| 1. Git-HOL |
| --- |

======================================================

Step 1: Configure Git

======================================================

Before you can use Git effectively, you need to tell Git who you are.

This information (name and email) will be recorded in every commit you make.

Without this, Git won’t be able to associate your changes with your identity.

1. Check if Git is installed and see the version:

git --version

# This confirms Git is installed. Example output: "git version 2.43.0"

2. Set your name for all Git projects on this computer:

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

# Replace "Your Name" with your real full name. Example: "Anuj Mishra"

# The --global flag means it applies to all repositories on this system.

3. Set your email for all Git projects on this computer:

git config --global user.email "your.email@example.com"

# Replace with the same email you use for GitHub or your Git hosting service.

4. Verify that the name and email were saved correctly:

git config --list

# This lists all Git configuration settings. Look for 'user.name' and 'user.email'.

======================================================

Step 2: Set Default Editor (Optional but Recommended)

======================================================

Git sometimes needs to open an editor — for example, if you run 'git commit' without the -m flag,

or if you need to resolve merge conflicts. By default, Git might use Vim or Nano, which may be

unfamiliar to beginners. You can change it to an editor you are comfortable with.

1. If you are on Windows and want to use Notepad++ as your Git editor:

git config --global core.editor "notepad++ -multiInst -notabbar -nosession -noPlugin"

# This command tells Git to use Notepad++ and also starts it without extra tabs or plugins.

2. If you are on Linux or Mac and want to use Visual Studio Code as your Git editor:

git config --global core.editor "code --wait"

# The '--wait' option tells VS Code to keep the terminal waiting until you close the file.

3. You can check which editor is currently set by running:

git config core.editor

======================================================

Step 3: Create a Local Repository and Add a File

======================================================

A Git repository is basically a folder with your project files plus a hidden '.git' folder that stores

the entire history of your changes. In this step, you will create a project, initialize Git, and make your first commit.

1. Create a new folder for your project:

mkdir GitDemo

# 'mkdir' stands for Make Directory.

# This creates a folder named GitDemo in your current location.

2. Move into the new folder:

cd GitDemo

# 'cd' stands for Change Directory.

# This makes GitDemo your working directory for all next commands.

3. Initialize a Git repository in this folder:

git init

# This creates a hidden '.git' folder inside GitDemo where Git stores all version history.

# You will now see: "Initialized empty Git repository in /path/to/GitDemo/.git/"

4. Verify that '.git' was created:

ls -a

# The -a flag lists ALL files, including hidden ones.

# You should see a folder named '.git' — that’s the heart of your repository.

5. Create a new file named welcome.txt and add some text to it:

echo "Welcome to the GitDemo project" > welcome.txt

# This creates a text file with one line of content.

6. View the file to ensure it’s correct:

cat welcome.txt

# This will print: Welcome to the GitDemo project

7. Check the repository status:

git status

# This shows the file 'welcome.txt' under "Untracked files".

# "Untracked" means Git knows the file exists, but it is NOT being tracked yet.

8. Stage the file for committing:

git add welcome.txt

# This moves the file into the "staging area".

# The staging area is like a holding zone for files that will be included in your next commit.

9. Check status again:

git status

# Now the file should appear under "Changes to be committed".

# This means it’s staged and ready for commit.

10. Commit the file to the repository:

git commit -m "Add welcome.txt"

# The '-m' flag lets you add a commit message directly in the command.

# This creates a permanent snapshot of the file in the Git history.

11. Check status once more:

git status

# It should now say "nothing to commit, working tree clean".

# This means all your changes are saved in Git.

======================================================

Step 4: Link to a Remote Repository and Synchronize

======================================================

A remote repository is a copy of your Git repository stored on a server (e.g., GitHub, GitLab, Bitbucket).

This lets you share your code with others and keep it backed up online.

1. Link your local repository to a remote one:

git remote add origin <URL>

# Replace <URL> with your repository's HTTPS or SSH link from GitHub or GitLab.

# 'origin' is just the default name for your primary remote.

2. Push your local commits to the remote master branch:

git push origin master

# 'push' uploads your commits to the remote server so others (or you from another computer) can see them.

3. Pull the latest changes from the remote:

git pull origin master

# 'pull' fetches changes from the remote repository and merges them into your local branch.

# This ensures you have the latest version from others who may have worked on the project.