Git, Python and ML Shortcuts for internship

230057896

Sahas Talasila

Git

1. Start by using `mkdir` command to create the project folder.

2. then use:

```
cd /path/to/your/project
```

3. Then initialise using the command below so that git will actually track the changes in your project folder:

```
git init
```

4. Then add a 'README.md' markdown file which will allow for a user to understand how the project actually works:

```
touch README.md
```

5. Add a '.gitignore' file which will prevent other users from getting access to certain files that they should not have access to (API keys, certain personal details)

6. Then add to the git tracking:

```
git add .
```

7. Then save and commit the changes with a descriptive commit message:

```
git commit -m "Initial commit"
```

8. Log in to GitHub, go to GitHub, and create a new repository called TestProject. Make sure to select "Private" so that only you can access it.

9. Link your local project to GitHub Once the repository is created, copy its URL and add it to your local repository:

```
git remote add origin https://github.com/YOUR_USERNAME/TestProject.git
```

10. Then for simplicity, rename to 'main'.

```
git branch -M main
```

11. Then, finally push to the main branch and push to the remote repository on Github

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

```
git pull origin main --rebase
```

for moving from one directory to another

```
mv
```

Follow these steps to clone a repository:

1. Navigate to the GitHub repository page.
2. Click on the green **Code** button.

3. Copy the repository URL (HTTPS or SSH).
4. Open a terminal and run the following command:

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

Replace `username` and `repository` with the actual repository owner and name.

Navigating and Using the Repository Once cloned, move into the repository directory:

```
cd repository
```

You can now edit files, create branches, and push changes.

1.1 Cloning

Cloning a repository is the first step in working on a GitHub project locally. After cloning, you can modify code, commit changes, and sync them with the remote repository.

For more details, visit Git Documentation.

1.2 Git Branching

Creating a new branch:

`git branch <branch-name>`

This command creates a new branch, allowing you to work on features separately from the main branch.

Listing all branches:

```
    git branch
```

Use this command to view all branches in your repository.

Switching to an existing branch:

```
git checkout <branch-name>
```

Alternatively, with newer Git versions:

```
git switch <branch-name>
```

These commands allow you to move between different branches.

Renaming a branch:

```
git branch -m <new-branch-name>
```

Useful for reorganizing branch names after creation.

Deleting a branch:

```
git branch -d <branch-name>
```

Removes a branch that has been merged. Use '-D' instead if you need to force-delete an unmerged branch.

Pushing a branch to a remote repository:

```
git push -u origin <branch-name>
```

This command sends the branch to the remote server for collaboration.

Fetching remote branches:

```
git fetch --all
```

Syncs your local repository with all available remote branches.

Deleting a remote branch:

```
git push origin --delete <branch-name>
```

Used to clean up obsolete branches from the remote repository.

Merging a branch into the current branch:

```
git merge <branch-name>
```

Brings the changes from another branch into the active branch.

Checking branch status:

```
git status
```

Displays the current branch, uncommitted changes, and files staged for commit.

1.3 Creating a Pull Request in Git

A **Pull Request (PR)** allows you to propose changes to a repository and request a review before merging. Follow these steps:

1. Create a new branch:

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

This creates a new branch for your changes.

2. Make changes and commit: Edit files, then add and commit your changes:

```
git add .
git commit -m "Added new feature"
```

3. Push the branch to the remote repository:

```
git push origin feature-branch
```

4. Create a Pull Request on GitHub: 1. Go to the repository on GitHub. 2. Click **"Pull requests"**, then **"New pull request"**. 3. Select ‘main‘ as the base and ‘feature-branch‘ as the compare branch. 4. Add a title and description. 5. Click **"Create pull request"**.

5. Review and Merge: Once reviewed, merge the PR:

```
git merge feature-branch
```

Now your changes are successfully integrated!

1.4 Resolving Merge Conflicts in Git

Sometimes, when merging branches, Git detects conflicts. A file with conflicts may look like this:

```
<<<<<<< HEAD
Your current branch changes
=======
The other branch's changes
>>>>>>> branch-name/
```

To resolve the conflict: 1. Manually edit the file to **keep the correct changes**. 2. Remove the conflict markers (‘<<<<<<<’, ‘=======’, ‘>>>>>>>’). 3. Stage and commit the resolved file:

```
git add <conflicted-file>
git commit -m "Resolved merge conflict"
```

Now the conflict is resolved, and the merge is

Command	Description	Usage Scenario
git init	Initializes a new Git repository	Starting a new project
git clone <repo-url>	Clones an existing repository	Getting a copy of a remote project
git add <file>	Stages a file for commit	Preparing changes for commit
git commit -m ”message”	Saves changes with a message	Recording changes in history
git status	Shows the current state of the repository	Checking which files are modified or staged
git log	Displays commit history	Reviewing past changes
git diff	Shows differences between commits or branches	Comparing changes before committing
git branch	Lists all branches	Checking available branches
git checkout <branch>	Switches to another branch	Working on a different feature
git switch <branch>	Alternative to checkout for switching branches	Moving between branches efficiently
git merge <branch>	Merges another branch into the current one	Combining changes from different branches
git pull origin <branch>	Fetches and merges changes from a remote repository	Updating local repository with latest changes
git push origin <branch>	Sends local commits to a remote repository	Sharing changes with others
git remote -v	Shows remote repository URLs	Checking remote connections
git reset –hard <commit>	Resets repository to a specific commit	Undoing changes completely
git stash	Temporarily saves uncommitted changes	Switching tasks without committing

Command	Description	Usage Scenario
git tag <tag-name>	Creates a tag for a specific commit	Marking important versions
git rm <file>	Removes a file from the repository	Deleting files from version control
git show <commit>	Displays details of a specific commit	Inspecting changes in a commit

1.5 Bash Commands and Their Uses

Command	Description	Usage Scenario
ls	Lists files in a directory	Checking available files
cd <directory>	Changes the current directory	Navigating through folders
mkdir <directory>	Creates a new directory	Organizing files
rm -rf <directory>	Deletes a directory and its contents	Removing unwanted files
touch <file>	Creates an empty file	Initializing new files
echo "text" >file.txt	Writes text to a file	Creating simple text files
cat <file>	Displays file contents	Viewing file data
grep "pattern" <file>	Searches for a pattern in a file	Finding specific text in logs
chmod +x <file>	Makes a file executable	Running scripts
ps aux	Lists running processes	Checking system activity
kill <PID>	Terminates a process	Stopping unresponsive applications
tar -cvf archive.tar <directory>	Creates a compressed archive	Backing up files
scp <file>userserver:/path	Securely copies files to a remote server	Transferring files over SSH
ping <host>	Checks network connectivity	Troubleshooting network issues
curl <URL>	Fetches data from a URL	Testing API responses

1.6 Setting Up SSH

Follow these steps to configure SSH for secure access.

1. Check if SSH is installed:

```
ssh -V
```

2. Generate an SSH key:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Press **Enter** to save it to the default location (‘`~/.ssh/id_rsa`‘).

3. Add the SSH key to your agent:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

4. Copy and add the SSH key to a remote service (e.g., GitHub):

```
cat ~/.ssh/id_rsa.pub
```

Copy the output and add it to your GitHub SSH settings.

5. Test the SSH connection:

```
ssh -T git@github.com
```

If successful, your SSH setup is complete!

Python Key tricks

1. Efficient Data Loading with Pandas Using '`low_memory=False`' prevents memory fragmentation, and specifying 'dtype' reduces memory usage.

2. Vectorized Operations with NumPy Vectorization eliminates slow Python loops, improving performance.

3. Memory-Efficient Generators for Large Datasets Generators prevent excessive memory usage when handling large datasets.

4. Parallel Processing with Multiprocessing Multiprocessing speeds up computations by **parallelizing operations**.

5. Custom Data Preprocessing Without Sklearn Custom normalization avoids reliance on third-party ML libraries.

6. Loading and Preprocessing Data Using TensorFlow Using 'tf.data' provides efficient streaming for large datasets.

7. PyTorch-Based Neural Network for ML PyTorch offers flexible ML models with **custom architecture**.

8. Using TensorFlow for Deep Learning TensorFlow simplifies deep learning training without external dependencies.

9. Making Predictions with TensorFlow and PyTorch Both TensorFlow and PyTorch provide easy inference methods.

10. Saving and Loading Models Without Sklearn Saving and loading models ensures reproducibility in ML workflows.

2.1 1. Define the Problem

Before building a Machine Learning model, it is essential to understand the objective:

- Identify whether the task is **classification**, **regression**, or **clustering**.

- Determine the data sources required.
- Establish success metrics (e.g. accuracy, precision, mean squared error).

2.2 2. Collect and Preprocess Data

Data preparation is crucial for model accuracy. Steps include:

2.3 3. Split the Data

Dividing the data ensures reliable model evaluation:

2.4 4. Choose the Model Type

Select an appropriate model depending on the problem:

- **Linear models**: Suitable for numerical trends.
- **Decision trees and random forests**: Effective for structured data.
- **Neural networks**: Ideal for deep learning and complex patterns.

2.5 5. Train the Model

The model is trained using the provided dataset:

2.6 6. Evaluate the Model

Assess model performance using various metrics:

2.7 7. Tune Hyperparameters

Optimising hyperparameters improves model efficiency:

2.8 8. Deploy the Model

Once trained, the model can be deployed for real-world use:

2.9 9. Monitor and Improve the Model

Continual monitoring ensures long-term success:

- Track performance using automated logging.
- Retrain the model periodically with new data.
- Implement MLOps frameworks for efficient model updates.