

# Linux Setup

The setup process for UE4 on Linux is pretty straight forward, and the [official instructions](#) are very well documented. Follow those steps, and return here once you're done. Below, I just outline some of the issues I encountered using UE4 on Linux post-installation and how I solved them.

## Marketplace Assets

Epic Games doesn't seem to provide any support for the UE4 Marketplace for Linux. As a result, Epic also does not support downloading and adding any assets to your project. Bummer.

I would like to note a blog post on this same issue by [alexandra-zaharia](#), I would have very much preferred her solution but after attempting it I could not get it to work. I could install Epic Games through Lutris, but symlinking my projects (or copying them) into Wine Windows FS did not result in the Epic Games launcher detecting the projects.

What I ended up doing was using the **unofficial** [nmrugg/UE4Launcher](#) on GitHub. It works great, but I do miss some things like asset engine version information and browsing the marketplace.

You can install assets with it though, as long as you own them and they're linked to the Epic account you sign in with. That's all I need for now, I'm just playing with the idea of learning some C++ for UE4.

## UI Scaling

### [Related question on StackOverflow](#)

Initially the UI for Unreal was very large and practically unusable. This could be due to the fact that I'm running on an integrated graphics CPU, but I'm not sure. To fix this, I just opened a project and navigated to `Edit->Editor Preferences` and then unchecked the `Enable High DPI Support` option under the `General/Appearance` settings menus. After restarting UE4, everything was good and UI was normal again.

## Application Launchers

The shortcuts initialized by Unreal during the build process didn't work for me. I couldn't figure out why, so I just made my own. I placed these files into `~/.local/share/applications/`

You will need to change the `Path` value to point to the same directories on your local system for the following configurations.

For the Unreal Engine Editor, the following `UnrealEngine.desktop` file will launch a window to open or create a UE4 project

```
#!/usr/bin/env xdg-open

[Desktop Entry]
Version=1.0
Type=Application
Exec=UE4Editor
Path=/home/kapper/Code/Clones/UnrealEngine/Engine/Binaries/Linux
Name=Unreal Engine Editor
Icon=ubinary
Terminal=false
StartupWMClass=UE4Editor
MimeType=application/uproject
Comment=Open or create UE4 projects
```

And for the open source UE4Launcher, I created the following `UnrealEngineLauncher.desktop` file to run `npm start` in the UE4Launcher repository directory.

```
#!/usr/bin/env xdg-open

[Desktop Entry]
Version=1.0
Type=Application
Exec=npm start
Path=/home/kapper/Code/Clones/UE4Launcher/
Name=Unreal Engine Launcher
Icon=ubinary
Terminal=false
StartupWMClass=UE4Editor
MimeType=application/uproject
Comment=UE4 Project and asset management
```

## Hardlinks for Editor Layouts

In the editor you can create and save layouts under the `Window->Load Layout` toolbar menus. After creating a layout, it will by default be output into the `UnrealEngine/Engine/Saved/Config/Layouts/` directory.

I thought it'd be really useful to store the layout in the root directory of a project (tracked by github) and hardlink it from there to `~/Code/Clones/UnrealEngine/Engine/Saved/Config/Layouts/` instead.

This way, when you clone the project for development on another machine you keep your layout, and UE4 will still recognize it within the menus for loading and saving layouts.

To create the hardlink, I ran this command

```
In ~/Code/Clones/UnrealEngine/Engine/Saved/Config/Layouts/EditorLayoutKapper.ini  
~/Code/GameDev/UnrealProjects/ThirdPerson/EditorLayoutKapper.ini
```

## Debugging

See [Build Configuration documentaiton](#) for more information of the various build targets that are made available when generating a C++ project solution with UE4. Also see [Compiling Projects](#) for how to build and debug with Visual Studio.

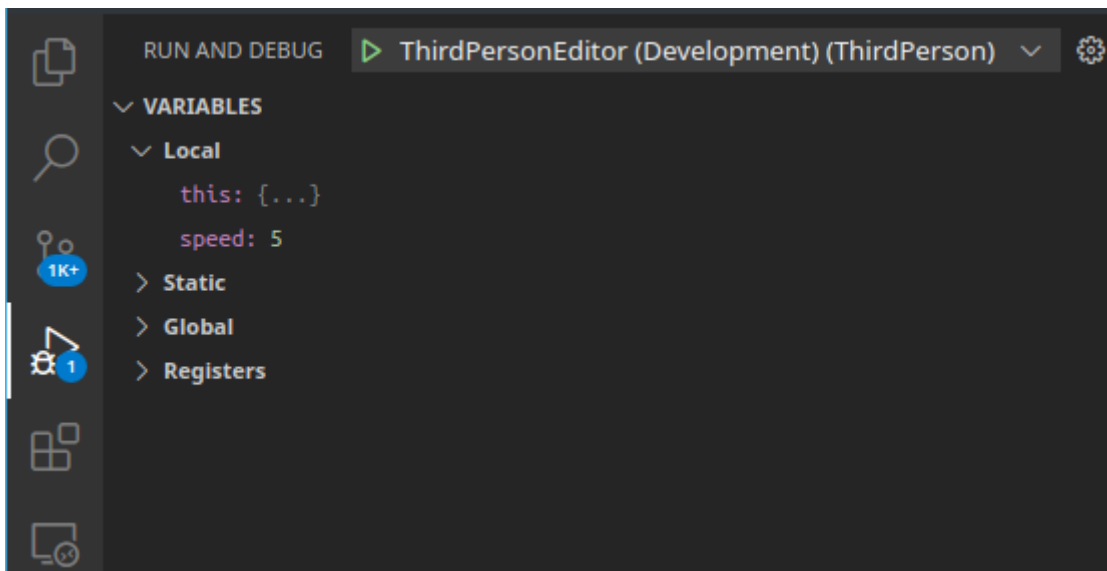
I watched [this video on YouTube](#) to learn how to debug UE4 projects using vscode. The instructions are for windows but it was very similar for linux. Just had to [install vscode on Linux](#) first.

Once you have vscode installed, open the UE4 editor as you normally would to edit your project. Then, go to `Edit->Editor Preferences` and navigate to the `Source Code` settings menu from the left. Select Visual Studio Code from the drop down. You'll get a prompt to restart the editor, which you should do before continuing.

Once the UE4 editor has been restarted, navigate to `File->Generate Visual Studio Code Project`, there will be some loading screen that appears and once that finishes you can go to `File->Open Visual Studio Code`

Once inside our UE4 vscode project, push `CTRL+SHIFT+B` and run the `<YOUR_PROJECT_NAME>Editor (Development)` build task. This task was defined for us by UE4 when we generated our vscode project in the previous step, and the entry within my `launch.json` is seen below.

```
{  
  "name": "ThirdPersonEditor (Development)",  
  "request": "launch",  
  "preLaunchTask": "ThirdPersonEditor Linux Development Build",  
  "program": "/home/kapper/Code/Clones/UnrealEngine/Engine/Binaries/Linux/UE4Editor",  
  "args": [  
    "/home/kapper/Code/GameDev/UnrealProjects/unrealgame/ThirdPerson.uproject"  
  ],  
  "cwd": "/home/kapper/Code/Clones/UnrealEngine",  
  "type": "lldb"  
},
```



Running this task for the first time could take a bit of time depending on the size of your project. When this finishes building, it should automatically start a new UE4 editor with your project open and in debug mode. If it doesn't, just select the `<YOUR_PROJECT_NAME>Editor (Development)` debug configuration in the sidebar and click the run button.

I also had to run the following command to install a missing dependency. The first time I tried to build there was an error and after reading it I traced it to this package and installed. The development build was successful after installing.

```
sudo apt install libopenvr-dev
```

Now you can run the same debug configuration and once the UE4 editor launches make sure whatever script you're debugging is active within your scene or your code breakpoint will not be hit.

Debugging on Windows under Visual Studios is very well documented, so while I do also have a windows development setup for UE4 I wont cover that here.

No luck with CLion, I seen memory usage during the build jump from 6GB to 16GB, plus an additional 4GB of swap space was also ate up.

Possibly a better solution: [Running UE4 from Qt Creator](#)

Since writing this, I've been using [Rider for Unreal Engine](#), It's in EA right now so you have to apply for access but I was accepted almost immediately so you shouldn't have any issues.

---

Revision #12

Created 11 January 2022 19:15:29 by Shaun Reed

Updated 13 January 2022 01:08:22 by Shaun Reed