1. What Is Git?

Git is a distributed version control system that tracks changes in your files over time. It allows you to record snapshots of your project, manage multiple versions, and easily collaborate with others. By using Git, you can experiment without fear because you have a complete history of your work.

---

2. Installing Git

For Windows:
- Download the installer from Git for Windows.
- Run the installer, accepting the default options unless you have specific preferences.

For macOS:
- You can install Git via Homebrew with:`brew install git`
- Alternatively, download Git from git-scm.com.

For Linux:
- Use your package manager. For Debian-based distros:
`bash sudo apt update sudo apt install git` After installation, verify it by running:

```
git --version
```

---

3. Configuring Git

Before you start using Git, set your user details so that your commits are properly identified.

Run these commands in your terminal:

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

These settings store your identity globally. You can also set configurations per repository if needed.

---

4. Creating Your First Repository

A repository (or “repo”) stores your project files and the history of changes.

To create a new repository in an existing project folder: 1. Open your terminal and navigate to your project folder. 2. Initialize the repository:

`bash` `git init` This command creates a hidden `.git` folder that manages all version control data.

Or, to start with a new directory:

```
mkdir my-project
cd my-project
git init
```

---

5. Tracking and Committing Files

A. Check Repository Status

After initializing your repository, see what Git recognizes (untracked files, modified files, etc.) by running:

```
git status
```

B. Staging Files

To include changes in your next commit, you must stage them. For example, to stage one file:

```
git add filename.ext
```

Or to stage all changes in your directory:

```
git add .
```

C. Making a Commit

A commit is a snapshot of your current project state. Write concise and meaningful commit messages. Commit your staged files with:

```
git commit -m "Add initial project structure"
```

You can review your commit history later using:

```
git log
```

---

6. Working with Branches

Branches allow you to work on new features or fixes without affecting the main codebase.

A. Creating and Switching Branches

Make a new branch and switch to it with:

```
git checkout -b feature-branch
```

This creates a branch called `feature-branch` and switches you to it.

B. Merging Branches

Once your feature is complete, switch back to your main branch (often called `main` or `master`) and merge the changes:

```
git checkout main
git merge feature-branch
```

If you encounter conflicts, Git will guide you through the process to resolve them before finalizing the merge.

---

7. Working with Remote Repositories

Remote repositories (for example on GitHub, GitLab, or Bitbucket) allow you to back up your code and collaborate with others.

A. Adding a Remote Repository

After creating a repository on the hosting service, add it as the remote:

```
git remote add origin https://github.com/yourusername/your-
        repo.git
```

B. Pushing Changes

Send your commits to the remote repository with:

```
git push -u origin main
```

The `-u` flag sets the remote branch as the default upstream for future pushes.

C. Cloning a Repository

To work on an existing remote repository, clone it to your machine:

```
git clone https://github.com/username/repo.git
```

D. Pulling Changes

Before you start working, fetch the latest changes from the remote:

```
git pull
```

---

8. Helpful Git Commands

- **`git status`**: Check which files have changed.
- **`git diff`**: See the differences between your working directory and the last commit.
- **`git branch`**: List, create, or delete branches.
- **`git log`**: View commit history.
- **`git remote -v`**: Show the URLs for your remote repositories.

Using these commands frequently will help you stay oriented and manage your project effectively.

---

9. Best Practices for Beginners

- **Commit Often:** Save snapshots of your project as you progress.
- **Write Descriptive Messages:** Help you (and others) understand your changes later.
- **Use Branches:** Experiment and develop features separately to keep the main branch stable.
- **Regularly Pull Updates:** If collaborating, ensure you integrate others' changes periodically to avoid large conflicts.
- **Explore with Confidence:** Don't be afraid—Git is designed to help you recover from mistakes via commands such as `git reset`, `git revert`, and even branching out if things go astray.
