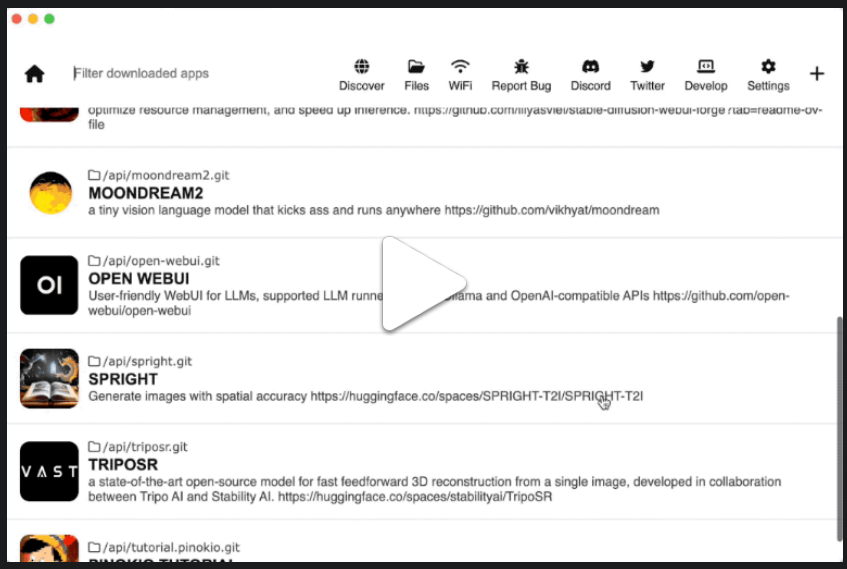
PINOKIO

introduction



Pinokio is a browser that lets you **locally install, run, and automate any AI on your computer**. Everything you can run in your command line can be **automated with Pinokio script**, with a user-friendly UI.

You can use Pinokio to automate anything, including:

1. Install AI apps and models
2. Manage and Run AI apps
3. Create workflows to orchestrate installed AI apps
4. Run any command to automate things on your machine
5. and more...

features

Here's what makes Pinokio special:

1. **Local:** Everything gets installed and runs locally. None of your data is stored on someone else's server.
2. **Free:** Pinokio is an open source application that is 100% free to use with no restriction. There is no one to pay for API access, since everything runs on your local machine. Play with AI as much as you want, for free forever.

3. **Private:** You don't need to worry about submitting private data just to run AI, everything runs 100% privately on your own machine.
4. **User-friendly Interface:** Pinokio provides a user-friendly GUI for running and automating anything that you would normally need to use the terminal for.
5. **Batteries Included:** Pinokio is a self-contained system. You do not need to install any other program. Pinokio can automate anything, including program/library installations. The only program you need is Pinokio.
6. **Cross Platform:** Pinokio works on ALL operating systems (Windows, Mac, Linux).
7. **Save Storage and Resources:** Pinokio has a lot of optimization features that will save you hundreds of gigabytes of disk space. Also, many other resource optimization features (such as memory) all possible with Pinokio.
8. **Expressive Scripting Language:** Pinokio script is a powerful automation scripting language with features like memory, dynamic templating, and extensible low level APIs.
9. **Portable:** Everything is stored under an isolated folder and everything exists as a file, which means you can easily back up everything or delete apps simply by deleting files.

---

architecture

Pinokio takes inspiration from how traditional computers work.

Just like how a computer can do all kinds of things thanks to its comprehensive architecture, Pinokio as a virtual computer is a comprehensive platform for running and automating anything you can imagine with AI.

1. File System: Where and how Pinokio stores files.
2. Processor: How pinokio runs tasks.
3. Memory: How pinokio implements a state machine using its built-in native memory.
4. Script: The programming language that operates pinokio.
5. UI: The UI (user interface) through which users access apps.

INSTALL

> 1. Windows
> 2. Mac
> 3. Linux

windows

Make sure to follow **ALL steps below!**

Step 1. Download

Download for Windows

Step 2. Unzip

Unzip the downloaded file and you will see a .exe installer file.

Step 3. Install

Run the installer file and you will be presented with the following Windows warning:

This message shows up because the app was downloaded from the Web, and this is what Windows does for apps downloaded from the web.

To bypass this,

1. Click **"More Info"**
2. Then click **"Run anyway"**

---

mac

Make sure to follow **BOTH step 1 AND step 2.**

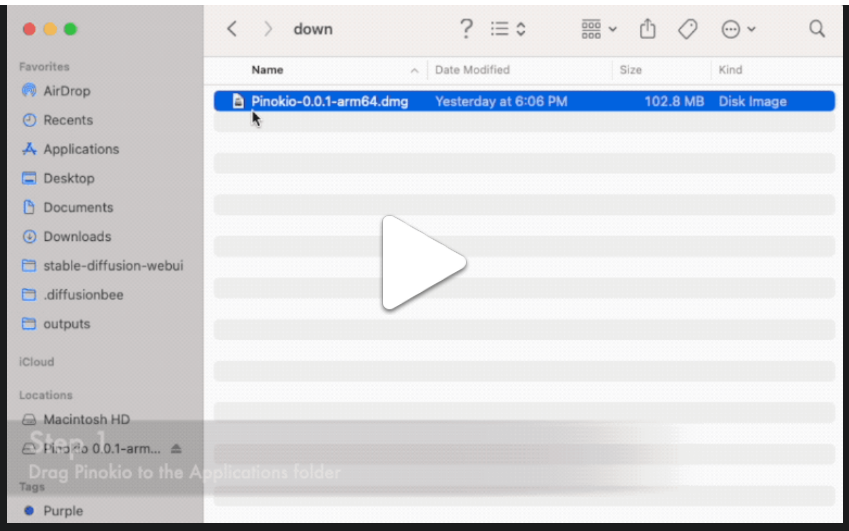
Step 1. Download

Download for M1/M2/M3 Mac **Download for Intel Mac**

Step 2. Install (IMPORTANT!!)

After downloading the dmg files, **you MUST make a patch**, as shown below:

1. Run the downloaded DMG installer file
2. Drag the "Pinokio" app to the Applications folder
3. Run the "patch.command"
4. Open the Pinokio app in the applications folder



linux

For linux, you can download and install directly from the latest release on Github:

Go to the Releases Page

COMMUNITY HELP

To stay on top of all the new APIs and app integrations,

x (twitter)

> Follow @cocktailpeanut on X to stay updated on all the new scripts being released and feature updates.

discord

> Join the Pinokio discord to ask questions and get help.

QUICKSTART

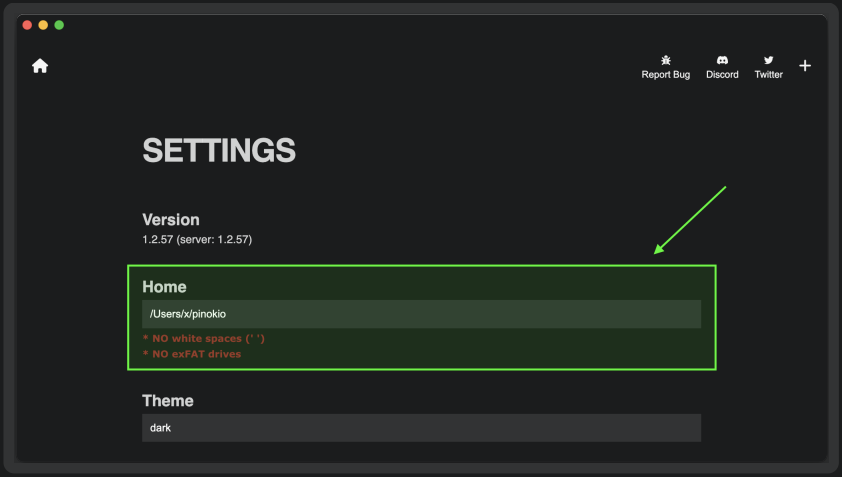
pinokio file system

Pinokio is a self-contained platform that lets you install apps in an isolated manner.

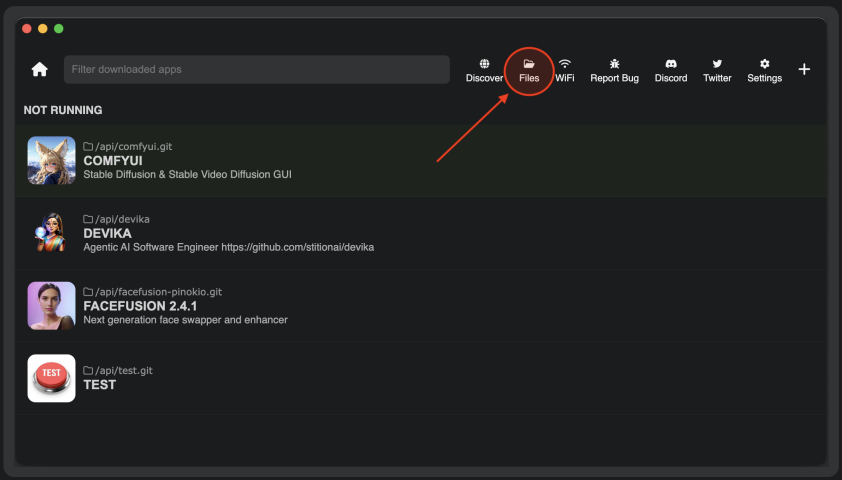
1. **Isolated Environment:** no need to worry about messing up your global system configurations and environments
2. **Batteries Included:** no need to manually install required programs just to install something (such as **ffpeg**, **node.js**, **visual studio**, **conda**, **python**, **pip**, etc.). Pinokio takes care of it automatically.

To achieve this, Pinokio **stores everything under a single isolated folder ("pinokio home")**, so it never has to rely on your system-wide configs and programs but runs everything in a self-contained manner.

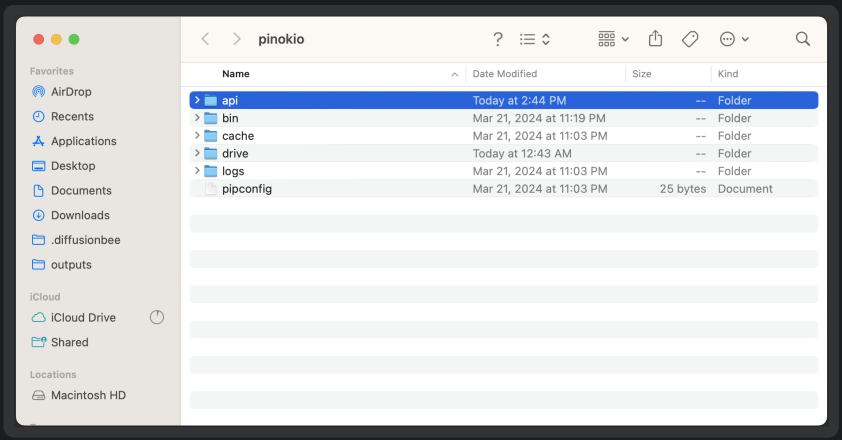
You can set the **pinokio home** folder when you first set up Pinokio, as well as later change it to a new location from the **settings** tab.



So where are the files stored? Click the "Files" button from the home page:



This will open Pinokio's home folder in your file explorer:



Let's quickly go through what each folder does:

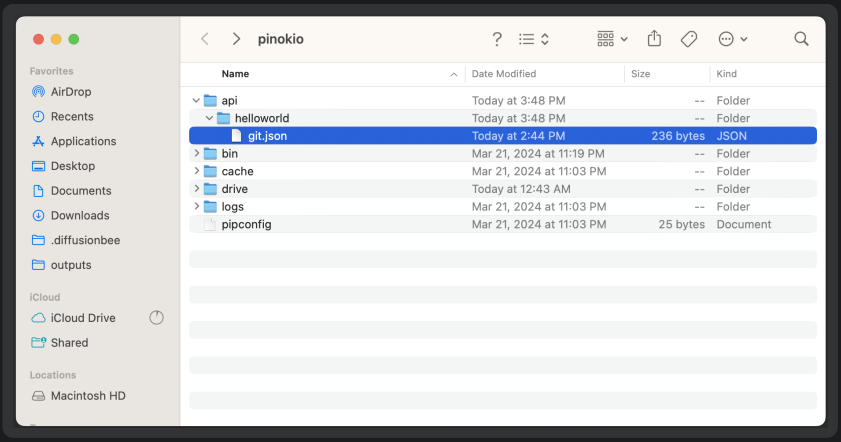
1. `api`: stores all the downloaded apps (scripts).
   - The folders inside this folder are displayed on your Pinokio's home.
2. `bin`: stores globally installed modules shared by multiple apps so you don't need to install them redundantly.
   - For example, `ffmpeg`, `nodejs`, `python`, etc.
3. `cache`: stores all the files automatically cached by apps you run.
   - When something doesn't work, deleting this folder and starting fresh may fix it.
   - It is OK to delete the `cache` folder as it will be re-populated by the apps you use as you start using apps.
4. `drive`: stores all the virtual drives created by the fs.link Pinokio API
5. `logs`: stores all the log files for each app.

> **You can learn more about the file system here**

---

hello world

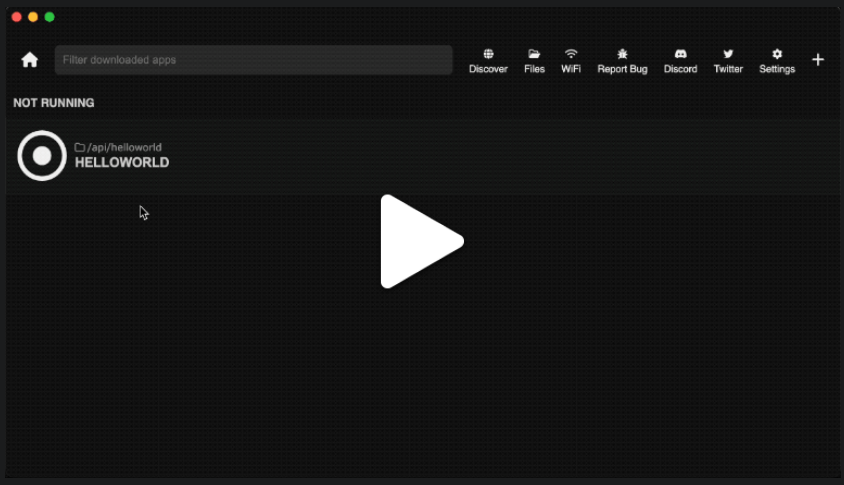
Let's write a script that clones a git repository.



1. Create a folder named `helloworld` under the Pinokio api folder.
2. Create a file named `git.json` under the the Pinokio `api/helloworld` folder.

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "git clone https://github.com/pinokiocomputer/test"
    }
  }]
}
```

Now when you go back to Pinokio, you will see your `helloworld` repository show up. Navigate into it and click the `git.json` tab to run it:



You will see that an `api/helloworld/test` folder has been cloned from the https://github.com/pinokiocomputer/test repository.

templates

We can also dynamically change what commmands to run, and how to run them, using templates.

As an example, let's write a script that runs `dir` on windows, and `ls` on linux and mac.

In your `api/helloworld` folder, create a file named `files.json`:

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "{{platform === 'win32' ? 'dir' : 'ls'}}"
    }
  }]
}
```

1. The `{{ }}` template expression contains a JavaScript expression
2. There are several variables available inside every template expression, and one of them is platform.
3. The value of `platform` is either `darwin` (mac), `win32` (windows), or `linux` (linux).

This means, on Windows, the above script is equivalent to:

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "dir"
    }
  }]
}
```

Or if it's not windows (mac or linux), it's equivalent to:

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "ls"
    }
  }]
}
```

> **You can learn more about templates here**

run in daemon mode

When a Pinokio script finishes running, every shell session that was spawned through the script gets disposed of, and all the related processes get shut down.

For example, let's try launching a local web server using http-server. Create a new folder named `httpserver` under the Pinokio `api` folder, and create a new script named `index.json`:

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "npx -y http-server"
    }
  }]
}
```

Then go back to Pinokio and you'll see this app show up on the home page. Click through and click the `index.json` tab on the sidebar, and it will start this script, which should launch the web server using `npx http-server`.

But the problem is, right after it launches the server it will immediately shut down and you won't be able to use the web server.

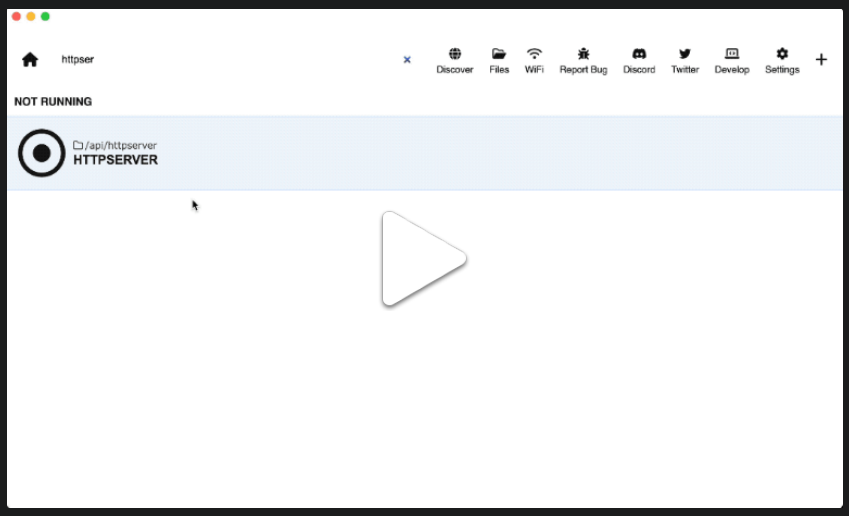
This is because Pinokio automatically shuts down all processes associated with the script when it finishes running all the steps in the `run` array.

To avoid this, you need to tell Pinokio this app should stay up even after all the steps have run. We simply need to add a `daemon` attribute:

```json
{
  "daemon": true,
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "npx -y http-server"
    }
  }]
}
```

Now retry starting the script, and you'll see that the web server starts running and does not shut down.

The web server will serve all the files in the current folder (in this case just `index.json`), like this:



You can stop the script by pressing the "stop" button at the top of the page.

> **Learn more about daemon mode** here

---

run multiple commands

You can also run multiple commands with one `shell.run` call.

Let's try an example. We are going to install, initialize, and launch a documentation engine in one script.

Things like this used to be not accessible for normal people (since you have to run these things in the terminal), but with Pinokio, it's as easy as one click.

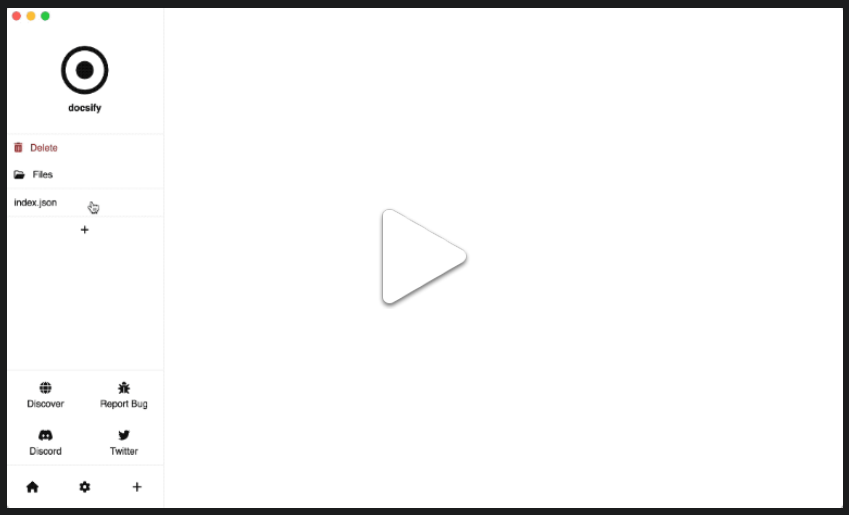
1. Create a folder named `docsify` under the Pinokio `api` folder
2. Create a file named `index.json` under the `api/docsify` folder. The `index.json` file should look like the following:

```json
{
  "daemon": true,
  "run": [{
    "method": "shell.run",
    "params": {
      "message": [
        "npx -y docsify-cli init docs",
        "npx -y docsify-cli serve docs"
      ]
    }
  }]
}
```

This example does 2 things:

1. Initialize a docsify Documentation project
2. Launch the docsify dev server

When you click the dev server link from the Pinokio terminal, it will open the documentation page in a web browser:



> **Learn more ablut the `shell.run` API** here

---

install packages into venv

One of the common use cases for Pinokio is to:

1. Create/activate a venv
2. Install dependencies into the activated venv

Let's try a simple example. This example is a minimal gradio app from the official gradio tutorial

First, create a folder named `gradio_demo` under Pinokio's `api` folder.

Next, create a file named `app.py` in the `api/gradio_demo` folder.

```python
# app.py
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"],
)
demo.launch()
```

We also need a `requirements.txt` file that looks like this:

```
# requirements.txt
gradio
```

Finally, we need an `install.json` script that will install the dependencies from the `requirements.txt` file:

```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "venv": "env",
      "message": "pip install -r requirements.txt"
    }
  }]
}
```

The folder structure will look like this:

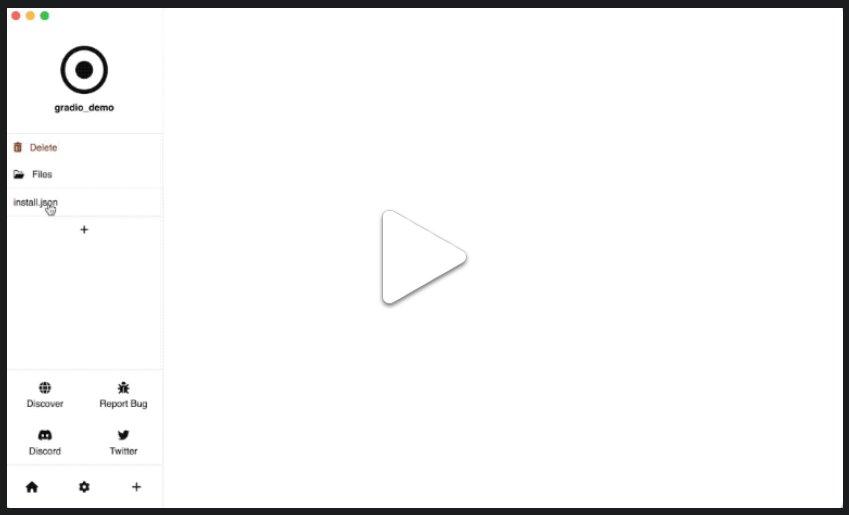
```
/PINOKIO_HOME
  /api
    /gradio_demo
```

```
app.py
requirements.txt
install.json
```

Go back to Pinokio and you'll see the `gradio_demo` app. Click into the UI and click the `install.json` tab, and it will:

1. Create a `venv` folder at path `env`
2. Activate the `env` environment
3. Run `pip install -r requirements.txt`, which will install the `gradio` dependency into the `env` envrionment.

Here's what the installation process looks like (note that a new `env` folder has been created at the end):



> **Learn more about the venv API here**

---

run an app in venv

> **continued from the last section.**

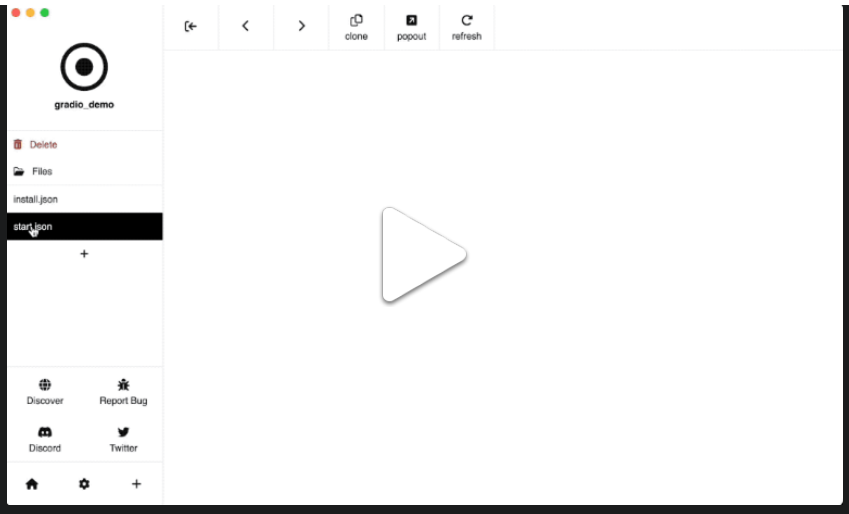
Now let's write a simple script that will launch the gradio server from the `app.py` from the last section. Create a file named `start.json` in the same folder:

```json
{
  "daemon": true,
  "run": [{
    "method": "shell.run",
    "params": {
      "venv": "env",
      "message": "python app.py"
    }
  }]
}
```

Go back to Pinokio and you'll see that the `start.json` file now shows up on the sidebar as well. Click to start the `start.json` script. This will:

1. activate the `env` environment we created from the install step
2. run `python app.py` in **daemon mode** (`daemon: true`), which will launch the gradio server and keep it running.

It will look something like this:



> Learn more about the venv API here

---

download a file

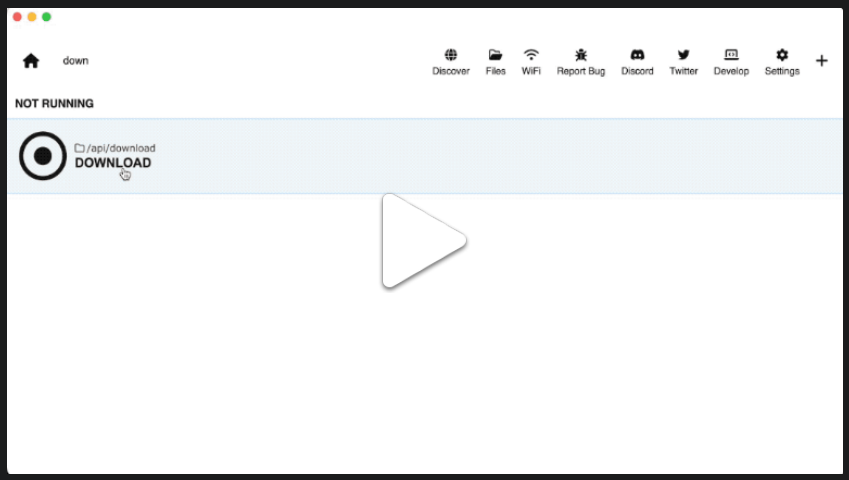
Pinokio has a cross-platform API for downloading files easily and reliably (including automatic retries, etc.).

Let's try writing a simple script that downloads a PDF.

First create a folder named `download` under the Pinokio `api` folder, and then create a file named `index.json`:

```json
{
  "run": [{
    "method": "fs.download",
    "params": {
      "uri": "https://arxiv.org/pdf/1706.03762.pdf",
      "dir": "pdf"
    }
  }]
}
```

This will download the file at https://arxiv.org/pdf/1706.03762.pdf to a folder named `pdf` (The `fs.download` API automatically creates a folder at the location if it doesn't already exist). Here's what it looks like:



> Learn more about the `fs.download` API here

---

call a script from another script

In many cases you may want to call a script from another script. Some examples:

1. An orchestration script that spins up `stable diffusion` and then `llama`.
2. An agent that starts `stable diffusion`, and immediately makes a request to generate an image, and finally stops the `stable diffusion` server to save resources, automatically.
3. An agent that makes a request to a `llama` endpoint, and then feeds the response to a `stable diffusion` endpoint.

We can achieve this using the `script` APIs:

- `script.start`: Start a remote script (Download first if it doesn't exist yet)
- `script.return`: If the current script was a child process, specify the return value, which will be made available in the next step of the caller script.

Here's an example. Let's create a simple `caller.json` and `callee.json`:

`caller.json`:

```json
{
  "run": [{
    "method": "script.start",
    "params": {
      "uri": "callee.json",
      "params": { "a": 1, "b": 2 }
    }
  }, {
    "method": "log",
    "params": {
      "json2": "{{input}}"
    }
  }]
}
```

First step, the `caller.json` will call `callee.json` with the params `{ "a": 1, "b": 2 }`.

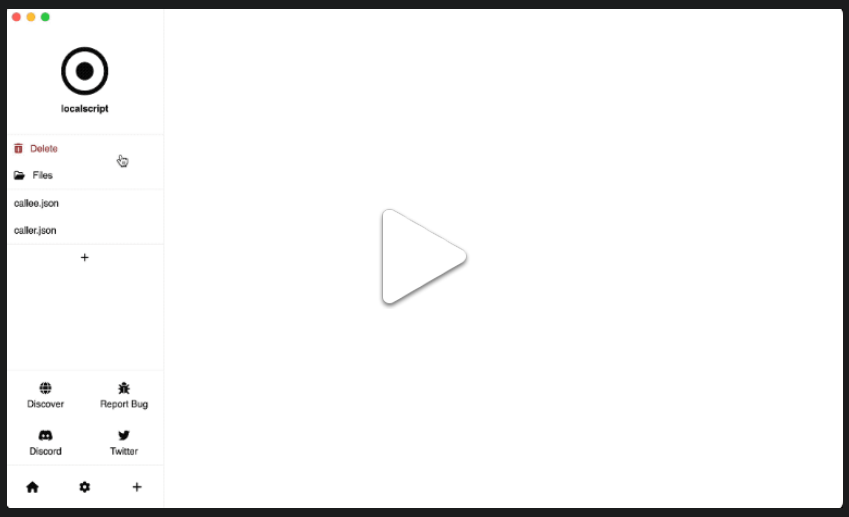
This params object will be passed into the `callee.json` as `args`:

`callee.json`:

```json
{
  "run": [{
    "method": "script.return",
    "params": {
      "ressponse": "{{args.a + args.b}}"
    }
  }]
}
```

The `callee.json` script immediately returns the value `{{args.a + args.b}}` with the `script.return` call.

Finally, the `caller.json` will call the last step `log`, which will print the value `{{input}}`, which is the return value from `callee.json`. This will print `3`:



install, start, and stop remote scripts

The last section explained how you can call a script from within the same repository. But what if you want to call scripts from other repositories?

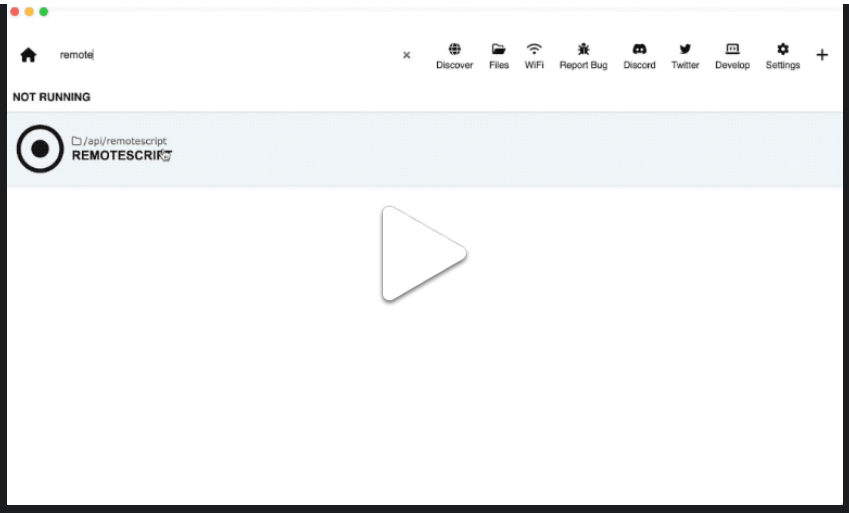
The `script.start` API can also download and run remote scripts on the fly.

Create a folder named `remotescript` under Pinokio `api` folder and create a file named `install.json` under the `api/remotescript`:

```json
{
  "run": [{
    "method": "script.start",
    "params": {
      "uri": "https://github.com/cocktailpeanutlabs/moondream2.git/install.js"
    }
  }, {
    "method": "script.start",
    "params": {
      "uri": "https://github.com/cocktailpeanutlabs/moondream2.git/start.js"
    }
  }, {
    "id": "run",
    "method": "gradio.predict",
    "params": {
      "uri": "{{kernel.script.local('https://github.com/cocktailpeanutlabs/moondream2.git/sta
      "path": "/answer_question_1",
      "params": [
        { "path": "https://media.timeout.com/images/105795964/750/422/image.jpg" },
        "Explain what is going on here"
      ]
    }
  }, {
    "method": "log",
    "params": {
      "json2": "{{input}}"
    }
  }, {
    "method": "script.stop",
    "params": {
      "uri": "https://github.com/cocktailpeanutlabs/moondream2.git/start.js"
    }
  }]
}
```

1. The first step starts the script https://github.com/cocktailpeanutlabs/moondream2.git/install.js.
   - If the `moondream2.git` repository already exists on Pinokio, it will run the install.js script.
   - If it doesn't already exist, Pinokio automatically clones the `https://github.com/cocktailpeanutlabs/moondream2.git` repository first, and then starts the install.js script after that.
2. After the install has finished, it then launches the gradio app using the https://github.com/cocktailpeanutlabs/moondream2.git/start.js script. This script will return after the server has started.
3. Now we run `gradio.predict`, using the kernel.script.local() API to get the local variable object for the start.js script, and then getting its `url` value (which is programmatically set inside the `moondream2.git/start.js` script).
   - Basically, this step makes a request to the gradio endpoint to ask the LLM "Explain what is going on here", passing an image.
4. Next, the return value from the `gradio.predict` is logged to the terminal using the `log` API.
5. Finally, we stop the `moondream2/start.js` script to shut down the moondream gradio server using the `script.stop` API.
   - If we don't call the `script.stop`, the moondream2 app will keep running even after this script halts.

Here's what it would look like:



> The ability to run `script.start`, and then `script.stop` is very useful for running AI on personal computers, because most personal computers do not have unbounded memory, and your computer will quickly run out of memory if you cannot shut down these AI engines programmatically.
>
> With `script.stop` you can start a script, get its response, and immediatley shut it down once the task has finished, which will free up the system memory, which you can use for running other subsequent AI tasks.

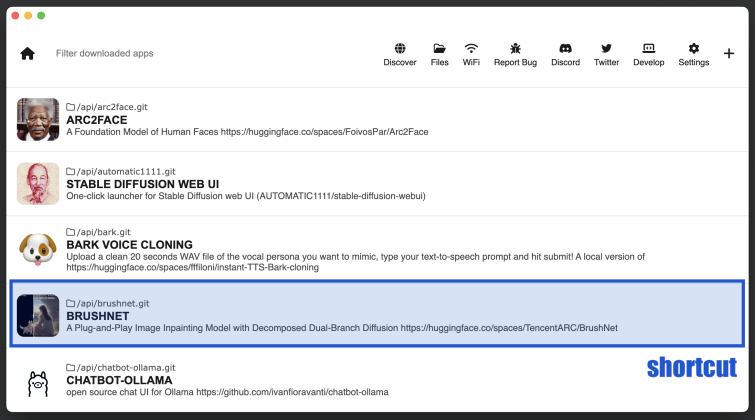
---

build ui with pinokio.js

Pinokio apps have a simple structure:

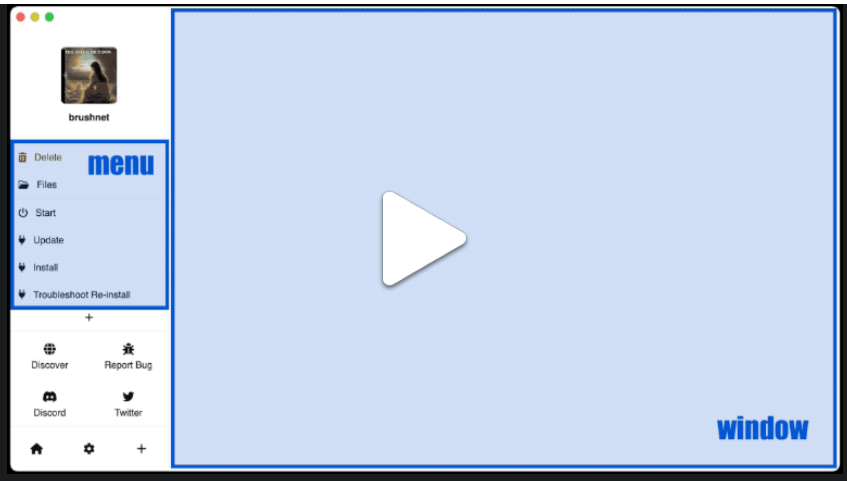
1. shortcut: The app shortcut that shows up on Pinokio home.
2. app: The main UI layout for the app

Shortcut



App

- **Menu:** The sidebar that displays all the links you can run (as well as their running status)
- **Window:** The viewport that displays a **web page**, or **a terminal** that runs the scripts



By default if you do not have a `pinokio.js` file in your project,

- the **shortcut** displays the folder name as the title, and a default icon as the app's icon.
- the **menu** displays all `.js` or `.json` files in your repository root.

While this is convenient for getting started, it's not flexible enough:

1. You can't control what gets displayed in the menu bar
2. You can't control how the scripts are launched (by passing `params` for example)
3. You can't control how the app is displayed
   - The title of the app will be your folder name
   - There is no description
   - The icon will just show a default icon.

To customize how your app itself behaves, you will want to write a UI script named `pinokio.js`.

Let's try writing a minimal UI:

1. Create a folder named `downloader` in the `/PINOKIO_HOME/api` folder
2. Add any icon to the `/PINOKIO_HOME/api/downloader` folder and name it `icon.png`
3. Create a file named `/PINOKIO_HOME/api/downloader/download.json`
4. Create a file named `/PINOKIO_HOME/api/downloader/pinokio.js`

/PINOKIO_HOME/api/downloader/icon.png

/PINOKIO_HOME/api/downloader/download.json

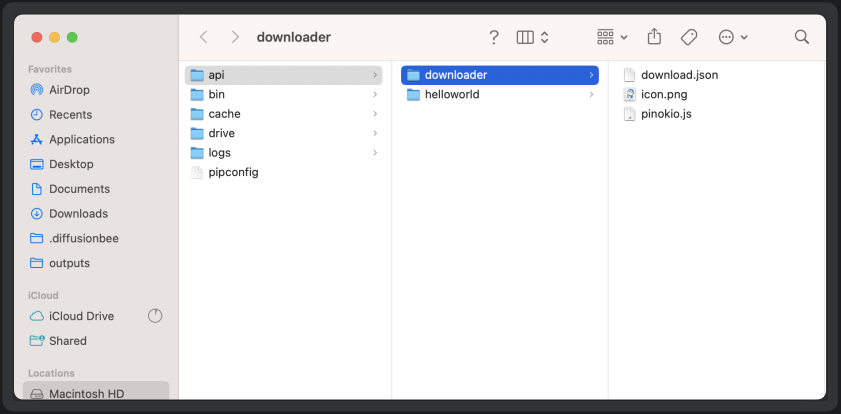
```json
{
  "run": [{
    "method": "shell.run",
    "params": {
      "message": "git clone {{input.url}}"
    }
  }]
}
```

/PINOKIO_HOME/api/downloader/pinokio.js

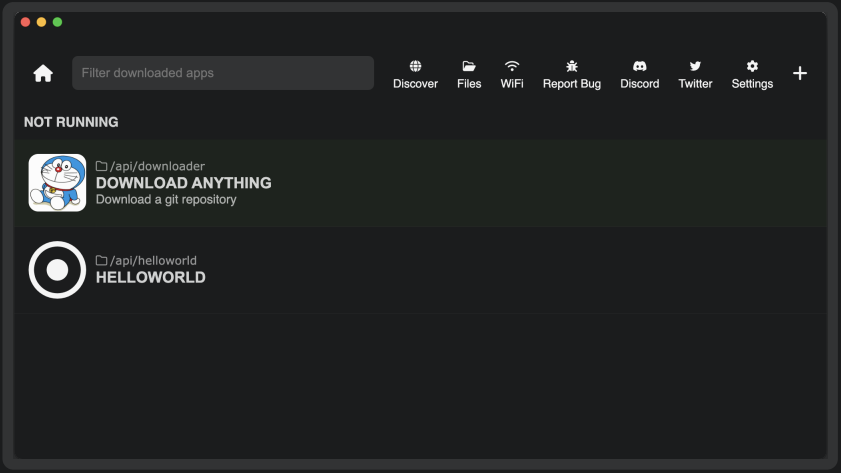
```js
module.exports = {
  title: "Download Anything",
  description: "Download a git repository",
  icon: "icon.png",
  menu: [{
    text: "Start",
```

```
      href: "download.json",
      params: {
        url: "https://github.com/cocktailpeanut/dalai"
      }
    }]
  }
```

The end result will look like this in your file explorer:

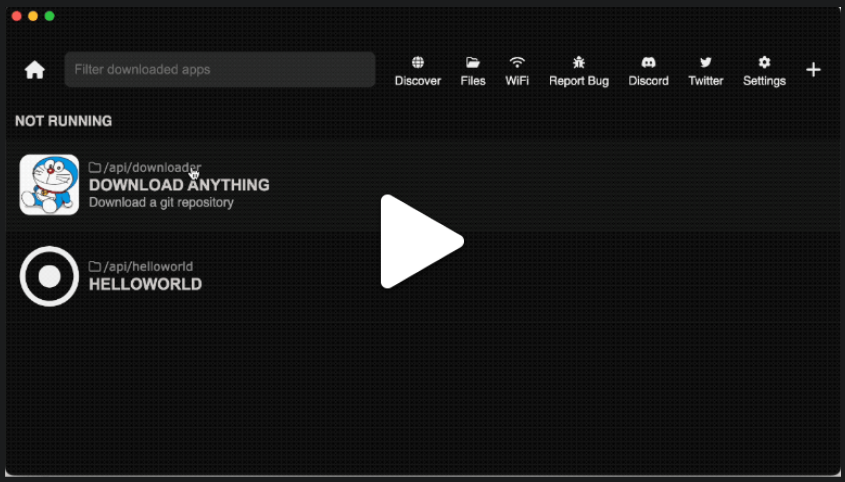


Now go back to Pinokio and refresh, and you will see your app show up:



- the title displays `Download Anything`
- the description displays `Download a git repository`
- the icon is the `icon.png` we've added

Now when you click into the app, you will see the following:



1. You will see the menu item `Start`.
2. Click this to run the `download.json` which is specified by the `href` attribute.
3. Also note that the script is passing the value of https://github.com/cocktailpeanut/dalai as the `params.url` value.
4. The `params` passed to the `download.json` is made available as the `input` variable, so the `git clone {{input.url}}` will be instantiated as `git clone https://github.com/cocktailpeanut/dalai`.

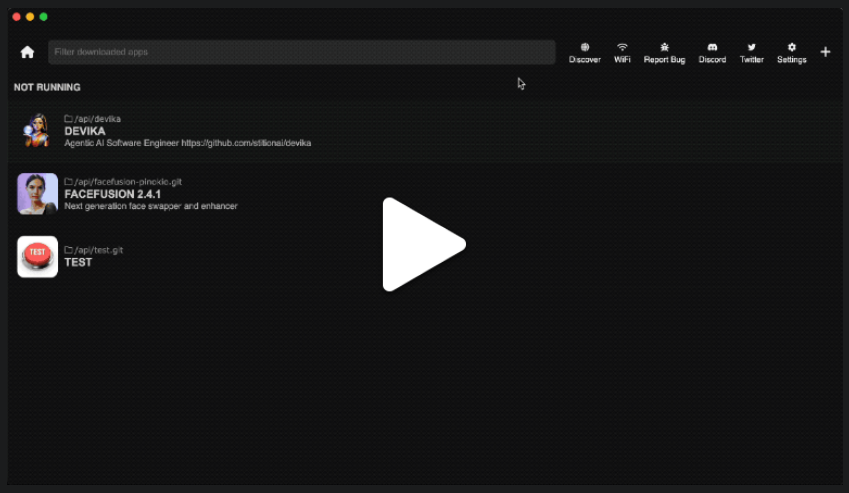
publish your script

Once you have a working script repository, you can publish to any git hosting service and share the URL, and anyone will be able to install and run your script.

install script from any git url

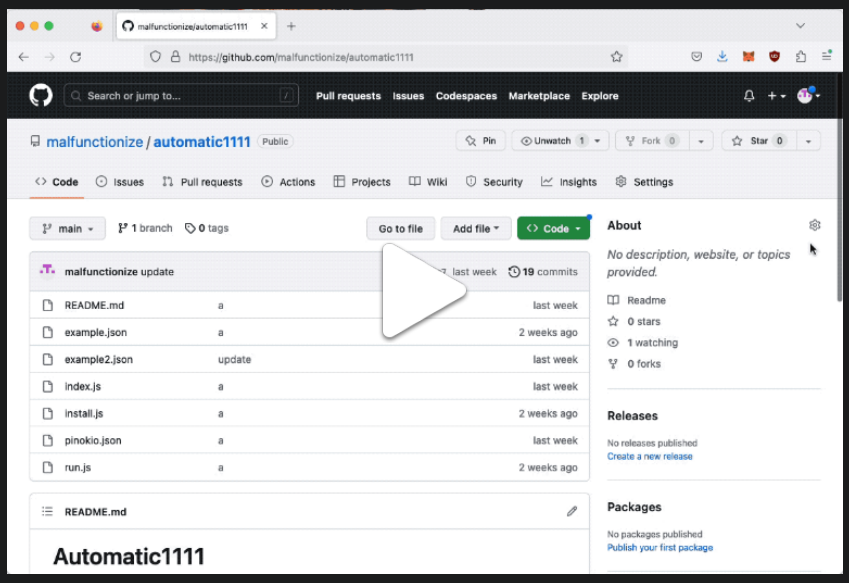
You can install any pinokio script repository very easily:

1. Click the "Download from URL" button at the top of the Discover page.
2. Enter the git URL (You can optionally specify the branch as well).

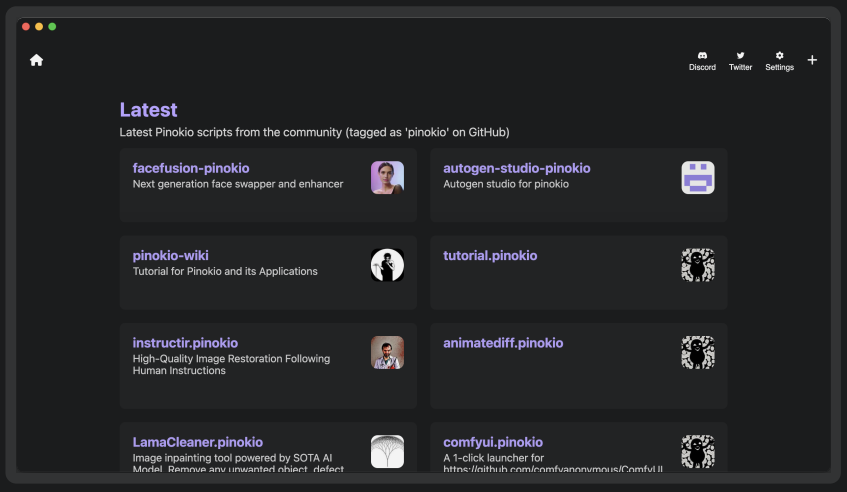


list your script on the directory

If you published to github, you can tag your repository with "pinokio" to make it show up in the "latest" section of the Discover page.



Now it will automatically show up on the "latest" section (at the bottom of the "Discover" page):



> **Pinokio constructs the "Latest" section automatically from GitHub "/repositories" API at https://api.github.com/search/repositories?q=topic:pinokio&sort=updated&direction=desc**
>
> **So if you tagged your repository as "pinokio" but doesn't show up, check in the API result, and try to figure out why it's not included in there.**

---

auto-generate app launchers

While it is important to understand how all this works, in most cases you may want a simple "launcher combo", which includes:

1. **App install script:** Installs the app dependencies
2. **App Launch script:** Starts the app
3. **UI:** Displays the launcher UI.
4. **Reset script:** Resets the app state when something goes wrong.
5. **Update script:** Updates the app to the latest version with 1 click.

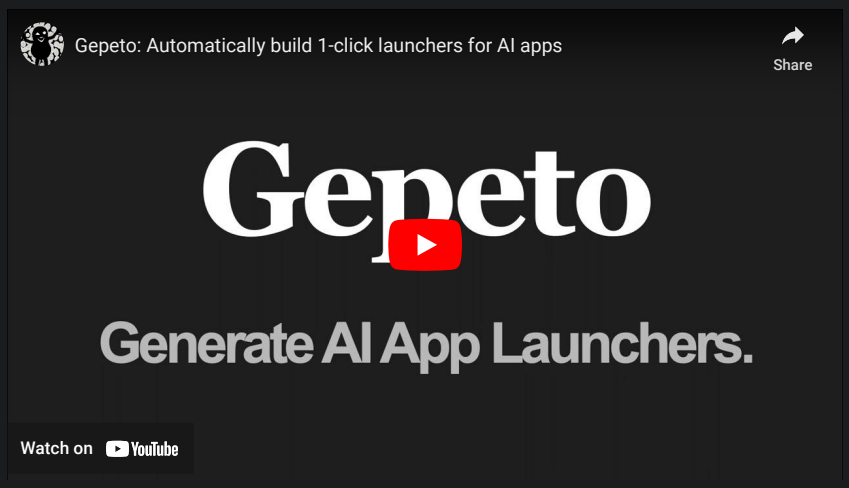
This use case is needed so often, that we've implemented a program that automatically generates these scripts instantly. It's called Gepeto.

GEPETO

Gepeto is a program that lets you **automatically generate Pinokio scripts, specifically for app launchers.**

Let's start by actually generating an app and its launcher in 1 minute.

gepeto quickstart

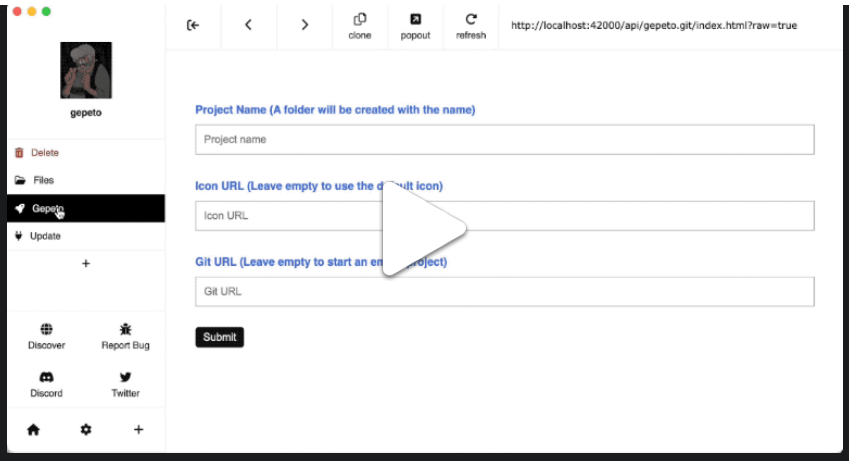


1. Install Gepeto on Pinokio

If you don't have gepeto installed already, find it on Pinokio and install first.

2. Generate Scripts with Gepeto

You will see a simple web UI that lets you fill out a form. For simplicity, just enter `Helloworld` as the project name, and press **submit**.

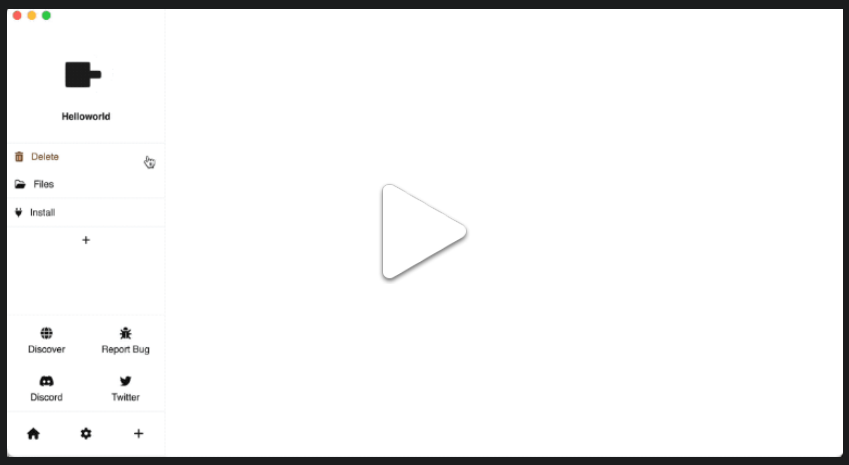


This will initialize a project. When you go back to Pinokio home,

1. You will see a new entry named `Helloworld`. Click into it and you'll see the launcher screen.
2. Also, check your `/PINOKIO_HOME/api` folder, you will find a new folder named `Helloworld` with some script files.

3. Install and Start the App

Now let's click the **install** button to install the app, and when it's over, click **start** to launch.



You will see a minimal gradio app, where you can enter a prompt and it will generate an image using Stable Diffusion XL Turbo.

So what just happened? We've just **created an empty project**, which comes with a minimal demo app.

Let's take a look at each generated file in the next section.

---

creating an empty project

Gepeto automatically generates a minimal set of scripts required for an app launcher. A typical app launcher has the following features:

1. **Install:** Install the dependencies required to run the app. (`install.js`)
2. **Launch:** Launch the app itself. (`start.js`)
3. **Reset install:** Reset all the installed dependencies in case you need to reinstall fresh. (`reset.js`)
4. **Update:** Update to the latest version when the project gets updated. (`update.js`)
5. **GUI:** The script that describes what the launcher will look like and behave on Pinokio home and as a sidebar menu. (`pinokio.js`)

Here's what it looks like: