What is Markdown?

Markdown is a lightweight writing language used to create formatted documents using plain text. It converts simple symbols like #, *, > into:

Headings

Bullet points

Tables

Links

Images

Code blocks

Etc.

It is used everywhere: GitHub, VS Code, Jupyter Notebooks, documentation, websites, and reports.

☑ Why is Markdown Needed?

Markdown is important because:

1 It makes documentation easy

You can write clean, structured documents without Word or formatting buttons.

2 It is used heavily on GitHub

README files, project documentation, wiki pages, pull requests, issues, and changelogs all use Markdown.

3 It is faster than using Word/Google Docs

You type symbols → it automatically formats.

4 It works everywhere

You can render Markdown on:

GitHub

VS Code

Websites

Notebooks

Documentation tools

5 It can convert into PDF, Word, HTML

Using tools like Markdown Preview Enhanced or Pandoc.

☑ How to Make Use of Markdown

Create a file ending with .md Example: README.md

Open it in VS Code or GitHub

Write Markdown commands (see list below)

Preview it

In VS Code → Ctrl + Shift + V

In GitHub → Preview appears automatically

Convert to PDF/HTML if needed (via VS Code extension)

That's all — Markdown is extremely simple.

☑ Full Markdown Commands, How to Use Them & When to Use Them

1. Headings

Heading 1

Heading 2

Heading 3

Heading 4

Use when: Creating titles, subtitles, and sections.

2. Bold Text **bold text**

Use when: Highlighting important points.

3. Italic Text *italic text*

Use when: Emphasizing words.

4. Bold + Italic ***bold and italic***

5. Bullet List

- Item 1
- Item 2

- Another item

Use when: Listing points.

6. Numbered List
7. Step one

8. Step two
9. Step three

Use when: Procedures, instructions.

7. Task List (checkboxes)

- [ ] To do
- [x] Completed task

Use when: Project tracking (GitHub supports this).

8. Links link text

Use when: Adding sources or references.

9. Images alt text

Use when: Showing diagrams, screenshots, logos.

10. Code (inline) `code`

Use when: Writing code inside a sentence.

11. Code Block (multi-line)

```
print("Hello World")
```

Use when: Adding programming code examples.

12. Blockquote

> This is a quote.

Use when: Highlighting notes or important statements.

13. Horizontal Line

Use when: Separating sections.

14. Tables | Name | Age | |------|-----| | John | 25 | | Ada | 22 |

Use when: Presenting structured data.

15. Highlight / Alert (GitHub & VS Code)

> **Note:** This is important. **Warning:** Be careful.

Use when: Adding warnings or notes.

16. Strikethrough ~~deleted text~~

17. Footnotes This is a sentence with a footnote.[^1]

[^1]: This is the footnote text.

18. Collapsible Section (GitHub only)

▶ Click to expand

Hidden content here.

Use when: Hiding long content.

19. Emoji 😄👍🔥

Use when: Enhancing readability.

20. Escaping Markdown Commands \This will show star instead of italics*

Use when: You need to show symbols without formatting.

GIT WORK FLOW

☑ Git Workflow (Simple and Complete)

Git has 3 main areas where your files move:

Working Directory → where you write/edit your files

Staging Area → where you prepare files for commit

Local Repository → where commits are stored on your computer

Remote Repository (GitHub) → where the project is shared online

🔚 FULL GIT WORKFLOW (Step-by-step) 1️⃣ Create or Clone a Repository A. Create a new project git init

This creates a new empty Git repository.

B. Or download an existing project git clone https://github.com/user/repo.git

2️⃣ Make Changes in Working Directory

You edit files normally:

write code

modify documents

delete/update files

Git detects these changes automatically.

3️⃣ Check What Has Changed git status

Shows:

modified files

untracked files

staged files

4️⃣ Stage Files (Prepare for Commit) git add filename

or stage all changes:

git add .

Staged files are ready to be saved.

5️⃣ Commit the Changes

A commit is a snapshot of your project.

git commit -m "Explain what you changed"

This saves your changes into the local repository.

6️⃣ Connect to GitHub (Only once per project) git remote add origin https://github.com/user/repo.git

7️⃣ Push to GitHub (Upload your commits) git push origin main

or

git push origin master

Now your code is online.

8️⃣ Continue Working

As you continue editing:

Modify → Stage → Commit → Push Modify → Stage → Commit → Push

9️⃣ Pull Updates from Others (If you're collaborating) git pull origin main

This brings new changes from GitHub to your computer.

🔀 Optional: Create Branches for New Features git branch feature-login git checkout feature-login

Or combined:

git checkout -b feature-login

After finishing:

git add . git commit -m "Completed feature" git push origin feature-login

Then create a pull request on GitHub.

Summary of the Git Workflow#

Working Directory → git add → Staging Area → git commit → Local Repo → git push → GitHub

To update your local copy:

GitHub → git pull → Local Repo → Working Directory

GIT COMMANDS

☑ MODULE 1 — Git Setup & Configuration

1. git --version

What it does: Checks your Git version How to use:

git --version

2. git config

What it does: Sets your Git identity How to use:

git config --global user.name "Your Name" git config --global user.email "you@example.com"

3. git help

What it does: Shows Git help and manual How to use:

git help commit

☑ MODULE 2 — Creating or Getting a Repository 4. git init

What it does: Creates a new Git repository How to use:

git init

5. git clone

What it does: Downloads a project from GitHub How to use:

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

☑ MODULE 3 — Checking the Status of Your Work 6. git status

What it does: Shows file changes (tracked, untracked, staged) How to use:

git status

7. git diff

What it does: Shows the exact line changes How to use:

git diff # unstaged changes git diff --staged # staged changes

☑ MODULE 4 — Staging and Committing Changes 8. git add

What it does: Moves files to staging area How to use:

git add file.txt # add single file git add folder/ # add folder git add . # add everything

9. git commit

What it does: Saves your changes permanently How to use:

git commit -m "Your message"

10. git commit -am

(Shortcut — adds AND commits tracked files)

git commit -am "Quick commit"

☑ MODULE 5 — Working with Remote Repositories (GitHub) 11. git remote add origin URL

What it does: Connects your local repo to GitHub

git remote add origin https://github.com/user/repo.git

12. git remote -v

What it does: Shows connected remote repositories

git remote -v

13. git push

What it does: Uploads commits to GitHub

git push origin main

14. git pull

What it does: Downloads new changes from GitHub

git pull origin main

15. git fetch

What it does: Downloads changes WITHOUT merging

git fetch origin

☑ MODULE 6 — Branching & Merging 16. git branch

What it does: Lists, creates, or deletes branches

git branch # show branches git branch new-feature # create branch git branch -d old-branch # delete branch

17. git checkout

What it does: Switches to another branch

git checkout new-feature

18. git checkout -b

(Creates & switches)

git checkout -b login-page

19. git merge

What it does: Combines branches

git checkout main git merge new-feature

☑ MODULE 7 — Undoing Changes & Fixing Mistakes 20. git reset

What it does: Unstage or remove commits

git reset file.txt # unstage git reset --hard HEAD~1 # delete last commit (dangerous)

21. git restore

What it does: Restores modified files

git restore file.txt

22. git revert

What it does: Creates a new commit that cancels a previous commit

git revert abc1234

☑ MODULE 8 — Viewing History 23. git log

What it does: Shows commit history

git log git log --oneline git log --graph --decorate

24. git show

What it does: Shows details of a commit

git show abc1234

☑ MODULE 9 — Working with Files 25. git rm

What it does: Removes file from repo

git rm file.txt

26. git mv

What it does: Renames or moves a file

git mv old.txt new.txt

☑ MODULE 10 — Tagging Releases 27. git tag

What it does: Marks a specific version/release

git tag v1.0 git tag -a v1.0 -m "Version 1" git push origin v1.0

GIT BRANCHING

How Branching Works (Practical Example) Step 1 — Check branches git branch

Step 2 — Create a branch git branch login-feature

Step 3 — Switch to it git checkout login-feature

Step 4 — Do your work, commit git add . git commit -m "Added login feature"

Step 5 — Push to GitHub git push origin login-feature

Step 6 — Create Pull Request (PR) on GitHub

A PR lets others review and approve your code.

Step 7 — Merge into main

Click Merge on GitHub after approval.