ENVIRONMENTS

https://github.com/apache/incubator-nuttx

NuttX requires a POSIX development environment such as you would find under Linux or macOS. NuttX may also be installed and built on Windows system if you also provide such a POSIX development environment. Options for a POSIX development environment under Windows include:

An installation of Linux on a virtual machine (VM) in Windows. I have not been happy using a VM myself. I have had stability problems with open source VMs and commercial VMs cost more than I want to spend. Sharing files with Linux running in a VM is awkward; sharing devices connected to the Windows box with Linux in a VM is, at the very least, confusing; Using Windows tools (such as Segger J-Link) with files built under the Linux VM is not a possibility.

The Cygwin environment. Instructions for installation of Cygwin on a Windows system are provided in the following paragraph, "Installing Cygwin". Cygwin is a mature, well-tested, and very convenient environment. It is especially convenient if you need to integrate with Windows tools and files. Downsides are that the installation time is very long and the compile times are slow.

Ubuntu/Bash shell under Windows 10. This is a new option under Windows 10. See the section "Ubuntu Bash under Windows 10" below. This is an improvement over Cygwin if your concern is compile time; its build performance is comparable to native Linux, certainly better than the Cygwin build time. It also installs in a tiny fraction of the time as Cygwin, perhaps 20 minutes for the basic Ubuntu install (vs. more than a day for the complete Cygwin install).

There have been even more recent ports of Linux environment to Windows. I need to update this section to include some mention of these alternatives.

The MSYS environment. MSYS derives from an older version of Cygwin simplified and adapted to work more naturally in the Windows environment. See http://www.mingw.org/wiki/MSYS if you are interested in using MSYS. The advantages of the MSYS environment is that it is better integrted with the native Windows environment and lighter weight; it uses only a minimal number of add-on POSIX-land tools.

The download link in that Wiki takes you to the SourceForge download site. The SourceForge MSYS project has been stagnant for some time. The MSYS project has more recently moved to http://odsn.net/projects/sfnet_mingwbundle. Downloads of current .zip files are available there but no instructions for the installation.

MSYS2 appears to be a re-write of MSYS based on a newer version of Cygwin. Is it available at https://www.msys2.org. A windows installer is available at that site along with very good installation instructions. The download is relatively quick (at least compared to Cygwin) and the 'pacman' package management tool supports supports simple system updates. For example, 'pacman -S git' will install the GIT command line utilities.

Other POSIX environments. Check out:
UnxUtils: https://sourceforge.net/projects/unxutils/, https://en.wikipedia.org/wiki/UnxUtils
MobaXterm: https://mobaxterm.mobatek.net/
Gow: https://github.com/bmatzelle/gow/wiki

Disclaimer: In principle, these should work. However, I have never used any of these environments and cannot guarantee that there is not some less-than-obvious issues.

NuttX can also be installed and built on a native Windows system, but with some potential tool-related issues (see the discussion "Native Windows Build" under "Building NuttX" below). GNUWin32 is used to provide compatible native windows tools.
Installing Cygwin

Installing Cygwin on your Windows PC is simple, but time consuming. See http://www.cygwin.com/ for installation instructions. Basically you just need to download a tiny setup.exe program and it does the real, network installation for you.

Some Cygwin installation tips:

Install at C:\cygwin

Install everything: "Only the minimal base packages from the Cygwin distribution are installed by default. Clicking on categories and packages in the setup.exe package installation screen will provide you with the ability to control what is installed or updated. Clicking on the "Default" field next to the "All" category will provide you with the opportunity to install every Cygwin package. Be advised that this will download and install hundreds of megabytes to your computer."

If you use the "default" installation, you will be missing many of the Cygwin utilities that you will need to build NuttX. The build will fail in numerous places because of missing packages.

NOTE: The last time I installed everything, the download was about 5GiB. The server I selected was also very slow so it took over a day to do the whole install!

NOTE: You don't really have to install everything but I cannot answer the question "Then what should I install?" I don't know the answer to that and so will continue to recommend installing everything.

You should certainly be able to omit "Science", "Math", and "Publishing". You can try omitting KDE, Gnome, GTK, and other graphics packages if you don't plan to use them.

Perhaps a minimum set would be those packages listed below for the "Ubuntu Bash under Windows 10" installation?

UPDATE: Sergey Frolov had success with the following minimal Cygwin configuration:

After starting the Cygwin installer, keep the recommended packages that are pre-selected in the default configuration.

Using the installation tools, add the following packages:

make (GNU make)	bison	libgmp3-dev
gcc-core	byacc	libmpfr-dev
gcc-g++	gperf	libmpc-dev
flex	gdb	automake-1.15
libncurses-dev	libgmp-dev	curl

After installing Cygwin, you will get lots of links for installed tools and shells. I use the RXVT native shell. It is fast and reliable and does not require you to run the Cygwin X server (which is neither fast nor reliable). Unless otherwise noted, the rest of these instructions assume that you are at a bash command line prompt in either Linux or in Cygwin shell.

Using MSYS

MSYS is an environment the derives from Cygwin. Thus, most things said about Cygwin apply equally to MSYS. This section will, then, focus on the differences when using MSYS, specifically MSYS2.

Here is it assumed that you have already downloaded and installed MSYS2 from https://www.msys2.org using the windows installer available at that location. It is also assumed that you have brought in the necessary tools using the 'pacman' package management tool Tools needed including:

pacman -S git
pacman -S make
pacman -S gcc
pacman -S gdb

And possibly others depending upon your usage. Then you will need to build and install kconfig-frontends per the instructions of the top-level README.txt file in the tools repository. This requires the following additional tools:

pacman -S bison
pacman -S curl
pacman -S gperf
pacman -S ncurses-devel
pacman -S automake-wrapper
pacman -S autoconf
pacman -S pkg-config

Because of some versioning issues, I had to run 'aclocal' prior to running the kconfig-frontends configure script. See "Configuring NuttX" below for further information.

Unlike Cygwin, MSYS does not support symbolic links. The 'ln -s' command will, in fact, copy a directory! This means that you Make.defs file will have to include definitions like:

```
ifeq ($(CONFIG_WINDOWS_MSYS),y)
  DIRLINK = $(TOPDIR)/tools/copydir.sh
  DIRUNLINK = $(TOPDIR)/tools/unlink.sh
endif
```

This will force the directory copies to work in a way that can be handled by the NuttX build system. NOTE: The default link.sh script has been updated so that is should now be MSYS2 compatible. The above is preferred but no longer necessary in the Make.defs file.

To build the simulator under MSYS, you also need:

pacman -S zlib-devel

It appears that you cannot use directory names with spaces in them like "/c/Program\ Files (86)" in the MSYS path variable. I worked around this by create Windows junctions like this:

Open the a windows command terminal,

cd to c:\msys64, then

mklink /j programfiles "C:/Program\ Files" and

mklink /j programfiles86 "C:/Program\ Files\ \(x86\)"

They then show up as /programfiles and /programfiles86 with the MSYS2 sandbox. Those paths can then be used with the PATH variable. I had to do something similar for the path to the GNU Tools "ARM Embedded Toolchain" which also has spaces in the path name.

Ubuntu Bash under Windows 10

A better version of a command-line only Ubuntu under Windows 10 (beta) has recently been made available from Microsoft.

Installation

Installation instructions abound on the Internet complete with screen shots. I will attempt to duplicate those instructions in full here. Here are the simplified installation steps:

Open Settings.

Click on Update & security.

Click on For Developers.

Under Use developer features, select the Developer mode option to setup the environment to install Bash.

A message box should pop up. Click Yes to turn on developer mode.

After the necessary components install, you'll need to restart your computer.

Once your computer reboots:

Open Control Panel.

Click on Programs.

Click on Turn Windows features on or off.

A list of features will pop up, check the Windows Subsystem for Linux (beta) option.

Click OK.

Once the components installed on your computer, click the Restart now button to complete the task.

After your computer restarts, you will notice that Bash will not appear in the Recently added list of apps, this is because Bash isn't actually installed yet. Now that you have setup the necessary components, use the following steps to complete the installation of Bash:

Open Start, do a search for bash.exe, and press Enter.

On the command prompt, type y and press Enter to download and install Bash from the Windows Store. This will take awhile.

Then you'll need to create a default UNIX user account. This account doesn't have to be the same as your Windows account. Enter the username in the required field and press Enter (you can't use the username admin).

Close the bash.exe command prompt.

Now that you completed the installation and setup, you can open the Bash tool from the Start menu like you would with any other app.
Accessing Windows Files from Ubuntu

File systems will be mounted under /mnt so for example C:\Program Files appears at /mnt/c/Program Files. This is as opposed to Cygwin where the same directory would appear at /cygdrive/c/Program Files.

With these differences (perhaps a few other Windows quirks) the Ubuntu install works just like Ubuntu running natively on your PC.

A good tip for file sharing is to use symbolic links within your Ubuntu home directory. For example, suppose you have your projects directory at C:\Documents\projects. Then you can set up a link to the projects/ directory in your Ubuntu directory like:

ln -s /mnt/c/Documents/projects projects

Accessing Ubuntu Files From Windows

In Ubuntu Userspace for Windows, the Ubuntu file system root directory is at:

%localappdata%\lxss\rootfs

Or

C:\Users\Username\AppData\Local\lxss\rootfs

However, I am unable to see my files under the rootfs\home directory. After some looking around, I find the home directory %localappdata%\lxss\home.

With that trick access to the /home directory, you should actually be able to use Windows tools outside of the Ubuntu sandbox with versions of NuttX built within the sandbox using that path.
Executing Windows Tools from Ubuntu

You can also execute Windows tools from within the Ubuntu sandbox:

```
/mnt/c/Program\ Files\ \(x86\)/Microchip/xc32/v1.43/bin/xc32-gcc.exe --version
Unable to translate current working directory. Using C:\WINDOWS\System32
xc32-gcc.exe (Microchip Technology) 4.8.3 MPLAB XC32 Compiler v1.43 Build date: Mar  1 2017
...
```

The error message indicates that there are more issues: You cannot mix Windows tools that use Windows style paths in an environment that uses POSIX paths. I think you would have to use Linux tools only from within the Ubuntu sandbox.
Install Ubuntu Software

Use sudo apt-get install <package name>. As examples, this is how you would get GIT:

sudo apt-get install git

This will get you a compiler for your host PC:

sudo apt-get install gcc

This will get you an ARM compiler for your target:

sudo apt-get install gcc-arm-none-eabi

NOTE: That is just an example. I am not sure if apt-get will give you a current or usable compiler. You should carefully select your toolchain for the needs of your project.

You will also need to get the kconfig-frontends configuration as described below under NuttX Configuration Tool. In order to build the kconfig-frontends configuration tool you will also need: make, gperf, flex, bison, and libncurses-dev.

That is enough to do a basic NuttX build.

Integrating with Windows Tools

If you want to integrate with Windows native tools, then you would need deal with the same kind of craziness as with integrating Cygwin with native toolchains, see the section Cygwin Build Problems below.

However, there is currently no build support for using Windows native tools with Ubuntu under Windows. This tool combination is made to work with Cygwin through the use of the cygpath -w tool that converts paths from say /cydrive/c/Program Files to C:\Program Files. There is, however, no corresponding tool to convert /mnt/c/Program Files in the Ubuntu environment.

Graphics Support

The Ubuntu version support by Microsoft is a command-line only version. There is no support for Linux graphics utilities.

This limitation is not a limitation of Ubuntu, however, only in what Microsoft is willing to support. If you install a X-Server, then you can also use basic graphics utilities. See for example:

http://www.howtogeek.com/261575/how-to-run-graphical-linux-desktop-applications-from-windows-10s-bash-shell/

Many Linux graphics programs would, however, also require a graphics framework like GTK or Qt. So this might be a trip down the rabbit hole.