Setup your environment on a Windows 10 machine

This tutorial draws heavily from this terrific guide by Michael Treat.

Installation of WSL

For this setup you need a Windows 10 machine, with all updates done.

Enable WSL Feature in Windows.

First we need to enable WSL in Windows.

- Right click on the start menu and click on Settings.
- In the Search box, type Turn Windows Features On Or Off and click on the item that populates in the list.
- A window will pop up with a list of folders with checkboxes next to them. Scroll down and check the box for Windows Subsystem for Linux.

This will install the needed files. Follow any directions that pop up and restart your computer when asked.

Install the Ubuntu app from the Windows Store.

- Go to Microsoft store and install the Ubuntu App
- Follow the on-screen prompts to install the app.
- When the app is ready, the button that said 'Install' will change to say 'Launch'. Click Launch. This will start the Ubuntu installation. This installation only happens the first time the app is launched. It's the actual Ubuntu (or Linux) OS installing and mounting to your Windows FS.

WARNING	Anytime you uninstall the app and reinstall it you will lose any data that lives on the Linux Filesystem. This inlcudes databases, configs, `.profile's, and anything else you might have stored on the NON-Windows Filesystem. Make sure to back this data up!

Finish Installing the Ubuntu App.

Then we need to setup your user name and password for Ubuntu.

- It will ask you to enter a username. This will be the root / admin user for the Ubuntu FS. Don't use capital letters in this name.
- It will then ask you to enter and confirm a password. Also note that it will protect your

password by not displaying it to the screen when you type, but it is registering your key strokes.

NOTE Security is important at all levels, so even though you have to use this password often, don't be tempted to make it too simple (as I suggessted in an earlier version). Essentially all of your Window's files can be viewed and modified by this user, so keep that password safe and strong.

Finally, the prompt will change and you will be on a command line. Type pwd to see where you currently are on your machine, you should be at `/home/<your username>`. This is the root level of your Ubuntu user.

Updating the `.profile` file

In order to change how your terminal looks, we need to add some code to a file that lives in your Ubuntu user's root directory.

1. Open the Ubuntu app and type `ls -a`. You should see a `.profile` file there. If not, then type `sudo touch .profile`.
2. Type `sudo nano .profile`. This will open the file in the command line editor Nano.
3. Copy and paste this code into the editor. You can paste with right-click:

```
# This allows you to switch between the Ubuntu root and your Windows Root.
# wr evaluates to the absolute path to your Windows user's root.
export wr=~/../../mnt/c/Users/<Windows Username>/

# This gives us a quick way of moving directly to the Windows root
alias cdwr='cd "$wr"'

# This brings you to your Windows Working directory immediatly when you open a new
terminal.
cdwr
```

After pasting that in, you will need to add your Windows username right after `/Users/`, e.g., `/Users/MyName/`.

If your path has a space, you can use an backslash escape character to include the space, e.g., `/Users/My\ Name/`.

After that you're done in this editor, so press `Ctrl-x` at the same time to quit. It will ask if you want to save changes. Hit `y` and the editor will save your changes. It will then ask what to name the file. Just hit enter to keep the same name.

Close the Ubuntu app and open it to enable your changes.

Extra Info

In the file you pasted there was a section at the bottom that exports `wr` and sets up the `cdwr` alias.

This add the variable `$wr` to your environment. If you want to use a relative path, but don't want to go all the way up to the Ubuntu FS and then work down to the Windows FS, you can use the `$wr` variable as a shortcut to the Windows root instead, e.g.:

```
cd $wr/about_me/scripts
```

Instead of:

```
cd ~/../../mnt/c/Users/MTreat/Development/about_me/scripts
```

This also works with tab completion as well, which is awesome.

- `cdwr`

Now when you type `cdwr` it will bring you to the root of your Windows User! This makes navigating between the two file systems super easy.

- To navigate to the Ubuntu root, you will type the normal `cd ~`.
- To navigate to the Windows Root, you will type `cdwr` with NO space!

If you decide to add a directory to your Windows User's root to hold all of your work, e.g., `/Users/MichaelLeonTreat/Development`, you can come back to this file and update the `export wr` line so that it moves directly into that directory. Just add the name of the directory to the end of the path after your username.

In case you ever need it, the Ubuntu FS lives on your Windows FS on the path that looks very similar to this:

`C:\Users\<user>\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs`

If you want to create your own custom command line prompt you can check out http://bashrcgenerator.com or http://ezprompt.net and use the code that provides instead of the code here.

Install Git

Git is a version control system that allows you to track your projects' changes over time, and allows for an extremely collaborative process to exist.

1. Visit https://git-scm.com/ to download and install Git.
2. Follow the onscreen instructions.
   - Choose the default values for each prompt
3. Continue choosing the default options to finish the installation.

NOTE Git for Windows also comes with a terminal called Git Bash. This is what a lot of Windows users have used in the past as their solution to the POSIX / Unix-like terminal problem. We will be using the Ubuntu app instead.

Verifying Git

Now that we have Git installed on both of the file systems, lets check which Git Ubuntu is using.

1. Open a new terminal (the Ubuntu App) and type `whereis git`. This will show you all the places git is on your computer.
2. Now type `which git`. This will show you which git is executed when you type `git`. Notice that it only shows the one in Ubuntu - that is the git that will be used when you are in your terminals.

Set the Git Config

Create an account on GitHub.

Then add your email and name to the Git config. This will allow you to commit and push things to GitHub. Make sure to include the space after `.email` and `.name`, and always remember to close your quotes ' ' and " ".

1. Type `git config --global user.email 'your email here in single quotes'`.
2. Type `git config --global user.name 'Your Name In Single Quotes'`.

Once you are done, type `git config -l` and verify that it has your name and email saved correctly.

Set up SSH Keys

SSH stands for "secure shell" and is a protocol for encrypting communications with remote systems. You will need an SSH key pair which can be found in two text files that contain your "public" and "private" keys. We will add your public key to your GitHub settings so that you can securely communicate with the GitHub.com server without having to authenticate with a username and password.

- Open a terminal
- Execute `ls ~/.ssh`
- If you see `No such file or directory`, then execute `ssh-keygen`, hit `Enter` to accept all defaults
- You should have two files at least like `id_rsa` and `id_rsa.pub` which represent an SSH key pair. The `.pub` file contains the *public* key which you need to copy to GitHub. The other file is your *private* key that you should never touch, copy, use, or email. If that key is compromised, you should delete the files and use `ssh-keygen` to generate a new pair. You can generate as many pairs as you like, saving them into different files.
- Copy the contents of `~/.ssh/id_rsa.pub`. Otherwise, you can use `cat ~/.ssh/id_rsa.pub` to "concatenate" the contents to the screen, then copy the text to your clipboard.
- On GitHub.com, go to your user settings by clicking on your name/icon in the upper-right corner

to view a drop-down list. Click on "Settings" (2nd from the bottom of the list).

- In the left side, there is a table. Click on "SSH and GPG keys".
- Click on the big green "New SSH Key".
- Give your key a name like "laptop" and paste in the *public* key value.
- Click the big green "Add SSH key" button.

Forking the repo

- Go to https://github.com/kyclark/tiny_python_projects
- Click on the "Fork" button in the upper-right of the page, just below the big black bar.
- Fork into *your* repository.
- Verify that you have something like https://github.com/id/tiny_python_projects where `<id>` is *your GitHub username*.
- Go to the page of the repo and clone the repo on your computer using the command git clone `<ssh_clone>`

NOTE If you get a permission error, try to run this:

```
$ cd
$ sudo umount /mnt/c
$ sudo mount -t drvfs C: /mnt/c -o metadata
```

Then go back to the Windows subsystem (cdwr command) and try to clone again.

Set an upstream repository to get updates

In order to get updates from my GitHub copy of the repo, you must set it as an "upstream" repository.

Change into your local repo checkout, e.g.,

```
$ cd /tiny_python_projects
$ git remote add upstream https://github.com/kyclark/tiny_python_projectw.git
```

Install Python

To install python run the following at the WSL:

```
$ sudo apt update && upgrade
$ sudo apt install python3 python3-pip ipython3
```

Install PyCharm

NOTE you can only use and invoke PyCharm for the files in the Windows filesystem (also accessible form the WSL at /mnt/c/Users/<user-name>).

PyCharm is a complete IDE for Python, allowing you to run directly the code through the interface. It has a lot of options and possibilities, but is more difficult to use. Alternatively, you can install Notepadd++, which is a good old notepad, that will allow you to edit your code (see below).

- Download the community edition and install PyCharm.
- In order to enable interactive coding you should also have python installed in Windows. Go to the Microsoft store and install the latest Python.
- For each project you create you'll have to add the python interpreter properlt for PyCharm to run. If you have issues with the interpreter, see this help page.

Create the alias to launch PyCharm from WSL

1. Open a new terminal (the Ubuntu App) and type `whereis git`. This will show you all the places Git is on your computer.
2. Open your bash configuration:

```
$ nano ~/.bashrc
```

3. Add to the end of the file:

```
$ alias charm="/mnt/c/Users/<user-
name>/AppData/Local/JetBrains/Toolbox/apps/PyCharm-P/ch-
0/<version>/bin/pycharm64.exe"
```

4. You're done in this editor, so press `Ctrl-x` at the same time to quit. It will ask if you want to save changes. Hit `y` and the editor will save your changes. It will then ask what to name the file. Just hit enter to keep the same name.
5. Update your bash profile:

```
$ source ~/.bashrc
```

Now you can use `charm .` & to open PyCharm projects from WSL.

Install Notepad++

To download and install Notepad, go to the download page. And follow the instruction.

You’re done! Good job!