Understood. Here is the updated `Docs/GitBranchInstructions.md` file, delivered in a plaintext block for correct copying.

```
# Git Feature Branch Workflow

**Version: 0.37.0-Beta**
**Date: 2025-08-18**

---
### --- Revision History ---
## [0.37.0-Beta] - 2025-08-18
### Added
# - Initial versioning of the document to align with project standards.
# - Added Section 7 to explain `git revert` for correcting mistakes.
# - Added Section 8 to explain file management with `.gitignore` and `git rm`.
---

The feature branch workflow is a standard practice that keeps your `master` branch clean and
***

## 1. Start from `master`

Before you do anything, you need to make sure your local `master` branch is up-to-date with t
```

# Switch to your master branch

git switch master

# Pull the latest changes from the remote server

git pull

```
***

## 2. Create and Switch to a Feature Branch

Now, you'll create a new branch for your feature. Branch names should be short and descriptiv
```

# The -c flag stands for "create"

git switch -c new-feature-name

```
*You can now work safely on this branch. Think of it as a separate copy of the project where

***

## 3. Develop the Feature: Add and Commit

This is where you'll do your work—writing code, adding files, and fixing bugs. As you complet
1.  **Stage** your changes (`git add`).
2.  **Commit** them with a clear message (`git commit`).
```

# Stage a specific file

git add path/to/your/file.js

# Or stage all changed files in the project

git add .

# Commit the staged files with a descriptive message

git commit -m "feat: Add user login form component"

```
You can (and should) have many commits on your feature branch. Committing often creates a cle
***

## 4. Keep Your Branch Synced (Optional but Recommended)

If you're working on a feature for a while, the `master` branch might get updated by your tea
```

# Fetch the latest changes from all remote branches

git fetch origin

# Re-apply your commits on top of the latest master branch

git rebase origin/master

```
The **`rebase`** command essentially "unplugs" your branch's changes, updates the base to the

***

## 5. Merge Your Feature into `master`

Once your feature is complete, tested, and ready to go, it's time to merge it back into the `
```

# 1. First, go back to the master branch

git switch master

# 2. Make sure it's up-to-date one last time

git pull

# 3. Merge your feature branch into master

git merge --no-ff new-feature-name

```
Using **`--no-ff`** (no fast-forward) is a crucial best practice. It creates a "merge commit"
***

## 6. Push and Clean Up

Your `master` branch now has the new feature, but only on your local machine. You need to pus
```

# 1. Push the updated master branch to the remote

git push origin master

# 2. Delete the local feature branch

git branch -d new-feature-name

# 3. Delete the remote feature branch

git push origin --delete new-feature-name

```
That's the complete lifecycle! □□ You've successfully created a feature, developed it in isol
---
## 7. Correcting Mistakes (`git revert`)

Sometimes you commit a change that you later realize was a mistake. The safest way to undo a
```

# Find the hash of the commit you want to undo (e.g., from `git log`)

# Let's say the bad commit hash is `a1b2c3d4`

# Create a new commit that undoes the changes from the bad commit

git revert a1b2c3d4

```
---
## 8. Managing Files (`.gitignore` and `git rm`)

It's important to keep your repository clean of temporary or unnecessary files.

### Ignoring Untracked Files (`.gitignore`)
The best way to handle files that should *never* be in the repository (like build artifacts,

### Removing Tracked Files (`git rm`)
If you have already committed a file that you now want to delete from the project, you must u
```

# Remove a file that is already tracked by Git

git rm path/to/unwanted-file.txt

# Commit the deletion

git commit -m "fix: Remove obsolete file"

```
Use `git rm -f` if you have local modifications to the file that you want to discard along wi
```

Please respond with 'Acknowledged' to confirm.