IC101 - Introduction to IC Tools & Workspace

Learn how to set up IC tools and workspace on your device.

UofT ASIC Team

Contents

1 IC101 — Onboarding onto Tools 5

I 1–2. Terminal and git

2 Getting started 9

3 Terminal and git 11

4 Lab 01: Terminal and git 13

II 3–4. Docker

5 Why a containerized toolchain 17

6 What Docker needs 19

7 Lab 02: Install Docker 21

III 5–6. The workbench

8 The workbench desktop (noVNC) 25

9 What the smoke test checks 27

10 Lab 03: Launch the workbench 29

IV 7. Wrap-up

11 Where each tool sits in the tapeout flow 37

Appendices 41

A Troubleshooting 43

1. IC101 — Onboarding onto Tools

In IC101 you set up the containerized open-source toolchain that later courses build on. By the end you'll have a Linux desktop with the team's chip design (EDA) tools running in a browser tab on your laptop, a setup the course calls the workbench. A short check script called the smoke test then confirms that the tools and the SKY130 PDK work. The PDK (process design kit) is the set of files that describes the manufacturing process the tools design against.

IC101 is part of the UofT ASIC Team education materials, published at https://uoftasic.com/ic101/. The runnable tooling lives in the shared `workspace` repo rather than in this course repo.

At a glance

Track	Core (intro): IC101 comes before both the analog and the digital track.
Who it's for	Everyone starting the UofT ASIC courses
Prerequisites	None; Lab 01 starts from opening a terminal.
Tools	git, Docker, IIC-OSIC-TOOLS (the container image with the EDA tools), noVNC (shows the container's desktop in your browser)
Time	About 2–3 hours, much of it waiting for installs and the first image download
Machine	A laptop with roughly 15 GB of free disk space and permission to install software, and a stable network connection, because the toolchain image is several GB. On a Mac you'll also give Docker Desktop at least 4 GB of RAM in its settings (Lab 02, Part 1).
Shared workspace	The `workspace` repo, which you clone once in Lab 03 and reuse in every course that uses the tools.

Course path

Work down this table in order; the course ends with row 7. A Read row points you to guide pages, which explain things and have nothing to run. A Do row is one lab, and every hands-on step in the course is in a lab. Each lab's "Before you start" section lists the pages it expects you to have read.

#		What	Time
1	Read	Getting started · Terminal and git	10 min
2	Do	Lab 01: Terminal and git. Open a terminal, install git and check that `git --version` prints a version.	15 min

#		What	Time
3	Read	Why a containerized toolchain · What Docker needs	10 min
4	Do	Lab 02: Install Docker. Install Docker and run its `hello-world` test.	30 min–1 h
5	Read	The workbench desktop · What the smoke test checks	15 min
6	Do	Lab 03: Launch the workbench. Clone `workspace`, open the desktop in your browser, pass the smoke test, then restart the container and check which of your files survived.	30 min–1 h
7	Read	Where each tool sits in the tapeout flow	10 min

If a step fails, look it up in Troubleshooting.

After IC101

AD101 starts the analog track and DD101 starts the digital track. You'll find every course listed on the portal at edu.uoftasic.com.

I 1–2. Terminal and git

2 **Getting started** 9
2 What you need 9
2 What you'll set up 9
2 This repo and the workspace repo 9
2 How the course works 9

3 **Terminal and git** 11
3 The terminal 11
3 Git 11

4 **Lab 01: Terminal and git** 13
4 Before you start 13
4 Part 1: Open a terminal 13
4 Part 2: Install git 13
4 Finishing 14

2. Getting started

IC101 is the first course in the UofT ASIC Team's Internal Education Initiative, and it has no prerequisites. By the end of it, the team's open-source EDA toolchain runs in a browser tab on your own laptop, and a smoke test has confirmed that the tools and the SKY130 PDK work. Every later course that uses the tools starts from this same setup.

What you need

- A laptop with roughly 15 GB of free disk space. On a Mac, Docker also needs at least 4 GB of RAM.
- Permission to install software: Docker Desktop, and git if you don't already have it. On a school- or work-managed laptop this may need admin or IT permission.
- A stable network connection, because the toolchain image you download is several GB.

You don't need to have used a terminal before, since Lab 01 starts by opening one.

What you'll set up

The setup has three layers, and each lab adds one:

1. git (Lab 01), to copy the team's `workspace` repo onto your laptop.
2. Docker (Lab 02), to download and run the toolchain container.
3. The `workspace` repo (Lab 03), whose launch script starts the IIC-OSIC-TOOLS container with your `workspace` folder mounted inside it, so that you can open the container's Linux desktop in your browser.

Why a containerized toolchain explains why the team ships the tools this way instead of asking you to install each one.

This repo and the workspace repo

This `ic101` repo holds only the course pages you're reading. The Docker launch scripts, the file that pins the PDK version, and the helper commands you use inside the container live in a separate, shared repo called `workspace`. You clone it once in Lab 03 and keep the same clone for every later course.

How the course works

The home page has a Course path table. Work down it in order:

- A **Read** row is one or two guide pages, which explain the concepts and give you nothing to run.
- A **Do** row is one lab. Labs hold every hands-on step, in numbered parts, and each part ends with a "You should see" check so you know it worked before you move on.

When a step fails, look it up in Troubleshooting, which is organized by the error you see.

Next: Terminal and git.

3. Terminal and git

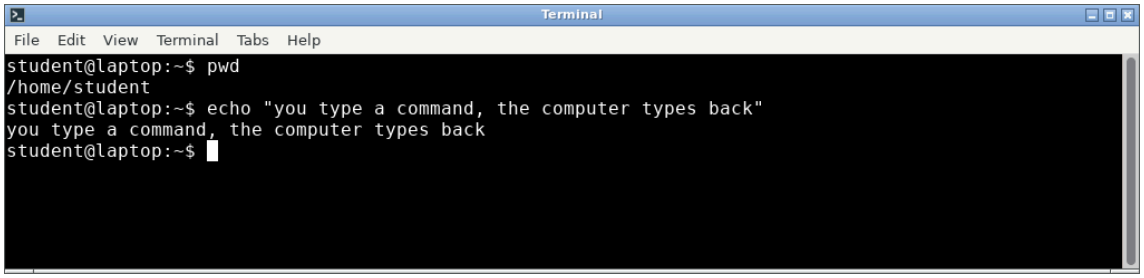
Every later step in IC101 assumes you have a way to type commands and that git is installed. This page covers just enough of both to get you through the course, and it isn't meant as a full tutorial on either. You'll open a terminal and install git in Lab 01.

The terminal

A **terminal** (also called a shell, or the command line) is a text window where you type commands instead of clicking buttons. Every operating system comes with one:

OS	Terminal app
macOS	Terminal
Windows	Windows Terminal, or PowerShell
Linux	Varies by Linux distribution (distro), for example GNOME Terminal or Konsole

When the terminal opens, it shows a **prompt**, which often ends in `$` or `>`, and waits for you to type. When a page in this course says "run this command", it means you click into the terminal window, type or paste the command, and press Enter.



A shell prompt waiting for a command. The `$` marks where you type, but it isn't part of the command, so don't type the `$` itself.

Git

Git is a tool for downloading a folder of files from a site like GitHub onto your computer and keeping the two copies in sync. The folder is called a **repository**, or **repo**, and copying a repo onto your computer for the first time is called **cloning** it.

In IC101 we only use git for one thing. In Lab 03 you run a single command, `git clone`, to copy the team's shared `workspace` repo onto your laptop, and that's all the git you need for this course.

Next: Lab 01: Terminal and git.

4. Lab 01: Terminal and git

You'll open a terminal on your own computer, install git if you don't already have it, and check that git runs. Every later lab starts with you typing commands into this terminal.

Before you start

- Read Getting started and Terminal and git.
- You need permission to install software on this computer.

Time: about 15 minutes, or less if git is already installed.

Part 1: Open a terminal

1. Open the terminal app for your OS:

OS	App	How to open it
macOS	Terminal	Press `Cmd+Space`, type `Terminal`, press Enter
Windows	Windows Terminal (or PowerShell)	Press the Windows key, type `Terminal` (or `PowerShell`), press Enter
Linux	Varies by Linux distribution (distro), for example GNOME Terminal or Konsole	Press the Super (Windows) key, type `Terminal`, press Enter

2. Click inside the window, so that what you type goes to the terminal and not to some other app.
3. Whenever a step in a lab says "Run:", type or paste the command shown under it into the terminal, then press Enter.

You should see a window with a prompt, which often ends in `$` or `>`, waiting for you to type.

Part 2: Install git

Follow the steps for your OS, and skip the other two.

macOS

1. Run:

   ```
   git --version
   ```

   If it prints a version number, git is already installed and you can skip to Finishing.
2. If git isn't installed yet, macOS instead offers to install the **Xcode Command Line Tools**, which include git. Accept the offer and wait for the install to finish.
3. Run `git --version` again.

Windows

1. Download and run the installer from git-scm.com.

2. The installer's default options are fine for this course, so accept them as you go through it.
3. Close any terminal you already had open, and open a new Windows Terminal or PowerShell window. A terminal that was already open during the install won't find git, so you need a new one.
4. Run:

```
git --version
```

Linux

1. Run:

```
sudo apt update && sudo apt install -y git
git --version
```

   `sudo` asks for your password. Nothing appears on screen while you type it, which is normal, so type it anyway and press Enter. If your distro doesn't use `apt`, use its own package manager instead, for example `dnf` on Fedora or `pacman` on Arch.

On every OS, you should see `git --version` print a version number, in a line like `git version 2.34.1`. Your number may be different, and some systems add more text after it. If you see `command not found` instead (on Windows, `'git' is not recognized...`), see Troubleshooting.

Finishing

You're done when `git --version`, run in a terminal on your own computer, prints a version number rather than `command not found`.

Submit

Nothing to submit.

Next on the Course path: read Why a containerized toolchain and What Docker needs, then do Lab 02: Install Docker.

II

3–4. Docker

5 **Why a containerized toolchain 17**
5 What the team standardizes 17
5 What IC101 doesn't cover 17

6 **What Docker needs 19**
6 Disk, memory and permission 19
6 Windows: WSL2 and virtualization 19
6 Linux: the docker group 19
6 Pulling the image 19
6 The hello-world check 19

7 **Lab 02: Install Docker 21**
7 Before you start 21
7 Part 1: Install Docker 21
7 Part 2: Run hello-world 22
7 Finishing 22

5. Why a containerized toolchain

A **container** is a self-contained bundle of software that runs the same way on every computer, whether that computer runs Windows, macOS or Linux, so you don't have to install each program separately. You can think of it as one pre-built package that holds an entire toolchain, ready to run. Strictly speaking, the package you download is called an **image**, and a container is a copy of that image running on your computer. You download the image once, and after that you can remove a container and start a new one from the same image without downloading anything, as you'll do in Lab 03.

Open-source ASIC flows use dozens of tools: schematic capture, SPICE, layout, DRC/LVS, synthesis, and place and route (P&R). Installing each of them natively on Windows, macOS and Linux is fragile, because tool versions drift from one machine to the next and the PDKs on different machines disagree, until sorting out "works on my machine" becomes the course.

What the team standardizes

The team uses the IIC-OSIC-TOOLS Docker image, which IIC-JKU maintains. The team's `workspace` repo wraps that image with:

- launch scripts for noVNC (a Linux desktop in your browser) and, optionally, X11
- a pinned SKY130 PDK (`pdk/volare.lock`)
- shared shell helpers (`common/.designinit`)
- a starter folder for each course, under `modules/`

Apart from git, the only thing you install yourself is Docker. Everything else runs inside one container, and the `workspace` folder on your laptop appears inside that container at `/foss/designs`. noVNC shows the container's desktop in Chrome, Firefox or Edge, so you can open terminals, XSchem, Magic and the other tools without installing any EDA package on your own operating system. The workbench desktop explains how that desktop starts.

What IC101 doesn't cover

- It isn't a theory-heavy introduction to semiconductors; later courses cover that.
- It isn't a substitute for the ECE curriculum.
- It doesn't give each course its own Docker image, because one workspace serves every tool-heavy course.

Next: What Docker needs.

6. What Docker needs

Docker is the program that downloads the toolchain image described in Why a containerized toolchain and runs it as a container. The rest of the course runs inside that container, so none of it works until Docker is installed and running. You'll install it in Lab 02. This page explains what the installer asks of your computer, and why.

Disk, memory and permission

- Roughly 15 GB of free disk space, which covers the image, the PDK and room to work.
- Permission to install Docker Desktop, or Docker Engine on Linux. On a school- or work-managed laptop this may need admin or IT permission.
- On a Mac, Docker Desktop needs at least 4 GB of RAM, which you set under **Settings → Resources**. Apple Silicon Macs (M1/M2/M3) work too, because the IIC-OSIC-TOOLS image is multi-arch, which means it's built for their processors as well as for Intel ones.

Windows: WSL2 and virtualization

WSL2 is a compatibility layer that lets Windows run a lightweight Linux environment behind the scenes. Docker Desktop needs it and uses it by default, and the installer offers to set it up for you. Keep that default by leaving the installer's **Use WSL2 instead of Hyper-V** option checked.

Docker also needs your processor's virtualization support switched on in the BIOS/UEFI setup, the settings screen your computer shows before Windows starts. The setting is usually called Virtualization, Intel VT-x or AMD-V. If it's off, Docker shows a window asking you to enable it, and Lab 02 walks you through turning it on.

Linux: the docker group

On Linux, your user has to be in the `docker` group before it can talk to Docker without `sudo`, and until it is, `docker` commands fail with `permission denied`. Adding yourself to the group doesn't take effect straight away: the change applies only after you log out and back in, and terminals that were already open won't pick it up.

Pulling the image

Pulling an image means downloading it. The first time you start the workbench in Lab 03, the launch script pulls `hpretl/iic-osic-tools:<tag>`, where the tag is currently `2026.08` in the workspace scripts. The image is several GB, so use a stable network connection and expect it to take a few minutes. The IIC-OSIC-TOOLS releases page lists the tags.

The hello-world check

The last step of Lab 02 is `docker run --rm hello-world`, which runs Docker's own test image. When Docker is working, that command prints a message starting with `Hello from Docker!`.

Next: Lab 02: Install Docker.

7. Lab 02: Install Docker

In this lab you'll install Docker Desktop (or Docker Engine on Linux) and run Docker's `hello-world` test to show that it works. Lab 03 runs the course workbench in Docker, so it needs Docker installed and running.

Before you start

- Finish Lab 01, so that you know how to open a terminal.
- Read Why a containerized toolchain and What Docker needs.
- Check that you have roughly 15 GB of free disk and permission to install software. On a school- or work-managed laptop you may need admin or IT permission.

Time: 30 minutes to 1 hour. Allow longer on Windows if you have to change a BIOS setting.

Part 1: Install Docker

Follow the steps for your operating system.

macOS

1. Download Docker Desktop for Mac.
2. Open the downloaded file, drag Docker into Applications, then launch it.
3. Wait until the Docker icon in the menu bar stops animating, then click the icon. Its status should read **Engine running**.
4. In Docker Desktop, open **Settings → Resources** and allocate at least 4 GB of RAM.

Windows

1. Install Docker Desktop for Windows. During setup, leave the default **Use WSL2 instead of Hyper-V** option checked.
2. If a window asks you to enable virtualization in BIOS/UEFI:
   a. Restart your computer and enter the BIOS/UEFI setup. You get there by pressing a key right after the computer powers on, usually `F2`, `Del` or `Esc`; the exact key is shown briefly on screen as it starts.
   b. Find the setting called **Virtualization**, **Intel VT-x** or **AMD-V**, and enable it.
   c. Save and exit. The computer boots back into Windows.
3. Open Docker Desktop and wait until it shows **Engine running** in the bottom-left corner.

Linux

1. Install Docker Engine or Docker Desktop for your distro, from your distro's package manager or following docs.docker.com.
2. Add your user to the `docker` group, so you don't need `sudo` for every `docker` command:

   ```
   sudo usermod -aG docker $USER
   ```

3. Log out and log back in, or restart. A group change only takes effect when you log in, so it doesn't apply to the session you're already in.

You should see **Engine running** in Docker Desktop. If you're on Linux with Docker Engine, run `groups` after you log back in, and you should see `docker` in the list.

Part 2: Run hello-world

1. Open a terminal. On Windows, use Windows Terminal or PowerShell.
2. Run:

```
docker version
docker run --rm hello-world
```

You should see version information from `docker version`, with a `Client` section and a `Server` section, and then `hello-world` prints a paragraph starting with `Hello from Docker!`. If the `Server` section is missing or shows an error, the Docker engine hasn't started yet, so wait for **Engine running** and try again. On Linux, an error that mentions `permission denied` means the `docker` group change from Part 1 hasn't applied yet, and logging out and back in will fix it.

If `hello-world` fails, fix Docker before you go on, because every later step in the course depends on it. Start with the Troubleshooting page.

Finishing

You're done when `docker run --rm hello-world` prints `Hello from Docker!` on your own computer, you have roughly 15 GB of free disk for an image several gigabytes in size, and, on a Mac, Docker Desktop has at least 4 GB of RAM.

Submit

Nothing to submit.

Next on the Course path: read The workbench desktop and What the smoke test checks, then do Lab 03: Launch the workbench.

III

5–6. The workbench

8 The workbench desktop (noVNC) . . 25
8 The workspace repo 25
8 What `start_vnc` does 25
8 Settings you can change 25
8 The desktop in your browser 26
8 X11 instead of noVNC 26

9 What the smoke test checks 27
9 Two terms 27
9 The environment file 27
9 The checks 27
9 When the PDK check fails 27
9 Course modules and `mod` 28

10 Lab 03: Launch the workbench 29
10 Before you start 29
10 Part 1: Clone the workspace 29
10 Part 2: Start the container 29
10 Part 3: Open the desktop 30
10 Part 4: Run the smoke test 31
10 Part 5: Restart the container 32
10 Finishing 33
10 Answers 33

8. The workbench desktop (noVNC)

To launch the desktop, follow Lab 03, Parts 1–3. This page explains what those steps do. If the desktop page is blank or port 80 is taken, see Troubleshooting.

The workbench is a container started from IIC-OSIC-TOOLS, a toolchain image that IIC-JKU maintains. You start it with a script from the team's `workspace` repo, and **noVNC** shows the container's Linux desktop in your browser.

The workspace repo

You clone the `workspace` repo once, into a folder named `workspace`, and keep that clone for every later course. These are the parts IC101 uses:

Path in `workspace`	Purpose
`scripts/start_vnc.sh` / `.bat`	Launches IIC-OSIC-TOOLS with noVNC
`scripts/smoke_test.sh`	Checks that the tools inside the container work
`common/.designinit`	Sets up the environment and SKY130
`modules/ic101_setup/`	Scratch space for this course
`pdk/`	`volare.lock`, which records the PDK version the team uses. The PDK itself ships inside the image, at `/foss/pdks/sky130A`.

Later courses add their own folders under `modules/` with `mod add <course>`, a command described in What the smoke test checks.

What `start_vnc` does

`start_vnc.sh` (macOS and Linux) and `start_vnc.bat` (Windows) do the same three things:

1. Pull the image `hpretl/iic-osic-tools:2026.08`. The first run downloads several GB, which can take several minutes, and later runs reuse the copy already on your computer.
2. Run the container, named `asic-edu-osic`, with your `workspace` folder mounted at `/foss/designs`. Anything you save under `/foss/designs` is a file in your `workspace` folder on your laptop, so it survives when the container is stopped or removed.
3. Expose the desktop at http://localhost/, on port 80 by default.

Running `docker rm -f asic-edu-osic` on your laptop stops and removes the container, and running `start_vnc` again brings it back. You'll try both in Lab 03, Part 5.

Settings you can change

The script reads the environment variables below. You won't need any of them in Lab 03 unless something else on your laptop is already using port 80.

Variable	Default	What it changes
`HOST_PORT`	`80`	The port on your laptop. With `HOST_PORT=8080` the desktop is at `http://localhost:8080/`. Use it when something else is already using port 80.
`VNC_PW`	`abc123`	The desktop password
`VNC_RESOLUTION`	`1280x800`	The desktop size, for example `1920x1080`
`DOCKER_TAG`	`2026.08`	Which image tag to pull

On macOS and Linux you set them by putting them in front of the command, for example `VNC_PW='your-password' HOST_PORT=8080 ./scripts/start_vnc.sh`.

The desktop in your browser

noVNC serves a full Linux desktop in Chrome, Firefox or Edge, and the password is `abc123` unless you set `VNC_PW`. Terminals, XSchem, Magic and the rest of the tools all open from that desktop, and they run inside the container rather than on your laptop.

Plain `Ctrl+V` doesn't paste text from your laptop into the desktop. To paste there, press **Ctrl+Shift+V**, or use the clipboard icon in the noVNC sidebar.

X11 instead of noVNC

On Linux, or on macOS with XQuartz, `./scripts/start_x.sh` shows the tools in windows on your own desktop over X11, which can be faster than streaming a whole desktop to the browser. We use noVNC in IC101 so that everyone sees the same interface.

Next: What the smoke test checks.

9. What the smoke test checks

The **smoke test** is a script in the workspace, `scripts/smoke_test.sh`, that checks whether the EDA tools and the SKY130 PDK resolve inside the container. You'll run it in Lab 03, Part 4. This page explains what each check looks at and what a failure usually means.

Two terms

- Your **`PATH`** is the list of folders your shell searches when you type a command name. A tool "resolves" when the shell finds it on the `PATH` after you type its name.
- A **PDK** (process design kit) is the set of files that describe how one chip fab's manufacturing process behaves, so that tools can simulate your designs and check them against it. The courses use SKY130, an open PDK, which the tools call `sky130A`. Inside the container its files are in `/foss/pdks/sky130A` (`$PDK_ROOT` is `/foss/pdks`).

The environment file

Inside the container, your `workspace` folder appears as `/foss/designs`. The file `/foss/designs/common/.designinit` sets up the shell for the courses: it sets the environment variables the tools read, points the PDK paths at SKY130, and defines the `mod` command. It leaves your `PATH` alone, because the image's own setup already puts the tools on it in every terminal. Lab 03 has you load it with `. /foss/designs/common/.designinit` before you run the smoke test. The leading `.` tells the shell to run the file in your current shell rather than in a new one, so its settings stay in the terminal you're using.

The checks

Check	Role
`ngspice`	Circuit simulation
`xschem`	Schematic capture
`magic`	Layout
`netgen`	LVS (netlist comparison)
`yosys`	RTL synthesis
`sky130A` under `$PDK_ROOT`	The SKY130 PDK is present and is the one in use
`mod`, `add_module.sh`, `ic101_setup`	The course module helpers work: `mod` is defined, `add_module.sh` is present, and `mod` lists and enters this course's `ic101_setup` folder

When every check passes, you'll see a column of `OK` lines, along with an ngspice version line and a `--- module helpers ---` heading, and the last line is `=== All checks passed ===`.

When the PDK check fails

If this check fails, the PDK is almost never missing, because SKY130 ships inside the image at `/foss/pdks/sky130A`. The usual cause is that the image starts on a different PDK, `ihp-sg13g2`, and the tools look in the wrong place until the workspace environment switches everything over to SKY130. Loading `.designinit` is what makes that switch. Troubleshooting has the exact messages and the fix.

Course modules and `mod`

Each course keeps its files in a folder under `modules/` in your workspace, which the container sees as `/foss/designs/modules/`. Once you've loaded `.designinit`, the `mod` command moves you between those folders:

Command	What it does
`mod`	List the course modules
`mod ic101_setup`	Go to this course's scratch folder, `/foss/designs/modules/ic101_setup`
`mod add ad101`	Clone `github.com/uoftasic/ad101` into `modules/ad101` (a later course)
`mod ad101`	Go to `modules/ad101`, once it's cloned

On your laptop, running `./scripts/add_module.sh ad101` from your `workspace` clone does the same thing as `mod add ad101`, because that folder is the one the container mounts as `/foss/designs`, so both commands write into the same `modules/` folder.

`mod add <course>` first prints the address it clones from, then git's own `Cloning into ...` line. If the clone works, the output ends with:

```
OK  module ready: /foss/designs/modules/<course>
    run: mod <course>
cwd: /foss/designs/modules/<course>
```

The `cwd:` line means `mod add` has already moved you into the new folder. If it stops and asks for a GitHub username instead, see Troubleshooting.

Next: Lab 03: Launch the workbench.

10. Lab 03: Launch the workbench

You'll clone the team's `workspace` repo, start the IIC-OSIC-TOOLS container, open its desktop in your browser, and run the smoke test. Then you'll restart the container and check which of your files survived. When the smoke test prints `=== All checks passed ===` after that restart, your setup is ready for every later course.

Other courses link here for their workbench setup, so if one of them sent you, do this whole lab and then go back to that course.

Before you start

- You need git and Docker on your laptop. In a terminal there, run:

  ```
  git --version
  docker run --rm hello-world
  ```

  The first should print a version number, and the second a paragraph starting with `Hello from Docker!`. If git is missing, do Lab 01 first. If Docker isn't installed, do Lab 02. If it's installed but the command fails, Docker may still be starting: wait until Docker Desktop shows **Engine running** and try again. If it still fails, or you run Docker Engine on Linux without Docker Desktop, see Troubleshooting.
- On a Mac, give Docker Desktop at least 4 GB of RAM under **Settings → Resources** (Lab 02, Part 1).
- Read The workbench desktop and What the smoke test checks.
- Have roughly 15 GB of free disk and a stable network connection, because the first launch downloads an image that is several GB in size.

Time: 30 minutes to 1 hour, most of it waiting for the first download.

Part 1: Clone the workspace

1. Open a terminal on your laptop.
2. Run:

   ```
   git clone https://github.com/uoftasic/workspace.git
   cd workspace
   ```

   The first command creates a folder named `workspace` inside the folder your terminal is in, and puts the team's scripts and configuration in it. The second moves your terminal into `workspace`. Remember which folder you ran this in, because you'll run the laptop commands in Parts 2 and 5 from inside `workspace`, and later courses start the workbench from it too.

You should see git print `Cloning into 'workspace'...` and finish without an error. If you get `git: command not found`, go back to Lab 01. If git says `fatal: destination path 'workspace' already exists`, you've already cloned it in this folder, so run `cd workspace` and use that clone.

Part 2: Start the container

1. From inside `workspace`, start the container.

   On macOS or Linux:

   ```
   ./scripts/start_vnc.sh
   ```

On Windows, run it from Windows Terminal or PowerShell:

```
scripts\start_vnc.bat
```

You can also double-click `scripts/start_vnc.bat` in File Explorer, but the window may close when the script ends, before you can read what it printed.

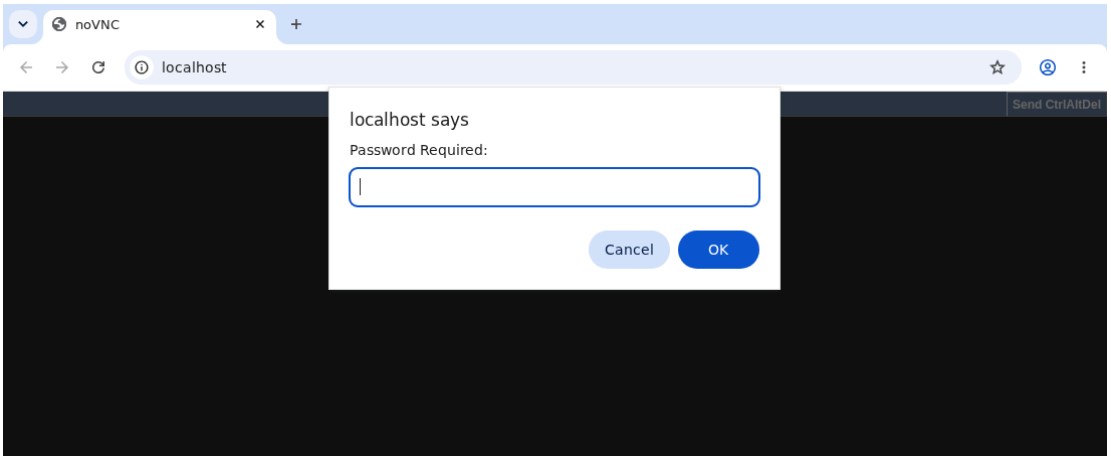
2. Wait for the script to finish. The first run downloads `hpretl/iic-osic-tools:2026.08` and can take several minutes.

You should see the script end with a block that starts `=== Open in your browser ===`, followed by the desktop's address and password.

If the script fails because port 80 is in use, start it on another port (on macOS or Linux, `HOST_PORT=8080 ./scripts/start_vnc.sh`) and use `http://localhost:8080/` in Part 3. Other failures are in Troubleshooting.

Part 3: Open the desktop

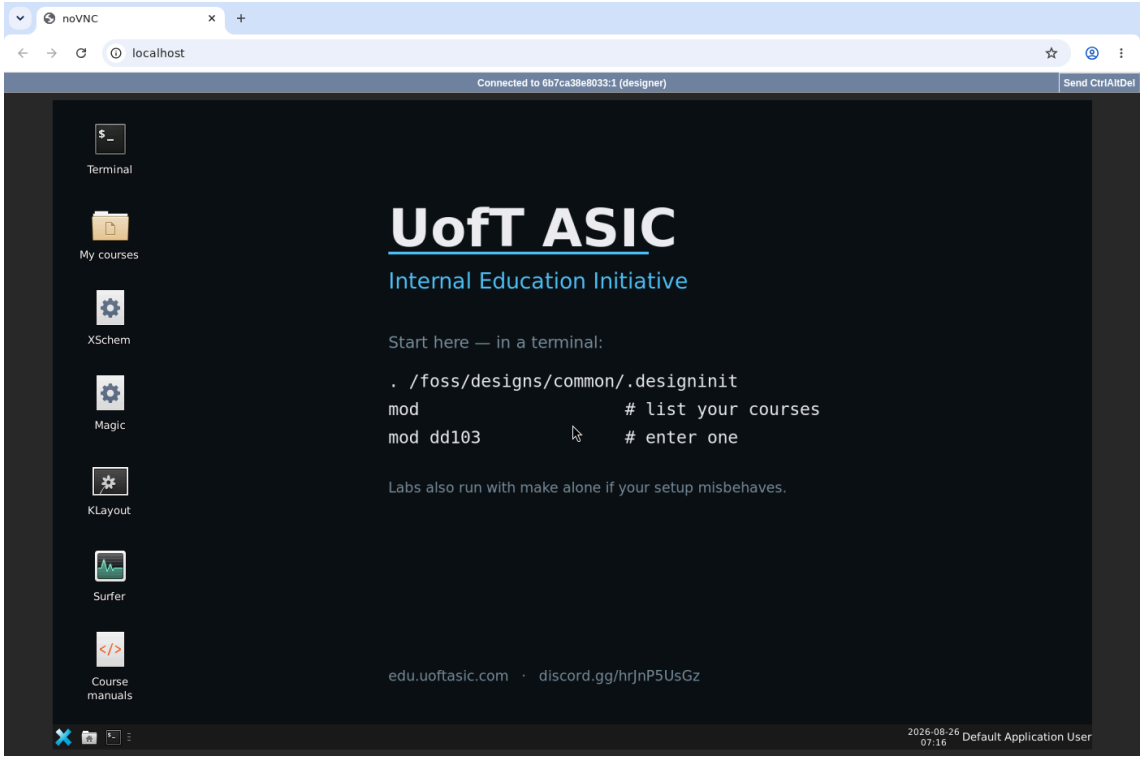
1. In your browser, go to http://localhost/, or `http://localhost:<HOST_PORT>/` if you set a port in Part 2.



The address bar shows plain `localhost`, which means the default port, 80. With `HOST_PORT=8080` it would show `localhost:8080`.

2. Enter the password the script printed: `abc123`, unless you changed it with `VNC_PW` (Settings you can change).

You should see a Linux desktop with a column of icons down the left, including **Terminal**, **XSchem** and **Magic**.



The desktop, running inside the browser tab. `Connected to ...` in the top bar means the VNC session is live. Every later course that uses the tools runs on this machine.

If the page is blank or won't load, wait 30–60 s and reload. If it's still blank, see Troubleshooting.

Part 4: Run the smoke test

Every command in this part runs in a terminal inside the noVNC desktop, not on your laptop. Inside the container, your `workspace` folder is `/foss/designs`.

1. Open **Terminal** from the icons on the left of the desktop. To paste a command from this page into it, press **Ctrl+Shift+V**, or use the clipboard icon in the noVNC sidebar. Plain `Ctrl+V` doesn't paste into the desktop.
2. Load the workspace environment:

   ```
   . /foss/designs/common/.designinit
   ```

   `.designinit` points the tools at the SKY130 PDK and defines the `mod` command you'll use in steps 4 and 5. The `.` and the space after it are part of the command, because the `.` is what loads the file into this terminal.

   You should see:

   ```
   ASIC-EDU: PDK=sky130A  PDK_ROOT=/foss/pdks  designs=/foss/designs
             run 'mod' to list course modules, or 'mod <name>' to enter one.
   ```

   A new terminal on the desktop loads this file by itself, so you may already have seen these two lines when the terminal opened. Running it again does no harm, and it's also the fix if a terminal ever says `bash: mod: command not found`.
3. Run the smoke test:

   ```
   /foss/designs/scripts/smoke_test.sh
   ```

   You should see the same two `ASIC-EDU` lines, then:

```
=== ASIC-EDU workspace smoke test ===
OK  ngspice
OK  xschem
OK  magic
OK  netgen
OK  yosys
OK  PDK sky130A at /foss/pdks/sky130A
    ngspice-47
--- module helpers ---
OK  mod function loaded
OK  add_module.sh present
OK  mod lists ic101_setup
OK  mod ic101_setup
=== All checks passed ===
```

The indented `ngspice-47` line is the ngspice version. If any line says `FAIL` or `WARN`, see Smoke test: tool or helper not found or Smoke test: PDK is not sky130A.

4. List the course modules:

```
mod
```

You should see `Available modules:` and, under it, `ic101_setup`.

5. Go to IC101's scratch folder:

```
mod ic101_setup
```

You should see `cwd: /foss/designs/modules/ic101_setup`.

Part 5: Restart the container

Step 3 below is how you stop the workbench, and steps 4 and 5 are how you start it again at the beginning of every later session. Before you restart it, you'll save two test files and predict which of them the restart keeps.

1. In the desktop terminal, save one file in your workspace and one in the container's `/tmp` folder:

```
echo "saved in the workspace" > /foss/designs/modules/ic101_setup/restart-test.txt
echo "saved in the container" > /tmp/restart-test.txt
```

2. Write down which of the two files you expect to find after you remove the container and start a new one.
3. In a terminal on your laptop (not the desktop), stop and remove the container. The launch script named it `asic-edu-osic`:

```
docker rm -f asic-edu-osic
```

It prints the name, `asic-edu-osic`, when the container is gone.

4. From inside your `workspace` folder, start the container again as in Part 2: `./scripts/start_vnc.sh` on macOS or Linux, `scripts\start_vnc.bat` on Windows. This time the image is already on your computer, so there's nothing to download.
5. Reload http://localhost/ (or your `HOST_PORT` address), enter the password, and open **Terminal** from the icons on the left. If the page is blank, wait 30–60 s and reload, as in Part 3.
6. Look for both files:

```
cat /foss/designs/modules/ic101_setup/restart-test.txt
cat /tmp/restart-test.txt
```

7. Run the smoke test again:

```
/foss/designs/scripts/smoke_test.sh
```

In step 6 you should see the workspace file's text, then an error for the `/tmp` file:

```
saved in the workspace
cat: /tmp/restart-test.txt: No such file or directory
```

In step 7, the smoke test should end with `=== All checks passed ===`, as in Part 4. Compare the `cat` output with what you wrote down in step 2.

Answer: A1, in Answers at the end.

Keep your course work under `/foss/designs`. Even if you skip step 3, `start_vnc` removes any old `asic-edu-osic` container before it starts a new one, so every launch gives you a fresh container and anything you saved outside `/foss/designs` is lost. You can leave `restart-test.txt` where it is, since `ic101_setup` is IC101's scratch folder.

Finishing

You're done when, after the restart in Part 5, the smoke test ends with `=== All checks passed ===` in a desktop terminal and `restart-test.txt` is still in `ic101_setup`.

Submit

Nothing to submit. Keep the `workspace` clone, though, because every later course that uses the tools runs in it.

Next on the Course path: read Where each tool sits in the tapeout flow. If another course sent you here, go back to the page that sent you. Courses that use the tools add their own files to your workspace with `mod add <course>`, which What the smoke test checks describes.

Answers

A1. Part 5: Restart the container

`/foss/designs` is your `workspace` folder on your laptop, mounted into the container, so the first file is on your own disk and survives. `/tmp` exists only inside the container, and `docker rm -f` deleted the container with everything in it.

IV 7. Wrap-up

11 Where each tool sits in the tapeout flow 37
11 Analog and mixed-signal flow 37
11 Digital flow (open-source ASIC) 38
11 Shared infrastructure 38
11 Pick a track 38

11. Where each tool sits in the tapeout flow

The smoke test you just passed checked a list of tools, and this page shows where each of them fits on the way from an idea to a chip. You won't use them to design anything in IC101; the design work itself comes in later courses. Both flows below end at **tapeout**, the point where a finished chip design is sent off to be manufactured. What you send to the fab is a **GDSII** file, the format that describes the final layout.

Analog and mixed-signal flow

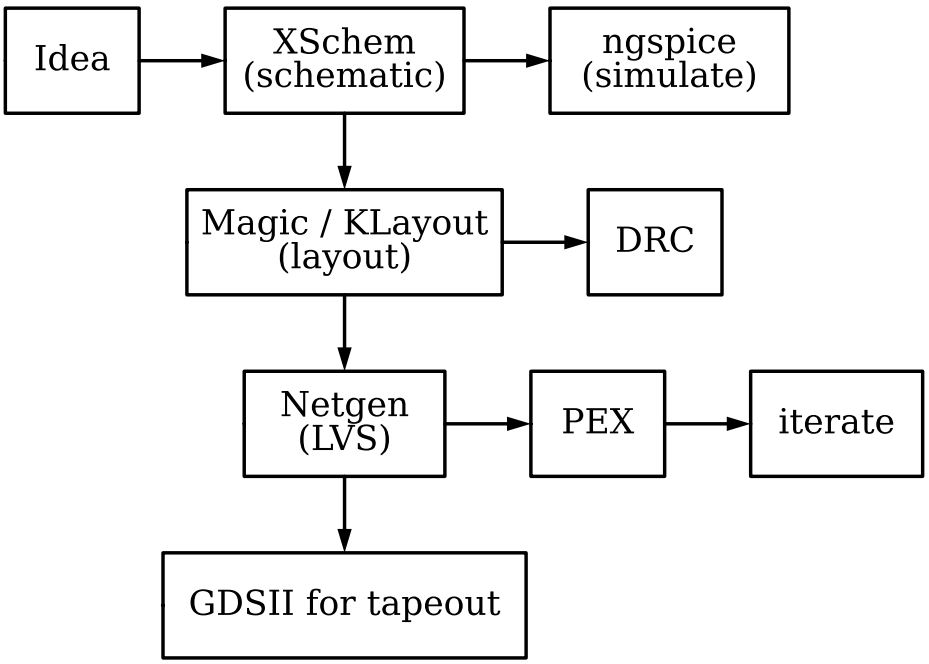


Figure 11.1: From an idea to the GDSII file for tapeout.

Tool	Stage
XSchem	Draw and edit schematics
ngspice	Simulate circuits from those schematics
Magic / KLayout	Full-custom layout; DRC
Netgen	Layout vs schematic (LVS)

Digital flow (open-source ASIC)

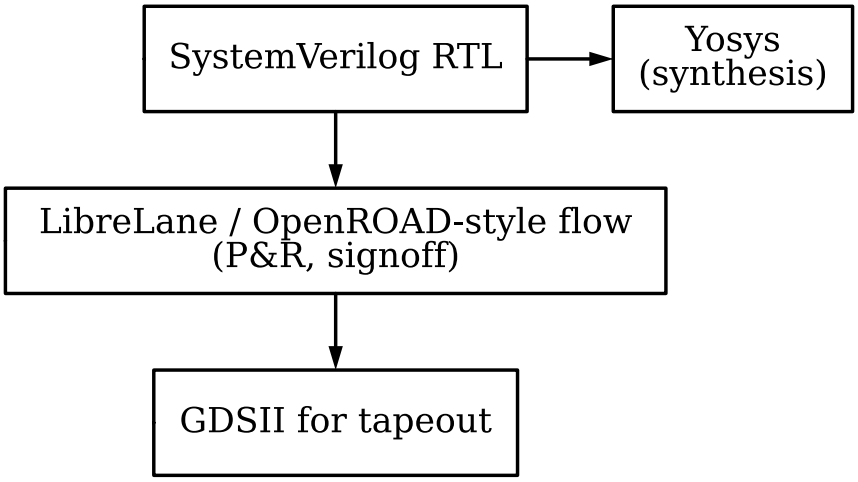


Figure 11.2: From SystemVerilog RTL to the GDSII file for tapeout.

Tool	Stage
Yosys	RTL → gate-level netlist
Verilator / cocotb	Pre-silicon verification (later courses)
LibreLane	Automated RTL-to-GDSII (DD103 and later)

The early digital courses also use FPGAs, as a second path alongside the open ASIC flow. In DD102 you take a design through the open FPGA toolchain (Yosys, nextpnr and IceStorm). That toolchain is already in the workbench image, so you won't have any vendor software to install.

Shared infrastructure

Piece	Role
Docker + IIC-OSIC-TOOLS	One environment that holds all the tools above
SKY130 PDK	The process design kit for the open SKY130 process
noVNC	Lets you reach the workbench desktop from a browser
`workspace` `modules/`	Each course's files, mounted into the container

Pick a track

This is the end of IC101. Both tracks start from here, and both assume the workbench you've just built. You can take them in either order, or take both.

Track	Start with	What you do first
Analog	AD101: Signals	Use notes and chords to learn to read signals in time and in fre-

Track	Start with	What you do first
		quency, and to see what a filter does to them
Digital	DD101: Intro to Digital Logic	Build logic circuits in browser simulators, from a single NAND gate up to a full adder

Each track then runs in order, AD101 → AD102 → AD103 → AD104 and DD101 → DD102 → DD103 → DD104, and the end of every course links to the next one. Every course that uses the tools reuses this workspace. Some early courses run in a web browser instead, and for those you won't need Docker until their labs say so.

A few links are worth keeping to hand:

- If you change the Docker tag or the PDK pin, run the smoke test again (Lab 03, Part 4).
- When something breaks, start with Troubleshooting.
- Every course is listed in one place, at edu.uoftasic.com.

Appendices

A. Troubleshooting

This page collects common setup problems in IC101 and the shared workspace, including the ones that come up when you load a later course with the workspace's `mod` command. Each entry gives the symptom you'll see, what causes it, and how to fix it.

I don't know how to open a terminal

Symptom: A page says "run this command" and you're not sure where to type it.
Cause: Every lab assumes you already have a terminal window open to type commands into.
Fix: Lab 01, Part 1 walks you through opening one, with steps for each OS.

git: command not found

Symptom: Running `git clone ...` prints `git: command not found` (on Windows, `'git' is not recognized...`) instead of downloading anything.
Cause: git isn't installed yet, or, on Windows, the terminal was already open when you installed it.
Fix: Follow Lab 01, Part 2, then open a new terminal window and try again. On Windows you need the new window, because a terminal that was already open won't pick up a program you installed after it started.

Docker won't start or hello-world fails

Symptom: Docker Desktop never shows **Engine running**, or `docker run --rm hello-world` prints an error instead of `Hello from Docker!`.
Cause: Docker hasn't finished starting, or a one-time OS setting hasn't been applied: WSL2 on Windows, or `docker` group membership on Linux.
Fix:

- Wait until Docker Desktop shows **Engine running** before you run any `docker` command. It can take a minute to get there after you open it.
- On Windows, make sure WSL2 is installed and selected as Docker's backend. See What Docker needs.
- On Linux, check that your user is in the `docker` group: run `groups` and look for `docker` in the output. If you've only just added yourself to the group, log out and back in so that it takes effect.
- Free up disk space. When the disk is full, large image pulls fail without a clear error, so a full disk is easy to miss.

permission denied on a docker command (Linux)

Symptom: A `docker` command fails with a message containing `permission denied`.
Cause: Your Linux user isn't in the `docker` group yet, so it can't talk to Docker without `sudo`.
Fix:

```
sudo usermod -aG docker $USER
```

Then log out and log back in (group changes don't apply to a terminal that's already open), and run the command again.

Port 80 already in use

Symptom: `start_vnc` fails, or the browser won't load `http://localhost/`, because another program on your computer is already using port 80.

Cause: Only one program can listen on a port at a time, and something else, often another web server running on your computer, started listening on port 80 first.

Fix: Start the workbench on a different host port instead:

```
HOST_PORT=8080 ./scripts/start_vnc.sh
```

Then open `http://localhost:8080/`.

Container starts but the browser page is blank

Symptom: `start_vnc` finishes without errors, but `http://localhost/` shows a blank page or won't connect.

Cause: The script exits before the desktop inside the container has finished starting, so for a short while there's nothing for the browser to show.

Fix:

- Wait about 30–60 s after the first pull or start for the services to come up, then reload the page.
- Run `docker ps` and check that it lists a container named `asic-edu-osic`.
- Read that container's logs with `docker logs asic-edu-osic`.
- Check that the address uses the port you actually set: `80` by default, or your `HOST_PORT` if you set one.

.designinit: No such file or directory

Symptom: Loading the environment with `. /foss/designs/common/.designinit` prints `bash: /foss/designs/common/.designinit: No such file or directory`.

Cause: Your `workspace` folder isn't mounted at `/foss/designs`, so the container has only an empty folder there. That happens when the container wasn't started with `start_vnc` from your `workspace` clone: for example, when someone started it with a plain `docker run` of the `hpretl/iic-osic-tools` image.

Fix:

1. In the desktop terminal, run `ls /foss/designs/`. If the workspace is mounted, the list includes `common`, `modules` and `scripts`, among others. If it lists nothing, the workspace isn't mounted.
2. If you haven't cloned the `workspace` repo yet, do Lab 03 from the start. If you have, start the workbench again from your clone, as in Lab 03, Part 2: `./scripts/start_vnc.sh` on macOS or Linux, `scripts\start_vnc.bat` on Windows.

mod: command not found

Symptom: A `mod` command prints `bash: mod: command not found`.

Cause: `mod` is defined in `/foss/designs/common/.designinit`, and this terminal hasn't loaded that file yet.

Fix: Load it, then run your `mod` command again:

```
. /foss/designs/common/.designinit
```

If that prints `No such file or directory`, see .designinit: No such file or directory.

Smoke test: tool or helper not found

Symptom: `smoke_test.sh` prints `FAIL` for a command (for example `ngspice` or `magic`) or for one of the module helper lines (`mod`, `add_module.sh`, `ic101_setup`) instead of `OK`.

Cause: The smoke test loads `common/.designinit` itself before it checks anything, so a terminal that hasn't loaded the file can't make a check fail. The most it can cause is the `WARN  your shell started on PDK ...` line described in Smoke test: PDK is not sky130A. A `FAIL` on a module helper line means the script couldn't find your workspace at `/foss/designs`, where those checks look for `scripts/add_module.sh` and `modules/ic101_setup`. A `FAIL` on a tool means the tool isn't on the `PATH`, which the image sets up in every terminal, so the container is probably not running the image the scripts expect.

Fix: Loading the environment by hand is still a quick first check, because it fails with `No such file or directory` when the workspace isn't mounted:

```
. /foss/designs/common/.designinit
```

Then check that the workspace is mounted (`ls /foss/designs/scripts` should list files) and run `/foss/designs/scripts/smoke_test.sh` again. If that folder is empty or missing, see .designinit: No such file or directory. If tools are still missing after that, your image may not be the one the scripts expect, so re-pull the tag they use (`DOCKER_TAG`, which defaults to `2026.08`).

Smoke test: PDK is not sky130A

Symptom: The smoke test's PDK line starts with `FAIL  PDK is` or `FAIL  PDKPATH is`, or else the checks pass but one line reads `WARN  your shell started on PDK ...`.

Cause: The PDK is there, since SKY130 ships inside the image at `/foss/pdks/sky130A`, but the image starts on a different PDK, `ihp-sg13g2`, so `PDKPATH` and the ngspice and KLayout search paths all point at that one instead. The `WARN` means the smoke test corrected the environment for its own run, but any tools you start from that terminal will still use the wrong PDK.

Fix: Load the workspace environment, which switches all of those variables over to SKY130, and then run the smoke test again:

```
. /foss/designs/.designinit
/foss/designs/scripts/smoke_test.sh
```

All `/foss/designs/.designinit` does is load `/foss/designs/common/.designinit`, so you can use either path.

If you want to switch PDK by hand, the image has its own switcher, `sak-pdk-script.sh`, and you must **source** it, with a leading `.`:

```
. sak-pdk-script.sh sky130A
```

If you run it as a plain command instead, it prints the settings but changes nothing in your shell, which looks a lot like it worked. Its usage message calls it `sak-pdk`, but there's no command by that name on the `PATH`, so type the full `sak-pdk-script.sh`.

mod: no such module

Symptom: `mod <course>` prints `mod: no such module: <course>`, followed by `Available modules:` and a list of the modules you do have.

Cause: There's no folder for that course under `/foss/designs/modules/` yet. Usually that's because you ran `mod <course>` before `mod add <course>`, or because the name is misspelled.

Fix: Check the name against the list it printed. If the course isn't in the list, add it with `mod add <course>`, using the name your course's Lab 01 gives. `mod add` clones the course and moves you into its folder, and from then on `mod <course>` works.

mod add asks for a GitHub username

Symptom: `mod add <course>` stops at `Username for 'https://github.com':` and waits, or it fails with `fatal: could not read Username for 'https://github.com'`, followed by `add_module: clone failed for <course>`.

Cause: GitHub wouldn't let git read `github.com/uoftasic/<course>` without logging in. `mod add` is meant to work without a login, so either the course name is misspelled or that course's repo isn't published yet.

Fix:

- Don't type a username or password. If git is waiting at the prompt, press **Ctrl+C** to stop it. Stopping it there leaves nothing behind, so you can run `mod add` again straight away.
- Check the spelling of the course name and run `mod add <course>` again.
- If the name is right, open `https://github.com/uoftasic/<course>` in your browser. A 404 page there means the repo isn't published yet. Whether or not you see a 404, ask on the team Discord, because only the course team can fix it.

Copy and paste into noVNC

Symptom: `Ctrl+V` on your laptop's keyboard doesn't paste into the noVNC desktop.

Cause: noVNC streams a remote desktop into your browser, and your laptop's clipboard isn't shared with that desktop automatically.

Fix: Press **Ctrl+Shift+V** instead of `Ctrl+V` to paste into the desktop, or use the clipboard icon in the noVNC sidebar.

Still stuck

If none of these fit, ask on the team Discord. Include your OS, your Docker version, the exact command you ran, and the part of the log that shows the error, but leave out passwords and other secrets.