GIT & GITHUB by Shivam

GitHub

LinkedIn

In powershell, `#` is used to comment a line.

Website Learn Git through Visualization

These are my notes in the beginning and I have attached some more printed codes at the end.

▼ GIT

```
git init
#This will initialise the repo.

ls -a
#This shows all the files that are hidden.

ls .\.git\
#Shows whats inside that folder. Here it'll show what's inside git folder.

touch names.txt
#Creates new file with that name.

git status
#To know the status of that folder and what changes we have made and not commmite
d.

git add .
#This will put all files into the staging area.

git add names.txt
#This will add this file into the staging area. Still commit has to be done.

git restore --staged names.txt
#This will removes that file from staging area and you will have to again add that
file to commit.

git commit -m "names.txt file added"
#-m is used to add any message.


rm -f .git/index.lock
#Another git process seems to be running in this repository.
#If this error comes then try deleting the This will remove the index.lock file.

git rm -rf .\names.txt
```

```
#To delete a file


notepad.exe names.txt
#Used to add something inside that file.

cat names.txt
#displays whatever is inside that file.

git log
#To know the history of all your commits.

q
#Press q to exit. git hist is using a pager tool so you can scroll up and down the
results before returning to the console.

git stash
#Takes your uncommitted changes (both staged and unstaged), saves them away for la
ter use, and then reverts them from your working copy.

git stash pop
git stash apply
#Throws away the stash after applying it, whereas git stash apply leaves it in the
stash list for possible later reuse. This happens unless there are conflicts after
git stash pop, in which case it will not remove the stash, leaving it to behave ex
actly as git stash apply.
```

▼ GITHUB

```
git remote add origin https://github.com/shivxmr/Classromm-Git.git
#Write this code in powershell. You will get the URL after creating a repository i
n GitHub.

#Origin means your own account.

git remote -v
#Will show all things attached to your folder.

git push origin master
#master is the branch name where you are pushing. Can be main/head etc.

git branch feature
#Creates a new branch named feature

git checkout feature
#Goes inside feature
git commit
#This commit will happen inside feature branch.
#Now suppose you want to come inside master branch again.
git checkout master
#Goes inside master.

git merge feature
```

```
#If your code is not finalised but you want other people to use feature branch cod
e then write this code. It will merge feature branch with the existing branch you
 are working on.
```

If you want to copy someone else's project into your account, `FORK` it through GitHub. By this way you can clone his project into your local. You will get a URL once you fork it, then write this code to get the project into your local.

```
git clone <URL>
#This URL you will get once you clone into your account.

git remote add upstream <URL>
#Do this to add the place from where you have cloned the project. The URL will be the
 original account's URL that has the original project.

git remote -v
#Will show all things attached to your folder.
#Now this will also show the upstream URL.

#Sometimes when there are conflicting requests you might have to force push.
git push origin main -f

  #If you want your cloned project to be up-to-date with the upstream project then you
can click FETCH UPSTREAM on top right. Code for this is:
git fetch --all --prune
git reset --hard upstream/main
git push origin main
#Now push these changes and your branch will be updated with the upstream branch.

#To do all the above things in a single line:
git pull upstream main

#If you don't want to do all these things just click on FETCH UPSTREAM and everything
 will be done.
```

GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name “[firstname lastname]”`
set a name that is identifiable for credit when review version history
`git config --global user.email “[valid-email]”`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m “[descriptive message]”`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch's history into the current one
`git log`
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`	delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`	change an existing file path and stage the move
`git log --stat -M`	show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`	system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`	Transmit local branch commits to the remote repository branch
`git pull`	fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`	apply any commits of current branch ahead of specified one
`git reset --hard [commit]`	clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`	Save modified and staged changes
`git stash list`	list stack-order of stashed file changes
`git stash pop`	write working from top of stash stack
`git stash drop`	discard the changes from top of stash stack