Ch 5 - .gitignore, untracking files and working with commits

Sometimes we want some files to be ignored from our project. eg - .tmp, .log, .dat, .ini, --pycache--

We don't need these files but they are still getting created. These files are to be ignored from the project. There is a special file .gitignore which helps use to avoid them to go to staging area.

At this moment many useless files are generated. Inside .gitignore, you can put the file/folder name that is to be ignored

Now git status command doesn't show the logs.

To ignore files of any extension → *.ini
*.log

Avoid a folder → FolderName/

→ Git automatically ignores blank Folders

Track a file of ignored extension → !rough.log

Ignore all .tmp folders in any directory → folder/**/*.tmp

Only a file added to staging area can also be unstaged
git restore --staged myFile.py

A file inside gitignore which was previously staged, on modifying it git status shows it is modified.

You need to remove this file from the tracking system git rm --cached something.txt

Changing a commit
git commit --amend -m "Changed commit"

Matching code with some commit
git checkout changed.py → This command matches changed.py with latest commit.

git checkout -f → This command revives you! It changes the code or images back it initial commit.
It is used to checkout all files

You can also match all files with older commits by commit-hash.

git checkout <commit-hash>

git log can be used to find the hash

$ git diff - This command compares working tree with staging area

This doesn't work if files are staged. You need to run the git restore command to unstage the file.

$ git diff --staged - Compares staging area with recent commit

$ git config --local alias.sts status.

→ Instead of git status you can now use git sts