

Publishing your lab

ar020 · 11 July 2026

A demolab repo compiles to plain files, and publishing is deliberately boring: build once, put the output on a static host. demolab ships with *no* active deploy workflow; going public is an explicit opt-in, one command plus a couple of clicks in the GitHub UI. This page walks the whole path.

What a build emits

`demolab` build

One build turns the repository into two committed-or-served outputs under `artifacts/`:

- `artifacts/site/`: the website. HTML with real, selectable math and **relative links**, so it works at a domain root, under a subpath, or straight off a file system. This directory is regenerable build output (`demolab clean` deletes it), so it stays out of git; CI rebuilds it on every deploy.
- `artifacts/pdfs/`: a PDF per entry, plus the bound book, and any standalone slide decks compiled by `demolab slides`.

Everything below is just ways of getting `artifacts/site/` onto a server.

Opting in to GitHub Pages

`demolab` deploy-setup

This copies two workflow files from the engine into your repo's `.github/workflows/`:

- `deploy.yml`: the production deploy. On every push to `main` (or a manual dispatch) it installs Typst and uv, runs `uv sync` and `uv run demolab build`, and publishes `artifacts/site/` to the root of a `gh-pages` branch as a single snapshot commit.
- `preview.yml`: per-PR previews. Every pull request gets its own built copy of the site under `pr-preview/pr-N/` on the same `gh-pages` branch, a sticky comment on the PR linking to it, and an automatic teardown when the PR closes or merges. So a reviewer reads the rendered pages and PDFs at a URL, not a raw diff.

Both files are engine templates: rerun `demolab deploy-setup` after an engine update to refresh them.

The GitHub-side clicks

Three settings can't be scripted, so the command prints them and you click. In the repo on GitHub:

1. *Settings* → *Pages* → *Source*: choose **Deploy from a branch**, then branch `gh-pages / (root)`. The branch won't exist until the first push to `main` runs the workflow; flip the setting once it does.
2. *Settings* → *General* → *Pull Requests*: enable **Automatically delete head branches**.
3. (Recommended) *Settings* → *Branches*: protect `main`.

Then commit the workflows and push: `main` deploys the site, and each PR gets its own preview URL.

The branch-based source matters. The workflows use the older build-then-commit-to-`gh-pages` flow rather than the GitHub-Actions Pages flow, precisely so production and the PR previews

can share one branch: the production deploy prunes stale files but explicitly preserves `pr-preview/`, and the two use separate concurrency groups so they never race a commit.

One caveat: pull requests from forks get a read-only token and can't write `gh-pages`, so previews are skipped for external contributors. That's expected, and fine for a personal lab; don't "fix" it by switching the trigger to `pull_request_target`, which would run untrusted PR code with a write token.

Custom domains

Set your domain in *Settings* → *Pages* as usual. GitHub records it by writing a `CNAME` file to the `gh-pages` branch, and `deploy.yml` excludes that file from its cleanup, so the domain survives every deploy. Nothing demolab-specific to configure.

Any static host

GitHub Pages is the packaged path, not a dependency. The site is plain files with relative links and no server-side anything, so any static host works the same way:

```
demolab build
# then upload artifacts/site/ to Netlify, Cloudflare Pages, S3, your own nginx, ...
```

Point the host at `artifacts/site/` as the publish directory and you're done.

Inside another project

If your lab isn't a standalone repo but the docs folder of a bigger project, the recipe changes: `deploy-setup`'s workflows assume the lab is the repo root, so you skip them and add a Pages workflow to the **host** repo instead, modelled on `deploy.yml`. The EMBED-DOCS runbook walks that path end to end.