SOURCE

August 2025, Issue 2
Suggested Donation £3:00

SOURCE:
a quarterly
periodical that takes a
look into interesting
opensource projects and
applications. In this issue we'll
look at getting started with the Godot
game engine, consider Logseq and
Syncthing to make a second brain, explore
adding Termux, a terminal emulator app on an
android phone, simple approaches to contributing to
openstreetmap... and more!

Pay what you feel via Paypal here

PayPal

Pay what you feel via KoFi

Support me on Ko-fi

Welcome to SOURCE

Hello, I'm Jo, AKA Concretedog, and first of all, thank you for taking interest in SOURCE.

I've been writing about technology, maker/hardware hacker/tinkerer culture for a while now but I often have stories, ideas or longer form tutorials that don't fit easily in any of my clients platforms.

So SOURCE is an attempt to fill this gap. The idea so far is for quarterly issues on a suggested donation or "pay what you feel" model. No subscriptions. It's going to focus on opensource projects and technologies, be it hardware, software or possibly other areas of open culture, open governance and more. We'll see!

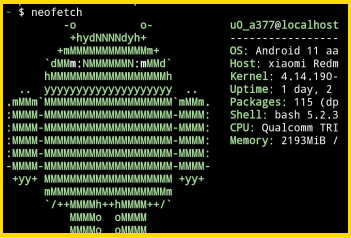
If you downloaded, read or shared SOURCE issue one MANY thanks. The issue covered Computational Fluid Dynamics with FreeCAD and was a great learning experience and has travelled well and even garnered a tiny amount of donations. Thank you very much.

This issue 2 is a fuller affair, and it's nice to look around at a few different areas. I've really enjoyed tinkering with Termux, breathing new life and functionality into my old droid phone. Tackling starting Godot has been an area I've needed to nudge myself into for a long time and at a personal level it's cool that this project is giving me the impetus to move these explorations along. I hope you enjoy it too.

Contents

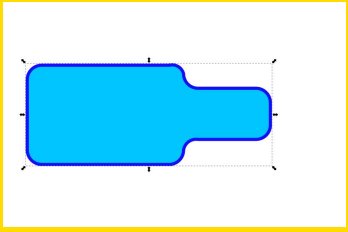
Page 4, Logseq and Syncthing: Building a Second Brain.

Page 7. Making a Dot Go with Godot.

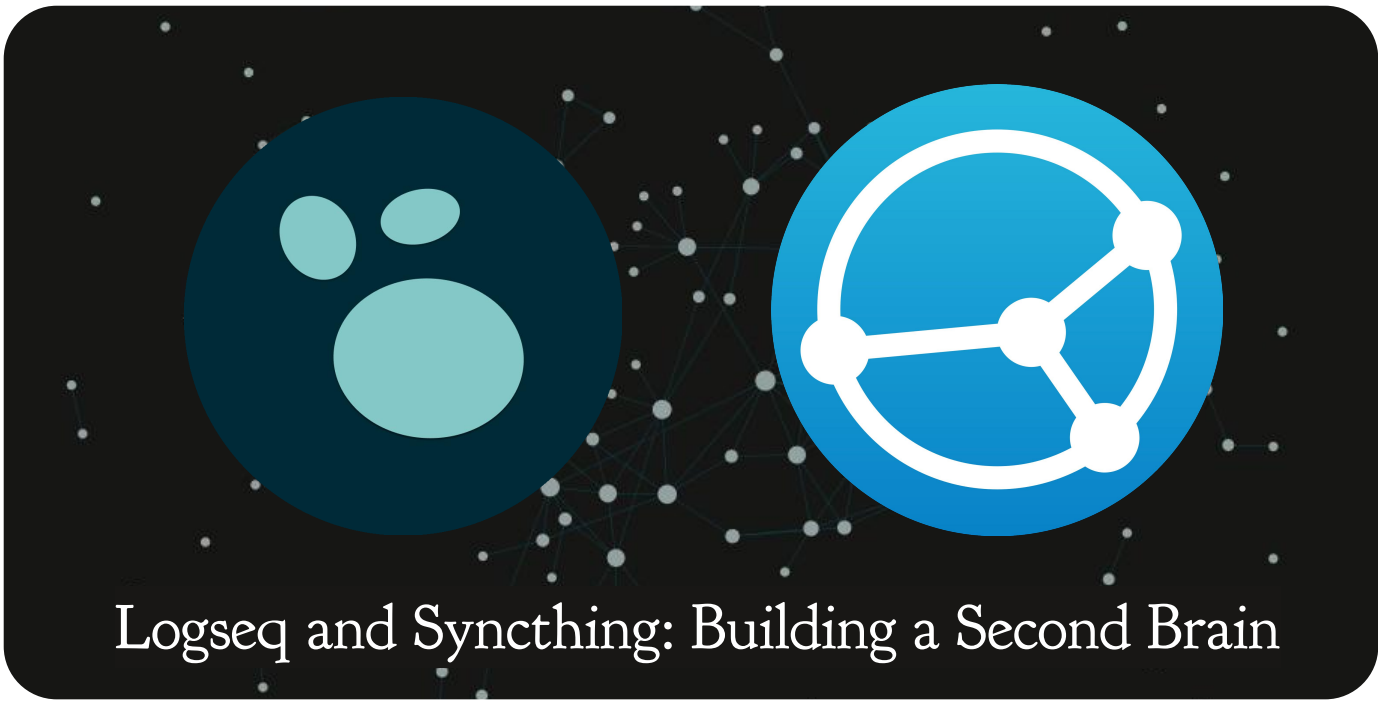


Page 13.Termux: Terminal Power in your Handheld Droid.

Page 16. Street Complete: Contributing to Open Street Map.



Page 18. Inkscape: Using the Corner Path Effect

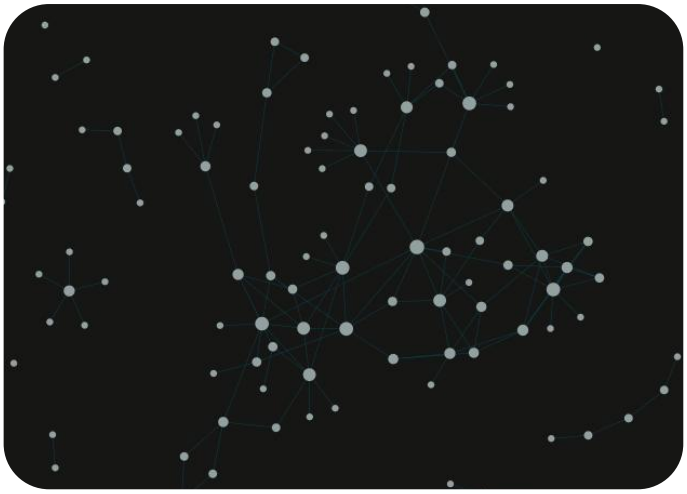


Logseq and Syncthing: Building a Second Brain

The heady blend of my neurology, my physical environment and my many project area's, I've become increasingly aware of needing supportive systems to try and keep me generally on track. Whilst I still fail at this in many ways, one little setup and combination of two applications has really made sense to me and been really impactful in my workflow? (sometimes the "flow" is more like "splutter" but "worksplutter" doesn't read as well).

The two applications are both free and opensource of course, and they are Logseq and Syncthing. Logseq is a note taker and outliner application and often gets discussed in sentences as a little "like obsidian" etc. I've always been interested in note taking (cue references to my continued obsession with pens and index cards) but I really got interested in finding an application for note taking that allowed me to build, ahem, a "second brain". This was inspired by reading Tiago Forte's book how to build a second brain. I don't do Tiago's entire process, it's similar to the older "Getting Things Done" type approaches of ubuquitous capture which is a little too much for me. However I've realised that a system that allowed me to collate and connect notes, as well as tag and retrieve stuff easily was very much needed. What wasn't immediately apparent was how Logseq would actually change how I viewed all my information management and impact in other areas, like enabling a more distraction free work environment.

Logseq can be operated as a journal. I have it on all my Linux machines as an app image and I can quickly open it and be presented with a today's date heading and a bullet point underneath at which I can start typing. On set up you set up a local folder where all your Logseq stuff will be saved. The files created by Logseq are pure markdown and it creates a simple set of directories in your local folder where it stores files. You can tag notes using a double set of square brackets with a string inside so [[important_quotes]] would create a tag for example. However clicking that tag will take you to a bespoke page for that tag where any linked references will be created. This is really fluid in use as if I know exactly where to tag an item I don't need to go to that particular page, I can just open Logseq and at the next bullet point in today's journal I can simply insert that tag and write or copy the data I want to add. Simple.



A good example of tagging information as a second brain is, well for me one of my most visited tags in my local directory is "commodity codes". So I sell a small amount of weird electronic stuff on my Tindie store. This includes an opensource robot platform, some screw switches for rocketry, some adapted switches for drop release systems, a nichrome burning circuit and more. I invariably sell most of this stuff outside the UK and as such each order requires one or more commodity codes, or tarriff codes, or export codes, that describes the tarrif sector the product sits in. hese are quite often hard to initially identify and it takes quite a lot of work to land on a code you feel confidant represents the product. Furthermore, I need different combinations of codes for each order and I also am not overrrun with orders, so I don't retain a memory of the codes. So a simple page "commodity codes" gets me to that info instantly. Interestingly though on that page I also have tags like [[HS tarrif]] [[tindie]] etc so that I don't have to rely on my wet brain producing the single tag, it can splurge out any number of things I might call those codes or recall a word in a related area.

There are a thousand other features in Logseq and I don't really tend to use many of them. One that I do use, and again has really changed computing for me, you can upload an asset into a Logseq note. To do this you start typing "/upload" and in the context menu that appears you select "upload an asset" then a file manager launches and you can select any file to add to the note, images and pdf's are previewed in the note directly. Interestingly what this does is it creates a copy of the asset in an "assets" folder in your local Logseq folder. For me this is perfect and although I use a more traditional folder on my hard drive for larger projects I actually now store huge amounts of general files in this folder. You can of course tag files and more and a single file may end up tagged across multiple relevant subject areas or pages.

Just for the sake of the story, let's skip to what Syncthing is and how it impacts into my setup. So one of the things proprietary note taking applications have is synchronising across systems. So for example, google keep, which I did use a lot historically, is synchronised across all devices signed into by the same google account, so often I'd make a note or paste a link on my phone and then review it later on my laptop. That's pretty crucial in terms of modern note taking and second brain stuff. Logseq in it's current form doesn't sync (Sync is on their roadmap and may signal a change in directon for the Logseq project). Up steps Syncthing. Syncthing is a free and opensource continuous file synchronisation application. It synchronises user specified files and

folders between user specified machines, using the internet, but never placing your files and folders on any external servers or clouds (aka other people computers).

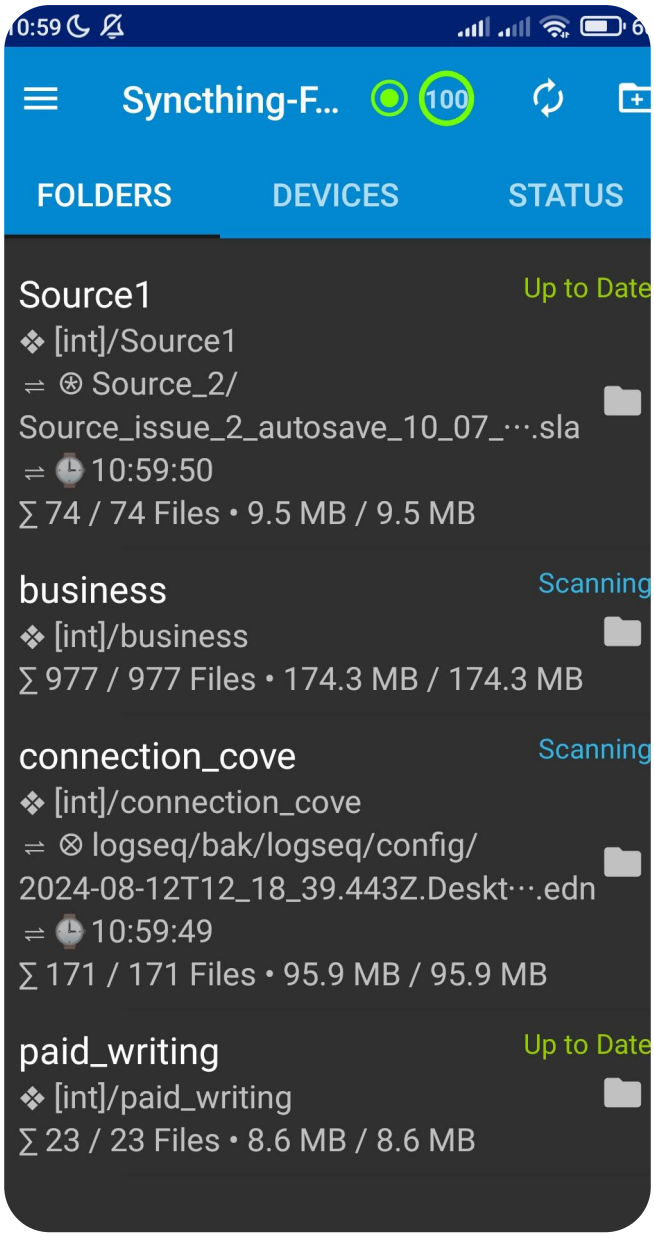
> Remember the old adage; "There is no cloud, it's just someone elses computer."

It does this in real time, and so as it never uses external servers you need to have the syncing machines online at the same time. I have Syncthing installed on most of my machines (pretty easy to set up on Debian and Arch machines) and I also have the recommended Syncthing fork android application set up on my phone (I also have the android version of Logseq installed). With Syncthing set up I can have my Logseq folder synchronising across my phone and numerous machines. It works incredibly well and the my phone is always on with the Syncthing service running and as it's my ubiquitous device it's often the glue that keeps my varied other machines up to date.

Whilst I ensure this isn't my only backup system, it is a very handy way to ensure backup of key files and folders, it means my Logseq second brain is always ready and up to date, but also has provided another interesting solution. I've become aware of how many distractions I am capable of falling victim too, I've realised, and this will be different for all of us, that a good solution for me is to have some work machines, that I never log into my distraction accounts, or even allow to have my passwords stored for my distraction accounts on. So the machine I write on, I have allowed myself to access my business email account, but I'm not logged in to my mastodon or other socials, but the machine has my Logseq and Syncthing set up on board. This means that I can park distraction or manage known personal entry points to distraction.

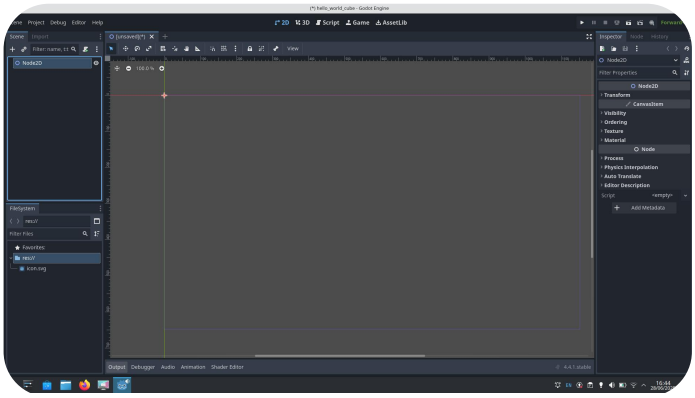
As an example, this morning I was working on an article about using a Raspberry Pi Pico, flashed with the Picomite firmware, a boot to basic computer, and interacting with it using a Picocom, a serial interface in the terminal emulator. I wanted to message my friend with a screenshot quickly to show them I had it up and running. However, I can't log in to my messaging applications on this work machine easily. So, I've uploaded a quick screenshot to Logseq which will be synced to my phone. This means later, when I allow myself, I can go retrieve my very distracting phone and the image will be ready to send to my friend in my gallery application.

I've really only scratched the surface of this useful combination of applications, I occasionally use Logseq to annotate PDF's and it's pretty stellar for this as you can highlight text, tag a connected note to the highlight and then when you return to the tag it will open the PDF on screen at the references point. Theres heaps more I haven't used... those are distractions for another day!

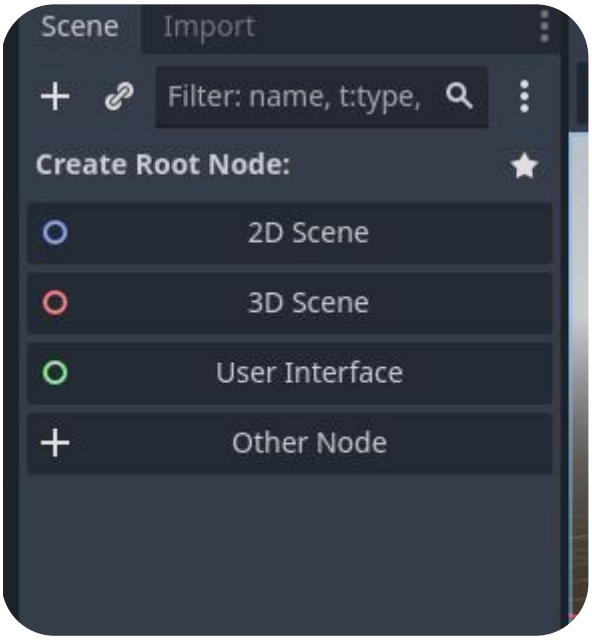


The Godot engine is a great free and opensource option for those looking to get started in game development. Let's dive in to a very fundamental getting started tutorial to explore just the basics. In this tutorial we are using debian 12 and the Godot download page offers us an executable download.

Once downloaded, a double click launches a getting started dialogue which recognises that you don't have any current Godot projects. Click the button to create a new project and give your project a name, note that the name you give create a new folder at the specified location rather than a typical "project file". That's because, neatly, Godot sets up a folder that contains the entire Godot toolset making it really easy to move projects around. Having created a new project you'll immediately land on a new project page with a 3D preview window. We aren't going to dive into a 3D project so our first thing is to left click on the 2D tab from the collection of 5 tabs in the centre of the screen.

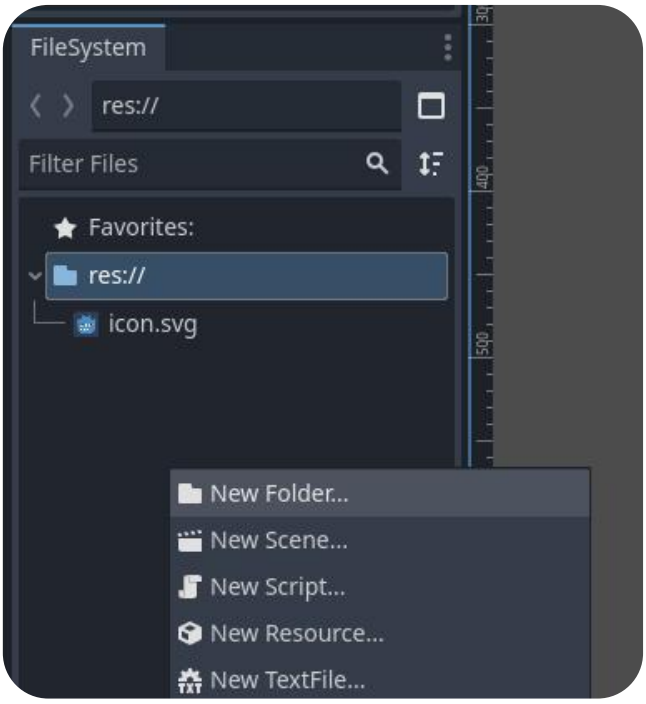


You should now see a 2D preview window. Godot, when working in 2D, places the X axis as left to right across the screen and the Y axis up and down the screen. In the upper left hand side of the screen you will see a set of tabs called "scene" and "import" with "scene" being currently active. In the dialogue there are some options listed under the title "Create Root Node". As we are going to create a very simple 2D scene click the "2D Scene" option to create a "Node2D" item.

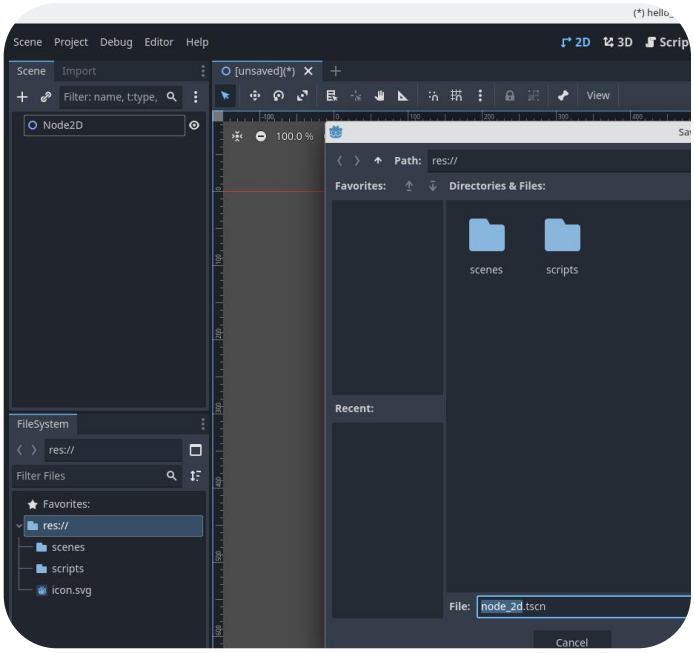


Before we start to do anything more exciting let's do some basic housekeeping for this project. On the lower left hand side of the screen you will see a

"FileSystem" tab inside which we will see a listing for "res://". Right click on this an in the context menu you can click the first option to "Create New" and from the expansion menu select "Folder". Call this folder "scenes". Repeat this process to create a second folder called "scripts".

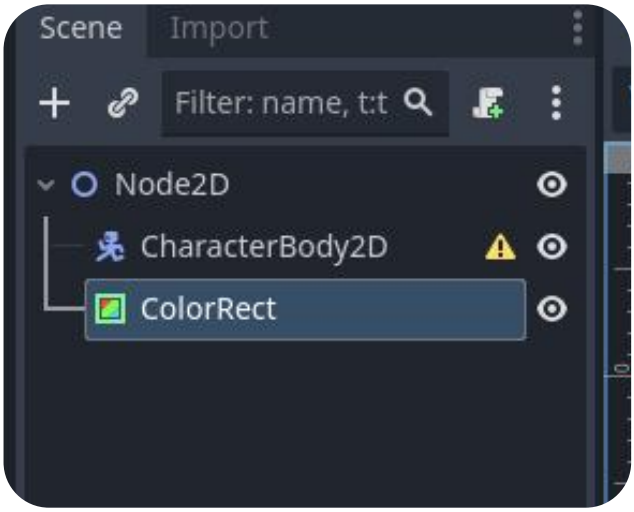


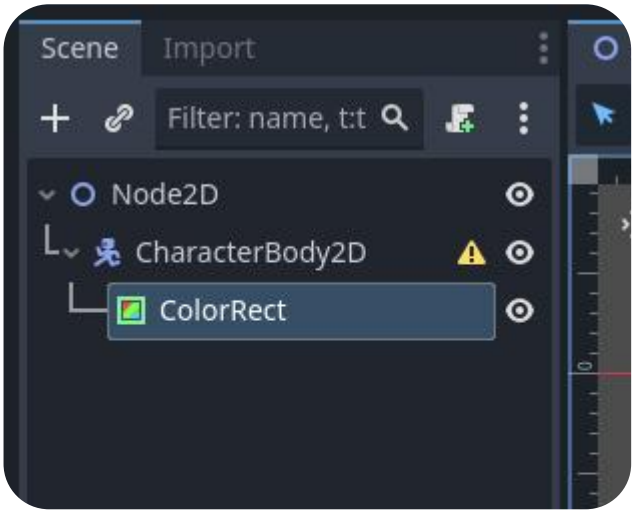
Once you have these two folders set up. Click away from the FileSystem tab by left clicking on the large preview window area. Then press "control and S" or alternatively navigate the main menu's by selecting "Scene > Save Scene". In the resulting window you should be in the "res://" directory and see the two folders you just created. Double click to enter the "scenes" folder and then change the default name of the file from "node_2d.tscn" to "main.tscn" and left click the save button.



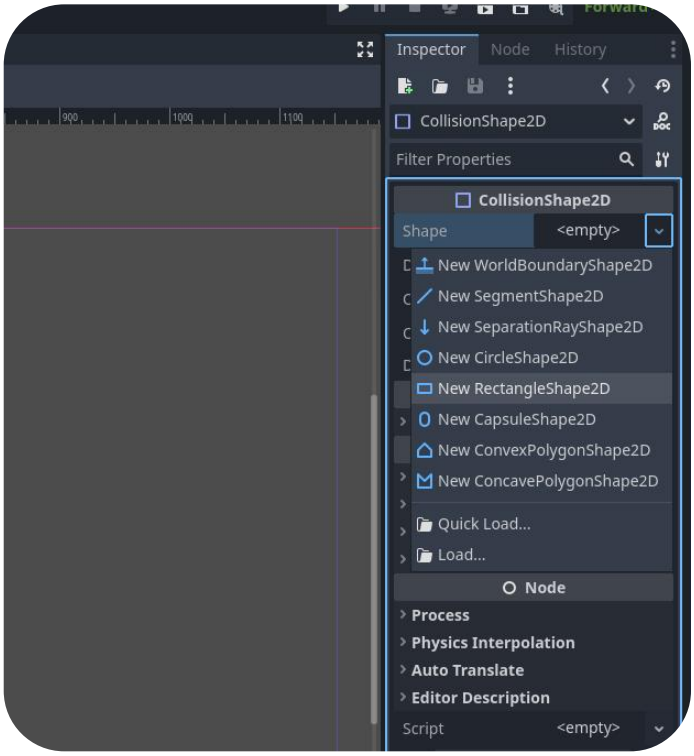
With the housekeeping a little out of the way let's now highlight our "Node2D" item and right click and then select "+ Child Node". You'll see a "Create New Node" dialogue appear with a long list of options. At the top of this window you'll see a search bar, let's use the search bar to search for a node type called "CharacterBody2D". Once identified in the list left click to select the item and click the "Create" button. This node will be our "Dot" that we will be able to move around our screen.

Let's add an object for to our as yet empty CharacterBody2D node. With the CharacterBody2D item highlighted in the "Scene" tab once again right click and select to add a Child Node. This time search for a "ColorRect" object to insert, yes you guessed it, a small coloured rectangle. Note that it's really easy to place child nodes in the wrong position in the hierachy, in the image you can see that the ColorRect child node has been accidentally added as a sibling of the "CharacterBody2D" node rather than a child. Make sure that the node hierarchy looks like the second image below.

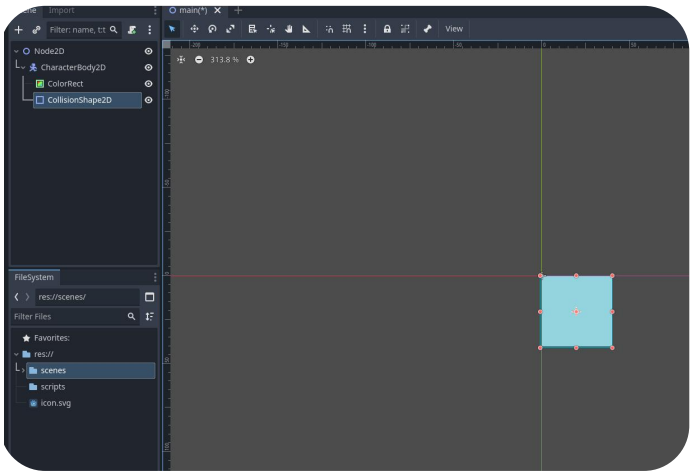




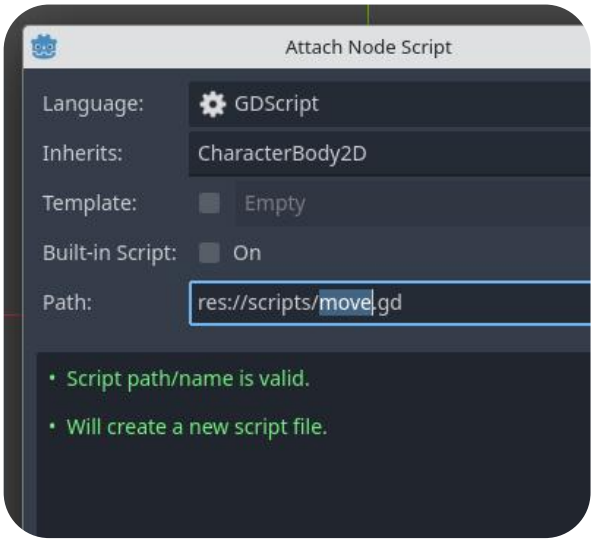
The next node we will add we do want to be a sibling of the ColorRect node we added as a child node of the CharacterBody2D. So highlight CharacterBody2D and then add a child node. This time search for "CollisionShape2D" and add this item. Note that as we have added child nodes they sometimes appear with a small yellow triangular warning icon. Hovering the pointer over these often gives tips as to what is needed to be added to the node to allow it to function. The CollisionShape2D node we just added has such a warning. Hovering over it with the pointer it suggests that a Shape Resource needs to be added. To do this we need to use the tabbed area on the right hand side of the screen. You should find a selection of tabs titled "Inspector", "Node" and "History" with the Inspector tab uppermost and active. With the CollisionShape2D child node highlighted in the Scene tab on the left hand side of the screen you should see a "CollissionShape2D" tab and dialogue open in the "Inspector" tab. There is a "shape" parameter which is currently "empty" and you click to open a dropdown menu. From the dropdown left click to select and create a "NewRectangleShape2D".



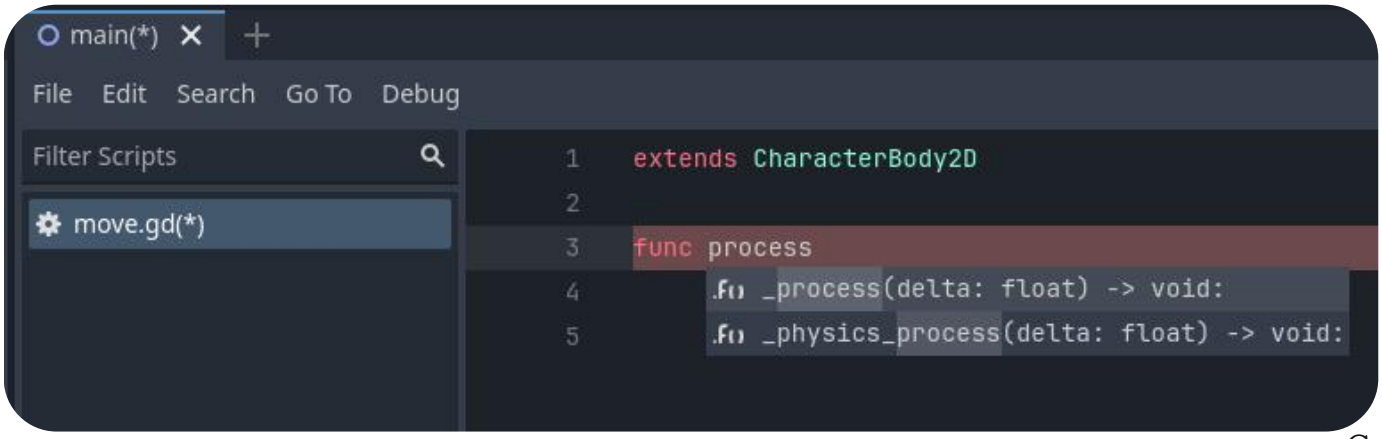
You should see a collection of 9 red dots appear close to your original ColorRect object. The dots are the corner and side handles and centre point of an adjustable rectangle shape. Grab these and move them so that the light blue rectangle directly covers the ColorRect rectangle placed previously.



Moving on we need to create a script connected to our CharacterBody2D node. To do this highlight the CharacterBody2D node item and then left click on the "Attach a new or existing script to the selected node" button found to the right of the filter input box on the Scene tab at the upper left of the screen. In the resulting "Open Script/ Choose Location" dialogue navigate into the "scripts" folder we prepared earlier and name the script "move.gd" then click "open". You'll now jump to a second window titled "Attach Node Script". We are going to leave everything as it is on this window and click "Create" however just note for future reference that in this window you can select a scripting language, and set which node in your project the script effects or which node "inherits" the script.

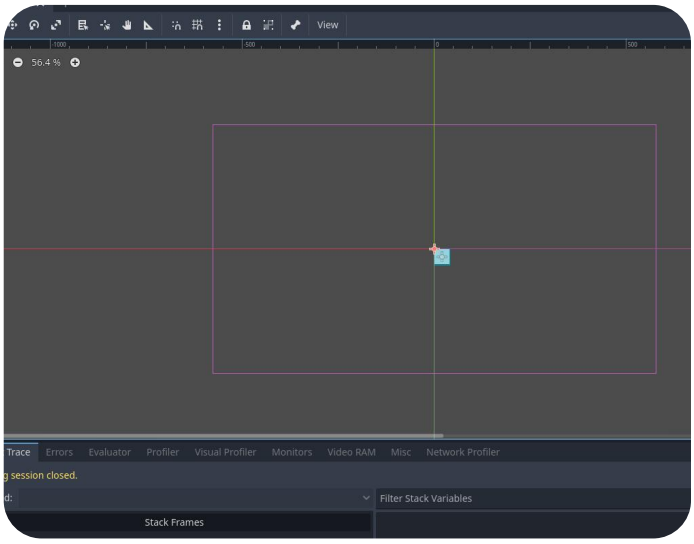


You should now see that the main preview area of the screen is the script editor environment and you have a new script loaded called move.gd and it contains the single line "extends CharacterBody2D". To show that scripts can be edited repeatedly we'll start to write a function and then go do another task and then return to the script.



Notice that the script editor has line numbers but Godot/GD script is not worries about empty lines. So use the return key to move a couple of lines below line one and let's begin to add a function.

Type "func" to declare a function and then start typing "process". You'll see Godot autosuggest completions for the line. Whilst this might seem a little overwhelming at first it does make it incredibly quick to work with as you learn more. You'll also see options that maybe intrigue you and you end up reading up about different functionality so it's a useful learning feature. We are going to select the first suggestion which reads "_process(delta: float) -> void:" . Note in later images that we have added comments to our simple script explaining each line. An inline comment is added by using a # symbol. This first line creates a function that will be processed in every frame of gameplay.

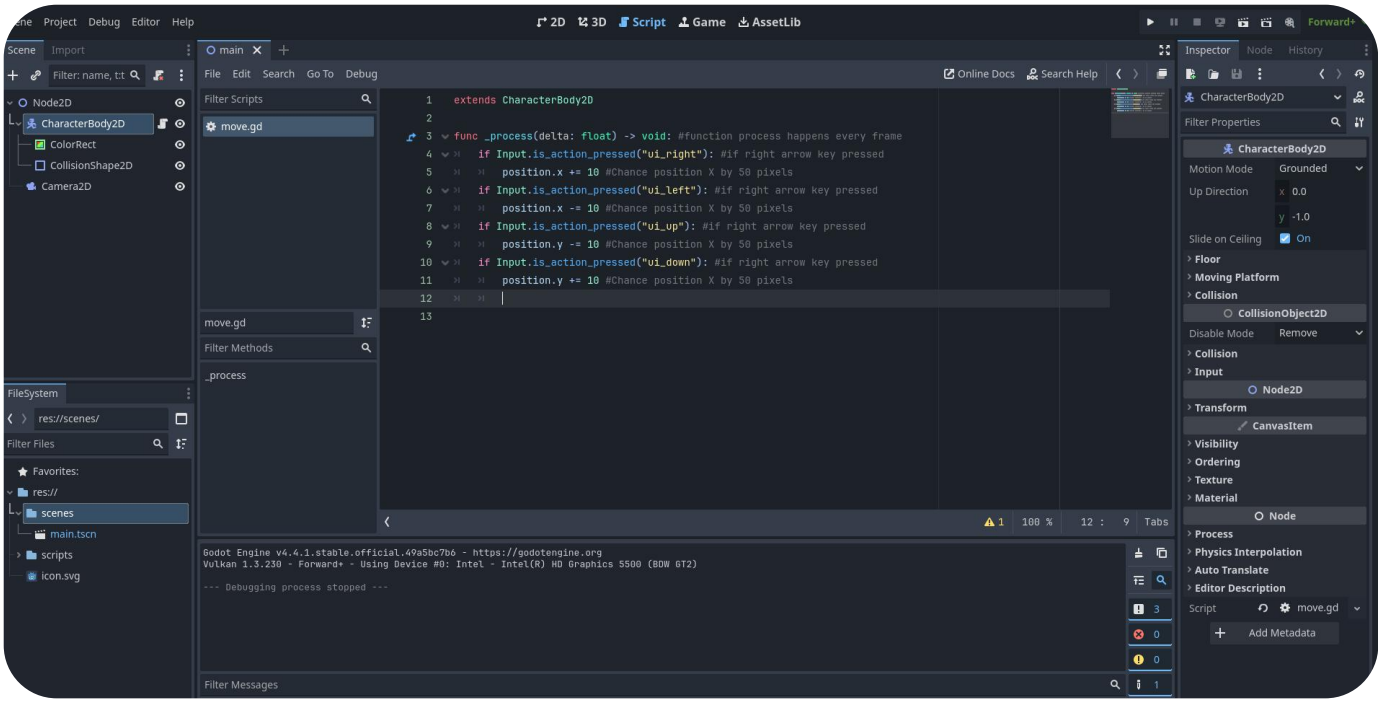


We can use the main set of tabs in the top centre of the screen to navigate through our project. When we created our move.gd script it automatically jumped to the "Script" tab whereas previously most of the time we had been working in the "2D" tab.

Click the "2D" tab once more and let's set a camera node up that will define the view area of our little project. This camera node needs to be attached to our uppermost main "Node 2D" parent node in out project. Left click to highlight "Node 2D" and then click the "+" icon to add a node. Input camera into the search bar and then select a "Camera2D" node item. This should now appear in the growing tree of nodes but directly connected with a line to the "Node 2D" object.

You should now also see a thin pink line appear in the 2D preview that forms part of a rectangle around the centred rectangle we made earlier. Don't worry if you can only see part of it. At the top of the 2D preview window we can see some tools, the currently selected one being a standard pointer icon. Further along to the right of this icon we can see a hand shaped icon. If we select this tool we can then pull the canvas around in the 2D preview and we can also use either the centre mouse wheel or button or the zoom tools to move in and out. Move the canvas so you can see the entire pink box that forms the camera view.

Next move back to the script tab and let's finish our small script. We are going to create 4 "if" statements inside our function that runs every frame and each if statement will detect if an arrow key is pressed and if it is move the rectangle in the corresponding direction. With the cursor at the end of the "func _process(delta: float) -> void:" press the return key and note that cursor will be indented to signify the following lines are inside the function.



Type the line;

“If Input.is_action_pressed(”ui_right”):”

Noting that almost all that line of text is available to select from the auto suggested options that pop up as you type. This line of code basically checks if a button, ui_right, which is the right arrow key has been pressed. At the end of the line press the return key and the next line should begin indented as the statement will be within the If statement from the previous line.

On this line type;

“position.x += 10”

This means that If the right arrow key is pressed then the position of the object inheriting this script (our rectangle) will move 10 pixels in the positive X axis direction. Next add virtually the same if statement but replacing the “ui_right” for “ui_left” in the If statement and then in the position line change “+=10” to “-=10” this makes the rectangle move in the negative direction of the X axis.

Continue to add 2 more If statement for the Y axis. You should end up with a script that looks like the text block below.

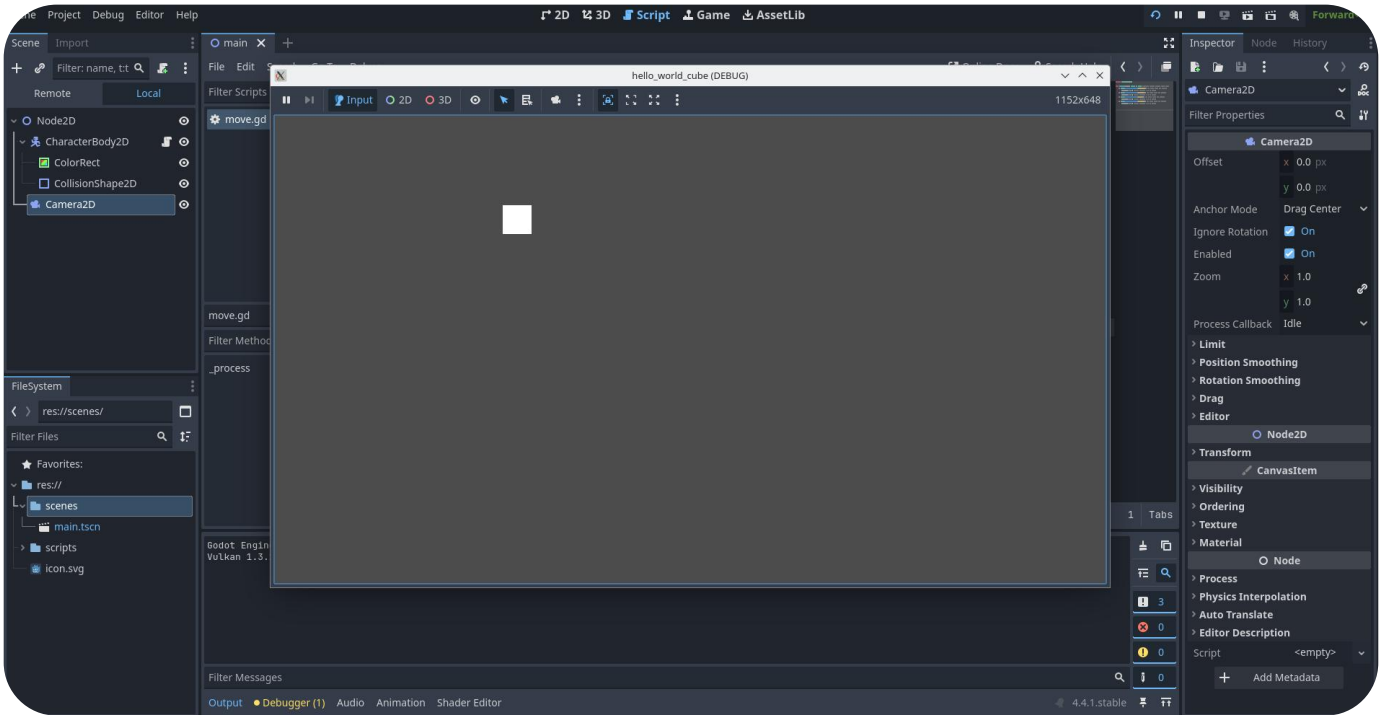
```
extends CharacterBody2D

func _process(delta: float) -> void: #function process happens every frame
	if Input.is_action_pressed("ui_right"): #if right arrow key pressed
		position.x += 10 #Change position X by 10 pixels
	if Input.is_action_pressed("ui_left"): #if right arrow key pressed
		position.x -= 10 #Change position X by 10 pixels
	if Input.is_action_pressed("ui_up"): #if right arrow key pressed
		position.y -= 10 #Change position X by 10 pixels
	if Input.is_action_pressed("ui_down"): #if right arrow key pressed
		position.y += 10 #Change position X by 10 pixels
```

Finally you can in the script window click “file > save” to save the move.gd script.

We should now be all set to run our small project. In the upper right hand side of the screen you can see a triangle shaped “play” button icon. Click this and you’ll get a small warning dialogue saying that “no main scene has ever been defined”, click “select current” and your scene/game should launch in a new window. You’ll notice it is the same size as the pink camera rectangle with your small rectangle in the centre. Toggling the arrow keys on your keyboard and well, you should be able to make the dot go with Godot!

Whilst this isn’t going to win you a game design award it’s certainly a good start and a peek under the hood of the Godot environment. Going further there are heaps of online tutorials and content on video platforms as well as written tutorials. If you are particularly interested in learning GDScript then a fun interactive learning environment is available online here.



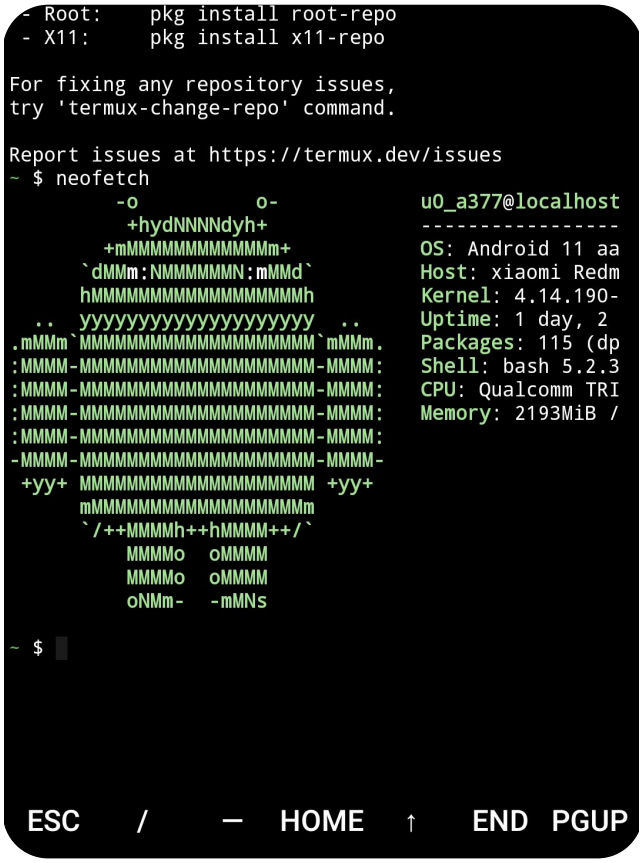
Termux:
Terminal Power in your Handheld Droid

I've played a lot with Linux phones, ran an Ubuntu touch device (a very old Pixel) as an everyday machine for a while, and tinkered with a huge variety of different distro's on my original Pinephone community edition. There's a few apps that keep me on my Droid device though. My business banking is totally app based and there's a few other bits and bobs that lure me back.

We also see in 2025 that, from version 15 of Android on Pixel phones, they'll ship with a native linux terminal application. If you aren't at the cutting edge of Android, and don't particularly feel the need for a full blown Linux distro phone then there is Termux.

Termux, is a terminal emulator for Android and it's frankly excellent. Over the years Termux availability on the Google Play Store has changed numerous times but it has continuously been available on platforms such as F-Droid. However, perhaps it's more simple to just seek out the most up to date apk file available on the projects repository.

Excellently you can run Termux on pretty much any Android device without the device being rooted, although there are some interesting applications for Termux if you do have a rooted device.

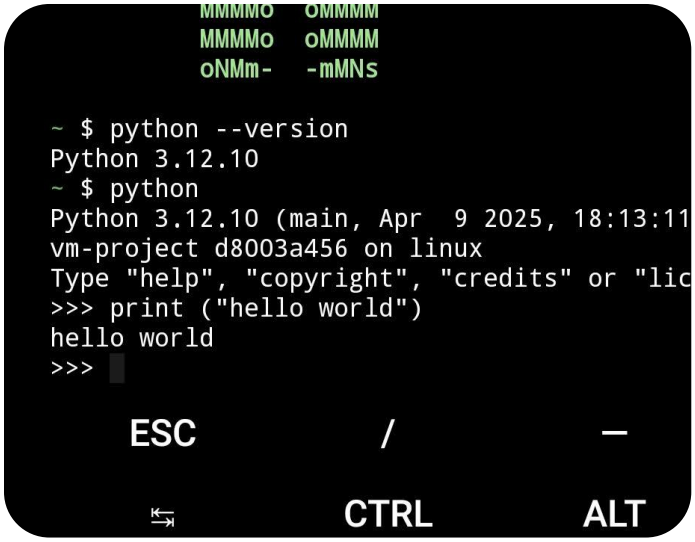


Once downloaded and installed you can launch into a familiar looking terminal. It's pretty easy to get going with but if you've used things like "apt" or "pacman" to install packages on other Linux systems you'll need to read the terminal header text that quickly describes the "pkg" command. You can use "pkg install <packagenamehere>" to install a package but perhaps more usefully you can do a "pkg search <query>" to search the 1000's of packages available for Termux.

A good test example is to run "pkg install neofetch" to install neofetch and then, once completed, you can type "neofetch" at the prompt to witness your system specification in that lovely neofetch style.

Beyond Neofetch there's lots to explore, but before you jump in too deeply it's worth running a storage setup command to allow Termux to create and edit files on your device. You can do this by running the command "termux-storage setup". This will now create a "storage" area on your device which you can locate using any file manager. Within the "storage" area you can then use familiar commands like "cd" and "ls" to navigate and list files and directories. After setting up the storage a neat idea is to run the "pkg list" command but put the output into a text file by using "pkg list-all > package_list.txt" you can run the command to list directly in the terminal but as a default Termux can only scroll back 2000 lines and this won't be enough.

With your storage sorted you can start to think about what you might use Termux for. I'm no dev, and not the greatest coder however it's super useful for me to be able to install Python and pip to tinker with python scripts whenever the mode takes me. You'll discover more familiar terminal commands that just work as you go along, when I installed Python I curiously after a successful install ran "python --version" and it did indeed correctly display the Python version (3.12) successfully.



Alongside Python it's useful to have some kind of text editor. I often use the ubiquitous "nano" to create and modify text files and scripts and find it a nice straightforward and distraction free way to sometimes create bodies of text. Nano runs happily in Termux (its pre-installed) and you can install other popular editors, vim, neovim or emacs for example.

Moving to areas I haven't explored, there are also options to use things like Proot and Chroot and to install x11 for a more fully featured GUI based Linux environment, looking around the scene people are running all manner of distros this way with varied results. For me I think that's an interesting exercise but probably not something I have a huge use for. Another area where Termux seems a popular solution is using it to SSH into other systems, again it's perfect to have a pocket device with this capability.

As a final thought, at first I questioned how good Termux would be on a small screen and whether it was really useful. I have to say it is a pretty good experience, it's fun to have a terminal that autorotates and it automatically adds some useful keys (page up, page down, tab, control, escape, home and more) on top of your regular screen keyboard. I tend to use the "hacker keyboard" application on my android devices with 4 lines so It's fair to say that it can get pretty cramped for screen real estate quickly, but it's certainly workable. For more extended Termux tinkering I tend to pair my cheap portable folding keyboard, which disables the on screen keyboard, then it's then a joyous, lightweight, "Tech Tuareg" rig that still fits in your pocket.

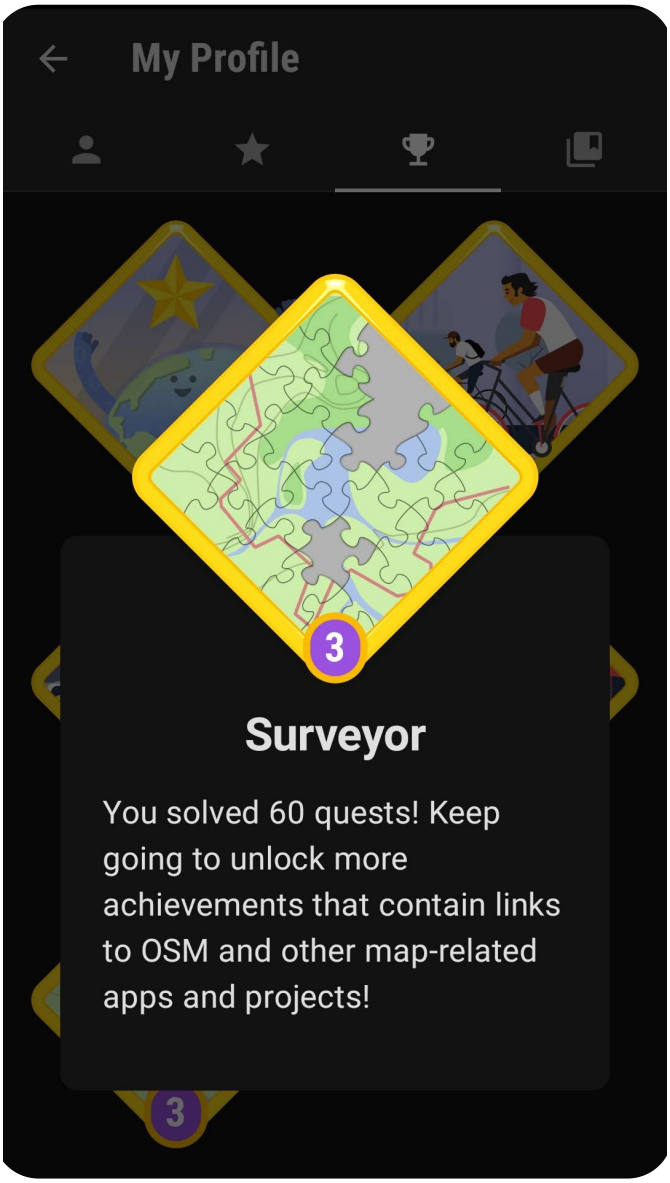
Street Complete: Contributing to Open Street Map

It can sometimes be tricky to work out how to contribute to an opensource project. Perhaps you aren't a developer, or perhaps time is a constraint that limits what you can do. As such it's always cool to share ways that we can incorporate contributions into our lives without much impact.

If you don't know Open Street Map (OSM) is an opensource mapping system that started in the UK and has rolled out to the world. OSM started in 2004 and it's become a hugely used project that attracts heaps of users and developers. Founded by Steve Coast the first map entry was surveyed by Steve riding around Regents park in London on a GPS equipped bicycle and since then many online and offline map editor environments have been created to add data to the project. However with many of us having GPS enabled phones with healthy data packages, mobile phones now make great sense in terms of quick, impactful, surveying.

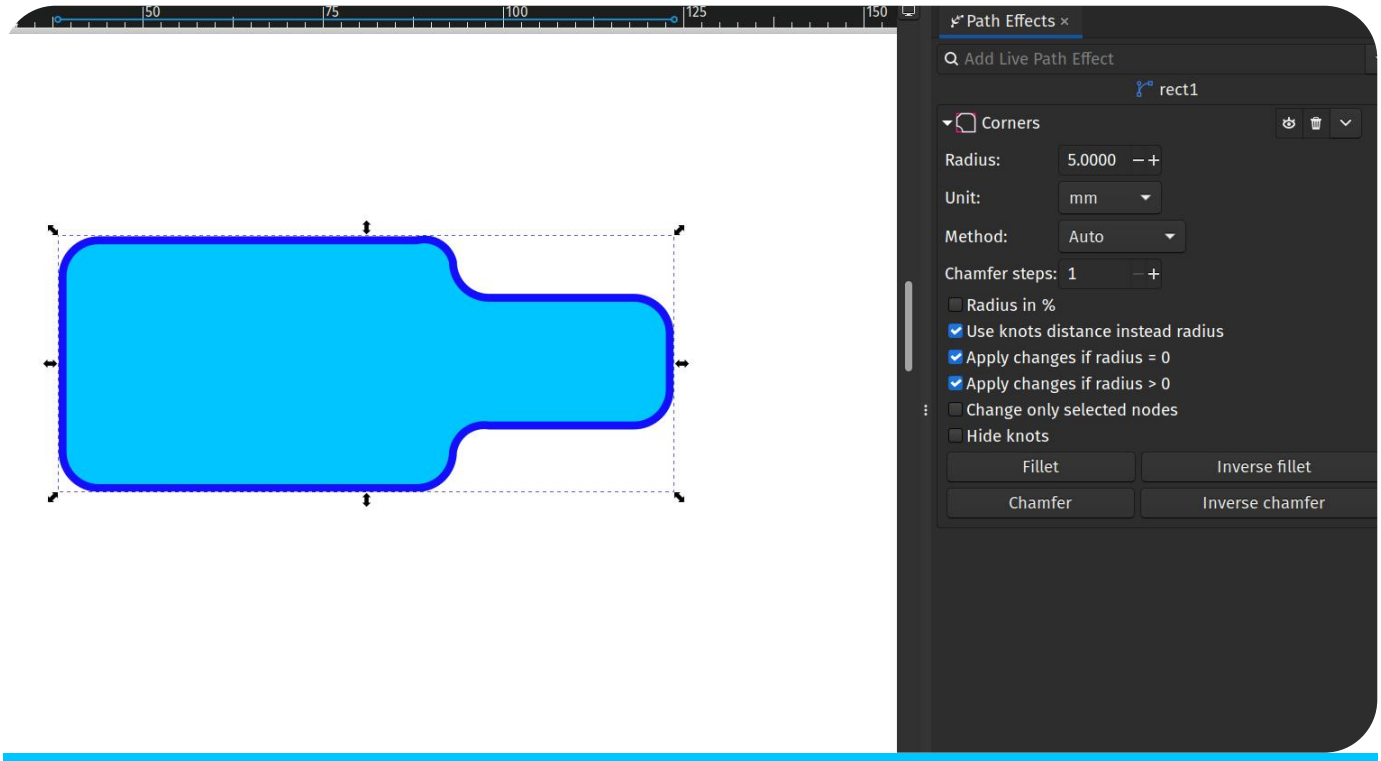
One way to achieve this is the "Street Complete" application on android. It's available via the Google Play store but, if you're avoiding there then it's also available via F-Droid or you can directly find .apk files on the Github repository release page.

Once you have the application installed, you'll need to give it a few permissions, turn on location services and you are ready to go. It's incredibly intuitive and needs little real explanation. You are presented with a map and your location and the map is populated with a lot of icons dotted around. Clicking on an icon will reveal a question about something at that location. These questions are varied but with lots of common simple ones like, "what is the surface of this road/path?", "What is this telegraph pole made from?" or things like "What times is mail collected from this post box?". For many questions there are multiple choice answers and you simply click an icon. For others there are text input boxes etc.



Whilst the gold standard is to fill in answers at the locations of the questions, you can also fill in answers not at the location, ticking a box to confirm that you realise you aren't at the location. This means you can actually get started usually from your/ desk or couch as it's pretty likely you'll know some answers for the questions local to your accommodation or place of work.

It's nicely gamified with reward medals and ranking systems and it does a good job of motivating you to contribute more. As you tick off question nodes you might think that you'll run out in the areas you can survey easily (for example my standard dog walking routes). Rest assured, more questions appear, whilst this might seem never ending, it actually means that the OSM for your area becomes incredibly detailed and well surveyed.

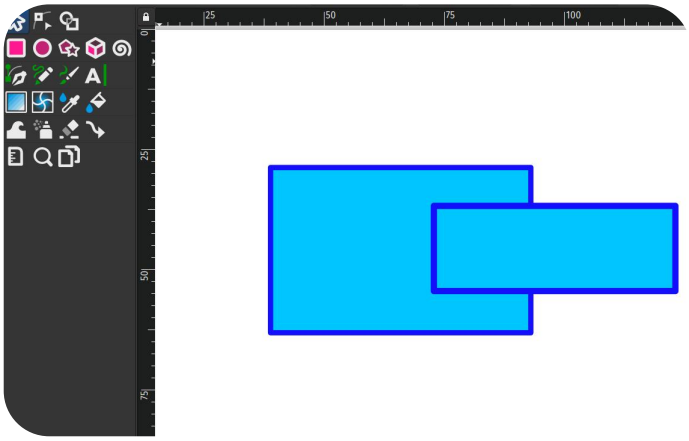


Inkscape: Using the Corner Path Effect

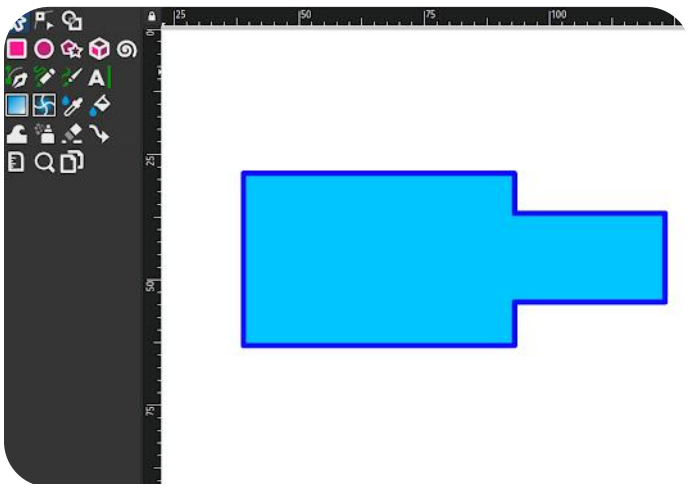
The Corner path effect in Inkscape can be added to reasonably complex paths to create radius corners. It's a relatively straightforward path effect to use. As an example lets create a shape with a pair of rectangles and then apply the Corner path effect.

To begin click the Rectangle Tool icon and then left click and hold on the canvas to start drawing a rectangle, drag the tool across the canvas to create a rectangle. Create a second rectangle and roughly arrange them as seen in the image.

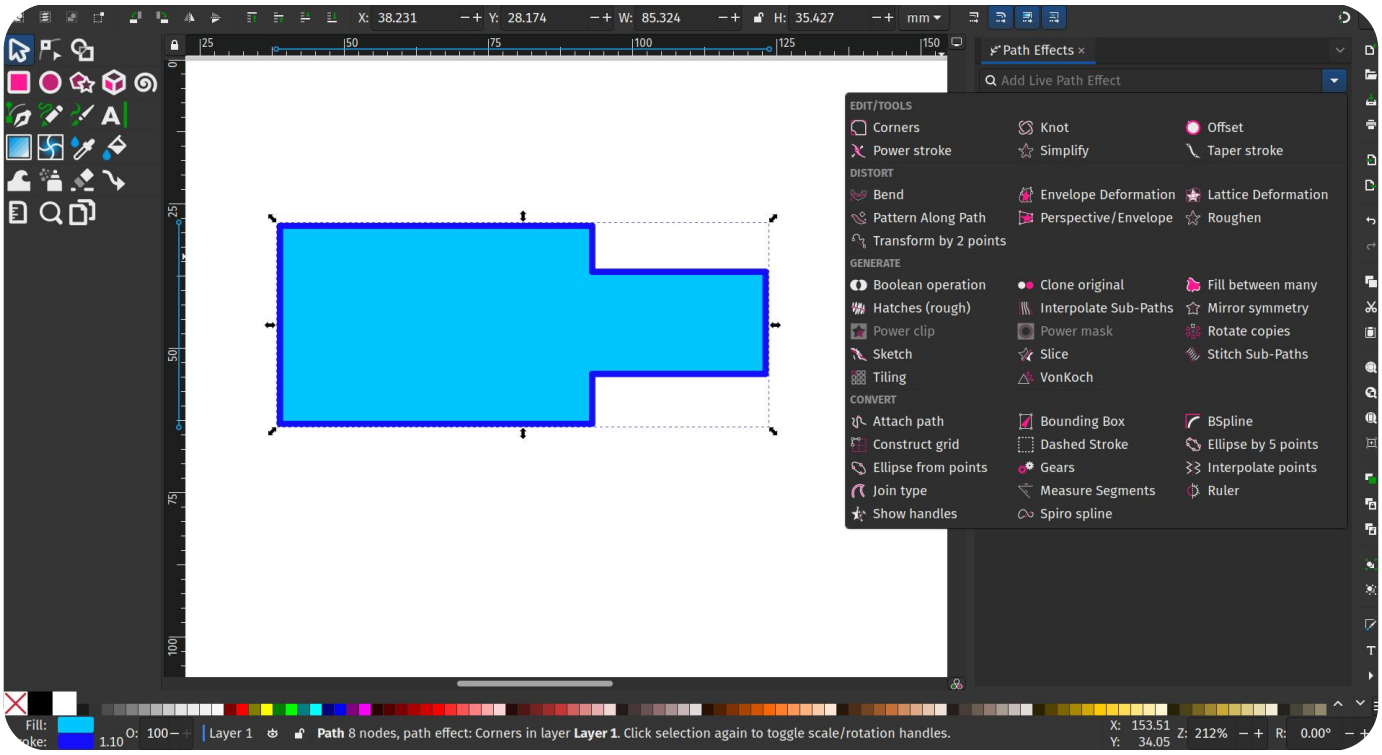
Quicktip: If we just want to add rounded corners to a single rectangle or square we can select the item with the Node Tool and then either drag a circular node or type a radius into the Rx or Ry dialogue boxes.



Next press "Control A" to select both rectangles and then Path – Union to make a single object from the two rectangles as shown in the image.

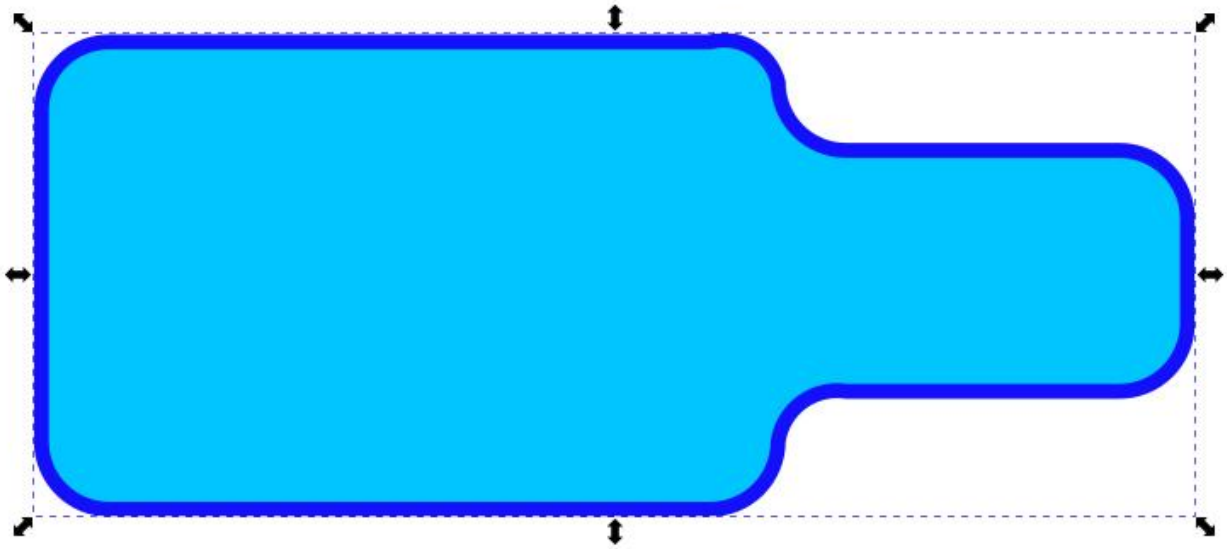


To add the Corner path effect first select the union shape we just created and then click Path – Path Effects to open the Path Effects tab on the right hand side of the screen. You should see in the tab that the object, named "rect1" is listed below a search input box which has a dropdown menu. Left click on the arrow at the right side of the input box and you should see the available path effects as shown in the image on the next page.



Left click to select the Corners path effect and you should see the dialogue with the parameters for the corners path effect in the right hand tab. As a simple example of applying a corner effect let's set the Units selection to "mm" and then type "5" into the Radius input box.

You should now see that the corner path effect is added to both the internal and external corners of our shape, you can see this in the header image at the top of this post. To remove the corner path effect at any time you can click the dustbin logo in the corners path effects dialogue.



Thanks for Reading!

Thanks so much for reading SOURCE. Whilst it's a lot of work putting these together, I'm enjoying learning new things and incrementally increasing my knowledge of Scribus which I'm using to do the page layout. I have a few thoughts for the next issue, I'd like it to have a general lean towards tools for open hardware and open hardware projects. If you do want to support SOURCE then please do consider paying what you feel it's worth. If I can garner a reasonable amount of donations then I can justify keeping going, (although I am commited to doing 4 issues whatever they bring!). If you aren't in a position to pay what you feel, then a great thing you can do is to share SOURCE. Either directly by any means, post it, host it, compress it and email it, or link to it. Finally... if you enjoyed something in SOURCE do feel free to comment or mention it to me directly. I'm @concretedog on mastodon.

Take good care and see you in issue 3!
Jo AKA Concretedog.

Pay what you feel via Paypal here

PayPal

Pay what you feel via KoFi

Support me on Ko-fi