repository (GitHub)

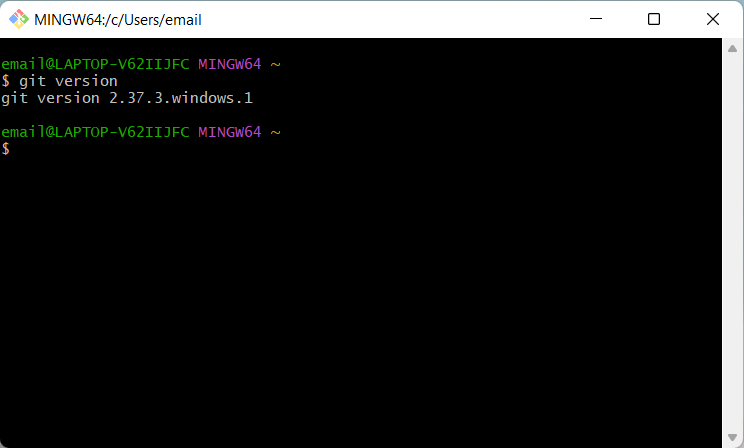


4.Master branch  
  


**Why command line?**- History   
- New features   
- Online help   
- Power  
- Consistent   
 a. Terminal on mac and Linux   
 b. Git Bash on windows   
  
**Installation:  
  
Windows:** git-scm.com

Step-1: Download the latest version of Git from this URL: <https://git-scm.com/download/win>

Step-2: After installation, Open the Git bash and type $ git version

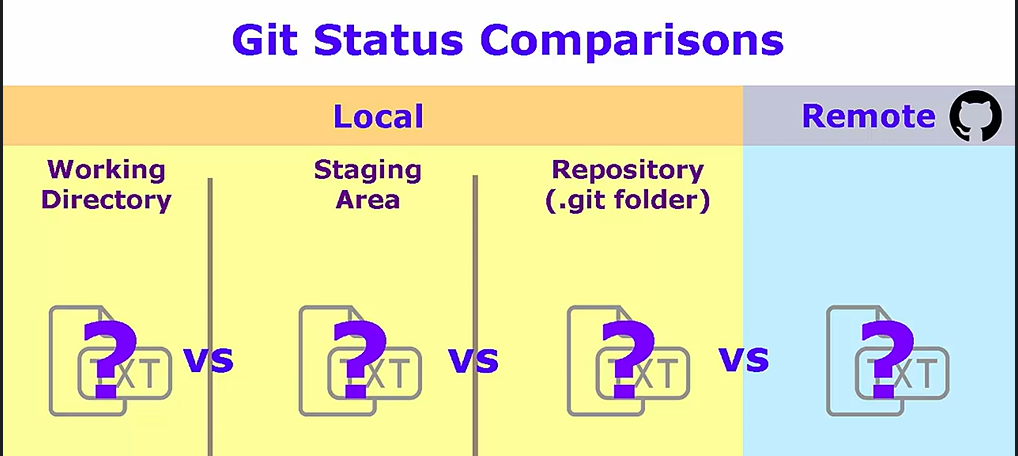


**Git workflow with GitHub**

1. Install Git as per above instructions
2. Now go to <https://github.com/> and create an account
3. Create a New Repository from your account interface (make it public/private)
4. Setup the project folder in local machine ( If not created or exist then move to the folder)

$ mkdir project\_name   
$ cd project\_name

1. Configure the Git with remote username and password to commit in the remote repo   
     
   $ git config --global user.name “email”  
   $ git config --global user.email “email”
2. Cloning from an existing git hub repo   
   $ git clone <repo\_url>
3. $ git status



**Git Setup:** After Git installation you can go and open the Git bash command prompt. And follow below steps to setup

1. **Identity setup**$ git config --global user.name "John Doe"

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

1. **Editor setup (optional)**$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin
2. **Configure default branch** **name** **instead of master**  
     
   $git config –global init.defaultBranch main
3. **Checking your settings**$git config –list  
     
   $git config user.name
4. **Getting Help**$git help // With Help verb: $ git help push

You can create Git repository:  
  
1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or   
  
2. You can clone an existing Git repository from elsewhere.

1. **Initializing a repository in an existing Directory**a. Move to the particular directory location   
    $ cd <directory\_path>  
     
    $ cd C:/Users/user/my\_project  
     
   b. Initialize the Git   
    $ git init  
     
   c. Add all the files and commit locally$ git add –-all // in case of specific file > $git add <file>

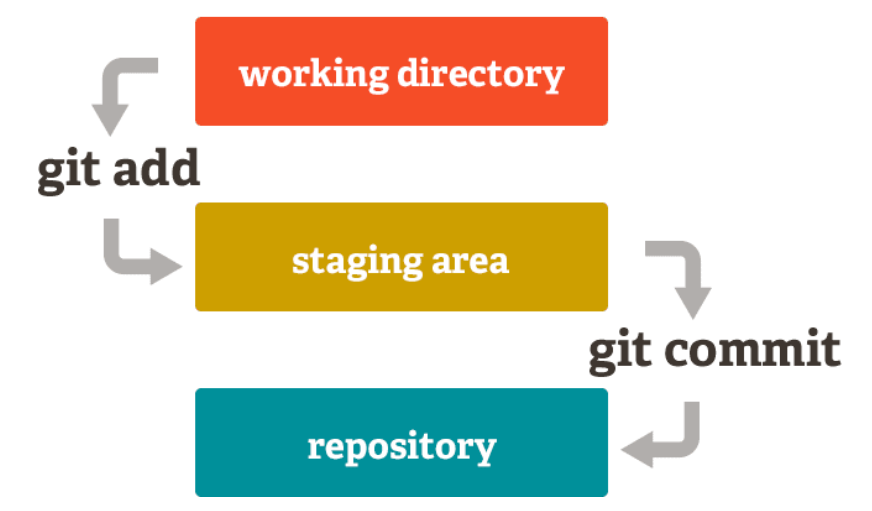
$git commit -m ‘Initial project commit’

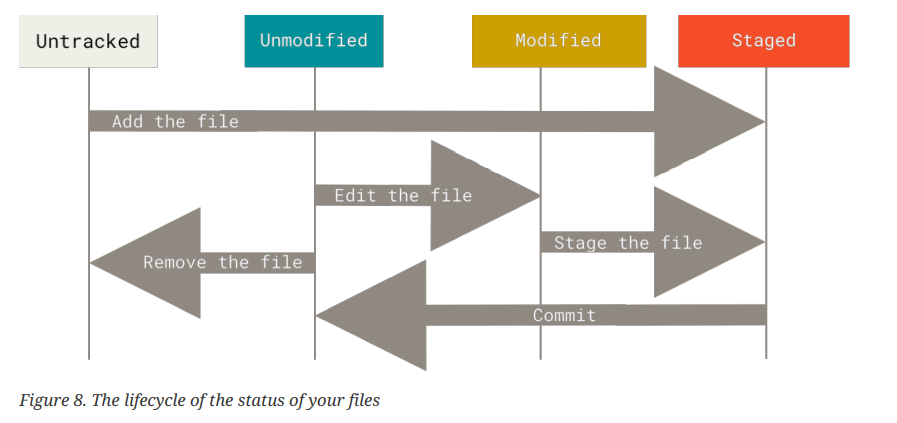
1. **Cloning an existing directory**a. Go to the locally created directory   
    $cd <path> // $ cd .. (precious directory)  
     
    $ git clone https://github.com/libgit2/libgit2 **mylibgit**  
     
   b. $ git init  
     
   c. $ git clone <path>

**Recording changes to the repository:**you should have a bona fide Git repository on your local machine, and a checkout or working copy of all of its files in front of you. Typically, you’ll want to start making changes and committing snapshots of those changes into your repository each time the project reaches a state you want to record.

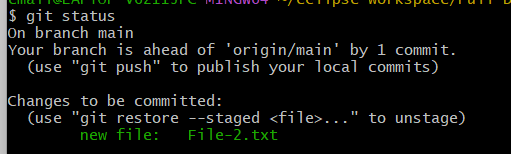
Remember that each file in your working directory can be in one of two states: tracked or untracked. Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.

The staging area is like a rough draft space, it's where you can git add the version of a file or multiple files that you want to save in your next commit

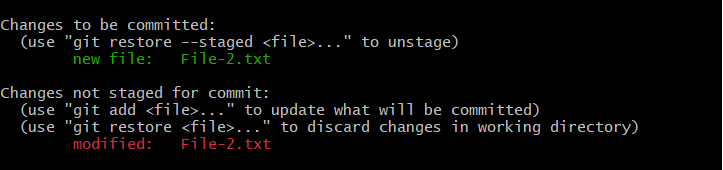




Checking the status of your files   
  
$ git status



Let’s change a file that was already tracked. If you change a previously tracked file called File-2.txt and then run your git status command again, you get something that looks like this:



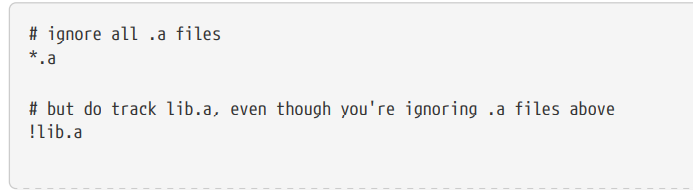
Often, you’ll have a class of files that you don’t want Git to automatically add or even show you as being untracked. These are generally automatically generated files such as log files or files produced by your build system. In such cases, you can create a file listing patterns to match them named .gitignore.

Here is an example .gitignore file:

$ cat .gitignore

\*.[oa]

\*~



To see what you’ve changed but not yet staged, type git diff with no other arguments:

**$ git diff**

If you want to see what you’ve staged that will go into your next commit, you can use git diff –staged

**$ git diff --staged**

Alternatively, you can type your commit message inline with the commit command by specifying it after a -m flag, like this:

**$ git commit -m "Story 182: fix benchmarks for speed"**

**Removing Files**$ rm PROJECTS.md   
$ git status

**Renaming files**$ git mv file\_from file\_to  
  
$ git mv file\_from file\_to

**Watch Log**

$ git log

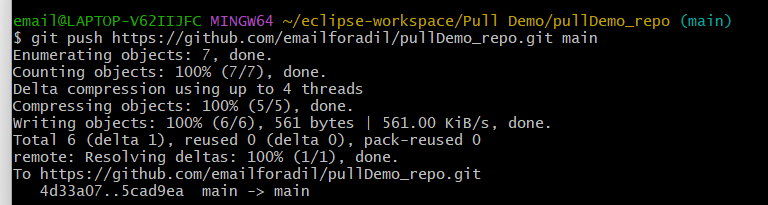
$ git log --since=2.weeks

As an example, if you commit and then realize you forgot to stage the changes in a file you wanted to add to this commit, you can do something like this:

$ git commit -m 'Initial commit'

$ git add forgotten\_file

$ git commit --amend

**Pushing to remotes   
  
$ git push <remote\_name> <branch\_name>**  
  
****

**Pull any changes from remote**To rename a branch, you'd use the same git push command, but you would add one more argument: the name of the new branch.

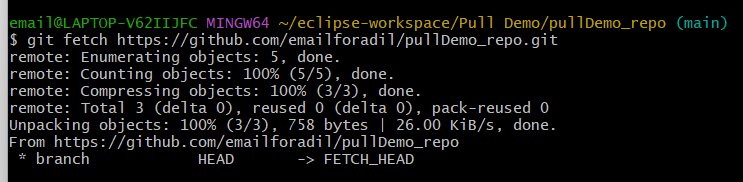
For example:

**git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>**

This pushes the LOCALBRANCHNAME to your REMOTENAME, but it is renamed to REMOTEBRANCHNAME.

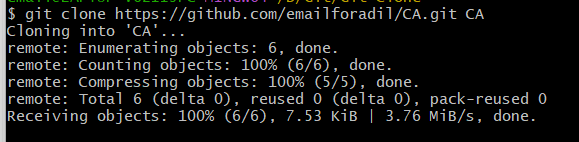
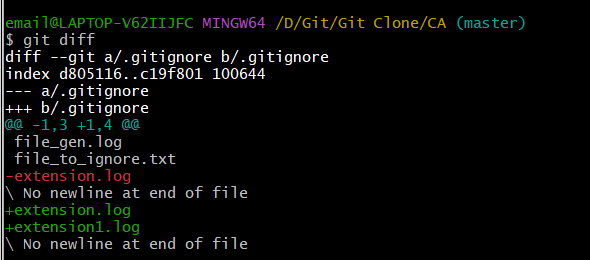
Use git fetch to retrieve new work done by other people. Fetching from a repository grabs all the new remote-tracking branches and tags *without* merging those changes into your own branches.

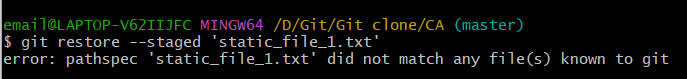
**$ git fetch *remotename***



**Rename Master branch to main  
  
$ git branch -m master main**

**Git useful commands:**

1. Move to any local directory   
     
   $ cd “/D” // $ cd “/<directory\_name>”
2. Set the username and email   
     
   $ git config --global user.name ‘Adil’  
   $ git config – global user.email ‘emailforadil@gmail.com’
3. List down all the config list   
     
   $ git config –-list
4. Set editor   
     
   $git config –global core.editor vim
5. Create a new directory and go to git bash type  
     
   $ git status
6. Initialize a new git repo   
      
   $ git init
7. Add all the new files in Git   
     
   $ git add –a
8. Create snapshot of all the staging files   
     
   $ git commit -m’This is initial commit’
9. To know the log   
   $ git log
10. Modify/Add in any existing file or new   
      
    $ git status   
      
    $ git add Important URL.txt  
      
    $ git status
11. Remove a git repository   
      
    $ rm -rf .git
12. Cloning a remote repository   
      
    $ git clone <URL> <local folder you want to create>  
      
    
13. Present working directory   
     $ pwd
14. List content   
     $ ls
15. Ignoring   
      
      
     $ touch .gitignore // add all the ignore files into this file  
      
     $ git status // all the files in .gitignore should not show here  
      
      
     Creating a new file   
      
     $ touch <file\_name.ext>  
      
     Ignore the file with specific extension  
       
     **\*.log**   
      
    // write it in .gitignore and it will ignore all the files with .log extension  
      
     Ignoring directory   
      
     **dir/**
16. Difference between working directory and staging area   
      
     $ git diff // diff between staging area and working directory  
      
       
      
       
     git add –a
17. Compare with the previous commit and with current staging area  
      
    $ git diff –-staged
18. Ignoring the staging area   
    $ git commit -a -m’Your message’
19. Moving and renaming files in Git  
      
    After renaming the file  
      
    $ git add --all  
    $ git status   
    $ git commit -m’Renamed the file..’  
    $ git status   
      
    **$ git rm <file\_name>** // remove and staged  
      
    **$ git mv <file\_name> <renamed\_file\_name>**
20. Log  
     $ git log  
     $ git log -p // detailed log   
     $ git log –p -3  
     $ git log –stat  
     $ git log –-pretty = oneline  
     $ git log –-since=2.days  
     $ git commit –ammend // to change in the last commit  
     // After changes you can press “i” then insert > “ESQ” > “shift+:” > “w” > “q” > Enter - You will exit from the editor
21. Quit from the log text  
     $ q
22. Unstaging and modifying files in Git  
       
     $ git restore –-stage <file\_Name>

  
  
 **Undo any changes in the file // with previous commit match** **$ git checkout – “file\_name”  
 $ git checkout –f** // Take changes from last commit

**Working with remote repository**