

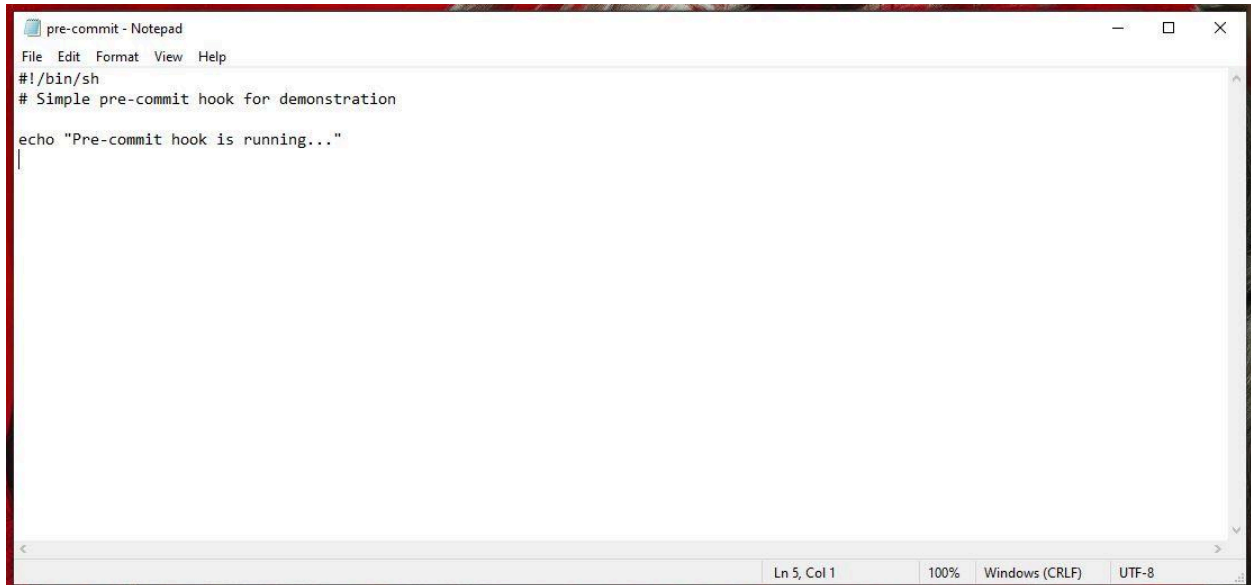
```
Command Prompt

C:\Users\joeva\Desktop\Test>cd .git/hooks
C:\Users\joeva\Desktop\Test\.git\hooks>echo.> pre-commit
C:\Users\joeva\Desktop\Test\.git\hooks>notepad pre-commit
C:\Users\joeva\Desktop\Test\.git\hooks>git add newfile.txt
fatal: this operation must be run in a work tree
C:\Users\joeva\Desktop\Test\.git\hooks>cd ..
C:\Users\joeva\Desktop\Test\.git>cd ..
C:\Users\joeva\Desktop\Test>git add newfile.txt
C:\Users\joeva\Desktop\Test>git commit -m "Test pre-commit hook"
Pre-commit hook is running...
[detached HEAD 625d2dc] Test pre-commit hook
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 newfile.txt
C:\Users\joeva\Desktop\Test>
```

```
MINGW64:/c/Users/joeva/Desktop/Test/.git/hooks

joeva@DESKTOP-81IFNOB MINGW64 ~/Desktop/Test/.git/hooks (GIT_DIR!)
$ chmod +x pre-commit

joeva@DESKTOP-81IFNOB MINGW64 ~/Desktop/Test/.git/hooks (GIT_DIR!)
$ |
```



```
pre-commit - Notepad
File Edit Format View Help
#!/bin/sh
# Simple pre-commit hook for demonstration

echo "Pre-commit hook is running..."
```

1. Pre-Commit Hook

- **Location:** `.git/hooks/pre-commit`
- **Trigger:** Before a commit is created.
- **Use Cases:**
 - **Code Formatting:** Ensure code adheres to style guidelines (e.g., using linters like `eslint` or `prettier`).
 - **Validation:** Run unit tests or checks to prevent committing broken code.
 - **Preventing Large Files:** Block commits of large files or sensitive data.

2. Commit-Msg Hook

- **Location:** `.git/hooks/commit-msg`
- **Trigger:** After the commit message is created but before the commit is completed.
- **Use Cases:**
 - **Commit Message Validation:** Enforce commit message formats or conventions (e.g., including issue numbers or adhering to a specific structure).
 - **Automated Checks:** Ensure that commit messages meet project guidelines.

3. Post-Commit Hook

- **Location:** `.git/hooks/post-commit`
- **Trigger:** After a commit is created.
- **Use Cases:**
 - **Notifications:** Send notifications or messages (e.g., Slack notifications) about new commits.

- **Analytics:** Log commit data or update external systems with commit information.

4. Pre-Push Hook

- **Location:** `.git/hooks/pre-push`
- **Trigger:** Before changes are pushed to a remote repository.
- **Use Cases:**
 - **Test Runs:** Run tests to ensure code quality before pushing changes.
 - **Linting:** Perform code linting to maintain code standards.
 - **Security Checks:** Ensure no sensitive data or secrets are pushed.

5. Post-Push Hook

- **Location:** `.git/hooks/post-push`
- **Trigger:** After changes are pushed to a remote repository.
- **Use Cases:**
 - **Deployment:** Trigger deployments or updates to staging/production environments.
 - **Syncing:** Synchronize additional resources or external services with the latest changes.

6. Pre-Rebase Hook

- **Location:** `.git/hooks/pre-rebase`
- **Trigger:** Before a rebase operation.
- **Use Cases:**
 - **Check Dependencies:** Ensure that rebasing will not break dependencies or functionality.
 - **Code Reviews:** Prompt for a code review before rebasing.

7. Post-Rewrite Hook

- **Location:** `.git/hooks/post-rewrite`
- **Trigger:** After a rewrite operation (e.g., `git commit --amend` or `git rebase`).
- **Use Cases:**
 - **Notification:** Notify team members about changes in commit history.
 - **Update Tools:** Update external tools or systems affected by history changes.

8. Pre-Receive Hook

- **Location:** Server-side hook (`hooks/pre-receive`)
- **Trigger:** Before changes are accepted by the remote repository.
- **Use Cases:**

- **Access Control:** Implement access control or authentication mechanisms.
- **Policy Enforcement:** Enforce repository policies (e.g., branch protection, commit message rules).

9. Update Hook

- **Location:** Server-side hook (`hooks/update`)
- **Trigger:** Before the reference is updated in the remote repository.
- **Use Cases:**
 - **Branch Protection:** Prevent updates to certain branches or enforce branch naming conventions.
 - **Integration Checks:** Ensure incoming changes meet certain integration criteria.

10. Post-Receive Hook

- **Location:** Server-side hook (`hooks/post-receive`)
- **Trigger:** After changes are accepted by the remote repository.
- **Use Cases:**
 - **Deployments:** Trigger automated deployments or builds after code is pushed.
 - **Notifications:** Send notifications or updates about the changes.

Command Line Experience

When I first started using Git, I dove straight into the command line. It was like learning a new language—at first, it felt overwhelming with all the commands and options. I remember being daunted by the sheer number of Git commands and their syntax. To stage changes, I'd use `git add .`, and to commit, it was `git commit -m "message"`. For me, the command line was a powerful tool, offering full control over every aspect of my version control.

However, the learning curve was steep. Complex operations like merging branches or resolving conflicts often left me feeling lost. I had to remember the exact commands and their parameters, and dealing with errors required me to dig into the Git documentation or search for solutions online. The command line is incredibly efficient once you're familiar with it, but it demands a lot of memorization and a deeper understanding of Git's internals.

Graphical Interface Experience

Switching to a graphical Git tool like Sourcetree was like stepping into a whole new world. The visual representation of branches, commits, and changes was a game-changer. With

Sourcetree, I could see a clear history of my commits in a graph, which made understanding my project's state so much easier. Staging and committing changes became more intuitive—I just selected files, wrote my commit message, and clicked a button.

GitKraken offered a similar experience with its sleek interface and drag-and-drop functionality for merging branches. It was a huge relief not to worry about remembering commands. The graphical tools made complex operations like rebases and merges much more manageable. I could see all my branches laid out and visually handle conflicts, which felt much less stressful compared to the command line.

The graphical interfaces also made collaboration smoother. Seeing the status of my remote branches and incoming changes in real-time was really helpful. For tasks like reviewing commit history or exploring different branches, the visual tools provided a much clearer view.