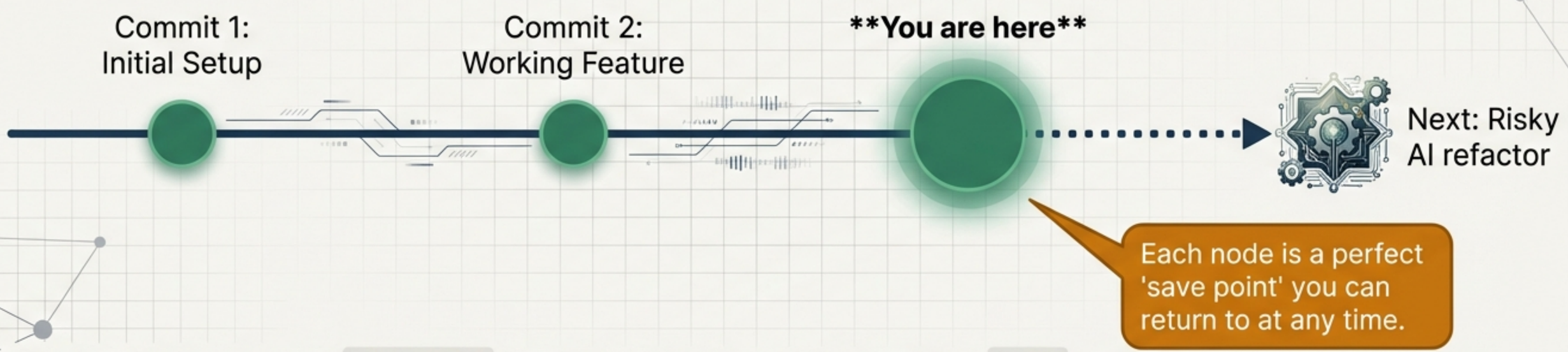
The Power and Peril of AI Code

AI assistants provide incredible speed, but introduce risk. Will the generated code work? Will it break your project? Fear of breaking things stifles the very experimentation AI is meant to enable.

We'll embrace a new mindset: Git is not about memorizing commands. It's about creating a **safety net** for fearless development. This is your system for controlling chaos and turning AI suggestions into reliable code.

Git is Your Project's Save System

Think of a **commit** as a **perfect, restorable snapshot** of your **project**. It's like saving your game **before a boss fight**. This is the foundation of fearless experimentation. Before you ask an AI to refactor your code, you create a save point. If the AI fails, you simply reload.



Everything starts with `git init`. This command creates a hidden `.git` directory in your project folder, which acts as the repository's database. It's where your entire project history and all your save points will live.

The Three States of Being: Working, Staged, Committed

Creating a save point is a deliberate, two-step process that gives you precise control. You don't just save everything; you choose exactly what goes into each snapshot.

1. Working Directory

You modify files. `git status` shows them as "modified" (red).

2. Staging Area

You choose which changes are ready. This is your checklist. `git status` shows them as "Changes to be committed" (green).

3. Repository

You create the snapshot. This saves everything in the staging area as a permanent record.

Rewind Time, No Consequences

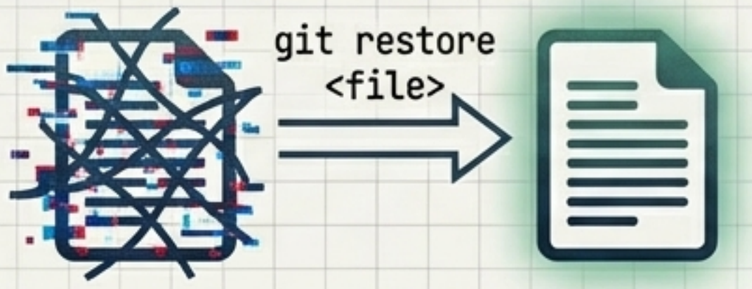
In Git, mistakes are inevitable, but they are also temporary. You have a non-destructive undo button for every stage of your work before a commit.

See Your Changes

```
+ const newFeature = 'enabled';
+ console.log(newFeature);

- const oldFeature = 'deprecated';
- // console.log(oldFeature);
```

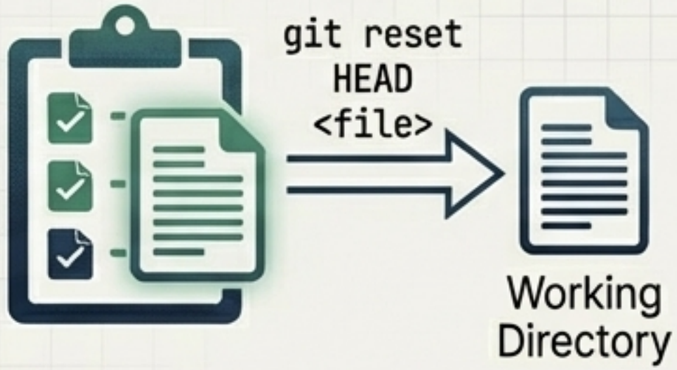
Use `git diff` to inspect what you've modified since your last commit. It shows additions (`+`) and deletions (`-`) line-by-line. This is your inspection checkpoint.

Discard Unstaged Changes



An AI suggestion broke a file? `git restore <file>` instantly reverts it to the last committed version.

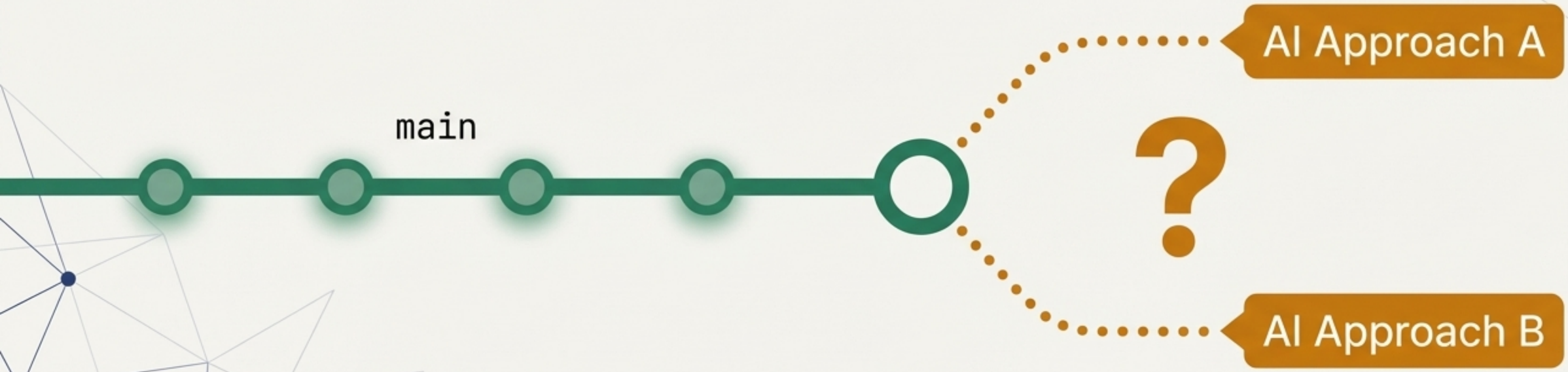
Unstage an Accident



Staged the wrong file? `git reset HEAD <file>` moves it out of the staging area without deleting your changes.

The Innovator's Dilemma: How Do You Test Two AI Suggestions at Once?

Your project is in a stable state on the `main` branch. An AI suggests two completely different, risky ways to implement a new feature. Committing one, then undoing it to try the other, is slow and messy.

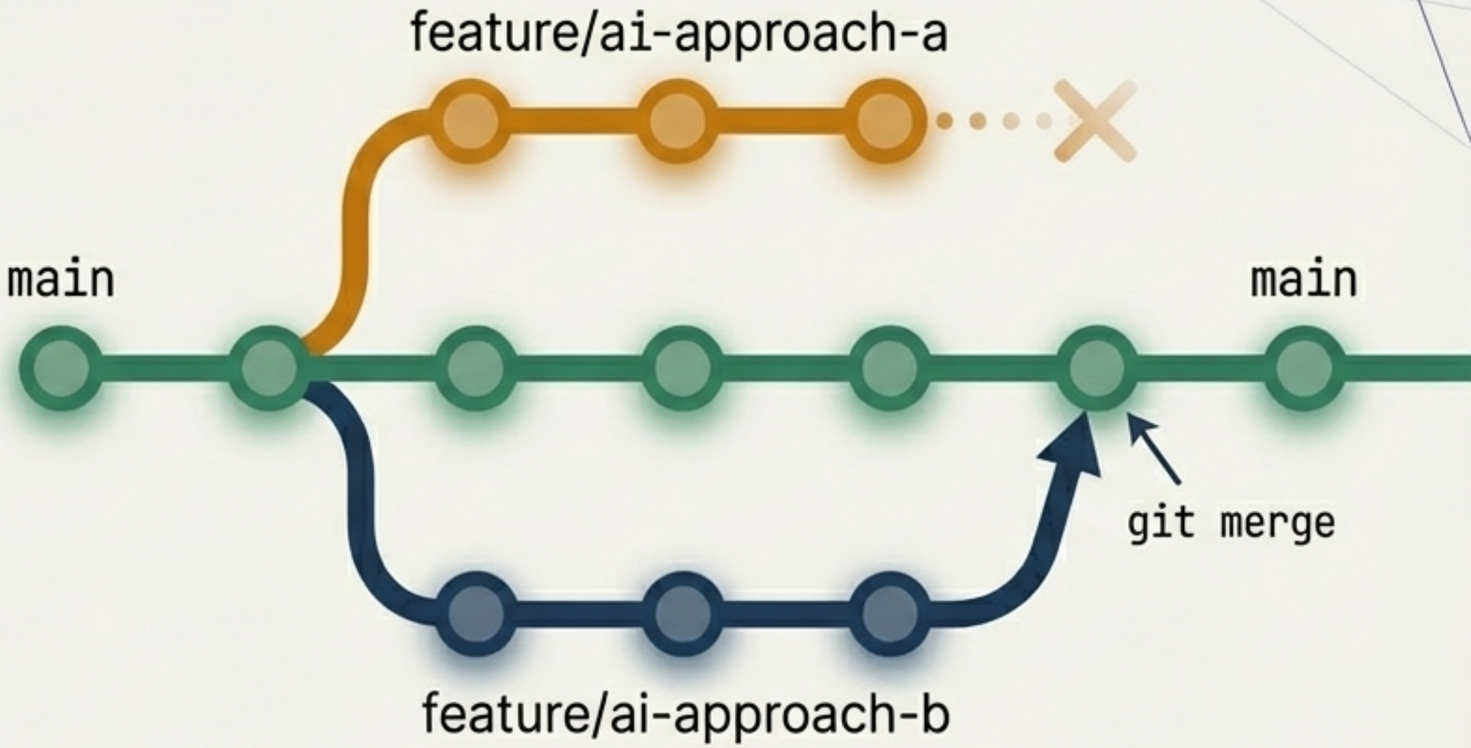


This is a common scenario in AI-driven development. You need a way to explore both paths in parallel, without either one affecting your stable `main` codebase until you've chosen a winner.

Branches: Disposable, Parallel Universes for Your Code

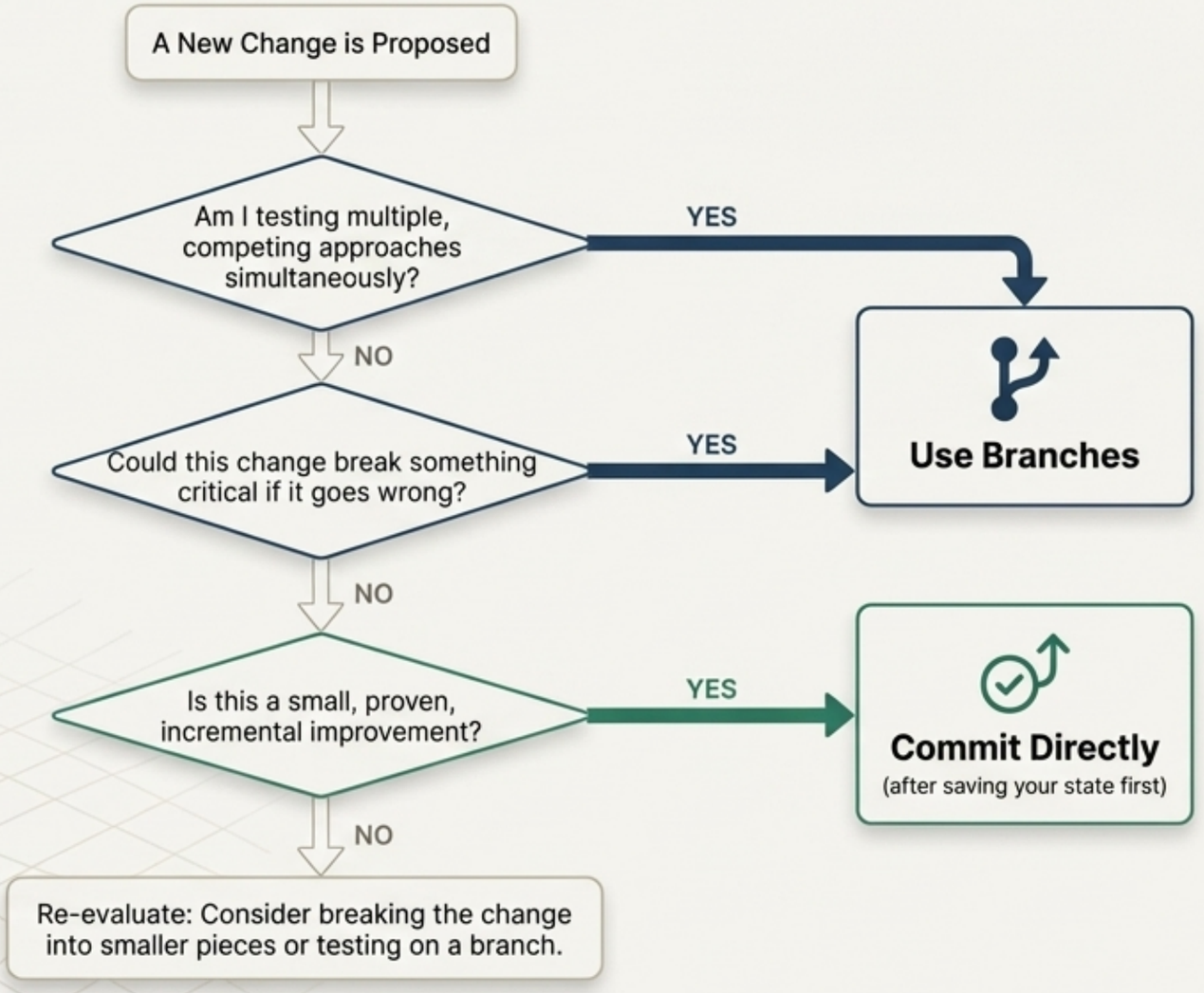
A branch is an independent timeline. You can create a branch to test an idea, and anything you do there—add, commit, even break everything—has zero impact on your `main` branch.

1. **Create & Switch**
   `git switch -c feature/ai-approach-a` creates a new branch and moves you to it.
2. **Experiment**
   Let the AI generate code. Commit your changes on this branch.
3. **Decide**
   Test the feature. If it works, you merge it. If it fails, you simply switch back to `main` and delete the branch with `git branch -d feature/ai-approach-a`. Your main code remains pristine.



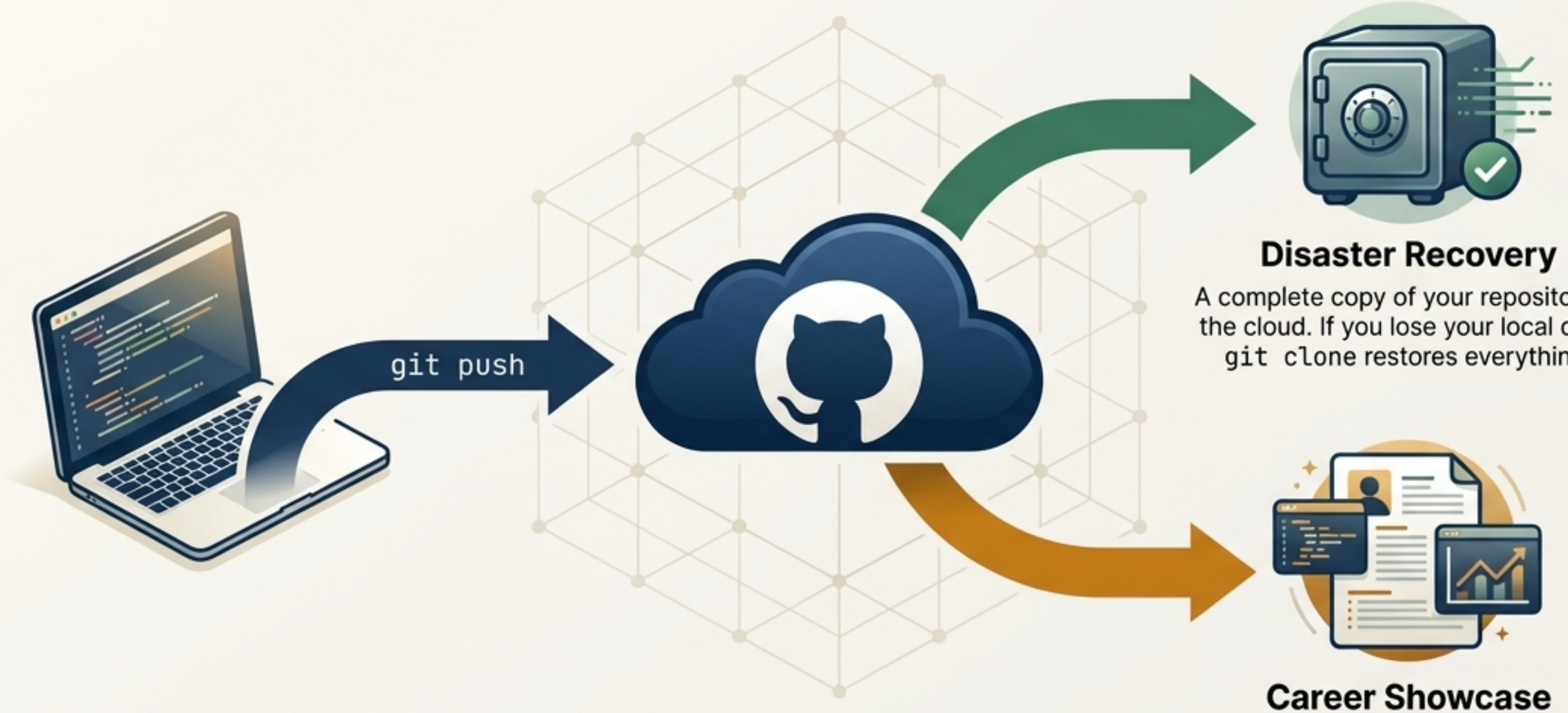
Your Strategic Choice: Iterate or Isolate?

Use this framework before letting an AI make a change. Your decision will determine your workflow.



GitHub: Your Cloud Backup and Professional Portfolio

What if your computer crashes? All your local Git history is gone. Pushing your code to a **remote repository on GitHub serves two critical functions at once.**



Disaster Recovery

A complete copy of your repository in the cloud. If you lose your local data, `git clone` restores everything.

Career Showcase

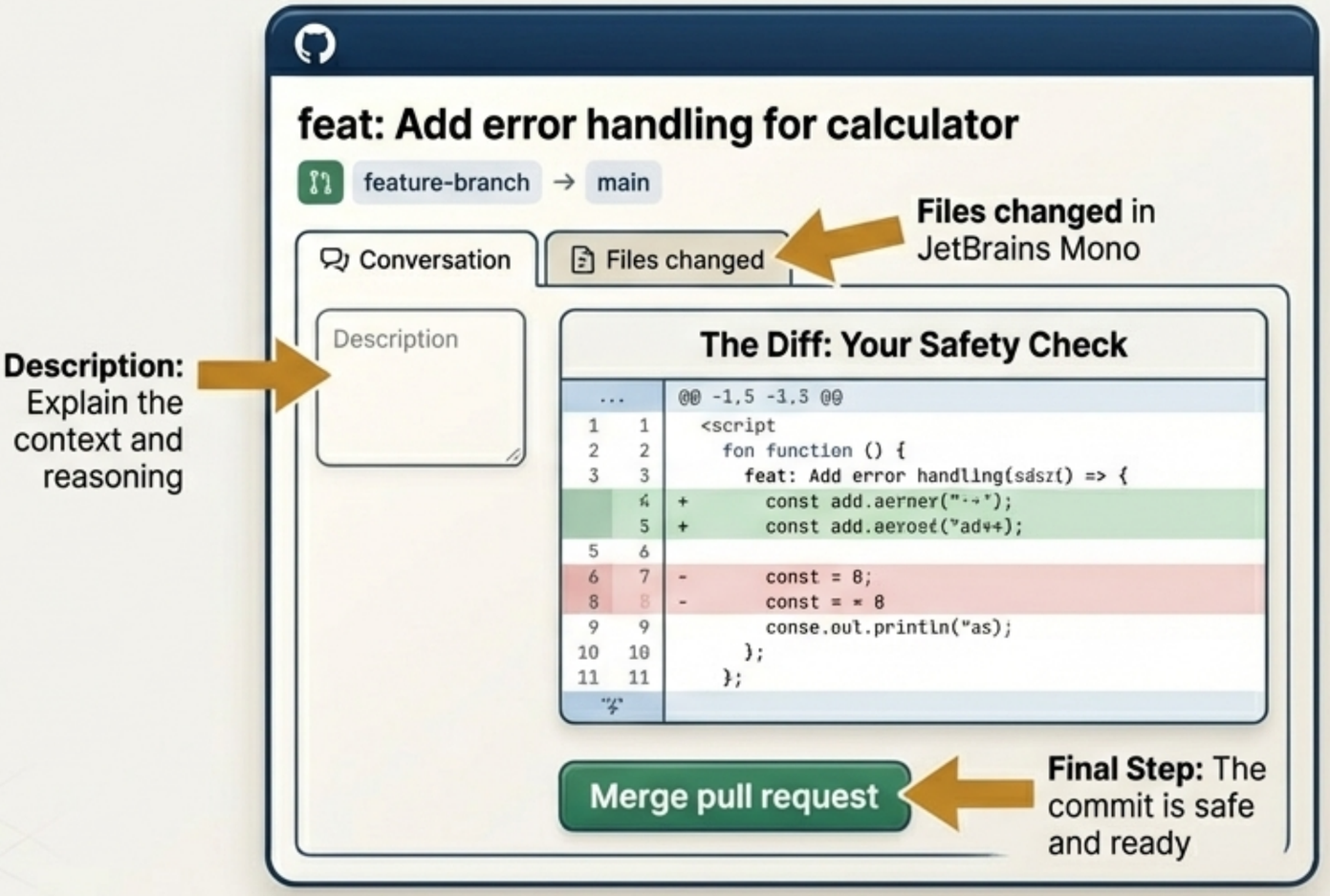
Your GitHub profile is the modern developer's resume, providing tangible proof of what you can build.

The Professional Workflow: Review Before You Merge

On professional teams (and for your own quality control), you never merge branches directly into `main`. You open a **Pull Request (PR)**. A PR is a formal proposal to merge changes, creating an opportunity for review.

The PR Workflow

1. Push your feature branch to GitHub.
2. On GitHub, open a PR to merge your `feature-branch` into `main`.
3. Review the "diff"—a visual comparison of all changes.
4. Discuss, approve, and then merge.



The New Ethic: Documenting Your AI Collaborator

In AI-native development, acknowledging your AI partner is a sign of professional integrity and builds trust. It tells reviewers which code might need extra scrutiny and properly attributes the work.

Description: feat: Add error handling for calculator

```
### AI Assistance

- **AI Tool Used**: Claude Code
- **What AI Generated**: "Initial implementation of the user validation logic."
- **My Modifications**: "I refactored the validation function, added error handling for empty inputs which the AI missed, and wrote all unit tests."
```

Transparency builds trust.

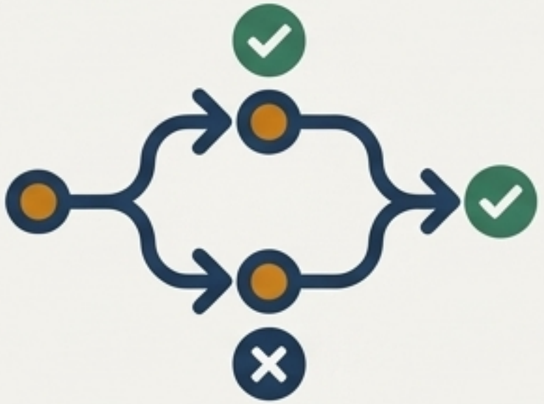
This section gives reviewers crucial context for their evaluation.

From Actions to Intelligence: Recognizing the Patterns

The commands you've learned are not just a list to memorize. You've been unconsciously using recurring, intelligent workflows. Making these patterns explicit is the key to mastery.

Commit-Before-Experiment

The "save your game" pattern. Always commit your working state before letting an AI make a risky change.



Branch-Test-Merge

The "parallel universe" pattern. Test competing or risky AI suggestions in isolation before integrating the winner.

Push-for-Backup

The "disaster recovery" pattern. Regularly back up your local commits to the cloud to prevent catastrophic data loss.

Codify Your Knowledge into a `git-workflow.md`

The difference between learning and true intelligence is making your knowledge explicit and reusable. Create a personal guide in your project root called `git-workflow.md`.

This document isn't for others; it's for your future self. It should capture your principles, your decision frameworks, and the questions you ask yourself to ensure you're following best practices.
This document becomes your personal “safety engineer,” ensuring apply these patterns automatically on every future project.

git-workflow.md

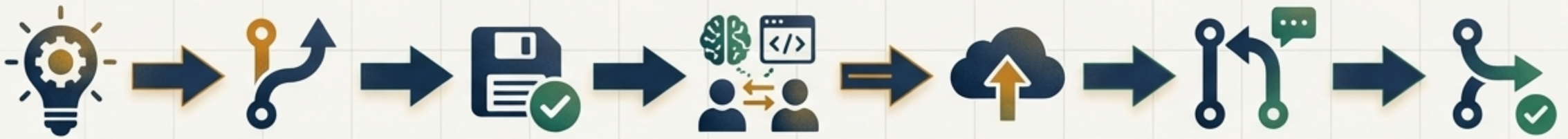
```
### My Principles
  • **Commit Intentionally**: Every commit is a conscious
    decision.
  • **Parallel Safety**: Never modify `main` directly when
    uncertain.
  • **Backup Discipline**: The remote is not optional; it's
    my safety net.

### Questions to Ask Myself Before a Change
  • If this experiment fails, will I want to return to this
    exact state?
  • Could this break something critical?
  • Is my commit message clear enough for me in six months?
```

The Full Loop: Your AI-Dev Superpower

This is how all the pieces fit together for safe, rapid, and professional AI-assisted development. From a local idea to production-ready code, you now have a **repeatable and robust workflow.**

Local Actions

Remote Actions (GitHub)



Local Idea

1. Create a branch
(`git switch -c ...`)

2. Save Point
(`git commit`)

3. AI Collaboration & Local Commits

4. Cloud Backup
(`git push`)

5. Formal Review
(Open a Pull Request with AI Transparency)

6. Integration
(Merge PR into `main`)

7. Cleanup
(Delete feature branch)

Welcome to Fearless Development

You now possess both the mindset and the toolset to leverage AI to its full potential—safely, professionally, and without fear. You have a system to manage the chaos and a workflow to build robust software.

Start your next project. Apply your documented workflow. Build your portfolio. The safety net is in place.

Your Core Command Reference

A quick reference for the essential commands and their purpose in your workflow.

The Goal	The Command
Initialize a new repository	`git init`
Check the project status	`git status`
Stage a file for commit	`git add <file>`
Create a save point	`git commit -m "Message"`
See your changes	`git diff`
Discard changes in a file	`git restore <file>`
Create and switch to a new branch	`git switch -c <branch-name>`
Merge a branch into your current one	`git merge <branch-name>`
Push your commits to GitHub	`git push origin main`
Download a repository from GitHub	`git clone <url>`