Running Your Project Using Windows Subsystem for Linux (WSL)

With WSL you can run Linux natively on Windows. Follow these instructions to start your Django project inside a Linux environment on Windows.

Please be aware that setting this up can take an hour or longer. So, follow all the steps carefully to prevent surprises!

Install WSL

Open Powershell and run:

```
wsl --install
```

This will download Ubuntu Linux (the most popular Linux distribution) by default and will take a few minutes. If nothing happens, you need to explicitly specify Ubuntu:

```
wsl --install -d ubuntu
```

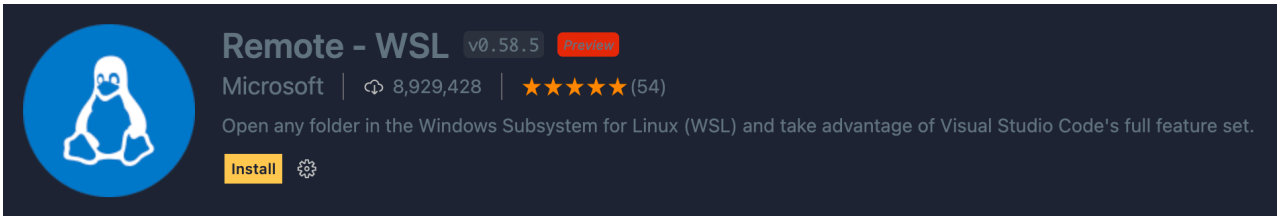
Once the installation is complete, you'll see a new terminal window running Linux. The first time, you'll need to setup an account to log into Linux.

Once you set up your account, you'll see Linux command prompt indicated by a $.

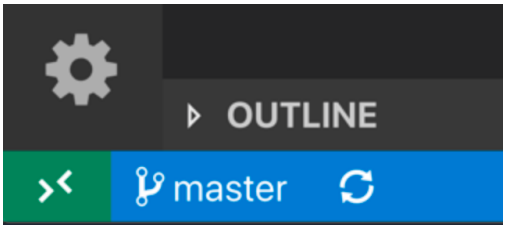
You can run pwd to print the current working directory. Note that Linux has a different file system and path than Windows.

Install VSCode Extension

The next step is to tell VSCode to use WSL as your integrated development environment. To do that, open the **Extensions** panel and install **Remote - WSL** extension from Microsoft.



This extension adds a few commands to VSCode. Open the **Command Palette** and type **Remote-WSL**. Alternatively, you can click on the green remote indicator in the lower left corner of the status bar.



From the list of commands, select **Remote-WSL: Reopen Folder in WSL.**

This will re-open our project folder in WSL. The first time, it may take a few seconds until the required components are downloaded.

The green remote indicator on the status bar will then show that your project is running on **WSL: Ubuntu**

Open a terminal window in VSCode. You'll see the command prompt has changed from Windows to Linux style (starting with $).

Update the Package List

On Linux, we use APT (Advanced Package Tool) to install or uninstall software, just like how we use Pip in Python projects.

As a best practice, we should always update the current packages.

```
sudo apt update && sudo apt upgrade
```

Sudo allows us to elevate the current user to have root (admin) privileges. This is required for running certain commands like updating the package list. Using **&&**, we can chain two commands.

After a couple of minutes, you'll be told how much more space will be required to upgrade these libraries. After that, allow 15-20 minutes for the update to complete. Depending on your connection speed, updating may complete in less or more time.

Upgrade Python to 3.9

If you open the **Pipfile** in our project, you'll see that our Project requires **Python 3.9** or higher. Our Linux environment, however, may include an older version of Python. Run the following command to see the version of Python installed:

```
python3 --version
```

If you see Python 3.8, you need to upgrade Python to 3.9:

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

```
sudo apt install python3.9
```

Here, first we tell Linux where it can find the Python 3.9 package (by adding the repository) so we can install it using APT.

Verify Python 3.9 is installed:

```
python3.9 --version
```

If you run "python3 —version" again, you'll still see version 3.8. That's because the **python3** command is linked to **python3.8** binary. To change the link, we need to remove it and add it again:

```
sudo rm /usr/bin/python3
```

```
sudo ln -s python3.9 /usr/bin/python3
```

Now, check the default version again:

```
python3 --version
```

Install pip and pipenv

```
sudo apt install python3-pip pipenv
```

Verify that **pip** and **pipenv** are successfully installed.

```
pip --version
```

```
pipenv --version
```

Install MySQL

Our Django project uses **mysqlclient** to talk to MySQL. In order to install this with pip later, first we need to install a couple of packages in our Linux environment or the installation of mysqlclient will fail.

```
sudo apt install python3.9-dev
```

```
sudo apt install libmysqlclient-dev
```

We could install both these packages in one go but I deliberately split the command into two so you ensure each package is successfully installed. To verify, run:

```
pip install mysqlclient
```

You shouldn't see any errors. If you do, stop and troubleshoot. Remember, Google is your best friend!

Next, we install MySQL server:

```
sudo apt install mysql-server
```

Verify MySQL is installed correctly:

```
mysql --version
```

Start the server:

```
sudo service mysql start
```

Configure MySQL

Connect to MySQL using the root user.

```
sudo mysql -u root -p
```

By default, the root user doesn't have a password. So press Enter when asked for the password.

Create a database called **storefront3**. Note the **;** at the end of the command. If you forget that, the prompt will break into a new line where you can enter a ; to terminate the command.

```
CREATE DATABASE storefront3;
```

Next, set the password for the root user. Note that both **root** and **localhost** are surrounded by single quotes.

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'P@ssword';
```

Type **quit** to exit.

Here, we set the password of the root user to **P@ssword**. Open **settings.py**, and set the same password in the **DATABASES** setting.

Start the Project

Congratulations! You've done all the hard parts. Now, we're ready to start our project.

First, install the project dependencies:

```
pipenv install
```

Activate the virtual environment:

```
pipenv shell
```

Run the migrations:

```
python manage.py migrate
```

Optionally, seed the database:

```
python manage.py seed_db
```

Start the web server:

```
python manage.py runserver
```

Now, your application is running on WSL. Next time you open VSCode, your application will continue to run on WSL.

Remember, every time you open a terminal Window, you need to activate the virtual environment:

```
pipenv shell
```