

Claude Code Getting Started Guide for Aspiring Web Developers

A Practical Guide for Building Prototype Websites

Table of Contents

1. [Choosing Your Development Machine](#)
 2. [Installing Homebrew](#)
 3. [Setting Up Git and GitHub](#)
 4. [Installing a Better Terminal](#)
 5. [Installing a Code Editor](#)
 6. [Installing Claude Code Prerequisites](#)
 7. [Installing Claude Code](#)
 8. [Installing Docker](#)
 9. [Creating Your First Project](#)
 10. [Using Claude Code Effectively](#)
 11. [Planning Mode and One-Shot Builds](#)
 12. [Recommended Free Tools](#)
 13. [Example Prompts to Try](#)
 14. [Working with Your Tech Partner](#)
-

1. Choosing Your Development Machine

Recommendation: Use the MacBook Air

Why Mac wins for web development:

Factor	MacBook Air	Windows Laptop
Terminal/CLI	Native Unix (bash/zsh)	Needs WSL2 setup
Node.js/npm	Works seamlessly	Extra configuration
Docker	Native support	Requires Hyper-V/WSL2
Git	Pre-installed	Needs installation
Path issues	Rare	Common with spaces/backslashes
Industry standard	Most web devs use Mac	Less common

The 250GB SSD concern: This is manageable. A typical web project is under 500MB. You can store 50+ projects easily. Use GitHub as your backup/archive.

Tips to manage storage:

- Delete `node_modules` folders from inactive projects (run `npm install` to restore)
- Use `npx` to run tools without installing globally
- Clean Docker images periodically: `docker system prune`

2. Installing Homebrew

Homebrew is the standard package manager for macOS. It lets you install developer tools with simple terminal commands instead of hunting for download links.

Install Homebrew

Open Terminal and run:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install"
```

The installer will:

1. Ask for your password (this is your Mac login password)
2. Show you what it will install
3. Press Enter to continue

About the password prompt: This is the only time in this entire guide you'll be asked for your

password. Homebrew needs it once during installation to create its directory (`/opt/homebrew`). After that, all `brew install` commands run without elevated privileges.

Good practice: Avoid using `sudo` (administrator/root access) on a development machine whenever possible. It's a security risk—if you accidentally run malicious code with `sudo`, it can damage your entire system. Homebrew is specifically designed to work without `sudo`, which is one reason it's the preferred package manager for developers.

Important: After installation, the installer will show you two commands to run. They look like this (copy the exact commands shown in YOUR terminal):

```
echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Verify Homebrew is working

```
brew --version
```

You should see something like `Homebrew 4.x.x` .

Homebrew basics

```
# Install a program
brew install <package-name>

# Install a GUI application (like VS Code, Docker, etc.)
brew install --cask <app-name>

# Update Homebrew itself
brew update

# Upgrade all installed packages
brew upgrade

# See what you have installed
brew list

# Search for a package
brew search <keyword>
```

Why Homebrew is great

- **One command installs:** No downloading .dmg files and dragging to Applications
- **Easy updates:** `brew upgrade` updates everything at once
- **Clean uninstalls:** `brew uninstall <package>` removes everything cleanly
- **Consistent:** Every tool installs the same way

Quick install (for the impatient)

Once Homebrew is set up, you can install most tools in one go:

```
# Install all the GUI apps at once
brew install --cask iterm2 visual-studio-code docker macdown

# Install Git
brew install git nvm
```

Then follow the individual sections below for configuration steps.

3. Setting Up Git and GitHub

Git tracks your code changes. GitHub stores your code online so your tech partner can access it.

Step 1: Install Git via Homebrew

macOS comes with Git, but it's often outdated. Install the latest version:

```
brew install git
```

Verify installation: `bash git --version`

Step 2: Create a GitHub account

1. Go to github.com
2. Click "Sign up"
3. Use your student email (you get free GitHub Pro as a college student!)
4. Verify your email

Step 3: Configure Git with your identity

```
git config --global user.name "Your Name"
git config --global user.email "your-email@myuniversity.edu"
```

Step 4: Set up SSH keys (so you don't type passwords constantly)

```
# Generate a new SSH key
ssh-keygen -t ed25519 -C "your-email@myuniversity.edu"

# Press Enter to accept default location
# Enter a passphrase (or leave blank for no passphrase)

# Start the SSH agent
eval "$(ssh-agent -s)"

# Add your key to the agent
ssh-add ~/.ssh/id_ed25519

# Copy your public key to clipboard
pbcopy < ~/.ssh/id_ed25519.pub
```

Now add the key to GitHub:

1. Go to GitHub → Settings → SSH and GPG keys
2. Click "New SSH key"
3. Paste your key and save

Step 5: Test your connection

```
ssh -T git@github.com
```

You should see: "Hi username! You've successfully authenticated..."

////////////////////////////////////

4. Installing a Better Terminal

The default Terminal.app works, but **iTerm2** is much better.

Install iTerm2

```
brew install --cask iterm2
```

That's it! Open iTerm2 from your Applications folder or Spotlight (Cmd + Space, type "iTerm").

Quick iTerm2 setup

1. Open iTerm2 → Preferences (Cmd + ,)
2. Profiles → Colors → Color Presets → Choose "Solarized Dark" or "Tango Dark"
3. Profiles → Text → Font → Change to 14pt for readability

Useful iTerm2 features

- **Split panes:** Cmd + D (vertical) or Cmd + Shift + D (horizontal)
 - **New tab:** Cmd + T
 - **Search:** Cmd + F
 - **Paste history:** Cmd + Shift + H
- ////////////////////////////////////

5. Installing a Code Editor

While Claude Code works in the terminal, you'll want an editor to view and occasionally tweak files.

Recommendation: Visual Studio Code (VS Code)

It's free, visual, beginner-friendly, and has great Git integration.

Install VS Code

```
brew install --cask visual-studio-code
```

Open VS Code from Applications or Spotlight (Cmd + Space, type "Visual Studio Code").

Essential extensions for web development

Open VS Code, click the Extensions icon (4 squares) on the left, and install:

1. **Prettier** - Auto-formats your code
2. **Live Server** - Preview websites instantly

3. **GitLens** - See who changed what
4. **Auto Rename Tag** - Rename HTML tags in pairs

Enable opening VS Code from terminal

1. Open VS Code
2. Press Cmd + Shift + P
3. Type "shell command"
4. Select "Install 'code' command in PATH"

Now you can type `code .` in any folder to open it in VS Code.

6. Installing Claude Code Prerequisites

Claude Code requires Node.js version 18 or higher.

Install Node.js using nvm (Node Version Manager)

Using nvm lets you easily switch Node versions. Install it via Homebrew:

```
# Install nvm
brew install nvm

# Create nvm's working directory
mkdir ~/.nvm
```

Add nvm to your shell profile. Run this command:

```
echo 'export NVM_DIR="$HOME/.nvm"
[ -s "/opt/homebrew/opt/nvm/nvm.sh" ] && \. "/opt/homebrew/opt/nvm/nvm.sh"
[ -s "/opt/homebrew/opt/nvm/etc/bash_completion.d/nvm" ] && \. "/opt/homebr
```

Now restart iTerm2 (close and reopen), then install Node.js:

```
# Verify nvm is working
nvm --version

# Install the latest LTS (Long Term Support) Node.js
nvm install --lts

# Verify Node.js installation
node --version    # Should show v20.x.x or higher
npm --version     # Should show 10.x.x or higher
```

7. Installing Claude Code

Step 1: Install Claude Code globally

```
npm install -g @anthropic-ai/claude-code
```

Step 2: Authenticate

```
claude
```

The first time you run this, it will:

1. Open your browser
2. Ask you to log in to your Anthropic account (create one at console.anthropic.com if needed)
3. Authorize Claude Code

Step 3: Verify it works

```
claude --version
```

Important: API usage costs

Claude Code uses the Anthropic API, which has usage costs. As a student building prototypes:

- Start with the free tier if available
 - Monitor your usage at console.anthropic.com
 - Consider setting spending limits
-

8. Installing Docker

Docker lets you run databases, servers, and other infrastructure in isolated "containers." Claude Code can create and manage Docker containers for you.

Install Docker Desktop

```
brew install --cask docker
```

After installation:

1. Open Docker Desktop from Applications or Spotlight
2. Complete the setup wizard (you can skip the tutorial)
3. Docker will start running in the background

Verify Docker is running

The Docker whale icon should appear in your menu bar. In terminal:

```
docker --version  
docker ps      # Should show empty list (no containers yet)
```

Keep Docker running

When working with Claude Code on projects that need databases:

- **Always start Docker Desktop first** before running Claude Code
- Claude Code will automatically create containers when needed
- Docker must be running or database commands will fail

Common Docker commands

```
# See running containers
docker ps

# See all containers (including stopped)
docker ps -a

# Stop all running containers
docker stop $(docker ps -q)

# Clean up unused resources (reclaim disk space)
docker system prune

# Remove all stopped containers and unused images (more aggressive)
docker system prune -a
```

Example: Claude Code with Docker

When you tell Claude Code something like "create a web app with a PostgreSQL database," it will:

1. Create a `docker-compose.yml` file
2. Run `docker-compose up` to start the database
3. Configure your app to connect to it

You just need Docker Desktop running—Claude Code handles the rest.

////////////////////////////////////

9. Creating Your First Project

Create a projects directory

Use `term2` or `Terminal` to run the following commands.

```
# Create a folder for all your projects
mkdir -p ~/Projects

# In case you are wondering, the tilda ~ means your home directory, which y
# if you type cd without any directory specified.

# Navigate to it using change directory (cd)
cd ~/Projects

# Verify the full path of the present working directory (pwd)
pwd
```

Create a new project

```
# Create a project folder
mkdir my-first-website

# Change directory to your new project folder
cd my-first-website

# Initialize Git
git init

# Create a basic README
echo "# My First Website" > README.md

# Make your first commit
git add .
git commit -m "Initial commit"
```

Connect to GitHub

1. Go to GitHub and click "New repository"
2. Name it `my-first-website`
3. Keep it **Private** (for client work)
4. **Don't** initialize with README (you already have one)
5. Click "Create repository"

GitHub will show you commands. Run the "push an existing repository" ones:

```
git remote add origin git@github.com:YOUR-USERNAME/my-first-website.git
git branch -M main
git push -u origin main
```

Start Claude Code in your project

```
# Make sure you're in the project directory
cd ~/Projects/my-first-website

# Start Claude Code
claude
```

10. Using Claude Code Effectively

Basic interaction

Once Claude Code is running, you type natural language requests:

```
> Create a simple landing page for a coffee shop called "Boiler Brew"
```

Claude Code will:

1. Understand your request
2. Create the necessary files
3. Show you what it's doing
4. Ask questions if needed

Essential commands

Command	What it does
<code>/help</code>	Show available commands
<code>/clear</code>	Clear conversation history
<code>/cost</code>	Show API usage costs
<code>Ctrl+C</code>	Cancel current operation
<code>Ctrl+D</code> or <code>/exit</code>	Exit Claude Code
<code>Shift+Return</code> or <code>Shiift+Enter</code>	Next line of prompt entry
<code>Return</code> or <code>Enter</code>	Accept prompt and begin processing

Tips for good prompts

Be specific about what you want:

Bad: "Make a website"

Good: "Create a single-page portfolio website for a photographer with a header gallery section showing 6 images in a grid, an about section, and a contact form. Use modern CSS with a dark theme."

Mention technologies if you have preferences:

"Create a landing page using HTML, Tailwind CSS, and vanilla JavaScript. No frameworks needed."

Describe the visual style:

"Use a minimalist design with lots of white space, a sans-serif font like Inter, and a blue (#0066CC) accent color."

11. Planning Mode and One-Shot Builds

The Power of Planning

Good planning leads to cleaner builds. Claude Code has a **Planning Mode** that helps you think through the project before writing code.

It takes time up front to think through and design what you want to have Claude build for you.

This up-front time investment pays dividends, however. Every 10 minutes invested in forethought and planning prior to the build will save hours of rework, testing and tedious debugging later.

Using Planning Mode

Start your session by asking Claude Code to plan:

```
> I want to build a portfolio website for a freelance graphic designer.
Before we build anything, let's create a detailed plan. Please enter
planning mode and help me think through:
- What pages we need
- What components each page should have
- What the file structure should look like
- What technologies to use
Save the plan to CLAUDE.md when we're done.
```

Claude Code will:

1. Enter planning mode
2. Explore options with you
3. Ask clarifying questions
4. Create a comprehensive plan
5. Save it to `CLAUDE.md`

The CLAUDE.md file

This is your project's "memory." It contains:

- Project overview
- Technical decisions
- File structure
- Implementation plan
- Notes for future sessions

Always create this file at the start of a project. Claude Code reads it automatically when you start a session in that folder.

Example CLAUDE.md structure

Project: Designer Portfolio

Overview

Portfolio website for Jane Doe, freelance graphic designer.

Tech Stack

- HTML5
- Tailwind CSS (via CDN)
- Vanilla JavaScript
- No backend needed (static site)

Pages

1. Home - Hero with name/tagline, featured work (3 items)
2. Portfolio - Grid gallery with category filters
3. About - Bio, skills, photo
4. Contact - Form (using Formspree for handling)

File Structure

```
/
├── index.html
├── portfolio.html
├── about.html
├── contact.html
├── css/
│   └── custom.css
├── js/
│   └── main.js
└── images/
    └── (placeholder images for now)
```

Design Notes

- Color palette: Navy (#1a365d), Gold (#d69e2e), White
- Font: Playfair Display for headings, Inter for body
- Mobile-first responsive design

Implementation Order

1. Set up file structure
2. Create shared header/footer
3. Build home page
4. Build portfolio page with filter
5. Build about page
6. Build contact page with form
7. Add responsive adjustments
8. Test all pages

One-Shot Build Strategy

After planning, aim for a **one-shot build**—giving Claude Code enough context to build everything correctly the first time.

```
> Now let's build this. Using the plan in CLAUDE.md, create the complete portfolio website. Build all pages, make sure navigation works between them, and include placeholder content. I want to be able to open index.html in a browser and have a fully functional site.
```

Why one-shot builds are better:

- Less back-and-forth means lower API costs
- Fewer iterations mean fewer bugs
- Complete context leads to more consistent code
- You learn to communicate requirements clearly

Tips for successful one-shot builds:

1. Spend 30% of your time on the plan
2. Be very specific about requirements
3. Include examples or references when possible
4. Describe edge cases upfront
5. Review the plan before building

In the early stages of building and iteration, it may be better, faster and more efficient to go back and update your plan, delete all of the previous work and do another one-shot build from scratch, with your updated plan.

Minor tweaks like changing colors, updating text or even adding new pages can be done as modifications to the existing build; however, major revisions of existing pages and functionality may be better done as a new one-shot build. Remember to save your color, text and page tweaks back to your plan, in case you decide to do another one-shot build later.

Managing Multiple Builds (Beginner Workflow)

When you're learning Claude Code, don't worry about Git right away. Use simple numbered directories for your early experiments:

```
~/Projects/client-portfolio/
├── CLAUDE.md           # Your master plan (keep at top level)
├── builds/
│   ├── v0.1/          # First attempt - learning the ropes
│   ├── v0.2/          # Better prompts, improved design
│   └── v0.3/          # Almost there...
└── v1.0/              # Ready for client! Initialize Git here.
```

Why this works:

1. **Focus on one skill at a time** - Learn Claude Code first, Git later
2. **Easy comparison** - Open two builds side-by-side in VS Code or Finder
3. **No fear of breaking things** - Can't mess up a Git repo if there isn't one
4. **Keep your experiments** - Failed attempts are learning opportunities, not clutter

The workflow:

1. Create your project folder and write your plan in `CLAUDE.md`
2. Set up and enter your first build directory:

```
# Create the builds folder and v0.1
mkdir -p ~/Projects/client-portfolio/builds/v0.1

# Change into the build directory
cd ~/Projects/client-portfolio/builds/v0.1

# Copy your plan here so Claude can read it
cp ../../CLAUDE.md .
```

3. Start Claude Code and do your one-shot build:

```
# Make sure you're in the build directory
pwd

# Should show: /Users/yourname/Projects/client-portfolio/builds/v0.1

# Start Claude Code
claude
```

Important: Claude Code creates files in your current directory. Always `cd` into the build folder before running `claude`.

4. Review the result. Update your master plan (`../../CLAUDE.md`) with lessons learned
5. Create the next build and try again:

```
# Go back to project folder
cd ~/Projects/client-portfolio

# Create next build directory
mkdir builds/v0.2
cd builds/v0.2

# Copy your updated plan
cp ../../CLAUDE.md .

# Run Claude Code again
claude
```

6. Repeat until you have something worth showing a client
7. When you're happy, promote your best build to v1.0:

```
# Go back to project folder
cd ~/Projects/client-portfolio

# Copy your best build to v1.0 (let's say v0.3 was the winner)
# NOTE: The cp command below creates the v1.0 directory automatically
# Don't pre-create the v1.0 directory using mkdir, otherwise the v0.3 w
# the pre-existing v1.0 directory:
cp -r builds/v0.3 v1.0

# The CLAUDE.md should already be in there from when you built it
```

8. **Now** initialize Git in `v1.0/` and push to GitHub

When to initialize Git (the v1.0 moment):

- The build is ready to show a client
- You're handing off to your tech partner for deployment
- You want to start tracking incremental changes (not wholesale rebuilds)

```
cd ~/Projects/client-portfolio/v1.0
git init
git add .
git commit -m "Initial v1.0 build - ready for deployment"

# Then connect to GitHub as shown in Section 9
```

After v1.0: Now Git makes sense. Use it to track changes, fixes, and enhancements. Your Git history will be clean and meaningful—showing intentional changes rather than your learning journey.

Pro tip: Keep your `builds/` folder around even after v1.0. You might want to reference an old approach, or your client might say "I liked that thing from the second version better."

Cleaning Up Old Builds

Once you're confident you no longer need your experimental builds, you can delete them to free up disk space.

When it's safe to clean up:

- The client has approved the final version
- The project has been deployed and is running smoothly
- You're certain you won't need to reference old approaches

Option 1: Use Finder (safest)

Simply drag the `builds` folder to the Trash. This way you can recover it if you change your mind (until you empty the Trash).

Option 2: Delete from the terminal

```
# From the project folder
cd ~/Projects/client-portfolio

# Delete all experimental builds at once
rm -rf builds/

# Or delete specific versions selectively
rm -rf builds/v0.1 builds/v0.2
```

Warning: `rm -rf` is permanent—there's no undo or Trash. Double-check you're deleting the

right folder before pressing Enter. If you're not 100% sure, use Finder instead.

12. Recommended Free Tools

Markdown Editor/Viewer

For editing CLAUDE.md and documentation:

MacDown (Free, Mac-only)

```
brew install --cask macdown
```

- Live preview as you type
- Simple and lightweight

Alternative: VS Code has built-in Markdown preview (Cmd + Shift + V)

Code Editor (Beyond VS Code)

If you find VS Code overwhelming:

Zed (Free, Mac-only)

```
brew install --cask zed
```

- Very fast and minimal
- Good for beginners
- Built-in AI features

Design/Prototyping

Figma (Free tier available)

- Browser-based, no install needed
- Great for planning website layouts visually
- Your designer clients might send you Figma files

Browser for Development

Chrome or Firefox Developer Edition

- Built-in DevTools (right-click → Inspect)
- Essential for debugging CSS and JavaScript

13. Example Prompts to Try

Starter project (try this first!)

```
Create a simple personal landing page with:  
- A centered hero section with my name "Your Name" and tagline "Engineering"  
- A brief about section  
- Links to GitHub, LinkedIn, and email  
- Use a clean, modern design with a dark theme  
- Make it responsive for mobile
```

Business landing page

```
Create a landing page for a local tutoring business called "Boilermaker Tut  
- Hero section with headline, subheadline, and a "Book a Session" button  
- Section listing 4 subjects offered (Math, Physics, Chemistry, CS)  
- Testimonials section with 3 fake student reviews  
- Contact section with address, phone, and email  
- Use Purdue's colors (Old Gold and Black) as the color scheme  
- Include smooth scroll navigation
```

Portfolio website

```
Build a 3-page portfolio website for a graphic designer:  
- Home page with hero, featured work preview (3 items), and call to action  
- Portfolio page with a grid of 9 project thumbnails with hover effects  
- Contact page with a styled form
```

```
Use placeholder images from picsum.photos. Make the design minimal  
and typography-focused.
```

Interactive component

Create an interactive pricing calculator for web design services:

- Checkboxes for features (Landing Page \$500, Additional Pages \$200 each, Contact Form \$150, Gallery \$300, E-commerce \$1000)
- Slider for number of additional pages (0-10)
- Live total that updates as options change
- "Get Quote" button that shows a summary

Make it look professional with good spacing and animations.

With Docker (database project)

Create a simple bookmark manager web app with:

- Form to add a bookmark (URL, title, tags)
- List view of all bookmarks with search/filter
- Ability to delete bookmarks
- Use Node.js with Express for the backend
- PostgreSQL database for storage (use Docker)
- Simple frontend with vanilla HTML/CSS/JS

Make sure to include the docker-compose.yml file and instructions to run everything.

14. Working with Your Offshore Tech Partner

Sharing access via GitHub

Add your offshore tech partner as a collaborator:

1. Go to your repository on GitHub
2. Settings → Collaborators
3. Add their GitHub username or email

Branch workflow

Keep your main branch stable:

```
# Create a new branch for each feature

git checkout -b feature/new-landing-page

# Do your work with Claude Code...

# Commit your changes

git add .
git commit -m "Add new landing page design"

# Push the branch

git push -u origin feature/new-landing-page

# Create a Pull Request on GitHub for your partner to review
```

Communication through code

Add notes in CLAUDE.md for your partner:

```
## Deployment Notes
- Site is static HTML/CSS/JS
- Can be deployed to Netlify, Vercel, or any static host
- No backend required
- Contact form uses Formspree (account: xyz@email.com)

## For Tech Partner
- Database credentials are in .env (not committed)
- Docker setup: run `docker-compose up` before starting
- See /docs folder for API documentation
```

.gitignore essentials

Create a `.gitignore` file to avoid committing unnecessary files:

```
# Ask Claude Code to create it
> Create a .gitignore file appropriate for a Node.js web project
```

Or create it manually:

```
node_modules/  
.env  
.DS_Store  
*.log  
dist/  
.cache/
```

Quick Start Checklist

- ☐ Choose MacBook Air as your development machine
- ☐ Install Homebrew
- ☐ Install Git (`brew install git`)
- ☐ Create GitHub account with Purdue email
- ☐ Set up Git config and SSH keys
- ☐ Install iTerm2 (`brew install --cask iterm2`)
- ☐ Install VS Code (`brew install --cask visual-studio-code`)
- ☐ Install VS Code extensions
- ☐ Install nvm (`brew install nvm`)
- ☐ Install Node.js (`nvm install --lts`)
- ☐ Install Claude Code (`npm install -g @anthropic-ai/claude-code`)
- ☐ Create Anthropic account for API access
- ☐ Install Docker Desktop (`brew install --cask docker`)
- ☐ Create your ~/Projects directory
- ☐ Create your first project folder
- ☐ Initialize Git and connect to GitHub
- ☐ Run Claude Code and try the starter prompt!

Getting Help

- **Claude Code issues:** Type `/help` or ask Claude Code itself
- **Git problems:** Ask Claude Code "I'm having a git issue where..."
- **General questions:** Claude Code can explain concepts too

Remember: Claude Code is your pair programmer. Describe what you want clearly, plan before you build, and don't be afraid to ask it questions. Good luck with your web prototyping business!

*Guide created for Engineering students getting started with Claude Code Last updated:
November 2025*