Git

Installation

https://git-scm.com/

Setup/Configuration

rmian03@RazaXps03 MINGW64 ~ $ git --version git version 2.42.0.windows.2	Check the version of git installed on your machine.
$ git config --global user.name "Your Name" $ git config --global user.email "your email"	Configure the name and email that Git will associate with your work
$ git config user.name $ git config user.email	Check what user and email are set

Git Basic Commands

$ git status	Shows the current state of the repo, including staged, unstaged, and untracked files.
$ git init	Initializes a new Git repository in the current directory.
$ git add <file>	It's used to stage the changes before committing them. To add all files to stage, use (.), instead of the file name.
$ git commit -m "message"	It's like hitting the "Save" button on your changes. It's used to commit the changes with a message.
$ git log	View commit history. Shows a list of all the "Save" points (or "commits") made in the project.
$ git log --oneline	Shows each commit as a single line.

Branching & Merging

$ git branch	List all Branches
$ git branch <newBranch>	Create a new Branch
$ git checkout <newbranch> $ git switch <newBranch>	Switch to a Branch
$ git checkout -b <newBranch> $ git switch -c <newBranch>	Create and Switch to a new branch in one command.
$ git branch -d <newBranch>	Delete a Branch
$ git branch -m oldName newName	Rename a Branch
$ git merge <feature>	Combine changes from one branch into another. To merge changes from **feature** branch into the **main** branch, you would: (1) Checkout or switch to the main branch, (2) then run the git merge command.

Git Diff and Stashing

$ git diff	Compares the differences between the working directory (unstaged changes) and the staging area (staged changes). If there aren't any staged changes, then it compares with the last commit.
$ git diff HEAD	View changes compared to the last commit.
$ git diff –staged $ git diff --cached	View changes between staged files and the last commit.
$ git diff <commit_id1> <commit_id2>	View changes between two specific commits.
$ git diff branch1..branch2	View changes between two branches.
$ git diff path/to/file	View changes made in a specific file.
$ git diff –name-only	See only the summary of changes

`$ git stash` `$ git stash save "msg"`	Save changes for later without committing.
`$ git stash list`	List all stashed changes
`$ git stash apply`	Re-applies the stashed changes, but keeps them in the stash. Useful if you want to apply the same stashes changes to multiple branches
`$ git stash pop`	Re-applies the stashed changes and removes them from the stash
`$ git stash drop`	Deletes the latest stash
`$ git stash clear`	Deletes all stashes.

Restore, Reset & Revert

`$ git restore <filename>`
`$ git restore .`
`$ git checkout -- <filename>`
	Discards changes in the working directory. Note: Use (.) to do the same for all files. Does not affect the changes in the staging area. Does not affect untracked files.

`$ git restore --staged <filename>`	Unstages a file but keeps changes in the working directory.

`$ git reset`
`$ git reset --mixed`
`$ git reset HEAD`
	Unstages a file, but keeps modifications (similar to **git restore --staged**).
	Note: You can specify a file with git reset or leave it blank to affect all files.

`$ git reset --soft <commit>` `$ git reset --soft HEAD~n`	Moves the HEAD back to a specified commit, but keeps all the changes staged. Does not affect the working directory either.
`$ git reset --hard` `$ git reset --hard HEAD~n`	Moves the HEAD back to a specific commit and discards all changes in the staging area and the working directory. IRREVERSIBLE. Does not affect untracked files. If you want to remove untracked files use: **git clean -f**
`$ git revert HEAD` `$ git revert <commit>`	Creates a new commit that undoes the changes from the last or a specified commit
`$ git revert HEAD~2..HEAD`	Reverts two commits: HEAD~1 and HEAD. The commit at HEAD~2 is excluded from this range.
`$ git revert -n <commit>` `$ git --no-commit <commit>`	Applies the changes to undo the commit but won't commit them. You can then make additional changes before committing

Git Tags

`$ git tag <tag>`	Create a lightweight tag. Pointer to a specific commit, typically to indicate a version release or a major milestone. Example: v1.0.1
`$ git tag -a <tag> -m "message"`	Create an annotated tag.
`$ git tag <tag> <commitHash>` `$ git tag -a <tag> <commitHash> -m "message"`	Tag a specific commit with lightweight or annotated tags.
`$ git tag`	List all tags.
`$ git show <tag>`	See more details about a specific tag
`$ git tag -l "*beta*"`	List all tags that include "beta" in their name.
`$ git push origin <tag>` `$ git push origin --tags`	Push a specific or all tages to the remote repository.
`$ git tag checkout <tag>`	Check out the state of the repository at the point in time of a specific tag. If you want to make changes from a tag, create a new branch from there: **git checkout -b newBranch <tag>**
`$ git diff <tag1> <tag2>`	Compare the changes between two tags.
`$ git tag -d <oldTag>`	Delete old tag locally
`$ git push origin --delete <oldTag>`	Delete old tag on the remote repository

Git Rebase

`$ git rebase <upstream-branch>` `$ git rebase main`	Move or combine commits from one branch onto another. Running **Git rebase main** on your feature branch takes the commits from the feature branch and re-applies them on top of the latest version of the main branch, creating a cleaner history without merge commits. **Note**: **git rebase main** does not merge changes into main; it simply updates your feature branch to have a cleaner change history on top of the latest main branch.

`$ git rebase -i <base-branch>`
`or`
`$ git rebase -i main`
	Interactively rebase your current branch onto the main branch, enabling you to review and modify the commits your are rebasing. This process opens an editor where you can decide what to do with each commit. Each commit is prefixed with the word pick. You can replace the word pick with any of the following commands:
	• **pick**: Use the commit as it is (no changes).
	• **reword**: Use the commit, but modify the commit message.
	• **edit**: Pause the rebase process and allow changes to the commit (e.g., to edit the code in the commit).
	• **squash** (or **s**): Combine this commit with the previous one. The commit message can be edited.
	• **fixup** (or **f**): Like squash, but the commit message is discarded.
	• **drop**: Remove the commit completely.
	• **exec**: Run a shell command during the rebase.

`$ git rebase --continue`	After resolving conflicts during a rebase, continue the process.
`$ git rebase --abort`	Abort the rebase and return to the state before it started.

GitHub Clone & Remote

`$ git clone <URL>`	Download a repository to your local machine.
`$ git remote -v`	View remote repository connections.
`$ git remote add <remoteName> <remoteURL>`	Add remote repository
`$ git remote remove <name>`	Remove a remote repository.
`$ git remote rename <oldName> <newname>`	Rename a remote repository
`$ git remote show <name>`	See more information about a remote repository.
`$ git remote set-url <name> <newUrl>`	Change the URL of an existing remote repository.

GitHub Push

`$ git push <remote> <branch>`	Push changes to the remote repository.
`$ git push --set-upstream <remote> <newBranch>` `$ git push -u <remote> <newBranch>`	Push a new local branch that doesn't exist on the remote. Set it up to track and link to the local branch, so that next time we can just use "git push" without specifying anything.
`$ git push –all <remote>`	Push all branches to remote.
`$ git push <remote> --delete <oldBranch>`	Delete a branch from the remote repository.
`$ git push --dry-run <remote> <branch>`	Simulate a push and see what would happen without actually making any changes to the remote.

GitHub Fetch

`$ git fetch` `$ git fetch <remote>`	Download changes from the remote without applying them.
`$ git fetch <remote> <refspec>`	<refspec>: Specifies what branches or commits to fetch. You can leave this out to fetch all branches.
`$ git fetch --prune`	Removes references to remote branches that have been deleted from the remote repository, cleaning up stale branches in your local copy while fetching the latest updates

`$ git fetch --dry-run <remote>`	Simulates fetching updates from the specified remote, showing what changes would be fetched without actually downloading or applying them
`$ git fetch --tags`	Fetches new tages from the remote.
`$ git fetch --depth=5`	Fetches only the latest 5 commits from the remote.

GitHub Pull

`$ git pull <remote> <branch>`	Fetch and Merge changes from the remote repository into your local branch.
`$ git pull --rebase <remote> <branch>`	Fetches the latest changes from the remote and then rebases your local commits on top of those changes. Rebase rewrites your commit history, keeping it linear without merge commits
`$ git pull --all`	Pull all branches at once.
`$ git pull --ff-only <remote> <branch>`	Updates your current branch with changes from the specified remote branch, but only if a fast-forward merge is possible, meaning no merge commits will be created
`$ git pull --no-commit <remote> <branch>`	Fetches changes from the specified remote branch and merges them into your current branch without automatically creating a commit, allowing you to review or modify the merge before committing
`$ git pull --dry-run <remote> <branch>`	Simulates pulling changes from the remote branch, showing what would be updated without actually applying any changes to your local branch
`$ git pull --verbose origin main`	Shows detailed output about what is being fetched and merged.

GitHub Workflow

Issues	Use GitHub Issues to track tasks, bugs, and feature requests
Pull Requests (PR)	Propose changes, review code, and merge into the main project.
Labels & Milestones	Organize issues and PRs with labels and milestones.
Projects	Use project boards for Kanban-style task management.
Forks	Forking creates a personal copy of someone else's GitHub repository in your account, allowing you to freely make changes without affecting the original project.