Commits: The Save Button That Remembers Why

What a commit actually is, and how to write messages future-you can read

You've written some code, edited a config, or fixed a template. It works. Now what? In GitHub's world, you commit.

A commit is a save with a memory. Not just "here is the current state of the files," but "here is what changed, and here is why." Every commit is a labeled snapshot you can return to. String them together and you get the full history of a project: who changed what, when, and for what reason.

That history is the entire point of using GitHub. Lose the discipline of good commits and you have an expensive backup drive. Keep it and you have a time machine.

A commit is two moves, not one

Git makes you do this in two steps, which trips up everyone at first:

1. **Stage** the changes you want to include (`git add`).
2. **Commit** the staged changes with a message (`git commit`).

```
git add intake-form.html          # stage one file
git add .                          # or stage everything you changed
git commit -m "Add email field to intake form"
```

The staging step feels like busywork until you see why it exists: it lets you commit some changes now and hold others back. You fixed a bug and also left a half-finished note to yourself; stage and commit the fix, leave the note for later. One commit, one idea.

Write the message for the person who reads it in six months

That person is you. You will not remember why you changed that file. The message is where you tell yourself.

Bad messages, all real, all useless:

```
update
stuff
fixed it
asdf
```

Good messages say what changed and, when it matters, why:

```
Add conflict check to intake before matter creation
Fix date parsing that broke on single-digit days
Remove client name from example config (privilege)
```

The working rule: finish the sentence "If applied, this commit will ___." "If applied, this commit will *add conflict check to intake*." That is why commit messages read as commands, not past tense.

Commit small, commit often

A commit should be one coherent change. Not a week of work in one giant blob; not every keystroke either. One fix, one feature, one cleanup. Small commits are easier to understand, easier to review, and far easier to undo when one of them turns out to be the mistake. (Undoing a bad commit is its own tip; see Git Rollback.)

No terminal? Same thing, with buttons

GitHub Desktop does all of this without the command line. You check boxes next to the files you want (that is staging), type a summary in the message box, and click "Commit." Then "Push" to send it up. Same two moves, same discipline, no memorizing commands.

The playbook

```
git add <files>                              # stage what belongs together
git commit -m "Clear, present-tense summary" # save with a reason
git log --oneline                            # read your history back
```

Small commits, honest messages. That habit is what makes the whole history worth having.