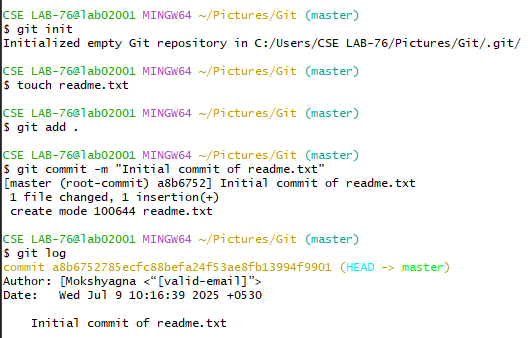
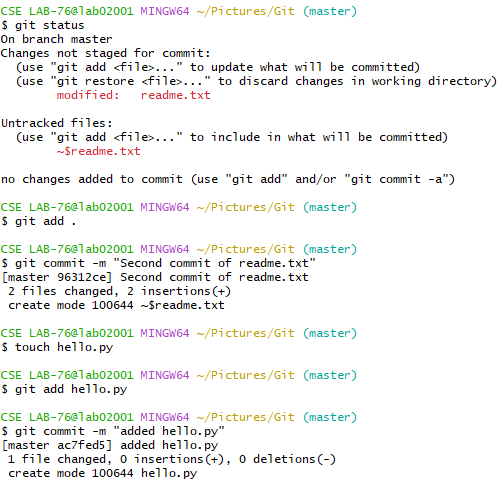
**Module 1: Git Fundamentals (Local Version Control)**

**Lab 1: Getting Started with Git**

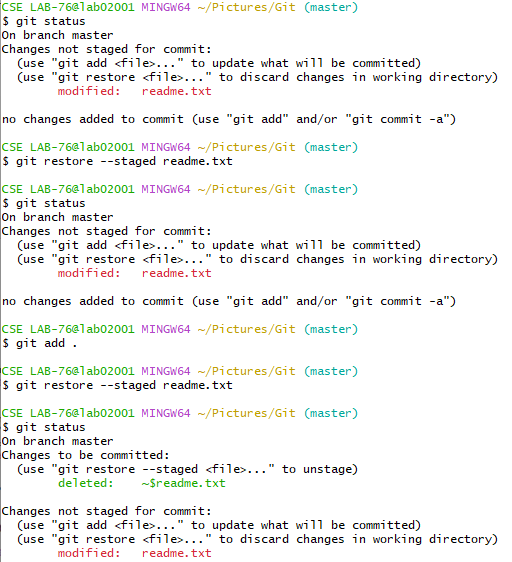
* **Objective:** To introduce basic Git commands for initializing a repository, tracking changes, and committing.
* **Programs:**
  1. **Setup and First Commit:**
     + Install Git on your system.
     + Configure Git with your username and email.
     + Create a new directory for a project (e.g., my\_first\_repo).
     + Initialize a Git repository within this directory (git init).
     + Create a simple text file (e.g., README.txt) with some content.
     + Add the file to the staging area (git add README.txt).
     + Commit the changes with a meaningful message (git commit -m "Initial commit of README.txt").
     + View the commit history (git log).

****

* 1. **Modifying and Tracking Changes:**
     + Modify README.txt and add new content.
     + Check the status of your repository (git status).
     + Stage the modified file.
     + Commit the changes.
     + Create another file (e.g., hello.py).
     + Add and commit hello.py.
     + Practice git diff to see unstaged and staged changes.

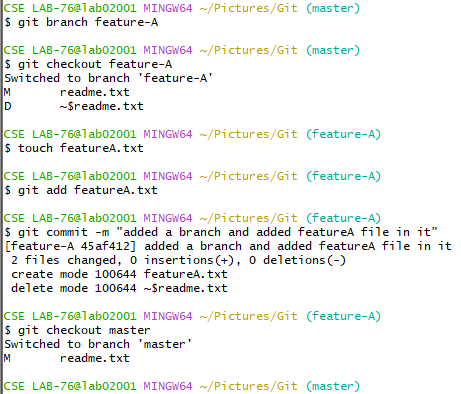


* 1. **Undoing Changes (Basic):**
     + Make a change to README.txt.
     + Use git restore README.txt to discard unstaged changes.
     + Make a change and stage it.
     + Use git restore --staged README.txt to unstage a file.

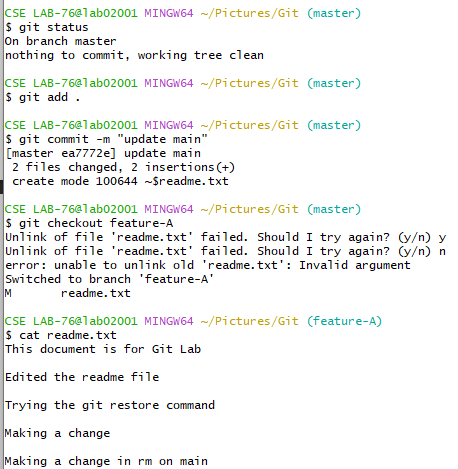


**Lab 2: Branching and Merging**

* **Objective:** To understand the concepts of branching, merging, and resolving conflicts in Git.
* **Programs:**
  1. **Create and Switch Branches:**
     + Create a new branch named feature-A (git branch feature-A).
     + Switch to feature-A (git checkout feature-A).
     + Create a new file (e.g., featureA.txt) and commit it on feature-A.
     + Switch back to the main (or master) branch.



* 1. **Merging Branches:**
     + On the main branch, make a change to README.txt and commit it.
     + Merge feature-A into main (git merge feature-A).
     + Observe the merge process.
  2. **Conflict Resolution:**
     + On main, modify a line in README.txt.
     + On feature-A, modify the *same line* in README.txt differently.
     + Try to merge feature-A into main. Observe the merge conflict.
     + Manually resolve the conflict in README.txt.
     + Add the resolved file and commit the merge.

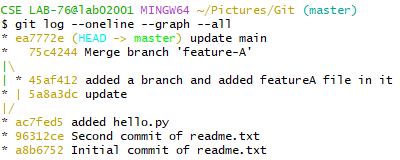


* 1. **Deleting Branches:**
     + Delete feature-A after it's merged (git branch -d feature-A).
     + Attempt to delete an unmerged branch and observe the warning/error. Use -D to force delete.

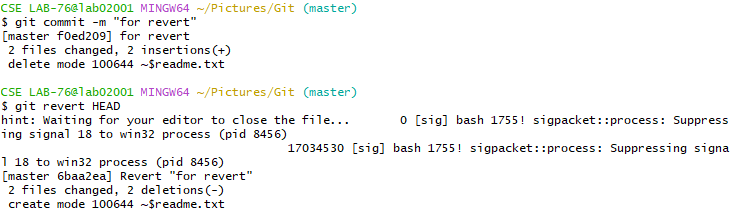


**Lab 3: Git History and Advanced Operations**

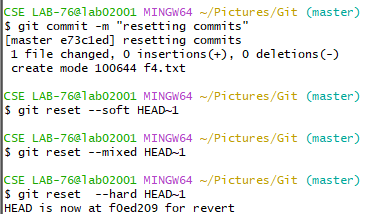
* **Objective:** To explore Git history, revert changes, and rebase branches.
* **Programs:**
  1. **Viewing History:**
     + Use git log --oneline --graph --all for a clear view of the branch history.
     + Explore git log --decorate, git log --stat.



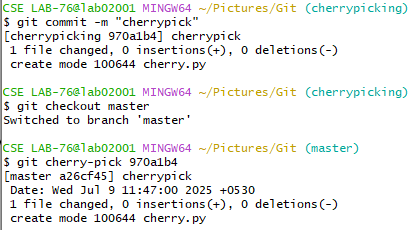
* 1. **Reverting Commits:**
     + Make a new commit with some changes.
     + Revert the last commit using git revert HEAD. Understand that revert creates a new commit to undo previous changes.



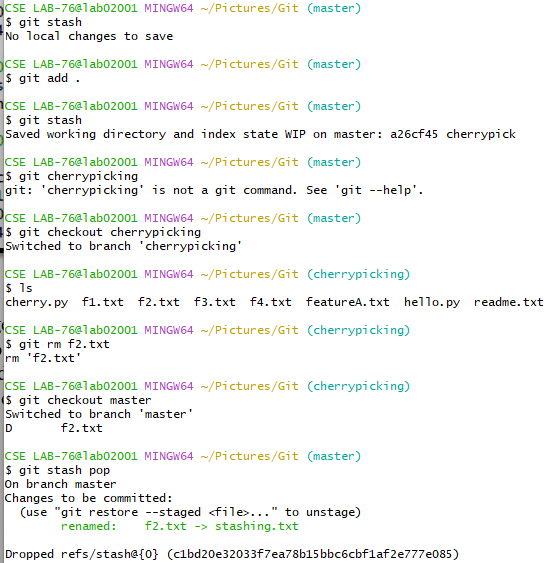
* 1. **Resetting Commits:**
     + Make a few commits.
     + Use git reset --soft HEAD~1 to uncommit but keep changes staged.
     + Use git reset --mixed HEAD~1 to uncommit and unstage changes.
     + Use git reset --hard HEAD~1 to uncommit, unstage, and discard changes (use with caution!).



* 1. **Cherry-picking:**
     + Create a new branch, make a commit with a specific small feature.
     + Switch to main and use git cherry-pick <commit-hash> to apply that specific commit to main.



* 1. **Stashing Changes:**
     + Make some changes but don't commit them.
     + Use git stash to temporarily save uncommitted changes.
     + Switch branches, do some work.
     + Come back to the original branch and use git stash pop to reapply the stashed changes.



**Module 2: GitHub Fundamentals (Remote Collaboration)**

**Lab 4: Connecting Local Git to GitHub**

* **Objective:** To learn how to create a GitHub repository, link it to a local Git repository, and push/pull changes.
* **Programs:**
  1. **Creating a GitHub Repository:**
     + Create a new public repository on GitHub (e.g., my-github-project).
     + Do *not* initialize it with a README, .gitignore, or license.
  2. **Connecting Local to Remote:**
     + From your existing local Git repository (my\_first\_repo from Lab 1), add the GitHub remote (git remote add origin <GitHub\_Repo\_URL>).
     + Push your local main branch to GitHub (git push -u origin main).
     + Verify the files appear on GitHub.
  3. **Cloning an Existing Repository:**
     + Create a new directory on your local machine.
     + Clone a repository from GitHub (e.g., a public repository, or your own my-github-project) using git clone <GitHub\_Repo\_URL>.
     + Explore the cloned repository.

**Lab 5: Collaborative Workflow with GitHub**

* **Objective:** To practice common collaborative workflows involving multiple contributors, pull requests, and code review.
* **Programs:**
  1. **Simulating Teamwork (Two Users/Clones):**
     + **User 1 (Student A):** Modify a file in the cloned repository, commit, and push to main.
     + **User 2 (Student B - or a second clone of the same repo):** Clone the *same* repository. Modify a *different* file, commit, and push to main.
     + **User 1:** Pull the changes from User 2 (git pull origin main). Observe how changes are integrated.
  2. **Feature Branch Workflow with Pull Requests:**
     + **User 1:** Create a new branch (e.g., feature-login-page).
     + Make changes related to a "login page" feature, commit them.
     + Push the feature-login-page branch to GitHub.
     + **On GitHub:** Create a Pull Request (PR) from feature-login-page to main.
     + **User 2 (Reviewer):** Review the PR, add comments/suggestions.
     + **User 1:** Address review comments by making new commits on feature-login-page and pushing them.
     + **User 2:** Approve the PR.
     + **User 1 or 2:** Merge the PR on GitHub.
     + **All Users:** Pull the latest changes to their local main branch.
  3. **Handling Merge Conflicts on GitHub Pull Requests:**
     + **User 1:** Create branch-X, modify file.txt.
     + **User 2:** Create branch-Y, modify the *same lines* in file.txt differently.
     + Both push their branches and create PRs to main.
     + Observe the merge conflict reported on GitHub.
     + **User 1:** Attempt to resolve the conflict directly on GitHub (if possible and simple), or pull main into branch-X locally, resolve, and push.
     + Merge the resolved PR.

**Lab 6: GitHub Features and Best Practices**

* **Objective:** To explore advanced GitHub features like issues, project boards, and using .gitignore.
* **Programs:**
  1. **GitHub Issues and Project Boards:**
     + On your GitHub repository, create a few issues (e.g., "Implement user registration", "Fix database connection bug").
     + Create a Project Board and link issues to it. Move issues through "To Do", "In Progress", "Done" columns.
     + Practice linking commits to issues using keywords (e.g., git commit -m "Fix #123: Resolved database issue").
  2. **Using .gitignore:**
     + Create a new project.
     + Add some temporary files or build artifacts (e.g., .log files, \_\_pycache\_\_ directories for Python, target/ for Java Maven projects, node\_modules for Node.js).
     + Observe them showing up in git status.
     + Create a .gitignore file and add patterns to ignore these files/directories.
     + Verify that they are no longer tracked by Git.
     + Commit .gitignore.
  3. **Contributing to Open Source (Conceptual / Optional Practical):**
     + Discuss the process of forking a repository, cloning your fork, making changes, pushing to your fork, and creating a pull request to the upstream repository.
     + (Optional Practical) Find a small open-source project on GitHub and walk through the steps to contribute a minor fix or feature.

**Module 3: Advanced Git & Project-Based Scenarios**

**Lab 7: Collaborative Project Simulation**

* **Objective:** To simulate a more realistic software development scenario involving multiple features, bug fixes, and continuous integration practices.
* **Scenario:** Develop a simple command-line application (e.g., a "To-Do List Manager", "Simple Calculator", or a "Contact Book") collaboratively.
* **Programs:**
  1. **Project Setup:**
     + One student creates the initial project repository on GitHub.
     + Other students clone the repository.
  2. **Feature Development (Parallel):**
     + Each student (or pair) selects a distinct feature or module to implement (e.g., "Add task", "View tasks", "Delete task", "Mark task as complete").
     + Each feature is developed on a separate branch.
     + Regularly pull main into feature branches to stay updated.
  3. **Code Review and Merging:**
     + Students submit Pull Requests for their completed features.
     + Peers review the code, suggest improvements.
     + Resolve conflicts if any arise during merging feature branches to main.
  4. **Bug Fixing:**
     + Introduce a known bug into the main branch.
     + Assign a student to create a bugfix-XYZ branch, fix the bug, and submit a PR.
     + Review and merge the bug fix.

**Lab 8: Git Hooks and Automation (Optional/Advanced)**

* **Objective:** To understand how Git hooks can automate tasks during development.
* **Programs:**
  1. **Pre-commit Hook:**
     + Write a simple pre-commit hook (shell script) that checks for common issues (e.g., ensures no debug statements are present, checks for minimum line length in commit messages).
     + Test the hook by trying to commit code that violates the rule.
  2. **Post-receive Hook (Conceptual):**
     + Discuss how post-receive hooks can be used on remote servers (like GitHub Enterprise or self-hosted Git servers) to trigger automated builds, tests, or deployments upon receiving a push. (Practical implementation might require server access, so this can be theoretical.)

**Lab 9: Git Rebase and Interactive Rebase (Advanced)**

* **Objective:** To understand git rebase for cleaning up commit history and integrating changes.
* **Programs:**
  1. **Simple Rebase:**
     + On main, make a commit.
     + Create feature-branch, make two commits.
     + On main, make another commit.
     + Switch to feature-branch and rebase it onto main (git rebase main). Observe the commit history.
  2. **Interactive Rebase (git rebase -i):**
     + Make several small, unorganized commits on a feature branch.
     + Use git rebase -i HEAD~N (where N is the number of commits) to:
       - Squash multiple commits into one.
       - Reorder commits.
       - Edit a commit message.
       - Delete a commit.
     + Discuss when rebase is appropriate (before pushing shared branches) versus merge.