Git & GitHub Hands-On Workshop for Data Science

Tools: Git, GitHub, Poetry, VS Code or Terminal

Workshop Goals

By the end of this session, you will be able to:
- Use Git to track and manage your code.
- Create and merge branches safely.
- Resolve conflicts and collaborate using GitHub.
- Use advanced Git tools (stash, cherry-pick, tags).
- Build and maintain a reproducible project structure.
- Manage Python environments with Poetry.

Prerequisites

Make sure you have the following installed:
- Poetry
- A GitHub account
- A code editor (VS Code recommended)

Setup (if required):

- Vs code - install from https://code.visualstudio.com/download
- Git - git conda install -c conda-forge git
- Poetry – pip install poetry

Clone and Explore

1. Clone the starter repo:

```
git clone https://github.com/aakc3007/open-data-version-control.git
cd open-data-version-control
```

2. Explore the folders:
   - notebooks/: for EDA and model training
   - src/: modular code (model_code/, utils/, tests/, data/, models/)

- pyproject.toml: dependency file
- .gitignore, README.md

3. Create an env

```
conda create -n python_310 python=3.10
conda activate python_310
pip install poetry
poetry install
```

Git Basics

1. Configure Git identity:

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

2. Check status and initialize Git:

```
git status
```

---------- Just wait here ----------

3. Add and commit a new file:

```
echo "# helper functions" > src/utils/helper.py
git add src/utils/helper.py
git commit -m "Add helper.py with placeholder"
```

4. View commit history:

```
git log --oneline
```

---------- Issue ----------

5. Need to pull and merge all commits

```
git pull origin main –-rebase
```

6. If conflict:

```
git add src/utils/helper.py
git rebase --continue
git push origin main
```

New personal repo

1. Create a new repo

git clone --bare https://github.com/aakc3007/open-data-version-control.git
cd open-data-version-control.git
git push --mirror <target_repo link>
cd ..
rm -rf open-data-version-control.git

Branching & Merging (Add changes to main without pushing a new branch)

1. Create a new branch:

```
git checkout -b <branch_name>
```

2. Add code and commit:

```
echo "print('cleaning data')" > src/utils/clean.py
git add src/utils/clean.py
git commit -m "Add basic data cleaning script"
```

3. Merge it back into main:

```
git checkout main
git merge <branch_name>
```

Conflict Handling

1. Simulate conflict: Edit the same file on two branches

git branch
>> main

Edit the helpers.py to put this in
“Try and create new conflict in main”

git add .
git commit -m “Added conflict in main”

git checkout -b conflicting-branch-1

Edit the helpers.py to put this in
“added conflict in conflicting-branch-1”

git commit -am "Conflict from conflicting-branch-1"

git checkout main
git checkout -b conflicting-branch-2

Edit the helpers.py to put this in
“added conflict in conflicting-branch-2”

git commit -am "conflict from conflicting-branch-2"

git checkout conflicting-branch-2
git merge conflicting-branch-1

3. Fix conflict manually, then:

```
git add .
git commit -m "Resolve merge conflict in helper.py"
```

Push to GitHub & Pull Requests

1. Push to GitHub:

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

Add remote (if needed):
git remote add origin <github repo link>

2. Create a new branch and push:

```
git checkout -b pr-branch
echo "print('Git is awesome')" > print.py
git add print.py
git commit -m "Print message for demo"
git push -u origin pr-branch
```

3. Open GitHub → Create Pull Request → Review → Merge

Advanced Git Tools

- Stash work:

1. Add updates in main branch

  git checkout main

  Make changes in the clean.py file

  git status

2. Stash these changes and create another branch

git stash

git checkout -b stash-branch

echo "print('New work')" >> newfile.py
git add newfile.py
git commit -m "New branch work"

3. Go back to main and restore changes

  Make changes in clean.py

  git checkout main

  git stash pop

- Cherry-pick commit:
  git log --oneline # copy commit hash
  git checkout main
  git cherry-pick <commit-hash>

- Tag a version:
  git tag -a v1.0 -m "First release version"
  git push origin v1.0

- Cherry-pick work:

1. Add changes in main branch

git checkout main
git push -u origin main

Add changed in clean.py/helper.py

git commit -am “Added changed in clean.py”

git log --oneline

2. Go to old branch and pick changes

git checkout stash-branch
git cherry-pick <commit-hash>

if conflicts then resolve it

- Tag a Version:

1. After merging and testing, tag it with a release version
git tag -a v1.0 -m “first release version”

2. Push annotated tag
git push origin v1.0