**Targeting the Windows Subsystem for Linux from Visual Studio**

★★★★★

★★★★

★★★

★★

★

February 8, 2017 by [Marc Goodner - MSFT](https://social.msdn.microsoft.com/profile/Marc+Goodner+-+MSFT) // [61 Comments](https://blogs.msdn.microsoft.com/vcblog/2017/02/08/targeting-windows-subsystem-for-linux-from-visual-studio/#comments)

* [0](https://blogs.msdn.microsoft.com/vcblog/2017/02/08/targeting-windows-subsystem-for-linux-from-visual-studio/?ranMID=24542&ranEAID=XdSn0e3h3*k&ranSiteID=XdSn0e3h3.k-jfq1NoT__87ZYfDESnGlJA&tduid=(36bf82c8398c9a99e60f31caa7a09209)(256380)(2459594)(XdSn0e3h3.k-jfq1NoT__87ZYfDESnGlJA)())
* [0](https://blogs.msdn.microsoft.com/vcblog/2017/02/08/targeting-windows-subsystem-for-linux-from-visual-studio/?ranMID=24542&ranEAID=XdSn0e3h3*k&ranSiteID=XdSn0e3h3.k-jfq1NoT__87ZYfDESnGlJA&tduid=(36bf82c8398c9a99e60f31caa7a09209)(256380)(2459594)(XdSn0e3h3.k-jfq1NoT__87ZYfDESnGlJA)())

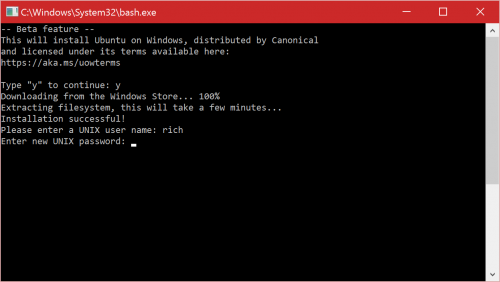
The [Windows Subsystem for Linux (WSL)](https://msdn.microsoft.com/en-us/commandline/wsl/about) was first introduced at Build in 2016 and was delivered as an early beta in Windows 10 Anniversary Update. Since then, the WSL team has been hard at work, dramatically improving WSL’s abilty to run an ever increasing number of native Linux command-line binaries and tools, including most mainstream developer tools, platforms and languages, and many daemons/services\* including MySQL, Apache, and SSH.

With the Linux development with C++ workload in Visual Studio 2017 you can use the full power of Visual Studio for your C/C++ Linux development. Because WSL is just another Linux system, you can target it from Visual Studio by following [our guide on using the Linux workload](http://aka.ms/vslinux).  This gives you a lot of flexibility to keep your entire development cycle locally on your development machine without needing the complexity of a separate VM or machine. It is, however, worth covering how to configure SSH on Bash/WSL in a bit more detail.

**Install WSL**

If you’ve not already done so, you’ll first need to [enable developer mode and install WSL itself](https://msdn.microsoft.com/en-us/commandline/wsl/install_guide). This only takes a few seconds, but does require a reboot. You need to be on a current Insider build or wait for the release of Creators Update to use WSL with the Linux workload in VS.

When you run Bash for the first time, you’ll need to follow the on-screen instructions to accept Canonical’s license, download the Ubuntu image, and install it on your machine. You’ll then need to choose a UNIX username and password. This needn’t be the same as your Windows login username and password if you prefer. You’ll only need to enter the UNIX username and password in the future when you use sudo to elevate a command, or to login “remotely” (see below).

[](https://msdnshared.blob.core.windows.net/media/2017/02/bash-install.png)

Now you’ll have a vanilla Ubuntu instance on your machine within which you can run any ELF-64 Linux binary, including those that you download using apt-get!

Before we continue, let’s install the build-essential package so you have some key developer tools including the GNU C++ compiler, linker, etc.:

$ sudo apt install -y build-essential

We’ll also need to install gdbserver, a program that allows you to debug with a remote GDB debugger.

$ sudo apt install -y gdbserver

**Install & configure SSH**

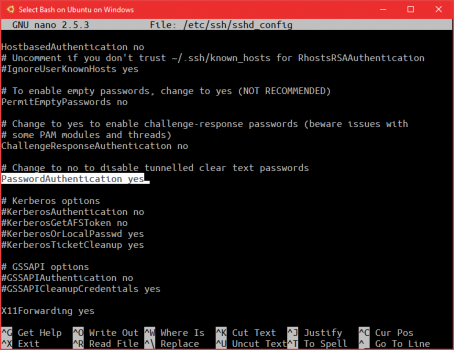
Let’s use the ‘apt’ package manager to download and install SSH on Bash/WSL:

$ sudo apt install -y openssh-server

Before we start SSH, you will need to configure SSH, but you only need to do this once. Run the following commands to edit the sshd config file:

$ sudo nano /etc/ssh/sshd\_config

Scroll down the “PasswordAuthentication” setting and make sure it’s set to “yes”:

[](https://msdnshared.blob.core.windows.net/media/2017/02/bash-nanosshd.png)

Hit CTRL + X to exit, then Y to save.

Now generate SSH keys for the SSH instance:

$ sudo ssh-keygen -A

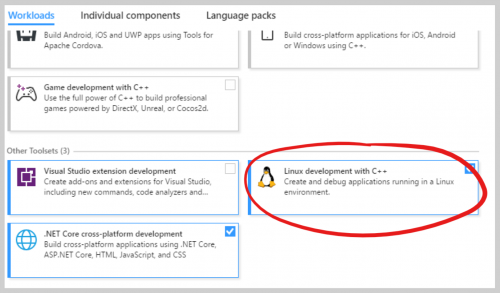
Start SSH before connecting from Visual Studio:

$ sudo service ssh start

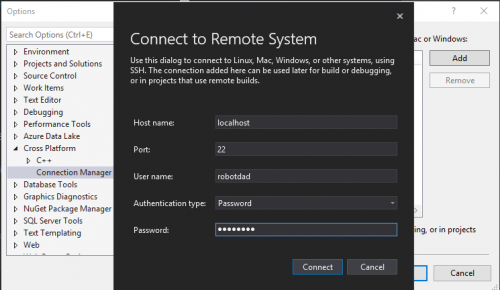
**\*Note: You will need to do this every time you start your first Bash console. As a precaution, WSL currently tears-down all Linux processes when you close your last Bash console!**.

**Install & configure Visual Studio**

For the best experience, we recommend installing [Visual Studio 2017 RC](https://www.visualstudio.com/vs/visual-studio-2017-rc/) (or later) to use Visual C++ for Linux. Be sure to select the Visual C++ for Linux workload during the installation process.

[](https://msdnshared.blob.core.windows.net/media/2017/02/bashinstalllinux.png)

Now you can connect to the Windows Subsystem for Linux from Visual Studio by going to Tools > Options > Cross Platform > Connection Manager. Click add and enter “localhost” for the hostname and your WSL user/password.

[](https://msdnshared.blob.core.windows.net/media/2017/02/bashconnectionmanager.png)

Now you can use this connection with any of your existing C++ Linux projects or create a new Linux project under File > New Project > Visual C++ > Cross Platform > Linux.

In the future, we’ll publish a more detailed post showing the advantages of working with WSL, particularly leveraging the compatibility of binaries built using the Linux workload to deploy on remote Linux systems.

For now, now that, starting with Windows 10 Creators Update, Bash on the Windows Subsystem for Linux (Bash/WSL) is a real Linux system from the perspective of Visual Studio.