THE CLAUDE STARTER KIT

Run Claude Code Free on Windows

A step-by-step setup with OpenRouter, in plain English

Fabian Markl
FabiMarkl.com

Start here

This guide walks you through running Claude Code on a Windows PC using a free AI model from OpenRouter. You do not need a Claude subscription to follow it, and you do not need to know how to code.

Everything is written for someone who has never opened a terminal before. Every technical word gets explained in plain English the first time it appears, and there is a full word list in Part 1 you can flip back to any time.

What you'll have when you finish

- ✓ Claude Code installed and running on your Windows PC
- ✓ A free OpenRouter model doing the work instead of a paid Claude plan
- ✓ A setup you do once that survives restarts and works in every project folder
- ✓ Your API key stored safely, and out of anything you might share
- ✓ A real project built by Claude Code to prove the whole thing works
- ✓ A troubleshooting page for when something goes sideways

> ⚠ **WATCH OUT**
> Read this before you start: free models are genuinely free, but they are not the same as paid Claude. They are slower, they forget more, and you get a limited number of requests per day. This setup is excellent for learning how Claude Code works and building small projects. It is not the setup you want for serious daily work. Part 6 explains exactly where the limits bite and what to do about it.

> 💡 **TIP**
> You do not have to understand every step to complete it. If a step confuses you, do it anyway, then ask Claude Code afterwards: “Explain what step 5 of my setup actually did, like I'm completely new to this.” It will happily explain its own plumbing.

PART 1

What You're Actually Doing

The whole idea in one page, then every tech word explained.

THE 60-SECOND VERSION

What this setup really is

Claude Code is a tool that sits on your computer and does real work in your files. It can create files, read them, edit them, run things, and fix its own mistakes. It is not a chat window that hands you text to copy. It actually does the work.

But Claude Code needs an AI brain behind it, and normally that brain is Anthropic's Claude, which you pay for. What we are doing in this guide is pointing Claude Code at a different brain: a free model hosted by a company called OpenRouter.

> 💡 **THINK OF IT LIKE…**
> Claude Code is the driver. The AI model is the engine. OpenRouter is a garage full of engines you can swap in, and some of those engines cost nothing to use. This guide is you swapping the expensive engine for a free one. The driver never notices the difference, it just drives whatever engine is bolted in.

That is the entire trick. Three settings tell Claude Code to send its requests to OpenRouter instead of to Anthropic, and one more setting tells it which free engine to use. Everything else in this guide is just installing the tools and making the settings stick.

EVERY TECH WORD IN THIS GUIDE

Flip back here whenever a word stops making sense

You do not need to memorise any of this. Skim it once so the words feel familiar, then carry on.

Words about your computer

Word	What it actually means
Terminal	A window where you type commands instead of clicking buttons. Windows calls its version PowerShell. It looks intimidating and it is genuinely just a text box that runs instructions.
PowerShell	The terminal that comes built into Windows. This is where you will paste most of the commands in this guide.
Command	One line of text you type into the terminal, then press Enter. That is it. If nothing appears to happen, that usually means it worked.
Git	A tool that keeps a history of every change made to your project files, so you can look back or undo. Think of it as unlimited undo that survives closing the app.
Git Bash	A second, different terminal that gets installed alongside Git. Some coding tools prefer it. You will not have to use it directly.
VS Code	A free program from Microsoft for opening project folders and looking at files. It is where you will see what Claude Code has built.
Folder / project	A normal Windows folder. Claude Code works inside one folder at a time, and everything it

Word	What it actually means
folder	builds lands there.
Node.js	A piece of software that lets your computer run programs written in JavaScript. You are not writing JavaScript. It is just required by some add-ons.
npm and npx	Two commands that arrive with Node.js. They download and run small helper programs. You will only use them to check the install worked.
PATH	A list Windows keeps of where to look for programs. If a program is not on the list, Windows says “not recognised” even though it is installed.
Environment variable	A named setting that lives in your terminal and that programs can read. This is how you tell Claude Code where to send its requests.
MCP server	An add-on that gives Claude Code a new ability, like reading your email or browsing a website. Not needed today, but Node.js is what makes them possible later.

Words about AI

Word	What it actually means
Model	The actual AI brain. Different models have different strengths, speeds and prices.
Model ID	The exact name a model is filed under. It has to be typed perfectly or nothing works. This guide uses openrouter/free, which picks a working free model for you.
:free	A tag on the end of a model ID that says “give me the free version of this model”. Without it you get charged.
API key	A long password that proves a request came from your account. Treat it exactly like a password, because that is what it is.
OpenRouter	A middleman service that gives you access to hundreds of AI models through one account and one key, including free ones.
Token	A chunk of text, roughly three quarters of a word. AI usage is measured in tokens rather than words.
Context window	How much the model can hold in its head at once. When you hit the limit, it starts forgetting the earlier part of the conversation.
Opus, Sonnet, Haiku	The names of Anthropic's big, medium and small models. Claude Code reaches for different sizes for different jobs, which is why you will set all three to the same free model.
Subagent	A helper Claude Code spins up to go off and do a side task on its own. It also needs a model assigned to it.
CLAUDE.md	A plain text file in your project where you write notes for Claude Code. It reads this every time it starts, so it remembers how your project works.

> 💡 **TIP**
> If you hit a word later that is not on these lists, paste it straight into Claude Code and ask: “What does this mean, explained simply, in the context of setting up a coding tool?”

PART 2

Install The Four Tools

Git, VS Code, Node.js, and Claude Code itself. In this order.

BEFORE YOU INSTALL ANYTHING

The short checklist

You need a PC running Windows 10 or newer, an internet connection, and about 30 minutes. Everything in this part is free.

Install these four in the order given. The order matters, because each one is easier to verify once the one before it is working.

Tool	Do you truly need it?	Why it's in this guide
Git	Recommended	Gives you a reliable coding environment and the Git Bash terminal that many tools expect. Claude Code can fall back to PowerShell without it.
VS Code	Recommended	The easiest way to see the files Claude Code creates, and it has a terminal built in. Claude Code runs fine without it.
Node.js	Optional today	Not needed for Claude Code itself, and not needed for the demo project. Needed later if you want to add MCP add-ons. Installing it now saves a headache later.
Claude Code	Required	This is the actual tool. Everything else is supporting cast.

A note on Node.js: you will see people say Claude Code needs Node.js. That used to be true. The Windows installer used in this guide does not need it. We install it anyway because the moment you want to add an MCP add-on, you will need it, and installing it later while Claude Code is running is more annoying than doing it now.

STEP 1: INSTALL GIT FOR WINDOWS

Roughly 5 minutes

What it is: A tool that records every change made to the files in a project, so any version can be reviewed or restored. It also installs Git Bash, an alternative terminal that many coding tools look for.

Why you need it: It gives you a solid, standard Windows coding setup. Claude Code can work without it by falling back to PowerShell, but you will hit fewer odd problems with it installed.

Download Git for Windows from git-scm.com and run the installer. Click through with the default options. There are a lot of screens and you do not need to change anything on any of them.

When it finishes, open PowerShell and check it worked:

💻 **TYPE THIS**

```
git --version
```

👀 **WHAT IT LOOKS LIKE** — *a successful check*

You type: git --version
You see: git version 2.47.1.windows.1
The numbers will differ. Any version number means it worked.

⚠ **WATCH OUT**
Installing Git does not connect anything to GitHub, does not upload your files, and does not publish anything anywhere. It is a local tool. Git and GitHub are different things, and this guide never touches GitHub.

STEP 2: INSTALL VISUAL STUDIO CODE

Roughly 3 minutes

What it is: A free editor from Microsoft for opening a project folder and looking at the files inside it. It also has a terminal built into the bottom of the window.

Why you need it: It is how you will actually see what Claude Code builds. Without it you are working blind, guessing at what changed.

Download it from code.visualstudio.com and install with the default options.

Then open VS Code, go to File, then Open Folder, and either pick an existing folder or create a new empty one for your project. This folder is where Claude Code will build things.

💡 **TIP**
Make a fresh empty folder for your first project, somewhere easy to find, and give it a simple name with no spaces, like claude-test. Spaces in folder names mean you have to wrap paths in quotes later, which is one more thing to get wrong.

To open the terminal inside VS Code, use the Terminal menu at the top, then New Terminal. A panel opens at the bottom. That panel is PowerShell, and it is already pointed at your project folder, which saves you navigating there manually.

You do not need to install the Claude extension for VS Code. This guide uses the terminal version of Claude Code, which works in the VS Code terminal with nothing extra installed.

STEP 3: INSTALL NODE.JS

Roughly 4 minutes

What it is: Software that lets your computer run JavaScript programs outside a web browser. It comes with two commands, npm and npx, that download and run small helper tools.

Why you need it: Claude Code does not need it. MCP add-ons do. Installing it now means that when you later want to give Claude Code a new ability, it just works.

Go to nodejs.org and download the version labelled LTS. LTS stands for Long Term Support, which is the boring stable one, which is the one you want. Install with default options.

Now close VS Code completely and reopen it. This matters: new programs get added to that PATH list we talked about, and open windows do not notice until they restart.

Open a new terminal and check all three commands:

💻 **TYPE THIS**

```
node --version
npm --version
npx --version
```

👀 **WHAT IT LOOKS LIKE** — *a successful check*

You see three version numbers, one after each command.
Example: v22.14.0 then 10.9.2 then 10.9.2
If any says “not recognized”, close VS Code fully and reopen it, then try again.

⚠ **WATCH OUT**

Do not skip this and plan to have Claude Code install Node.js for you later. It can try, but installers need administrator permission and throw security prompts that Claude Code cannot click for you. Two minutes now saves a stuck session later.

STEP 4: INSTALL CLAUDE CODE

Roughly 3 minutes

What it is: The actual tool this guide is about. It runs in your terminal, works inside one project folder, and can create, read and edit files on your behalf.

Why you need it: This is the thing. Everything up to now was groundwork.

Open PowerShell and paste this single line, then press Enter:

💻 **TYPE THIS**

```
irm https://claude.ai/install.ps1 | iex
```

In plain English, that line means: download the official install script from Anthropic, and run it. The irm part fetches it, the iex part runs it, and the vertical bar passes the first into the second.

When it finishes, Claude Code will normally be sitting here:

💻 **WHERE IT LANDS**

```
C:\Users\YOUR_USERNAME\.local\bin\claude.exe
```

💡 **TIP**

A folder name starting with a dot, like .local, is a convention meaning “configuration, not for everyday poking”. Windows shows these folders normally, but many programs hide them. Nothing is wrong if you cannot see it at first.

STEP 5: MAKE THE CLAUDE COMMAND WORK

Roughly 2 minutes

Installing a program and being able to type its name are two different things on Windows. For the second one to work, the program's folder has to be on the PATH, that list Windows checks when you type a command.

💡 **THINK OF IT LIKE…**

PATH is like the contacts list on your phone. The person exists whether or not they are in your

> contacts, but until you save the number, you cannot just type their name to call them. Adding to PATH is saving the number.

You could do this by hand through the Windows Environment Variables screen, which is fiddly and easy to break. Instead, copy all six lines below into PowerShell and press Enter:

💻 **TYPE THIS**

```
$p = "$env:USERPROFILE\.local\bin";
$u = [Environment]::GetEnvironmentVariable("Path","User");
$n = "$u;$p";
if ($u -notlike "*$p*") {
  [Environment]::SetEnvironmentVariable("Path",$n,"User")
};
```

It reads your current PATH, checks whether the Claude folder is already on it, and adds it only if it is missing. Running it twice does no harm.

> 💡 **TIP**
> Every line ends with a semicolon on purpose. Copying from a PDF sometimes loses the line breaks, and sometimes adds extra ones. The semicolons mean PowerShell understands the command either way, so it works however your copy and paste behaves.

> 🔄 **IT WILL STILL SAY "NOT RECOGNIZED" IN THIS WINDOW**
> This catches almost everyone. Windows only hands the new PATH to programs that start after the change, so the terminal you are sitting in still has the old one. Typing claude there will fail even though the fix worked perfectly. You have not done anything wrong, and you do not need to run it again.

The same trap applies to VS Code, and it catches even more people. If VS Code was already open, opening a new terminal inside it will not help: that terminal inherits its settings from VS Code, and VS Code is still holding the old PATH.

So close VS Code properly, using File then Exit rather than only clicking the X. Then reopen it, open a fresh terminal, and test:

💻 **TYPE THIS**

```
claude --version
where.exe claude
```

> 👀 **WHAT IT LOOKS LIKE** — *a successful check*
> claude --version: shows a version number
> where.exe claude: shows the full path to claude.exe
> Both working means the command is properly wired up.

If Windows still says the command is not recognised, you can always start Claude Code by its full address instead. This always works, it is just more typing:

💻 **TYPE THIS**

```
& "$env:USERPROFILE\.local\bin\claude.exe"
```

Does this work on German or other non-English Windows?

Yes, and you do not need to change anything. The commands use $env:USERPROFILE, which asks Windows for your real profile location rather than assuming it. Even when File Explorer displays your user folder as Benutzer, the actual underlying path is still C:\Users.

> ⚠ **WATCH OUT**
> Never manually retype C:\Users as C:\Benutzer in any command. The German name you see in File Explorer is a display label, not the real path, and typing it will break the command.

You can see your own real path any time with:

💻 **TYPE THIS**

```
$env:USERPROFILE
```

PART 3

Connect It To OpenRouter

Get a key, pick a free engine, bolt it in.

STEP 6: CREATE YOUR OPENROUTER API KEY

Roughly 4 minutes

What it is: OpenRouter is a middleman that gives you one account and one key for hundreds of AI models. An API key is the long password that proves a request came from your account.

Why you need it: It is what lets Claude Code use a free model instead of a paid Claude plan.

Go to openrouter.ai, create a free account, then find the Keys section in your account settings and create a new key. Copy it somewhere safe immediately, because most services only show a key once.

> 🔐 **TREAT THIS LIKE A PASSWORD**
> Anyone who gets your API key can spend money through your account. Never paste it into a chat with a stranger, never put it in a public forum post asking for help, never commit it to a code repository, and never leave it visible on screen while recording or screen sharing.

If a key ever does get out, do not panic and do not try to hide it. Go to your OpenRouter keys page, delete that key, and create a new one. It takes about ten seconds, and the old key stops working the moment you delete it.

> 💡 **TIP**
> Name your keys something descriptive when you create them, like my-laptop or work-pc. When you have four of them and need to remove one, you will be glad you can tell them apart.

STEP 7: PICK A FREE MODEL

Roughly 1 minute

You could go to the models page, filter for free, and copy a specific model ID. You can do that, and there is a good reason not to.

Use this instead:

> 💻 **USE THIS AS YOUR MODEL ID**

```
openrouter/free
```

> ⭐ **WHY THIS ONE, RATHER THAN NAMING A MODEL**
> openrouter/free is not a model, it is a chooser. It looks at the free models available right now, throws out any that cannot do what was asked, and sends your request to one that can. That means it handles the three things most likely to break this setup, all by itself: models that quietly stop being free, models that cannot edit files, and models that are overloaded at that moment.

What it saves you from

The problem	Naming one model	Using openrouter/free
Model retired or no longer free	Everything stops until you find a new ID	It just picks another one
Model can't edit files	It chats and changes nothing, looking broken	Filtered out before it gets your request
Model busy or overloaded	Errors and failed builds	Routed elsewhere
Guide goes out of date	The ID printed here eventually dies	This line keeps working

Tool calling, which you may see called function calling, is the ability to actually create and edit files rather than just describe what it would do. A model without it greets you warmly and changes nothing, which looks exactly like a broken setup. Letting the router filter for it removes the whole problem.

> 💡 **TIP**
> Want a specific model anyway? Copy its ID from OpenRouter exactly, including the :free tag, for example nvidia/nemotron-3-ultra-550b-a55b:free. It has to match character for character. Worth doing if you find one you particularly like, but start with openrouter/free.

> ⚠ **WATCH OUT**
> If you do name a specific model, remember that which models are free changes regularly, with no warning. Do not treat any model ID in any guide, including this one, as permanent, and keep it somewhere easy to swap out.

STEP 8: LOG OUT OF ANY EXISTING CLAUDE LOGIN

Roughly 1 minute

Skip this only if you have never signed into Claude Code before. If you have ever logged in with an Anthropic account, that saved login sits in the background and quietly overrides the settings you are about to make.

Start Claude Code, and inside it type:

💻 **TYPE THIS INSIDE CLAUDE CODE**

```
/logout
```

Then close Claude Code completely before carrying on with the next step.

> ⚠ **THIS IS THE MOST COMMON REASON THE SETUP FAILS**
> People follow every step correctly, and Claude Code carries on using their paid Anthropic account, because a cached login beats the environment settings. If your usage is showing up on your Anthropic bill instead of OpenRouter, this step is the one you missed.

STEP 9: POINT CLAUDE CODE AT OPENROUTER

Roughly 3 minutes

Open your project folder in VS Code and open a terminal. Now paste the block below, with two edits first: swap YOUR_OPENROUTER_KEY for the key you created, and swap the model ID for the one you chose.

💻 **TYPE THIS**

```
$env:OPENROUTER_API_KEY = "YOUR_OPENROUTER_KEY"
$env:ANTHROPIC_BASE_URL = "https://openrouter.ai/api"
$env:ANTHROPIC_AUTH_TOKEN = $env:OPENROUTER_API_KEY
$env:ANTHROPIC_API_KEY = ""

$model = "openrouter/free"
$env:ANTHROPIC_DEFAULT_OPUS_MODEL = $model
$env:ANTHROPIC_DEFAULT_SONNET_MODEL = $model
$env:ANTHROPIC_DEFAULT_HAIKU_MODEL = $model
$env:CLAUDE_CODE_SUBAGENT_MODEL = $model

claude
```

What each line is actually doing

Line	In plain English
OPENROUTER_API_KEY	Stores your key in a named slot so the next lines can reuse it without you pasting it twice.
ANTHROPIC_BASE_URL	The important one. Tells Claude Code to send its requests to OpenRouter instead of to Anthropic.
ANTHROPIC_AUTH_TOKEN	Hands your OpenRouter key over as the credential for those requests.
ANTHROPIC_API_KEY	Deliberately emptied. If a leftover Anthropic key is sitting here, Claude Code may use it and bill you.
$model	Just a shortcut so you type the model ID once instead of four times.
OPUS / SONNET / HAIKU	Claude Code reaches for a big, medium or small model depending on the job. All three now point at your free model.
SUBAGENT_MODEL	Covers the helpers Claude Code spins up for side tasks, so they use the free model too.
claude	Starts it.

⚠ **THE SINGLE MOST COMMON TYPO**

The base URL is https://openrouter.ai/api and it must not have /v1 on the end. The address with /v1 is OpenRouter's other, differently-shaped interface. Add those two characters and every request fails with an error that will not tell you why.

💡 **TIP**

These settings live only in the terminal window you pasted them into. Close that window and they are gone. That is not a mistake, and Step 11 makes them permanent. For now, keep this terminal open.

STEP 10: CONFIRM IT ACTUALLY CONNECTED

Roughly 2 minutes

Inside Claude Code, type:

💻 **TYPE THIS INSIDE CLAUDE CODE**

```
/status
```

👀 **WHAT IT LOOKS LIKE** — *a correct connection*

Base URL: https://openrouter.ai/api
Auth: coming from ANTHROPIC_AUTH_TOKEN, not a logged-in account
Model: the OpenRouter model ID you chose

If any of those three show something else, go back to Step 8 and Step 9. Nine times out of ten it is a stale login or the /v1 typo.

Now prove it can actually do the work

Getting a friendly hello back is not proof of anything. A model that cannot use tools will greet you warmly and then fail to create a single file. Give it something small and real:

✍ **TRY THIS PROMPT**

“Create a file called test.txt in this folder containing three bullet points about why I should keep a project journal. Then read the file back to me and tell me exactly how many characters it contains.”

That one prompt forces it to create a file, read a file, and do something with what it read. If a real test.txt appears in your folder, your setup works.

PART 4

Make It Stick, And Keep It Safe

Set it up once, for every project you'll ever start.

STEP 11: SAVE THE SETUP PERMANENTLY

Roughly 3 minutes

Right now your settings vanish the moment you close the terminal. Claude Code can read them from a settings file instead, and where you put that file decides how much work you are signing up for.

📍 THERE ARE TWO PLACES THIS FILE CAN GO

Inside one project folder, where it applies to that project only. Or in your user folder, where it applies to every project you will ever create. Most guides show you the first one, which quietly commits you to redoing this setup in every new folder for the rest of your life.

⚠ ANSWER THIS BEFORE YOU CHOOSE

Do you also use Claude Code with a paid Anthropic account? If no, take Option A below and enjoy it. If yes, skip to Option C. The user-level file captures every Claude Code session on your whole computer, not just the ones you meant. Your paid projects will quietly start running on the free model, and the first sign is usually a confusing error rather than a helpful message.

Where the file lives	What it covers	Use it when
C:\Users\you\.claude\settings.json	Every project, forever. Set up once.	This is the one you want. Do this by default.
.claude\settings.local.json in a project	That one folder only. Beats the user file when both exist.	You want one specific project on a different model.

Option A: set it up once for every project (recommended)

Run this once in a PowerShell terminal. Either the one inside VS Code or a normal Windows PowerShell window, it does not matter which, and it does not matter which folder you are in. It asks you for your key rather than assuming anything is already set up:

💻 **TYPE THIS**

```
$k = Read-Host "Paste your OpenRouter API key";
$m = "openrouter/free";
$d = "$env:USERPROFILE\.claude";
New-Item -ItemType Directory -Path $d -Force | Out-Null;
$e = @{};
$e.OPENROUTER_API_KEY = $k;
$e.ANTHROPIC_BASE_URL = "https://openrouter.ai/api";
$e.ANTHROPIC_AUTH_TOKEN = $k;
$e.ANTHROPIC_API_KEY = "";
$e.ANTHROPIC_DEFAULT_OPUS_MODEL = $m;
$e.ANTHROPIC_DEFAULT_SONNET_MODEL = $m;
$e.ANTHROPIC_DEFAULT_HAIKU_MODEL = $m;
$e.CLAUDE_CODE_SUBAGENT_MODEL = $m;
$j = @{env=$e} | ConvertTo-Json;
$j | Set-Content "$d\settings.json" -Encoding UTF8;
Write-Host "Saved to $d\settings.json";
```

📋 **WHY EVERY LINE ENDS IN A SEMICOLON**

Copying commands out of a PDF is unreliable: sometimes the line breaks disappear and everything arrives as one long run-on command, sometimes extra breaks get added in the middle of a word. Either way PowerShell answers with a wall of red text. Short lines that each end in a semicolon survive both, because PowerShell can tell where one instruction ends and the next begins no matter what your copy and paste does to the spacing. Select all sixteen lines, paste them in one go, and press Enter.

PowerShell will stop and ask for your OpenRouter API key, the one from Step 6. Paste it and press Enter.

The key does appear on screen as you paste it, so cover it if anyone is watching. It does not get saved into your command history, though, and it is never written into the command itself. That means you can copy, share or reuse these lines safely: they contain no secret.

It then confirms it has saved, and everything now lives in:

💻 **WHERE YOUR SETTINGS NOW LIVE**

```
C:\Users\YOUR_USERNAME\.claude\settings.json
```

⚠ **IF YOU ALREADY HAD A SETTINGS FILE**

This block writes a fresh settings.json, so anything already in that file is replaced. For most people it does not exist yet and there is nothing to lose. If you have used Claude Code before and customised it, make a copy first with the one-liner below, then paste anything you were missing back in afterwards.

💻 **OPTIONAL BACKUP, RUN THIS FIRST**

```
Copy-Item "$env:USERPROFILE\.claude\settings.json" `
  "$env:USERPROFILE\.claude\settings-backup.json" -ErrorAction SilentlyContinue
```

💡 **TIP**

Your key does appear on screen as you paste it, so this is not the moment to be screen sharing.

> If anyone did see it, delete that key in OpenRouter and make a new one, as in Step 6.

That is the last time you need to think about this. Close VS Code, reopen any folder at all, new or old, and just type:

💻 **TYPE THIS**

```
claude
```

It starts connected to OpenRouter. Run /status to confirm, then try it in a completely different folder to prove it follows you around.

> 🔒 **THIS IS ALSO THE SAFER OPTION**
> It is tempting to think one global file is riskier than a file tucked inside a project. It is the opposite. Per-project means a copy of your API key in every folder you work in, and every one of those is a folder you might zip up, share, or push somewhere. One file, in one place you never share, is far easier to keep track of.

Option B: override it for one specific project

Only needed if you want one project on a different model, say a paid one for serious work while everything else stays free. In that project's folder, create .claude\settings.local.json with just the bits you want to change:

💻 **OPTIONAL, PER-PROJECT ONLY**

```
{
  "env": {
    "ANTHROPIC_DEFAULT_SONNET_MODEL": "a-different-model-id"
  }
}
```

A project file beats the user file wherever the two disagree. Anything you leave out falls back to your global setup, so you do not repeat the key or the base URL here.

Already set up? Switch model without redoing anything

If your settings file already exists and you only want to change which model it uses, this edits that one setting and leaves everything else, including your key, exactly as it is:

💻 **SWAP THE MODEL, KEEP EVERYTHING ELSE**

```
$settingsFile = Join-Path $env:USERPROFILE ".claude\settings.json"
$settings = Get-Content $settingsFile -Raw | ConvertFrom-Json

$model = "openrouter/free"

$settings.env.ANTHROPIC_DEFAULT_OPUS_MODEL   = $model
$settings.env.ANTHROPIC_DEFAULT_SONNET_MODEL = $model
$settings.env.ANTHROPIC_DEFAULT_HAIKU_MODEL  = $model
$settings.env.CLAUDE_CODE_SUBAGENT_MODEL     = $model

$settings |
    ConvertTo-Json -Depth 10 |
    Set-Content $settingsFile -Encoding UTF8
```

Change the $model line to any model ID and rerun it whenever you want to switch. Restart Claude Code afterwards, then check with /status.

> **This one is for updating a file that already exists.** If you have not run Option A yet, it will complain that it cannot find the settings, which is correct: do Option A first.

Option C: keep free and paid side by side

This is the one to use if you have a paid Claude subscription as well. It gives you two separate commands: claude stays exactly as it was on your paid account, and a new claude-free command runs on OpenRouter. It works in every folder, and there is no settings file to copy anywhere.

First, if you already ran Option A, undo it. This renames the global file rather than deleting it, so nothing is lost:

💻 **UNDO OPTION A**

```
Rename-Item "$env:USERPROFILE\.claude\settings.json" `
  "settings-openrouter-backup.json"
```

Now add the new command to your PowerShell profile, which is a file PowerShell reads every time it starts. Swap in your key and model first:

💻 TYPE THIS

```
if (-not (Test-Path $PROFILE)) {
    New-Item -ItemType File -Path $PROFILE -Force | Out-Null
}

@'
function claude-free {
    $key = "YOUR_OPENROUTER_KEY"
    $m   = "openrouter/free"
    $env:OPENROUTER_API_KEY = $key
    $env:ANTHROPIC_BASE_URL = "https://openrouter.ai/api"
    $env:ANTHROPIC_AUTH_TOKEN = $key
    $env:ANTHROPIC_API_KEY = ""
    $env:ANTHROPIC_DEFAULT_OPUS_MODEL = $m
    $env:ANTHROPIC_DEFAULT_SONNET_MODEL = $m
    $env:ANTHROPIC_DEFAULT_HAIKU_MODEL = $m
    $env:CLAUDE_CODE_SUBAGENT_MODEL = $m
    claude @args
}
'@ | Add-Content -Path $PROFILE -Encoding UTF8
```

Close PowerShell, open it again, and you now have two commands in any folder:

💻 TYPE THIS

```
claude           # your paid account, untouched
claude-free      # the free OpenRouter model
```

⚠ WATCH OUT

One quirk worth knowing: once you run claude-free in a terminal window, that window stays in free mode until you close it. If you want to go back to your paid account, open a fresh terminal window. Nothing is broken, the settings just linger for the life of that window.

💡 TIP

Do not want to touch any of this by hand? Ask Claude Code: “Add a claude-free function to my PowerShell profile that sets my OpenRouter environment variables and then runs claude, so my normal claude command stays on my paid account.”

KEEPING YOUR KEY SAFE

Two minutes that can save you real money

Your API key is now sitting in your settings file as readable plain text. Convenient for you, and equally convenient for anyone who gets hold of that file.

Three rules:

- Never open that file on screen while recording, streaming or screen sharing.
- Never share or upload the whole project folder while the key is inside it.

- If it ever gets exposed, even briefly, revoke it in OpenRouter immediately and make a new one. Revoking takes ten seconds and is free.

If your project uses Git

If you took Option A, your key lives in your user folder, well away from any project, and Git will never see it. You can skip this.

If you put a key inside a project folder, though, Git can publish it the moment that project gets uploaded anywhere. To prevent that, add this one line to a file named .gitignore in the project folder:

💻 **ADD THIS LINE TO .GITIGNORE**

```
.claude/settings.local.json
```

Not sure whether your folder uses Git? Check with:

💻 **TYPE THIS**

```
git status
```

👀 **WHAT IT LOOKS LIKE** — *what the answer means*

fatal: not a git repository: this folder does not use Git. Skip the .gitignore step entirely.
A list of files: this folder does use Git. Add that line now.

💡 **TIP**

Git has nothing to do with whether Claude Code or OpenRouter work. This step is purely about not leaking your key. If it is confusing, ask Claude Code: “Is this folder using Git, and if so please create a .gitignore that excludes my Claude settings file.”

PART 5

Actually Using It

The daily rhythm, and a project worth showing off.

PICKING UP WHERE YOU LEFT OFF

The commands you'll use every day

Your files stay in the folder forever. Your conversation does not. Typing plain claude starts a brand new conversation with no memory of yesterday, which is the single most common frustration for new users.

Command	What it does	When to use it
claude	Starts a fresh conversation	New task, unrelated to what you were doing before
claude -c	Continues your most recent conversation	Carrying on with yesterday's work. This is the one you'll use most
claude -r	Lets you pick from older conversations	You want a specific session from last week
/status	Shows your connection and model	Checking you're still on OpenRouter
/logout	Clears a saved login	Claude Code is ignoring your OpenRouter settings

The -c is short for --continue and -r is short for --resume. Both spellings work.

A normal next-day routine looks like this:

💻 TYPE THIS

```
# Open your project folder in VS Code, open a terminal, then:
claude -c
```

For projects that run for weeks

Do not rely on one enormous conversation to hold everything. Free models have small context windows, which means they start forgetting the beginning of long conversations, and they forget quietly rather than telling you.

Instead, keep the important things in a CLAUDE.md file in your project folder. Claude Code reads it automatically every single time it starts. Put in it: what the project is, decisions you have made and why, how to run it, and where you got to.

✍ TRY THIS PROMPT

"Create a CLAUDE.md file in this folder. Include what this project is, the decisions we have made so far and why, how to run it, and what the current status is. Keep it short and factual, and update it whenever we make a significant decision."

A DEMO THAT ACTUALLY FINISHES

Proof that free is worth someone's time

Three demos, smallest first. Keep the prompts short and let the model work, then open the file in your browser to see what you got.

IF A BUILD STOPS HALFWAY

First, check your model ID is openrouter/free rather than a specific model you picked by name. Pinning one model is the most common reason a bigger build keeps stalling, because you are tied to a single endpoint and get an empty response whenever it is busy. Beyond that: retry the same prompt, since it often works second time, and anything already written stays on disk, so you can say “carry on from where you stopped”. If one build keeps failing, ask for a single index.html with the CSS and JavaScript inside it instead.

Demo 1: the two-minute proof

One prompt, one file. Use this when you just need to show someone quickly that this is real.

TRY THIS PROMPT

“Create a single index.html with the CSS and JavaScript inside it. No separate files, no libraries.

Build a clean hourly rate calculator: enter a monthly income goal, working days per month, and hours per day, and show the hourly rate I need to charge.

Modern design, works on mobile.”

Double-click the file to open it. That is the entire demo, and it takes about two minutes.

Demo 2: the ROI calculator

A step up: three separate files, sliders that stay in sync with number fields, and a chart. Still one prompt.

TRY THIS PROMPT

“Build a responsive AI Automation ROI Calculator using separate index.html, style.css, and script.js files.

Include inputs for employees, weekly repetitive hours, hourly cost, automation percentage, and monthly software cost.

Show monthly hours saved, gross and net monthly savings, annual savings, ROI, and payback period.

Add synced sliders and number fields, input validation, a simple savings chart, and a polished professional design.”

It writes the three files one after another, and you will see each appear as it goes.

Demo 3: ContentFlow, a real app

The best demo in the guide, and still one prompt. A board you drag cards around, that remembers where you put them. It reads as a real app rather than a web page, and it is genuinely useful afterwards.

✍ TRY THIS PROMPT

"Build a responsive web app called ContentFlow.

Create separate index.html, style.css, and script.js files.

The app helps a creator manage content from idea to publication.

Include:
- a board with five columns: Ideas, Writing, Recording, Editing, Published
- cards with a title, content type, deadline, and status
- a form to create new cards
- drag and drop between columns
- the ability to edit and delete cards
- localStorage so everything remains after refreshing
- a clean modern design
- mobile support
- light and dark mode

Add a few realistic sample projects.

Do not build a landing page.
Do not use external frameworks.
Test all main interactions."

Drag a card from Ideas across to Recording, then reload the page. It is still where you left it.

✅ FOUR THINGS TO TRY ONCE IT OPENS

Add a card of your own. Click it and edit the text. Drag it from one column to another. Then refresh the browser and watch it stay exactly where you put it. That last one is the difference between a web page and a piece of software, and it is worth doing just to see it work.

When something is broken, say so

Free models make more mistakes than paid ones, and you will find something that does not quite work. You do not need to know why. Describe what you see in plain English and let it investigate:

✍ TRY THIS PROMPT

"Dragging a card into the Published column does not save it. Find out why and fix it."

💡 TIP

Be specific about what you did and what happened instead. "It is broken" gives it nothing to work with. "I dragged a card to Published, refreshed, and it went back to Ideas" usually gets it fixed first try.

PART 6

When It Goes Wrong

The real limits, and how to get unstuck.

THE FREE TIER LIMITS NOBODY WARNS YOU ABOUT

Read this before you conclude it's broken

This is the part most guides leave out, and it is the reason most people give up on this setup thinking they did something wrong.

The limit	What it actually means for you
50 requests per day	On a free OpenRouter account. Claude Code sends many requests for a single task, because reading a file, editing it and checking the result are separate requests. One decent-sized build can eat your whole daily allowance.
1,000 per day after $10	Adding $10 of credit once lifts the daily ceiling to 1,000 free requests, permanently. The credit does not expire, and free models still cost nothing to run. For anyone using this seriously, this is the best ten dollars in the whole setup.
20 requests per minute	Applies to free models whether or not you have credit. Hit it and things pause briefly. Wait a moment and carry on.
Smaller context window	Free models hold less in their head than paid Claude. Long sessions start losing earlier details. This is why CLAUDE.md matters.
Availability wobbles	Free models are shared and can be slow, queued, or briefly unavailable. Not your setup, not your fault. Using openrouter/free softens this, because it can route around a model that is struggling.

IF YOU ONLY CHANGE ONE THING

Put $10 of credit on your OpenRouter account. You keep using free models at zero cost per request, but your daily allowance goes from 50 to 1,000. It is the difference between hitting a wall halfway through your first real project and actually being able to work.

TROUBLESHOOTING

Match the symptom, apply the fix

What you see	What to do
claude is not recognized	PATH did not take. Close VS Code fully and reopen. Still stuck? Redo Step 5, or start it directly with the full path shown there.
It's still using my Anthropic account	A cached login is winning. Run /logout inside Claude Code, close it, then redo Step 9.
Every request errors instantly	Check your base URL is exactly https://openrouter.ai/api with no /v1 on the end. Then check your key was pasted whole, with no missing characters.
It chats but never creates files	Your model cannot do tool calling. Switch your model ID to openrouter/free, which only picks models that can.
It stops partway with a limit error	You have hit the daily or per-minute cap. Wait, or add $10 of credit to lift the daily ceiling.
API returned an empty or	The most common free-model failure. The free endpoint accepted the request

malformed response (HTTP 200)	and sent back nothing usable. Usually too much in one go: a long task, a big file just written, or a skill that loaded a lot of background information. See the section below, it is fixable.
My paid Claude projects switched to the free model	You used Option A, which applies to every project on the computer. Switch to Option C in Step 11 to keep the two apart.
It forgot what we were doing	Context window filled up. Start a fresh session and keep the important details in CLAUDE.md.
Model not found	The model ID is wrong, or that model is no longer free. Switch to openrouter/free using the swap script in Step 11.
node or npm not recognized	Close VS Code completely and reopen it. New installs are not visible to already-open windows.
“claude is not recognized”, even after the PATH fix	You are testing in a window that was already open. Close VS Code with File then Exit, reopen it, and try in a fresh terminal. Programs only get the new PATH when they start.
Settings vanished after restart	You skipped Step 11. Do Option A, which stores them in your user folder and covers every project.
Works in one folder but not another	You saved the settings into a project folder instead of your user folder. Redo Step 11 using Option A.

THE EMPTY RESPONSE ERROR, AND HOW TO GET AROUND IT

The one you'll meet most

Sooner or later a build will stop dead with a message about an empty or malformed response. It usually lands mid-task, often right after a big file has been written, which makes it feel like something you did. It is not.

What has happened is that the free endpoint took your request and returned nothing usable. Free models run on shared capacity with tighter limits than the paid versions, and when a request gets too big they tend to fail silently rather than explain themselves.

Fixes, in the order worth trying

1. **Ask for one file at a time**
   This is the big one. Instead of “build me a three-file app”, ask for index.html on its own, then the CSS, then the JavaScript. Each request stays small enough to survive. It is slower to type and far faster overall than watching long builds collapse.
2. **Check how full the conversation is**
   Run /context inside Claude Code to see how much room is left. If it is filling up, run /compact to squeeze it down, or just start a fresh session and point it at the files already on disk.
3. **Simply try again**
   Free endpoints are genuinely flaky and the same request often works on the second attempt. Do not rebuild anything before retrying.
4. **Keep the request plain**
   Elaborate prompts can pull in extra background information before any work starts, and that eats the space your actual task needs. Short and direct goes further on a free model.
5. **Check you are on openrouter/free**

If you pinned a specific model by name, this is very often the cause. One model means one endpoint, and when that endpoint is busy you get empty responses. The automatic option moves your job to a different free model instead of failing. In testing, builds that kept dying on a named model went through first time after switching.

6. **Add the $10 credit**
   It lifts more than the daily cap. A funded account is treated better when free capacity is tight, so heavy builds tend to stumble less.

> 💡 **TIP**
> If a build dies halfway, the files it already wrote are still on disk and still good. Start a new session and say: “Read the files in this folder, work out what is missing, and carry on from there.” You rarely need to start over.

> 🧩 **IF ONE BUILD KEEPS FAILING**
> Ask for it one file at a time instead. Request the HTML on its own, then the styling, then the behaviour, checking the page in your browser between each step. Each request stays small enough to finish, and you end up with the same app.

BEING HONEST ABOUT WHAT THIS IS

So you know when to move on

This setup is genuinely useful, and it is genuinely not the same as paid Claude. Both things are true, and knowing the difference saves you frustration.

OpenRouter's own guidance notes that Claude Code is built and tested against Anthropic's own models, so pointing it at a different model means occasional rough edges: an instruction ignored, a tool used oddly, a task needing two attempts. Nothing is broken when that happens. You are running a car on fuel it was not tuned for, and mostly it runs fine.

The nice part is that switching is one line. Change the model ID in your settings file to a paid model on OpenRouter, or drop the OpenRouter settings entirely and log back into Anthropic. Nothing else about your setup changes.

> 💡 **TIP**
> Wondering whether free is enough for what you want to do? That is the whole of Part 7, including a week-long test you can run to answer it with your own numbers instead of someone else's opinion.

WHAT YOU DO NOT NEED

Ignore anyone who tells you otherwise

- You do not need to install Claude Code a second time inside VS Code.
- You do not need to hand-edit Windows System Variables.
- You do not need to add Claude to both user and system PATH.
- You do not need to install Claude through npm after using the Windows installer.
- You do not need the VS Code extension to use the terminal version.
- You do not need Node.js for the plain HTML, CSS and JavaScript demos in Part 5.

- You do not need to connect anything to GitHub.
- You do not need to type C:\Benutzer into any command, ever.

PART 7

Free, Paid, Or Both

Where the free setup stops making sense, and how to find your own line.

THERE ARE THREE OPTIONS, NOT TWO

Most people only ever hear about two

The choice gets framed as free versus a monthly subscription, which leaves out the option that suits a lot of people best. Here are all three, and you can move between them whenever you like.

Option	What it costs	What it's for
Free models on OpenRouter	Nothing, or ten dollars once to lift the daily limit	Learning, small builds, experimenting without a commitment. What this guide sets up.
Paid models on OpenRouter	Only what you use, from your credit balance	Occasional serious work, with no monthly commitment. Same setup, one line changed.
A Claude subscription	A flat monthly fee, around twenty dollars for the entry plan	Regular daily work on the newest models, with the fewest surprises.

💡 **THE MIDDLE OPTION IS THE UNDERRATED ONE**

Everything you have just built works with paid models too. Put credit on your OpenRouter account, change one line to a paid model ID with the swap script in Step 11, and you are paying pennies per task with no subscription at all. If you only do this properly twice a month, that is almost certainly cheaper than a monthly plan, and you keep the exact setup you already know.

THE PAID PLAN RUNS OUT TOO

The part the comparison usually skips

It is tempting to read this as free being limited and paid being unlimited. That is not the deal. A monthly subscription comes with its own usage caps, and Claude Code eats through them noticeably faster than chatting does.

The reason is the same one that drains your free daily allowance. Agentic work sends many requests for a single task, because reading a file, editing it, and checking the result are separate round trips. A morning of real work on a big project can put you into a waiting period on a paid plan, which surprises people who expected paying to remove the ceiling entirely.

⚖️ **SO THE REAL QUESTION ISN'T "LIMITED OR NOT"**

All three options are limited. They just have differently shaped limits, and the right one for you depends entirely on the shape of your usage rather than on how much you are willing to spend.

Option	The shape of the limit	Suits you if
Free models	A hard daily request count. Runs out fast, resets tomorrow.	You work in short bursts, or you are learning and can stop when it stops.
Paid models on credit	Only your balance. No daily cap, no waiting, you just spend as you go.	Your usage is bursty. Nothing for two weeks, then a heavy day you cannot

		afford to have interrupted.
A subscription	Usage caps over a rolling window. Generous for steady work, hittable on heavy days.	You use it most days at a fairly even pace.

This is why the answer is genuinely different for different people, and why nobody can hand you a single recommendation. Someone building daily gets far more from a subscription. Someone who does one intense project a month will spend less, and wait less, on credit.

> 💡 **TIP**
> Worth knowing before you commit: you can put credit on OpenRouter and use a paid model through the setup you already built, with no subscription at all. It is the cheapest way to find out what having a frontier model actually changes for your kind of work, without signing up to anything monthly.

WHAT YOU'RE ACTUALLY GIVING UP

The difference, in concrete terms

The gap between free and frontier models is not really about intelligence in the abstract. It shows up in specific, recognisable ways, and these are the ones you will feel.

Where it shows	What the difference feels like
Juggling several files	The clearest gap. Free models handle one file well and struggle to keep three in mind at once. This is why Part 5 builds in a single file.
Long jobs	Free models lose the thread on tasks with many steps. Paid ones hold a plan together for much longer.
Fixing its own mistakes	Paid models notice an error and correct course. Free ones are more likely to repeat the same wrong thing until you step in.
Big existing projects	Understanding a codebase means holding a lot in mind at once, and that is exactly where the smaller context window bites.
Subtle bugs	First-guess fixes are fine on both. The bug that needs actual reasoning is where you feel the difference.
Speed	Free models are slower and sometimes queue. Not a dealbreaker alone, but it compounds across a session.
Interruptions	Free endpoints fail sometimes. Usually a retry, occasionally a lost session.

WHAT FREE IS GENUINELY GOOD AT

Not a consolation prize

Everything below is well within reach of the free setup, and this is a longer list than people expect.

- Single-file tools, calculators, trackers and dashboards, exactly like the Part 5 demo.
- Learning how Claude Code works, without paying to find out whether you like it.
- Small scripts that rename files, tidy spreadsheets, or convert things in bulk.
- Explaining code you did not write, in plain English.
- First drafts and boilerplate you were going to rewrite anyway.
- Landing pages, forms and prototypes that need to look right more than be clever.

- Trying a rough idea at eleven at night to see whether it is worth pursuing.

Where it will frustrate you

- Anything with a real deadline, where a failed session costs you more than the subscription.
- Multi-file applications built in one go, which is the failure most people hit first.
- Existing projects of any size, rather than starting from an empty folder.
- Long sessions where you need it to remember decisions from an hour ago.
- Work you are being paid for, where your own time is the expensive part.

NOBODY CAN TELL YOU YOUR LINE

So here's how to find it in a week

Anyone who tells you exactly when to upgrade is guessing, because it depends entirely on what you build and what your time is worth. The good news is that this is easy to measure, and a week is enough.

1. **Pick three real tasks**
   Not test projects. Three things you actually want built this week. Write them down before you start, so you cannot quietly pick easy ones.
2. **Do all three on the free setup**
   Work in passes, one file at a time, exactly as Part 5 describes. Give it a fair run.
3. **Keep a tally as you go**
   Three numbers per task: did it finish, how many times you had to retry or restart, and roughly how long it took start to finish.
4. **Read your own numbers**
   You are not judging the output quality, you are counting friction. Retries and restarts are the signal, because that is where your time actually goes.

> **HOW TO READ YOUR TALLY**
> All three finished with barely a retry: stay free, and put the ten dollars on for the higher daily limit. Roughly one in three needed real wrestling: you are at the line, so try paid models on OpenRouter credit before committing monthly. Most tasks fought you, or you gave up on one: your time is going into working around the tool rather than using it, and a subscription will pay for itself quickly.

THE ONLY SUM THAT MATTERS

It's simpler than it looks

Put a rough value on an hour of your time. It does not need to be precise. Then ask one question: does the paid version save you more than one hour a month?

If an hour of your time is worth twenty dollars or more, the entry-level subscription only has to save you a single hour a month to break even. In practice, most people who use this daily cross that line in the first week, mostly through sessions that do not need restarting.

One honest caveat on that sum: a subscription buys you better models and fewer failed sessions, not an unlimited supply. If your working pattern is a heavy day every few weeks rather than a steady hour each morning, credit on OpenRouter may cost you less and interrupt you less than a monthly plan would. Cheapest is not always the same as the plan with the biggest name.

But if you are learning, building for yourself, or working evenings on something with no deadline, that sum comes out completely differently. An hour lost to a retry costs you nothing except the retry. That is a genuinely good reason to stay free, and not a compromise.

🚀 **WHY THIS MATTERS (TIME & MONEY)**

The real trap is not picking wrong, it is paying for something you have not tested, or grinding away on free out of stubbornness when the tool is costing you more time than it saves. Both are fixed by the same thing: run the week, count the retries, and decide with your own numbers.

One caution on numbers: plan prices, usage caps and which models are included change regularly, and any figure printed in a guide has a shelf life. Treat the twenty dollars here as a rough anchor, and check the current plans before you buy anything.

YOU'RE SET UP

Now Go Build Something

The setup was never the point. What you make with it is.

YOUR FIVE-MINUTE RECAP

Pin this somewhere

💻 **TYPE THIS**

```
# Open your project folder in VS Code, then in the terminal:

claude -c        # continue where you left off
claude           # start something new
claude -r        # pick an older conversation

# Inside Claude Code:
/status          # check you're still on OpenRouter
/logout          # fix it ignoring your settings
```

🚀 **WHY THIS MATTERS (TIME & MONEY)**

You now have a coding assistant that builds real, working software on your machine, for nothing per request. The genuine win is not the money saved. It is that trying an idea now costs you a prompt and ten minutes, so you will try far more ideas than you ever would have when each one meant a weekend.

If a step in this guide did not go the way it says it should, the fastest route through is almost always to describe exactly what you saw to Claude Code itself and ask it what to check next. It is remarkably good at debugging its own setup.

More free guides, prompts and starter kits

fabimarkl.com/library