Environment Setup (Windows)

The tips below may be helpful in setting up your environment and getting term 2 projects up and running. **Windows 10 Users please note that Ubuntu BASH is the recommended option.**

Ubuntu BASH on Windows

Steps

- Ensure all dependencies are present per project resources
- Follow these the instructions in the uWebSocketIO starter guide which includes instructions for setting up Ubuntu BASH.
- open Ubuntu Bash and clone the project repository
- on the command line execute `./install-ubuntu.sh`
- build and run according to the instructions in the project repository README

Trouble Shooting

- **.sh files not recognized on run:** Try `chmod a+x <filename.sh>` for example `chmod a+x install-ubuntu.sh`
- **Messages regarding 404s, missing resources, and a variety of other error messages** can be addressed by updates and other means, per this post and this post, and this post. The general steps are:

Not all steps will be necessary, for example, installing git and cloning the project repository, if this has already been done.

1. `sudo apt-get update`
2. `sudo apt-get install git`
3. `sudo apt-get install cmake`
4. `sudo apt-get install openssl`

5. `sudo apt-get install libssl-dev`
6. `git clone https://github.com/udacity/CarND-Kidnapped-Vehicle-Project or whatever CarND project`
7. `sudo rm /usr/lib/libuWS.so`
8. navigate to CarND-Kidnapped-Vehicle-Project/
9. `./install-ubuntu.sh`
10. at the top level of the project repository `mkdir build && cd build`
11. from /build `cmake .. && make`
12. Launch the simulator from Windows and execute the run command for the project, for example `./ExtendedKF` or `./particle_filter` (Make sure you also run the simulator on the Windows host machine) If you see this message, it is working `Listening to port 4567 Connected!!!`

After following these steps there may be some messages regarding makefile not found or can't create symbolic link to websockets. There is likely nothing wrong with the installation. Before doing any other troubleshooting make sure that steps 10 and 11 have been executed from the top level of the project directory, then test the installation using step 12.

Step 9 may fail for number of reasons as listed below:

- `install-ubuntu.sh` has only rw but no x permission. Run `chmod a+x install-ubuntu.sh` to give execution permission
- Cannot find the package `libuv1-dev`
  - To install the package run `sudo apt-get install libuv1.dev`
  - If you still cannot install the package run the following to get the package and install it:
    - `sudo add-apt-repository ppa:acooks/libwebsockets6`
    - `sudo apt-get update`
    - `sudo apt-get install libuv1.dev`
- May complain about the version of cmake you have. You need a version greater than 3.0. Here is a link which describes how to get version 3.8. Look at Teocci's

response in this link

- Installing cmake requires g++ compiler. Install a g++ version 4.9 or greater. Here are the steps:
  - `sudo add-apt-repository ppa:ubuntu-toolchain-r/test`
  - `sudo apt-get update`
  - `sudo apt-get install g++-4.9`

A Note Regarding Step 11 This step can fail if the bash shell is still referring to an older g++ version. To address this, please refer to **this Ask Ubuntu post**.

Docker on Windows

The best place to start is to follow the instructions **here**. A common pitfall is to not log in to the docker container. The first time you run `docker run -it -p 4567:4567 -v 'pwd':/work udacity/controls_kit:latest` the controls_kit may download, but the system may not log you in to the container. If you are logged in instead of a `$` prompt, you should see something like this: `root@27b126542a51:/work#`. If you are not logged into the container commands such as `apt-get` and `make` will not be recognized, so be sure to execute `docker run -it -p 4567:4567 -v 'pwd':/work udacity/controls_kit:latest` again, if you do not see the correct prompt.

A thoughtful student has created a docker specific **starter guide** for the EKF project. The following is an abridged version.

Tip regarding port forwarding when running code on vm and simulator on host

When using a virtual machine and running the simulator on the host machine, it is critical to set up port forwarding, as described **here**.

Transferring Files Between Native and Virtual Environments

Many prefer to use text editors in Windows rather than those that ship with Ubuntu BASH or Docker (vim, nano, etc.)

Options for addressing this include:

- **All Systems:** setup a git repo, edit files in Windows, push to the repo from Windows, pull the repo from the virtual environment
- **Ubuntu BASH:** edit files in windows, mount the c drive in Ubuntu BASH (`cd /mnt/c`), navigate to the files, copy to the desired location in Ubuntu BASH, navigated to the appropriate Ubuntu BASH folder
- **Docker on Windows:** See this starter guide for suggestions.

Note Regarding Ubuntu Bash on Windows

The Ubuntu Bash system can be accessed from Windows, any files altered in this way may no longer be recognizable by Ubuntu BASH. This often manifest itself in the file disappearing from Ubuntu BASH.

IDE Profile to Develop Natively in Windows with Visual Studio

A student contributed IDE profile can be found here. More detail can be found here

NEXT