✅ Git Basics

- `git init`: Initializes a local Git repo.
- `git add`: Stages files for commit.
- `git commit`: Saves changes to the local repo.
- `git status`, `git log`: View current repo state and commit history.

✅ GitHub & Remote Setup

- GitHub is a **hosting platform** for Git repos.
- `git remote add origin <URL>`: Links local repo to GitHub.
- `git push`: Uploads commits to GitHub.
- `git pull`: Pulls changes from GitHub.

✅ Branching & Collaboration

- `git branch`: Lists branches.
- `git checkout -b feature/xyz`: Creates and switches to a new branch.
- Pull Requests: Merge new features safely into `main`.
- `git merge`: Combines changes from other branches.
- `git branch -d`: Deletes a local branch.

✅ Useful Tools

- `.gitignore`: Avoids pushing sensitive files like `.env`.
- SSH Keys: Secure authentication for Git operations.

✅ CI/CD with Vercel

- Connect your GitHub repo to Vercel.
- Auto-deploys every time you `git push` to main.
- Great for static sites and small front-end projects.

🔷 1. `git --version`

- **Syntax**:

```
git --version
```

- **Purpose**: Check Git installation and version.
- **Explanation**: Verifies that Git is installed and shows the current installed version.
- **Output**:

```
git version 2.42.0
```

- **Use Cases**:
  - Verifying Git is installed.
  - Checking for version compatibility.

🔷 2. `git config --global user.name "Your Name"`

- **Syntax**:

```
git config --global user.name "Your Name”

git config --global user.email "you@example.com"
```

- **Purpose**: Sets global Git identity.
- **Explanation**: Associates commits with your name and email across all projects on your system.
- **Output**: No output unless you omit value to view:

```
git config --global user.name
```

- **Use Cases**:
  - Required before making commits.
  - Helps track contributions by identity.

🔷 3. `git init`

- **Syntax**:

  ```
  git init
  ```

- **Purpose**: Initialize a new Git repository.
- **Explanation**: Creates a `.git` directory in your project folder that starts tracking your files.
- **Output**:

  ```
  Initialized empty Git repository in /path/to/
  project/.git/
  ```

- **Use Cases**:
  - Starting version control on a local project.
  - Setting up a new repo before connecting to GitHub.

🔷 4. `git status`

- **Syntax**:

  ```
  git status
  ```

- **Purpose**: Show current working directory state.
- **Explanation**: Displays untracked, modified, staged, and committed files.
- **Output**:
  Lists:
  - Untracked files (not added yet)
  - Changes to be committed
  - Changes not staged
- **Use Cases**:
  - Checking what files need staging or committing.
  - Validating what will happen before a commit.

🔷 5. `git add`

- **Syntax**:

```
git add <filename>
git add .
```

- **Purpose**: Add files to staging area.
- **Explanation**: Prepares files to be committed to the repository.
- **Output**: No output, but changes reflected in `git status`.
- **Use Cases**:
    - Add single or multiple files before a commit.
    - Use `git add .` to stage everything at once.

🔷 6. `git commit -m "message"`

- **Syntax**:

```
git commit -m "Your commit message"
```

- **Purpose**: Save staged changes to local repository.
- **Explanation**: Records a snapshot of the project with a message.
- **Output**:

```
[main abc1234] Your commit message
1 file changed, 2 insertions(+)
```

- **Use Cases:**
    - Log progress or completed features.
    - Create meaningful messages for change history.

🔷 7. `git log`

- **Syntax**:

```
git log
```

- **Purpose**: Show commit history.
- **Explanation**: Lists all commits in reverse chronological order.
- **Output**:

```
commit 7f3d2a9 Author: ...

Date: ...
Message: ...
```

- **Use Cases**:
    - Reviewing project history.
    - Copying commit hashes for reverts or cherry-picking.

🔷 8. `git remote add origin <URL>`

- **Syntax**:

```
git remote add origin https://github.com/username/
repo.git
```

- **Purpose**: Link local repo to remote (e.g., GitHub).
- **Explanation**: Names the remote as `origin` and allows pushing/pulling.
- **Output**: None.
- **Use Cases**:
    - Pushing code to GitHub.
    - Collaborating with others.

🔷 9. `git push -u origin main`

- **Syntax**:

```
git push -u origin main
```

- **Purpose**: Upload local commits to remote repo.
- **Explanation**: Pushes `main` branch to remote and sets upstream for future `git push`.
- **Output**:

```
Enumerating objects...
To https://github.com/...
```

- **Use Cases**:
  - First push of a branch.
  - Updating remote with local work.

🔷 10. `git pull`

- **Syntax**:

```
git pull
```

- **Purpose**: Fetch and merge changes from remote.
- **Explanation**: Combines `git fetch` and `git merge` to keep local in sync.
- **Output**:

```
Updating abc1234..def5678
```

- **Use Cases**:
  - Staying up to date with team changes.
  - Getting updated README or new commits.

🔷 11. `git branch`

- **Syntax**:

```
git branch
git branch <branch-name>
```

- **Purpose**: View, create, or delete branches.
- **Explanation**: Helps manage multiple lines of development.

- **Output**:
  ```
  * main

   feature/login
  ```
- **Use Cases**:
  - Feature development.
  - Isolating experiments.

🔷 12. `git checkout -b <branch-name>`

- **Syntax**:
  ```
  git checkout -b feature/login
  ```
- **Purpose**: Create and switch to a new branch.
- **Explanation**: Shortcut for `git branch` + `git checkout`.
- **Output**:
  ```
  Switched to a new branch 'feature/login'
  ```
- **Use Cases**:
  - Start a new feature.
  - Isolate code before merging.

🔷 13. `git merge <branch-name>`

- **Syntax**:
  ```
  git merge feature/login
  ```
- **Purpose**: Merge one branch into another.
- **Explanation**: Integrates changes into the current branch.
- **Output**:
  ```
  Merge made by the 'recursive' strategy.
  ```

- **Use Cases**:
  - Bring features into main.
  - Complete development cycles.

🔷 14. `git clone <repo-url>`

- **Syntax**:

```
git clone https://github.com/user/repo.git
```

- **Purpose**: Copy remote repository to your machine.
- **Explanation**: Brings entire project history and files.
- **Output**:

```
Cloning into 'repo'...
```

- **Use Cases**:
  - Contributing to projects.
  - Downloading templates or codebases.

🔷 15. `git .gitignore`

- **Syntax**:
  Contents of `.gitignore` file:

```
.env

node_modules/
*.log
```

- **Purpose**: Ignore specific files/folders.
- **Explanation**: Prevents sensitive or unnecessary files from being tracked/pushed.
- **Output**: No output; affects tracking behavior.
- **Use Cases**:
  - Keep secrets out of Git.
  - Prevent cluttering commits with generated files.