**SCRIPT MODULE 8**

**Best Practices in Commits & Branching**

**Commit Best Practices**

1. **Atomic Commits  
   → Each commit should represent one logical change.  
   *(e.g., Fixing a typo is separate from refactoring a function.)***
2. **Write Meaningful Commit Messages  
   → Start with a verb: Add, Fix, Refactor, Update, Remove.  
   → Format:  
   feat: Add login form validation  
   fix: Resolve broken image path on homepage**
3. **Use Conventional Commits (Optional)  
   → Follow standards like feat, fix, docs, refactor, etc., for clarity.  
   → Helps with automated changelogs and CI tools.**
4. **Commit Frequently, But Not Excessively  
   → Avoid one huge commit or dozens of tiny ones for minor edits.**
5. **Avoid Committing Sensitive Data  
   → No passwords, API keys, or .env files!**
6. **Use .gitignore Wisely  
   → Exclude logs, OS files, IDE settings, compiled files.**

**Branching main practices**

**Follow Naming Conventions  
→ feature/login-page, bugfix/navbar, hotfix/login-crash**

**Use Short-Lived Feature Branches  
→ Create a branch for a specific task, and delete it after merging.**

**Keep Branches Up-to-Date with Main  
→ Regularly pull from main (git pull origin main) to avoid merge hell.**

**Avoid Working Directly on Main  
→ Keep main stable, always work in branches.**

**Use Pull Requests (PRs) for Merging  
→ Ensures peer review, feedback, and traceability.**

**Always Review Before Merging  
→ Self-review your code and check for broken logic or untracked files.**

* **Mention how you used branching in a team:  
  *“We used feature branches for development, bugfix branches during testing, and hotfixes for urgent issues in production.”***
* **Talk about avoiding merge conflicts by communicating regularly:  
  *“We coordinated our branching strategies during daily stand-ups.”***

**"When working in real-world teams, Git history becomes a record of collaboration. So, commit messages must be clear and readable.**

**"In any collaborative coding environment, how you manage your commits and branches speaks volumes about your professionalism. Think of commits like journal entries—they should tell the story of your development progress. And branching is like working in your own safe sandbox before merging your work into the main project. These best practices not only make your life easier, they also help your teammates and future employers understand your work better."**

**Real Git Questions in Interviews**

* **What’s the difference between git fetch vs git pull?**
  + **Fetches latest commits and branches from the remote, but doesn't merge them into your local branch. Think of it as checking for updates.**
    - **Checking your mailbox but not opening the letters.**
  + **Git pull: It does both git fetch + git merge. It updates your local branch with remote changes.**
    - **Opening the mailbox and reading all the letters (updates your local branch).**
* **How do you resolve a merge conflict?**
*  Git will mark the conflict in the affected file using <<<<<<<, =======, and >>>>>>>.
*  Open the file in VS Code or your preferred editor.
*  Choose which code you want to keep (or combine).
*  Delete the conflict markers.
*  Save the file.
  + **git add <conflicted-file>**
  + **git commit -m "Resolved merge conflict"**
  + **Tip: Use git status to see which files are in conflict.**
* **When would you use revert vs reset?**
  + **Git revert: To undo a commit without changing history. Creates a new commit that reverses changes.**
  + **Git reset: Moves the HEAD pointer to an older commit. Rewrites history.**
  + **Use revert when collaborating, reset when solo.**
* **What is a fast-forward merge?**
  + **A fast-forward merge occurs when there’s no new commit on the target branch (usually main) since the feature branch diverged.**
* **How do you undo the last commit?**
  + **Soft Undo (keeps changes): git reset --soft HEAD~1**
  + **Hard Undo (removes changes permanently): git reset --hard HEAD~1**
* **Explain your Git branching strategy in your last project**

**"Interviewers love Git questions because Git reveals how you think and collaborate. They test both theory and real-world handling.**

**So, practice answering:**

* **How you resolved a conflict.**
* **What branching model your team followed.**
* **And how you rolled back a mistake using revert vs reset."**

**Tip: Use GitHub Graph to visually explain your branching strategy.**

**Intro to Open Source Contribution**

* **Build real-world experience**
* **Collaborate with developers globally**
* **Get visibility in tech communities**
* **Strengthen your resume and GitHub profile**

**"Open Source is the ultimate real-world playground. It’s not only about code — you can help with documentation, translations, bug testing, and more.**

**Recruiters highly value contributions to known projects like Mozilla, TensorFlow, or even small community tools.**

**It’s also a great way to learn team workflows, Git etiquette, and improve your portfolio."**

**What Are ‘Good First Issues’?**

* **Beginner-friendly bugs/tasks**
* **Usually labeled as: good first issue**
* **Ideal for learning Git workflow and codebase**

**"Many repositories label beginner tasks as good first issue. These are well-described, easy-to-solve issues meant for newcomers.**

**You can search GitHub using this query:**

**is:issue is:open label:"good first issue"**

**You'll find hundreds of opportunities to get started without being overwhelmed."**

**The Fork-Branch-PR Workflow**

1. **Fork a repository**
2. **Clone to your machine**
3. **Create a feature branch**
4. **Make changes → git push**
5. **Raise a Pull Request (PR) to the original repo**

**"You don’t get direct write access to public repositories. Instead, you fork the repo — which means creating a copy in your own account.**

**Then you:**

* **Clone your fork**
* **Create a branch**
* **Make changes**
* **Push and raise a Pull Request**

**This is called the Fork-Branch-PR model, and it’s used by all major open-source projects."**

**GitHub Profile Optimization**

* **Pin top repositories**
* **Use a profile README**
* **Contribute regularly**
* **Add project documentation and visuals**
* **Get stars, forks, and PRs**

**"Your GitHub profile is like a live resume. Recruiters actually check it.**

**You can:**

* **Create a profile-level README (same name as your username)**
* **Pin 6 best projects**
* **Add GIFs/screenshots in your README to make them visual**
* **Keep active contributions (even if small — consistency matters!)**

**Bonus: Add tech stacks used and links to demo or video walkthroughs."**

**Hands-On Activity (Optional)**

* **Fork a public repo with good first issue**
* **Fix the issue and raise a Pull Request**
* **Add a README to your own project repo**
* **Customize your GitHub profile README**
* **Use GitHub Projects board for any personal project**

**Speaker Notes:**

**"If you're ready, try this:**

* **Go to any project with good first issues, fork it, and submit a PR.**
* **Or, create your own GitHub Project board to manage tasks.**
* **Write a meaningful README with images and badges — this shows polish and professionalism."**

**Summary & Career Tips**

* **Use Git daily, even solo**
* **Contribute to open source = real-world experience**
* **Make GitHub your live resume**
* **Practice interview scenarios**

**"Git is no longer optional. Whether you're in dev, data, DevOps, or QA — Git skills show you're a team player.**

**By contributing to open source and optimizing your GitHub profile, you’ll naturally stand out in job interviews.**

**Most importantly — practice Git daily. Your comfort level will shine through."**

**Module-Wise Recap**

**🔹 Module 1: Introduction to Git & GitHub**

***“We began with the ‘why’ — why version control matters. We explored Git as a version control tool and GitHub as a cloud-based collaboration platform. We compared it with real-world examples like saving different versions of your work or checkpoints in video games.”***

**🔹 Module 2: Git Setup & Local Repo Management**

***“You learned how to install Git, set your name and email, and create local repositories. You understood how to track files, move them through the working directory ➡️ staging area ➡️ repository, and how to commit like a pro.”***

**🔹 Module 3: GitHub Integration & Remote Commands**

***“This was the first time we linked local work to GitHub. We created remote repositories, cloned them, and learned how to push and pull changes. We also explored the GitHub interface and how commits are reflected there.”***

**🔹 Module 4: Branching & Merging**

***“Branching gave us the superpower to work on features independently. We learned how to merge safely, deal with merge conflicts, and understand fast-forward vs recursive merges. This module truly mimicked how teams work in real life.”***

**🔹 Module 5: GitHub Collaboration (Issues, PRs, Projects)**

***“This was a collaboration-focused module. We raised issues, assigned them, used labels, and made pull requests (PRs). We saw how GitHub Projects can be used to organize and automate team workflows.”***

**🔹 Module 6: Git Internals & Undoing Mistakes**

***“We dug deeper into Git with commands like git log, git reflog, reset, revert, and stash. You now know how to safely undo mistakes and understand the state machine of Git.”***

**🔹 Module 7: Introduction to GitHub Actions (CI/CD)**

***“This was our first taste of DevOps. You understood what CI/CD is, and how automation can help you run tasks like testing code, checking file standards, or even deploying projects — all triggered by GitHub events.”***

**🔹 Module 8: Best Practices**

***“We ended with best practices — meaningful commits, clean branches, helpful commit messages, and proper collaboration techniques that will make you look like a true professional on any team.”***

**Closing Thoughts**

***“Git and GitHub are essential for any modern developer, analyst, or engineer. With what you've learned, you can now manage your code confidently, collaborate in teams, contribute to open source, and even automate workflows. This isn’t just about commands — it’s about building reliable, trackable, and collaborative workflows for your career.”***