GitHub & Remotes

What a remote is, how git clone/push/fetch/pull work, the difference between fetch and pull (fetch: download without integrating, pull: fetch + integrate), tracking branches and git push -u, and synchronizing local and remote branches.

Intermediate 35 min

Last updated: August 27, 2026
Read online: https://www.learnforgex.com/en/topics/github-and-remotes

---

Everything so far has happened entirely on your machine. This lesson connects your local repository to GitHub — so your work is backed up, shareable, and synchronized with the rest of your team. The core idea to hold onto throughout: your local repository and the one on GitHub are two independent, complete copies of the same project, and Git commands exist specifically to move commits between them in a controlled way.

What Is a Remote?

A **remote** is simply a named reference to another copy of the repository — usually one hosted on GitHub. It's not a live connection or a mount; it's just a URL Git remembers, plus a local record of what that other copy's branches looked like the last time you checked.

Local Repository vs Remote Repository

Your local repository and a remote repository are both *complete* — the remote isn't a "partial" or "server-only" version. Each side has its own full commit history. When you interact with GitHub, you're always explicitly transferring commits in one direction or the other (`push` sends yours there, `pull` / `fetch` bring theirs here) — nothing syncs automatically in the background.

git remote

```
git remote            # list remote names (usually just "origin")
git remote -v         # same, but show the actual URLs (fetch + push)
```

By convention, the main remote you push to and pull from is named `origin` — this is just a name, not a Git keyword, but nearly every project uses it.

Adding origin

If you created a repository locally first (`git init`) and want to connect it to a new, empty GitHub repository:

```
git remote add origin https://github.com/your-username/task-tracker.git
git push -u origin main
```

`git remote add` registers the URL under the name `origin`; the `-u` flag on the following push is covered in detail a few sections down.

git clone

If the repository already exists on GitHub and you want a local copy, `git clone` does it in one step — it downloads the entire history *and* automatically sets up `origin` pointing back at it:

```
git clone https://github.com/your-username/task-tracker.git
```

`clone` is how you'll start working on almost every existing project — you'll rarely `git init` a brand-new empty repo once you're working with an established team.

git push

`git push` sends your local commits to the remote:

```
git push origin main
```

This uploads any commits `main` has locally that `origin`'s copy of `main` doesn't have yet. If the remote has commits your local branch doesn't have, Git will refuse the push (this is a safety feature — we cover the deliberate override, `--force`, later in this course).

git fetch

`git fetch` downloads any new commits from the remote **without touching your working tree or your local branches**:

```
git fetch origin
```

This is the first half of the two-step process that keeps your local repository aware of what's happened on GitHub. After a fetch, your local `main` hasn't moved at all — Git has just updated its *record* of where `origin/main` currently points.

git pull

`git pull` does the fetch, then immediately merges the fetched changes into your current branch:

```
git pull origin main
```

`git fetch` downloads remote changes without integrating them; `git pull` is fetch + integrate. This is the single most important distinction in this lesson. `pull` is convenient, but it means a merge (or, depending on your Git configuration, a rebase) happens immediately and automatically — which is exactly why some developers prefer to `fetch` first, look at what changed, and merge deliberately afterward.

Fetch vs Pull

- **Downloads remote commits** — both `git fetch` and `git pull` do.
- **Updates your working tree** — `git fetch`: no. `git pull`: yes.
- **Merges into your current branch** — `git fetch`: no. `git pull`: yes, automatically.
- **When to use** — `git fetch` when you want to look before integrating; `git pull` when you just want to be up to date, now.

What Happens If You Have Uncommitted Changes and Run git pull

Because `pull` ends with a merge, it's subject to the same safety rule as switching branches: **Git will refuse to pull if the merge would overwrite uncommitted changes it can't safely combine.** You'll see an error telling you to commit or stash your changes first. If your uncommitted changes don't touch any of the same lines the incoming commits touch, Git *may* be able to merge them automatically — but don't rely on this; committing (or stashing, covered later in this course) before pulling is the safe habit.

Tracking Branches

When you clone a repository, your local `main` is automatically set up to **track** `origin/main` — Git remembers this relationship, which is what lets you run plain `git push` / `git pull` without specifying `origin main` every time once the connection exists.

```
git branch -vv    # show local branches with what they're tracking
```

git push -u

When you push a *brand-new* local branch for the first time, you need `-u` (short for `--set-upstream`) to establish the tracking relationship:

```
git push -u origin add-password-reset
```

After this one-time setup, plain `git push` and `git pull` on that branch know exactly where to send/receive commits, without you specifying `origin add-password-reset` every time.

Deleting Remote Branches

Deleting a local branch (`git branch -d`) has no effect on the remote copy — they're independent. To delete a branch on GitHub itself:

```
git push origin --delete add-password-reset
```

Synchronizing Local and Remote Branches

Putting the full picture together, a typical day working with a remote looks like:

```
#!/bin/sh
# A typical day: check for remote changes, pull them in, push a new branch,
# and keep it up to date with main.

git fetch origin
# Output:
# From github.com:your-username/task-tracker
#    a1b2c3d..e5f6a7b  main       -> origin/main

# Two new commits exist on origin/main that we don't have locally yet.
git pull origin main
# Output:
# Updating a1b2c3d..e5f6a7b
# Fast-forward
#  TaskController.java | 9 +++++++++
#  1 file changed, 9 insertions(+)

# Start a new feature branch and push it for the first time.
git switch -c add-due-dates
# ... edit Task.java, TaskController.java ...
git add .
git commit -m "Add due date field to Task"
git push -u origin add-due-dates
# Output:
# * [new branch]      add-due-dates -> add-due-dates
# branch 'add-due-dates' set up to track 'origin/add-due-dates'.

# From now on, plain push/pull on this branch work without extra arguments.
git push
```

Common Mistakes

- **Assuming `git fetch` updates your files** — it only updates Git's *knowledge* of the remote; your working tree is untouched until you merge (or pull).
- **Running `git pull` with uncommitted changes and being surprised by a conflict or a refusal** — commit or stash first.
- **Forgetting `-u` on the first push of a new branch**, then being confused why plain `git push` afterward complains about no upstream branch.

Best Practices

- `git fetch` before you start work each day to see what's changed, before deciding how to integrate it.
- Always use `-u` the first time you push a new branch — it pays off on every push/pull after that.
- Delete a branch on GitHub as part of finishing a Pull Request (covered in the next lesson) — don't let stale remote branches accumulate.

Summary, Cheat Sheet, and Glossary

Summary

- A remote is a named reference (usually `origin`) to another copy of the repository, typically on GitHub.
- `git clone` downloads a repository and sets up `origin` automatically; `git remote add` connects an existing local repo to a new remote.
- `git fetch` downloads changes without integrating them; `git pull` = fetch + merge, and can be blocked by uncommitted local changes just like switching branches.
- `git push -u` establishes the tracking relationship the first time you push a new branch.

Cheat Sheet

```
git clone <url>                   # download a repo, set up origin automatically
git remote -v                     # show remote URLs
git fetch origin                  # download changes, don't integrate
git pull origin main              # download AND integrate changes
git push origin main              # upload your commits
git push -u origin <branch>       # push a new branch, set up tracking
git push origin --delete <branch> # delete a branch on the remote
```

Glossary

- **Remote** — a named reference to another copy of the repository (usually `origin`, on GitHub).
- **Fetch** — download remote commits without merging them into your branches.
- **Pull** — fetch, then immediately merge into your current branch.
- **Tracking branch** — a local branch linked to a specific remote branch, enabling plain `push` / `pull`.