All-in-one

git

Cheat sheet

SETUP:

#configuring user information used across all local repositories.

#set a name that is identifiable for credit when review version history

```
git config --global user.name “[firstname lastname]”
```

#set an email address that will be associated with each history marker

```
git config --global user.email “[valid email]”
```

#set automatic command line coloring for Git for easy reviewing

```
git config --global color.ui auto
```

MOST COMMONLY USED GIT COMMANDS:

#create new repository in current directory
```
git init
```

#add a file as it's ready for your next commit (stage)
```
git add <file>
```

#add all files as they are ready for your next commit (stage)
```
git add .
```

#show modified files in working directory, staged for your next commit
```
git status
```

```
#commit your staged content as a new commit snapshot
git commit -m “[descriptive message]”

# Create a new connection to a remote repository by giving it name and it's url
git remote add <remote_name> <remote_url>

# push your local branch to specified remote.
git push <remote_name> <branch_name>

# Download new changes from the branch_name on the remote.
git pull <remote_name> <branch_name>

#Display the entire commit history using the default format.
git log
```

CREATING REPOSITORIES:

```
#create new repository in current directory
git init

#clone a remote repository
git clone <url>

#for example cloning the entire axios repo locally
git clone https://github.com/axios/axios.git
```

STAGE & SNAPSHOT:

```
#show modified files in working directory, staged for your next commit
git status

#add a file as it's ready for your next commit (stage)
git add <file>

#add all files as they are ready for your next commit (stage)
git add .

#unstage a file while retaining the changes in working directory
git reset <file>
```

```
#diff of what is changed but not staged
git diff

#diff of what is staged but not yet commited
git diff --staged

#commit your staged content as a new commit snapshot
git commit -m “[descriptive message]”
```

BRANCH & MERGE:

#branches helps to encapsulate your code changes.

#list your all branches. a '*' will appear next to the currently active branch
```
git branch
```

#create a new branch at the current commit
```
git branch <branch_name>
```

#switch to another branch and check it out into your working directory.
```
git checkout
```

#Create and check out a new branch named [branch_name]. above two commands in one.
```
git checkout -b <branch_name>
```

#merge the specified branch’s history into the current one
```
git merge <branch>
```

INSPECT & COMPARE:

#Inspect the logs and differences

#Display the entire commit history using the default format.

```
git log
```

#Limit number of commits by <limit>. e.g. ”git log -5” will limit to 5 commits

```
git log -<limit>
```

#Only display commits that have the specified file

```
git log -- <file_name>
```

#display each commit to a single line

```
git log --oneline
```

```
# show the commits that changed file, even across renames
git log --follow <file>

# show the commits on branchA that are not on branchB
git log branchB..branchA

#show the diff of what is in branchA that is not in branchB
git diff branchB..branchA

#Show difference between working directory and last commit.
git diff HEAD

#Show difference between staged changes and last commit
git diff --cached
```

TRACKING PATH CHANGES:

```
# remove files & path changes

#delete the file from project and stage the removal for commit
git rm <file>

#change an existing file path and stage the move
git mv <existingpath><newpath>

#show all commit logs with indication of any paths that moved
git log --stat -M
```

SHARE & UPDATE:

```
#Retrieving updates from another repository and updating local repos
git remote add <alias><url>

#fetch down all the branches from that Git remote
git fetch <alias>

#merge a remote branch into your current branch to bring it up to date
git merge <alias>/<branch>

#Transmit local branch commits to the remote repository branch
git push <alias><branch>
```

SHARE & UPDATE:

#Rewriting branches, updating commits and clearing history

#apply any commits of current branch ahead of specified one
```
git rebase <branch>
```

#clear staging area, rewrite working tree from specified commit
```
git reset --hard <commit>
```

DELETE COMMITS HISTORY FROM GIT/GITHUB:

- *In case, accidently you've pushed any sensitive information to your github repository like your SECRET_KEY, API_KE, PASSWORDS or .env files, even after deleting it from github repo the information remains visible in previous commits.*

- *So, in that case it's better to delete all your commit history rather than deleing the whole repo.*

Deleting the .git folder may cause problems in our git repository. If we want to delete all of our commits history, but keep the code in its current state, try this:

```
# Check out a temporary branch to hold our commit:
git checkout --orphan TEMP_BRANCH

# Add all the files to the emp branch:
git add .

# Commit the changes:
git commit -m "Initial commit"

# Delete the old branch (most probably old_branch would be a master branch):
git branch -D <old_branch_name>

# Rename the temporary branch (TEMP_BRANCH) to master:
git branch -m master

# Finally, force update to our repository:
git push -f origin master
```

Let's connect on linkedin!

https://www.linkedin.com/in/shubham-waje/

Subscribe to the newsletter for programming related articles

https://blog.learncodeonline.in/