**My Git Notes**

* **Git config command :-**

$ git config --global user.name Avinash #for setting name and gmail globally

$ git config --global user.email [avinash.h@simformsolutions.com](mailto:avinash.h@simformsolutions.com)

git config --global alias.<alianname> <gitcommand> # Create shortcut for a Git command. E.g. alias.glog “log --graph --oneline” will set ”git glog” equivalent to ”git log --graph -- oneline.

* **Git init command :-**

**$ git init**

Initialized empty Git repository in C:/Users/avinash/Desktop/git/.git/

* **Git add command:-**

**We have to add at least one time to new file for changing its state from**  **untrack to**  **staging**

$ git add gitdemo\_file1.c #adding file to staging phase #also use for merge conflict

$ git add –A #for adding all files to staging phase

* **Git commit command:-**

$ git commit

$ git commit -m "Added more dummy files" #escaping vi editor and added comment directly

$ git commit –a –m “commit with avoiding staging phase” #we can skip 1 step of adding staging phase and it will combing staging plus commit command in one.

* **Git checkout command: -**

Use when we want our file back as last committed, by using this it discard current changes of the files and get back the content of last commit.

$ git checkout <filename>

$ git checkout –f #all files

* **Git log command: -**

Shows the logs of commits.

$ git log

commit a69d0c973774782d454ee7fb0fcbb8ad0608e42d (HEAD -> master)

Author: Avinash <avinash.h@simformsolutions.com>

Date: Fri Dec 17 13:12:47 2021 +0530

Added more dummy files

commit cf0755b417e7b489a952b3b96249b9ec2aa61641

Author: Avinash <avinash.h@simformsolutions.com>

Date: Fri Dec 17 12:57:49 2021 +0530

this is the first commit

* **$ git** log -p –1 #shows the last 1 commit and diff’s output
* **Git diff command: -**

Diff command compare the current directory with the staging

avinash@SF-CPU-315 MINGW64 ~/Desktop/git (master)

$ git diff

diff --git a/gitdemo\_file2.c b/gitdemo\_file2.c

index d9c892c..5b0833d 100644

--- a/gitdemo\_file2.c

+++ b/gitdemo\_file2.c

@@ -5,6 +5,7 @@ int main()

{

printf("Hello this is 2nd dummy file ");

+ hey there

* **$ git diff –staged #Compare staging output with the last commit**
* **Git rm command: -**

$git rm <file name> #removes the files (at least 1 time file need to be added)

$git rm --cached <file name> #removes files from staging phase and put back to untrackeing phase

$git –rf .git #deletes the whole repository

* **Git ignore command: -**

We create a .gitignore file and in that file if anything we want to exclude from the getting to push or hiding from git then we put those file names in there.

For example,

in .gitignore file if we write log.txt and \*.cpp then it will ignore all the log.txt and cpp files. Like that we also give the path from where to ignore by “/”.

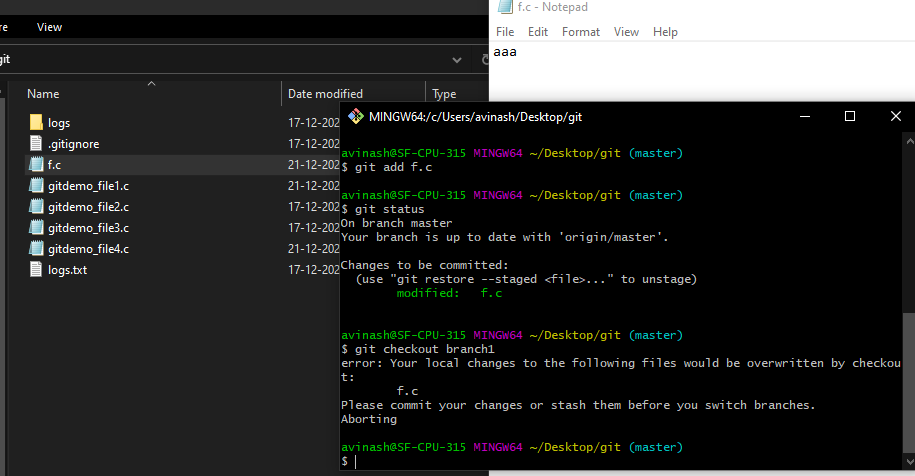
* **Git branch command: -**

$ git branch <branchname> #creates the new branch independent of master.

$git checkout <branchname> #for switching the branch.

$git checkout –b <branchname> # creates the new branch and switch to the newly added branch

$git branch –v #show branches with last commits



Note: - If we changes the branch and created or modified file then make sure you commit that modification so that when you change the branch then in that branch you can’t see other branch’s file. If you can’t commit then you keep seeing your other branches files.

* **Git merge command: -**

$ git merge <branchname> #merging the branch

$ git branch –merged #see which branches are merged

$ git branch --no-merged #see which branches are unmerged

* **Git remote command: -**
* $ git remote add origin https://github.com/avihargun/gitdemo\_javagroup.git

#Add new link of respective repo

* git remote set-url origin [repo-url] #rewrting url name
* $ git remote –v #showing links

origin https://github.com/avihargun/gitdemo\_javagroup.git (fetch)

origin https://github.com/avihargun/gitdemo\_javagroup.git (push)

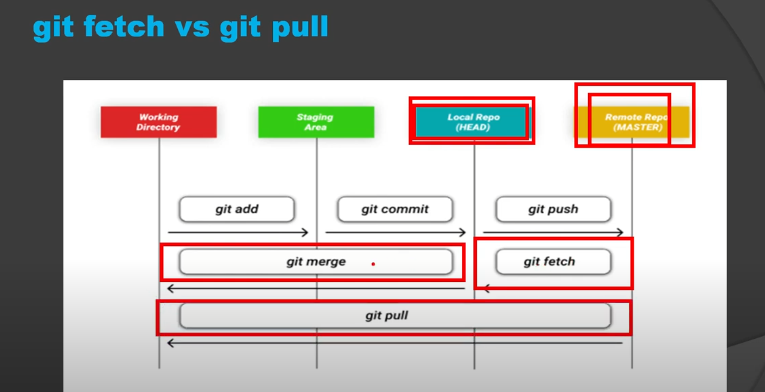
* $ git config --get remote.origin.url #also used for showing
* **Git push command: -**
  + $ git push origin version1 #push branch also
  + $ git push <origin-remote name> <version1 -branch name>

#other name #we can directly rename on GitHub

* + avinash$ git remote add bob /home/bob/myrepo

#witch this I need to only write bob instead of whole link for fetching or pulling etc.

* **Git fetch vs git pull : -**



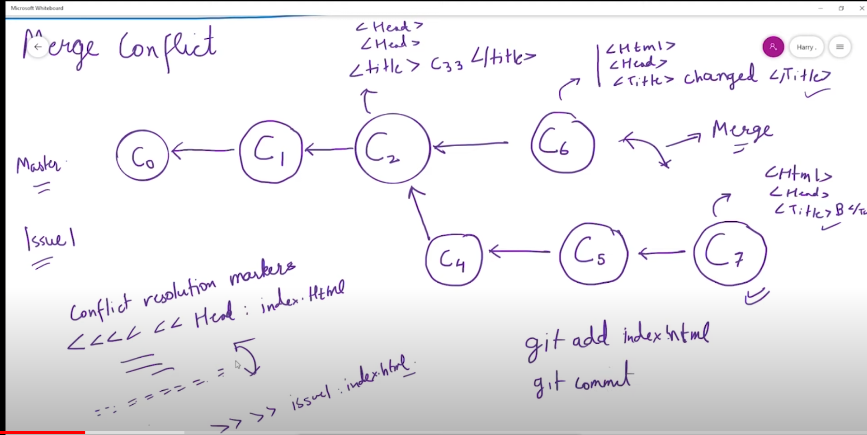
Git fetch only fetch remote repo to local and after merging it reflects to our working directory

Git pull =git fetch + git merge

$git pull <link>

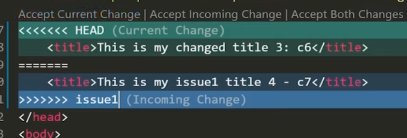
$git fetch <link> <branch>

* **Basic merge conflict : -**



In above diagram if we try to merge c7 and c6 then git is confuse which <title> part it writes ( both, c6’s or c7’s)

And if we open and try to see the files we can see <<<<<< or >>>>>> symbol that called conflict resolution markers



We can erase those parts which we don’t want and then run git add <filename> to resolute conflict and commit them.

* **Git clone: -**

$git clone <link>

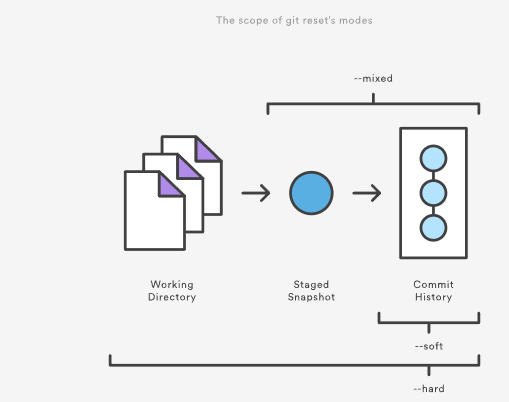
* **Git revert: -**

$git revert <commit> # Create new commit that undoes all of the changes made in , then apply it to the current branch.

$git revert -n <commit> # it not commit directly but undoes file change and we have can do commit explicitly after reviewing changes

* **Git reset command: -(by default –mixed)**

$git reset –merge



$ git reset <commit> # Move the current branch tip backward to , reset the staging area to match, but leave the working directory alone

1.delete commit 2.if any file in staging area it moves to untracked 3.do not delete out file which we are working just removes from staging to match with particular commit staging .

$git reset –hard <commit> # resets both the staging area & working directory to match. Deletes uncommitted changes, and all commits after .

$git reset –soft <commit> #leave as it is staging area and working directory just move to mentioned commit

* **Git restore command: -**

$ git restore --staged <filename> # unstage file but keep your changes

* **Git stash command: -**

$ git stash save "do some work" #will save your work and we can use later

$ git stash list #shows list of saved stash

$ git stash apply <stash@{0}> #apply that particular stash but still shows in list

$ git stash pop #pops and apply very first stash

$ git stash drop <stash@{0}> #for delete the stash

$ git stash clear #clear all the changes of saved stash

* **Git blame command: -**

The git blame command is used to examine the contents of a file line by line and see when each line was last modified and who the author of the modifications was.

$ git blame <filename> #shows full output

$ git blame –L 1,5 <filename> #restrict line 1 to 5

$ git blame -e <file> # -e option shows the authors email address instead of username.

git blame -w README.md

The -w option ignores whitespace changes. If a previous author has modified the spacing of a file by switching from tabs to spaces or adding new lines this, unfortunately, obscures the output of git blame by showing these changes.

git blame -M README.md

The -M option detects moved or copied lines within in the same file. This will report the original author of the lines instead of the last author that moved or copied the lines.

git blame -C README.md

The -C option detects lines that were moved or copied from other files. This will report the original author of the lines instead of the last author that moved or copied the lines.

* **Git rebase command: -**

git rebase <base> #Rebase the current branch onto. can be a commit ID, branch name, a tag, or a relative reference to HEAD.