

Windows Subsystem for Linux Documentation

Article • 06/27/2022

Windows Subsystem for Linux (WSL) lets developers run a GNU/Linux environment -- including most command-line tools, utilities, and applications -- directly on Windows, unmodified, without the overhead of a traditional virtual machine or dual-boot setup.

[Install WSL](#)

Learn more

- [What is the Windows Subsystem for Linux \(WSL\)?](#)
- [What's new with WSL 2?](#)
- [Comparing WSL 1 and WSL 2](#)
- [Frequently Asked Questions](#)

Get started

- [Install WSL](#)
- [Install Linux on Windows Server](#)
- [Manual install steps](#)
- [Best practices for setting up a WSL development environment](#)

Try WSL preview features by joining the Windows Insiders Program

To try the most recent features or updates to WSL, join the [Windows Insiders Program](#). Once you have joined Windows Insiders, you can choose the channel you would like to receive preview builds from inside the Windows settings menu. You can choose from:

- Dev channel: Most recent updates, but low stability.
- Beta channel: Ideal for early adopters, more reliable builds than the Dev channel.
- Release Preview channel: Preview fixes and key features on the next version of Windows just before its available to the general public.

Team blogs

- Overview post with a collection of videos and blogs ↗
- Command-Line blog ↗ (Active)
- Windows Subsystem for Linux Blog (Historical)

Provide feedback

- GitHub issue tracker: WSL ↗
- GitHub issue tracker: WSL documentation ↗

Related videos

WSL BASICS

1. [What is the Windows Subsystem for Linux \(WSL\)?](#) ↗ | One Dev Question (0:40)
2. [I'm a Windows developer. Why should I use WSL?](#) ↗ | One Dev Question (0:58)
3. [I'm a Linux developer. Why should I use WSL?](#) ↗ | One Dev Question (1:04)
4. [What is Linux?](#) ↗ | One Dev Question (1:31)
5. [What is a Linux distro?](#) ↗ | One Dev Question (1:04)
6. [How is WSL different than a virtual machine or dual booting?](#) ↗ | One Dev Question
7. [Why was the Windows Subsystem for Linux created?](#) ↗ | One Dev Question (1:14)
8. [How do I access files on my computer in WSL?](#) ↗ | One Dev Question (1:41)
9. [How is WSL integrated with Windows?](#) ↗ | One Dev Question (1:34)
10. [How do I configure a WSL distro to launch in the home directory in Terminal?](#) ↗ | One Dev Question (0:47)
11. [Can I use WSL for scripting?](#) ↗ | One Dev Question (1:04)
12. [Why would I want to use Linux tools on Windows?](#) ↗ | One Dev Question (1:20)
13. [In WSL, can I use distros other than the ones in the Microsoft Store?](#) ↗ | One Dev Question (1:03)

WSL DEMOS

1. [WSL2: Code faster on the Windows Subsystem for Linux!](#) ↗ | Tabs vs Spaces (13:42)
2. [WSL: Run Linux GUI Apps](#) ↗ | Tabs vs Spaces (17:16)
3. [WSL 2: Connect USB devices](#) ↗ | Tabs vs Spaces (10:08)
4. [GPU Accelerated Machine Learning with WSL 2](#) ↗ | Tabs vs Spaces (16:28)
5. [Visual Studio Code: Remote Dev with SSH, VMs, and WSL](#) ↗ | Tabs vs Spaces (29:33)
6. [Windows Dev Tool Updates: WSL, Terminal, Package Manager, and more](#) ↗ | Tabs vs Spaces (20:46)

7. Build Node.JS apps with WSL ↗ | Highlight (3:15)
8. New memory reclaim feature in WSL 2 ↗ | Demo (6:01)
9. Web development on Windows (in 2019) ↗ | Demo (10:39)

WSL DEEP DIVES

1. [WSL on Windows 11 - Demos with Craig Loewen and Scott Hanselman ↗](#) | Windows Wednesday (35:48)
2. [WSL and Linux Distributions – Hayden Barnes and Kayla Cinnamon ↗](#) | Windows Wednesday (37:00)
3. [Customize your terminal with Oh My Posh and WSL Linux distros ↗](#) | Windows Wednesday (33:14)
4. [Web dev Sarah Tamsin and Craig Loewen chat about web development, content creation, and WSL ↗](#) | Dev Perspectives (12:22)
5. [How WSL accesses Linux files from Windows ↗](#) | Deep dive (24:59)
6. [Windows subsystem for Linux architecture: a deep dive ↗](#) | Build 2019 (58:10)

What is the Windows Subsystem for Linux?

Article • 11/28/2023

Windows Subsystem for Linux (WSL) is a feature of Windows that allows you to run a Linux environment on your Windows machine, without the need for a separate virtual machine or dual booting. WSL is designed to provide a seamless and productive experience for developers who want to use both Windows and Linux at the same time.

- Use WSL to install and run various Linux distributions, such as Ubuntu, Debian, Kali, and more. [Install Linux distributions](#) and receive automatic updates from the [Microsoft Store](#), import Linux distributions not available in the Microsoft Store, or [build your own customer Linux distribution](#).
- Store files in an isolated Linux file system, specific to the installed distribution.
- Run command-line tools, such as BASH.
- Run common BASH command-line tools such as `grep`, `sed`, `awk`, or other ELF-64 binaries.
- Run Bash scripts and GNU/Linux command-line applications including:
 - Tools: vim, emacs, tmux
 - Languages: [NodeJS](#), JavaScript, [Python](#), Ruby, C/C++, C# & F#, Rust, Go, etc.
 - Services: SSHD, [MySQL](#), Apache, lighttpd, [MongoDB](#), [PostgreSQL](#).
- Install additional software using your own GNU/Linux distribution package manager.
- Invoke Windows applications using a Unix-like command-line shell.
- Invoke GNU/Linux applications on Windows.
- [Run GNU/Linux graphical applications](#) integrated directly to your Windows desktop
- Use your device [GPU to accelerate Machine Learning workloads running on Linux](#).

[Install WSL](#)

<https://www.youtube-nocookie.com/embed/48k317kOxqg> ↗

What is WSL 2?

WSL 2 is the default distro type when installing a Linux distribution. WSL 2 uses virtualization technology to run a Linux kernel inside of a lightweight utility virtual machine (VM). Linux distributions run as isolated containers inside of the WSL 2 managed VM. Linux distributions running via WSL 2 will share the same network

namespace, device tree (other than `/dev/pts`), CPU/Kernel/Memory/Swap, `/init` binary, but have their own PID namespace, Mount namespace, User namespace, Cgroup namespace, and `init` process.

WSL 2 increases file system performance and adds full system call compatibility in comparison to the WSL 1 architecture. Learn more about how [WSL 1 and WSL 2 compare](#).

Individual Linux distributions can be run with either the WSL 1 or WSL 2 architecture. Each distribution can be upgraded or downgraded at any time and you can run WSL 1 and WSL 2 distributions side by side. See the [Set WSL version command](#).

<https://www.youtube-nocookie.com/embed/MrZolfGm8Zk>

Microsoft Loves Linux

Learn more about [Linux resources at Microsoft](#), including Microsoft tools that run on Linux, Linux training courses, Cloud Solution Architecture for Linux, and Microsoft + Linux news, events, and partnerships. **Microsoft Loves Linux!**

Comparing WSL Versions

Article • 12/15/2023

Learn more about different WSL versions, including why WSL 2 is now the default and the specific scenarios or exceptions that may warrant switching your installed Linux distribution to the earlier WSL 1 architecture.

Comparing WSL 1 and WSL 2

This guide will compare WSL 1 and WSL 2, including [exceptions for using WSL 1 rather than WSL 2](#). The primary differences between WSL 1 and WSL 2 are the use of an actual Linux kernel inside a managed VM, support for full system call compatibility, and performance across the Linux and Windows operating systems. WSL 2 is the current default version when installing a Linux distribution and uses the latest and greatest in virtualization technology to run a Linux kernel inside of a lightweight utility virtual machine (VM). WSL2 runs Linux distributions as isolated containers inside the managed VM. If your distribution is currently running WSL 1 and you want to update to WSL 2, see [update from WSL 1 to WSL 2](#).

Comparing features

[] [Expand table](#)

Feature	WSL 1	WSL 2
Integration between Windows and Linux	✓	✓
Fast boot times	✓	✓
Small resource foot print compared to traditional Virtual Machines	✓	✓
Runs with current versions of VMware and VirtualBox	✓	✗
Managed VM	✗	✓
Full Linux Kernel	✗	✓
Full system call compatibility	✗	✓
Performance across OS file systems	✓	✗
systemd support	✗	✓
IPv6 support	✓	✓

As you can tell from the comparison table above, the WSL 2 architecture outperforms WSL 1 in several ways, with the exception of performance across OS file systems, which can be addressed by storing your project files on the same operating system as the tools you are running to work on the project.

WSL 2 is only available in Windows 11 or Windows 10, Version 1903, Build 18362 or later. Check your Windows version by selecting the **Windows logo key + R**, type `winver`, select **OK**. (Or enter the `ver` command in Windows Command Prompt). You may need to [update to the latest Windows version](#). For builds lower than 14393, WSL is not supported at all.

For more info on the latest WSL 2 updates, see the [Windows Command Line blog](#), including [Systemd support is now available in WSL](#) and [WSL September 2023 update](#) for more info on IPv6 support.

Note

WSL 2 will work with [VMware 15.5.5+](#) and although [VirtualBox 6+](#) states that there is WSL support, there are still significant challenges that make it unsupported. Learn more in our [FAQs](#).

What's new in WSL 2

WSL 2 is a major overhaul of the underlying architecture and uses virtualization technology and a Linux kernel to enable new features. The primary goals of this update are to increase file system performance and add full system call compatibility.

- [WSL 2 system requirements](#)
- [Set your Linux distribution version from WSL 1 to WSL 2](#)
- [Frequently Asked Questions about WSL 2](#)

WSL 2 architecture

A traditional VM experience can be slow to boot up, is isolated, consumes a lot of resources, and requires your time to manage it. WSL 2 does not have these attributes.

WSL 2 provides the benefits of WSL 1, including seamless integration between Windows and Linux, fast boot times, a small resource footprint, and requires no VM configuration or management. While WSL 2 does use a VM, it is managed and run behind the scenes, leaving you with the same user experience as WSL 1.

Full Linux kernel

The Linux kernel in WSL 2 is built by Microsoft from the latest stable branch, based on the source available at kernel.org. This kernel has been specially tuned for WSL 2, optimizing for size and performance to provide an amazing Linux experience on Windows. The kernel will be serviced by Windows updates, which means you will get the latest security fixes and kernel improvements without needing to manage it yourself.

The [WSL 2 Linux kernel is open source](#). If you'd like to learn more, check out the blog post [Shipping a Linux Kernel with Windows](#) written by the team that built it.

Learn more in the [Release Notes for Windows Subsystem for Linux kernel](#).

Increased file IO performance

File intensive operations like git clone, npm install, apt update, apt upgrade, and more are all noticeably faster with WSL 2.

The actual speed increase will depend on which app you're running and how it is interacting with the file system. Initial versions of WSL 2 run up to 20x faster compared to WSL 1 when unpacking a zipped tarball, and around 2-5x faster when using git clone, npm install and cmake on various projects.

Full system call compatibility

Linux binaries use system calls to perform functions such as accessing files, requesting memory, creating processes, and more. Whereas WSL 1 used a translation layer that was built by the WSL team, WSL 2 includes its own Linux kernel with full system call compatibility. Benefits include:

- A whole new set of apps that you can run inside of WSL, such as [Docker](#) and more.
- Any updates to the Linux kernel are immediately ready for use. (You don't have to wait for the WSL team to implement updates and add the changes).

Exceptions for using WSL 1 rather than WSL 2

We recommend that you use WSL 2 as it offers faster performance and 100% system call compatibility. However, there are a few specific scenarios where you might prefer using WSL 1. Consider using WSL 1 if:

- Your project files must be stored in the Windows file system. WSL 1 offers faster access to files mounted from Windows.
 - If you will be using your WSL Linux distribution to access project files on the Windows file system, and these files cannot be stored on the Linux file system, you will achieve faster performance across the OS files systems by using WSL 1.
- A project which requires cross-compilation using both Windows and Linux tools on the same files.
 - File performance across the Windows and Linux operating systems is faster in WSL 1 than WSL 2, so if you are using Windows applications to access Linux files, you will currently achieve faster performance with WSL 1.
- Your project needs access to a serial port or USB device. *However*, USB device support is now available for WSL 2 via the [USBIPD-WIN](#) project. See [Connect USB devices](#) for set up steps.
- WSL 2 does not include support for accessing serial ports. Learn more in the [FAQs](#) or in [WSL GitHub repo issue on serial support ↗](#).
- You have strict memory requirements
 - WSL 2's memory usage grows and shrinks as you use it. When a process frees memory this is automatically returned to Windows. However, as of right now WSL 2 does not yet release cached pages in memory back to Windows until the WSL instance is shut down. If you have long running WSL sessions, or access a very large amount of files, this cache can take up memory on Windows. We are tracking the work to improve this experience on [the WSL GitHub repository issue 4166 ↗](#).
- For those using VirtualBox, be sure to use the latest version of both VirtualBox and WSL 2. See the [related FAQ](#).
- If you rely on a Linux distribution to have an IP address in the same network as your host machine, you may need to set up a workaround in order to run WSL 2. WSL 2 is running as a hyper-v virtual machine. This is a change from the bridged network adapter used in WSL 1, meaning that WSL 2 uses a Network Address Translation (NAT) service for its virtual network, instead of making it bridged to the host Network Interface Card (NIC) resulting in a unique IP address that will change on restart. To learn more about the issue and workaround that forwards TCP ports of WSL 2 services to the host OS, see [WSL GitHub repository issue 4150, NIC Bridge mode \(TCP Workaround\) ↗](#).

Note

Consider trying the VS Code [Remote WSL Extension ↗](#) to enable you to store your project files on the Linux file system, using Linux command line tools, but also using VS Code on Windows to author, edit, debug, or run your project in an

internet browser without any of the performance slow-downs associated with working across the Linux and Windows file systems. [Learn more](#).

WSL in the Microsoft Store

WSL has lifted the update functionality out of the Windows OS Image into a package that is available via the Microsoft Store. This means faster updates and servicing as soon as they're available, rather than needing to wait for an update of your Windows operating system.

WSL was originally included in the Windows operating system as an optional component that need to be enabled in order to install a Linux distribution. WSL in the Store has the same user experience, and is the same product, but receives updates and servicing as a store package, rather than as an entire OS update. Beginning in Windows version 19044 or higher, running the `wsl.exe --install` command will install the WSL servicing update from the Microsoft Store. ([See the blog post announcing this update ↗](#)). If you are already using WSL, you can update to ensure that you're receiving the latest WSL features and servicing from the store by running `wsl.exe --update`.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Basic commands for WSL

Article • 11/28/2023

The WSL commands below are listed in a format supported by PowerShell or Windows Command Prompt. To run these commands from a Bash / Linux distribution command line, you must replace `wsl` with `wsl.exe`. For a full list of commands, run `wsl --help`. If you have not yet done so, we recommend [updating to the version of WSL installed from Microsoft Store](#) in order to receive WSL updates as soon as they are available. ([Learn more about installing WSL via Microsoft Store](#)).

Install

PowerShell

```
wsl --install
```

Install WSL and the default Ubuntu distribution of Linux. [Learn more](#). You can also use this command to install additional Linux distributions by running `wsl --install <Distribution Name>`. For a valid list of distribution names, run `wsl --list --online`.

Options include:

- `--distribution`: Specify the Linux distribution to install. You can find available distributions by running `wsl --list --online`.
- `--no-launch`: Install the Linux distribution but do not launch it automatically.
- `--web-download`: Install from an online source rather than using the Microsoft Store.

When WSL is not installed options include:

- `--inbox`: Installs WSL using the Windows component instead of using the Microsoft Store. (*WSL updates will be received via Windows updates, rather than pushed out as-available via the store*).
- `--enable-wsl1`: Enables WSL 1 during the install of the Microsoft Store version of WSL by also enabling the "Windows Subsystem for Linux" optional component.
- `--no-distribution`: Do not install a distribution when installing WSL.

 Note

If you run WSL on Windows 10 or an older version, you may need to include the `-d` flag with the `--install` command to specify a distribution: `wsl --install -d <distribution name>`.

List available Linux distributions

PowerShell

```
wsl --list --online
```

See a list of the Linux distributions available through the online store. This command can also be entered as: `wsl -l -o`.

List installed Linux distributions

PowerShell

```
wsl --list --verbose
```

See a list of the Linux distributions installed on your Windows machine, including the state (whether the distribution is running or stopped) and the version of WSL running the distribution (WSL 1 or WSL 2). [Comparing WSL 1 and WSL 2](#). This command can also be entered as: `wsl -l -v`. Additional options that can be used with the list command include: `--all` to list all distributions, `--running` to list only distributions that are currently running, or `--quiet` to only show distribution names.

Set WSL version to 1 or 2

PowerShell

```
wsl --set-version <distribution name> <versionNumber>
```

To designate the version of WSL (1 or 2) that a Linux distribution is running on, replace `<distribution name>` with the name of the distribution and replace `<versionNumber>` with 1 or 2. [Comparing WSL 1 and WSL 2](#). WSL 2 is only available in Windows 11 or Windows 10, Version 1903, Build 18362 or later.

 **Warning**

Switching between WSL 1 and WSL 2 can be time-consuming and result in failures due to the differences between the two architectures. For distributions with large projects, we recommend backing up files before attempting a conversion.

Set default WSL version

PowerShell

```
wsl --set-default-version <Version>
```

To set a default version of WSL 1 or WSL 2, replace `<Version>` with either the number 1 or 2. For example, `wsl --set-default-version 2`. The number represents the version of WSL to default to for new Linux distribution installations. [Comparing WSL 1 and WSL 2](#). WSL 2 is only available in Windows 11 or Windows 10, Version 1903, Build 18362 or later.

Set default Linux distribution

PowerShell

```
wsl --set-default <Distribution Name>
```

To set the default Linux distribution that WSL commands will use to run, replace `<Distribution Name>` with the name of your preferred Linux distribution.

Change directory to home

PowerShell

```
wsl ~
```

The `~` can be used with `wsl` to start in the user's home directory. To jump from any directory back to home from within a WSL command prompt, you can use the command: `cd ~`.

Run a specific Linux distribution from PowerShell or CMD

```
PowerShell
```

```
wsl --distribution <Distribution Name> --user <User Name>
```

To run a specific Linux distribution with a specific user, replace `<Distribution Name>` with the name of your preferred Linux distribution (ie. Debian) and `<User Name>` with the name of an existing user (ie. root). If the user doesn't exist in the WSL distribution, you will receive an error. To print the current user name, use the command `whoami`.

Update WSL

```
PowerShell
```

```
wsl --update
```

Update your WSL version to the latest version. Options include:

- `--web-download`: Download the latest update from the GitHub rather than the Microsoft Store.

Check WSL status

```
PowerShell
```

```
wsl --status
```

See general information about your WSL configuration, such as default distribution type, default distribution, and kernel version.

Check WSL version

```
PowerShell
```

```
wsl --version
```

Check the version information about WSL and its components.

Help command

```
PowerShell
```

```
wsl --help
```

See a list of options and commands available with WSL.

Run as a specific user

```
PowerShell
```

```
wsl --user <Username>
```

To run WSL as a specified user, replace `<Username>` with the name of a user that exists in the WSL distribution.

Change the default user for a distribution

```
PowerShell
```

```
<DistributionName> config --default-user <Username>
```

Change the default user for your distribution log-in. The user has to already exist inside the distribution in order to become the default user.

For example: `ubuntu config --default-user johndoe` would change the default user for the Ubuntu distribution to the "johndoe" user.

ⓘ Note

If you are having trouble figuring out the name of your distribution, use the command `wsl -l`.

⚠ Warning

This command will not work for imported distributions, because these distributions do not have an executable launcher. You can instead change the default user for imported distributions using the `/etc/wsl.conf` file. See the Automount options in the [Advanced Settings Configuration](#) doc.

Shutdown

PowerShell

```
wsl --shutdown
```

Immediately terminates all running distributions and the WSL 2 lightweight utility virtual machine. This command may be necessary in instances that require you to restart the WSL 2 virtual machine environment, such as [changing memory usage limits](#) or making a change to your [.wslconfig file](#).

Terminate

PowerShell

```
wsl --terminate <Distribution Name>
```

To terminate the specified distribution, or stop it from running, replace `<Distribution Name>` with the name of the targeted distribution.

Identify IP address

- `wsl hostname -i` for the IP address of your Linux distribution installed via WSL 2 (the WSL 2 VM address)
- `cat /etc/resolv.conf` for the IP address of the Windows machine as seen from WSL 2 (the WSL 2 VM)

Export a distribution

PowerShell

```
wsl --export <Distribution Name> <FileName>
```

Exports a snapshot of the specified distribution as a new distribution file. Defaults to tar format. The filename can be `-` for standard input. Options include:

- `--vhdx`: Specifies the export distribution should be a .vhdx file instead of a tar file (this is only supported using WSL 2)

Import a distribution

PowerShell

```
wsl --import <Distribution Name> <InstallLocation> <FileName>
```

Imports the specified tar file as a new distribution. The filename can be `-` for standard input. Options include:

- `--vhd`: Specifies the import distribution should be a .vhdx file instead of a tar file (this is only supported using WSL 2)
- `--version <1/2>`: Specifies whether to import the distribution as a WSL 1 or WSL 2 distribution

Import a distribution in place

PowerShell

```
wsl --import-in-place <Distribution Name> <FileName>
```

Imports the specified .vhdx file as a new distribution. The virtual hard disk must be formatted in the ext4 filesystem type.

Unregister or uninstall a Linux distribution

While Linux distributions can be installed through the Microsoft Store, they can't be uninstalled through the store.

To unregister and uninstall a WSL distribution:

PowerShell

```
wsl --unregister <DistributionName>
```

Replacing `<DistributionName>` with the name of your targeted Linux distribution will unregister that distribution from WSL so it can be reinstalled or cleaned up. **Caution:** Once unregistered, all data, settings, and software associated with that distribution will be permanently lost. Reinstalling from the store will install a clean copy of the distribution. For example, `wsl --unregister Ubuntu` would remove Ubuntu from the distributions available in WSL. Running `wsl --list` will reveal that it is no longer listed.

You can also uninstall the Linux distribution app on your Windows machine just like any other store application. To reinstall, find the distribution in the Microsoft Store and select "Launch".

Mount a disk or device

PowerShell

```
wsl --mount <DiskPath>
```

Attach and mount a physical disk in all WSL2 distributions by replacing `<DiskPath>` with the directory\file path where the disk is located. See [Mount a Linux disk in WSL 2](#). Options include:

- `--vhd`: Specifies that `<Disk>` refers to a virtual hard disk.
- `--name`: Mount the disk using a custom name for the mountpoint
- `--bare`: Attach the disk to WSL2, but don't mount it.
- `--type <Filesystem>`: Filesystem type to use when mounting a disk, if not specified defaults to ext4. This command can also be entered as: `wsl --mount -t <Filesystem>`. You can detect the filesystem type using the command: `blkid <BlockDevice>`, for example: `blkid <dev/sdb1>`.
- `--partition <Partition Number>`: Index number of the partition to mount, if not specified defaults to the whole disk.
- `--options <MountOptions>`: There are some filesystem-specific options that can be included when mounting a disk. For example, [ext4 mount options](#) like: `wsl --mount -o "data=ordered"` or `wsl --mount -o "data=writeback"`. However, only filesystem-specific options are supported at this time. Generic options, such as `ro`, `rw`, or `noatime`, are not supported.

ⓘ Note

If you're running a 32-bit process in order to access `wsl.exe` (a 64-bit tool), you may need to run the command in the following manner: `C:\Windows\Sysnative\wsl.exe --command`.

Unmount disks

PowerShell

```
wsl --unmount <DiskPath>
```

Unmount a disk given at the disk path, if no disk path is given then this command will unmount and detach ALL mounted disks.

Deprecated WSL commands

PowerShell

```
wslconfig.exe [Argument] [Options]
```

PowerShell

```
bash [Options]
```

PowerShell

```
lxrun /[Argument]
```

These commands were the original wsl syntax for configuring Linux distributions installed with WSL, but have been replaced with the `wsl` or `wsl.exe` command syntax.



Collaborate with us on
GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for
Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

[Open a documentation issue](#)

[Provide product feedback](#)

How to install Linux on Windows with WSL

Article • 08/28/2023

Developers can access the power of both Windows and Linux at the same time on a Windows machine. The Windows Subsystem for Linux (WSL) lets developers install a Linux distribution (such as Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, etc) and use Linux applications, utilities, and Bash command-line tools directly on Windows, unmodified, without the overhead of a traditional virtual machine or dualboot setup.

Prerequisites

You must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11 to use the commands below. If you are on earlier versions please see [the manual install page](#).

Install WSL command

You can now install everything you need to run WSL with a single command. Open PowerShell or Windows Command Prompt in **administrator** mode by right-clicking and selecting "Run as administrator", enter the `wsl --install` command, then restart your machine.

```
PowerShell
```

```
wsl --install
```

This command will enable the features necessary to run WSL and install the Ubuntu distribution of Linux. ([This default distribution can be changed](#)).

If you're running an older build, or just prefer not to use the install command and would like step-by-step directions, see [WSL manual installation steps for older versions](#).

The first time you launch a newly installed Linux distribution, a console window will open and you'll be asked to wait for files to de-compress and be stored on your machine. All future launches should take less than a second.

 Note

The above command only works if WSL is not installed at all. If you run `wsl --install` and see the WSL help text, please try running `wsl --list --online` to see a list of available distros and run `wsl --install -d <DistroName>` to install a distro. To uninstall WSL, see [Uninstall legacy version of WSL](#) or [unregister](#) or [uninstall a Linux distribution](#).

Change the default Linux distribution installed

By default, the installed Linux distribution will be Ubuntu. This can be changed using the `-d` flag.

- To change the distribution installed, enter: `wsl --install -d <Distribution Name>`. Replace `<Distribution Name>` with the name of the distribution you would like to install.
- To see a list of available Linux distributions available for download through the online store, enter: `wsl --list --online` or `wsl -l -o`.
- To install additional Linux distributions after the initial install, you may also use the command: `wsl --install -d <Distribution Name>`.

Tip

If you want to install additional distributions from inside a Linux/Bash command line (rather than from PowerShell or Command Prompt), you must use .exe in the command: `wsl.exe --install -d <Distribution Name>` or to list available distributions: `wsl.exe -l -o`.

If you run into an issue during the install process, check the [installation section of the troubleshooting guide](#).

To install a Linux distribution that is not listed as available, you can [import any Linux distribution](#) using a TAR file. Or in some cases, [as with Arch Linux ↗](#), you can install using an `.appx` file. You can also create your own [custom Linux distribution](#) to use with WSL.

Set up your Linux user info

Once you have installed WSL, you will need to create a user account and password for your newly installed Linux distribution. See the [Best practices for setting up a WSL development environment](#) guide to learn more.

Set up and best practices

We recommend following our [Best practices for setting up a WSL development environment](#) guide for a step-by-step walk-through of how to set up a user name and password for your installed Linux distribution(s), using basic WSL commands, installing and customizing Windows Terminal, set up for Git version control, code editing and debugging using the VS Code remote server, good practices for file storage, setting up a database, mounting an external drive, setting up GPU acceleration, and more.

Check which version of WSL you are running

You can list your installed Linux distributions and check the version of WSL each is set to by entering the command: `wsl -l -v` in PowerShell or Windows Command Prompt.

To set the default version to WSL 1 or WSL 2 when a new Linux distribution is installed, use the command: `wsl --set-default-version <Version#>`, replacing `<Version#>` with either 1 or 2.

To set the default Linux distribution used with the `wsl` command, enter: `wsl -s <DistributionName>` or `wsl --set-default <DistributionName>`, replacing `<DistributionName>` with the name of the Linux distribution you would like to use. For example, from PowerShell/CMD, enter: `wsl -s Debian` to set the default distribution to Debian. Now running `wsl npm init` from Powershell will run the `npm init` command in Debian.

To run a specific wsl distribution from within PowerShell or Windows Command Prompt without changing your default distribution, use the command: `wsl -d <DistributionName>`, replacing `<DistributionName>` with the name of the distribution you want to use.

Learn more in the guide to [Basic commands for WSL](#).

Upgrade version from WSL 1 to WSL 2

New Linux installations, installed using the `wsl --install` command, will be set to WSL 2 by default.

The `wsl --set-version` command can be used to downgrade from WSL 2 to WSL 1 or to update previously installed Linux distributions from WSL 1 to WSL 2.

To see whether your Linux distribution is set to WSL 1 or WSL 2, use the command: `wsl -l -v`.

To change versions, use the command: `wsl --set-version <distro name> 2` replacing `<distro name>` with the name of the Linux distribution that you want to update. For example, `wsl --set-version Ubuntu-20.04 2` will set your Ubuntu 20.04 distribution to use WSL 2.

If you manually installed WSL prior to the `wsl --install` command being available, you may also need to [enable the virtual machine optional component](#) used by WSL 2 and [install the kernel package](#) if you haven't already done so.

To learn more, see the [Command reference for WSL](#) for a list of WSL commands, [Comparing WSL 1 and WSL 2](#) for guidance on which to use for your work scenario, or [Best practices for setting up a WSL development environment](#) for general guidance on setting up a good development workflow with WSL.

Ways to run multiple Linux distributions with WSL

WSL supports running as many different Linux distributions as you would like to install. This can include choosing distributions from the [Microsoft Store](#), [importing a custom distribution](#), or [building your own custom distribution](#).

There are several ways to run your Linux distributions once installed:

- [Install Windows Terminal \(Recommended\)](#) Using Windows Terminal supports as many command lines as you would like to install and enables you to open them in multiple tabs or window panes and quickly switch between multiple Linux distributions or other command lines (PowerShell, Command Prompt, Azure CLI, etc). You can fully customize your terminal with unique color schemes, font styles, sizes, background images, and custom keyboard shortcuts. [Learn more](#).
- You can directly open your Linux distribution by visiting the Windows Start menu and typing the name of your installed distributions. For example: "Ubuntu". This will open Ubuntu in its own console window.
- From Windows Command Prompt or PowerShell, you can enter the name of your installed distribution. For example: `ubuntu`
- From Windows Command Prompt or PowerShell, you can open your default Linux distribution inside your current command line, by entering: `wsl.exe`.
- From Windows Command Prompt or PowerShell, you can use your default Linux distribution inside your current command line, without entering a new one, by

entering: `wsl [command]`. Replacing `[command]` with a WSL command, such as: `wsl -l -v` to list installed distributions or `wsl pwd` to see where the current directory path is mounted in wsl. From PowerShell, the command `get-date` will provide the date from the Windows file system and `wsl date` will provide the date from the Linux file system.

The method you select should depend on what you're doing. If you've opened a WSL command line within a Windows Prompt or PowerShell window and want to exit, enter the command: `exit`.

Want to try the latest WSL preview features?

Try the most recent features or updates to WSL by joining the [Windows Insiders Program](#). Once you have joined Windows Insiders, you can choose the channel you would like to receive preview builds from inside the Windows settings menu to automatically receive any WSL updates or preview features associated with that build. You can choose from:

- Dev channel: Most recent updates, but low stability.
- Beta channel: Ideal for early adopters, more reliable builds than the Dev channel.
- Release Preview channel: Preview fixes and key features on the next version of Windows just before its available to the general public.

Additional resources

- [Windows Command Line Blog: Install WSL with a single command now available in Windows 10 version 2004 and higher](#)

Manual installation steps for older versions of WSL

Article • 11/20/2023

For simplicity, we generally recommend using the `wsl --install` to install Windows Subsystem for Linux, but if you're running an older build of Windows, that may not be supported. We have included the manual installation steps below. If you run into an issue during the install process, check the [installation section of the troubleshooting guide](#).

Step 1 - Enable the Windows Subsystem for Linux

You must first enable the "Windows Subsystem for Linux" optional feature before installing any Linux distributions on Windows.

Open PowerShell as Administrator (Start menu > PowerShell > right-click > Run as Administrator) and enter this command:

```
PowerShell  
  
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

We recommend now moving on to step #2, updating to WSL 2, but if you wish to only install WSL 1, you can now **restart** your machine and move on to [Step 6 - Install your Linux distribution of choice](#). To update to WSL 2, **wait to restart** your machine and move on to the next step.

Step 2 - Check requirements for running WSL 2

To update to WSL 2, you must be running Windows 10...

- For x64 systems: **Version 1903** or later, with **Build 18362.1049** or later.
- For ARM64 systems: **Version 2004** or later, with **Build 19041** or later.

or Windows 11.

 Note

Builds lower than 18362 do not support WSL 2. Use the [Windows Update Assistant](#) to update your version of Windows. The Windows version 1903 support is also only for x64 systems. If you are using an Arm64 version of Windows, you will need to upgrade to Windows 10 version 2004 or later for full access to WSL 2. For more info, see [WSL 2 support coming to Windows 10 Versions 1903 and 1909](#).

To check your version and build number, select **Windows logo key + R**, type `winver`, select **OK**. [Update to the latest Windows version](#) in the Settings menu.

Note

If you are running Windows 10 version 1903 or 1909, open "Settings" from your Windows menu, navigate to "Update & Security" and select "Check for Updates". Your Build number must be 18362.1049+ or 18363.1049+, with the minor build # over .1049. Read more: [WSL 2 Support is coming to Windows 10 Versions 1903 and 1909](#).

Step 3 - Enable Virtual Machine feature

Before installing WSL 2, you must enable the **Virtual Machine Platform** optional feature. Your machine will require [virtualization capabilities](#) to use this feature.

Open PowerShell as Administrator and run:

```
PowerShell
```

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all  
/norestart
```

Restart your machine to complete the WSL install and update to WSL 2.

Step 4 - Download the Linux kernel update package

The Linux kernel update package installs the most recent version of the [WSL 2 Linux kernel](#) for running WSL inside the Windows operating system image. (To run [WSL from the Microsoft Store](#), with more frequently pushed updates, use `wsl.exe --install` or `wsl.exe --update`).

1. Download the latest package:

- [WSL2 Linux kernel update package for x64 machines ↗](#)

ⓘ Note

If you're using an ARM64 machine, please download the [ARM64 package ↗](#) instead. If you're not sure what kind of machine you have, open Command Prompt or PowerShell and enter: `systeminfo | find "System Type"`. **Caveat:** On non-English Windows versions, you might have to modify the search text, translating the "System Type" string. You may also need to escape the quotations for the find command. For example, in German `systeminfo | find '"Systemtyp"'`.

2. Run the update package downloaded in the previous step. (Double-click to run - you will be prompted for elevated permissions, select 'yes' to approve this installation.)

Once the installation is complete, move on to the next step - setting WSL 2 as your default version when installing new Linux distributions. (Skip this step if you want your new Linux installs to be set to WSL 1).

ⓘ Note

For more information, read the article [changes to updating the WSL2 Linux kernel ↗](#), available on the [Windows Command Line Blog ↗](#).

Step 5 - Set WSL 2 as your default version

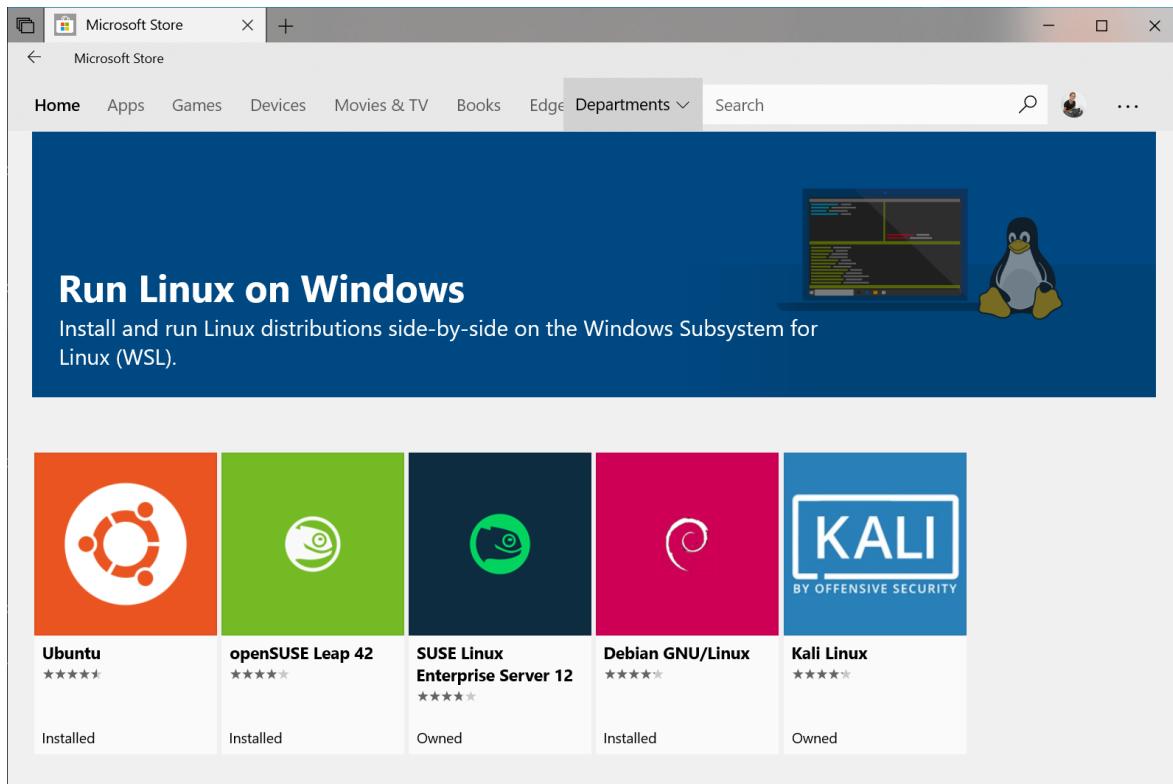
Open PowerShell and run this command to set WSL 2 as the default version when installing a new Linux distribution:

```
PowerShell
```

```
wsl --set-default-version 2
```

Step 6 - Install your Linux distribution of choice

1. Open the [Microsoft Store ↗](#) and select your favorite Linux distribution.



The following links will open the Microsoft store page for each distribution:

- [Ubuntu 18.04 LTS ↗](#)
- [Ubuntu 20.04 LTS ↗](#)
- [Ubuntu 22.04 LTS ↗](#)
- [openSUSE Leap 15.1 ↗](#)
- [SUSE Linux Enterprise Server 12 SP5 ↗](#)
- [SUSE Linux Enterprise Server 15 SP1 ↗](#)
- [Kali Linux ↗](#)
- [Debian GNU/Linux ↗](#)
- [Fedora Remix for WSL ↗](#)
- [Pengwin ↗](#)
- [Pengwin Enterprise ↗](#)
- [Alpine WSL ↗](#)
- [Raft\(Free Trial\) ↗](#)
- [Alma Linux ↗](#)

2. From the distribution's page, select "Get".

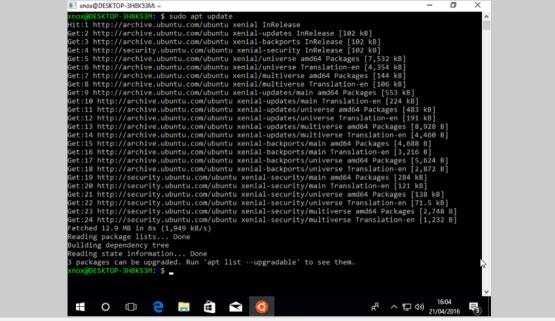


Ubuntu
Canonical Group Limited
★★★★★ 22
Free

Get **Share**

ESRB Everyone

Screenshots



Description

Ubuntu on Windows allows one to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

To use this feature, one first needs to use "Turn Windows features on or off" and select "Windows Subsystem for Linux", click OK, reboot, and use this app.

The above step can also be performed using Administrator PowerShell prompt:
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux

...

More

Available on

PC

What's new in this version

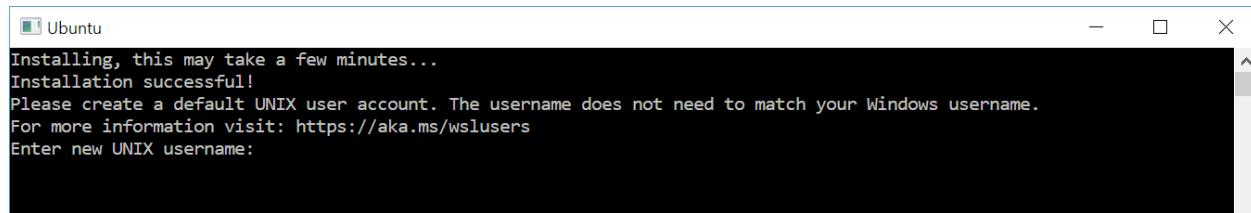
20170619.1 build of Ubuntu 16.04 LTS

Features

Ubuntu
bash
ssh

The first time you launch a newly installed Linux distribution, a console window will open and you'll be asked to wait for a minute or two for files to de-compress and be stored on your PC. All future launches should take less than a second.

You will then need to [create a user account and password for your new Linux distribution](#).



CONGRATULATIONS! You've successfully installed and set up a Linux distribution that is completely integrated with your Windows operating system!

Troubleshooting installation

If you run into an issue during the install process, check the [installation section of the troubleshooting guide](#).

Downloading distributions

There are some scenarios in which you may not be able (or want) to, install WSL Linux distributions using the Microsoft Store. You may be running a Windows Server or Long-Term Servicing (LTSC) desktop OS SKU that doesn't support Microsoft Store, or your corporate network policies and/or admins do not permit Microsoft Store usage in your environment. In these cases, while WSL itself is available, you may need to download Linux distributions directly.

If the Microsoft Store app is not available, you can download and manually install Linux distributions using these links:

- [Ubuntu ↗](#)
- [Ubuntu 22.04 LTS ↗](#)
- [Ubuntu 20.04 ↗](#)
- [Ubuntu 20.04 ARM ↗](#)
- [Ubuntu 18.04 ↗](#)
- [Ubuntu 18.04 ARM ↗](#)
- [Ubuntu 16.04 ↗](#)
- [Debian GNU/Linux ↗](#)
- [Kali Linux ↗](#)
- [SUSE Linux Enterprise Server 12 ↗](#)
- [SUSE Linux Enterprise Server 15 SP2 ↗](#)
- [SUSE Linux Enterprise Server 15 SP3 ↗](#)
- [openSUSE Tumbleweed ↗](#)
- [openSUSE Leap 15.3 ↗](#)
- [openSUSE Leap 15.2 ↗](#)
- [Oracle Linux 8.5 ↗](#)
- [Oracle Linux 7.9 ↗](#)
- [Fedora Remix for WSL ↗](#)

This will cause the `<distro>.appx` packages to download to a folder of your choosing.

If you prefer, you can also download your preferred distribution(s) via the command line, you can use PowerShell with the [Invoke-WebRequest](#) cmdlet. For example, to download Ubuntu 20.04:

PowerShell

```
Invoke-WebRequest -Uri https://aka.ms/wslubuntu2004 -OutFile Ubuntu.appx -  
UseBasicParsing
```

💡 Tip

If the download is taking a long time, turn off the progress bar by setting

```
$ProgressPreference = 'SilentlyContinue'
```

You also have the option to use the [curl command-line utility](#) for downloading. To download Ubuntu 20.04 with curl:

Console

```
curl.exe -L -o ubuntu-2004.appx https://aka.ms/wslubuntu2004
```

In this example, `curl.exe` is executed (not just `curl`) to ensure that, in PowerShell, the real curl executable is invoked, not the PowerShell curl alias for [Invoke-WebRequest](#).

Once the distribution has been downloaded, navigate to the folder containing the download and run the following command in that directory, where `app-name` is the name of the Linux distribution .appx file.

Powershell

```
Add-AppxPackage .\app_name.appx
```

Once the Appx package has finished downloading, you can start running the new distribution by double-clicking the appx file. (The command `wsl -l` will not show that the distribution is installed until this step is complete).

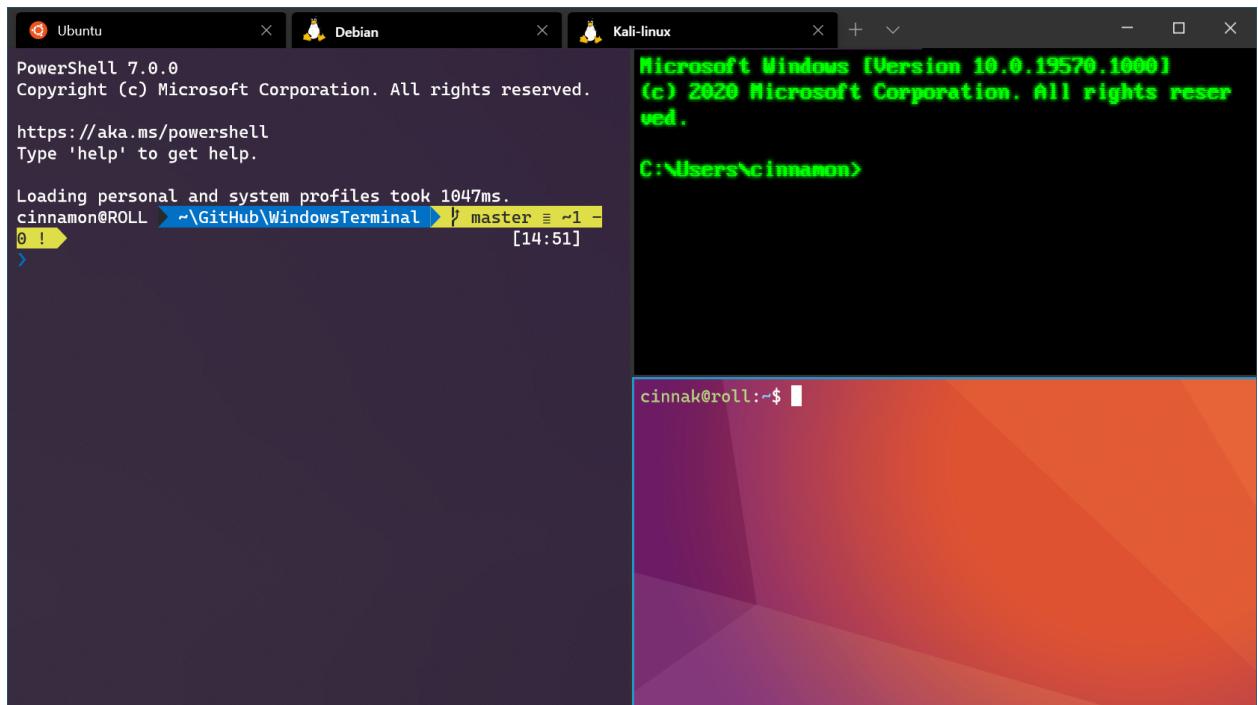
If you are using Windows server, or run into problems running the command above you can find the alternate install instructions on the [Windows Server](#) documentation page to install the `.appx` file by changing it to a zip file.

Once your distribution is installed, follow the instructions to [create a user account and password for your new Linux distribution](#).

Install Windows Terminal (optional)

Using Windows Terminal enables you to open multiple tabs or window panes to display and quickly switch between multiple Linux distributions or other command lines (PowerShell, Command Prompt, Azure CLI, etc). You can fully customize your terminal with unique color schemes, font styles, sizes, background images, and custom keyboard shortcuts. [Learn more](#).

[Install Windows Terminal](#).



Windows Server Installation Guide

Article • 06/20/2022

The Windows Subsystem for Linux (WSL) is available for installation on Windows Server 2019 (version 1709) and later. This guide will walk through the steps of enabling WSL on your machine.

Install WSL on Windows Server 2022

[Windows Server 2022](#) now supports a simple WSL installation using the command:

```
Bash
```

```
wsl --install
```

You can now install everything you need to run WSL on Windows Server 2022 by entering this command in an **administrator** PowerShell or Windows Command Prompt and then restarting your machine.

This command will enable the required optional components, download the latest Linux kernel, set WSL 2 as your default, and install a Linux distribution for you (*Ubuntu by default*).

See the standard WSL docs for more information on how to:

- [Change the default Linux distribution installed.](#)
- [Set up your Linux username and password.](#)
- [Check which version of WSL you are running](#)
- [Update and upgrade packages.](#)
- [Add additional distributions.](#)
- [Use Git with WSL.](#)

Install WSL on previous versions of Windows Server

To install WSL on Windows Server 2019 (version 1709+), you can follow the manual install steps below.

Enable the Windows Subsystem for Linux

Before you can run Linux distributions on Windows, you must enable the "Windows Subsystem for Linux" optional feature and reboot.

Open PowerShell as Administrator and run:

```
PowerShell
```

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

Download a Linux distribution

See the [Downloading distributions](#) section of the manual installation page for instructions and links to download your preferred Linux distribution.

Extract and install a Linux distribution

Now that you've downloaded a Linux distribution, in order to extract its contents and manually install, follow these steps:

1. Extract the `<DistributionName>.appx` package's contents, using PowerShell:

```
PowerShell
```

```
Rename-Item .\Ubuntu.appx .\Ubuntu.zip  
Expand-Archive .\Ubuntu.zip .\Ubuntu
```

2. Once the distribution has been downloaded, navigate to the folder containing the download and run the following command in that directory, where `app-name` is the name of the Linux distribution .appx file.

```
Powershell
```

```
Add-AppxPackage .\app_name.appx
```

⊗ Caution

Installation failed with error 0x8007007e: If you receive this error, then your system doesn't support WSL. Ensure that you're running Windows build 16215 or later. **Check your build.** Also check to **confirm that WSL is enabled** and your computer was restarted after the feature was enabled.

3. Add your Linux distribution path to the Windows environment PATH

(`C:\Users\Administrator\Ubuntu` in this example), using PowerShell:

PowerShell

```
$userenv = [System.Environment]::GetEnvironmentVariable("Path", "User")
[System.Environment]::SetEnvironmentVariable("PATH", $userenv +
";C:\Users\Administrator\Ubuntu", "User")
```

You can now launch your distribution from any path by typing `<DistributionName>.exe`.
For example: `ubuntu.exe`.

Once installation is complete, you can [create a user account and password for your new Linux distribution](#).

Set up a WSL development environment

Article • 11/20/2023

A step-by-step guide to the best practices for setting up a WSL development environment. Learn how to run the command to install the default Bash shell that uses Ubuntu or can be set to install other Linux distributions, use basic WSL commands, set up Visual Studio Code or Visual Studio, Git, Windows Credential Manager, databases like MongoDB, Postgres, or MySQL, set up GPU acceleration, run GUI apps, and more.

Get started

Windows Subsystem for Linux comes with the Windows operating system, but you must enable it and install a Linux distribution before you can begin using it.

To use the simplified --install command, you must be running a recent build of Windows (Build 20262+). To check your version and build number, select **Windows logo key + R**, type **winver**, select **OK**. You can update using the [Settings menu](#) or [Windows Update Assistant](#).

If you prefer to install a Linux distribution other than Ubuntu, or would prefer to complete these steps manually, see the [WSL installation page](#) for more details.

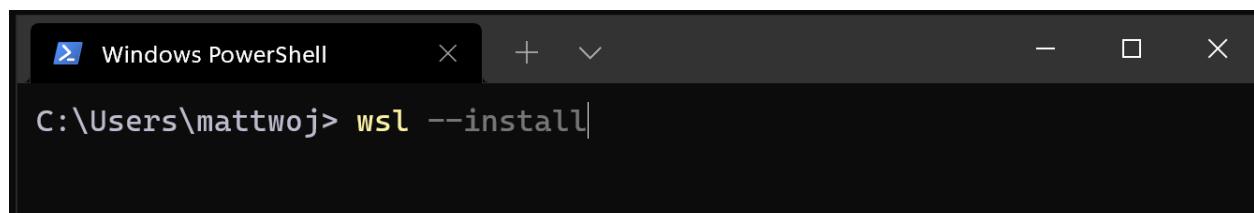
Open PowerShell (or Windows Command Prompt) and enter:

```
PowerShell  
wsl --install
```

The --install command performs the following actions:

- Enables the optional WSL and Virtual Machine Platform components
- Downloads and installs the latest Linux kernel
- Sets WSL 2 as the default
- Downloads and installs the Ubuntu Linux distribution (reboot may be required)

You will need to restart your machine during this installation process.



Check the [troubleshooting installation](#) article if you run into any issues.

Set up your Linux username and password

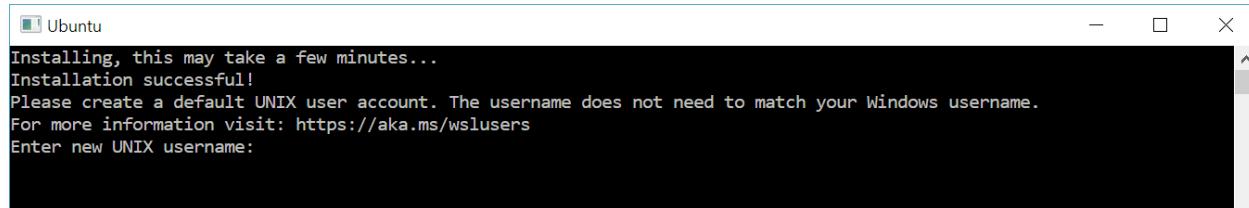
Once the process of installing your Linux distribution with WSL is complete, open the distribution (Ubuntu by default) using the Start menu. You will be asked to create a **User Name** and **Password** for your Linux distribution.

- This **User Name** and **Password** is specific to each separate Linux distribution that you install and has no bearing on your Windows user name.
- Please note that whilst entering the **Password**, nothing will appear on screen. This is called blind typing. You won't see what you are typing, this is completely normal.
- Once you create a **User Name** and **Password**, the account will be your default user for the distribution and automatically sign-in on launch.
- This account will be considered the Linux administrator, with the ability to run `sudo` (Super User Do) administrative commands.
- Each Linux distribution running on WSL has its own Linux user accounts and passwords. You will have to configure a Linux user account every time you add a distribution, reinstall, or reset.

ⓘ Note

Linux distributions installed with WSL are a per-user installation and can't be shared with other Windows user accounts. Encountering a username error?

[StackExchange: What characters should I use or not use in usernames on Linux?](#) ↗



To change or reset your password, open the Linux distribution and enter the command: `passwd`. You will be asked to enter your current password, then asked to enter your new password, and then to confirm your new password.

If you forgot the password for your Linux distribution:

1. Open PowerShell and enter the root of your default WSL distribution using the command: `wsl -u root`

If you need to update the forgotten password on a distribution that is not your default, use the command: `wsl -d Debian -u root`, replacing `Debian` with the name of your targeted distribution.

2. Once your WSL distribution has been opened at the root level inside PowerShell, you can use this command to update your password: `passwd <username>` where `<username>` is the username of the account in the distribution whose password you've forgotten.
3. You will be prompted to enter a new UNIX password and then confirm that password. Once you're told that the password has updated successfully, close WSL inside of PowerShell using the command: `exit`.

Update and upgrade packages

We recommend that you regularly update and upgrade your packages using the preferred package manager for the distribution. For Ubuntu or Debian, use the command:

Bash

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

Windows does not automatically update or upgrade your Linux distribution(s). This is a task that most Linux users prefer to control themselves.

Add additional distributions

To add additional Linux distributions, you can install via the [Microsoft Store](#), via the [--import command](#), or by [sideloading your own custom distribution](#). You may also want to [set up custom WSL images for distribution across your enterprise company](#).

Set up Windows Terminal

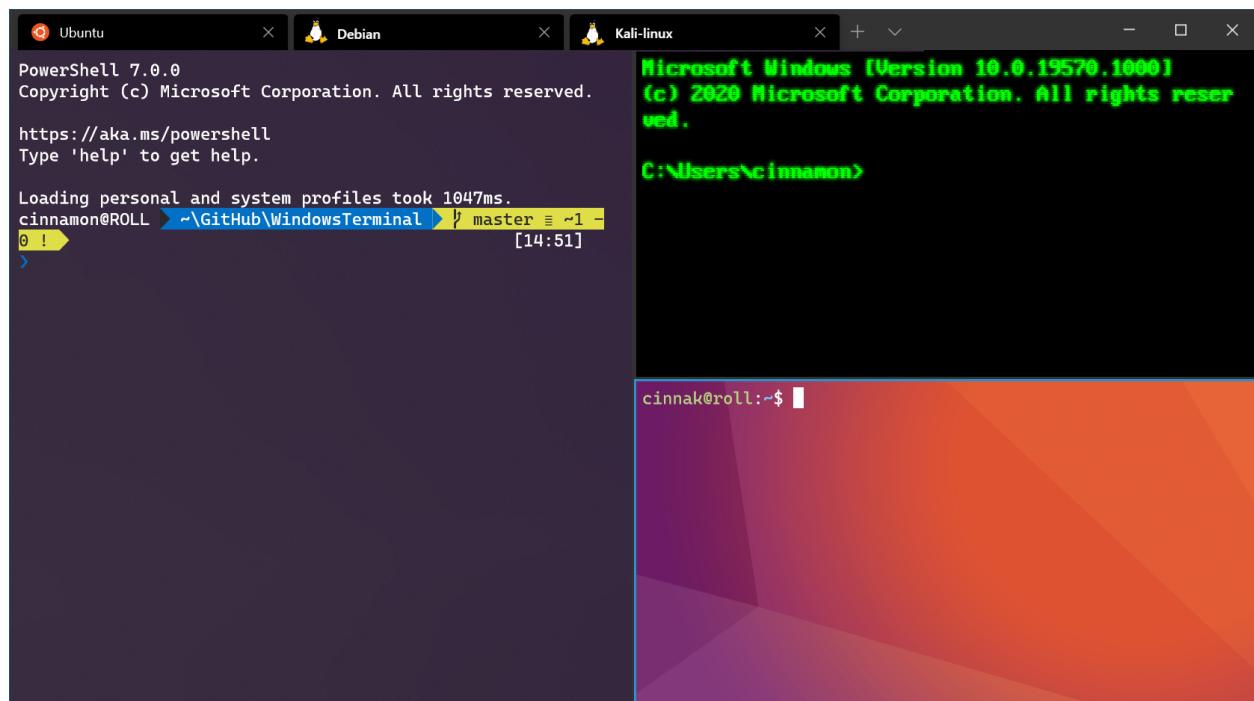
Windows Terminal can run any application with a command line interface. Its main features include multiple tabs, panes, Unicode and UTF-8 character support, a GPU

accelerated text rendering engine, and the ability to create your own themes and customize text, colors, backgrounds, and shortcuts.

Whenever a new WSL Linux distribution is installed, a new instance will be created for it inside the Windows Terminal that can be customized to your preferences.

We recommend using WSL with Windows Terminal, especially if you plan to work with multiple command lines. See the Windows Terminal docs for help with setting it up and customizing your preferences, including:

- [Install Windows Terminal or Windows Terminal \(Preview\)](#) from the Microsoft Store
- [Use the Command Palette](#)
- Set up [custom actions](#) like keyboard shortcuts to make the terminal feel natural to your preferences
- Set up the [default startup profile](#)
- Customize the appearance: [theme](#), [color schemes](#), [name and starting directory](#), [background image](#), etc.
- Learn how to use [command line arguments](#) like opening a terminal with multiple command lines split into window panes or tabs
- Learn about the [search feature](#)
- Find [tips and tricks](#), like how to rename or color a tab, use mouse interactions, or enable "Quake mode"
- Find tutorials on how to set up a [customized command prompt](#), [SSH profiles](#), or [tab titles](#)
- Find a [custom terminal gallery](#) and a [troubleshooting guide](#)

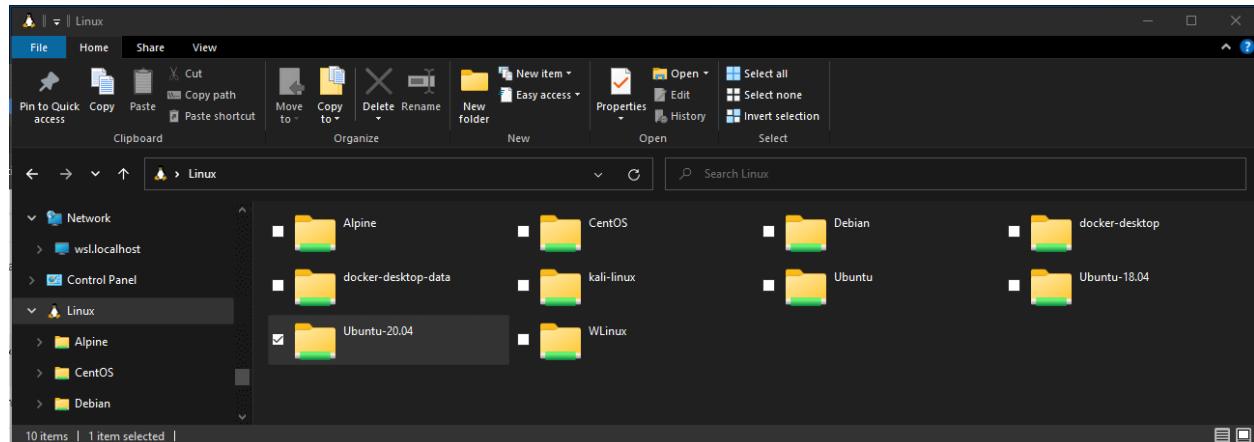


File storage

- To open your WSL project in Windows File Explorer, enter: `explorer.exe .`
Be sure to add the period at the end of the command to open the current directory.
- **Store your project files on the same operating system as the tools you plan to use.**
For the fastest performance speed, store your files in the WSL file system if you are working on them with Linux tools in a Linux command line (Ubuntu, OpenSUSE, etc). If you're working in a Windows command line (PowerShell, Command Prompt) with Windows tools, store your files in the Windows file system. Files can be accessed across the operating systems, but it may significantly slow down performance.

For example, when storing your WSL project files:

- Use the Linux file system root directory: `\wsl$\<DistroName>\home\<UserName>\Project`
- Not the Windows file system root directory: `C:\Users\<UserName>\Project` or `/mnt/c/Users/<UserName>/Project$`



Set up your favorite code editor

We recommend using Visual Studio Code or Visual Studio, as they directly support remote development and debugging with WSL. Visual Studio Code allows you to use WSL as a full-featured development environment. Visual Studio offers native WSL support for C++ cross-platform development.

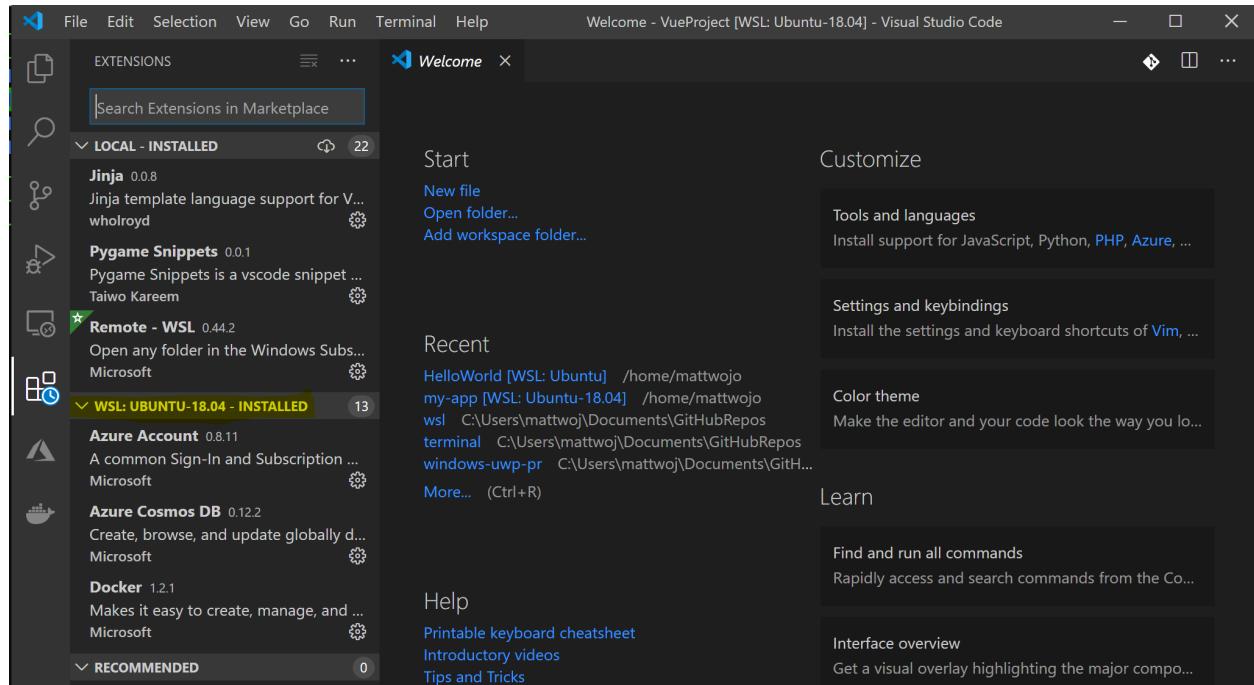
Use Visual Studio Code

Follow this step-by-step guide to [Get started using Visual Studio Code with WSL](#), which includes installing the [Remote Development extension pack](#). This extension enables

you to run WSL, SSH, or a development container for editing and debugging with the full set of Visual Studio Code features. Quickly swap between different, separate development environments and make updates without worrying about impacting your local machine.

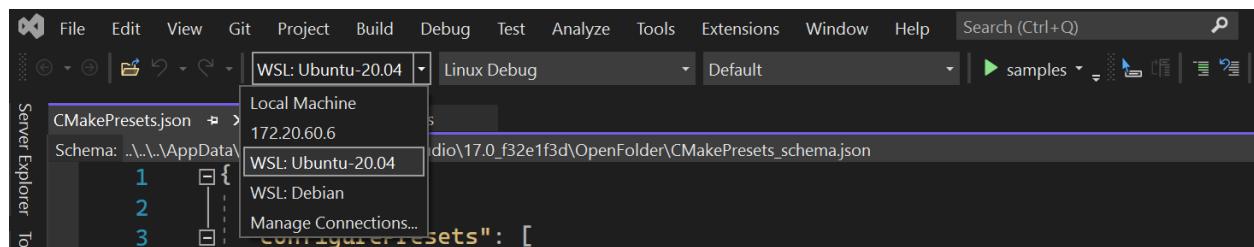
Once VS Code is installed and set up, you can open your WSL project with a VS Code remote server by entering: `code .`

Be sure to add the period at the end of the command to open the current directory.



Use Visual Studio

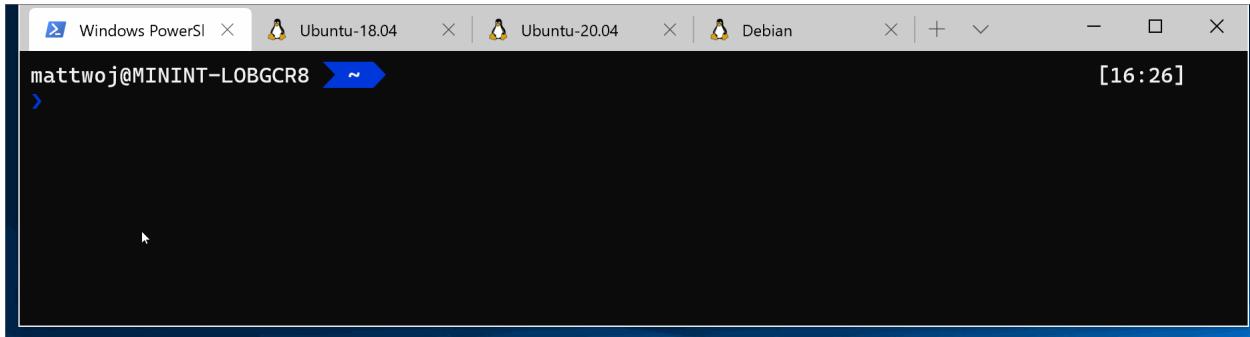
Follow this step-by-step guide to [Get started using Visual Studio with WSL for C++ cross-platform development](#). Visual Studio 2022 enables you to build and debug CMake projects on Windows, WSL distributions, and SSH connections from the same instance of Visual Studio.



Set up version management with Git

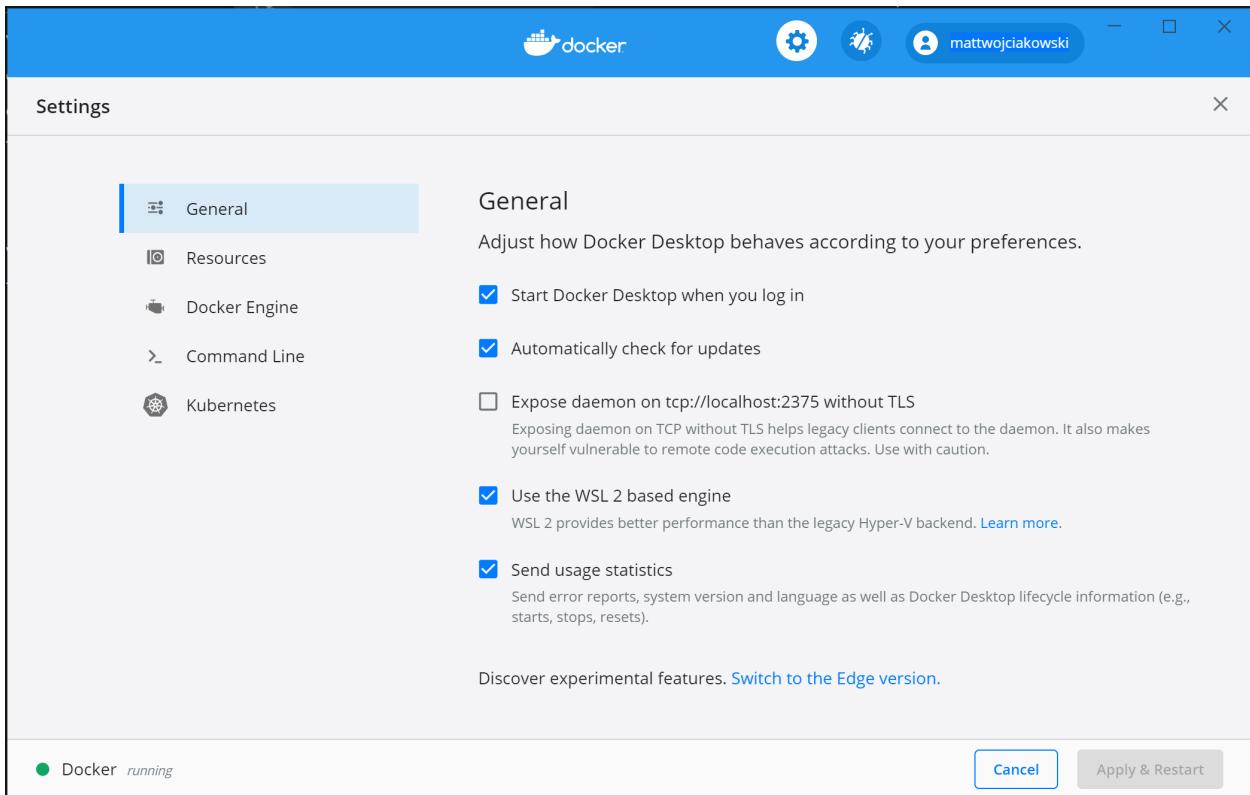
Follow this step-by-step guide to [Get started using Git on WSL](#) and connect your project to the Git version control system, along with using the credential manager for

authentication, using Git Ignore files, understanding Git line endings, and using the Git commands built-in to VS Code.



Set up remote development containers with Docker

Follow this step-by-step guide to [Get started with Docker remote containers on WSL 2](#) and connect your project to a remote development container with Docker Desktop for Windows.



Set up a database

Follow this step-by-step guide to [Get started with databases on WSL](#) and connect your project to a database in the WSL environment. Get started with MySQL, PostgreSQL, MongoDB, Redis, Microsoft SQL Server, or SQLite.

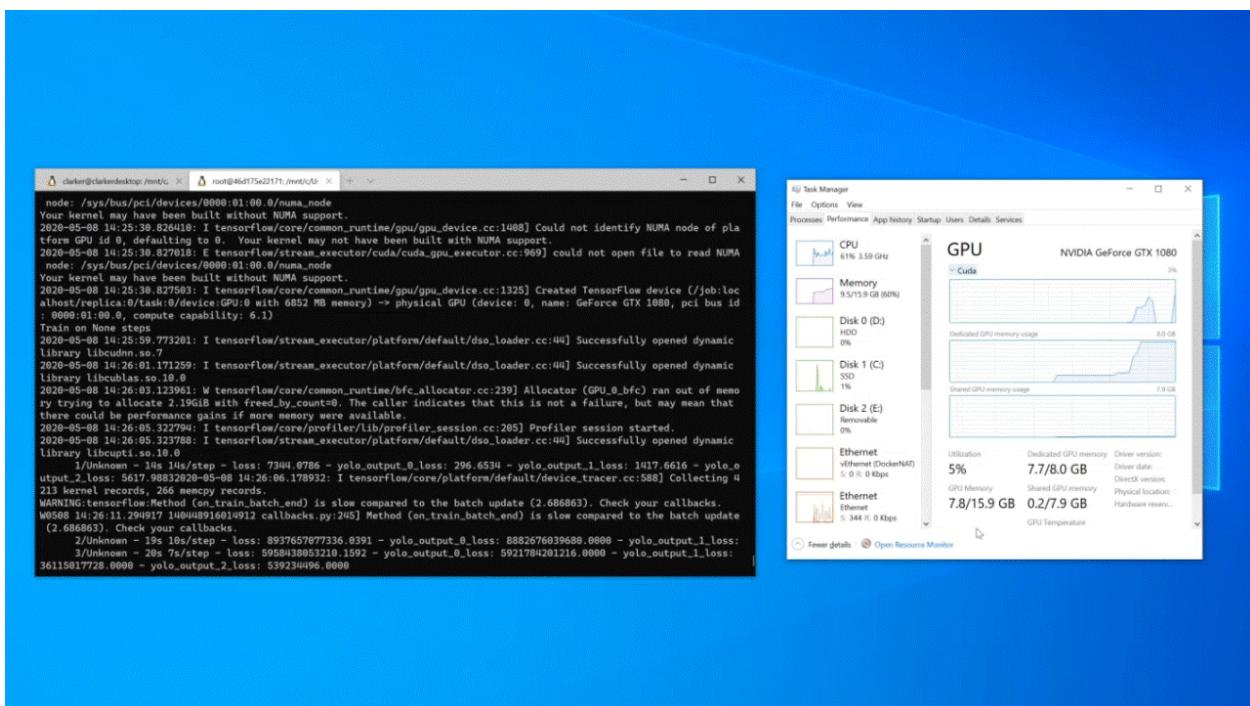
```

Ubuntu
db version v3.6.3
git version: 9586e557d54ef70f9ca4b43c26892cd55257e1a5
OpenSSL version: OpenSSL 1.1.1  11 Sep 2018
allocator: tcmalloc
modules: none
build environment:
  distarch: x86_64
  target_arch: x86_64
mattwojo@MININT-LOBGCR8:~$ sudo service mongodb start
 * Starting database mongodb
[m  OK ]
mattwojo@MININT-LOBGCR8:~$ 

```

Set up GPU acceleration for faster performance

Follow this step-by-step guide to set up [GPU accelerated machine learning training in WSL](#) and leverage your computer's GPU (graphics processing unit) to accelerate performance heavy workloads.



Basic WSL commands

The Linux distributions that you install via WSL are best managed using PowerShell or Windows Command Prompt (CMD). See the [WSL command reference guide](#) for a list of basic commands to be familiar with when using WSL.

In addition, many commands are interoperable between Windows and Linux. Here are a couple of examples:

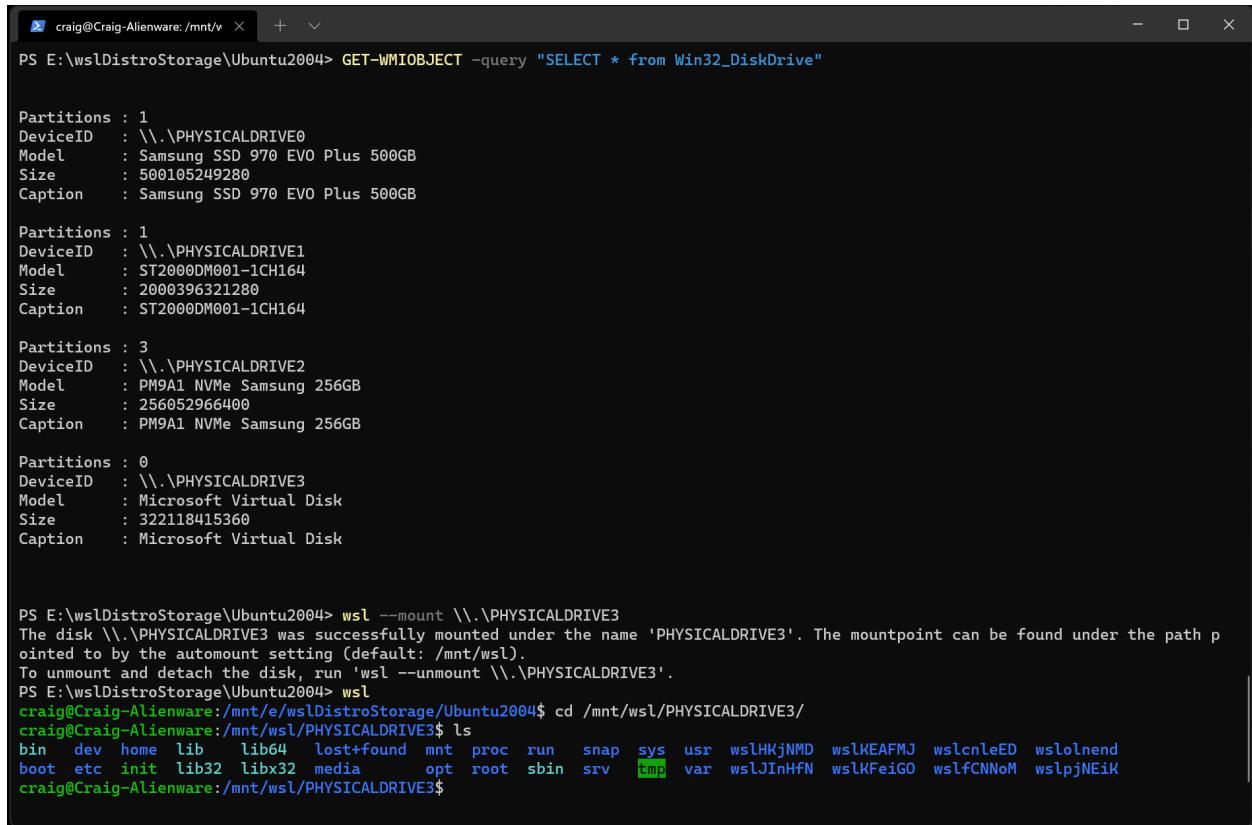
- [Run Linux tools from a Windows command line](#): Open PowerShell and display the directory contents of `C:\temp>` using the Linux `ls -la` command by entering: `wsl ls -la`

- **Mix Linux and Windows commands:** In this example, the Linux command `ls -la` is used to list files in the directory, then the PowerShell command `findstr` is used to filter the results for words containing "git": `wsl ls -la | findstr "git"`. This could also be done mixing the Windows `dir` command with the Linux `grep` command:
`dir | wsl grep git.`

- **Run a Windows tool directly from the WSL command line:** `<tool-name>.exe` For example, to open your `.bashrc` file (the shell script that runs whenever your Linux command line is started), enter: `notepad.exe .bashrc`
- **Run the Windows ipconfig.exe tool with the Linux Grep tool:** From Bash enter the command `ipconfig.exe | grep IPv4 | cut -d: -f2` or from PowerShell enter `ipconfig.exe | wsl grep IPv4 | wsl cut -d: -f2` This example demonstrates the `ipconfig` tool on the Windows file system being used to display the current TCP/IP network configuration values and then being filtered to only the IPv4 result with `grep`, a Linux tool.

Mount an external drive or USB

Follow this step-by-step guide to [Get started mounting a Linux disk in WSL 2](#).



```

PS E:\wslDistroStorage\Ubuntu2004> GET-WMIOBJECT -query "SELECT * from Win32_DiskDrive"
Partitions : 1
DeviceID   : \\.\PHYSICALDRIVE0
Model      : Samsung SSD 970 EVO Plus 500GB
Size       : 500105249280
Caption    : Samsung SSD 970 EVO Plus 500GB

Partitions : 1
DeviceID   : \\.\PHYSICALDRIVE1
Model      : ST2000DM001-1CH164
Size       : 2000396321280
Caption    : ST2000DM001-1CH164

Partitions : 3
DeviceID   : \\.\PHYSICALDRIVE2
Model      : PM9A1 NVMe Samsung 256GB
Size       : 256052966400
Caption    : PM9A1 NVMe Samsung 256GB

Partitions : 0
DeviceID   : \\.\PHYSICALDRIVE3
Model      : Microsoft Virtual Disk
Size       : 322118415360
Caption    : Microsoft Virtual Disk

PS E:\wslDistroStorage\Ubuntu2004> wsl --mount \\.\PHYSICALDRIVE3
The disk \\.\PHYSICALDRIVE3 was successfully mounted under the name 'PHYSICALDRIVE3'. The mountpoint can be found under the path pointed to by the automount setting (default: /mnt/wsl).
To unmount and detach the disk, run 'wsl --unmount \\.\PHYSICALDRIVE3'.
PS E:\wslDistroStorage\Ubuntu2004> wsl
craig@Craig-Alienware:/mnt/e/wslDistroStorage/Ubuntu2004$ cd /mnt/wsl/PHYSICALDRIVE3/
craig@Craig-Alienware:/mnt/wsl/PHYSICALDRIVE3$ ls
bin  dev  home  lib  lib64  lost+found  mnt  proc  run  snap  sys  usr  wslHKjNMD  wslKEAFMJ  wslcnleED  wslolnend
boot etc  init  lib32  libx32  media  opt  root  sbin  srv  tmp  var  wslJInHfn  wslKFeiGO  wslfcNNoM  wslpjNEiK
craig@Craig-Alienware:/mnt/wsl/PHYSICALDRIVE3$
```

Run Linux GUI apps

Follow this tutorial to learn how to set up and [run Linux GUI apps on WSL](#).

Additional resources

- [Set up your development environment on Windows](#): Learn more about setting up your development environment for your preferred language or framework, such as React, Python, NodeJS, Vue, etc.
- [Troubleshooting](#): Find common issues, where to report bugs, where to request new features, and how to contribute to the docs.
- [FAQs](#): Find a list of frequently asked questions.
- [Release Notes](#): Review the WSL Release Notes for a history of past build updates. You can also find the [release notes for the WSL Linux Kernel](#).

Get started using Visual Studio Code with Windows Subsystem for Linux

Article • 10/04/2022

Visual Studio Code, along with the WSL extension, enables you to use WSL as your full-time development environment directly from VS Code. You can:

- develop in a Linux-based environment
- use Linux-specific toolchains and utilities
- run and debug your Linux-based applications from the comfort of Windows while maintaining access to productivity tools like Outlook and Office
- use the VS Code built-in terminal to run your Linux distribution of choice
- take advantage of VS Code features like [Intellisense code completion](#), [linting](#), [debug support](#), [code snippets](#), and [unit testing](#)
- easily manage your version control with VS Code's built-in [Git support](#)
- run commands and VS Code extensions directly on your WSL projects
- edit files in your Linux or mounted Windows filesystem (for example /mnt/c) without worrying about pathing issues, binary compatibility, or other cross-OS challenges

Install VS Code and the WSL extension

- Visit the [VS Code install page](#) and select the 32 or 64 bit installer. Install Visual Studio Code on Windows (not in your WSL file system).
- When prompted to **Select Additional Tasks** during installation, be sure to check the **Add to PATH** option so you can easily open a folder in WSL using the `code` command.
- Install the [Remote Development extension pack](#). This extension pack includes the WSL extension, in addition to the Remote - SSH, and Dev Containers extensions, enabling you to open any folder in a container, on a remote machine, or in WSL.

Important

In order to install the WSL extension, you will need the [1.35 May release](#) version or later of VS Code. We do not recommend using WSL in VS Code without the WSL extension as you will lose support for auto-complete, debugging, linting, etc. Fun

fact: this WSL extension is installed in \$HOME/.vscode/extensions (enter the command `ls $HOME\.vscode\extensions\` in PowerShell).

Update your Linux distribution

Some WSL Linux distributions are lacking libraries that are required by the VS Code server to start up. You can add additional libraries into your Linux distribution by using its package manager.

For example, to update Debian or Ubuntu, use:

```
Bash
```

```
sudo apt-get update
```

To add wget (to retrieve content from web servers) and ca-certificates (to allow SSL-based applications to check for the authenticity of SSL connections), enter:

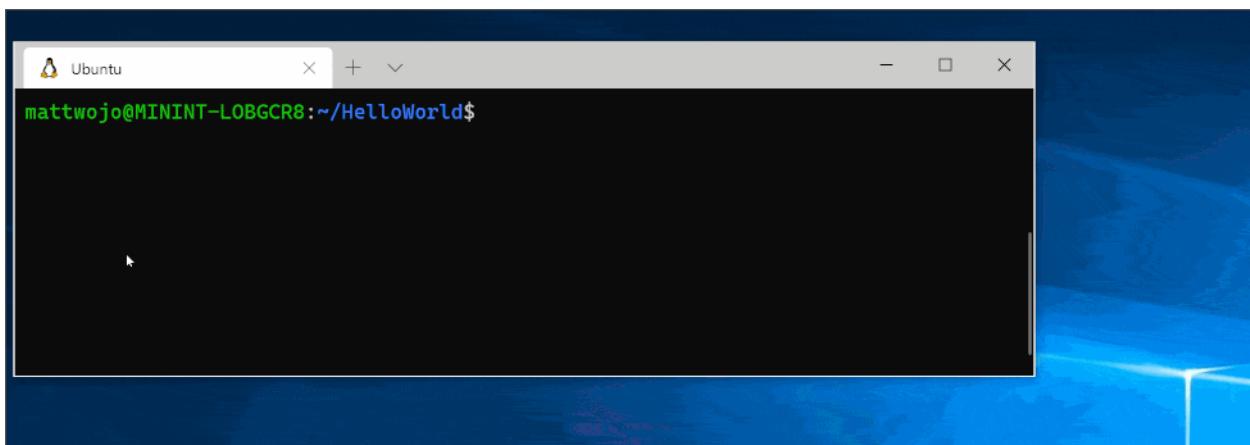
```
Bash
```

```
sudo apt-get install wget ca-certificates
```

Open a WSL project in Visual Studio Code

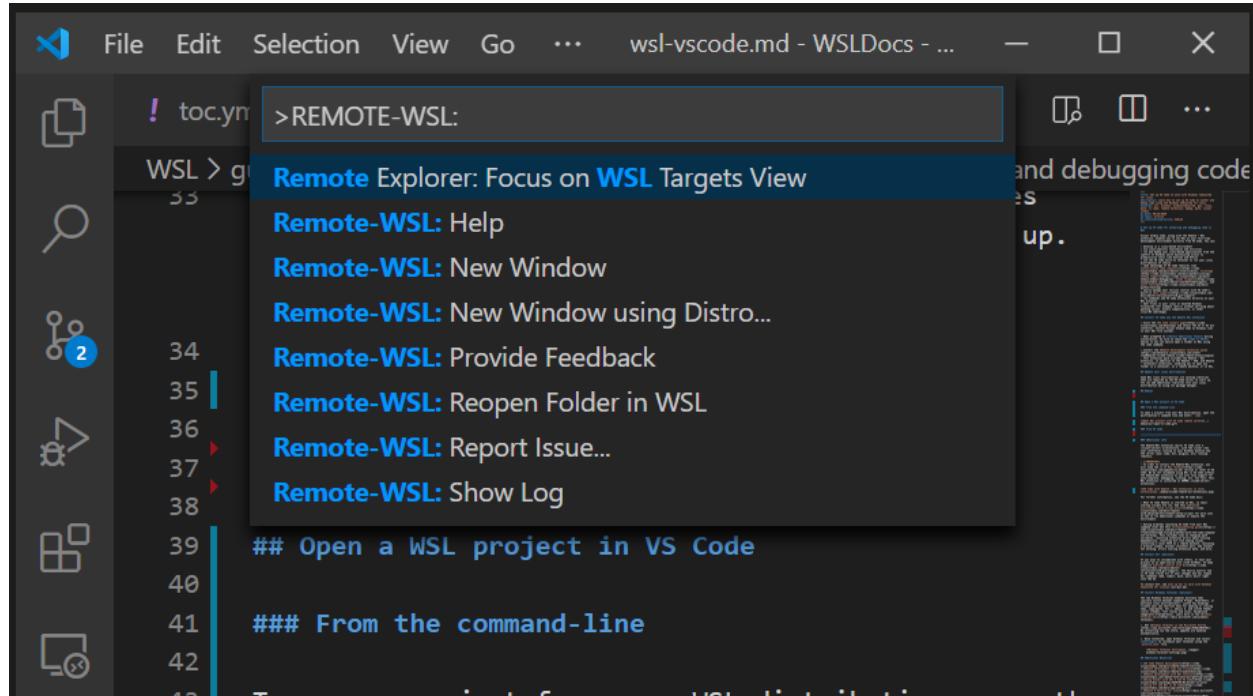
From the command-line

To open a project from your WSL distribution, open the distribution's command line and enter: `code .`



From VS Code

You can also access more VS Code WSL options by using the shortcut: **CTRL+SHIFT+P** in VS Code to bring up the command palette. If you then type `WSL` you will see a list of the options available, allowing you to reopen the folder in a WSL session, specify which distribution you want to open in, and more.



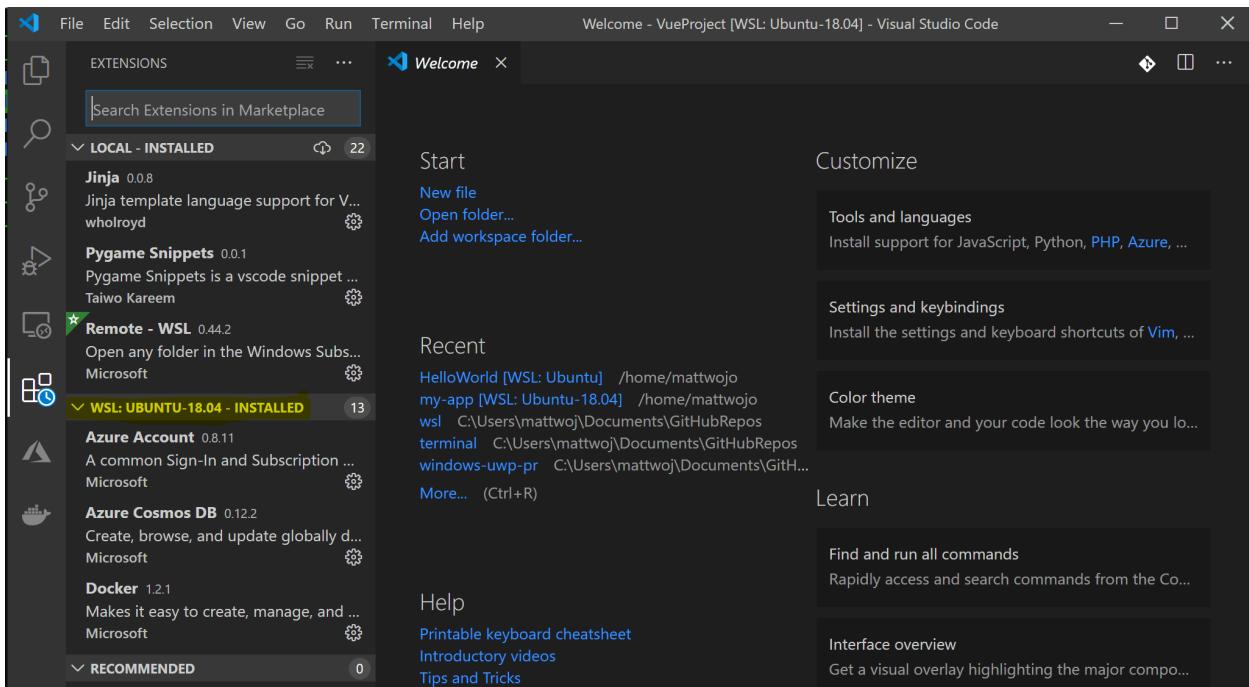
Extensions inside of VS Code WSL

The WSL extension splits VS Code into a “client-server” architecture, with the client (the user interface) running on your Windows machine and the server (your code, Git, plugins, etc) running “remotely” in your WSL distribution.

When running the WSL extension, selecting the 'Extensions' tab will display a list of extensions split between your local machine and your WSL distribution.

Installing a local extension, like a [theme](#), only needs to be installed once.

Some extensions, like the [Python extension](#) or anything that handles things like linting or debugging, must be installed separately on each WSL distribution. VS Code will display a warning icon along with a green "Install in WSL" button, if you have an extension locally installed that is not installed on your WSL distribution.



For further information, see the VS Code docs:

- When VS Code is started in WSL, no shell startup scripts are run. See this [advanced environment setup script article](#) for more info on how to run additional commands or modify the environment.
- Having problems launching VS Code from your WSL command line? This [troubleshooting guide](#) includes tips on changing path variables, resolving extension errors about missing dependencies, resolving Git line ending issues, installing a local VSIX on a remote machine, launching a browser window, blocker localhost port, web sockets not working, errors storing extension data, and more.

Install Git (optional)

If you plan to collaborate with others, or host your project on an open-source site (like GitHub), VS Code supports [version control with Git](#). The Source Control tab in VS Code tracks all of your changes and has common Git commands (add, commit, push, pull) built right into the UI.

To install Git, see [set up Git to work with Windows Subsystem for Linux](#).

Install Windows Terminal (optional)

The new Windows Terminal enables multiple tabs (quickly switch between Command Prompt, PowerShell, or multiple Linux distributions), custom key bindings (create your own shortcut keys for opening or closing tabs, copy+paste, etc.), emojis ☺, and custom

themes (color schemes, font styles and sizes, background image/blur/transparency). Learn more in the [Windows Terminal docs](#).

1. Get [Windows Terminal in the Microsoft Store](#): By installing via the store, updates are handled automatically.
2. Once installed, open Windows Terminal and select **Settings** to customize your terminal using the `profile.json` file.

Additional Resources

- [VS Code WSL documentation](#)
- [VS Code WSL tutorial](#)
- [Remote development tips and tricks](#)
- [Using Docker with WSL 2 and VS Code](#)
- [Using C++ and WSL in VS Code](#)
- [Remote R Service for Linux](#)

A few additional extensions you may want to consider include:

- [Keymaps from other editors](#): These extensions can help your environment feel right at home if you're transitioning from another text editor (like Atom, Sublime, Vim, eMacs, Notepad++, etc).
- [Settings Sync](#): Enables you to synchronize your VS Code settings across different installations using GitHub. If you work on different machines, this helps keep your environment consistent across them.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Get started using Git on Windows Subsystem for Linux

Article • 06/21/2023

Git is the most commonly used version control system. With Git, you can track changes you make to files, so you have a record of what has been done, and have the ability to revert to earlier versions of the files if needed. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.

Git can be installed on Windows AND on WSL

An important consideration: when you enable WSL and install a Linux distribution, you are installing a new file system, separated from the Windows NTFS C:\ drive on your machine. In Linux, drives are not given letters. They are given mount points. The root of your file system / is the mount point of your root partition, or folder, in the case of WSL. Not everything under / is the same drive. For example, on my laptop, I've installed two versions of Ubuntu (20.04 and 18.04), as well as Debian. If I open those distributions, select the home directory with the command cd ~, and then enter the command explorer.exe ., Windows File Explorer will open and show me the directory path for that distribution.

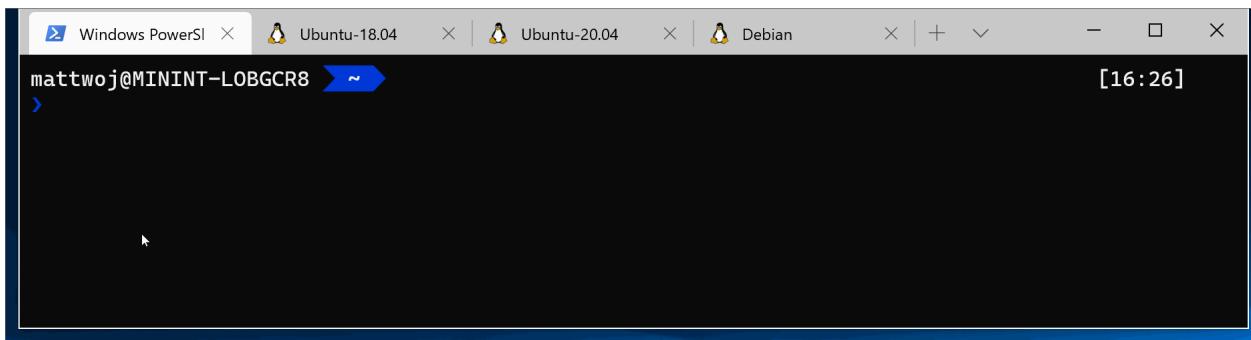
[+] Expand table

Linux distro	Windows Path to access home folder
Ubuntu 20.04	\\\wsl\$\Ubuntu-20.04\home\username
Ubuntu 18.04	\\\wsl\$\Ubuntu-18.04\home\username
Debian	\\\wsl\$\Debian\home\username
Windows PowerShell	C:\Users\username

Tip

If you are seeking to access the Windows file directory from your WSL distribution command line, instead of C:\Users\username, the directory would be accessed using /mnt/c/Users/username, because the Linux distribution views your Windows file system as a mounted drive.

You will need to install Git on each file system that you intend to use it with.



Installing Git

Git comes already installed with most of the Windows Subsystem for Linux distributions, however, you may want to update to the latest version. You also will need to set up your git config file.

To install Git, see the [Git Download for Linux ↗](#) site. Each Linux distribution has their own package manager and install command.

For the latest stable Git version in Ubuntu/Debian, enter the command:

```
Bash  
  
sudo apt-get install git
```

ⓘ Note

You also may want to [install Git for Windows ↗](#) if you haven't already.

Git config file setup

To set up your Git config file, open a command line for the distribution you're working in and set your name with this command (replacing "Your Name" with your preferred username):

```
Bash  
  
git config --global user.name "Your Name"
```

Set your email with this command (replacing "youremail@domain.com" with the email you prefer):

Bash

```
git config --global user.email "youremail@domain.com"
```

Tip

If you don't yet have a GitHub account, you can [sign-up for one on GitHub](#). If you've never worked with Git before, [GitHub Guides](#) can help you get started. If you need to edit your Git config, you can do so with a built-in text editor like nano:

```
nano ~/.gitconfig
```

We recommend that you [secure your account with two-factor authentication \(2FA\)](#).

Git Credential Manager setup

[Git Credential Manager \(GCM\)](#) is a secure Git credential helper built on [.NET](#) that can be used with both WSL1 and WSL2. It enables multi-factor authentication support for GitHub repos, [Azure DevOps](#), Azure DevOps Server, and Bitbucket.

GCM integrates into the authentication flow for services like GitHub and, once you're authenticated to your hosting provider, requests a new authentication token. It then stores the token securely in the [Windows Credential Manager](#). After the first time, you can use Git to talk to your hosting provider without needing to re-authenticate. It will just access the token in the Windows Credential Manager.

In order to use GCM with WSL you must be on Windows 10 Version 1903 or later. This is the first version of Windows that includes the required `wsl.exe` tool that GCM uses to interoperate with Git in your WSL distributions.

It is recommended to install the [latest Git for Windows](#) in order to share credentials & settings between WSL and the Windows host. Git Credential Manager is included with Git for Windows and the latest version is included in each new Git for Windows release. During the installation, you will be asked to select a credential helper, with GCM set as the default.

If you have a reason not to install Git for Windows, you can install GCM as a Linux application directly in your WSL distribution, but note that doing so means GCM is running as a Linux application and cannot utilize the authentication or credential storage features of the host Windows operating system. See the GCM repo for instructions on how to [configure WSL without Git for Windows](#).

To set up GCM for use with a WSL distribution, open your distribution and enter this command:

If GIT installed is >= v2.39.0

```
Bash
```

```
git config --global credential.helper "/mnt/c/Program Files/Git/mingw64/bin/git-credential-manager.exe"
```

else if GIT installed is >= v2.36.1

```
Bash
```

```
git config --global credential.helper "/mnt/c/Program Files/Git/mingw64/libexec/git-core/git-credential-manager.exe"
```

else if version is < v2.36.1 enter this command:

```
Bash
```

```
git config --global credential.helper "/mnt/c/Program Files/Git/mingw64/bin/git-credential-manager-core.exe"
```

ⓘ Note

Using GCM as a credential helper for a WSL Git installation means that any configuration set in WSL Git is NOT respected by GCM (by default). This is because GCM is running as a Windows application, and therefore will use the Git for Windows installation to query configuration. This means things like proxy settings for GCM need to be set in Git for Windows as well as WSL Git as they are stored in different files (`%USERPROFILE%\.gitconfig` vs `\wsl$\distro\home\$USER\.gitconfig`). You can configure WSL so that GCM will use the WSL Git configuration, but this means that proxy settings will be unique to the specific WSL installation and not shared with others or the Windows host.

Git with SSH

Git Credential Manager only works with HTTP(S) remotes. You can still use Git with SSH:

- [Azure DevOps SSH](#)
- [GitHub SSH ↗](#)

- Bitbucket SSH ↗

Additional configuration for Azure

If you intend to work with [Azure Repos ↗](#) or [Azure DevOps ↗](#), some additional configuration is required:

Bash

```
git config --global credential.https://dev.azure.com.useHttpPath true
```

Now any git operation you perform within your WSL distribution will use GCM. If you already have credentials cached for a host, it will access them from the credential manager. If not, you'll receive a dialog response requesting your credentials, even if you're in a Linux console.

💡 Tip

If you are using a GPG key for code signing security, you may need to [associate your GPG key with your GitHub email ↗](#).

Adding a Git Ignore file

We recommend adding a [.gitignore file ↗](#) to your projects. GitHub offers [a collection of useful .gitignore templates ↗](#) with recommended .gitignore file setups organized according to your use-case. For example, here is [GitHub's default gitignore template for a Node.js project ↗](#).

If you choose to [create a new repo using the GitHub website ↗](#), there are check boxes available to initialize your repo with a README file, .gitignore file set up for your specific project type, and options to add a license if you need one.

Git and VS Code

Visual Studio Code comes with built-in support for Git, including a source control tab that will show your changes and handle a variety of git commands for you. Learn more about [VS Code's Git support ↗](#).

Git line endings

If you are working with the same repository folder between Windows, WSL, or a container, be sure to set up consistent line endings.

Since Windows and Linux use different default line endings, Git may report a large number of modified files that have no differences aside from their line endings. To prevent this from happening, you can disable line ending conversion using a `.gitattributes` file or globally on the Windows side. See this [VS Code doc about resolving Git line ending issues ↗](#).

Additional resources

- [WSL & VS Code](#)
- [GitHub Learning Lab: Online courses ↗](#)
- [Git Visualization Tool ↗](#)
- [Git Tools - Credential Storage ↗](#)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Get started with databases on Windows Subsystem for Linux

Article • 09/19/2023

This step-by-step guide will help you get started connecting your project in WSL to a database. Get started with MySQL, PostgreSQL, MongoDB, Redis, Microsoft SQL Server, or SQLite.

Prerequisites

- Running Windows 11 or Windows 10, [updated to version 2004, Build 19041](#) or later.
- [Install a Linux distribution using WSL and create a Linux user name and password.](#)

Differences between database systems

Some [popular choices](#) for a database system include:

- [MySQL](#) (SQL)
- [PostgreSQL](#) (SQL)
- [Microsoft SQL Server](#) (SQL)
- [SQLite](#) (SQL)
- [MongoDB](#) (NoSQL)
- [Redis](#) (NoSQL)

MySQL is an open-source SQL relational database, organizing data into one or more tables in which data types may be related to each other. It is vertically scalable, which means one ultimate machine will do the work for you. It is currently the most widely used of the four database systems.

PostgreSQL (sometimes referred to as Postgres) is also an open-source SQL relational database with an emphasis on extensibility and standards compliance. It can handle JSON now too, but it is generally better for structured data, vertical scaling, and ACID-compliant needs like eCommerce and financial transactions.

Microsoft SQL Server includes SQL Server on Windows, SQL Server on Linux, and SQL on Azure. These are also relational database management systems set up on servers with primary function of storing and retrieving data as requested by software applications.

SQLite is an open-source self-contained, file-based, “serverless” database, known for its portability, reliability, and good performance even in low-memory environments.

MongoDB is an open-source NoSQL document database designed to work with JSON and store schema-free data. It is horizontally scalable, which means multiple smaller machines will do the work for you. It's good for flexibility and unstructured data, and caching real-time analytics.

Redis is an open-source NoSQL in-memory data structure store. It uses key-value pairs for storage instead of documents.

Install MySQL

To install MySQL on a Linux distribution running on WSL, just follow the [Installing MySQL on Linux](#) instructions in the MySQL docs. You may need to first [enable systemd support](#) in your `wsl.conf` configuration file.

Example using the Ubuntu distribution:

1. Open your Ubuntu command line and update the packages available: `sudo apt update`
2. Once the packages have updated, install MySQL with: `sudo apt install mysql-server`
3. Confirm installation and get the version number: `mysql --version`
4. Start MySQL Server / check status: `systemctl status mysql`
5. To open the MySQL prompt, enter: `sudo mysql`
6. To see what databases you have available, in the MySQL prompt, enter: `SHOW DATABASES;`
7. To create a new database, enter: `CREATE DATABASE database_name;`
8. To delete a database, enter: `DROP DATABASE database_name;`

For more about working with MySQL databases, see the [MySQL docs](#).

To work with MySQL databases in VS Code, try the [MySQL extension](#).

You may also want to run the included security script. This changes some of the less secure default options for things like remote root logins and sample users. This script also includes steps to change password for MySQL root user. To run the security script:

1. Start a MySQL server: `sudo service mysql start`
2. Start the security script prompts: `sudo mysql_secure_installation`

3. The first prompt will ask whether you'd like to set up the VALIDATE PASSWORD COMPONENT, which can be used to test the strength of your MySQL password. If you want to set some simple password, you should not set this component.
4. You will then set/change password for the MySQL root user, decide whether or not to remove anonymous users, decide whether to allow the root user to login both locally and remotely, decide whether to remove the test database, and, lastly, decide whether to reload the privilege tables immediately.

Install PostgreSQL

To install PostgreSQL on WSL (ie. Ubuntu):

1. Open your WSL terminal (ie. Ubuntu).
2. Update your Ubuntu packages: `sudo apt update`
3. Once the packages have updated, install PostgreSQL (and the -contrib package which has some helpful utilities) with: `sudo apt install postgresql postgresql-contrib`
4. Confirm installation and get the version number: `psql --version`

There are 3 commands you need to know once PostgreSQL is installed:

- `sudo service postgresql status` for checking the status of your database.
- `sudo service postgresql start` to start running your database.
- `sudo service postgresql stop` to stop running your database.

The default admin user, `postgres`, needs a password assigned in order to connect to a database. To set a password:

1. Enter the command: `sudo passwd postgres`
2. You will get a prompt to enter your new password.
3. Close and reopen your terminal.

To run PostgreSQL with `psql` shell:

1. Start your postgres service: `sudo service postgresql start`
2. Connect to the postgres service and open the psql shell: `sudo -u postgres psql`

Once you have successfully entered the psql shell, you will see your command line change to look like this: `postgres=#`

ⓘ Note

Alternatively, you can open the psql shell by switching to the postgres user with: `su`

- `postgres` and then entering the command: `psql`.

To exit `postgres=#` enter: `\q` or use the shortcut key: `Ctrl+D`

To see what user accounts have been created on your PostgreSQL installation, use from your WSL terminal: `psql --command="\du"` ...or just `\du` if you have the psql shell open. This command will display columns: Account User Name, List of Roles Attributes, and Member of role group(s). To exit back to the command line, enter: `q`.

For more about working with PostgreSQL databases, see the [PostgreSQL docs](#).

To work with PostgreSQL databases in VS Code, try the [PostgreSQL extension](#).

Install MongoDB

To install MongoDB, see the Mongodbs docs: [Install MongoDB Community Edition on Linux](#)

Installing MongoDB may require slightly different steps depending on the Linux distribution being used for installation. Also note that MongoDB installation may differ depending on the version # that you are aiming to install. Use the version drop-down list in the top-left corner of the MongoDB documentation to select the version that aligns with your goal. Lastly, you may need to [enable systemd support](#) in the `wsl.conf` configuration file of the Linux distribution that you are using with WSL. The `systemctl` command is a part of the systemd init system and may not work if your distribution is using systemv.

VS Code supports working with MongoDB databases via the [Azure CosmosDB extension](#), you can create, manage and query MongoDB databases from within VS Code. To learn more, visit the VS Code docs: [Working with MongoDB](#).

Learn more in the MongoDB docs:

- [Introduction to using MongoDB](#)
- [Create users](#)
- [CRUD: Create, Read, Update, Delete](#)
- [Reference Docs](#)

Install Microsoft SQL Server

To install SQL Server on a Linux distribution run by WSL: [SQL Server on Linux](#). To work with Microsoft SQL Server databases in VS Code, try the [MSSQL extension](#).

Install SQLite

To install SQLite on WSL (ie. Ubuntu):

1. Open your WSL terminal (ie. Ubuntu).
2. Update your Ubuntu packages: `sudo apt update`
3. Once the packages have updated, install SQLite3 with: `sudo apt install sqlite3`
4. Confirm installation and get the version number: `sqlite3 --version`

To create a test database, called "example.db", enter: `sqlite3 example.db`

To see a list of your SQLite databases, enter: `.databases`

To see the status of your database, enter: `.dbinfo ?DB?`

Database will be empty after creation. You can create a new table for your database with
`CREATE TABLE empty (kol INTEGER);`.

Now entering the `.dbinfo ?DB?` will show the database you have created.

To exit the SQLite prompt, enter: `.exit`

For more information about working with a SQLite database, see the [SQLite docs](#).

To work with SQLite databases in VS Code, try the [SQLite extension](#).

Install Redis

To install Redis on WSL (ie. Ubuntu):

1. Open your WSL terminal (ie. Ubuntu).
2. Update your Ubuntu packages: `sudo apt update`
3. Once the packages have updated, install Redis with: `sudo apt install redis-server`
4. Confirm installation and get the version number: `redis-server --version`

To start running your Redis server: `sudo service redis-server start`

Check to see if redis is working (redis-cli is the command line interface utility to talk with Redis): `redis-cli ping` this should return a reply of "PONG".

To stop running your Redis server: `sudo service redis-server stop`

For more information about working with a Redis database, see the [Redis docs ↗](#).

To work with Redis databases in VS Code, try the [Redis extension ↗](#).

See services running and set up profile aliases

To see the services that you currently have running on your WSL distribution, enter:

```
service --status-all
```

Typing out `sudo service mongodb start` or `sudo service postgres start` and `sudo -u postrest psql` can get tedious. However, you could consider setting up aliases in your `.profile` file on WSL to make these commands quicker to use and easier to remember.

To set up your own custom alias, or shortcut, for executing these commands:

1. Open your WSL terminal and enter `cd ~` to be sure you're in the root directory.
2. Open the `.profile` file, which controls the settings for your terminal, with the terminal text editor, Nano: `sudo nano .profile`
3. At the bottom of the file (don't change the `# set PATH` settings), add the following:

Bash

```
# My Aliases
alias start-pg='sudo service postgresql start'
alias run-pg='sudo -u postgres psql'
```

This will allow you to enter `start-pg` to start running the postgresql service and `run-pg` to open the psql shell. You can change `start-pg` and `run-pg` to whatever names you want, just be careful not to overwrite a command that postgres already uses!

4. Once you've added your new aliases, exit the Nano text editor using `Ctrl+X` -- select `Y` (Yes) when prompted to save and Enter (leaving the file name as `.profile`).
5. Close and re-open your WSL terminal, then try your new alias commands.

Troubleshooting

Error: directory-sync fdatasync Invalid argument

Ensure that you are running your Linux distribution in WSL 2 mode. For help switching from WSL 1 to WSL 2, see [Set your distribution version to WSL 1 or WSL 2](#).

Additional resources

- [Setting up your development environment on Windows](#)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

-  [Open a documentation issue](#)
-  [Provide product feedback](#)

Get started with Docker remote containers on WSL 2

Article • 01/10/2024

This step-by-step guide will help you get started developing with remote containers by [setting up Docker Desktop for Windows with WSL 2](#) (Windows Subsystem for Linux, version 2).

Docker Desktop for Windows provides a development environment for building, shipping, and running dockerized apps. By enabling the WSL 2 based engine, you can run both Linux and Windows containers in Docker Desktop on the same machine. (Docker Desktop is free for personal use and small businesses, for info on Pro, Team, or Business pricing, see the [Docker site FAQs](#)).

Note

We recommend using Docker Desktop due to its [integration with Windows and Windows Subsystem for Linux](#). However, while Docker Desktop supports running both Linux and Windows containers, you can **not** run both simultaneously. To run Linux and Windows containers simultaneously, you would need to install and run a separate Docker instance in WSL. If you need to run simultaneous containers or just prefer to install a container engine directly in your Linux distribution, follow the Linux installation instructions for that container service, such as [Install Docker Engine on Ubuntu](#) or [Install Podman for running Linux containers](#).

Overview of Docker containers

Docker is a tool used to create, deploy, and run applications using containers. Containers enable developers to package an app with all of the parts it needs (libraries, frameworks, dependencies, etc) and ship it all out as one package. Using a container ensures that the app will run the same regardless of any customized settings or previously installed libraries on the computer running it that could differ from the machine that was used to write and test the app's code. This permits developers to focus on writing code without worrying about the system that code will be run on.

Docker containers are similar to virtual machines, but don't create an entire virtual operating system. Instead, Docker enables the app to use the same Linux kernel as the system that it's running on. This allows the app package to only require parts not already on the host computer, reducing the package size and improving performance.

Continuous availability, using Docker containers with tools like [Kubernetes](#), is another reason for the popularity of containers. This enables multiple versions of your app container to be created at different times. Rather than needing to take down an entire system for updates or maintenance, each container (and its specific microservices) can be replaced on the fly. You can prepare a new container with all of your updates, set up the container for production, and just point to the new container once it's ready. You can also archive different versions of your app using containers and keep them running as a safety fallback if needed.

To learn more, check out [Introduction to Docker containers](#).

Prerequisites

- WSL version 1.1.3.0 or later.
- Windows 11 64-bit: Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2 or higher.
- Windows 10 64-bit (Recommended): Home or Pro 22H2 (build 19045) or higher, or Enterprise or Education 22H2 (build 19045) or higher. (Minimum): Home or Pro 21H2 (build 19044) or higher, or Enterprise or Education 21H2 (build 19044) or higher. [Update Windows](#)
- 64-bit processor with [Second Level Address Translation \(SLAT\)](#).
- 4GB system RAM.
- Enable hardware virtualization in BIOS.
- [Install WSL and set up a user name and password for your Linux distribution running in WSL 2](#).
- [Install Visual Studio Code](#) (*optional*). This will provide the best experience, including the ability to code and debug inside a remote Docker container and connected to your Linux distribution.
- [Install Windows Terminal](#) (*optional*). This will provide the best experience, including the ability to customize and open multiple terminals in the same interface (including Ubuntu, Debian, PowerShell, Azure CLI, or whatever you prefer to use).
- [Sign up for a Docker ID at Docker Hub](#) (*optional*).
- See the [Docker Desktop license agreement](#) for updates on the terms of use.

For more information, see the [Docker docs System requirements to Install Docker Desktop on Windows](#).

To learn how to install Docker on Windows Server, see [Get started: Prep Windows for containers](#).

Note

WSL can run distributions in both WSL version 1 or WSL 2 mode. You can check this by opening PowerShell and entering: `wsl -l -v`. Ensure that your distribution is set to use WSL 2 by entering: `wsl --set-version <distro> 2`. Replace `<distro>` with the distro name (e.g. Ubuntu 18.04).

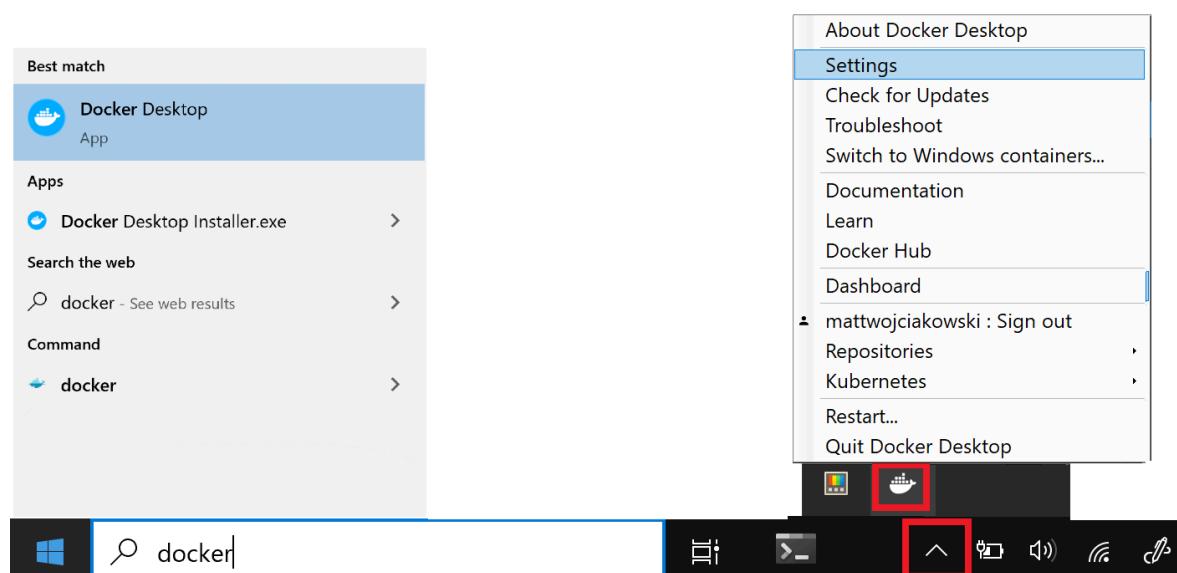
In WSL version 1, due to fundamental differences between Windows and Linux, the Docker Engine couldn't run directly inside WSL, so the Docker team developed an alternative solution using Hyper-V VMs and LinuxKit. However, since WSL 2 now runs on a Linux kernel with full system call capacity, Docker can fully run in WSL 2. This means that Linux containers can run natively without emulation, resulting in better performance and interoperability between your Windows and Linux tools.

Install Docker Desktop

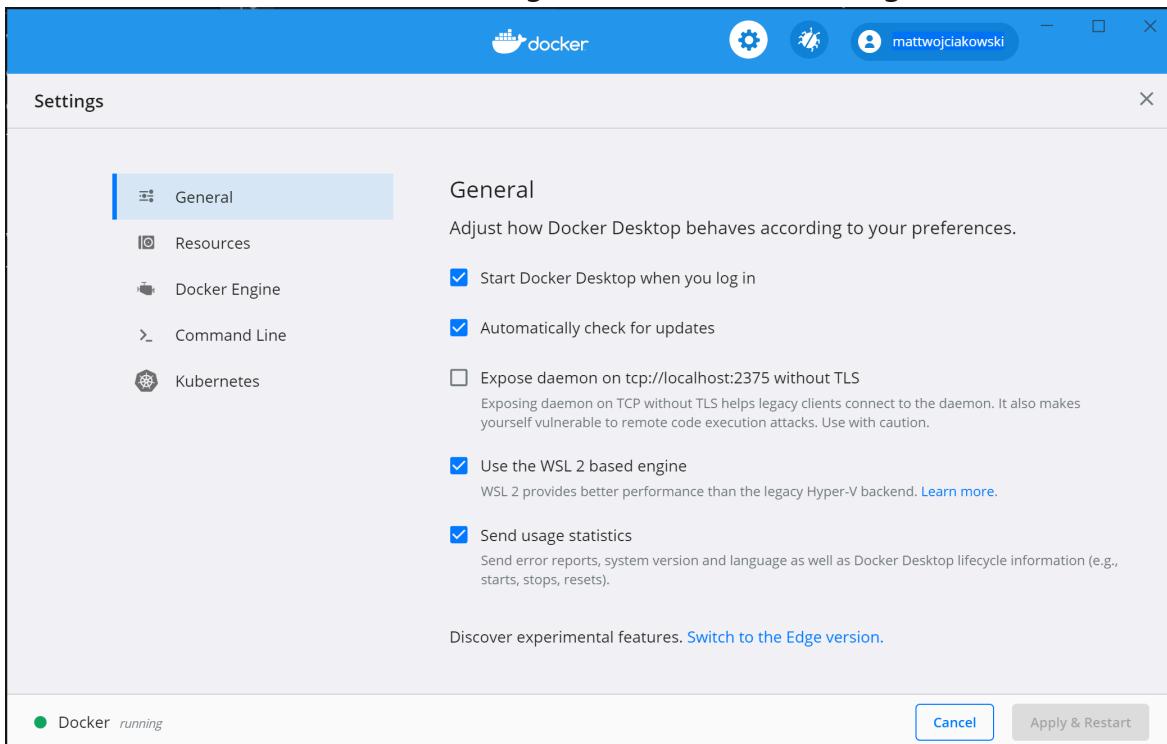
With the WSL 2 backend supported in Docker Desktop for Windows, you can work in a Linux-based development environment and build Linux-based containers, while using Visual Studio Code for code editing and debugging, and running your container in the Microsoft Edge browser on Windows.

To install Docker (after already [installing WSL](#)):

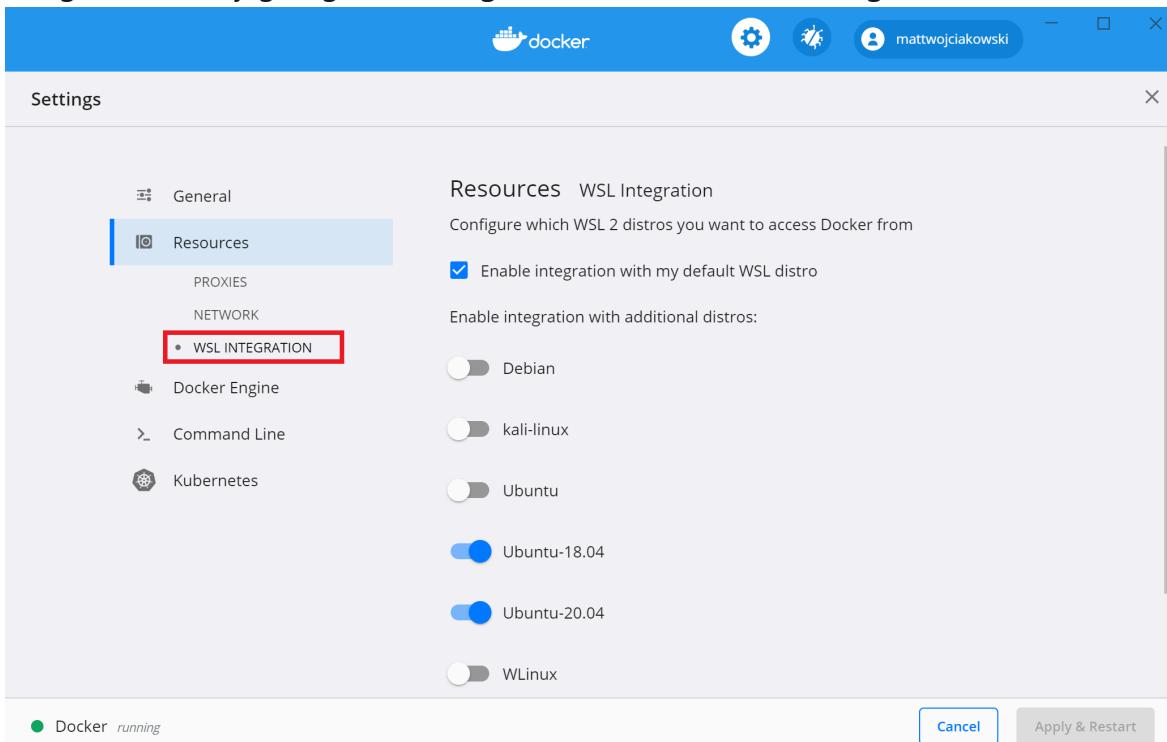
1. Download [Docker Desktop](#) and follow the installation instructions.
2. Once installed, start Docker Desktop from the Windows Start menu, then select the Docker icon from the hidden icons menu of your taskbar. Right-click the icon to display the Docker commands menu and select "Settings".



3. Ensure that "Use the WSL 2 based engine" is checked in **Settings > General**.



4. Select from your installed WSL 2 distributions which you want to enable Docker integration on by going to: **Settings > Resources > WSL Integration**.



5. To confirm that Docker has been installed, open a WSL distribution (e.g. Ubuntu) and display the version and build number by entering: `docker --version`

6. Test that your installation works correctly by running a simple built-in Docker image using: `docker run hello-world`

 Tip

Here are a few helpful Docker commands to know:

- List the commands available in the Docker CLI by entering: `docker`
- List information for a specific command with: `docker <COMMAND> --help`
- List the docker images on your machine (which is just the hello-world image at this point), with: `docker image ls --all`
- List the containers on your machine, with: `docker container ls --all` or `docker ps -a` (without the `-a` show all flag, only running containers will be displayed)
- List system-wide information regarding the Docker installation, including statistics and resources (CPU & memory) available to you in the WSL 2 context, with: `docker info`

Develop in remote containers using VS Code

To get started developing apps using Docker with WSL 2, we recommend using VS Code, along with the WSL, Dev Containers, and Docker extensions.

- [Install the VS Code WSL extension ↗](#). This extension enables you to open your Linux project running on WSL in VS Code (no need to worry about pathing issues, binary compatibility, or other cross-OS challenges).
- [Install the VS Code Dev Containers extension ↗](#). This extension enables you to open your project folder or repo inside of a container, taking advantage of Visual Studio Code's full feature set to do your development work within the container.
- [Install the VS Code Docker extension ↗](#). This extension adds the functionality to build, manage, and deploy containerized applications from inside VS Code. (You need the Dev Containers extension to actually use the container as your dev environment.)

Let's use Docker to create a development container for an existing app project.

1. For this example, I'll use the source code from my [Hello World tutorial for Django](#) in the Python development environment set up docs. You can skip this step if you prefer to use your own project source code. To download my HelloWorld-Django web app from GitHub, open a WSL terminal (Ubuntu for example) and enter: `git clone https://github.com/mattwojo/helloworld-django.git`

 Note

Always store your code in the same file system that you're using tools in. This will result in faster file access performance. In this example, we are using a Linux distro (Ubuntu) and want to store our project files on the WSL file system `\ws1\`. Storing project files on the Windows file system would significantly slow things down when using Linux tools in WSL to access those files.

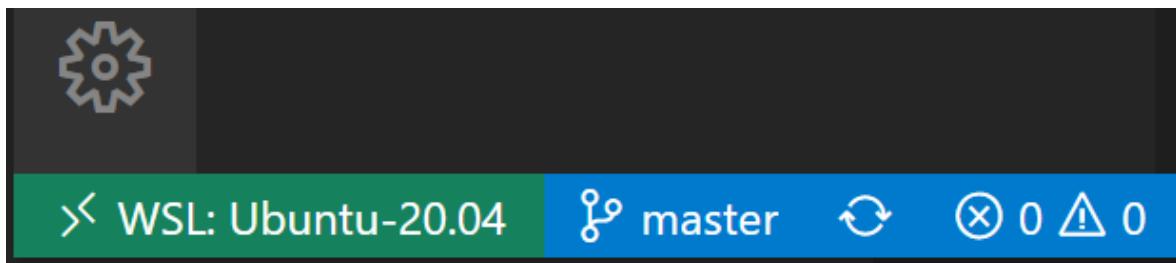
- From your WSL terminal, change directories to the source code folder for this project:

```
Bash  
cd helloworld-django
```

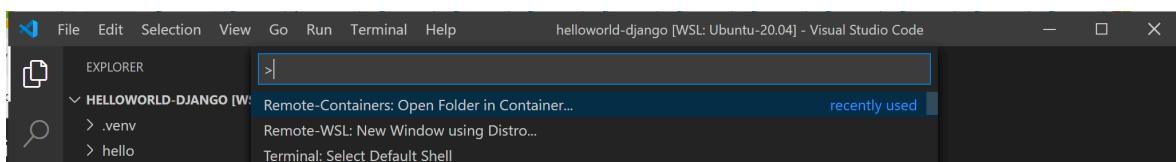
- Open the project in VS Code running on the local WSL extension server by entering:

```
Bash  
code .
```

Confirm that you are connected to your WSL Linux distro by checking the green remote indicator in the bottom-left corner of your VS Code instance.

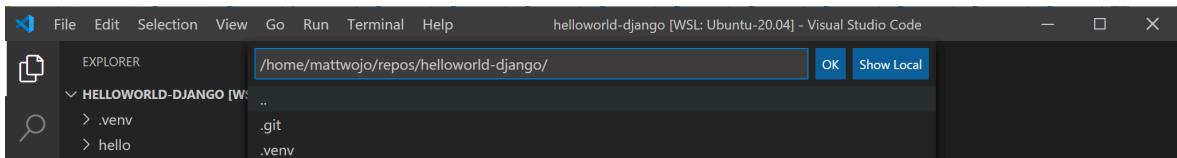


- From the VS Code command palette (Ctrl + Shift + P), enter: **Dev Containers: Reopen in Container** as we are using a folder already opened using the WSL extension. Alternatively, use **Dev Containers: Open Folder in Container...** to choose a WSL folder using the local `\ws1\$` share (from the Windows side). See the Visual Studio Code [Quick start: Open an existing folder in a container](#) for more details. If these commands don't display as you begin to type, check to ensure that you've installed the Dev Containers extension linked above.

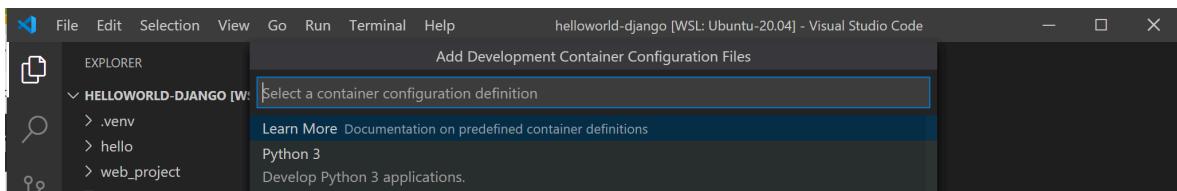


5. Select the project folder that you wish to containerize. In my case, this is

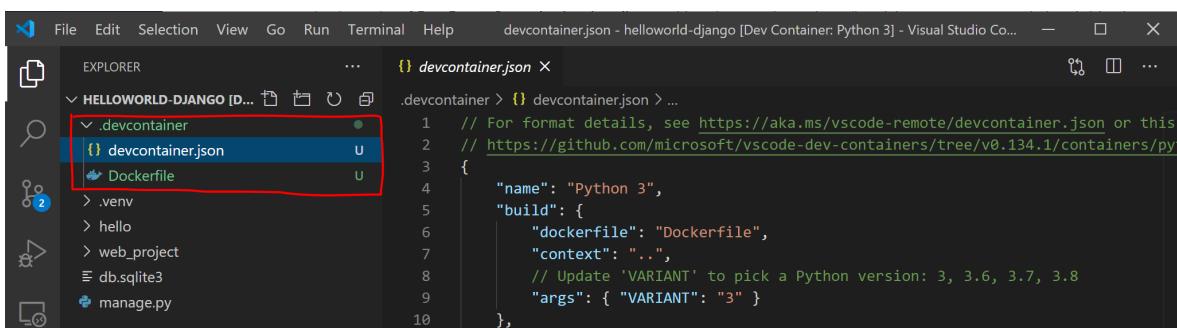
\ws1\Ubuntu-20.04\home\mattwojo\repos\helloworld-django\



6. A list of container definitions will appear, since there is no dev container configuration in the project folder (repo) yet. The list of container configuration definitions that appears is filtered based on your project type. For my Django project, I'll select Python 3.

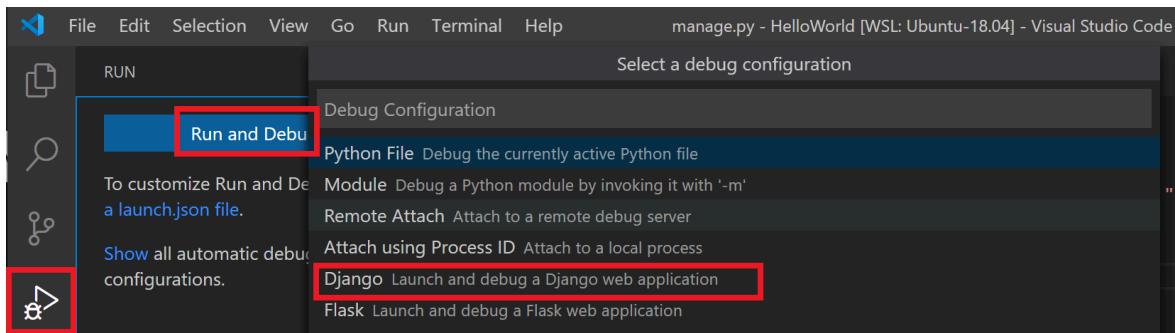


7. A new instance of VS Code will open, begin building our new image, and once the build completed, will start our container. You will see that a new `.devcontainer` folder has appeared with container configuration information inside a `Dockerfile` and `devcontainer.json` file.

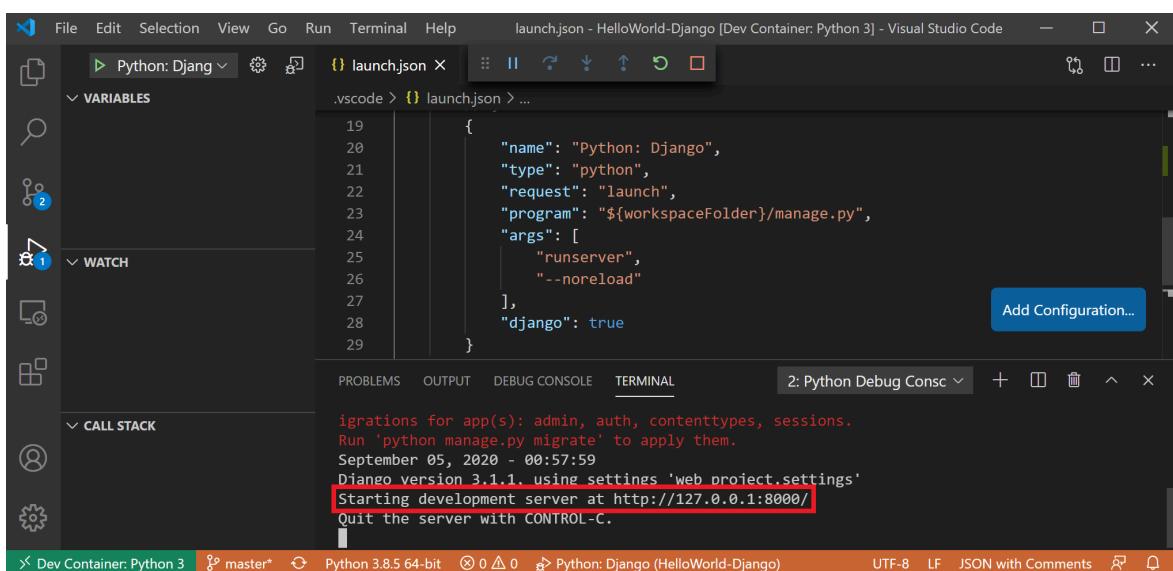


8. To confirm that your project is still connected to both WSL and within a container, open the VS Code integrated terminal (Ctrl + Shift + ~). Check the operating system by entering: `uname` and the Python version with: `python3 --version`. You can see that the `uname` came back as "Linux", so you are still connected to the WSL 2 engine, and Python version number will be based on the container config that may differ from the Python version installed on your WSL distribution.

9. To run and debug your app inside of the container using Visual Studio Code, first open the **Run** menu (Ctrl+Shift+D or select the tab on the far left menu bar). Then select **Run and Debug** to select a debug configuration and choose the configuration that best suits your project (in my example, this will be "Django"). This will create a `launch.json` file in the `.vscode` folder of your project with instructions on how to run your app.



10. From inside VS Code, select **Run > Start debugging** (or just press the **F5** key). This will open a terminal inside VS Code and you should see a result saying something like: "Starting development server at <http://127.0.0.1:8000/>". Quit the server with **CONTROL-C**." Hold down the Control key and select the address displayed to open your app in your default web browser and see your project running inside of its container.



You have now successfully configured a remote development container using Docker Desktop, powered by the WSL 2 backend, that you can code in, build, run, deploy, or debug using VS Code!

Troubleshooting

WSL docker context deprecated

If you were using an early Tech Preview of Docker for WSL, you may have a Docker context called "wsl" that is now deprecated and no longer used. You can check with the command: `docker context ls`. You can remove this "wsl" context to avoid errors with the command: `docker context rm wsl` as you want to use the default context for both Windows and WSL2.

Possible errors you might encounter with this deprecated wsl context include:

```
wsl open //./pipe/docker_wsl: The system cannot find the file specified. or error during connect: Get http://%2F%2F.%2Fpipe%2Fdocker_wsl/v1.40/images/json?all=1: open //./pipe/docker_wsl: The system cannot find the file specified.
```

For more on this issue, see [How to set up Docker within Windows System for Linux \(WSL2\) on Windows 10](#).

Trouble finding docker image storage folder

Docker creates two distro folders to store data:

- \wsl\$\docker-desktop
- \wsl\$\docker-desktop-data

You can find these folders by opening your WSL Linux distribution and entering:
`explorer.exe .` to view the folder in Windows File Explorer. Enter: `\wsl\<distro name>\mnt\wsl` replacing `<distro name>` with the name of your distribution (ie. Ubuntu-20.04) to see these folders.

Find more on locating docker storage locations in WSL, see this [issue from the WSL repo](#) or this [StackOverflow post](#).

For more help with general troubleshooting issues in WSL, see the [Troubleshooting doc](#).

Additional resources

- [Docker docs: Best practices for Docker Desktop with WSL 2](#)
- [Feedback for Docker Desktop for Windows: File an issue](#)
- [VS Code Blog: Guidelines for choosing a development environment](#)
- [VS Code Blog: Using Docker in WSL 2](#)
- [VS Code Blog: Using Remote Containers in WSL 2](#)
- [Hanselminutes Podcast: Making Docker lovely for Developers with Simon Ferquel](#)

 Collaborate with us on
GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

more information, see [our contributor guide](#).

 [Open a documentation issue](#)

 [Provide product feedback](#)

Walkthrough: Build and debug C++ with WSL 2 and Visual Studio 2022

Article • 03/20/2024

Visual Studio 2022 introduces a native C++ toolset for Windows Subsystem for Linux version 2 (WSL 2) development. This toolset is available now in [Visual Studio 2022 version 17.0](#) or higher.

WSL 2 is the new, recommended version of the [Windows Subsystem for Linux](#) (WSL). It provides better Linux file system performance, GUI support, and full system call compatibility. Visual Studio's WSL 2 toolset allows you to use Visual Studio to build and debug C++ code on WSL 2 distros without adding an SSH connection. You can already build and debug C++ code on WSL 1 distros using the native [WSL 1 toolset](#) introduced in Visual Studio 2019 version 16.1.

Visual Studio's WSL 2 toolset supports both CMake and MSBuild-based Linux projects. CMake is our recommendation for all C++ cross-platform development with Visual Studio. We recommend CMake because it build and debug the same project on Windows, WSL, and remote systems.

For a video presentation of the information in this topic, see [Video: Debug C++ with WSL 2 Distributions and Visual Studio 2022](#).

WSL 2 toolset background

C++ cross-platform support in Visual Studio assumes all source files originate in the Windows file system. When targeting a WSL 2 distro, Visual Studio executes a local `rsync` command to copy files from the Windows file system to the WSL file system. The local `rsync` copy doesn't require any user intervention. It occurs automatically when Visual Studio detects you're using a WSL 2 distro. To learn more about the differences between WSL 1 and WSL 2, see [Comparing WSL 1 and WSL 2](#).

CMake Presets integration in Visual Studio supports the WSL 2 toolset. To learn more, see [CMake Presets integration in Visual Studio and Visual Studio Code](#) and [Configure and build with CMake Presets in Visual Studio](#). There's also more advanced information in this article under [Advanced WSL 2 and CMake projects considerations](#).

Install the build tools

Install the tools necessary to build and debug on WSL 2. You'll install a recent version of CMake using Visual Studio's CMake binary deployment in a later step.

1. Install WSL and a WSL 2 distro by following the instructions at [Install WSL](#).
2. Assuming your distro uses `apt` (this walkthrough uses Ubuntu), use the following commands to install the required build tools on your WSL 2 distro:

```
Bash
```

```
sudo apt update  
sudo apt install g++ gdb make ninja-build rsync zip
```

The `apt` commands above install:

- A C++ compiler
- `gdb`
- `CMake`
- `rsync`
- `zip`
- An underlying build system generator

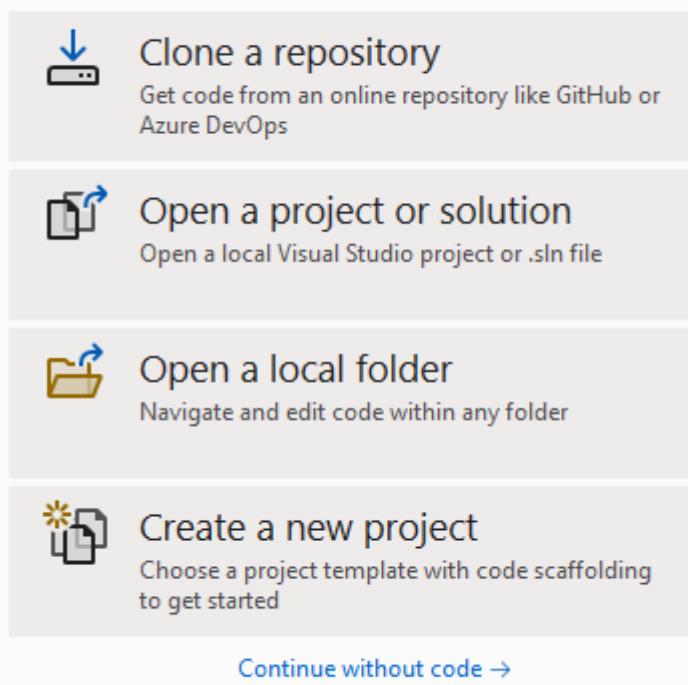
Cross-platform CMake development with a WSL 2 distro

This walkthrough uses GCC and Ninja on Ubuntu. And Visual Studio 2022 version 17.0 Preview 2 or later.

Visual Studio defines a CMake project as a folder with a `CMakeLists.txt` file at the project root. In this walkthrough, you create a new CMake project by using the Visual Studio **CMake Project** template:

3. From the Visual Studio **Get started** screen, select **Create a new project**.

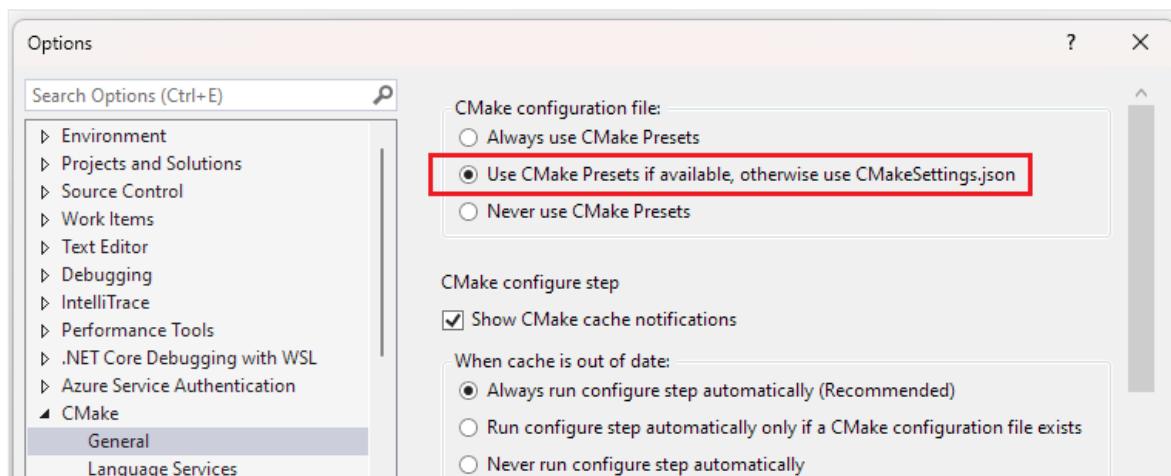
Get started



The available options are:

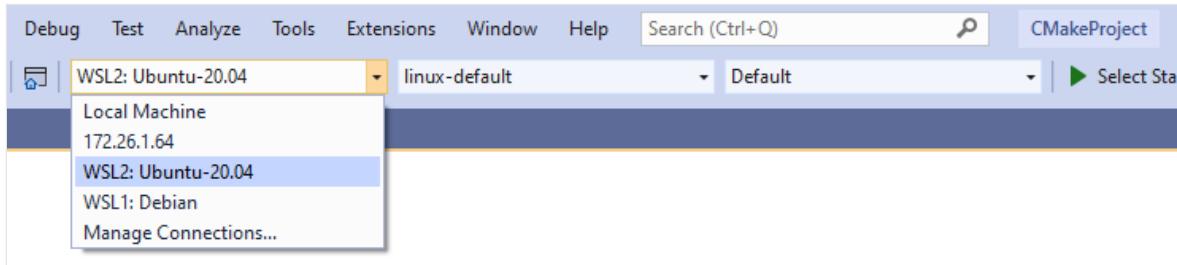
Clone a repository, Open a project or solution, Open a local folder, Create a new project, or Continue without code.":::

4. In the **Search for templates** textbox, type "cmake". Choose the **CMake Project** type and select **Next**. Give the project a name and location, and then select **Create**.
5. Enable Visual Studio's CMake Presets integration. Select **Tools > Options > CMake > General**. Select **Prefer using CMake Presets for configure, build, and test**, then select **OK**. Instead, you could have added a `CMakePresets.json` file to the root of the project. For more information, see [Enable CMake Presets integration](#).



6. To activate the integration: from the main menu, select **File > Close Folder**. The **Get started** page appears. Under **Open recent**, select the folder you just closed to reopen the folder.

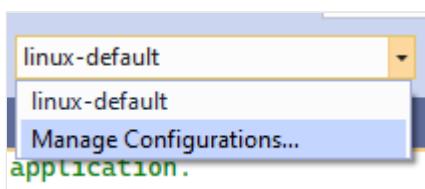
7. There are three dropdowns across the Visual Studio main menu bar. Use the dropdown on the left to select your active target system. This is the system where CMake is invoked to configure and build the project. Visual Studio queries for WSL installations with `wsl -l -v`. In the following image, **WSL2: Ubuntu-20.04** is shown selected as the **Target System**.



ⓘ Note

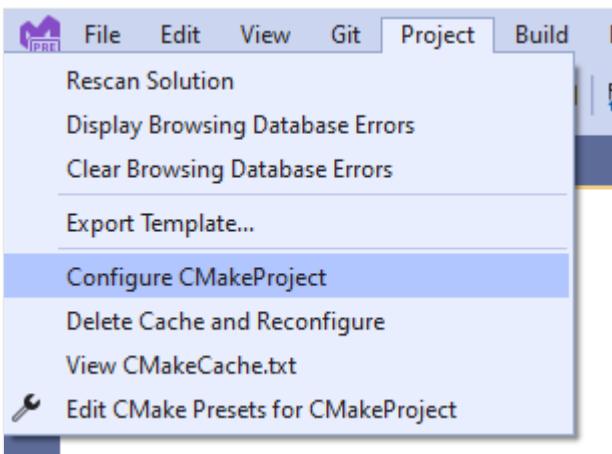
If Visual Studio starts to configure your project automatically, read step 11 to manage CMake binary deployment, and then continue to the step below. To customize this behavior, see [Modify automatic configuration and cache notifications](#).

8. Use the dropdown in the middle to select your active Configure Preset. Configure Presets tell Visual Studio how to invoke CMake and generate the underlying build system. In step 7, the active Configure Preset is the **linux-default** Preset created by Visual Studio. To create a custom Configure Preset, select **Manage Configurations...** For more information about Configure Presets, see [Select a Configure Preset](#) and [Edit Presets](#).

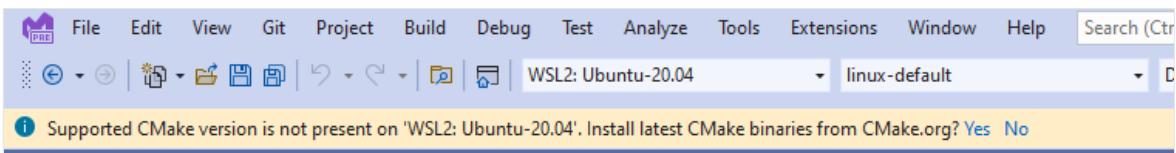


9. Use the dropdown on the right to select your active Build Preset. Build Presets tell Visual Studio how to invoke build. In the illustration for step 7, the active Build Preset is the **Default** Build Preset created by Visual Studio. For more information about Build Presets, see [Select a Build Preset](#).

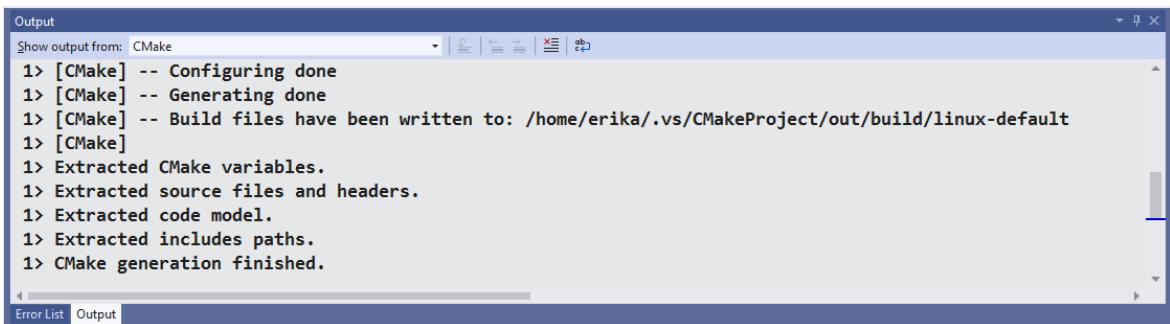
10. Configure the project on WSL 2. If project generation doesn't start automatically, then manually invoke configure with **Project > Configure project-name**



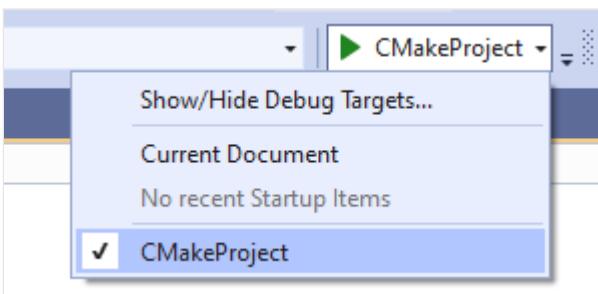
11. If you don't have a supported version of CMake installed on your WSL 2 distro, then Visual Studio prompts you beneath the main menu ribbon to deploy a recent version of CMake. Select **Yes** to deploy CMake binaries to your WSL 2 distro.



12. Confirm that the configure step completed and that you can see the **CMake generation finished** message in the **Output** window under the **CMake** pane. Build files are written to a directory in the WSL 2 distro's file system.

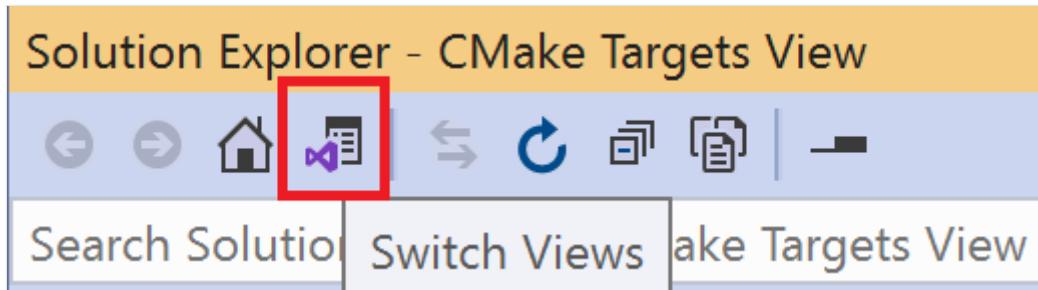


13. Select the active debug target. The debug dropdown menu lists all the CMake targets available to the project.

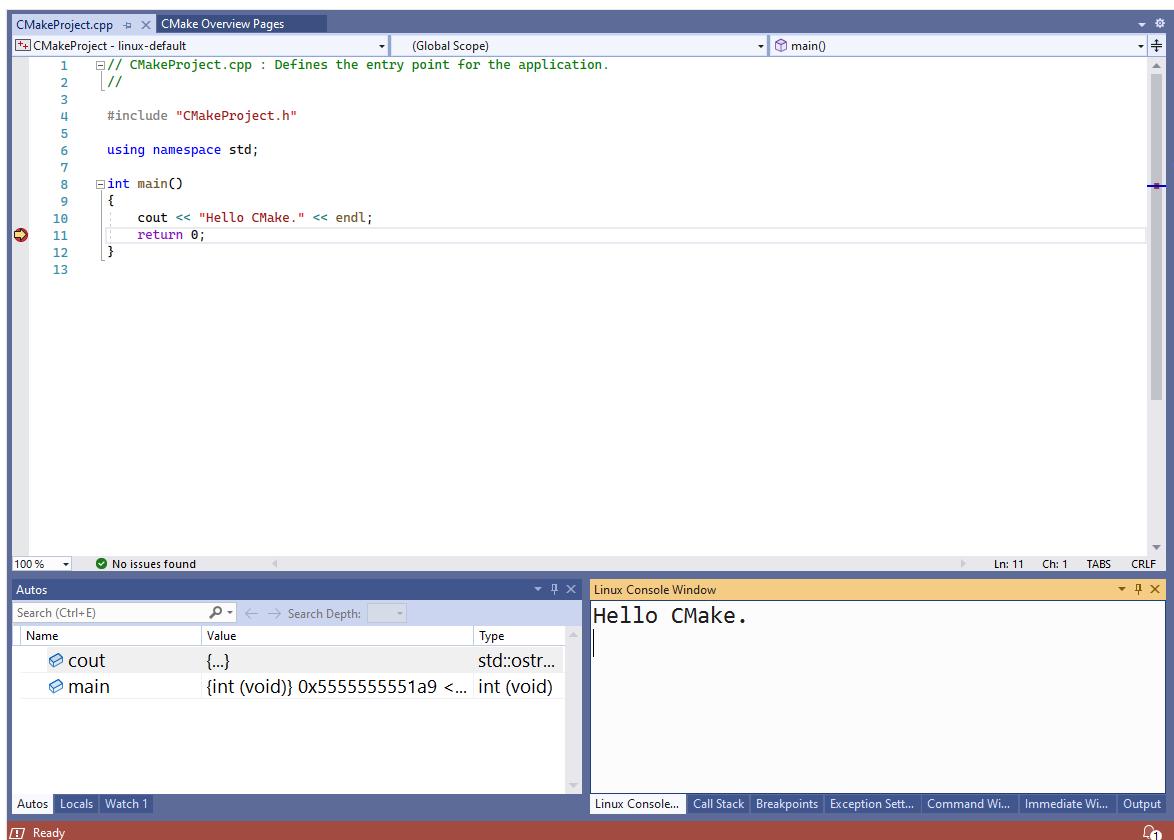


14. Expand the project subfolder in the **Solution Explorer**. In the `CMakeProject.cpp` file, set a breakpoint in `main()`. You can also navigate to CMake targets view by

selecting the View Picker button in the **Solution Explorer**, highlighted in following screenshot:



15. Select **Debug > Start**, or press **F5**. Your project builds, the executable launches on your WSL 2 distro, and Visual Studio halts execution at the breakpoint. The output of your program (in this case, "Hello CMake.") is visible in the Linux Console Window:



You've now built and debugged a C++ app with WSL 2 and Visual Studio 2022.

Advanced WSL 2 and CMake projects considerations

Visual Studio only provides native support for WSL 2 for CMake projects that use `CMakePresets.json` as the active configuration file. To migrate from `CMakeSettings.json` to `CMakePresets.json`, see [Enable CMake Presets integration in Visual Studio](#).

If you're targeting a WSL 2 distribution and you don't want to use the WSL 2 toolset, then in the Visual Studio Remote Settings vendor map in `CMakePresets.json`, set `forceWSL1Toolset` to `true`. For more information, see [Visual Studio Remote Settings vendor map](#).

If `forceWSL1Toolset` is set to `true`, then Visual Studio doesn't maintain a copy of your source files in the WSL file system. Instead, it accesses source files in the mounted Windows drive (`/mnt/` ...).

In most cases, it's best to use the WSL 2 toolset with WSL 2 distributions because WSL 2 is slower when project files are instead stored in the Windows file system. To learn more about file system performance in WSL 2, see [Comparing WSL 1 and WSL 2](#).

Specify advanced settings such as the path to the directory on WSL 2 where the project is copied, copy source options, and rsync command arguments, in the Visual Studio Remote Settings vendor map in `CMakePresets.json`. For more information, see [Visual Studio Remote Settings vendor map](#).

System headers are still automatically copied to the Windows file system to supply the native IntelliSense experience. You can customize the headers that are included or excluded from this copy in the Visual Studio Remote Settings vendor map in `CMakePresets.json`.

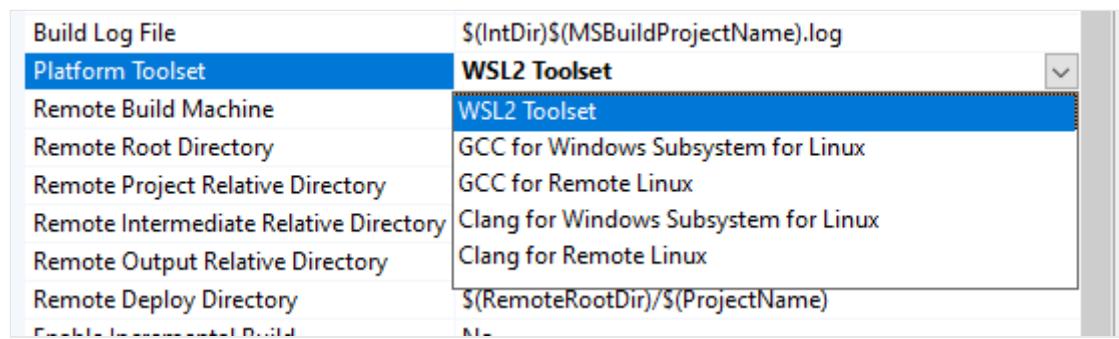
You can change the IntelliSense mode, or specify other IntelliSense options, in the Visual Studio Settings vendor map in `CMakePresets.json`. For details about the vendor map, see [Visual Studio Remote Settings vendor map](#).

WSL 2 and MSBuild-based Linux projects

CMake is recommended for all C++ cross-platform development with Visual Studio because it allows you to build and debug the same project on Windows, WSL, and remote systems.

But you may have a MSBuild-based Linux project.

If you have a MSBuild-based Linux project, then you can upgrade to the WSL 2 toolset in Visual Studio. Right-click the project in the solution explorer, then choose **Properties** > **General** > **Platform Toolset**:



If you're targeting a WSL 2 distribution and you don't want to use the WSL 2 toolset, then in the **Platform Toolset** dropdown, select the **GCC for Windows Subsystem for Linux** or **Clang for Windows Subsystem for Linux** toolset. If either of these toolsets are selected, Visual Studio doesn't maintain a copy of your source files in the WSL file system and instead accesses source files over the mounted Windows drive (`/mnt/...`). System headers are still automatically copied to the Windows file system to provide a native IntelliSense experience. Customize the headers that are included or excluded from this copy in **Property Pages > General**.

In most cases, it's best to use the WSL 2 toolset with WSL 2 distributions because WSL 2 is slower when project files are stored in the Windows file system. To learn more, see [Comparing WSL 1 and WSL 2](#).

See also

[Video: Debug C++ with WSL 2 Distributions and Visual Studio 2022](#) ↗

[Download Visual Studio 2022](#) ↗

[Create a CMake Linux project in Visual Studio](#)

[Tutorial: Debug a CMake project on a remote Windows machine](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback](#) ↗ | [Get help at Microsoft Q&A](#)

Get started with GPU acceleration for ML in WSL

Article • 03/20/2023

Machine learning (ML) is becoming a key part of many development workflows. Whether you're a data scientist, ML engineer, or starting your learning journey with ML the Windows Subsystem for Linux (WSL) offers a great environment to run the most common and popular GPU accelerated ML tools.

There are lots of different ways to set up these tools. For example, [NVIDIA CUDA in WSL](#), [TensorFlow-DirectML](#) and [PyTorch-DirectML](#) all offer different ways you can use your GPU for ML with WSL. To learn more about the reasons for choosing one versus another, see [GPU accelerated ML training](#).

This guide will show how to set up:

- NVIDIA CUDA if you have an NVIDIA graphics card and run a sample ML framework container
- TensorFlow-DirectML and PyTorch-DirectML on your AMD, Intel, or NVIDIA graphics card

Prerequisites

- Ensure you are running [Windows 11](#) or [Windows 10, version 21H2](#) or higher.
- [Install WSL and set up a username and password for your Linux distribution](#).

Setting up NVIDIA CUDA with Docker

1. [Download and install the latest driver for your NVIDIA GPU](#)
2. Install [Docker Desktop](#) or install the Docker engine directly in WSL by running the following command

Bash

```
curl https://get.docker.com | sh
```

Bash

```
sudo service docker start
```

3. If you installed the Docker engine directly then [install the NVIDIA Container Toolkit](#) following the steps below.

Set up the stable repository for the NVIDIA Container Toolkit by running the following commands:

Bash

```
distribution=$( . /etc/os-release; echo $ID$VERSION_ID )
```

Bash

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-docker-keyring.gpg
```

Bash

```
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-docker-keyring.gpg] https://#g' | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

Install the NVIDIA runtime packages and dependencies by running the commands:

Bash

```
sudo apt-get update
```

Bash

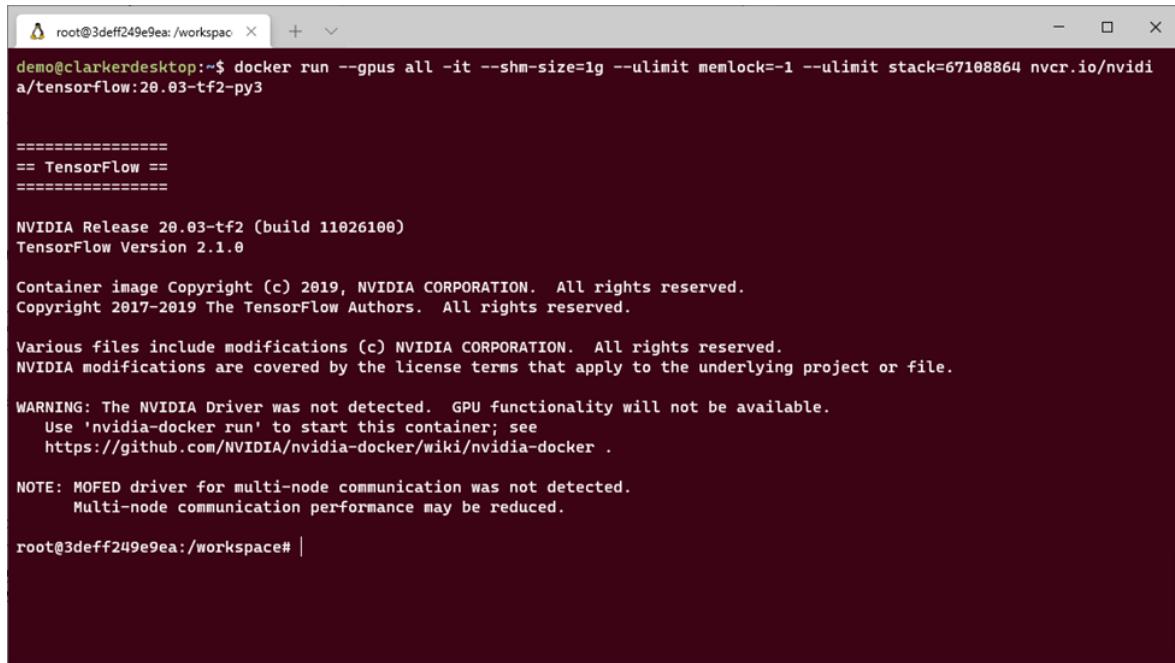
```
sudo apt-get install -y nvidia-docker2
```

4. Run a machine learning framework container and sample.

To run a machine learning framework container and start using your GPU with this NVIDIA NGC TensorFlow container, enter the command:

Bash

```
docker run --gpus all -it --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 nvcr.io/nvidia/tensorflow:20.03-tf2-py3
```



```
root@3deff249e9ea:/workspace ~ + - X
demo@clarkerdesktop:~$ docker run --gpus all -it --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 nvcr.io/nvidia/tensorflow:20.03-tf2-py3

=====
== TensorFlow ==
=====

NVIDIA Release 20.03-tf2 (build 11026100)
TensorFlow Version 2.1.0

Container image Copyright (c) 2019, NVIDIA CORPORATION. All rights reserved.
Copyright 2017-2019 The TensorFlow Authors. All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

WARNING: The NVIDIA Driver was not detected. GPU functionality will not be available.
Use 'nvidia-docker run' to start this container; see
https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker .

NOTE: MOFED driver for multi-node communication was not detected.
Multi-node communication performance may be reduced.

root@3deff249e9ea:/workspace# |
```

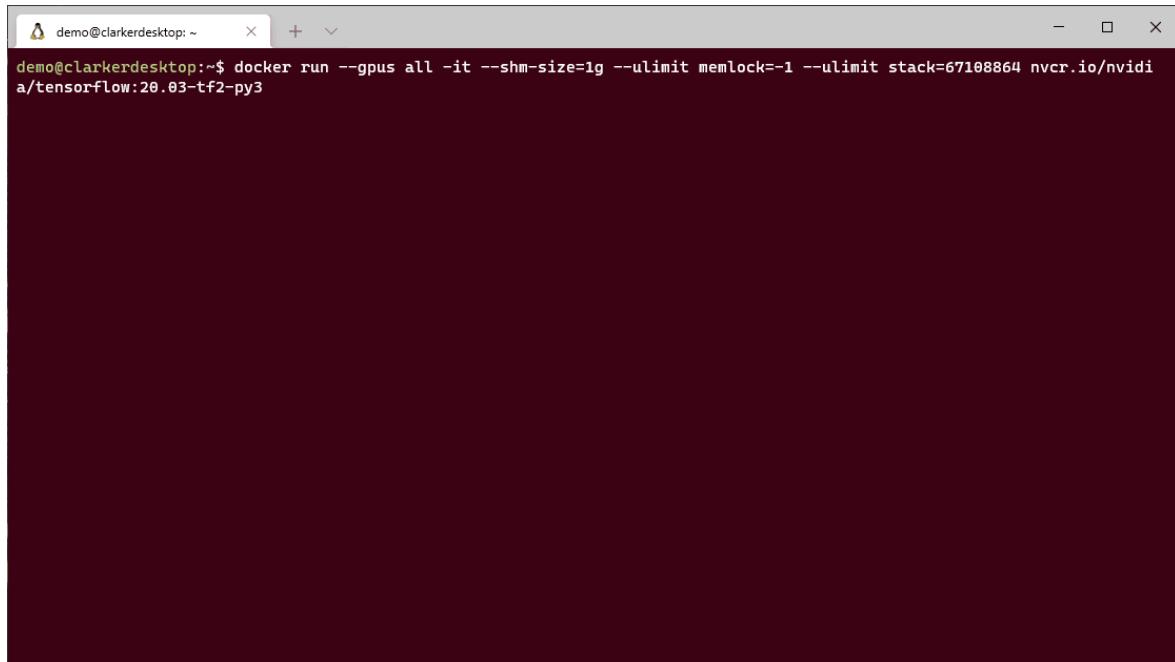
You can run a pre-trained model sample that is built into this container by running the commands:

Bash

```
cd nvidia-examples/cnn/
```

Bash

```
python resnet.py --batch_size=64
```



```
demo@clarkerdesktop:~$ docker run --gpus all -it --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 nvcr.io/nvidia/tensorflow:20.03-tf2-py3
```



Additional ways to get setup and utilize NVIDIA CUDA can be found in the [NVIDIA CUDA on WSL User Guide](#).

Setting up TensorFlow-DirectML or PyTorch-DirectML

1. Download and install the latest driver from your GPU vendors website: [AMD](#), [Intel](#), or [NVIDIA](#).
2. Setup a Python environment.

We recommend setting up a virtual Python environment. There are many tools you can use to setup a virtual Python environment — for these instructions, we'll use [Anaconda's Miniconda](#).

Bash

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Bash

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Bash

```
conda create --name directml python=3.7 -y
```

Bash

```
conda activate directml
```

3. Install the machine learning framework backed by DirectML of your choice.

TensorFlow-DirectML:

Bash

```
pip install tensorflow-directml
```

PyTorch-DirectML:

```
Bash
```

```
sudo apt install libblas3 libomp5 liblapack3
```

```
Bash
```

```
pip install pytorch-directml
```

4. Run a quick addition sample in an interactive Python session for [TensorFlow-DirectML](#) or [PyTorch-DirectML](#) to make sure everything is working.

If you have questions or run into issues, visit the [DirectML repo on GitHub ↗](#).

Multiple GPUs

If you have multiple GPUs on your machine you can also access them inside of WSL. However, you will only be able to access one at a time. To choose a specific GPU please set the environment variable below to the name of your GPU as it appears in device manager:

```
Bash
```

```
export MESA_D3D12_DEFAULT_ADAPTER_NAME="<NameFromDeviceManager>"
```

This will do a string match, so if you set it to "NVIDIA" it will match the first GPU that starts with "NVIDIA".

Additional Resources

- [Guidance for setting up NVIDIA CUDA in WSL ↗](#)
- [Guidance for setting up TensorFlow with DirectML in WSL](#)
- [TensorFlow with DirectML Samples ↗](#)
- [Guidance for setting up PyTorch with DirectML in WSL](#)
- [PyTorch with DirectML Samples ↗](#)

Run Linux GUI apps on the Windows Subsystem for Linux

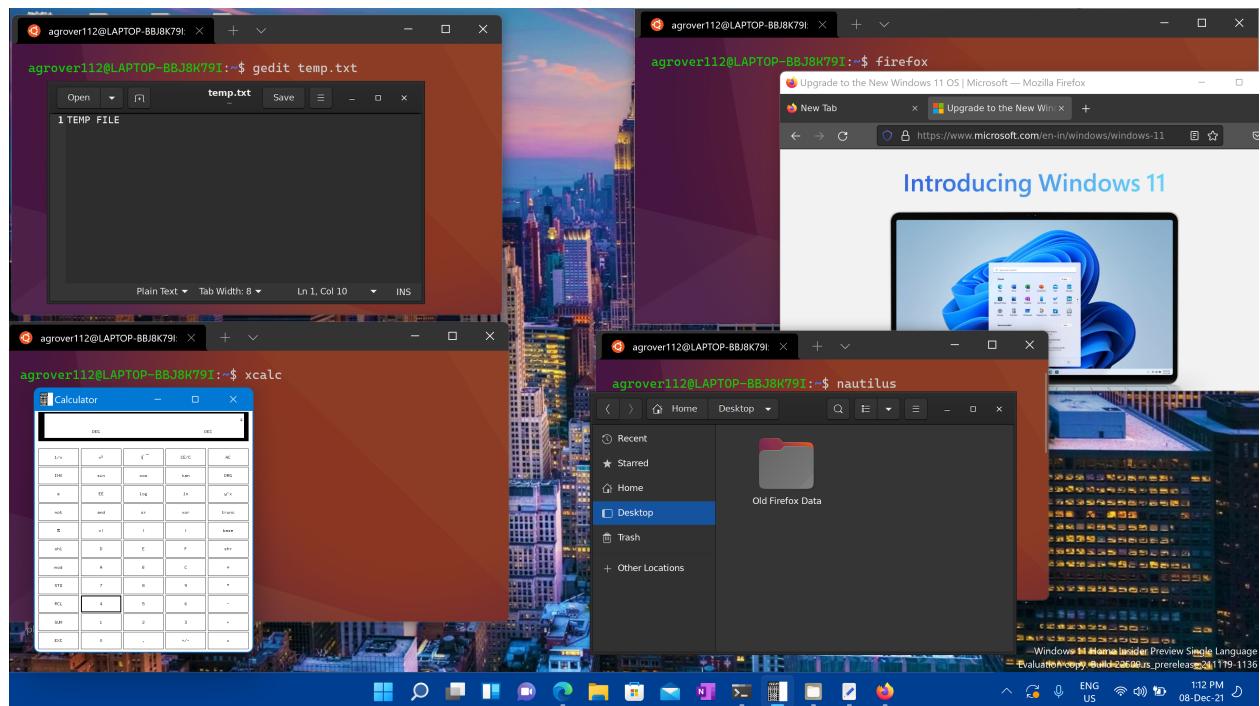
Article • 01/10/2024

Windows Subsystem for Linux (WSL) now supports running Linux GUI applications (X11 and Wayland) on Windows in a fully integrated desktop experience.

WSL 2 enables Linux GUI applications to feel native and natural to use on Windows.

- Launch Linux apps from the Windows Start menu
- Pin Linux apps to the Windows task bar
- Use alt-tab to switch between Linux and Windows apps
- Cut + Paste across Windows and Linux apps

You can now integrate both Windows and Linux applications into your workflow for a seamless desktop experience.



Install support for Linux GUI apps

Prerequisites

- You will need to be on Windows 10 Build 19044+ or Windows 11 to access this feature.
- Installed driver for vGPU

To run Linux GUI apps, you should first install the driver matching your system below. This will enable you to use a virtual GPU (vGPU) so you can benefit from hardware accelerated OpenGL rendering.

- [Intel GPU driver ↗](#)
- [AMD GPU driver ↗](#)
- [NVIDIA GPU driver ↗](#)

Fresh install - No prior WSL installation

You can now install everything you need to run Windows Subsystem for Linux (WSL) by entering this command in an administrator PowerShell or Windows Command Prompt and then restarting your machine.

```
PowerShell
```

```
wsl --install
```

Once your machine has finished rebooting, installation will continue and you will be asked to enter a username and password. This will be your Linux credential for the Ubuntu distribution.

You're now ready to begin using Linux GUI apps on WSL!

For more info check [install WSL](#).

Existing WSL install

If you already have WSL installed on your machine, you can update to the latest version that includes Linux GUI support by running the update command from an elevated command prompt.

1. Select **Start**, type **PowerShell**, right-click **Windows PowerShell**, and then select **Run as administrator**.
2. Enter the WSL update command:

```
PowerShell
```

```
wsl --update
```

3. You will need to restart WSL for the update to take effect. You can restart WSL by running the shutdown command in PowerShell.

```
PowerShell
```

```
wsl --shutdown
```

ⓘ Note

Linux GUI apps are only supported with WSL 2 and will not work with a Linux distribution configured for WSL 1. Read about [how to change your distribution from WSL 1 to WSL 2](#).

Run Linux GUI apps

You can run the following commands from your Linux terminal to download and install these popular Linux applications. If you are using a different distribution than Ubuntu, it may use a different package manager than apt. Once the Linux application is installed, you can find it in your **Start** menu under the distribution name. For example: `Ubuntu -> Microsoft Edge`.

ⓘ Note

Support for GUI apps on WSL does not provide a full desktop experience. It relies on Windows desktop, so installing desktop-focused tools or apps may not be supported. To request additional support, you can file an issue in the [WSLg repo on GitHub](#).

Update the packages in your distribution

```
Bash
```

```
sudo apt update
```

Install Gnome Text Editor

Gnome Text Editor is the default text editor of the GNOME desktop environment.

```
Bash
```

```
sudo apt install gnome-text-editor -y
```

To launch your bashrc file in the editor, enter: `gnome-text-editor ~/bashrc`

ⓘ Note

GNOME Text Editor  replaces gedit as GNOME/Ubuntu's default text editor in Ubuntu 22.10. If you're running an older version of Ubuntu and want to use **gedit** , the previous default text editor, use `sudo apt install gedit -y`.

Install GIMP

GIMP is a free and open-source raster graphics editor used for image manipulation and image editing, free-form drawing, transcoding between different image file formats, and more specialized tasks.

Bash

```
sudo apt install gimp -y
```

To launch, enter: `gimp`

Install Nautilus

Nautilus, also known as GNOME Files, is the file manager for the GNOME desktop. (Similar to Windows File Explorer).

Bash

```
sudo apt install nautilus -y
```

To launch, enter: `nautilus`

Install VLC

VLC is a free and open source cross-platform multimedia player and framework that plays most multimedia files.

Bash

```
sudo apt install vlc -y
```

To launch, enter: `vlc`

Install X11 apps

X11 is the Linux windowing system and this is a miscellaneous collection of apps and tools that ship with it, such as the xclock, xcalc calculator, xclipboard for cut and paste, xev for event testing, etc. See the [x.org docs](#) for more info.

Bash

```
sudo apt install x11-apps -y
```

To launch, enter the name of the tool you would like to use. For example:

- `xcalc`, `xclock`, `xeyes`

Install Google Chrome for Linux

To install the Google Chrome for Linux:

1. Change directories into the temp folder: `cd /tmp`
2. Use wget to download it: `wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb`
3. Install the package: `sudo apt install --fix-missing ./google-chrome-stable_current_amd64.deb`

*The `--fix-missing` option is used to fix missing dependencies that may arise during the installation process. The `./` in the command specifies the current directory where the .deb file is located. If the .deb file is located in a different directory, you will need to specify the path to the file in the command.

To launch, enter: `google-chrome`

Install Microsoft Edge browser for Linux

Find information on how to [install the Microsoft Edge browser for Linux using the command line on the Edge Insider site](#). Select **Get instructions** under the Command line installation section of the page.

To launch, enter: `microsoft-edge`

Troubleshooting

If you have any problem starting GUI applications please check this guide first:

[Diagnosing "cannot open display" type issues with WSLg ↗](#)

Install Node.js on Windows Subsystem for Linux (WSL2)

Article • 03/01/2024

If you prefer using Node.js in a Linux environment, find performance speed and system call compatibility important, want to run [Docker containers](#) that leverage Linux workspaces and avoid having to maintain both Linux and Windows build scripts, or just prefer using a Bash command line, then you want to install Node.js on the Windows Subsystem for Linux (more specifically, WSL 2).

Using Windows Subsystem for Linux (WSL), might also enable you to install your preferred Linux distribution (Ubuntu is our default) so that you can have consistency between your development environment (where you write code) and production environment (the server where your code is deployed).

ⓘ Note

If you are new to developing with Node.js and want to get up and running quickly so that you can learn, [install Node.js on Windows](#). This recommendation also applies if you plan to use a Windows Server production environment.

Install WSL 2

WSL 2 is the most recent version available for Windows and we recommend it for professional Node.js development workflows. To enable and install WSL 2, follow the steps in the [WSL install documentation](#). These steps will include choosing a Linux distribution (for example, Ubuntu).

Once you have installed WSL 2 and a Linux distribution, open the Linux distribution (it can be found in your Windows start menu) and check the version and codename using the command: `lsb_release -dc`.

We recommend updating your Linux distribution regularly, including immediately after you install, to ensure you have the most recent packages. Windows doesn't automatically handle this update. To update your distribution, use the command: `sudo apt update && sudo apt upgrade`.

Install Windows Terminal (optional)

Windows Terminal is an improved command line shell that allows you to run multiple tabs so that you can quickly switch between Linux command lines, Windows Command Prompt, PowerShell, Azure CLI, or whatever you prefer to use. You can also create custom key bindings (shortcut keys for opening or closing tabs, copy+paste, etc.), use the search feature, customize your terminal with themes (color schemes, font styles and sizes, background image/blur/transparency), and more. [Learn more in the Windows Terminal docs](#).

[Install Windows Terminal using the Microsoft Store](#): By installing via the store, updates are handled automatically.

Install nvm, node.js, and npm

Besides choosing whether to install on Windows or WSL, there are additional choices to make when installing Node.js. We recommend using a version manager as versions change very quickly. You will likely need to switch between multiple versions of Node.js based on the needs of different projects you're working on. Node Version Manager, more commonly called nvm, is the most popular way to install multiple versions of Node.js. We will walk through the steps to install nvm and then use it to install Node.js and Node Package Manager (npm). There are [alternative version managers](#) to consider as well covered in the next section.

ⓘ Important

It is always recommended to remove any existing installations of Node.js or npm from your operating system before installing a version manager as the different types of installation can lead to strange and confusing conflicts. For example, the version of Node that can be installed with Ubuntu's `apt-get` command is currently outdated. For help with removing previous installations, see [How to remove nodejs from ubuntu](#).)

For the most current information on installing NVM, see [Installing and Updating in the NVM repo on GitHub](#).

1. Open your Ubuntu command line (or distribution of your choice).
2. Install cURL (a tool used for downloading content from the internet in the command-line) with: `sudo apt-get install curl`
3. Install nvm, with: `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/master/install.sh | bash`

(!) Note

Installing a newer version of NVM using cURL will replace the older one, leaving the version of Node you've used NVM to install intact. For more information, see the [GitHub project page for the latest release information on NVM ↗](#).

4. To verify installation, enter: `command -v nvm` ...this should return 'nvm', if you receive 'command not found' or no response at all, close your current terminal, reopen it, and try again. [Learn more in the nvm github repo ↗](#).

5. List which versions of Node are currently installed (should be none at this point):

```
nvm ls
```

```
mattwojo@MININT-LOBGCR8:~$ nvm --version
0.34.0
mattwojo@MININT-LOBGCR8:~$ nvm ls

->      system
iojs -> N/A (default)
node -> stable (-> N/A) (default)
unstable -> N/A (default)
lts/* -> lts/dubnium (-> N/A)
lts/argon -> v4.9.1 (-> N/A)
lts/boron -> v6.17.1 (-> N/A)
lts/carbon -> v8.16.1 (-> N/A)
lts/dubnium -> v10.16.3 (-> N/A)
mattwojo@MININT-LOBGCR8:~$ nvm use node
N/A: version "node -> N/A" is not yet installed.
```

6. Install both the current and stable LTS versions of Node.js. In a later step, you'll learn how to switch between active versions of Node.js with an `nvm` command.

- Install the current stable LTS release of Node.js (recommended for production applications): `nvm install --lts`
- Install the current release of Node.js (for testing latest Node.js features and improvements, but more likely to have issues): `nvm install node`

7. List what versions of Node are installed: `nvm ls` ...now you should see the two versions that you just installed listed.

```
mattwojo@MININT-LOBGCR8: ~
mattwojo@MININT-LOBGCR8:~$ nvm ls
->      v10.16.3
      v12.9.0
      system
default -> node  (-> v12.9.0)
node -> stable  (-> v12.9.0) (default)
stable -> 12.9  (-> v12.9.0) (default)
iojs -> N/A (default)
unstable -> N/A (default)
lts/* -> lts/dubnium (-> v10.16.3)
lts/argon -> v4.9.1 (-> N/A)
lts/boron -> v6.17.1 (-> N/A)
lts/carbon -> v8.16.1 (-> N/A)
lts/dubnium -> v10.16.3
mattwojo@MININT-LOBGCR8:~$
```

8. Verify that Node.js is installed and the currently default version with: `node --version`. Then verify that you have npm as well, with: `npm --version` (You can also use `which node` or `which npm` to see the path used for the default versions).

9. To change the version of Node.js you would like to use for a project, create a new project directory `mkdir NodeTest`, and enter the directory `cd NodeTest`, then enter `nvm use node` to switch to the Current version, or `nvm use --lts` to switch to the LTS version. You can also use the specific number for any additional versions you've installed, like `nvm use v8.2.1`. (To list all of the versions of Node.js available, use the command: `nvm ls-remote`).

If you are using NVM to install Node.js and NPM, you should not need to use the SUDO command to install new packages.

Alternative version managers

While nvm is currently the most popular version manager for node, there are a few alternatives to consider:

- [n](#) is a long-standing `nvm` alternative that accomplishes the same thing with slightly different commands and is installed via `npm` rather than a bash script.
- [fpm](#) is a newer version manager, claiming to be much faster than `nvm`. (It also uses [Azure Pipelines](#).)
- [Volta](#) is a new version manager from the LinkedIn team that claims improved speed and cross-platform support.

- [asdf-vm](#) is a single CLI for multiple languages, like ike gvm, nvm, rbenv & pyenv (and more) all in one.
- [nvs](#) (Node Version Switcher) is a cross-platform [nvm](#) alternative with the ability to integrate with VS Code.

Install Visual Studio Code

We recommend using Visual Studio Code with the [Remote-development extension pack](#) for Node.js projects. This splits VS Code into a “client-server” architecture, with the client (the VS Code user interface) running on your Windows operating system and the server (your code, Git, plugins, etc) running “remotely” on your WSL Linux distribution.

ⓘ Note

This “remote” scenario is a bit different than you may be accustomed to. WSL supports an actual Linux distribution where your project code is running, separately from your Windows operating system, but still on your local machine. The Remote-WSL extension connects with your Linux subsystem as if it were a remote server, though it’s not running in the cloud... it’s still running on your local machine in the WSL environment that you enabled to run alongside Windows.

- Linux-based Intellisense and linting is supported.
- Your project will automatically build in Linux.
- You can use all your extensions running on Linux ([ES Lint](#), [NPM Intellisense](#), [ES6 snippets, etc.](#)).

Other code editors, like IntelliJ, Sublime Text, Brackets, etc. will also work with a WSL 2 Node.js development environment, but may not have the same sort of remote features that VS Code offers. These code editors may run into trouble accessing the WSL shared network location (\wsl\$\Ubuntu\home) and will try to build your Linux files using Windows tools, which likely not what you want. The Remote-WSL Extension in VS Code handles this compatibility for you, with other IDEs you may need to set up an X server. [Support for running GUI apps in WSL](#) (like a code editor IDE) is coming soon.

Terminal-based text editors (vim, emacs, nano) are also helpful for making quick changes from right inside your console. The article, [Emacs, Nano, or Vim: Choose your Terminal-Based Text Editor Wisely](#) does a nice job explaining some differences and a bit about how to use each.

To install VS Code and the Remote-WSL Extension:

1. Download and install VS Code for Windows [↗](#). VS Code is also available for Linux, but Windows Subsystem for Linux does not support GUI apps, so we need to install it on Windows. Not to worry, you'll still be able to integrate with your Linux command line and tools using the Remote - WSL Extension.
2. Install the [Remote - WSL Extension ↗](#) on VS Code. This allows you to use WSL as your integrated development environment and will handle compatibility and pathing for you. [Learn more ↗](#).

Important

If you already have VS Code installed, you need to ensure that you have the [1.35 May release ↗](#) or later in order to install the [Remote - WSL Extension ↗](#). We do not recommend using WSL in VS Code without the Remote-WSL extension as you will lose support for auto-complete, debugging, linting, etc. Fun fact: This WSL extension is installed in `$HOME/.vscode-server/extensions`.

Helpful VS Code Extensions

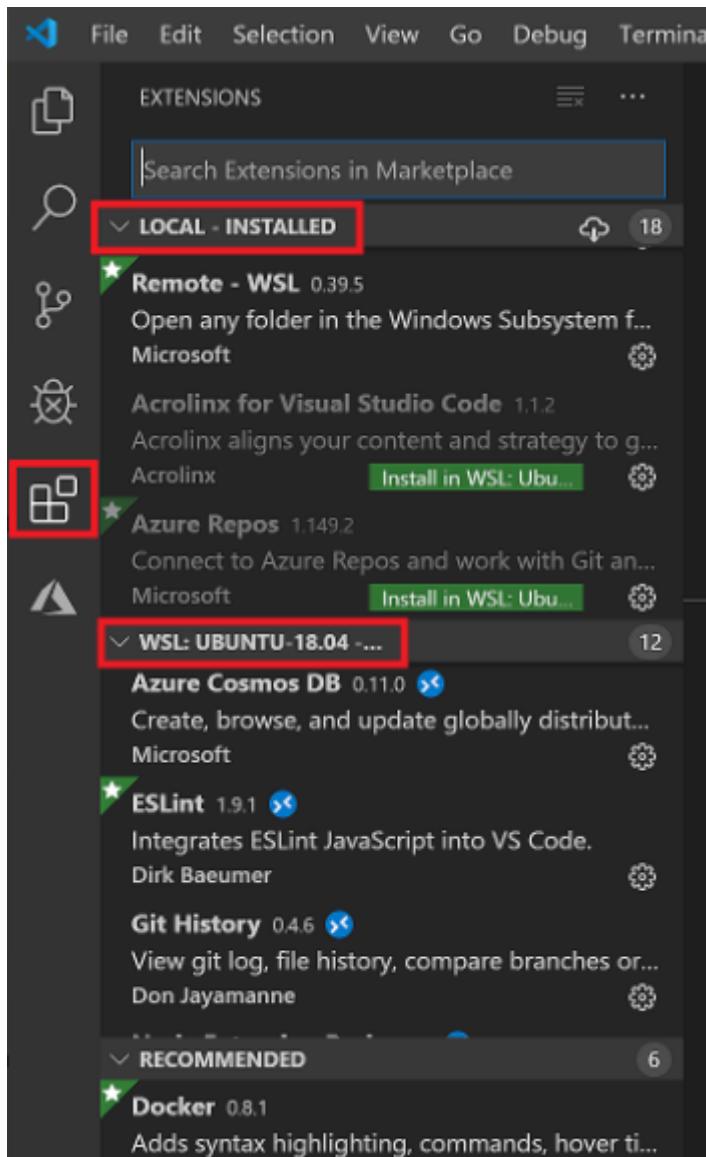
While VS Code comes with many features for Node.js development out of the box, there are some helpful extensions to consider installing available in the [Node.js Extension Pack ↗](#). Install them all or pick and choose which seem the most useful to you.

To install the Node.js extension pack:

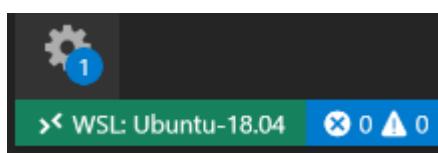
1. Open the **Extensions** window (Ctrl+Shift+X) in VS Code.

The Extensions window is now divided into three sections (because you installed the Remote-WSL extension).

- "Local - Installed": The extensions installed for use with your Windows operating system.
- "WSL:Ubuntu-18.04-Installed": The extensions installed for use with your Ubuntu operating system (WSL).
- "Recommended": Extensions recommended by VS Code based on the file types in your current project.



2. In the search box at the top of the Extensions window, enter: **Node Extension Pack** (or the name of whatever extension you are looking for). The extension will be installed for either your Local or WSL instances of VS Code depending on where you have the current project opened. You can tell by selecting the remote link in the bottom-left corner of your VS Code window (in green). It will either give you the option to open or close a remote connection. Install your Node.js extensions in the "WSL:Ubuntu-18.04" environment.



A few additional extensions you may want to consider include:

- [JavaScript Debugger](#): Once you finish developing on the server side with Node.js, you'll need to develop and test the client side. This extension is a DAP-based JavaScript debugger. It debugs Node.js, Chrome, Edge, WebView2, VS Code extensions, and more.

- [Keymaps from other editors](#) : These extensions can help your environment feel right at home if you're transitioning from another text editor (like Atom, Sublime, Vim, eMacs, Notepad++, etc).
- [Settings Sync](#) : Enables you to synchronize your VS Code settings across different installations using GitHub. If you work on different machines, this helps keep your environment consistent across them.

Set up Git (optional)

To set up Git for a Node.js project on WSL, see the article [Get started using Git on Windows Subsystem for Linux](#) in the WSL documentation.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Getting started with Linux and Bash

Article • 11/28/2022

This tutorial will help those new to Linux to get started installing and updating packages using the Ubuntu distribution of Linux that is installed by default using WSL, as well as using some basic commands with the Bash command line.

Installing and Updating Software

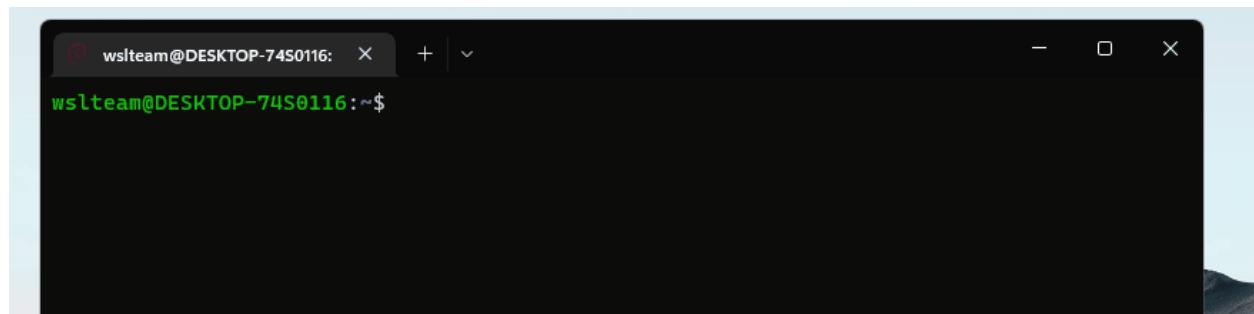
You can install and update software programs directly from the command line using the preferred package manager for the distribution you are running.

In Ubuntu, for example, first update the list of software available by running 'sudo apt update'. Then, you can directly get software by using the 'sudo apt-get install' command followed by the name of the program you wish to install:

```
Bash  
sudo apt-get install <app_name>
```

To update programs that have already been installed, you can run:

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



Tip

Different distributions of Linux often have different package managers and will require using an install command specific to the associated package manager. For example, the main package manager for Arch Linux is called [pacman](#) and the install command would be `sudo pacman -S <app_name>`. The main package manager

for OpenSuse is called **Zypper** ↗ and the install command would be `sudo zypper install <app_name>`. The main package manager for Alpine is called **apk** ↗ and the install command would be `sudo apk add <app_name>`. The two main package managers for Red Hat distributions, like CentOS, are **YUM** and **RPM** ↗ and an install command could be `sudo yum install <app_name>` or `sudo rpm -i <app_name>`. Refer to the documentation of the distribution you are working with to find out what tools are available for you to install and update software.

Working with files and directories

To view the path of the directory you are currently in, use the ‘pwd’ command:

```
Bash
```

```
pwd
```

To create a new directory, use the ‘mkdir’ command followed by the name of the directory you want to create:

```
Bash
```

```
mkdir hello_world
```

To change directories, use the ‘cd’ command followed by the name of the directory you want to navigate to:

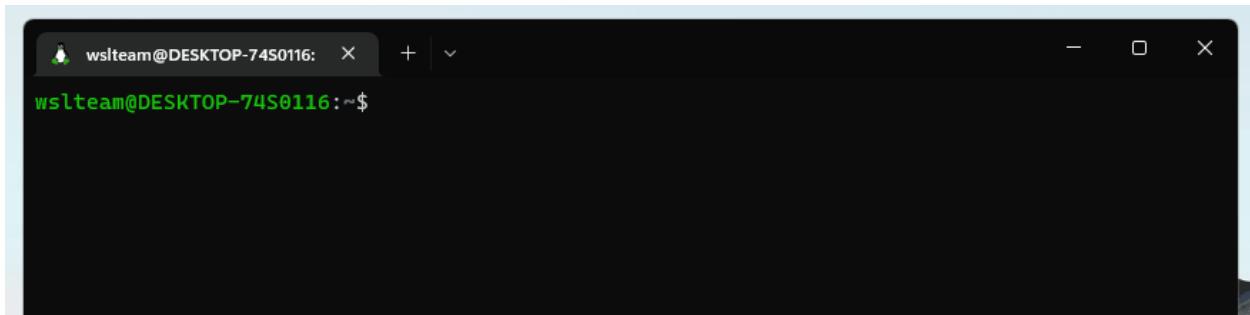
```
Bash
```

```
cd hello_world
```

To see the contents within the directory you are currently in, type ‘ls’ into the command line:

```
Bash
```

```
ls
```

A screenshot of a Windows Subsystem for Linux (WSL) terminal window. The title bar shows "wslteam@DESKTOP-74S0116: ~" and the command line prompt is "\$". The main area of the terminal is completely black, indicating no output.

[↗](#)

By default, the 'ls' command will print the name of all the files and directories only. To get additional information such as the last time a file was modified or file permissions, use the flag "-l":

Bash

```
ls -l
```

You can create a new file via the 'touch' command followed by the name of the file you would like to create:

Bash

```
touch hello_world.txt
```

You can edit files using any downloaded graphical text-editor or the VS Code Remote – WSL extension. You can learn more about getting started with VS Code [here](#)

If you prefer to edit a file directly from the command-line, you'll need to use a command-line editor such as vim, emacs, or nano. Many distributions come with one or more of these programs installed, but you can always install these programs by following the installation instructions outlined in the guide from [above](#)↗ .

To edit your file with your preferred method of editing, simply run the program name followed by the name of the file you'd like to edit:

Bash

```
code hello_world.txt
```

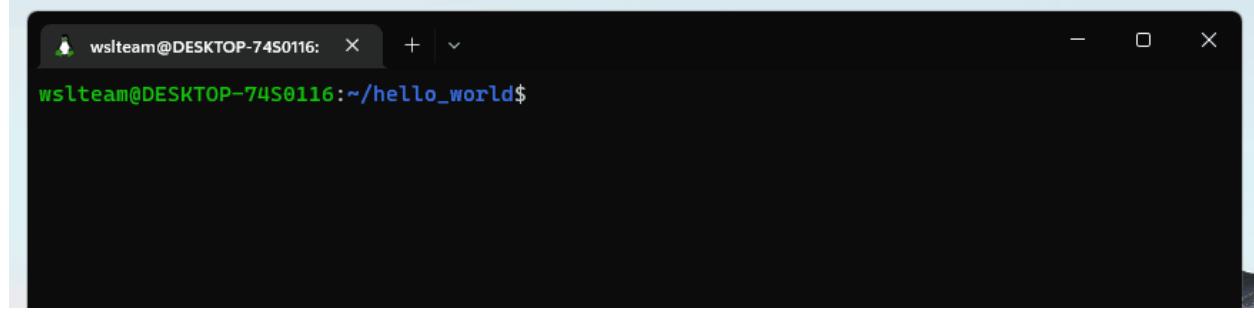
Bash

```
notepad.exe hello_world.txt
```

To see the contents of a file in the command line, use the 'cat' command followed by the file you'd like to read:

```
Bash

cat hello_world.txt
```



The screenshot shows a terminal window titled 'Bash'. The command 'cat hello_world.txt' is entered and executed. The output is a blank black screen, indicating that the file 'hello_world.txt' is empty.

Using Pipes and Redirect Operators

A pipe '|' redirects the output from one command as input into another command. For example, lhscmd | rhscmd would direct the output from lhscmd to rhscmd. Pipes can be used in a variety of ways to quickly accomplish tasks through the command line. Below are just a few simple examples of how pipes can be used.

Imagine you want to quickly sort the contents of a file. Take the fruits.txt example below:

```
Bash

cat fruits.txt

Orange

Banana

Apple

Pear

Plum

Kiwi

Strawberry

Peach
```

You can quickly sort this list by using a pipe:

Bash

```
$ cat fruits.txt | sort
```

Apple

Banana

Kiwi

Orange

Peach

Pear

Plum

Strawberry

By default, the output of the 'cat' command is sent to standard output; however, the '|' allows us to instead redirect the output as the input to another command, 'sort'.

Another use case is searching. You can use 'grep' which is a helpful command that searches input for a particular search string.

Bash

```
cat fruits.txt | grep P
```

Pear

Plum

Peach

You can also use redirect operators like '>' to pass the output to a file or stream. For example, if you wanted to create a new .txt file with the sorted contents of fruit.txt:

Bash

```
cat fruits.txt | sort > sorted_fruit.txt
```

Bash

```
$ cat sorted_fruit.txt
```

Apple

Banana

Kiwi

Orange

Peach

Pear

Plum

Strawberry

By default, the output of the sort command is sent to standard output; however, the '>' operator allows us to instead redirect the output into a new file named sorted_fruits.txt.

You can use pipes and redirect operators in many interesting ways to more efficiently complete tasks directly from the command line.

Recommended content

- [Microsoft Learn: Introduction to Bash](#)
- [Command Line for Beginners ↗](#)
- [Microsoft Learn: Get Started with WSL](#)

Working across Windows and Linux file systems

Article • 03/03/2022

There are a number of considerations to keep in mind when working between Windows and Linux file systems. We have outlined a few of them for you in this guide, including some examples of interoperability support for mixing Windows and Linux-based commands.

File storage and performance across file systems

We recommend against working across operating systems with your files, unless you have a specific reason for doing so. For the fastest performance speed, store your files in the WSL file system if you are working in a Linux command line (Ubuntu, OpenSUSE, etc). If you're working in a Windows command line (PowerShell, Command Prompt), store your files in the Windows file system.

For example, when storing your WSL project files:

- Use the Linux file system root directory: `\wsl$\Ubuntu\home<user name>\Project`
- Not the Windows file system root directory: `/mnt/c/Users/<user name>/Project$` or `C:\Users\<user name>\Project`

When you see `/mnt/` in the file path of a WSL command line, it means that you are working from a mounted drive. So the Windows file system C:/ drive (`C:\Users\<user name>\Project`) will look like this when mounted in a WSL command line: `/mnt/c/Users/<user name>/Project$`. It is possible to store your project files on a mounted drive, but your performance speed will improve if you store them directly on the `\wsl$` drive.

View your current directory in Windows File Explorer

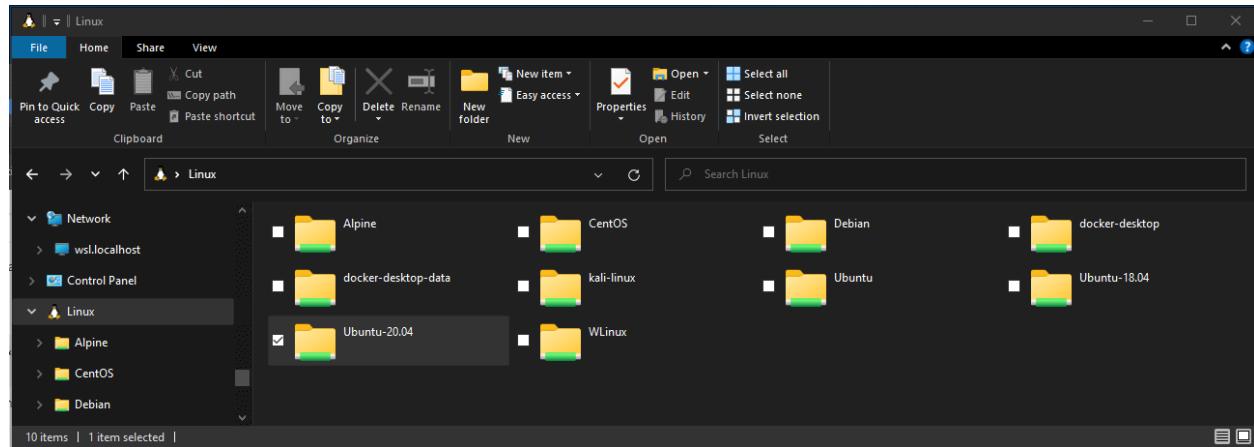
You can view the directory where your files are stored by opening the Windows File Explorer from the command line, using:

Bash

```
explorer.exe .
```

Alternatively, you can also use the command: `powershell.exe /c start .` Be sure to add the period at the end of the command to open the current directory.

To view all of your available Linux distributions and their root file systems in Windows File explorer, in the address bar enter: `\\\wsl$`



Filename and directory case sensitivity

Case sensitivity determines whether uppercase (FOO.txt) and lowercase (foo.txt) letters are handled as distinct (case-sensitive) or equivalent (case-insensitive) in a file name or directory. Windows and Linux file systems handle case sensitivity in different ways – Windows is case-insensitive and Linux is case-sensitive. Learn more about how to adjust case sensitivity, particularly when mounting disks with WSL, in the [Adjust case sensitivity](#) how-to article.

Interoperability between Windows and Linux commands

Windows and Linux tools and commands can be used interchangeably with WSL.

- Run Windows tools (ie. notepad.exe) from a Linux command line (ie. Ubuntu).
- Run Linux tools (ie. grep) from a Windows command line (ie. PowerShell).
- Share environment variables between Linux and Windows. (Build 17063+)

Run Linux tools from a Windows command line

Run Linux binaries from the Windows Command Prompt (CMD) or PowerShell using `wsl`

`<command>` (or `wsl.exe <command>`).

For example:

PowerShell

```
C:\temp> wsl ls -la  
<- contents of C:\temp ->
```

Binaries invoked in this way:

- Use the same working directory as the current CMD or PowerShell prompt.
- Run as the WSL default user.
- Have the same Windows administrative rights as the calling process and terminal.

The Linux command following `wsl` (or `wsl.exe`) is handled like any command run in WSL. Things such as `sudo`, piping, and file redirection work.

Example using `sudo` to update your default Linux distribution:

PowerShell

```
C:\temp> wsl sudo apt-get update
```

Your default Linux distribution user name will be listed after running this command and you will be asked for your password. After entering your password correctly, your distribution will download updates.

Mixing Linux and Windows commands

Here are a few examples of mixing Linux and Windows commands using PowerShell.

To use the Linux command `ls -la` to list files and the PowerShell command `findstr` to filter the results for words containing "git", combine the commands:

PowerShell

```
wsl ls -la | findstr "git"
```

To use the PowerShell command `dir` to list files and the Linux command `grep` to filter the results for words containing "git", combine the commands:

```
PowerShell
```

```
C:\temp> dir | wsl grep git
```

To use the Linux command `ls -la` to list files and the PowerShell command `> out.txt` to print that list to a text file named "out.txt", combine the commands:

```
PowerShell
```

```
C:\temp> wsl ls -la > out.txt
```

The commands passed into `wsl.exe` are forwarded to the WSL process without modification. File paths must be specified in the WSL format.

To use the Linux command `ls -la` to list files in the `/proc/cpuinfo` Linux file system path, using PowerShell:

```
PowerShell
```

```
C:\temp> wsl ls -la /proc/cpuinfo
```

To use the Linux command `ls -la` to list files in the `C:\Program Files` Windows file system path, using PowerShell:

```
PowerShell
```

```
C:\temp> wsl ls -la "/mnt/c/Program Files"
```

Run Windows tools from Linux

WSL can run Windows tools directly from the WSL command line using `[tool-name].exe`. For example, `notepad.exe`.

Applications run this way have the following properties:

- Retain the working directory as the WSL command prompt (for the most part -- exceptions are explained below).
- Have the same permission rights as the WSL process.
- Run as the active Windows user.
- Appear in the Windows Task Manager as if directly executed from the CMD prompt.

Windows executables run in WSL are handled similarly to native Linux executables -- piping, redirects, and even backgrounding work as expected.

To run the Windows tool `ipconfig.exe`, use the Linux tool `grep` to filter the "IPv4" results, and use the Linux tool `cut` to remove the column fields, from a Linux distribution (for example, Ubuntu) enter:

```
Bash
```

```
ipconfig.exe | grep IPv4 | cut -d: -f2
```

Let's try an example mixing Windows and Linux commands. Open your Linux distribution (ie. Ubuntu) and create a text file: `touch foo.txt`. Now use the Linux command `ls -la` to list the direct files and their creation details, plus the Windows PowerShell tool `findstr.exe` to filter the results so only your `foo.txt` file shows in the results:

```
Bash
```

```
ls -la | findstr.exe foo.txt
```

Windows tools must include the file extension, match the file case, and be executable. Non-executables including batch scripts. CMD native commands like `dir` can be run with `cmd.exe /C` command.

For example, list the contents of your Windows files system C:\ directory, by entering:

```
Bash
```

```
cmd.exe /C dir
```

Or use the `ping` command to send an echo request to the microsoft.com website:

```
Bash
```

```
ping.exe www.microsoft.com
```

Parameters are passed to the Windows binary unmodified. As an example, the following command will open `C:\temp\foo.txt` in `notepad.exe`:

```
Bash
```

```
notepad.exe "C:\temp\foo.txt"
```

This will also work:

Bash

```
notepad.exe C:\\temp\\foo.txt
```

Share environment variables between Windows and WSL with WSLENV

WSL and Windows share a special environment variable, `WSLENV`, created to bridge Windows and Linux distributions running on WSL.

Properties of `WSLENV` variable:

- It is shared; it exists in both Windows and WSL environments.
- It is a list of environment variables to share between Windows and WSL.
- It can format environment variables to work well in Windows and WSL.
- It can assist in the flow between WSL and Win32.

ⓘ Note

Prior to 17063, only Windows environment variable that WSL could access was `PATH` (so you could launch Win32 executables from under WSL). Starting in 17063, `WSLENV` begins being supported. `WSLENV` is case sensitive.

WSLENV flags

There are four flags available in `WSLENV` to influence how the environment variable is translated.

`WSLENV` flags:

- `/p` - translates the path between WSL/Linux style paths and Win32 paths.
- `/1` - indicates the environment variable is a list of paths.
- `/u` - indicates that this environment variable should only be included when running WSL from Win32.
- `/w` - indicates that this environment variable should only be included when running Win32 from WSL.

Flags can be combined as needed.

Read more about [WSLENV](#), including FAQs and examples of setting the value of WSLENV to a concatenation of other pre-defined environment vars, each suffixed with a slash followed by flags to specify how the value should be translated and passing variables with a script. This article also includes an example for setting up a dev environment with the [Go programming language](#), configured to share a GOPATH between WSL and Win32.

Disable interoperability

Users may disable the ability to run Windows tools for a single WSL session by running the following command as root:

Bash

```
echo 0 > /proc/sys/fs/binfmt_misc/WSLInterop
```

To re-enable Windows binaries, exit all WSL sessions and re-run bash.exe or run the following command as root:

Bash

```
echo 1 > /proc/sys/fs/binfmt_misc/WSLInterop
```

Disabling interop will not persist between WSL sessions -- interop will be enabled again when a new session is launched.

Advanced settings configuration in WSL

Article • 01/17/2024

The `wsl.conf` and `.wslconfig` files are used to configure advanced settings options, on a per-distribution basis (`wsl.conf`) and globally across all WSL 2 distributions (`.wslconfig`). This guide will cover each of the settings options, when to use each file type, where to store the file, sample settings files and tips.

What is the difference between `wsl.conf` and `.wslconfig`?

You can configure the settings for your installed Linux distributions that will automatically be applied every time you launch WSL in two ways, by using:

- `.wslconfig` to configure **global settings** across all installed distributions running on WSL 2.
- `wsl.conf` to configure **local settings** per-distribution for each Linux distribution running on WSL 1 or WSL 2.

Both file types are used for configuring WSL settings, but the location where the file is stored, the scope of the configuration, the type of options that can be configured, and the version of WSL running your distribution all impact which file type to choose.

WSL 1 and WSL 2 run with different architecture and will impact the configuration settings. WSL 2 runs as a lightweight virtual machine (VM), so uses virtualization settings that allow you to control the amount of memory or processors used (which may be familiar if you use Hyper-V or VirtualBox). [Check which version of WSL you are running.](#)

The 8 second rule for configuration changes

You must wait until the subsystem running your Linux distribution completely stops running and restarts for configuration setting updates to appear. This typically takes about 8 seconds after closing ALL instances of the distribution shell.

If you launch a distribution (e.g. Ubuntu), modify the configuration file, close the distribution, and then re-launch it, you might assume that your configuration changes have immediately gone into effect. This is not currently the case as the subsystem could still be running. You must wait for the subsystem to stop before relaunching in order to give enough time for your changes to be picked up. You can check to see whether your Linux distribution (shell) is still running after closing it by using PowerShell with the

command: `wsl --list --running`. If no distributions are running, you will receive the response: "There are no running distributions." You can now restart the distribution to see your configuration updates applied.

The command `wsl --shutdown` is a fast path to restarting WSL 2 distributions, but it will shut down all running distributions, so use wisely. You can also use `wsl --terminate <distroName>` to terminate a specific distribution that's running instantly.

wsl.conf

Configure **local settings** with `wsl.conf` per-distribution for each Linux distribution running on WSL 1 or WSL 2.

- Stored in the `/etc` directory of the distribution as a unix file.
- Used to configure settings on a per-distribution basis. Settings configured in this file will only be applied to the specific Linux distribution that contains the directory where this file is stored.
- Can be used for distributions run by either version, WSL 1 or WSL 2.
- To get to the `/etc` directory for an installed distribution, use the distribution's command line with `cd /` to access the root directory, then `ls` to list files or `explorer.exe .` to view in Windows File Explorer. The directory path should look something like: `/etc/wsl.conf`.

ⓘ Note

Adjusting per-distribution settings with the `wsl.conf` file is only available in Windows Build 17093 and later.

Configuration settings for wsl.conf

The `wsl.conf` file configures settings on a per-distribution basis. (*For global configuration of WSL 2 distributions see [.wslconfig](#).*)

The `wsl.conf` file supports four sections: `automount`, `network`, `interop`, and `user`. (*Modeled after .ini file conventions, keys are declared under a section, like .gitconfig files.*) See [wsl.conf](#) for info on where to store the `wsl.conf` file.

systemd support

Many Linux distributions run "systemd" by default (including Ubuntu) and WSL has recently added support for this system/service manager so that WSL is even more similar to using your favorite Linux distributions on a bare metal machine. You will need version 0.67.6+ of WSL to enable systemd. Check your WSL version with command `wsl --version`. If you need to update, you can grab the [latest version of WSL in the Microsoft Store](#). Learn more in [blog announcement](#).

To enable systemd, open your `wsl.conf` file in a text editor using `sudo` for admin permissions and add these lines to the `/etc/wsl.conf`:

```
Bash
[boot]
systemd=true
```

You will then need to close your WSL distribution using `wsl.exe --shutdown` from PowerShell to restart your WSL instances. Once your distribution restarts, systemd should be running. You can confirm using the command: `systemctl list-unit-files --type=service`, which will show the status of your services.

Automount settings

wsl.conf section label: `[automount]`

[Expand table](#)

key	value	default	notes
enabled	boolean	true	<code>true</code> causes fixed drives (i.e <code>C:/</code> or <code>D:/</code>) to be automatically mounted with DrvFs under <code>/mnt</code> . <code>false</code> means drives won't be mounted automatically, but you could still mount them manually or via <code>fstab</code> .
mountFsTab	boolean	true	<code>true</code> sets <code>/etc/fstab</code> to be processed on WSL start. <code>/etc/fstab</code> is a file where you can declare other filesystems, like an SMB share. Thus, you can mount these filesystems automatically in WSL on start up.
root	string	<code>/mnt/</code>	Sets the directory where fixed drives will be automatically mounted. By default this is set to <code>/mnt/</code> , so your Windows file system C-drive is mounted to <code>/mnt/c/</code> . If you change <code>/mnt/</code> to

key	value	default	notes
			/windir/, you should expect to see your fixed C-drive mounted to /windir/c.
options	comma-separated list of values, such as uid, gid, etc, see automount options below	empty string	The automount option values are listed below and are appended to the default DrvFs mount options string. Only DrvFs-specific options can be specified.

The automount options are applied as the mount options for all automatically mounted drives. To change the options for a specific drive only, use the `/etc/fstab` file instead. Options that the mount binary would normally parse into a flag are not supported. If you want to explicitly specify those options, you must include every drive for which you want to do so in `/etc/fstab`.

Automount options

Setting different mount options for Windows drives (DrvFs) can control how file permissions are calculated for Windows files. The following options are available:

[Expand table](#)

Key	Description	Default
uid	The User ID used for the owner of all files	The default User ID of your WSL distro (on first installation this defaults to 1000)
gid	The Group ID used for the owner of all files	The default group ID of your WSL distro (on first installation this defaults to 1000)
umask	An octal mask of permissions to exclude for all files and directories	022
fmask	An octal mask of permissions to exclude for all files	000
dmask	An octal mask of permissions to exclude for all directories	000
metadata	Whether metadata is added to Windows files to support Linux system permissions	disabled

Key	Description	Default
case	Determines directories treated as case sensitive and whether new directories created with WSL will have the flag set. See case sensitivity for a detailed explanation of the options. Options include <code>off</code> , <code>dir</code> , or <code>force</code> .	<code>off</code>

By default, WSL sets the uid and gid to the value of the default user. For example, in Ubuntu, the default user is uid=1000, gid=1000. If this value is used to specify a different gid or uid option, the default user value will be overwritten. Otherwise, the default value will always be appended.

User file-creation mode mask (umask) sets permission for newly created files. The default is 022, only you can write data but anyone can read data. Values can be changed to reflect different permission settings. For example, `umask=077` changes permission to be completely private, no other user can read or write data. To further specify permission, fmask (files) and dmask (directories) can also be used.

ⓘ Note

The permission masks are put through a logical OR operation before being applied to files or directories.

What is DrvFs?

DrvFs is a filesystem plugin to WSL that was designed to support interop between WSL and the Windows filesystem. DrvFs enables WSL to mount drives with supported file systems under /mnt, such as /mnt/c, /mnt/d, etc. For more information about specifying the default case sensitivity behavior when mounting Windows or Linux drives or directories, see the [case sensitivity](#) page.

Network settings

wsl.conf section label: `[network]`

[\[\] Expand table](#)

key	value	default	notes
generateHosts	boolean	<code>true</code>	<code>true</code> sets WSL to generate <code>/etc/hosts</code> . The <code>hosts</code> file contains a static map of hostnames corresponding IP address.

key	value	default	notes
generateResolvConf	boolean	true	<code>true</code> sets WSL to generate <code>/etc/resolv.conf</code> . The <code>resolv.conf</code> contains a DNS list that are capable of resolving a given hostname to its IP address.
hostname	string	Windows hostname	Sets hostname to be used for WSL distribution.

Interop settings

wsl.conf section label: `[interop]`

These options are available in Insider Build 17713 and later.

[\[+\] Expand table](#)

key	value	default	notes
enabled	boolean	true	Setting this key will determine whether WSL will support launching Windows processes.
appendWindowsPath	boolean	true	Setting this key will determine whether WSL will add Windows path elements to the \$PATH environment variable.

User settings

wsl.conf section label: `[user]`

These options are available in Build 18980 and later.

[\[+\] Expand table](#)

key	value	default	notes
default	string	The initial username created on first run	Setting this key specifies which user to run as when first starting a WSL session.

Boot settings

The Boot setting is only available on Windows 11 and Server 2022.

wsl.conf section label: `[boot]`

key	value	default	notes
command	string	""	A string of the command that you would like to run when the WSL instance starts. This command is run as the root user. e.g: <code>service docker start</code> .

Example wsl.conf file

The `wsl.conf` sample file below demonstrates some of the configuration options available. In this example, the distribution is Ubuntu-20.04 and the file path is `\wsl.localhost\Ubuntu-20.04\etc\wsl.conf`.

Bash

```
# Automatically mount Windows drive when the distribution is launched
[automount]

# Set to true will automount fixed drives (C:/ or D:/) with DrvFs under the
# root directory set above. Set to false means drives won't be mounted
# automatically, but need to be mounted manually or with fstab.
enabled = true

# Sets the directory where fixed drives will be automatically mounted. This
# example changes the mount location, so your C-drive would be /c, rather than
# the default /mnt/c.
root = /

# DrvFs-specific options can be specified.
options = "metadata,uid=1003,gid=1003,umask=077,fmask=11,case=off"

# Sets the `/etc/fstab` file to be processed when a WSL distribution is
# launched.
mountFsTab = true

# Network host settings that enable the DNS server used by WSL 2. This
# example changes the hostname, sets generateHosts to false, preventing WSL
# from the default behavior of auto-generating /etc/hosts, and sets
# generateResolvConf to false, preventing WSL from auto-generating
# /etc/resolv.conf, so that you can create your own (ie. nameserver 1.1.1.1).
[network]
hostname = DemoHost
generateHosts = false
generateResolvConf = false

# Set whether WSL supports interop processes like launching Windows apps and
# adding path variables. Setting these to false will block the launch of
# Windows processes and block adding $PATH environment variables.
[interop]
enabled = false
```

```
appendWindowsPath = false

# Set the user when launching a distribution with WSL.
[user]
default = DemoUser

# Set a command to run when a new WSL instance launches. This example starts
# the Docker container service.
[boot]
command = service docker start
```

.wslconfig

Configure **global settings** with `.wslconfig` across all installed distributions running on WSL.

- The `.wslconfig` file does not exist by default. It must be created and stored in your `%UserProfile%` directory to apply these configuration settings.
- Used to configure settings globally across all installed Linux distributions running as the WSL 2 version.
- Can be used **only for distributions run by WSL 2**. Distributions running as WSL 1 will not be affected by this configuration as they are not running as a virtual machine.
- To get to your `%UserProfile%` directory, in PowerShell, use `cd ~` to access your home directory (which is typically your user profile, `c:\Users\<UserName>`) or you can open Windows File Explorer and enter `%UserProfile%` in the address bar. The directory path should look something like: `c:\Users\<UserName>\.wslconfig`.

WSL will detect the existence of these files, read the contents, and automatically apply the configuration settings every time you launch WSL. If the file is missing or malformed (improper markup formatting), WSL will continue to launch as normal without the configuration settings applied.

Configuration settings for .wslconfig

The `.wslconfig` file configures settings globally for all Linux distributions running with WSL 2. (*For per-distribution configuration see [wsl.conf](#)*).

See [.wslconfig](#) for info on where to store the `.wslconfig` file.

Note

Configuring global settings with `.wslconfig` are only available for distributions running as WSL 2 in Windows Build 19041 and later. Keep in mind you may need to run `wsl --shutdown` to shut down the WSL 2 VM and then restart your WSL instance for these changes to take effect.

This file can contain the following options that affect the VM that powers any WSL 2 distribution:

Main WSL settings

.wslconfig section label: `[ws12]`

[\[+\]](#) Expand table

key	value	default	notes
kernel	path	The Microsoft built kernel provided inbox	An absolute Windows path to a custom Linux kernel.
memory	size	50% of total memory on Windows	How much memory to assign to the WSL 2 VM.
processors	number	The same number of logical processors on Windows	How many logical processors to assign to the WSL 2 VM.
localhostForwarding	boolean	<code>true</code>	Boolean specifying if ports bound to wildcard or localhost in the WSL 2 VM should be connectable from the host via <code>localhost:port</code> .
kernelCommandLine	string	Blank	Additional kernel command line arguments.

key	value	default	notes
safeMode	boolean	false	Run WSL in "Safe Mode" which disables many features and is intended to be used to recover distributions that are in bad states. Only available for Windows 11 and WSL version 0.66.2+.
swap	size	25% of memory size on Windows rounded up to the nearest GB	How much swap space to add to the WSL 2 VM, 0 for no swap file. Swap storage is disk-based RAM used when memory demand exceeds limit on hardware device.
swapFile	path	%USERPROFILE%\AppData\Local\Temp\swap.vhdx	An absolute Windows path to the swap virtual hard disk.
pageReporting	boolean	true	Default true setting enables Windows to reclaim unused memory allocated to WSL 2 virtual machine.
guiApplications	boolean*	true	Boolean to turn on or off support for GUI applications (WSLg) in WSL. Only available for Windows 11.

key	value	default	notes
debugConsole	boolean*	false	Boolean to turn on an output console Window that shows the contents of <code>dmesg</code> upon start of a WSL 2 distro instance. Only available for Windows 11.
nestedVirtualization	boolean*	true	Boolean to turn on or off nested virtualization, enabling other nested VMs to run inside WSL 2. Only available for Windows 11.
vmlIdleTimeout	number*	60000	The number of milliseconds that a VM is idle, before it is shut down. Only available for Windows 11.
dnsProxy	bool	true	Only applicable to networkingMode = NAT. Boolean to inform WSL to configure the DNS Server in Linux to the NAT on the host. Setting to false will mirror DNS servers from Windows to Linux.
networkingMode**	string	NAT	If the value is mirrored then this turns on mirrored networking

key	value	default	notes
			mode. Default or unrecognized strings result in NAT networking.
firewall**	bool	true	Setting this to true allows the Windows Firewall rules, as well as rules specific to Hyper-V traffic, to filter WSL network traffic.
dnsTunneling**	bool	false	Changes how DNS requests are proxied from WSL to Windows
autoProxy*	bool	false	Enforces WSL to use Windows' HTTP proxy information

Entries with the `path` value must be Windows paths with escaped backslashes, e.g:

`C:\\\\Temp\\\\myCustomKernel`

Entries with the `size` value must be a size followed by a unit, for example, `8GB` or `512MB`.

Entries with an `*` after the value type are only available on Windows 11.

Entries with an `**` after the value type require [Windows 11 version 22H2](#) or higher.

Experimental settings

These settings are opt-in previews of experimental features that we aim to make default in the future.

.wslconfig section label: `[experimental]`

[] Expand table

Setting name	Value	Default	Notes
<code>autoMemoryReclaim</code>	string	disabled	Automatically releases cached memory after detecting idle CPU usage. Set to <code>gradual</code> for slow release, and <code>dropcache</code> for instant release of cached memory.
<code>sparseVhd</code>	bool	false	When set to true, any newly created VHD will be set to sparse automatically.
<code>useWindowsDnsCache</code> **	bool	false	Only applicable when <code>ws12.dnsTunneling</code> is set to true. When this option is set to false, DNS requests tunneled from Linux will bypass cached names within Windows to always put the requests on the wire.
<code>bestEffortDnsParsing</code> **	bool	false	Only applicable when <code>ws12.dnsTunneling</code> is set to true. When set to true, Windows will extract the question from the DNS request and attempt to resolve it, ignoring the unknown records.
<code>initialAutoProxyTimeout</code> *	string	1000	Only applicable when <code>ws12.autoProxy</code> is set to true. Configures how long (in milliseconds) WSL will wait for retrieving HTTP proxy information when starting a WSL container. If proxy settings are resolved after this time, the WSL instance must be restarted to use the retrieved proxy settings.
<code>ignoredPorts</code> **	string	null	Only applicable when <code>ws12.networkingMode</code> is set to <code>mirrored</code> . Specifies which ports Linux applications can bind to, even if that port is used in Windows. This enables applications to listen on a port for traffic purely within Linux, so those applications are not blocked even when that port is used for other purposes on Windows. For example, WSL will allow binding to port 53 in Linux for Docker Desktop, as it is listening only to requests from within the Linux container. Should be formatted in a comma separated list, e.g: <code>3000,9000,9090</code>
<code>hostAddressLoopback</code> **	bool	false	Only applicable when <code>ws12.networkingMode</code> is set to <code>mirrored</code> . When set to <code>True</code> , will allow the Container to connect to the Host, or the Host to connect to the Container, by an IP address that's assigned to the Host. The <code>127.0.0.1</code> loopback address can always be used, this option allows for all additionally assigned local IP addresses to be

Setting name	Value	Default	Notes
			used as well. Only IPv4 addresses assigned to the host are supported.

Entries with an * after the value type are only available on Windows 11.

Entries with an ** after the value type require [Windows version 22H2](#) or higher.

Example .wslconfig file

The `.wslconfig` sample file below demonstrates some of the configuration options available. In this example, the file path is `C:\Users\<UserName>\.wslconfig`.

Bash

```
# Settings apply across all Linux distros running on WSL 2
[wsl2]

# Limits VM memory to use no more than 4 GB, this can be set as whole
# numbers using GB or MB
memory=4GB

# Sets the VM to use two virtual processors
processors=2

# Specify a custom Linux kernel to use with your installed distros. The
# default kernel used can be found at https://github.com/microsoft/WSL2-Linux-
# Kernel
kernel=C:\\temp\\myCustomKernel

# Sets additional kernel parameters, in this case enabling older Linux base
# images such as Centos 6
kernelCommandLine = vsyscall=emulate

# Sets amount of swap storage space to 8GB, default is 25% of available RAM
swap=8GB

# Sets swapfile path location, default is
%USERPROFILE%\AppData\Local\Temp\swap.vhdx
swapfile=C:\\temp\\wsl-swap.vhdx

# Disable page reporting so WSL retains all allocated memory claimed from
# Windows and releases none back when free
pageReporting=false

# Turn on default connection to bind WSL 2 localhost to Windows localhost.
# Setting is ignored when networkingMode=mirrored
localhostforwarding=true

# Disables nested virtualization
```

```
nestedVirtualization=false

# Turns on output console showing contents of dmesg when opening a WSL 2
# distro for debugging
debugConsole=true

# Enable experimental features
[experimental]
sparseVhd=true
```

Additional resources

- [Windows Command Line Blog: Automatically Configuring WSL ↗](#)
- [Windows Command Line Blog: Chmod/Chown, DrvFs, file metadata ↗](#)



Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

[Open a documentation issue](#)

[Provide product feedback](#)

File Permissions for WSL

Article • 12/29/2021

This page details how Linux file permissions are interpreted across the Windows Subsystem for Linux, especially when accessing resources inside of Windows on the NT file system. This documentation assumes a basic understanding of the [Linux file system permissions structure](#) and the [umask command](#).

When accessing Windows files from WSL the file permissions are either calculated from Windows permissions, or are read from metadata that has been added to the file by WSL. This metadata is not enabled by default.

WSL metadata on Windows files

When metadata is enabled as a mount option in WSL, extended attributes on Windows NT files can be added and interpreted to supply Linux file system permissions.

WSL can add four NTFS extended attributes:

[] [Expand table](#)

Attribute Name	Description
\$LXUID	User Owner ID
\$LXGID	Group Owner ID
\$LXMOD	File mode (File systems permission octals and type, e.g: 0777)
\$LXDEV	Device, if it is a device file

Additionally, any file that is not a regular file or directory (e.g: symlinks, FIFOs, block devices, unix sockets, and character devices) also have an NTFS [reparse point](#). This makes it much faster to determine the kind of file in a given directory without having to query its extended attributes.

File Access Scenarios

Below is a description of how permissions are determined when accessing files in different ways using the Windows Subsystem for Linux.

Accessing Files in the Windows drive file system (DrvFS) from Linux

These scenarios occur when you are accessing your Windows files from WSL, most likely via `/mnt/c`.

Reading file permissions from an existing Windows file

The result depends on if the file already has existing metadata.

DrvFS file does not have metadata (default)

If the file has no metadata associated with it then we translate the effective permissions of the Windows user to read/write/execute bits and set them to this as the same value for user, group, and other. For example, if your Windows user account has read and execute access but not write access to the file then this will be shown as `r-x` for user, group and other. If the file has the 'Read Only' attribute set in Windows then we do not grant write access in Linux.

The file has metadata

If the file has metadata present, we simply use those metadata values instead of translating effective permissions of the Windows user.

Changing file permissions on an existing Windows file using chmod

The result depends on if the file already has existing metadata.

chmod file does not have metadata (default)

Chmod will only have one effect, if you remove all the write attributes of a file then the 'read only' attribute on the Windows file will be set, since this is the same behavior as CIFS (Common Internet File System) which is the SMB (Server Message Block) client in Linux.

chmod file has metadata

Chmod will change or add metadata depending on the file's already existing metadata.

Please keep in mind that you cannot give yourself more access than what you have on Windows, even if the metadata says that is the case. For example, you could set the

metadata to display that you have write permissions to a file using `chmod 777`, but if you tried to access that file you would still not be able to write to it. This is thanks to interoperability, as any read or write commands to Windows files are routed through your Windows user permissions.

Creating a file in DriveFS

The result depends on if metadata is enabled.

Metadata is not enabled (default)

The Windows permissions of the newly created file will be the same as if you created the file in Windows without a specific security descriptor, it will inherit the parent's permissions.

Metadata is enabled

The file's permission bits are set to follow the Linux umask, and the file will be saved with metadata.

Which Linux user and Linux group owns the file?

The result depends on if the file already has existing metadata.

User file does not have metadata (default)

In the default scenario, when automounting Windows drives, we specify that the user ID (UID) for any file is set to the user ID of your WSL user and the group ID (GID) is set to the principal group ID of your WSL user.

User file has metadata

The UID and GID specified in the metadata is applied as the user owner and group owner of the file.

Accessing Linux files from Windows using `\wsl$`

Accessing Linux files via `\wsl$` will use the default user of your WSL distribution.

Therefore any Windows app accessing Linux files will have the same permissions as the default user.

Creating a new file

The default umask is applied when creating a new file inside of a WSL distribution from Windows. The default umask is `022`, or in other words it allows all permissions except write permissions to groups and others.

Accessing files in the Linux root file system from Linux

Any files created, modified, or accessed in the Linux root file system follow standard Linux conventions, such as applying the umask to a newly created file.

Configuring file permissions

You can configure your file permissions inside of your Windows drives using the mount options in `wsl.conf`. The mount options allow you to set `umask`, `dmask` and `fmask` permissions masks. The `umask` is applied to all files, the `dmask` is applied just to directories and the `fmask` is applied just to files. These permission masks are then put through a logical OR operation when being applied to files, e.g: If you have a `umask` value of `023` and an `fmask` value of `022` then the resulting permissions mask for files will be `023`.

Learn more: [Per distribution configuration options with `wsl.conf`](#).

Accessing network applications with WSL

Article • 11/16/2023

There are a few considerations to be aware of when working with networking apps and WSL. By default WSL uses a [NAT based architecture](#), and we recommend trying the new [Mirrored networking mode](#) to get the latest features and improvements.

Default networking mode: NAT

By default, WSL uses a NAT (Network Address Translation) based architecture for networking. Keep the following considerations in mind when working with a NAT-based networking architecture:

Accessing Linux networking apps from Windows (localhost)

If you are building a networking app (for example an app running on a NodeJS or SQL server) in your Linux distribution, you can access it from a Windows app (like your Edge or Chrome internet browser) using `localhost` (just like you normally would).

Accessing Windows networking apps from Linux (host IP)

If you want to access a networking app running on Windows (for example an app running on a NodeJS or SQL server) from your Linux distribution (ie Ubuntu), then you need to use the IP address of your host machine. While this is not a common scenario, you can follow these steps to make it work.

1. Obtain the IP address of your host machine by running this command from your Linux distribution: `ip route show | grep -i default | awk '{ print $3}'`
2. Connect to any Windows server using the copied IP address.

The picture below shows an example of this by connecting to a Node.js server running in Windows via curl.

The screenshot shows two terminal windows side-by-side. The left window is a WSL terminal where the user runs commands to check the /etc/resolv.conf file and curl a local node.js application at 192.168.96.193:5000. The right window is a Windows PowerShell window running a Node.js script that prints "Hello World!" and handles an incoming HTTP request.

```
caloewen@LOE-WSL:/mnt/c$ cat /etc/resolv.conf
# This file was automatically generated by WSL. To stop automatic generation of this file, add the following entry to /etc/wsl.conf:
# [network]
# generateResolvConf = false
nameserver 192.168.96.193
caloewen@LOE-WSL:/mnt/c$ curl http://192.168.96.193:5000/
Hello World!
caloewen@LOE-WSL:/mnt/c$ |
```

```
E:\WSLTesting\nodeTest> node app.js
Server running at http://127.0.0.1:5000/
Received HTTP request
```

Connecting via remote IP addresses

When using remote IP addresses to connect to your applications, they will be treated as connections from the Local Area Network (LAN). This means that you will need to make sure your application can accept LAN connections.

For example, you may need to bind your application to `0.0.0.0` instead of `127.0.0.1`. In the example of a Python app using Flask, this can be done with the command: `app.run(host='0.0.0.0')`. Keep security in mind when making these changes as this will allow connections from your LAN.

Accessing a WSL 2 distribution from your local area network (LAN)

When using a WSL 1 distribution, if your computer was set up to be accessed by your LAN, then applications run in WSL could be accessed on your LAN as well.

This isn't the default case in WSL 2. WSL 2 has a virtualized ethernet adapter with its own unique IP address. Currently, to enable this workflow you will need to go through the same steps as you would for a regular virtual machine. (We are looking into ways to improve this experience.)

Here's an example of using the [Netsh interface portproxy](#) Windows command to add a port proxy that listens on your host port and connects that port proxy to the IP address for the WSL 2 VM.

The screenshot shows a PowerShell window with a single command entered: `netsh interface portproxy add v4tov4 listenport=<yourPortToForward> listenaddress=0.0.0.0 connectport=<yourPortToConnectToInWSL> connectaddress=(wsl hostname -I)`.

```
PowerShell
netsh interface portproxy add v4tov4 listenport=<yourPortToForward>
listenaddress=0.0.0.0 connectport=<yourPortToConnectToInWSL> connectaddress=
(wsl hostname -I)
```

In this example, you will need to update `<yourPortToForward>` to a port number, for example `listenport=4000`. `listenaddress=0.0.0.0` means that incoming requests will be accepted from ANY IP address. The Listen Address specifies the IPv4 address for which to listen and can be changed to values that include: IP address, computer NetBIOS name, or computer DNS name. If an address isn't specified, the default is the local computer. You need to update the `<yourPortToConnectToInWSL>` value to a port number where you want WSL to connect, for example `connectport=4000`. Lastly, the `connectaddress` value needs to be the IP address of your Linux distribution installed via WSL 2 (the WSL 2 VM address), which can be found by entering the command: `wsl.exe hostname -I`.

So this command may look something like:

PowerShell

```
netsh interface portproxy add v4tov4 listenport=4000 listenaddress=0.0.0.0  
connectport=4000 connectaddress=192.168.101.100
```

To obtain the IP address, use:

- `wsl hostname -I` for the IP address of your Linux distribution installed via WSL 2 (the WSL 2 VM address)
- `cat /etc/resolv.conf` for the IP address of the Windows machine as seen from WSL 2 (the WSL 2 VM)

Using `listenaddress=0.0.0.0` will listen on all [IPv4 ports](#).

ⓘ Note

Using a lowercase "i" with the `hostname` command will generate a different result than using an uppercase "I". `wsl hostname -i` is your local machine (127.0.1.1 is a placeholder diagnostic address), whereas `wsl hostname -I` will return your local machine's IP address as seen by other machines and should be used to identify the `connectaddress` of your Linux distribution running via WSL 2.

IPv6 access

- `wsl hostname -i` for the IP address of your Linux distribution installed via WSL 2 (the WSL 2 VM address)

- `ip route show | grep -i default | awk '{ print $3}'` for the IP address of the Windows machine as seen from WSL 2 (the WSL 2 VM)

Using `listenaddress=0.0.0.0` will listen on all [IPv4 ports](#).

Mirrored mode networking

You can [set networkingMode=mirrored under \[wsl2\]](#) in the `.wslconfig` file to enable mirrored mode networking. Enabling this changes WSL to an entirely new networking architecture which has the goal of 'mirroring' the network interfaces that you have on Windows into Linux, to add new networking features and improve compatibility.

Here are the current benefits to enabling this mode:

- IPv6 support
- Connect to Windows servers from within Linux using the localhost address `127.0.0.1`. IPv6 localhost address `::1` is not supported
- Improved networking compatibility for VPNs
- Multicast support
- Connect to WSL directly from your local area network (LAN)

ⓘ Note

Run the following command in PowerShell window with admin privileges to [Configure Hyper-V firewall](#) settings to allow inbound connections: `Set-NetFirewallHyperVVMSetting -Name '{40E0AC32-46A5-438A-A0B2-2B479E8F2E90}' -DefaultInboundAction Allow OR New-NetFirewallHyperVRule -Name "MyWebServer" -DisplayName "My Web Server" -Direction Inbound -VMCreatorId '{40E0AC32-46A5-438A-A0B2-2B479E8F2E90}' -Protocol TCP -LocalPorts 80.`

This new mode addresses networking issues seen with using a NAT (Network Address Translation) based architecture. Find known issues or file feedback on any bugs identified in the [WSL product repo on GitHub](#).

DNS Tunneling

Setting [dnsTunneling=true under \[wsl2\]](#) in the `.wslconfig` file has WSL use a virtualization feature to answer DNS requests from within WSL, instead of requesting them over a networking packet. This feature is aimed to improve compatibility with VPNs, and other complex networking set ups.

Auto Proxy

Setting [autoProxy=true under \[wsl2\]](#) in the [.wslconfig file](#) enforces WSL to use Windows' HTTP proxy information. If you have a proxy already set up in Windows, enabling this feature will make that proxy be set automatically in WSL as well.

WSL and firewall

On machines running Windows 11 22H2 and higher, with WSL 2.0.9 and higher, the Hyper-V firewall feature will be turned on by default. This will ensure that:

- See [Windows Defender Firewall with Advanced Security](#) to learn more about Windows security features that will automatically apply to WSL.
- See [Configure Hyper-V firewall](#) to learn more about applying these rules and settings both locally and via online tools like Intune.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Use systemd to manage Linux services with WSL

Article • 06/16/2023

Windows Subsystem for Linux (WSL) now supports systemd, an init system and service manager used by many popular Linux distributions such as Ubuntu, Debian, and more. ([What is systemd?](#)).

The init system default has recently changed from SystemV, with [systemd now the default for the current version of Ubuntu](#) that will be installed using the [wsl --install command](#) default. Linux distributions other than the current version of Ubuntu may still use the WSL init, similar to SystemV init. To change to systemd, see [How to enable systemd](#).

What is systemd in Linux?

According to [systemd.io](#): "systemd is a suite of basic building blocks for a Linux system. It provides a system and service manager that runs as PID 1 and starts the rest of the system."

Primarily an init system and service manager, systemd includes features like on-demand starting of daemons, mount and automount point maintenance, snapshot support, and processes tracking using Linux control groups.

Most major Linux distributions now run systemd, so enabling it on WSL brings the experience even closer to using bare-metal Linux. See the [video announcement with systemd demos](#) or [examples of using systemd](#) below to learn more about what systemd has to offer.

How to enable systemd?

Systemd is [now the default for the current version of Ubuntu](#) that will be installed using the [wsl --install command](#) default.

To enable systemd for any other Linux distributions running on WSL 2 (changing the default from using the systemv init):

1. Ensure that your WSL version is 0.67.6 or newer. (To check, run `wsl --version`. To update, run `wsl --update` or [download the latest version from the Microsoft Store](#).)

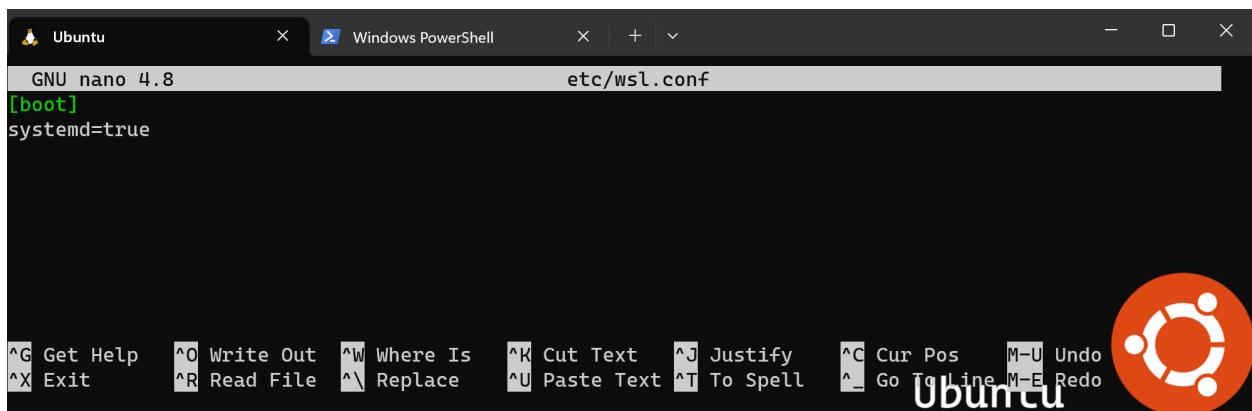
2. Open a command line for your Linux distribution and enter `cd /` to access the root directory, then `ls` to list the files. You will see a directory named "etc" that contains the WSL configuration file for the distribution. Open this file so that you can make an update with the Nano text editor by entering: `nano /etc/wsl.conf`.

3. Add these lines in the `wsl.conf` file that you now have open to change the init used to systemd:

```
Bash

[boot]
systemd=true
```

4. Exit the Nano text editor (Ctrl + X, select Y to save your change). You will then need to close the Linux distribution. You can use the command `wsl.exe --shutdown` in PowerShell to restart all WSL instances.



Once your Linux distribution restarts, systemd will be running. You can confirm using the command: `systemctl list-unit-files --type=service`, which will show the status of any services associated with your Linux distribution.

Learn more about [Advanced settings configuration in WSL](#), including the difference between the `wsl.conf` (distribution-specific) and `.wslconfig` (global) config files, how to update automount settings, etc.

Systemd demo video

Microsoft partnered with Canonical to bring systemd support to WSL. See Craig Loewen (PM for WSL at Microsoft) and Oliver Smith (PM for Ubuntu on WSL at Canonical) announce systemd support and show some demos of what it enables.

<https://learn-video.azurefd.net/vod/player?show=tabs-vs-spaces&ep=wsl-partnering-with-canonical-to-support-systemd&locale=en-us&embedUrl=%2Fwindows%2Fwsl%2Fsystemd>

- [Systemd support blog announcement ↗](#)
- [Oliver's tutorials based on these demos on the Ubuntu blog ↗](#) - includes "Use snap to create a Nextcloud instance in minutes on WSL", "Manage your web projects with LXD", and "[Run a .Net Echo Bot as a systemd service on Ubuntu WSL](#)" ↗
- [Craig's microk8s demo on GitHub ↗](#)

Systemd examples

A few examples of Linux applications that depend on systemd are:

- [snap ↗](#): a software packaging and deployment system developed by Canonical for operating systems that use the Linux kernel and the systemd init system. The packages are called "snaps", the command line tool for building snaps is called "Snapcraft", the central repository where snaps can be downloaded/installed is called the "Snap Store", and the daemon required to run snaps (download from the store, mount into place, confine, and run apps out of them) is called "snapd". The entire system is sometimes referred to as "snappy." Try running the command:
`snap install spotify` or `snap install postman`.
- [microk8s ↗](#): an open-source, low-ops, minimal production Kubernetes that automates deployment, scaling, and management of containerized apps. Follow the instructions to [Install MicroK8s on WSL2](#) ↗, check out the [Get Started Tutorial](#) ↗, or watch the video on [Kubernetes on Windows with MicroK8s and WSL 2](#) ↗.
- [systemctl ↗](#): a command-line utility used to control and inspect systemd and to help you interact with services on your Linux distribution. Try the command:
`systemctl list-units --type=service` to see which services are available and their status.

A few related tutorials demonstrating ways to use systemd:

- [Understanding and Using Systemd](#) ↗
- [Systemd Essentials: Working with the Services, Units, and the Journal](#) ↗
- [How To Sandbox Processes With Systemd On Ubuntu 20.04](#) ↗

How does enabling systemd affect WSL architecture?

Enabling support for systemd required changes to the WSL architecture. As systemd requires PID 1, the WSL init process started within the Linux distribution becomes a child process of the systemd. Because the WSL init process is responsible for providing the infrastructure for communication between the Linux and Windows components, changing this hierarchy required rethinking some of the assumptions made with the WSL init process. Additional modifications had to be made to ensure a clean shutdown (as that shutdown is controlled by systemd now) and to have compatibility with [WSLg](#), the component of WSL that runs Linux Graphical User Interfaces (GUIs), or the Linux apps that display in windows rather than the command line.

It is also important to note that with these changes, systemd services will NOT keep your WSL instance alive. Your WSL instance will stay alive in the same way it did previous to this update, which you can read more about in this [Background Task Support blog post from 2017 ↗](#).

Import any Linux distribution to use with WSL

Article • 02/14/2022

You can use any Linux distribution inside of the Windows Subsystem for Linux (WSL), even if it is not available in the [Microsoft Store](#), by importing it with a tar file.

This article shows how to import the Linux distribution, [CentOS](#), for use with WSL by obtaining its tar file using a Docker container. This process can be applied to import any Linux distribution.

Obtain a tar file for the distribution

First you'll need to obtain a tar file that contains all the Linux binaries for the distribution.

You can obtain a tar file in a variety of ways, two of which include:

- Download a provided tar file. You can find an example for Alpine in the "Mini Root Filesystem" section of the [Alpine Linux downloads](#) site.
- Find a Linux distribution container and export an instance as a tar file. The example below will show this process using the [CentOS container](#).

Obtaining a tar file for CentOS example

In this example, we'll use Docker inside of a WSL distribution to obtain the tar file for CentOS.

Prerequisites

- You must have [WSL enabled](#) with a Linux distribution installed running WSL 2.
- You must have [Docker Desktop for Windows](#) installed with the [WSL 2 engine enabled and integration checked](#). See the [Docker Desktop license agreement](#) for updates on the terms of use.

Export the tar from a container

1. Open the command line (Bash) for a Linux distribution that you've already installed from the Microsoft Store (Ubuntu in this example).

2. Start the Docker service:

```
Bash  
sudo service docker start
```

3. Run the CentOS container inside Docker:

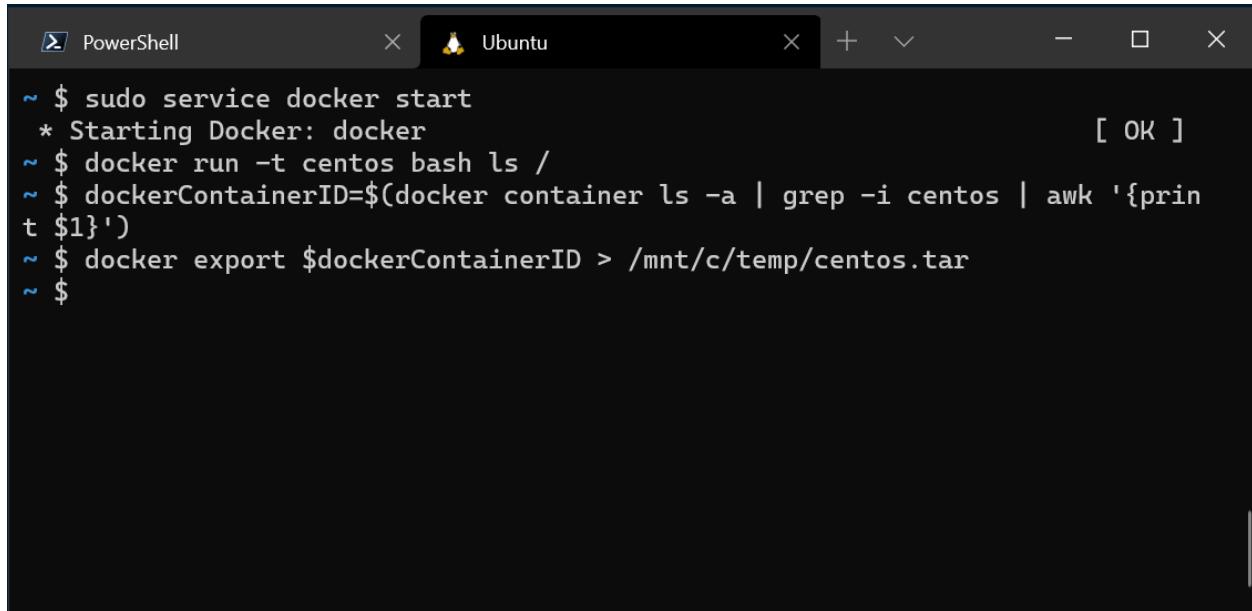
```
Bash  
docker run -t centos bash ls /
```

4. Grab the CentOS container ID using grep and awk:

```
Bash  
dockerContainerID=$(docker container ls -a | grep -i centos | awk  
'{print $1}')
```

5. Export the container ID to a tar file on your mounted c-drive:

```
Bash  
docker export $dockerContainerID > /mnt/c/temp/centos.tar
```



The screenshot shows a terminal window with two tabs: "PowerShell" and "Ubuntu". The "Ubuntu" tab is active and displays the following command history:

```
~ $ sudo service docker start
* Starting Docker: docker [ OK ]
~ $ docker run -t centos bash ls /
~ $ dockerContainerID=$(docker container ls -a | grep -i centos | awk '{print $1}')
~ $ docker export $dockerContainerID > /mnt/c/temp/centos.tar
~ $
```

This process exports the CentOS tar file from the Docker container so that we can now import it for use locally with WSL.

Import the tar file into WSL

Once you have a tar file ready, you can import it using the command: `wsl --import <Distro> <InstallLocation> <FileName>`.

Importing CentOS example

To import the CentOS distribution tar file into WSL:

1. Open PowerShell and ensure that you have a folder created where you'd like the distribution to be stored.

```
PowerShell

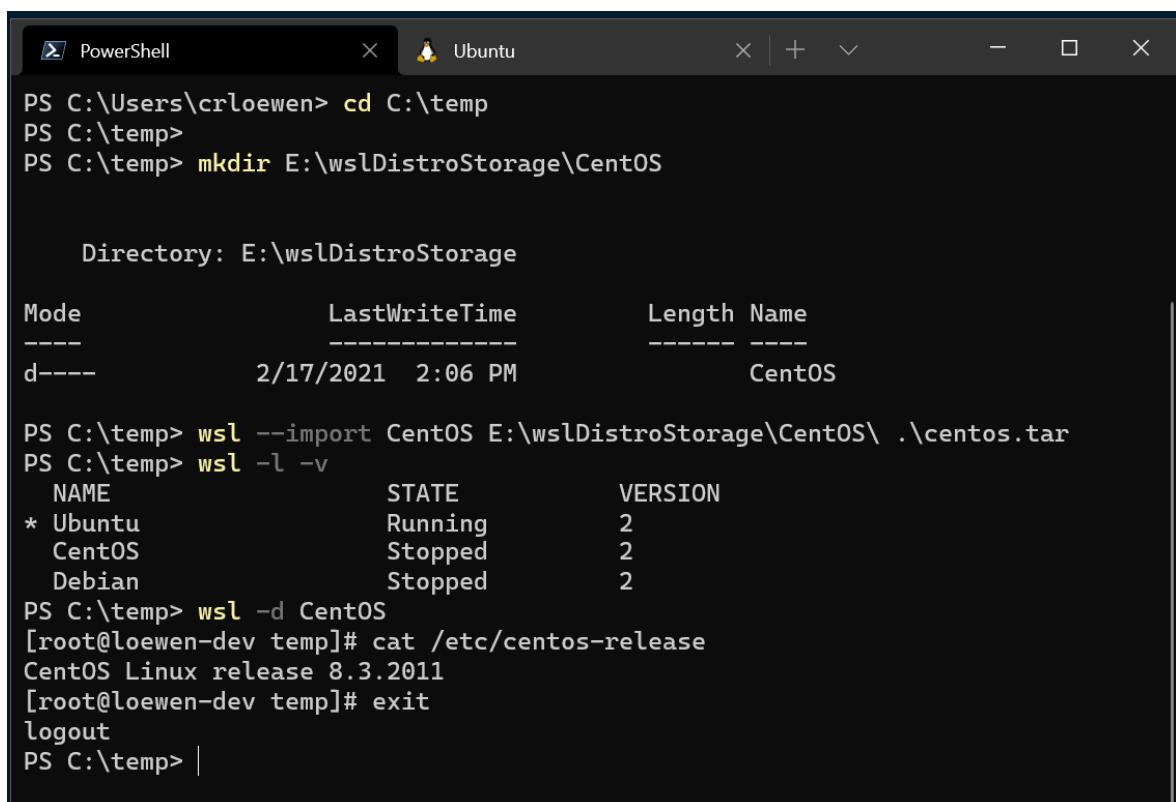
cd C:\temp
mkdir E:\wslDistroStorage\CentOS
```

2. Use the command `wsl --import <DistroName> <InstallLocation> <InstallTarFile>` to import the tar file.

```
PowerShell

wsl --import CentOS E:\wslDistroStorage\CentOS .\centos.tar
```

3. Use the command `wsl -l -v` to check which distributions you have installed.



```
PowerShell          X   Ubuntu
PS C:\Users\crloewen> cd C:\temp
PS C:\temp>
PS C:\temp> mkdir E:\wslDistroStorage\CentOS

Directory: E:\wslDistroStorage

Mode          LastWriteTime      Length Name
----          -----          ----
d---  2/17/2021  2:06 PM           CentOS

PS C:\temp> wsl --import CentOS E:\wslDistroStorage\CentOS\ .\centos.tar
PS C:\temp> wsl -l -v
  NAME          STATE      VERSION
* Ubuntu        Running       2
  CentOS        Stopped       2
  Debian        Stopped       2
PS C:\temp> wsl -d CentOS
[root@loewen-dev temp]# cat /etc/centos-release
CentOS Linux release 8.3.2011
[root@loewen-dev temp]# exit
logout
PS C:\temp> |
```

4. Finally, use the command `wsl -d CentOS` to run your newly imported CentOS Linux distribution.

Add WSL specific components like a default user

By default when using --import, you are always started as the root user. You can set up your own user account, but note that the set up process will vary slightly based on each different Linux distribution.

To set up user account with the CentOS distribution we just imported, first open PowerShell and boot into CentOS, using the command:

```
PowerShell
```

```
wsl -d CentOS
```

Next, open your CentOS command line. Use this command to install sudo and password setting tools into CentOS, create a user account, and set it as the default user. In this example, the username will be 'caloewen'.

ⓘ Note

You will want to add the username to the sudoers file so that enables the user to use sudo. The command `adduser -G wheel $myUsername` adds the user `myUsername` to the wheel group. Users in the wheel group are automatically granted sudo privileges and can perform tasks requiring elevated permission.

```
Bash
```

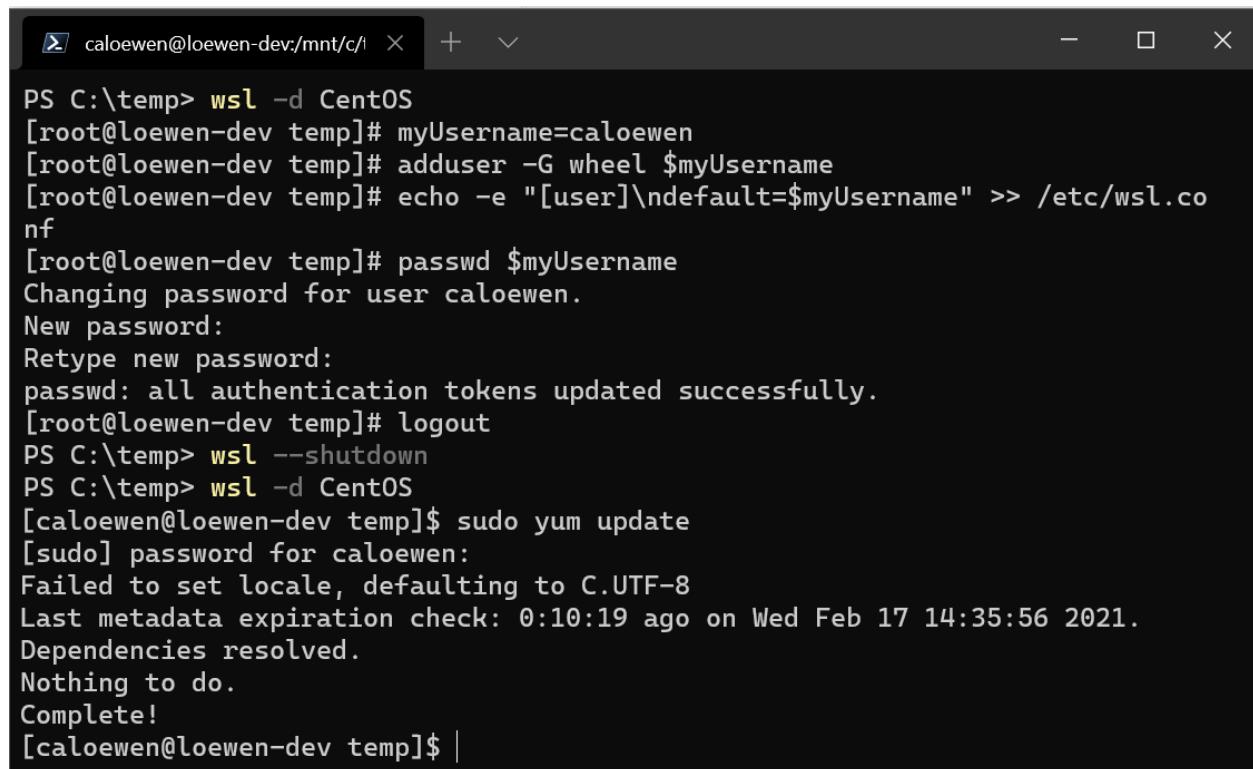
```
yum update -y && yum install passwd sudo -y  
myUsername=caloewen  
adduser -G wheel $myUsername  
echo -e "[user]\ndefault=$myUsername" >> /etc/wsl.conf  
passwd $myUsername
```

You must now quit out of that instance and ensure that all WSL instances are terminated. Start your distribution again to see your new default user by running this command in PowerShell:

```
PowerShell
```

```
wsl --terminate CentOS  
wsl -d CentOS
```

You will now see [caloewen@loewen-dev]\$ as the output based on this example.



A screenshot of a Windows terminal window titled "caloewen@loewen-dev:/mnt/c/" with a dark theme. The window contains a command-line session:

```
PS C:\temp> wsl -d CentOS
[root@loewen-dev temp]# myUsername=caloewen
[root@loewen-dev temp]# adduser -G wheel $myUsername
[root@loewen-dev temp]# echo -e "[user]\ndefault=$myUsername" >> /etc/wsl.conf
[root@loewen-dev temp]# passwd $myUsername
Changing password for user caloewen.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@loewen-dev temp]# logout
PS C:\temp> wsl --shutdown
PS C:\temp> wsl -d CentOS
[caloewen@loewen-dev temp]$ sudo yum update
[sudo] password for caloewen:
Failed to set locale, defaulting to C.UTF-8
Last metadata expiration check: 0:10:19 ago on Wed Feb 17 14:35:56 2021.
Dependencies resolved.
Nothing to do.
Complete!
[caloewen@loewen-dev temp]$ |
```

To learn more about configuring WSL settings, see [Configure settings with .wslconfig and wsl.conf](#).

Use a custom Linux distribution

You can create your own customized Linux distribution, packaged as a UWP app, that will behave exactly like the WSL distributions available in the Microsoft Store. To learn how, see [Creating a Custom Linux Distribution for WSL](#).

Creating a Custom Linux Distribution for WSL

Article • 09/28/2021

Use our open source WSL sample to build WSL distro packages for the Microsoft Store and/or to create custom Linux distro packages for sideloading. You can find the [distro launcher repo ↗](#) on GitHub.

This project enables:

- Linux distribution maintainers to package and submit a Linux distribution as an appx that runs on WSL
- Developers to create custom Linux distributions that can be sideloaded onto their dev machine

Background

We distribute Linux distros for WSL as UWP applications through the Microsoft Store. You can install those applications that will then run on WSL - the subsystem that sits in the Windows kernel. This delivery mechanism has many benefits as discussed in an [earlier blog post ↗](#).

Sideload a Custom Linux Distro Package

You can create a custom Linux distro package as an application to sideload on your personal machine. Please note that your custom package would not be distributed through the Microsoft Store unless you submit as a distribution maintainer. To set up your machine to sideload apps, you will need to enable this in the system settings under "For Developers". Be sure to either have developer mode, or sideload apps selected

For Linux Distro Maintainers

To submit to the Store, you will need to work with us to receive publishing approval. If you are a Linux distribution owner interested in adding your distribution to the Microsoft Store, please contact wslpartners@microsoft.com.

Getting Started

Follow the instructions on the [Distro Launcher GitHub repo](#) to create a custom Linux distro package.

Team Blogs

- [Open Sourcing a WSL Sample for Linux Distribution Maintainers and Sideload](#)
[Custom Linux Distributions](#)
- [Command-Line blog](#)

Provide Feedback

- [Distro Launcher GitHub repo](#)
- [GitHub issue tracker for WSL](#)

Mount a Linux disk in WSL 2

Article • 07/18/2023

If you want to access a Linux disk format that isn't supported by Windows, you can use WSL 2 to mount your disk and access its content. This tutorial will cover the steps to identify the disk and partition to attach to WSL2, how to mount them, and how to access them.

If you are connecting an external drive and do not have success with these mounting instructions, you may want to try the instructions to [Connect USB devices](#). The `wsl --mount` command does not currently support USB/flash drives/SD card readers, ([learn more about this issue ↗](#)).

ⓘ Note

Administrator access is required to attach a disk to WSL 2. The WSL 2 `mount` command does not support mounting a disk (or partitions that belong to the disk) that is currently in use. `wsl --mount` always attaches the entire disk even if only a partition is requested. You can't mount the Windows installation disk.

Prerequisites

You will need to be on Windows 11 Build 22000 or later, or be running the Microsoft Store version of WSL. To check your WSL and Windows version, use the command:

```
wsl.exe --version
```

Differences between mounting an external drive with Windows formatting versus Linux formatting

External drives formatted for Windows typically use the NTFS file system formatting. External drives formatted for Linux typically use the Ext4 file system formatting.

If you have mounted an NTFS-formatted drive on your Windows file system, you can access that drive from your Linux distribution using WSL by creating a mounted directory (`sudo mkdir /mnt/d`, replacing 'd' with whatever drive letter you'd like to use) and then using the `drvfs` file system interop plugin, with the command:

Bash

```
sudo mount -t drvfs D: /mnt/d
```

[Learn more about mounting scenarios ↗](#).

If you have an Ext4-formatted drive, you cannot mount it on your Windows file system. In order to mount an Ext4-formatted drive on your Linux distribution with WSL, you can use the `wsl --mount` command following the instructions below.

Mounting an unpartitioned disk

If you have a disk that doesn't have any partitions, you can mount it directly using the `wsl --mount` command. First you need to identify the disk.

1. **Identify the disk** - To list the available disks in Windows, run:

PowerShell

```
GET-CimInstance -query "SELECT * from Win32_DiskDrive"
```

The disks paths are available under the 'DeviceID' columns. Usually under the `\.\.\PHYSICALDRIVE*` format.

2. **Mount the disk** - Using PowerShell, you can mount the disk using the Disk path discovered above, run:

PowerShell

```
wsl --mount <DiskPath>
```

```

craig@craig-Alienware:/mnt/v ~ + - x
PS E:\wslDistroStorage\Ubuntu2004> GET-WMIOBJECT -query "SELECT * from Win32_DiskDrive"

Partitions : 1
DeviceID   : \\.\PHYSICALDRIVE0
Model      : Samsung SSD 970 EVO Plus 500GB
Size       : 500105249280
Caption    : Samsung SSD 970 EVO Plus 500GB

Partitions : 1
DeviceID   : \\.\PHYSICALDRIVE1
Model      : ST2000DM001-1CH164
Size       : 2000396321280
Caption    : ST2000DM001-1CH164

Partitions : 3
DeviceID   : \\.\PHYSICALDRIVE2
Model      : PM9A1 NVMe Samsung 256GB
Size       : 256052966400
Caption    : PM9A1 NVMe Samsung 256GB

Partitions : 0
DeviceID   : \\.\PHYSICALDRIVE3
Model      : Microsoft Virtual Disk
Size       : 322118415360
Caption    : Microsoft Virtual Disk

PS E:\wslDistroStorage\Ubuntu2004> wsl --mount \\.\PHYSICALDRIVE3
The disk \\.\PHYSICALDRIVE3 was successfully mounted under the name 'PHYSICALDRIVE3'. The mountpoint can be found under the path pointed to by the automount setting (default: /mnt/wsl).
To unmount and detach the disk, run 'wsl --unmount \\.\PHYSICALDRIVE3'.
PS E:\wslDistroStorage\Ubuntu2004> wsl
craig@craig-Alienware:/mnt/e/wslDistroStorage/Ubuntu2004$ cd /mnt/wsl/PHYSICALDRIVE3/
craig@craig-Alienware:/mnt/wsl/PHYSICALDRIVE3$ ls
bin  dev  home  lib  lib64  lost+found  mnt  proc  run  snap  sys  usr  wslHKjnMD  wslKEAFMJ  wslcnleED  wslolnend
boot etc  init  lib32  libx32  media  opt  root  sbin  srv  tmp  var  wslJInHfN  wslKFeigo  wslfCNNoM  wslpjNEik
craig@craig-Alienware:/mnt/wsl/PHYSICALDRIVE3$
```

Mounting a partitioned disk

If you have a disk that you aren't sure what file format it is in, or what partitions it has, you can follow the steps below to mount it.

1. Identify the disk - To list the available disks in Windows, run:

```

PowerShell
GET-CimInstance -query "SELECT * from Win32_DiskDrive"
```

The disks paths are listed after 'DeviceID', usually in the `\.\PHYSICALDRIVE*` format.

2. List and select the partitions to mount in WSL 2 - Once the disk is identified, run:

```

PowerShell
wsl --mount <DiskPath> --bare
```

This will make the disk available in WSL 2. (In the case of our example, the `<DiskPath>` is `\.\PHYSICALDRIVE*`.

3. Once attached, the partition can be listed by running the following command inside WSL 2:

```
Bash
```

```
lsblk
```

This will display the available block devices and their partitions.

Inside Linux, a block device is identified as `/dev/<Device><Partition>`. For example, `/dev/sdb3`, is the partition number 3 of disk `sdb`.

Example output:

```
Bash
```

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb      8:16   0    1G  0 disk 
└─sdb2   8:18   0   50M  0 part 
└─sdb3   8:19   0  873M  0 part 
└─sdb1   8:17   0  100M  0 part 
sdc      8:32   0  256G  0 disk /
sda      8:0    0  256G  0 disk
```

Identifying the filesystem type

If you don't know the type of filesystem of a disk or partition, you can use this command:

```
Bash
```

```
blkid <BlockDevice>
```

This will output the detected filesystem type (under the `TYPE="<Filesystem>"` format).

Mount the selected partitions

Once you have identified the partitions you want to mount, run this command on each partition:

```
PowerShell
```

```
wsl --mount <DiskPath> --partition <PartitionNumber> --type <Filesystem>
```

 Note

If you wish to mount the entire disk as a single volume (i.e. if the disk isn't partitioned), `--partition` can be omitted.

If omitted, the default filesystem type is "ext4".

Access the disk content

Once mounted, the disk can be accessed under the path pointed to by the config value: `automount.root`. The default value is `/mnt/wsl`.

From Windows, the disk can be accessed from File Explorer by navigating to: `\\\wsl$\\<Distro>\\<Mountpoint>` (pick any Linux distribution).

Unmount the disk

If you want to unmount and detach the disk from WSL 2, run:

```
PowerShell
```

```
wsl --unmount <DiskPath>
```

Mount a VHD in WSL

ⓘ Note

[WSL from the Microsoft Store](#) ↗ introduces a new argument to directly mount a VHD: `wsl --mount --vhd <pathToVHD>`

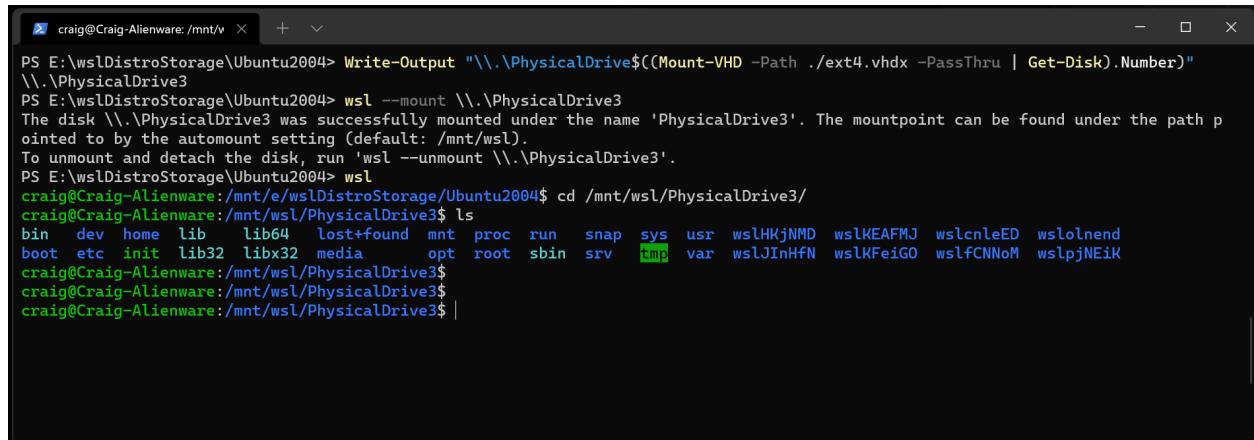
You can also mount virtual hard disk files (VHD) into WSL using `wsl --mount`. To do this, you first need to mount the VHD into Windows using the [Mount-VHD](#) command in Windows. Be sure to run this command with administrator privileges. Below is an example where we use this command, and also output the disk path. Be sure to replace `<pathToVHD>` with your actual VHD path.

```
PowerShell
```

```
Write-Output "\\.\PhysicalDrive$((Mount-VHD -Path <pathToVHD> -PassThru | Get-Disk).Number)"
```

You can use the output above to obtain the disk path for this VHD and mount that into WSL following the instructions in the previous section.

You can also use this technique to mount and interact with the virtual hard disks of other WSL distros, as each WSL 2 distro is stored via a virtual hard disk file called: `ext4.vhdx`. By default the VHDs for WSL 2 distros are stored in this path: `C:\Users\[user]\AppData\Local\Packages\[distro]\LocalState\[distroPackageName]`, please exercise caution accessing these system files, this is a power user workflow. Make sure to run `wsl --shutdown` before interacting with this disk to ensure the disk is not in use.



The screenshot shows a Windows terminal window with the following command history:

```
PS E:\wslDistroStorage\Ubuntu2004> Write-Output "\.\PhysicalDrive$((Mount-VHD -Path ./ext4.vhdx -PassThru | Get-Disk).Number)" \\.\PhysicalDrive3
PS E:\wslDistroStorage\Ubuntu2004> wsl --mount \\.\PhysicalDrive3
The disk \\.\PhysicalDrive3 was successfully mounted under the name 'PhysicalDrive3'. The mountpoint can be found under the path pointed to by the automount setting (default: /mnt/wsl).
To unmount and detach the disk, run 'wsl --unmount \\.\PhysicalDrive3'.
PS E:\wslDistroStorage\Ubuntu2004> wsl
craig@craig-Alienware:/mnt/e/wslDistroStorage/Ubuntu2004$ cd /mnt/wsl/PhysicalDrive3/
craig@craig-Alienware:/mnt/wsl/PhysicalDrive3$ ls
bin dev home lib lib64 lost+found mnt proc run snap sys usr wslHKjNMD wslKEAFMJ wslcnleED wslolnend
boot etc init lib32 libx32 media opt root sbin srv tmp var wslJInHFN wslKFeiGO wslfCNNoM wslpjNEiK
craig@craig-Alienware:/mnt/wsl/PhysicalDrive3$ 
craig@craig-Alienware:/mnt/wsl/PhysicalDrive3$ 
craig@craig-Alienware:/mnt/wsl/PhysicalDrive3$ |
```

Command line reference

Mounting a specific filesystem

By default, WSL 2 will attempt to mount the device as ext4. To specify another filesystem, run:

PowerShell

```
wsl --mount <DiskPath> -t <FileSystem>
```

For example, to mount a disk as fat, run:

```
wsl --mount <Diskpath> -t vfat
```

ⓘ Note

To list the available filesystems in WSL2, run: `cat /proc/filesystems`

When a disk has been mounted via WSL2 (Linux file system), it is no longer

available to mount via an ext4 driver on the Windows file system.

Mounting a specific partition

By default, WSL 2 attempts to mount the entire disk. To mount a specific partition, run:

```
wsl --mount <Diskpath> -p <PartitionIndex>
```

This only works if the disk is either MBR (Master Boot Record) or GPT (GUID Partition Table). [Read about partition styles - MBR and GPT](#).

Specifying mount options

To specify mount options, run:

PowerShell

```
wsl --mount <DiskPath> -o <MountOptions>
```

Example:

PowerShell

```
wsl --mount <DiskPath> -o "data=ordered"
```

ⓘ Note

Only filesystem specific options are supported at this time. Generic options such as `ro`, `rw`, `noatime`, ... are not supported.

Attaching the disk without mounting it

If the disk scheme isn't supported by any of the above options, you can attach the disk to WSL 2 without mounting it by running:

PowerShell

```
wsl --mount <DiskPath> --bare
```

This will make the block device available inside WSL 2 so it can be mounted manually from there. Use `lsblk` to list the available block devices inside WSL 2.

Specifying the mount name

ⓘ Note

This option is only available with [WSL from the Microsoft Store](#) ↗

By default the mountpoint name is generated based on the physical disk or VHD name. This can be overridden with `--name`. Example:

PowerShell

```
wsl --mount <DiskPath> --name myDisk
```

Detaching a disk

To detach a disk from WSL 2, run:

PowerShell

```
wsl --unmount [DiskPath]
```

If `Diskpath` is omitted, all attached disks are unmounted and detached.

ⓘ Note

If one disk fails to unmount, WSL 2 can be forced to exit by running `wsl --shutdown`, which will detach the disk.

Limitations

- At this time, only entire disks can be attached to WSL 2, meaning that it's not possible to attach only a partition. Concretely, this means that it's not possible to use `wsl --mount` to read a partition on the boot device, because that device can't be detached from Windows.

- Only filesystems that are natively supported in the kernel can be mounted by `wsl --mount`. This means that it's not possible to use installed filesystem drivers (such as ntfs-3g for example) by calling `wsl --mount`.
- Filesystems not directly supported by the kernel can be mounted via a `--bare` attach and then invoking the relevant FUSE driver.

 Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Connect USB devices

Article • 02/02/2024

This guide will walk through the steps necessary to connect a USB device to a Linux distribution running on WSL 2 using the USB/IP open-source project, [usbipd-win](#).

Setting up the USB/IP project on your Windows machine will enable common developer USB scenarios like flashing an Arduino or accessing a smartcard reader.

Prerequisites

- Running Windows 11 (Build 22000 or later). (*Windows 10 support is possible, see note below*).
- A machine with an x64 processor is required. (x86 and Arm64 are currently not supported with usbipd-win).
- WSL is [installed and set up with the latest version](#).
- Linux distribution installed and [set to WSL 2](#).

ⓘ Note

To check your Windows version and build number, select **Windows logo key + R**, type `winver`, select **OK**. You can update to the latest Windows version by selecting **Start > Settings > Windows Update > Check for updates**. To check your Linux kernel version, open your Linux distribution and enter the command: `uname -a`. To manually update to the latest kernel, open PowerShell and enter the command: `'wsl --update'`.

ⓘ Important

WSL now supports both Windows 10 and Windows 11 via the Microsoft Store, meaning that Windows 10 users now have access to the latest kernel versions without needing to compile from source. See [WSL in the Microsoft Store is now generally available on Windows 10 and 11](#) for info on how to update to the Store-supported version of WSL. If you are unable to update to the Store-supported version of WSL and automatically receive kernel updates, see the [USBIPD-WIN project repo](#) for instructions on connecting USB devices to a Linux distribution running on WSL 2 by building your own USBIP enabled WSL 2 kernel.

Install the USBIPD-WIN project

Support for connecting USB devices is not natively available in WSL, so you will need to install the open-source usbipd-win project.

1. Go to the [latest release page for the usbipd-win project](#).
2. Select the .msi file, which will download the installer. (You may get a warning asking you to confirm that you trust this download).
3. Run the downloaded usbipd-win_x.msi installer file.

ⓘ Note

Alternatively, you can also install the usbipd-win project using **Windows Package Manager** (winget). If you have already installed winget, just use the command:

```
winget install --interactive --exact dorsel.usbipd-win
```

 to install usbipd-win. If you leave out --interactive, winget may immediately restart your computer if that is required to install the drivers.

This will install:

- A service called `usbipd` (display name: USBIP Device Host). You can check the status of this service using the Services app from Windows.
- A command line tool `usbipd`. The location of this tool will be added to the PATH environment variable.
- A firewall rule called `usbipd` to allow all local subnets to connect to the service. You can modify this firewall rule to fine tune access control.

Attach a USB device

Before attaching your USB device, ensure that a WSL command line is open. This will keep the WSL 2 lightweight VM active.

ⓘ Note

This doc assumes that you have [usbipd-win 4.0.0 or higher installed](#)

1. List all of the USB devices connected to Windows by opening PowerShell in *administrator* mode and entering the following command. Once the devices are listed, select and copy the bus ID of the device you'd like to attach to WSL.

```
PowerShell
```

```
usbipd list
```

2. Before attaching the USB device, the command `usbipd bind` must be used to share the device, allowing it to be attached to WSL. This requires administrator privileges. Select the bus ID of the device you would like to use in WSL and run the following command. After running the command, verify that the device is shared using the command `usbipd list` again.

```
PowerShell
```

```
usbipd bind --busid 4-4
```

3. To attach the USB device, run the following command. (You no longer need to use an elevated administrator prompt.) Ensure that a WSL command prompt is open in order to keep the WSL 2 lightweight VM active. Note that as long as the USB device is attached to WSL, it cannot be used by Windows. Once attached to WSL, the USB device can be used by any distribution running as WSL 2. Verify that the device is attached using `usbipd list`. From the WSL prompt, run `lsusb` to verify that the USB device is listed and can be interacted with using Linux tools.

```
PowerShell
```

```
usbipd attach --wsl --busid <busid>
```

4. Open Ubuntu (or your preferred WSL command line) and list the attached USB devices using the command:

```
Bash
```

```
lsusb
```

You should see the device you just attached and be able to interact with it using normal Linux tools. Depending on your application, you may need to configure udev rules to allow non-root users to access the device.

5. Once you are done using the device in WSL, you can either physically disconnect the USB device or run this command from PowerShell:

```
PowerShell
```

```
usbipd detach --busid <busid>
```

To learn more about how this works, see the [Windows Command Line Blog](#) and the [usbipd-win repo on GitHub](#).

For a video demonstration, see [WSL 2: Connect USB devices \(Tabs vs Spaces show\)](#).

 Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

-  [Open a documentation issue](#)
-  [Provide product feedback](#)

Adjust case sensitivity

Article • 04/27/2022

Case sensitivity determines whether uppercase (FOO.txt) and lowercase (foo.txt) letters are handled as distinct (case-sensitive) or equivalent (case-insensitive) in a file name or directory.

- Case-sensitive: FOO.txt ≠ foo.txt ≠ Foo.txt
- Case-insensitive: FOO.txt = foo.txt = Foo.txt

Differences between Windows and Linux case sensitivity

When working with both Linux and Windows files and directories, you may need to adjust how case sensitivity is handled.

Standard behavior:

- Windows file system treats file and directory names as case-insensitive. FOO.txt and foo.txt will be treated as equivalent files.
- Linux file system treats file and directory names as case-sensitive. FOO.txt and foo.txt will be treated as distinct files.

The Windows file system supports setting case sensitivity with attribute flags per directory. While the standard behavior is to be case-insensitive, you can assign an attribute flag to make a directory case sensitive, so that it will recognize Linux files and folders that may differ only by case.

This may be especially true when mounting drives to the Windows Subsystem for Linux (WSL) file system. When working in the WSL file system, you are running Linux, thus files and directories are treated as case-sensitive by default.

Note

In the past, if you had files whose name differed only by case, these files could not be accessed by Windows, because Windows applications treat the file system as case insensitive and cannot distinguish between files whose names only differ in case. While Windows File Explorer will show both files, only one will open regardless of which you select.

Change the case sensitivity of files and directories

The following steps explain how to change a directory on the Windows file system so that it is case-sensitive and will recognize files and folders that differ only by case.

⚠️ Warning

Some Windows applications, using the assumption that the file system is case insensitive, don't use the correct case to refer to files. For example, it's not uncommon for applications to transform filenames to use all upper or lower case. In directories marked as case sensitive, this means that these applications can no longer access the files. Additionally, if Windows applications create new directories in a directory tree where you are using case sensitive files, these directories are not case sensitive. This can make it difficult to work with Windows tools in case sensitive directories, so exercise caution when changing Windows file system case-sensitivity settings.

Inspect current case sensitivity

To check if a directory is case sensitive in the Windows filesystem, run the command:

```
PowerShell
```

```
fsutil.exe file queryCaseSensitiveInfo <path>
```

Replace `<path>` with your file path. For a directory in the Windows (NTFS) file system, the `<path>` will look like: `c:\Users\user1\case-test` or if you are already in the `user1` directory, you could just run: `fsutil.exe file setCaseSensitiveInfo case-test`

Modify case sensitivity

Support for per-directory case sensitivity began in Windows 10, build 17107. In Windows 10, build 17692, support was updated to include inspecting and modifying the case sensitivity flag for a directory from inside WSL. Case sensitivity is exposed using an extended attribute named `system.wsl_case_sensitive`. The value of this attribute will be 0 for case insensitive directories, and 1 for case sensitive directories.

Changing the case-sensitivity of a directory requires that you run **elevated permissions** (run as Administrator). Changing the case-sensitivity flag also requires "Write attributes",

“Create files”, “Create folders” and “Delete subfolders and files” permissions on the directory. [See the troubleshooting section for more about this.](#)

To change a directory in the Windows file system so that it is case-sensitive (FOO ≠ foo), run PowerShell as Administrator and use the command:

```
PowerShell
```

```
fsutil.exe file setCaseSensitiveInfo <path> enable
```

To change a directory in the Windows file system back to the case-insensitive default (FOO = foo), run PowerShell as Administrator and use the command:

```
PowerShell
```

```
fsutil.exe file setCaseSensitiveInfo <path> disable
```

A directory must be empty in order to change the case sensitivity flag attribute on that directory. You cannot disable the case sensitivity flag on a directory containing folders/files whose names differ on only by case.

Case sensitivity inheritance

When creating new directories, those directories will inherit the case sensitivity from its parent directory.

Warning

There is an exception to this inheritance policy when running in WSL 1 mode. When a distribution is running in WSL 1 mode, the per-directory case sensitivity flag is not inherited; directories created in a case sensitive directory are not automatically case sensitive themselves. You must explicitly mark each directory as case sensitive

Case sensitivity options for mounting a drive in WSL configuration file

Case sensitivity can be managed when mounting a drive on the Windows Subsystem for Linux using the WSL config file. Each Linux distribution that you have installed can have its own WSL config file, called `/etc/wsl.conf`. For more information about how to mount a drive, see [Get started mounting a Linux disk in WSL 2](#).

To configure the case sensitivity option in the `wsl.conf` file when mounting a drive:

1. Open the Linux distribution you will be using (ie. Ubuntu).
2. Change directories up until you see the `etc` folder (this may require you to `cd ..` up from the `home` directory).
3. List the files in the `etc` directory to see if a `wsl.conf` file already exists (use the `ls` command, or `explorer.exe .` to view the directory with Windows File Explorer).
4. If the `wsl.conf` file does not already exist, you can create it using: `sudo touch wsl.conf` or by running `sudo nano /etc/wsl.conf`, which will create the file upon saving from the Nano editor.
5. The following options are available for you to add into your `wsl.conf` file:

Default setting: `dir` for enabling case sensitivity per directory.

Bash

```
[automount]
options = case = dir
```

Case sensitivity unavailable (all directories on mounted NTFS drives will be case insensitive): `off`

Bash

```
[automount]
options = case = off
```

Treat all directories on the (NTFS) drive as case sensitive: `force`

Bash

```
[automount]
options = case = force
```

This option is only supported for mounting drives on Linux distributions running as WSL 1 and may require a registration key. To add a registration key, you can use this command from an elevated (admin) command prompt: `reg.exe add`

```
HKLM\SYSTEM\CurrentControlSet\Services\lxss /v DrvFsAllowForceCaseSensitivity /t
REG_DWORD /d 1.
```

You will need to restart WSL after making any changes to the `wsl.conf` file in order for those changes to take effect. You can restart WSL using the command: `wsl --shutdown`

💡 Tip

To mount a drive (which uses the DrvFs filesystem plugin to make the disk available under /mnt, such as /mnt/c, /mnt/d, etc) with a specific case sensitivity setting for ALL drives, use `/etc/wsl.conf` as described above. To set the default mount options for one specific drive, use the [/etc/fstab file](#) to specify these options. For more WSL configuration options, see [Configure per distro launch settings with wslconf](#).

Changing the case sensitivity on a drive mounted to a WSL distribution

NTFS-formatted drives mounted to a WSL distribution will be case-insensitive by default. To change the case sensitivity for a directory on a drive mounted to a WSL distribution (ie. Ubuntu), follow the same steps as listed above for the Windows file system. (EXT4 drives will be case-sensitive by default).

To enable case-sensitivity on a directory (FOO ≠ foo), use the command:

Bash

```
fsutil.exe file setCaseSensitiveInfo <path> enable
```

To disable case-sensitivity on a directory and return to the case-insensitive default (FOO = foo), use the command:

Bash

```
fsutil.exe file setCaseSensitiveInfo <path> disable
```

⚠ Note

If you change the case sensitive flag on an existing directory for a mounted drive while WSL is running, ensure WSL has no references to that directory or else the change will not be effective. This means the directory must not be open by any WSL processes, including using the directory (or its descendants) as the current working directory.

Configure case sensitivity with Git

The Git version control system also has a configuration setting that can be used to adjust case sensitivity for the files you are working with. If you are using Git, you may want to adjust the [git config core.ignorecase](#) setting.

To set Git to be case-sensitive (FOO.txt ≠ foo.txt), enter:

```
git config core.ignorecase false
```

To set Git to be case-insensitive (FOO.txt = foo.txt), enter:

```
git config core.ignorecase true
```

Setting this option to false on a case-insensitive file system may lead to confusing errors, false conflicts, or duplicate files.

For more information, see the [Git Config documentation](#).

Troubleshooting

My directory has files that are mixed case and require case sensitivity but Windows FS tools will not recognize these files

To use Windows file system tools to work on a Linux directory that contains mixed case files, you will need to create a brand new directory and set it to be case-sensitive, then copy the files into that directory (using git clone or untar). The files will remain mixed case. (Note that if you have already tried moving the files to a case-insensitive directory and there were conflicts, there were likely some files that were overwritten and will no longer be available.)

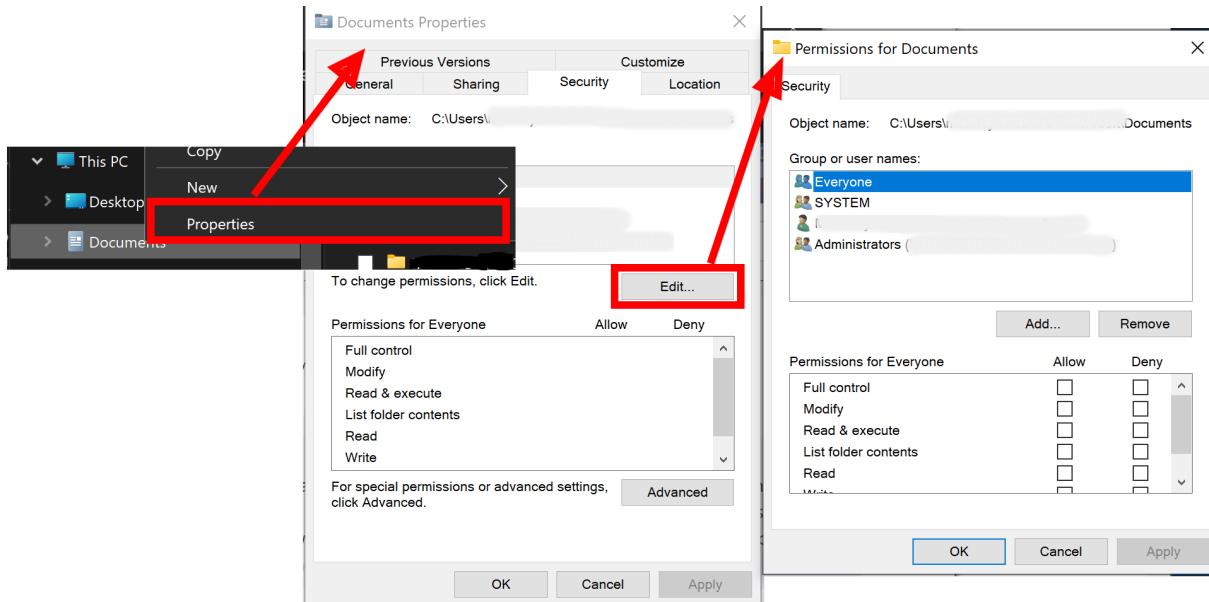
Error: The directory is not empty

You cannot change the case sensitivity setting on a directory that contains other files or directories. Try creating a new directory, changing the setting, then copying your mixed-case files into it.

Error: Access denied

Ensure that you have the “Write attributes”, “Create files”, “Create folders” and “Delete subfolders and files” permissions on the directory required for changing case-sensitivity. To check these settings, open the directory in Windows File Explorer (from command

line, use the command: `explorer.exe .`). Right-click the directory and select **Properties** to open the Document Properties window, then select **Edit** to view or change permissions for the directory.



Error: A local NTFS volume is required for this operation

The case sensitivity attribute can only be set on directories within an NTFS-formatted file system. Directories in the WSL (Linux) file system are case sensitive by default (and cannot be set to be case insensitive using the `fsutil.exe` tool).

Additional resources

- DevBlog: Per-directory case sensitivity and WSL ↗
- DevBlog: Improved per-directory case sensitivity support in WSL ↗

How to manage WSL disk space

Article • 11/10/2023

This guide covers how to manage the disk space used by Linux distributions installed using WSL 2, including:

- [How to check the amount of disk space available in the VHD](#)
- [How to expand the size of the VHD](#)
- [How to repair the VHD if an error occurs](#)
- [How to locate the .vhdx file and disk path for any installed Linux distributions](#)

Windows Subsystem for Linux (WSL 2) uses a virtualization platform to install Linux distributions alongside the host Windows operating system, creating a Virtual Hard Disk (VHD) to store files for each of the Linux distributions that you install. These VHDs use the [ext4 file system type](#) and are represented on your Windows hard drive as an `ext4.vhdx` file.

WSL 2 automatically resizes these VHD files to meet storage needs. By default each VHD file used by WSL 2 is initially allocated a 1TB maximum amount of disk space (prior to [WSL release 0.58.0](#) this default was set to a 512GB max and 256GB max prior to that).

If the storage space required by your Linux files exceeds this maximum size, you will see errors stating that you've run out of disk space. To fix this error, follow the guidance below on [How to expand the size of your WSL 2 Virtual Hard Disk](#).

How to check your available disk space

Check the amount of disk space available in the VHD for a Linux distribution installed with WSL 2 by using the Linux `df` command.

To check available disk space, open a PowerShell command line and enter this command (replacing `<distribution-name>` with the actual distribution name):

```
PowerShell

wsl.exe --system -d <distribution-name> df -h /mnt/wslg/distro
```

If this command does not work for you please upgrade to the Store version of WSL using the `wsl --update` command, or try `wsl df -h /`.

The output will include:

- **Filesystem:** Identifier for the VHD file system
- **Size:** Total size of the disk (the maximum amount of space allocated to the VHD)
- **Used:** Amount of space currently being used in the VHD
- **Avail:** Amount of space left in the VHD (Allocated size minus amount used)
- **Use%:** Percentage of disk space remaining (Used / Allocated size)
- **Mounted on:** Directory path where the disk is mounted

If you see that you are near to reaching the available amount of disk space allocated to your VHD, or have already received an error due to no disk space remaining, see the next section for steps on how to expand the maximum amount of disk space allocated to the VHD associated with your Linux distribution. The amount of disk space allocated to your VHD by WSL will always show the default maximum amount (1TB in the most recent version of WSL), even if the amount of disk space on your actual Windows device is less than that. WSL mounts a VHD that will expand in size as you use it, so your Linux distribution sees that it can grow to the allocated maximum size of 1TB.

How to expand the size of your WSL 2 Virtual Hard Disk

To expand the VHD size for a Linux distribution beyond the **default 1TB maximum** amount of allocated disk space, follow the steps below. (*For earlier WSL releases that have not yet been updated, this max default may be set to 512GB or 256GB*).

1. Terminate all WSL instances using the command: `wsl.exe --shutdown`
2. Copy the directory path to the `ext4.vhdx` file associated with the Linux distribution installed on your machine. For help, see [How to locate the vhdx file and disk path for your Linux distribution](#).
3. Open Windows Command Prompt with admin privileges and then open the `diskpart` command interpreter by entering:

Windows Command Prompt

`diskpart`

4. You will now have a `DISKPART>` prompt. Enter the following command, replacing `<pathToVHD>` with the directory path to the `ext4.vhdx` file associated with the Linux distribution (copied in step #2).

Windows Command Prompt

```
Select vdisk file=<pathToVHD>"
```

5. Display the details associated with this virtual disk, including the **Virtual size**, representing the current maximum size the VHD is allocated:

```
Windows Command Prompt
```

```
detail vdisk
```

6. You will need to convert the **Virtual size** to megabytes. For example, if **Virtual size: 512 GB**, this is equal to **512000 MB**. The new value you enter must be greater than this original value. To double the virtual size of 512 GB to 1024 GB, you would enter the value in MB as: **1024000**. Be careful not to enter a value higher than you actually want as the process of reducing a virtual disk size is much more complicated.
7. Enter the value for the new maximum size you want to allocate to this Linux distribution using the Windows Command Prompt **DISKPART** prompt:

```
Windows Command Prompt
```

```
expand vdisk maximum=<sizeInMegaBytes>
```

8. Exit the **DISKPART** prompt:

```
Windows Command Prompt
```

```
exit
```

9. Launch this Linux distribution. (*Ensure it is running in WSL 2. You can confirm this using the command: `wsl.exe -L -v`. WSL 1 is not supported*).

10. Make WSL aware that it can expand the file system size for this distribution by running these commands from your WSL distribution command line. You may see this message in response to the first **mount** command: "/dev: none already mounted on /dev." This message can safely be ignored.

```
Bash
```

```
sudo mount -t devtmpfs none /dev  
mount | grep ext4
```

11. Copy the name of this entry, which will look like: `/dev/sdX` (with the X representing any other character). In the following example the value of X is **b**:

```
Bash
```

```
sudo resize2fs /dev/sdb <sizeInMegabytes>M
```

Using the example from above, we changed the vhd size to **2048000**, so the command would be: `sudo resize2fs /dev/sdb 2048000M`.

ⓘ Note

You may need to install **resize2fs**. If so, you can use this command to install it: `sudo apt install resize2fs`.

The output will look similar to the following:

```
Bash
```

```
resize2fs 1.44.1 (24-Mar-2021)
Filesystem at /dev/sdb is mounted on /; on-line resizing required
old_desc_blocks = 32, new_desc_blocks = 38
The filesystem on /dev/sdb is now 78643200 (4k) blocks long.
```

The virtual drive (ext4.vhdx) for this Linux distribution has now successfully been expanded to the new size.

ⓘ Important

We recommend that you do not modify, move, or access the WSL related files located inside of your `AppData` folder using Windows tools or editors. Doing so could cause your Linux distribution to become corrupted. If you would like to access your Linux files from Windows, that is possible via the path `\\\wsl$\<distribution-name>\`. Open your WSL distribution and enter `explorer.exe .` to view that folder. To learn more, see the blog post: [Accessing Linux files from Windows](#).

How to repair a VHD mounting error

If you encounter an error related to "mounting the distribution disk", this could be due to a sudden shutdown or power outage and may result in the Linux distribution VHD being switched to *read-only* to avoid data loss. You can repair and restore the distribution using the `e2fsck` Linux command by following the steps below.

Use the `lsblk` command to identify the block device name

When WSL 2 installs a Linux distribution, it is mounting the distribution as a Virtual Hard Disk (VHD) with its own file system. Linux refers to these hard drives as "block devices" and you can view information about them by using the `lsblk` command.

To find the names of the block devices currently being used by WSL 2, open your distribution and enter the command: `lsblk`. (Or open PowerShell and enter the command: `wsl.exe lsblk`.) The output will look something like this:

```
Bash

NAME MAJ:MIN RM    SIZE RO TYPE MOUNTPOINTS
sda      8:0     0 363.1M  1 disk
sdb      8:16    0     8G  0 disk [SWAP]
sdc      8:32    0   1.5T  0 disk
sdd      8:48    0     1T  0 disk /mnt/wslg/distro
```

Information about the block device includes:

- **NAME:** The name assigned to the device will be `sd[a-z]`, referring to the SCSI Disk with a letter designation for each disk being used. `sda` is always the system distribution.
- **MAJ:MIN:** Represents numbers used by the Linux kernel to internally identify the devices with the first number representing the device type (8 is used for Small Computer System Interface/SCSI disks).
- **RM:** Let's us know if the device is removable (1) or not (0).
- **SIZE:** Total size of the volume.
- **RO:** Let's us know if the device is read-only (1) or not (0).
- **TYPE:** Refers to the device type (disk in this case).
- **MOUNTPOINTS:** Refers to the current directory on the files system where the block device is located (SWAP is for preconfigured inactive memory so no mountpoint).

Read-only fallback error

If WSL encounters a "mounting error" when opening a Linux distribution, the distribution may be set as read-only as a fallback. If that happens, the distribution may display the following error during startup:

PowerShell

```
An error occurred mounting the distribution disk, it was mounted read-only as a fallback.
```

When a distribution is started as read-only, any attempts to write to the filesystem will fail with an error like this:

Bash

```
$ touch file  
touch: cannot touch 'file': Read-only file system
```

To repair a disk mount error in WSL, and restore it back to a usable / writeable state again, you can use the `wsl.exe --mount` command to re-mount the disk with the following steps:

1. Shutdown all WSL distributions by opening PowerShell and entering the command:

PowerShell

```
wsl.exe --shutdown
```

2. Open PowerShell as administrator (in an elevated command prompt) and enter the mount command, replacing `<path-to-ext4.vhdx>` with the path to the distribution's .vhdx file. For help locating this file, see [How to locate the VHD file and disk path for your Linux distribution](#).

PowerShell

```
wsl.exe --mount <path-to-ext4.vhdx> --vhd --bare
```

3. Use the `wsl.exe lsblk` command from PowerShell to identify the block device name for the distribution (sd[a-z]) and then enter the following command to repair the disk (replacing `<device>` with the correct block device name, like "sdc"). The `e2fsck` command checks ext4 file systems (the type used by distributions installed with WSL) for errors and repairs them accordingly.

PowerShell

```
wsl.exe sudo e2fsck -f /dev/<device>
```

⚠ Note

If you only have a single Linux distribution installed, you may encounter an "ext file in use" error and will need to install an additional distribution in order to run `wsl.exe lsblk`. You can uninstall the distribution once the repair is complete.

4. Once the repair is complete, unmount the disk in PowerShell by entering:

```
PowerShell  
wsl.exe --unmount
```

⚠ Warning

You can use the command: `sudo mount -o remount,rw /` to return a read-only distribution to a usable/writable state, but all changes will be in-memory and so will be lost when the distribution is restarted. We recommend using the steps listed above to mount and repair the disk instead.

How to locate the .vhdx file and disk path for your Linux distribution

To locate the `.vhdx` file and directory path for a Linux distribution, open PowerShell and use the following script, replacing `<distribution-name>` with the actual distribution name:

```
PowerShell  
(Get-ChildItem -Path HKCU:\Software\Microsoft\Windows\CurrentVersion\Lxss |  
Where-Object { $_.GetValue("DistributionName") -eq '<distribution-name>' })  
.GetValue("BasePath") + "\ext4.vhdx"
```

The result will display a path looking something like `%LOCALAPPDATA%\Packages\<PackageFamilyName>\LocalState\<disk>.vhdx`. For example:

```
PowerShell
```

```
C:\Users\User\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_7  
9rhkp1fndgsc\LocalState\ext4.vhdx
```

This is the path to the `ext4.vhdx` file associated with the Linux distribution that you listed.

 [Collaborate with us on GitHub](#)

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

WSL Plugins

Article • 11/15/2023

Windows applications can now create and interact with Linux processes running inside of the Windows Subsystem for Linux (WSL) with WSL plugins. This article gives an overview of how they work, and how to get started using them.

Understanding Plugin functionality

WSL plugins provide these core functionalities:

- Allows applications to specify a Windows executable that starts when the WSL virtual machine is started
- The Windows executable can create Linux processes inside of WSL, and it can communicate directly to them using a virtualized socket

Using these, Windows applications can build ontop of WSL experiences and provide additional functionality related to the Windows Subsystem for Linux.

Installing a Plugin

As a WSL plugin creator, you can install your plugin on a machine by setting a registry key to point to your plugin's DLL file.

And as a WSL user, any application that you use will automatically install WSL plugins as part of their normal install process.

Creating your own Plugin

To start a plugin project you will need to build a Win32 dll. The simplest way to get set up with this is to try our WSL plugin sample project. You can do this by cloning the [WSL plugin sample repository](#) to a local folder with `git clone` and open it in Visual Studio.

When you open the project please navigate to the `dllmain.cpp` file (<https://github.com/microsoft/wsl-plugin-sample/blob/main/plugin.cpp>) and you will see the list of functions available to WSL plugins.

You can then press the "Build" tab and build your project, which will output a DLL ready you to use, likely under `wsl-plugin-sample\x64\Debug\WSLPluginSample.dll`.

Testing your Plugin

WSL plugins will only run if they are [digitally signed](#). To test this you will need to enable test signing on your machine.

Enabling test signing and creating a test certification

Open an elevated PowerShell Window and [enable test signing](#) by running this command:

PowerShell

```
## If this command results in "The value is protected by Secure Boot policy  
and cannot be modified or deleted"  
## Then reboot the PC, go into BIOS settings, and disable Secure Boot.  
## BitLocker may also affect your ability to modify this setting.  
Bcdedit.exe -set TESTSIGNING ON
```

Once test signing is enabled (A reboot may required), in an elevated Powershell command prompt that is at the directory of your WSLPluginSample.dll file created above we will create a WSL test certificate:

PowerShell

```
# Create the cert  
$certname = "WSLPluginTestCert"  
$cert = New-SelfSignedCertificate -Subject "CN=$certname" -CertStoreLocation  
"Cert:\CurrentUser\My" -KeyExportPolicy Exportable -KeySpec Signature -  
KeyLength 2048 -KeyAlgorithm RSA -HashAlgorithm SHA256 -Type CodeSigningCert  
  
# Export it to a local path  
Export-Certificate -Cert $cert -FilePath ".\$certname.cer"  
  
# Sign the DLL file  
Set-AuthenticodeSignature -FilePath "C:\dev\Path\To\Your\WSLPlugin.dll" -  
Certificate $cert
```

Last import the certificate to the Trusted Root Certification Authority:

PowerShell

```
certutil -addstore "Root" ".\$certname.cer"
```

See the [how to create a self signed certificate](#) docs page for more info.

Install the plugin

In the same elevated PowerShell window, run the command below to install the plugin, and be sure to change the path to the plugin DLL to your existing path:

```
PowerShell
```

```
Reg.exe add  
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Lxss\Plugins /v  
demo-plugin /t REG_SZ /d C:\Path\to\plugin.dll /f
```

To use the plugin, restart the wsl service via:

```
PowerShell
```

```
sc.exe stop wslservice  
wsl.exe echo "test"
```

Your plugin should now be loaded. See the [Troubleshooting and additional information](#) section for more information if the plugin failed to load.

And then when you are finished, you can run this command to remove the plugin (Please keep in mind you will need to restart the WSL service for it to take effect):

```
PowerShell
```

```
Reg.exe delete  
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Lxss\Plugins /v  
demo-plugin
```

Troubleshooting and additional information

Common error codes:

- Wsl/Service/CreateInstance/CreateVm/Plugin/ERROR_MOD_NOT_FOUND -> The plugin DLL could not be loaded. Check that the plugin registration path is correct
- Wsl/Service/CreateInstance/CreateVm/Plugin/TRUST_E_NOSIGNATURE -> The plugin DLL is not signed, or its signature is not trusted by the computer
 - Please [enable test signing](#) and see the [signing section above](#) on how to set up a [test certificate](#).
- Wsl/Service/CreateInstance/CreateVm/Plugin/* -> The plugin DLL returned an error in WSLPLUGINAPI_ENTRYPOINTV1 or OnVmStarted()

- Wsl/Service/CreateInstance/Plugin/* -> The plugin DLL returned an error in OnDistributionStarted()

Plugins Linux user space

Linux processes created via ExecuteBinary() will run in the root namespace of the WSL2 Virtual Machine. This namespace is not associated to any distribution and has a very minimal Mariner based root file system.

That file system is a writable tmpfs, meaning that all changes made to it will be dropped when the WSL2 Virtual Machine is shut down.

You can inspect the content of the root namespace by running `ws1 --debug-shell` while WSL is running.

Additional considerations

- All WSL plugin hooks are synchronous, meaning that WSL will wait for the plugin hooks to be completed before continuing.
- Any error returned by a plugin is considered fatal by WSL (meaning that the user's distribution will not start)
- The plugin code runs in the same address space as the WSL service. Any crash in a plugin will crash the entire WSL service, potentially causing data loss



Collaborate with us on
GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

[Open a documentation issue](#)

[Provide product feedback](#)

Enterprise environment: Set up Windows Subsystem for Linux for your company

Article • 11/16/2023

This guidance is intended for IT Administrators or Security Analysts responsible for setting up enterprise work environments with the goal of distributing software across multiple machines and maintaining a consistent level of security settings across those work machines.

Many companies use [Microsoft Intune](#) and [Microsoft Defender](#) to manage these security settings. However, setting up WSL and accessing Linux distributions in this context requires some specific setup. This guidance provides what you need to know to enable the secure use of Linux with WSL in an enterprise environment.

Recommended Enterprise set up with Microsoft Defender for Endpoint, Intune, and Advanced Networking Controls

There are a variety of ways to set up a secure enterprise environment, but we recommend the following for setting up a secure environment that utilizes WSL.

Pre-requisites

To get started ensure that all enterprise devices have the following minimum versions installed:

- Windows 10 22H2 or higher, or Windows 11 22H2 or higher
 - Advanced networking features are only available on Windows 11 22H2 or higher.
- [WSL version 2.0.9](#) or higher
 - You can check the WSL version by running `wsl --version`.

Enable Microsoft Defender for Endpoint (MDE) integration

[Microsoft Defender for Endpoint](#) is an enterprise endpoint security platform designed to help enterprise networks prevent, detect, investigate, and respond to advanced threats. MDE now integrates with WSL as a [WSL plugin](#), which allows security teams to see and continuously monitor for security events in all running WSL distributions with Defender for Endpoint while minimally impacting performance on developer workloads.

See [Microsoft Defender for Endpoint plug-in for WSL](#) to learn more about how to get started.

Configure recommended settings with Intune

[Microsoft Intune](#) is a cloud-based endpoint management solution. It manages user access to organizational resources and simplifies app and device management across your many devices, including mobile devices, desktop computers, and virtual endpoints. You can use Microsoft Intune to manage devices inside of your organization, which now also includes managing access to WSL and its key security settings.

See [Intune settings for WSL](#) for guidance on using InTune to manage WSL as a Windows component and the recommended settings.

Use advanced networking features and controls

Starting from Windows 11 22H2 and WSL 2.0.9 or later, Windows firewall rules will automatically apply to WSL. This ensures that the firewall rules set on the Windows host will automatically apply to all WSL distributions by default. For guidance on customizing the firewall settings for WSL, visit [Configure Hyper-V firewall](#).

Additionally, we recommend configuring [settings under \[wsl2\]](#) in the `.wslconfig` file to suit your specific Enterprise scenario.

Mirrored mode networking

`networkingMode=mirrored` enables mirrored mode networking. This new networking mode improves compatibility with complex networking environments, especially VPNs and more, as well as adding support for new networking features unavailable in the default NAT mode like IPv6.

DNS Tunneling

`dnsTunneling=true` changes how WSL obtains DNS information. This setting improves compatibility in different networking environments, and makes use of virtualization

features to obtain DNS information rather than a networking packet. It's recommended to turn this on if experiencing any connectivity issues, and can be especially helpful when using VPNs, advanced firewall settings, and more.

Auto proxy

`autoProxy=true` enforces WSL to use Windows' HTTP proxy information. We recommend turning this setting on when using a proxy on Windows, as it will make that proxy automatically apply to your WSL distributions.

Creating a custom WSL image

What is commonly referred to as an "image", is simply a snapshot of your software and its components saved to a file. In the case of the Windows Subsystem for Linux, your image would consist of the subsystem, its distributions, and whatever software and packages are installed on the distribution.

To begin creating your WSL image, first [install the Windows Subsystem for Linux](#).

Once installed, use The Microsoft Store for Business to download and install the Linux distribution that's right for you. Create an account with the [Microsoft Store for Business ↗](#).

Exporting your WSL image

Export your custom WSL image by running `wsl --export <Distro> <FileName>`, which will wrap your image in a tar file and make it ready for distribution on to other machines. You can [create custom distributions including CentOS, RedHat and more using the custom distro guide](#).

Distributing your WSL image

Distribute the WSL image from a share or storage device by running `wsl --import <Distro> <InstallLocation> <FileName>`, which will import the specified tar file as a new distribution.

Update and patch Linux distributions and packages

Using Linux configuration manager tools is strongly recommended for monitoring and managing Linux user space. There are a host of Linux configuration managers to choose from. See this blog post on [Running Puppet quickly in WSL 2](#).

Windows file system access

When a Linux binary inside of WSL accesses a Windows file, it does so with the user permissions of the Windows user that ran `wsl.exe`. So even though a Linux user has root access inside of WSL, they cannot do Windows administrator level operations on Windows if the Windows user does not have those permission. With regards to Windows file and Windows executable access from WSL, running a shell like `bash` has the same security level permissions as running `powershell` from Windows as that user.

Supported

- Sharing an approved image internally using `wsl --import` and `wsl --export`
- Creating your own WSL distro for your Enterprise using the [WSL Distro Launcher repo](#)
- Monitor security events inside of WSL distros using Microsoft Defender for Endpoint (MDE)
- Use firewall settings to control networking in WSL (Includes syncing Windows firewall settings to WSL)
- Control access to WSL and its key security settings with Intune or group policy

Here's a list of features for which we don't yet have support for, but are investigating.

Currently unsupported

Below is a list of commonly asked features that are currently unsupported within WSL. These requests are on our backlog and we are investigating ways to add them.

- Managing updates and patching of the Linux distributions and packages using Windows tools
- Having Windows update also update WSL distro contents
- Controlling which distributions users in your Enterprise can access
- Controlling root access for users

Intune settings for WSL

Article • 01/10/2024

You can now use management tools like Intune to manage WSL as a Windows component.

To access these settings please navigate to your Microsoft Intune admin center portal, and then select: `Devices -> Configuration Profiles -> Create -> New Policy -> Windows 10 and later -> Settings catalog`, create a name for the new profile and search for "Windows Subsystem for Linux" to see and add the full list of available settings.

Recommended settings

To maximize security in an enterprise environment, we recommend that you specify these settings:

[+] [Expand table](#)

Setting Name	Value	Description
Allow the Inbox version of the Windows Subsystem for Linux	Disabled	When set to disabled, this policy disables the inbox version (optional component) of the Windows Subsystem For Linux. If this policy is disabled, only the store version of WSL can be used.
Allow WSL1	Disabled	When set to disabled, this policy disables WSL1. When disabled, only WSL2 distributions can be used.
Allow the debug shell	Disabled	When set to disabled, this policy disables the debug shell (wsl.exe --debug-shell). This policy only applies to Store WSL.
Allow custom kernel configuration	Disabled	When set to disabled, this policy disables custom kernel configuration via .wslconfig (wsl2.kernel). This policy only applies to Store WSL.
Allow kernel command line configuration	Disabled	When set to disabled, this policy disables kernel command line configuration via .wslconfig (wsl2.kernelCommandLine). This policy only applies to Store WSL.
Allow custom system distribution configuration	Disabled	When set to disabled, this policy disables custom system distribution configuration via .wslconfig (wsl2.systemDistro). This policy only applies to Store WSL.

Setting Name	Value	Description
Allow custom networking configuration	Disabled	When set to disabled, this policy disables custom networking configuration via .wslconfig (wsl2.networkingmode). This policy only applies to Store WSL.
Allow user setting firewall configuration	Disabled	When set to disabled, this policy disables firewall configuration via .wslconfig (wsl2.firewall). This policy only applies to Store WSL.
Allow nested virtualization	Disabled	When set to disabled, this policy disables nested virtualization configuration via .wslconfig (wsl2.nestedVirtualization). This policy only applies to Store WSL.
Allow kernel debugging	Disabled	When set to disabled, this policy disables kernel debugging configuration via .wslconfig (wsl2.kernelDebugPort). This policy only applies to Store WSL.

Control access to WSL

The `AllowWSL`, `AllowInboxWSL`, and `AllowWSL1` settings control user access to WSL. You can configure these settings to enable or disable access to the in-Windows version of WSL, WSL 1 distros, or WSL itself.

This will allow you to configure WSL to ensure that users are only using the latest version of WSL with Enterprise feature support.

Control WSL commands

`AllowDebugShell` and `AllowDiskMount` control whether users can run the `wsl --debug-shell` and `wsl --mount` commands. Learn more about how to [Mount a disk in WSL 2](#) using the `wsl --mount` command.

Control access to WSL settings in `.wslconfig`

The last group of settings that end with `*UserSettingConfigurable` control access to WSL advanced settings in `.wslconfig`. When these are set to disabled then users will only be able to use the default value for that setting, and not able to configure it to custom values. Learn more about [Configuration setting for .wslconfig](#), including a list of settings that can be configured globally for all Linux distributions running with WSL 2.

Full list of available settings

[Expand table](#)

Setting Name	Description
Allow the Windows Subsystem For Linux	When set to disabled, this policy disables access to the Windows Subsystem For Linux for all users on the machine.
Allow the Inbox version of the Windows Subsystem For Linux	When set to disabled, this policy disables the inbox version (optional component) of the Windows Subsystem For Linux. If this policy is disabled, only the store version of WSL can be used.
Allow WSL1	When set to disabled, this policy disables WSL1. When disabled, only WSL2 distributions can be used.
Allow the debug shell	When set to disabled, this policy disables the debug shell (wsl.exe --debug-shell). This policy only applies to Store WSL.
Allow passthrough disk mount	When set to disabled, this policy disables passthrough disk mounting in WSL2 (wsl.exe --mount). This policy only applies to Store WSL.
Allow custom kernel configuration	When set to disabled, this policy disables custom kernel configuration via .wslconfig (wsl2.kernel). This policy only applies to Store WSL.
Allow kernel command line configuration	When set to disabled, this policy disables kernel command line configuration via .wslconfig (wsl2.kernelCommandLine). This policy only applies to Store WSL.
Allow custom system distribution configuration	When set to disabled, this policy disables custom system distribution configuration via .wslconfig (wsl2.systemDistro). This policy only applies to Store WSL.
Allow custom networking configuration	When set to disabled, this policy disables custom networking configuration via .wslconfig (wsl2.networkingmode). This policy only applies to Store WSL.
Allow user setting firewall configuration	When set to disabled, this policy disables firewall configuration via .wslconfig (wsl2.firewall). This policy only applies to Store WSL.
Allow nested virtualization	When set to disabled, this policy disables nested virtualization configuration via .wslconfig (wsl2.nestedVirtualization). This policy only applies to Store WSL.
Allow kernel debugging	When set to disabled, this policy disables kernel debugging configuration via .wslconfig (wsl2.kernelDebugPort). This policy only applies to Store WSL.

 Collaborate with us on
GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Frequently Asked Questions about Windows Subsystem for Linux

FAQ

General

What is Windows Subsystem for Linux (WSL)?

The Windows Subsystem for Linux (WSL) is a feature of the Windows operating system that enables you to run a Linux file system, along with Linux command-line tools and GUI apps, directly on Windows, alongside your traditional Windows desktop and apps.

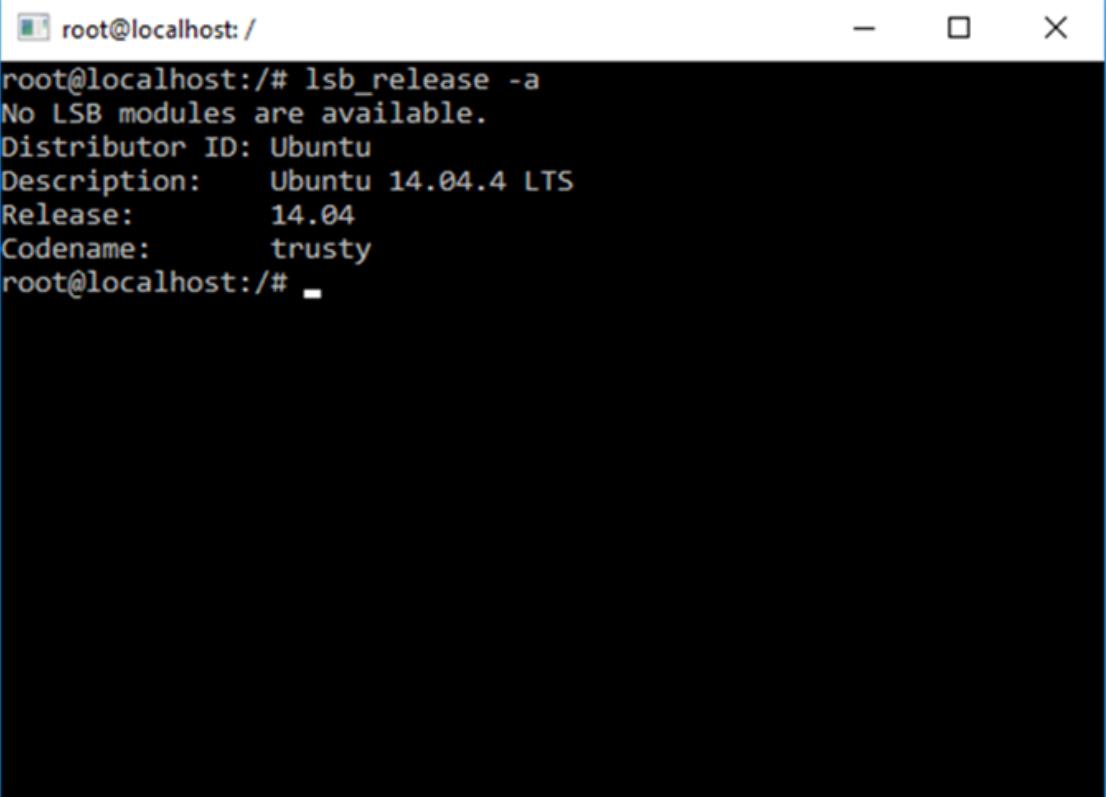
See the [about page](#) for more details.

Who is WSL for?

This is primarily a tool for developers, especially web developers, those working on open source projects, or deploying to Linux server environments. WSL is for anyone who likes using Bash, common Linux tools (`sed`, `awk`, etc.) and Linux-first frameworks (Ruby, Python, etc.) but also enjoys using Windows productivity tools.

What can I do with WSL?

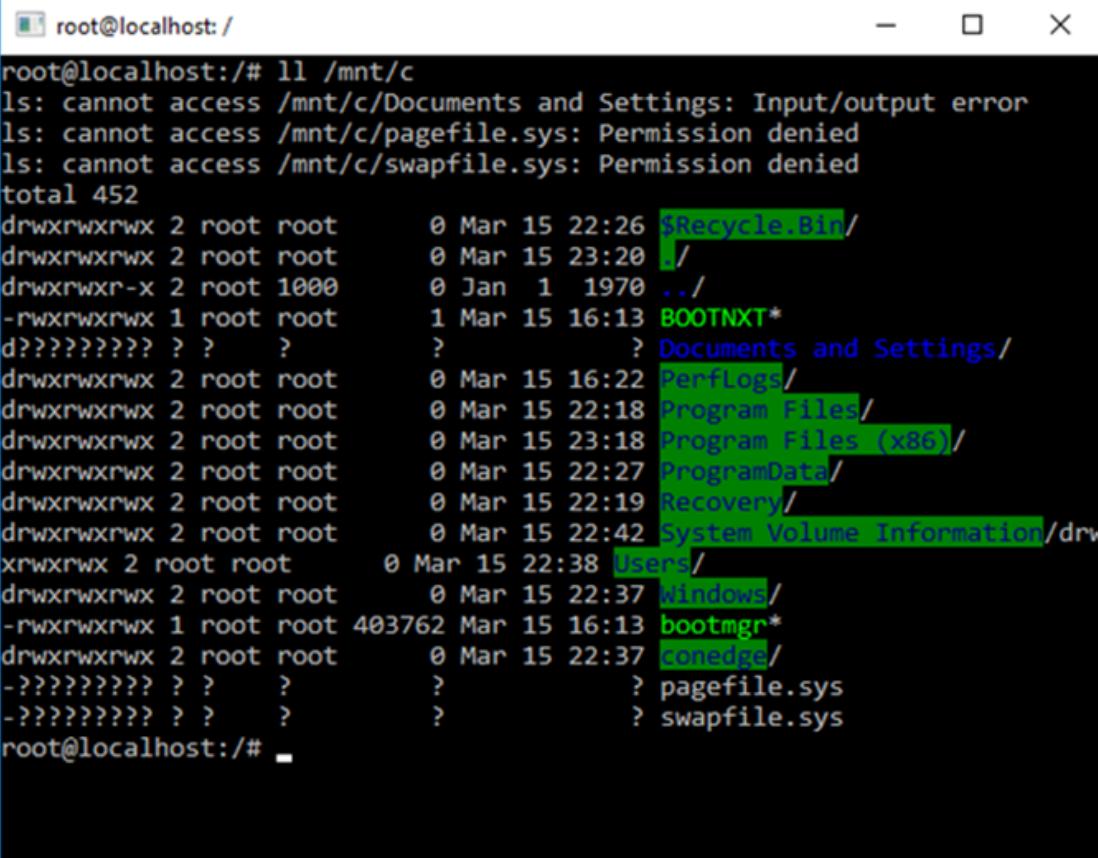
WSL enables you to run Linux in a Bash shell with your choice of distribution (Ubuntu, Debian, OpenSUSE, Kali, Alpine, etc). Using Bash, you can run command-line Linux tools and apps. For example, type `lsb_release -a` and hit enter; you'll see details of the Linux distro currently running:



A screenshot of a terminal window titled "root@localhost:/". The window contains the following text output from the command `lsb_release -a`:

```
root@localhost:/# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.4 LTS
Release:        14.04
Codename:       trusty
root@localhost:/#
```

You can also access your local machine's file system from within the Linux Bash shell – you'll find your local drives mounted under the `/mnt` folder. For example, your `C:` drive is mounted under `/mnt/c:`



```
root@localhost:/# ls /mnt/c
ls: cannot access /mnt/c/Documents and Settings: Input/output error
ls: cannot access /mnt/c/pagefile.sys: Permission denied
ls: cannot access /mnt/c/swapfile.sys: Permission denied
total 452
drwxrwxrwx 2 root root      0 Mar 15 22:26 $Recycle.Bin/
drwxrwxrwx 2 root root      0 Mar 15 23:20 /
drwxrwxr-x 2 root 1000      0 Jan  1 1970 ...
-rwxrwxrwx 1 root root     1 Mar 15 16:13 BOOTNXT*
d????????? ? ?      ?      ? Documents and Settings/
drwxrwxrwx 2 root root     0 Mar 15 16:22 PerfLogs/
drwxrwxrwx 2 root root     0 Mar 15 22:18 Program Files/
drwxrwxrwx 2 root root     0 Mar 15 23:18 Program Files (x86)/
drwxrwxrwx 2 root root     0 Mar 15 22:27 ProgramData/
drwxrwxrwx 2 root root     0 Mar 15 22:19 Recovery/
drwxrwxrwx 2 root root     0 Mar 15 22:42 System Volume Information/drw
xrwxrwx 2 root root      0 Mar 15 22:38 Users/
drwxrwxrwx 2 root root     0 Mar 15 22:37 Windows/
-rwxrwxrwx 1 root root 403762 Mar 15 16:13 bootmgr*
drwxrwxrwx 2 root root     0 Mar 15 22:37 conedge/
-????????? ? ?      ?      ? pagefile.sys
-????????? ? ?      ?      ? swapfile.sys
root@localhost:/#
```

Could you describe a typical development workflow that incorporates WSL?

WSL targets a developer audience with the intent to be used as part of an inner development loop. Let's say that Sam is creating a CI/CD pipeline (Continuous Integration & Continuous Delivery) and wants to test it first on a local machine (laptop) before deploying it to the cloud. Sam can enable WSL (& WSL 2 to improve speed and performance), and then use a genuine Linux Ubuntu instance locally (on the laptop) with whatever Bash commands and tools they prefer. Once the development pipeline is verified locally, Sam can then push that CI/CD pipeline up to the cloud (i.e. Azure) by making it into a Docker container and pushing the container to a cloud instance where it runs on a production-ready Ubuntu VM.

What is Bash?

Bash [🔗](#) is a popular text-based shell and command-language. It is the default shell included within Ubuntu and other Linux distros. Users type commands into a shell to execute scripts and/or run commands and tools to accomplish many tasks.

How does this work?

Check out this article on the Windows Command Line blog: [A Deep Dive Into How WSL Allows Windows to Access Linux Files](#) which goes into detail about the underlying technology.

Why would I use WSL rather than Linux in a VM?

WSL requires fewer resources (CPU, memory, and storage) than a full virtual machine. WSL also allows you to run Linux command-line tools and apps alongside your Windows command-line, desktop and store apps, and to access your Windows files from within Linux. This enables you to use Windows apps and Linux command-line tools on the same set of files if you wish.

Why would I use, for example, Ruby on Linux instead of on Windows?

Some cross-platform tools were built assuming that the environment in which they run behaves like Linux. For example, some tools assume that they are able to access very long file paths or that specific files/folders exist. This often causes problems on Windows which often behaves differently from Linux.

Many languages like Ruby and Node.js are often ported to, and run great, on Windows. However, not all of the Ruby Gem or node/NPM library owners port their libraries to support Windows, and many have Linux-specific dependencies. This can often result in systems built using such tools and libraries suffering from build and sometimes runtime errors or unwanted behaviors on Windows.

These are just some of issues that caused many people to ask Microsoft to improve Windows' command-line tools and what drove us to partner with Canonical to enable native Bash and Linux command-line tools to run on Windows.

What does this mean for PowerShell?

While working with OSS projects, there are numerous scenarios where it's immensely useful to drop into Bash from a PowerShell prompt. Bash support is complementary and strengthens the value of the command-line on Windows, allowing PowerShell and the PowerShell community to leverage other popular technologies.

Read more on the PowerShell team blog -- [Bash for Windows: Why it's awesome and what it means for PowerShell](#)

What processors does WSL support?

WSL supports x64 and Arm CPUs.

How do I access my C: drive?

Mount points for hard drives on the local machine are automatically created and provide easy access to the Windows file system.

`/mnt/<drive letter>/`

Example usage would be `cd /mnt/c` to access `c:\`

How do I set up Git Credential Manager? (How do I use my Windows Git permissions in WSL?)

See the tutorial [Get started using Git on Windows Subsystem for Linux](#), which features a section on setting up Git Credential Manager and storing authentication tokens in Windows Credential Manager.

How do I use a Windows file with a Linux app?

One of the benefits of WSL is being able to access your files via both Windows and Linux apps or tools.

WSL mounts your machine's fixed drives under the `/mnt/<drive>` folder in your Linux distros. For example, your `c:` drive is mounted under `/mnt/c/`.

Using your mounted drives, you can edit code in, for example, `c:\dev\myproj\` using [Visual Studio](#) or [VS Code](#), and build/test that code in Linux by accessing the same files via `/mnt/c/dev/myproj`.

Learn more in [Working across Windows and Linux file systems](#) article.

Are files in the Linux drive different from the mounted Windows drive?

1. Files under the Linux root (i.e. `/`) are controlled by WSL which aligns with Linux behavior, including but not limited to:
 - Files which contain invalid Windows filename characters

- Symlinks created for non-admin users
- Changing file attributes through `chmod` and `chown`
- File/folder case sensitivity

2. Files in mounted drives are controlled by Windows and have the following behaviors:

- Support case sensitivity
- All permissions are set to best reflect the Windows permissions

How do I uninstall a WSL Distribution?

To remove a distribution from WSL and *delete all of the data associated with that Linux distribution*, run `wsl --unregister <distroName>` where `<distroName>` is the name of your Linux distro, which can be seen from the list in the `wsl -l` command.

Additionally, you can then uninstall the Linux distro app on your machine just like any other store application.

To learn more about wsl commands, see the article, [Basic commands for WSL](#).

How do I run an OpenSSH server?

OpenSSH ships with Windows as an optional feature. See the [Install OpenSSH](#) doc. Administrator privileges in Windows are required to run OpenSSH in WSL. To run an OpenSSH server, run your WSL distribution (i.e. Ubuntu) or Windows Terminal as an administrator. There are several resources out there covering SSH scenarios with WSL. Check out Scott Hanselman's blog articles: [How to SSH into a Windows 10 Machine from Linux OR Windows OR anywhere ↗](#), [How to SSH into WSL2 on Windows 10 from an external machine ↗](#), [THE EASY WAY how to SSH into Bash and WSL2 on Windows 10 from an external machine ↗](#), and [How to use Windows 10's built-in OpenSSH to automatically SSH into a remote Linux machine ↗](#).

How do I change the display language of WSL?

WSL install will try to automatically change the Ubuntu locale to match the locale of your Windows install. If you do not want this behavior you can run this command to change the Ubuntu locale after install completes. You will have to relaunch your WSL distribution for this change to take effect.

The below example changes the locale to en-US:

```
Bash
```

```
sudo update-locale LANG=en_US.UTF8
```

Why do I not have internet access from WSL?

Some users have reported issues with specific firewall applications blocking internet access in WSL. The firewalls reported are:

1. Kaspersky
2. AVG
3. Avast
4. Symantec Endpoint Protection
5. F-Secure

In some cases turning off the firewall allows for access. In some cases simply having the firewall installed looks to block access.

How do I access a port from WSL in Windows?

WSL shares the IP address of Windows, as it is running on Windows. As such you can access any ports on localhost e.g. if you had web content on port 1234 you could <https://localhost:1234> into your Windows browser. For more information, see [Accessing network applications](#).

How can I back up my WSL distributions?

The best way to backup or move your distributions is via the [export/import](#) commands available in Windows Version 1809 and later. You can export your entire distribution to a tarball using the `wsl --export` command. You can then import this distribution back into WSL using the `wsl --import` command, which can name a new drive location for the import, allowing you to backup and save states of (or move) your WSL distributions. To learn more about moving your WSL distributions, see [How can I transfer my WSL files from one machine to another?](#).

Traditional backup services that backup files in your AppData folders (like Windows Backup) will not corrupt your Linux files.

Can I use WSL for production scenarios?

WSL has been designed and built to use with inner loop development workflows. There are design features in WSL that make it great for this purpose but may make it challenging for production-related scenarios compared to other products. Our goal is to make clear how WSL differs from a regular VM environment, so you can make the decision on whether it fits your business needs.

The main differences between WSL and a traditional production environment are:

- WSL has a lightweight utility VM that starts, stops and manages resources automatically.
- If you have no open file handles to Windows processes, the WSL VM will automatically be shut down. This means if you are using it as a web server, SSH into it to run your server and then exit, the VM could shut down because it is detecting that users are finished using it and will clean up its resources.
- WSL users have full access to their Linux instances. The lifetime of the VM, the registered WSL distributions, etc., are all accessible by the user and can be modified by the user.
- WSL automatically gives file access to Windows files.
- Windows paths are appended to your path by default, which could cause unexpected behavior for certain Linux applications compared to a traditional Linux environment.
- WSL can run Windows executables from Linux, which could also lead to a different environment than a traditional Linux VM.
- The Linux kernel used by WSL is updated automatically.
- GPU access in WSL happens through a `/dev/dxg` device, which routes GPU calls out to the Windows GPU. This setup is different than a traditional Linux set up.
- There are other smaller differences compared to bare metal Linux and more differences are expected to arise in the future as the inner loop development workflow is prioritized.

How can I transfer my WSL files from one machine to another?

There are a few ways you can accomplish this task:

- The easiest way is to use the `wsl --export --vhd` command to export your WSL distribution to a VHD file. You can then copy this file to another machine, and import it using `wsl --import --vhd`. See the [import](#) and [export](#) commands in the [WSL basic commands](#) doc for more information.
- The implementation above requires a lot of disk space. If you don't have a lot of disk space you can use Linux techniques to move your files over:

- Use `tar -czf <tarballName> <directory>` to create a tarball of your files. You can then copy these specific files over to your new machine and run `tar -xzf <tarballName>` to extract them.
- You can also export a list of installed packages via `apt` with a command like so:
`dpkg --get-selections | grep -v deinstall | awk '{print $1}' > package_list.txt` and then reinstall those same packages on another machine with a command like `sudo apt install -y $(cat package_list.txt)` after transferring the file over.

How can I move my WSL distribution to a different drive or location?

You can do this using PowerShell. Below are the necessary commands and explanations for each step. Please open a PowerShell window and adjust the values in between the `< >` tags to fit your specific use case:

PowerShell

```
# Create a folder where you would like to store your distro
New-Item -ItemType Directory -Path <Install location, e.g:
D:\WSLDistros\Ubuntu>

# Export your distro to that folder as a VHD
wsl --export --vhd <Distroname, e.g: Ubuntu> <Install Location with
filename, e.g: D:\WSLDistros\Ubuntu\ext4.vhdx>

# Unregister your old distro
# Please note this will erase your existing distro's file contents, please
ensure the backup file you created in the 2nd step is present at the
location and that the export operation completed successfully.
# Please exercise caution when using this command, as it is destructive and
could cause data loss.
wsl --unregister <Distroname>

# Import your VHD backup
wsl --import-in-place <Distroname> <Install Location with filename>
```

WSL 2

Is WSL 2 available on Windows 10 Home and Windows 11 Home?

Yes. WSL 2 is available on all Desktop SKUs where WSL is available, including Windows 10 Home and Windows 11 Home.

Specifically, WSL2 requires two features to be enabled:

1. "Virtual Machine Platform" (a subset of Hyper-V)
2. "Windows Subsystem for Linux"

Does WSL 2 use Hyper-V?

The newest version of WSL uses a *subset* of Hyper-V architecture to enable its virtualization. This subset is provided as an optional component named "Virtual Machine Platform," available on all Desktop SKUs.

What will happen to WSL 1? Will it be abandoned?

We currently have no plans to deprecate WSL 1. You can run WSL 1 and WSL 2 distros side by side, and can upgrade and downgrade any distro at any time. Adding WSL 2 as a new architecture presents a better platform for the WSL team to deliver features that make WSL an amazing way to run a Linux environment in Windows.

Will I be able to run WSL 2 and other 3rd party virtualization tools such as VMware, or VirtualBox?

Some 3rd party applications cannot work when Hyper-V is in use, which means they will not be able to run when WSL 2 is enabled, such as VMware and VirtualBox. However, recently both VirtualBox and VMware have released versions that support Hyper-V and WSL2. Learn more about [VirtualBox's changes here](#) and [VMware's changes here](#). For troubleshooting issues, take a look at the [VirtualBox issue discussions in the WSL repo on GitHub](#). StackOverflow also offers a helpful tip: [How to get VirtualBox 6.0 and WSL working at the same time](#).

We are consistently working on solutions to support third-party integration of Hyper-V. For example, we expose a set of APIs called [Hypervisor Platform](#) that third-party virtualization providers can use to make their software compatible with Hyper-V. This lets applications use the Hyper-V architecture for their emulation such as [the Google Android Emulator](#), and VirtualBox 6 and above which are both now compatible with Hyper-V.

See the WSL issues repo for more background and discussion on [WSL 2 issues with VirtualBox 6.1](#).

*If you're looking for a Windows virtual machine, VMWare, Hyper-V, VirtualBox, and Parallels VM downloads are [available on the Windows Dev Center](#).

Can I access the GPU in WSL 2? Are there plans to increase hardware support?

We have released support for accessing the GPU inside of WSL 2 distributions! This means you can now use WSL for machine learning, artificial intelligence, and data science scenarios more easily when big data sets are involved. Check out the [get started with GPU support](#) tutorial. As of right now WSL 2 does not include serial support, or USB device support. We are investigating the best way to add these features. However, USB support is now available through the USBIPD-WIN project. See [Connect USB devices](#) for steps to set up USB device support.

Can WSL 2 use networking applications?

Yes, in general networking applications will work better and be faster with WSL 2 since it offers full system call compatibility. However, the WSL 2 architecture uses virtualized networking components, which means that WSL 2 will behave similarly to a virtual machine -- WSL 2 distributions will have a different IP address than the host machine (Windows OS). For more information, see [Accessing network applications with WSL](#).

Can I run WSL 2 in a virtual machine?

Yes! You need to make sure that the virtual machine has nested virtualization enabled. This can be enabled in your parent Hyper-V host by running the following command in a PowerShell window with Administrator privileges:

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

Make sure to replace '<VMName>' with the name of your virtual machine.

Can I use wsl.conf in WSL 2?

WSL 2 supports the same wsl.conf file that WSL 1 uses. This means that any configuration options that you had set in a WSL 1 distro, such as automounting Windows drives, enabling or disabling interop, changing the directory where Windows drives will be mounted, etc. will all work inside of WSL 2. You can learn more about the

configuration options in WSL in the [Distribution Management](#) page. Learn more about support for mounting drives, disks, devices, or virtual hard disks (VHDs) in the [Mount a Linux disk in WSL 2](#) article.

Where can I provide feedback?

The [WSL product repo issues](#) enables you to:

- **Search existing issues** to see if there are any associated with a problem that you are having. Note that in the search bar, you can remove "is:open" to include issues that have already been resolved in your search. Please consider commenting or giving a thumbs up to any open issues that you would like to express your interest in moving forward as a priority.
- **File a new issue.** If you have found a problem with WSL and there does not appear to be an existing issue, you can select the green *New issue* button and then choose *WSL - Bug Report*. You will need to include a title for the issue, your Windows build number (run `cmd.exe /c ver` to see your current build #), whether you're running WSL 1 or 2, your current Linux Kernel version # (run `wsl.exe --status` or `cat /proc/version`), the version # of your distribution (run `lsb_release -r`), any other software versions involved, the repro steps, expected behavior, actual behavior, and diagnostic logs if available and appropriate. For more info, see [contributing to WSL](#).
- **File a feature request** by selecting the green *New issue* button and then select *Feature request*. You will need to address a few questions describing your request.

You can also:

- **File a documentation issue** using the [WSL docs repo](#). To contribute to the WSL docs, see the [Microsoft Docs contributor guide](#).
- **File a Windows Terminal issue** using the [Windows Terminal product repo](#) if your problem is related more to the Windows Terminal, Windows Console, or the command-line UI.

If you'd like to stay up to date with the latest WSL news you can do so with:

- Our [command-line team blog](#)
- Twitter. Please follow [@craigaloewen](#) on Twitter to learn of news, updates, etc.



Collaborate with us on
GitHub



Windows Subsystem for
Linux feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Troubleshooting Windows Subsystem for Linux

Article • 02/22/2024

We have covered some common troubleshooting scenarios associated with WSL below, but please consider searching the issues filed in the [WSL product repo on GitHub](#) as well.

File an issue, bug report, feature request

The [WSL product repo issues](#) enables you to:

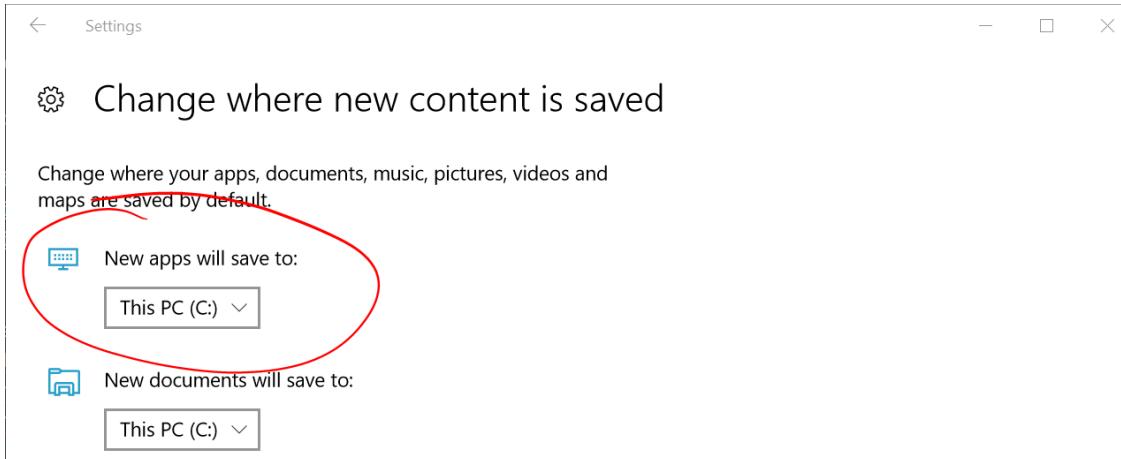
- **Search existing issues** to see if there are any associated with a problem that you are having. Note that in the search bar, you can remove "is:open" to include issues that have already been resolved in your search. Please consider commenting or giving a thumbs up to any open issues that you would like to express your interest in moving forward as a priority.
- **File a new issue.** If you have found a problem with WSL and there does not appear to be an existing issue, you can select the green *New issue* button and then choose *WSL - Bug Report*. You will need to include a title for the issue, your Windows build number (run `cmd.exe /c ver` to see your current build #), whether you're running WSL 1 or 2, your current Linux Kernel version # (run `wsl.exe --status` or `cat /proc/version`), the version # of your distribution (run `lsb_release -r`), any other software versions involved, the repro steps, expected behavior, actual behavior, and diagnostic logs if available and appropriate. For more info, see [contributing to WSL](#).
- **File a feature request** by selecting the green *New issue* button and then select *Feature request*. You will need to address a few questions describing your request.

You can also:

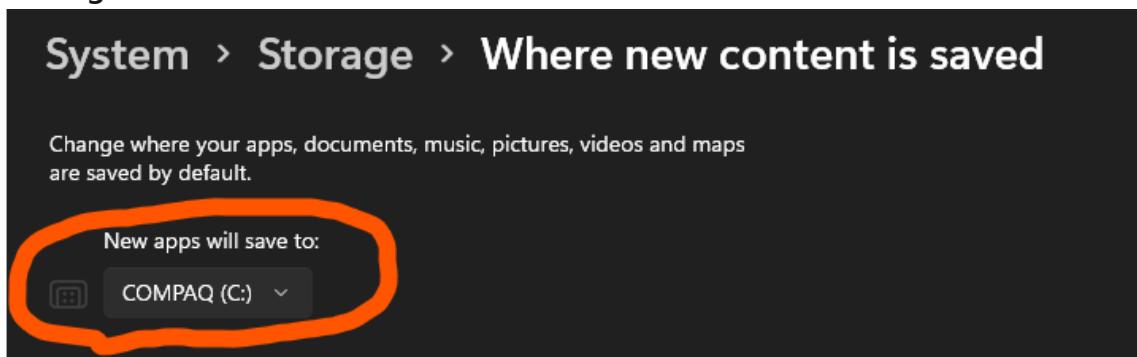
- **File a documentation issue** using the [WSL docs repo](#). To contribute to the WSL docs, see the [Microsoft Docs contributor guide](#).
- **File a Windows Terminal issue** using the [Windows Terminal product repo](#) if your problem is related more to the Windows Terminal, Windows Console, or the command-line UI.

Installation issues

- Installation failed with error 0x80070003
 - The Windows Subsystem for Linux only runs on your system drive (usually this is your C: drive). Make sure that distributions are stored on your system drive:
 - On Windows 10 open **Settings** -> **System** -> **Storage** -> **More Storage**
Settings: Change where new content is saved



- On Windows 11 open **Settings** -> **System** -> **Storage** -> **Advanced storage settings** -> **Where new content is saved**

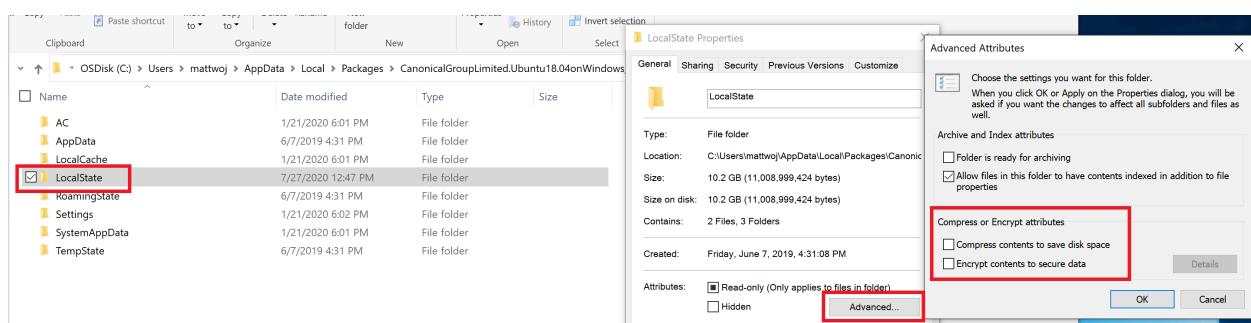


- WslRegisterDistribution failed with error 0x8007019e
 - The Windows Subsystem for Linux optional component is not enabled:
 - Open **Control Panel** -> **Programs and Features** -> **Turn Windows Feature on or off** -> Check **Windows Subsystem for Linux** or using the PowerShell cmdlet mentioned at the beginning of this article.
- Installation failed with error 0x80070003 or error 0x80370102
 - Please make sure that virtualization is enabled inside of your computer's BIOS. The instructions on how to do this will vary from computer to computer, and will most likely be under CPU related options.
 - WSL2 requires that your CPU supports the Second Level Address Translation (SLAT) feature, which was introduced in Intel Nehalem processors (Intel Core 1st Generation) and AMD Opteron. Older CPUs (such as the Intel Core 2 Duo) will not be able to run WSL2, even if the Virtual Machine Platform is successfully installed.

- Error when trying to upgrade: `Invalid command line option: wsl --set-version`

Ubuntu 2

- Ensure that you have the Windows Subsystem for Linux enabled, and that you're using Windows Build version 18362 or later. To enable WSL run this command in a PowerShell prompt with admin privileges: `Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux`.
- The requested operation could not be completed due to a virtual disk system limitation. Virtual hard disk files must be uncompressed and unencrypted and must not be sparse.
 - Deselect "Compress contents" (as well as "Encrypt contents" if that's checked) by opening the profile folder for your Linux distribution. It should be located in a folder on your Windows file system, something like:
`%USERPROFILE%\AppData\Local\Packages\CanonicalGroupLimited...`
 - In this Linux distro profile, there should be a LocalState folder. Right-click this folder to display a menu of options. Select Properties > Advanced and then ensure that the "Compress contents to save disk space" and "Encrypt contents to secure data" checkboxes are unselected (not checked). If you are asked whether to apply this to just to the current folder or to all subfolders and files, select "just this folder" because you are only clearing the compress flag. After this, the `wsl --set-version` command should work.



① Note

In my case, the LocalState folder for my Ubuntu 18.04 distribution was located at
 C:\Users<my-user-name>\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu18.04onWindows_79rhkp1fndgsc

Check [WSL Docs GitHub thread #4103](#) where this issue is being tracked for updated information.

- The term 'wsl' is not recognized as the name of a cmdlet, function, script file, or operable program.

- Ensure that the [Windows Subsystem for Linux Optional Component](#) is installed. Additionally, if you are using an ARM64 device and running this command from PowerShell, you will receive this error. Instead run `wsl.exe` from [PowerShell Core](#), or Command Prompt.
- **Error: Windows Subsystem for Linux has no installed distributions.**
 - If you receive this error after you have already installed WSL distributions:
 1. Run the distribution at least once before invoking it from the command line.
 2. Check whether you may be running separate user accounts. Running your primary user account with elevated permissions (in admin mode) should not result in this error, but you should ensure that you aren't accidentally running the built-in Administrator account that comes with Windows. This is a separate user account and will not show any installed WSL distributions by design. For more info, see [Enable and Disable the Built-in Administrator Account](#).
 3. The WSL executable is only installed to the native system directory. When you're running a 32-bit process on 64-bit Windows (or on ARM64, any non-native combination), the hosted non-native process actually sees a different System32 folder. (The one a 32-bit process sees on x64 Windows is stored on disk at `\Windows\SysWOW64`.) You can access the "native" system32 from a hosted process by looking in the virtual folder: `\Windows\sysnative`. It won't actually be present on disk, mind you, but the filesystem path resolver will find it.
 - **Error: This update only applies to machines with the Windows Subsystem for Linux.**
 - To install the Linux kernel update MSI package, WSL is required and should be enabled first. If it fails, it you will see the message: `This update only applies to machines with the Windows Subsystem for Linux`.
 - There are three possible reason you see this message:
 1. You are still in old version of Windows which doesn't support WSL 2. See step #2 for version requirements and links to update.
 2. WSL is not enabled. You will need to return to step #1 and ensure that the optional WSL feature is enabled on your machine.
 3. After you enabled WSL, a reboot is required for it to take effect, reboot your machine and try again.
 - **Error: WSL 2 requires an update to its kernel component. For information please visit <https://aka.ms/wsl2kernel>.**

- If the Linux kernel package is missing in the %SystemRoot%\system32\lxss\tools folder, you will encounter this error. Resolve it by installing the Linux kernel update MSI package in step #4 of these installation instructions. You may need to uninstall the MSI from '[Add or Remove Programs](#)', and install it again.

Common issues

I'm on Windows 10 version 1903 and I still do not see options for WSL 2

This is likely because your machine has not yet taken the backport for WSL 2. The simplest way to resolve this is by going to Windows Settings and clicking 'Check for Updates' to install the latest updates on your system. See [the full instructions on taking the backport ↗](#).

If you hit 'Check for Updates' and still do not receive the update you can [install KB KB4566116 manually ↗](#).

Error: 0x1bc when `wsl --set-default-version 2`

This may happen when 'Display Language' or 'System Locale' setting is not English.

PowerShell

```
wsl --set-default-version 2
Error: 0x1bc
For information on key differences with WSL 2 please visit
https://aka.ms/ws12
```

The actual error for `0x1bc` is:

PowerShell

```
WSL 2 requires an update to its kernel component. For information please
visit https://aka.ms/ws12kernel
```

For more information, please refer to issue [5749 ↗](#)

Cannot access WSL files from Windows

A 9p protocol file server provides the service on the Linux side to allow Windows to access the Linux file system. If you cannot access WSL using `\wsl$` on Windows, it could be because 9P did not start correctly.

To check this, you can check the start up logs using: `dmesg |grep 9p`, and this will show you any errors. A successful output looks like the following:

Bash

```
[ 0.363323] 9p: Installing v9fs 9p2000 file system support
[ 0.363336] FS-Cache: Netfs '9p' registered for caching
[ 0.398989] 9pnet: Installing 9P2000 support
```

Please see [this GitHub thread ↗](#) for further discussion on this issue.

Can't start WSL 2 distribution and only see 'WSL 2' in output

If your display language is not English, then it is possible you are seeing a truncated version of an error text.

PowerShell

```
C:\Users\me>wsl
WSL 2
```

To resolve this issue, please visit <https://aka.ms/wsl2kernel> and install the kernel manually by following the directions on that doc page.

command not found when executing Windows .exe in Linux

Users can run Windows executables like `notepad.exe` directly from Linux. Sometimes, you may hit "command not found" like below:

Bash

```
$ notepad.exe
-bash: notepad.exe: command not found
```

If there are no Win32 paths in your `$PATH`, interop isn't going to find the .exe. You can verify it by running `echo $PATH` in Linux. It's expected that you will see a Win32 path (for

example, /mnt/c/Windows) in the output. If you can't see any Windows paths then most likely your PATH is being overwritten by your Linux shell.

Here is an example that /etc/profile on Debian contributed to the problem:

Bash

```
if [ "`id -u`" -eq 0 ]; then
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
    PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
fi
```

The correct way on Debian is to remove above lines. You may also append \$PATH during the assignment like below, but this lead to some [other problems](#) with WSL and VSCode..

Bash

```
if [ "`id -u`" -eq 0 ]; then
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:$PATH"
else
    PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:$PATH"
fi
```

For more information, see issue [5296](#) and issue [5779](#).

"Error: 0x80370102 The virtual machine could not be started because a required feature is not installed."

Please enable the Virtual Machine Platform Windows feature and ensure virtualization is enabled in the BIOS.

1. Check the [Hyper-V system requirements](#)
2. If your machine is a VM, enable [nested virtualization](#) manually. Launch powershell with admin, and run the following command, replacing <VMName> with the name of the virtual machine on your host system (you can find the name in your Hyper-V Manager):

PowerShell

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

3. Please follow guidelines from your PC's manufacturer on how to enable virtualization. In general, this can involve using the system BIOS to ensure that these features are enabled on your CPU. Instructions for this process can vary from machine to machine, please see [this article](#) from Bleeping Computer for an example.
4. Restart your machine after enabling the `Virtual Machine Platform` optional component.
5. Make sure that the hypervisor launch is enabled in your boot configuration. You can validate this by running (elevated powershell):

```
PowerShell
```

```
bcdedit /enum | findstr -i hypervisorlaunchtype
```

If you see `hypervisorlaunchtype Off`, then the hypervisor is disabled. To enable it run in an elevated powershell:

```
bcdedit /set hypervisorlaunchtype Auto
```

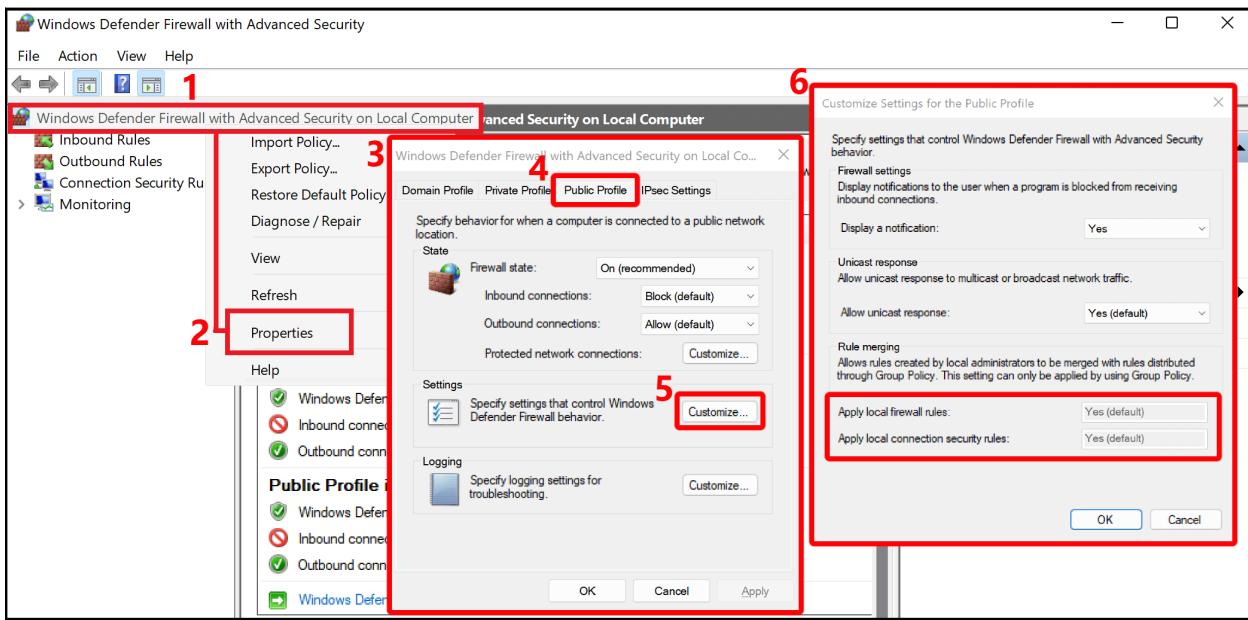
6. Additionally, if you have 3rd party hypervisors installed (Such as VMware or VirtualBox) then please ensure you have these on the latest versions which can support HyperV ([VMware 15.5.5+](#) and [VirtualBox 6+](#)) or are turned off.
7. If you are receiving this error on an Azure Virtual Machine, check to ensure that [Trusted Launch](#) is disabled. [Nested Virtualization is not supported on Azure virtual machines](#).

Learn more about how to [Configure Nested Virtualization](#) when running Hyper-V in a Virtual Machine.

WSL has no network connection on my work machine or in an Enterprise environment

Business or Enterprise environments may have [Windows Defender Firewall settings configured](#) to block unauthorized network traffic. If [local rule merging](#) is set to "No" then WSL networking will not work by default, and your administrator will need to add a firewall rule to allow it.

You can confirm local rule merging's setting by following these steps:



1. Open "Windows Defender Firewall with advanced security" (*this is different than "Windows Defender Firewall" in the Control Panel*)
2. Right-click on the "Windows Defender Firewall with advanced security on Local Computer" tab
3. Select "Properties"
4. Select the "Public Profile" tab on the new Window that opens
5. Select "Customize" under the "Settings" section
6. Check in the "Customize Settings for the Public Profile" window that opens to see if "Rule Merging" is set to "No". This will block access to WSL.

You can find instructions on how to change this Firewall setting in [Configure Hyper-V firewall](#).

WSL has no network connectivity once connected to a VPN

If after connecting to a VPN on Windows, bash loses network connectivity, try this workaround from within bash. This workaround will allow you to manually override the DNS resolution through `/etc/resolv.conf`.

1. Take a note of the DNS server of the VPN from doing `ipconfig.exe /all`
2. Make a copy of the existing `resolv.conf` `sudo cp /etc/resolv.conf /etc/resolv.conf.new`
3. Unlink the current `resolv.conf` `sudo unlink /etc/resolv.conf`
4. `sudo mv /etc/resolv.conf.new /etc/resolv.conf`
5. Edit `/etc/wsl.conf` and add this content to the file. (More info on this set up can be found in [Advanced settings configuration](#))

```
[network]
generateResolvConf=false
```

6. Open `/etc/resolv.conf` and
 - a. Delete the first line from the file which has a comment describing automatic generation
 - b. Add the DNS entry from (1) above as the very first entry in the list of DNS servers.
 - c. Close the file.

Once you have disconnected the VPN, you will have to revert the changes to `/etc/resolv.conf`. To do this, do:

1. `cd /etc`
2. `sudo mv resolv.conf resolv.conf.new`
3. `sudo ln -s ../../run/resolvconf/resolv.conf resolv.conf`

Cisco Anyconnect VPN issues with WSL in NAT mode

The Cisco AnyConnect VPN modifies routes in a way which prevents NAT from working. There is a workaround specific to WSL 2: See [Cisco AnyConnect Secure Mobility Client Administrator Guide, Release 4.10 - Troubleshoot AnyConnect ↴](#).

WSL connectivity issues with VPNs when Mirrored networking mode is on

Mirrored networking mode is currently an [experimental setting in the WSL Configuration](#). The traditional NAT networking architecture of WSL can be updated to an entirely new networking mode called "Mirrored networking mode". When the experimental `networkingMode` is set to `mirrored`, the network interfaces that you have on Windows are mirrored into Linux to improve compatibility. Learn more in the Command Line blog: [WSL September 2023 update ↴](#).

Some VPNs have been tested and confirmed to be incompatible with WSL, including:

- "Bitdefender" version 26.0.2.1
- "OpenVPN" version 2.6.501
- "Mcafee Safe Connect" version 2.16.1.124

Considerations when using autoProxy for HttpProxy Mirroring in WSL

HTTP/S proxy mirroring can be configured using the `autoProxy` setting in the [experimental section of the WSL Configuration file](#). When applying this setting, note these considerations:

- **PAC Proxy:** WSL will configure the setting in Linux by Setting the "WSL_PAC_URL" environment variable. Linux does not support PAC proxies by default.
- **Interactions with WSLENV:** User defined environment variables take precedence over those specified by this feature.

When enabled, the following apply to proxy settings on your Linux distributions:

- The Linux environment variable, `HTTP_PROXY`, is set to the one or more HTTP proxies found installed in the Windows HTTP proxy configuration.
- The Linux environment variable, `HTTPS_PROXY`, is set to the one or more HTTPS proxies found installed in the Windows HTTP proxy configuration.
- The Linux environment variable, `NO_PROXY`, is set to bypass any HTTP/S proxies found in the Windows configuration targets.
- Every environment variable, except `WSL_PAC_URL`, is set to both lower case and upper case. For example: `HTTP_PROXY` and `http_proxy`.

Learn more in the Command Line blog: [WSL September 2023 update ↗](#).

Networking considerations with DNS tunneling

When WSL can't connect to the internet, it might be because the DNS call to the Windows host is blocked. This is because the networking packet for DNS sent by the WSL VM to the Windows host is blocked by the existing networking configuration. DNS tunneling fixes this by using a virtualization feature to communicate with Windows directly, allowing the DNS name to be resolved without sending a networking packet. This feature should improve network compatibility and allow you to get better internet connectivity even if you have a VPN, specific firewall setup, or other networking configurations.

DNS Tunneling can be configured using the `dnsTunneling` setting in the [experimental section of the WSL Configuration file](#). When applying this setting, note these considerations:

- Native Docker can have connectivity issues in WSL when DNS tunneling is enabled – if the network has a policy to block DNS traffic to: 8.8.8.8

- If you use a VPN with WSL, turn on DNS tunneling. Many VPNs use NRPT policies, which are only applied to WSL DNS queries when DNS tunneling is enabled.
- The `/etc/resolv.conf` file in your Linux distribution has a 3 DNS servers maximum limitation, while Windows may use more than 3 DNS servers. Using DNS tunneling removes this limitation – all Windows DNS servers can now be used by Linux.
- WSL will use Windows DNS suffixes in the following order (similar to the order used by the Windows DNS client):
 1. Global DNS suffixes
 2. Supplemental DNS suffixes
 3. Per-interface DNS suffixes
 4. If DNS encryption (DoH, DoT) is enabled on Windows, encryption will be applied to DNS queries from WSL. If users want to enable DoH, DoT inside Linux, they need to disable DNS tunneling.
- DNS queries from Docker containers (either Docker Desktop or native Docker running in WSL) will bypass DNS tunneling. DNS tunneling cannot be leveraged to apply host DNS settings and policies to Docker DNS traffic.
- Docker Desktop has its own way (different from DNS tunneling) of applying host DNS settings and policies to DNS queries from Docker containers.

Learn more in the Command Line blog: [WSL September 2023 update ↗](#).

Issues with steering the inbound traffic received by the Windows host to the WSL Virtual Machine

When using Mirrored networking mode (the experimental `networkingMode` set to `mirrored`), some inbound traffic received by the Windows host will never be steered to the Linux VM. This traffic is as follows:

- UDP port 68 (DHCP)
- TCP port 135 (DCE endpoint resolution)
- UDP port 5353 (mDNS)
- TCP port 1900 (UPnP)
- TCP port 2869 (SSDP)
- TCP port 5004 (RTP)
- TCP port 3702 (WSD)
- TCP port 5357 (WSD)
- TCP port 5358 (WSD)

WSL will automatically configure certain Linux networking settings when using mirrored networking mode. Any user configurations of these settings while using mirrored

networking mode is not supported. Here is the list of settings WSL will configure:

[Expand table](#)

Setting Name	Value
https://sysctl-explorer.net/net/ipv4/accept_local/	Enabled (1)
https://sysctl-explorer.net/net/ipv4/route_localnet/	Enabled (1)
https://sysctl-explorer.net/net/ipv4/rp_filter/	Disabled (0)
https://sysctl-explorer.net/net/ipv6/accept_ra/	Disabled (0)
https://sysctl-explorer.net/net/ipv6/autoconf/	Disabled (0)
https://sysctl-explorer.net/net/ipv6/use_tempaddr/	Disabled (0)
addr_gen_mode	Disabled (0)
disable_ipv6	Disabled (0)
https://sysctl-explorer.net/net/ipv4/arp_filter/	Enabled (1)

Docker container issues in WSL2 with Mirrored networking mode enabled when running under the default networking namespace

There is a known issue in which Docker Desktop containers with published ports (`docker run -publish/-p`) will fail to be created. The WSL team is working with the Docker Desktop team to address this issue. To work around the issue, use the host's networking namespace in the Docker container. Set the network type via the "`--network host`" option used in the "docker run" command. An alternative workaround is to list the published port number in the `ignoredPorts` setting of the [experimental section in the WSL Configuration file](#).

Docker container issues when its Network Manager is running

There is a known issue with Docker containers which have the Network Manager service running. Symptoms include failures when trying to make loopback connections to the host. It is recommended to stop the Network Manager service for WSL networking to be configured properly.

```
sudo systemctl disable network-manager.service
```

DNS suffixes in WSL

Depending on the configurations in the .wslconfig file, WSL will have the following behavior wrt DNS suffixes:

When networkingMode is set to NAT:

Case 1) By default no DNS suffix is configured in Linux

Case 2) If DNS tunneling is enabled (dnsTunneling is set to true in .wslconfig) All Windows DNS suffixes are configured in Linux, in the "search" setting of /etc/resolv.conf

The suffixes are configured in /etc/resolv.conf in the following order, similar to the order in which Windows DNS client tries suffixes when resolving a name: global DNS suffixes first, then supplemental DNS suffixes, then per-interface DNS suffixes.

When there is a change in the Windows DNS suffixes, that change will be automatically reflected in Linux

Case 3) If DNS tunneling is disabled and SharedAccess DNS proxy is disabled (dnsTunneling is set to false and dnsProxy is set to false in .wslconfig) A single DNS suffix is configured in Linux, in the "domain" setting of /etc/resolv.conf

When there is a change in the Windows DNS suffixes, that change is not reflected in Linux

The single DNS suffix configured in Linux is chosen from the per-interface DNS suffixes (global and supplemental suffixes are ignored)

If Windows has multiple interfaces, a heuristic is used to choose the single DNS suffix that will be configured in Linux. For example if there is a VPN interface on Windows, the suffix is chosen from that interface. If no VPN interface is present, the suffix is chosen from the interface that is most likely to give Internet connectivity.

When networkingMode is set to Mirrored:

All Windows DNS suffixes are configured in Linux, in the "search" setting of /etc/resolv.conf

The suffixes are configured in /etc/resolv.conf in the same order as in case 2) from NAT mode

When there is a change in the Windows DNS suffixes, that change will be automatically reflected in Linux

Note: supplemental DNS suffixes can be configured in Windows using [SetInterfaceDnsSettings - Win32 apps | Microsoft Learn](#), with the flag `DNS_SETTING_SUPPLEMENTAL_SEARCH_LIST` set in the `Settings` parameter

Troubleshooting DNS in WSL

The default DNS configuration when WSL starts a container in NAT mode is to have the NAT device on the Windows Host serve as the DNS 'server' for the WSL container. When DNS queries are sent from the WSL container to that NAT device on the Windows Host, the DNS packet is forwarded from the NAT device to the shared access service on the Host; the response is sent in the reverse direction back to the WSL container. This packet forwarding process to shared access requires a Firewall rule to allow that inbound DNS packet, which is created by the HNS service when WSL initially asks HNS to create the NAT virtual network for its WSL container.

Due to this NAT - shared access design, there are a few known configurations which can break name resolution from WSL.

1. An Enterprise can push policy that does not allow locally defined Firewall rules, only allowing Enterprise-policy defined rules.

When this is set by an Enterprise, the HNS-created Firewall rule is ignored, as it's a locally defined rule. For this configuration to work the Enterprise must create a Firewall rule to allow UDP port 53 to the shared access service, or WSL can be set to use DNS Tunneling. One can see if this is configured to not allow locally defined Firewall rules by running the following. Note that this will show settings for all 3 profiles: Domain, Private, and Public. If it's set on any profile, then packets will be blocked if the WSL vNIC is assigned that profile (default is Public). This is only a snippet of the first Firewall profile that is returned in Powershell:

PowerShell

```
PS C:\> Get-NetFirewallProfile -PolicyStore ActiveStore
Name          : Domain
Enabled       : True
DefaultInboundAction : Block
DefaultOutboundAction : Allow
AllowInboundRules   : True
AllowLocalFirewallRules : False
```

`AllowLocalFirewallRules:False` means the locally defined firewall rules, like that by HNS, will not be applied or used.

2. And Enterprise can push down Group Policy and MDM policy settings that block all inbound rules.

These settings override any Allow-Inbound Firewall rule. This setting will thus block the HNS-created UDP Firewall rule, and thus will prevent WSL from resolving names. For this configuration to work, **WSL must be set to use DNS Tunneling**. This setting will always block the NAT DNS proxy.

From Group Policy:

Computer Configuration \ Administrative Templates \ Network \ Network Connections \ Windows Defender Firewall \ Domain Profile | Standard Profile

"Windows Defender Firewall: Do not allow exceptions" - Enabled

From MDM Policy:

`./Vendor/MSFT/Firewall/MdmStore/PrivateProfile/Shielded`

`./Vendor/MSFT/Firewall/MdmStore/DomainProfile/Shielded`

`./Vendor/MSFT/Firewall/MdmStore/PublicProfile/Shielded`

One can see if this is configured to not allow any inbound Firewall rules by running the following (see above caveats on Firewall Profiles). This is only a snippet of the first Firewall profile that is returned in Powershell:

powerShell

```
PS C:\> Get-NetFirewallProfile -PolicyStore ActiveStore
Name          : Domain
Enabled       : True
DefaultInboundAction : Block
DefaultOutboundAction : Allow
AllowInboundRules : False
```

`AllowInboundRules: False` means that no inbound Firewall rules will be applied.

3. A user goes through the Windows Security setting apps and checks the control for "Blocks all incoming connections, including those in the list of allowed apps."

Windows supports a user-opt-in for the same setting that can be applied by an Enterprise referenced in #2 above. Users can open the "Windows Security" settings

page, selects the “Firewall & network protection” option, selects the Firewall Profile they want to configure (Domain, Private, or Public), and under “Incoming connections” check the control labeled “Blocks all incoming connections, including those in the list of allowed apps.”

If this is set for the Public profile (this is the default profile for the WSL vNIC), the Firewall rule created by HNS to allow the UDP packets to shared access will be blocked.

This must be unchecked for the NAT DNS proxy configuration to work from WSL, **or WSL can be set to use DNS Tunneling.**

4. The HNS Firewall rule to allow the DNS packets to shared access can become invalid, referencing a previous WSL interface identifier. This is a flaw within HNS which has been fixed with the latest Windows 11 release. On earlier releases, if this occurs, it’s not easily discoverable, but it has a simple work around:

- Stop WSL

```
wsl.exe -shutdown
```

- Delete the old HNS Firewall rule. This Powershell command should work in most cases:

```
Get-NetFirewallRule -Name "HNS*" | Get-NetFirewallPortFilter | where Protocol -eq UDP | where LocalPort -eq 53 | Remove-NetFirewallRule
```

- Remove all HNS endpoints. Note: if HNS is used to manage other containers, such as MDAG or Windows Sandbox, those should also be stopped.

```
hnsdiag.exe delete all
```

- Reboot or restart the HNS service

```
Restart-Service hns
```

- When WSL is restarted, HNS will create new Firewall rules, correctly targeting the WSL interface.

Troubleshooting Network Access Issues on Windows

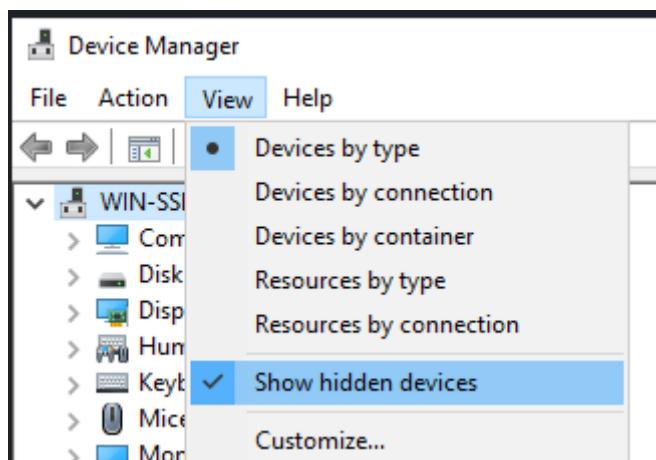
If you have no network access, it might be due to a misconfiguration. Please see if the FSE driver is running: ‘sc queryex FSE’. If that does not show FSE running, please check if the PortTrackerEnabledMode registry value exists under this registry key: reg query HKLM\System\CurrentControlSet\Services\Tcpip\Parameters. If FSE is not running or

installed, and PortTrackerEnabledMode exists, please delete that registry value and reboot

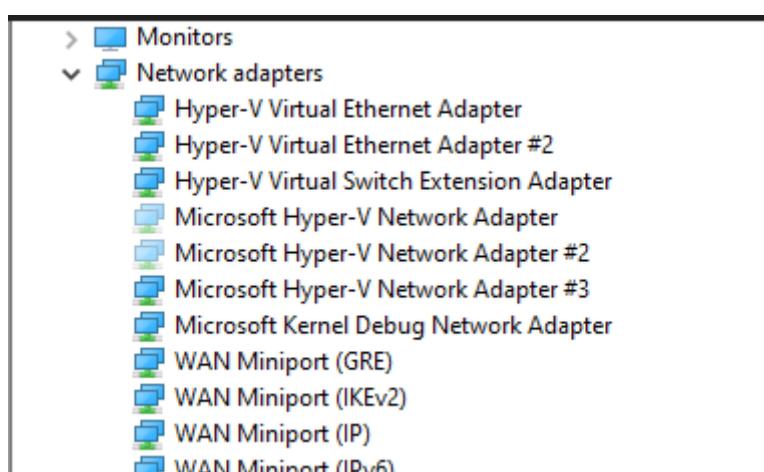
Manual way to delete phantom adapters

Ghost adapters, or phantom Plug and Play (PnP) devices, refer to hardware components that appear in your system but are not physically connected. These “ghost” devices can cause confusion and clutter in your system settings. If you see ghost adapters when running WSL in a Virtual Machine (VM), follow these manual steps to find and delete these Phantom PnP devices. Microsoft is working on an automated solution that will not require manual intervention. More information will be coming soon.

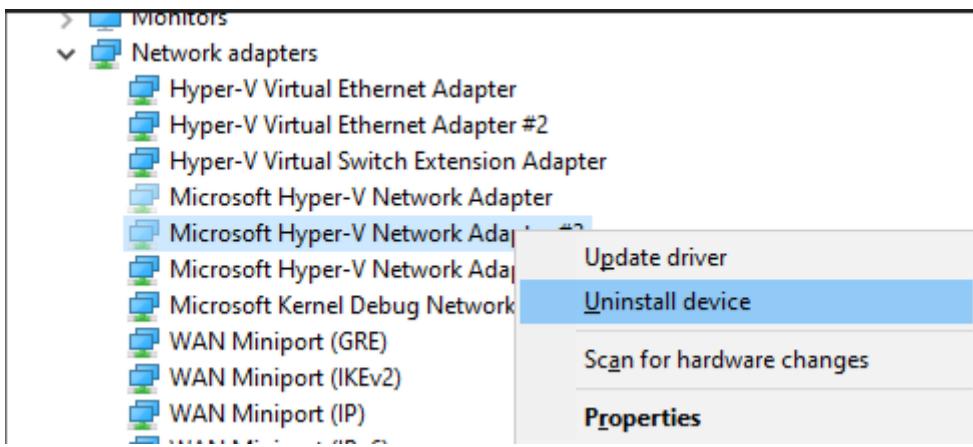
1. Open Device Manager
2. View > Show hidden devices



3. Open Network adapters



4. Right-click over the Ghosted network adapter and select Uninstall Device



Starting WSL or installing a distribution returns an error code

Follow the instructions to [Collect WSL logs](#) in the WSL repo on GitHub to collect detailed logs and file an issue on our GitHub.

Updating WSL

There are two components of Windows Subsystem for Linux that can require updating.

1. To update the Windows Subsystem for Linux itself, use the command `wsl --update` in PowerShell or CMD.
2. To update the specific Linux distribution user binaries, use the command: `apt-get update | apt-get upgrade` in the Linux distribution that you are seeking to update.

Apt-get upgrade errors

Some packages use features that we haven't implemented yet. `udev`, for example, isn't supported yet and causes several `apt-get upgrade` errors.

To fix issues related to `udev`, follow the following steps:

1. Write the following to `/usr/sbin/policy-rc.d` and save your changes.

```
Bash  
  
#!/bin/sh  
exit 101
```

2. Add execute permissions to `/usr/sbin/policy-rc.d`:

```
Bash
```

```
chmod +x /usr/sbin/policy-rc.d
```

3. Run the following commands:

```
Bash
```

```
dpkg-divert --local --rename --add /sbin/initctl  
ln -s /bin/true /sbin/initctl
```

"Error: 0x80040306" on installation

This has to do with the fact that we do not support legacy console. To turn off legacy console:

1. Open cmd.exe
2. Right click title bar -> Properties -> Uncheck Use legacy console
3. Click OK

"Error: 0x80040154" after Windows update

The Windows Subsystem for Linux feature may be disabled during a Windows update. If this happens the Windows feature must be re-enabled. Instructions for enabling the Windows Subsystem for Linux can be found in the [Manual Installation Guide](#).

Changing the display language

WSL install will try to automatically change the Ubuntu locale to match the locale of your Windows install. If you do not want this behavior you can run this command to change the Ubuntu locale after install completes. You will have to relaunch bash.exe for this change to take effect.

The below example changes to locale to en-us :

```
Bash
```

```
sudo update-locale LANG=en_US.UTF8
```

Installation issues after Windows system restore

1. Delete the `%windir%\System32\Tasks\Microsoft\Windows\Windows Subsystem for Linux` folder.

Note: Do not do this if your optional feature is fully installed and working.

2. Enable the WSL optional feature (if not already)
3. Reboot
4. `lxrun /uninstall /full`
5. Install bash

No internet access in WSL

Some users have reported issues with specific firewall applications blocking internet access in WSL. The firewalls reported are:

1. Kaspersky
2. AVG
3. Avast
4. Symantec Endpoint Protection

In some cases turning off the firewall allows for access. In some cases simply having the firewall installed looks to block access.

If you are using Microsoft Defender Firewall, unchecking "Blocks all incoming connections, including those in the list of allowed apps." allows for access.

Permission Denied error when using ping

For [Windows Anniversary Update, version 1607](#), administrator privileges in Windows are required to run ping in WSL. To run ping, run Bash on Ubuntu on Windows as an administrator, or run `bash.exe` from a CMD/PowerShell prompt with administrator privileges.

For later versions of Windows, [Build 14926+](#), administrator privileges are no longer required.

Bash is hung

If while working with bash, you find that bash is hung (or deadlocked) and not responding to inputs, help us diagnose the issue by collecting and reporting a memory dump. Note that these steps will crash your system. Do not do this if you are not comfortable with that or save your work prior to doing this.

To collect a memory dump

1. Change the memory dump type to "complete memory dump". While changing the dump type, take a note of your current type.
2. Use the [steps](#) to configure crash using keyboard control.
3. Repro the hang or deadlock.
4. Crash the system using the key sequence from (2).
5. The system will crash and collect the memory dump.
6. Once the system reboots, report the memory.dmp to secure@microsoft.com. The default location of the dump file is %SystemRoot%\memory.dmp or C:\Windows\memory.dmp if C: is the system drive. In the email, note that the dump is for the WSL or Bash on Windows team.
7. Restore the memory dump type to the original setting.

Check your build number

To find your PC's architecture and Windows build number, open **Settings > System > About**

Look for the **OS Build** and **System Type** fields.

The screenshot shows two sections of the Windows Settings app:

- Device specifications:**
 - Device name: [REDACTED] Desktop-PC
 - Processor: [REDACTED]
 - Installed RAM: [REDACTED]
 - Device ID: [REDACTED]
 - Product ID: [REDACTED]
 - System type:** 64-bit operating system, x64-based processor
 - Pen and touch: No pen or touch input is available for this display
- Windows specifications:**
 - Edition: Windows 11 Pro
 - Version: Dev
 - Installed on: 26.02.2022
 - OS build:** 22563.1
 - Experience: Windows Feature Experience Pack 1000.22563.1.0
 - Microsoft Services Agreement
 - Microsoft Software Licence Terms

To find your Windows Server build number, run the following in PowerShell:

```
PowerShell
```

```
systeminfo | Select-String "^\$OS Name", "^\$OS Version"
```

Confirm WSL is enabled

You can confirm that the Windows Subsystem for Linux is enabled by running the following in an elevated PowerShell window:

```
PowerShell
```

```
Get-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

OpenSSH-Server connection issues

Trying to connect your SSH server failed with the following error: "Connection closed by 127.0.0.1 port 22".

1. Make sure your OpenSSH Server is running:

```
Bash
```

```
sudo service ssh status
```

and you've followed this tutorial: <https://ubuntu.com/server/docs/service-openssh> ↗

2. Stop the sshd service and start sshd in debug mode:

```
Bash
```

```
sudo service ssh stop  
sudo /usr/sbin/sshd -d
```

3. Check the startup logs and make sure HostKeys are available and you don't see log messages such as:

```
BASH
```

```
debug1: sshd version OpenSSH_7.2, OpenSSL 1.0.2g 1 Mar 2016  
debug1: key_load_private: incorrect passphrase supplied to decrypt  
private key  
debug1: key_load_public: No such file or directory  
Could not load host key: /etc/ssh/ssh_host_rsa_key
```

```
debug1: key_load_private: No such file or directory
debug1: key_load_public: No such file or directory
Could not load host key: /etc/ssh/ssh_host_dsa_key
debug1: key_load_private: No such file or directory
debug1: key_load_public: No such file or directory
Could not load host key: /etc/ssh/ssh_host_ecdsa_key
debug1: key_load_private: No such file or directory
debug1: key_load_public: No such file or directory
Could not load host key: /etc/ssh/ssh_host_ed25519_key
```

If you do see such messages and the keys are missing under `/etc/ssh/`, you will have to regenerate the keys or just purge&install openssh-server:

BASH

```
sudo apt-get purge openssh-server
sudo apt-get install openssh-server
```

"The referenced assembly could not be found." when enabling the WSL optional feature

This error is related to being in a bad install state. Please complete the following steps to try and fix this issue:

- If you are running the enable WSL feature command from PowerShell, try using the GUI instead by opening the start menu, searching for 'Turn Windows features on or off' and then in the list select 'Windows Subsystem for Linux' which will install the optional component.
- Update your version of Windows by going to Settings, Updates, and clicking 'Check for Updates'
- If both of those fail and you need to access WSL please consider upgrading in place by reinstalling Windows using installation media and selecting 'Keep Everything' to ensure your apps and files are preserved. You can find instructions on how to do so at the [Reinstall Windows 10 page ↗](#).

Correct (SSH related) permission errors

If you're seeing this error:

Bash

```
@@@@@@@WARNING: UNPROTECTED PRIVATE KEY FILE!@@@@@@@  
Permissions 0777 for '/home/user/.ssh/private-key.pem' are too open.
```

To fix this, append the following to the `/etc/wsl.conf` file:

Bash

```
[automount]  
enabled = true  
options = metadata,uid=1000,gid=1000,umask=0022
```

Please note that adding this command will include metadata and modify the file permissions on the Windows files seen from WSL. Please see the [File System Permissions](#) for more information.

Fails to use WSL remotely by using OpenSSH on Windows

If you are using `openssh-server` on Windows and trying to access WSL remotely, you may see this error:

Windows Command Prompt

```
The file cannot be accessed by the system.
```

It's a [known issue](#), when using the Store version of WSL. You can work around this today by using WSL 1, or by using the in-Windows version of WSL. See <https://aka.ms/wslstoreinfo> for more info.

Running Windows commands fails inside a distribution

Some distributions [available in Microsoft Store](#) are yet not fully compatible to run Windows commands out of the box. If you get an error `-bash: powershell.exe: command not found` running `powershell.exe /c start .` or any other Windows command, you can resolve it following these steps:

1. In your WSL distribution run `echo $PATH`.

If it does not include: `/mnt/c/Windows/system32` something is redefining the standard PATH variable.

2. Check profile settings with `cat /etc/profile`.

If it contains assignment of the PATH variable, edit the file to comment out PATH

assignment block with a # character.

3. Check if wsl.conf is present `cat /etc/wsl.conf` and make sure it does not contain `appendWindowsPath=false`, otherwise comment it out.
4. Restart distribution by typing `wsl -t` followed by distribution name or run `wsl --shutdown` either in cmd or PowerShell.

Unable to boot after installing WSL 2

We are aware of an issue affecting users where they are unable to boot after installing WSL 2. While we fully diagnose those issue, users have reported that [changing the buffer size](#) or [installing the right drivers](#) can help address this. Please view this [GitHub issue](#) to see the latest updates on this issue.

WSL 2 errors when ICS is disabled

Internet Connection Sharing (ICS) is a required component of WSL 2. The ICS service is used by the Host Network Service (HNS) to create the underlying virtual network which WSL 2 relies on for NAT, DNS, DHCP, and host connection sharing.

Disabling the ICS service (SharedAccess) or disabling ICS through group policy will prevent the WSL HNS network from being created. This will result in failures when creating a new WSL version 2 image, and the following error when trying to convert a version 1 image to version 2.

Console

There are no more endpoints available from the endpoint mapper.

Systems that require WSL 2 should leave the ICS service (SharedAccess) in its default start state, Manual (Trigger Start), and any policy that disables ICS should be overwritten or removed. While disabling the ICS service will break WSL 2, and we do not recommend disabling ICS, portions of ICS can be disabled [using these instructions](#)

Using older versions of Windows and WSL

There are several differences to note if you're running an older version of Windows and WSL, like the Windows 10 Creators Update (Oct 2017, Build 16299) or Anniversary Update (Aug 2016, Build 14393). We recommend that you [update to the latest Windows version](#), but if that's not possible, we have outlined some of the differences below.

Interoperability command differences:

- `bash.exe` has been replaced with `wsl.exe`. Linux commands can be run from the Windows Command Prompt or from PowerShell, but for early Windows versions, you may need to use the `bash` command. For example: `C:\temp> bash -c "ls -la"`. The WSL commands passed into `bash -c` are forwarded to the WSL process without modification. File paths must be specified in the WSL format and care must be taken to escape relevant characters. For example: `C:\temp> bash -c "ls -la /proc/cpuinfo"` or `C:\temp> bash -c "ls -la \"/mnt/c/Program Files\""`.
- To see what commands are available for a particular distribution, run `[distro.exe] /?`. For example, with Ubuntu: `C:\> ubuntu.exe /?`.
- Windows path is included in the WSL `$PATH`.
- When calling a Windows tool from a WSL distribution in an earlier version of Windows 10, you will need to specify the directory path. For example, to call the Windows Notepad app from your WSL command line, enter:
`/mnt/c/Windows/System32/notepad.exe`
- To change the default user to `root` use this command in PowerShell: `c:\> 1xrun /setdefaultuser root` and then run Bash.exe to log in: `C:\> bash.exe`. Reset your password using the distributions password command: `$ passwd username` and then close the Linux command line: `$ exit`. From Windows command prompt or Powershell, reset your default user back to your normal Linux user account: `C:\> 1xrun.exe /setdefaultuser username`.

Uninstall legacy version of WSL

If you originally installed WSL on a version of Windows 10 prior to Creators update (Oct 2017, Build 16299), we recommend that you migrate any necessary files, data, etc. from the older Linux distribution you installed, to a newer distribution installed via the Microsoft Store. To remove the legacy distribution from your machine, run the following from a Command Line or PowerShell instance: `wsl --unregister Legacy`. You also have the option to manually remove the older legacy distribution by deleting the `%localappdata%\lxss\` folder (and all it's sub-contents) using Windows File Explorer or with PowerShell: `rm -Recurse $env:localappdata/lxss/`.

 Collaborate with us on
GitHub

The source for this content can
be found on GitHub, where you
can also create and review



Windows Subsystem for
Linux feedback

Windows Subsystem for Linux is an
open source project. Select a link to
provide feedback:

issues and pull requests. For more information, see [our contributor guide](#).

 Open a documentation issue

 Provide product feedback

Release Notes for Windows Subsystem for Linux

Article • 06/27/2022

Build 21364

For general Windows information on build 21364 visit the [Windows blog ↗](#).

- GUI apps are now available! For more information see [this blog post ↗](#).
- Resolve error when accessing files via \\wsl.localhost\.
- Fix potential deadlock in LxssManager service.

Build 21354

For general Windows information on build 21354 visit the [Windows blog ↗](#).

- Switch the \wsl prefix to \wsl.localhost to avoid issues when there is a machine on the network named "wsl". \wsl\$ will continue to work.
- Enable Linux quick access icon for wow processes.
- Update issue where version 2 was always being passed via wslapi RegisterDistribution.
- Change the fmask of the /usr/lib/wsl/lib directory to 222 so files are marked as executable [GH 3847]
- Fix wsl service crash if Virtual Machine Platform is not enabled.

Build 21286

For general Windows information on build 21286 visit the [Windows blog ↗](#).

- Introduce wsl.exe --cd command to set current working directory of a command.
- Improve mapping of NTSTATUS to Linux error codes. [GH 6063]
- Improve wsl.exe --mount error reporting.
- Added an option to /etc/wsl.conf to enable start up commands:

Console

```
[boot]
command=<string>
```

Build 20226

For general Windows information on build 20226 visit the [Windows blog ↗](#).

- Fix crash in LxssManager service. [GH 5902]

Build 20211

For general Windows information on build 20211 visit the [Windows blog ↗](#).

- Introduce `wsl.exe --mount` for mounting physical or virtual disks. For more information see [Access Linux filesystems in Windows and WSL 2 ↗](#).
- Fix crash in LxssManager service when checking if the VM is idle. [GH 5768]
- Support for compressed VHD files. [GH 4103]
- Ensure that Linux user mode libs installed to `c:\windows\system32\lxss\lib` are preserved across OS upgrade. [GH 5848]
- Added the ability to list available distributions that can be installed with `wsl --install --list-distributions`.
- WSL instances are now terminated when the user logs off.

Build 20190

For general Windows information on build 20190 visit the [Windows blog ↗](#).

- Fix bug preventing WSL1 instances from launching. [GH 5633]
- Fix hang when redirecting Windows process output. [GH 5648]
- Add `%userprofile%\wslconfig` option to control the VM idle timeout (`wsl2.vmIdleTimeout=<time_in_ms>`).
- Support launching app execution aliases from WSL.
- Added support for installing the WSL2 kernel and distributions to `wsl.exe --install`.

Build 20175

For general Windows information on build 20175 visit the [Windows blog ↗](#).

- Adjust default memory assignment of WSL2 VM to be 50% of host memory or 8GB, whichever is less [GH 4166].
- Change `\wsl$` prefix to `\wsl` to support URI parsing. The old `\wsl$` path is still supported.
- Enable nested virtualization for WSL2 by default on amd64. You can disable this via `%userprofile%\wslconfig` (`[wsl2] nestedVirtualization=false`).

- Make wsl.exe --update demand start Microsoft Update.
- Support renaming over a read-only file in DrvFs.
- Ensure error messages are always printed in the correct codepage.

Build 20150

For general Windows information on build 20150 visit the [Windows blog ↗](#).

- WSL2 GPU compute see [Windows blog ↗](#) for more information.
- Introduce wsl.exe --install command line option to easily set up WSL.
- Introduce wsl.exe --update command line option to manage updates to the WSL2 kernel.
- Set WSL2 as the default.
- Increase WSL2 vm graceful shutdown timeout.
- Fix virtio-9p race condition when mapping device memory.
- Don't run an elevated 9p server if UAC is disabled.

Build 19640

For general Windows information on build 19640 visit the [Windows blog ↗](#).

- [WSL2] Stability improvements for virtio-9p (drvfs).

Build 19555

For general Windows information on build 19555 visit the [Windows blog ↗](#).

- [WSL2] Use a memory cgroup to limit the amount of memory used by install and conversion operations [GH 4669]
- Make wsl.exe present when the Windows Subsystem for Linux optional component is not enabled to improve feature discoverability.
- Change wsl.exe to print help text if the WSL optional component is not installed
- Fix race condition when creating instances
- Create wslclient.dll that contains all command line functionality
- Prevent crash during LxssManagerUser service stop
- Fix wslapi.dll fast fail when distroName parameter is NULL

Build 19041

For general Windows information on build 19041 visit the [Windows blog ↗](#).

- [WSL2] Clear the signal mask before launching the processes
- [WSL2] Update Linux kernel to 4.19.84
- Handle creation of /etc/resolv.conf symlink when the symlink is non-relative

Build 19028

For general Windows information on build 19028 visit the [Windows blog ↗](#).

- [WSL2] Update Linux kernel to 4.19.81
- [WSL2] Change the default permission of /dev/net/tun to 0666 [GH 4629]
- [WSL2] Tweak default amount of memory assigned to Linux VM to be 80% of host memory
- [WSL2] fix interop server to handle requests with a timeout so bad callers cannot hang the server

Build 19018

For general Windows information on build 19018 visit the [Windows blog ↗](#).

- [WSL2] Use cache=mmap as the default for 9p mounts to fix dotnet apps
- [WSL2] Fixes for localhost relay [GH 4340]
- [WSL2] Introduce a cross-distro shared tmpfs mount for sharing state between distros
- Fix restoring persistent network drive for \\wsl\$

Build 19013

For general Windows information on build 19013 visit the [Windows blog ↗](#).

- [WSL2] Improve memory performance of WSL utility VM. Memory that is no longer in use will be freed back to the host.
- [WSL2] Update kernel version to 4.19.79. (add CONFIG_HIGH_RES_TIMERS, CONFIG_TASK_XACCT, CONFIG_TASK_IO_ACCOUNTING, CONFIG_SCHED_HRTICK, and CONFIG_BRIDGE_VLAN_FILTERING).
- [WSL2] Fix input relay to handle cases where stdin is a pipe handle that is not closed [GH 4424]
- Make the check for \\wsl\$ case-insensitive.

Console

```
[ws12]
pageReporting = <bool>      # Enable or disable the free memory page reporting
```

```
feature (default true).  
idleThreshold = <integer> # Set the idle threshold for memory compaction, 0  
disables the feature (default 1).
```

Build 19002

For general Windows information on build 19002 visit the [Windows blog ↗](#).

- [WSL] Fix issue with handling of some Unicode characters:
[https://github.com/microsoft/terminal/issues/2770 ↗](https://github.com/microsoft/terminal/issues/2770)
- [WSL] Fix rare cases where distros could be unregistered if launched immediately after a build-to-build upgrade.
- [WSL] Fix minor issue with wsl.exe --shutdown where instance idle timers were not cancelled.

Build 18995

For general Windows information on build 18995 visit the [Windows blog ↗](#).

- [WSL2] Fix an issue where DrvFs mounts stopped working after an operation was interrupted (e.g. ctrl-c) [GH 4377]
- [WSL2] Fix handling of very large hvsocket messages [GH 4105]
- [WSL2] Fix issue with interop when stdin is a file [GH 4475]
- [WSL2] Fix service crash when unexpected network state is encountered [GH 4474]
- [WSL2] Query the distro name from the interop server if the current process does not have the environment variable
- [WSL2] Fix issue with interop whe stdin is a file
- [WSL2] Update Linux kernel version to 4.19.72
- [WSL2] Add ability to specify additional kernel command line parameters via .wslconfig

```
[wsl2]  
kernelCommandLine = <string> # Additional kernel command line arguments
```

Build 18990

For general Windows information on build 18990 visit the [Windows blog ↗](#).

- Improve the performance for directory listings in \\wsl\$

- [WSL2] Inject additional boot entropy [GH 4416]
- [WSL2] Fix for Windows interop when using su / sudo [GH 4465]

Build 18980

For general Windows information on build 18980 visit the [Windows blog ↗](#).

- Fix reading symlinks that deny FILE_READ_DATA. This includes all the symlinks Windows creates for backwards compatibility such as "C:\Document and Settings" and a bunch of symlinks in the user profile directory
- Make unexpected filesystem state non-fatal [GH 4334, 4305]
- [WSL2] Add support for arm64 if your CPU / firmware supports virtualization
- [WSL2] Allow unprivileged users to view kernel log
- [WSL2] Fix output relay when stdout / stderr sockets have been closed [GH 4375]
- [WSL2] Support battery and AC adapter passthrough
- [WSL2] Update Linux kernel to 4.19.67
- Add the ability to set default username in /etc/wsl.conf:

```
[user]
default=<string>
```

Build 18975

For general Windows information on build 18975 visit the [Windows blog ↗](#).

- [WSL2] Fixed a number of localhost reliability issues [GH 4340]

Build 18970

For general Windows information on build 18970 visit the [Windows blog ↗](#).

- [WSL2] Sync time with host time when system resumes from sleep state [GH 4245]
- [WSL2] Create NT symlinks on the Windows volumes when possible.
- [WSL2] Create distros in UTS, IPC, PID, and Mount namespaces.
- [WSL2] Fix localhost port relay when server binds to localhost directly [GH 4353]
- [WSL2] Fix interop when output is redirected [GH 4337]
- [WSL2] Support translating absolute NT symlinks.
- [WSL2] Update kernel to 4.19.59
- [WSL2] Properly set subnet mask for eth0.

- [WSL2] Change logic to break out of console worker loop when exit event is signaled.
- [WSL2] Eject distribution vhd when the distro is not running.
- [WSL2] Fix config parsing library to correctly handle empty values.
- [WSL2] Support Docker Desktop by creating cross distro mounts. A distro can opt-in to this behavior by adding the following line to the /etc/wsl.conf file:

```
[automount]
crossDistro = true
```

Build 18945

For general Windows information on build 18945 visit the [Windows blog ↗](#).

WSL

- [WSL2] Allow listening tcp sockets in WSL2 to be accessible from the host by using localhost:port
- [WSL2] Fixes for install / conversion failures and additional diagnostics to track down future issues [GH 4105]
- [WSL2] Improve diagnosability of WSL2 network issues
- [WSL2] Update kernel version to 4.19.55
- [WSL2] Update kernel with config options required for docker [GH 4165]
- [WSL2] Increase the number of CPUs assigned to the lightweight utility VM to be the same as the host (was previously capped at 8 by CONFIG_NR_CPUS in the kernel config) [GH 4137]
- [WSL2] Create a swap file for the WSL2 lightweight VM
- [WSL2] Allow user mounts to be visible via \\wsl\$\\distro (for example sshfs) [GH 4172]
- [WSL2] Improve 9p filesystem performance
- [WSL2] Ensure vhd ACL does not grow unbounded [GH 4126]
- [WSL2] Update kernel config to support squashfs and xt_conntrack [GH 4107, 4123]
- [WSL2] Fix for interop.enabled /etc/wsl.conf option [GH 4140]
- [WSL2] Return ENOTSUP if the file system does not support EAs
- [WSL2] Fix CopyFile hang with \\wsl\$
- Switch default umask to 0022 and add filesystem.umask setting to /etc/wsl.conf
- Fix wslpath to properly resolve symlinks, this was regressed in 19h1 [GH 4078]

- Introduce %UserProfile%\wslconfig file for tweaking WSL2 settings

```
[wsl2]
kernel=<path>          # An absolute Windows path to a custom Linux
kernel.
memory=<size>           # How much memory to assign to the WSL2 VM.
processors=<number>       # How many processors to assign to the WSL2 VM.
swap=<size>              # How much swap space to add to the WSL2 VM. 0
for no swap file.
swapFile=<path>          # An absolute Windows path to the swap vhd.
localhostForwarding=<bool> # Boolean specifying if ports bound to wildcard
or localhost in the WSL2 VM should be connectable from the host via
localhost:port (default true).

# <path> entries must be absolute Windows paths with escaped backslashes,
for example C:\\Users\\Ben\\kernel
# <size> entries must be size followed by unit, for example 8GB or 512MB
```

Build 18917

For general Windows information on build 18917 visit the [Windows blog ↗](#).

WSL

- WSL 2 is now available! Please see [blog ↗](#) for more details.
- Fix a regression where launching Windows processes via symlinks did not work correctly [GH 3999]
- Add wsl.exe --list --verbose, wsl.exe --list --quiet, and wsl.exe --import --version options to wsl.exe
- Add wsl.exe --shutdown option
- Plan 9: Allow opening a directory for write to succeed

Build 18890

For general Windows information on build 18890 visit the [Windows blog ↗](#).

WSL

- Non-blocking socket leak [GH 2913]
- EOF input to terminal can block subsequent reads [GH 3421]
- Update resolv.conf header to refer to wsl.conf [discussed in GH 3928]

- Deadlock in epoll delete code [GH 3922]
- Handle spaces in arguments to --import and --export [GH 3932]
- Extending mmap'd files does not work properly [GH 3939]
- Fix issue with ARM64 \\wsl\$ access not working properly
- Add better default icon for wsl.exe

Build 18342

For general Windows information on build 18342 visit the [Windows blog ↗](#).

WSL

- We've added the ability for users to access Linux files in a WSL distro from Windows. These files can be accessed through the command line, and also Windows apps, like file explorer, VSCode, etc. can interact with these files. Access your files by navigating to \\wsl\$\\<distro_name>, or see a list of running distributions by navigating to \\wsl\$
- Add additional CPU info tags and fix Cpus_allowed[_list] values [GH 2234]
- Support exec from non-leader thread [GH 3800]
- Treat configuration update failures as non-fatal [GH 3785]
- Update binfmt to properly handle offsets [GH 3768]
- Enable mapping network drives for Plan 9 [GH 3854]
- Support Windows -> Linux and Linux -> Windows path translation for bind mounts
- Create read-only sections for mappings on files opened read-only

Build 18334

For general Windows information on build 18334 visit the [Windows blog ↗](#).

WSL

- Redesign the way that Windows time zone is mapped to a Linux time zone [GH 3747]
- Fix memory leaks and add new string translation functions [GH 3746]
- SIGCONT on a threadgroup with no threads is a no-op [GH 3741]
- Correctly display socket and epoll file descriptors in /proc/self/fd

Build 18305

For general Windows information on build 18305 visit the [Windows blog](#).

WSL

- pthreads lose access to files when the primary thread exits [GH 3589]
- TIOCSCTTY should ignore the "force" parameter unless it is required [GH 3652]
- wsl.exe command line improvements and addition of import / export functionality.

```
Usage: wsl.exe [Argument] [Options...] [CommandLine]
```

Arguments to run Linux binaries:

If no command line is provided, wsl.exe launches the default shell.

```
--exec, -e <CommandLine>
    Execute the specified command without using the default Linux shell.
```

```
--
```

```
    Pass the remaining command line as is.
```

Options:

```
--distribution, -d <DistributionName>
    Run the specified distribution.
```

```
--user, -u <UserName>
    Run as the specified user.
```

Arguments to manage Windows Subsystem for Linux:

```
--export <DistributionName> <FileName>
    Exports the distribution to a tar file.
    The filename can be - for standard output.
```

```
--import <DistributionName> <InstallLocation> <FileName>
    Imports the specified tar file as a new distribution.
    The filename can be - for standard input.
```

```
--list, -l [Options]
    Lists distributions.
```

Options:

```
--all
    List all distributions, including distributions that are
currently
    being installed or uninstalled.
```

```
--running
    List only distributions that are currently running.
```

```
-setdefault, -s <DistributionName>
```

```
Sets the distribution as the default.  
  
--terminate, -t <DistributionName>  
    Terminates the distribution.  
  
--unregister <DistributionName>  
    Unregisters the distribution.  
  
--upgrade <DistributionName>  
    Upgrades the distribution to the WslFs file system format.  
  
--help  
    Display usage information.
```

Build 18277

For general Windows information on build 18277 visit the [Windows blog ↗](#).

WSL

- Fix "no such interface supported" error introduced in build 18272 [GH 3645]
- Ignore the MNT_FORCE flag for umount syscall [GH 3605]
- Switch WSL interop to use the official CreatePseudoConsole API
- Maintain no timeout value when FUTEX_WAIT restarts

Build 18272

For general Windows information on build 18272 visit the [Windows blog ↗](#).

WSL

- **WARNING:** There is an issue in this build that makes WSL inoperable. When trying to launch your distribution you will see a "No such interface supported" error. The issue has been fixed and will be in next week's Insider Fast build. If you've installed this build you can roll back to the previous Windows build using "Go back to the previous version of Windows 10" in Settings->Update & Security->Recovery.

Build 18267

For general Windows information on build 18267 visit the [Windows blog ↗](#).

WSL

- Fix issue where zombie process may not be reaped and remain indefinitely.
- WslRegisterDistribution hangs if error message exceeds max length [GH 3592]
- Allow fsync to succeed for read-only files on DrvFs [GH 3556]
- Ensure that /bin and /sbin directories exist before creating symlinks inside [GH 3584]
- Added an instance termination timeout mechanism for WSL instances. The timeout is currently set to 15 seconds, meaning the instance will terminate 15 seconds after the last WSL process exits. To terminate a distribution immediately, use:

```
wslconfig.exe /terminate <DistributionName>
```

Build 17763 (1809)

For general Windows information on build 17763 visit the [Windows blog](#).

WSL

- Setpriority syscall permission check too strict for changing same thread priority [GH 1838]
- Ensure that unbiased interrupt time is used for boot time to avoid returning negative values for clock_gettime(CLOCK_BOOTTIME) [GH 3434]
- Handle symlinks in the WSL binfmt interpreter [GH 3424]
- Better handling of threadgroup leader file descriptor cleanup.
- Switch WSL to use KeQueryInterruptTimePrecise instead of KeQueryPerformanceCounter to avoid overflow [GH 3252]
- Ptrace attach may cause bad return value from system calls [GH 1731]
- Fix several AF_UNIX related issues [GH 3371]
- Fix issue that could cause WSL interop to fail if the current working directory is less than 5 characters long [GH 3379]
- Avoid one second delay failing loopback connections to non-existent ports [GH 3286]
- Add /proc/sys/fs/file-max stub file [GH 2893]
- More accurate IPV6 scope information.
- PR_SET_PTRACER support [GH 3053]
- Pipe filesystem inadvertently clearing edge-triggered epoll event [GH 3276]
- Win32 executable launched via NTFS symlink doesn't respect symlink name [GH 2909]
- Improved zombie support [GH 1353]

- Add wsl.conf entries for controlling Windows interop behavior [GH 1493]

```
[interop]

enabled=false # enable launch of Windows binaries; default is true

appendWindowsPath=false # append Windows path to $PATH variable;
default is true
```

- Fix for getsockname not always returning UNIX socket family type [GH 1774]
- Add support for TIOCSTI [GH 1863]
- Non-blocking sockets in the process of connecting should return EAGAIN for write attempts [GH 2846]
- Support interop on mounted VHDs [GH 3246, 3291]
- Fix permission checking issue on root folder [GH 3304]
- Limited support for TTY keyboard ioctl KDGKBTYPE, KDGKBMODE and KDSKBMODE.
- Windows UI apps should execute even when launched in the background.
- Add wsl -u or --user option [GH 1203]
- Fix WSL launch issues when fast startup is enabled [GH 2576]
- Unix sockets need to retain disconnected peer credentials [GH 3183]
- Non-blocking Unix sockets failing indefinitely with EAGAIN [GH 3191]
- case=off is the new default drvfs mount type [GH 2937, 3212, 3328]
 - See [blog ↗](#) for more information.
- Add wslconfig /terminate to stop running distributions.
- Fix issue with the WSL shell context menu entries that do not correctly handle paths with spaces.
- Expose per-directory case sensitivity as an extended attribute
- ARM64: Emulate cache maintenance operations. Resolve [dotnet issue ↗](#).
- DrvFs: only unescape characters in the private range that correspond to an escaped character.
- Fix off-by-one error in ELF parser interpreter length validation [GH 3154]
- WSL absolute timers with a time in the past do not fire [GH 3091]
- Ensure newly created reparse points are listed as such in the parent directory.
- Atomically create case sensitive directories in DrvFs.
- Fixed an additional issue where multithreaded operations could return ENOENT even though the file exists. [GH 2712]
- Fixed WSL launch failure when UMCI is enabled. [GH 3020]
- Add explorer context menu to launch WSL [GH 437, 603, 1836]. To use, hold shift and right-click when in an explorer window.
- Fix Unix socket non-blocking behavior [GH 2822, 3100]

- Fix hanging NETLINK command as reported in GH 2026.
- Add support for mount propagation flags [GH 2911].
- Fix issue with truncate not causing inotify events [GH 2978].
- Add --exec option for wsl.exe to invoke a single binary without a shell.
- Add --distribution option for wsl.exe to select a specific distro.
- Limited support for dmesg. Applications can now log to dmesg. WSL driver logs limited information to dmesg. In future, this can be extended to carry other information/diagnostics from the driver.
 - Note: dmesg is currently supported through the `/dev/kmsg` device interface. `syslog` syscall interface is not yet supported. And, so, some of the `dmesg` command line options such as `-s`, `-c` don't work.
- Change default gid and mode of serial devices to match native [GH 3042]
- DrvFs now supports extended attributes.
 - Note: DrvFs has some limitations on the name of extended attributes. Some characters (like '/', '.' and '*') are not allowed, and extended attribute names are not case sensitive on DrvFs

Build 18252 (Skip Ahead)

For general Windows information on build 18252 visit the [Windows Blog ↗](#).

WSL

- Move init and bsdtar binaries out of lssmanager.dll and into a separate tools folder
- Fix race around closing file descriptor when using CLONE_FILES
- Handle optional fields in /proc/pid/mountinfo when translating DrvFs paths
- Allow DrvFs mknod to succeed without metadata support for S_IFREG
- Readonly files created on DrvFs should have the readonly attribute set [GH 3411]
- Add /sbin/mountdrv helper to handle DrvFs mounting
- Use POSIX rename in DrvFs.
- Allow path translation on volumes without a volume GUID.

Build 17738 (Fast)

For general Windows information on build 17738 visit the [Windows Blog ↗](#).

WSL

- Setpriority syscall permission check too strict for changing same thread priority [GH 1838]
- Ensure that unbiased interrupt time is used for boot time to avoid returning negative values for clock_gettime(CLOCK_BOOTTIME) [GH 3434]
- Handle symlinks in the WSL binfmt interpreter [GH 3424]
- Better handling of threadgroup leader file descriptor cleanup.

Build 17728 (Fast)

For general Windows information on build 17728 visit the [Windows Blog](#).

WSL

- Switch WSL to use KeQueryInterruptTimePrecise instead of KeQueryPerformanceCounter to avoid overflow [GH 3252]
- Ptrace attach may cause bad return value from system calls [GH 1731]
- Fix a number of AF_UNIX related issues [GH 3371]
- Fix issue that could cause WSL interop to fail if the current working directory is less than 5 characters long [GH 3379]

Build 18204 (Skip Ahead)

For general Windows information on build 18204 visit the [Windows Blog](#).

WSL

- Pipe filesystem inadvertently clearing edge-triggered epoll event [GH 3276]
- Win32 executable launched via NTFS symlink doesn't respect symlink name [GH 2909]

Build 17723 (Fast)

For general Windows information on build 17723 visit the [Windows Blog](#).

WSL

- Avoid one second delay failing loopback connections to non-existent ports [GH 3286]
- Add /proc/sys/fs/file-max stub file [GH 2893]

- More accurate IPV6 scope information.
- PR_SET_PTRACER support [GH 3053]
- Pipe filesystem inadvertently clearing edge-triggered epoll event [GH 3276]
- Win32 executable launched via NTFS symlink doesn't respect symlink name [GH 2909]

Build 17713

For general Windows information on build 17713 visit the [Windows Blog ↗](#).

WSL

- Improved zombie support [GH 1353]
- Add wsl.conf entries for controlling Windows interop behavior [GH 1493]

```
[interop]

enabled=false # enable launch of Windows binaries; default is true

appendWindowsPath=false # append Windows path to $PATH variable;
default is true
```

- Fix for getsockname not always returning UNIX socket family type [GH 1774]
- Add support for TIOCSTI [GH 1863]
- Non-blocking sockets in the process of connecting should return EAGAIN for write attempts [GH 2846]
- Support interop on mounted VHDs [GH 3246, 3291]
- Fix permission checking issue on root folder [GH 3304]
- Limited support for TTY keyboard ioctls KDGKBTYPE, KDGKBMODE and KDSKBMODE.
- Windows UI apps should execute even when launched in the background.

Build 17704

For general Windows information on build 17704 visit the [Windows Blog ↗](#).

WSL

- Add wsl -u or --user option [GH 1203]
- Fix WSL launch issues when fast startup is enabled [GH 2576]

- Unix sockets need to retain disconnected peer credentials [GH 3183]
- Non-blocking Unix sockets failing indefinitely with EAGAIN [GH 3191]
- case=off is the new default drvfs mount type [GH 2937, 3212, 3328]
 - See [blog ↗](#) for more information.
- Add wslconfig /terminate to stop running distributions.

Build 17692

For general Windows information on build 17692 visit the [Windows Blog ↗](#).

WSL

- Fix issue with the WSL shell context menu entries that do not correctly handle paths with spaces.
- Expose per-directory case sensitivity as an extended attribute
- ARM64: Emulate cache maintenance operations. Resolve [dotnet issue ↗](#).
- DrvFs: only unescape characters in the private range that correspond to an escaped character.

Build 17686

For general Windows information on build 17686 visit the [Windows Blog ↗](#).

WSL

- Fix off-by-one error in ELF parser interpreter length validation [GH 3154]
- WSL absolute timers with a time in the past do not fire [GH 3091]
- Ensure newly created reparse points are listed as such in the parent directory.
- Atomically create case sensitive directories in DrvFs.

Build 17677

For general Windows information on build 17677 visit the [Windows Blog ↗](#).

WSL

- Fixed an additional issue where multithreaded operations could return ENOENT even though the file exists. [GH 2712]
- Fixed WSL launch failure when UMCI is enabled. [GH 3020]

Build 17666

For general Windows information on build 17666 visit the [Windows Blog ↗](#).

WSL

WARNING: There is an issue preventing WSL from running on some AMD chipsets [GH 3134]. A fix is ready and making its way to the Insider Build branch.

- Add explorer context menu to launch WSL [GH 437, 603, 1836]. To use hold shift and right-click when in an explorer window.
- Fix unix socket non-blocking behavior [GH 2822, 3100]
- Fix hanging NETLINK command as reported in GH 2026.
- Add support for mount propagation flags [GH 2911].
- Fix issue with truncate not causing inotify events [GH 2978].
- Add --exec option for wsl.exe to invoke a single binary without a shell.
- Add --distribution option for wsl.exe to select a specific distro.

Build 17655 (Skip Ahead)

For general Windows information on build 17655 visit the [Windows Blog ↗](#).

WSL

- Limited support for dmesg. Applications can now log to dmesg. WSL driver logs limited information to dmesg. In future, this can be extended to carry other information/diagnostics from the driver.
 - Note: dmesg is currently supported through the `/dev/kmsg` device interface. `syslog` syscall interface is not yet supported. And, so, some of the `dmesg` command line options such as `-s`, `-c` don't work.
- Fixed an issue where multithreaded operations could return ENOENT even though the file exists. [GH 2712]

Build 17639 (Skip Ahead)

For general Windows information on build 17639 visit the [Windows Blog ↗](#).

WSL

- Change default gid and mode of serial devices to match native [GH 3042]
- DrvFs now supports extended attributes.
 - Note: DrvFs has some limitations on the name of extended attributes. In particular, some characters (like '/', ':' and '*') are not allowed, and extended attribute names are not case sensitive on DrvFs

Build 17133 (Fast)

For general Windows information on build 17133 visit the [Windows Blog ↗](#).

WSL

- Fix for hang in WSL. [GH 3039, 3034]

Build 17128 (Fast)

For general Windows information on build 17128 visit the [Windows Blog ↗](#).

WSL

- None

Build 17627 (Skip Ahead)

For general Windows information on build 17627 visit the [Windows Blog ↗](#).

WSL

- Add support for the futex pi-aware operations. [GH 1006]
 - Note that priorities are not currently a supported WSL feature so there are limitations, but standard usage should be unblocked.
- Windows firewall support for WSL processes. [GH 1852]
 - For example, to allow the WSL python process to listen on any port, use the elevated Windows cmd:
`netsh.exe advfirewall firewall add rule
name=wsl_python dir=in action=allow program="C:\users\
<username>\appdata\local\packages\canonicalgroup\limited.ubuntuonwindows_79r
hkp1fndgsc\localstate\rootfs\usr\bin\python2.7" enable=yes`
 - For additional details on how to add firewall rules, see [link ↗](#)
- Respect user's default shell when using wsl.exe. [GH 2372]

- Report all network interfaces as ethernet. [GH 2996]
- Better handling of corrupt /etc/passwd file. [GH 3001]

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17618 (Skip Ahead)

For general Windows information on build 17618 visit the [Windows Blog ↗](#).

WSL

- Introduce pseudoconsole functionality for NT interop [GH 988, 1366, 1433, 1542, 2370, 2406].
- The legacy install mechanism (lxrun.exe) has been deprecated. The supported mechanism for installing distributions is through the Microsoft Store.

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17110

For general Windows information on build 17110 visit the [Windows Blog ↗](#).

WSL

- Allow /init to be terminated from Windows [GH 2928].
- DrvFs now uses per-directory case sensitivity by default (equivalent to the "case=dir" mount option).

- Using "case=force" (the old behavior) requires setting a registry key. Run the following command to enable "case=force" if you need to use it: reg add HKLM\SYSTEM\CurrentControlSet\Services\lxss /v DrvFsAllowForceCaseSensitivity /t REG_DWORD /d 1
- If you have existing directories created with WSL in older version of Windows which need to be case sensitive, use fsutil.exe to mark them as case sensitive: fsutil.exe file setcasesensitiveinfo <path> enable
- NULL terminate strings returned from the uname syscall.

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17107

For general Windows information on build 17107 visit the [Windows Blog](#).

WSL

- Support TCSETSF and TCSETSW on master pty endpoints [GH 2552].
- Starting simultaneous interop processes can result in EINVAL [GH 2813].
- Fix PTRACE_ATTACH to show proper tracing status in /proc/pid/status.
- Fix race where short-lived processes cloned with both the CLEARTID and SETTID flags could exit without clearing the TID address.
- Display a message when upgrading the Linux file system directories when moving from a pre-17093 build. For more details on the 17093 file system changes, see the release notes for [17093](#).

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17101

For general Windows information on build 17101 visit the [Windows Blog](#).

WSL

- Support for signalfd. [GH 129]
- Support file-names containing illegal NTFS characters by encoding them as private Unicode characters. [GH 1514]
- Auto mount will fallback to read-only when write is not supported. [GH 2603]
- Allow pasting of Unicode surrogate pairs (like emoji characters). [GH 2765]
- Pseudo-files in /proc and /sys should return read and write ready from select, poll, epoll, et al. [GH 2838]
- Fix issue that could cause service to go into infinite loop when the registry has been tampered with or is corrupt.
- Fix netlink messages to work with newer (upstream 4.14) version of iproute2.

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17093

For general Windows information on build 17093 visit the [Windows Blog](#).

Important:

When starting WSL for the first time after upgrading to this build, it needs to perform some work upgrading the Linux file system directories. This may take up to several minutes, so WSL may appear to start slowly. This should only happen once for each distribution you have installed from the store.

- Improved case sensitivity support in DrvFs.
 - DrvFs now supports per-directory case sensitivity. This is a new flag that can be set on directories to indicate all operations in those directories should be

treated as case sensitive, which allows even Windows applications to correctly open files that differ only by case.

- DrvFs has new mount options controlling case sensitivity on a per-directory basis
 - case=force: all directories are treated as case sensitive (except for the drive root). New directories created with WSL are marked as case sensitive. This is the legacy behavior except for marking new directories case sensitive.
 - case=dir: only directories with the per-directory case sensitivity flag are treated as case sensitive; other directories are case insensitive. New directories created with WSL are marked as case sensitive.
 - case=off: only directories with the per-directory case sensitivity flag are treated as case sensitive; other directories are case insensitive. New directories created with WSL are marked as case insensitive.
- Note: directories created by WSL in previous releases do not have this flag set, so will not be treated as case sensitive if you use the "case=dir" option. A way to set this flag on existing directories is coming soon.
- Example of mounting with these options (for existing drives, you must first unmount before you can mount with different options): sudo mount -t drvfs C:/mnt/c -o case=dir
- For now, case=force is still the default option. This will be changed to case=dir in the future.
- You can now use forward slashes in Windows paths when mounting DrvFs, e.g.:
sudo mount -t drvfs //server/share /mnt/share
- WSL now processes the /etc/fstab file during instance start [GH 2636].
 - This is done prior to automatically mounting DrvFs drives; any drives that were already mounted by fstab will not be remounted automatically, allowing you to change the mount point for specific drives.
 - This behavior can be turned off using wsl.conf.
- The mount, mountinfo and mountstats files in /proc properly escape special characters like backslashes and spaces [GH 2799]
- Special files created with DrvFs such as WSL symbolic links, or fifos and sockets when metadata is enabled, can now be copied and moved from Windows.

WSL is more configurable with wsl.conf

We added a method for you to automatically configure certain functionality in WSL that will be applied every time you launch the subsystem. This includes automount options and network configuration. Learn more about it in our blog post at:

<https://aka.ms/wslconf>

AF_UNIX allows socket connections between Linux processes on WSL and Windows native processes

WSL and Windows applications can now communicate with each other over Unix sockets. Imagine you want to run a service in Windows and make it available to both Windows and WSL apps. Now, that's possible with Unix sockets. Read more in our blog post at <https://aka.ms/afunixinterop>

WSL

- Support mmap() with MAP_NORESERVE [GH 121, 2784]
- Support CLONE_PTRACE and CLONE_UNTRACED [GH 121, 2781]
- Handle non-SIGCHLD termination signal in clone [GH 121, 2781]
- Stub /proc/sys/fs/inotify/max_user_instances and /proc/sys/fs/inotify/max_user_watches [GH 1705]
- Error loading ELF binaries that contain load headers with non-zero offsets [GH 1884]
- Zero out trailing page bytes when loading images.
- Reduce cases where execve silently terminates process

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17083

For general Windows information on build 17083 visit the [Windows Blog](#).

WSL

- Fixed bugcheck related to epoll [GH 2798, 2801, 2857]
- Fixed hangs when turning off ASLR [GH 1185, 2870]
- Ensure mmap operations appear atomic [GH 2732]

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17074

For general Windows information on build 17074 visit the [Windows Blog ↗](#).

WSL

- Fixed storage format of DrvFs metadata [GH 2777]
Important: DrvFs metadata created before this build will show up incorrectly or not at all. To fix affected files, use chmod and chown to re-apply the metadata.
- Fixed issue with multiple signals and restartable syscalls.

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17063

For general Windows information on build 17063 visit the [Windows Blog ↗](#).

WSL

- DrvFs supports additional Linux metadata. This allows setting the owner and mode of files using chmod/chown, and also the creation of special files such as fifos, unix sockets and device files. This is disabled by default for now since it's still experimental. **Note:** We fixed a bug in the metadata format used by DrvFs. While metadata works on this build for experimentation, future builds will not correctly read metadata created by this build. You might need to manually update owner for modified files and devices with a custom device ID will have to be recreated.

To enable, mount DrvFs with the metadata option (to enable it on an existing mount, you must first unmount it):

Bash

```
mount -t drvfs C: /mnt/c -o metadata
```

Linux permissions are added as additional metadata to the file; they do not affect the Windows permissions. Remember, editing a file using a Windows editor may remove the metadata. In this case, the file will revert to its default permissions.

- Added mount options to DrvFs to control files without metadata.
 - uid: the user ID used for the owner of all files.
 - gid: the group ID used for the owner of all files.
 - umask: an octal mask of permissions to exclude for all files and directories.
 - fmask: an octal mask of permissions to exclude for all regular files.
 - dmask: an octal mask of permissions to exclude for all directories.

For example:

```
mount -t drvfs C: /mnt/c -o uid=1000,gid=1000,umask=22,fmask=111
```

Combine with the metadata option to specify default permissions for files without metadata.

- Introduced a new environment variable, `WSLENV`, to configure how environment variables flow between WSL and Win32.

For example:

Bash

```
WSLENV=GOPATH/1:USERPROFILE/pu:DISPLAY
```

`WSLENV` is a colon-delimited list of environment variables that can be included when launching WSL processes from Win32 or Win32 processes from WSL. Each variable can be suffixed with a slash followed by flags to specify how it is translated.

- p: The value is a path that should be translated between WSL paths and Win32 paths.

- l: The value is a list of paths. In WSL, it is a colon-delimited list. In Win32, it is a semicolon-delimited list.
- u: The value should only be included when invoking WSL from Win32
- w: The value should only be included when invoking Win32 from WSL

You can set `WSLENV` in `.bashrc` or in the custom Windows environment for your user.

- drvfs mounts correctly preserves timestamps from tar, cp -p (GH 1939)
- drvfs symlinks report the correct size (GH 2641)
- read/write works for very large IO sizes (GH 2653)
- waitpid works with process group IDs (GH 2534)
- significantly improved mmap performance for large reserve regions; improves ghc performance (GH 1671)
- personality supports for READ_IMPLIES_EXEC; fixes maxima and clisp (GH 1185)
- mprotect supports PROT_GROWSDOWN; fixes clisp (GH 1128)
- page fault fixes in overcommit mode; fixes sbcl (GH 1128)
- clone supports more flags combinations
- Support select/epoll of epoll files (previously a no-op).
- Notify ptrace of unimplemented syscalls.
- Ignore interfaces that are not up when generating resolv.conf nameservers [GH 2694]
- Enumerate network interfaces with no physical address. [GH 2685]
- Additional bug fixes and improvements.

Linux tools available to developers on Windows

- Windows Command line Toolchain includes bsdtar (tar) and curl. Read [this blog ↗](#) to learn more about the addition of these two new tools and see how they're shaping the developer experience on Windows.
- `AF_UNIX` is available in the Windows Insider SDK (17061+). Read [this blog ↗](#) to learn more about `AF_UNIX` and how developers on Windows can use it.

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17046

For general Windows information on build 17046 visit the [Windows Blog ↗](#).

Fixed

WSL

- Allow processes to run without an active terminal. [GH 709, 1007, 1511, 2252, 2391, et al.]
- Better support of CLONE_VFORK and CLONE_VM. [GH 1878, 2615]
- Skip TDI filter drivers for WSL networking operations. [GH 1554]
- DrvFs creates NT symlinks when certain conditions are met. [GH 353, 1475, 2602]
 - The link target must be relative, must not cross any mount points or symlinks, and must exist.
 - The user must have SE_CREATE_SYMBOLIC_LINK_PRIVILEGE (this normally requires you to launch wsl.exe elevated), unless Developer Mode is turned on.
 - In all other situations, DrvFs still creates WSL symlinks.
- Allow running elevated and non-elevated WSL instances simultaneously.
- Support /proc/sys/kernel/yama/ptrace_scope
- Add wslpath to do WSL<->Windows path conversions. [GH 522, 1243, 1834, 2327, et al.]

```
wslpath usage:  
-a      force result to absolute path format  
-u      translate from a Windows path to a WSL path (default)  
-w      translate from a WSL path to a Windows path  
-m      translate from a WSL path to a Windows path, with '/' instead  
       of '\\'  
  
EX: wslpath 'c:\users'
```

Console

- No fixes.

LTP Results:

Testing in progress.

Build 17040

For general Windows information on build 17040 visit the [Windows Blog ↗](#).

Fixed

WSL

- No fixes since 17035.

Console

- No fixes since 17035.

LTP Results:

Testing in progress.

Build 17035

For general Windows information on build 17035 visit the [Windows Blog ↗](#).

Fixed

WSL

- Accessing files on DrvFs could occasionally fail with EINVAL. [GH 2448]

Console

- Some color loss when inserting/deleting lines in VT mode.

LTP Results:

Testing in progress.

Build 17025

For general Windows information on build 17025 visit the [Windows Blog ↗](#).

Fixed

WSL

- Start initial processes in a new foreground process group [GH 1653, 2510].
- SIGHUP delivery fixes [GH 2496].
- Generate default virtual bridge name if none provided [GH 2497].
- Implement /proc/sys/kernel/random/boot_id [GH 2518].
- More interop stdout/stderr pipe fixes.
- Stub syncfs system call.

Console

- Fix input VT translation for third party consoles [GH 111]

LTP Results:

Testing in progress.

Build 17017

For general Windows information on build 17017 visit the [Windows Blog ↗](#).

Fixed

WSL

- Ignore empty ELF program headers [GH 330].
- Allow LxssManager to create WSL instances for non-interactive users (ssh and scheduled task support) [GH 777, 1602].
- Support WSL->Win32->WSL ("inception") scenarios [GH 1228].

- Limited support for termination of console apps invoked via interop [GH 1614].
- Support mount options for devpts [GH 1948].
- Ptrace blocking child startup [GH 2333].
- EPOLLET missing some events [GH 2462].
- Return more data for PTRACE_GETSIGINFO.
- Getdents with lseek gives incorrect results.
- Fix some Win32 interop app hangs, waiting for input on a pipe that has no more data.
- O_ASYNC support for tty/pty files.
- Additional improvements and bug fixes

Console

- No Console related changes in this release.

LTP Results:

Testing in progress.

Fall Creators Update

Build 16288

For general Windows information on build 16288 visit the [Windows Blog ↗](#).

Fixed

WSL

- Correctly initialize and report uid, gid, and mode for socket file descriptors [GH 2490]
- Additional improvements and bug fixes

Console

- No Console related changes in this release.

LTP Results:

No change since 16273

Build 16278

For general Windows information on build 162738 visit the [Windows Blog](#).

Fixed

WSL

- Explicitly unmap mapped views of file backed sections when tearing down LX MM state [GH 2415]
- Additional improvements and bug fixes

Console

- No Console related changes in this release.

LTP Results:

No change since 16273

Build 16275

For general Windows information on build 162735 visit the [Windows Blog](#).

Fixed

WSL

- No WSL related changes in this release.

Console

- No Console related changes in this release.

LTP Results:

No change since 16273

Build 16273

For general Windows information on build 16273 visit the [Windows Blog](#).

Fixed

WSL

- Fixed an issue where DrvFs sometimes reported the wrong file type for directories [GH 2392]
- Allow creation of NETLINK_KOBJECT_UEVENT sockets to unblock programs that use uevent [GH 1121, 2293, 2242, 2295, 2235, 648, 637]
- Add support for non-blocking connect [GH 903, 1391, 1584, 1585, 1829, 2290, 2314]
- Implement CLONE_FS clone system call flag [GH 2242]
- Fix issues around not handling tabs or quotes correctly in NT interop [GH 1625, 2164]
- Resolve access denied error when trying to re-launch WSL instances [GH 651, 2095]
- Implement futex FUTEX_REQUEUE and FUTEX_CMP_REQUEUE operations [GH 2242]
- Fix permissions for various SysFs files [GH 2214]
- Fix Haskell stack hang during setup [GH 2290]
- Implement binfmt_misc 'C' 'O' and 'P' flags [GH 2103]
- Add /proc/sys/kernel /shmmax /shmmni & /threads-max [GH 1753]
- Add partial support for ioprio_set system call [GH 498]
- Stub SO_REUSEPORT & adding support for SO_PASSCRED for netlink sockets [GH 69]
- Return different error codes from RegisterDistribution if a distribution is currently being installed or uninstalled.
- Allow unregistration of partially installed WSL distributions via wslconfig.exe
- Fix python socket test hang from udp::msg_peek
- Additional improvements and bug fixes

Console

- No Console related changes in this release.

LTP Results:

Total Tests: 1904

Total Skipped Tests: 209

Total Failures: 229

Build 16257

For general Windows information on build 16257 visit the [Windows Blog ↗](#).

Fixed

WSL

- Implement prlimit64 system call
- Add support for ulimit -n (setrlimit RLIMIT_NOFILE) [GH 1688]
- Stub MSG_MORE for TCP sockets [GH 2351]
- Fix invalid AT_EXECFN auxiliary vector behavior [GH 2133]
- Fix copy/paste behavior for console/tty, and add better full buffer handling [GH 2204, 2131]
- Set AT_SECURE in auxiliary vector for set-user-ID and set-group-ID programs [GH 2031]
- Pseudo-terminal master endpoint not handling TIOCPGRP [GH 1063]
- Fix lseek does to rewind directories in LxFs [GH 2310]
- /dev/ptmx locks up after heavy usage [GH 1882]
- Additional improvements and bug fixes

Console

- Fix for horizontal Lines/Underscores Everywhere [GH 2168]
- Fix for process order changed making NPM harder to close [GH 2170]
- Added our new color scheme:
<https://blogs.msdn.microsoft.com/commandline/2017/08/02/updating-the-windows-console-colors/> ↗

LTP Results:

No change since 16251

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

`prlimit64`

Known Issues

[GitHub Issue 2392: Windows Folders not recognized by WSL ... ↗](#)

In build 16257, WSL has issues when enumerating Windows files/folders via `/mnt/c/...`. This issue has been fixed and should be released in Insiders build during week commencing 8/14/2017.

Build 16251

For general Windows information on build 16251 visit the [Windows Blog ↗](#).

Fixed

WSL

- Remove beta tag from WSL optional component, see [blog post ↗](#) for details.
- Correctly initialize saved-set uid and gid for set-user-ID and set-group-ID binaries on exec [GH 962, 1415, 2072]
- Added support for ptrace PTRACE_O_TRACEEXIT [GH 555]
- Added support for ptrace PTRACE_GETFPREGS and PTRACE_GETREGSET with NT_FPREGSET [GH 555]
- Fixed ptrace to stop on ignored signals
- Additional improvements and bug fixes

Console

- No Console related changes in this release.

LTP Results:

Number of Passing Tests: 768

Number of Failing Tests: 244

Number of Skipped Tests: 96

Build 16241

For general Windows information on build 16241 visit the [Windows Blog ↗](#).

Fixed

WSL

- No WSL related changes in this release.

Console

- Fix for outputting the wrong character for the crossing-lines DEC, originally reported [here ↗](#)
- Fix for no output text being displayed in codepage 65001 (utf8)
- Do not transfer changes made to one color's RGB values to other parts of the palette on selection change. This will make the console properties sheet a lot easier to use.
- Ctrl+S doesn't appear to work correctly
- Un-Bold/-Dim completely absent from ANSI escape codes [GH 2174]
- Console doesn't correctly support Vim color themes [GH 1706]
- Cannot paste particular characters [GH 2149]
- Reflow resize interacts strangely with resizing a bash window when stuff is on the edit/command line [GH ConEmu 1123]
- Ctrl-L leaves the screen dirty [GH 1978]
- Console rendering bug when displaying VT on HDPI [GH 1907]
- Japanese characters look strange with Unicode Character U+30FB [GH 2146]
- Additional improvements and bug fixes

Build 16237

For general Windows information on build 16237 visit the [Windows Blog ↗](#).

Fixed

- Use default attributes for files without EAs in lxfs (root, root, 0000)
- Added support for distributions that use extended attributes
- Fix padding for entries returned by getdents and getdents64
- Fix permissions check for the shmctl SHM_STAT system call [GH 2068]
- Fixed incorrect initial epoll state for ttys [GH 2231]
- Fix DrvFs readdir not returning all entries [GH 2077]
- Fix LxFs readdir when files are unlinked [GH 2077]
- Allow unlinked drvfs files to be reopened through procfs
- Added global registry key override for disabling WSL features (interop / drive mounting)
- Fix incorrect block count in "stat" for DrvFs (and LxFs) [GH 1894]
- Additional improvements and bug fixes

Build 16232

For general Windows information on build 16232 visit the [Windows Blog ↗](#).

Fixed

- No WSL related changes in this release.

Build 16226

For general Windows information on build 16226 visit the [Windows Blog ↗](#).

Fixed

- xattr related syscalls support (getxattr, setxattr, listxattr, removexattr).
- security.capability xattr support.
- Improved compatibility with certain file systems and filters, including non-MS SMB servers. [GH #1952]
- Improved support for OneDrive placeholders, GVFS placeholders, and Compact OS compressed files.
- Additional improvements and bug fixes

Build 16215

For general Windows information on build 16215 visit the [Windows Blog](#).

Fixed

- WSL no longer requires developer mode.
- Support directory junctions in drvfs.
- Handle uninstalling of WSL distribution appx packages.
- Update procfs to show private and shared mappings.
- Add ability for wslconfig.exe to clean up distributions that are partially installed or uninstalled.
- Added support for IP_MTU_DISCOVER for TCP sockets. [GH 1639, 2115, 2205]
- Infer protocol family for routes to AF_INADDR.
- Serial device improvements [GH 1929].

Build 16199

For general Windows information on build 16199 visit the [Windows Blog](#).

Fixed

- No WSL related changes in these releases.

Build 16193

For general Windows information on build 16193 visit the [Windows Blog](#).

Fixed

- Race condition between sending SIGCONT and a threadgroup terminating [GH 1973]
- change tty and pty devices to report FILE_DEVICE_NAMED_PIPE instead of FILE_DEVICE_CONSOLE [GH 1840]

- SSH fix for IP_OPTIONS
- Moved DrvFs mounting to init daemon [GH 1862, 1968, 1767, 1933]
- Added support in DrvFs for following NT symlinks.

Build 16184

For general Windows information on build 16184 visit the [Windows Blog ↗](#).

Fixed

- Removed apt package maintenance task (lxrun.exe /update)
- Fixed output not showing up in from Windows processes in node.js [GH 1840]
- Relax alignment requirements in lxcore [GH 1794]
- Fixed handling of the AT_EMPTY_PATH flag in a numer of system calls.
- Fixed issue where deleting DrvFs files with open handles will cause the file to exhibit undefined behavior [GH 544,966,1357,1535,1615]
- /etc/hosts will now inherit entries from the Windows hosts file (%windir%\system32\drivers\etc\hosts) [GH 1495]

Build 16179

For general Windows information on build 16179 visit the [Windows Blog ↗](#).

Fixed

- No WSL changes this week.

Build 16176

For general Windows information on build 16176 visit the [Windows Blog ↗](#).

Fixed

- Enabled serial support

- Added IP socket option IP_OPTIONS [GH 1116]
- Implemented pwritev function (while uploading file to nginx/PHP-FPM) [GH 1506]
- Added IP socket options IP_MULTICAST_IF & IPV6_MULTICAST_IF [GH 990]
- Support for socket option IP_MULTICAST_LOOP & IPV6_MULTICAST_LOOP [GH 1678]
- Added IP(V6)_MTU socket option for apps node, traceroute, dig, nslookup, host
- Added IP socket option IPV6_UNICAST_HOPS
- [Filesystem Improvements](#)
 - Allow mounting of UNC paths
 - Enable CDFS support in drvfs
 - Correctly handle permissions for network file systems in drvfs
 - Add support for remote drives to drvfs
 - Enable FAT support in drvfs
- Additional fixes and improvements

LTP Results

No changes since 15042

Build 16170

For general Windows information on build 16170 visit the [Windows Blog ↗](#).

We released a new [blog post](#) discussing our efforts to test WSL.

Fixed

- Support socket option IP_ADD_MEMBERSHIP & IPV6_ADD_MEMBERSHIP [GH 1678]
- Add support for PTRACE_OLDSETOPTIONS. [GH 1692]
- Additional fixes and improvements

LTP Results

No changes since 15042

Build 15046 to Windows 10 Creators Update

There are no more WSL fixes or features planned for inclusion in the Creators Update to Windows 10. Release notes for WSL will resume in the coming weeks for additions targeting the next major Windows Update. For general Windows information on build 15046 and future Insider releases visit the [Windows Blog](#).

Build 15042

For general Windows information on build 15042 visit the [Windows Blog](#).

Fixed

- Fix for a deadlock when removing a path ending in ".."
- Fixed an issue where FIONBIO not returning 0 on success [GH 1683]
- Fixed issue with zero-length reads of inet datagram sockets
- Fix possible deadlock due to race condition in drvfs inode lookup [GH 1675]
- Extended support for unix socket ancillary data; SCM_CREDENTIALS and SCM_RIGHTS [GH 514, 613, 1326]
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 737

Number of non-Passing (failing, skipped, etc...): 255

Build 15031

For general Windows information on build 15031 visit the [Windows Blog](#).

Fixed

- Fixed a bug where time(2) would sporadically misbehave.
- Fixed an issue where *SIGPROCMASK syscalls could corrupt signal mask.
- Now return full command line length in WSL process creation notification. [GH 1632]
- WSL now reports thread exit through ptrace for GDB hangs. [GH 1196]
- Fixed bug where ptys would hang after heavy tmux IO. [GH 1358]

- Fixed timeout validation in many system calls (futex, semtimedop, ppoll, sigtimedwait, itimer, timer_create)
- Added eventfd EFD_SEMAPHORE support [GH 452]
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 737

Number of non-Passing (failing, skipped, etc...): 255

Build 15025

For general Windows information on build 15025 visit the [Windows Blog](#).

Fixed

- Fix for bug that broke grep 2.27 [GH 1578]
- Implemented EFD_SEMAPHORE flag for eventfd2 syscall [GH 452]
- Implemented /proc/[pid]/net/ipv6_route [GH 1608]
- Signal driven IO support for unix stream sockets [GH 393, 68]
- Support F_GETPIPE_SZ and F_SETPIPE_SZ [GH 1012]
- Implement recvmmmsg() syscall [GH 1531]
- Fixed bug where epoll_wait() wasn't waiting [GH 1609]
- Implement /proc/version_signature
- Tee syscall now returns failure if both file descriptors refer to the same pipe
- Implemented correct behavior for SO_PEERCRED for Unix sockets
- Fixed tkill syscall invalid parameter handling
- Changes to increase the performance of drvfs
- Minor fix for Ruby IO blocking
- Fixed recvmsg() returning EINVAL for the MSG_DONTWAIT flag for inet sockets [GH 1296]
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 732

Number of non-Passing (failing, skipped, etc...): 255

Build 15019

For general Windows information on build 15019 visit the [Windows Blog](#).

Fixed

- Fixed bug that incorrectly reported CPU usage in procfs for tools like htop (GH 823, 945, 971)
- When calling open() with O_TRUNC on an existing file inotify now generates IN MODIFY before IN_OPEN
- Fixes to unix socket getsockopt SO_ERROR to enable postgres [GH 61, 1354]
- Implement /proc/sys/net/core/somaxconn for the GO language
- Apt-get package update background task now runs hidden from view
- Clear scope for ipv6 localhost (Spring-Framework(Java) failure).
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 714

Number of non-Passing (failing, skipped, etc...): 249

Build 15014

For general Windows information on build 15014 visit the [Windows Blog](#).

Fixed

- Ctrl+C now works as intended
- htop and ps auxw now show correct resource utilization (GH #516)
- Basic translation of NT exceptions to signals. (GH #513)
- fallocate now fails with ENOSPC when running out of space instead of EINVAL (GH #1571)
- Added /proc/sys/kernel/sem.
- Implemented semop and semtimedop system calls
- Fixed nslookup errors with IP_RECVTOS & IPV6_RECVTCLASS socket option (GH 69)
- Support for socket options IP_RECVTTL and IPV6_RECVHOPLIMIT
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 709

Number of non-Passing (failing, skipped, etc...): 255

Syscall Summary

Total Syscalls: 384

Total Implemented: 235

Total Stubbed: 22

Total Unimplemented: 127

Build 15007

For general Windows information on build 15007 visit the [Windows Blog ↗](#).

Known Issue

- There is a known bug where the console does not recognize some Ctrl + <key> input. This includes the ctrl-c command which will act as a normal 'c' keypress.
 - Workaround: Map an alternate key to Ctrl+C. For example, to map Ctrl+K to Ctrl+C do: `stty intr ^k`. This mapping is per terminal and will have to be done *every* time bash is launched. Users can explore the option to include this in their `.bashrc`

Fixed

- Corrected the issue where running WSL would consume 100% of a CPU core
- Socket option IP_PKTINFO, IPV6_RECV_PKTINFO now supported. (GH #851, 987)
- Truncate network interface physical address to 16 bytes in lcore (GH #1452, 1414, 1343, 468, 308)
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 709

Number of non-Passing (failing, skipped, etc...): 255

Build 15002

For general Windows information on build 15002 visit the [Windows Blog](#).

Known Issue

Two known issues:

- There is a known bug where the console does not recognize some Ctrl + <key> input. This includes the ctrl-c command which will act as a normal 'c' keypress.
 - Workaround: Map an alternate key to Ctrl+C. For example, to map Ctrl+K to Ctrl+C do: `stty intr ^k`. This mapping is per terminal and will have to be done every time bash is launched. Users can explore the option to include this in their `.bashrc`
- While WSL is running a system thread will consume 100% of a CPU core. The root cause has been addressed and fixed internally.

Fixed

- All bash sessions must now be created at the same permission level. Attempting to start a session at a different level will be blocked. This means admin and non-admin consoles cannot run at the same time. (GH #626)
- Implemented the following NETLINK_ROUTE messages (requires Windows admin)
 - RTM_NEWSADDR (supports `ip addr add`)
 - RTM_NEWRROUTE (supports `ip route add`)
 - RTM_DELADDR (supports `ip addr del`)
 - RTM_DELROUTE (supports `ip route del`)
- Scheduled task checking for packages to update will no longer run on a metered connection (GH #1371)
- Fixed error where piping gets stuck i.e. `bash -c "ls -alR /" | bash -c "cat"` (GH #1214)
- Implemented TCP_KEEP_CNT socket option (GH #843)
- Implemented IP_MTU_DISCOVER_INET socket option (GH #720, 717, 170, 69)
- Removed legacy functionality to run NT binaries from init with NT path lookup. (GH #1325)
- Fix mode of `/dev/kmsg` to allow group / other read access (0644) (GH #1321)
- Implemented `/proc/sys/kernel/random/uuid` (GH #1092)
- Corrected error where process start time was showing as year 2432 (GH #974)
- Switched default TERM environment variable to xterm-256color (GH #1446)

- Modified the way that process commit is calculated during process fork. (GH #1286)
- Implemented /proc/sys/vm/overcommit_memory. (GH #1286)
- Implemented /proc/net/route file (GH #69)
- Fixed error where shortcut name was incorrectly localized (GH #696)
- Fixed elf parsing logic that is incorrectly validating the program headers must be less than (or equal to) PATH_MAX. (GH #1048)
- Implemented statfs callback for procfs, sysfs, cgroupfs, and binfmtfs (GH #1378)
- Fixed AptPackageIndexUpdate windows that won't close (GH #1184, also discussed in GH #1193)
- Added ASLR personality ADDR_NO_RANDOMIZE support. (GH #1148, 1128)
- Improved PTRACE_GETSIGINFO, SIGSEGV, for proper gdb stack traces during AV (GH #875)
- Elf parsing no longer fails for patchelf binaries. (GH #471)
- VPN DNS propagated to /etc/resolv.conf (GH #416, 1350)
- Improvements to TCP close for more reliable data transfer. (GH #610, 616, 1025, 1335)
- Now return correct error code when too many files are opened (EMFILE). (GH #1126, 2090)
- Windows Audit log now reports the image name in process create audit.
- Now gracefully fail when launching bash.exe from within a bash window
- Added error message when interop is unable to access a working directory under LxFs (i.e. notepad.exe .bashrc)
- Fixed issue where Windows path was truncated in WSL
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 690

Number of non-Passing (failing, skipped, etc...): 274

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

`shmctl`

`shmget`

shmdt

shmat

Build 14986

For general Windows information on build 14986 visit the [Windows Blog](#).

Fixed

- Fixed bugchecks with Netlink and Pty IOCTLS
- Kernel version now reports 4.4.0-43 for consistency with Xenial
- Bash.exe now launches when input directed to 'nul:' (GH #1259)
- Thread IDs now reported correctly in procfs (GH #967)
- IN_UNMOUNT | IN_Q_OVERFLOW | IN_IGNORED | IN_ISDIR flags now supported in inotify_add_watch() (GH #1280)
- Implement timer_create and related system calls. This enables GHC support (GH #307)
- Fixed issue where ping returned a time of 0.000ms (GH #1296)
- Return correct error code when too many files are opened.
- Fixed issue in WSL where Netlink request for network interface data would fail with EINVAL if the interface's hardware address is 32-bytes (such as the Teredo interface)
 - Note that the Linux "ip" utility contains a bug where it will crash if WSL reports a 32-byte hardware address. This is a bug in "ip", not WSL. The "ip" utility hard-codes the length of the string buffer used to print the hardware address, and that buffer is too small to print a 32-byte hardware address.
- Additional fixes and improvements

LTP Results:

Number of Passing Test: 669

Number of non-Passing (failing, skipped, etc...): 258

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

```
timer_create  
timer_delete  
timer_gettime  
timer_settime
```

Build 14971

For general Windows information on build 14971 visit the [Windows Blog ↗](#).

Fixed

- Due to circumstances beyond our control there are no updates in this build for the Windows Subsystem for Linux. Regularly scheduled updates will resume on the next release.

LTP Results:

Unchanged from 14965

Number of Passing Test: 664

Number of non-Passing (failing, skipped, etc...): 263

Build 14965

For general Windows information on build 14965 visit the [Windows Blog ↗](#).

Fixed

- Support for Netlink sockets NETLINK_ROUTE protocol's RTM_GETLINK and RTM_GETADDR (GH #468)
 - Enables ifconfig and ip commands for network enumeration
- /sbin is now in the user's path by default
- NT user path now appended to the WSL path by default (i.e. you can now type notepad.exe without adding System32 to the Linux path)
- Added support for /proc/sys/kernel/cap_last_cap

- NT Binaries can now be launched from WSL when the current working directory contains non-ansi characters (GH #1254)
- Allow shutdown on disconnected unix stream socket.
- Added support for PR_GET_PDEATHSIG.
- Added support for CLONE_PARENT
- Fixed error where piping gets stuck i.e. bash -c "ls -alR /" | bash -c "cat" (GH #1214)
- Handle requests to connect to the current terminal.
- Mark `/proc/<pid>/oom_score_adj` as writable.
- Add `/sys/fs/cgroup` folder.
- `sched_setaffinity` should return number of affinity bits mask
- Fix ELF validation logic which incorrectly assumes interpreter paths must be less than 64 characters long. (GH #743)
- Open file descriptors can keep console window open (GH #1187)
- Fixed error where `rename()` failed with trailing slash on target name (GH #1008)
- Implement `/proc/net/dev` file
- Fixed 0.000ms pings due to timer resolution.
- Implemented `/proc/self/environ` (GH #730)
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 664

Number of non-Passing (failing, skipped, etc...): 263

Build 14959

For general Windows information on build 14959 visit the [Windows Blog ↗](#).

Fixed

- Improved Pico Process notification for Windows. Additional information found on the [WSL Blog](#).
- Improved stability with Windows interoperability
- Fixed error 0x80070057 when launching bash.exe when Enterprise Data Protection (EDP) is enabled
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 665

Number of non-Passing (failing, skipped, etc...): 263

Build 14955

For general Windows information on build 14955 visit the [Windows Blog ↗](#).

Fixed

- Due to circumstances beyond our control there are no updates in this build for the Windows Subsystem for Linux. Regularly scheduled updates will resume on the next release.

LTP Results:

Number of Passing Test: 665

Number of non-Passing (failing, skipped, etc...): 263

Build 14951

For general Windows information on build 14951 visit the [Windows Blog ↗](#).

New Feature: Windows / Ubuntu Interoperability

Windows binaries can now be invoked directly from the WSL command line. This gives users the ability to interact with their Windows environment and system in a way that has not been possible. As a quick example, it is now possible for users to run the following commands:

Bash

```
$ export PATH=$PATH:/mnt/c/Windows/System32
$ notepad.exe
$ ipconfig.exe | grep IPv4 | cut -d: -f2
$ ls -la | findstr.exe foo.txt
$ cmd.exe /c dir
```

More information can be found at:

- [WSL Team Blog for Interop](#)
- [WSL File Systems Documentation](#)

Fixed

- Ubuntu 16.04 (Xenial) is now installed for all new WSL instances. Users with existing 14.04 (Trusty) instances will not be automatically upgraded.
- Locale set during install is now displayed
- Terminal improvements including bug where redirecting a WSL process to a file does not always work
- Console lifetime should be tied to bash.exe lifetime
- Console window size should use visible size, not buffer size
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 665

Number of non-Passing (failing, skipped, etc...): 263

Build 14946

For general Windows information on build 14946 visit the [Windows Blog](#).

Fixed

- Fixed an issue that prevented creating WSL user accounts for users with NT usernames that contain spaces or quotes.
- Change VolFs and DrvFs to return 0 for directory's link count in stat
- Support IPV6_MULTICAST_HOPS socket option.

- Limit to a single console I/O loop per tty. Example: the following command is possible:
 - bash -c "echo data" | bash -c "ssh user@example.com 'cat > foo.txt'"
- replace spaces with tabs in /proc/cpuinfo (GH #1115)
- DrvFs now appears in mountinfo with a name that matches mounted Windows volume
- /home and /root now appear in mountinfo with correct names
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 665

Number of non-Passing (failing, skipped, etc...): 263

Build 14942

For general Windows information on build 14942 visit the [Windows Blog](#).

Fixed

- A number of bugchecks addressed, including the "ATTEMPTED EXECUTE OF NOEXECUTE MEMORY" networking crash which was blocking SSH
- inotify support for notifications generated from Windows applications on DrvFs is now in
- Implement TCP_KEEPIDLE and TCP_KEEPINTVL for mongod. (GH #695)
- Implement the pivot_root system call
- Implement socket option for SO_DONTROUTE
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 665

Number of non-Passing (failing, skipped, etc...): 263

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

pivot_root

Build 14936

For general Windows information on build 14936 visit the [Windows Blog ↗](#).

Note: WSL will install Ubuntu version 16.04 (Xenial) instead of Ubuntu 14.04 (Trusty) in an upcoming release. This change will apply to Insiders installing new instances (lxrun.exe /install or first run of bash.exe). Existing instances with Trusty will not be upgraded automatically. Users can upgrade their Trusty image to Xenial using the do-release-upgrade command.

Known Issue

WSL is experiencing an issue with some socket implementations. The bugcheck manifests itself as a crash with the error "ATTEMPTED EXECUTE OF NOEXECUTE MEMORY". The most common manifestation of this issue is a crash when using ssh. The root cause is fixed on internal builds and will be pushed to Insiders at the earliest opportunity.

Fixed

- Implemented the chroot system call
- Improvements in inotify including support for notifications generated from ~~Windows applications on DrvFs~~
 - Correction: Inotify support for changes originating from Windows applications not available at this time.
- Socket binding to `IPV6::<port n>` now supports `IPV6_V6ONLY` (GH #68, #157, #393, #460, #674, #740, #982, #996)
- WNOWAIT behavior for waitid systemcall implemented (GH #638)
- Support for IP socket options `IP_HDRINCL` and `IP_TTL`
- Zero-length read() should return immediately (GH #975)
- Correctly handle filenames and filename prefixes that don't include a NULL terminator in a .tar file.
- epoll support for /dev/null
- Fix /dev/alarm time source

- Bash -c now able to redirect to a file
- Additional bugfixes and improvements

LTP Results:

Number of Passing Test: 664

Number of non-Passing (failing, skipped, etc...): 264

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

`chroot`

Build 14931

For general Windows information on build 14931 visit the [Windows Blog ↗](#).

Fixed

- Due to circumstances beyond our control there are no updates in this build for the Windows Subsystem for Linux. Regularly scheduled updates will resume in the next release.

Build 14926

For general Windows information on build 14926 visit the [Windows Blog ↗](#).

Fixed

- Ping now works in consoles which do not have administrator privileges
- Ping6 now supported, also without administrator privileges
- Inotify support for files modified through WSL. (GH #216)
 - Flags supported:
 - `inotify_init1`: `LX_O_CLOEXEC`, `LX_O_NONBLOCK`

- inotify_add_watch events: LX_IN_ACCESS, LX_IN MODIFY, LX_IN_ATTRIB, LX_IN_CLOSE_WRITE, LX_IN_CLOSE_NOWRITE, LX_IN_OPEN, LX_IN_MOVED_FROM, LX_IN_MOVED_TO, LX_IN_CREATE, LX_IN_DELETE, LX_IN_DELETE_SELF, LX_IN_MOVE_SELF
- inotify_add_watch attributes: LX_IN_DONT_FOLLOW, LX_IN_EXCL_UNLINK, LX_IN_MASK_ADD, LX_IN_ONESHOT, LX_IN_ONLYDIR
- read output: LX_IN_ISDIR, LX_IN_IGNORED
- Known issue: Modifying files from Windows applications does not generate any events
- Unix socket now supports SCM_CREDENTIALS

LTP Results:

Number of Passing Test: 651

Number of non-Passing (failing, skipped, etc...): 258

Build 14915

For general Windows information on build 14915 visit the [Windows Blog ↗](#).

Fixed

- Socketpair for unix datagram sockets (GH #262)
- Unix socket support for SO_REUSEADDR
- UNIX socket support for SO_BROADCAST (GH #568)
- Unix socket support for SOCK_SEQPACKET (GH #758, #546)
- Adding support for unix datagram socket send, recv and shutdown
- Fix bugcheck due to invalid mmap parameter validation for non-fixed addresses. (GH #847)
- Support for suspend / resume of terminal states
- Support for TIOCPKT ioctl to unblock the Screen utility (GH #774)
 - Known issue: Function keys not operational
- Corrected a race in TimerFd that could cause a freed member 'ReaderReady' to be accessed by LxpTimerFdWorkerRoutine (GH #814)
- Enable restartable system call support for futex, poll, and clock_nanosleep
- Added bind mount support
- unshare for mount namespace support
 - Known issue: When creating a new mount namespace with `unshare(CLONE_NEWNS)` the current working directory will continue to point to the

old namespace

- Additional improvements and bug fixes

Build 14905

For general Windows information on build 14905 visit the [Windows Blog ↗](#).

Fixed

- Restartable system calls are now supported (GH #349, GH #520)
- Symlinks to directories ending in / now operational (GH #650)
- Implemented RNDGETENTCNT ioctl for /dev/random
- Implemented the /proc/[pid]/mounts, /proc/[pid]/mountinfo and /proc/[pid]/mountstats files
- Additional bugfixes and improvements

Build 14901

First Insider build for the post Windows 10 Anniversary Update release.

For general Windows information on build 14901 visit the [Windows Blog ↗](#).

Fixed

- Fixed trailing slash issue
 - Commands such as `$ mv a/c/ a/b/` now work
- Installing now prompts if Ubuntu locale should be set to Windows locale
- Procfs support for ns folder
- Added mount and unmount for tmpfs, procfs and sysfs file systems
- Fix mknod[at] 32-bit ABI signature
- Unix sockets moved to dispatch model
- INET socket recv buffer size set using the setsockopt should be honored
- Implement MSG_CMSG_CLOEXEC unix socket receive message flag
- Linux process stdin/stdout pipe redirection (GH #2)
 - Allows for piping of bash -c commands in CMD. Example: `>dir | bash -c "grep foo"`
- Bash can now be installed on systems with multiple pagefiles (GH #538, #358)

- Default INET Socket buffer size should match that of default Ubuntu setup
- Align xattr syscalls to listxattr
- Only return interfaces with a valid IPv4 address from SIOCGIFCONF
- Fix signal default action when injected by ptrace
- implement /proc/sys/vm/min_free_kbytes
- Use machine context register values when restoring context in sigreturn
 - This resolves the issue where java and javac were hanging for some users
- Implement /proc/sys/kernel/hostname

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

`waitid`

`epoll_pwait`

Build 14388 to Windows 10 Anniversary Update

For general Windows information on build 14388 visit the [Windows Blog ↗](#).

Fixed

- Fixes to prepare for the Windows 10 Anniversary Update on 8/2
 - More information about WSL in the Anniversary Update can be found on our [blog](#)

Build 14376

For general Windows information on build 14376 visit the [Windows Blog ↗](#).

Fixed

- Removed some instances where apt-get hangs (GH #493)
- Fixed an issue where empty mounts were not handled correctly
- Fixed an issue where ramdisks were not mounted correctly

- Change unix socket accept to support flags (partial GH #451)
- Fixed common network related bluescreen
- Fixed bluescreen when accessing /proc/[pid]/task (GH #523)
- Fixed high CPU utilization for some pty scenarios (GH #488, #504)
- Additional bugfixes and improvements

Build 14371

For general Windows information on build 14371 visit the [Windows Blog ↗](#).

Fixed

- Corrected timing race with SIGCHLD and wait() when using ptrace
- Corrected some behavior when paths have a trailing / (GH #432)
- Fixed issue with rename/unlink failing due to open handles to children
- Additional bugfixes and improvements

Build 14366

For general Windows information on build 14366 visit the [Windows Blog ↗](#).

Fixed

- Fix in file creation through symlinks
- Added listxattr for Python (GH 385)
- Additional bugfixes and improvements

Syscall Support

- Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

`listxattr`

Build 14361

For general Windows information on build 14361 visit the [Windows Blog](#).

Fixed

- DrvFs is now case sensitive when running in Bash on Ubuntu on Windows.
 - Users may case.txt and CASE.TXT on their /mnt/c drives
 - Case sensitivity is only supported within Bash on Ubuntu on Windows. When outside of Bash NTFS will report the files correctly, but unexpected behavior may occur interacting with the files from Windows.
 - The root of each volume (i.e. /mnt/c) is not case sensitive
 - More information on handling these files in Windows can be found [here](#).
- Greatly enhanced pty / tty support. Applications like TMUX now supported (GH #40)
- Fixed install issue where user accounts not always created
- Optimized command line arg structure allowing for extremely long argument list. (GH #153)
- Now able to delete and chmod read_only files from DrvFs
- Fixed some instances where the terminal hangs on disconnect (GH #43)
- chmod and chown now work on tty devices
- Allow connection to 0.0.0.0 and :: as localhost (GH #388)
- Sendmsg/recvmsg now handle an IO vector length of >1 (partial GH #376)
- Users can now opt-out of auto-generated hosts file (GH #398)
- Automatically match Linux locale to the NT locale during install (GH #11)
- Added the /proc/sys/vm/swappiness file (GH #306)
- strace now exits correctly
- Allow pipes to be reopened through /proc/self/fd (GH #222)
- Hide directories under %LOCALAPPDATA%\lxss from DrvFs (GH #270)
- Better handling of bash.exe ~. Commands like "bash ~ -c ls" now supported (GH #467)
- Sockets now notify epoll read available during shutdown (GH #271)
- Ixrun /uninstall does a better job of deleting the files and folders
- Corrected ps -f (GH #246)
- Improved support for x11 apps such as xEmacs (GH #481)
- Updated initial thread stack size to match default Ubuntu setting and reporting the size correctly to the get_rlimit syscall (GH #172, #258)
- Improved reporting of pico process image names (e.g., for auditing)
- Implemented /proc/mountinfo for df command
- Fixed symlink error code for child name . and ..

- Additional improvements bugfixes and improvements

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

GETTIMER

MKNODAT

RENAMEAT

SENDFILE

SENDFILE64

SYNC_FILE_RANGE

Build 14352

For general Windows information on build 14352 visit the [Windows Blog ↗](#).

Fixed

- Fixed issue where large files were not downloaded / created correctly. This should unblock npm and other scenarios (GH #3, GH #313)
- Removed some instances where sockets hang
- Corrected some ptrace errors
- Fixed issue with WSL allowing filenames longer than 255 characters
- Improved support for non-English characters
- Add current Windows timezone data and set as default
- Unique device id's for each mount point (jre fix – GH #49)
- Correct issue with paths containing "." and ".."
- Added Fifo support (GH #71)
- Updated format of resolv.conf to match native Ubuntu format
- Some procfs cleanup
- Enabled ping for Administrator consoles (GH #18)

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all

parameters supported at this time.

FALLOCATE
EXECVE
LGETXATTR
FGETXATTR

Build 14342

For general Windows information on build 14342 the [Windows Blog](#).

Information on VolFs and DriveFs can be found on the [WSL Blog](#).

Fixed

- Fixed install issue when the Windows user had Unicode characters in the username
- The apt-get update udev workaround in the FAQ is now provided by default on first run
- Enabled symlinks in DriveFs (`/mnt/<drive>`) directories
- Symlinks now work between DriveFs and VolFs
- Addressed top level path parsing issue: `ls .//` will now work as expected
- npm install on DriveFs and the `-g` options are now working
- Fixed issue preventing PHP server from launching
- Updated default environment values, such as `$PATH` to closer match native Ubuntu
- Added a weekly maintenance task in Windows to update the apt package cache
- Fixed issue with ELF header validation, WSL now supports all Melkor options
- Zsh shell is functional
- Precompiled Go binaries are now supported
- Prompting on Bash.exe first run is now localized correctly
- `/proc/meminfo` now returns correct information
- Sockets now supported in VFS
- `/dev` now mounted as tempfs
- Fifo now supported
- Multi-core systems now showing correctly in `/proc/cpuinfo`
- Additional improvements and error messages downloading during first run
- Syscall improvements and bugfixes. Supported syscall list below.
- Additional bugfixes and improvements

Known Issues

- Not resolving '..' correctly on DriveFs in some cases

Syscall Support

Below are a list of new or enhanced syscalls that have some implementation in WSL. The syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

FCHOWNAT

GETEUID

GETGID

GETRESUID

GETXATTR

PTRACE

SETGID

SETGROUPS

SETHOSTNAME

SETXATTR

Build 14332

For general Windows information on build 14332 visit the [Windows Blog ↗](#).

Fixed

- Better resolv.conf generation including prioritizing DNS entries
- Issue with moving files and directories between /mnt and non-/mnt drives
- Tar files can now be created with symlinks
- Added default /run/lock directory on instance creation
- Update /dev/null to return proper stat info
- Additional errors when downloading during first run
- Syscall improvements and bugfixes. Supported syscall list below.
- Additional improvements bugfixes and improvements

Syscall Support

Below is the new syscall that has some implementation in WSL. The syscall on this list is supported in at least one scenario, but may not have all parameters supported at this

time.

READLINKAT

Build 14328

For general Windows information on build 14332 visit the [Windows Blog ↗](#).

New Features

- Now support Linux users. Installing Bash on Ubuntu on Windows will prompt for creation of a Linux user. For more information, visit <https://aka.ms/wslusers> ↗
- Hostname is now set to the Windows computer name, no more @localhost
- For more information on build 14328, visit: <https://aka.ms/wip14328> ↗

Fixed

- Symlink improvements for non `/mnt/<drive>` files
 - npm install now works
 - jdk / jre now installable using instructions found [here](#) ↗ .
 - known issue: symlinks do not work for Windows mounts. Functionality will be available in a later build
- top and htop now display
- Additional error messages for some install failures
- Syscall improvements and bugfixes. Supported syscall list below.
- Additional improvements bugfixes and improvements

Syscall Support

Below is a list of syscalls that have some implementation in WSL. Syscalls on this list are supported in at least one scenario, but may not have all parameters supported at this time.

ACCEPT

ACCEPT4

ACCESS

ALARM

ARCH_PRCTL

BIND

BRK
CAPGET
CAPSET
CHDIR
CHMOD
CHOWN
CLOCK_GETRES
CLOCK_GETTIME
CLOCK_NANOSLEEP
CLONE
CLOSE
CONNECT
CREAT
DUP
DUP2
DUP3
EPOLL_CREATE
EPOLL_CREATE1
EPOLL_CTL
EPOLL_WAIT
EVENTFD
EVENTFD2
EXECVE
EXIT
EXIT_GROUP
FACCESSAT
FADVISE64
FCHDIR
FCHMOD
FCHMODAT
FCHOWN
FCHOWNAT
FCNTL64
FDATASYNC
FLOCK
FORK
FSETXATTR
FSTAT64

FSTATAT64

FSTATFS64

FSYNC

FTRUNCATE

FTRUNCATE64

FUTEX

GETCPU

GETCWD

GETDENTS

GETDENTS64

GETEGID

GETEGID16

GETEUID

GETEUID16

GETGID

GETGID16

GETGROUPS

GETPEERNAME

GETPGID

GETPGRP

GETPID

GETPPID

GETPRIORITY

GETRESgid

GETRESgid16

GETRESuid

GETRESuid16

GETRLIMIT

GETRUSAGE

GETSID

GETSOCKNAME

GETSOCKOPT

GETTID

GETTIMEOFDAY

GETUID

GETUID16

GETXATTR

GET_ROBUST_LIST

GET_THREAD_AREA
INOTIFY_ADD_WATCH
INOTIFY_INIT
INOTIFY_RM_WATCH
IOCTL
IOPRIO_GET
IOPRIO_SET
KEYCTL
KILL
LCHOWN
LINK
LINKAT
LISTEN
LLSEEK
LSEEK
LSTAT64
MADVISE
MKDIR
MKDIRAT
MKNOD
MLOCK
MMAP
MMAP2
MOUNT
MPROTECT
MREMAP
MSYNC
MUNLOCK
MUNMAP
NANOSLEEP
NEWUNAME
OPEN
OPENAT
PAUSE
PERF_EVENT_OPEN
PERSONALITY
PIPE
PIPE2

POLL
PPOLL
PRCTL
PREAD64
PROCESS_VM_READV
PROCESS_VM_WRITEV
PSELECT6
PTRACE
PWRITE64
READ
READLINK
READV
REBOOT
RECV
RECVFROM
RECVMSG
RENAME
RMDIR
RT_SIGACTION
RT_SIGPENDING
RT_SIGPROCMASK
RT_SIGRETURN
RT_SIGSUSPEND
RT_SIGTIMEDWAIT
SCHED_GETAFFINITY
SCHED_GETPARAM
SCHED_GETSCHEDULER
SCHED_GET_PRIORITY_MAX
SCHED_GET_PRIORITY_MIN
SCHED_SETAFFINITY
SCHED_SETPARAM
SCHED_SETSCHEDULER
SCHED_YIELD
SELECT
SEND
SENDMMSG
SENDMSG
SENDTO

SETDOMAINNAME

SETGID

SETGROUPS

SETHOSTNAME

SETITIMER

SETPGID

SETPRIORITY

SETREGID

SETRESGID

SETRESUID

SETREUID

SETRLIMIT

SETSID

SETSOCKOPT

SETTIMEOFDAY

SETOUID

SETXATTR

SET_ROBUST_LIST

SET_THREAD_AREA

SET_TID_ADDRESS

SHUTDOWN

SIGACTION

SIGALTSTACK

SIGPENDING

SIGPROCMASK

SIGRETURN

SIGSUSPEND

SOCKET

SOCKETCALL

SOCKETPAIR

SPLICE

STAT64

STATFS64

SYMLINK

SYMLINKAT

SYNC

SYSINFO

TEE

TGKILL

TIME

TIMERFD_CREATE

TIMERFD_GETTIME

TIMERFD_SETTIME

TIMES

TKILL

TRUNCATE

TRUNCATE64

UMASK

UMOUNT

UMOUNT2

UNLINK

UNLINKAT

UNSHARE

UTIME

UTIMENSAT

UTIMES

VFORK

WAIT4

WAITPID

WRITE

WRITEV

Release Notes for Windows Subsystem for Linux kernel

Article • 08/19/2022

We've added support for WSL 2 distributions, [which use a full Linux kernel ↗](#). This Linux kernel is open source, with its source code available at the [WSL2-Linux-Kernel ↗](#) repository. This Linux kernel is delivered to your machine via Microsoft Update, and follows a separate release schedule to the Windows Subsystem for Linux which is delivered as part of the Windows image.

5.15.57.1

Release Date: Prerelease 2022/08/02

[Official GitHub release link ↗](#)

- Initial release of the WSL2 kernel based on the v5.15 kernel series
- Release rolling-lts/wsl/5.15.57.1
- Update to stable kernel version v5.15.57
- Enable Retbleed mitigations in x86_64 builds
- Enable nftables and traffic control
- Enable VGEM driver
- Fix 9p filesystem regressions since the last v5.10 WSL2 kernel
- Enable support for the Precision Time Protocol (PTP) clock device
- Enable the Landlock Linux Security Module (LSM)
 - [https://landlock.io/ ↗](https://landlock.io/)
- Enable the Miscellaneous Control Group (CGroup)
 - [https://www.kernel.org/doc/html/v5.15/admin-guide/cgroup-v2.html#misc ↗](https://www.kernel.org/doc/html/v5.15/admin-guide/cgroup-v2.html#misc)
- Disable support for the Ceph Distributed File System

5.10.102.1

Release Date: Prerelease 2022/05/09

[Official Github release link ↗](#)

- Release rolling-lts/wsl/5.10.102.1
- Update to upstream stable kernel release 5.10.102
- Disable unprivileged BPF by default
- It can be re-enabled by setting the kernel.unprivileged_bpf_disabled sysctl to 0

- Update Dxgkrnl Version to 2216
- Fix out of bounds array access for ioctl()
- Implement wait for sync VM bus messages as "killable" to allow killing a process waiting for a synchronous call to the host
- Flush device for termination when process is destroyed to avoid a deadlock when the guest process is killed

5.10.93.2

Release Date: Prerelease 2022/02/08

[Official Github release link ↗](#)

- Release rolling-lts/wsl/5.10.93.2
- Update to upstream stable kernel release 5.10.93
- Enable CH341 and CP210X USB Serial drivers
- Fix README.md build instructions to include dwarves dependency for pahole
- Switched Dxgkrnl Version to 2111
- Removed the limit of existing and total sysmem allocations
- Properly flush the device for termination during process cleanup
- Fixed SPDX-License-Identifier for d3dkmthk.h

5.10.81.1

Release Date: Prerelease 2022/02/01

[Official Github release link ↗](#)

- Release rolling-lts/wsl/5.10.81.1
- Update to upstream stable kernel release 5.10.81
- Unify the kernel configurations by enabling missing options on arm64
- Enable non-arch specific ACPI options
- Enable options related to device-mapper RAID
- Enable Btrfs
- Enable LZO and ZSTD compression

5.10.74.3

Release Date: Prerelease 2021/11/10

[Official Github release link ↗](#)

- Release rolling-lts/wsl/5.10.74.3
- Update to upstream stable kernel release 5.10.74
- Enable BPF Type Format (CONFIG_DEBUG_INFO_BTF) for use by eBPF tools (microsoft/WSL#7437)
- Updated Dxgkrnl version to 2110
- Enable the Buffer Sharing and Sync File Frameworks (CONFIG_DMA_SHARED_BUFFER, CONFIG_SYNC_FILE) for Dxgkrnl usage
- Fix Dxgkrnl build failure with GCC versions older than 8.1 (microsoft/WSL#7558)

5.10.60.1

Release Date: 2021/11/02 (Prerelease 2021/10/05)

[Official Github release link ↗](#)

- Release rolling-lts/wsl/5.10.60.1
- Update to upstream stable kernel release 5.10.60
- Enable virtio-pmem with support for PCI BAR-relative addresses
- Enable vPCI support under Hyper-V for arm64
- Enable io_uring support
- Enable USB over IP support
- Enable paravirtualized spinlock support for x86_64
- Refresh dxgkrnl driver to pick up bug fixes and code cleanups
- Enable NFS client support for NFSv4.1
- Enable USB kernel configuration options for interacting with an Arduino over USB
- Provide a WSL2-specific README.md

5.10.43.3

Release Date: Prerelease 2021/07/12

[Official Github release link ↗](#)

- Version rolling-lts/wsl/5.10.43.3
- Update to upstream stable kernel release 5.10.43
- Improved dxgkrnl driver
- New revision of arm64 Linux on Hyper-V series (v9)
- Always use the Hyper-V hypercall interface on arm64 guests to support running on all versions of Windows

5.10.16.3

Release Date: 2021/07/20 (Prerelease 2021/04/16)

[Official Github release link ↗](#)

- Fixes [GH 5324 ↗](#)
- Adds support for LUKS encrypted disks using wsl --mount

5.4.91

Release Date: Prerelease 2021/02/22

[Official Github release link ↗](#)

5.4.72

Release Date: 2021/01/21

[Official Github release link ↗](#)

- Fix config for 5.4.72

5.4.51-microsoft-standard

Release Date: Prerelease - 2020/10/22

[Official Github release link ↗](#).

- Stable release of 5.4.51

4.19.128-microsoft-standard

Release Date: 2020/09/15

[Official Github release link ↗](#).

- This is a stable release of 4.19.128
- Fix dxgkrnl driver IOCTL memory corruption

4.19.121-microsoft-standard

Release Date: Prerelease

[Official Github release link ↗](#).

- Drivers: hv: vmbus: hook up dxgkrnl
- Added support for GPU Compute

4.19.104-microsoft-standard

Release Date: 2020/06/09

[Official Github release link ↗](#).

- Update WSL config for 4.19.104

4.19.84-microsoft-standard

Release Date: 2019/12/11

[Official Github release link ↗](#).

- This is the 4.19.84 stable release

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Release Notes for Windows Subsystem for Linux in the Microsoft Store

Article • 02/22/2024

The release notes for [WSL inside of the Microsoft Store](#) can be found on the [Microsoft/WSL GitHub repository releases page](#). Please see that list for latest updates.

Known Issues:

- Launching Windows Subsystem for Linux from session zero does not currently work (for example from an ssh connection).



Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).



Windows Subsystem for Linux feedback

Windows Subsystem for Linux is an open source project. Select a link to provide feedback:

[Open a documentation issue](#)

[Provide product feedback](#)