✅ Git Basics

• `git --version`
Checks if Git is installed and shows the installed version.
Example: `git version 2.42.0`

• `git config --global user.name "Your Name"`
• `git config --global user.email "you@example.com"`
Sets your global identity (used in every repo unless overridden). Essential for associating commits with your GitHub account.

• `git init`
Initializes an empty Git repository in your project folder by creating a `.git/` directory. This begins tracking changes.

• `git status`
Shows the current working state — untracked, modified, or staged files. Helps ensure you know what will be committed.

• `git add <filename>`
Adds a specific file to the **staging area**, meaning it's ready to be committed.
Use `git add .` to stage **all modified and untracked files**.

• `git commit -m "message"`
Takes everything in the staging area and saves it to the local repo with a descriptive message.
`-m` stands for message.

• `git commit -am "message"`
Shortcut to **add AND commit** modified files (not untracked ones).
You skip `git add`, but only for files already tracked by Git.

• `git log`
Shows a list of all commits in reverse chronological order, along with author, date, and commit hash.

✅ Working with Remote Repositories (GitHub)

• `git remote add origin https://github.com/username/repo.git`
Connects your local repo to a remote GitHub repo and gives it an alias (usually `origin`).

• `git push -u origin main`
Pushes your current branch (`main`) to the remote repo and sets it as the upstream. From now, `git push` will be enough.

• `git push`
Sends local commits to the remote repo on the currently tracked branch.

• `git pull`
Pulls the latest commits from the remote repo and merges them into your local copy.
Equivalent to: `git fetch` + `git merge`.

• `git fetch`
Fetches changes from remote, but does **not** automatically merge them into your current branch. Safe way to inspect updates first.

✅ Branching & Collaboration

• `git branch`
Shows all branches. The `*` indicates the branch you're currently on.

• `git branch <branch-name>`
Creates a new branch but doesn't switch to it.

• `git checkout <branch-name>`
Switches to another branch.

• `git checkout -b feature/login`
Creates a **new branch** and immediately switches to it. Common naming:
feature/login
bug/navbar-issue
fix/signup-flow

• `git merge feature/login`
Merges the named branch into the currently active branch. Used to bring finished feature work back into `main`.

• `git branch -d feature/login`
Deletes a local branch after it's merged.

• **Pull Requests (on GitHub)**
After pushing a new branch, you can open a *Pull Request* on GitHub to propose merging your changes into `main`.
Allows for code reviews, discussion, and safe integration.

✅ .gitignore File

• Contents example:

```
bash
CopyEdit
.env
node_modules/
*.log
.DS_Store
```

This file tells Git to **ignore** certain files or folders — typically sensitive info like environment variables, API keys, or unnecessary files.

✅ SSH Keys

• `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`
Generates a public/private key pair for authentication.

• Copy public key using:
```
cat ~/.ssh/id_rsa.pub
```

• Add this to GitHub under:
GitHub → Settings → SSH and GPG Keys → New SSH Key

• Test connection:
```
ssh -T git@github.com
```

• If needed:
```
eval $(ssh-agent -s)
ssh-add ~/.ssh/id_rsa
```

✅ Cloning Repositories

• `git clone https://github.com/user/repo.git`
Clones an entire remote repository and its history to your machine.

• `git clone <url> .`
Clones into the **current directory** (useful when you're already in a folder).

✅ CI/CD with Vercel

• Vercel auto-deploys GitHub repos every time you `git push` to the `main` branch (or any tracked branch).
• You can connect your GitHub account on https://vercel.com, choose the repo, and deploy it instantly.
• Ideal for frontend projects — static sites, SPAs, etc.
• No setup needed for simple HTML/CSS/JS projects.

✅ Pro Tips

• `git diff`
See what's changed but not yet staged.

• `git reset <file>`
Unstage a file (moves it out of staging area).

- `git rm --cached <file>`
Stop tracking a file (useful if accidentally added sensitive files before `.gitignore`).

- `git revert <commit-hash>`
Undo a specific commit without changing history.

- `git reset --hard HEAD~1`
Dangerous! Resets the last commit entirely (and deletes changes).