***Git & GitHub: The Complete Walkthrough (with Branching Strategy)***

*Whether you're just starting out or looking to polish your Git & GitHub workflow, this guide is your one-stop shop for everything from initialization to advanced branching techniques. Let's break it all down in a practical, developer-friendly format.*

***What are Git & GitHub?***

* ***Git*** *is a version control system — like a time machine for your code.*
* ***GitHub*** *is a web platform that hosts Git repositories, adding collaboration tools like pull requests, issue tracking, and CI/CD pipelines.*

*Think of Git as your local brain, and GitHub as your team’s collective memory in the cloud.*

***Getting Started: Installing and Configuring Git***

# Install Git (Linux)

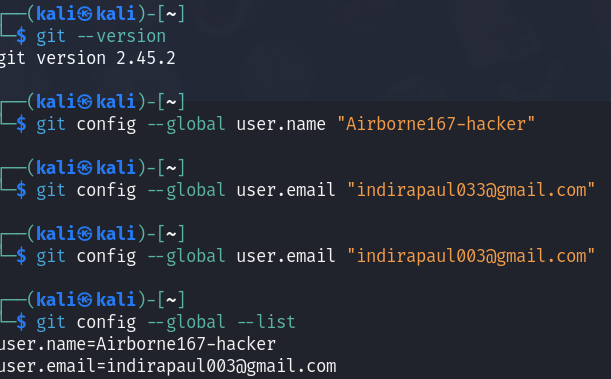
sudo apt install git

# Set your identity

git config --global user.name "My Name"

git config --global user.email "my@email.com"  
***This ensures every commit my make is properly credited.***





***Initializing a New Git Project***

*mkdir my-portfolio*

*cd my-portfolio*

*git init*

*touch index.html*

*git add index.html*

*git commit -m "Initial commit"*

***just created my first Git repository and made my first commit.***



***Connecting to GitHub***

***1st Step: Create a Repo on GitHub***

***Go to*** [***https://github.com***](https://github.com) ***→ Click New repository***

* ***Name: my-portfolio***
* ***Leave it empty (don't add README, .gitignore, etc.)***
* ***Click Create Repository***

***2nd step: Run These Commands in Kali Terminal Assuming I’m in my project folder (my-portfolio) and already ran git init, git add, and git commit.***

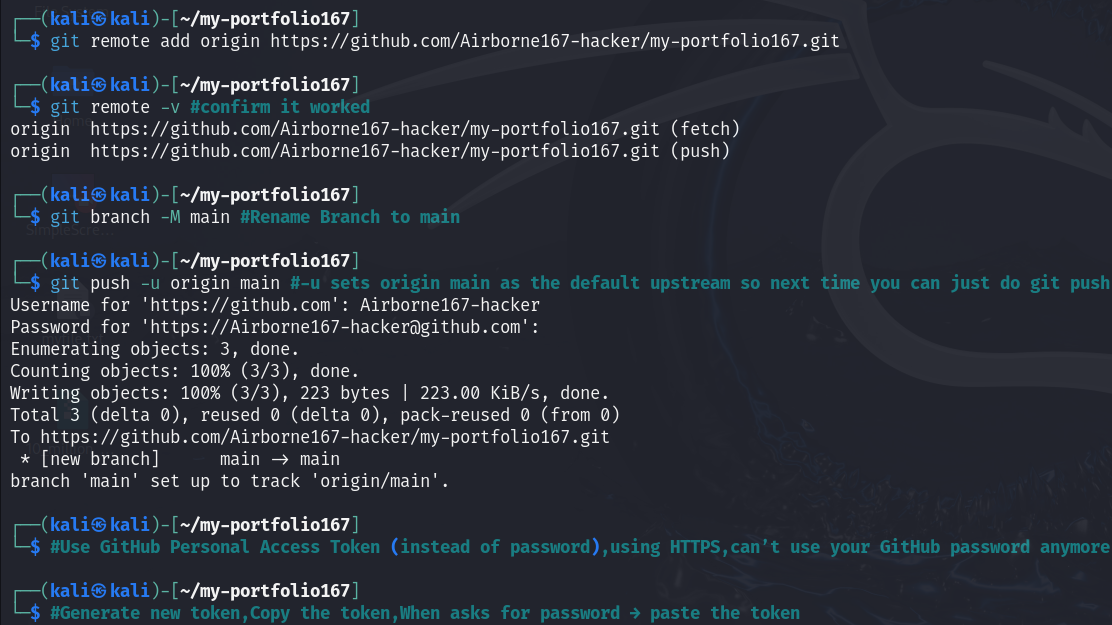
***Connect Local Repo to Remote***

*git remote add origin* [*https://github.com/myusername/my-portfolio.git*](https://github.com/myusername/my-portfolio.git)

*git branch -M main*

*git push -u origin main*

***local code is now live on GitHub***



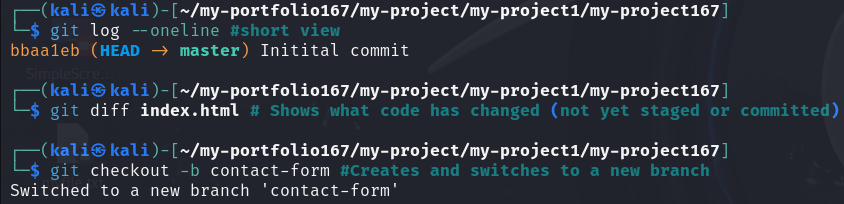
***Essential Git Commands (Use Every Day)***

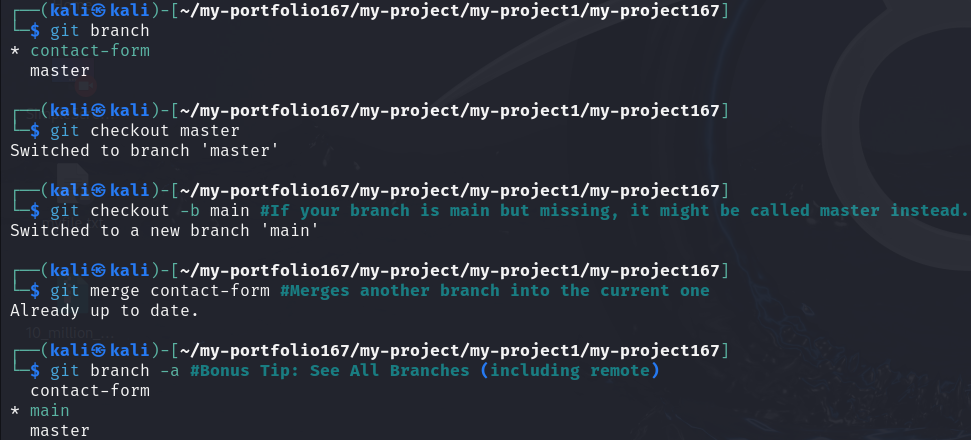
| ***Command*** | ***What It Does*** | ***Real Example*** |
| --- | --- | --- |
| *git status* | *Shows which files are staged or not* | *git status* |
| *git add* | *Stages all changed files* | *git add .* |
| *git commit -m "msg"* | *Saves changes with a commit message* | *git commit -m "Added navbar"* |
| *git log* | *Displays commit history* | *git log --oneline* |
| *git diff* | *Shows what has changed* | *git diff index.html* |
| *git checkout -b feature-x* | *Creates and switches to a new branch* | *git checkout -b contact-form* |
| *git merge branch-name* | *Merges another branch into current* | *git merge contact-form* |
| *git push origin main* | *Sends your changes to GitHub (remote)* | *git push origin main* |
| *git pull origin main* | *Pulls latest code from GitHub* | *git pull origin main* |
| *git fetch origin* | *Downloads latest changes (doesn't merge)* | *git fetch origin* |

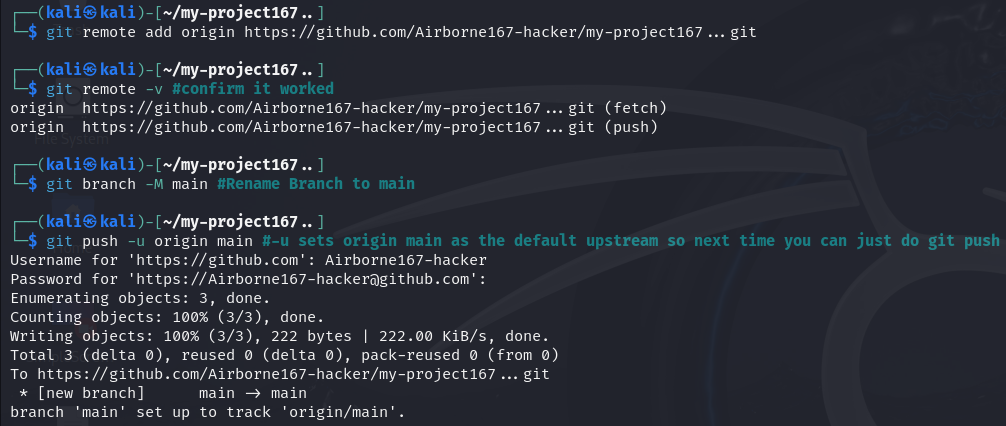
*git init Initializes a new Git git init*

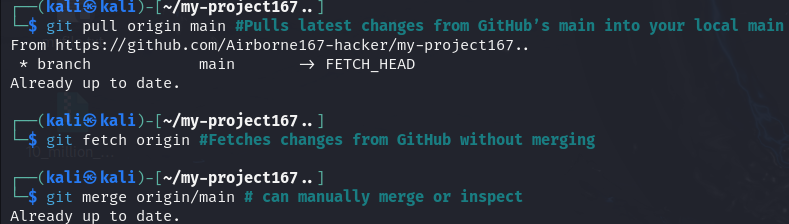
*repository in your folder*

***Use git pull when you want to get and apply the latest updates  
Use git fetch if you want to review updates before merging them***









***Git Branching : A branch lets you work on features independently***

*Creating & Switching Branches*

*git checkout -b feature/login*

*# Make changes*

*git add*

*git commit -m "Add login feature"*

*git checkout main*

*git merge feature/login*

***just worked in isolation, and then brought changes back into the main codebase.***

***Git Branching Strategy: Feature-Driven Workflow***

*A simple, powerful Git branching model:*

***main***

***│***

***├── dev***

***│ ├── feature/login***

***│ ├── feature/signup***

***│ └── bugfix/navbar***

***└── hotfix/prod-issue***

***Main Branch (main)***

* *Always production-ready*
* *Protected (no direct commits)*

***Development Branch (dev)***

* *Integration branch for all features*

***Feature Branches (feature/\*)***

* *One branch per feature*
* *Merge into dev when done*

***Hotfix Branches (hotfix/\*)***

* *Emergency fixes off main*

***Use Pull Requests (PRs) for code reviews before merging to main.***

***Viewing Changes & Undoing Mistakes***

*git log --oneline*

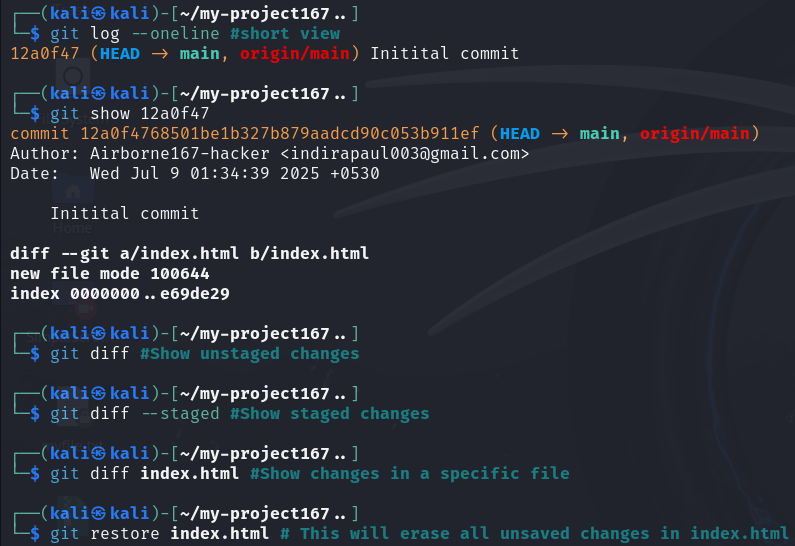
*git show <commit-id>*

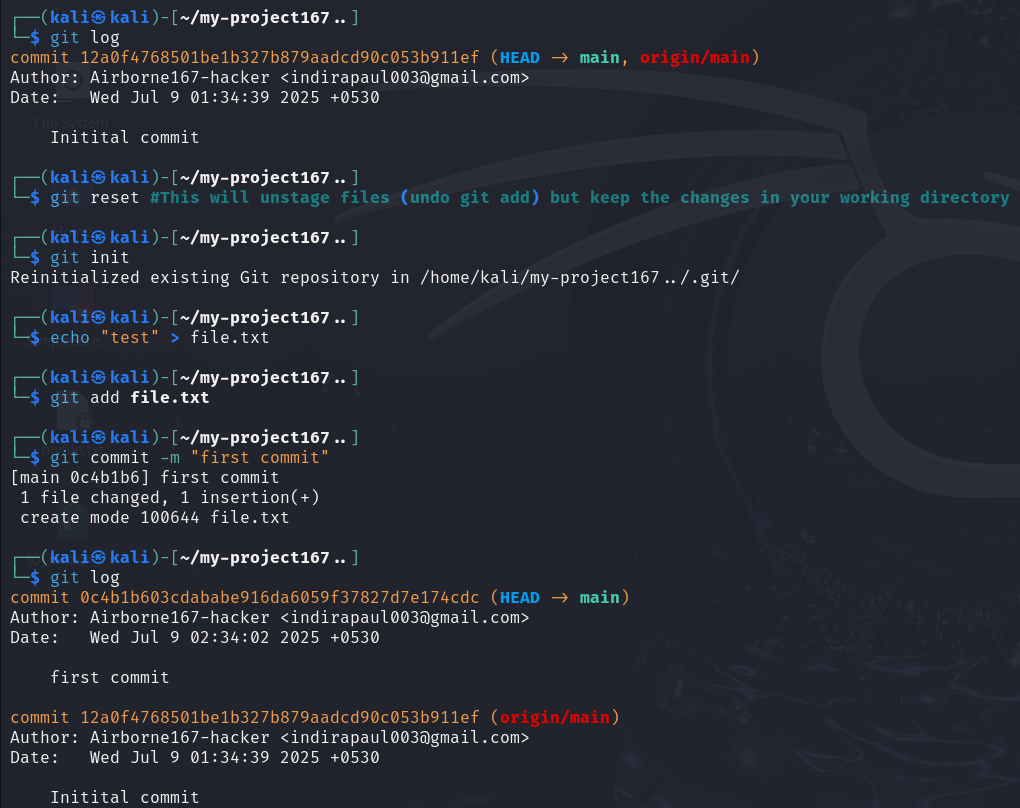
*git diff*

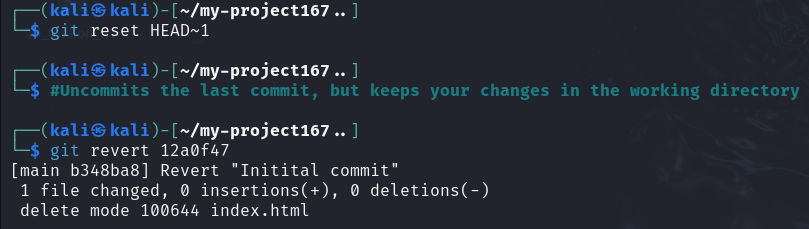
*git restore file.txt # Undo file changes*

*git reset HEAD~1 # Undo last commit (keep changes)*

*git revert <commit-id> # Undo via new commit (Creates a* ***new commit*** *that undoes the changes of a previous commit ---****safe for shared branches)***







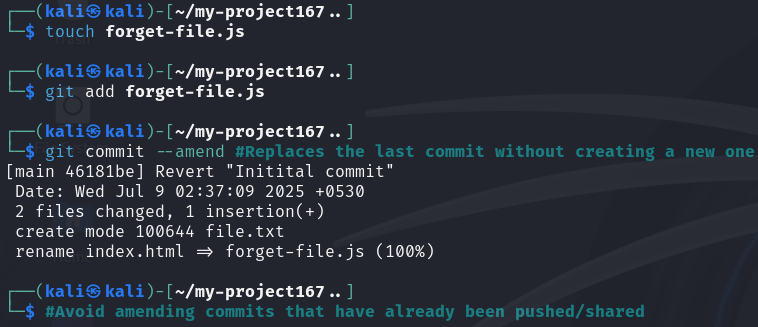
***Advanced: Rewriting History & Stashing***

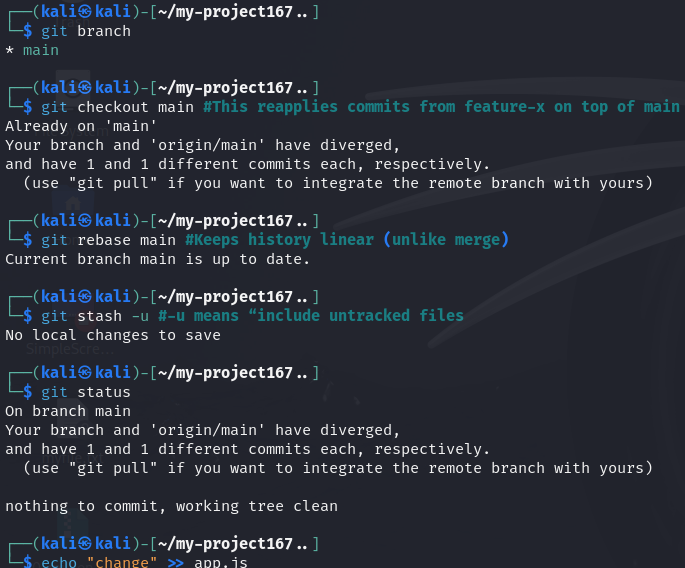
*git commit --amend # Edit last commit*

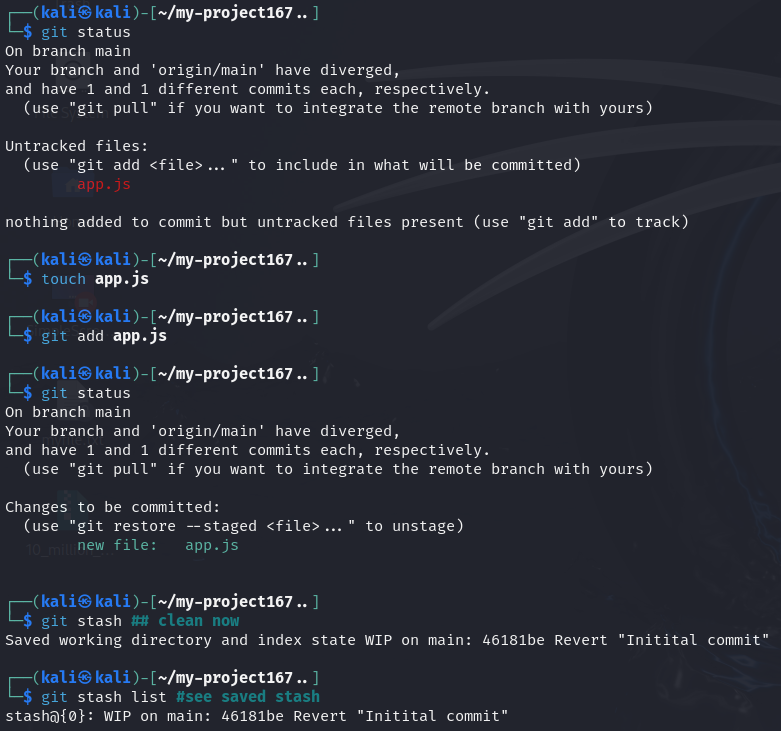
*git rebase <branch> # Replay commits over another branch*

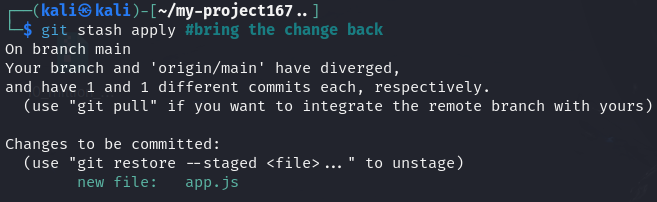
*git stash -u # Temporarily shelve changes [If I have untracked files (like new.js or .txt files), and want to stash them]*

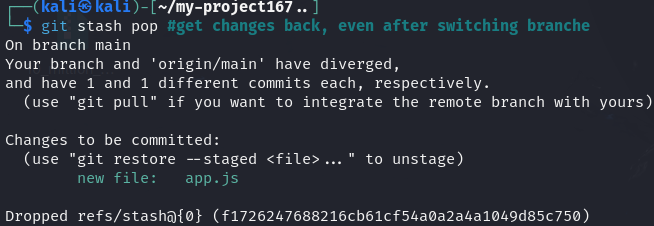
*git stash pop # Reapply stashed work*











***Team Collaboration with GitHub***

| ***GitHub Feature*** | ***Purpose*** |
| --- | --- |
| *Pull Requests* | *Propose & review code changes* |
| *Issues* | *Track bugs and tasks* |
| *Projects* | *Kanban boards for planning* |
| *Wiki* | *Share documentation* |
| *GitHub Pages* | *Host static sites* |
| *GitHub Actions* | *Automate testing & deployment* |

***Real-World Workflow Example***

*git checkout -b feature/search-bar*

*# Make changes*

*git add .*

*git commit -m "Add search bar"*

*git push origin feature/search-bar*

***Go to GitHub → Open a Pull Request → Team reviews → Merge to dev or main.***

***Git Cheat Sheet***

*# Basics*

*git init*

*git add .*

*git commit -m "message"*

*git status*

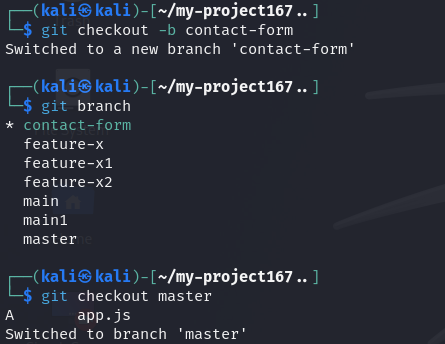
*git log*

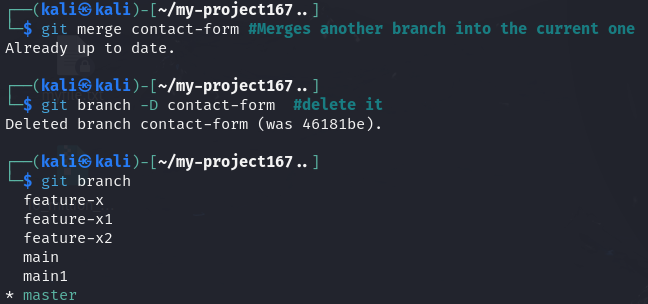
*# Branching*

*git checkout -b feature-x*

*git merge feature-x*

*git branch -d feature-x (****Use after merging the branch****)*





*# Remote*

*git remote add origin <url>*

*git push origin main*

*git pull origin main*

*# Undo*

*git reset --soft HEAD~1*

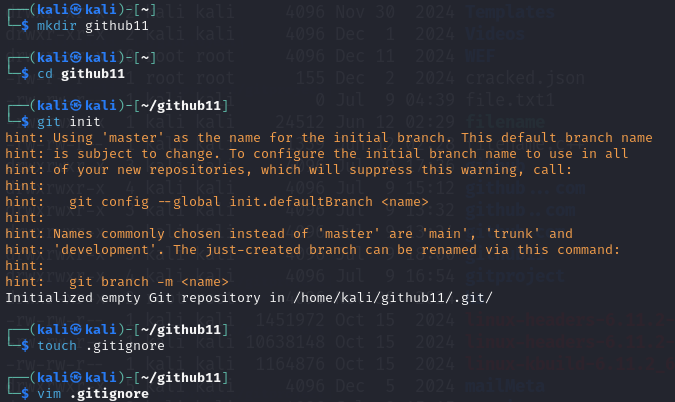
*git revert <commit>*

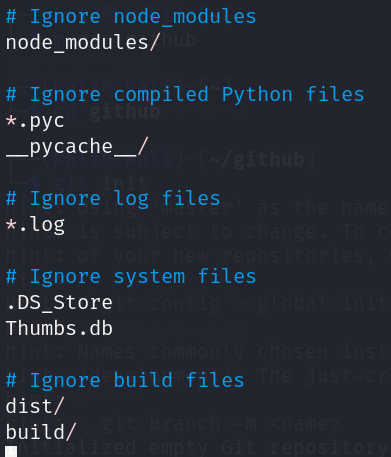
***1.Always branch from dev or main***

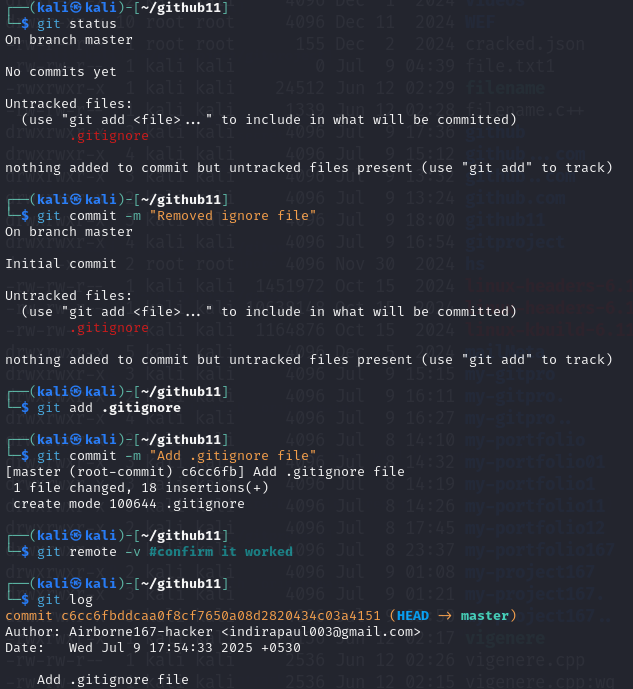
***2.Never commit directly to main***

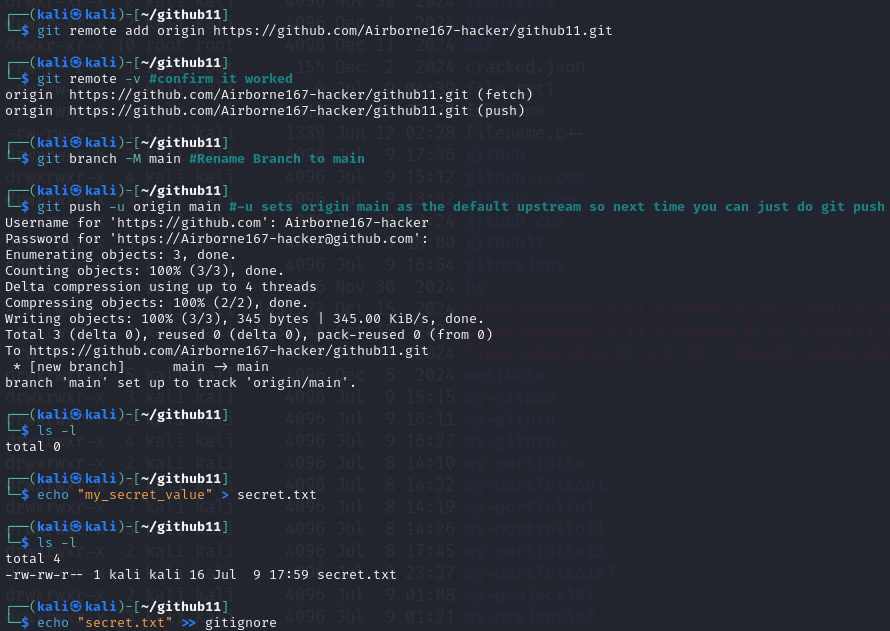
***3.Use .gitignore to exclude unwanted files***

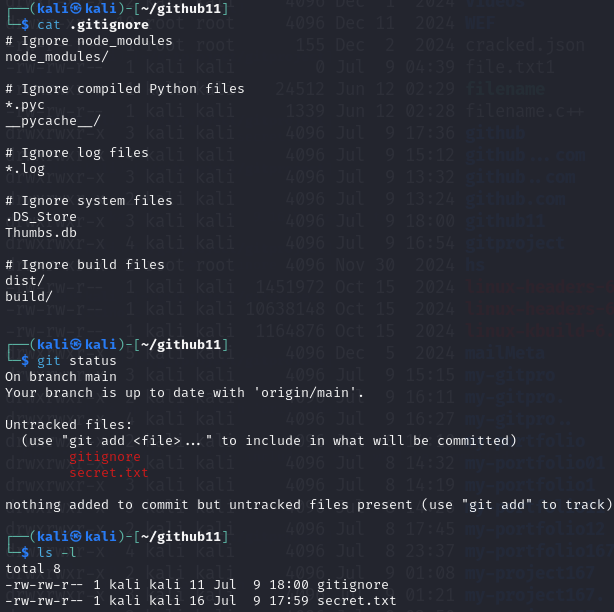
***FOR GITIGNORE***

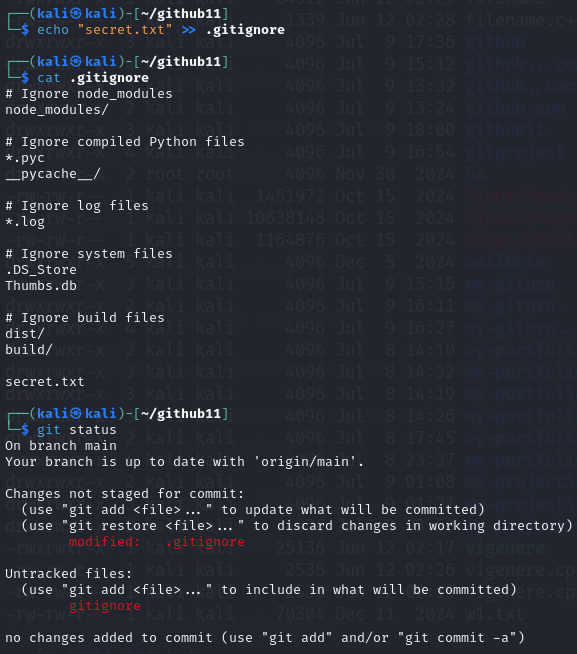












***Untracked files: (use "git add <file>..." to include in what will be committed)***

***gitignore***

***[That means Git is not tracking secret.txt anymore — success]***

***4.Write clear, descriptive commit messages***

***5.Pull before you push***

***Git & GitHub is about understanding how version control makes your development process safer, smarter, and more collaborative.***