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

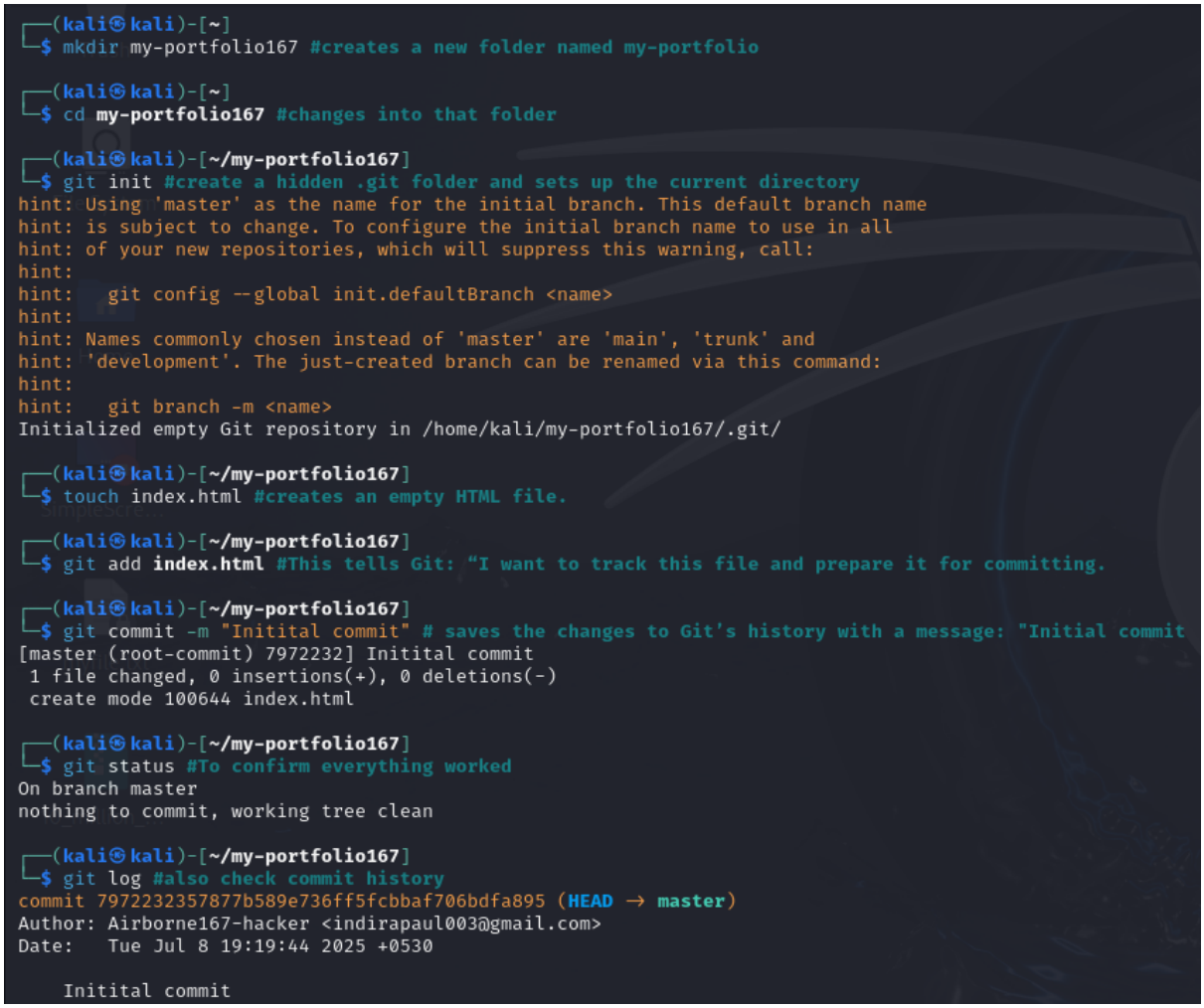
git init

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 → 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.

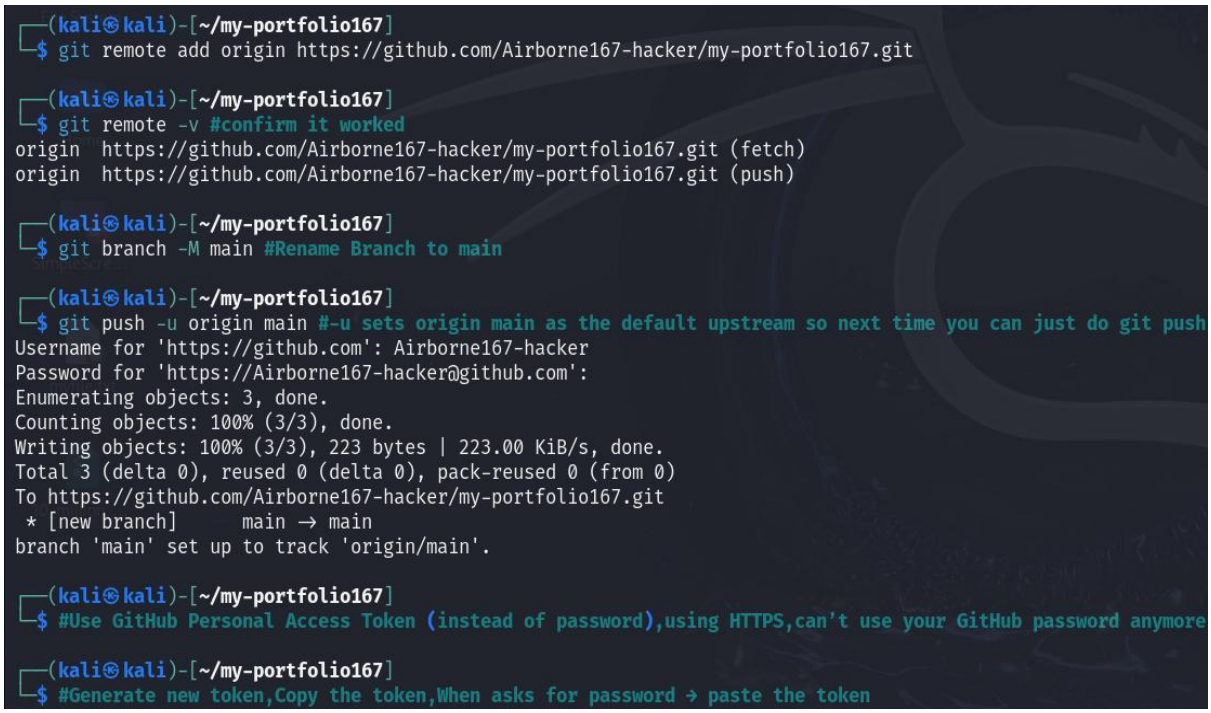
Connect Local Repo to Remote

git remote add origin https://github.com/myusername/my-portfolio.git

git branch -M main

git push -u origin main

local code is now live on GitHub

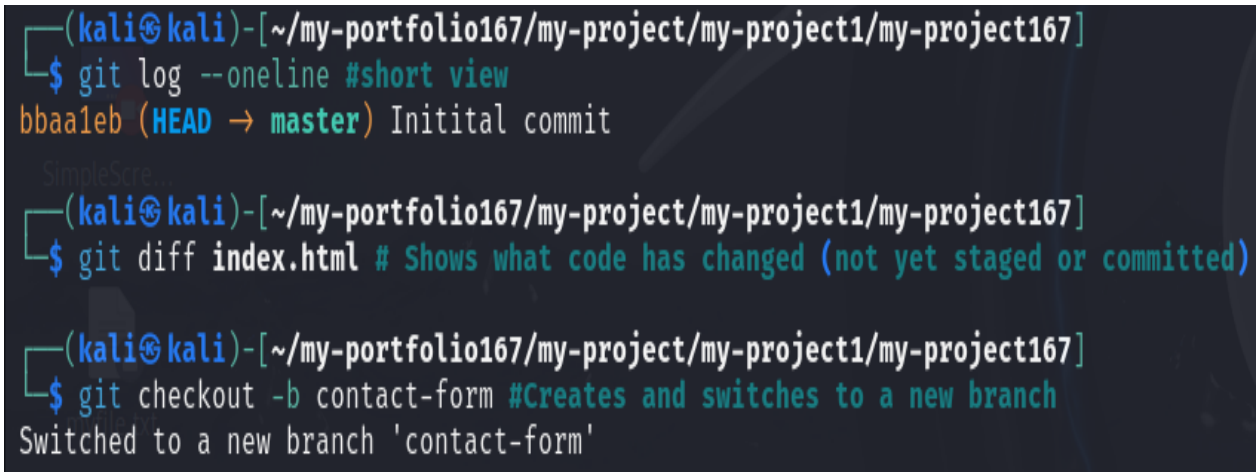


Essential Git Commands (Use Every Day)

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*

Command	*What It Does*	*Real Example*
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 repository in your folder*	*git init*

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



```
┌──(kali㉿kali)-[~/my-portfolio167/my-project/my-project1/my-project167]
└─$ git branch
* contact-form
  master

┌──(kali㉿kali)-[~/my-portfolio167/my-project/my-project1/my-project167]
└─$ git checkout master
Switched to branch 'master'

┌──(kali㉿kali)-[~/my-portfolio167/my-project/my-project1/my-project167]
└─$ git checkout -b main #If your branch is main but missing, it might be called master instead.
Switched to a new branch 'main'

┌──(kali㉿kali)-[~/my-portfolio167/my-project/my-project1/my-project167]
└─$ git merge contact-form #Merges another branch into the current one
Already up to date.

┌──(kali㉿kali)-[~/my-portfolio167/my-project/my-project1/my-project167]
└─$ git branch -a #Bonus Tip: See All Branches (including remote)
  contact-form
* main
  master
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git remote add origin https://github.com/Airborne167-hacker/my-project167...git

┌──(kali㉿kali)-[~/my-project167..]
└─$ git remote -v #confirm it worked
origin  https://github.com/Airborne167-hacker/my-project167...git (fetch)
origin  https://github.com/Airborne167-hacker/my-project167...git (push)

┌──(kali㉿kali)-[~/my-project167..]
└─$ git branch -M main #Rename Branch to main

┌──(kali㉿kali)-[~/my-project167..]
└─$ git push -u origin main #-u sets origin main as the default upstream so next time you can just do git push
Username for 'https://github.com': Airborne167-hacker
Password for 'https://Airborne167-hacker@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 222 bytes | 222.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Airborne167-hacker/my-project167...git
 * [new branch]      main → main
branch 'main' set up to track 'origin/main'.
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git pull origin main #Pulls latest changes from GitHub's main into your local main
From https://github.com/Airborne167-hacker/my-project167..
 * branch            main       → FETCH_HEAD
Already up to date.

┌──(kali㉿kali)-[~/my-project167..]
└─$ git fetch origin #Fetches changes from GitHub without merging

┌──(kali㉿kali)-[~/my-project167..]
└─$ git merge origin/main # can manually merge or inspect
Already up to date.
```

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.

<u>Viewing Changes & Undoing Mistakes</u>

git log --oneline

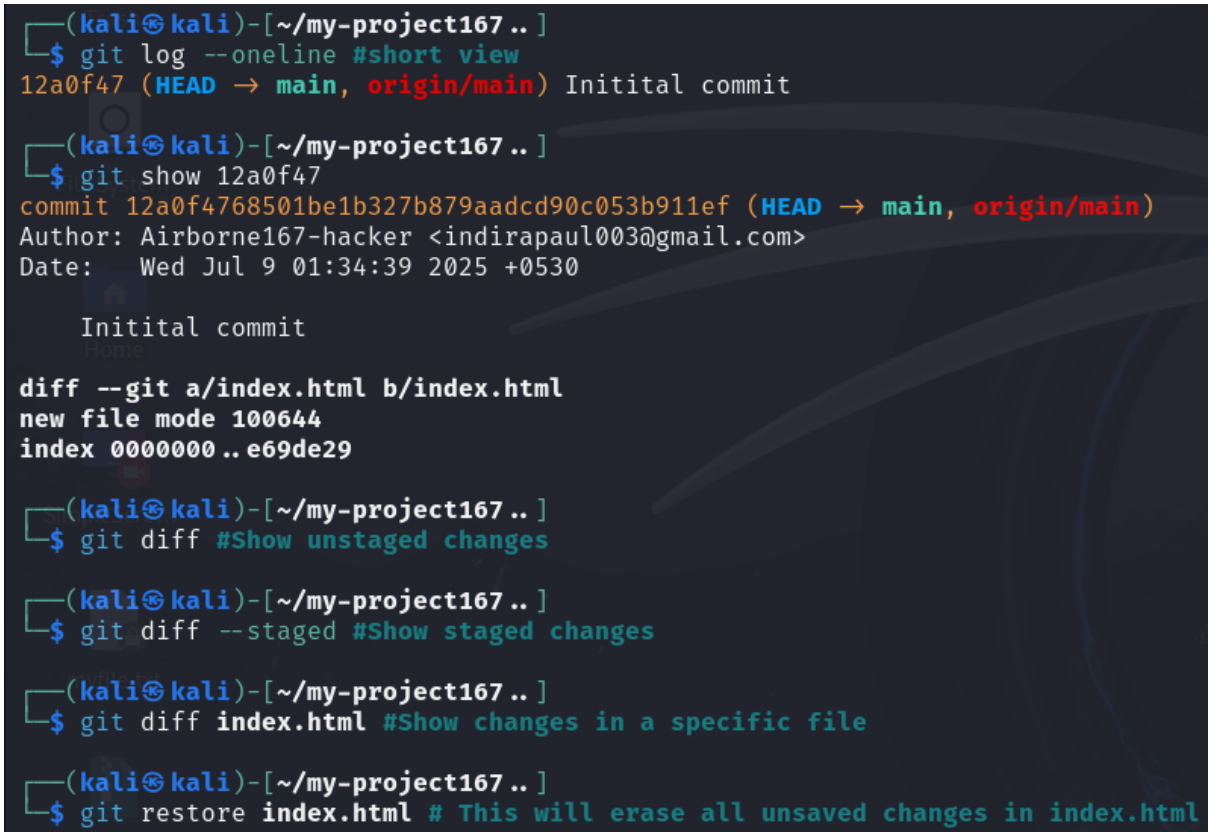
git show <commit-id>

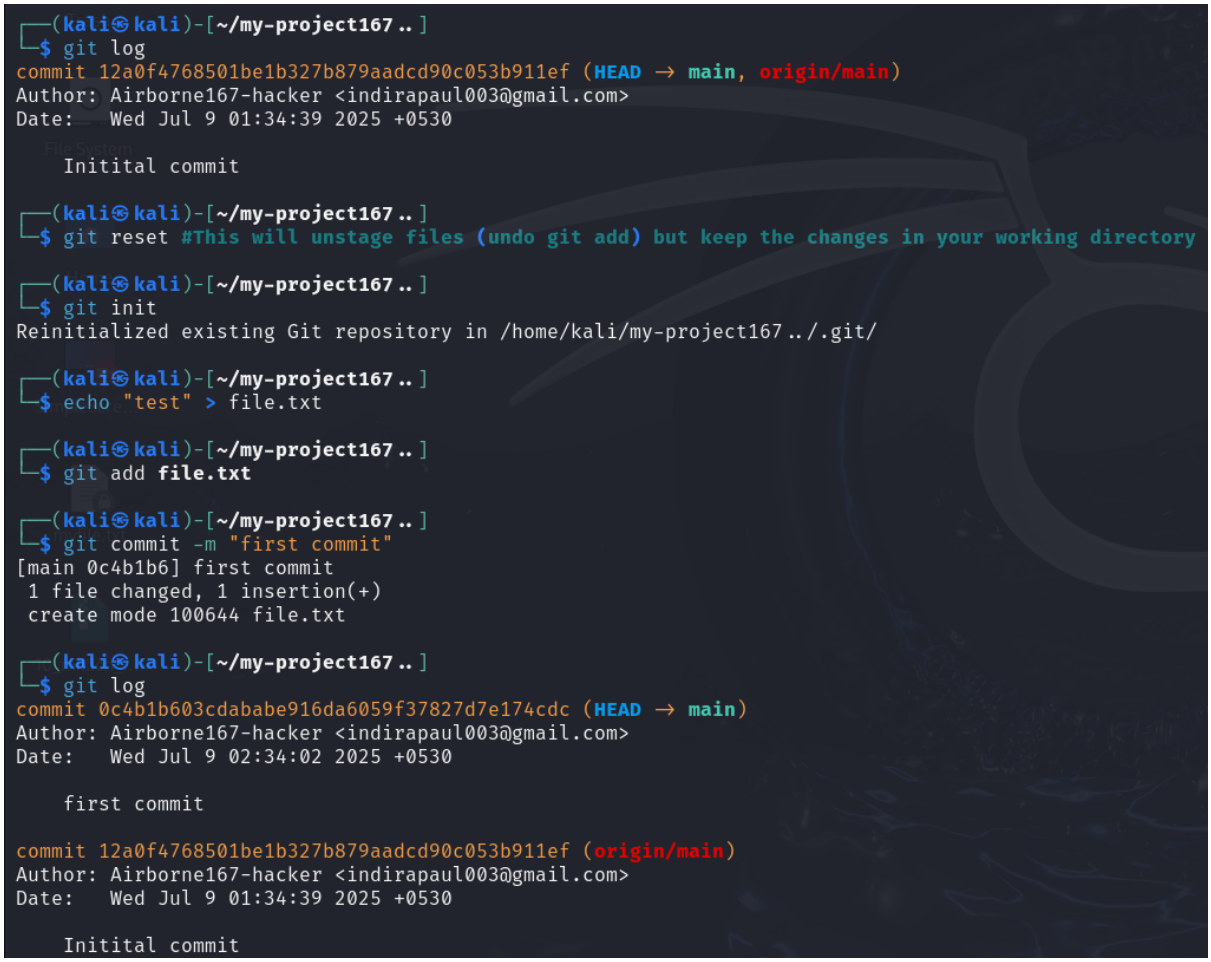
git diff

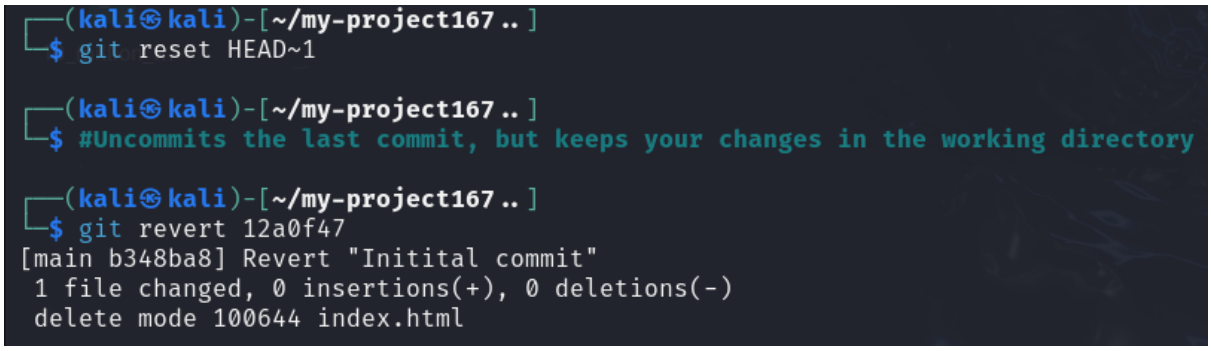
git restore file.txt # Undo file changes

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

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ touch forget-file.js

┌──(kali㉿kali)-[~/my-project167..]
└─$ git add forget-file.js

┌──(kali㉿kali)-[~/my-project167..]
└─$ git commit --amend #Replaces the last commit without creating a new one
[main 46181be] Revert "Initital commit"
 Date: Wed Jul 9 02:37:09 2025 +0530
 2 files changed, 1 insertion(+)
 create mode 100644 file.txt
 rename index.html ⇒ forget-file.js (100%)

┌──(kali㉿kali)-[~/my-project167..]
└─$ #Avoid amending commits that have already been pushed/shared
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git branch
* main

┌──(kali㉿kali)-[~/my-project167..]
└─$ git checkout main #This reapplies commits from feature-x on top of main
Already on 'main'
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

┌──(kali㉿kali)-[~/my-project167..]
└─$ git rebase main #Keeps history linear (unlike merge)
Current branch main is up to date.

┌──(kali㉿kali)-[~/my-project167..]
└─$ git stash -u #-u means “include untracked files
No local changes to save

┌──(kali㉿kali)-[~/my-project167..]
└─$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

┌──(kali㉿kali)-[~/my-project167..]
└─$ echo "change" >> app.js
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

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

nothing added to commit but untracked files present (use "git add" to track)

┌──(kali㉿kali)-[~/my-project167..]
└─$ touch app.js

┌──(kali㉿kali)-[~/my-project167..]
└─$ git add app.js

┌──(kali㉿kali)-[~/my-project167..]
└─$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app.js


┌──(kali㉿kali)-[~/my-project167..]
└─$ git stash ## clean now
Saved working directory and index state WIP on main: 46181be Revert "Initital commit"

┌──(kali㉿kali)-[~/my-project167..]
└─$ git stash list #see saved stash
stash@{0}: WIP on main: 46181be Revert "Initital commit"
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git stash apply #bring the change back
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app.js
```

```
┌──(kali㉿kali)-[~/my-project167..]
└─$ git stash pop #get changes back, even after switching branche
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   app.js

Dropped refs/stash@{0} (f1726247688216cb61cf54a0a2a4a1049d85c750)
```

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

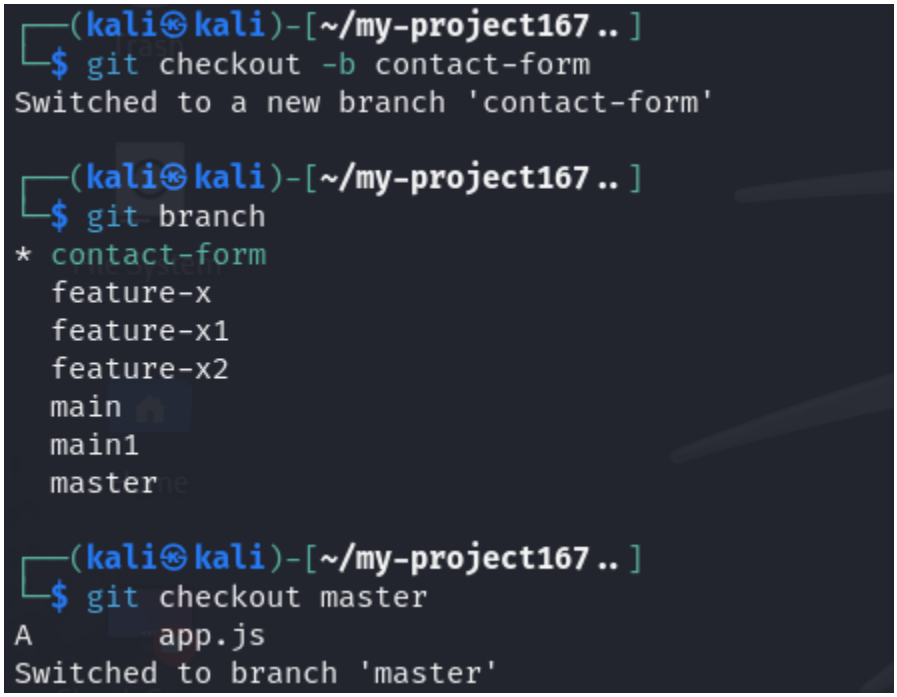
git log

 # 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

<u>FOR GITIGNORE</u>

```
┌──(kali㉿kali)-[~]
└─$ mkdir github11

┌──(kali㉿kali)-[~]
└─$ cd github11

┌──(kali㉿kali)-[~/github11]
└─$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/kali/github11/.git/

┌──(kali㉿kali)-[~/github11]
└─$ touch .gitignore

┌──(kali㉿kali)-[~/github11]
└─$ vim .gitignore
```

```
# Ignore node_modules
node_modules/

# Ignore compiled Python files
*.pyc
__pycache__/

# Ignore log files
*.log

# Ignore system files
.DS_Store
Thumbs.db

# Ignore build files
dist/
build/
```

```
┌──(kali㉿kali)-[~/github11]
└─$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

┌──(kali㉿kali)-[~/github11]
└─$ git commit -m "Removed ignore file"
On branch master

Initial commit

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

nothing added to commit but untracked files present (use "git add" to track)

┌──(kali㉿kali)-[~/github11]
└─$ git add .gitignore

┌──(kali㉿kali)-[~/github11]
└─$ git commit -m "Add .gitignore file"
[master (root-commit) c6cc6fb] Add .gitignore file
 1 file changed, 18 insertions(+)
 create mode 100644 .gitignore

┌──(kali㉿kali)-[~/github11]
└─$ git remote -v #confirm it worked

┌──(kali㉿kali)-[~/github11]
└─$ git log
commit c6cc6fbddcaa0f8cf7650a08d2820434c03a4151 (HEAD → master)
Author: Airborne167-hacker <indirapaul003@gmail.com>
Date:   Wed Jul 9 17:54:33 2025 +0530

    Add .gitignore file
```

```
┌──(kali㉿kali)-[~/github11]
└─$ git remote add origin https://github.com/Airborne167-hacker/github11.git

┌──(kali㉿kali)-[~/github11]
└─$ git remote -v #confirm it worked
origin  https://github.com/Airborne167-hacker/github11.git (fetch)
origin  https://github.com/Airborne167-hacker/github11.git (push)

┌──(kali㉿kali)-[~/github11]
└─$ git branch -M main #Rename Branch to main

┌──(kali㉿kali)-[~/github11]
└─$ git push -u origin main #-u sets origin main as the default upstream so next time you can just do git push
Username for 'https://github.com': Airborne167-hacker
Password for 'https://Airborne167-hacker@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Airborne167-hacker/github11.git
 * [new branch]      main → main
branch 'main' set up to track 'origin/main'.

┌──(kali㉿kali)-[~/github11]
└─$ ls -l
total 0

┌──(kali㉿kali)-[~/github11]
└─$ echo "my_secret_value" > secret.txt

┌──(kali㉿kali)-[~/github11]
└─$ ls -l
total 4
-rw-rw-r-- 1 kali kali 16 Jul  9 17:59 secret.txt

┌──(kali㉿kali)-[~/github11]
└─$ echo "secret.txt" >> gitignore
```

```
┌──(kali㉿kali)-[~/github11]
└─$ cat .gitignore
# Ignore node_modules
node_modules/

# Ignore compiled Python files
*.pyc
__pycache__/

# Ignore log files
*.log

# Ignore system files
.DS_Store
Thumbs.db

# Ignore build files
dist/
build/


┌──(kali㉿kali)-[~/github11]
└─$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)

┌──(kali㉿kali)-[~/github11]
└─$ ls -l
total 8
-rw-rw-r-- 1 kali kali 11 Jul  9 18:00 gitignore
-rw-rw-r-- 1 kali kali 16 Jul  9 17:59 secret.txt
```

```
┌──(kali㉿kali)-[~/github11]
└─$ echo "secret.txt" >> .gitignore

┌──(kali㉿kali)-[~/github11]
└─$ cat .gitignore
# Ignore node_modules
node_modules/

# Ignore compiled Python files
*.pyc
__pycache__/

# Ignore log files
*.log

# Ignore system files
.DS_Store
Thumbs.db

# Ignore build files
dist/
build/

secret.txt

┌──(kali㉿kali)-[~/github11]
└─$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

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

no changes added to commit (use "git add" and/or "git commit -a")
```

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.