Get Yourself Online

Friday, August 14, 2026 · interactive version at `aiclass.oaksedtech.com`

By next week you can take a change from idea to live on the internet without looking anything up.

Below is the list. Each row has two halves: the left is the thing to make work, the right is the thing to understand. If you can't answer the right, you haven't finished the row — no matter what's on your screen.

> **Nothing here explains how. Finding out is the assignment.** Use AI, videos, docs, each other — there's no cheating on the how. You're judged on whether it works, whether you can explain it, and how you got yourself unstuck.

The list

Accounts

Figure out	Understand it when you can answer
Get a GitHub account, with your real name and a photo on the profile	Why a stranger reading your profile is the point of it
Apply for the GitHub Student Pack	What it gives you that you'd otherwise pay for
Launch a Codespace, use its terminal, and stop it when you're done	What a Codespace actually is, and what's still running after you close the tab
Find the core-hours meter	Why a 2-core machine burns your quota twice as fast as clock time

The Git loop

Figure out	Understand it when you can answer
Make a repo, and get it open in a Codespace	The difference between Git and GitHub
Make a branch	Why you don't just edit `main`
Commit a change	Why a commit is not a save, and what a good message says
Push it	Why committing ten times and pushing once is normal
Open a pull request, and merge it	What a PR is for if you could just merge it yourself
Pull the merged change back down	Where your copy and GitHub's copy disagree, and what happens if you ignore it

Run that loop start to finish, twice, without notes. That's the bar.

Live on the internet

Figure out	Understand it when you can answer
Get something — anything, a heading will do — live at a public URL	Why pushing isn't deploying
Find out what happens to your URL when you push again	What "deploy" means in the thing you chose
Work out what the three layers of a web page are	Which layer a visitor can read, and how you know
Write a `.gitignore`, working, **before** your first push	What it does and does not protect you from

Secrets

Everything you commit is permanent, and on a public repo it's instantly public. Bots scrape new commits within seconds of a push. API keys, tokens, passwords, `.env` files: never.

Figure out	Understand it when you can answer
Where a secret key is safe to live, and where it isn't	Why "it's in the JavaScript" means "it's public"
What you're supposed to do if you commit a key by accident	Why the answer is not "delete it in the next commit"

Hand in

1	Your GitHub profile
2	A pull request you opened and merged
3	Your live URL
4	What broke, and how you got unstuck

> **DONE WHEN**
> All four handed in, and you can walk through the Git loop out loud with the screen off.

What you're actually graded on

Three things, and only three:

1	**Engagement** — Mr. Smith's call, from what he sees in the room. If you're not participating, he'll notice. That's the whole mechanism
2	**The blank paper quiz** — Tuesday, next week
3	**The portfolio** — graded when it's finished, end of this month

Everything above is how you get ready for those. Nothing on the list is worth marks on its own — it's worth marks because you can't do 2 or 3 without it.

Tuesday — blank paper quiz

You get a blank sheet with no questions on it, and write down everything you know about this material. That's the whole test, and the right-hand column of the list above is where most of the marks are.

What the page looks like	What it says about you
The loop end to end, the vocabulary in your own words, why each step exists, and the specific things that broke on you	You know this. Nothing else to prove.
The foundations are there — Git vs GitHub, commit vs push, what a branch is for — but holes where it gets specific	You've got the shape, not the detail. The holes are your homework.
Scattered words, or a page that could have been written before the assignment	Nothing's landed yet. We start over, and that's fine — but we start over.

You can't cram recall, only rehearse it: sit down with a blank page now and do the quiz to yourself. Whatever you can't produce is your study list. Re-reading your notes will feel like it's working and won't be.

Limits worth knowing before you assume you're stuck

- **GitHub Pages needs a public repo** on the free plan. Other hosts don't.
- Your profile needs to look like a real person for the Student Pack. Nothing beyond that belongs in a repo — not your address, not your schedule, not your phone number.

When you're stuck

That's not the assignment going wrong. That's the assignment.

Before you ask a person: search the exact error text, read the real documentation, ask an AI to explain it three different ways until one lands, find a video. Note where the answer came from — hand-in 4 is that note.

PROMPT — BUILD YOUR OWN WAY IN

```
I need to learn X. Don't explain it yet.

Find me the three best resources — the official docs page, one video, one worked example.
Tell me what order to use them in and what I should be able to do after each. Then quiz me
on it and tell me what I still don't have.
```

Practice

None of this is handed in. It's how you find out whether the right-hand column actually landed. The interactive version — flashcards, a commit-message critic, a Git loop you can break safely — is at `aiclass.oaksedtech.com/assignments/02-practice`. Everything below is fair game on Tuesday.

Vocabulary — know these cold

Term	What it means	Like
Repository	One project's folder plus its entire history	A Doc where every version is saved
Commit	One saved snapshot, with a message explaining why	Hitting save and writing a note
Push	Upload your commits to GitHub	Syncing your photos to the cloud
Pull	Download commits other people made	Refreshing for everyone's latest
Clone	Make your own local copy of a repo	Downloading the whole album
Fork	Copy someone else's repo into your account	Making your own remix
Branch	A parallel timeline where you can experiment safely	A "what if" copy
Merge	Fold a branch's changes back into main	Accepting the "what if"
Pull Request	A formal request to merge, with room for review	"Look this over first"
main	The default branch. The real version	The master copy
Deploy	Put the code on a live public URL	Printing the book
Worktree	Two branches checked out in two folders at once	Two desks, one filing cabinet

Quiz yourself

Answer out loud before you check. Answers at the bottom.

1. Git vs GitHub?
2. Commit vs push?
3. Why branch instead of editing main?
4. What's a pull request for, if you could just merge?
5. Why isn't pushing the same as deploying?
6. You need a secret API key for a weather service. Where does it go, and why?

PROMPT — MAKE IT QUIZ YOU PROPERLY

```
Quiz me on: Git vs GitHub, the commit/push/pull/branch/merge/PR workflow, worktrees,
deploying, Codespaces core-hours, installing an AI coding agent, and the three layers of the
web stack.

One at a time, wait for my answer. Mix definitions, "what happens if...", and "here's an
error, what did I do wrong." Don't accept vague answers. Score me after 12 and tell me what
to review.
```

The portfolio — due end of this month

Not this week's job. The list comes first, and there's no point styling a page you can't deploy yet. But this is one of the three things you're graded on, so the sooner you start thinking about it the less it costs you later.

That live page becomes your portfolio, and you'll add your work to it every unit. Nobody's handing you a template — working out what makes a portfolio good, and what makes yours worth looking at, is the assignment.

Which makes this the place to plan before you build. **Every project this year starts the same way — Goal, Scaffold, Evaluate — and you write all three down before you make anything.** Write it in a file in the repo, so it's committed and you can hold yourself to it later.

This isn't paperwork. It's the handoff. An AI will build whatever you ask, instantly, and has no idea what you actually want or when it's finished — so these three, once you've got them aligned, are exactly what you hand it so it can go and work without you standing over it. Goal says what to build. Scaffold says in what order. Evaluate is the test it has to pass before either of you calls it done. Get all three right and you're driving. Leave one vague and you're just accepting whatever came out.

PLAN · YOUR PORTFOLIO
01 Goal — what you are making, and why it is worth making Who is this for, and what do you want them to think after ten seconds on it? One sentence, specific. A portfolio for everyone is a portfolio for nobody, and "to show my work" isn't a goal — every portfolio shows work. What's yours for?
02 Scaffold — how you get there: the pieces, and the order The pieces, in the order you'll build them, with what depends on what. What's the smallest version that could be live today? What are you deliberately not building yet? If a step can't be done in one sitting, it's not a step yet — break it down until it is.
03 Evaluate — how we will know it is good How will you know it's actually good — written down **now**, before you build, while you can still be honest about it. Not "it looks nice." Something specific enough that someone else could check it for you and get the same answer you would. A plan with no standard in it can't fail, and a thing that can't fail can never be finished either — and an AI with no standard will cheerfully tell you it's done.

Do this before you write a line of it, or you'll end up with whatever the AI defaulted to. That's the point of the prompt below — same trick as the one above, make it find you the map instead of walking it for you.

PROMPT — PLAN THE PORTFOLIO BEFORE YOU BUILD IT

```
I'm building a portfolio site that I'll add school and personal projects to all year.
Don't build it, and don't give me code or a template yet.

First: find me four or five real portfolios worth stealing ideas from — a mix of students,
designers, and developers — and for each one tell me what it does well and what it does
badly.

Then ask me the questions you'd need answered to plan mine: who's meant to read it, what it's
for, what I actually want out of it. One at a time, and don't move on until my answer is
specific.

Then help me write three things down, and push back where I'm being vague:
  GOAL — who it's for and what they should think after ten seconds
  SCAFFOLD — the pieces in build order, what depends on what, what I'm not building yet
  EVALUATE — how we'll tell whether the finished thing is any good

Tell me which parts you're least sure about and why.
```

Then, and only then, hand it over and let it work:

PROMPT — HAND IT THE PLAN AND LET IT RUN

```
Here is my plan. Goal, Scaffold, Evaluate:

[paste your three]

Before you build anything: tell me where this plan is ambiguous, and where two people could
read it differently. Ask me about anything you'd otherwise have to guess. I'd rather answer
five questions now than get the wrong thing.

Once it's clear, work through the scaffold in order and build it. Then — separately, and
strictly against my Evaluate section, not against your own taste — go back over what you
built and tell me where it falls short of the standard I wrote. Be hard on it.
```

If it built the wrong thing, that's information: your plan let it. Fix the plan, not just the page.

> **DONE WHEN**
> It's live, something of yours is on it, and you can say why it looks the way it does — and "because the AI made it that way" isn't a reason.

Extra — get an agent in your terminal

Only if you already have a paid Claude or ChatGPT plan — both agents need one, and neither free tier includes them.

Get it running **inside your Codespace**, not on your laptop. Things to look up, in order:

- The official install command for Claude Code, or for the Codex CLI
- How to authenticate it from inside a container. This is the awkward part. The first link you're offered won't work, and working out why is the exercise
- How to get it to read your whole project before you ask it for anything

- How to stop your login vanishing every time you create a new Codespace

> **DONE WHEN**
> It has explained your own repo back to you in plain English, and you've had it change something that you then reviewed line by line.

Answers

1. Git is the program taking snapshots on your computer. GitHub hosts copies so others can collaborate. You can use Git with no GitHub at all.

2. A commit saves locally. A push uploads to GitHub. You can commit ten times and push once.

3. `main` should always work. A branch lets you break things without risking the live version — if it fails, delete it and nothing was ever at risk.

4. Review, and the record of why a change happened, kept with the code.

5. Pushing stores your code. Deploying serves it at a public URL.

6. On the server, behind an API layer. Anything shipped to the browser is public — the frontend can be read by anyone with Ctrl/Cmd+U, so a key there is a key given away.

The interactive version of the list is at `aiclass.oaksedtech.com`, and the sandbox, flashcards and self-checks live one page over under Practice. This sheet is the same assignment on paper.