GIT

What is Git?

Git is a version control system. It helps developers track changes in their code, collaborate with others, and manage different versions of a project.

Why to use Git?

- **Track changes** - Git saves the history of changes so you can go back to previous versions of your project anytime.
- **Collaboration** - Multiple people can work on the same project at the same time.
- **Branching** - You can create separate “branches” to test new features without affecting the main project.
- **Safety** - Mistakes can be easily undone.
- **Remote sharing** - Share and sync your code with others using platforms like GitHub or GitLab.

How It works Git (simplified)?

- **Repository (Repo)** - A folder that Git tracks. It can be on your computer or online. This is the folder if your project and Git tracks it.
- **Commit** - A snapshot of your project at a certain time (like saving a game).
- **Branch** - A separate line of development. Example: main, feature-login
- **Merge** - Combining changes from different branches.
- **Push** - Send your changes to a shared (remote) repository (repo).
- **Pull** - Get the lates changes from the shared repository (repo).

Git commands:

Initial configuration:

Set url and the credentials of the user for the repository:
git remote set-url origin https://username:token@github.com/ownerusername/reponame.git

Set up initial Git configuration globally. This information is related to the commits:
git config --global user.name "Your Personal Name"
git config --global user.email "your-personal-email@example.com"

Set up initial Git configuration locally for specific repository:
git config user.name "Your Personal Name"
git config user.email "your-personal-email@example.com"

Set credential manager globally. Our user names and personal access tokens is more secure to be saved in the credential manager for each OS instead in gitconfig file as a plain text:
git config --global credential.helper manager

Getting repository:

Initialize a new local repository:
git init

Clone remote repository with it's full history locally:
git clone <repository_url>

Clone remote repository with it's full history locally in custom folder:
git clone <repository_url> <folder_name>

Working directory:

Restore all working directory changes to the state of the last commit:
git restore .

Stage and Unstage:

Add file/s to the index area (staging file):
git add <file1_name> <file2_name>

Add all content (files/folders) of the current directory to the staging area:
git add .

Restore all staged changes to the same changes in the working directory:
git restore --staged .

Commiting

Make a commit (commit all staged files)
git commit -m "Commit message"

Make a commit - will be opened the default editor for setting of commit message:
git commit

Update already made commit but first the new change must be staged:
git commit --amend -m "New commit message"

Getting repository state

Show the full status of the repository:
git status

Show the status of the repo in shorter way:
git status -s

Getting repository history

Viewing the history in detail view:
git log

Viewing the history in short view in single lines:
git log --oneline

Viewing staged and unstaged info

Viewing all changes in the working directory - (difftool must be configured before that):
git difftool

Viewing changes in the staging area (index) - (difftool must be configured before that):
git difftool --staged

Stashing

Make a stash with specified name:
git stash push -m "Your stash message"

Lists all stashes:
git stash list

Apply the stash to the working directory. The 0 is used but can be 1, 2, 3 depending from the stash number:
git stash apply stash@{0}

Deletes the current stash:
git stash drop stash@{0}

Deletes all stashes:
git stash clear

Branching and merging locally

Creates local branch and switch to it:
git switch -C <branch_name>

Deletes local branch:
git branch -d <branch_name>

Merges branch <branch_name> into the current one in which we are:
git merge <branch_name>

Merge branch <branch_name> with merge commit even if fast forward merge is possible:
git merge --no-ff <branch_name>

Performs squash merge:
git merge --squash <branch_name>

Aborts the merge:
git merge --abort

Change the base of the current branch:
git rebase master

Applies the commit on the current branch:
git cherry-pick <commit_hash>

Pulling and Pushing

Pulls remote work (updates) and merge them into our local branch.
Before that is needed our local branch to be set to track it's remote one:
git pull

Fetches the remote work (updates) but NOT merge it into the local branch.
Before that is needed our local branch to be set to track it's remote one:
git fetch

Pushes the latest work from our branch into the remote repository.
See the section "BRANCHING AND MERGING REMOTELY" for more info:
git push

Branching and merging remotely

Push the local branch_name to the remote repository:
git push -u origin branch_name

Delete remote branch:
git push -d origin branch_name

Show remote branches:
git branch -r

Set local branch to track remote branch:
git switch -C branch_name origin/branch_name

Tags

Creates tag on the last commit:
git tag <tag_name>

Creates tag on earlier commit:
git tag <tag_name> <commit_hash>

Lists all tags:
git tag

Deletes the tag locally:
git tag -d <tag_name>

Push the tag in the remote repository:
git push origin <tag_name>

Deletes remote tag:
git push origin --delete tag <tag_name>

Other useful commands

Checks out the given commit (reset the repo state to this commit state):
git checkout <commit_hash>

Removes all untracked files from the repository:
git clean -fd

Creates alias called "lg" for the command "log --oneline":
git config --global alias.lg “log --oneline"

Shows the author of each line in the file:
git blame <file1_name>

//NEVER NEVER NEVER REWRITE HISTORY THAT IS ALREADY PUBLIC!!!