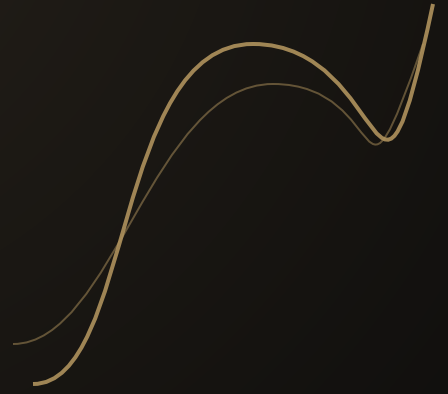




WEB DEVELOPMENT DIVISION

Frame 4 Eternity

Web Development Client Guide

How your website gets built — from first setup to live launch. A plain-English walkthrough of the tools, workflow, and safety nets behind every F4E web build.

Table of Contents

01	Introduction: What Is "Vibe Coding"?
02	The Tool Stack & Monthly Costs
03	Step 1 — Install Cursor (Your Code Editor)
04	Step 2 — Subscribe to an AI Coding Assistant
05	Step 3 — Install Git & Set Up GitHub
06	Step 4 — Set Up Firebase Hosting
07	Step 5 — Configure Your Cursor Workspace & Colors
08	Step 6 — The Vibe Coding Workflow
09	Step 7 — Essential Git Commands
10	Step 8 — Pre-Deploy Checklist
11	Step 9 — Working From Multiple Computers
12	Step 10 — Sharing Access With Clients
13	Step 11 — Reverting / Undoing Commits & Deploys
A	Appendix A — Mac vs Windows Command Reference
B	Appendix B — Troubleshooting Common Issues
C	Appendix C — Quick Reference Cheat Sheet

What Is "Vibe Coding"?

The philosophy behind how F4E builds websites.

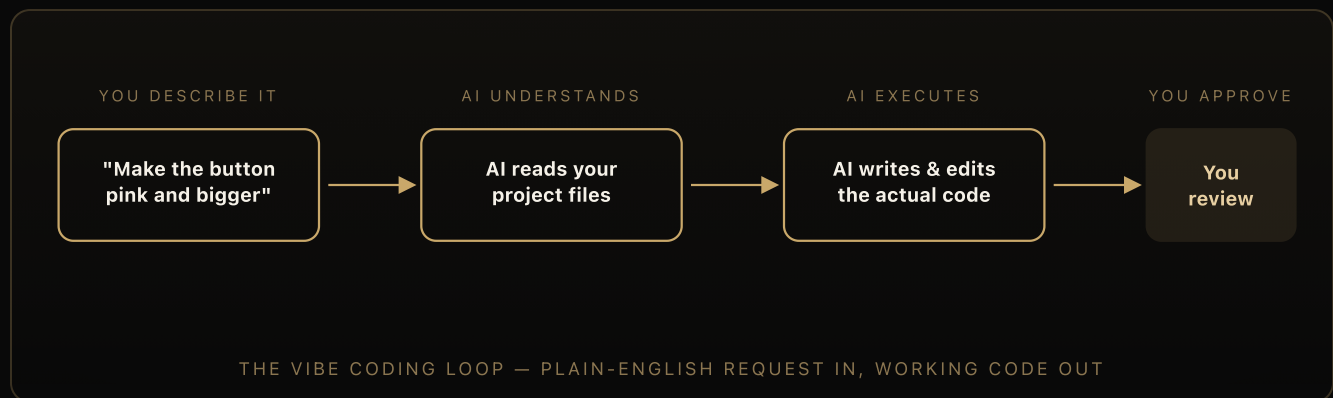
Welcome to the Frame 4 Eternity LLC web development workflow. This guide walks you through exactly how your website gets built, step by step — from the tools we install on day one to the moment your site goes live on the internet.

The approach we use is called "**vibe coding**." Here's what that means in plain English:

Instead of writing hundreds of lines of code by hand, we use AI-powered tools that let us *describe what we want in casual language* — "make the button pink," "add a contact form that emails the client," "change the font to something more elegant" — and the AI writes, edits, and executes the code on our behalf. You don't need to know how to code. You need to know what you want your website to look like and do.

This guide covers the full setup so that:

- **You (the client)** can make small edits yourself if you want to
- **We (F4E)** can build, deploy, and maintain your site efficiently
- **Both of us** can work from different computers without stepping on each other's toes



The Tool Stack & Monthly Costs

Here's the complete toolkit. Each tool costs about \$20/month (or is free). Here's what each one does and why we need it.

TOOL	MONTHLY COST	WHAT IT DOES	DO YOU NEED IT?
Cursor	\$20/mo (Pro) or Free (Hobby)	Your code editor — where you view, edit, and manage all website files. Think Microsoft Word for websites, but with AI built in.	Yes — main workspace
ChatGPT Plus (Codex)	\$20/mo	AI coding agent from OpenAI. You describe what you want in plain English; it writes and edits code. Terminal + web.	Pick one of two AI tools
Claude Pro (Claude Code)	\$20/mo (\$17/mo annual)	AI coding agent from Anthropic. Same concept — describe it, it executes. Runs in your terminal.	Pick one of two AI tools
GitHub	Free (public) / \$4/mo (private)	Your website's "save history." Every change is tracked — you can always go back to any previous version.	Yes — free tier works
Firebase	Free (Spark plan)	Your website's hosting — where the site "lives." Includes a free <code>.web.app</code> URL and SSL.	Yes — free tier covers most sites

TOTAL MONTHLY COST

~\$20–24/month (one AI tool + GitHub if private repos + Firebase free tier)

Which AI Tool Should You Pick?

FEATURE	CHATGPT PLUS (CODEX)	CLAUDE PRO (CLAUDE CODE)
Price	\$20/mo	\$20/mo (\$17/mo annual)
How it works	Terminal-based agent + cloud tasks	Terminal-based agent (local only)
Best for	Fire-and-forget — describe, walk away, review	Interactive editing — steer in real time
Runs your dev server?	No (cloud mode can't reach localhost)	Yes — starts local preview to verify
Model lock-in	OpenAI models only	Anthropic models only
Setup file	<code>agent.md</code>	<code>CLAUDE.md</code>
IDE integration	VS Code / Cursor extension	Runs in terminal (inside Cursor)

Our recommendation: New to this? Start with **ChatGPT Plus (Codex)** — most beginner-friendly for "describe it and forget it." Want more hands-on steering? **Claude Code** is excellent for interactive back-and-forth.

Both tools also work inside Cursor's built-in AI chat — you don't have to choose just one. Cursor can route between multiple AI models. The \$20/mo subscription gives you *more usage* and *terminal access* on top of what Cursor provides free.

Install Cursor

Your code editor — built on VS Code, with AI layered on top.

Download & Install

On Mac

1. Go to cursor.com/download
2. Apple Silicon (M1–M4) → **Apple Silicon (ARM64)**. Older Intel Macs → **x64**.
3. Open the downloaded `.dmg` and drag Cursor to Applications.
4. If you get a security warning, run:

```
xattr -cr /Applications/Cursor.app
```

5. Open Cursor from Applications (or `Cmd+Space` → "Cursor" → Enter).

Alternative (Homebrew):

```
brew install --cask cursor
```

On Windows

1. Go to cursor.com/download (auto-detects Windows)
2. Run the installer and follow the prompts
3. Open Cursor from the Start menu

```
winget install Anywhere.Cursor  
# OR  
choco install cursor
```

First-Time Setup

1. Open Cursor — you'll see a welcome screen
2. Sign in with email, Google, or GitHub (required for AI features)
3. New users get a **14-day Pro trial** — no card required. After that, free Hobby plan (limited AI requests), upgrade anytime for \$20/mo.
4. Import from VS Code (optional): Settings → General → Account → "Import from VS Code"

Install the cursor Terminal Command

Lets you open any folder in Cursor directly from the terminal.

1. Open Cursor
2. Press `Cmd+Shift+P` (Mac) / `Ctrl+Shift+P` (Windows)
3. Type **"Install 'cursor' command in PATH"** and select it

```
cursor /path/to/your/project  
# Mac fallback if the command doesn't work:  
open -a Cursor "/path/to/your/project"
```

Essential Keyboard Shortcuts

Memorize the ones for your OS — you'll use these every day.

ACTION	MAC	WINDOWS	WHAT IT DOES
AI Chat	<code>Cmd+L</code>	<code>Ctrl+L</code>	Open the AI chat panel — ask questions about your code
Inline Edit	<code>Cmd+K</code>	<code>Ctrl+K</code>	The magic button — highlight code, describe a change, AI rewrites it
Composer / Agent	<code>Cmd+I</code>	<code>Ctrl+I</code>	Multi-file AI tasks — create files, refactor across the project
Command Palette	<code>Cmd+Shift+P</code>	<code>Ctrl+Shift+P</code>	Search for any command in Cursor
Quick Open	<code>Cmd+P</code>	<code>Ctrl+P</code>	Jump to any file by name
Toggle Terminal	<code>Cmd+`</code>	<code>Ctrl+`</code>	Open/close the built-in terminal
Extensions	<code>Cmd+Shift+X</code>	<code>Ctrl+Shift+X</code>	Browse and install extensions



WHERE TO LOOK

The **Cmd+K** inline-edit bar

Highlight any code, press **Cmd+K** / **Ctrl+K** — a small input bar appears right above your selection where you type the change in plain English.

Subscribe to an AI Coding Assistant

The "vibe coder" — the tool that lets you describe edits and have the AI execute them.

Option A: ChatGPT Plus (includes Codex)

What you get for \$20/month: ChatGPT web chat, Codex CLI (terminal agent), Codex Web (cloud tasks), 5x higher usage limits.

1. Go to chat.openai.com and sign up / sign in
2. Click "Upgrade to Plus" (\$20/month)
3. Install the Codex CLI (*requires Node.js* — see Step 3.3):

```
npm install -g @openai/codex
```

4. Authenticate:

```
codex auth
```

5. Test it:

```
codex exec "Explain what files are in this directory"
```

Give Codex a task in plain English, and it reads your project, edits files, runs commands, and returns a reviewable result:

```
codex exec "Add a contact form to index.html that collects name, email, and message. Style it with CSS to match the existing site."
```

PERSISTENT INSTRUCTIONS

Create an `agent.md` file in your project root so the AI always knows your conventions:

```
# Agent Instructions
- This is a static HTML website hosted on Firebase
- All CSS goes in public/css/ and all JS goes in public/js/
- Use the brand color #991C52 for primary buttons and links
- Always test with `firebase emulators:start --only hosting` before deploying
```

Option B: Claude Pro (includes Claude Code)

What you get for \$20/month (\$17/mo annual): Claude web chat, Claude Code (terminal agent), higher usage limits.

1. Go to claude.ai and sign up / sign in
2. Click "Upgrade to Pro"
3. Install Claude Code (*requires Node.js*):

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

4. Authenticate:

```
claude
```

5. Test it:

```
claude "Explain what files are in this directory"
```

Claude Code runs locally in your terminal — it can see your working directory, git state, and running processes, and can even start your dev server to verify changes:

```
claude "Build a landing page with a hero section, three feature cards,  
and a contact form. Put it in public/index.html and style it with  
public/css/style.css."
```

Define persistent instructions in a `CLAUDE.md` file (same format as `agent.md` above).

Can You Use Both?

Yes. Many developers subscribe to one (\$20/mo) and use Cursor's built-in AI chat for day-to-day edits, then use the terminal agent for bigger, multi-file tasks:

Cursor's Cmd+K / Composer

Quick, visual, interactive edits — like editing a Word doc with AI assistance.

Codex / Claude Code (terminal)

Larger, autonomous tasks — like asking a developer to build a feature and show you when done.

Install Git & Set Up GitHub

GitHub is where your website's code lives — every change tracked, like Google Drive with a time machine.

Create a GitHub Account

1. Go to github.com and sign up (free)
2. Choose a username (visible on your repos)
3. Verify your email

Install Git

Mac: pre-installed on most Macs. Verify:

```
git --version
```

If missing, macOS will prompt you to install the Xcode Command Line Tools.

Windows: Download from git-scm.com/download/win. During install, choose "**Git from the command line and also from 3rd-party apps**" for PATH. This also installs **Git Bash** — a Unix-style terminal that makes Mac tutorials work on Windows.

Install Node.js (Required for Firebase + AI Tools)

```
# Mac — check first
node -v
# If not installed:
brew install node
```

Windows: download the LTS version from nodejs.org, run the installer (includes npm), then verify:

```
node -v
npm -v
```

Git Identity, GitHub CLI & Your First Repo

Configure Git With Your Identity

Run once on each computer you use:

```
git config --global user.name "Your Name"
git config --global user.email "your-email@example.com"

# Example:
git config --global user.name "Jon Bass"
git config --global user.email "jonbass.jcb@gmail.com"
```

Install & Authenticate GitHub CLI

OS	COMMAND
Mac (Homebrew)	<code>brew install gh</code>
Windows (winget)	<code>winget install GitHub.cli</code>
Windows (choco)	<code>choco install gh</code>

```
gh auth login
# Choose: GitHub.com → HTTPS → Login with a web browser →
# copy the one-time code → paste in the browser link

gh auth status # should show your username, "Logged in to github.com"
```

Create Your First Repository

Option A — create on GitHub's website, then clone:

1. Go to github.com/new, name your repo (e.g. `my-client-website`)
2. Set it to **Private** (recommended for client work), check "Add a README"
3. Click "Create repository", then clone it:

```
git clone https://github.com/your-username/my-client-website.git
cd my-client-website
```

Option B — from the terminal:

```
mkdir my-client-website && cd my-client-website
git init
gh repo create my-client-website --private --source=. --push
```

The .gitignore File

```
# .gitignore
node_modules/
.firebase/
*.log
.DS_Store
.env
```

Set Up Firebase Hosting

Where your website "lives" on the internet.

Firebase Hosting provides: a free `your-project.web.app` URL, free SSL (the padlock icon), a global CDN, custom domain support, and a free tier (Spark plan) that covers most static websites.

Create a Firebase Project

1. Go to the [Firebase Console](#)
2. Click "Add project"
3. Name your project — this becomes part of your free URL
4. Google Analytics is optional — can skip
5. Click "Create project"

IMPORTANT

We use **classic Firebase Hosting** (free Spark plan) for static websites — NOT "Firebase App Hosting" (requires the paid Blaze plan, for dynamic/server-rendered apps like Next.js). For static marketing/portfolio sites, classic Hosting is all you need.

Install & Log In to the Firebase CLI

```
npm install -g firebase-tools
firebase --version

firebase login
# Opens a browser window — log in with your Google account
✓ Success! Logged in as your-email@gmail.com
```

Initialize, Structure & Deploy

Initialize Firebase Hosting

```
cd my-client-website
firebase init hosting
```

QUESTION	ANSWER	WHY
Select a Firebase project	The project from the step above	Links this folder to that project
Public directory?	<code>public</code>	Folder containing your HTML/CSS/JS/images
Single-page app?	<code>No</code> (for most static sites)	Only Yes for a React/Vue SPA
Auto-deploy via GitHub?	<code>No</code> (for now)	Deploy manually to keep full control
Overwrite existing index.html?	<code>No</code>	Don't overwrite existing work

This creates `firebase.json` (hosting config) and `.firebaserc` (project alias).

Project Structure

```
my-client-website/
├── public/           ← your website files go here
│   ├── index.html
│   ├── about.html
│   ├── css/style.css
│   ├── js/main.js
│   └── images/
├── functions/       ← optional, needs Blaze plan
├── firebase.json
├── .firebaserc
└── .gitignore
```

KEY RULE

Everything inside `public/` gets deployed to the internet. Everything outside is configuration and tooling.

Your First Deploy

```
firebase deploy --only hosting

=== Deploying to 'my-client-website'...
i  hosting: uploading files in public/
+  hosting: 12 files uploaded successfully
+  Deploy complete!

Hosting URL: https://my-client-website.web.app
```

Preview Channels & Custom Domains

Preview Channels (Test Before Going Live)

```
firebase hosting:channel:deploy preview
# Creates a temporary URL:
https://my-client-website--preview-abc123.web.app
```

Preview channel URLs expire after about 7 days — great for sharing a work-in-progress with clients for feedback.

Setting Up a Custom Domain

- 1. Firebase Console → **Hosting** → **Add custom domain**
- 2. Enter your domain name
- 3. Firebase shows DNS records to add at your registrar (GoDaddy, Namecheap, etc.)

RECORD	HOST	VALUE
A	yourdomain.com	199.36.158.100
TXT	yourdomain.com	hosting-site=your-project-id
CNAME	www	your-project-id.web.app

- 4. Remove any old DNS records from previous hosts (Wix, Google Sites, etc.)
- 5. Wait for DNS propagation (minutes to a few hours)
- 6. Firebase provides zero-config SSL once the domain is verified



CUSTOM DOMAIN PATH — REGISTRAR → DNS RECORDS → FIREBASE HOSTING → AUTOMATIC SSL

firebase.json & Command Reference

The firebase.json Configuration File

```
{
  "hosting": {
    "public": "public",
    "ignore": ["firebase.json", "**/.*", "**/node_modules/**"],
    "rewrites": [{ "source": "**", "destination": "/index.html" }]
  }
}
```

- `"public": "public"` — serve files from the `public/` folder
- `"ignore"` — files excluded from deployment
- `"rewrites"` — optional; sends all routes to `index.html` (SPAs only). Remove for multi-page sites.

Common Firebase Commands

COMMAND	WHAT IT DOES
<code>firebase login</code>	Log in to your Firebase account
<code>firebase init hosting</code>	Set up Firebase Hosting in a project folder
<code>firebase deploy --only hosting</code>	Deploy your <code>public/</code> folder to the live URL
<code>firebase hosting:channel:deploy preview</code>	Deploy to a temporary preview URL
<code>firebase emulators:start --only hosting</code>	Run a local preview server (usually <code>localhost:5000</code>)
<code>firebase use --add</code>	Add a project alias (for managing multiple projects)
<code>firebase use --json</code>	See which project is currently active
<code>firebase projects:create <project-id></code>	Create a new Firebase project from the CLI
<code>firebase deploy --only hosting --project <id></code>	Deploy to a specific project

Configure Cursor Workspace & Colors

Color-code each client's workspace so you can tell at a glance which project you're in.

When you're working on multiple client projects simultaneously, it's easy to accidentally edit the wrong one. Color-coding prevents this:



GOTCHA (LEARNED THE HARD WAY)

If you open a *folder* in Cursor instead of the *workspace file* (e.g. `PamBass.code-workspace`), workspace color settings won't apply. Always open the workspace file if colors are defined there — or add the same colors to `.vscode/settings.json` in the folder.

Creating the Color Settings File

Method A — Via the Cursor UI (easiest)

1. Open your project in Cursor
2. `Cmd+Shift+P` / `Ctrl+Shift+P`
3. Type "**Preferences: Open Workspace Settings (JSON)**" — opens/creates `.vscode/settings.json`

Method B — Create Manually

```
# Mac
mkdir -p .vscode
touch .vscode/settings.json

# Windows (PowerShell)
mkdir .vscode
New-Item .vscode/settings.json
```

The Color Settings Template

```
{
  "window.title": "YOUR CLIENT NAME – PRODUCTION",
  "window.titleBarStyle": "custom",
  "workbench.colorCustomizations": {
    "titleBar.activeBackground": "#6D184E",
    "titleBar.activeForeground": "FFFFFF",
    "titleBar.inactiveBackground": "#3C142E",
    "titleBar.inactiveForeground": "#D8B9CB",
    "statusBar.background": "#A22972",
    "statusBar.foreground": "FFFFFF",
    "statusBar.border": "#F09BCB",
    "activityBar.background": "#481235",
    "activityBar.foreground": "FFFFFF",
    "activityBarBadge.background": "#FF641E",
    "activityBarBadge.foreground": "FFFFFF",
    "sideBar.border": "#A22972",
    "panel.border": "#A22972",
    "focusBorder": "#A22972"
  }
}
```

Color Properties Reference

PROPERTY	WHAT IT COLORS
<code>titleBar.activeBackground</code>	Top bar when window is focused
<code>titleBar.inactiveBackground</code>	Top bar when window is in the background
<code>statusBar.background</code>	The bottom status bar
<code>activityBar.background</code>	Left sidebar (file icons)
<code>activityBarBadge.background</code>	Notification badge on sidebar icons
<code>sideBar.border</code> / <code>panel.border</code>	Border around sidebar / bottom panel
<code>focusBorder</code>	Outline color when an element is focused

Real F4E Color Examples

Lights On Nebraska (Yellow / Gold)

```
{
  "window.title": "LIGHTS ON — REDESIGN | ${activeEditorShort}",
  "workbench.colorCustomizations": {
    "titleBar.activeForeground": "#111111",
    "titleBar.inactiveBackground": "#BFA327",
    "activityBar.background": "#E8B900",
    "activityBar.foreground": "#111111",
    "activityBarBadge.background": "#FF641E",
    "statusBar.background": "#FFD21F",
    "statusBar.foreground": "#111111",
    "statusBar.debuggingBackground": "#FF641E",
    "sideBar.border": "#FFD21F",
    "panel.border": "#FFD21F",
    "focusBorder": "#FF641E"
  }
}
```

Sincere Practice (Hot Pink)

```
{
  "window.title": "SINCERE PRACTICE — ${activeEditorShort}",
  "window.titleBarStyle": "custom",
  "workbench.colorCustomizations": {
    "titleBar.activeBackground": "#991C52",
    "titleBar.inactiveBackground": "#5C1032",
    "statusBar.background": "#991C52",
    "activityBar.background": "#6D1438",
    "activityBarBadge.background": "#E299C6",
    "activityBarBadge.foreground": "#3C0A1E",
    "focusBorder": "#E299C6"
  }
}
```

Changing the Overall Editor Theme

1. `Cmd+Shift+P` / `Ctrl+Shift+P`
2. Type "**Preferences: Color Theme**"
3. Browse with arrow keys (live preview), Enter to select

Popular themes: Default Dark Modern, Cursor Dark, Default Light Modern, One Dark Pro, Dracula. Install more from the Extensions marketplace (`Cmd+Shift+X`).

The Vibe Coding Workflow

Four ways to build with AI in Cursor.

Method 1 — Cmd+K (Inline Edit): Quick, Targeted Changes

1. Open a file in Cursor, e.g. `public/index.html`
2. Highlight the text inside an `<h1>` tag
3. Press `Cmd+K` / `Ctrl+K`
4. Type: *"Change this to 'Welcome to Sincere Practice' and make it larger."*
5. Cursor shows a diff. `Cmd+Y` to accept, `Cmd+N` to reject.

Use Cmd+K for: text content, colors/fonts/spacing, adding a class, layout changes, generating a terminal command.

Cmd+K in the terminal: place your cursor there, press `Cmd+K`, describe the task — *"Deploy the site to Firebase", "Create a new git branch called feature-contact-form"* — Cursor generates the command; press Enter to run it.

Method 2 — Composer / Agent (Cmd+I): Multi-File Tasks

1. Press `Cmd+I` / `Ctrl+I`
2. Type: *"Create a new about.html page with a hero section, a bio section, and a contact CTA. Link it from the navigation in index.html. Use the existing CSS variables for colors."*
3. The AI creates the file, edits `index.html`, writes the CSS
4. Review the diff view — accept or reject each file

Use Composer for: new pages, multi-file features, refactoring, boilerplate.

AI Chat & Terminal Agents

Method 3 — AI Chat (Cmd+L): Questions & Guidance

1. Press `Cmd+L` / `Ctrl+L`
2. Ask: "How do I make the navigation bar sticky at the top of the page?"
3. The AI reads your codebase and gives context-aware advice, with follow-ups

@-symbols for context: `@file` a specific file, `@folder` a whole folder, `@codebase` search the entire project.

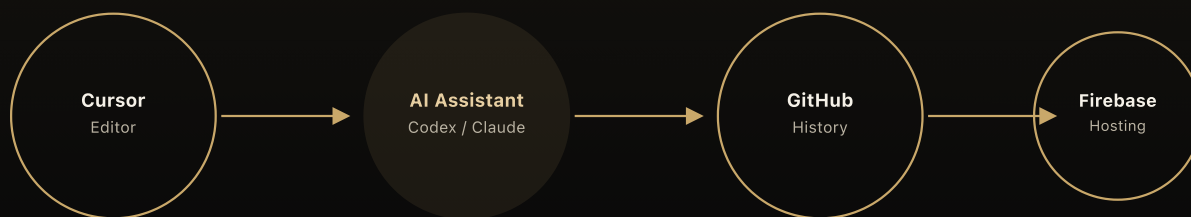
Example: "@codebase How is the contact form currently handled? I want to add email validation."

Method 4 — Codex CLI / Claude Code: Big Autonomous Tasks

```
# Codex
codex exec "Build a complete contact form in public/contact.html.
Include name, email, phone, message. Add CSS in public/css/contact.css
matching the brand color #991C52. Add JS validation in public/js/contact.js."

# Claude Code
claude "Build a complete contact form in public/contact.html. ...
Then start the local preview server so I can see it."
```

The AI reads your project structure, creates the files, writes the code, (Claude Code) starts your local dev server to verify, and shows you what it created.



HOW THE FOUR TOOLS CONNECT — EDITOR → AI → VERSION HISTORY → LIVE HOSTING

A Typical Vibe Coding Session

```
# 1. Open your project in Cursor
cursor /path/to/your/project

# 2. Pull the latest changes
git pull

# 3. Start the local preview server
firebase emulators:start --only hosting
# (Opens at http://localhost:5000)

# 4. Make edits using Cmd+K, Composer, or AI Chat

# 5. Check the preview in your browser

# 6. Commit and deploy
git add .
git commit -m "Add contact form and update navigation"
git push
firebase deploy --only hosting

# 7. Verify the live site
open https://your-project.web.app # Mac
start https://your-project.web.app # Windows
```

Persistent AI Instructions (Project Conventions)

Create `agent.md` (Codex) or `CLAUDE.md` (Claude Code) in your repo root:

```
# Project: Sincere Practice Website

## Stack
- Static HTML/CSS/JS website, hosted on Firebase Hosting (Spark/free)
- Public directory: public/

## Brand Guidelines
- Primary color: #991C52 (deep pink)
- Hover color: #E299C6 (light pink)
- Font: Inter, sans-serif
- Background: #FAFAFA (off-white)

## Conventions
- CSS in public/css/, JS in public/js/, images in public/images/
- Use semantic HTML5 elements, mobile-responsive (test at 375px)
- Always test with `firebase emulators:start --only hosting` before deploying
- Deploy with `firebase deploy --only hosting`
```

Essential Git Commands

Every change is a "commit" — a snapshot you can always return to.

COMMAND	WHAT IT DOES	WHEN TO USE IT
<code>git status</code>	Shows which files changed	Before committing
<code>git status -sb</code>	Short status with branch info	Quick check
<code>git pull</code>	Download latest changes from GitHub	Before you start working — always
<code>git add .</code>	Stage all changed files	Save everything
<code>git add <file></code>	Stage a specific file	Commit only certain files
<code>git commit -m "message"</code>	Save a snapshot with a description	After staging
<code>git push</code>	Upload commits to GitHub	After committing
<code>git log --oneline</code>	See recent commit history	Review what's been done
<code>git diff</code>	See unstaged changes	Before committing
<code>git diff --cached</code>	See staged changes	After add, before commit
<code>git branch</code>	List all branches	See what exists
<code>git checkout <branch></code>	Switch to a branch	Move between branches
<code>git fetch origin</code>	Download remote info without merging	Check for remote changes

The Daily Workflow, Step by Step

```
# Start of session
cd /path/to/your/project
git pull                # get any changes from other computers
git branch --show-current # confirm you're on the right branch
git status -sb          # check for uncommitted changes

# After making edits (in Cursor)
git status -sb
git diff --stat
git diff --check        # catch whitespace errors
git add .
git diff --cached --stat
git diff --cached --check
git commit -m "Add contact form to index.html and update navigation"
git pull                # in case someone pushed while you worked
git push
```

Staging Specific Files

```
git add public/index.html public/css/style.css public/js/main.js
git commit -m "Update homepage styling and add interactive script"
git push
```

Writing Good Commit Messages

Good

```
git commit -m "Add contact form with
name, email, and message fields"
git commit -m "Fix mobile nav menu
overlap on iPhone"
git commit -m "Update brand colors
to match new logo (#991C52)"
```

Bad — too vague

```
git commit -m "updates"
git commit -m "fix"
git commit -m "stuff"
```

Working With Branches

Branches let you work on changes without affecting the main version — think of it as a "draft" copy.

```
# Create and switch to a feature branch
git branch feature-contact-form
git checkout feature-contact-form

# Make changes, commit, push
git add .
git commit -m "Add contact form"
git push -u origin feature-contact-form

# When ready, merge back into main
git checkout main
git merge feature-contact-form
git push

# Delete the branch when done
git branch -d feature-contact-form
```

ALWAYS RUN BEFORE COMMITTING

```
git diff --check
```

Checks for whitespace errors (trailing spaces, mixed tabs/spaces) that can cause subtle bugs — a quick sanity check before every commit.

Pre-Deploy Checklist

The F4E standard launch procedure — run through this before every deploy.

- ☐ **Pull latest changes** — `git pull`
- ☐ **Check your branch** — `git branch --show-current` (should be `main` or your intended branch)
- ☐ **Check for uncommitted changes** — `git status -sb` should show "nothing to commit, working tree clean"
- ☐ **Check for whitespace errors** — `git diff --check` should show no output
- ☐ **Verify commits are pushed** — `git log --oneline -5`, then `git push` (or confirm "Everything up-to-date")
- ☐ **Test locally** — `firebase emulators:start --only hosting`, check `localhost:5000`
- ☐ **Deploy a preview** (optional, recommended for big changes) — `firebase hosting:channel:deploy preview`, share with the client
- ☐ **Deploy to live** — `firebase deploy --only hosting`
- ☐ **Verify the live site** — open the URL, check links and forms
- ☐ **Check the deploy output** — confirm "Deploy complete!" and note the URL

The Full Deploy Sequence (Copy-Paste Ready)

```
# Pre-deploy checks
git pull
git status -sb
git diff --check
git log --oneline -5

# Deploy
firebase deploy --only hosting

# Verify
open https://your-project.web.app # Mac
start https://your-project.web.app # Windows
```

Dry Runs & Reading Deploy Output

Dry Run (Test a Push Without Actually Pushing)

```
git push --dry-run origin main
```

Shows what *would* be pushed without actually doing it — great for verifying you're pushing the right branch.

What a Successful Deploy Looks Like



```
=== Deploying to 'helen-sincere-website-firebase'...

i  hosting: preparing to upload files...
+  hosting: 38 files uploaded successfully
+  hosting: 0 files deleted

+  Deploy complete!

Hosting URL: https://helen-sincere-website-firebase.web.app
Project Console: https://console.firebase.google.com/project/helen-sincere-website-firebase/overview
```

IF YOU SEE AN ERROR

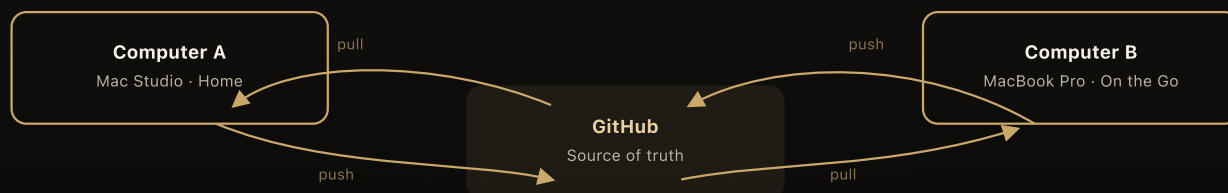
Check: Are you logged in? (`firebase login`) • Is the right project selected? (`firebase use --json`) • Does `public/` exist and contain your files?

Working From Multiple Computers

Follow a few rules to avoid overwriting each other's work.

THE GOLDEN RULE

Before you start working on any computer, always run `git pull` first. Before you push, run `git pull` again, then push.



The Multi-Computer Daily Flow

```

# Computer A — morning
git pull
# ... edit in Cursor ...
git add . && git commit -m "Update homepage hero section"
git push

# Computer B — afternoon
git pull                # gets what Computer A pushed this morning
# ... edit in Cursor ...
git add . && git commit -m "Add about page styling"
git pull                # pull again before pushing, just in case
git push
  
```

Rebasing, Rejected Pushes & Merge Conflicts

Clean History With --rebase

```
git pull --rebase origin main
```

Puts your local commits *on top of* the remote commits instead of creating a merge commit — the result is a clean, linear history.

What Happens If You Forget to Pull?

```
git push
! [rejected]          main -> main (fetch first)
error: failed to push some refs to 'origin/main'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally.
```

This is Git protecting you. Fix:

```
git pull      # get the remote changes, resolve conflicts if any
git push      # now push succeeds
```

Resolving Merge Conflicts

If both computers edited the *same lines* of the *same file*, Git can't auto-merge:

```
git pull
Auto-merging public/index.html
CONFLICT (content): Merge conflict in public/index.html
```

Open the file in Cursor — you'll see conflict markers:

```
<<<<<<< HEAD
<h1>Welcome to Sincere Practice</h1>
=====
<h1>Welcome to Sincere Practice Counseling</h1>
>>>>>> origin/main
```

Everything between `<<<<<<< HEAD` and `=====` is YOUR version. Everything between `=====` and `>>>>>> origin/main` is the REMOTE version.

1. Edit the file to keep the version you want (or combine both)
2. Remove the conflict markers
3. Save, then stage and commit:

```
git add public/index.html
git commit -m "Resolve merge conflict in index.html"
git push
```

Cursor has built-in merge conflict resolution — open a conflicted file and you'll see "Accept Current Change," "Accept Incoming Change," and "Accept Both Changes" buttons above each conflict.

Parallel Work, Stashing & Per-Computer Logins

Using Branches for Parallel Work

```
# On Computer A
git checkout -b computer-a-edits
git add . && git commit -m "Work from home studio"
git push -u origin computer-a-edits

# On Computer B
git checkout -b computer-b-edits
git add . && git commit -m "Work from laptop"
git push -u origin computer-b-edits

# Later, merge both into main
git checkout main
git pull
git merge computer-a-edits
git merge computer-b-edits
git push
```

Never Leave Uncommitted Work

```
# Not ready to commit but need to switch computers:
git stash          # temporarily save your changes
git push          # push any committed work

# On the other computer:
git pull
git stash pop      # restore your saved changes
```

Firebase & GitHub Login on Each Computer

Cursor doesn't sync credentials between machines — run these once on *each* computer:

```
firebase login
gh auth login
```

Sharing Access With Clients

GitHub + Firebase — handing off or granting access without handing over your whole account.

When a project is handed off to a client (or a client wants to make their own edits, view analytics, or bring in their own developer down the line), you don't share passwords — you add them as a scoped collaborator on the two systems that matter: the code (GitHub) and the hosting (Firebase). Each has its own permission levels, so you can grant exactly as much access as the situation calls for.

Adding a Collaborator on GitHub

1. Go to the repository on GitHub.com
2. **Settings → Collaborators and teams**
3. Click **Add people**, enter their GitHub username or email
4. Choose a role:

ROLE	CAN DO	GOOD FOR
Read	View code and history only	A client who just wants visibility
Write	Push commits, open pull requests	A second developer or the client's own dev
Admin	Everything, including repo settings and access management	Full ownership handoff only

They'll receive an email invite — once accepted, they can `git clone` the repo and follow the same workflow described in this guide.



WHERE TO LOOK

GitHub → Repo → Settings → Collaborators and teams

This is the same screen used to see who currently has access and to revoke it later.

Adding a User on Firebase

1. Open the [Firebase Console](#) for the project
2. **Project settings (gear icon) → Users and permissions**
3. Click **Add member**, enter their Google account email
4. Choose a role:

ROLE	CAN DO	GOOD FOR
Viewer	View project settings, hosting history, analytics	A client who wants visibility only
Editor	Deploy, roll back, manage hosting — no billing/user access	A second developer
Owner	Everything, including billing and adding/removing members	Full project handoff

HANDOFF NOTE

If a project is being fully transferred to the client, add them as **Owner** on Firebase and **Admin** on GitHub, confirm they can log in and deploy independently, then remove F4E's access if the engagement has ended — or keep it as Editor/Write if F4E continues maintaining the site.



WHERE TO LOOK

Firebase Console → ⚙️ Project settings → Users and permissions

Same screen for adding, changing, and removing access at any point in the project.

A client only needs a free Google account and a free GitHub account to receive either kind of access — no paid tier is required just to view or collaborate.

Reverting / Undoing

Mistakes happen. With Git and Firebase, you can almost always undo them.

COMMAND	WHEN TO USE	WHAT IT DOES	SAFE FOR SHARED BRANCHES?
<code>git reset</code>	Commit NOT yet pushed	Removes the commit from history	No — local only
<code>git revert</code>	Commit HAS been pushed	Creates a new commit that undoes the old one	Yes

THE GOLDEN RULE

Use `git reset` for local mistakes. Use `git revert` for mistakes already on GitHub.



Undoing Local & Pushed Commits

Undo a Local Commit (Not Yet Pushed)

```
# A: undo commit, KEEP changes staged (safest)
git reset --soft HEAD~1

# B: undo commit, unstage changes (default)
git reset --mixed HEAD~1

# C: undo commit AND discard changes (destructive!)
git reset --hard HEAD~1
```

Undo a Commit That Was Already Pushed

```
git log --oneline -10
# a1b2c3d Add contact form
# 4b05dd8 Build Sincere Practice v1.2 site ← this one broke something
# 9a21cde Initial commit

git revert 4b05dd8
# Git opens your editor for a commit message – default is fine, save & close
git push
```

Result: a new commit that reverses the changes from `4b05dd8`, without erasing history. Everyone who pulls gets the revert.

Force Push (Use With Caution)

```
git reset --hard <good-commit-hash>
git push --force-with-lease
```

WARNING

`--force-with-lease` is safer than `--force` — it refuses to overwrite if someone else pushed since your last fetch. But it still rewrites history. Only do this on branches nobody else is using. Never force-push to `main` if others are collaborating.

Undoing Merges & Firebase Deploys

Undo Multiple Commits / a Merge

```
git revert <oldest-commit-hash>..<newest-commit-hash>

# Undo a merge commit - -m 1 reverts to the branch you merged INTO
git revert -m 1 <merge-commit-hash>
```

Undo a Firebase Deploy

Option A — revert in Git, then redeploy:

```
git revert <bad-commit-hash>
git push
firebase deploy --only hosting
```

Option B — roll back in the Firebase Console (no git needed):

1. Firebase Console → your project → **Hosting**
2. Find the **release history** — a list of all past deploys
3. Click the  menu next to the version you want → **Rollback**
4. The site immediately reverts to that version



FIREBASE KEEPS A FULL RELEASE HISTORY — ROLL BACK TO ANY PAST DEPLOY WITHOUT TOUCHING CODE

Quick Reference: Undo Scenarios

SCENARIO	COMMAND
Committed locally, not pushed	<code>git reset --soft HEAD~1</code>
Committed and pushed, want to undo	<code>git revert <hash> && git push</code>
Pushed, no one else has pulled	<code>git reset --hard <good-hash> && git push --force-with-lease</code>
Deployed a broken site	<code>git revert <hash> && git push && firebase deploy --only hosting</code>
Deployed, want to roll back without git	Firebase Console → Hosting → Release History → Rollback
Merged a branch, want to undo	<code>git revert -m 1 <merge-hash></code>
Want to see what a commit changed	<code>git show <hash></code>

Mac vs Windows Command Reference

`git`, `npm`, and `firebase` commands are identical on Mac and Windows — the differences are in shell built-ins and file-system commands.

Terminal Applications

OS	DEFAULT TERMINAL	RECOMMENDED FOR WEB DEV
Mac	Terminal.app	Terminal.app or iTerm2 · default shell: zsh
Windows	CMD or PowerShell	Git Bash (installed with Git for Windows) — Unix-style commands so Mac tutorials just work

PRO TIP

Install Git Bash on Windows — gives you a Unix-like terminal where `ls`, `cat`, `rm`, `mkdir` work the same as on Mac. Or use **WSL** for a full Linux environment on Windows.

Command Comparison

TASK	MAC (BASH/ZSH)	WIN (CMD)	WIN (POWERSHELL)	WIN (GIT BASH)
List files	<code>ls</code>	<code>dir</code>	<code>ls</code>	<code>ls</code>
Current dir	<code>pwd</code>	<code>cd</code>	<code>pwd</code>	<code>pwd</code>
Make folder	<code>mkdir name</code>	<code>mkdir name</code>	<code>mkdir name</code>	<code>mkdir name</code>
Remove file	<code>rm file</code>	<code>del file</code>	<code>Remove-Item file</code>	<code>rm file</code>
Remove folder	<code>rm -rf dir</code>	<code>rmdir /s dir</code>	<code>Remove-Item -Recurse -Force dir</code>	<code>rm -rf dir</code>
Copy file	<code>cp a b</code>	<code>copy a b</code>	<code>Copy-Item a b</code>	<code>cp a b</code>
Clear screen	<code>clear</code>	<code>cls</code>	<code>clear</code>	<code>clear</code>
Show file contents	<code>cat file</code>	<code>type file</code>	<code>Get-Content file</code>	<code>cat file</code>
Set env variable	<code>export VAR=x</code>	<code>set VAR=x</code>	<code>\$env:VAR="x"</code>	<code>export VAR=x</code>
Path separator	<code>/</code>	<code>\</code>	Both work	<code>/</code>
Open browser	<code>open https://...</code>	<code>start https://...</code>	<code>Start-Process https://...</code>	<code>start https://...</code>

Same Commands, Everywhere

Git / npm / Firebase Commands (Identical on All Platforms)

```
git pull
git push
git add .
git commit -m "message"
git status
git clone https://github.com/...
npm install
npm install -g firebase-tools
firebase login
firebase init hosting
firebase deploy --only hosting
```

Key Differences to Watch For

- **Path separators:** Mac `/Users/jon/projects/my-site` · Windows CMD `C:\Users\jon\projects\my-site` · PowerShell/Git Bash accept both
- **Environment variables:** Mac `export VAR="x"` · CMD `set VAR=x` · PowerShell `$env:VAR="x"`
- **Opening URLs:** Mac `open url` · Windows `start url`
- **rm -rf (destructive!):** Mac/Git Bash `rm -rf folder` · CMD `rmdir /s folder` · PowerShell `Remove-Item -Recurse -Force folder`

Cursor's Integrated Terminal

1. `Cmd+Shift+P` / `Ctrl+Shift+P`
2. Type **"Terminal: Select Default Profile"**
3. Mac: zsh (default), bash, or fish · Windows: PowerShell, Command Prompt, or Git Bash



WHERE TO LOOK

Command Palette → "Terminal: Select Default Profile"

Recommended for Windows: set the default to Git Bash so all Mac-based tutorials work seamlessly.

Troubleshooting Common Issues

B.1 — GitHub CLI Token Expired

Symptom: `gh auth status` shows "invalid token."

Reality: An expired `gh` token does NOT block normal git pushes or Firebase deploys — Git and Firebase both authenticate independently of `gh`.

```
gh auth logout -h github.com
gh auth login -h github.com
# If the web login times out:
gh auth login -h github.com --web
```

B.2 — Wrong Firebase Project in .firebaserc

Symptom: Deploy goes to the wrong URL or fails with "project not found."

```
firebase use --json      # check active project
firebase use --add       # switch to the correct one
```

```
{ "projects": { "default": "helen-sincere-website-firebase" } }
```

B.3 — Firebase Preview Channel Expired

Cause: preview URLs expire after ~7 days.

```
firebase hosting:channel:deploy preview
```

B.4 — DNS Propagation Delays

Changes can take minutes to a few hours. Verify records at your registrar, remove old records (Wix, Google Sites, etc.); Firebase shows "Connecting..." until verification completes.

B.5 — Node Version Warnings

`npm warn EBADENGINE` is harmless noise — safe to ignore.

B.6 — Merge Conflict After Pulling

See Section 11 — Resolving Merge Conflicts.

More Troubleshooting

B.7 — Cursor Workspace Colors Not Showing

Cause: you opened the folder instead of the workspace file.

Fix: open the `.code-workspace` file directly, or add the same colors to `.vscode/settings.json` in the folder.

B.8 — "Failed to authenticate via web browser: context deadline exceeded"

```
gh auth logout -h github.com -u your-username
gh auth login -h github.com
# Choose HTTPS + "Login with a web browser," paste the code quickly
```

B.9 — Deploy Says "No files found in public/"

```
ls public/          # Mac/Git Bash
dir public\         # Windows CMD
mkdir public        # if missing — then move your files in
```

B.10 — "Permission denied (publickey)" When Pushing

```
ls -al ~/.ssh
ssh-keygen -t ed25519 -C "your-email@example.com"
cat ~/.ssh/id_ed25519.pub
# Add this to GitHub: Settings → SSH and GPG keys → New SSH key
ssh -T git@github.com

# Or switch to HTTPS instead:
git remote set-url origin https://github.com/username/repo.git
```

Quick Reference Cheat Sheet

The Complete Setup (One-Time, Per Computer)

```
# 1. Install Cursor from cursor.com/download
# 2. Install Git (Mac: built-in · Windows: git-scm.com)
# 3. Install Node.js from nodejs.org

git config --global user.name "Your Name"
git config --global user.email "your-email@example.com"

brew install gh # Mac – or winget install GitHub.cli
gh auth login

npm install -g firebase-tools
firebase login

npm install -g @openai/codex # or
npm install -g @anthropic-ai/claude-code

git clone https://github.com/your-username/your-project.git
cd your-project
firebase init hosting
cursor .
```

Daily Workflow & Emergency Commands

The Daily Workflow (Every Session)

```
git pull
firebase emulators:start --only hosting

# ... edit in Cursor (Cmd+K, Composer, AI Chat) ...

git status -sb
git diff --check
git add .
git commit -m "Describe what changed"
git pull
git push
firebase deploy --only hosting
open https://your-project.web.app
```

Emergency Commands

```
# Undo last local commit (keep changes)
git reset --soft HEAD~1

# Undo a pushed commit (safe for shared branches)
git revert <commit-hash> && git push

# Roll back a Firebase deploy
# → Firebase Console → Hosting → Release History → Rollback

# Force undo a push (ONLY if no one else has pulled)
git reset --hard <good-commit-hash>
git push --force-with-lease
```

Shortcuts & Monthly Costs

Cursor Keyboard Shortcuts

SHORTCUT	MAC	WINDOWS
AI Chat	Cmd+L	Ctrl+L
Inline Edit	Cmd+K	Ctrl+K
Composer / Agent	Cmd+I	Ctrl+I
Command Palette	Cmd+Shift+P	Ctrl+Shift+P
Quick Open	Cmd+P	Ctrl+P
Toggle Terminal	Cmd+`	Ctrl+`
Extensions	Cmd+Shift+X	Ctrl+Shift+X
Accept AI edit	Cmd+Y	Ctrl+Y
Reject AI edit	Cmd+N	Ctrl+N

Monthly Cost Summary

TOOL	COST	REQUIRED?
Cursor Pro	\$20/mo	Recommended (free Hobby tier also works)
ChatGPT Plus (Codex)	\$20/mo	Pick one AI tool
Claude Pro (Claude Code)	\$20/mo (\$17 annual)	Pick one AI tool
GitHub	Free (public) / \$4/mo (private)	Yes
Firebase	Free (Spark plan)	Yes
Total	~\$20–24/mo	



DOCUMENT END

Built With Frame 4 Eternity

This guide was prepared by Frame 4 Eternity LLC (F4E) — Web Development Division. For questions, edits, or to request a new website build, contact F4E.