perplexity

this is an image for the git and github.

i didn't get actually which command will do what? can u explain in details for each command also add extra command which are important

Here's a detailed explanation of each Git command from the cheat sheet, along with additional important commands:

Repository Commands

These commands deal with initializing or cloning repositories.

- `git init`: Initializes a new Git repository in the current directory. It sets up a `.git` folder to track changes.
  - Example: `git init` creates a local repository.
- `git clone <repo-url>`: Clones an existing repository from a remote URL into your local machine.
  - Example: `git clone https://github.com/user/repo.git` downloads the repository and its history.

Basic Commands

These are essential for tracking and managing changes.

- `git status`: Displays the current state of your working directory and staging area. It shows which files are modified, staged, or untracked.
  - Example: `git status` helps you check what changes need to be committed.
- `git add <file>`: Stages changes (files) for the next commit. Use `git add -A` to stage all changes.
  - Example: `git add file.txt` stages `file.txt`.
- `git commit -m "Message"`: Saves staged changes to the repository with a descriptive message.
  - Example: `git commit -m "Added login feature"` commits changes with the message.
- `git log`: Shows the commit history of the repository.
  - Example: `git log --oneline` gives a concise view of commits.

Branching Commands

These commands manage branches for parallel development.

- `git branch`: Lists all branches in the repository. The current branch is marked with an asterisk (`*`).
  - Example: `git branch -a` lists all branches (local and remote).
- `git branch <branch-name>`: Creates a new branch.
  - Example: `git branch feature-login`.
- `git checkout <branch-name>`: Switches to another branch.
  - Example: `git checkout feature-login`.
- `git merge <branch-name>`: Merges changes from another branch into the current branch.
  - Example: `git merge feature-login`.
- `git branch -d <branch-name>`: Deletes a branch locally (only if merged).
  - Example: `git branch -d feature-login`.

Remote Repository Commands

These commands interact with remote repositories like GitHub.

- `git remote`: Lists configured remote repositories.
  - Example: `git remote -v`.
- `git remote add <name> <url>`: Adds a new remote repository.
  - Example: `git remote add origin https://github.com/user/repo.git`.
- `git push <remote> <branch>`: Pushes committed changes to a remote repository.
  - Example: `git push origin main`.
- `git pull <remote> <branch>`: Fetches and merges changes from a remote repository into your current branch.
  - Example: `git pull origin main`.

Undoing Changes

These commands help revert or reset changes.

- `git pull`: Fetches and merges changes from the remote repository into your local one (same as above).
- `git fetch`: Fetches updates from the remote repository without merging them.
  - Example: `git fetch origin`.
- `git reset --hard HEAD`: Resets your working directory to match the last commit, discarding all uncommitted changes.

    - Example: `git reset --hard HEAD`.
- `git revert <commit-hash>`: Creates a new commit that undoes the changes made in a specific commit without altering history.
    - Example: `git revert abc123`.

Additional Useful Commands

Here are some extra commands that can be helpful:

File Management

- `git rm <file>`: Removes files from the working directory and stages their deletion for commit.
    - Example: `git rm file.txt`.
- `git mv <old-name> <new-name>`: Renames or moves files within the repository.
    - Example: `git mv file1.txt file2.txt`.

Viewing Differences

- `git diff [file]`: Shows differences between your working directory and staging area or between commits.
    - Example: `git diff HEAD~1 HEAD`.

Stashing Changes

- `git stash save "message"`: Temporarily saves uncommitted changes for later use.
    - Example: `git stash save "Work in progress"`.
- `git stash apply [stash-id]`: Applies stashed changes back into your working directory.

Tagging

- `git tag <tag-name>`: Creates a tag for marking specific points in history (e.g., version releases).
    - Example: `git tag v1.0`.

Configuration

- **Global Configurations**
    - Set username globally:

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

    - Set email globally:

```
git config --global user.email "[email protected]"
```

These commands cover most scenarios you'll encounter when using Git. Mastering them will help you effectively manage repositories and collaborate on projects!

⁂