Getting started

IC102-D is about everything around a digital Tiny Tapeout design except the design itself. You start from Tiny Tapeout's Verilog template, as every project on the team does, and by the end of the course you'll have a repository of your own with every workflow green, a pull request that a classmate reviewed and approved, and a revert you found with `git bisect` and got through every check. The Verilog comes with each lab, so you don't need to know any. This page says how the course is laid out and what you need before Lab 01, and it explains why you'll type every git command in the workbench rather than on your laptop.

How the course is laid out

The Course path on the home page alternates Read rows and Do rows. A Read row is one or more guide pages like this one. They explain, show real output from the team's repositories and from Tiny Tapeout's, and never ask you to run anything; a "Check yourself" block at the end of a page asks a few questions you can answer on paper. A Do row is a lab writeup, and every command in the course is in one of those. Each lab is split into parts, each part opens with the question it answers and ends with what you should see, and each lab ends with a "You're done when" line that you check against your own repository.

The files a lab runs (scripts, a design, a test, a checker) are in the course repository under `labs/`, one folder per lab, and the test job's tools are in `labs/common/tt-tools`. You get all of it with `mod add ic102-d` in Lab 01, Part 1.

Five labs take about eight hours, and a good part of that is waiting: a `gds` run takes around five minutes, and a lab may need three or four of them. Each lab says when you can read ahead while a run is going.

What you need

- **The workbench from IC101.** You'll run the course's tools in it, and every git command too, for reasons given in the next section. If you haven't set it up, do IC101 Lab 03 first.
- **A GitHub account of your own.** Every repository you make in this course goes on it, as a public repository, for three reasons: GitHub's standard runners are free and unlimited on public repositories, Pages (which the `viewer` job needs) is available on a free account only for public ones, and so are branch rules. You don't need a paid plan, and the course doesn't use the team's GitHub organization at all: Lab 04's review happens between your account and a partner's.
- **A token for git**, which Lab 01, Part 2 has you make. GitHub doesn't accept your password from git.

Where you run git

You'll type every git command in a terminal on the workbench desktop, in a clone under `/foss/designs/projects/`, named after its repository: `ttsky-ic102-d` for your project, and two more in Labs 04 and 05. `/foss/designs` is the `workspace` folder on your laptop, so your clones survive when the container stops, and you can open the files in an editor on your laptop if you prefer one. (On Windows, set that editor to save shell scripts with Unix line endings, because bash can't run a script whose lines end in the Windows way.) The course itself sits next to them, in `/foss/designs/modules/ic102-d`, which is where `mod` keeps courses. The alternative, cloning with git on the laptop and running the tests in the workbench, runs into two problems:

- On a Linux laptop whose user ID isn't 1000, the container's user doesn't own a clone your laptop made, and git in the container refuses to touch it with `fatal: detected dubious ownership in repository`.
- Lab 05's `git bisect run` checks out one old commit after another and runs the test at each, so git and the test have to run on the same machine.

Working in a container changes two things about how you use git. First, anything outside `/foss/designs` is deleted when the container is replaced, and that includes the file where `git config --global` saves your settings. So the labs have you set your name, email address and credential cache with plain `git config` inside each clone, which saves them in that clone's `.git/config`, on your own disk. Second, git asks for your token on the first push after each restart, because the cache that remembers it for 8 hours lives in the container's memory.

The token is a fine-grained personal access token for your own account, with access to all of your repositories and two permissions, both "Read and write":

- **Contents**, which is what pushing commits needs.
- **Workflows**, which lets a token add or change files under `.github/workflows/`. You won't edit a workflow in this course, but two of your pushes carry workflow files: the first push of your Lab 04 review repository, and the first push of Lab 05's history.

GitHub's documentation gives the same two permissions, for the same reasons, to a GitHub App that uses git (choosing permissions for Git access).

It needs all of your repositories, rather than a list, because the ones for Labs 04 and 05 don't exist yet when you make it in Lab 01. Every push in the course goes to a repository you own, including Lab 04's: your pull request to your partner comes from your own fork of their repository. GitHub also offers older "classic" tokens, whose `public_repo` scope covers every public repository your account can push to, including those of any organization you belong to. A fine-grained token reaches only your own repositories, which is all this course needs, so a token that leaks can do less harm.

Public repositories

Everything you push to a public repository can be read by anyone: your code, your commit messages, the name and email address on every commit, and the `author` field in `info.yaml`, which also goes into the datasheet. Lab 01 has you set your commit address to the no-reply address GitHub gives every account, before your first commit, because a commit can't be changed later without changing its hash and the hash of every commit after it. How open-source projects work says more.

Your tools and the runner's

The workbench has its own versions of the simulation tools, and GitHub's runner installs the versions Tiny Tapeout's workflows ask for. They differ: the runner's `test` job uses Icarus Verilog 12.0 and cocotb 2.1.0, and the workbench has 14.0 and 2.0.1. Lab 01 installs the runner's versions next to the workbench's, and `make test` runs your test with them, so a test that passes there passes on the runner. Your tools and the runner's tools explains what that covers and what it doesn't.

AI tools

You may use AI tools such as Claude, ChatGPT or Copilot in this course: to explain an error, to draft a test, or to help with a pull request description. You answer for everything you push, however it was written, so check what a tool gives you, and say where you used one. A pull request description is the place for that, and Pull requests and review explains why it matters to your reviewer. Lab 04 has an optional part where you review a pull request an AI tool wrote.

What the course doesn't cover

Tiny Tapeout puts designs on real chips, a few times a year, in shuttles with a deadline and a price per tile. This course stops just before that. You'll get a project to the state Tiny Tapeout asks for at submission, with every check green, but you won't create a project on Tiny Tapeout's site, pay for tiles or use a coupon. Hardening a design on your own machine, the FPGA workflow (which the template switches off) and writing workflows of your own are also left out.

Getting help

Ask in the team Discord. A question about a red cross gets answered fastest when it includes the link to the run, the job and step that failed, and the first error line copied as text, not as a picture, followed by what you've already tried. Logs on GitHub are deleted after 90 days at most, so text pasted into a message also lasts longer than a link to the log.

Check yourself

1. Why does the course have you run `git config user.email ...` in every clone, instead of once with `--global`?

Answer: A1, in Answers at the end.

2. Your project repository is private on a free account. Which parts of this course stop working?

Answer: A2, in Answers at the end.

3. A classmate made a classic token with the `public_repo` scope instead, and it works for every lab. What else can it push to, and why does that matter?

Answer: A3, in Answers at the end.

Answers

A1. Check yourself

Because `--global` saves the setting in your home folder inside the container, and the container is replaced, with everything outside `/foss/designs`, each time you start it. Without `--global` the setting goes in the clone's own `.git/config`, which is on your disk.

A2. Check yourself

The runners are no longer free and unlimited (a private repository draws on a monthly allowance), Pages isn't available, so the `viewer` job can't pass, and branch rules aren't available either. That's why the course uses public repositories.

A3. Check yourself

It can push to every public repository the classmate's account can write to, not just their own: if they belong to an organization whose members can write to its repositories, all of those too. If the token leaked, whoever found it could push to all of them. A fine-grained token made for their own account reaches only their own repositories, which is all the course needs.