ADVANCED TOPIC OF GIT AND GITHUB IN DS

Yes, there are additional Git and GitHub commands and practices that are particularly useful in data science projects. These commands help in managing data files, collaborating efficiently, and maintaining a clean project history. Here are some more advanced and specific commands:

**1. Working with Large Files and Datasets**

* **Git Large File Storage (LFS)**:
  + Git LFS is an extension for Git that allows you to version large files such as datasets, models, and binaries that are too big for regular Git.
  + **Install Git LFS**:

Command : git lfs install

* + **Track Large Files**:

Command : git lfs track "\*.csv"

* + - This tracks all CSV files with Git LFS.
  + **Check Status of LFS Files**:

Command : git lfs status

**2. Managing Commits and History**

* **Interactive Rebase**:

Command : git rebase -i <commit-hash>

* + Allows you to edit, combine, or delete commits in your history. Useful for cleaning up commits before pushing to a shared repository.
* **Squash Commits**:
  + During a rebase, you can squash multiple commits into one. This is useful when you want to consolidate your work before merging it into the main branch.
* **Cherry-Pick a Commit**:

Command : git cherry-pick <commit-hash>

* + Applies a specific commit from another branch into your current branch. This is useful if you want to bring in a particular feature or fix without merging an entire branch.

**3. Advanced Collaboration**

* **Stashing Changes**:

Command : git stash

* + Temporarily saves your changes and cleans your working directory. Useful when you need to switch branches but aren't ready to commit your changes.
  + **Apply Stashed Changes**:

Command : git stash apply

* + **List Stashed Changes**:

Command : git stash list

* **Resolving Merge Conflicts**:
  + If you encounter conflicts while merging branches, Git will prompt you to resolve them manually. After resolving the conflicts, mark them as resolved:

Command : git add <resolved-file>

Command : git commit

**4. Automating and Streamlining Workflows**

* **Git Hooks**:
  + Git hooks are scripts that run automatically at certain points in your Git workflow, such as before a commit or after a merge.
  + **Create a Pre-commit Hook** (e.g., to run tests before committing):

Command : nano .git/hooks/pre-commit

* + - Add your script to automate tasks, such as running tests or checking code style.
* **Git Aliases**:
  + Create shortcuts for frequently used commands.

Command : git config --global alias.co checkout

Command : git config --global alias.br branch

**5. Managing Collaboration on GitHub**

* **GitHub Actions**:
  + Automate workflows directly in GitHub, such as running tests or deploying models when you push code to the repository.
  + **Create a Workflow**:
    - Add a YAML file in .github/workflows/ to define your automation pipeline.
    - Example:

Python code

name: Python application

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: 3.x

- name: Install dependencies

run: pip install -r requirements.txt

- name: Run tests

run: pytest

* **GitHub Pages**:
  + Host project documentation or Jupyter notebooks directly from your repository on GitHub Pages.
  + **Enable GitHub Pages**:
    - Go to the repository settings and enable GitHub Pages, usually from the gh-pages branch or /docs folder.
* **GitHub Issues and Projects**:
  + Use GitHub Issues to track bugs, enhancements, and tasks.
  + Use GitHub Projects to manage and visualize project tasks in a Kanban board.

**6. Handling Multiple Collaborators**

* **Forking and Pull Requests**:
  + **Fork a Repository**:
    - Forking creates your own copy of someone else’s repository under your account. You can make changes and then submit those changes back to the original repository via a pull request.
  + **Create a Pull Request**:
    - After pushing changes to your fork, you can create a pull request on GitHub to propose your changes.
* **Code Reviews**:
  + Use GitHub’s built-in code review features to review pull requests, add comments, and suggest changes before merging code into the main branch.

**7. Debugging and Troubleshooting**

* **Blame a File**:

Command : git blame <file-name>

* + Shows who last modified each line of a file. Useful for tracking down the origin of bugs or changes in the code.
* **Reflog**:

Command : git reflog

* + Shows a log of all actions (e.g., commits, checkouts) that have been performed on your local repository. Useful for recovering lost commits or undoing changes.

**8. Integration with Other Tools**

* **Jupyter Notebooks in Git**:
  + Managing Jupyter notebooks in Git can be tricky because of the large JSON format. Use nbstripout to strip output from notebooks before committing:

Command : pip install nbstripout

nbstripout --install

* + This prevents large outputs from cluttering your Git history.
* **Versioning Data**:
  + For versioning data along with code, tools like **DVC (Data Version Control)** can be integrated with Git:

dvc init

dvc add data.csv

git add data.csv.dvc .gitignore

By mastering these additional commands and tools, you can handle more complex workflows, collaborate more effectively, and maintain a robust and clean version control system for your data science projects.