Git Basic Commands

Show us untrack files or files in the working directory

```
git status
```

Add all files Untracked files from you working to the staging area

```
git add .
Or
git add -A
```

Add only one file Untracked from you working to the staging area

```
git add <file name>
```

Remove a file in the staging, and put it back to the working directory

```
git rm --cached <file name>
```

To remove a file in the local repository, and put it back in the working directory

```
git rm --cached <file name>
```

Commit the changes from the staging area in our local repo or take a snapshot of the project. File here are ready to push to Github

```
-m = message
git commit -m "(committed message)"
```

Commit a specific file in the local repo

```
git commit <file name> -m "(committed message)"
```

See all the files that you commit in your local repository

```
git ls-files
```

See the log of all the commits with author information

```
git log
```

See the log of all the commits in one line with the commit ID

```
git log --oneline
```

See all the information about a particular commit

```
git show <commit ID>
```

Git diff

- The file can have 3 different stages: file in a working directory different from a file in the staging area, file in the staging area different from the file in the local repository.
- To see the difference between the file in the working directory and the same file that was committed already in the local repository, and we have not yet pushed it to the remote repository.
  - + = new content was add
  - = content was removed

NB: The better way to compare 2 files is to use `VS Code`. This is because while working in the real world, we might have a thousand lines of code to compare and the good tool here will be `VS Code`

Example:

```
rm -rf *
ls
echo "Hello Tia" > file.txt
git add file.txt
git commit -m "first commit"
echo "Hello John" >> file.txt
git diff file.txt
```

See any modification of a file in the local repository

```
git diff <file name>
git diff file.txt
```

See any modification of a file in the staging area

```
git diff --staged <file name>
git diff --staged file.txt
```

Remove a file from the local repository and working directory. We need to add it and commit it.

```
git rm -f <file name>
git rm -f file.txt
```

Delete a file in your local directory and keep a copy in your working directory

```
git rm --cached <file name>
git rm --cached file.txt
```

Delete a file in your working directory and keep a copy in your local directory

```
rm -rf <file name>
rm -rf file.txt
```

What's the difference between git fetch and git pull?

git fetch and git pull

- **Git fetch** really only downloads new data from a remote repository - but it doesn't integrate any of this new data into your working files. Fetch is great for getting a fresh view of all the things that happened in a remote repository. Git fetch will never manipulate, destroy, or screw up anything.
- **Git pull**, in contrast, is used with a different goal in mind: to update your current `HEAD` branch with the latest changes from the remote server. This means that pull not only downloads new data; it also directly integrates it into your current working copy files. This has a couple of consequences:
  - Since "git pull" tries to merge remote changes with your local ones, a so-called `merge conflict` can occur.
  - It's highly recommended to start a "git pull" only with a clean working copy. This means that you should not have any uncommitted local changes before you pull. Use `Git's Stash` feature to save your local changes temporarily.

Git pull will fetch and merge files to the local repository

```
git pull = git fetch + git merge
```

Update your local repository from the remote repository if there were any update

```
git pull origin
git fetch origin_git
git fetch origin_bitbucket
```

Check if there were any updates on the remote repository and it will not download. This will not download the changes

```
git fetch  origin
git fetch  origin_git
git fetch  origin_bitbucket
```