Getting started

IC102-A is about the repository around an analog Tiny Tapeout design rather than the design itself. Every analog project on a Tiny Tapeout shuttle lives in a GitHub repository made from Tiny Tapeout's analog template, and the workflows in that repository check your layout on GitHub's machines each time you push. In this course you make one of those repositories, learn to read what its checks report, put a small layout in it that passes them, take a change through review with a partner, and use the history to find a commit that broke something.

You don't need to know how to draw layout. Each lab ships the layout it needs: Tiny Tapeout's empty frame in Lab 01, the CMOS inverter from AD104 Lab 03 in Lab 03, and a prepared project in Lab 05. You'll edit a few lines of the scripts that place and wire them, and the course's checks tell you whether the result is right. If you've done AD104, the cell will be familiar; if you haven't, nothing here depends on it.

How the course is laid out

The course path on the home page lists every page in the order to work through them. A Read row points to guide pages like this one, which explain things and have no commands for you to run. A Do row points to a lab writeup, and every command in the course is in a lab. Each lab is split into parts with numbered steps. Most parts start with the question they answer, and every part ends with a "You should see" or "Check" line that tells you whether it worked. When a step asks a question, its answer is in a collapsed block under it; try it first, then open the block. Some steps ask you to predict a result before you see it, which is worth doing honestly, because a wrong prediction is the quickest way to find out what you misunderstood.

The files the labs run are in the course's repository, under `labs/`. You'll get them into the workbench with `mod add ic102-a` in Lab 01.

Two places your work happens

Your work in this course happens in two places, the workbench and GitHub, and it's worth keeping them straight from the start.

The workbench is the IIC-OSIC-TOOLS desktop you set up in IC101. That's where you build layouts with Magic, run checks, and use git: every command in the labs is typed into a terminal on that desktop. Your repositories go in a `projects` folder of your workspace, `/foss/designs/projects`, each in a folder with the repository's name. `/foss/designs` is the `workspace` folder on your own disk, so they survive when the workbench restarts, and `projects/` keeps them apart from `modules/`, where `mod` puts the courses. Anything outside `/foss/designs` doesn't survive a restart, because the container's home folder is thrown away with the container. That's why, in Lab 01, you'll store git's settings in each repository rather than in your home folder.

GitHub is where your repository lives, and where its workflows run each time you push, on GitHub's machines (the *runners*). What a cloud runner is explains what those are. Some things can only be done on GitHub: making the repository, turning on Pages, opening and reviewing pull requests, and reading a run's results.

Tiny Tapeout's precheck, the most important of those results, doesn't use the workbench's tools. It installs its own versions of Magic, KLayout and the PDK on the runner, and a layout that passes with one set can fail with the other. So in Lab 01 you'll install precheck's versions next to the workbench's, and `make precheck` then tells you, on your own machine and before you push, what precheck on the runner will say. Your tools and the runner's tools has the details. IC102-A follows Tiny Tapeout's versions on purpose; the other courses use the workbench's.

Your GitHub account

You'll work in your own GitHub account throughout, so all you need is a free account: every repository you make in this course belongs to your account, and none goes in the team's organization. Every one of them is public, because GitHub Pages, which the template's `viewer` job needs, works only in public repositories on a free account, and because GitHub's runners are free and unlimited for public repositories. Remember that everything in a public

repository is public: your commits, the `author` field in `info.yaml`, and the email address in every commit. Lab 01 has you use GitHub's no-reply address for that reason.

These are the repositories you'll make, and each one's clone goes in `/foss/designs/projects/` under the same name:

Repository	Lab	Made how
`ttsky-ic102-a`	01 to 03	**Use this template** on Tiny Tapeout's analog template
`ttsky-ic102-a-review`	04	an empty repository, into which you push a copy of the course's sandbox
`ttsky-ic102-a-review-fork`	04	**Fork** on your partner's `ttsky-ic102-a-review`, renamed, because you already have a repository with that name
`ttsky-ic102-a-bisect`	05	an empty repository, into which you push a prepared history

Lab 04 is the one lab you do with a partner from the course. Each of you sends a pull request to the other's `ttsky-ic102-a-review`, and reviews the one that arrives in your own.

Pushing from the workbench

GitHub doesn't accept your account's password for `git push`. From the workbench you push over HTTPS with a **personal access token**, a long random string that stands in for a password and that you can limit to particular repositories and permissions. Lab 01 has you make a fine-grained token whose resource owner is your own account, for "All repositories", with 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).

"All repositories" is there because the Lab 04 and Lab 05 repositories don't exist yet when you make the token in Lab 01. Every push in the course goes to a repository you own: the four in the table above, including your fork of your partner's, from which your pull request goes to them. A fine-grained token for your own account covers those, and any other repository in your account, and nothing outside it. The other kind, a classic token with the `public_repo` scope, could push to every public repository your account can write to, including those of any organization you belong to (in `uoftasic`, members get write access to every repository by default), which is far more than the course needs. A fine-grained token can't push to a repository someone else owns, even if they add you as a collaborator, so in Lab 04 you send your change to your partner from a fork, which you own.

When `git push` asks for a username and password, you give your GitHub username and paste the token. A line you run in each clone tells git to keep it in memory for 8 hours, so you're asked about once a session, and again after the workbench restarts. Keep the token in a password manager, not in a file under `/foss/designs`: that folder is your workspace, itself a git repository, and a token saved there is one careless `git add` away from being published. Delete the token when you've finished the course.

Using AI tools

You may use AI tools such as Claude, ChatGPT or Copilot to help with the work in this course, for example to explain an error message, write a script or draft a pull request description. The rules are the ones in Pull requests and review: you answer for every line you submit, however it was written, and you say in a pull request which parts a tool wrote and what you checked yourself. The labs are built so that a check tells you when your work is right, whoever or whatever wrote it: LVS compares your layout with the reference either way.

Where the course stops

IC102-A takes a project as far as "ready to submit": every workflow green, including precheck, on a commit you made and can name. It doesn't cover submitting to a shuttle. You won't create a project on `app.tinytapeout.com`, pay, or use a coupon, and nothing in the course depends on a shuttle being open. Every project you make here is practice, and Lab 04's review repositories are never submitted.

Getting help

Ask in the team Discord. A question that can be answered quickly has three things in it: the lab, part and step you're on; the command you ran; and the output, pasted as text rather than a photo, from the first line that looks wrong. If it's about a run on GitHub, include the link to the run. Logs on GitHub are deleted after 90 days, so if the question might matter later, paste the error line itself as well.

Check yourself

1. You restart the workbench, open a terminal, and `git commit` stops with `Author identity unknown`. You set your name and email in Lab 01. What happened?

   Answer: A1, in Answers at the end.

2. Why does the course install a second Magic when the workbench already has one?

   Answer: A2, in Answers at the end.

Answers

A1. Check yourself

You set them with `--global`, which writes them to the container's home folder, and the restart threw that folder away. Set them again without `--global`, inside the repository, where they're stored in `.git/config` on your own disk.

A2. Check yourself

Because precheck, which decides whether your layout passes, uses Magic 8.3.568 with a particular commit of the PDK, and the workbench has Magic 8.3.681 with a different one. Running precheck's own versions locally means a pass on your machine predicts a pass on the runner.