Clone, Pull, Push: The Everyday GitHub Loop

Get a repo onto your machine, sync changes down, send yours back up

A repo lives in two places: on GitHub's servers, and on your machine. Almost everything you do in Git is moving changes between those two copies. Three commands cover ninety percent of it: clone, pull, push.

Get these three and the daily rhythm of GitHub stops being mysterious.

Clone: get the repo onto your machine

Cloning makes a full local copy of a repo, complete history and all. You do it once per repo, at the start.

```
git clone https://github.com/yourname/your-repo.git
cd your-repo
```

You now have the entire project in a folder, plus a hidden `.git` directory that tracks its history. Clone is download, but smarter: it remembers where it came from, so pull and push know where to sync.

Pull: bring down changes made elsewhere

If you work from two machines, or anyone else touches the repo, GitHub's copy can move ahead of yours. `git pull` brings those changes down and merges them into your local copy.

```
git pull
```

Make a habit of pulling *before* you start working. It is the difference between building on the current version and building on a stale one, then having to untangle the two. Pull first, every session.

Push: send your commits up

You've committed your changes locally (see the Commits tip). Those commits still live only on your machine until you push them.

```
git push
```

Now GitHub has them, they are backed up off your laptop, and anyone else on the repo can pull them down. A commit you never push is a commit only your hard drive knows about; if the laptop dies, so does the work.

The loop, start to finish

Here is a normal working session, top to bottom:

```
git pull                            # 1. get the latest
# ...edit files, do your work...
git add .                           # 2. stage what changed
git commit -m "Describe the change" # 3. save it locally
git push                            # 4. send it up
```

Pull, work, commit, push. That is the everyday loop. You will run it hundreds of times.

The one error you will hit

Sooner or later `git push` gets rejected because GitHub has changes you do not: someone else pushed, or you pushed from another machine. The fix is almost always: pull, let Git merge, then push again.

```
git pull      # brings their changes in, may ask you to resolve a conflict
git push      # now it goes through
```

Merge conflicts sound scary and are not; they get their own tip. For now, know that "pull, then push" clears most rejected pushes.

No terminal? The buttons map one to one

In GitHub Desktop: "Clone repository" to start, "Fetch/Pull origin" to bring changes down, "Push origin" to send yours up. Same three moves, same order.

The playbook

```
git clone <url>    # once, to start
git pull           # before you work, and to sync
git push           # after you commit, to publish
```

Pull before, push after. Build that reflex and you will rarely fight the tool.