

# Linux용 Windows 하위 시스템 설명서

아티클 • 2023. 03. 21. • 읽는 데 5분 걸림

WSL(Linux용 Windows 하위 시스템)을 사용하면 개발자가 기존 가상 머신의 오버헤드 또는 듀얼 부팅 설정 없이 대부분의 명령줄 도구, 유ти리티 및 애플리케이션을 비롯한 GNU/Linux 환경을 수정하지 않고 Windows에서 직접 실행할 수 있습니다.

WSL 설치

## 자세한 정보

- WSL(Linux용 Windows 하위 시스템)이란?
- WSL 2의 새로운 기능
- WSL 1과 WSL 2 비교
- 질문과 대답

## 시작

- WSL 설치
- Windows Server에 Linux 설치
- 수동 설치 단계
- WSL 개발 환경 설정에 대한 모범 사례

## Windows 참가자 프로그램에 참여하여 WSL 미리 보기 기능 사용해 보기

WSL의 최신 기능이나 업데이트를 시도하려면 [Windows 참가자 프로그램](#)에 참여하세요. Windows 참가자에 가입하면 Windows 설정 메뉴에서 미리 보기 빌드를 수신할 채널을 선택할 수 있습니다. 다음 중 하나를 선택할 수 있습니다.

- 개발 채널: 가장 최근의 업데이트이지만 안정성은 낮습니다.
- 베타 채널: 개발 채널보다 더 안정적인 빌드인 얼리 어답터에게 이상적입니다.
- 릴리스 미리 보기 채널: 일반 대중에게 제공되기 직전에 다음 버전의 Windows 수정 사항 및 주요 기능을 미리 봅니다.

## 팀 블로그

- [비디오 및 블로그 모음이 포함된 게시물에 대한 개요](#)

- 명령줄 블로그 ↗ (활성)
- Linux용 Windows 하위 시스템 블로그(기록)

## 피드백 제공

- GitHub 문제 추적기: WSL ↗
- GitHub 문제 추적기: WSL 설명서 ↗

## 관련 비디오

### WSL BASICS

1. WSL(Linux용 Windows 하위 시스템)이란? ↗ | One Dev Question(0:40)
2. Windows 개발자입니다. WSL을 사용해야 하는 이유는 무엇인가요? ↗ | One Dev Question(0:58)
3. Linux 개발자입니다. WSL을 사용해야 하는 이유는 무엇인가요? ↗ | One Dev Question(1:04)
4. Linux란? ↗ | One Dev Question(1:31)
5. Linux 배포판이란? ↗ | One Dev Question(1:04)
6. WSL은 가상 머신 또는 이중 부팅과 어떻게 다른가요? ↗ | One Dev Question
7. Linux용 Windows 하위 시스템이 만들어진 이유는 무엇인가요? ↗ | One Dev Question(1:14)
8. WSL에서 내 컴퓨터의 파일에 액세스하려면 어떻게 해야 하나요? ↗ | One Dev Question(1:41)
9. WSL은 Windows와 어떻게 통합하나요? ↗ | One Dev Question(1:34)
10. 터미널의 홈 디렉터리에서 시작하도록 WSL 배포판을 구성하려면 어떻게 해야 하나요? ↗ | One Dev Question(0:47)
11. 스크립팅에 WSL을 사용할 수 있나요? ↗ | One Dev Question(1:04)
12. Windows에서 Linux 도구를 사용하려는 이유는 무엇인가요? ↗ | One Dev Question(1:20)
13. WSL에서 Microsoft Store의 배포판 이외의 배포판을 사용할 수 있나요? ↗ | One Dev Question(1:03)

### WSL DEMOS

1. WSL2: Linux용 Windows 하위 시스템에서 더 빠르게 코딩하세요! ↗ | 탭 및 공백 (13:42)
2. WSL: Linux GUI Apps 실행 ↗ | 탭 및 공백(17:16)
3. WSL 2: USB 디바이스 연결 ↗ | 탭 및 공백(10:08)
4. WSL 2를 사용한 GPU 가속 Machine Learning ↗ | 탭 및 공백(16:28)
5. Visual Studio Code: SSH, VM 및 WSL을 사용한 원격 개발 ↗ | 탭 및 공백(29:33)

6. Windows 개발 도구 업데이트: WSL, 터미널, 패키지 관리자 등 ↗ | 탭 및 공백(20:46)
7. WSL을 사용하여 Node.js 앱 빌드 ↗ | 강조 표시(3:15)
8. WSL 2의 새로운 메모리 회수 기능 ↗ | 데모(6:01)
9. Windows에서 웹 개발(2019) ↗ | 데모(10:39)

## WSL 심층 분석

1. Windows 11 WSL - Craig Loewen과 Scott Hanselman의 데모 ↗ | Windows 수요일 (35:48)
2. WSL 및 Linux 배포 – Hayden Barnes와 Kayla Cinnamon ↗ | Windows 수요일(37:00)
3. Oh My Posh 및 WSL Linux 배포판으로 터미널 사용자 지정 ↗ | Windows 수요일 (33:14)
4. 웹 개발자 Sarah Tamsin과 Craig Loewen이 웹 개발 및 콘텐츠 제작, WSL에 대해 채팅 ↗ | 개발 관점(12:22)
5. WSL이 Windows에서 Linux 파일에 액세스하는 방법 ↗ | 심층 분석(24:59)
6. Linux 아키텍처용 Windows 하위 시스템: 심층 분석 ↗ | 빌드 2019(58:10)

# Linux용 Windows 하위 시스템이란?

아티클 • 2023. 03. 21. • 읽는 데 2분 걸림

Linux용 Windows 하위 시스템을 사용하면 개발자가 기존 가상 머신의 오버헤드 또는 듀얼 부팅 설정 없이 대부분의 명령줄 도구, 유ти리티 및 애플리케이션을 비롯한 GNU/Linux 환경을 수정하지 않고 Windows에서 직접 실행할 수 있습니다.

다음을 수행할 수 있습니다.

- Microsoft Store [\[↗\]](#)에서 즐겨찾는 GNU/Linux 배포를 선택합니다.
- `grep`, `sed`, `awk` 또는 다른 ELF-64 이진 파일과 같은 일반적인 명령줄 도구를 실행합니다.
- 다음을 포함하여 Bash 셸 스크립트 및 GNU/Linux 명령줄 애플리케이션을 실행합니다.
  - 도구: vim, emacs, tmux
  - 언어: NodeJS, Javascript, Python, Ruby, C/C++, C# & F#, Rust, Go 등
  - 서비스: SSHD, MySQL, Apache, lighttpd, MongoDB, PostgreSQL.
- 자체 GNU/Linux 배포 패키지 관리자를 사용하여 추가 소프트웨어를 설치합니다.
- Unix와 같은 명령줄 셸을 사용하여 Windows 애플리케이션을 호출합니다.
- Windows에서 GNU/Linux 애플리케이션을 호출합니다.
- Windows 데스크톱에 직접 통합된 GNU/Linux 그래픽 애플리케이션 실행
- 기계 학습, 데이터 과학 시나리오 등에 GPU 가속 사용

WSL 설치

<https://www.youtube-nocookie.com/embed/48k317kOxqg> [\[↗\]](#)

## WSL 2란?

WSL 2는 Linux용 Windows 하위 시스템 아키텍처의 새로운 버전으로, Linux용 Windows 하위 시스템이 Windows에서 ELF64 Linux 이진 파일을 실행할 수 있게 해줍니다. WSL 2의 주 목표는 파일 시스템 성능을 높이고 전체 시스템 호출 호환성을 추가하는 것입니다.

이 새 아키텍처는 이러한 Linux 이진 파일이 Windows 및 컴퓨터의 하드웨어와 상호 작용하는 방식을 변경하되, WSL 1(현재 널리 사용 가능한 버전)과 동일한 사용자 환경을 제공합니다.

개별 Linux 배포는 WSL 1 또는 WSL 2 아키텍처를 사용하여 실행할 수 있습니다. 언제든지 각 배포를 업그레이드하거나 다운그레이드할 수 있으며 WSL 1 및 WSL 2 배포를 함께 실행할 수 있습니다. WSL 2는 실제 Linux 커널을 실행하는 이점을 제공하는 완전히 새로운 아키텍처를 사용합니다.

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

# WSL 버전 비교

아티클 • 2023. 03. 21. • 읽는 데 14분 걸림

WSL 2가 기본값인 이유와 설치된 Linux 배포를 이전 WSL 1 아키텍처로 전환할 수 있는 특정 시나리오 또는 예외를 포함하여 다양한 WSL 버전에 대해서 자세히 알아봅니다.

## WSL 1과 WSL 2 비교

이 가이드에서는 [WSL 2 대신 WSL 1 사용에 대한 예외](#)를 포함하여 WSL 1과 WSL 2를 비교합니다. WSL 1과 WSL 2의 주요 차이점은 관리되는 VM 내에서 실제 Linux 커널 사용, 전체 시스템 호출 호환성 지원, Linux 및 Windows 운영 체제 전반의 성능입니다. WSL 2는 Linux 배포판을 설치할 때 현재 기본 버전이며, 최신의 가장 뛰어난 가상화 기술을 사용하여 경량 유ти리티 VM(가상 머신) 내에서 Linux 커널을 실행합니다. 배포가 현재 WSL 1을 실행하고 있고 WSL 2로 업데이트하려는 경우, [WSL 1에서 WSL 2로 업데이트](#)를 참조하십시오.

## 기능 비교

기능	WSL 1	WSL 2
Windows와 Linux 통합	✓	✓
빠른 부팅 시간	✓	✓
기존 Virtual Machines보다 작은 리소스 공간	✓	✓
현재 버전의 VMware 및 VirtualBox에서 실행	✓	✓
관리 VM	✗	✓
전체 Linux 커널	✗	✓
전체 시스템 호출 호환성	✗	✓
OS 파일 시스템 간 성능	✓	✗

위의 비교 표에서 알 수 있듯이 WSL 2 아키텍처는 여러 가지 면에서 WSL 1보다 성능이 우수합니다. 단, OS 파일 시스템의 성능은 제외하면 프로젝트에서 작업하기 위해 실행 중인 도구와 동일한 운영 체제에 프로젝트 파일을 저장하여 해결할 수 있습니다.

WSL 2는 Windows 11 또는 Windows 10, 버전 1903, 빌드 18362 이상에서만 사용할 수 있습니다. **Windows 로고 키 + R**을 선택하고 `winver`를 입력한 다음, **확인**을 선택하여 Windows 버전을 확인합니다. (또는 Windows 명령 프롬프트에서 `ver` 명령을 입력합니

다.) 최신 Windows 버전을 업데이트해야 할 수도 있습니다. 14393보다 낮은 빌드의 경우 WSL은 전혀 지원되지 않습니다.

### ① 참고

WSL 2는 VMware 15.5.5 이상  [및 VirtualBox 6 이상](#) 에서 작동합니다. FAQ에서 자세히 알아보세요.

## WSL 2의 새로운 기능

WSL 2는 기본 아키텍처를 전체적으로 점검했으며, 가상화 기술과 Linux 커널을 사용하여 새로운 기능을 사용하도록 설정합니다. 이 업데이트의 주요 목표는 파일 시스템 성능을 높이고, 전체 시스템 호출 호환성을 추가하는 것입니다.

- [WSL 2 시스템 요구 사항](#)
- [Linux 배포 버전을 WSL 1에서 WSL 2로 설정](#)
- [WSL 2에 대한 질문과 대답](#)

## WSL 2 아키텍처

기존 VM 환경은 부팅 속도가 느리고, 격리되어 있으며, 많은 리소스를 사용하고, 관리하는데 시간이 걸릴 수 있습니다. WSL 2는 이러한 단점이 없습니다.

WSL 2는 Windows와 Linux 간의 원활한 통합, 빠른 부팅 시간, 작은 리소스 설치 공간을 포함하여 WSL 1의 이점을 제공하며 VM을 구성하거나 관리할 필요가 없습니다. WSL 2는 VM을 사용하지만 WSL 1과 동일한 사용자 환경을 유지하면서 백그라운드에서 관리되고 실행됩니다.

## 전체 Linux 커널

WSL 2의 Linux 커널은 kernel.org에서 제공되는 원본을 기반으로 하여 Microsoft를 통해 안정적인 최신 분기에서 구축되었습니다. 이 커널은 WSL 2에 맞게 특별히 튜닝되어 크기와 성능을 최적화하여 Windows에서 놀라운 Linux 환경을 제공합니다. 커널은 Windows 업데이트를 통해 서비스를 제공하므로 직접 관리할 필요 없이 최신 보안 수정과 향상된 커널 기능을 얻을 수 있습니다.

WSL 2 Linux 커널은 [오픈 소스](#)입니다. 자세히 알아보려면 구축한 팀에서 작성한 [Windows를 사용하여 Linux 커널 전달](#) 블로그 게시물을 확인하세요.

[Linux용 Windows 하위 시스템 커널의 릴리스 정보](#)에서 자세히 알아봅니다.

## 파일 IO 성능 향상

git clone, npm install, apt update, apt upgrade 등과 같은 파일 집약적 작업은 모두 WSL 2를 통해 훨씬 더 빠르게 수행됩니다.

실제 속도 증가는 실행되는 앱과 이 앱에서 파일 시스템과 상호 작용하는 방법에 따라 달라집니다. 압축된 tarball의 압축을 푸는 경우 WSL 2의 초기 버전은 WSL 1보다 최대 20배 더 빠르게 실행되며, 다양한 프로젝트에서 git clone, npm install 및 cmake를 사용하는 경우 약 2~5배 더 빠르게 실행됩니다.

## 전체 시스템 호출 호환성

Linux 이진 파일은 시스템 호출을 사용하여 파일 액세스, 메모리 요청, 프로세스 만들기 등의 함수를 수행합니다. WSL 1은 WSL 팀에서 개발한 번역 계층을 사용했지만 WSL 2에는 전체 시스템 호출 호환성을 지원하는 자체 Linux 커널이 포함되어 있습니다. 이점은 다음과 같습니다.

- [Docker](#) 등과 같이 WSL 내에서 실행할 수 있는 새로운 앱의 전체 세트입니다.
- Linux 커널에 대한 모든 업데이트를 즉시 사용할 수 있습니다. (WSL 팀에서 업데이트를 구현하고 변경 내용을 추가할 때까지 기다릴 필요가 없습니다.)

## WSL 2가 아닌 WSL 1의 사용에 대한 예외

더 빠른 성능과 100% 시스템 호출 호환성을 제공하는 WSL 2를 사용하는 것이 좋습니다. 그러나 WSL 1을 사용하는 것이 더 나을 수 있는 몇 가지 특정 시나리오가 있습니다. 다음과 같은 경우 WSL 1을 사용하는 것이 좋습니다.

- 프로젝트 파일을 Windows 파일 시스템에 저장해야 합니다. WSL 1을 사용하면 Windows에서 탑재된 파일에 더 빠르게 액세스할 수 있습니다.
  - WSL Linux 배포를 사용하여 Windows 파일 시스템의 프로젝트 파일에 액세스하고 이러한 파일을 Linux 파일 시스템에 저장할 수 없는 경우 WSL 1을 사용하여 OS 파일 시스템에서 더 빠른 성능을 얻을 수 있습니다.
- Windows 및 Linux 도구를 모두 동일한 파일에 사용하여 크로스 컴파일해야 하는 프로젝트입니다.
  - Windows 및 Linux 운영 체제 전체의 파일 성능은 WSL 2보다 WSL 1에서 더 빠르므로 Windows 애플리케이션을 사용하여 Linux 파일에 액세스하는 경우 현재 WSL 1에서 더 빠른 성능을 얻을 수 있습니다.
- 프로젝트가 직렬 포트 또는 USB 디바이스에 액세스해야 합니다. 그러나 USB 디바이스 지원은 이제 USBIPD-WIN 프로젝트를 통해 WSL 2에 사용할 수 있습니다. 설정 단계는 [USB 디바이스 연결](#)을 참조하세요.

- WSL 2는 직렬 포트 액세스를 지원하지 않습니다. [FAQ](#) 또는 [WSL GitHub 리포지토리](#)에서 직렬 지원에 대해 자세히 알아보세요.
- 메모리 요구 사항이 엄격합니다.
  - WSL 2의 메모리는 사용자가 사용함에 따라 사용량이 증가하고 축소됩니다. 프로세스에서 메모리를 해제하면 자동으로 Windows에 반환됩니다. 그러나 현재 WSL 2는 WSL 인스턴스가 종료되어야만 메모리의 캐시된 페이지를 Windows로 다시 해제합니다. WSL 세션이 오랫동안 실행되거나 사용자가 매우 많은 파일에 액세스하는 경우 이 캐시가 Windows의 메모리를 사용할 수 있습니다. [WSL GitHub 리포지토리 문제 4166](#)에서 이 환경을 개선하기 위한 작업을 추적하고 있습니다.
- VirtualBox를 사용하는 경우 실행 중인 버전과 WSL 2와 호환되는지 여부를 고려해야 할 수 있습니다. 전체 논의는 [는 WSL GitHub 리포지토리 문제 798](#)을 참조하세요. VirtualBox v6.1.16은 WSL 2에서 잘 작동하지만 다른 버전에 문제가 발생할 수 있습니다.)
- Linux 배포판을 사용하여 호스트 머신과 동일한 네트워크에 IP 주소를 사용하는 경우 WSL 2를 실행하기 위해 해결 방법을 설정해야 할 수 있습니다. WSL 2는 hyper-v 가상 머신으로 실행되고 있습니다. 이는 WSL 1에서 사용되는 브리지된 네트워크 어댑터의 변경입니다. 즉, WSL 2는 호스트 NIC(네트워크 인터페이스 카드)에 브리지되는 대신 해당 가상 네트워크에 NAT(Network Address Translation) 서비스를 사용하므로 다시 시작할 때 고유한 IP 주소가 변경됩니다. WSL 2 서비스의 TCP 포트를 호스트 OS에 전달하는 문제 및 해결 방법에 대한 자세한 내용은 [WSL GitHub 리포지토리 문제 4150, NIC 브리지 모드\(TCP 해결 방법\)](#)을 참조하세요.

### ① 참고

VS Code 원격 WSL 확장 [에서](#) Linux 명령줄 도구를 사용하여 프로젝트 파일을 Linux 파일 시스템에 저장하도록 시도하는 것을 고려할 수 있지만, Linux 및 Windows 간의 작업과 관련된 성능을 저하시키지 않고 Windows에서 VS Code를 사용하여 인터넷 브라우저에서 프로젝트를 작성, 편집, 디버그 또는 실행할 수도 있습니다. 자세한 정보를 알아보세요.

## Microsoft Store에서 WSL

WSL은 Windows OS 이미지에서 Microsoft Store를 통해 사용할 수 있는 패키지로 업데이트 기능을 해제했습니다. 즉, Windows 운영 체제의 업데이트를 기다릴 필요 없이 사용 가능한 즉시 업데이트와 서비스가 더 빨라집니다.

WSL은 원래 Linux 배포를 설치하기 위해 사용하도록 설정해야 하는 선택적 구성 요소로 Windows 운영 체제에 포함되었습니다. 스토어의 WSL은 동일한 사용자 환경을 가지며 동일한 제품이지만 전체 OS 업데이트가 아닌 스토어 패키지로 업데이트 및 서비스를 받습니다.

니다. Windows 버전 19044 이상부터 `wsl.exe --install` 명령을 실행하면 Microsoft Store에서 WSL 서비스 업데이트가 설치됩니다. ([이 업데이트를 발표하는 블로그 게시물을 참조하세요 ↗](#).) 이미 WSL을 사용 중인 경우 `wsl.exe --update`을 실행하여 최신 WSL 기능을 수신하고 저장소에서 서비스를 받을 수 있도록 업데이트할 수 있습니다.

# WSL의 기본 명령

아티클 • 2023. 03. 21. • 읽는 데 13분 걸림

아래의 WSL 명령은 PowerShell 또는 Windows 명령 프롬프트에서 지원하는 형식으로 나열되어 있습니다. Bash/Linux 배포 명령줄에서 이러한 명령을 실행하려면 `wsl`을 `wsl.exe`로 바꿔야 합니다. 명령의 전체 목록을 보려면 `wsl --help`을(를) 실행합니다.

## 설치

PowerShell

```
wsl --install
```

WSL 및 Linux의 기본 Ubuntu 배포판을 설치합니다. [자세한 정보를 알아보세요](#). 이 명령을 사용하여 `wsl --install <Distribution Name>`을(를) 실행하여 추가 Linux 배포를 설치할 수도 있습니다. 유효한 배포 이름 목록을 보려면 `wsl --list --online`을(를) 실행합니다.

표시되는 옵션은 다음과 같습니다.

- `--distribution`: 설치할 Linux 배포를 지정합니다. `wsl --list --online`을(를) 실행하여 사용 가능한 배포를 찾을 수 있습니다.
- `--no-launch`: Linux 배포를 설치하지만 자동으로 시작하지는 않습니다.
- `--web-download`: Microsoft Store를 사용하는 대신 온라인 원본에서 설치합니다.

WSL이 설치되지 않은 경우 옵션은 다음과 같습니다.

- `--inbox`: Microsoft Store를 사용하는 대신 Windows 구성 요소를 사용하여 WSL을 설치합니다. (*WSL 업데이트는 스토어를 통해 사용 가능으로 푸시되는 대신 Windows 업데이트를 통해 수신됩니다.*)
- `--enable-wsl1`: Microsoft Store 버전의 WSL을 설치하는 동안 "Linux용 Windows 하위 시스템" 선택적 구성 요소를 사용하도록 설정하여 WSL 1을 사용하도록 합니다.
- `--no-distribution`: WSL을 설치할 때 배포를 설치하지 마세요.

### ① 참고

Windows 10 또는 이전 버전에서 WSL을 실행하는 경우 배포를 지정하기 위해 명령에 플래그 `--install` 를 `wsl --instal -d <distribution name>` 포함 `-d` 해야 할 수 있습니다.

# 사용 가능한 Linux 배포판 나열

PowerShell

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

온라인 스토어를 통해 받을 수 있는 Linux 배포판 목록을 참조하세요. 이 명령은 `wsl -l -o`으로 입력할 수도 있습니다.

## 설치된 Linux 배포판 나열

PowerShell

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

상태(배포판이 실행 중인지 또는 중지되었는지 여부) 및 배포판을 실행하는 WSL 버전 (WSL 1 또는 WSL 2)을 포함하여 Windows 머신에 설치된 Linux 배포 목록을 참조하세요. [WSL 1과 WSL 2를 비교해 보세요](#). 이 명령은 `wsl -l -v`로 입력할 수도 있습니다. `list` 명령과 함께 사용할 수 있는 추가 옵션으로는 모든 배포판을 나열하는 `--all`, 현재 실행 중인 배포판만 나열하는 `--running`, 배포판 이름만 표시하는 `--quiet`가 있습니다.

## WSL 버전을 1에서 2로 설정

PowerShell

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

Linux 배포판이 실행 중인 WSL 버전(1 또는 2)을 지정하려면 `<distribution name>`을 배포판 이름으로 바꾸고 `<versionNumber>`를 1 또는 2로 바꿉니다. [WSL 1과 WSL 2를 비교해 보세요](#).

## 기본 WSL 버전 설정

PowerShell

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

WSL 1 또는 WSL 2의 기본 버전을 설정하려면 `<Version>`을 숫자 1 또는 2로 바꿔서 새 Linux 배포판 설치의 설치 기본값으로 사용할 WSL 버전을 표시합니다. 예: `wsl --set-`

default-version 2. WSL 1과 WSL 2를 비교해 보세요.

## 기본 Linux 배포판 설정

PowerShell

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

WSL 명령에서 실행에 사용할 기본 Linux 배포판을 설정하려면 `<Distribution Name>`을 기본 Linux 배포판의 이름으로 바꿉니다.

## 디렉터리를 홈으로 변경

PowerShell

```
wsl ~
```

`~`는 wsl과 함께 사용하여 사용자의 홈 디렉터리에서 시작할 수 있습니다. WSL 명령 프롬프트 내 디렉터리에서 홈으로 다시 이동하기 위해 `cd ~` 명령을 사용할 수 있습니다.

## PowerShell 또는 CMD에서 특정 Linux 배포판 실행

PowerShell

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

특정 사용자로 특정 Linux 배포판을 실행하려면 `<Distribution Name>`을 기본 Linux 배포판의 이름(즉, Debian)으로 바꾸고 `<User Name>`을 기존 사용자의 이름(예: 루트)으로 바꿉니다. 해당 사용자가 WSL 배포판에 없는 경우 오류가 발생합니다. 현재 사용자 이름을 출력하려면 `whoami` 명령을 사용합니다.

## WSL 업데이트

PowerShell

```
wsl --update
```

WSL 버전을 최신 버전으로 업데이트합니다. 표시되는 옵션은 다음과 같습니다.

- `--web-download`: Microsoft Store가 아닌 GitHub에서 최신 업데이트를 다운로드합니다.

## WSL 상태 확인

PowerShell

```
wsl --status
```

기본 배포판 유형, 기본 배포판 및 커널 버전과 같은 WSL 구성에 대한 일반 정보를 참조하세요.

## WSL 버전 확인

PowerShell

```
wsl --version
```

WSL 및 해당 구성 요소에 대한 버전 정보를 확인합니다.

## Help 명령

PowerShell

```
wsl --help
```

WSL에서 사용할 수 있는 옵션 및 명령 목록을 참조하세요.

## 특정 사용자로 실행

PowerShell

```
wsl -u <Username>, `wsl --user <Username>
```

WSL을 지정된 사용자로 실행하려면 `<Username>`를 WSL 배포에 있는 사용자의 이름으로 바꿉니다.

# 배포의 기본 사용자 변경

PowerShell

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

배포 로그인에 대한 기본 사용자를 변경합니다. 사용자가 기본 사용자가 될 수 있도록 배포 내에 이미 있어야 합니다.

예를 들어 `wsl config --default-user johndoe`는 Ubuntu 배포에 대한 기본 사용자를 "johndoe" 사용자로 변경합니다.

## ① 참고

배포 이름을 확인하는 데 문제가 있는 경우 `wsl -l` 명령을 사용합니다.

## ⚠ 경고

가져온 배포에는 실행 가능한 시작 관리자가 없기 때문에 이 명령은 작동하지 않습니다. 대신 `/etc/wsl.conf` 파일을 사용하여 가져온 배포의 기본 사용자를 변경할 수 있습니다. [고급 설정 구성 문서](#)의 자동 탑재 옵션을 참조하세요.

# Shutdown

PowerShell

```
wsl --shutdown
```

실행 중인 모든 배포판과 WSL 2 경량 유틸리티 가상 머신을 즉시 종료합니다. 이 명령은 [메모리 사용 제한 변경](#) 또는 [.wslconfig 파일 변경](#)처럼 WSL 2 가상 머신 환경을 다시 시작해야 하는 인스턴스에서 필요할 수 있습니다.

# 종료

PowerShell

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

지정된 배포판을 종료하거나 실행을 중지하려면 <Distribution Name>을 대상 배포판의 이름으로 바꿉니다.

## 배포 가져오기 및 내보내기

PowerShell

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

PowerShell

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

지정된 tar 파일을 새 배포로 가져오고 내보냅니다. 파일 이름은 표준 입력을 위한 것입니다. 표시되는 옵션은 다음과 같습니다.

- `--vhd`: 가져오기/내보내기 배포 지정은 tar 파일이 아닌 .vhdx 파일이어야 합니다.
- `--version`: 가져오기 전용의 경우, 배포를 WSL 1 또는 WSL 2 배포로 가져올지 여부를 지정합니다.

## 배포 위치로 가져오기

PowerShell

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

지정된 .vhdx 파일을 새 배포로 가져옵니다. 가상 하드 디스크는 ext4 파일 시스템 형식으로 포맷되어야 합니다.

## Linux 배포판 등록 취소 또는 제거

Linux 배포는 Microsoft Store를 통해 설치할 수 있지만 이를 통해 제거할 수는 없습니다.

WSL 배포를 등록 취소하고 제거하려면 다음을 수행합니다.

PowerShell

```
wsl --unregister <DirectoryName>
```

<DistributionName>를 대상 Linux 배포의 이름으로 바꾸면 WSL에서 해당 배포를 등록 취소하여 다시 설치하거나 정리할 수 있습니다. **주의:** 등록이 취소되면 해당 배포와 관련된 모든 데이터, 설정 및 소프트웨어가 영구적으로 손실됩니다. 스토어에서 다시 설치하면 배포의 새 복사본이 설치됩니다. 예를 들어 `wsl --unregister Ubuntu`는 WSL에서 사용할 수 있는 배포에서 Ubuntu를 제거합니다. `wsl --list`를 실행하면 더 이상 나열되지 않습니다.

다른 스토어 애플리케이션과 마찬가지로 Windows 머신에서 Linux 배포판 앱을 제거할 수도 있습니다. 다시 설치하려면 Microsoft Store에서 해당 배포를 찾아 "시작"을 선택합니다.

## 디스크 또는 디바이스 탑재

PowerShell

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

<DiskPath>를 디스크가 있는 디렉터리\파일 경로로 바꿔서 모든 WSL2 배포판에 물리적 디스크를 연결하고 탑재합니다. [WSL 2에 Linux 디스크 탑재를 참조하세요](#). 다음 옵션을 사용할 수 있습니다.

- `--vhd: <Disk>`(이)가 가상 하드 디스크를 참조하도록 지정합니다.
- `--name`: 탑재 지점에 대한 사용자 지정 이름을 사용하여 디스크를 탑재합니다.
- `--bare`: WSL2에 디스크를 연결하지만 탑재하지는 않습니다.
- `--type <Filesystem>`: 디스크를 탑재할 때 사용되는 파일 시스템 유형입니다. 지정하지 않으면 기본값은 ext4입니다. 이 명령은 `wsl --mount -t <Filesystem>`으로 입력할 수도 있습니다. `blkid <BlockDevice>` 명령을 사용하여 파일 시스템 형식을 검색할 수 있습니다(예: `blkid <dev/sdb1>`).
- `--partition <Partition Number>`: 탑재할 파티션의 인덱스 번호입니다. 지정하지 않으면 전체 디스크가 기본값입니다.
- `--options <MountOptions>`: 디스크를 탑재할 때 포함할 수 있는 몇 가지 파일 시스템 관련 옵션이 있습니다. `wsl --mount -o "data-ordered"` 또는 `wsl --mount -o "data=writeback"` 같은 [ext4 탑재 옵션](#)을 예로 들 수 있습니다. 그러나 현재는 파일 시스템 관련 옵션만 지원됩니다. `ro`, `rw` 또는 `noatime`과 같은 일반 옵션은 지원되지 않습니다.

### ① 참고

wsl.exe(64비트 도구)에 액세스하기 위해 32비트 프로세스를 실행하는 경우

`C:\Windows\Sysnative\wsl.exe --command`와 같은 방식으로 이 명령을 실행해야 할

수도 있습니다.

## 디스크 분리

PowerShell

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

디스크 경로에 지정된 디스크를 분리합니다. 디스크 경로가 지정되지 않은 경우, 이 명령은 탑재된 모든 디스크를 장착 해제하여 분리합니다.

## 사용되지 않은 WSL 명령

PowerShell

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

PowerShell

```
bash [Options]
```

PowerShell

```
lxrun /[Argument]
```

이러한 명령은 WSL과 함께 설치된 Linux 배포판을 구성하는 원래 wsl 구문이지만 `wsl` 또는 `wsl.exe` 명령 구문으로 대체되었습니다.

# WSL을 사용하여 Windows에 Linux 설치

아티클 • 2023. 02. 06. • 읽는 데 13분 걸림

개발자는 Windows 컴퓨터에서 동시에 Windows와 Linux의 기능에 액세스할 수 있습니다. WSL(Linux용 Windows 하위 시스템)을 사용하면 개발자가 Linux 배포판(예: Ubuntu, OpenSUSE, Kali, Debian, Arch Linux)을 설치하고 기존 가상 머신 또는 이중 부팅 설정의 오버헤드 없이 Windows에서 직접 Linux 애플리케이션, 유ти리티 및 Bash 명령줄 도구를 사용할 수 있습니다.

## 사전 요구 사항

아래 명령을 사용하려면 Windows 10 버전 2004 이상(빌드 19041 이상) 또는 Windows 11을 실행해야 합니다. 이전 버전을 사용 중인 경우 [수동 설치 페이지](#)를 참조하세요.

## WSL 설치 명령

이제 단일 명령으로 WSL을 실행하는 데 필요한 모든 항목을 설치할 수 있습니다. 관리자 모드에서 PowerShell 또는 Windows 명령 프롬프트를 마우스 오른쪽 단추로 클릭하고 "관리자 권한으로 실행"을 선택하여 열고 `wsl --install` 명령을 입력한 다음 컴퓨터를 다시 시작합니다.

```
PowerShell  
wsl --install
```

이 명령은 WSL을 실행하고 Linux의 Ubuntu 배포를 설치하는 데 필요한 기능을 사용하도록 설정합니다. ([이 기본 분포는 변경할 수 있습니다](#)).

이전 빌드를 실행 중이거나, 설치 명령을 사용하지 않고 단계별 지침을 원하는 경우 [이전 버전의 WSL 수동 설치 단계](#)를 참조하세요.

새로 설치된 Linux 배포판을 처음 시작하면 콘솔 창이 열리고 파일의 압축이 풀리고 머신에 저장될 때까지 기다리라는 메시지가 표시됩니다. 이후의 모든 시작은 1초도 걸리지 않습니다.

### ① 참고

위의 명령은 WSL이 전혀 설치되지 않은 경우에만 작동합니다. `wsl --install`을 실행하고 WSL 도움말 텍스트를 보는 경우 `wsl --list --online`을 실행하여 사용 가능한 배포판 목록을 확인하고 `wsl --install -d <DistroName>`을 실행하여 배포판을 설

치해 보세요. WSL을 제거하려면 WSL의 레거시 버전 제거 또는 Linux 배포판 등록 취소 또는 제거를 참조하세요.

## 설치된 기본 Linux 배포판 변경

기본적으로 설치된 Linux 배포는 Ubuntu입니다. `-d` 플래그를 사용하여 변경할 수 있습니다.

- 설치된 배포판을 변경하려면 `wsl --install -d <Distribution Name>`을 입력합니다. `<Distribution Name>`을 설치하려는 배포판의 이름으로 바꿉니다.
- 온라인 스토어를 통해 다운로드할 수 있는 Linux 배포판 목록을 보려면 `wsl --list --online` 또는 `wsl -l -o`를 입력합니다.
- 초기 설치 후 추가 Linux 배포판을 설치하려면 `wsl --install -d <Distribution Name>` 명령을 사용합니다.

### 💡 팁

PowerShell 또는 명령 프롬프트가 아닌 Linux/Bash 명령줄 내에서 추가 배포판을 설치하려면 `wsl.exe --install -d <Distribution Name>` 명령 또는 `wsl.exe -l -o` 명령 (사용 가능한 배포판을 나열하려는 경우)에서 .exe를 사용해야 합니다.

설치 과정에서 문제가 발생하는 경우 [문제 해결 가이드의 설치 섹션](#)을 참조하세요.

사용 가능한 것으로 나열되지 않은 Linux 배포를 설치하려면 TAR 파일을 사용하여 [Linux 배포를 가져올](#) 수 있습니다. 또는 경우에 따라 [Arch Linux](#)와 마찬가지로 `.appx` 파일을 사용하여 설치할 수 있습니다. WSL과 함께 사용할 [사용자 지정 Linux 배포](#)를 만들 수도 있습니다.

## Linux 사용자 정보 설정

WSL을 설치한 후에는 새로 설치된 Linux 배포판의 사용자 계정 및 암호를 만들어야 합니다. 자세한 내용은 [WSL 개발 환경 설정에 대한 모범 사례](#)를 참조하세요.

## 설정 및 모범 사례

설치된 Linux 배포판의 사용자 이름 및 암호를 설정하는 방법, 기본 WSL 명령 사용, Windows 터미널 설치 및 사용자 지정, Git 버전 제어 설정, VS Code 원격 서버를 사용하여 코드 편집 및 디버깅, 파일 스토리지 모범 사례, 데이터베이스 설정, 외부 드라이브 탐

재, GPU 가속 설정 방법을 안내하는 단계별 연습에 대한 [WSL 개발 환경 설정에 대한 모범 사례](#) 가이드를 따르는 것이 좋습니다

## 실행 중인 WSL 버전 확인

PowerShell 또는 Windows 명령 프롬프트에서 `wsl -l -v` 명령을 입력하여 설치된 Linux 배포판을 나열하고 각각 설정된 WSL 버전을 확인할 수 있습니다.

새 Linux 배포판이 설치될 때 기본 버전을 WSL 1 또는 WSL 2로 설정하려면 `wsl --set-default-version <Version#>` 명령을 사용하여 `<Version#>`를 1 또는 2로 바꿉니다.

`wsl` 명령과 함께 사용되는 기본 Linux 배포판을 설정하려면 `wsl -s <DistributionName>` 또는 `wsl --setdefault <DistributionName>`를 입력하고 `<DistributionName>`를 사용하려는 Linux 배포판의 이름으로 바꿉니다. 예를 들어 PowerShell/CMD에서 `wsl -s Debian`를 입력하여 기본 배포를 Debian으로 설정합니다. 이제 Powershell에서 `wsl npm init`를 실행하면 Debian에서 `npm init` 명령이 실행됩니다.

기본 배포를 변경하지 않고 PowerShell 또는 Windows 명령 프롬프트 내에서 특정 `wsl` 배포를 실행하려면 `wsl -d <DistributionName>` 명령을 사용하여 `<DistributionName>`를 사용하려는 배포 이름으로 바꿉니다.

[WSL의 기본 명령](#) 가이드에서 자세히 알아봅니다.

## WSL 1에서 WSL 2로 버전 업그레이드

`wsl --install` 명령을 사용하여 설치된 새 Linux 설치는 기본적으로 WSL 2로 설정됩니다.

`wsl --set-version` 명령을 사용하면 WSL 2에서 WSL 1로 다운그레이드하거나 이전에 설치된 Linux 배포를 WSL 1에서 WSL 2로 업데이트할 수 있습니다.

Linux 배포판이 WSL 1 또는 WSL 2로 설정되어 있는지 확인하려면 `wsl -l -v` 명령을 사용합니다.

버전을 변경하려면 `wsl --set-version <distro name> 2` 명령을 사용하여 `<distro name>`을 업데이트하려는 Linux 배포판의 이름으로 바꿉니다. 예를 들어 `wsl --set-version Ubuntu-20.04 2`는 WSL 2를 사용하도록 Ubuntu 20.04 배포를 설정합니다.

`wsl --install` 명령을 사용하기 전에 WSL을 수동으로 설치한 경우 WSL 2에서 사용하는 [가상 머신 선택적 구성 요소를 사용하도록 설정](#)하고 아직 설치하지 않은 경우 [커널 패키지를 설치](#)해야 할 수도 있습니다.

자세히 알아보려면 WSL 명령 목록은 [WSL 명령 참조](#), 작업 시나리오에 사용할 지침은 [WSL 1 및 WSL 2 비교](#) 또는 WSL을 사용하는 우수한 개발 워크플로를 설정하기 위한 일반적인 지침은 [WSL 개발 환경 설정 모범 사례](#)를 참조하세요.

## WSL을 사용하여 여러 Linux 배포판을 실행하는 방법

WSL은 설치하려는 만큼의 다양한 Linux 배포판의 실행을 지원합니다. 여기에는 [Microsoft Store](#)에서 배포판 선택, [사용자 지정 배포판 가져오기](#) 또는 [고유한 사용자 지정 배포 빌드](#)가 포함될 수 있습니다.

설치 후 Linux 배포판을 실행할 수 있는 여러 가지 방법은 다음과 같습니다.

1. [Windows 터미널 설치\(권장\)](#) Windows 터미널을 사용하여 설치하려는 만큼의 명령 줄을 지원하고, 여러 탭 또는 창에서 명령줄을 열고, 여러 Linux 배포판 또는 다른 명령줄(PowerShell, 명령 프롬프트, PowerShell, Azure CLI 등) 간에 빠르게 전환할 수 있습니다. 고유한 색 구성표, 글꼴 스타일, 크기, 배경 이미지 및 사용자 지정 바로 가기 키를 사용하여 터미널을 원하는 대로 사용자 지정할 수 있습니다. [자세한 정보](#)
2. Windows 시작 메뉴를 방문해 설치된 배포의 이름을 입력하여 Linux 배포판을 직접 열 수 있습니다. 예: "Ubuntu". 그러면 자체 콘솔 창에서 Ubuntu가 열립니다.
3. Windows 명령 프롬프트 또는 PowerShell에서 설치된 배포의 이름을 입력할 수 있습니다. 예를 들면 다음과 같습니다. `ubuntu`
4. Windows 명령 프롬프트 또는 PowerShell에서 `wsl.exe`를 입력하여 현재 명령줄 내에서 기본 Linux 배포판을 열 수 있습니다.
5. Windows 명령 프롬프트 또는 PowerShell에서 `wsl [command]`를 입력하여 새 배포를 입력하지 않고 현재 명령줄 내에서 기본 Linux 배포판을 열 수 있습니다. `[command]`를 WSL 명령(예: `wsl -l -v`)으로 대체하여 설치된 배포를 나열하거나 `wsl pwd`를 통해 현재 디렉터리 경로가 wsl에 탑재된 위치를 확인합니다. PowerShell에서 명령 `get-date`는 Windows 파일 시스템의 날짜를 제공하고 `wsl date`는 Linux 파일 시스템의 날짜를 제공합니다.

선택하는 방법은 수행하는 작업에 따라 달라집니다. Windows 프롬프트 또는 PowerShell 창 내에서 WSL 명령줄을 열고 종료하려면 `exit` 명령을 입력합니다.

## 최신 WSL 미리 보기 기능 사용해 보기

WSL의 최신 기능이나 업데이트를 사용해 보려면 [Windows 참가자 프로그램](#)에 참여하세요. Windows 참가자 프로그램에 참여하면 Windows 설정 메뉴 내에서 미리 보기 빌드를 받을 채널을 선택하여 해당 빌드와 연결된 WSL 업데이트 또는 미리 보기 기능을 자동으로 받을 수 있습니다. 다음 중 하나를 선택할 수 있습니다.

- 개발 채널: 가장 최근의 업데이트이지만 안정성은 낮습니다.
- 베타 채널: 개발 채널보다 더 안정적인 빌드인 얼리 어답터에게 이상적입니다.
- 릴리스 미리 보기 채널: 일반 대중에게 제공되기 직전에 다음 버전의 Windows 설정 사항 및 주요 기능을 미리 봅니다.

## 추가 자료

- [Windows 명령줄 블로그](#): 이제 Windows 10 버전 2004 이상에서 사용할 수 있는 단일 명령으로 WSL 설치 ↗

# 이전 버전 WSL의 수동 설치 단계

아티클 • 2023. 02. 06. • 읽는 데 11분 걸림

간단하게 하려면 일반적으로 `wsl --install`을 사용하여 Linux용 Windows 하위 시스템을 설치하는 것이 좋지만, 이전 빌드의 Windows를 실행하는 경우에는 수동 설치 단계가 지원되지 않을 수 있습니다. 아래에 수동 설치 단계가 설명되어 있습니다. 설치 과정에서 문제가 발생하는 경우 [문제 해결 가이드의 설치 섹션](#)을 참조하세요.

## 1단계 - Linux용 Windows 하위 시스템 사용

Windows에서 Linux 배포를 설치하려면 먼저 "Linux용 Windows 하위 시스템" 옵션 기능을 사용하도록 설정합니다.

PowerShell을 관리자 권한(시작 메뉴 > PowerShell >에서 관리자 권한으로 실행 >을 마우스 오른쪽 단추로 클릭)으로 열고 다음 명령을 입력합니다.

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

이제 2단계로 이동하여 WSL 2로 업데이트하는 것이 좋습니다. 그러나 WSL 1만 설치하려면 이제 머신을 다시 시작하여 [6단계 - 선택한 Linux 배포 설치](#)로 이동할 수 있습니다. WSL 2로 업데이트하려면 머신이 다시 시작될 때까지 기다린 후 다음 단계로 이동합니다.

## 2단계 - WSL 2 실행을 위한 요구 사항 확인

WSL 2로 업데이트하려면 Windows 10을 실행해야 합니다.

- x64 시스템의 경우: **버전 1903 이상, 빌드 18362 이상**
- ARM64 시스템의 경우: **버전 2004 이상, 빌드 19041 이상**

또는 Windows 11.

### ① 참고

18362보다 낮은 빌드는 WSL 2를 지원하지 않습니다. [Windows Update Assistant ↗](#)를 사용하여 Windows 버전을 업데이트합니다.

버전 및 빌드 번호를 확인하려면 Windows 로고 키 + R을 선택하고, winver를 입력하고, 확인을 선택합니다. [설정] 메뉴에서 최신 Windows 버전으로 업데이트합니다.

#### ① 참고

Windows 10 버전 1903 또는 1909를 실행하고 있는 경우 Windows 메뉴에서 "설정"을 열고, "업데이트 & 보안"으로 이동하여 "업데이트 확인"을 선택합니다. 빌드 번호는 18362.1049 이상 또는 18363.1049 이상이고, 부 빌드 번호는 .1049 이상이어야 합니다. 자세한 정보: [WSL 2 지원이 Windows 10 버전 1903 및 1909에 제공됨](#).

## 3단계 - Virtual Machine 기능 사용

WSL 2를 설치하려면 먼저 Virtual Machine 플랫폼 옵션 기능을 사용하도록 설정해야 합니다. 이 기능을 사용하려면 머신에 [가상화 기능](#)이 필요합니다.

PowerShell을 관리자 권한으로 열어 실행합니다.

```
PowerShell
```

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

머신을 다시 시작하여 WSL 설치를 완료하고 WSL 2로 업데이트합니다.

## 4단계 - Linux 커널 업데이트 패키지 다운로드

1. 최신 패키지를 다운로드합니다.

- [x64 머신용 최신 WSL2 Linux 커널 업데이트 패키지](#)

#### ① 참고

ARM64 머신을 사용하는 경우 [ARM64 패키지](#)를 대신 다운로드하세요. 사용하고 있는 머신의 종류를 잘 모르는 경우 명령 프롬프트 또는 PowerShell을 열고 `systeminfo | find "System Type"`을 입력합니다. 주의: 비 영어 Windows 버전에서는 "시스템 유형" 문자열을 변환하여 검색 텍스트를 수정해야 할 수 있습니다. `find` 명령에 대한 따옴표는 이스케이프해야 할 수도 있습니다. 예를 들어 독일어 `systeminfo | find '"Systemtyp"'`입니다.

- 이전 단계에서 다운로드한 업데이트 패키지를 실행합니다. (실행하려면 두 번 클릭 - 관리자 권한을 요구하는 메시지가 표시되면 '예'를 선택하여 이 설치를 승인합니다.)

설치가 완료되면 새 Linux 배포를 설치할 때 WSL 2를 기본 버전으로 설정하는 다음 단계로 이동합니다. (새 Linux 설치를 WSL 1로 설정하려면 이 단계를 건너뜁니다.)

### ① 참고

자세한 내용은 [Windows 명령줄 블로그](#)에서 사용할 수 있는 [WSL2 Linux 커널업데이트 변경](#) 문서를 참조하세요.

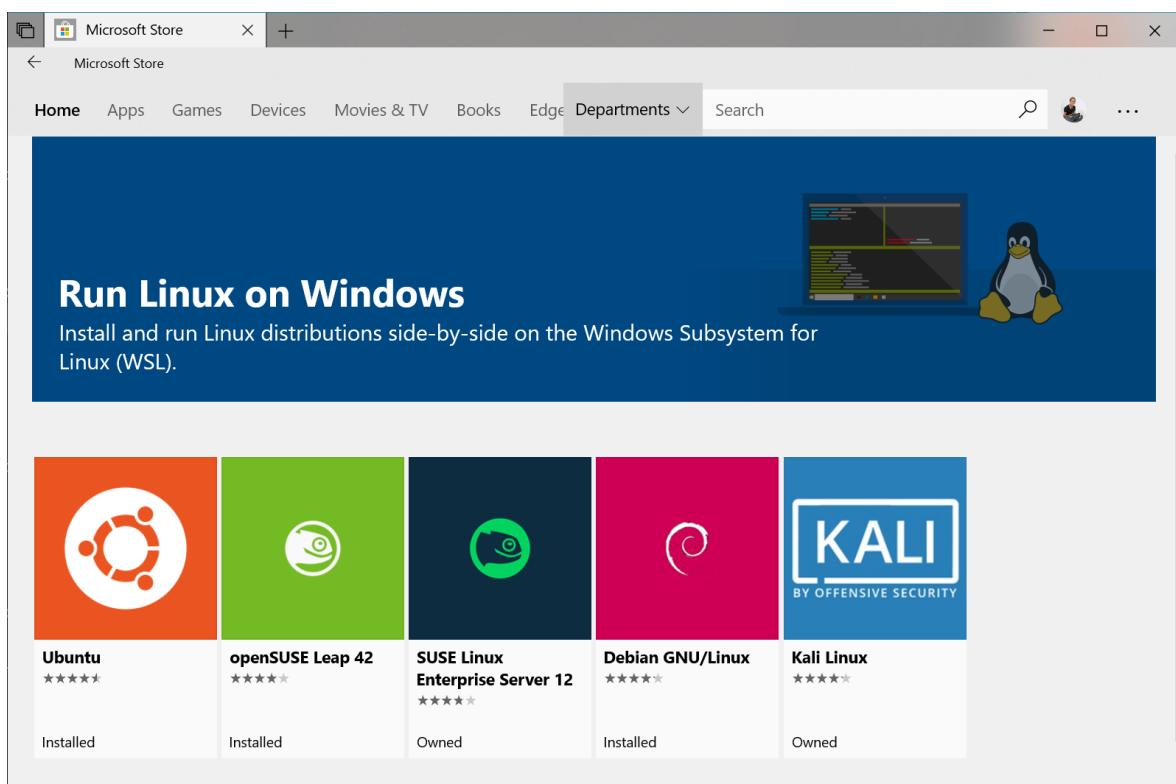
## 5단계 - WSL 2를 기본 버전으로 설정

PowerShell을 열고 이 명령을 실행하여 새 Linux 배포를 설치할 때 WSL 2를 기본 버전으로 설정합니다.

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

## 6단계 - 선택한 Linux 배포 설치

- Microsoft Store를 열고 즐겨 찾는 Linux 배포를 선택합니다.



각 배포에 대한 Microsoft Store 페이지를 여는 링크는 다음과 같습니다.

- [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\(평가판\) ↗](#)

## 2. 배포 페이지에서 "가져오기"를 선택합니다.

The screenshot shows the Microsoft Store interface for the Ubuntu 18.04 LTS app. The app is developed by Canonical Group Limited and has a rating of 22 stars. It is listed as 'Free'. The 'Get' button is highlighted in blue. The 'Share' button is greyed out. The ESRB rating is 'Everyone'. Below the main app info, there's a 'Screenshots' section showing a terminal window with command-line output related to package updates. To the right, there's a 'Description' section with text about using Ubuntu on Windows, a 'Available on' section showing a PC icon, and a 'Features' section listing 'Ubuntu', 'bash', and 'ssh'.

**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

...

**Available on**

PC

**What's new in this version**

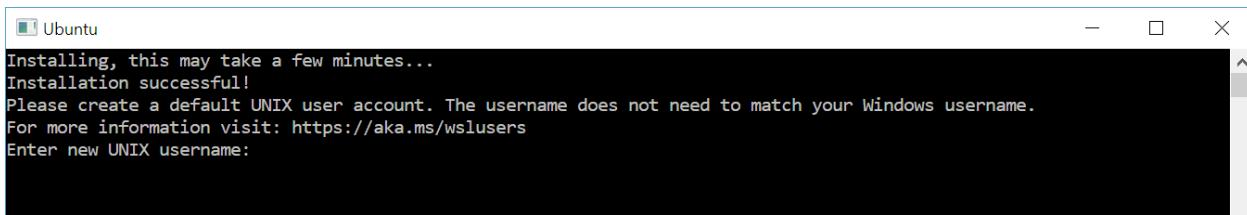
20170619.1 build of Ubuntu 16.04 LTS

**Features**

Ubuntu  
bash  
ssh

새로 설치된 Linux 배포를 처음 시작하면 콘솔 창이 열리고 파일이 압축 해제되어 PC에 저장될 때까지 1~2분 정도 기다려야 합니다. 이후의 모든 시작은 1초도 걸리지 않습니다.

새 Linux 배포를 위한 사용자 계정 및 암호를 만들어야 합니다.



```
Ubuntu
Installing, this may take a few minutes...
Installation successful!
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username:
```

축하합니다! Windows 운영 체제와 완전히 통합된 Linux 배포를 성공적으로 설치하고 설정했습니다.

## 설치 문제 해결

설치 과정에서 문제가 발생하는 경우 [문제 해결 가이드의 설치 섹션](#)을 참조하세요.

## 배포판 다운로드

Microsoft Store를 사용하여 WSL Linux 배포판을 설치할 수 없는(또는 설치하고 싶지 않은) 몇 가지 시나리오가 있습니다. 사용자가 Microsoft Store를 지원하지 않는 Windows Server 또는 LTSC(장기 서비스) 데스크톱 OS SKU를 실행하고 있거나, 회사 네트워크 정책 및/또는 관리자가 사용자 환경에서 Microsoft Store 사용을 허용하지 않는 경우입니다. 이러한 경우 WSL 자체는 제공되지만 Linux 배포판을 직접 다운로드해야 할 수도 있습니다.

Microsoft Store 앱을 사용할 수 없는 경우 다음 링크를 사용하여 Linux 배포판을 다운로드하고 수동으로 설치하면 됩니다.

- [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 ↗](#)

이렇게 하면 `<distro>.appx` 패키지가 선택한 폴더에 다운로드됩니다.

원한다면 명령줄을 통해 원하는 배포판을 다운로드할 수도 있습니다. PowerShell에서 `Invoke-WebRequest` cmdlet을 사용하면 됩니다. 예를 들어 Ubuntu 20.04를 다운로드하려면 다음을 수행합니다.

PowerShell

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

### 💡 팁

다운로드가 오래 걸릴 경우 `$ProgressPreference = 'SilentlyContinue'` 를 설정하여 진행률 표시줄을 비활성화하세요.

다운로드에 curl 명령줄 유틸리티<sup>2</sup>를 사용할 수도 있습니다. curl을 사용하여 Ubuntu 20.04를 다운로드하려면 다음을 수행합니다.

콘솔

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

이 예제에서는 PowerShell에서 `Invoke-WebRequest`의 PowerShell curl 별칭이 아닌 실제 curl 실행 파일이 호출되도록 `curl`이 아닌 `curl.exe`가 실행됩니다.

배포판이 다운로드되면 다운로드 파일이 들어 있는 폴더로 이동하여 해당 디렉터리에서 다음 명령을 실행합니다. 여기서 `app-name`은 Linux 배포판 .appx 파일의 이름입니다.

Powershell

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

Appx 패키지 다운로드가 완료되면 appx 파일을 두 번 클릭하여 새 배포 실행을 시작할 수 있습니다. (`wsl -l` 명령은 이 단계가 완료될 때까지 배포가 설치되었음을 표시하지 않습니다.)

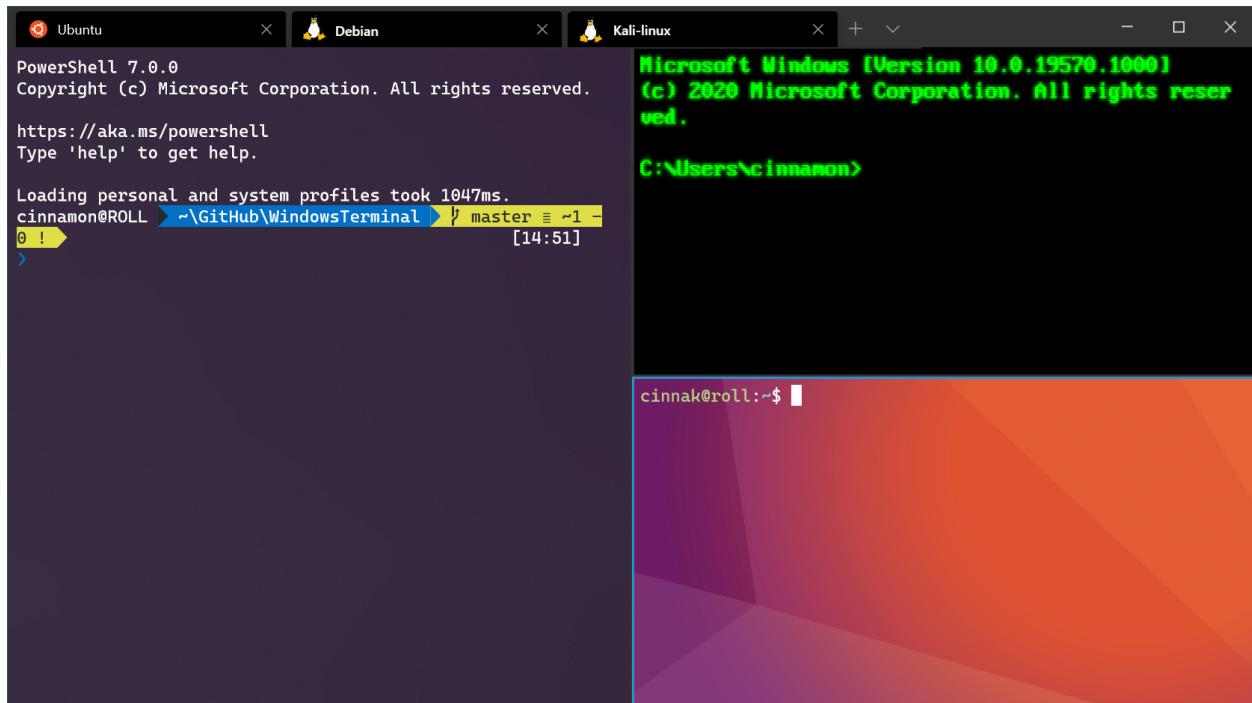
Windows Server를 사용 중이거나 위의 명령을 실행하는 동안 문제가 발생하면 [Windows Server](#) 설명서 페이지에서 대체 설치 지침을 찾아 .appx 파일을 zip 파일로 변경하여 설치할 수 있습니다.

배포판이 설치되면 지침에 따라 [새 Linux 배포판의 사용자 계정 및 암호를 만듭니다](#).

# Windows 터미널 설치(선택 사항)

Windows 터미널을 사용하면 여러 탭 또는 창을 열어 여러 Linux 배포판 또는 다른 명령줄 (PowerShell, 명령 프롬프트, Azure CLI 등)을 표시하고 빠르게 전환할 수 있습니다. 고유한 색 구성표, 글꼴 스타일, 크기, 배경 이미지 및 사용자 지정 바로 가기 키를 사용하여 터미널을 원하는 대로 사용자 지정할 수 있습니다. [자세한 정보](#)

Windows 터미널을 설치합니다.



# Windows Server 설치 가이드

아티클 • 2023. 03. 21. • 읽는 데 4분 걸림

WSL(Linux용 Windows 하위 시스템)은 Windows Server 2019(버전 1709) 이상에 설치할 수 있습니다. 이 가이드에서는 컴퓨터에서 WSL을 사용하는 단계를 안내합니다.

## Windows Server 2022에 WSL 설치

[Windows Server 2022](#)는 이제 다음 명령을 사용하여 간단한 WSL 설치를 지원합니다.

Bash

```
wsl --install
```

이제 관리자 PowerShell 또는 Windows 명령 프롬프트에서 이 명령을 입력한 다음, 컴퓨터를 다시 시작하여 Windows Server 2022에서 WSL을 실행하는 데 필요한 모든 항목을 설치할 수 있습니다.

이 명령은 필요한 선택적 구성 요소를 사용하도록 설정하고, 최신 Linux 커널을 다운로드하고, WSL 2를 기본값으로 설정하고, Linux 배포(기본적으로 Ubuntu)를 설치합니다.

다음 방법에 대한 자세한 내용은 표준 WSL 문서를 참조하세요.

- 설치된 기본 Linux 배포판 변경
- Linux 사용자 이름 및 암호 설정
- 실행 중인 WSL 버전 확인
- 패키지 업데이트 및 업그레이드
- 추가 배포 추가
- WSL에서 Git 사용

## 이전 버전의 Windows Server에 WSL 설치

WSL을 Windows Server 2019(버전 1709 이상)에 설치하려면 아래 수동 설치 단계를 수행하면 됩니다.

## Linux용 Windows 하위 시스템 사용

Windows에서 Linux 배포판을 실행하려면 먼저 "Linux용 Windows 하위 시스템" 옵션 기능을 사용하도록 설정하고 다시 부팅해야 합니다.

PowerShell을 관리자 권한으로 열어 실행합니다.

PowerShell

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

## Linux 배포 다운로드

기본 설정 Linux 배포판을 다운로드하기 위한 지침과 링크는 수동 설치 페이지의 [배포판 다운로드](#) 섹션을 참조하세요.

## Linux 배포 추출 및 설치

Linux 배포를 다운로드했으므로 콘텐츠를 추출하고 수동으로 설치하기 위해 다음 단계를 수행합니다.

1. PowerShell을 사용하여 `<DistributionName>.appx` 패키지의 콘텐츠를 추출합니다.

PowerShell

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

2. 배포판이 다운로드되면 다운로드 파일이 들어 있는 폴더로 이동하여 해당 디렉터리에서 다음 명령을 실행합니다. 여기서 `app-name`은 Linux 배포판 .appx 파일의 이름입니다.

Powershell

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

### ⊗ 주의

**0x8007007e 오류로 인한 설치 실패:** 이 오류가 발생하는 경우 시스템에서 WSL을 지원하지 않습니다. Windows 빌드 16215 이상을 실행하고 있는 확인합니다. **빌드를 확인하세요. 또한 WSL를 사용하는지** 그리고 기능이 활성화된 후 컴퓨터를 다시 시작했는지 확인합니다.

3. PowerShell을 사용하여 Windows 환경 경로(이 예에서는 `C:\Users\Administrator\Ubuntu`)에 Linux 배포판 경로를 추가합니다.

### PowerShell

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

이제 `<DistributionName>.exe`를 입력하여 모든 경로에서 배포를 시작할 수 있습니다. 예를 들면 `ubuntu.exe`와 같습니다.

설치가 완료되면 새 Linux 배포판의 사용자 계정 및 암호를 만들 수 있습니다.

# WSL 개발 환경 설정

아티클 • 2023. 03. 21. • 읽는 데 17분 걸림

WSL 개발 환경 설정을 위한 모범 사례에 대한 단계별 가이드입니다. Ubuntu를 사용하거나 다른 Linux 배포를 설치하도록 설정할 수 있는 기본 Bash 셸 설치 명령 실행, 기본 WSL 명령 사용, Visual Studio Code 또는 Visual Studio, Git, Windows 자격 증명 관리자와 MongoDB, Postgres 또는 MySQL과 같은 데이터베이스 설정, GPU 가속 설정, GUI 앱 실행 방법을 알아봅니다.

## 시작하기

Linux용 Windows 하위 시스템은 Windows 운영 체제와 함께 제공되지만 사용을 시작하려면 이를 사용하도록 설정하고 Linux 배포판을 설치해야 합니다.

단순화된 --install 명령을 사용하려면 최신 Windows 빌드(빌드 20262+)를 실행해야 합니다. 버전 및 빌드 번호를 확인하려면 **Windows 로고 키 + R**을 선택하고, **winver**를 입력하고, **확인**을 선택합니다. [설정 메뉴](#) 또는 [Windows 업데이트 도우미 ↗](#)를 사용하여 업데이트할 수 있습니다.

Ubuntu 이외의 Linux 배포판을 설치하거나 이러한 단계를 수동으로 완료하려는 경우 자세한 내용은 [WSL 설치 페이지](#)를 참조하세요.

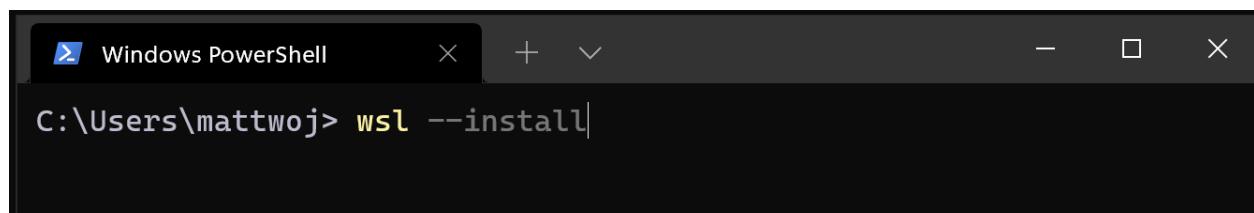
PowerShell(또는 Windows 명령 프롬프트)을 열고 다음을 입력합니다.

```
PowerShell
wsl --install
```

--install 명령은 다음 작업을 수행됩니다.

- 선택 사항인 WSL 및 Virtual Machine 플랫폼 구성 요소를 사용하도록 설정합니다.
- 최신 Linux 커널을 다운로드하여 설치합니다.
- WSL 2를 기본값으로 설정합니다.
- Ubuntu Linux 배포판 다운로드 및 설치(다시 부팅이 필요할 수 있음)

이 설치 프로세스 중에 컴퓨터를 다시 시작해야 합니다.



문제가 발생하면 [설치 문제 해결](#) 문서를 확인합니다.

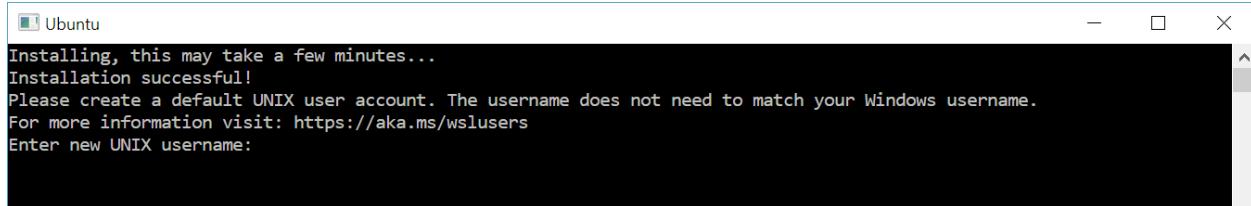
## Linux 사용자 이름 및 암호 설정

WSL을 사용하여 Linux 배포판을 설치하는 프로세스가 완료되면 시작 메뉴를 사용하여 배포판(기본적으로 Ubuntu)을 엽니다. Linux 배포판의 **사용자 이름과 암호**를 만들라는 메시지가 표시됩니다.

- 이 **사용자 이름 및 암호**는 설치하는 각각의 개별 Linux 배포에만 적용되며, Windows 사용자 이름과는 관련이 없습니다.
- **암호**를 입력하는 동안에는 화면에 아무것도 나타나지 않습니다. 이를 블라인드 타이핑이라고 합니다. 입력하는 내용을 볼 수 없습니다. 이는 완전히 정상입니다.
- **사용자 이름 및 암호**를 만들면 해당 계정이 배포의 기본 사용자가 되고 시작 시 자동으로 로그인됩니다.
- 이 계정은 `sudo`(슈퍼 사용자 작업) 관리 명령을 실행할 수 있는 Linux 관리자로 간주됩니다.
- WSL에서 실행되는 각 Linux 배포에는 고유한 Linux 사용자 계정과 암호가 있습니다. 배포를 추가하거나, 다시 설치하거나, 다시 설정할 때마다 Linux 사용자 계정을 구성해야 합니다.

### ① 참고

WSL과 함께 설치된 Linux 배포는 사용자별 설치이며 다른 Windows 사용자 계정과 공유할 수 없습니다. 사용자 이름 오류가 발생했나요? [StackExchange: Linux에서 사용자 이름에 어떤 문자를 사용해야 하나요?](#)



```
Ubuntu
Installing, this may take a few minutes...
Installation successful!
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username:
```

암호를 변경하거나 초기화하려면 Linux 배포판을 열고 `passwd` 명령을 입력합니다. 현재 암호를 입력하고 새 암호를 입력하라는 메시지가 표시된 다음, 새 암호를 확인하라는 메시지가 표시됩니다.

Linux 배포용 암호를 잊은 경우 다음을 수행합니다.

1. PowerShell을 열고, `wsl -u root` 명령을 사용하여 기본 WSL 배포의 루트를 입력합니다.

기본값이 아닌 배포에서 잊어버린 암호를 업데이트해야 하는 경우 `Debian`을 대상 배포의 이름으로 바꾼 `wsl -d Debian -u root` 명령을 사용합니다.

2. WSL 배포가 PowerShell 내의 루트 수준에서 열리면 `passwd <username>` 명령을 사용하여 암호를 업데이트할 수 있습니다. 여기서 `<username>`은 암호를 잊어버린 배포 계정 사용자 이름입니다.

3. 새 UNIX 암호를 입력한 다음, 해당 암호를 확인하라는 메시지가 표시됩니다. 암호가 성공적으로 업데이트되었다는 메시지가 표시되면 `exit` 명령을 사용하여 PowerShell 내에서 WSL을 닫습니다.

## 패키지 업데이트 및 업그레이드

배포의 기본 설정 패키지 관리자를 사용하여 패키지를 정기적으로 업데이트하고 업그레이드하는 것이 좋습니다. Ubuntu 또는 Debian의 경우 다음 명령을 사용합니다.

Bash

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

Windows는 Linux 배포를 자동으로 업데이트하거나 업그레이드하지 않습니다. 이는 대부분의 Linux 사용자가 직접 제어하는 것을 선호하는 작업입니다.

## 추가 배포 추가

추가 Linux 배포판을 추가하려면 [Microsoft Store](#) 나 `--import` 명령을 사용하거나 사용자 지정 배포판을 테스트용으로 로드하여 설치할 수 있습니다. [엔터프라이즈 회사 전체에 배포할 사용자 지정 WSL 이미지를 설정](#)할 수도 있습니다.

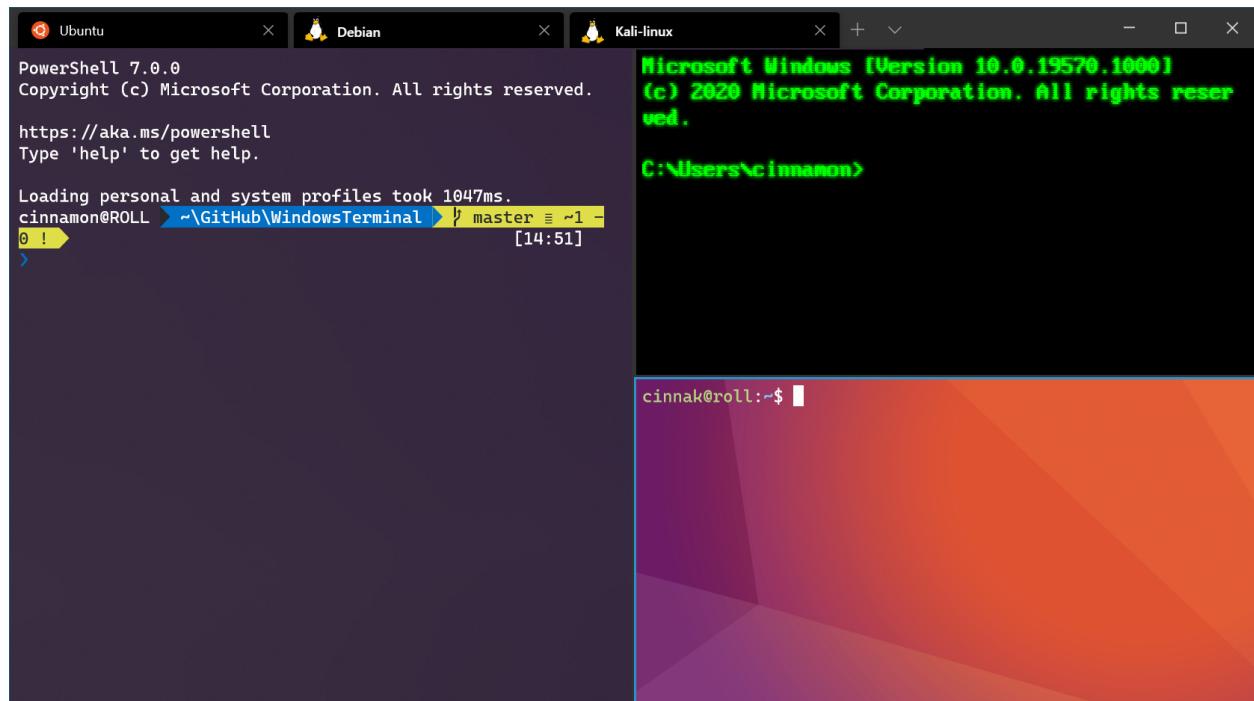
## Windows Terminal 설정

Windows Terminal은 명령줄 인터페이스로 모든 애플리케이션을 실행할 수 있습니다. 주요 기능에는 여러 탭, 창, 유니코드 및 UTF-8 문자 지원, GPU 가속 텍스트 렌더링 엔진, 사용자 고유의 테마를 만들고 텍스트, 색, 배경 및 바로 가기를 사용자 지정하는 기능이 있습니다.

새 WSL Linux 배포가 설치될 때마다 기본 설정에 맞게 사용자 지정할 수 있는 Windows Terminal 내부에 새 인스턴스가 만들어집니다.

특히 여러 명령줄로 작업하려는 경우 Windows Terminal에서 WSL을 사용하는 것이 좋습니다. 설정 및 기본 설정 사용자 지정에 대한 도움말은 Windows Terminal 문서를 참조하세요. 여기에는 다음이 포함됩니다.

- Microsoft Store에서 [Windows Terminal 또는 Windows Terminal\(미리 보기\)](#) 설치
- [명령 팔레트 사용](#)
- 바로 가기 키와 같은 [사용자 지정 작업](#)을 설정하여 터미널이 사용자의 기호에 맞게 자연스럽게 느껴지도록 합니다.
- [기본 시작 프로필 설정](#)
- 모양 사용자 지정: [테마](#), [색 구성표](#), [이름 및 시작 디렉터리](#), [배경 이미지](#) 등
- 창이나 탭으로 분할된 여러 명령줄이 있는 터미널 열기와 같은 [명령줄 인수](#) 사용 방법에 대해 알아봅니다.
- [검색 기능](#)에 대해 자세히 알아보기
- 탭의 이름을 바꾸거나 색상을 지정하는 방법, 마우스 상호 작용을 사용하는 방법 또는 'Quake 모드'를 사용하는 방법과 같은 [도움말과 요령](#)을 찾아보세요.
- [사용자 지정 명령 프롬프트](#), [SSH 프로필](#) 또는 [탭 제목](#)을 설정하는 방법에 대한 자습서 찾기
- [사용자 지정 터미널 갤러리](#) 및 [문제 해결 가이드](#) 찾기



## File Storage

- Windows 파일 탐색기에서 WSL 프로젝트를 열려면 다음을 입력합니다.  
`explorer.exe .`

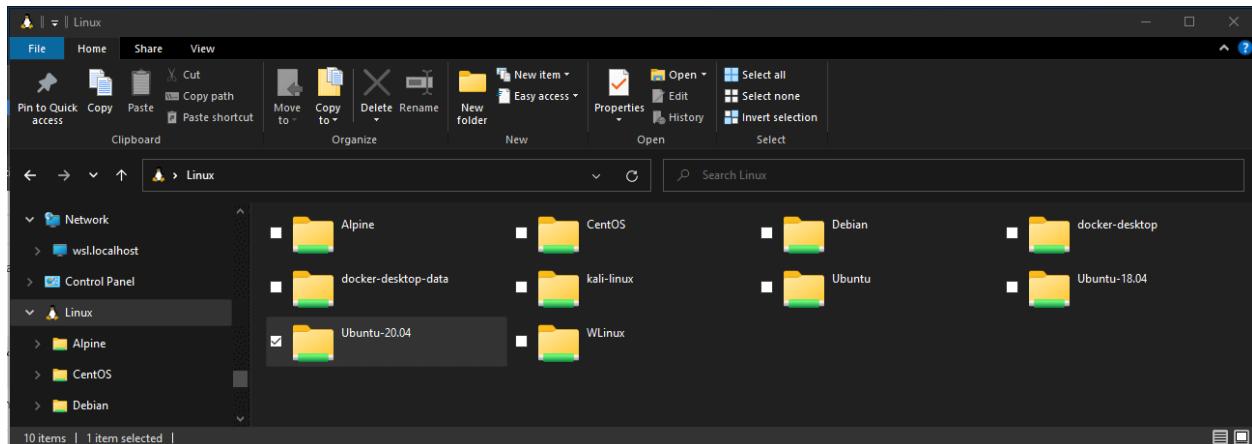
현재 디렉터리를 열려면 명령 끝에 마침표를 추가해야 합니다.

- 사용하려는 도구와 동일한 운영 체제에 프로젝트 파일을 저장합니다.

가장 빠른 성능 속도를 위해 Linux 명령줄(Ubuntu, OpenSUSE 등)에서 Linux 도구로 작업하는 경우 WSL 파일 시스템에 파일을 저장합니다. Windows 도구를 사용하여 Windows 명령줄(PowerShell, 명령 프롬프트)에서 작업하는 경우 파일을 Windows 파일 시스템에 저장합니다. 여러 운영 체제에서 파일에 액세스할 수 있지만 성능이 크게 저하될 수 있습니다.

예를 들어 WSL 프로젝트 파일을 저장하는 경우 다음과 같습니다.

- Linux 파일 시스템 루트 디렉터리(`\wsl$\<DistroName>\home\<UserName>\Project`)를 사용합니다.
- Windows 파일 시스템 루트 디렉터리를 사용하지 않는 경우: `C:\Users\<UserName>\Project` 또는 `/mnt/c/Users/<UserName>/Project$`



## 좋아하는 코드 편집기 설정

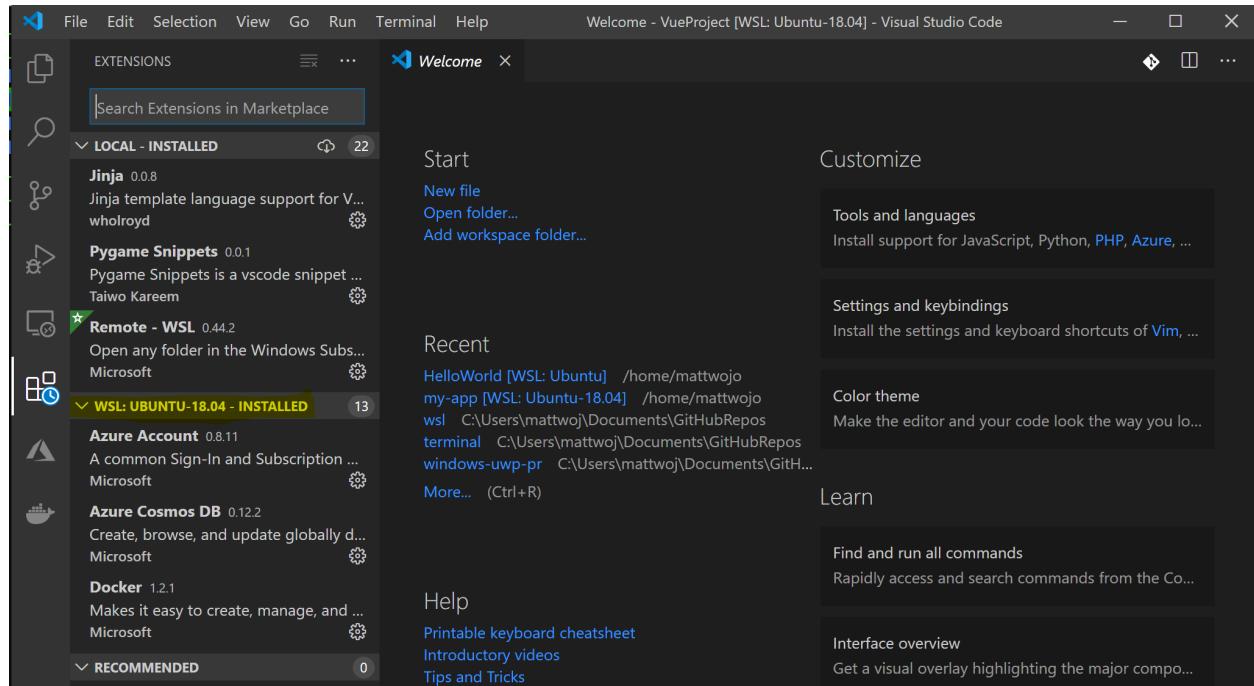
WSL을 사용한 원격 개발 및 디버깅을 직접 지원하므로 Visual Studio Code 또는 Visual Studio를 사용하는 것이 좋습니다. Visual Studio Code를 사용하면 WSL을 모든 기능을 갖춘 개발 환경으로 사용할 수 있습니다. Visual Studio는 C++ 플랫폼 간 개발을 위한 네이티브 WSL 지원을 제공합니다.

## Visual Studio Code 사용

원격 개발 확장 팩 설치가 포함된 WSL과 함께 Visual Studio Code를 사용하여 시작하려면 이 단계별 가이드를 따릅니다. 이 확장을 사용하면 전체 Visual Studio Code 기능 집합으로 편집 및 디버깅을 위해 WSL, SSH 또는 개발 컨테이너를 실행할 수 있습니다. 서로 다른 별도의 개발 환경 간에 신속하게 전환하고 로컬 컴퓨터에 영향을 미칠 염려 없이 업데이트합니다.

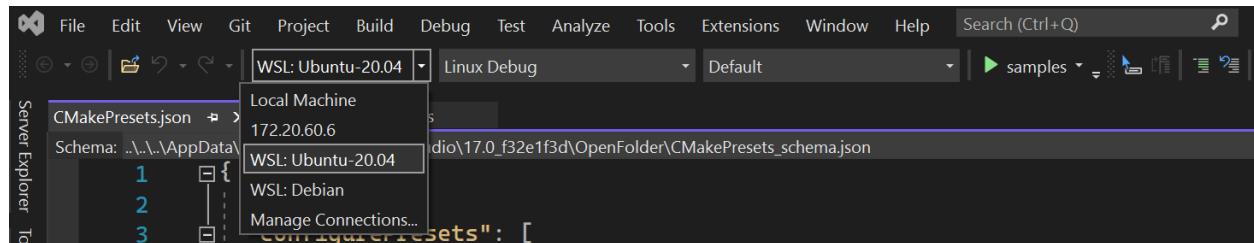
VS Code가 설치되고 설정되면 다음을 입력하여 VS Code 원격 서버로 WSL 프로젝트를 열 수 있습니다. `code .`

현재 디렉터리를 열려면 명령 끝에 마침표를 추가해야 합니다.



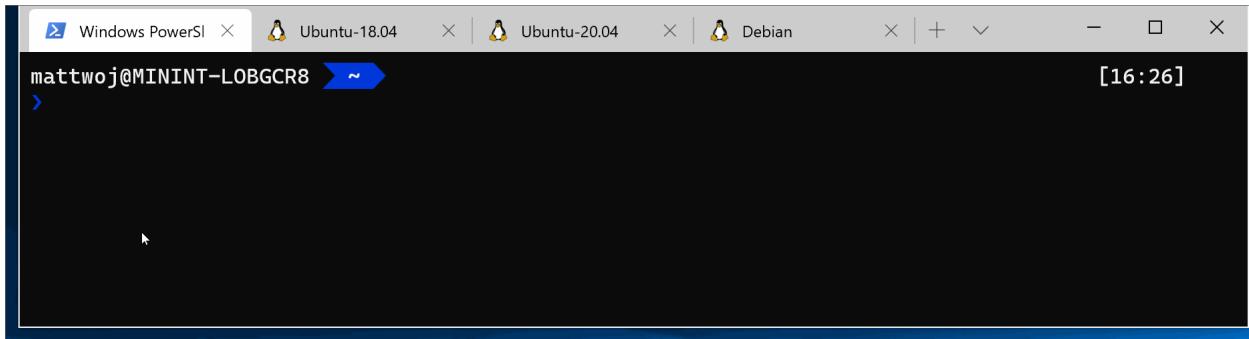
## Visual Studio 사용

C++ 플랫폼 간 개발을 위해 WSL과 함께 Visual Studio를 사용하여 시작하려면 이 단계별 가이드를 따릅니다. Visual Studio 2022를 사용하면 동일한 Visual Studio 인스턴스에서 Windows, WSL 배포 및 SSH 연결의 CMake 프로젝트를 빌드하고 디버그할 수 있습니다.



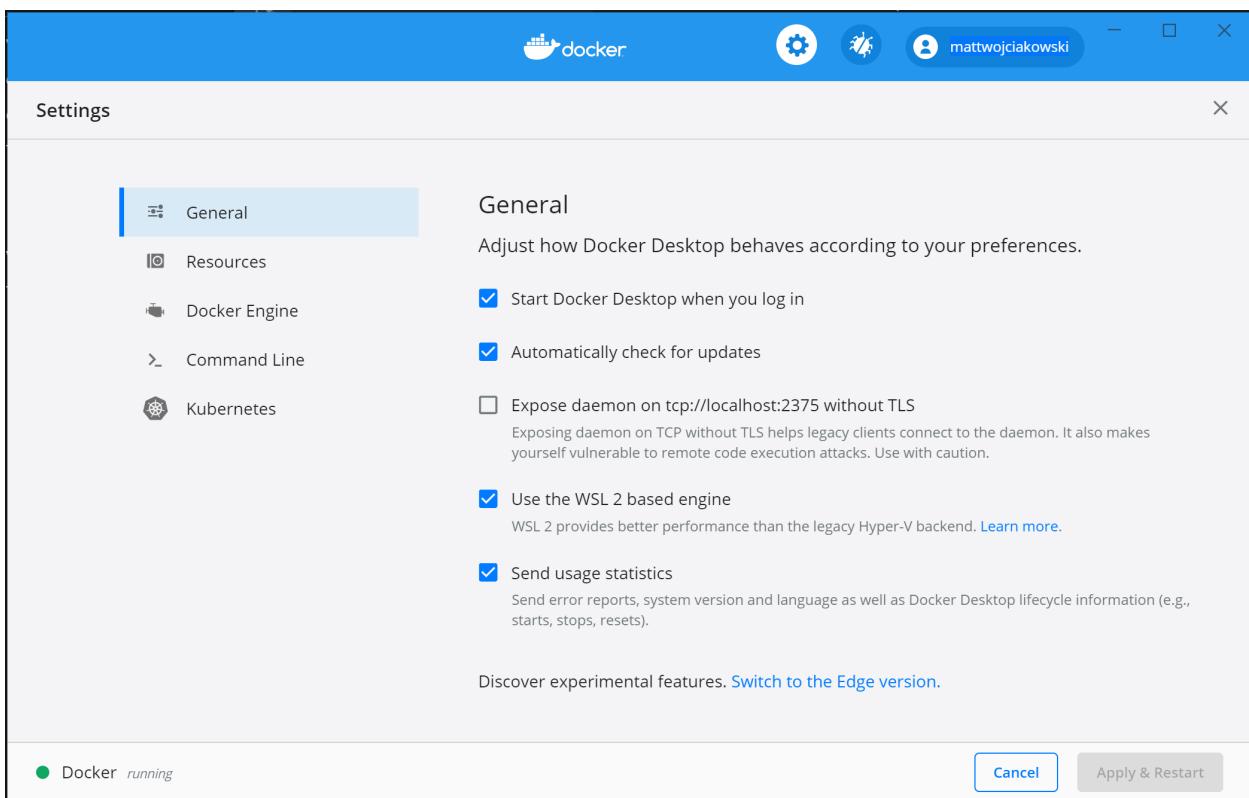
## Git으로 버전 관리 설정

이 단계별 가이드에 따라 WSL에서 Git 사용을 시작하고, 인증을 위해 자격 증명 관리자를 사용하고, Git 무시 파일을 사용하고, Git 줄 끝을 이해하고, VS Code에 기본 제공된 Git 명령을 사용하여 프로젝트를 Git 버전 제어 시스템에 연결합니다.



## Docker로 원격 개발 컨테이너 설정

이 단계별 가이드에 따라 [WSL 2에서 Docker 원격 컨테이너를 시작](#)하고 Windows용 Docker Desktop을 사용하여 프로젝트를 원격 개발 컨테이너에 연결합니다.



## 데이터베이스 설정

이 단계별 가이드에 따라 [WSL에서 데이터베이스 시작](#) 및 프로젝트를 WSL 환경의 데이터베이스에 연결합니다. MySQL, PostgreSQL, MongoDB, Redis, Microsoft SQL Server 또는 SQLite로 시작합니다.

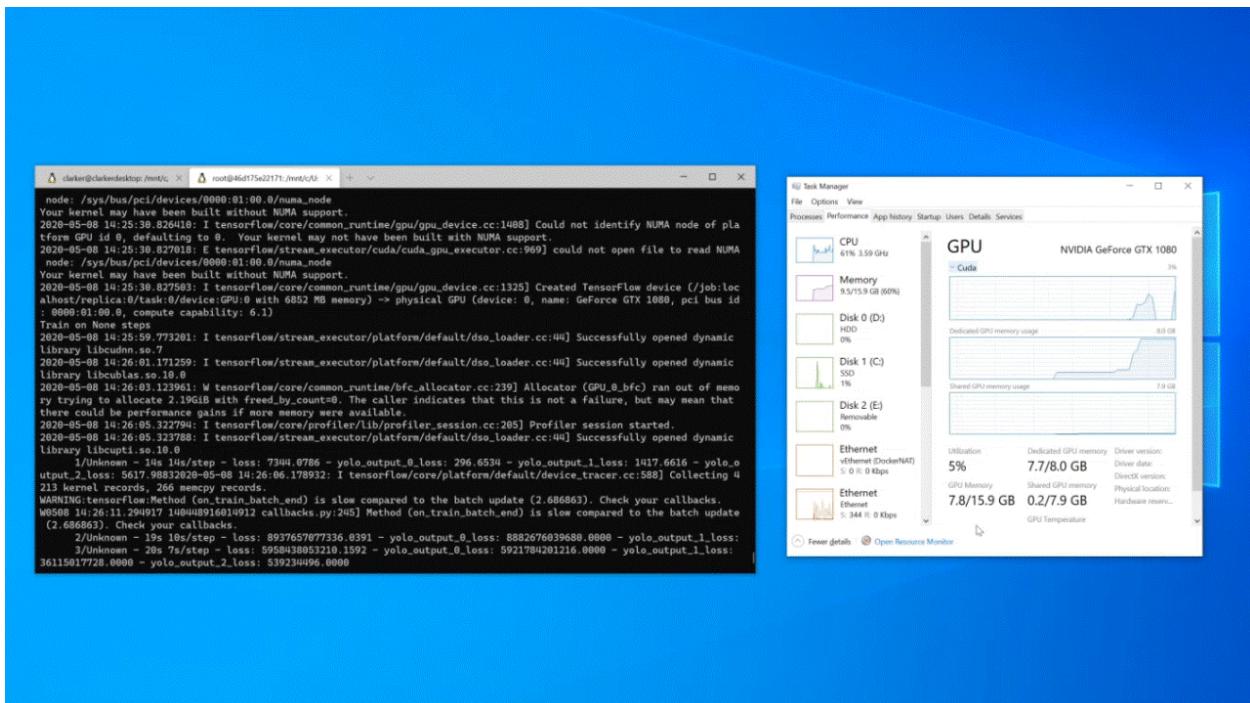
```

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:~$ 

```

## 더 빠른 성능을 위해 GPU 가속 설정

이 단계별 가이드에 따라 [WSL에서 GPU 가속 기계 학습 교육](#)을 설정하고 컴퓨터의 GPU(그래픽 처리 장치)를 활용하여 성능이 높은 워크로드를 가속화합니다.



## 기본 WSL 명령

WSL을 통해 설치하는 Linux 배포판은 PowerShell 또는 Windows 명령 프롬프트(CMD)를 사용하여 가장 잘 관리됩니다. WSL을 사용할 때 익숙해져야 할 기본 명령 목록은 [WSL 명령 참조 가이드](#)를 참조하세요.

또한 많은 명령이 Windows와 Linux 간에 상호 운용 가능합니다. 다음은 몇 가지 예입니다.

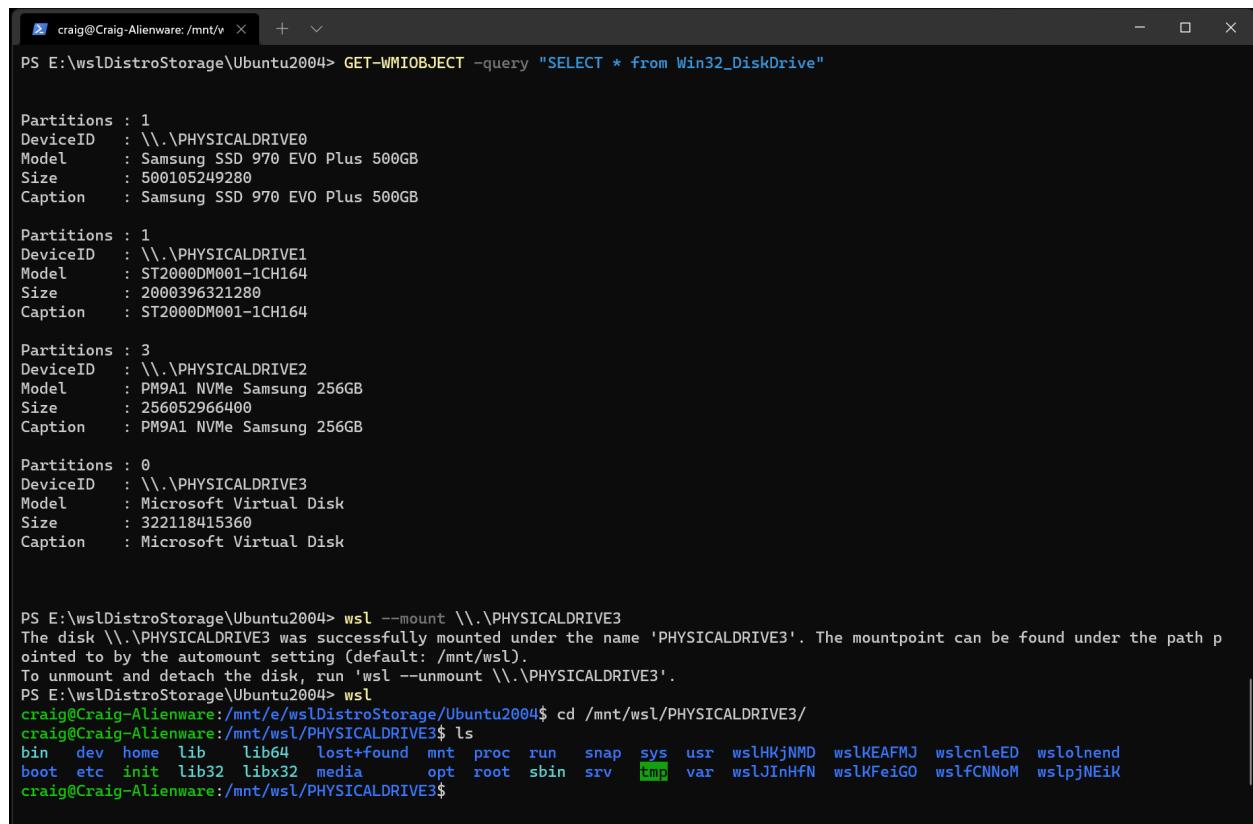
- **Windows 명령줄에서 Linux 도구 실행:** PowerShell을 열고 다음을 입력하여 Linux `ls -la` 명령을 사용하여 `c:\temp>`의 디렉터리 콘텐츠를 표시합니다. `wsl ls -la`
- **Linux 및 Windows 명령 혼합:** 이 예에서는 Linux 명령 `ls -la`를 사용하여 디렉터리의 파일을 나열한 다음 PowerShell 명령 `findstr`을 사용하여 "git"이 포함된 단어에

대한 결과를 필터링합니다. `wsl ls -la | findstr "git"`. 이는 Windows `dir` 명령과 Linux `grep` 명령을 혼합하여 수행할 수도 있습니다. `dir | wsl grep git`.

- **WSL 명령줄에서 직접 Windows 도구 실행:** <tool-name>.exe 예를 들어, .bashrc 파일(Linux 명령줄이 시작될 때마다 실행되는 셸 스크립트)을 열려면 다음을 입력합니다. `notepad.exe .bashrc`
- **Linux Grep 도구로 Windows ipconfig.exe 도구 실행:** `ipconfig.exe | grep IPv4 | cut -d: -f2` 이 예는 현재 TCP/IP 네트워크 구성 값을 표시하는 데 사용된 다음 Linux 도구인 grep을 사용하여 IPv4 결과로만 필터링되는 Windows 파일 시스템의 ipconfig 도구를 보여 줍니다.

## 외장형 드라이브 또는 USB 탑재

WSL 2에서 Linux 디스크 탑재 시작에 대한 단계별 가이드를 따릅니다.



The screenshot shows a Windows Command Prompt window titled 'craig@Craig-Alienware: /mnt/v'. It displays the output of several WSL commands related to disk management:

```
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$
```

## Linux GUI 앱 실행

이 자습서에 따라 [WSL에서 Linux GUI 앱을 설정하고 실행하는 방법](#)을 알아봅니다.

## 추가 자료

- [Windows에서 개발 환경 설정](#): React, Python, NodeJS, Vue 등과 같이 기본 설정 언어 또는 프레임워크에 맞게 개발 환경을 설정하는 방법에 대해 자세히 알아봅니다.
- [문제 해결](#): 일반적인 문제, 버그 보고 위치, 새 기능 요청 위치 및 문서에 기여하는 방법을 찾습니다.
- [FAQ](#): 질문과 대답 목록을 찾습니다.
- [릴리스 정보](#): WSL 릴리스 정보에서 지난 빌드 업데이트 기록을 검토합니다. [WSL Linux 커널의 릴리스 정보](#)도 찾을 수 있습니다.

# Linux용 Windows 하위 시스템에서 Visual Studio Code 사용 시작

아티클 • 2023. 03. 21. • 읽는 데 9분 걸림

WSL 확장과 함께 Visual Studio Code를 사용하면 VS Code에서 직접 WSL을 정규 개발 환경으로 사용할 수 있습니다. 다음을 수행할 수 있습니다.

- Linux 기반 환경에서 개발
- Linux 관련 도구 체인 및 유틸리티 사용
- Outlook 및 Office와 같은 생산성 도구에 대한 액세스를 유지하면서 Windows의 편안함에서 Linux 기반 애플리케이션을 실행 및 디버그합니다.
- VS Code 기본 제공 터미널을 사용하여 선택한 Linux 배포판 실행
- [Intellisense](#) [코드 완료](#), [linting](#), [디버그 지원](#), [코드 조각](#) 및 [단위 테스트](#)와 같은 VS Code 기능 활용
- VS Code의 기본 제공 [Git 지원](#)으로 버전 제어를 쉽게 관리
- WSL 프로젝트에서 직접 명령 및 VS Code 확장 실행
- 경로 지정 문제, 이진 파일 호환성 또는 기타 교차 OS 문제에 대한 걱정 없이 Linux 또는 탑재된 Windows 파일 시스템(예: /mnt/c)에서 파일 편집

## VS Code 및 WSL 확장 설치

- [VS Code 설치 페이지](#)를 방문하여 32비트 또는 64비트 설치 프로그램을 선택합니다. Windows에 Visual Studio Code를 설치합니다(WSL 파일 시스템이 아님).
- 설치 중에 **추가 작업 선택** 메시지가 표시되면 code 명령을 사용하여 WSL에서 폴더를 쉽게 열 수 있도록 PATH에 **추가** 옵션을 선택해야 합니다.
- [원격 개발 확장 팩](#)을 설치합니다. 이 확장 팩에는 원격 - SSH 및 Dev Containers 확장 외에도 WSL 확장이 포함되어 있어 컨테이너, 원격 컴퓨터 또는 WSL에서 모든 폴더를 열 수 있습니다.

### ① 중요

WSL 확장을 설치하려면 VS Code의 [1.35 May 릴리스](#) 버전 이상이 필요합니다. 자동 완성, 디버깅, linting 등에 대한 지원을 잊지 않도록 WSL 확장 없이 VS Code에서 WSL을 사용하지 않는 것이 좋습니다. 재미있게도 이 WSL 확장은 \$HOME/.vscode/extensions에 설치됩니다(PowerShell에서 `ls $HOME\.vscode\extensions\` 입력).

# Linux 배포 업데이트

일부 WSL Linux 배포에는 VS Code 서버를 시작하는 데 필요한 라이브러리가 부족합니다. 패키지 관리자를 사용하여 Linux 배포판에 라이브러리를 더 추가할 수 있습니다.

예를 들어, Debian 또는 Ubuntu를 업데이트하려면 다음을 사용합니다.

```
Bash
```

```
sudo apt-get update
```

wget(웹 서버에서 콘텐츠 검색) 및 ca-certificates(SSL 기반 애플리케이션이 SSL 연결의 신뢰성을 확인하도록 허용)를 추가하려면 다음을 입력합니다.

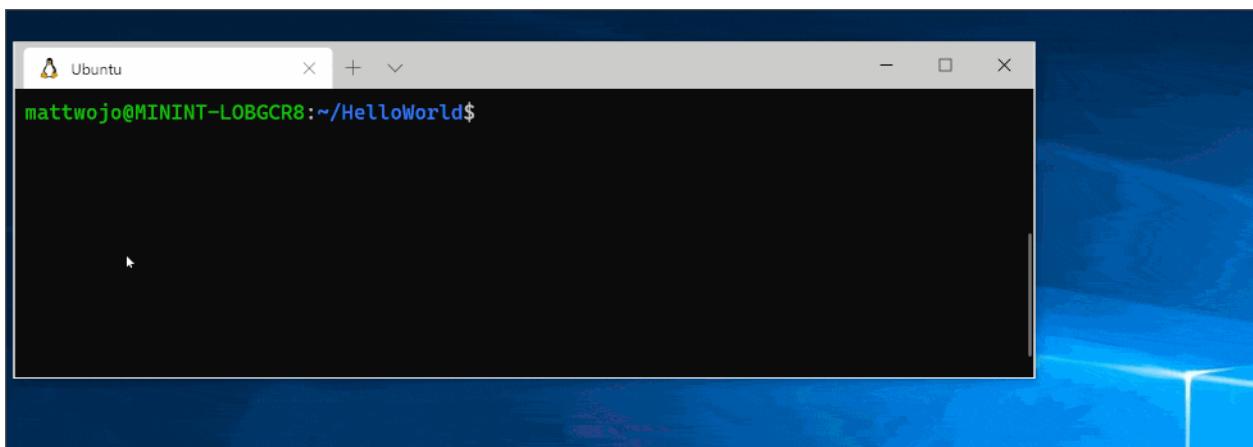
```
Bash
```

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

## Visual Studio Code에서 WSL 프로젝트 열기

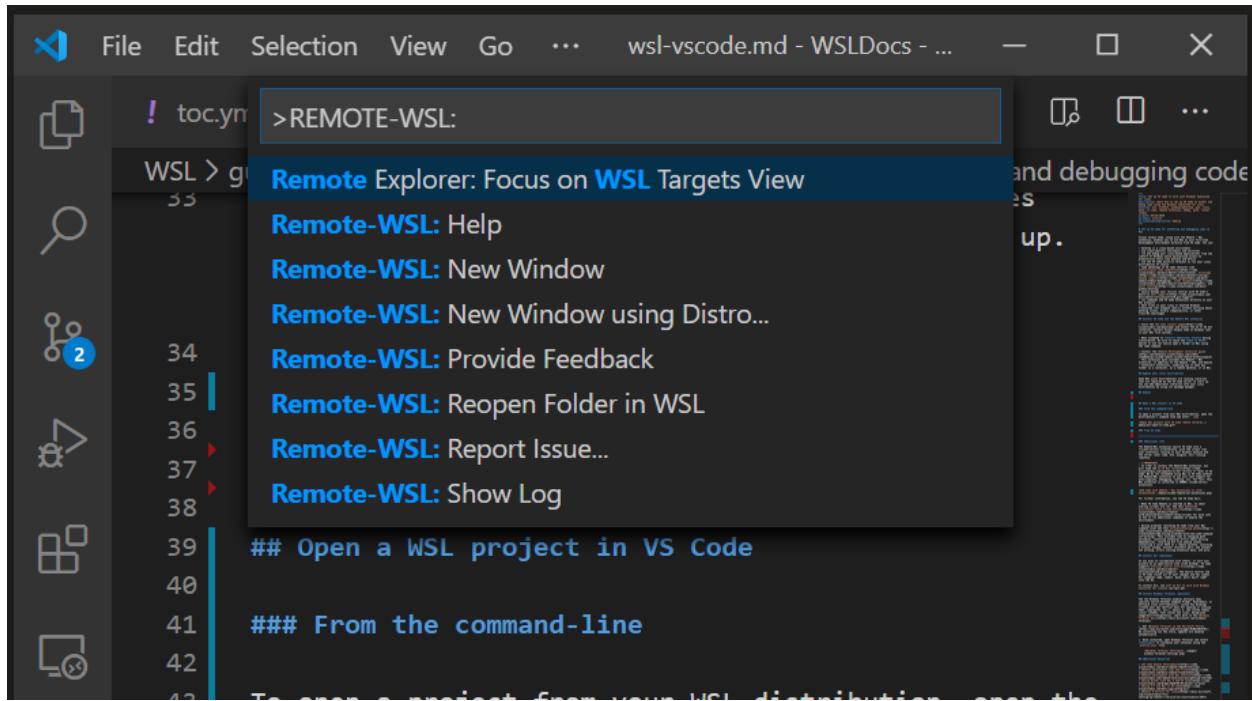
### 명령줄에서

WSL 배포에서 프로젝트를 열려면 배포의 명령줄을 열고 다음을 입력합니다. `code .`



### VS Code에서

VS Code에서 `CTRL+SHIFT+P` 바로 가기를 사용하여 더 많은 VS Code WSL 옵션에 액세스하여 명령 팔레트를 불러올 수도 있습니다. 그런 다음 `wsL`을 입력하면 사용 가능한 옵션 목록이 표시되어 WSL 세션에서 폴더를 다시 열고 열려는 배포를 지정하는 등의 작업을 수행할 수 있습니다.



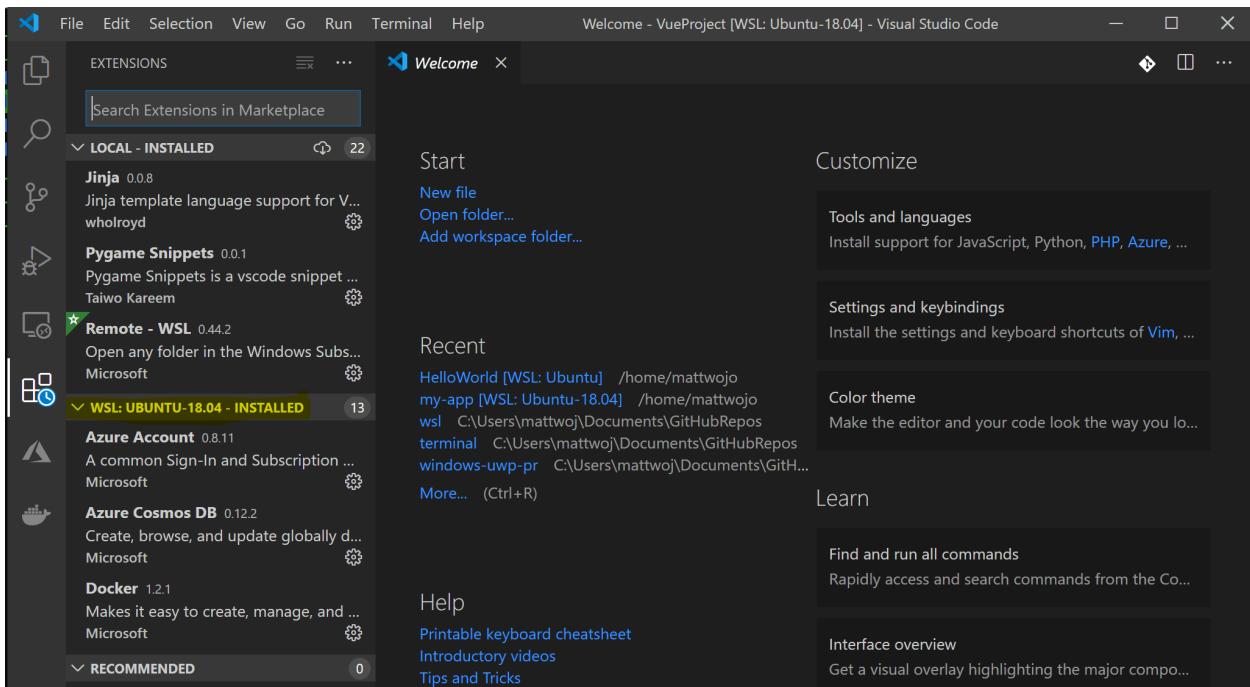
## VS Code WSL 내부의 확장

WSL 확장은 VS Code를 "클라이언트-서버" 아키텍처로 분할합니다. 클라이언트(사용자 인터페이스)는 Windows 컴퓨터에서 실행되고 서버(코드, Git, 플러그인 등)는 WSL 배포에서 "원격"으로 실행됩니다.

WSL 확장을 실행할 때 '확장' 탭을 선택하면 로컬 컴퓨터와 WSL 배포 간에 분할된 확장 목록이 표시됩니다.

[테마](#) 와 같은 로컬 확장은 한 번만 설치하면 됩니다.

[Python 확장](#) 과 같은 일부 확장 또는 Linting 또는 디버깅과 같은 작업을 처리하는 확장은 각 WSL 배포에 별도로 설치해야 합니다. VS Code는 WSL 배포판에 설치되지 않은 확장이 로컬에 설치된 경우 녹색 "WSL에 설치" 단추와 함께 경고 아이콘 을 표시합니다.



자세한 내용은 VS Code 문서를 참조하세요.

- VS Code가 WSL에서 시작되면 셀 시작 스크립트가 실행되지 않습니다. 추가 명령을 실행하거나 환경을 수정하는 방법에 대한 자세한 내용은 이 [고급 환경 설정 스크립트 문서](#)를 참조하세요.
- WSL 명령줄에서 VS Code를 시작하는 데 문제가 있나요? 이 [문제 해결 가이드](#)에는 경로 변수 변경, 누락된 종속성에 대한 확장 오류 해결, Git 줄 끝 문제 해결, 원격 컴퓨터에 로컬 VSIX 설치, 브라우저 창 시작, 블로커 localhost 포트, 웹 소켓이 아닌 것에 대한 팁이 포함되어 있습니다. 작업, 확장 데이터 저장 오류 등.

## Git 설치(선택 사항)

다른 사람과 협업할 계획이거나 GitHub 같은 오픈 소스 사이트에 프로젝트를 호스팅할 계획인 분들을 위해 VS Code는 [Git을 사용한 버전 제어](#)를 지원합니다. VS Code의 소스 제어 탭은 모든 변경 내용을 추적하며, UI에 바로 빌드된 일반적인 Git 명령(추가, 커밋, 푸시, 끌어오기)를 포함하고 있습니다.

Git을 설치하려면 [Linux용 Windows 하위 시스템과 작동하도록 Git 설정](#)을 참조하세요.

## Windows 터미널 설치(선택 사항)

새 Windows 터미널을 사용하면 여러 탭(명령 프롬프트, PowerShell 또는 여러 Linux 배포판 간에 신속하게 전환), 사용자 지정 키 바인딩(탭을 열거나 닫는 개발자 고유의 바로 가기 키 만들기, 복사+붙여넣기 등), 이모지 ☺ 및 사용자 지정 테마(색 구성표, 글꼴 스타일 및 크기, 배경 이미지/흐림/투명도)를 사용할 수 있습니다. [Windows 터미널 문서](#)에서 자세히 알아보세요.

1. 다음과 같이 Microsoft Store에서 Windows 터미널 [을 받습니다](#). Microsoft Store를 통해 설치하면 업데이트가 자동으로 처리됩니다.
2. 설치가 완료되면 Windows 터미널을 열고 설정을 선택한 다음, `profile.json` 파일을 사용하여 터미널을 사용자 지정합니다.

## 추가 리소스

- [VS Code WSL 설명서](#)
- [VS Code WSL 자습서](#)
- [원격 개발 도움말 및 요령](#)
- [WSL 2 및 VS Code와 함께 Docker 사용](#)
- [VS Code에서 C++ 및 WSL 사용](#)
- [Linux용 원격 R 서비스](#)

다음과 같은 몇 가지 추가 확장도 고려해 볼 수 있습니다.

- [다른 편집기의 키맵](#): Atom, Sublime, Vim, eMacs, 메모장++ 등의 다른 텍스트 편집기에서 전환할 때 이러한 확장을 사용하여 익숙한 환경을 만들 수 있습니다.
- [설정 동기화](#): GitHub를 사용하는 여러 설치에서 VS Code 설정을 동기화할 수 있습니다. 여러 머신에서 작업하는 경우 이렇게 하면 여러 머신의 환경을 일관되게 유지할 수 있습니다.
- [Chrome용 디버거](#): Linux로 서버 쪽 개발을 완료하면 클라이언트 쪽을 개발하고 테스트해야 합니다. 이 확장은 VS Code 편집기를 Chrome 브라우저 디버깅 서비스와 통합하여 효율성을 높입니다.

# Linux용 Windows 하위 시스템에서 Git 사용 시작

아티클 • 2023. 03. 21. • 읽는 데 11분 걸림

Git은 가장 일반적으로 사용되는 버전 제어 시스템입니다. Git을 사용하면 파일 변경 내용을 추적할 수 있으므로 수행된 작업을 기록하고 필요한 경우 파일의 이전 버전으로 되돌릴 수 있습니다. Git은 또한 협업을 더 쉽게 만들어 여러 사람이 변경한 내용을 모두 하나의 원본으로 병합할 수 있습니다.

## Git은 Windows 및 WSL에 설치할 수 있습니다.

중요한 고려 사항: WSL을 사용하도록 설정하고 Linux 배포판을 설치할 때 컴퓨터의 Windows NTFS C:\ 드라이브와 분리된 새 파일 컴퓨터를 설치하게 됩니다. Linux에서는 드라이브에 문자가 지정되지 않습니다. 탑재 지점이 제공됩니다. 파일 시스템 /의 루트는 WSL의 경우 루트 파티션 또는 폴더의 탑재 지점입니다. / 아래의 모든 항목이 동일한 드라이브는 아닙니다. 예를 들어, 내 랩톱에 Debian뿐만 아니라 두 가지 버전의 Ubuntu(20.04 및 18.04)를 설치했습니다. 해당 배포판을 열면 cd ~ 명령으로 홈 디렉터리를 선택한 다음 explorer.exe . 명령을 입력하면 Windows 파일 탐색기가 열리고 해당 배포의 디렉터리 경로가 표시됩니다.

Linux 배포판	홈 폴더에 액세스하기 위한 Windows 경로
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

### 💡 팁

WSL 배포 명령줄에서 Windows 파일 디렉터리에 액세스하려는 경우 C:\Users\username 대신 /mnt/c/Users/username을 사용하여 디렉터리에 액세스합니다. Linux 배포는 Windows 파일 시스템을 탑재된 드라이브로 보기 때문입니다.

사용하려는 각 파일 시스템에 Git을 설치해야 합니다.

## Git 설치

Git은 대부분의 Linux 배포용 Windows 하위 시스템과 함께 이미 설치되어 제공되지만 최신 버전으로 업데이트하는 것이 좋습니다. 또한 git 구성 파일을 설정해야 합니다.

Git을 설치하려면 [Linux용 Git 다운로드](#) 사이트를 참조하세요. 각 Linux 배포에는 자체 패키지 관리자와 설치 명령이 있습니다.

Ubuntu/Debian의 안정적인 최신 Git 버전을 보려면 다음 명령을 입력합니다.

Bash

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

### ① 참고

아직 설치하지 않은 경우 [Windows용 Git을 설치](#) 할 수도 있습니다.

## Git 구성 파일 설정

Git 구성 파일을 설정하려면 작업 중인 배포판의 명령줄을 열고 다음 명령으로 이름을 설정합니다("Your Name"을 원하는 사용자 이름으로 바꾸기).

Bash

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

다음 명령으로 이메일을 설정합니다("youremail@domain.com"을 원하는 이메일로 바꿉니다).

Bash

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

## 💡 팁

아직 GitHub 계정이 없다면 [GitHub에서 등록](#)할 수 있습니다. 이전에 Git를 사용한 경험이 없는 경우 [GitHub 가이드](#)를 보면 시작하는 데 도움이 될 수 있습니다. Git config를 편집해야 하는 경우 nano: `nano ~/.gitconfig`와 같은 기본 제공 텍스트 편집기를 사용하여 편집할 수 있습니다.

2FA(2단계 인증)를 사용하여 계정을 [보호](#)하는 것이 좋습니다.

## Git 자격 증명 관리자 설정

[GCM\(Git 자격 증명 관리자\)](#)은 WSL1과 WSL2 모두에서 사용할 수 있는 [.NET](#)에 빌드된 안전한 Git 자격 증명 도우미입니다. GitHub 리포지토리, [Azure DevOps](#), Azure DevOps Server 및 Bitbucket에 대한 다단계 인증 지원을 사용하도록 설정합니다.

GCM은 GitHub와 같은 서비스의 인증 흐름에 통합되며 호스팅 공급자에 인증되면 새 인증 토큰을 요청합니다. 그런 다음, [Windows 자격 증명 관리자](#)에 토큰을 안전하게 저장합니다. 처음 인증하고 나면 다시 인증할 필요 없이 Git을 사용하여 호스팅 공급자와 통신할 수 있습니다. Git이 Windows 자격 증명 관리자의 토큰에 액세스할 것입니다.

WSL과 함께 GCM을 사용하려면 Windows 10 버전 1903 이상이어야 합니다. 이는 GCM이 WSL 배포에서 Git과 상호 운용하는 데 사용하는 필수 `wsl.exe` 도구를 포함하는 Windows의 첫 번째 버전입니다.

WSL과 Windows 호스트 간에 자격 증명 & 설정을 공유하려면 [Windows용 최신 Git](#)을 설치하는 것이 좋습니다. Git 자격 증명 관리자는 Windows용 Git에 포함되어 있으며 최신 버전은 각각의 새로운 Windows용 Git 릴리스에 포함되어 있습니다. 설치 중에 GCM이 기본값으로 설정된 자격 증명 헬퍼를 선택하라는 메시지가 표시됩니다.

Windows용 Git을 설치하지 않아야 할 이유가 있는 경우 GCM을 WSL 배포판에서 Linux 애플리케이션으로 직접 설치할 수 있지만 이렇게 하면 GCM이 Linux 애플리케이션으로 실행 중이며 호스트 Windows 운영 체제의 인증 또는 자격 증명 스토리지 기능을 활용할 수 없습니다. [Windows용 Git 없이 WSL을 구성](#)하는 방법에 대한 지침은 GCM 리포지토리를 참조하세요.

WSL 배포와 함께 사용하기 위해 GCM을 설정하려면 배포를 열고 다음 명령을 입력합니다.

설치된 GIT가 = v2.39.0인 > 경우

Bash

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

설치된 GIT가 = v2.36.1이 >면 그렇지 않습니다.

Bash

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

버전이 < v2.36.1인 경우 다음 명령을 입력합니다.

Bash

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

### ① 참고

GCM을 WSL Git 설치에 대한 자격 증명 도우미로 사용하는 것은 WSL Git에 설정된 모든 구성이 GCM에서 준수되지 않음을 의미합니다(기본적으로). 이는 GCM이 Windows 애플리케이션으로 실행 중이기 때문에 Windows용 Git 설치를 사용하여 구성을 쿼리하기 때문입니다. 즉, GCM에 대한 프록시 설정과 같은 항목은 서로 다른 파일(%USERPROFILE%\.gitconfig 및 \\wsl\$\distro\home\\$USER\.gitconfig)에 저장되므로 Windows용 Git 및 WSL Git에서 설정해야 합니다. GCM이 WSL Git 구성을 사용하도록 WSL을 구성할 수 있지만 이는 프록시 설정이 특정 WSL 설치에 대해 고유하고 다른 사람 또는 Windows 호스트와 공유되지 않음을 의미합니다.

## SSH와 함께 Git 사용

Git 자격 증명 관리자는 HTTP(S) 원격에서만 작동합니다. SSH와 함께 Git을 계속 사용할 수 있습니다.

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

## Azure에 대한 추가 구성

[Azure Repos ↗](#) 또는 [Azure DevOps ↗](#)로 작업하려면 몇 가지 추가 구성이 필요합니다.

Bash

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

이제 WSL 배포 내에서 수행하는 모든 git 작업은 GCM을 사용합니다. 호스트용으로 캐시된 자격 증명이 이미 있는 경우 자격 증명 관리자에서 액세스됩니다. 그렇지 않다면 자격 증명을 요청하는 대화 상자 응답이 수신됩니다(Linux 콘솔에 있을 경우도).

### 💡 팁

코드 서명 보안을 위해 GPG 키를 사용하는 경우 [GPG 키를 GitHub 이메일과 연결](#) 해야 할 수 있습니다.

## Git Ignore 파일 추가

프로젝트에 [.gitignore 파일](#)을 추가하는 것이 좋습니다. GitHub는 사용 사례에 따라 구성된 권장 .gitignore 파일 설정과 함께 [유용한 .gitignore 템플릿 컬렉션](#)을 제공합니다. 예를 들어, [Node.js 프로젝트용 GitHub의 기본 gitignore 템플릿](#)은 다음과 같습니다.

[GitHub 웹 사이트](#)를 사용하여 새 리포지토리 만들기 를 선택하면 특정 프로젝트 형식으로 설정된 .gitignore 파일인 README 파일을 사용하여 리포지토리를 초기화하는 확인란과 필요에 따라 라이선스를 추가할 수 있는 옵션이 제공됩니다.

## Git 및 VS Code

Visual Studio Code에는 변경 내용을 표시하고 다양한 git 명령을 처리하는 원본 제어 탭을 포함하여 Git에 대한 기본 제공 지원이 제공됩니다. [VS Code의 Git 지원](#)에 대해 자세히 알아봅니다.

## Git 줄 끝

Windows, WSL 또는 컨테이너 간에 동일한 리포지토리 폴더로 작업하는 경우 일관된 줄 끝을 설정해야 합니다.

Windows와 Linux는 서로 다른 기본 줄 끝을 사용하므로 Git은 줄 끝 외에 차이가 없는 많은 수정된 파일을 보고할 수 있습니다. 이를 방지하기 위해 [.gitattributes](#) 파일을 사용하거나 Windows 측에서 전체적으로 줄 끝 변환을 사용하지 않도록 설정할 수 있습니다. [Git 줄 끝 문제 해결에 대한 VS Code 문서](#)를 참조하세요.

## 추가 자료

- [WSL & VS Code](#)
- [GitHub 학습 랩: 온라인 과정 ↗](#)
- [Git 시작화 도구 ↗](#)
- [Git 도구 - 자격 증명 스토리지 ↗](#)

# Linux용 Windows 하위 시스템에서 데이터베이스 시작하기

아티클 • 2023. 03. 21. • 읽는 데 21분 걸림

이 단계별 가이드는 WSL의 프로젝트에서 데이터베이스로 장치 연결을 시작하는 데 도움이 됩니다. MySQL, PostgreSQL, MongoDB, Redis, Microsoft SQL Server 또는 SQLite로 시작합니다.

## 사전 요구 사항

- Windows 10 실행, [버전 2004, 빌드 19041](#) 이상으로 업데이트
- [WSL을 설치하고 Linux 배포에 대한 사용자 이름과 비밀번호를 생성합니다.](#)
- [WSL 2 모드](#)에서 Linux 배포 실행

## 데이터베이스 시스템 간의 차이점

데이터베이스 시스템에 가장 많이 사용되는 선택 항목은 다음과 같습니다.

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

**MySQL**은 데이터 형식이 서로 관련될 수 있는 하나 이상의 테이블로 데이터를 구성하는 오픈 소스 SQL 관계형 데이터베이스입니다. 수직으로 확장이 가능하기 때문에 한 개의 컴퓨터로 작업을 수행할 수 있습니다. 현재 4개의 데이터베이스 시스템에서 가장 널리 사용됩니다.

**PostgreSQL**(Postgres라고도 함)은 확장성 및 표준 준수에 중점을 둔 오픈 소스 SQL 관계형 데이터베이스입니다. 이제 JSON도 처리할 수 있지만, 일반적으로 정형 데이터, 수평적 크기 조정 및 전자 상거래 및 금융 거래와 같은 ACID 호환 요구에 더 적합합니다.

**Microsoft SQL Server**에는 Windows의 SQL Server, Linux의 SQL Server, Azure의 SQL이 포함됩니다. 또한, 소프트웨어 애플리케이션에서 요청한 대로 데이터를 저장하고 검색하는 기본 기능을 사용하여 서버에 설정된 관계형 데이터베이스 관리 시스템이기도 합니다.

**SQLite**는 메모리가 낮은 환경에서도 이식성, 안정성 및 우수한 성능으로 유명한 오픈 소스 자체 포함 파일 기반 “서버리스” 데이터베이스입니다.

**MongoDB**는 JSON을 사용하고 스키마가 없는 데이터를 저장하도록 고안된 오픈 소스 NoSQL 문서 데이터베이스입니다. 수평으로 확장이 가능하기 때문에 여러 대의 작은 컴퓨터로 작업을 수행할 수 있습니다. 우수한 유연성으로 비정형 데이터 및 실시간 분석 캐싱에 적합합니다.

**Redis**는 오픈 소스 NoSQL 인메모리 데이터 구조 저장소입니다. 문서 대신 저장소에 키 값의 쌍을 사용합니다. Redis는 유연성, 성능 및 광범위한 언어 지원으로 유명합니다. 캐시 또는 메시지 브로커로 사용할 수 있을 만큼 유연하며 목록, 집합, 해시와 같은 데이터 구조를 사용할 수 있습니다.

선택한 데이터베이스의 종류는 데이터베이스와 함께 사용하는 애플리케이션의 유형에 따라 달라집니다. 정형 및 비정형 데이터베이스의 장점과 단점을 살펴보고 사용 사례에 따라 선택하는 것이 좋습니다.

## MySQL 설치

WSL에 MySQL(예: Ubuntu)을 설치하려면 다음을 수행합니다.

1. WSL 터미널(예: Ubuntu)을 엽니다.
2. Ubuntu 패키지를 업데이트합니다. `sudo apt update`
3. 패키지가 업데이트되면 `sudo apt install mysql-server`을(를) 사용하여 MySQL을 설치합니다.
4. 설치를 확인하고 버전 번호(`mysql --version`)를 가져옵니다.

포함된 보안 스크립트를 실행할 수도 있습니다. 이렇게 하면 원격 루트 로그인 및 샘플 사용자와 같은 항목에 대해 덜 안전한 기본 옵션 중 일부가 변경됩니다. 보안 스크립트를 실행하려면 다음을 수행합니다.

1. MySQL 서버를 시작합니다. `sudo /etc/init.d/mysql start`
2. 보안 스크립트 프롬프트를 시작합니다. `sudo mysql_secure_installation`
3. 첫 번째 프롬프트에서는 MySQL 비밀번호의 강도를 테스트하는 데 사용할 수 있는 비밀번호 유효성 검사 플러그인을 설정할 것인지 묻는 메시지가 표시됩니다. 그런 다음 MySQL 루트 사용자에 대한 비밀번호를 설정하고, 익명 사용자를 제거할지 여부를 결정하고, 루트 사용자가 로컬 및 원격으로 로그인할 수 있도록 허용할지 여부를 결정하고, 테스트 데이터베이스를 제거할지 여부를 결정한 후, 마지막으로 권한 테이블을 즉시 다시 로드할지 여부를 결정합니다.

MySQL 프롬프트를 열려면 `sudo mysql`을(를) 입력합니다.

사용 가능한 데이터베이스를 보려면 MySQL 프롬프트에서 `SHOW DATABASES;`을(를) 입력합니다.

새 데이터베이스를 만들려면 `CREATE DATABASE database_name;`을(를) 입력합니다.

데이터베이스를 삭제하려면 `DROP DATABASE database_name;` 을(를) 입력합니다.

MySQL 데이터베이스 작업에 대한 자세한 내용은 [MySQL 문서](#)를 참조하십시오.

VS Code에서 MySQL 데이터베이스를 사용하려면 [MySQL 확장](#)을 사용해 보십시오.

## PostgreSQL 설치

WSL(예: Ubuntu)에 PostgreSQL을 설치하는 방법

1. WSL 터미널(예: Ubuntu)을 엽니다.
2. Ubuntu 패키지를 업데이트합니다. `sudo apt update`
3. 패키지가 업데이트된 후에는 PostgreSQL(몇 가지 유용한 유ти리티가 포함된 - contrib 패키지)을 설치합니다. `sudo apt install postgresql postgresql-contrib`
4. 설치를 확인하고 버전 번호(`psql --version`)를 가져옵니다.

PostgreSQL이 설치되면 다음 세 가지 명령을 알고 있어야 합니다.

- `sudo service postgresql status`: 데이터베이스의 상태 확인
- `sudo service postgresql start` 데이터베이스 실행 시작
- `sudo service postgresql stop`: 데이터베이스 실행 중지

기본 관리자 사용자 `postgres`에는 데이터베이스에 연결하기 위해 할당된 암호가 필요합니다. 암호를 설정하려면 다음을 수행합니다.

1. `sudo passwd postgres` 명령을 입력합니다.
2. 새 암호를 입력하라는 메시지가 표시됩니다.
3. 터미널을 닫았다가 다시 엽니다.

[psql](#) 셸을 사용하여 PostgreSQL 실행

1. Postgres 서비스를 시작합니다. `sudo service postgresql start`
2. Postgres 서비스에 연결하고 psql 셸을 엽니다. `sudo -u postgres psql`

psql 셸을 성공적으로 입력하면 명령줄 변경 내용이 다음과 같이 표시됩니다. `postgres=#`

### ① 참고

또는 `su - postgres`를 사용하여 postgres 사용자로 전환한 다음, `psql` 명령을 입력하여 psql 셸을 열 수 있습니다.

`postgres=#`를 종료하려면 `\q` 을(를) 입력하거나 바로가기 키(Ctrl+D)를 사용합니다.

PostgreSQL 설치에 생성된 사용자 계정을 확인하려면 WSL 터미널에서 `psql -c "\du"`를 사용하거나 `psql` 셸이 열려 있는 경우 `\du`를 사용합니다. 이 명령은 계정 사용자 이름, 역할 특성 목록 및 역할 그룹의 구성원 열을 표시합니다. 명령줄로 돌아가려면 `q`를 입력합니다.

PostgreSQL 데이터베이스 작업에 대한 자세한 내용은 [PostgreSQL 문서](#)를 참조하십시오.

VS Code에서 PostgreSQL 데이터베이스를 사용하려면 [PostgreSQL 확장](#)을 사용해 보십시오.

## MongoDB 설치

WSL(Ubuntu 20.04)에 MongoDB(버전 5.0)를 설치하려면 다음을 수행합니다.

1. WSL 터미널(예: Ubuntu)을 열고 홈 디렉터리로 이동합니다. `cd ~`
2. Ubuntu 패키지를 업데이트합니다. `sudo apt update`
3. MongoDB 패키지 관리 시스템에서 사용하는 공개 키를 가져옵니다. `wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add -`
4. MongoDB에 대한 목록 파일을 만듭니다. `echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list`
5. 로컬 패키지 데이터베이스를 다시 로드합니다. `sudo apt-get update`
6. MongoDB 패키지를 설치합니다. `sudo apt-get install -y mongodb-org`
7. 설치를 확인하고 버전 번호(`mongod --version`)를 가져옵니다.
8. 데이터를 저장할 디렉터리를 만듭니다. `mkdir -p ~/data/db`
9. Mongo 인스턴스를 실행합니다. `sudo mongod --dbpath ~/data/db`
10. MongoDB 인스턴스가 다음을 사용하여 실행 중인지 확인합니다. `ps -e | grep 'mongod'`
11. MongoDB 셸을 종료하려면 바로가기 키(Ctrl + C)를 사용합니다.

### 💡 팁

MongoDB를 설치하려면 설치에 사용되는 Linux 배포판에 따라 약간 다른 단계가 필요할 수 있습니다. [MongoDB 설치 자습서](#)를 참조하십시오. 또한 MongoDB 설치는 설치하려는 버전 #에 따라 다를 수 있습니다. MongoDB 설명서의 왼쪽 위 모서리에 있는 버전 드롭다운 목록을 사용하여 목표에 맞는 버전을 선택합니다.

## MongoDB init 시스템 차이점

위의 예제에서는 MongoDB를 직접 실행했습니다. 다른 자습서에서는 운영 체제의 기본 제공 init 시스템을 사용하여 MongoDB를 시작할 수 있습니다. 자습서 또는 문서에 사용된 `sudo systemctl status mongodb` 명령이 표시될 수 있습니다. 현재 WSL에서는 `systemd`(Linux의 서비스 관리 시스템)에 대한 지원을 제공하지 않습니다.

차이점은 볼 수 없지만 자습서에서 `sudo systemctl` 사용을 권장하는 경우 대신 `sudo /etc/init.d/`를 사용합니다. 예를 들어 WSL에 대한 `sudo systemctl status docker`는 `sudo /etc/init.d/docker status`가 될 수 있습니다. 또는 `sudo service docker status`를 사용할 수도 있습니다.

## Init 스크립트를 추가하여 MongoDB를 서비스로 시작합니다.

위의 설치 지침은 `/etc/init.d/`에 스크립트를 자동으로 포함하지 않는 MongoDB 버전을 설치합니다. 서비스 명령을 사용하려는 경우 [이 원본에서](#) `mongodb`에 대한 init.d 스크립트를 다운로드하고 수동으로 이 경로 `/etc/init.d/mongodb`에 파일로 배치한 다음 `sudo service mongodb start`을(를) 사용하여 Mongo를 서비스로 시작할 수 있습니다.

1. MongoDB에 대한 init.d 스크립트를 다운로드합니다. `curl`

```
https://raw.githubusercontent.com/mongodb/mongo/master/debian/init.d | sudo tee /etc/init.d/mongodb >/dev/null
```

2. 해당 스크립트 실행 권한을 할당합니다. `sudo chmod +x /etc/init.d/mongodb`

3. 이제 MongoDB 서비스 명령을 사용할 수 있습니다.

- `sudo service mongodb status`: 데이터베이스의 상태 확인 데이터베이스가 실행되고 있지 않으면 [실패] 응답이 표시됩니다.
- `sudo service mongodb start`: 데이터베이스 실행 시작 [확인] 응답이 표시됩니다.
- `sudo service mongodb stop`: 데이터베이스 실행 중지

4. 진단 명령을 사용하여 데이터베이스 서버에 연결되어 있는지 확인합니다. `mongo --eval 'db.runCommand({ connectionStatus: 1 })'` 현재 데이터베이스 버전, 서버 주소 및 포트, 상태 명령의 출력이 표시됩니다. 응답의 “확인” 필드에 대한 `1` 값은 서버가 작동하고 있음을 나타냅니다.

### ① 참고

MongoDB에는 데이터를 `/data/db`에 저장하고 포트 27017에서 실행하는 등의 몇 가지 기본 매개 변수가 있습니다. 또한 `mongod`는 디먼(데이터베이스에 대한 호스트 프로세스)이고 `mongo`는 `mongod`의 특정 인스턴스에 연결하는 명령줄 셸입니다.

VS Code는 [Azure CosmosDB 확장](#)을 통해 MongoDB 데이터베이스 작업을 지원합니다. VS Code 내에서 MongoDB 데이터베이스를 만들고, 관리하고, 쿼리할 수 있습니다. 자세한 내용은 VS Code 설명서: [MongoDB로 작업하기](#)를 참조하십시오.

다음 MongoDB 문서에서 자세히 알아보세요.

- [MongoDB 사용 소개](#)
- [사용자 만들기](#)
- [원격 호스트에서 MongoDB 인스턴스에 연결](#)
- [CRUD: 생성, 읽기, 업데이트, 삭제](#)
- [참조 문서](#)

## Microsoft SQL Server 설치

WSL(예: Ubuntu)에 SQL Server를 설치하려면 빠른 시작을 사용합니다. [SQL Server를 설치하고 Ubuntu에 데이터베이스를 만듭니다.](#)

### ① 참고

WSL에서 SQL Server를 설치하고 구성할 수 있지만, 지원되는 구성은 아닙니다. 또한, Linux에서 SQL Server는 WSL에 포함되지 않은 `systemd`가 필요합니다.

VS Code에서 Microsoft SQL Server 데이터베이스를 사용하려면, [MSSQL 확장](#)을 사용해 보십시오.

## SQLite 설치

WSL(예: Ubuntu)에 SQLite를 설치하려면 다음을 수행합니다.

1. WSL 터미널(예: Ubuntu)을 엽니다.
2. Ubuntu 패키지를 업데이트합니다. `sudo apt update`
3. 패키지가 업데이트되면 `sudo apt install sqlite3`을(를) 사용하여 SQLite3를 설치합니다.
4. 설치를 확인하고 버전 번호(`sqlite3 --version`)를 가져옵니다.

"example.db"라는 테스트 데이터베이스를 만들려면 `sqlite3 example.db`을(를) 입력합니다.

SQLite 데이터베이스 목록을 보려면 `.databases`을(를) 입력합니다.

데이터베이스의 상태를 보려면 `.dbinfo ?DB?`을(를) 입력합니다.

생성 후에는 데이터베이스가 비어 있습니다. `CREATE TABLE empty (kol INTEGER);` 을(를) 사용하여 데이터베이스에 대한 새 테이블을 만들 수 있습니다.

`.dbinfo ?DB?` 을(를) 입력하면 만든 데이터베이스가 표시됩니다.

SQLite 프롬프트를 종료하려면 `.exit` 을(를) 입력합니다.

SQLite 데이터베이스 작업에 대한 자세한 내용은 [SQLite 문서](#) 를 참조하십시오.

VS Code에서 SQLite 데이터베이스를 사용하려면 [SQLite 확장](#) 을 사용해 보십시오.

## Redis 설치

WSL(예: Ubuntu)에 Redis를 설치하려면 다음을 수행합니다.

1. WSL 터미널(예: Ubuntu)을 엽니다.
2. Ubuntu 패키지를 업데이트합니다. `sudo apt update`
3. 패키지가 업데이트되면 `sudo apt install redis-server` 을(를) 사용하여 Redis를 설치합니다.
4. 설치를 확인하고 버전 번호(`redis-server --version`)를 가져옵니다.

Redis 서버 실행을 시작하려면 다음을 수행합니다. `sudo service redis-server start`

Redis가 작동하는지 확인합니다.(Redis-cli는 Redis와 통신하는 명령줄 인터페이스 유필리티입니다.) `redis-cli ping` 그러면 "PONG"이라는 회신이 반환됩니다.

Redis 서버 실행을 중지하려면 다음을 수행합니다. `sudo service redis-server stop`

Redis 데이터베이스 작업에 대한 자세한 내용은 [Redis 문서](#) 를 참조하십시오.

VS Code에서 Redis 데이터베이스를 사용하려면 [Redis 확장](#) 을 사용해 보십시오.

## 프로필 별칭을 실행하고 설정하는 서비스를 참조하십시오.

현재 WSL 배포에서 실행 중인 서비스를 보려면 다음을 입력합니다. `service --status-all`

`sudo service mongodb start` 또는 `sudo service postgres start` 및 `sudo -u postrest psql` 을 입력하는 것은 지루할 수 있습니다. 그러나 WSL의 `.profile` 파일에 별칭을 설정하면 이러한 명령으로 더 빠르게 사용하고 더 쉽게 기억할 수 있습니다.

이러한 명령 실행을 위한 사용자 지정 별칭 또는 바로 가기를 설정하려면 다음을 수행합니다.

1. WSL 터미널을 열고 루트 디렉터리에 있도록 `cd ~`를 입력합니다.
2. 터미널 텍스트 편집기인 Nano를 사용하여 터미널의 설정을 제어하는 `.profile` 파일을 엽니다. `sudo nano .profile`
3. 파일의 아래쪽에서(`# set PATH` 설정을 변경하지 않음) 다음을 추가합니다.

Bash

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

이렇게 하면 `start-pg`를 입력하여 postgresql 서비스 실행을 시작하고 `run-pg`를 입력하여 psql 셸을 열 수 있습니다. `start-pg` 및 `run-pg`를 원하는 이름으로 변경할 수 있습니다. `postgres`가 이미 사용하는 명령을 덮어쓰지 않도록 주의하세요.

4. 새 별칭을 추가한 후에는 **Ctrl+X**를 사용하여 Nano 텍스트 편집기를 종료합니다. 저장 및 Enter에 대한 메시지가 표시되면 `Y(예)`를 선택합니다(파일 이름을 `.profile`로 그대로 둠).
5. WSL 터미널을 닫았다가 다시 연 다음, 새 별칭 명령을 시도합니다.

## 문제 해결

### 오류: 디렉터리 동기화 fdatasync 잘못된 인수

WSL 2 모드에서 Linux 배포를 실행하고 있는지 확인합니다. WSL 1에서 WSL 2로 전환하는 데 도움이 되는 도움말은 [배포 버전을 WSL 1 또는 WSL 2로 설정을 참조하십시오](#).

## 추가 자료

- [Windows에서 개발 환경 설정](#)

# WSL 2에서 Docker 원격 컨테이너 시작

아티클 • 2023. 03. 21. • 읽는 데 19분 걸림

이 단계별 가이드는 WSL 2(Linux용 Windows 하위 시스템, 버전 2)를 사용해 Windows용 Docker Desktop을 설정하여 원격 컨테이너로 개발을 시작하는 데 도움이 됩니다.

Windows용 Docker Desktop은 Docker화된 앱을 빌드, 배송 및 실행하기 위한 개발 환경을 제공합니다. WSL 2 기반 엔진을 사용하도록 설정하면 동일한 컴퓨터의 Docker Desktop에서 Linux 및 Windows 컨테이너를 모두 실행할 수 있습니다. (Docker Desktop은 개인용 및 중소기업용으로 무료입니다. Pro, Team 또는 Business 가격 책정에 대한 정보는 [Docker 사이트 FAQ](#)를 참조하세요.)

## ① 참고

Windows 및 Linux용 Windows 하위 시스템 통합 [\[1\]](#)되어 Docker Desktop을 사용하는 것이 좋습니다. 그러나 Docker Desktop은 Linux 및 Windows 컨테이너 실행을 모두 지원하지만 두 컨테이너를 동시에 실행할 수는 없습니다. Linux 및 Windows 컨테이너를 동시에 실행하려면 WSL에서 별도의 Docker 인스턴스를 설치하고 실행해야 합니다. 동시 컨테이너를 실행해야하거나 Linux 배포판에 직접 컨테이너 엔진을 설치하려는 경우 [Ubuntu에 Docker 엔진 설치](#) [\[2\]](#) 또는 [Linux 컨테이너 실행을 위한 Podman 설치](#) [\[3\]](#)와 같은 해당 컨테이너 서비스에 대한 Linux 설치 지침을 따릅니다.

## Docker 컨테이너 개요

Docker는 컨테이너를 사용하여 애플리케이션을 만들고, 배포하고, 실행하는 데 사용되는 도구입니다. 개발자는 컨테이너를 사용하여 필요한 모든 파트(라이브러리, 프레임워크, 종속성 등)와 함께 앱을 패키징하여 하나의 패키지로 제공할 수 있습니다. 컨테이너를 사용하면 앱을 실행하는 컴퓨터(코드를 작성하고 테스트하는 데 사용된 머신과 다를 수도 있음)의 사용자 지정된 설정 또는 이전에 설치한 라이브러리와 상관없이 앱이 동일하게 실행됩니다. 따라서 개발자는 코드가 실행될 시스템에 대해 신경 쓰지 않고 코드 작성에 집중할 수 있습니다.

Docker 컨테이너는 가상 머신과 비슷하지만 전체 가상 운영 체제를 만들지는 않습니다. 대신 Docker를 사용하면 앱이 실행 중인 시스템과 동일한 Linux 커널을 사용할 수 있습니다. 따라서 호스트 컴퓨터에 아직 없는 파트만 앱 패키지에 필요하므로 패키지 크기를 줄이고 성능을 높일 수 있습니다.

컨테이너가 인기 있는 또 다른 이유는 [Kubernetes](#) 같은 도구와 함께 Docker 컨테이너를 사용하면 지속적인 가용성을 얻을 수 있다는 점입니다. 이렇게 하면 여러 버전의 앱 컨테이너를 서로 다른 시간에 만들 수 있습니다. 업데이트 또는 유지 관리를 위해 전체 시스템

을 중단하는 대신 각 컨테이너(및 특정 마이크로 서비스)를 즉시 교체할 수 있습니다. 모든 업데이트가 포함된 새 컨테이너를 준비하고, 프로덕션용 컨테이너를 설정하고, 준비가 되면 새 컨테이너를 가리킬 수 있습니다. 컨테이너를 사용하여 여러 버전의 앱을 보관하고, 필요한 경우 안전 대비책으로 앱을 계속 실행할 수도 있습니다.

자세한 내용은 [Docker 컨테이너 소개](#)를 참조하세요.

## 사전 요구 사항

- 컴퓨터에서 Windows 10, [버전 2004로 업데이트됨](#), [빌드 18362](#) 이상을 실행 중인지 확인합니다.
- [WSL을 설치하고 WSL 2에서 실행되는 Linux 배포판의 사용자 이름과 암호를 설정합니다.](#)
- [Visual Studio Code를 설치합니다](#) (선택 사항). 이는 원격 Docker 컨테이너 내에서 코딩 및 디버그하고 Linux 배포에 연결된 기능을 포함하여 최상의 환경을 제공합니다.
- [Windows 터미널을 설치합니다](#)(선택 사항). 이렇게 하면 동일한 인터페이스 (Ubuntu, Debian, PowerShell, Azure CLI 또는 선호하는 항목 포함)에서 여러 터미널을 사용자 지정하고 여는 기능을 포함하여 최상의 환경을 제공합니다.
- [Docker Hub에서 Docker ID에 등록합니다](#) (선택 사항).
- 사용 약관에 대한 업데이트는 [Docker Desktop 라이선스 계약](#)을 참조하세요.

### ① 참고

WSL은 WSL 버전 1 또는 WSL 2 모드 모두에서 배포를 실행할 수 있습니다.

PowerShell을 열고 `wsl -l -v`를 입력하여 확인할 수 있습니다. `wsl --set-version <distro> 2`를 입력하여 배포가 WSL 2를 사용하도록 설정되었는지 확인합니다. `<distro>`를 배포판 이름(예: Ubuntu 18.04)으로 바꿉니다.

WSL 버전 1에서는 Windows와 Linux 간의 근본적인 차이점으로 인해 Docker 엔진이 WSL 내에서 직접 실행될 수 없었기 때문에 Docker 팀은 Hyper-V VM 및 LinuxKit을 사용하여 대체 솔루션을 개발했습니다. 그러나 WSL 2는 이제 전체 시스템 호출 용량을 갖춘 Linux 커널에서 실행되므로 Docker는 WSL 2에서 완전히 실행할 수 있습니다. 즉, Linux 컨테이너는 에뮬레이션 없이 기본적으로 실행될 수 있으므로 Windows와 Linux 도구 간의 성능과 상호 운용성이 향상됩니다.

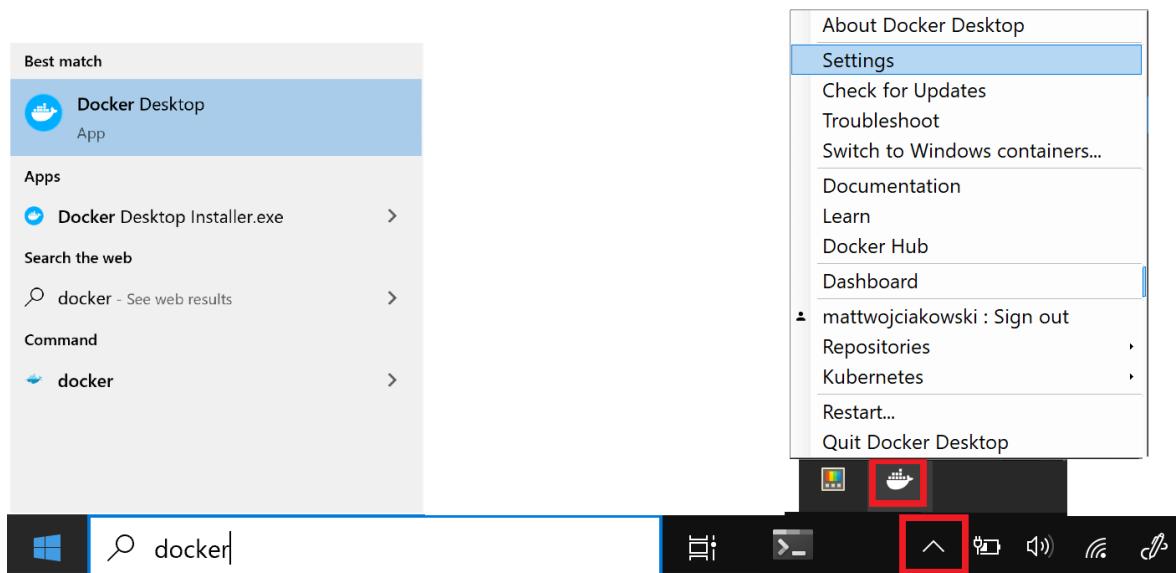
## Docker Desktop 설치

Windows용 Docker Desktop에서 지원되는 WSL 2 백 엔드를 사용하면 Linux 기반 개발 환경에서 작업하고 Linux 기반 컨테이너를 빌드하는 동시에 코드 편집 및 디버깅에 Visual

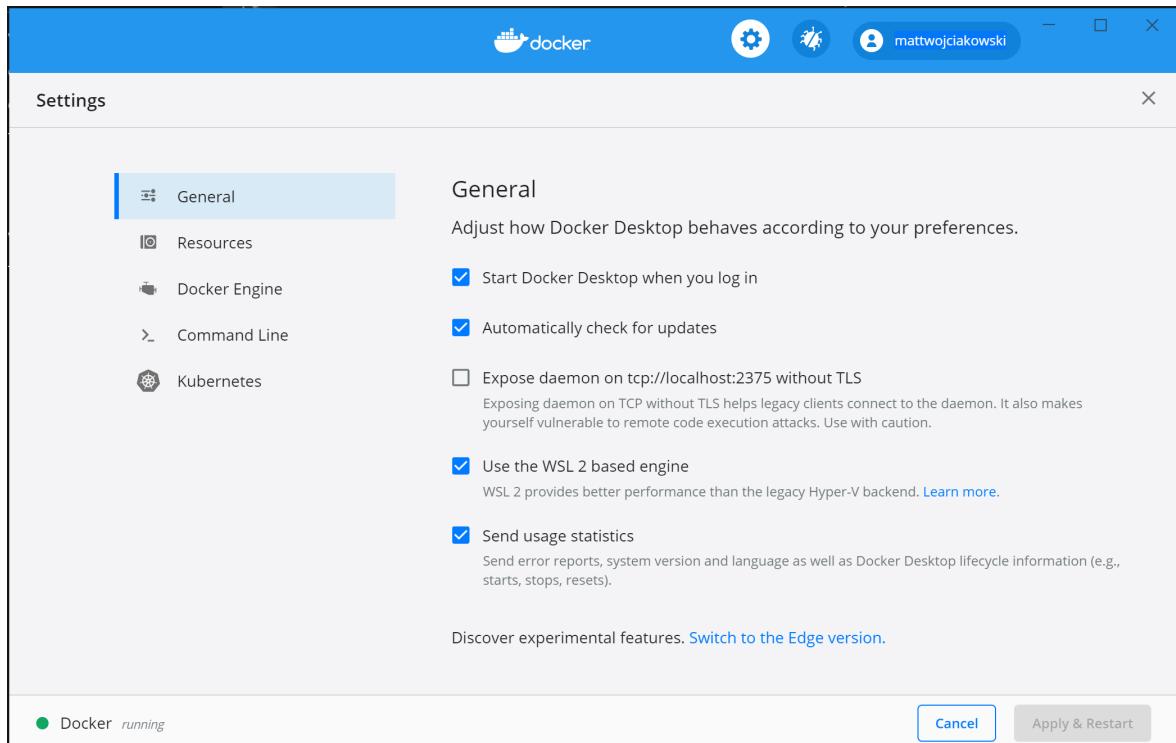
Studio Code를 사용하고 Windows의 Microsoft Edge 브라우저에서 컨테이너를 실행할 수 있습니다.

Docker를 설치하려면(이미 WSL을 설치한 후):

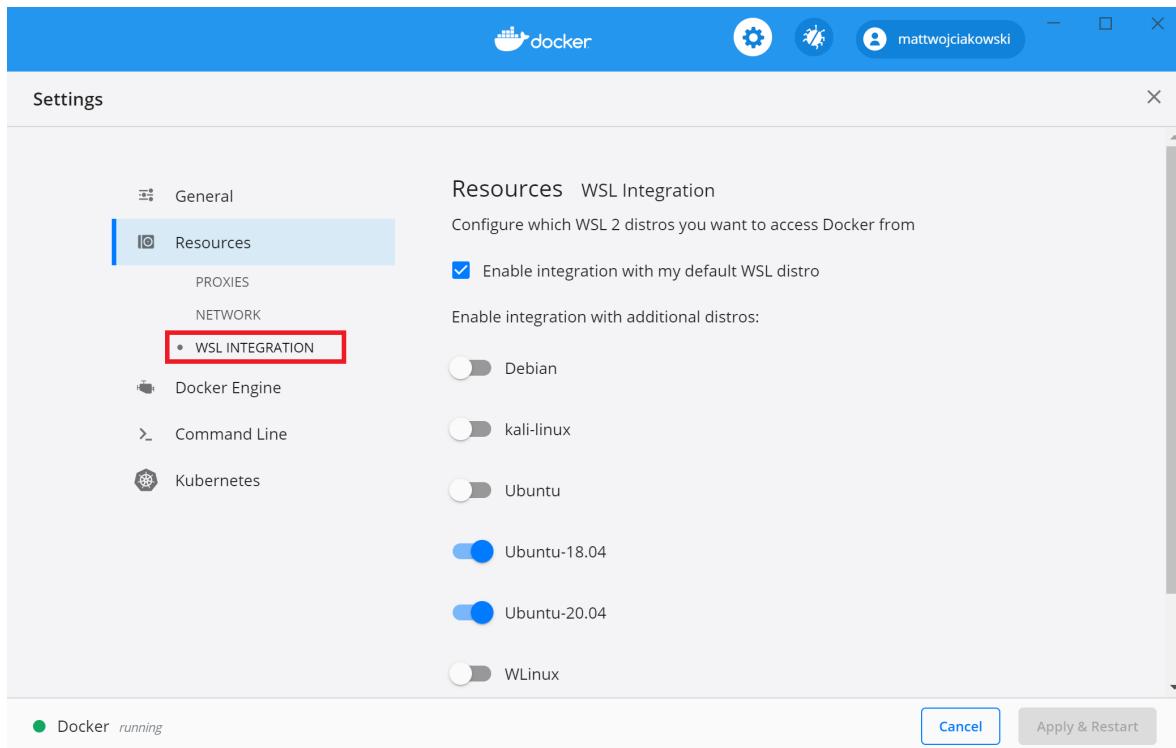
1. Docker Desktop [을 다운로드하고 설치 지침을 따릅니다.](#)
2. 설치가 완료되면 Windows 시작 메뉴에서 Docker Desktop을 시작한 다음 작업 표시 줄의 숨겨진 아이콘 메뉴에서 Docker 아이콘을 선택합니다. 아이콘을 마우스 오른쪽 단추로 클릭하여 Docker 명령 메뉴를 표시하고 "설정"을 선택합니다.



3. 설정>일반에서 "WSL 2 기반 엔진 사용"이 선택되어 있는지 확인합니다.



4. 설정>리소스>WSL 통합으로 이동하여 Docker 통합을 사용하도록 설정하려는 설치된 WSL 2 배포판에서 선택합니다.



5. Docker가 설치되었는지 확인하려면 WSL 배포(예: Ubuntu)를 열고 다음을 입력하여 버전 및 빌드 번호를 표시합니다. `docker --version`

6. `docker run hello-world` 명령을 사용하여 간단한 기본 제공 Docker 이미지를 실행하여 올바르게 설치되었는지 테스트합니다.

### 💡 팁

다음은 알아야 할 몇 가지 유용한 Docker 명령입니다.

- Docker CLI에서 사용할 수 있는 명령 나열: `docker`
- 특정 명령에 대한 정보 나열: `docker <COMMAND> --help`
- 머신의 docker 이미지 나열(현재는 hello-world 이미지만 나열됨): `docker image ls --all`
- `docker container ls --all` 또는 `docker ps -a`를 사용하여 컴퓨터의 컨테이너를 나열합니다(-a show all 플래그가 없으면 실행 중인 컨테이너만 표시됨).
- WSL 2 컨텍스트에서 사용할 수 있는 통계 및 리소스(CPU & 메모리)를 포함하여 Docker 설치에 관한 시스템 전체 정보를 다음과 함께 나열합니다. `docker info`

## VS Code를 사용하여 원격 컨테이너에서 개발

WSL 2와 함께 Docker를 사용하여 앱 개발을 시작하려면 WSL, Dev Containers 및 Docker 확장과 함께 VS Code를 사용하는 것이 좋습니다.

- [VS Code WSL 확장을 설치합니다](#). 이 확장을 사용하면 VS Code의 WSL에서 실행되는 Linux 프로젝트를 열 수 있습니다(경로 문제, 이진 파일 호환성 또는 기타 OS 간 문제에 대해 걱정할 필요 없음).
- [VS Code Dev Containers 확장을 설치합니다](#). 이 확장을 사용하면 컨테이너 내에서 개발 작업을 수행하도록 설정된 Visual Studio Code의 전체 기능을 활용하여 컨테이너 내부의 프로젝트 폴더 또는 리포지토리를 열 수 있습니다.
- [VS Code Docker 확장 설치](#). 이 확장은 VS Code 내부에서 컨테이너화된 애플리케이션을 빌드, 관리 및 배포하는 기능을 추가합니다. (실제로 컨테이너를 개발 환경으로 사용하려면 Dev Containers 확장이 필요합니다.)

Docker를 사용하여 기존 앱 프로젝트에 대한 개발 컨테이너를 만들어 보겠습니다.

1. 이 예에서는 Python 개발 환경 설정 문서의 [Django용 Hello World 자습서](#)의 소스 코드를 사용할 예정입니다. 자체 프로젝트 소스 코드를 사용하려면 이 단계를 건너뛸 수 있습니다. GitHub에서 내 HelloWorld-Django 웹앱을 다운로드하려면 WSL 터미널(예: Ubuntu)을 열고 다음을 입력합니다. `git clone https://github.com/mattwojo/helloworld-django.git`

#### ① 참고

항상 도구를 사용하는 것과 동일한 파일 시스템에 코드를 저장합니다. 이렇게 하면 파일 액세스 성능이 빨라집니다. 이 예에서는 Linux 배포판(Ubuntu)을 사용 중이며 프로젝트 파일을 WSL 파일 시스템 \\ws1\\에 저장하려고 합니다. Windows 파일 시스템에 프로젝트 파일을 저장하면 WSL에서 Linux 도구를 사용하여 해당 파일에 액세스할 때 작업 속도가 크게 느려집니다.

2. WSL 터미널에서 이 프로젝트의 소스 코드 폴더로 디렉터리를 변경합니다.

Bash

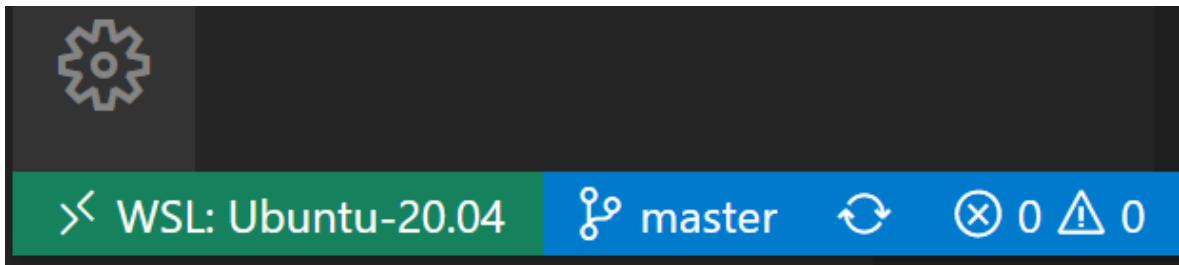
```
cd helloworld-django
```

3. 다음을 입력하여 로컬 WSL 확장 서버에서 실행 중인 VS Code에서 프로젝트를 엽니다.

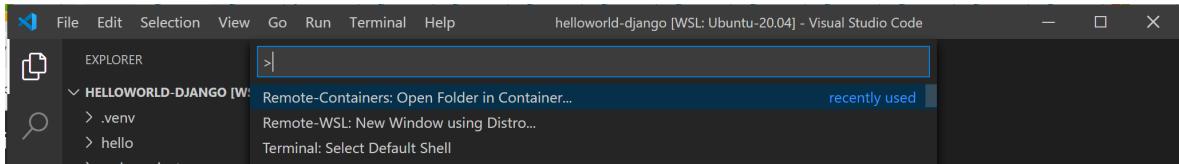
Bash

```
code .
```

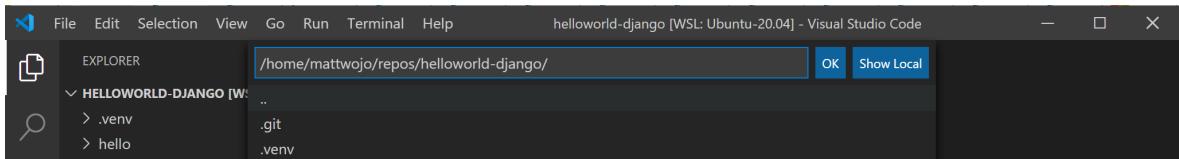
VS Code 인스턴스의 왼쪽 하단 모서리에 있는 녹색 원격 표시기를 확인하여 WSL Linux 배포판에 연결되어 있는지 확인합니다.



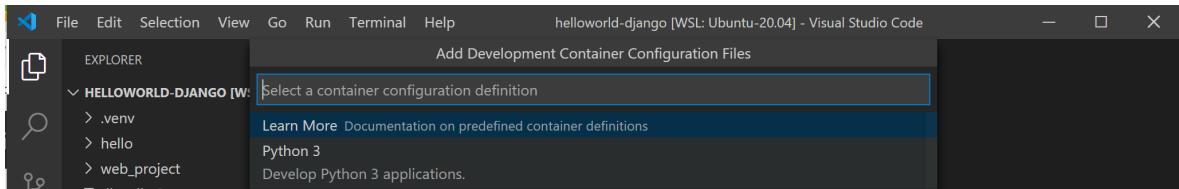
4. VS Code 명령 팔레트(Ctrl + Shift + P)에서 **Dev Containers: Open Folder in Container...**를 입력합니다. 입력을 시작할 때 이 명령이 표시되지 않으면 위에 링크된 Dev Containers 확장을 설치했는지 확인합니다.



5. 컨테이너화하려는 프로젝트 폴더를 선택합니다. 제 경우에는 `\wsl\Ubuntu-20.04\home\mattwojo\repos\helloworld-django`입니다.



6. 아직 프로젝트 폴더(repo)에 개발 컨테이너 구성이 없기 때문에 컨테이너 정의 목록이 나타납니다. 표시되는 컨테이너 구성 정의 목록은 프로젝트 형식에 따라 필터링됩니다. Django 프로젝트의 경우 Python 3을 선택할 예정입니다.



7. VS Code의 새 인스턴스가 열리고 새 이미지 빌드가 시작되면 빌드가 완료되면 컨테이너가 시작됩니다. `Dockerfile` 및 `devcontainer.json` 파일 내부의 컨테이너 구성 정보와 함께 새 `.devcontainer` 폴더가 나타나는 것을 볼 수 있습니다.

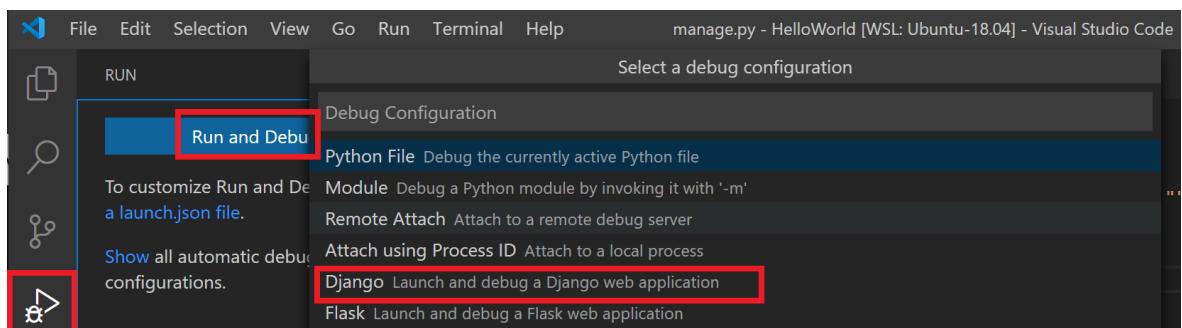
```

{
    "name": "Python 3",
    "build": {
        "dockerfile": "Dockerfile",
        "context": ".",
        // Update 'VARIANT' to pick a Python version: 3, 3.6, 3.7, 3.8
        "args": { "VARIANT": "3" }
    },
}

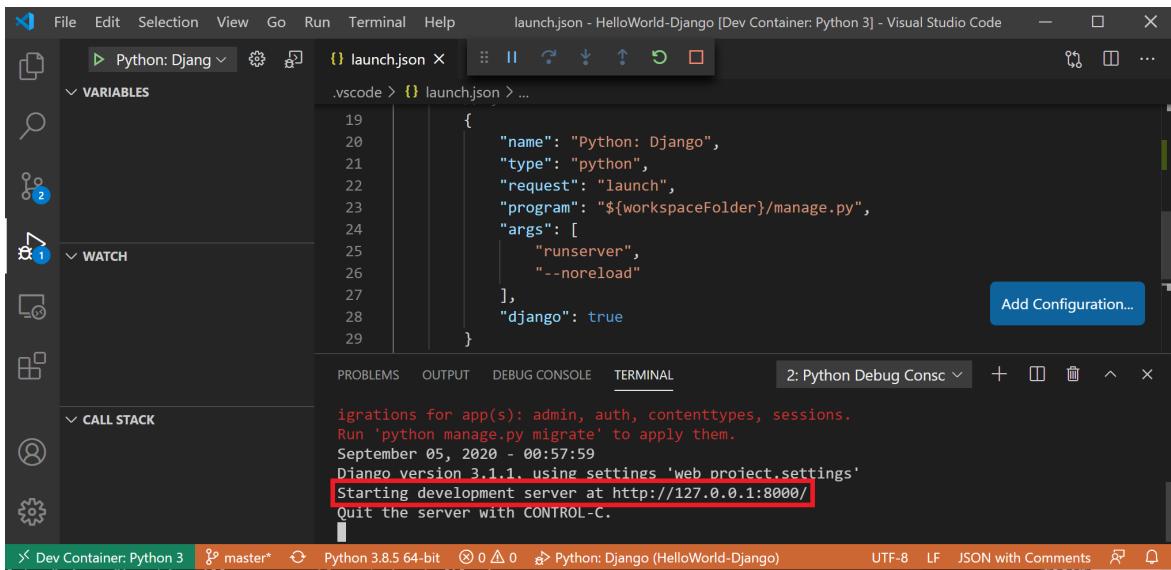
```

8. 프로젝트가 여전히 WSL과 컨테이너 내 모두에 연결되어 있는지 확인하려면 VS Code 통합 터미널(Ctrl + Shift + ~)을 엽니다. `uname`을 입력하여 운영 체제를 확인하고 및 `python3 --version`을 입력하여 Python 버전을 확인합니다. `uname`이 "Linux"를 반환하므로 여전히 WSL 2 엔진에 연결되어 있고 Python 버전 번호는 WSL 배포에 설치된 Python 버전과 다를 수 있는 컨테이너 구성을 기반으로 합니다.

9. Visual Studio Code를 사용하여 컨테이너 내에서 앱을 실행하고 디버그하려면 먼저 실행 메뉴를 엽니다(Ctrl+Shift+D 또는 맨 왼쪽 메뉴 모음에서 탭 선택). 그런 다음 실행 및 디버그를 선택하여 디버그 구성의 선택하고 프로젝트에 가장 적합한 구성을 선택합니다(현재 예에서는 "Django"). 이렇게 하면 앱 실행 방법에 대한 지침과 함께 프로젝트의 `.vscode` 폴더에 `launch.json` 파일이 만들어집니다.



10. VS Code 내부에서 실행>디버깅 시작을 선택하거나 F5 키를 누릅니다. 이렇게 하면 VS Code 내에서 터미널이 열리고 "<http://127.0.0.1:8000/>"에서 개발 서버 시작 중 - C로 서버 종료"와 같은 결과가 표시됩니다. 키를 누른 상태에서 표시되는 주소를 선택하여 기본 웹 브라우저에서 앱을 열고 컨테이너 내에서 실행 중인 프로젝트를 확인합니다.



이제 VS Code를 사용하여 코딩, 빌드, 실행, 배포 또는 디버그할 수 있는 WSL 2 백 엔드로 구동되는 Docker Desktop을 사용하여 원격 개발 컨테이너를 성공적으로 구성했습니다!

## 문제 해결

### WSL Docker 컨텍스트가 더 이상 사용되지 않음

WSL용 Docker의 초기 Tech Preview를 사용하고 있었다면 이제 더 이상 사용되지 않고 더 이상 사용되지 않는 "wsl"이라는 Docker 컨텍스트가 있을 수 있습니다. `docker context ls` 명령을 사용하여 확인할 수 있습니다. 이 "wsl" 컨텍스트를 제거하여 Windows 및 WSL2 모두에 대한 기본 컨텍스트를 사용하려는 경우 명령을 `docker context rm wsl` 사용하여 오류를 방지할 수 있습니다.

더 이상 사용되지 않는 이 wsl 컨텍스트에서 발생할 수 있는 오류는 다음과 같습니다.

```
docker wsl open //./pipe/docker_wsl: The system cannot find the file specified. 또는 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.
```

이 문제에 대한 자세한 내용은 [Windows 10의 Windows System for Linux\(WSL2\) 내에서 Docker를 설정하는 방법](#)을 참조하세요.

### Docker 이미지 스토리지 폴더를 찾는데 문제가 있습니다.

Docker는 데이터를 저장할 두 개의 distro 폴더를 만듭니다.

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

WSL Linux 배포를 열고 `explorer.exe` 를 입력하여 Windows 파일 탐색기에서 폴더를 보면 이러한 폴더를 찾을 수 있습니다. `\\\wsl\\<distro name>\\mnt\\wsl`을 입력합니다.

`<distro name>`을 배포 이름(예: Ubuntu-20.04)으로 바꾸면 이 폴더를 볼 수 있습니다.

WSL에서 Docker 스토리지 위치 찾기에 대해 자세히 알아봅니다. [WSL 스토리지의 문제 ↗](#) 또는 이 [StackOverflow 게시물 ↗](#)을 참조하세요.

WSL의 일반적인 문제 해결에 대한 추가 도움말은 [문제 해결 문서](#)를 참조하세요.

## 추가 자료

- Docker 문서: WSL 2를 사용하는 Docker Desktop의 모범 사례 ↗
- Windows용 Docker Desktop에 대한 피드백: 문제 제출 ↗
- VS Code 블로그: 개발 환경 선택 지침 ↗
- VS Code 블로그: WSL 2에서 Docker 사용 ↗
- VS Code 블로그: WSL 2에서 원격 컨테이너 사용 ↗
- Hanselminutes 팟캐스트: Simon Ferquel과 함께 개발자를 위한 Docker 만들기 ↗

# 연습: WSL 2 및 Visual Studio 2022를 사용하여 C++ 빌드 및 디버그

아티클 • 2023. 04. 03. • 읽는 데 16분 걸림

Visual Studio 2022에서는 WSL 2(Linux용 Windows 하위 시스템 버전 2) 개발을 위한 네이티브 C++ 도구를 도입합니다. 이 도구 집합은 이제 [Visual Studio 2022 버전 17.0](#) 이상에서 사용할 수 있습니다.

WSL 2는 새로운 권장 버전의 WSL([Linux용 Windows 하위 시스템](#))입니다. 이는 향상된 Linux 파일 시스템 성능, GUI 지원, 전체 시스템 호출 호환성을 제공합니다. Visual Studio의 WSL 2 도구 집합을 사용하면 SSH 연결을 추가하지 않고도 Visual Studio를 사용하여 WSL 2 배포판에서 C++ 코드를 빌드하고 디버그할 수 있습니다. 이미 Visual Studio 2019 버전 16.1에 도입된 네이티브 [WSL 1 도구 집합](#)을 사용하여 WSL 1 배포판에서 C++ 코드를 빌드하고 디버그할 수 있습니다.

Visual Studio의 WSL 2 도구 집합은 CMake 및 MSBuild 기반 Linux 프로젝트를 모두 지원합니다. CMake는 Visual Studio를 사용하는 모든 C++ 플랫폼 간 개발을 위한 권장 사항입니다. CMake는 Windows, WSL, 원격 시스템에서 동일한 프로젝트를 빌드하고 디버그하기 때문에 CMake를 권장합니다.

이 항목의 정보에 관한 비디오 프레젠테이션은 [비디오: WSL 2 배포 및 Visual Studio 2022를 사용하여 C++ 디버그](#)를 참조하세요.

## WSL 2 도구 집합 배경

Visual Studio의 C++ 플랫폼 간 지원에서는 모든 소스 파일이 Windows 파일 시스템에서 시작된 것으로 가정합니다. WSL 2 배포판을 대상으로 하는 경우 Visual Studio는 로컬 `rsync` 명령을 실행하여 Windows 파일 시스템에서 WSL 파일 시스템으로 파일을 복사합니다. 로컬 `rsync` 복사에는 사용자 개입이 필요하지 않습니다. 이 작업은 Visual Studio가 WSL 2 배포판이 사용 중이라는 것을 검색하면 자동으로 수행됩니다. WSL 1과 WSL 2의 차이점에 관한 자세한 내용은 [WSL 1 및 WSL 2 비교](#)를 참조하세요.

WSL 2 도구 집합은 Visual Studio에서 CMake 사전 설정 통합을 통해 지원됩니다. 자세한 내용은 [Visual Studio 및 Visual Studio Code의 CMake 사전 설정 통합](#) 및 [Visual Studio에서 CMake 사전 설정으로 구성 및 빌드](#)를 참조하세요. 이 문서의 [고급 WSL 2 및 CMake 프로젝트 고려 사항](#)에서 더 많은 고급 정보를 확인할 수 있습니다.

## 빌드 도구 설치

WSL 2에서 빌드 및 디버그하는 데 필요한 도구를 설치합니다. 이후 단계에서 Visual Studio의 CMake 이진 배포를 사용하여 최신 버전의 CMake를 설치합니다.

1. [WSL 설치](#)의 지침에 따라 WSL 및 WSL 2 배포판을 설치합니다.
2. 배포판이 `apt`를 사용한다고 가정하고(이 연습에서는 Ubuntu 사용) 다음 명령을 사용하여 WSL 2 배포판에 필요한 빌드 도구를 설치합니다.

Bash

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

위의 `apt` 명령은 다음을 설치합니다.

- C++ 컴파일러
- `gdb`
- `CMake`
- `rsync`
- `zip`
- 기본 빌드 시스템 생성기

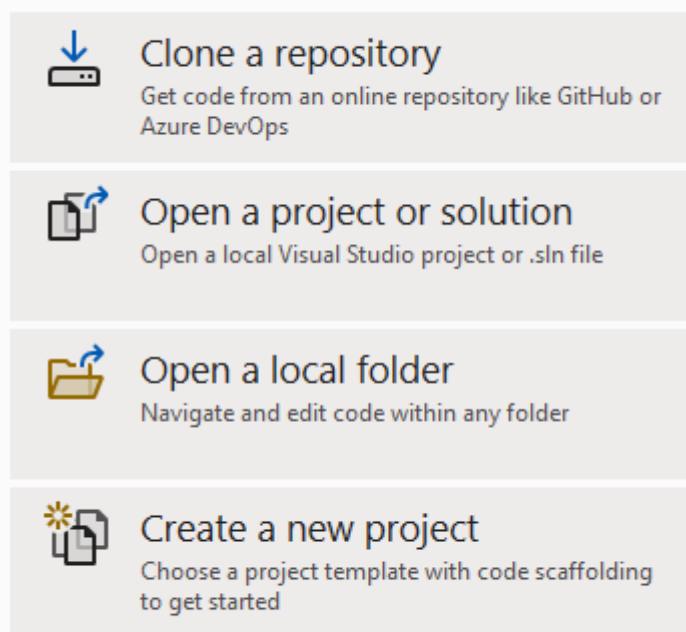
## WSL 2 배포판을 사용한 플랫폼 간 CMake 개발

이 연습에서는 Ubuntu에서 GCC 및 Ninja를 사용합니다. Visual Studio 2022 버전 17.0 미리 보기 2 이상도 사용합니다.

Visual Studio는 프로젝트 루트에 있는 `CMakeLists.txt` 파일을 사용하여 CMake 프로젝트를 풀더로 정의합니다. 이 연습에서는 Visual Studio CMake Project 템플릿을 사용하여 새 CMake 프로젝트를 만듭니다.

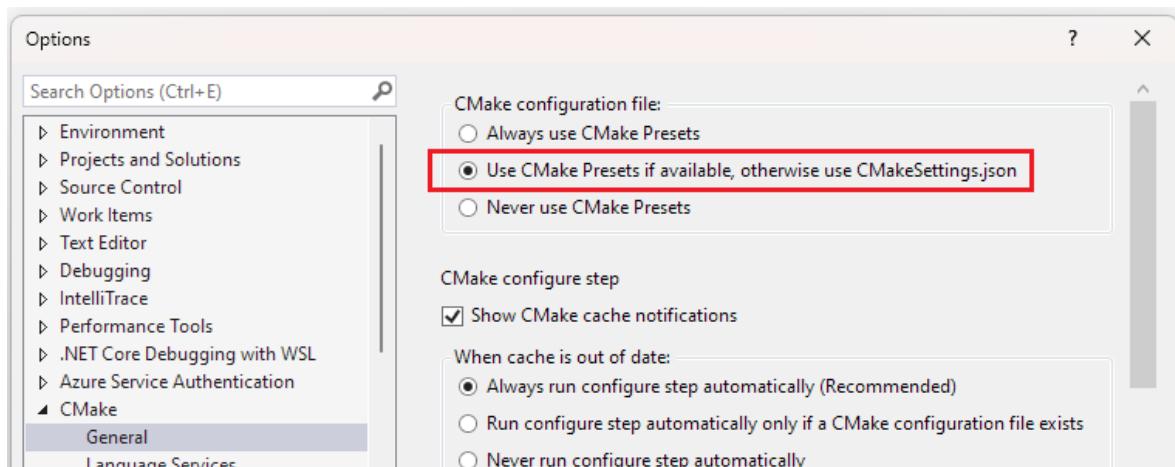
3. Visual Studio 시작 화면에서 새 프로젝트 만들기를 선택합니다.

## Get started



4. 템플릿 검색 텍스트 상자에 “cmake”를 입력합니다. CMake Project 형식을 선택하고 다음을 선택합니다. 프로젝트에 이름과 위치를 지정한 다음, 만들기를 선택합니다.

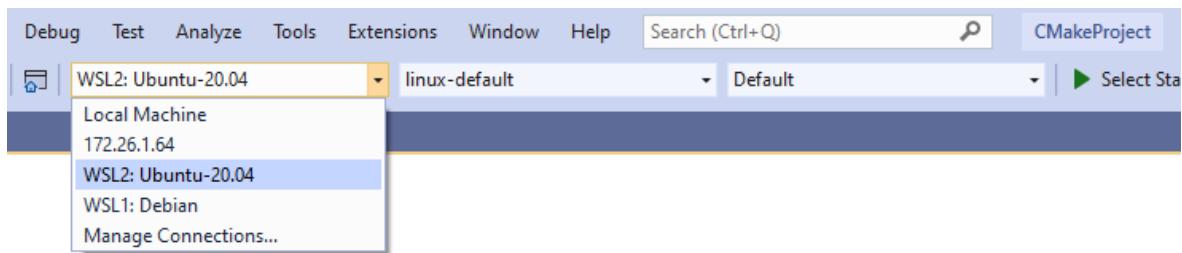
5. Visual Studio의 CMake 사전 설정 통합을 사용하도록 설정합니다. 도구>옵션 >CMake>일반을 선택합니다. 구성, 빌드 및 테스트에 CMake 사전 설정을 사용하는 것이 좋습니다. 를 선택한 다음, 확인을 선택합니다. 대신 프로젝트의 루트에 CMakePresets.json 파일을 추가했을 수 있습니다. 자세한 내용은 [CMake 사전 설정 통합 사용](#)을 참조하세요.



6. 통합을 활성화하려면: 주 메뉴에서 파일>폴더 닫기를 선택합니다. 시작 페이지가 나타납니다. 최근 파일 열기에서 방금 닫은 폴더를 선택하여 폴더를 다시 엽니다.

7. Visual Studio 주 메뉴 모음에 걸쳐 세 개의 드롭다운이 있습니다. 왼쪽의 드롭다운을 사용하여 활성 대상 시스템을 선택합니다. 이 시스템은 프로젝트를 구성하고 빌

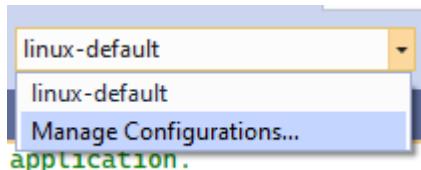
드하기 위해 CMake가 호출되는 시스템입니다. Visual Studio는 `wsl -l -v`를 사용하여 WSL 설치를 쿼리합니다. 다음 이미지에는 WSL2: Ubuntu-20.04가 대상 시스템으로 선택된 것으로 표시됩니다.



### ① 참고

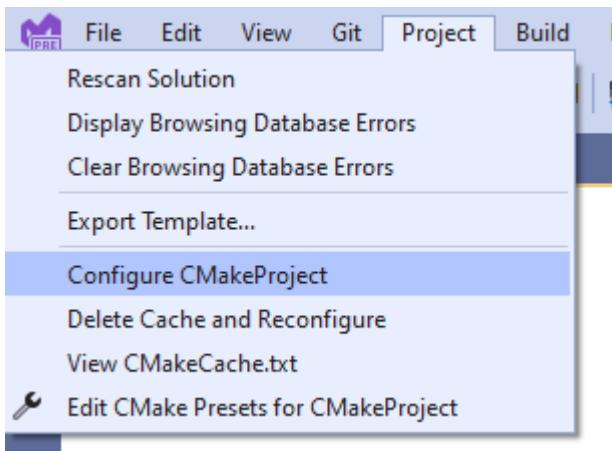
Visual Studio가 자동으로 프로젝트를 구성하기 시작하는 경우 11단계를 읽어서 CMake 이진 배포를 관리하고 아래 단계를 계속 진행합니다. 이 동작을 사용자 지정하려면 **자동 구성 및 캐시 알림 설정**을 참조하세요.

8. 가운데에 있는 드롭다운을 사용하여 활성 구성 사전 설정을 선택합니다. 구성 사전 설정은 CMake를 호출하고 기본 빌드 시스템을 생성하는 방법을 Visual Studio에 알려 줍니다. 7단계에서 활성 구성 사전 설정은 Visual Studio에서 만든 `linux-default` 사전 설정입니다. 사용자 지정 구성 사전 설정을 만들려면 **구성 관리...**를 선택합니다. 사전 설정 구성에 대한 자세한 내용은 [사전 설정 구성 및 사전 설정편집](#)을 참조하세요.

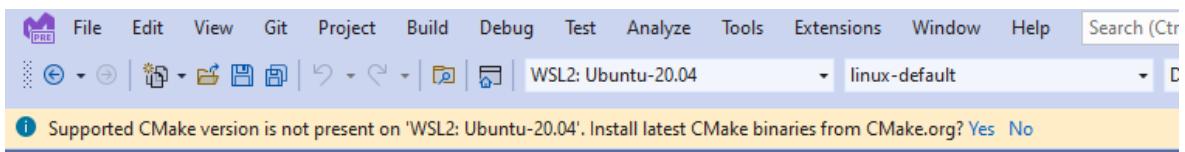


9. 오른쪽의 드롭다운을 사용하여 활성 빌드 사전 설정을 선택합니다. 빌드 사전 설정은 빌드를 호출하는 방법을 Visual Studio에 알려 줍니다. 7단계의 일러스트레이션에서 활성 빌드 사전 설정은 Visual Studio에서 만든 **기본** 빌드 사전 설정입니다. 빌드 사전 설정에 관한 자세한 내용은 [빌드 사전 설정 선택](#)을 참조하세요.

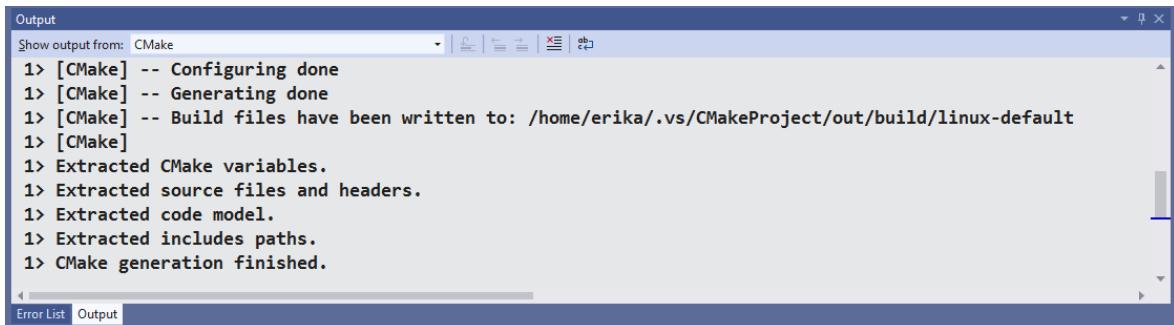
10. WSL 2에서 프로젝트를 구성합니다. 프로젝트 생성이 자동으로 시작되지 않는 경우 `ProjectConfigure project-name>`을 사용하여 구성을 수동으로 호출합니다.



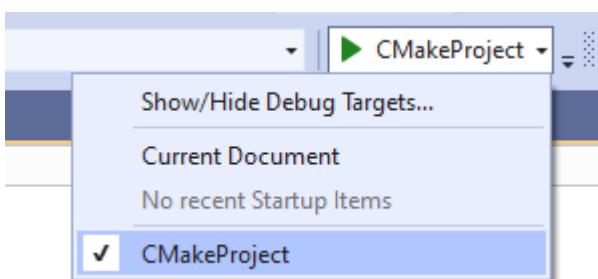
11. WSL 2 배포판에 지원되는 버전의 CMake가 설치되어 있지 않으면 Visual Studio는 주 메뉴 리본 바로 아래에 최신 버전의 CMake를 배포할지 묻는 메시지를 표시합니다. 예를 선택하여 WSL 2 배포판에 CMake 이진 파일을 배포합니다.



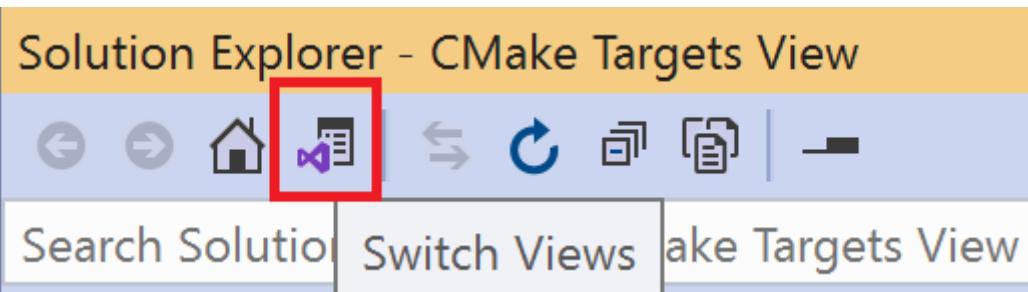
12. 구성 단계가 완료되었고 CMake 창 아래 출력 창에서 CMake 생성 완료 메시지를 볼 수 있는지 확인합니다. 빌드 파일은 WSL 2 배포판의 파일 시스템의 디렉터리에 기록됩니다.



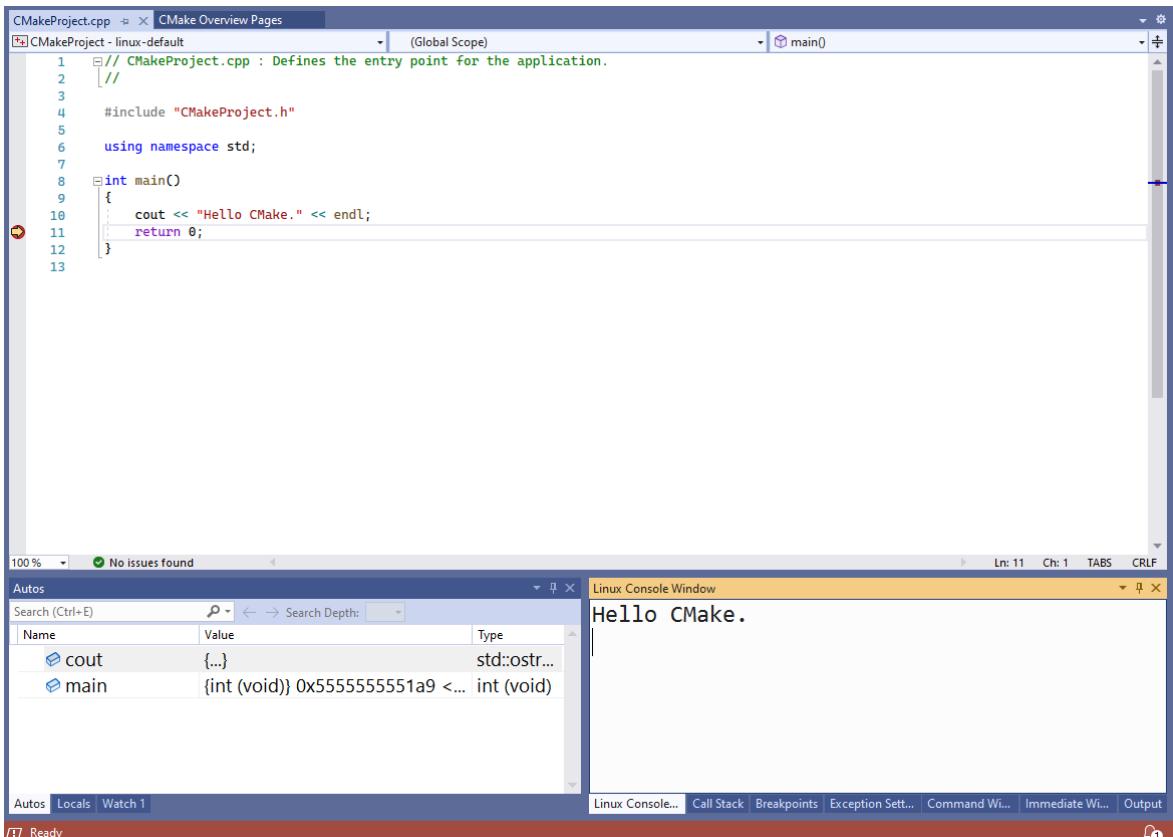
13. 활성 디버그 대상을 선택합니다. 디버그 드롭다운 메뉴에는 프로젝트에 사용할 수 있는 모든 CMake 대상이 나열됩니다.



14. 솔루션 탐색기에서 프로젝트 하위 폴더를 확장합니다. CMakeProject.cpp 파일의 main()에서 중단점을 설정합니다. 다음 스크린샷에서 강조 표시된 솔루션 탐색기의 보기 선택기 단추를 선택하여 CMake 대상 보기로 이동할 수도 있습니다.



15. 디버그>시작을 선택하거나 F5 키를 누릅니다. 프로젝트가 빌드되고, 실행 파일이 WSL 2 배포판에서 시작되고, Visual Studio가 중단점에서 실행을 중지합니다. Linux 콘솔 창에서 프로그램의 출력(이 경우 "Hello CMake.")을 볼 수 있습니다.



이제 WSL 2 및 Visual Studio 2022를 사용하여 C++ 앱을 빌드하고 디버그했습니다.

## 고급 WSL 2 및 CMake 프로젝트 고려 사항

Visual Studio는 `CMakePresets.json`을 활성 구성 파일로 사용하는 CMake 프로젝트에 대한 네이티브 WSL 2 지원만 제공합니다. `CMakeSettings.json`에서 `CMakePresets.json`으로 마이그레이션하려면 [Visual Studio에서 CMake 사전 설정 통합 사용](#)을 참조하세요.

WSL 2 배포를 대상으로 하고 WSL 2 도구 집합을 사용하지 않으려면 `CMakePresets.json`의 Visual Studio 원격 설정 공급업체 맵에서 `forceWSL1Toolset`을 `true`로 설정합니다. 자세한 내용은 [Visual Studio 원격 설정 공급업체 맵](#)을 참조하세요.

`forceWSL1Toolslet`이 `true`로 설정된 경우 Visual Studio는 WSL 파일 시스템에서 소스 파일 복사본을 유지 관리하지 않습니다. 대신 탑재된 Windows 드라이브(`/mnt/...`)에서 소스 파일에 액세스합니다.

대부분의 경우 WSL 2 배포판에서 WSL 2 도구 집합을 사용하는 것이 가장 좋습니다. 프로젝트 파일이 대신 Windows 파일 시스템에 저장되는 경우 WSL 2가 느리기 때문입니다. WSL 2의 파일 시스템 성능에 관한 자세한 내용은 [WSL 1 및 WSL 2 비교](#)를 참조하세요.

`CMakePresets.json`의 Visual Studio 원격 설정 공급업체 맵에서 프로젝트를 복사할 WSL 2의 디렉터리 경로, 복사 소스 옵션, rsync 명령 인수와 같은 고급 설정을 지정합니다. 자세한 내용은 [Visual Studio 원격 설정 공급업체 맵](#)을 참조하세요.

네이티브 IntelliSense 환경을 제공하기 위해 시스템 헤더이 Windows 파일 시스템에 자동으로 복사됩니다. `CMakePresets.json`의 Visual Studio 원격 설정 공급업체 맵에서 이 복사본에 포함되거나 제외되는 헤더를 사용자 지정할 수 있습니다.

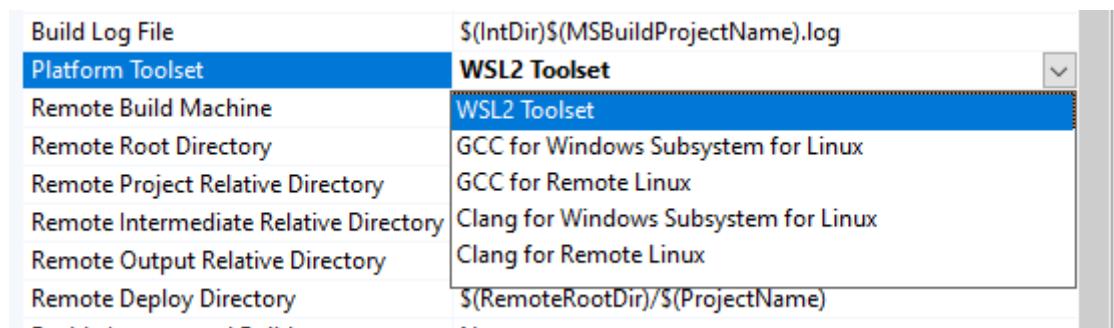
`CMakePresets.json`의 Visual Studio 설정 공급업체 맵에서 IntelliSense 모드를 변경하거나 다른 IntelliSense 옵션을 지정할 수 있습니다. 공급업체 맵에 관한 자세한 내용은 [Visual Studio 원격 설정 공급업체 맵](#)을 참조하세요.

## WSL 2 및 MSBuild 기반 Linux 프로젝트

CMake를 사용하면 Windows, WSL, 원격 시스템에서 동일한 프로젝트를 빌드하고 디버그 할 수 있으므로 CMake는 Visual Studio를 사용하여 모든 C++ 플랫폼 간 개발에 권장됩니다.

하지만 MSBuild 기반 Linux 프로젝트가 있을 수 있습니다.

MSBuild 기반 Linux 프로젝트가 있는 경우 Visual Studio에서 WSL 2 도구 집합으로 업그레이드할 수 있습니다. 솔루션 탐색기에서 프로젝트를 마우스 오른쪽 단추로 클릭한 다음 속성>일반>플랫폼 도구 집합을 선택합니다.



WSL 2 배포를 대상으로 하고 WSL 2 도구 집합을 사용하지 않으려면 **플랫폼 도구 집합** 드롭다운에서 **Linux용 Windows 하위 시스템 GCC 또는 Linux용 Windows 하위 시스템** 도구 집합용 **Clang**를 선택합니다. 이러한 도구 집합 중 하나가 선택되면 Visual Studio는

WSL 파일 시스템에서 소스 파일의 복사본을 유지 관리하지 않으면 대신 탑재된 Windows 드라이브(/mnt/...)를 통해 소스 파일에 액세스합니다. 네이티브 IntelliSense 환경을 제공하기 위해 시스템 헤더이 Windows 파일 시스템에 자동으로 복사됩니다. [속성 페이지>일반](#)에서 이 복사본에 포함되거나 제외되는 헤더를 사용자 지정합니다.

대부분의 경우 WSL 2 배포판에서 WSL 2 도구 집합을 사용하는 것이 가장 좋습니다. 프로젝트 파일이 Windows 파일 시스템에 저장되는 경우 WSL 2가 느리기 때문입니다. 자세한 내용은 [WSL 1 및 WSL 2 비교](#)를 참조하세요.

## 참고 항목

[비디오: WSL 2 배포 및 Visual Studio 2022를 사용하여 C++ 디버그 ↗](#)

[Visual Studio 2022 다운로드 ↗](#)

[Visual Studio에서 CMake Linux 프로젝트 만들기](#)

[자습서: 원격 Windows 머신에서 CMake 프로젝트 디버그](#)

# WSL에서 ML용 GPU 가속 시작

아티클 • 2023. 03. 21. • 읽는 데 5분 걸림

ML(기계 학습)은 많은 개발 워크플로의 핵심 부분이 되고 있습니다. 데이터 과학자, ML 엔지니어 또는 ML 학습 과정을 시작하는 사용자에게 WSL(Linux용 Windows 하위 시스템)은 가장 일반적이고 널리 사용되는 GPU 가속 ML 도구를 실행할 수 있는 훌륭한 환경을 제공합니다.

이러한 도구를 설정하는 방법에는 여러 가지가 있습니다. 예를 들어, [WSL의 NVIDIA CUDA](#), [TensorFlow-DirectML](#) 및 [PyTorch-DirectML](#)은 모두 WSL에서 ML용 GPU를 사용할 수 있는 다양한 방법을 제공합니다. 둘 중 하나를 선택하는 이유에 대해 자세히 알아보려면 [GPU 가속 ML 학습](#)을 참조하세요.

이 가이드는 다음을 설정하는 방법을 보여 줍니다.

- NVIDIA 그래픽 카드가 있고 샘플 ML 프레임워크 컨테이너를 실행하는 경우 NVIDIA CUDA
- AMD, Intel 또는 NVIDIA 그래픽 카드의 TensorFlow-DirectML 및 PyTorch-DirectML

## 사전 요구 사항

- [Windows 11](#) 또는 [Windows 10 버전 21H2](#) 이상을 실행 중인지 확인합니다.
- [WSL을 설치하고 Linux 배포용 사용자 이름과 암호를 설정합니다.](#)

## Docker로 NVIDIA CUDA 설정

1. [NVIDIA GPU용 최신 드라이버 다운로드 및 설치](#)
2. 다음 명령을 실행하여 Docker Desktop을 설치하거나 WSL에 직접 Docker 엔진을 설치합니다.

Bash

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

Bash

```
sudo service docker start
```

3. Docker 엔진을 직접 설치한 경우 아래 단계에 따라 [NVIDIA Container Toolkit](#)를 설치합니다.

다음 명령을 실행하여 NVIDIA Container Toolkit에 대한 안정적인 리포지토리를 설정합니다.

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
```

다음 명령을 실행하여 NVIDIA 런타임 패키지 및 종속성을 설치합니다.

Bash

```
sudo apt-get update
```

Bash

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

#### 4. 기계 학습 프레임워크 컨테이너 및 샘플을 실행합니다.

기계 학습 프레임워크 컨테이너를 실행하고 이 NVIDIA NGC TensorFlow 컨테이너에서 GPU 사용을 시작하려면 다음 명령을 입력합니다.

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# |
```

다음 명령을 실행하여 이 컨테이너에 기본 제공된 선행 학습된 모델 샘플을 실행할 수 있습니다.

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
```



NVIDIA CUDA를 설정하고 활용하는 추가 방법은 [WSL 사용자 가이드의 NVIDIA CUDA](#)에서 찾을 수 있습니다.

## TensorFlow-DirectML 또는 PyTorch-DirectML 설정

1. GPU 공급업체 웹 사이트([AMD](#), [Intel](#) 또는 [NVIDIA](#))에서 최신 드라이버를 다운로드하여 설치합니다.
2. Python 환경을 설정합니다.

가상 Python 환경을 설정하는 것이 좋습니다. 가상 Python 환경을 설정하는 데 사용할 수 있는 여러 도구가 있습니다. 이러한 지침에 대해서는 [Anaconda의 Miniconda](#)를 사용합니다.

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. 선택한 DirectML이 지원하는 기계 학습 프레임워크를 설치합니다.

TensorFlow-DirectML:

Bash

```
pip install tensorflow-directml
```

PyTorch-DirectML:

```
Bash
```

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

```
Bash
```

```
pip install pytorch-directml
```

4. [TensorFlow-DirectML](#) 또는 [PyTorch-DirectML](#)용 대화형 Python 세션에서 빠른 추가 샘플을 실행하여 모든 것이 제대로 작동하는지 확인합니다.

질문이 있거나 문제가 발생하면 [GitHub의 DirectML 리포지토리](#)를 방문합니다.

## 여러 GPU

컴퓨터에 여러 GPU가 있는 경우 WSL 내에서 액세스할 수도 있습니다. 그러나 한 번에 하나씩만 액세스할 수 있습니다. 특정 GPU를 선택하려면 아래 환경 변수를 디바이스 관리자에 표시되는 GPU 이름으로 설정하세요.

```
Bash
```

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

이렇게 하면 문자열 일치가 수행되므로 "NVIDIA"로 설정하면 "NVIDIA"로 시작하는 첫 번째 GPU와 일치합니다.

## 추가 리소스

- [WSL에서 NVIDIA CUDA 설정 지침](#)
- [WSL에서 DirectML로 TensorFlow를 설정하기 위한 지침](#)
- [DirectML 샘플이 포함된 TensorFlow](#)
- [WSL에서 DirectML로 PyTorch를 설정하기 위한 지침](#)
- [DirectML 샘플이 포함된 PyTorch](#)

# Linux용 Windows 하위 시스템 Linux GUI 앱 실행

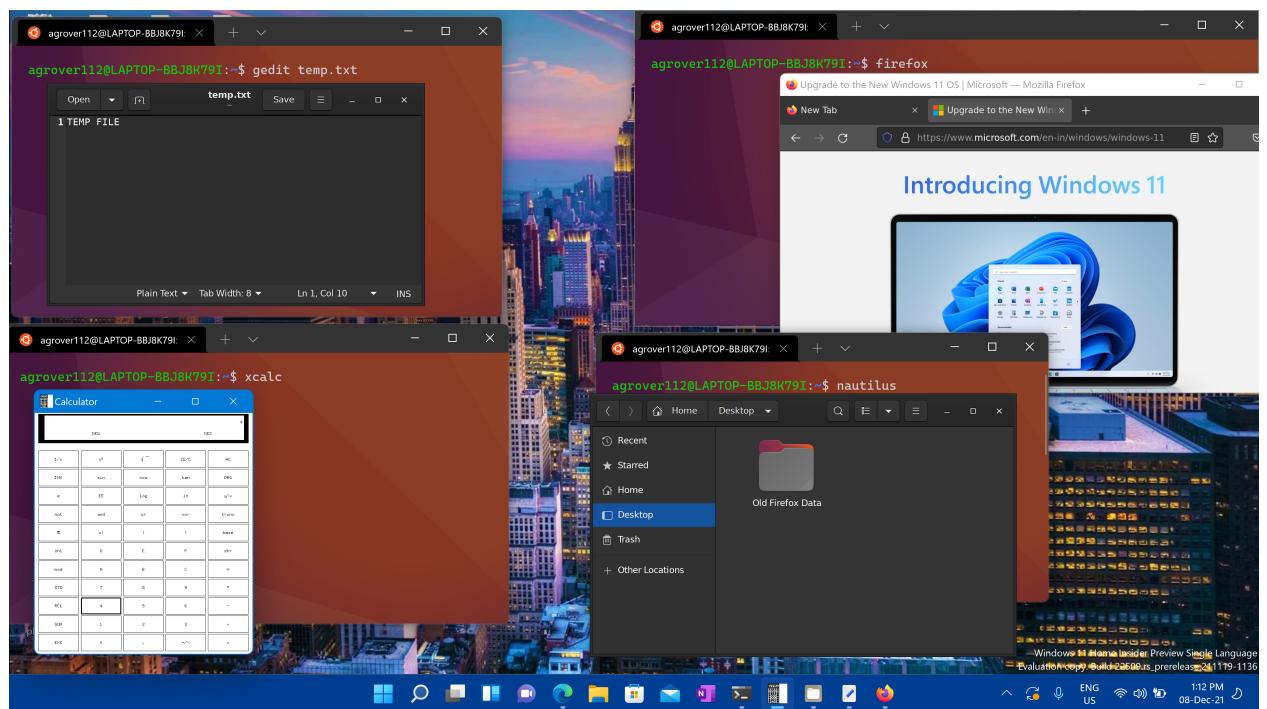
아티클 • 2023. 03. 21. • 읽는 데 7분 걸림

이제 WSL(Linux용 Windows 하위 시스템)은 완전히 통합된 데스크톱 환경에서 Windows에서 Linux GUI 애플리케이션(X11 및 Wayland) 실행을 지원합니다.

WSL 2를 사용하면 Linux GUI 애플리케이션이 Windows에서 사용하기에 기본적이고 자연스러운 느낌을 줍니다.

- Windows 시작 메뉴에서 Linux 앱 시작
- Windows 작업 표시줄에 Linux 앱 고정
- Alt-Tab을 사용하여 Linux와 Windows 앱 간 전환
- Windows 및 Linux 앱에서 잘라내기 + 붙여넣기

이제 원활한 데스크톱 환경을 위해 Windows 및 Linux 애플리케이션을 워크플로에 통합할 수 있습니다.



## Linux GUI 앱에 대한 설치 지원

### 사전 요구 사항

- 이 기능에 액세스하려면 Windows 10 Build 19044+ 또는 Windows 11이 필요합니다.

- vGPU용 설치된 드라이버

Linux GUI 앱을 실행하려면 먼저 아래 시스템과 일치하는 드라이버를 설치해야 합니다. 이렇게 하면 vGPU(가상 GPU)를 사용할 수 있으므로 하드웨어 가속 OpenGL 렌더링의 이점을 누릴 수 있습니다.

- [인텔 GPU 드라이버 ↗](#)
- [Amd GPU 드라이버 ↗](#)
- [엔비디아 GPU 드라이버 ↗](#)

## 새로 설치 - 이전 WSL 설치 없음

이제 관리자 PowerShell 또는 Windows 명령 프롬프트에서 이 명령을 입력한 다음, 머신을 다시 시작하여 WSL(Linux용 Windows 하위 시스템)을 실행하는 데 필요한 모든 것을 설치할 수 있습니다.

```
PowerShell  
wsl --install
```

컴퓨터 다시 부팅이 완료되면 설치가 계속되며 사용자 이름과 비밀번호를 입력하라는 메시지가 표시됩니다. Ubuntu 배포에 대한 Linux 자격 증명이 됩니다.

이제 WSL에서 Linux GUI 앱을 사용할 준비가 되었습니다.

자세한 내용은 [WSL 설치](#)를 참조하십시오.

## 기존 WSL 설치

컴퓨터에 WSL이 이미 설치된 경우 관리자 권한 명령 프롬프트에서 업데이트 명령을 실행하여 Linux GUI 지원을 포함하는 최신 버전으로 업데이트할 수 있습니다.

1. 시작을 클릭하고 PowerShell을 입력한 다음 Windows PowerShell을 마우스 오른쪽 단추로 클릭하고 관리자 권한으로 실행을 선택합니다.
2. WSL 업데이트 명령을 입력합니다.

```
PowerShell  
wsl --update
```

3. 업데이트를 적용하려면 WSL을 다시 시작해야 합니다. PowerShell에서 종료 명령을 실행하여 WSL을 다시 시작할 수 있습니다.

```
PowerShell
```

```
wsl --shutdown
```

### ① 참고

Linux GUI 앱은 WSL 2에서만 지원되며 WSL 1에 대해 구성된 Linux 배포판에서는 작동하지 않습니다. [배포를 WSL 1에서 WSL 2로 변경하는 방법](#)을 읽어보십시오.

## Linux GUI 앱 실행

Linux 터미널에서 다음 명령을 실행하여 인기 있는 Linux 애플리케이션을 다운로드하고 설치할 수 있습니다. Ubuntu와 다른 배포를 사용하는 경우, apt와 다른 패키지 관리자를 사용할 수 있습니다. Linux 애플리케이션이 설치되면 배포 이름 아래 **시작** 메뉴에서 찾을 수 있습니다. 예: Ubuntu -> Microsoft Edge

### ① 참고

WSL에서 GUI 앱에 대한 지원은 전체 데스크톱 환경을 제공하지 않습니다. Windows 데스크톱을 사용하므로 데스크톱 중심 도구 또는 앱 설치가 지원되지 않을 수 있습니다. 추가 지원을 요청하려면 [GitHub의 WSLg 리포지토리](#)로 문제를 제출할 수 있습니다.

## 배포에서 패키지 업데이트

```
Bash
```

```
sudo apt update
```

## Gedit 설치

Gedit은 GNOME 데스크톱 환경의 기본 텍스트 편집기입니다.

```
Bash
```

```
sudo apt install gedit -y
```

편집기에서 bashrc 파일을 시작하려면 `gedit ~/.bashrc` 을(를) 입력합니다.

## GIMP 설치

GIMP는 이미지 조작 및 이미지 편집, 자유 형식 그리기, 다양한 이미지 파일 형식 간의 코드 변환 및 보다 특수화된 작업에 사용되는 무료 오픈 소스 래스터 그래픽 편집기입니다.

Bash

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

시작하려면 `gimp`을(를) 입력합니다.

## Nautilus 설치

Nautilus(GNOME 파일이라고도 함)는 GNOME 데스크톱의 파일 관리자입니다. (Windows 파일 탐색기와 유사함)

Bash

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

시작하려면 `nautilus`을(를) 입력합니다.

## VLC 설치

VLC는 대부분의 멀티미디어 파일을 재생하는 무료 오픈 소스 플랫폼 간 멀티미디어 플레이어 및 프레임워크입니다.

Bash

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

시작하려면 `vlc`을(를) 입력합니다.

## X11 앱 설치

X11은 Linux 창 시스템이며 xclock, xcalc 계산기, 잘라내기 및 붙여넣기용 xclipboard, 이벤트 테스트용 xev 등과 함께 제공되는 앱 및 도구의 기타 컬렉션입니다. 자세한 내용은 [x.org 문서](#)를 참조하십시오.

Bash

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

시작하려면 사용하려는 도구의 이름을 입력합니다. 예를 들면 다음과 같습니다.

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

## Linux용 Google Chrome 설치

Linux용 Google Chrome을 설치하려면 다음을 수행합니다.

1. 디렉터리를 임시 폴더로 변경합니다. `cd /tmp`
2. wget을 사용하여 다운로드합니다. `sudo wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb`
3. 현재 안정적인 버전을 가져옵니다. `sudo dpkg -i google-chrome-stable_current_amd64.deb`
4. 패키지를 수정합니다. `sudo apt install --fix-broken -y`
5. 패키지를 구성합니다. `sudo dpkg -i google-chrome-stable_current_amd64.deb`

시작하려면 `google-chrome` 을(를) 입력합니다.

## Linux용 Microsoft Edge 브라우저 설치

Edge Insider 사이트의 명령줄을 사용하여 Linux용 Microsoft Edge 브라우저를 설치하는 방법 [방법](#)에 대한 정보를 찾습니다. 페이지의 명령줄 설치 섹션에서 지침 가져오기를 선택합니다.

시작하려면 `microsoft-edge` 을(를) 입력합니다.

## 문제 해결

GUI 애플리케이션을 시작하는 데 문제가 있는 경우 먼저 이 가이드를 확인하십시오.

[WSLg에서 "디스플레이를 열 수 없음" 유형 문제 진단](#)

# Linux용 Windows 하위 시스템의 Node.js 설치(WSL2)

아티클 • 2022. 09. 24. • 읽는 데 17분 걸림

Node.js에 능숙하거나, 성능 속도와 시스템 호출 호환성이 중요하거나, Linux 작업 영역을 활용하는 [Docker 컨테이너](#)를 실행하여 Linux 및 Windows 빌드 스크립트 모두 유지 관리하지 않으려고 하거나, Bash 명령줄만 사용하려고 하면 Linux용 Windows 하위 시스템(구체적으로는 WSL 2)에 Node.js를 설치합니다.

WSL(Linux 용 Windows 하위 시스템)을 사용하면 원하는 Linux 배포(Ubuntu가 기본)를 설치할 수 있으므로 코드를 작성하는 개발 환경과 코드가 배포된 서버인 프로덕션 환경 사이에서 일관성을 유지할 수 있습니다.

## ① 참고

Node.js를 사용하는 개발이 처음이고 빠르게 배워 시작하고 실행하려면 [Node.js를 직접 Windows에 설치합니다](#). 이 권장 사항은 Windows Server 프로덕션 환경을 사용하려는 경우에도 적용됩니다.

## WSL 2 설치

WSL 2는 Windows에서 사용할 수 있는 최신 버전이며 전문 Node.js 개발 워크플로에 권장됩니다. WSL 2를 사용하고 설치하려면 [WSL 설치 문서](#)의 단계를 따르세요. 이 단계에는 Linux 배포판(예: Ubuntu) 선택이 포함됩니다.

WSL 2와 Linux 배포판을 설치했으면 Linux 배포판(Windows 시작 메뉴에서 찾을 수 있음)을 열고 `lsb_release -dc` 명령을 사용하여 버전과 코드 이름을 확인합니다.

최신 패키지를 유지하기 위해 설치 직후를 포함하여 Linux 배포를 정기적으로 업데이트하는 것이 좋습니다. 이 업데이트는 Windows에서 자동으로 처리하지 않습니다. 배포를 업데이트하려면 `sudo apt update && sudo apt upgrade` 명령을 사용합니다.

## Windows 터미널 설치(선택 사항)

Windows 터미널은 Linux 명령줄, Windows 명령 프롬프트, PowerShell, Azure CLI 또는 사용하려는 탭으로 빠르고 전환할 수 있도록 탭 여러 개를 실행할 수 있게 해주는 향상된 명령줄 셸입니다. 사용자 지정 키 바인딩(탭 열기 또는 닫기, 복사+붙여 넣기 등의 바로 가기 키) 만들기, 검색 기능 사용, 테마(색 구성표, 글꼴 스타일 및 크기, 배경 이미지/흐림/투명

도)로 터미널 사용자 지정 등을 수행할 수도 있습니다. [Windows 터미널 문서에서 자세히 알아보세요.](#)

[Microsoft Store를 사용하여 Windows 터미널 설치](#): Store를 통해 설치하면 업데이트가 자동으로 처리됩니다.

## nvm, node.js 및 npm 설치

Node.js를 설치할 때 Windows 또는 WSL에 설치 여부 외에도 선택해야 할 사항이 더 있습니다. 버전이 매우 빠르게 바뀌므로 버전 관리자를 사용하는 것이 좋습니다. 작업 중인 서로 다른 프로젝트의 요구 사항에 따라 여러 Node.js 버전으로 전환해야 할 수도 있습니다. 흔히 nvm으로 불리는 노드 버전 관리자는 여러 버전의 Node.js를 설치하는 가장 인기 있는 방법입니다. nvm을 설치하는 단계와 nvm을 사용하여 Node.js 및 npm(노드 패키지 관리자)을 설치하는 단계를 살펴보겠습니다. 생각해 볼 수 있는 [또 다른 버전 관리자](#)도 있으며, 다음 섹션에서 설명하겠습니다.

### ⓘ 중요

여러 유형을 설치하면 비정상적이고 혼란스러운 충돌이 발생할 수 있으므로, 항상 운영 체제에 설치된 기존 Node.js 또는 npm을 제거한 후 버전 관리자를 설치하는 것이 좋습니다. 예를 들어 Ubuntu의 `apt-get` 명령을 사용하여 설치할 수 있는 노드 버전은 오래된 버전입니다. 이전 설치를 제거하는 방법에 대한 도움말은 [ubuntu에서 nodejs를 제거하는 방법](#)을 참조하세요.

1. Ubuntu 명령줄(또는 선택한 배포판)을 엽니다.
2. `sudo apt-get install curl` 명령을 사용하여 cURL(명령줄을 사용하여 인터넷에서 콘텐츠를 다운로드하는 데 사용되는 도구)을 설치합니다.
3. `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/master/install.sh | bash` 명령을 사용하여 nvm을 설치합니다.

### ⓘ 참고

cURL을 사용하여 최신 버전의 NVM을 설치하면 이전 버전이 대체되고, NVM을 사용하여 설치한 노드 버전은 그대로 유지됩니다. 자세한 내용은 [NVM에 대한 최신 릴리스 정보를 제공하는 GitHub 프로젝트 페이지](#)를 참조하세요.

4. 설치를 확인하려면 `command -v nvm` 명령을 입력합니다. 그러면 'nvm'이 반환됩니다. '명령을 찾을 수 없음' 메시지가 수신되거나 응답이 없으면 현재 터미널을 닫았다가

열고 다시 시도해 보세요. [nvm github](#) 리포지토리에서 자세한 정보를 알아보세요 ↴.

5. `nvm ls` 명령을 사용하여 현재 설치된 노드 버전을 나열합니다.

```
mattwojo@MININT-LOBGCR8: ~
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. Node.js의 현재 버전과 안정적인 LTS 버전을 모두 설치합니다. 이후 단계에서는 `nvm` 명령을 사용하여 Node.js의 활성 버전 간에 전환하는 방법을 알아봅니다.

- Node.js의 현재 안정적인 LTS 릴리스를 설치합니다(프로덕션 애플리케이션에 권장). `nvm install --lts`
- Node.js의 현재 릴리스를 설치합니다(최신 Node.js 기능 및 개선 사항을 테스트하지만 문제가 발생할 가능성이 높음). `nvm install node`

7. `nvm ls`를 사용하여 설치된 노드 버전을 나열합니다. 방금 설치한 두 가지 버전이 표시될 것입니다.

```
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. `node --version` 명령을 사용하여 Node.js가 설치되어 있는지 여부 및 현재 기본 버전을 확인합니다. 그리고 `npm --version` 명령을 사용하여 npm이 설치되어 있는지 확인합니다(`which node` 또는 `which npm` 명령을 사용하여 기본 버전에 사용되는 경로도 확인 가능).

9. 프로젝트에 사용할 Node.js 버전을 변경하려면 새 프로젝트 디렉터리 `mkdir NodeTest`를 만들고 `cd NodeTest` 디렉터리로 들어간 다음, `nvm use node`를 입력하여 현재 버전으로 전환하거나 `nvm use --lts`를 입력하여 LTS 버전으로 전환합니다. `nvm use v8.2.1`처럼 설치한 버전의 특정 번호를 사용할 수도 있습니다. (사용 가능한 모든 Node.js 버전을 나열하려면 `nvm ls-remote` 명령을 사용합니다.)

NVM을 사용하여 Node.js 및 NPM을 설치하는 경우 SUDO 명령을 사용하여 새 패키지를 설치할 필요가 없습니다.

## 대체 버전 관리자

현재 노드에 가장 많이 사용되는 버전 관리자는 nvm이지만, 고려해 볼 수 있는 다음과 같은 다른 버전 관리자도 있습니다.

- 오래전부터 nvm 대용으로 사용된 [n](#) 은 약간 다른 명령으로 동일한 작업을 수행하며, bash 스크립트 대신 npm을 통해 설치됩니다.
- fnm [은](#) 최신 버전의 관리자이며 nvm보다 훨씬 빠르다고 합니다. (마찬가지로 Azure Pipelines를 사용합니다.)
- Volta [는](#) LinkedIn 팀의 새로운 버전 관리자로, 속도 및 플랫폼 간 지원이 향상되었다고 합니다.

- [asdf-vm](#) 은 gvm, nvm, rbenv & pyenv 등의 여러 언어를 모두 관리할 수 있는 단일 CLI입니다.
- [nvs](#) (노드 버전 전환기)는 플랫폼 간 `nvm` 대안으로, VS Code와 통합할 수 있는 기능을 제공합니다.

## Visual Studio Code 설치

Node.js 프로젝트에 원격-개발 확장 팩과 함께 Visual Studio Code를 사용하는 것이 좋습니다. 이렇게 하면 VS Code가 “클라이언트-서버” 아키텍처로 분할되고, 클라이언트(VS Code 사용자 인터페이스)는 Windows 운영 체제에서 실행되고, 서버(코드, Git, 플러그인 등)는 WSL Linux 배포판에서 “원격”으로 실행됩니다.

### ① 참고

이 “원격” 시나리오는 여러분에게 익숙한 시나리오와 약간 다릅니다. WSL은 프로젝트 코드가 Windows 운영 체제와 별도로 실행되지만 여전히 로컬 컴퓨터에 있는 실제 Linux 배포판을 지원합니다. 원격 WSL 확장은 클라우드에서 실행되지 않지만 마치 원격 서버인 것처럼 Linux 하위 시스템에 연결합니다. Windows와 함께 실행되도록 설정한 WSL 환경의 로컬 컴퓨터에서 계속 실행됩니다.

- Linux 기반 Intellisense 및 린트가 지원됩니다.
- 프로젝트는 Linux에서 자동으로 빌드됩니다.
- Linux에서 실행되는 모든 확장([ES Lint, NPM Intellisense, ES6 코드 조각 등](#))을 사용할 수 있습니다.

IntelliJ, Sublime Text, Brackets 등 다른 코드 편집기는 WSL 2 Node.js 개발 환경에서도 작동하지만 VS Code에서 제공하는 원격 기능을 제공하지 않을 수 있습니다. 이 코드 편집기에서는 WSL 공유 네트워크 위치(\wsl\$\Ubuntu\home)에 액세스하는 중에 문제가 발생길 수 있으며 사용하지 않으려는 Windows 도구를 사용하여 Linux 파일을 빌드하려고 합니다. VS Code의 원격-WSL 확장은 X 서버를 설정하는 데 필요할 수 있는 다른 IDE와 함께 이 호환성 문제를 자동으로 처리합니다. [WSL에서 GUI 앱 실행 지원](#)(예: 코드 편집기 IDE)이 곧 제공될 예정입니다.

터미널 기반 텍스트 편집기(vim, emacs, nano)는 콘솔 내에서 바로 신속하게 변경할 때도 유용합니다. [Emacs, Nano 또는 Vim: 터미널 기반 텍스트 편집기 선택](#) 문서에서는 각 편집기의 차이점과 사용 방법을 설명합니다.

VS Code 및 원격 WSL 확장을 설치하는 방법은 다음과 같습니다.

1. [Windows용 VS Code를 다운로드하여 설치합니다](#). VS Code는 Linux에서도 사용 할 수 있지만, Linux용 Windows 하위 시스템은 GUI 앱을 지원하지 않으므로

Windows에 설치해야 합니다. 걱정하지 마세요. 여전히 Remote - WSL 확장을 사용하여 Linux 명령줄 및 도구와 통합할 수 있습니다.

2. [Remote - WSL 확장](#)을 VS Code에 설치합니다. 이를 통해 WSL을 통합 개발 환경으로 사용하고, 호환성과 패치를 처리할 수 있습니다. [자세한 정보를 알아보세요](#).

### ① 중요

VS Code가 이미 설치되어 있는 경우 [Remote - WSL 확장](#)을 설치하려면 [1.35 5월 릴리스](#) 이상이어야 합니다. 자동 완성, 디버깅, linting 등의 지원이 손실될 수 있으므로 VS Code에서 Remote - WSL 확장 없이 WSL을 사용하지 않는 것이 좋습니다. 재미있게도 이 WSL 확장은 \$HOME/.vscode-server/extensions에 설치됩니다.

## 유용한 VS Code 확장

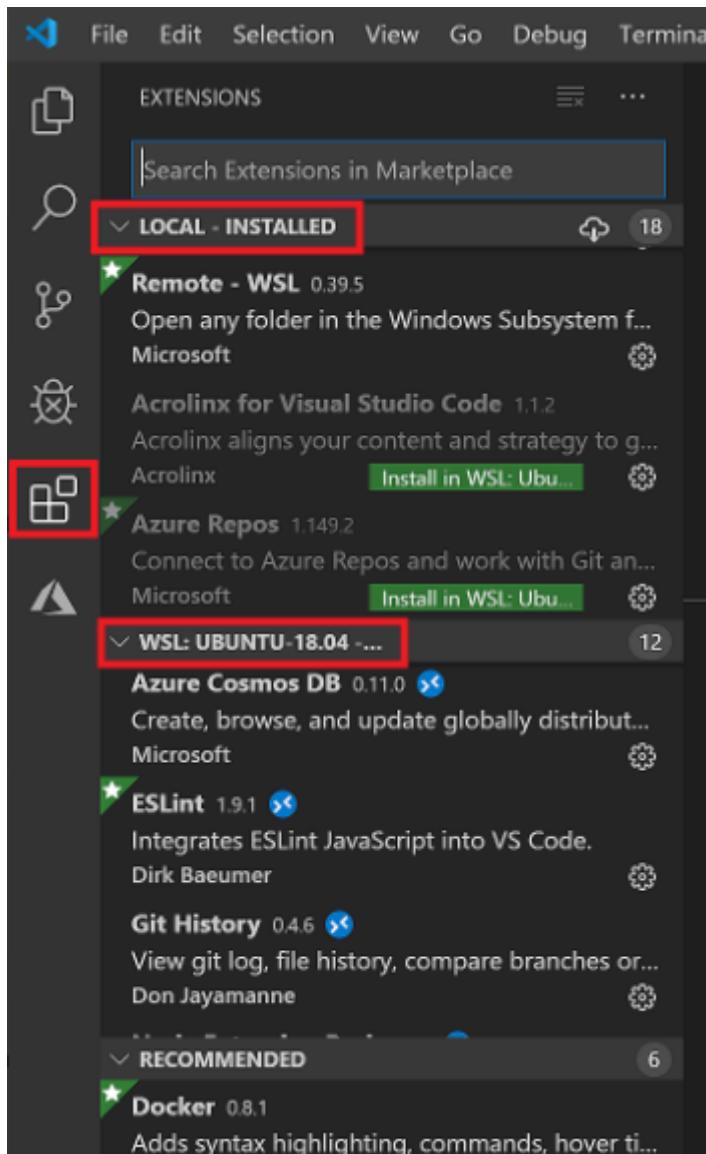
VS Code는 기본적으로 Node.js 개발을 위한 여러 기능과 함께 제공되지만, [Node.js 확장 팩](#)에 제공되는 확장 중에도 설치를 고려해 볼 수 있는 유용한 확장이 몇 개 있습니다. 전부 설치해도 되고 가장 유용하다고 생각되는 부분만 선택해서 설치해도 됩니다.

Node.js 확장 팩을 설치하는 방법은 다음과 같습니다.

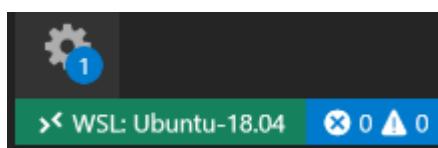
1. VS Code에서 **확장** 창을 엽니다(Ctrl+Shift+X).

이제 확장 창은 다음과 같은 세 가지 섹션으로 구분됩니다(원격 WSL 확장을 설치했기 때문에).

- "로컬 - 설치": Windows 운영 체제에서 사용하기 위해 설치된 확장입니다.
- "WSL:Ubuntu-18.04-설치": Ubuntu 운영 체제(WSL)에 사용하기 위해 설치된 확장입니다.
- "권장": 현재 프로젝트의 파일 형식에 따라 VS Code에서 권장하는 확장입니다.



2. [확장] 창의 맨 위에 있는 검색 상자에 **노드 확장 팩**(또는 찾고 있는 확장의 이름)을 입력합니다. 현재 프로젝트가 열려 있는 위치에 따라 VS Code의 로컬 또는 WSL 인스턴스에 대한 확장이 설치됩니다. VS Code 창의 왼쪽 아래 모서리에 있는 원격 링크(녹색)를 선택하여 알 수 있습니다. 원격 연결을 열거나 닫는 옵션이 제공됩니다. "WSL:Ubuntu-18.04" 환경에 Node.js 확장을 설치합니다.



다음과 같은 몇 가지 추가 확장도 고려해 볼 수 있습니다.

- [JavaScript 디버거](#): Node.js를 사용하여 서버 쪽에서 개발을 마친 후에는 클라이언트 쪽에서 개발하고 테스트해야 합니다. 이 확장은 DAP 기반 JavaScript 디버거입니다. Node.js, Chrome, Edge, WebView2, VS Code 확장 등을 디버그합니다.
- [다른 편집기의 키맵](#): Atom, Sublime, Vim, eMacs, 메모장++ 등의 다른 텍스트 편집기에서 전환할 때 이러한 확장을 사용하여 익숙한 환경을 만들 수 있습니다.

- [설정 동기화](#) : GitHub를 사용하는 여러 설치에서 VS Code 설정을 동기화할 수 있습니다. 여러 머신에서 작업하는 경우 이렇게 하면 여러 머신의 환경을 일관되게 유지할 수 있습니다.

## Git 설치(선택 사항)

WSL에서 Node.js 프로젝트를 위해 Git을 설정하려면 WSL 문서에서 [Linux용 Windows 하위 시스템에 Git 사용 시작](#) 문서를 참조하세요.

# Linux 및 Bash 시작하기

아티클 • 2023. 03. 21. • 읽는 데 7분 걸림

이 자습서는 Linux를 새로 사용하는 사용자가 기본적으로 WSL을 사용하여 설치되는 Linux의 Ubuntu 배포를 사용하여 패키지 설치 및 업데이트를 시작하고 Bash 명령줄에서 몇 가지 기본 명령을 사용하는 데 도움이 됩니다.

## 소프트웨어 설치 및 업데이트

실행 중인 배포에 대한 기본 패키지 관리자를 사용하여 명령줄에서 직접 소프트웨어 프로그램을 설치하고 업데이트할 수 있습니다.

예를 들어, Ubuntu에서 먼저 'sudo apt update'를 실행하여 사용 가능한 소프트웨어 목록을 업데이트합니다. 그런 다음, 'sudo apt-get install' 명령과 설치하려는 프로그램의 이름을 사용하여 소프트웨어를 직접 가져올 수 있습니다.

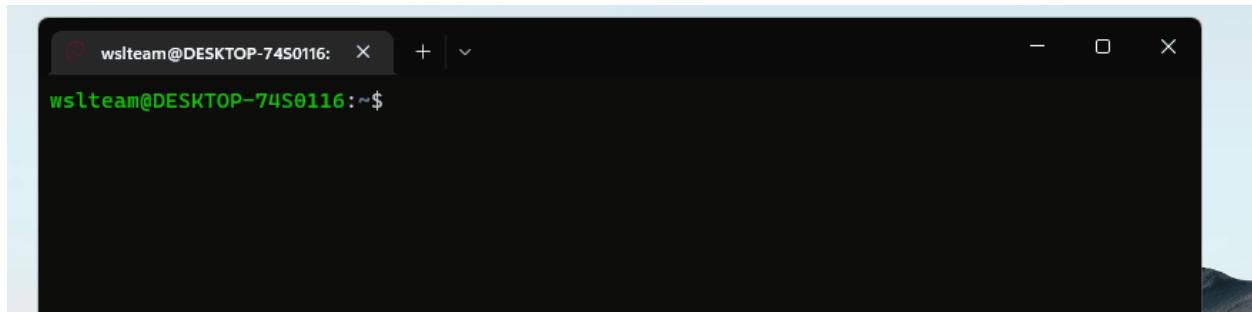
Bash

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

이미 설치된 프로그램을 업데이트하려면 다음을 실행할 수 있습니다.

Bash

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



### 💡 팁

Linux의 다양한 배포에서는 패키지 관리자가 다른 경우가 있기 때문에 연결된 패키지 관리자와 관련된 설치 명령을 사용해야 합니다. 예를 들어, Arch Linux의 기본 패키지 관리자를 **pacman** 이라고 하며, 설치 명령은 `sudo pacman -S <app_name>`입니다. 예를 들어, OpenSuse의 기본 패키지 관리자를 **Zypper** 이라고 하며, 설치 명령은

`sudo zypper install <app_name>` 입니다. 예를 들어, Alpine의 기본 패키지 관리자를 **apk**라고 하며, 설치 명령은 `sudo apk add <app_name>` 입니다. CentOS와 같은 Red Hat 배포판의 두 가지 주요 패키지 관리자는 **YUM**과 **RPM**이며 설치 명령은 `sudo yum install <app_name>` 또는 `sudo rpm -i <app_name>` 일 수 있습니다. 소프트웨어를 설치하고 업데이트하는 데 사용할 수 있는 도구를 알아보려면 작업 중인 배포 설명서를 참조하십시오.

## 파일 및 디렉터리 작업

현재 있는 디렉터리의 경로를 보려면 'pwd' 명령을 사용합니다.

Bash

```
pwd
```

새 디렉터리를 만들려면 'mkdir' 명령 뒤에 만들려는 디렉터리의 이름을 사용합니다.

Bash

```
mkdir hello_world
```

디렉터리를 변경하려면 'cd' 명령 뒤에 이동하려는 디렉터리의 이름을 사용합니다.

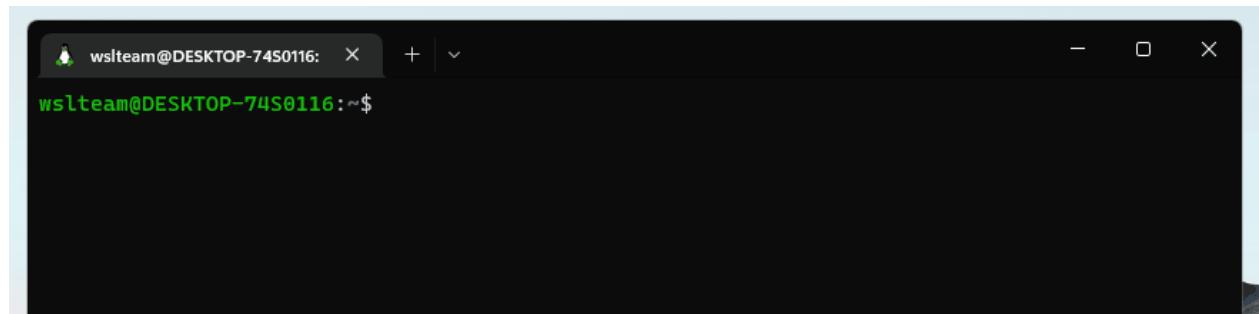
Bash

```
cd hello_world
```

현재 있는 디렉터리 내의 내용을 보려면 명령줄에 'ls'를 입력합니다.

Bash

```
ls
```



기본적으로 'ls' 명령은 모든 파일 및 디렉터리의 이름만 출력합니다. 파일이 마지막으로 수정된 시간 또는 파일 권한과 같은 추가 정보를 얻으려면 플래그 "-l"을 사용합니다.

```
Bash
```

```
ls -l
```

'touch' 명령과 만들려는 파일의 이름을 통해 새 파일을 만들 수 있습니다.

```
Bash
```

```
touch hello_world.txt
```

다운로드한 그래픽 텍스트 편집기 또는 VS Code Remote – WSL 확장을 사용하여 파일을 편집할 수 있습니다. [여기](#)에서 VS Code 시작하기에 대해 자세히 알아볼 수 있습니다.

명령줄에서 직접 파일을 편집하려는 경우, vim, emacs 또는 nano와 같은 명령줄 편집기를 사용해야 합니다. 많은 배포판에는 이러한 프로그램 중 하나 이상이 설치되어 있지만, [위](#)의 가이드에 설명한 설치 지침에 따라 항상 이러한 프로그램을 설치할 수 있습니다.

원하는 편집 방법으로 파일을 편집하려면 프로그램 이름 뒤에 편집하려는 파일 이름을 실행하면 됩니다.

```
Bash
```

```
code hello_world.txt
```

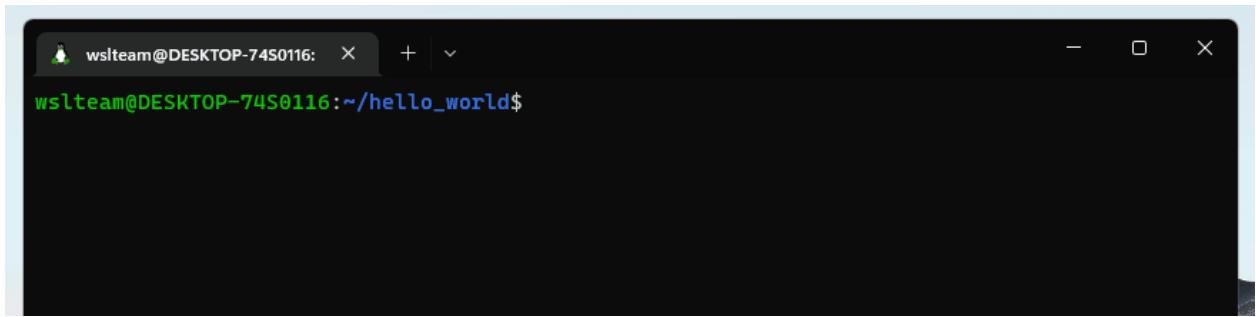
```
Bash
```

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

명령줄에서 파일의 내용을 보려면 'cat' 명령과 읽고 싶은 파일을 사용합니다.

```
Bash
```

```
cat hello_world.txt
```

A screenshot of a Windows Subsystem for Linux (WSL) terminal window titled "wslteam@DESKTOP-74S0116: ~". The window shows a black terminal screen with the prompt "wslteam@DESKTOP-74S0116:~/hello\_world\$".

wslteam@DESKTOP-74S0116:~/hello\_world\$

☞

## 파이프 및 리디렉션 연산자 사용

파이프 '|'은 한 명령의 출력을 입력으로 다른 명령으로 리디렉션합니다. 예를 들어, lhscmd | rhscmd는 출력을 lhscmd에서 rhscmd로 전달합니다. 파이프는 명령줄을 통해 작업을 신속하게 수행하는 다양한 방법으로 사용할 수 있습니다. 다음은 파이프를 사용하는 방법에 대한 몇 가지 간단한 예입니다.

파일의 내용을 빠르게 정렬하려는 경우를 상상해 보십시오. 아래 fruits.txt 예제를 참조하십시오.

Bash

```
cat fruits.txt
Orange
Banana
Apple
Pear
Plum
Kiwi
Strawberry
Peach
```

파이프를 사용하여 이 목록을 빠르게 정렬할 수 있습니다.

Bash

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

```
Banana
```

```
Kiwi
```

```
Orange
```

```
Peach
```

```
Pear
```

```
Plum
```

```
Strawberry
```

기본적으로 'cat' 명령의 출력은 표준 출력으로 전송됩니다. 그러나, '|'을 사용하면 출력을 다른 명령 'sort' 입력으로 리디렉션할 수 있습니다.

다른 사용 사례는 검색입니다. 특정 검색 문자열에 대한 입력을 검색하는 데 유용한 명령인 'grep'를 사용할 수 있습니다.

```
Bash
```

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

```
Pear
```

```
Plum
```

```
Peach
```

'>'와(과) 같은 리디렉션 연산자를 사용하여 출력을 파일 또는 스트림에 전달할 수도 있습니다. 예를 들어, fruit.txt의 정렬된 내용으로 새 .txt 파일을 만들려면 다음을 수행합니다.

```
Bash
```

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

```
Bash
```

```
$ cat sorted_fruit.txt
```

```
Apple
```

```
Banana
```

```
Kiwi
```

```
Orange
```

Peach

Pear

Plum

Strawberry

기본적으로, 정렬 명령의 출력은 표준 출력으로 전송됩니다. 그러나, '>' 연산자를 사용하면 출력을 sorted\_fruits.txt 라는 새 파일로 리디렉션할 수 있습니다.

여러 가지 흥미로운 방법으로 파이프 및 리디렉션 연산자를 사용하여 명령줄에서 직접 작업을 보다 효율적으로 완료할 수 있습니다.

## 권장 콘텐츠

- Microsoft Learn: Bash 소개
- 초보자를 위한 명령줄 ↗
- Microsoft Learn: WSL 시작하기

# Windows 및 Linux 파일 시스템 간 작업

아티클 • 2023. 03. 21. • 읽는 데 14분 걸림

Windows 및 Linux 파일 시스템 간에 작업할 때 기억해야 할 여러 가지 고려 사항이 있습니다. 이 가이드에서는 Windows 및 Linux 기반 명령을 혼합하기 위한 상호 운용성 지원의 몇 가지 예제를 포함하여 몇 가지 고려 사항이 설명되어 있습니다.

## 파일 시스템 전체의 파일 스토리지 및 성능

이를 수행하는 특별한 이유가 없으면 운영 체제 간에 작업하지 않는 것이 좋습니다. Linux 명령줄(Ubuntu, OpenSUSE 등)에서 작업하는 경우 가장 빠른 성능을 달성하려면 파일을 WSL 파일 시스템에 저장합니다. Windows 명령줄(PowerShell, 명령 프롬프트)에서 작업하는 경우 파일을 Windows 파일 시스템에 저장합니다.

예를 들어 WSL 프로젝트 파일을 저장하는 경우 다음과 같습니다.

- Linux 파일 시스템 루트 디렉터리(`\\\wsl$\Ubuntu\home\<user name>\Project`)를 사용 합니다.
- Windows 파일 시스템 루트 디렉터리를 사용하지 않는 경우: `/mnt/c/Users/<user name>/Project$` 또는 `C:\Users\<user name>\Project`

WSL 명령줄의 파일 경로에 `/mnt/`가 보이면 현재 탑재된 드라이브에서 작업하고 있는 것입니다. 따라서 Windows 파일 시스템 C:/ 드라이브(`C:\Users\<user name>\Project`)는 WSL 명령줄에 탑재될 때 `/mnt/c/Users/<user name>/Project$`와 같이 표시됩니다. 탑재된 드라이브에 프로젝트 파일을 저장할 수 있지만, `\\\wsl$` 드라이브에 직접 저장하면 성능 속도가 향상됩니다.

## Windows 파일 탐색기에서 현재 디렉터리 보기

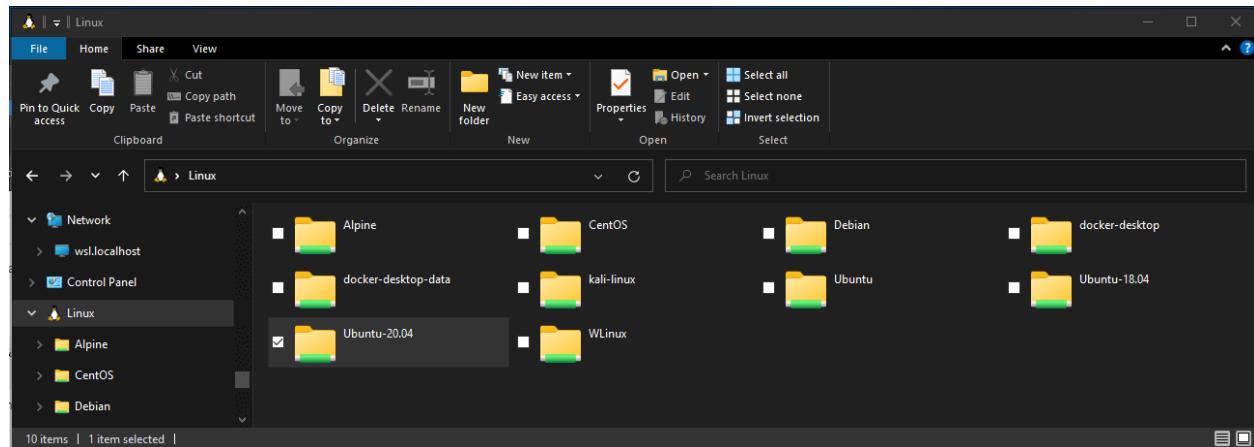
다음 명령을 사용하여 명령줄에서 Windows 파일 탐색기를 열면 파일이 저장되는 디렉터리를 볼 수 있습니다.

Bash

```
explorer.exe .
```

또는 `powershell.exe /c start .` 명령을 사용할 수도 있습니다. 현재 디렉터리를 열려면 명령 끝에 마침표를 추가해야 합니다.

Windows 파일 탐색기에서 사용 가능한 모든 Linux 배포판 및 해당 루트 파일 시스템을 보려면 주소 표시줄에 `\wsl$`를 입력합니다.



## 파일 이름 및 디렉터리 대/소문자 구분

대/소문자 구분은 대문자(FOO.txt) 및 소문자(foo.txt) 문자를 파일 이름이나 디렉터리에서 서로 다르게 취급할 것인지(대/소문자 구분) 아니면 똑같은 것으로 취급할 것인지(대/소문자 구분 안 함) 여부를 결정합니다. Windows 파일 시스템과 Linux 파일 시스템은 대/소문자 구분을 서로 다른 방식으로 처리합니다. Windows는 대/소문자를 구분하지 않는 반면 Linux는 대/소문자를 구분합니다. [대/소문자 구분 조정](#) 방법 문서에서 특히 WSL을 사용하여 디스크를 탑재할 때 대/소문자 구분을 조정하는 방법에 대해 자세히 알아보세요.

## Windows 명령과 Linux 명령 간의 상호 운용성

Windows와 Linux의 도구 및 명령은 WSL과 교환하여 사용할 수 있습니다.

- Linux 명령줄(즉, Ubuntu)에서 Windows 도구(즉, notepad.exe)를 실행합니다.
- Windows 명령줄(즉, PowerShell)에서 Linux 도구(즉, grep)를 실행합니다.
- Linux와 Windows 간에 환경 변수를 공유합니다. (빌드 17063 이상)

## Windows 명령줄에서 Linux 도구 실행

`wsl <command>`(또는 `wsl.exe <command>`)를 사용하여 CMD(Windows 명령 프롬프트) 또는 PowerShell에서 Linux 이진 파일을 실행합니다.

예:

```
PowerShell

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

이진 파일은 다음과 같은 방식으로 호출됩니다.

- 현재 CMD 또는 PowerShell 프롬프트와 동일한 작업 디렉터리를 사용합니다.
- WSL 기본 사용자로 실행합니다.
- 호출 프로세스 및 터미널과 동일한 Windows 관리 권한을 사용합니다.

wsl(또는 wsl.exe) 뒤의 Linux 명령은 WSL에서 실행되는 명령처럼 처리됩니다. sudo, 파이핑, 파일 리디렉션과 같은 작업이 작동합니다.

sudo를 사용하여 기본 Linux 배포를 업데이트하는 예제:

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

이 명령을 실행하면 기본 Linux 배포 사용자 이름이 나열되고 암호를 입력하라는 메시지가 표시됩니다. 암호가 올바르게 입력되면 배포에서 업데이트를 다운로드합니다.

## Linux 및 Windows 명령 혼합

PowerShell을 사용하여 Linux와 Windows 명령을 혼합하는 몇 가지 예제는 다음과 같습니다.

`ls -la` Linux 명령을 사용하여 파일을 나열하고 `findstr` PowerShell 명령을 사용하여 "git"이 포함된 단어에 대한 결과를 필터링하려면 명령을 다음과 같이 결합합니다.

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

`dir` PowerShell 명령을 사용하여 파일을 나열하고 `grep` Linux 명령을 사용하여 "git"이 포함된 단어에 대한 결과를 필터링하려면 명령을 다음과 같이 결합합니다.

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

`ls -la` Linux 명령을 사용하여 파일을 나열하고 `> out.txt` PowerShell 명령을 사용하여 해당 목록을 "out.txt"라는 텍스트 파일로 출력하려면 명령을 다음과 같이 결합합니다.

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

`wsl.exe`에 전달된 명령은 수정되지 않고 WSL 프로세스에 전달됩니다. 파일 경로는 WSL 형식으로 지정해야 합니다.

`ls -la` Linux 명령을 사용하여 `/proc/cpuinfo` Linux 파일 시스템 경로에 있는 파일을 나열하려면 PowerShell을 다음과 같이 사용합니다.

PowerShell

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

`ls -la` Linux 명령을 사용하여 `C:\Program Files` Windows 파일 시스템 경로에 있는 파일을 나열하려면 PowerShell을 다음과 같이 사용합니다.

PowerShell

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

## Linux에서 Windows 도구 실행

WSL에서 `[tool-name].exe`를 사용하여 WSL 명령줄에서 Windows 도구를 직접 실행할 수 있습니다. 정의합니다(예: `notepad.exe`).

이 방식으로 실행되는 애플리케이션에는 다음과 같은 속성이 있습니다.

- 작업 디렉터리를 WSL 명령 프롬프트로 유지합니다(대부분의 경우에 해당, 예외는 아래에 설명되어 있음).
- WSL 프로세스와 동일한 권한을 갖습니다.
- 활성 Windows 사용자로 실행합니다.
- CMD 프롬프트에서 직접 실행한 것처럼 Windows 작업 관리자에 표시됩니다.

WSL에서 실행되는 Windows 실행 파일은 네이티브 Linux 실행 파일과 비슷하게 처리됩니다(파이핑, 리디렉션 및 백그라운드 작업이 예상대로 작동).

`ipconfig.exe` Windows 도구를 실행하려면 `grep` Linux 도구를 사용하여 "IPv4" 결과를 필터링하고 `cut` Linux 도구를 사용하여 Linux 배포(예: Ubuntu)에서 열 필드를 제거합니다.

Bash

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

Windows와 Linux 명령이 혼합된 예제를 사용해 보겠습니다. Linux 배포(즉, Ubuntu)를 열고, `touch foo.txt`라는 텍스트 파일을 만듭니다. 이제 `ls -la` Linux 명령을 사용하여 파

일 및 해당 만들기 세부 정보를 직접 나열하고 `findstr.exe` Windows PowerShell 도구를 사용하여 결과를 필터링하여 `foo.txt` 파일만 결과에 표시합니다.

Bash

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

Windows 도구는 파일 확장명을 포함하고, 파일 대/소문자와 일치하며, 실행 파일이어야 합니다. 일괄 처리 스크립트가 포함된 비실행 파일이 있습니다. `dir`과 같은 CMD 기본 명령은 `cmd.exe /c` 명령을 사용하여 실행할 수 있습니다.

예를 들어 다음을 입력하여 Windows 파일 시스템의 C:\ 디렉터리에 있는 콘텐츠를 나열합니다.

Bash

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

또는 `ping` 명령을 사용하여 예코 요청을 microsoft.com 웹 사이트에 보냅니다.

Bash

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

매개 변수는 수정되지 않은 Windows 이진 파일에 전달됩니다. 예를 들어 다음 명령은 `notepad.exe`에서 `C:\temp\foo.txt`를 엽니다.

Bash

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

또한 다음과 같은 작업도 수행합니다.

Bash

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

## WSLENV를 사용하여 Windows와 WSL 간에 환경 변수 공유

WSL 및 Windows는 WSL에서 실행되는 Windows 및 Linux 배포를 연결하기 위해 만든 `WSLENV`라는 특수 환경 변수를 공유합니다.

`WSLENV` 변수의 속성은 다음과 같습니다.

- 공유되며, Windows 및 WSL 환경 모두에 있습니다.
- Windows와 WSL 간에 공유할 환경 변수의 목록입니다.
- Windows 및 WSL에서 제대로 작동하도록 환경 변수의 형식을 지정할 수 있습니다.
- WSL과 Win32 간의 흐름을 지원할 수 있습니다.

#### ① 참고

17063 이전에서는 `PATH`만 WSL에서 액세스할 수 있는 Windows 환경 변수였습니다 (이를 통해 WSL 아래에서 Win32 실행 파일을 시작할 수 있었음). `WSLENV`는 17063부터 지원됩니다. `WSLENV`는 대/소문자를 구분합니다.

## WSLENV 플래그

`WSLENV`에서 4개의 플래그를 사용하여 환경 변수가 변환되는 방법에 영향을 줄 수 있습니다.

`WSLENV` 플래그:

- `/p` - WSL/Linux 스타일 경로 및 Win32 경로 간의 경로를 변환합니다.
- `/l` - 환경 변수가 경로 목록임을 나타냅니다.
- `/u` - Win32에서 WSL을 실행하는 경우에만 이 환경 변수가 포함되어야 함을 나타냅니다.
- `/w` - WSL에서 Win32를 실행할 때만 경우에만 이 환경 변수가 포함되어야 함을 나타냅니다.

플래그는 필요에 따라 결합할 수 있습니다.

WSLENV 값을 미리 정의된 다른 환경 변수의 연결로 설정하고, 각 환경 변수에 접미사로 슬래시를 붙이고 그 뒤에 플래그를 추가하여 값을 변환하고 스크립트를 사용하여 변수를 전달하는 방법을 지정하는 FAQ 및 예제를 포함하여 [WSLENV에 대해 자세히 알아보세요](#). 또한 이 문서에는 WSL과 Win32 간에 GOPATH을 공유하도록 구성된 [Go 프로그래밍 언어](#)를 사용하여 개발 환경을 설정하는 예제가 포함되어 있습니다.

## 상호 운용성 사용 안 함

사용자는 다음 명령을 루트로 실행하여 단일 WSL 세션에 대해 Windows 도구를 실행하는 기능을 사용하지 않도록 설정할 수 있습니다.

Bash

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

Windows 이진 파일을 다시 사용하도록 설정하려면 모든 WSL 세션을 종료하고 bash.exe를 다시 실행하거나 다음 명령을 루트로 실행합니다.

Bash

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

interop를 사용하지 않도록 설정하는 것은 WSL 세션 간에 유지되지 않습니다. 새 세션이 시작되면 interop가 사용하도록 다시 설정됩니다.

# WSL의 고급 설정 구성

아티클 • 2023. 03. 21. • 읽는 데 24분 걸림

`wsl.conf` 및 `.wslconfig` 파일은 모든 WSL 2 배포(`.wslconfig`)에서 배포별 기반(`wsl.conf`) 및 전역적으로 고급 설정 옵션을 구성하는 데 사용됩니다. 이 가이드에서는 각 설정 옵션, 각 파일 형식을 사용하는 시기, 파일 저장 위치, 샘플 설정 파일 및 팁을 다룹니다.

## wsl.conf와 .wslconfig의 차이점은 무엇인가요?

다음을 사용하여 두 가지 방법으로 WSL을 시작할 때마다 자동으로 적용되는 설치된 Linux 배포판의 설정을 구성할 수 있습니다.

- `.wslconfig`는 WSL 2에서 실행되는 설치된 모든 배포에서 **전역적으로** 설정을 구성합니다.
- `wsl.conf`는 WSL 1 또는 WSL 2에서 실행되는 Linux 배포판에 대해 **배포별** 설정을 구성합니다.

두 파일 형식 모두 WSL 설정을 구성하는 데 사용되지만 파일이 저장되는 위치, 구성 범위 및 배포를 실행하는 WSL 버전은 모두 선택할 파일 형식에 영향을 미칩니다.

실행 중인 WSL 버전은 구성 설정에 영향을 미칩니다. WSL 2는 가벼운 VM(가상 머신)으로 실행되므로 사용되는 메모리 또는 프로세서의 양을 제어할 수 있는 가상화 설정을 사용합니다(Hyper-V 또는 VirtualBox를 사용하는 경우 익숙할 수 있음).

## wsl.conf

- 배포판의 `/etc` 디렉터리에 unix 파일로 저장됩니다.
- 배포별로 설정을 구성하는 데 사용됩니다. 이 파일에 구성된 설정은 이 파일이 저장된 디렉터리를 포함하는 특정 Linux 배포판에만 적용됩니다.
- WSL 1 또는 WSL 2 버전에서 실행되는 배포에 사용할 수 있습니다.
- 설치된 배포의 `/etc` 디렉터리로 이동하려면 `cd` 와 함께 배포의 명령줄을 사용하여 루트 디렉터리에 액세스한 다음 `ls`를 사용하여 파일을 나열하거나 `explorer.exe` 를 사용하여 Windows 파일 탐색기에서 봅니다. 디렉터리 경로는 다음과 같아야 합니다. `/etc/wsl.conf`.

## .wslconfig

- `%UserProfile%` 디렉터리에 저장됩니다.

- WSL 2 버전으로 실행되는 설치된 모든 Linux 배포에서 전역적으로 설정을 구성하는데 사용됩니다.
- **WSL 2에서 실행하는 배포에만** 사용할 수 있습니다. WSL 1로 실행되는 배포는 가상 머신으로 실행되지 않으므로 이 구성의 영향을 받지 않습니다.
- `%UserProfile%` 디렉터리로 이동하려면 PowerShell에서 `cd ~`를 사용하여 홈 디렉터리(일반적으로 사용자 프로필인 `C:\Users\<UserName>`)에 액세스하거나 Windows 파일 탐색기를 열고 주소 표시줄에 `%UserProfile%`를 입력할 수 있습니다. 디렉터리 경로는 다음과 같아야 합니다. `C:\Users\<UserName>\.wslconfig`.

WSL은 이러한 파일의 존재를 검색하고 콘텐츠를 읽고 WSL을 시작할 때마다 구성 설정을 자동으로 적용합니다. 파일이 없거나 형식이 잘못된 경우(부적절한 태그 형식) WSL은 구성 설정이 적용되지 않은 상태에서 계속 정상적으로 실행됩니다.

[실행 중인 WSL 버전을 확인합니다.](#)

#### ① 참고

wsl.conf 파일을 사용하여 배포별 설정을 조정하는 것은 Windows 빌드 17093 이상에서만 사용할 수 있습니다.

## 8초의 법칙

구성 설정 업데이트가 표시되려면 Linux 배포를 실행하는 하위 시스템이 실행을 완전히 중지하고 다시 시작할 때까지 기다려야 합니다. 일반적으로 배포 셀의 모든 인스턴스를 닫은 후 약 8초가 걸립니다.

배포판(예: Ubuntu)을 시작하는 경우 구성 파일을 수정하고 배포판을 닫은 다음 다시 시작합니다. 구성 변경 내용이 즉시 적용된다고 가정할 수 있습니다. 하위 시스템이 여전히 실행 중일 수 있으므로 현재는 그렇지 않습니다. 변경 내용을 적용할 수 있는 충분한 시간을 제공하려면 다시 시작하기 전에 하위 시스템이 중지될 때까지 기다려야 합니다. `wsl -list --running` 명령과 함께 PowerShell을 사용하여 Linux 배포(셀)를 닫은 후에도 여전히 실행 중인지 확인할 수 있습니다. 실행 중인 배포가 없으면 "실행 중인 배포가 없습니다."라는 응답을 받게 됩니다. 이제 배포를 다시 시작하여 적용된 구성 업데이트를 확인할 수 있습니다.

`wsl --shutdown` 명령은 WSL 2 배포를 다시 시작하는 빠른 경로이지만 실행 중인 모든 배포를 종료하므로 현명하게 사용합니다.

## wsl.conf에 대한 구성 설정

wsl.conf 파일은 배포별로 설정을 구성합니다. ([WSL 2 배포의 전역 구성은 .wslconfig 참조](#)).

wsl.conf 파일은 `automount`, `network`, `interop` 및 `user`의 네 가지 섹션을 지원합니다. ([.ini 파일 규칙을 따라 모델링된 키는 .gitconfig 파일과 같은 섹션에서 선언됩니다.](#)) wsl.conf 파일을 저장할 위치에 대한 정보는 [wsl.conf](#)를 참조하세요.

## 체계적인 지원

많은 Linux 배포판은 기본적으로 "systemd"(Ubuntu 포함)를 실행하며 WSL은 최근 이 컴퓨터/서비스 관리자에 대한 지원을 추가하여 WSL이 운영 체제 미설치 컴퓨터에서 선호하는 Linux 배포판을 사용하는 것과 훨씬 더 유사합니다. systemd를 사용하도록 설정하려면 WSL 버전 0.67.6 이상이 필요합니다. `wsl --version` 명령으로 WSL 버전을 확인합니다. 업데이트가 필요한 경우 [Microsoft Store에서 WSL의 최신 버전](#)을 다운로드할 수 있습니다. [블로그 발표](#)에서 자세히 알아봅니다.

systemd를 사용하도록 설정하려면 관리자 권한에 대해 `sudo`를 사용하여 텍스트 편집기에서 `wsl.conf` 파일을 열고 다음 줄을 `/etc/wsl.conf`에 추가합니다.

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

그런 다음 WSL 인스턴스를 다시 시작하려면 PowerShell에서 `wsl.exe --shutdown`을 사용하여 WSL 배포를 닫아야 합니다. 배포가 다시 시작되면 systemd가 실행 중이어야 합니다. 서비스 상태를 표시하는 `systemctl list-unit-files --type=service` 명령을 사용하여 확인할 수 있습니다.

## 자동 탑재 설정

섹션 레이블: `[automount]`

key	값	default	정보
사용	boolean	true	<code>true</code> 는 고정 드라이브(예: <code>c:/</code> 또는 <code>d:/</code> )가 <code>/mnt</code> 에서 DrvF로 자동으로 탑재되도록 합니다. <code>false</code> 는 드라이브가 자동으로 탑재되지 않음을 의미하지만, 수동으로 또는 <code>fstab</code> 를 통해 탑재할 수 있습니다.

key	값	default	정보
mountFsTab	boolean	true	<code>true</code> 는 WSL 시작 시 <code>/etc/fstab</code> 가 처리되도록 설정합니다. <code>/etc/fstab</code> 은 SMB 공유와 같은 다른 파일 시스템을 선언할 수 있는 파일입니다. 따라서 시작 시 이러한 파일 시스템을 WSL에 자동으로 탑재할 수 있습니다.
root	문자열	<code>/mnt/</code>	고정 드라이브가 자동으로 탑재될 디렉터리를 설정합니다. 기본적으로 이는 <code>/mnt/</code> 로 설정되어 있으므로 Windows 파일 시스템 C 드라이브는 <code>/mnt/c/</code> 에 탑재됩니다. <code>/mnt/</code> 를 <code>/windir/</code> 로 변경하면 고정 C 드라이브가 <code>/windir/c/</code> 에 탑재된 것을 볼 수 있습니다.
옵션	uid, gid 등과 같이 쉼표로 구분된 값 목록, 아래 자동 탑재 옵션 참조	빈 문자열	자동 탑재 옵션 값은 아래에 나열되어 있으며 기본 DrvFs 탑재 옵션 문자열에 추가됩니다. <b>DrvFs별 옵션만 지정할 수 있습니다.</b>

자동 탑재 옵션은 자동으로 탑재된 모든 드라이브의 탑재 옵션으로 적용됩니다. 특정 드라이브에 대한 옵션만 변경하려면 대신 `/etc/fstab` 파일을 사용합니다. 탑재 이진 파일이 일반적으로 플래그로 구문 분석되는 옵션은 지원되지 않습니다. 이러한 옵션을 명시적으로 지정하려면 원하는 모든 드라이브를 `/etc/fstab`에 포함시켜야 합니다.

## 자동 탑재 옵션

Windows 드라이브(DrvFs)에 다른 탑재 옵션을 설정하면 Windows 파일에 대한 파일 사용 권한이 계산되는 방법을 제어할 수 있습니다. 다음 옵션을 사용할 수 있습니다.

키	Description	기본값
uid	모든 파일의 소유자에게 사용되는 사용자 ID	WSL 배포판의 기본 사용자 ID(처음 설치할 때 기본값은 1000으로 설정됨)
gid	모든 파일의 소유자에게 사용되는 그룹 ID	WSL 배포판의 기본 그룹 ID(처음 설치할 때 기본값은 1000으로 설정됨)
umask	모든 파일 및 디렉터리에서 제외할 권한의 8진수 마스크	000
fmask	모든 파일에서 제외할 권한의 8진수 마스크	000
dmask	모든 디렉터리에서 제외할 권한의 8진수 마스크	000

키	Description	기본값
metadata	Linux 시스템 권한을 지원하기 위해 메타데이터가 Windows 파일에 추가되는지 여부	disabled
case	대/소문자를 구분하는 디렉터리로 취급되는 디렉터리와 WSL로 만든 새 디렉터리에 플래그가 설정되는지 여부를 결정합니다. 옵션에 대한 자세한 설명은 <a href="#">대/소문자 구분</a> 을 참조하세요. 옵션에는 <code>off</code> , <code>dir</code> 또는 <code>force</code> 가 포함됩니다.	off

기본적으로 WSL은 uid 및 gid를 기본 사용자 값으로 설정합니다. 예를 들어, Ubuntu에서 기본 사용자는 uid=1000, gid=1000입니다. 이 값을 사용하여 다른 gid 또는 uid 옵션을 지정하면 기본 사용자 값을 덮어씁니다. 그렇지 않으면 항상 기본값이 추가됩니다.

사용자 파일 만들기 모드 마스크(umask)는 새로 만들어진 파일에 대한 권한을 설정합니다. 기본값은 022이며 사용자만 데이터를 쓸 수 있지만 누구나 데이터를 읽을 수 있습니다. 다른 권한 설정을 반영하도록 값을 변경할 수 있습니다. 예를 들어, `umask=077`은 권한을 완전히 프라이빗으로 변경하고 다른 사용자는 데이터를 읽거나 쓸 수 없습니다. 권한을 추가로 지정하기 위해 fmask(파일) 및 dmask(디렉터리)를 사용할 수도 있습니다.

### ① 참고

권한 마스크는 파일 또는 디렉터리에 적용되기 전에 논리 OR 연산을 거칩니다.

## DrvFs란?

DrvFs는 WSL과 Windows 파일 시스템 간의 상호 운용성을 지원하도록 설계된 WSL의 파일 시스템 플러그인입니다. DrvFs를 사용하면 WSL이 /mnt/c, /mnt/d 등과 같은 /mnt 아래에 지원되는 파일 시스템이 있는 드라이브를 탑재할 수 있습니다. Windows 또는 Linux 드라이브나 디렉터리를 탑재할 때 기본 대/소문자 구분 동작을 지정하는 방법에 대한 자세한 내용은 [대/소문자 구분](#) 페이지를 참조하세요.

## 네트워크 설정

섹션 레이블: [network]

key	값	default	정보
generateHosts	boolean	true	<code>true</code> 는 WSL에서 <code>/etc/hosts</code> 를 생성하도록 설정합니다. <code>hosts</code> 파일에는 IP 주소에 해당하는 호스트 이름의 정적 맵이 포함됩니다.

key	값	default	정보
generateResolvConf	boolean <code>true</code>		<code>true</code> 는 WSL에서 <code>/etc/resolv.conf</code> 를 생성하도록 설정합니다. <code>resolv.conf</code> 에는 지정된 호스트 이름을 해당 IP 주소로 확인할 수 있는 DNS 목록이 포함됩니다.
hostname	문자열 호스트 이름	Windows	WSL 배포에 사용할 호스트 이름을 설정합니다.

## 상호 운용성 설정

섹션 레이블: [interop]

다음 옵션은 참가자 빌드 17713 이상에서 사용할 수 있습니다.

key	값	default	정보
사용	boolean <code>true</code>		이 키를 설정하면 WSL에서 Windows 프로세스 시작을 지원하는지 여부가 결정됩니다.
appendWindowsPath	boolean <code>true</code>		이 키를 설정하면 WSL에서 Windows 경로 요소를 \$PATH 환경 변수에 추가할지 여부가 결정됩니다.

## 사용자 설정

섹션 레이블: [user]

이러한 옵션은 빌드 18980 이상에서 사용할 수 있습니다.

key	값	default	정보
default	문자열 자	처음 실행 시 만들어진 초기 사용자 이름	이 키를 설정하면 WSL 세션을 처음 시작할 때 실행할 사용자를 지정합니다.

## 부팅 설정

부팅 설정은 Windows 11 및 Server 2022에서만 사용할 수 있습니다.

섹션 레이블: [boot]

key	값	default	정보

key	값	default	정보
명령을 사용합니다.	문자열	""	WSL 인스턴스가 시작될 때 실행할 명령의 문자열입니다. 이 명령은 루트 사용자로 실행됩니다. 예: <code>service docker start</code> .

## wsl.conf 파일 예

아래의 `wsl.conf` 샘플 파일은 사용 가능한 일부 구성 옵션을 보여 줍니다. 이 예에서 배포는 Ubuntu-20.04이고 파일 경로는 `\wsl.localhost\Ubuntu-20.04\etc\wsl.conf`입니다.

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 process 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
```

## .wslconfig에 대한 구성 설정

.wslconfig 파일은 WSL 2로 실행되는 모든 Linux 배포에 대해 전역적으로 설정을 구성합니다. ([배포별 구성은 wsl.conf 참조](#)).

.wslconfig 파일을 저장할 위치에 대한 정보는 [.wslconfig](#)를 참조하세요.

### ① 참고

.wslconfig 가 있는 전역 구성 옵션은 Windows 빌드 19041 이상에서 WSL 2로 실행되는 배포에만 사용할 수 있습니다. 이러한 변경 내용을 적용하려면 `wsl --shutdown` 을 실행하여 WSL 2 VM을 종료한 다음 WSL 인스턴스를 다시 시작해야 할 수 있습니다.

이 파일에는 모든 WSL 2 배포를 지원하는 VM에 영향을 주는 다음 옵션이 포함될 수 있습니다.

섹션 레이블: [ws12]

key	값	default	정보
커널(kernel)	path	받은 편지함에서 제공하는 Microsoft 빌드 커널	사용자 지정 Linux 커널에 대한 절대 Windows 경로입니다.
메모리	크기	Windows에서 총 메모리의 50% 또는 8GB 중 적은 쪽, 20175 이전 빌드에서의 경우 Windows에서 총 메모리의 80%	WSL 2 VM에 할당할 메모리 양입니다.
프로세서	number	Windows에서 동일한 수의 논리 프로세서	WSL 2 VM에 할당할 논리 프로세서 수입니다.

key	값	default	정보
localhostForwarding	boolean	true	WSL 2 VM의 와일드카드 또는 localhost에 바인딩된 포트를 localhost:port를 통해 호스트에서 연결할 수 있는지 여부를 지정하는 부울입니다.
kernelCommandLine	문자열	비어 있음	추가 커널 명령줄 인수입니다.
Safemode	boolean	false	많은 기능을 사용하지 않도록 설정하고 잘못된 상태의 배포를 복구하는 데 사용되는 "안전 모드"에서 WSL을 실행합니다. Windows 11 및 WSL 버전 0.66.2 이상에서만 사용할 수 있습니다.
swap	크기	Windows에서 메모리 크기의 25%, 가장 가까운 GB로 반올림됨	WSL 2 VM에 추가 할 스왑 공간의 양, 스왑 파일이 없는 경우 0입니다. 교환 스토리지는 메모리 수요가 하드웨어 디바이스의 한도를 초과 할 때 사용되는 디스크 기반 RAM입니다.
swapFile	path	%USERPROFILE%\AppData\Local\Temp\swap.vhdx	교환 가상 하드 디스크에 대한 절대 Windows 경로입니다.

key	값	default	정보
pageReporting	boolean	true	기본 true 설정을 사용하면 Windows가 WSL 2 가상 머신에 할당된 미사용 메모리를 회수할 수 있습니다.
guiApplications	boolean*	true	WSL에서 GUI 애플리케이션 ( <a href="#">WSLg</a> )에 대한 지원을 켜거나 끄는 부울입니다. Windows 11에서만 사용할 수 있습니다.
debugConsole	boolean*	false	WSL 2 distro 인스턴스 시작 시 <code>dmesg</code> 의 콘텐츠를 표시하는 출력 콘솔 창을 켜는 부울입니다. Windows 11에서만 사용할 수 있습니다.
nestedVirtualization	boolean*	true	중첩된 가상화를 켜거나 끄는 부울로, 다른 중첩된 VM이 WSL 2 내에서 실행될 수 있습니다. Windows 11에서만 사용할 수 있습니다.
vmlIdleTimeout	number*	60000	VM이 종료되기 전에 유휴 상태인 시간(밀리초)입니다. Windows 11에서만 사용할 수 있습니다.

`path` 값이 있는 항목은 이스케이프된 백슬래시가 있는 Windows 경로여야 합니다(예: `C:\\\\Temp\\\\myCustomKernel`).

`size` 값이 있는 항목은 크기 다음에 단위가 와야 합니다(예: `8GB` 또는 `512MB`).

값 형식 뒤에 \*가 있는 항목은 Windows 11에서만 사용할 수 있습니다.

## .wslconfig 파일 예

아래의 `.wslconfig` 샘플 파일은 사용 가능한 일부 구성 옵션을 보여 줍니다. 이 예에서 파일 경로는 `C:\Users\<UserName>\.wslconfig`입니다.

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 off default connection to bind WSL 2 localhost to Windows localhost
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
```

## 추가 자료

- Windows 명령줄 블로그: WSL 자동 구성 ↗
- Windows 명령줄 블로그: Chmod/Chown, DrvFs, 파일 메타데이터 ↗

# WSL에 대한 파일 권한

아티클 • 2023. 03. 21. • 읽는 데 9분 걸림

이 페이지에서는 Linux용 Windows 하위 시스템에서, 특히 NT 파일 시스템에서 Windows 내의 리소스에 액세스할 때 Linux 파일 권한을 해석하는 방법에 대해 자세히 설명합니다. 이 설명서에서는 [Linux 파일 시스템 권한 구조](#) 및 [umask 명령](#)을 기본적으로 이해하고 있다고 가정합니다.

WSL에서 Windows 파일에 액세스할 때 파일 권한은 Windows 권한에서 계산되거나 WSL에서 파일에 추가한 메타데이터로부터 읽어들입니다. 이 메타데이터는 기본적으로 사용하도록 설정되어 있지 않습니다.

## Windows 파일의 WSL 메타데이터

WSL에서 메타데이터를 탑재 옵션으로 사용하도록 설정된 경우 Windows NT 파일에 대한 확장 특성을 추가하고 해석하여 Linux 파일 시스템 권한을 제공할 수 있습니다.

WSL에서 추가할 수 있는 네 가지 NTFS 확장 특성은 다음과 같습니다.

특성 이름	설명
\$LXUID	사용자 소유자 ID
\$LXGID	그룹 소유자 ID
\$LXMOD	파일 모드(파일 시스템 권한 8진수 및 형식(예: 0777))
\$LXDEV	디바이스(디바이스 파일인 경우)

또한 일반 파일 또는 디렉터리가 아닌 파일(예: symlinks, FIFO, 블록 디바이스, Unix 소켓 및 문자 디바이스)에는 NTFS [재분석 지점](#)도 있습니다. 이렇게 하면 확장 특성을 쿼리하지 않고도 지정된 디렉터리에서 파일 종류를 더 빨리 확인할 수 있습니다.

## 파일 액세스 시나리오

다음은 Linux용 Windows 하위 시스템을 사용하여 다양한 방법으로 파일에 액세스할 때 권한이 결정되는 방법에 대해 설명합니다.

## Linux에서 Windows DrvFS(드라이브 파일 시스템)의 파일 액세스

이러한 시나리오는 대부분 `/mnt/c`를 통해 WSL에서 Windows 파일에 액세스할 때 발생합니다.

## 기존 Windows 파일에서 파일 권한 읽기

기존 메타데이터가 파일에 이미 있는지 여부에 따라 결과가 달라집니다.

### DrvFS 파일에 메타데이터가 없음(기본값)

메타데이터가 파일에 연결되어 있지 않으면 유효한 Windows 사용자 권한을 비트 읽기/쓰기/실행 권한으로 변환하여 이를 사용자, 그룹 및 기타에 대한 동일한 값으로 설정합니다. 예를 들어 파일에 대한 읽기 및 실행 액세스 권한은 Windows 사용자 계정에 있지만 쓰기 액세스 권한이 없는 경우 이는 사용자, 그룹 및 기타에 대해 `r-x`로 표시됩니다.

Windows에서 '읽기 전용' 특성이 파일에 설정되어 있으면 Linux에서 쓰기 액세스 권한이 부여되지 않습니다.

### 파일에 메타데이터가 있음

메타데이터가 파일에 있는 경우 유효한 Windows 사용자 권한을 변환하는 대신 해당 메타데이터 값을 사용합니다.

## chmod를 사용하여 기존 Windows 파일에 대한 파일 권한 변경

기존 메타데이터가 파일에 이미 있는지 여부에 따라 결과가 달라집니다.

### 메타데이터가 chmod 파일에 없음(기본값)

chmod에는 하나의 효과만 있습니다. 파일의 쓰기 특성을 모두 제거하면 Windows 파일의 '읽기 전용' 특성이 설정됩니다. 이는 Linux의 SMB(서버 메시지 블록) 클라이언트인 CIFS(일반 인터넷 파일 시스템)의 동작과 동일하기 때문입니다.

### 메타데이터가 chmod 파일에 있음

chmod는 파일의 기존 메타데이터에 따라 메타데이터를 변경하거나 추가합니다.

메타데이터에 해당하는 경우에도 Windows에 있는 것보다 더 많은 액세스 권한을 부여할 수 없습니다. 예를 들어 `chmod 777`을 사용하여 파일에 대한 쓰기 권한이 있음을 표시하도록 메타데이터를 설정할 수 있지만, 해당 파일에 액세스하려고 하는 경우 여전히 파일에 쓸 수 없습니다. 이는 Windows 파일에 대한 읽기 또는 쓰기 명령이 Windows 사용자 권한을 통해 라우팅되는 상호 운용성 때문입니다.

## DriveFS에서 파일 만들기

메타데이터를 사용하도록 설정되었는지 여부에 따라 결과가 달라집니다.

### 메타데이터를 사용할 수 없음(기본값)

새로 만든 파일의 Windows 권한은 특정 보안 설명자 없이 Windows에서 파일을 만든 경우와 동일하며 부모의 권한을 상속합니다.

### 메타데이터를 사용할 수 있음

파일의 권한 비트는 Linux umask를 따르도록 설정되고, 파일은 메타데이터와 함께 저장됩니다.

## 어떤 Linux 사용자 및 Linux 그룹에서 파일을 소유하나요?

기존 메타데이터가 파일에 이미 있는지 여부에 따라 결과가 달라집니다.

### 메타데이터가 사용자 파일에 없음(기본값)

기본 시나리오에서는 Windows 드라이브를 자동으로 탑재할 때 파일에 대한 UID(사용자 ID)가 WSL 사용자의 사용자 ID로 설정되고 GID(그룹 ID)가 WSL 사용자의 주 그룹 ID로 설정되도록 지정합니다.

### 메타데이터가 사용자 파일에 있음

메타데이터에 지정된 UID와 GID가 파일의 사용자 소유자 및 그룹 소유자로 적용됩니다.

## \\\wsl\\$ 을 사용하여 Windows에서 Linux 파일에 액세스

\\\wsl\\$을 통해 Linux 파일에 액세스하면 WSL 배포의 기본 사용자가 사용됩니다. 따라서 Linux 파일에 액세스하는 모든 Windows 앱은 기본 사용자와 동일한 권한을 갖습니다.

## 새 파일 만들기

Windows에서 WSL 배포 내에 새 파일을 만들 때 기본 umask가 적용됩니다. 기본 umask는 022입니다. 즉 그룹 및 기타에 대한 쓰기 권한을 제외한 모든 권한을 허용합니다.

## Linux에서 Linux 루트 파일 시스템의 파일 액세스

Linux 루트 파일 시스템에서 만들거나 수정하거나 액세스하는 모든 파일은 umask를 새로 만든 파일에 적용하는 것과 같은 표준 Linux 규칙을 따릅니다.

## 파일 권한 구성

wsl.conf의 탑재 옵션을 사용하여 Windows 드라이브 내에서 파일 권한을 구성할 수 있습니다. 탑재 옵션을 사용하면 `umask`, `dmask` 및 `fmask` 권한 마스크를 설정할 수 있습니다.

`umask`는 모든 파일에 적용되고, `dmask`는 디렉터리에만 적용되며, `fmask`는 파일에만 적용됩니다. 그런 다음, 이러한 권한 마스크는 파일에 적용될 때 논리 OR 연산을 수행합니다. 예를 들어 `umask` 값이 `023`이고 `fmask` 값이 `022`인 경우 파일에 대한 결과 권한 마스크는 `023`이 됩니다.

[wsl.conf를 사용하는 배포판별 구성 옵션](#)에 대해 자세히 알아보세요.

# WSL을 사용하여 네트워크 애플리케이션 액세스

아티클 • 2023. 03. 21. • 읽는 데 4분 걸림

네트워킹 앱을 작업할 때(Windows 앱에서 Linux 네트워킹 앱에 액세스하는 Linux 앱에서 Windows 네트워킹 앱에 액세스하는) 작업 중인 가상 머신의 IP 주소를 확인해야 하는 경우가 있으며, 이 주소는 로컬 물리적 머신의 IP 주소와 다릅니다.

## Windows에서 Linux 네트워킹 앱에 액세스 (localhost)

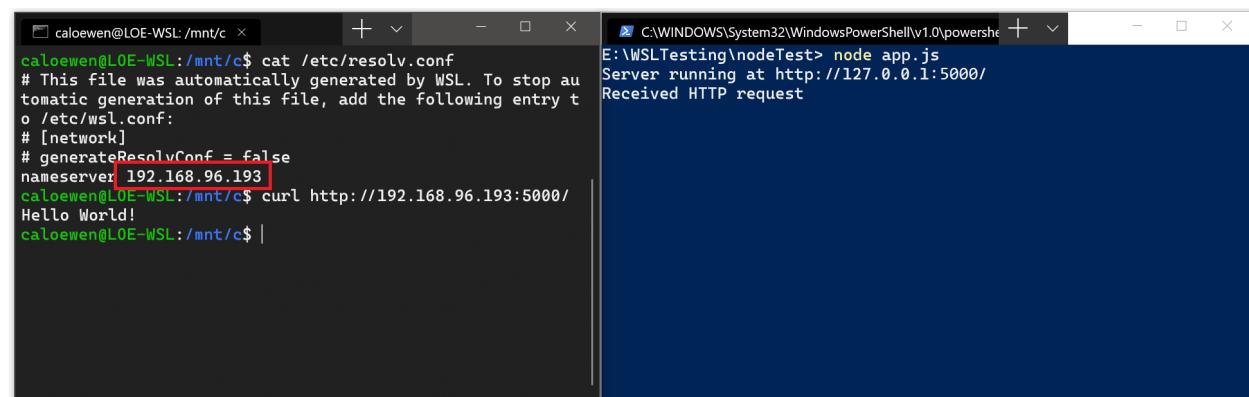
네트워킹 앱(예: NodeJS 또는 SQL 서버에서 실행되는 앱)을 Linux 배포에 구축하는 경우 `localhost`를 사용하여 Windows 앱(예: Edge 또는 Chrome 인터넷 브라우저)에서 이 앱에 액세스할 수 있습니다(일반적인 방법과 동일함).

## Linux에서 Windows 네트워킹 앱에 액세스(호스 트 IP)

Linux 배포(즉, Ubuntu)에서 Windows에서 실행되는 네트워킹 앱(예: NodeJS 또는 SQL 서버에서 실행되는 앱)에 액세스하려면 호스트 머신의 IP 주소를 사용해야 합니다. 일반적인 시나리오는 아니지만 다음 단계에 따라 작업을 수행할 수 있습니다.

1. Linux 배포에서 `cat /etc/resolv.conf` 명령을 실행하여 호스트 머신의 IP 주소를 가져옵니다.
2. IP 주소를 `nameserver`라는 용어 뒤에 복사합니다.
3. 복사한 IP 주소를 사용하여 Windows Server에 연결합니다.

아래 그림은 `curl`을 통해 Windows에서 실행 중인 Node.js 서버에 연결하여 이에 대한 예를 보여줍니다.



```
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
```

# 원격 IP 주소를 통해 연결

원격 IP 주소를 사용하여 애플리케이션에 연결하는 경우 LAN(Local Area Network)에서 들어오는 연결로 취급됩니다. 즉 애플리케이션에서 LAN 연결을 허용할 수 있는지 확인해야 합니다.

예를 들어 애플리케이션을 `127.0.0.1` 대신 `0.0.0.0`에 바인딩해야 할 수 있습니다. Flask를 사용하는 Python 앱의 예제에서는 `app.run(host='0.0.0.0')` 명령을 사용하여 이 작업을 수행할 수 있습니다. 이러한 변경을 수행하는 경우 LAN으로부터의 연결이 허용되므로 보안에 유의하세요.

## LAN(Local Area Network)에서 WSL 2 배포에 액세스

WSL 1 배포를 사용할 때 LAN에서 컴퓨터에 액세스하도록 설정된 경우 WSL에서 실행되는 애플리케이션도 LAN에서 액세스할 수 있습니다.

이는 WSL 2의 기본 사례가 아닙니다. WSL 2에는 고유한 IP 주소가 있는 가상화된 이더넷 어댑터가 있습니다. 현재 이 워크플로를 사용하도록 설정하려면 일반 가상 머신과 동일한 단계를 진행해야 합니다. (이 환경을 개선할 수 있는 방법을 찾고 있습니다.)

다음은 호스트의 포트 4000에서 수신 대기하고 IP 주소가 192.168.101.100인 WSL 2 VM에 포트 4000에 연결하는 포트 프록시를 추가하는 Windows 명령 예제입니다.

PowerShell

```
netsh interface portproxy add v4tov4 listenport=4000 listenaddress=0.0.0.0 connectport=4000 connectaddress=192.168.101.100
```

## IPv6 액세스

WSL 2 배포는 현재 IPv6 전용 주소에 연결할 수 없습니다. 이 기능을 추가하기 위한 작업이 진행 중입니다.

# 엔터프라이즈 환경: 회사를 위한 Linux용 Windows 하위 시스템 설정

아티클 • 2023. 03. 21. • 읽는 데 6분 걸림

관리자는 모든 개발자에게 동일한 승인 소프트웨어를 사용하도록 요구할 수 있습니다. 이러한 일관성은 잘 정의된 작업 환경을 만드는 데 도움이 됩니다. Linux용 Windows 하위 시스템은 한 머신에서 다음 머신으로 사용자 지정 WSL 이미지를 가져오고 내보낼 수 있도록 하여 이러한 일관성을 지원합니다. 자세한 내용은 다음 가이드를 참조하세요.

- 사용자 지정 WSL 이미지 만들기
- WSL 이미지 배포
- Linux 배포 및 패키지 업데이트 및 패치
- 엔터프라이즈 보안 및 제어 옵션

## 사용자 지정 WSL 이미지 만들기

일반적으로 "이미지"라고 하는 것은 단순히 소프트웨어 및 해당 구성 요소가 파일에 저장된 스냅샷입니다. Linux용 Windows 하위 시스템의 경우 이미지는 하위 시스템, 해당 배포 및 배포에 설치된 소프트웨어와 패키지로 구성됩니다.

WSL 이미지 만들기를 시작하려면 먼저 [Linux용 Windows 하위 시스템을 설치](#)합니다.

설치가 완료되면 비즈니스용 Microsoft Store를 사용하여 사용자에게 적합한 Linux 배포를 다운로드하고 설치합니다. [비즈니스용 Microsoft Store](#)를 사용하여 계정 만들기

## WSL 이미지 내보내기

`wsl --export <Distro> <FileName>`을 실행하여 사용자 지정 WSL 이미지를 내보냅니다. 그러면 해당 이미지가 tar 파일로 래핑되고 다른 머신에 배포할 준비가 됩니다.

## WSL 이미지 배포

`wsl --import <Distro> <InstallLocation> <FileName>`을 실행하여 공유 또는 스토리지 디바이스에서 WSL 이미지를 배포합니다. 그러면 지정된 tar 파일이 새 배포로 가져옵니다.

## Linux 배포 및 패키지 업데이트 및 패치

Linux 사용자 공간을 모니터링하고 관리하려면 Linux 구성 관리자 도구를 사용하는 것이 좋습니다. 선택할 수 있는 다양한 Linux 구성 관리자가 있습니다. WSL 2에서 Puppet을 설

치하는 방법에 대한이 [블로그 게시물](#)을 확인하세요.

## 엔터프라이즈 보안 및 제어 옵션

현재 WSL은 엔터프라이즈 시나리오에서 사용자 경험을 수정하는 것과 관련하여 제한된 제어 메커니즘을 제공합니다. 엔터프라이즈 기능은 계속 개발되고 있지만 지원되는 기능과 지원되지 않는 기능은 다음과 같습니다. 이 목록에 포함되지 않은 새로운 기능을 요청하려면 [GitHub 리포지토리](#)에서 문제를 제출하세요.

## WSL 방화벽 규칙 구성

Microsoft는 Windows에서 보안을 유지하고 로컬 디바이스로 들어오거나 나가는 무단 네트워크 트래픽을 차단하는 데 사용하는 방화벽 프로토콜을 구현합니다. 네트워크의 디바이스에 대한 보호를 최적화하려면 [모범 사례에 따라 Windows 방화벽을 구성합니다](#).

WSL과 관련하여 [로컬 정책 병합](#) 방화벽 정책이 "아니요"로 설정된 경우, WSL 네트워킹이 작동하지 않습니다. (자세한 내용은 [로컬 정책 병합 및 애플리케이션 규칙 설정](#)을 참조하십시오.)

이 구성을 변경하려면 Windows 방화벽 설정에 다음을 추가할 수 있습니다.

- 작업 허용, 방향 인바운드, 프로토콜 UDP, LocalPort 53, 프로그램:  
`%Systemroot%\System32\svchost.exe`, 서비스 SharedAccess

또한 [WSL은 내 작업 컴퓨터 또는 엔터프라이즈 환경에서 네트워크 연결이 없습니다](#).

## 지원됨

- `wsl --import` 및 `wsl --export`를 사용하여 내부적으로 승인된 이미지 공유
- [WSL Distro Launcher 리포지토리](#)를 사용하여 자신만의 엔터프라이즈용 WSL 배포판 만들기

다음은 아직 지원하지는 않지만 조사 중인 기능 목록입니다.

## 현재 지원되지 않음

다음은 현재 WSL에서 지원되지 않지만 자주 묻는 기능의 목록입니다. 이러한 요청은 백로그에 있으며 추가 방법을 조사하고 있습니다.

- WSL 내부의 사용자와 호스트 머신의 Windows 사용자 동기화
- Windows 도구를 사용하여 Linux 배포 및 패키지의 업데이트 및 패치 관리
- Windows 업데이트 시 WSL 배포 콘텐츠도 업데이트

- 엔터프라이즈의 사용자가 액세스할 수 있는 배포 제어
- WSL 내에서 필수 서비스(로깅 또는 모니터링) 실행
- SCCM 또는 Intune과 같은 Windows 구성 관리자 도구를 사용하여 Linux 인스턴스 모니터링
- McAfee 지원

# WSL과 함께 사용할 Linux 배포 가져오기

아티클 • 2023. 03. 21. • 읽는 데 6분 걸림

[Microsoft Store](#)에서 사용할 수 없는 경우에도 WSL(Linux용 Windows 하위 시스템) 내부의 Linux 배포판을 tar 파일과 함께 가져오면 사용할 수 있습니다.

이 문서에서는 Docker 컨테이너를 사용하여 tar 파일을 얻어 WSL과 함께 사용할 Linux 배포판 [CentOS](#)를 가져오는 방법을 보여 줍니다. 이 프로세스는 모든 Linux 배포판을 가져오는 데 적용할 수 있습니다.

## 배포용 tar 파일 가져오기

먼저 배포용 Linux 이진 파일이 모두 포함된 tar 파일을 가져와야 합니다.

다양한 방법으로 tar 파일을 가져올 수 있으며 그중 두 가지 방법은 다음과 같습니다.

- 제공된 tar 파일을 다운로드합니다. [Alpine Linux 다운로드](#) 사이트의 "Mini Root Filesystem" 섹션에서 Alpine의 예를 찾을 수 있습니다.
- Linux 배포 컨테이너를 찾아 인스턴스를 tar 파일로 내보냅니다. 아래 예는 [CentOS 컨테이너](#)를 사용하는 이 프로세스를 보여 줍니다.

## CentOS용 tar 파일 가져오기 예

이 예에서는 WSL 배포 내부의 Docker를 사용하여 CentOS용 tar 파일을 가져옵니다.

### 사전 요구 사항

- WSL 2를 실행하는 Linux 배포판을 사용하여 WSL을 사용하도록 설정해야 합니다.
- WSL 2 엔진이 사용하도록 설정되고 통합이 선택된 Windows용 Docker Desktop이 설치되어 있어야 합니다. 사용 약관 업데이트는 [Docker Desktop 라이선스 계약](#)을 참조하세요.

### 컨테이너에서 tar 내보내기

- Microsoft Store(이 예에서는 Ubuntu)에서 이미 설치한 Linux 배포판의 명령줄(Bash)을 엽니다.
- Docker 서비스를 시작합니다.

Bash

```
sudo service docker start
```

3. Docker 내에서 CentOS 컨테이너를 실행합니다.

Bash

```
docker run -t centos bash ls /
```

4. grep 및 awk를 사용하여 CentOS 컨테이너 ID를 가져옵니다.

Bash

```
dockerContainerID=$(docker container ls -a | grep -i centos | awk '{print $1}')
```

5. 컨테이너 ID를 탑재된 c-드라이브의 tar 파일로 내보냅니다.

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
~ $
```

이 프로세스는 Docker 컨테이너에서 CentOS tar 파일을 내보내므로 이제 WSL에서 로컬로 사용하기 위해 가져올 수 있습니다.

## tar 파일을 WSL로 가져오기

tar 파일이 준비되면 `wsl --import <Distro> <InstallLocation> <FileName>` 명령을 사용하여 가져올 수 있습니다.

# CentOS 가져오기 예

CentOS 배포 tar 파일을 WSL로 가져오려면:

- PowerShell을 열고 배포를 저장할 위치에 폴더를 만들었는지 확인합니다.

```
PowerShell

cd C:\temp
mkdir E:\wslDistroStorage\CentOS
```

- wsl --import <DistroName> <InstallLocation> <InstallTarFile> 명령을 사용하여 tar 파일을 가져옵니다.

```
PowerShell

wsl --import CentOS E:\wslDistroStorage\CentOS .\centos.tar
```

- wsl -l -v 명령을 사용하여 설치한 배포판을 확인합니다.

```
PowerShell          Ubuntu
PS C:\Users\crlloewen> 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           0    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> |
```

- 마지막으로 wsl -d CentOS 명령을 사용하여 새로 가져온 CentOS Linux 배포판을 실행합니다.

## 기본 사용자와 같은 WSL 특정 구성 요소 추가

--import를 사용할 때 기본적으로 항상 루트 사용자로 시작됩니다. 고유의 사용자 계정을 설정할 수 있지만 설정 프로세스는 각 Linux 배포판에 따라 약간 다를 수 있습니다.

방금 가져온 CentOS 배포로 사용자 계정을 설정하려면 먼저 PowerShell을 열고 다음 명령을 사용하여 CentOS로 부팅합니다.

```
PowerShell
```

```
wsl -d CentOS
```

다음으로 CentOS 명령줄을 엽니다. 이 명령을 사용하여 sudo 및 암호 설정 도구를 CentOS에 설치하고 사용자 계정을 만든 다음 기본 사용자로 설정합니다. 이 예에서 사용자 이름은 'caloewen'입니다.

### ① 참고

사용자가 sudo를 사용할 수 있도록 sudoers 파일에 사용자 이름을 추가할 수 있습니다. `adduser -G wheel $myUsername` 명령은 `myUsername` 사용자를 휠 그룹에 추가합니다. 휠 그룹의 사용자에게는 자동으로 sudo 권한이 부여되며 관리자 권한의 권한이 필요한 작업을 수행할 수 있습니다.

```
Bash
```

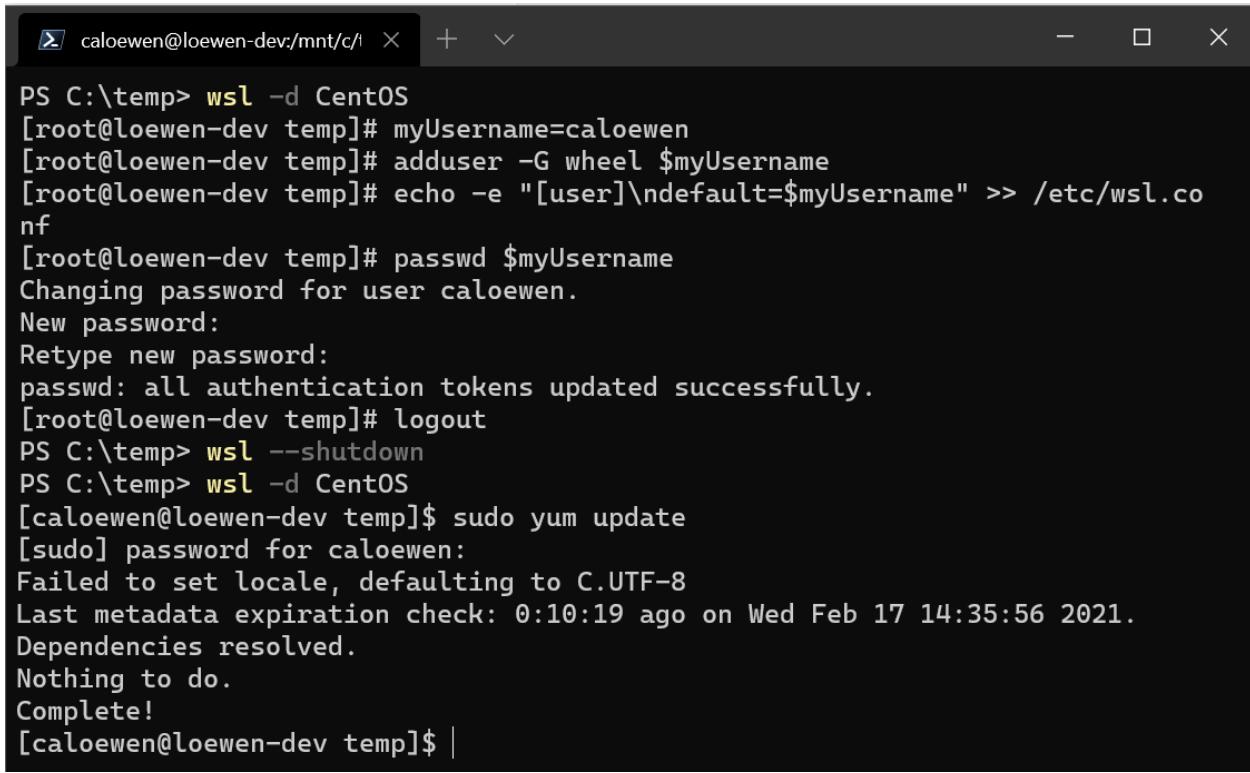
```
yum update -y && yum install passwd sudo -y  
myUsername=caloewen  
adduser -G wheel $myUsername  
echo -e "[user]\ndefault=$myUsername" >> /etc/wsl.conf  
passwd $myUsername
```

이제 해당 인스턴스를 종료하고 모든 WSL 인스턴스가 종료되었는지 확인해야 합니다. PowerShell에서 다음 명령을 실행하여 새 기본 사용자를 보려면 배포를 다시 시작합니다.

```
PowerShell
```

```
wsl --terminate CentOS  
wsl -d CentOS
```

이제 이 예를 기반으로 한 출력으로 [caloewen@loewen-dev]\$ 가 표시됩니다.



```
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
nf
[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]$ |
```

WSL 설정 구성에 대한 자세한 내용은 [.wslconfig](#) 및 [wsl.conf](#)로 설정 구성을 참조하세요.

## 사용자 지정 Linux 배포판 사용

Microsoft Store에서 사용할 수 있는 WSL 배포와 똑같이 작동하는 UWP 앱으로 패키지된 고유한 사용자 지정 Linux 배포를 만들 수 있습니다. 자세한 방법은 [WSL용 사용자 지정 Linux 배포판 만들기](#)를 참조하세요.

# WSL용 사용자 지정 Linux 배포판 만들기

아티클 • 2022. 09. 22. • 읽는 데 3분 걸림

오픈 소스 WSL 샘플을 사용하여 Microsoft Store 대한 WSL 배포판 패키지를 빌드하거나 테스트용으로 로드할 사용자 지정 Linux 배포판 패키지를 만듭니다. GitHub 배포판 시작 관리자 리포지토리 [☞](#)를 찾을 수 있습니다.

이 프로젝트를 사용하면 다음을 수행할 수 있습니다.

- Linux 배포 유지 관리자는 WSL에서 실행되는 appx로 Linux 배포를 패키지하고 제출 합니다.
- 개발자가 개발 머신에 테스트용으로 로드할 수 있는 사용자 지정 Linux 배포판 만들기

## 배경

WSL용 Linux 배포판은 Microsoft Store 통해 UWP 애플리케이션으로 배포합니다. 그런 다음, Windows 커널에 있는 하위 시스템인 WSL에서 실행되는 애플리케이션을 설치할 수 있습니다. 이 배달 메커니즘은 [이전 블로그 게시물](#) [☞](#)에서 설명한 대로 많은 이점을 제공합니다.

## 사용자 지정 Linux 배포판 패키지 테스트용 로드

사용자 지정 Linux 배포판 패키지를 개인 컴퓨터에 테스트용으로 로드하는 애플리케이션으로 만들 수 있습니다. 배포 유지 관리자로 제출하지 않는 한 사용자 지정 패키지는 Microsoft Store 통해 배포되지 않습니다. 앱을 테스트용으로 로드하도록 컴퓨터를 설정 하려면 "개발자용" 아래의 시스템 설정에서 이 기능을 사용하도록 설정해야 합니다. 개발자 모드가 있거나 앱을 테스트용으로 로드해야 합니다.

## Linux 배포판 유지 관리자의 경우

스토어에 제출하려면 Microsoft와 협력하여 게시 승인을 받아야 합니다. Microsoft Store 배포를 추가하는 데 관심이 있는 Linux 배포 소유자인 경우 문의 [wslpartners@microsoft.com](mailto:wslpartners@microsoft.com)하세요.

## 시작하기

[배포판 시작 관리자 GitHub 리포지토리](#) [☞](#)의 지침에 따라 사용자 지정 Linux 배포판 패키지를 만듭니다.

## 팀 블로그

- Linux 배포 유지 관리자용 WSL 샘플 소싱 및 사용자 지정 Linux 배포 테스트용 로드 열기 ↗
- 명령줄 블로그 ↗

## 피드백 제공

- 배포판 시작 관리자 GitHub 리포지토리 ↗
- WSL에 대한 GitHub 문제 추적기 ↗

# WSL 2에 Linux 디스크 탑재

아티클 • 2023. 03. 21. • 읽는 데 11분 걸림

Windows에서 지원하지 않는 Linux 디스크 형식에 액세스하려는 경우 WSL 2를 사용하여 디스크를 탑재하고 해당 콘텐츠에 액세스할 수 있습니다. 이 자습서에서는 WSL2에 연결 할 디스크 및 파티션을 식별하는 단계, 이를 탑재하는 방법 및 액세스하는 방법을 다룹니다.

USB 디바이스(플래시 드라이브, SD 카드 판독기 등) 연결 방법에 대한 지침을 찾고 있는 경우 [USB 디바이스 연결](#)을 참조하세요.

## ① 참고

디스크를 WSL 2에 연결하려면 관리자 액세스 권한이 필요합니다. WSL 2 `mount` 명령은 현재 사용 중인 디스크(또는 디스크에 속한 파티션) 탑재를 지원하지 않습니다. `wsl --mount`은 파티션만 요청하더라도 항상 전체 디스크를 연결합니다. Windows 설치 디스크를 탑재할 수 없습니다.

## 사전 요구 사항

Windows 11 빌드 22000 이상에 있거나 Microsoft Store 버전의 WSL을 실행해야 합니다. [Windows 참가자 프로그램](#)에 조인하여 최신 미리 보기 빌드를 가져올 수 있습니다.

## 파티션되지 않은 디스크 탑재

이 간단한 경우 파티션이 없는 디스크가 있는 경우 `wsl --mount` 명령을 사용하여 직접 탑재할 수 있습니다. 먼저 디스크를 식별해야 합니다.

1. **디스크 식별** - Windows에서 사용 가능한 디스크를 나열하려면 다음을 실행합니다.

PowerShell

```
GET-CimInstance -query "SELECT * from Win32_DiskDrive"
```

디스크 경로는 'DeviceID' 열에서 사용할 수 있습니다. 일반적으로 `\.\PHYSICALDRIVE*` 형식입니다.

2. **디스크 탑재** - PowerShell을 사용하여 위에서 찾은 디스크 경로를 사용하여 디스크를 탑재하고 다음을 실행할 수 있습니다.

## PowerShell

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

```
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$
```

## 파티션 디스크 탑재

어떤 파일 형식인지 또는 어떤 파티션이 있는지 확실하지 않은 디스크가 있는 경우 아래 단계에 따라 디스크를 탑재할 수 있습니다.

1. 디스크 식별 - Windows에서 사용 가능한 디스크를 나열하려면 다음을 실행합니다.

## PowerShell

```
GET-CimInstance -query "SELECT * from Win32_DiskDrive"
```

디스크 경로는 일반적으로 `\\.\PHYSICALDRIVE*` 형식으로 'DeviceID' 다음에 나열됩니다.

2. WSL 2에 탑재할 파티션 나열 및 선택 - 디스크가 식별되면 다음을 실행합니다.

## PowerShell

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

이렇게 하면 WSL 2에서 디스크를 사용할 수 있습니다. (이 예의 경우 <DiskPath>는 \\.\PHYSICALDRIVE\*입니다.)

3. 연결되면 WSL 2 내에서 다음 명령을 실행하여 파티션을 나열할 수 있습니다.

Bash

```
lsblk
```

그러면 사용 가능한 블록 디바이스와 해당 파티션이 표시됩니다.

Linux 내부에서 블록 디바이스는 /dev/<Device><Partition>으로 식별됩니다. 예를 들어, /dev/sdb3은 디스크 sdb의 파티션 번호 3입니다.

예제 출력:

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
```

## 파일 시스템 형식 식별

디스크 또는 파티션의 파일 시스템 형식을 모르는 경우 다음 명령을 사용할 수 있습니다.

Bash

```
blkid <BlockDevice>
```

이렇게 하면 검색된 파일 시스템 형식(TYPE=<Filesystem> 형식 아래)이 출력됩니다.

## 선택한 파티션 탑재

탑재할 파티션을 식별했으면 각 파티션에서 다음 명령을 실행합니다.

PowerShell

```
wsl --mount <DiskPath> --partition <PartitionNumber> --type <Filesystem>
```

## ① 참고

전체 디스크를 단일 볼륨으로 탑재하려는 경우(즉, 디스크가 분할되지 않은 경우) --partition을 생략할 수 있습니다.

생략하면 기본 파일 시스템 형식은 "ext4"입니다.

## 디스크 콘텐츠에 액세스

탑재되면 구성 값 `automount.root` 가 가리키는 경로에서 디스크에 액세스할 수 있습니다. 기본값은 `/mnt/wsl`입니다.

Windows에서 `\\\wsl$\\<Distro>\\<Mountpoint>`(Linux 배포판 선택)로 이동하여 파일 탐색기에서 디스크에 액세스할 수 있습니다.

## 디스크 분리

WSL 2에서 디스크 탑재를 해제하고 분리하려면 다음을 실행합니다.

PowerShell

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

## WSL에 VHD 탑재

## ① 참고

Microsoft Store의 WSL<sup>2</sup>에는 VHD를 직접 탑재하는 다음과 같은 새 인수가 도입되었습니다. `wsl --mount --vhd <pathToVHD>`

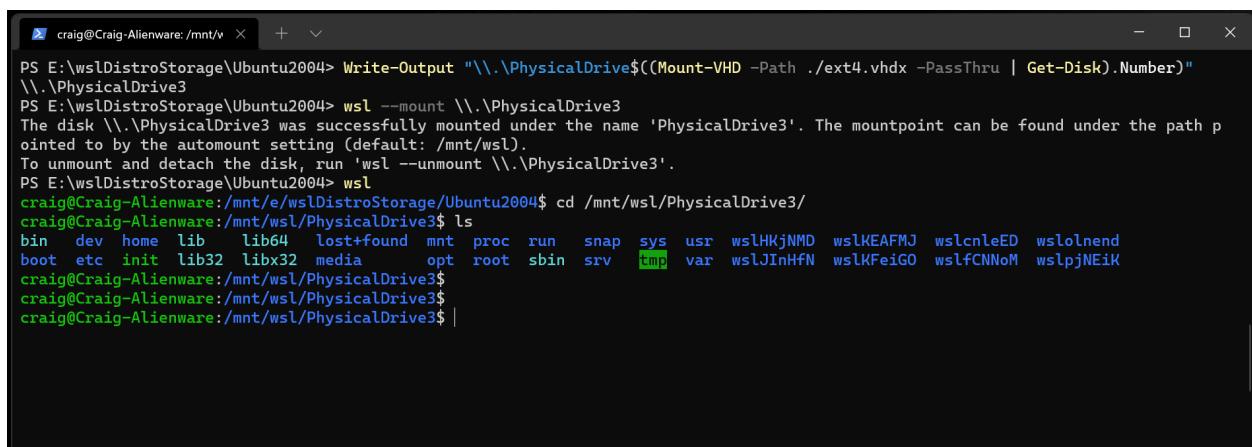
`wsl --mount`을 사용하여 VHD(가상 하드 디스크 파일)를 WSL에 탑재할 수도 있습니다. 이렇게 하려면 먼저 Windows에서 **Mount-VHD** 명령을 사용하여 VHD를 Windows에 탑재해야 합니다. 관리자 권한으로 이 명령을 실행해야 합니다. 다음은 이 명령을 사용하고 디스크 경로를 출력하는 예입니다. `<pathToVHD>`를 실제 VHD 경로로 바꿔야 합니다.

PowerShell

```
Write-Output "\\.\PhysicalDrive$((Mount-VHD -Path <pathToVHD> -PassThru | Get-Disk).Number)"
```

위의 출력을 사용하여 이 VHD의 디스크 경로를 가져오고 이전 섹션의 지침에 따라 WSL에 탑재할 수 있습니다.

또한 이 기술을 사용하여 다른 WSL 배포판의 가상 하드 디스크를 탑재하고 상호 작용할 수 있습니다. 각 WSL 2 배포판은 `ext4.vhdx`라는 가상 하드 디스크 파일을 통해 저장되기 때문입니다. 기본적으로 WSL 2 배포판용 VHD는 다음 경로에 저장됩니다. `c:\Users\[user]\AppData\Local\Packages\[distro]\LocalState\[distroPackageName]`. 이러한 시스템 파일에 액세스할 때 주의하세요. 이는 고급 사용자 워크플로입니다. 디스크가 사용 중이 아닌지 확인하려면 이 디스크와 상호 작용하기 전에 `wsl --shutdown`을 실행해야 합니다.



The screenshot shows a Windows PowerShell window titled 'craig@Craig-Alienware: /mnt/v'. The command `Write-Output "\\.\PhysicalDrive$((Mount-VHD -Path ./ext4.vhdx -PassThru | Get-Disk).Number)" \\.\PhysicalDrive3` is run, followed by `wsl --mount \\.\PhysicalDrive3`. A message indicates the disk 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'. Finally, the user runs `cd /mnt/wsl/PhysicalDrive3` and lists the contents of the directory, which includes standard Linux system directories like bin, dev, home, lib, lib64, lost+found, mnt, proc, run, snap, sys, usr, wslHKjNMD, wslKEAFMJ, wslcnleED, wsloInend, boot, etc, init, lib32, libx32, media, opt, root, sbin, srv, tmp, var, wslJInHfN, wslKFeiGO, wslfCNNoM, wslpjNEiK.

## 명령줄 참조

### 특정 파일 시스템 탑재

기본적으로 WSL 2는 디바이스를 ext4로 탑재하려고 시도합니다. 다른 파일 시스템을 지정하려면 다음을 실행합니다.

PowerShell

```
wsl --mount <DiskPath> -t <FileSystem>
```

예를 들어, 디스크를 fat으로 탑재하려면 다음을 실행합니다.

```
wsl --mount <Diskpath> -t vfat
```

① 참고

WSL2에서 사용 가능한 파일 시스템을 나열하려면 다음을 실행합니다. `cat`

```
/proc/filesystems
```

디스크가 WSL2(Linux 파일 시스템)를 통해 탑재되면 더 이상 Windows 파일 시스템에서 ext4 드라이버를 통해 탑재할 수 없습니다.

## 특정 파티션 탑재

기본적으로 WSL 2는 전체 디스크를 탑재하려고 시도합니다. 특정 파티션을 탑재하려면 다음을 실행합니다.

```
wsl --mount <Diskpath> -p <PartitionIndex>
```

디스크가 MBR(마스터 부트 레코드) 또는 GPT(GUID 파티션 테이블)인 경우에만 작동합니다. [파티션 스타일 - MBR 및 GPT에 대해 참조하세요](#).

## 탑재 옵션 지정

탑재 옵션을 지정하려면 다음을 실행합니다.

PowerShell

```
wsl --mount <DiskPath> -o <MountOptions>
```

예제:

PowerShell

```
wsl --mount <DiskPath> -o "data=ordered"
```

### ① 참고

현재 파일 시스템 관련 옵션만 지원됩니다. `ro`, `rw`, `noatime`, ...과 같은 일반 옵션은 지원되지 않습니다.

## 탑재하지 않고 디스크 연결

위의 옵션에서 디스크 구성표를 지원하지 않는 경우 다음을 실행하여 디스크를 탑재하지 않고 WSL 2에 디스크를 연결할 수 있습니다.

PowerShell

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

이렇게 하면 WSL 2 내부에서 블록 디바이스를 사용할 수 있으므로 여기에서 수동으로 탑재할 수 있습니다. `lsblk`을 사용하여 WSL 2 내에서 사용 가능한 블록 디바이스를 나열합니다.

## 탑재 이름 지정

① 참고

이 옵션은 Microsoft Store의 WSL 에서만 사용할 수 있습니다.

기본적으로 탑재 지점 이름은 실제 디스크 또는 VHD 이름을 기반으로 생성됩니다. 이는 `--name`으로 재정의할 수 있습니다. 예제:

PowerShell

```
wsl --mount <DiskPath> --name myDisk
```

## 디스크 분리

WSL 2에서 디스크를 분리하려면 다음을 실행합니다.

PowerShell

```
wsl --unmount [DiskPath]
```

`Diskpath`를 생략하면 연결된 모든 디스크가 분리되고 분리됩니다.

① 참고

하나의 디스크가 분리되지 않으면 `wsl --shutdown`을 실행하여 WSL 2를 강제로 종료할 수 있습니다. 그러면 디스크가 분리됩니다.

## 제한 사항

- 현재로서는 전체 디스크만 WSL 2에 연결할 수 있습니다. 즉, 파티션만 연결할 수는 없습니다. 구체적으로 이는 부팅 디바이스가 Windows에서 분리될 수 없기 때문에 `wsl --mount`를 사용하여 부팅 디바이스의 파티션을 읽을 수 없음을 의미합니다.
- 커널에서 기본적으로 지원되는 파일 시스템만 `wsl --mount`에 의해 탑재될 수 있습니다. 이는 `wsl --mount`를 호출하여 설치된 파일 시스템 드라이버(예: ntfs-3g)를 사용할 수 없음을 의미합니다.
- 커널에서 직접 지원하지 않는 파일 시스템은 `--bare` 연결을 통해 탑재한 다음 관련 FUSE 드라이버를 호출할 수 있습니다.

# USB 디바이스 연결

아티클 • 2023. 03. 21. • 읽는 데 7분 걸림

이 가이드에서는 USB/IP 오픈 소스 프로젝트인 [usbipd-win](#)을 사용하여 WSL 2에서 실행되는 Linux 배포판에 USB 디바이스를 연결하는 데 필요한 단계를 안내합니다.

Windows 장치에서 USB/IP 프로젝트를 설정하면 Arduino 플래시 또는 스마트 카드 리더 액세스와 같은 공통 개발자 USB 시나리오를 사용할 수 있습니다.

## 사전 요구 사항

- Windows 11 실행(빌드 22000 이상) (*Windows 10 지원 가능, 아래 참조*)
- x64/x86 프로세서가 있는 컴퓨터가 필요합니다. (Arm64는 현재 usbipd-win에서 지원되지 않습니다.)
- Linux 배포판이 설치되고 [WSL 2로 설정](#)됩니다.
- [Linux 커널 5.10.60.1 이상](#)을 실행합니다.

### ① 참고

Windows 버전 및 빌드 번호를 확인하려면 **Windows 로고 키 + R**을 선택하고, **winver**를 입력하고, **확인**을 선택합니다. **시작 > 설정 > Windows 업데이트 > 업데이트 확인**을 선택하여 최신 Windows 버전으로 업데이트할 수 있습니다. Linux 커널 버전을 확인하려면 Linux 배포판을 열고 `uname -a` 명령을 입력합니다. 최신 커널로 수동으로 업데이트하려면 PowerShell을 열고 '`wsl --update`' 명령을 입력합니다.

### ① 중요

WSL 2에서 실행되는 Linux 배포판에 USB 디바이스를 연결하는 데 Windows 11을 사용하는 것이 좋습니다. 그러나, Windows 10 [USBIPD-WIN 프로젝트 리포지토리의 지침](#)에 따라 고유한 USBIP 사용 WSL 2 커널을 빌드하여 USB 디바이스를 연결하는 데 사용할 수 있습니다.

## USBIPD-WIN 프로젝트 설치

USB 디바이스 연결에 대한 지원은 WSL에서 기본적으로 사용할 수 없으므로 오픈 소스 usbipd-win 프로젝트를 설치해야 합니다.

1. [usbipd-win 프로젝트의 최신 릴리스 페이지](#)로 이동합니다.

- 설치 관리자를 다운로드할 .msi 파일을 선택합니다. (다운로드한 설치 프로그램을 신뢰하는지 확인하라는 경고가 표시될 수 있습니다.)
- 다운로드한 usbipd-win\_x.msi 설치 관리자 파일을 실행합니다.

### ① 참고

또는 Windows 패키지 관리자(winget)를 사용하여 usbipd-win 프로젝트를 설치할 수도 있습니다. 이미 winget을 설치한 경우, `winget install --interactive --exact dorssel.usbipd-win` 명령을 사용하여 usbipd-win을 설치합니다. --interactive를 종료하면 드라이버를 설치하는 데 필요한 경우 winget이 컴퓨터를 즉시 다시 시작할 수 있습니다.

그러면 다음이 설치됩니다.

- 서비스 이름 `usbipd`(표시 이름: USBIP 장치 호스트) Windows의 서비스 앱을 사용하여 이 서비스의 상태를 확인할 수 있습니다.
- 명령줄 도구 `usbipd`입니다. 이 도구의 위치는 PATH 환경 변수에 추가됩니다.
- 모든 로컬 서브넷이 서비스에 연결하도록 허용하는 `usbipd`라는 방화벽 규칙. 이 방화벽 규칙을 수정하여 액세스 제어를 미세 조정할 수 있습니다.

## Linux에서 USBIP 도구 및 하드웨어 데이터베이스 설치

USB/IP 프로젝트 설치가 완료되면 사용자 공간 도구와 USB 하드웨어 식별자의 데이터베이스를 설치해야 합니다. 이러한 지침은 Ubuntu를 위한 것입니다. [다른 배포에는 다른 usbip 클라이언트 패키지가 필요할 수 있습니다](#).

Ubuntu에서 다음 명령을 실행합니다.

Bash

```
sudo apt install linux-tools-generic hwdata
sudo update-alternatives --install /usr/local/bin/usbip usbip
/usr/lib/linux-tools/*-generic/usbip 20
```

이 시점에서 USB 디바이스를 공유하기 위해 Windows에서 서비스가 실행되고 있으며 공유 디바이스에 연결하기 위해 필요한 도구가 WSL에 설치됩니다.

## USB 디바이스 연결

USB 디바이스를 연결하기 전에 WSL 명령줄이 열려 있는지 확인합니다. 이렇게 하면 WSL 2 경량 VM이 활성 상태로 유지됩니다.

1. 관리자 모드에서 PowerShell을 열고 명령을 입력하여 Windows에 연결된 모든 USB 디바이스를 나열합니다.

```
PowerShell
```

```
usbipd wsl list
```

2. WSL에 연결하려는 디바이스의 버스 ID를 선택하고 이 명령을 실행합니다. WSL에서 sudo 명령을 실행할 비밀번호를 묻는 메시지가 표시됩니다. 연결할 Linux 배포판은 기본 배포여야 합니다. (기본 배포를 변경하려면 [WSL용 기본 명령](#) 문서를 참조하십시오.)

```
PowerShell
```

```
usbipd wsl attach --busid <busid>
```

3. Ubuntu(또는 원하는 WSL 명령줄)를 열고 명령을 사용하여 연결된 USB 디바이스를 나열합니다.

```
Bash
```

```
lsusb
```

방금 연결한 디바이스가 표시되고 일반 Linux 도구를 사용하여 디바이스와 상호 작용할 수 있어야 합니다. 애플리케이션에 따라 루트가 아닌 사용자가 디바이스에 액세스할 수 있도록 udev 규칙을 구성해야 할 수 있습니다.

4. WSL에서 디바이스 사용을 완료한 후에는 USB 디바이스의 연결을 물리적으로 끊거나 관리자 모드의 PowerShell에서 이 명령을 실행할 수 있습니다.

```
PowerShell
```

```
usbipd wsl detach --busid <busid>
```

작동 방식에 대한 자세한 내용은 [GitHub의 Windows 명령줄 블로그](#) 및 [usbipd-win 리포지토리](#)를 참조하십시오.

비디오 데모는 [WSL 2: USB 디바이스 연결\(탭 및 공간 표시\)](#)을 참조하십시오.

# 대/소문자 구분 조정

아티클 • 2023. 03. 21. • 읽는 데 18분 걸림

대/소문자 구분은 대문자(FOO.txt) 및 소문자(foo.txt) 문자를 파일 이름이나 디렉터리에서 서로 다르게 취급할 것인지(대/소문자 구분) 아니면 똑같은 것으로 취급할 것인지(대/소문자 구분 안 함) 여부를 결정합니다.

- 대/소문자 구분: FOO.txt ≠ foo.txt ≠ Foo.txt
- 대/소문자 구분 없음: FOO.txt = foo.txt = Foo.txt

## Windows 및 Linux의 대/소문자 구분 차이점

Linux 및 Windows 파일 및 디렉터리를 모두 사용하는 경우, 대/소문자 구분 처리 방법을 조정해야 할 수 있습니다.

표준 동작:

- Windows 파일 시스템은 파일 및 디렉터리 이름을 대/소문자를 구분하지 않는 것으로 처리합니다. FOO.txt 및 foo.txt는 동일한 파일로 처리됩니다.
- Linux 파일 시스템은 파일 및 디렉터리 이름을 대/소문자를 구분하는 것으로 처리합니다. FOO.txt 및 foo.txt는 다른 파일로 처리됩니다.

Windows 파일 시스템은 디렉터리당 특성 플래그를 사용하여 대/소문자 구분 설정을 지원합니다. 표준 동작은 대/소문자를 구분하지 않지만, 특성 플래그를 할당하여 디렉터리 대/소문자를 구분하여 대/소문자별로만 다를 수 있는 Linux 파일 및 폴더를 인식할 수 있습니다.

WSL(Linux용 Windows 하위 시스템) 파일 시스템에 드라이브를 탑재할 때 특히 그렇습니다. WSL 파일 시스템에서 작업할 때 Linux를 실행하므로, 파일 및 디렉터리에서 기본적으로 대/소문자를 구분하는 것으로 처리됩니다.

### ① 참고

과거에 이름이 대/소문자만 다른 파일이 있는 경우, Windows 애플리케이션은 파일 시스템을 대/소문자를 구분하지 않는 것으로 간주하고 이름이 다른 파일만 구분할 수 없으므로 Windows에서 이러한 파일에 액세스할 수 없습니다. Windows 파일 탐색기는 두 파일을 모두 표시하지만, 선택한 파일에 관계없이 하나만 열립니다.

## 파일 및 디렉터리의 대/소문자 구분 변경

다음 단계에서는 대/소문자를 구분하고 대/소문자만 다른 파일 및 폴더를 인식하도록 Windows 파일 시스템에서 디렉터리를 변경하는 방법에 대해서 설명합니다.

### ⚠ 경고

일부 Windows 애플리케이션은 파일 시스템이 대/소문자를 구분하지 않는다고 가정하여 올바른 대/소문자를 사용하여 파일을 참조하지 않습니다. 예를 들어, 애플리케이션에서 파일 이름을 변환하여 대/소문자를 모두 사용하는 것은 드문 일이 아닙니다. 대/소문자를 구분하는 것으로 표시된 디렉터리의 경우, 이러한 애플리케이션이 더 이상 파일에 액세스할 수 없음을 의미합니다. 또한, Windows 애플리케이션이 대/소문자 구분 파일을 사용하는 디렉터리 트리에 새 디렉터리를 만드는 경우, 이러한 디렉터리가 대/소문자를 구분하지 않습니다. 이렇게 하면 대/소문자를 구분하는 디렉터리의 경우, Windows 도구로 작업하기가 어려울 수 있으므로 Windows 파일 시스템 대/소문자 구분 설정을 변경할 때는 주의해야 합니다.

## 현재 대/소문자 구분 검사

디렉터리가 Windows 파일 시스템에 대/소문자를 구분하는지 확인하려면 다음 명령을 실행합니다.

PowerShell

```
fsutil.exe file queryCaseSensitiveInfo <path>
```

<path> 을(를) 파일 경로로 바꿉니다. Windows(NTFS) 파일 시스템의 디렉터리의 경우, <path> 은(는) C:\Users\user1\case-test 와(과) 같이 표시되거나 user1 디렉터리에 이미 있는 경우, 다음을 실행할 수 있습니다. `fsutil.exe file setCaseSensitiveInfo case-test`

## 대/소문자 구분 조정

디렉터리별 대/소문자 구분에 대한 지원은 Windows 10, 빌드 17107에서 시작되었습니다. Windows 10, 빌드 17692에서는 WSL 내부에서 디렉터리에 대한 대/소문자 민감도 플래그 검사 및 수정을 포함하도록 지원이 업데이트되었습니다. 대/소문자 구분은 system.wsl\_case\_sensitive(이)라는 확장 특성을 사용하여 노출됩니다. 이 특성의 값은 대/소문자를 구분하지 않는 디렉터리에 대해 0, 대/소문자를 구분하는 디렉터리에 대해 1로 표시됩니다.

디렉터리의 대/소문자 구분을 변경하려면 **관리자 권한** (관리자 권한으로 실행)을 실행해야 합니다. 대/소문자 구분 플래그를 변경하려면 디렉터리에 대한 "쓰기 특성", "파일 만

"들기", "폴더 만들기" 및 "하위 폴더 및 파일 삭제" 권한이 필요합니다. 자세한 내용은 문제 해결 섹션을 참조하십시오.

대/소문자를 구분하도록(FOO ≠ foo) Windows 파일 시스템의 디렉터리를 변경하려면, 관리자 권한으로 PowerShell을 실행하고 다음 명령을 사용합니다.

PowerShell

```
fsutil.exe file setCaseSensitiveInfo <path> enable
```

대/소문자를 구분하지 않도록(FOO = foo) Windows 파일 시스템의 디렉터리를 다시 변경하려면, 관리자 권한으로 PowerShell을 실행하고 다음 명령을 사용합니다.

PowerShell

```
fsutil.exe file setCaseSensitiveInfo <path> disable
```

해당 디렉터리의 대/소문자 민감도 플래그 특성을 변경하려면 디렉터리가 비어 있어야 합니다. 이름이 대/소문자별로만 다른 폴더/파일이 포함된 디렉터리에서는 대/소문자 구분 플래그를 사용하지 않도록 설정할 수 없습니다.

## 대/소문자 구분 상속

새 디렉터리를 만들 때 해당 디렉터리가 부모 디렉터리에서 대/소문자 민감도를 상속합니다.

### ⚠ 경고

WSL 1 모드에서 실행할 때 이 상속 정책에는 예외가 있습니다. 배포가 WSL 1 모드에서 실행되는 경우, 디렉터리별 대/소문자 구분 플래그는 상속되지 않습니다. 대/소문자 구분 디렉터리에서 만든 디렉터리가 자동으로 대/소문자를 구분하지 않습니다. 각 디렉터리를 대/소문자를 구분하는 것으로 명시적으로 표시해야 합니다.

## WSL 구성 파일에서 드라이브를 탑재하기 위한 대/소문자 구분 옵션

WSL 구성 파일을 사용하여 Linux용 Windows 하위 시스템 드라이브를 탑재할 때 대/소문자 구분을 관리할 수 있습니다. 설치한 각 Linux 배포판에는 `/etc/wsl.conf`(이)라는 자체 WSL 구성 파일이 있을 수 있습니다. 드라이브를 탑재하는 방법에 대한 자세한 내용은 [WSL 2에서 Linux 디스크 탑재 시작하기](#)를 참조하십시오.

드라이브를 탑재할 때 `wsl.conf` 파일에서 대/소문자 민감도 옵션을 구성하려면 다음을 수행합니다.

1. 사용할 Linux 배포(예: Ubuntu)를 엽니다.
2. `etc` 폴더가 표시될 때까지 디렉터리를 변경합니다(`home` 디렉터리에서 `cd ..` 위로 이동해야 할 수 있음).
3. `etc` 디렉터리의 파일을 나열하여 `wsl.conf` 파일이 이미 있는지 확인합니다(`ls` 명령 또는 `explorer.exe .` 을(를) 사용하여 Windows 파일 탐색기로 디렉터리 보기).
4. `wsl.conf` 파일이 아직 없는 경우, `sudo touch wsl.conf` 을(를) 사용하거나 `sudo nano /etc/wsl.conf` 을(를) 실행하여 만들 수 있습니다. 그러면 Nano 편집에서 저장할 때 파일이 만들어질 수 있습니다.
5. `wsl.conf` 파일에 추가할 수 있는 옵션은 다음과 같습니다.

기본 설정: `dir` 디렉터리당 대/소문자 구분을 사용하도록 설정합니다.

Bash

```
[automount]
options = case = dir
```

대/소문자 구분 사용할 수 없음(탑재된 NTFS 드라이브의 모든 디렉터리가 대/소문자를 구분하지 않음) `off`

Bash

```
[automount]
options = case = off
```

(NTFS) 드라이브의 모든 디렉터리를 대/소문자를 구분하는 것으로 처리합니다. `force`

Bash

```
[automount]
options = case = force
```

이 옵션은 WSL 1로 실행되는 Linux 배포판에서 드라이브를 탑재하는 경우에만 지원되며 등록 키가 필요할 수 있습니다. 등록 키를 추가하려면, 관리자 권한(관리자) 명령 프롬프트 `reg.exe add HKLM\SYSTEM\CurrentControlSet\Services\lxss /v`

`DrvFsAllowForceCaseSensitivity /t REG_DWORD /d 1`에서 명령을 사용할 수 있습니다.

변경 내용을 적용하려면 `wsl.conf` 파일을 변경한 후 WSL을 다시 시작해야 합니다. `wsl -shutdown` 명령을 사용하여 WSL을 다시 시작할 수 있습니다.

## 💡 팁

모든 드라이브에 대한 특정 대/소문자 구분 설정을 사용하여 드라이브(DrvFs 파일 시스템 플러그인)을 사용하여 /mnt/c, /mnt/d 등)에서 디스크를 사용할 수 있도록 하려면 위에서 설명한 대로 `/etc/wsl.conf` 을(를) 사용합니다. 하나의 특정 드라이브에 대한 기본 탑재 옵션을 설정하려면, `/etc/fstab` 파일 ↗을 사용하여 이러한 옵션을 지정합니다. WSL 구성 옵션에 대한 자세한 내용은 [wslconf를 사용하여 배포판별 시작 설정 구성하기](#)를 참조하십시오.

## WSL 배포에 탑재된 드라이브의 대/소문자 구분 변경

WSL 배포에 탑재된 NTFS 형식의 드라이브는 기본적으로 대/소문자를 구분하지 않습니다. WSL 배포(예: Ubuntu)에 탑재된 드라이브의 디렉터리에 대한 대/소문자 구분을 변경하려면 Windows 파일 시스템에 대해 위에 나열된 것과 동일한 단계를 따릅니다. (EXT4 드라이브는 기본적으로 대/소문자를 구분합니다.)

디렉터리에서 대/소문자 구분(FOO ≠ foo)을 사용하도록 설정하려면 다음 명령을 사용합니다.

Bash

```
fsutil.exe file setCaseSensitiveInfo <path> enable
```

디렉터리에서 대/소문자 구분을 사용하지 않도록 설정하고 대/소문자를 구분하지 않는 기본값(FOO = foo)으로 돌아가려면 다음 명령을 사용합니다.

Bash

```
fsutil.exe file setCaseSensitiveInfo <path> disable
```

### ⓘ 참고

WSL이 실행되는 동안 탑재된 드라이브에 대한 기존 디렉터리의 대/소문자 구분 플래그를 변경하는 경우, WSL에 해당 디렉터리에 대한 참조가 없는지 확인합니다. 그렇지 않으면, 변경 내용이 적용되지 않습니다. 즉, 디렉터리(또는 해당 하위 항목)를 현재 작업 디렉터리로 사용하는 것을 포함하여 WSL 프로세스에서 디렉터리를 열어서는 안 됩니다.

## Git을 사용하여 대/소문자 구분 구성

Git 버전 제어 시스템에는 작업 중인 파일의 대/소문자 구분을 조정하는 데 사용할 수 있는 구성 설정도 있습니다. Git를 사용하는 경우 [git config core.ignorecase](#) 설정을 조정할 수 있습니다.

Git를 대/소문자 구분(FOO.txt ≠ foo.txt)으로 설정하려면 다음을 입력합니다.

```
git config core.ignorecase false
```

Git를 대/소문자 구분하지 않음(FOO.txt = foo.txt)으로 설정하려면 다음을 입력합니다.

```
git config core.ignorecase true
```

대/소문자를 구분하지 않는 파일 시스템에서 이 옵션을 false로 설정하면 혼동 오류, 거짓 충돌 또는 중복 파일이 발생할 수 있습니다.

자세한 내용은 [Git 구성 설명서](#)를 참조하십시오.

## 문제 해결

**내 디렉터리에는 대/소문자가 혼합되어 대/소문자 구분이 필요한 파일이 있지만, Windows FS 도구는 이러한 파일을 인식하지 않습니다.**

Windows 파일 시스템 도구를 사용하여 대/소문자 혼합 파일이 포함된 Linux 디렉터리에서 작업하려면, 새 디렉터리를 만들고 대/소문자를 구분하도록 설정한 다음 해당 디렉터리에 파일을 복사해야 합니다(git clone 또는 untar 사용). 파일은 대/소문자가 혼합된 상태로 유지됩니다. (파일을 대/소문자를 구분하지 않는 디렉터리로 이동하려고 했지만 충돌이 발생한 경우, 덮어쓰여서 더 이상 사용할 수 없는 파일이 있을 수 있습니다.)

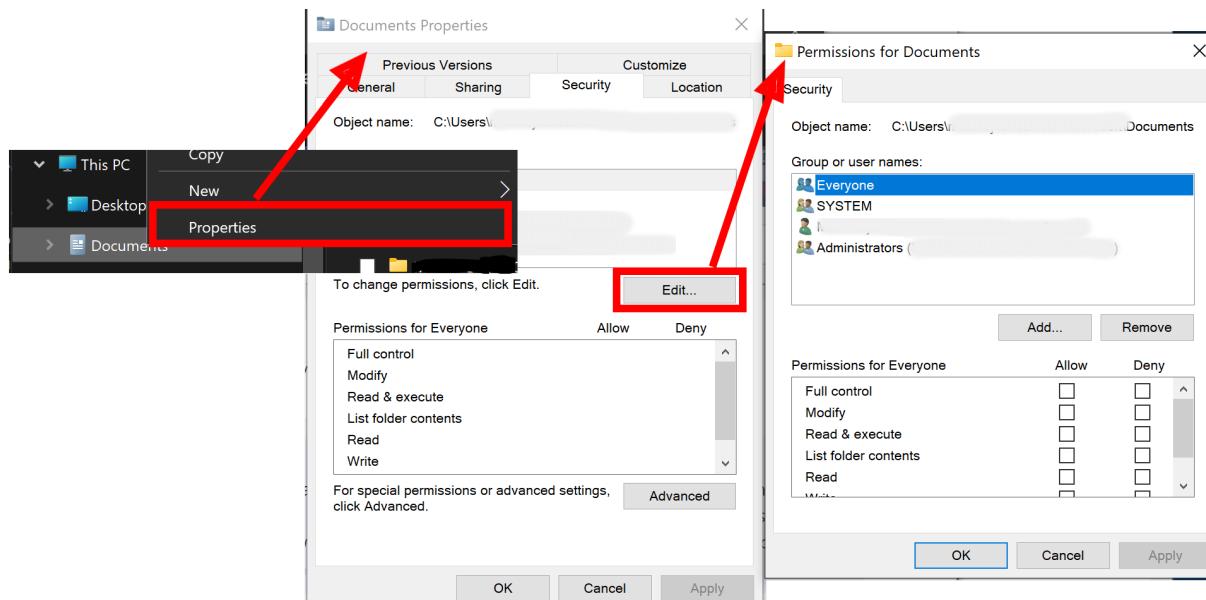
### 오류 : 디렉터리가 비어 있지 않습니다.

다른 파일 또는 디렉터리가 포함된 디렉터리에서는 대/소문자 구분 설정을 변경할 수 없습니다. 새 디렉터리를 만들고, 설정을 변경한 다음, 혼합 대/소문자 파일을 복사해 보십시오.

### 오류: 액세스 거부됨

대/소문자 구분 플래그를 변경하려면 디렉터리에 대한 "특성 쓰기", "파일 만들기", "폴더 만들기" 및 "하위 폴더 및 파일 삭제" 권한이 필요합니다. 이러한 설정을 확인하려면, Windows 파일 탐색기 디렉터리를 엽니다. (명령줄에서 `explorer.exe .` 명령을 사용합니

다.) 디렉터리를 마우스 오른쪽 단추로 클릭하고 속성을 선택하여 문서 속성 창 연 다음, 편집을 선택하여 디렉터리에 대한 권한을 보거나 변경합니다.



## 오류: 이 작업에 로컬 NTFS 볼륨이 필요합니다.

대/소문자 구분 특성은 NTFS 형식의 파일 시스템 내의 디렉터리에만 설정할 수 있습니다. WSL(Linux) 파일 시스템의 디렉터리는 기본적으로 대/소문자를 구분합니다. (fsutil.exe 도구를 사용하여 대/소문자를 구분하지 않도록 설정할 수 없습니다.)

## 추가 자료

- DevBlog: 디렉터리별 대/소문자 구분 및 WSL ↴
- DevBlog: WSL에서 디렉터리별 대/소문자 구분 지원 개선 ↴

# WSL 디스크 공간을 관리하는 방법

아티클 • 2023. 03. 21. • 읽는 데 16분 걸림

이 가이드에서는 다음을 포함하여 WSL 2를 사용하여 설치된 Linux 배포판에서 사용하는 디스크 공간을 관리하는 방법에 대해서 설명합니다.

- VHD에서 사용할 수 있는 디스크 공간의 양을 확인하는 방법
- VHD의 크기를 확장하는 방법
- 오류가 발생하는 경우 VHD를 복구하는 방법
- 설치된 Linux 배포판에 대한 .vhdx 파일 및 디스크 경로를 찾는 방법

WSL 2(Linux용 Windows 하위 시스템)는 가상화 플랫폼을 사용하여 호스트 Windows 운영 체제와 함께 Linux 배포판을 설치하고, 설치하는 각 Linux 배포판의 파일을 저장하는 VHD(가상 하드 디스크)를 만듭니다. 이러한 VHD는 [ext4 파일 시스템 형식](#)을 사용하며 Windows 하드 드라이브에 `ext4.vhdx` 파일로 표시됩니다.

WSL 2는 스토리지 요구 사항에 맞게 VHD 파일의 크기를 자동으로 조정합니다. WSL 2에서 사용하는 각 VHD 파일에는 최대 디스크 공간이 기본 1TB로 할당됩니다([WSL 릴리스 0.58.0](#) 이전에는 기본값이 최대 512GB, 그 이전에는 최대 256GB로 설정됨).

Linux 파일에 필요한 스토리지 공간이 최대 크기를 초과하면 디스크 공간이 부족하다는 오류가 표시됩니다. 이 오류를 해결하려면 [WSL 2 가상 하드 디스크의 크기를 확장하는 방법](#)에 대한 아래의 지침을 따르십시오.

## 사용 가능한 디스크 공간을 확인하는 방법

Linux `df` 명령을 사용하여 WSL 2와 함께 설치된 Linux 배포판에 대해 VHD에서 사용할 수 있는 디스크 공간의 양을 확인합니다.

사용 가능한 디스크 공간을 확인하려면 PowerShell 명령줄을 열고 이 명령을 입력합니다 (`<distribution-name>`을 실제 배포 이름으로 변경).

PowerShell

```
wsl.exe --system -d <distribution-name> df -h /mnt/wslg/distro
```

이 명령이 작동하지 않는 경우 명령을 사용하여 `wsl --update` WSL의 Store 버전으로 업그레이드하거나 를 시도 `wsl df -h /`하세요.

출력에는 다음 사항이 포함됩니다.

- **파일 시스템:** VHD 파일 시스템의 식별자

- **크기**: 디스크의 총 크기(VHD에 할당된 최대 공간의 양)
- **사용**: 현재 VHD에서 사용 중인 공간의 양
- **사용 가능**: VHD에 남아 있는 공간의 양(할당된 크기에서 사용된 양을 뺀 값)
- **사용 %**: 남은 디스크 공간의 백분율(사용/할당된 크기)
- **탑재**: 디스크가 탑재된 디렉터리 경로

VHD에 할당된 사용 가능한 디스크 공간 양에 가까워지거나 디스크 공간이 남아 있지 않아 이미 오류가 발생한 경우, Linux 배포판과 연결된 VHD에 할당된 최대 디스크 공간 크기를 확장하는 방법에 대한 단계는 다음 섹션을 참조하십시오. WSL에서 VHD에 할당된 디스크 공간의 양은 실제 Windows 디바이스의 디스크 공간 크기가 그보다 작더라도 항상 기본 최대 크기(WSL의 최신 버전에서는 1TB)를 표시합니다. WSL은 사용할 때 크기가 확장되는 VHD를 탑재하므로 Linux 배포판에서는 할당된 최대 크기인 1TB로 증가할 수 있습니다.

## WSL 2 가상 하드 디스크의 크기를 확장하는 방법

Linux 배포판에 대한 VHD 크기를 **기본 1TB의 최대 할당** 디스크 공간보다 확장하려면 아래 단계를 수행합니다. (아직 업데이트되지 않은 이전 WSL 릴리스의 경우, 최대 기본값은 512GB 또는 256GB로 설정될 수 있습니다.)

1. `wsl.exe --shutdown` 명령을 사용하여 모든 WSL 인스턴스를 종료합니다.
2. 디렉터리 경로를 컴퓨터에 설치된 Linux 배포판과 연관된 `ext4.vhdx` 파일로 복사합니다. 도움말은 [Linux 배포판에 대한 vhdx 파일 및 디스크 경로를 찾는 방법](#)을 참조하십시오.
3. 관리자 권한으로 Windows 명령 프롬프트를 열고 다음을 입력하여 `diskpart` 명령 인터프리터를 엽니다.

Windows 명령 프롬프트

`diskpart`

4. 이제 `DISKPART>` 프롬프트가 표시됩니다. 다음 명령을 입력하고 `<pathToVHD>`을 Linux 배포판과 연관된 `ext4.vhdx` 파일의 디렉터리 경로(2단계에서 복사된 경로)로 변경합니다.

Windows 명령 프롬프트

`Select vdisk file="<pathToVHD>"`

5. VHD가 할당된 현재 최대 크기를 나타내는 **가상 크기**를 포함하여 가상 디스크와 연관되는 세부 정보를 표시합니다.

```
Windows 명령 프롬프트
```

```
detail vdisk
```

6. **가상 크기**를 메가바이트로 변환해야 합니다. 예를 들어, **가상 크기: 512 GB**인 경우, **512000**로 변환합니다. 입력한 새 값은 이 원래 값보다 커야 합니다. 예를 들어, 가상 크기를 512GB에서 1024GB로 두 배로 늘리려면 MB로 변환하고 **1024000**의 값을 입력합니다. 가상 디스크 크기를 줄이는 프로세스가 훨씬 더 복잡하기 때문에 실제로 원하는 것보다 높은 값을 입력하지 않도록 주의하십시오.

7. Windows 명령 프롬프트 **DISKPART>** 프롬프트를 사용하여 이 Linux 배포판에 할당하려는 새 최대 크기 값을 입력하십시오.

```
Windows 명령 프롬프트
```

```
expand vdisk maximum=<sizeInMegabytes>
```

8. **DISKPART>** 프롬프트를 종료합니다.

```
Windows 명령 프롬프트
```

```
exit
```

9. 이 Linux 배포판을 시작합니다. (*WSL 2에서 실행 중인지 확인합니다.* `wsl.exe -l -v` 명령을 사용하여 확인할 수 있습니다. WSL 1은 지원되지 않습니다.)

10. WSL 배포 명령줄에서 다음 명령을 실행하여 WSL에서 파일 시스템의 크기를 확장할 수 있음을 알려줍니다. 첫 번째 **mount** 명령에 대한 응답으로 "/dev: none already mounted on /dev"와 같은 메시지가 표시될 수 있습니다. 이 메시지는 무시해도 됩니다.

```
Bash
```

```
sudo mount -t devtmpfs none /dev  
mount | grep ext4
```

11. 이 항목의 이름(예: `/dev/sdX`)을 복사합니다(여기서 X는 다른 문자를 나타냄). 다음 예에서 X의 값은 **b**입니다.

```
Bash
```

```
sudo resize2fs /dev/sdb <sizeInMegabytes>M
```

위의 예를 사용하여 VHD 크기를 2048000으로 변경했기 때문에 명령은 `sudo resize2fs /dev/sdb 2048000M`입니다.

### ① 참고

`resize2fs`를 설치해야 할 수도 있습니다. 이 경우 `sudo apt install resize2fs` 명령을 사용하여 설치할 수 있습니다.

출력은 다음과 비슷합니다.

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.
```

Linux 배포판에 대한 가상 드라이브(ext4.vhdx)가 새 크기로 확장되었습니다.

### ① 중요

Windows 도구 또는 편집기를 사용하여 AppData 폴더 내에 있는 WSL 관련 파일을 수정, 이동하거나 액세스하지 않는 것이 좋습니다. 이렇게 하면 Linux 배포가 손상될 수 있습니다. Windows에서 `\wsl$\<distribution-name>\` 경로를 통해 Linux 파일에 액세스할 수 있습니다. WSL 배포를 열고 `explorer.exe .`를 입력하여 해당 폴더를 확인합니다. 자세히 알아보려면 블로그 게시물 [Windows에서 Linux 파일 액세스](#)를 참조하세요.

## VHD 탑재 오류를 복구하는 방법

"배포 디스크 탑재"와 관련된 오류가 발생하는 경우, 이는 갑작스런 종료 또는 정전으로 인해 발생할 수 있으며 데이터 손실을 방지하기 위해 Linux 배포 VHD가 읽기 전용으로 전환될 수 있습니다. 아래 단계에 따라 `e2fsck` Linux 명령을 사용하여 배포를 복구하고 복원할 수 있습니다.

## lsblk 명령을 사용하여 차단 디바이스 이름 식별하기

WSL 2에서 Linux 배포를 설치하면 배포판이 자체 파일 시스템으로 VHD(가상 하드 디스크)로 탑재됩니다. Linux에서 이러한 하드 드라이브를 "차단 디바이스"라고 하며, `lsblk` 명령을 사용하여 해당 드라이브에 대한 정보를 볼 수 있습니다.

현재 WSL 2에서 사용 중인 차단 디바이스의 이름을 찾으려면 배포를 열고 `lsblk` 명령을 입력합니다. (또는 PowerShell을 열고 `wsl.exe lsblk` 명령을 입력합니다.) 출력은 다음과 같이 표시됩니다.

```
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
```

차단 디바이스에 대한 정보는 다음과 같습니다.

- **이름**: 디바이스에 할당된 이름은 `sd[a-z]`이며, 사용되는 각 디스크에 대한 문자 지정이 있는 SCSI 디스크를 참조합니다. `sda`은(는) 항상 시스템 배포입니다.
- **MAJ:MIN**: 디바이스 유형을 나타내는 첫 번째 번호로 디바이스를 내부적으로 식별하기 위해서 Linux 커널에서 사용하는 숫자를 나타냅니다. (8은 소형 컴퓨터 시스템 인터페이스/SCSI 디스크에 사용됩니다.)
- **RM**: 디바이스가 이동식인지 여부(이동식(1), 이동식 아님(0))를 알려줍니다.
- **크기**: 전체 크기입니다.
- **RO**: 디바이스가 읽기 전용인지 여부(읽기 전용(1), 읽기 전용 아님(0))를 알려줍니다.
- **유형**: 디바이스 유형(이 경우에는 디스크)을 참조합니다.
- **탑재 지점**: 차단 디바이스가 있는 파일 시스템의 현재 디렉터리를 나타냅니다. (SWAP은 미리 구성된 비활성 메모리용이므로 탑재 지점이 없습니다.)

## 읽기 전용 대체 오류

Linux 배포를 열 때 WSL에 "탑재 오류"가 발생하면 배포가 대체(fallback)로 읽기 전용으로 설정될 수 있습니다. 이 경우 시작 중에 배포에 다음 오류가 표시될 수 있습니다.

```
PowerShell

An error occurred mounting the distribution disk, it was mounted read-only
as a fallback.
```

배포가 읽기 전용으로 시작되면 파일 시스템에 쓰려고 하는 모든 시도가 실패하고 다음과 같은 오류가 발생합니다.

Bash

```
$ touch file  
touch: cannot touch 'file': Read-only file system
```

WSL에서 디스크 탑재 오류를 복구하고 사용 가능/쓰기 가능한 상태로 다시 복원하려면 `wsl.exe --mount` 명령을 사용하여 다음 단계를 수행하여 디스크를 다시 탑재할 수 있습니다.

1. PowerShell을 열고 명령을 입력하여 모든 WSL 배포를 종료합니다.

PowerShell

```
wsl.exe --shutdown
```

2. 관리자 권한(관리자 권한 명령 프롬프트)으로 PowerShell을 열고 탑재 명령을 입력하여 <path-to-ext4.vhdx> 을(를) 배포판의 .vhdx 파일 경로로 바꿉니다. 이 파일을 찾는 방법에 대한 도움말은 [Linux 배포에 대한 VHD 파일 및 디스크 경로를 찾는 방법](#)을 참조하십시오.

PowerShell

```
wsl.exe --mount <path-to-ext4.vhdx> --vhd --bare
```

3. PowerShell의 `wsl.exe lsblk` 명령을 사용하여 배포에 대한 블록 디바이스 이름 (`sd[a-z]`)을 식별한 다음, 다음 명령을 입력하여 디스크를 복구합니다(<device> 을 (를) "sdc"와 같은 올바른 블록 디바이스 이름으로 대체). `e2fsck` 명령은 ext4 파일 시스템(WSL과 함께 설치된 배포에서 사용하는 형식)에서 오류를 확인하고 그에 따라 복구합니다.

PowerShell

```
wsl.exe sudo e2fsck -f /dev/<device>
```

4. 복구가 완료되면 다음을 입력하여 PowerShell에서 디스크를 분리합니다.

PowerShell

```
wsl.exe --unmount
```

⚠ 경고

`sudo mount -o remount,rw /` 명령을 사용하여 읽기 전용 배포를 사용 가능/쓰기 가능한 상태로 반환할 수 있지만, 모든 변경 내용은 메모리 내이므로 배포가 다시 시작될 때 손실됩니다. 그 대신, 위에 나열된 단계를 사용하여 디스크를 탑재하고 복구하는 것이 좋습니다.

## Linux 배포판에 대한 .vhdx 파일 및 디스크 경로를 찾는 방법

Linux 배포에 대한 .vhdx 파일 및 디렉터리 경로를 찾으려면 PowerShell을 열고 다음 스크립트를 사용하여 <distribution-name>을(를) 실제 배포 이름으로 바꿉니다.

PowerShell

```
(Get-ChildItem -Path HKCU:\Software\Microsoft\Windows\CurrentVersion\Lxss |  
Where-Object { $_.GetValue("DistributionName") -eq '<distribution-name>'  
}).GetValue("BasePath") + "\ext4.vhdx"
```

결과는 %LOCALAPPDATA%\Packages\<PackageFamilyName>\LocalState\<disk>.vhdx 와(과) 같은 경로를 표시합니다. 예를 들면 다음과 같습니다.

PowerShell

```
C:\Users\User\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_7  
9rhkp1fndgsc\LocalState\ext4.vhdx
```

나열한 Linux 배포와 연관된 ext4.vhdx 파일에 대한 경로입니다.

# Linux용 Windows 하위 시스템에 대한 질문과 대답

FAQ

## 일반

### WSL(Linux용 Windows 하위 시스템)이란?

WSL(Linux용 Windows 하위 시스템)은 기존 Windows 데스크톱 및 앱과 함께 Windows에서 직접 Linux 명령줄 도구 및 GUI 앱과 함께 Linux 파일 시스템을 실행할 수 있는 Windows 운영 체제의 기능입니다.

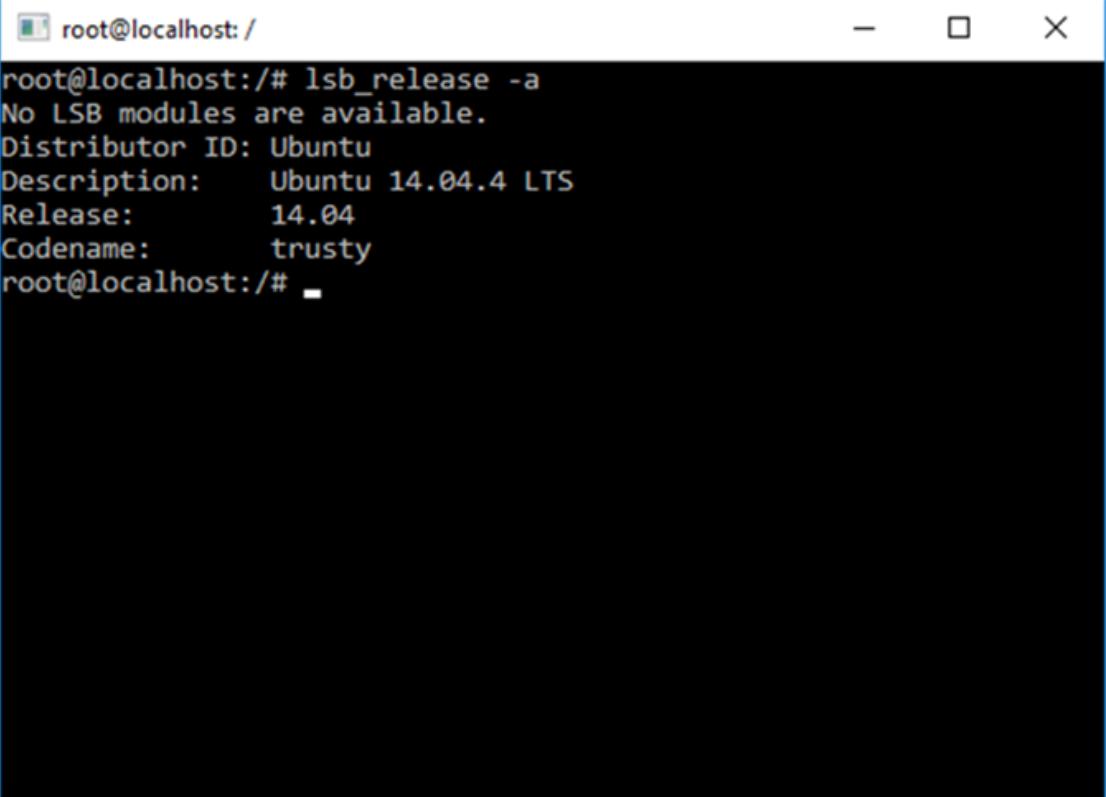
자세한 내용은 [정보 페이지](#)를 참조하세요.

### WSL은 누구를 위한 것인가요?

이는 주로 개발자, 특히 웹 개발자, 오픈 소스 프로젝트에서 작업하거나 Linux 서버 환경에 배포하는 개발자를 위한 도구입니다. WSL은 Bash, 일반적인 Linux 도구(`sed`, `awk` 등) 및 Linux 우선 프레임워크(Ruby, Python 등)를 사용하는 것을 좋아하지만, Windows 생산성 도구를 사용하는 것을 좋아하는 모든 사용자를 위한 것입니다.

### WSL로 무엇을 할 수 있나요?

WSL을 사용하면 배포(Ubuntu, Debian, OpenSUSE, Kali, Alpine 등)를 선택하여 Bash 셸에서 Linux를 실행할 수 있습니다. Bash를 사용하면 Linux 명령줄 도구 및 앱을 실행할 수 있습니다. 예를 들어 `lsb_release -a`를 입력하고 Enter 키를 누릅니다. 그러면 현재 실행되고 있는 Linux 배포판의 세부 정보가 표시됩니다.



A screenshot of a terminal window titled "root@localhost:/". The window contains the output of the command "lsb\_release -a". The output shows the following details:

```
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:/#
```

또한 Linux Bash 셸 내에서 로컬 머신의 파일 시스템에 액세스할 수 있습니다. 이 경우 로컬 드라이브는 `/mnt` 폴더 아래에 탑재되어 있습니다. 예를 들어 `c:` 드라이브가 `/mnt/c` 아래에 탑재되어 있습니다.

```
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 ../
-rw-rwxrwx 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/
-rw-rwxrwx 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:/#
```

## WSL이 통합된 일반적인 개발 워크플로를 설명할 수 있나요?

WSL은 내부 개발 루프의 일부로 사용하려는 개발자를 대상으로 합니다. Sam이 CI/CD 파이프라인(Continuous Integration & Continuous Delivery)을 만들고 있으며 클라우드에 배포하기 전에 로컬 머신(랩톱)에서 먼저 테스트하려 한다고 가정해 보겠습니다. Sam은 WSL(& WSL 2)을 사용하여 속도와 성능을 향상시킨 다음, 원하는 Bash 명령어와 도구를 사용하여 로컬(랩톱 상의)에서 정품 Linux Ubuntu 인스턴스를 사용할 수 있습니다. 개발 파이프라인이 로컬에서 확인되면 Sam은 해당 CI/CD 파이프라인을 Docker 컨테이너로 만들고 프로덕션 준비가 된 Ubuntu VM에서 실행되는 클라우드 인스턴스로 푸시하여 이 컨테이너를 클라우드(예: Azure)까지 푸시할 수 있습니다.

## Bash란?

Bash<sup>↗</sup>는 널리 사용되는 텍스트 기반 셸 및 명령 언어입니다. 이는 Ubuntu 및 기타 Linux 배포판에 포함된 기본 셸입니다. 사용자가 명령을 셸에 입력하여 스크립트를 실행하거나 명령과 도구를 실행하여 많은 작업을 수행합니다.

## 어떻게 작동하나요?

Windows 명령줄 블로그에서 이 문서를 확인하십시오. [WSL을 통해 Windows가 Linux 파일에 액세스할 수 있는 방법에 대한 자세한 내용](#)에서 기본 기술에 대해서 자세히 설명합니다.

## VM에서 Linux 대신 WSL을 사용하는 이유는 무엇인가요?

WSL에는 전체 가상 머신보다 적은 리소스(CPU, 메모리 및 스토리지)가 필요합니다. 또한 WSL을 사용하면 Windows 명령줄, 데스크톱 및 스토어 앱과 함께 Linux 명령줄 도구 및 앱을 실행하고 Linux 내에서 Windows 파일에 액세스할 수 있습니다. 이렇게 하면 원하는 경우 동일한 파일 세트에서 Windows 앱 및 Linux 명령줄 도구를 사용할 수 있습니다.

## 예를 들어 Windows 대신 Linux에서 Ruby를 사용하는 이유는 무엇인가요?

일부 플랫폼 간 도구는 이러한 도구가 실행되는 환경이 Linux처럼 동작한다고 가정하여 구축되었습니다. 예를 들어 일부 도구에서는 매우 긴 파일 경로에 액세스할 수 있거나 특정 파일/폴더가 있다고 가정합니다. 이로 인해 Windows에서 Linux와 다르게 동작하는 문제가 발생하는 경우가 많습니다.

Ruby 및 Node.js와 같은 많은 언어는 Windows에서 이식되어 매우 효율적으로 실행되는 경우가 많습니다. 그러나 일부 Ruby Gem 또는 Node/NPM 라이브러리 소유자는 Windows를 지원하기 위해 라이브러리를 이식하지 않으며 많은 소유자가 Linux 관련 종속성을 사용하고 있습니다. 이로 인해 이러한 도구와 라이브러리를 사용하여 빌드된 시스템의 Windows에서 빌드 및 때로는 런타임 오류 또는 원치 않는 동작이 발생하는 경우가 많습니다.

이러한 문제 중 일부는 Windows의 명령줄 도구를 개선하도록 많은 사용자가 Microsoft에 요청하게 하였으며, Windows에서 네이티브 Bash 및 Linux 명령줄 도구를 실행할 수 있도록 하기 위해 Microsoft에서 Canonical과 파트너 관계를 맺게 하였습니다.

## PowerShell에서 의미하는 것은 무엇인가요?

OSS 프로젝트에서 작업하는 동안 PowerShell 프롬프트에서 Bash를 사용하는 것이 매우 유용한 시나리오가 많이 있습니다. Bash 지원은 보완적이며 Windows에서 명령줄의 가치를 강화하여 PowerShell 및 PowerShell 커뮤니티에서 널리 사용되는 다른 기술을 활용할 수 있도록 합니다.

자세한 내용은 [Windows용 Bash: 뛰어난 기능 및 PowerShell에 대한 의미는 무엇일까요?](#)라는 PowerShell 팀 블로그를 참조하세요.

# WSL에서 지원하는 프로세서는 무엇인가요?

WSL은 x64 및 ARM CPU를 지원합니다.

## 내 C: 드라이브에는 어떻게 액세스하나요?

로컬 머신의 하드 드라이브에 대한 탑재 지점이 자동으로 만들어져 Windows 파일 시스템에 쉽게 액세스할 수 있습니다.

/mnt/<드라이브 문자>/

예를 들어 c:\에 액세스하려면 `cd /mnt/c`를 사용합니다.

## Git 자격 증명 관리자는 어떻게 설정하나요? (WSL에서 내 Windows Git 권한을 어떻게 사용하나요?)

Git 자격 증명 관리자를 설정하고 Windows 자격 증명 관리자에 인증 토큰을 저장하는 방법에 대한 섹션을 제공하는 [Linux용 Windows 하위 시스템 Git 사용 시작하기에 대한 자습서](#)를 참조하십시오.

## Linux 앱에서 Windows 파일을 사용하려면 어떻게 하나요?

WSL의 이점 중 하나는 Windows 및 Linux 앱 또는 도구를 통해 파일에 액세스할 수 있다는 것입니다.

WSL은 머신의 고정 드라이브를 Linux 배포판의 `/mnt/<drive>` 폴더 아래에 탑재합니다. 예를 들어 c: 드라이브는 `/mnt/c/` 아래에 탑재됩니다.

탑재된 드라이브를 사용하면 [Visual Studio](#) 또는 [VS Code](#)를 사용하여 `c:\dev\myproj\`에서 코드를 편집하고, `/mnt/c/dev/myproj`를 통해 동일한 파일에 액세스하여 Linux에서 해당 코드를 빌드/테스트할 수 있습니다.

[Windows 및 Linux 파일 시스템 간 작업 문서](#)에서 자세히 알아보십시오.

## Linux 드라이브의 파일이 탑재된 Windows 드라이브와 다른가요?

1. Linux 루트(예: `/`) 아래의 파일은 다음을 포함하지만 이에 국한되지 않는 Linux 특정 동작을 모방하는 WSL에서 제어됩니다.

- 잘못된 Windows 파일 이름 문자가 포함된 파일
- 관리자가 아닌 사용자에 대해 만들어진 symlink
- chmod 및 chown을 통한 파일 특성 변경
- 파일/폴더 대/소문자 구분

2. 탑재된 드라이브의 파일은 Windows에서 제어하며 다음과 같은 동작을 수행합니다.

- 대/소문자 구분 지원
- 모든 권한이 Windows 권한을 가장 잘 반영하도록 설정됨

## WSL 배포를 제거하려면 어떻게 하나요?

WSL에서 배포를 제거하고 해당 Linux 배포와 연결된 모든 데이터를 삭제하려면, `ws1 --unregister <distroName>` 을(를) 실행합니다. 여기서 `<distroName>` 은(는) `ws1 -l` 명령의 목록에서 볼 수 있는 Linux 배포판의 이름입니다.

다른 스토어 애플리케이션과 마찬가지로 Windows에서 Linux 배포판 앱을 제거할 수도 있습니다.

WSL 명령에 대한 자세한 내용은 [WSL에 대한 기본 명령](#) 문서를 참조하십시오.

## OpenSSH 서버를 실행하려면 어떻게 하나요?

OpenSSH는 선택적 기능으로 Windows와 함께 제공됩니다. [OpenSSH 설치](#) 문서를 참조하십시오. WSL에서 OpenSSH를 실행하려면 Windows의 관리자 권한이 필요합니다. OpenSSH 서버를 실행하려면 WSL 배포(예: Ubuntu) 또는 Windows 터미널을 관리자로 실행합니다. WSL을 사용하는 SSH 시나리오를 다루는 여러 리소스가 있습니다. Scott Hanselman의 블로그 문서: [Linux 또는 Windows에 SSH를 적용하는 방법](#), [외부 컴퓨터에서 Windows 10 WSL2로 SSH를 적용하는 방법](#), [외부 컴퓨터에서 Windows 10의 Bash 및 WSL2로 SSH를 적용하는 쉬운 방법](#), [Windows 10의 기본 제공 OpenSSH를 사용하여 원격 Linux 컴퓨터에 자동으로 SSH를 적용하는 방법](#)을 확인합니다.

## WSL의 표시 언어를 변경하려면 어떻게 하나요?

WSL 설치는 Windows 설치의 로캘과 일치하도록 Ubuntu 로캘을 자동으로 변경하려고 합니다. 이 동작을 원하지 않는 경우 설치가 완료된 후 다음 명령을 실행하여 Ubuntu 로 캠을 변경할 수 있습니다. 이 변경 내용이 적용되려면 WSL 배포를 다시 시작해야 합니다.

아래 예제에서는 로캘을 en-US로 변경합니다.

Bash

```
sudo update-locale LANG=en_US.UTF8
```

# WSL에서 인터넷에 액세스할 수 없는 이유는 무엇인가요?

일부 사용자가 WSL에서 인터넷 액세스를 차단하는 특정 방화벽 애플리케이션의 문제를 보고했습니다. 보고된 방화벽은 다음과 같습니다.

1. Kaspersky
2. AVG
3. Avast
4. Symantec Endpoint Protection
5. F-Secure

방화벽을 해제하면 액세스가 허용되는 경우도 있습니다. 단순히 방화벽을 설치하면 경우에 따라 액세스가 차단되는 것처럼 보입니다.

# Windows의 WSL에서 포트에 액세스하려면 어떻게 하나요?

WSL은 Windows에서 실행되는 Windows의 IP 주소를 공유합니다. 따라서 localhost의 모든 포트에 액세스할 수 있습니다. 예를 들어 1234 포트에 웹 콘텐츠가 있는 경우 <https://localhost:1234> 를 Windows 브라우저에 연결할 수 있습니다. 자세한 내용은 [네트워크 애플리케이션 액세스](#)를 참조하십시오.

# 내 WSL 배포판을 백업하거나 한 드라이브에서 다른 드라이브로 이동할 수 있나요?

배포판을 백업하거나 이동하는 가장 좋은 방법은 Window 버전 1809 이상에서 사용할 수 있는 [내보내기/가져오기 명령](#)을 사용하는 것입니다. `wsl --export` 명령을 사용하여 전체 배포를 tarball로 내보낼 수 있습니다. 그런 다음, `wsl --import` 명령을 사용하여 이 배포판을 WSL로 다시 가져올 수 있습니다. 이 명령을 사용하면 가져올 새 드라이브 위치의 이름을 지정할 수 있으므로 WSL 배포 상태를 백업, 저장하거나 이동할 수 있습니다.

AppData 폴더의 파일을 백업하는 기존 백업 서비스(예: Windows Backup)는 Linux 파일을 손상시키지 않습니다.

# 프로덕션 시나리오에 WSL을 사용할 수 있습니까?

WSL은 내부 루프 개발 워크플로와 함께 사용하도록 설계되었습니다. WSL에는 이러한 용도에 적합하지만 다른 제품에 비해 프로덕션 관련 시나리오에 어려운 디자인 기능이 있

습니다. WSL이 일반 VM 환경과 어떻게 다른지 명확히 하여 비즈니스 요구 사항에 맞는지 여부를 결정할 수 있도록 하는 것이 목표입니다.

WSL과 기존 프로덕션 환경 간의 주요 차이점은 다음과 같습니다.

- WSL에는 리소스를 자동으로 시작, 중지 및 관리하는 경량 유ти리티 VM이 있습니다.
- Windows 프로세스에 대한 열린 파일 핸들이 없는 경우, WSL VM이 자동으로 종료됩니다. 즉, 웹 서버로 SSH를 사용하여 서버를 실행한 다음 종료하는 경우, VM은 사용자가 사용을 완료하고 리소스를 정리한다는 것을 감지하기 때문에 VM이 종료될 수 있습니다.
- WSL 사용자는 Linux 인스턴스에 대한 모든 권한을 갖습니다. VM의 수명, 등록된 WSL 배포 등은 모두 사용자가 액세스할 수 있으며 사용자가 수정할 수 있습니다.
- WSL은 Windows 파일에 대한 파일 액세스 권한을 자동으로 제공합니다.
- Windows 경로는 기본적으로 경로에 추가되어 기존 Linux 환경에 비해 특정 Linux 애플리케이션에 대해 예기치 않은 동작이 발생할 수 있습니다.
- WSL은 Linux에서 Windows 실행 파일을 실행할 수 있으며, 이는 기존 Linux VM과 다른 환경으로 이어질 수도 있습니다.
- WSL에서 사용하는 Linux 커널이 자동으로 업데이트됩니다.
- WSL의 GPU 액세스는 GPU 호출을 Windows GPU로 라우팅하는 `/dev/dxg` 디바이스를 통해 발생합니다. 이 설정은 기존 Linux 설정과 다릅니다.
- 운영 체제 미설치 Linux에 비해 다른 작은 차이가 있으며, 내부 루프 개발 워크플로의 우선 순위가 지정됨에 따라 향후 더 많은 차이가 발생할 것으로 예상됩니다.

## 한 컴퓨터에서 다른 컴퓨터로 WSL 파일을 전송하려면 어떻게 해야 하나요?

이 작업을 수행할 수 있는 몇 가지 방법이 있습니다.

- 가장 쉬운 방법은 명령을 사용하여 `wsl --export --vhd` WSL 배포를 VHD 파일로 보내는 것입니다. 그런 다음, 이 파일을 다른 컴퓨터에 복사하고 이를 사용하여 `wsl --import --vhd` 가져올 수 있습니다. 자세한 내용은 [명령 문서를](#) 참조하세요.
- 위의 구현에는 많은 디스크 공간이 필요합니다. 디스크 공간이 많지 않은 경우 Linux 기술을 사용하여 파일을 이동할 수 있습니다.
  - 를 사용하여 `tar -czf <tarballName> <directory>` 파일의 tarball을 만듭니다. 그런 다음 이러한 특정 파일을 새 컴퓨터에 복사하고 이를 실행 `tar -xzf <tarballName>` 하여 추출할 수 있습니다.
  - 다음과 같은 `dpkg --get-selections | grep -v deinstall | awk '{print $1}' > package_list.txt` 명령을 사용하여 `apt` 설치된 패키지 목록을 내보낸 다음 파일

을 전송한 후 같은 `sudo apt install -y $(cat package_list.txt)` 명령을 사용하여 다른 컴퓨터에 동일한 패키지를 다시 설치할 수도 있습니다.

## WSL 2

### WSL 2가 Hyper-V를 사용하나요? Windows 10 Home 및 Windows 11 Home에서 사용할 수 있나요?

WSL 2는 Windows 10 Home과 Windows 11 Home을 포함하여 WSL이 현재 제공되는 모든 데스크톱 SKU에서 사용할 수 있습니다.

최신 버전의 WSL은 Hyper-V 아키텍처를 사용하여 가상화를 지원합니다. 이 아키텍처는 'Virtual Machine 플랫폼' 옵션 구성 요소에서 사용할 수 있습니다. 이 옵션 구성 요소는 모든 SKU에서 사용할 수 있습니다. 이 환경에 대한 자세한 내용은 WSL 2 릴리스를 앞두고 제공될 예정입니다.

### WSL 1은 어떻게 되나요? 중단되나요?

현재 WSL 1을 사용 중단할 계획은 없습니다. WSL 1과 WSL 2 배포판을 함께 실행하고 원하는 배포판을 언제든지 업그레이드하고 다운그레이드할 수 있습니다. WSL 2를 새 아키텍처로 추가하면 WSL 팀이 더 나은 플랫폼을 통해 Windows에서 Linux 환경을 원활하게 실행하는 데 필요한 기능을 제공할 수 있습니다.

### WSL 2 및 기타 타사 가상화 도구(예: VMware 또는 VirtualBox)를 실행할 수 있나요?

Hyper-V를 사용 중인 경우 일부 타사 애플리케이션을 작동할 수 없습니다. 즉, VMware 및 VirtualBox와 같이 WSL 2를 사용하도록 설정된 경우에는 일부 타사 애플리케이션이 실행되지 않습니다. 그러나 최근에 VirtualBox와 VMware는 모두 Hyper-V 및 WSL2를 지원하는 버전을 릴리스했습니다! [VirtualBox 변경 내용](#) 및 [VMware 변경 내용](#)에서 자세히 알아보세요. 문제 해결은 [GitHub의 WSL 리포지토리](#)에 있는 [VirtualBox 문제 토론](#)을 참조하세요.

저희는 타사의 Hyper-V 통합을 지원하기 위해 지속적으로 솔루션을 개발하고 있습니다. 예를 들어 타사 가상화 공급자가 Hyper-V와 호환되는 소프트웨어를 만드는 데 사용할 수 있는 [하이퍼바이저 플랫폼](#)이라는 API 세트를 제공합니다. 이를 통해 애플리케이션은 현재 Hyper-V와 호환되는 VirtualBox 6 이상 버전과 [Google Android Emulator](#) 같은 에뮬레이션에 Hyper-V 아키텍처를 사용할 수 있습니다.

[VirtualBox 6.1의 WSL 2 문제](#)에 대한 자세한 배경 및 논의는 WSL 문제 리포지토리를 참조하십시오.

\*Windows 가상 머신을 찾는 경우, [Windows 개발자 센터](#)에서 VMWare, Hyper-V, VirtualBox 및 Parallels VM 다운로드를 사용할 수 있습니다.

## WSL 2의 GPU에 액세스할 수 있나요? 하드웨어 지원을 늘릴 계획이 있나요?

WSL 2 배포판 내부의 GPU 액세스 지원을 릴리스했습니다. 즉, 이제 빅 데이터 세트가 관련될 때 기계 학습, AI 및 데이터 과학 시나리오에 대해 WSL을 더 쉽게 사용할 수 있습니다. [GPU 지원 시작](#) 자습서를 확인하세요. 현재 WSL 2에는 직렬 지원 또는 USB 디바이스 지원이 포함되어 있지 않습니다. 이러한 기능을 추가하는 가장 좋은 방법을 찾고 있습니다. 그러나, USB 지원은 이제 USBIPD-WIN 프로젝트를 통해 사용할 수 있습니다. USB 디바이스 지원을 설정하는 단계는 [USB 디바이스 연결](#)을 참조하십시오.

## WSL 2가 네트워킹 애플리케이션을 사용할 수 있나요?

예, 일반적인 네트워킹 애플리케이션은 전체 시스템 호출 호환성을 제공하므로 WSL 2에서 잘 작동하고 훨씬 빠릅니다. 그러나, WSL 2 아키텍처는 가상화된 네트워킹 구성 요소를 사용합니다. 즉, WSL 2는 가상 머신과 유사하게 작동합니다. WSL 2 배포판은 호스트 컴퓨터(Windows OS)와 다른 IP 주소를 갖습니다. 자세한 내용은 [WSL에서 네트워크 애플리케이션 액세스](#)를 참조하십시오.

## 가상 머신에서 WSL 2를 실행할 수 있나요?

예! 가상 머신에 중첩된 가상화가 사용 설정되었는지 확인해야 합니다. PowerShell 창에서 관리자 권한으로 다음 명령을 실행하면 부모 Hyper-V 호스트에서 이를 사용하도록 설정할 수 있습니다.

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

'<VMName>'을 가상 머신 이름으로 바꾸세요.

## WSL 2에서 wsl.conf를 사용할 수 있나요?

WSL 2는 WSL 1에서 사용하는 것과 동일한 wsl.conf 파일을 지원합니다. 즉, WSL 1 배포판에서 설정한 모든 구성 옵션(예: Windows 드라이브 자동 탑재, interop 사용 또는 사용 안 함, Windows 드라이브가 탑재될 디렉터리 변경 등)이 모두 WSL 2 내에서 작동합니다. [배포판 관리](#) 페이지에서 WSL의 구성 옵션에 대해 자세히 알아볼 수 있습니다. [WSL 2에서](#)

[Linux 디스크 탑재](#) 문서에서 드라이브, 디스크, 디바이스 또는 VHD(가상 하드 디스크) 탑재 지원에 대해서 자세히 알아봅니다.

## 피드백을 어디에 제공할 수 있나요?

[WSL 제품 리포지토리 문제](#) 를 통해 다음을 수행할 수 있습니다.

- **기존 문제를 검색**하여 현재 발생하는 문제와 관련된 문제가 있는지 확인합니다. 검색 창에서 "is:open"을 제거하여 이미 해결된 문제를 검색에 포함할 수 있습니다. 미 해결 문제 중에서 우선적으로 해결되기를 원하는 관심을 표시하려는 문제에 주석을 달거나 좋아요를 눌러 주시기 바랍니다.
- **새 문제를 제출합니다.** WSL에서 문제를 발견했는데 기존 문제가 아닌 것으로 보이는 경우 녹색 새 문제 단추를 선택한 다음, WSL - 버그 보고서를 선택하면 됩니다. 문제의 제목, Windows 빌드 번호(현재 빌드 번호를 보려면 cmd.exe /c ver 실행), WSL 1 또는 2 중에 현재 실행하고 있는 버전, 현재 Linux 커널 버전 번호(wsl1.exe --status 또는 cat /proc/version 실행), 배포판의 버전 번호(lsb\_release -r 실행), 관련된 기타 소프트웨어 버전, 재현 단계, 예상 동작, 실제 동작 및 진단 로그(사용 가능하고 적절한 경우)를 포함해야 합니다. 자세한 내용은 [WSL에 기여](#) 를 참조하세요.
- 녹색 새 문제 단추를 선택하고 기능 요청을 선택하여 기능 요청을 제출합니다. 요청을 설명하는 몇 가지 질문에 답해야 합니다.

다음도 가능합니다.

- [WSL 문서 리포지토리](#) 를 사용하여 설명서 문제를 제출합니다. WSL 문서에 기여하면 [Microsoft Docs 기여자 가이드](#)를 참조하세요.
- 문제가 Windows 터미널, Windows 콘솔 또는 명령줄 UI와 좀 더 관련이 있는 경우 [Windows 터미널 제품 리포지토리](#) 를 사용하여 Windows 터미널 문제를 제출합니다.

최신 WSL 뉴스를 최신 상태로 유지하려면 다음을 수행하세요.

- [명령줄 팀 블로그](#)
- Twitter. Twitter에서 [@craigaloewen](#) 을 팔로우하여 뉴스, 업데이트 등에 대해서 알아보십시오.

# Linux용 Windows 하위 시스템 문제 해결

아티클 • 2023. 03. 21. • 읽는 데 40분 걸림

아래에 WSL과 관련된 몇 가지 일반적인 문제 해결 시나리오가 나와 있지만, [GitHub의 WSL 제품 리포지토리](#)에 제출된 문제도 검색해 보는 것이 좋습니다.

## 문제, 버그 보고서, 기능 요청 제출

[WSL 제품 리포지토리 문제](#)를 통해 다음을 수행할 수 있습니다.

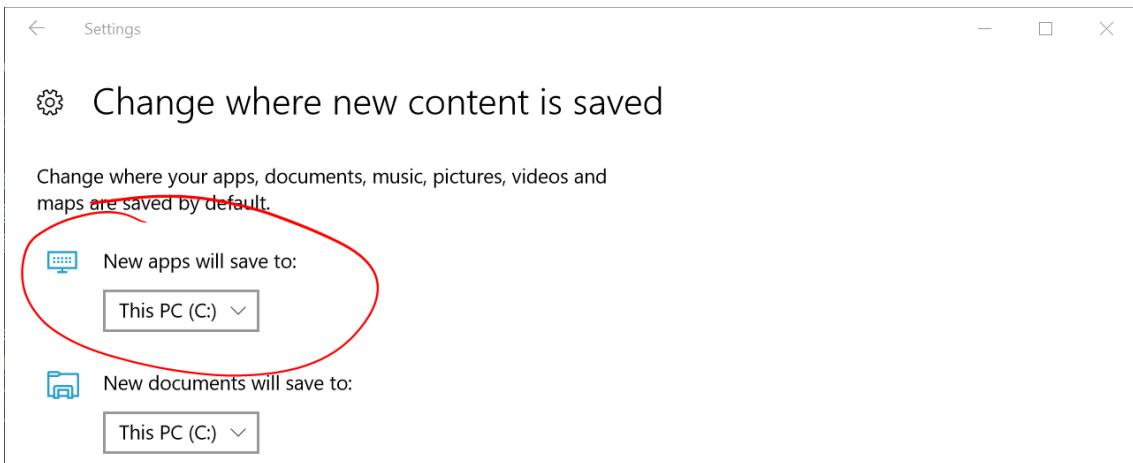
- **기존 문제를 검색**하여 현재 발생하는 문제와 관련된 문제가 있는지 확인합니다. 검색 창에서 "is:open"을 제거하여 이미 해결된 문제를 검색에 포함할 수 있습니다. 미 해결 문제 중에서 우선적으로 해결되기를 원하는 관심을 표시하려는 문제에 주석을 달거나 좋아요를 눌러 주시기 바랍니다.
- **새 문제를 제출합니다.** WSL에서 문제를 발견했는데 기존 문제가 아닌 것으로 보이는 경우 녹색 새 문제 단추를 선택한 다음, WSL - 버그 보고서를 선택하면 됩니다. 문제의 제목, Windows 빌드 번호(현재 빌드 번호를 보려면 `cmd.exe /c ver` 실행), WSL 1 또는 2 중에 현재 실행하고 있는 버전, 현재 Linux 커널 버전 번호(`wsl1.exe --status` 또는 `cat /proc/version` 실행), 배포판의 버전 번호(`lsb_release -r` 실행), 관련된 기타 소프트웨어 버전, 재현 단계, 예상 동작, 실제 동작 및 진단 로그(사용 가능하고 적절한 경우)를 포함해야 합니다. 자세한 내용은 [WSL에 기여](#)를 참조하세요.
- 녹색 새 문제 단추를 선택하고 기능 요청을 선택하여 **기능 요청을 제출합니다.** 요청을 설명하는 몇 가지 질문에 답해야 합니다.

다음도 가능합니다.

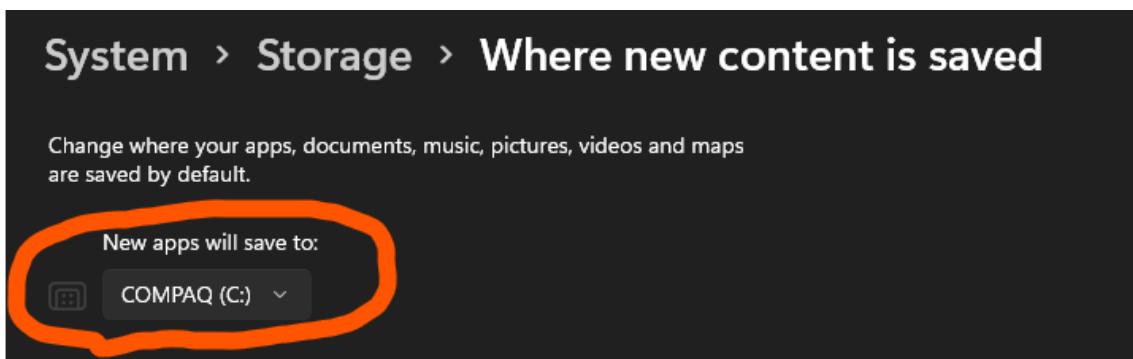
- [WSL 문서 리포지토리](#)를 사용하여 **설명서 문제를 제출합니다.** WSL 문서에 기여하려면 [Microsoft Docs 기여자 가이드](#)를 참조하세요.
- 문제가 Windows 터미널, Windows 콘솔 또는 명령줄 UI와 좀 더 관련이 있는 경우 [Windows 터미널 제품 리포지토리](#)를 사용하여 **Windows 터미널 문제를 제출합니다.**

## 설치 문제

- **0x80070003 오류로 인한 설치 실패**
  - Linux용 Windows 하위 시스템은 시스템 드라이브(일반적으로 `c:` 드라이브)에서만 실행됩니다. 배포가 시스템 드라이브에 저장되어 있는지 확인합니다.
  - Windows 10에서 설정 -> 시스템 -> 스토리지 -> 더 많은 스토리지 설정: 새 콘텐츠가 저장되는 위치 변경을 엽니다.

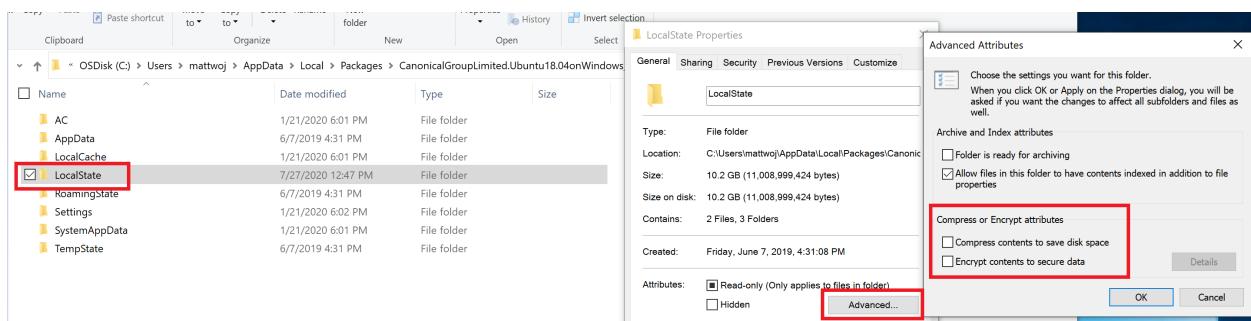


- Windows 11에서 설정 -> 시스템 -> 스토리지 -> 고급 스토리지 설정 -> 새 콘텐츠가 저장되는 위치를 엽니다.



- 0x8007019e 오류로 인한 WslRegisterDistribution 실패
  - 선택적인 Linux용 Windows 하위 시스템 구성 요소가 실행되지 않습니다.
  - 제어판 -> 프로그램 및 기능 -> Windows 기능 사용/사용 안 함 -> 을 열어 Linux용 Windows 하위 시스템을 선택하거나 이 문서의 시작 부분에서 설명한 PowerShell cmdlet을 사용합니다.
- 0x80070003 오류 또는 0x80370102 오류로 인해 설치하지 못했습니다.
  - 컴퓨터 BIOS 내에서 가상화를 사용하도록 설정했는지 확인합니다. 이 방법에 대한 지침은 컴퓨터마다 다르며, CPU 관련 옵션에 있을 가능성이 높습니다.
  - WSL2를 사용하려면 CPU가 Intel Nehalem 프로세서(Intel Core 1세대) 및 AMD Opteron에 도입된 SLAT(두 번째 수준 주소 변환) 기능을 지원해야 합니다. 이전 CPU(예: Intel Core 2 Duo)는 Virtual Machine 플랫폼을 성공적으로 설치하더라도 WSL2를 실행할 수 없습니다.
- 업그레이드 시도 중 오류: Invalid command line option: wsl --set-version Ubuntu 2
  - Linux용 Windows 하위 시스템을 사용하도록 설정했고 Windows 빌드 버전 18362 이상을 사용하고 있는지 확인합니다. WSL을 실행하도록 하려면 관리자 권한(Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux)으로 PowerShell 프롬프트에서 이 명령을 실행합니다.

- 가상 디스크 시스템 제한으로 인해 요청한 작업을 완료할 수 없습니다. 가상 하드 디스크 파일은 압축이 풀려 있는 상태이고 암호화되지 않아야 하며 스파스가 아니어야 합니다.
  - Linux 배포판의 프로필 폴더를 열어서 "내용 압축"과 "내용 암호화"를 선택 취소합니다. 이는 Windows 파일 시스템의 `%USERPROFILE%\AppData\Local\Packages\CanonicalGroupLimited...` 같은 폴더에 있을 것입니다.
  - 이 Linux 배포판 프로필에는 LocalState 폴더가 있을 것입니다. 이 폴더를 마우스 오른쪽 단추로 클릭하여 옵션 메뉴를 표시합니다. 속성 > 고급을 선택한 다음, "압축하여 디스크 공간 절약" 및 "데이터 보호를 위해 내용을 암호화" 확인란이 선택 취소되어 있는지 확인합니다(선택하지 않음). 이를 현재 폴더 또는 모든 하위 폴더와 파일에만 적용할지 묻는 메시지가 표시되면 압축 플래그만 지우도록 "이 폴더만"을 선택합니다. 그러면 `wsl --set-version` 명령이 작동할 것입니다.



## ① 참고

내 경우에는 Ubuntu 18.04 배포판의 LocalState 폴더가 `C:\Users<my-user-name>\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu18.04onWindows_79rhkp1fndgsc`에 있었습니다.

이 문제에 대한 업데이트된 정보를 추적할 수 있는 [WSL Docs GitHub 스레드 #4103](#) 을 확인하세요.

- cmdlet, 함수, 스크립트 파일 또는 실행 프로그램의 이름에는 'wsl'이라는 단어가 들어갈 수 없습니다.
  - [Linux용 Windows 하위 시스템 옵션 구성 요소가 설치되었는지](#) 확인하세요. 또는 ARM64 디바이스를 사용 중이고 PowerShell에서 이 명령을 실행하는 경우 이 오류가 표시됩니다. [PowerShell Core](#) 또는 명령 프롬프트에서 `wsl.exe`를 대신 실행하세요.
- 오류: [Linux용 Windows 하위 시스템에 설치된 배포가 없습니다.](#)
  - WSL 배포를 이미 설치한 후 이 오류가 표시되는 경우 다음을 수행합니다.
    1. 명령줄에서 호출하기 전에 배포를 한 번 이상 실행합니다.

2. 별도의 사용자 계정을 실행할 수 있는지 여부를 확인합니다. 상승된 권한(관리 모드)으로 기본 사용자 계정을 실행하면 이 오류가 발생하지 않지만, Windows 와 함께 제공되는 기본 관리자 계정을 실수로 실행하지 않도록 해야 합니다. 이 계정은 별도의 사용자 계정이며 설치된 WSL 배포는 디자인 별로 표시되지 않습니다. 자세한 내용은 [기본 제공 관리자 계정 사용 및 사용 안 함](#)을 참조하세요.

3. WSL 실행 파일은 네이티브 시스템 디렉터리에만 설치됩니다. 64비트 Windows(또는 ARM64, 비 네이티브 조합)에서 32비트 프로세스를 실행하는 경우 호스트된 비 네이티브 프로세스에는 실제로 다른 System32 폴더가 표시됩니다. (x64 Windows에서 32비트 프로세스에 표시되는 것은 디스크의 \Windows\SysWOW64에 저장됩니다.) 호스트 프로세스에서 가상 폴더 \Windows\sysnative를 확인하여 "네이티브" system32에 액세스할 수 있습니다. 실제로 디스크에는 존재하지 않지만 파일 시스템 경로 확인자가 찾을 것입니다.

- **오류: 이 업데이트는 Linux용 Windows 하위 시스템을 사용하는 머신에만 적용됩니다.**

- Linux 커널 업데이트 MSI 패키지를 설치하려면 WSL이 필요하며, 먼저 이를 사용하도록 설정해야 합니다. 실패하면 `This update only applies to machines with the Windows Subsystem for Linux` 메시지가 표시됩니다.

- 이 메시지가 표시되는 세 가지 가능한 원인은 다음과 같습니다.

1. WSL 2를 지원하지 않는 이전 버전의 Windows를 아직 사용하고 있습니다. 버전 요구 사항 및 업데이트에 대한 링크는 2단계를 참조하세요.

2. WSL을 사용하도록 설정되지 않았습니다. 1단계로 돌아가서 머신에서 선택적 WSL 기능을 사용하도록 설정되어 있는지 확인해야 합니다.

3. WSL을 사용하도록 설정한 후에는 다시 부팅해야 적용됩니다. 머신을 다시 부팅하고 다시 시도하세요.

- **오류: WSL 2에는 커널 구성 요소에 대한 업데이트가 필요합니다. 자세한 내용은 <https://aka.ms/wsl2kernel>을 방문하세요.**

- Linux 커널 패키지가 %SystemRoot%\system32\lxss\tools 폴더에 없는 경우 이 오류가 발생합니다. 이러한 설치 지침의 4단계에서 Linux 커널 업데이트 MSI 패키지를 설치하여 이 문제를 해결하세요. '[프로그램 추가/제거](#)'에서 MSI를 제거하고 다시 설치해야 할 수 있습니다.

## 일반적인 문제

# Windows 10 버전 1903을 사용하고 있는데 WSL 2에 대한 옵션이 아직 보이지 않음

이는 머신이 아직 WSL 2의 백포트를 가져오지 않았기 때문일 수 있습니다. 이를 해결하는 가장 간단한 방법은 Windows 설정으로 이동하여 '업데이트 확인'을 클릭하여 시스템에 최신 업데이트를 설치하는 것입니다. [백포트에 대한 전체 지침](#)을 확인하세요.

'업데이트 확인'을 눌렀는데도 업데이트를 아직 받지 못한 경우 [KB KB4566116을 수동으로 설치](#) 할 수 있습니다.

## 오류: `wsl --set-default-version 2` 인 경우 0x1bc

'표시 언어' 또는 '시스템 로캘' 설정이 영어가 아닌 경우 발생할 수 있습니다.

PowerShell

```
wsl --set-default-version 2
Error: 0x1bc
For information on key differences with WSL 2 please visit
https://aka.ms/wsl2
```

0x1bc의 실제 오류는 다음과 같습니다.

PowerShell

```
WSL 2 requires an update to its kernel component. For information please
visit https://aka.ms/wsl2kernel
```

자세한 내용은 문제 [5749](#)를 참조하십시오.

## Windows에서 WSL 파일에 액세스할 수 없음

9p 프로토콜 파일 서버는 Linux 쪽에서 서비스를 제공하여 Windows가 Linux 파일 시스템에 액세스할 수 있도록 합니다. Windows에서 `\wsl$`를 사용하여 WSL에 액세스할 수 없는 경우 9P가 제대로 시작되지 않았기 때문일 수 있습니다.

이를 확인하려면 `dmesg |grep 9p`를 사용하여 시작 로그를 확인할 수 있습니다. 그러면 오류가 표시됩니다. 성공적인 출력은 다음과 같습니다.

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
```

이 문제에 대한 자세한 내용은 [이 GitHub 스레드](#)를 참조하세요.

## WSL 2 배포를 시작할 수 없으며 출력에 'WSL 2'만 표시됨

표시 언어가 영어가 아닌 경우 오류 텍스트가 잘려서 표시될 수 있습니다.

PowerShell

```
C:\Users\me>wsl  
WSL 2
```

이 문제를 해결하려면 <https://aka.ms/ws12kernel>을 방문하여 해당 문서 페이지의 지침에 따라 수동으로 커널을 설치하세요.

## Linux에서 windows.exe를 실행하는 경우 `command not found`

사용자는 Linux에서 직접 notepad.exe와 같은 Windows 실행 파일을 실행할 수 있습니다. 경우에 따라 다음과 같이 "명령을 찾을 수 없음" 오류가 발생할 수 있습니다.

Bash

```
$ notepad.exe  
-bash: notepad.exe: command not found
```

\$PATH에 win32 경로가 없는 경우 interop에서 .exe를 찾지 못합니다. Linux에서 echo \$PATH를 실행하여 이를 확인할 수 있습니다. 출력에 win32 경로(예: /mnt/c/Windows)가 표시되어야 합니다. Windows 경로가 표시되지 않는 경우 Linux 셸에서 PATH를 덮어쓸 가능성이 높습니다.

다음은 Debian의 /etc/profile로 인해 문제가 발생한 예입니다.

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
```

Debian에서 올바른 방법은 위의 줄을 제거하는 것입니다. 아래와 같이 할당 시 \$PATH를 추가할 수도 있지만 이 경우에는 WSL 및 VSCode와 관련한 [몇 가지 다른 문제](#) 가 발생할

니다.

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
```

자세한 내용은 문제 [5296](#) 및 [5779](#) 를 참조하세요.

## "Error: 0x80370102 필요한 기능이 설치되어 있지 않아 가상 머신을 시작할 수 없습니다."

가상 머신 플랫폼 Windows 기능을 활성화하고 BIOS에서 가상화가 사용되고 있는지 확인하세요.

1. [Hyper-V 시스템 요구 사항](#)을 확인합니다.
2. 머신이 VM인 경우 [중첩된 가상화](#)를 수동으로 사용하도록 설정하세요. 관리자로 powershell을 시작하고 다음을 실행합니다.

PowerShell

```
Set-VMProcessor -VMName <VMName> -ExposeVirtualizationExtensions $true
```

3. 가상화를 사용하도록 설정하는 방법에 대한 자세한 지침은 PC 제조업체의 지침을 따르세요. 일반적으로 이는 시스템 BIOS를 사용하여 이러한 기능이 CPU에서 활성화되도록 할 수 있습니다. 이 프로세스에 대한 지침은 머신마다 다를 수 있습니다. 예는 Bleeping Computer의 [이 문서](#)를 참조하세요.
4. [Virtual Machine Platform](#) 선택적 구성 요소를 사용하도록 설정한 후 머신을 다시 시작합니다.
5. 부팅 구성에 하이퍼바이저 시작이 사용하도록 설정되어 있는지 확인합니다. 관리자 권한 powershell을 실행하여 유효성을 검사할 수 있습니다.

PowerShell

```
bcdedit /enum | findstr -i hypervisorlauchtype
```

`hypervisorlauchtype off` 가 표시되면 하이퍼바이저가 사용하지 않도록 설정된 것입니다. 이를 사용하도록 설정하려면 관리자 권한 powershell에서 다음을 실행합니

다.

```
bcdedit /set hypervisorlaunchtype Auto
```

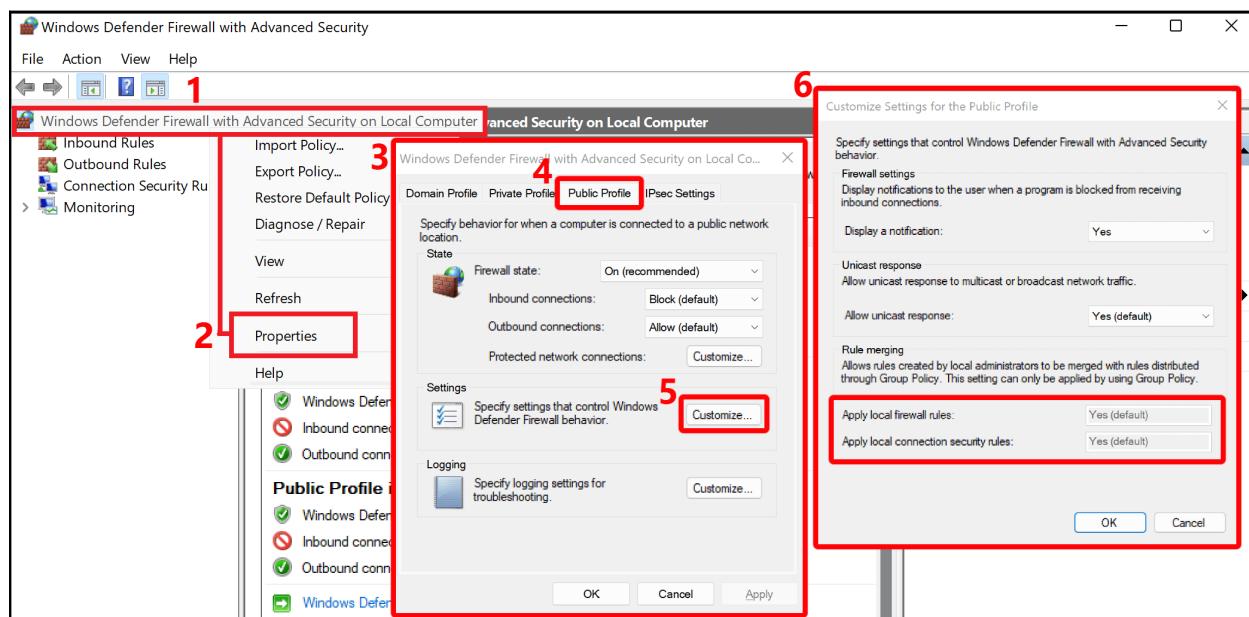
6. 또한 타사 하이퍼바이저(예: VMware 또는 VirtualBox)가 설치되어 있는 경우 HyperV([VMware 15.5.5+ ↗](#) 및 [VirtualBox 6+ ↗](#))를 지원할 수 있는 최신 버전에 설치되어 있거나 해당 하이퍼바이저가 꺼져 있는지 확인하세요.

Virtual Machine에서 Hyper-V를 실행할 때 [중첩된 가상화를 구성](#)하는 방법에 대해 자세히 알아봅니다.

## WSL은 내 작업 컴퓨터 또는 엔터프라이즈 환경에서 네트워크 연결이 없습니다.

비즈니스 또는 엔터프라이즈 환경에는 무단 네트워크 트래픽을 차단하도록 구성된 [Windows Defender 방화벽 설정](#)이 있을 수 있습니다. [로컬 규칙 병합](#)이 "아니요"로 설정된 경우 WSL 네트워킹은 기본적으로 작동하지 않으며 관리자는 이를 허용하기 위해 방화벽 규칙을 추가해야 합니다.

로컬 규칙 병합 설정은 다음 단계에 따라 확인할 수 있습니다.



1. "고급 보안이 포함된 Windows Defender 방화벽"을 엽니다(제어판의 "Windows Defender 방화벽"과 다름).
2. "로컬 컴퓨터의 고급 보안이 포함된 Windows Defender 방화벽" 탭을 마우스 오른쪽 단추로 클릭합니다.
3. "속성"을 선택합니다.
4. 열리는 새 창에서 "공개 프로필" 탭을 선택합니다.

5. "설정" 섹션에서 "사용자 지정"을 선택합니다.
6. "규칙 병합"이 "아니요"로 설정되어 있는지 확인하려면 열리는 "공개 프로필에 대한 설정 사용자 지정" 창을 확인합니다. 그러면 WSL에 대한 액세스가 차단됩니다.

이 방화벽 설정을 변경하는 방법에 대한 지침은 [엔터프라이즈 환경: 회사용 WSL 설정](#)에서 찾을 수 있습니다.

## VPN에 연결되면 WSL에 네트워크 연결이 없습니다.

Windows에서 VPN에 연결한 후에 Bash에서 네트워크 연결이 끊어지면 Bash 내에서 다음 해결 방법을 시도해 보세요. 이 해결 방법을 사용하면 `/etc/resolv.conf`를 통해 DNS 확인을 수동으로 재정의할 수 있습니다.

1. `ipconfig.exe /all`을 수행하여 VPN의 DNS 서버를 적어둡니다.
2. `sudo cp /etc/resolv.conf /etc/resolv.conf.new`를 수행하여 기존 resolv.conf의 복사본을 만듭니다.
3. `sudo unlink /etc/resolv.conf`를 수행하여 현재 resolv.conf의 연결을 해제합니다.
4. `sudo mv /etc/resolv.conf.new /etc/resolv.conf`
5. `/etc/wsl.conf`를 편집하고 이 콘텐츠를 파일에 추가합니다. (이 설정에 대한 자세한 내용은 [고급 설정 구성](#)에서 찾을 수 있습니다.)

```
[network]
generateResolvConf=false
```

6. `/etc/resolv.conf`를 엽니다.
  - a. 자동 생성을 설명하는 주석이 있는 파일에서 첫 번째 줄을 삭제합니다.
  - b. 위 (1)의 DNS 항목을 DNS 서버 목록의 첫 번째 항목으로 추가합니다.
  - c. 파일을 닫습니다.

VPN의 연결을 끊은 후에는 변경 내용을 `/etc/resolv.conf`로 되돌려야 합니다. 이렇게 하려면 다음을 수행합니다.

1. `cd /etc`
2. `sudo mv resolv.conf resolv.conf.new`
3. `sudo ln -s ../../run/resolvconf/resolv.conf resolv.conf`

## WSL을 시작하거나 배포를 설치하면 오류 코드가 반환됨

이 [지침](#)에 따라 자세한 로그를 수집하고 문제를 GitHub에 제출하세요.

# WSL 업데이트

Linux용 Windows 하위 시스템에는 업데이트가 가능한 두 가지 구성 요소가 있습니다.

1. Linux용 Windows 하위 시스템 자체를 업데이트하려면 PowerShell 또는 CMD에서 `wsl --update` 명령을 사용합니다.
2. 특정 Linux 배포판 사용자 바이너리를 업데이트하려면 업데이트하려는 Linux 배포판에서 `apt-get update | apt-get upgrade` 명령을 사용합니다.

## apt-get upgrade 오류

일부 패키지는 아직 구현하지 않은 기능을 사용합니다. 예를 들어 `udev`는 아직 지원되지 않으므로 여러 `apt-get upgrade` 오류가 발생합니다.

`udev`와 관련된 문제를 해결하려면 다음 단계를 수행합니다.

1. 다음을 `/usr/sbin/policy-rc.d`에 작성하고 변경 내용을 저장합니다.

```
Bash  
  
#!/bin/sh  
exit 101
```

2. 실행 권한을 `/usr/sbin/policy-rc.d`에 추가합니다.

```
Bash  
  
chmod +x /usr/sbin/policy-rc.d
```

3. 다음 명령을 실행 합니다.

```
Bash  
  
dpkg-divert --local --rename --add /sbin/initctl  
ln -s /bin/true /sbin/initctl
```

## "Error: 0x80040306" - 설치 시

이 오류는 레거시 콘솔을 지원하지 않는다는 사실과 관련이 있습니다. 레거시 콘솔을 해제하려면 다음을 수행합니다.

1. cmd.exe를 엽니다.

2. 마우스 오른쪽 단추로 제목 표시줄 -> 속성 -> 를 클릭하고, 레거시 콘솔 사용의 선택을 취소합니다.
3. 확인을 클릭합니다.

## "Error: 0x80040154" - Windows 업데이트 후

Windows 업데이트 중에 Linux용 Windows 하위 시스템 기능이 사용하지 않도록 설정될 수 있습니다. 이 문제가 발생하면 Windows 기능을 다시 사용하도록 설정해야 합니다. Linux용 Windows 하위 시스템을 사용하도록 설정하는 방법에 대한 지침은 [수동 설치 가이드](#)에서 확인할 수 있습니다.

## 표시 언어 변경

WSL 설치는 Windows 설치의 로캘과 일치하도록 Ubuntu 로캘을 자동으로 변경하려고 합니다. 이 동작을 원하지 않는 경우 설치가 완료된 후 다음 명령을 실행하여 Ubuntu 로캘을 변경할 수 있습니다. 이 변경 내용이 적용되려면 bash.exe를 다시 시작해야 합니다.

아래 예제에서는 로캘을 en-US로 변경합니다.

```
Bash
sudo update-locale LANG=en_US.UTF8
```

## Windows 시스템 복원 후의 설치 문제

1. %windir%\System32\Tasks\Microsoft\Windows\Windows Subsystem for Linux 폴더를 삭제합니다.  
참고: 선택적 기능이 완전히 설치되어 작동하는 경우에는 이 작업을 수행하지 마세요.
2. 선택적 WSL 기능을 사용하도록 설정합니다(아직 사용하지 않는 경우).
3. 다시 부팅
4. lxrund /uninstall /full을 실행합니다
5. Bash를 설치합니다.

## WSL에서 인터넷에 액세스할 수 없음

일부 사용자가 WSL에서 인터넷 액세스를 차단하는 특정 방화벽 애플리케이션의 문제를 보고했습니다. 보고된 방화벽은 다음과 같습니다.

1. Kaspersky
2. AVG

3. Avast

4. Symantec Endpoint Protection

방화벽을 해제하면 액세스가 허용되는 경우도 있습니다. 단순히 방화벽을 설치하면 경우에 따라 액세스가 차단되는 것처럼 보입니다.

Microsoft Defender 방화벽을 사용하는 경우 "허용되는 앱 목록에 있는 연결을 포함하여 들어오는 연결을 모두 차단합니다"를 선택 취소하면 액세스가 허용됩니다.

## ping 사용 시의 권한 거부 오류

[Windows 1주년 업데이트, 버전 1607](#)의 경우 WSL에서 ping을 실행하려면 Windows의 관리자 권한이 필요합니다. ping을 실행하려면 Windows의 Ubuntu에서 관리자 권한으로 Bash를 실행하거나 CMD/PowerShell 프롬프트에서 관리자 권한으로 bash.exe를 실행합니다.

이후 버전의 Windows인 [빌드 14926 이상](#)의 경우 관리자 권한이 더 이상 필요하지 않습니다.

## Bash가 중지됨

Bash로 작업하는 동안 Bash가 중지되거나 교착 상태가 되어 입력에 응답하지 않는 경우 메모리 덤프를 수집하고 보고하여 문제를 진단하는 데 도움이 됩니다. 다음 단계를 수행하면 시스템의 작동이 중단됩니다. 편안하지 않으면 이 작업을 수행하지 않거나 해당 작업을 저장한 후에 수행하세요.

메모리 덤프를 수집하려면 다음을 수행합니다.

1. 메모리 덤프 유형을 "전체 메모리 덤프"로 변경합니다. 덤프 유형을 변경하는 동안 현재 유형을 적어둡니다.
2. 이 [단계](#)를 사용하여 키보드 컨트롤을 사용하는 시스템 작동 중단을 구성합니다.
3. 중지 또는 교착 상태를 재현합니다.
4. (2)의 키 시퀀스를 사용하여 시스템의 작동을 중단시킵니다.
5. 시스템의 작동이 중단되고 메모리 덤프가 수집됩니다.
6. 시스템이 다시 부팅되면 memory.dmp를 secure@microsoft.com에 보고합니다. 덤프 파일의 기본 위치는 %SystemRoot%\memory.dmp 또는 C:\Windows\memory.dmp(C:가 시스템 드라이브인 경우)입니다. 이메일에서 덤프는 Windows 팀의 WSL 또는 Bash에 대한 것입니다.

7. 메모리 덤프 유형을 원래 설정으로 복원합니다.

## 빌드 번호 확인

PC의 아키텍처 및 Windows 빌드 번호를 확인하려면  
설정>시스템>정보를 차례로 엽니다.

OS 빌드 및 시스템 종류 필드를 찾습니다.

The screenshot shows two main sections of the Windows System Information window:

**Device specifications** section:

- Device name: 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

**Related links**: Domain or workgroup, System protection, Advanced system settings

**Windows specifications** section:

- 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

Windows Server 빌드 번호를 확인하려면 PowerShell에서 다음을 실행합니다.

```
PowerShell

systeminfo | Select-String "^\$OS Name", "^\$OS Version"
```

## WSL이 사용하도록 설정되었는지 확인

관리자 권한 PowerShell 창에서 다음을 실행하여 Linux용 Windows 하위 시스템을 사용하도록 설정했는지 확인할 수 있습니다.

```
PowerShell

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

# OpenSSH 서버 연결 문제

SSH 서버를 연결하려고 하면 "127.0.0.1 포트 22에 의해 연결이 닫혔습니다." 오류로 인해 실패합니다.

1. OpenSSH 서버가 실행되고 있는지 확인합니다.

```
Bash
```

```
sudo service ssh status
```

그리고 <https://ubuntu.com/server/docs/service-openssh> 자습서를 수행했습니다.

2. sshd 서비스를 중지한 다음, 디버그 모드에서 sshd를 시작합니다.

```
Bash
```

```
sudo service ssh stop  
sudo /usr/sbin/sshd -d
```

3. 시작 로그를 확인하고, HostKeys를 사용할 수 있고 다음과 같은 로그 메시지가 표시되지 않는지 확인합니다.

```
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
```

이러한 메시지가 표시되고 키가 `/etc/ssh/` 아래에 없으면 해당 키를 다시 생성하거나 openssh 서버를 제거& 후 설치해야 합니다.

```
BASH
```

```
sudo apt-get purge openssh-server  
sudo apt-get install openssh-server
```

## "참조 된 어셈블리를 찾을 수 없습니다." WSL 선택적 기능을 사용하도록 설정하는 경우

이 오류는 잘못된 설치 상태와 관련이 있습니다. 이 문제를 시도해 보고 해결하려면 다음 단계를 수행하세요.

- PowerShell에서 WSL 기능 사용 명령을 실행하는 경우 [시작] 메뉴를 열고, 'Windows 기능 사용/사용 안 함'을 검색하여 GUI를 대신 사용해 본 다음, 목록에서 선택적 구성 요소를 설치할 'Linux용 Windows 하위 시스템'을 선택합니다.
- [설정], [업데이트]로 차례로 이동하고, '업데이트 확인'을 클릭하여 Windows 버전을 업데이트합니다.
- 두 단계가 모두 실패하고 WSL에 액세스해야 하는 경우 설치 미디어를 사용하여 Windows를 다시 설치하고 '모두 유지'를 선택하여 앱과 파일이 유지되도록 하는 방식으로 업그레이드하는 것이 좋습니다. 이 작업을 수행하는 방법에 대한 지침은 [Windows 10 다시 설치](#) 페이지에서 확인할 수 있습니다.

## (SSH 관련) 권한 오류 해결

이 오류가 표시되는 경우:

Bash

```
@@@@@@@WARNING: UNPROTECTED PRIVATE KEY FILE!@@@@@@@  
@          Permissions 0777 for '/home/artur/.ssh/private-key.pem' are too open.
```

이 문제를 해결하려면 다음을 `/etc/wsl.conf` 파일에 추가합니다.

Bash

```
[automount]  
enabled = true  
options = metadata,uid=1000,gid=1000,umask=0022
```

이 명령을 추가하면 메타데이터가 포함되고 WSL에서 보이는 Windows 파일에 대한 파일 권한이 수정됩니다. 자세한 내용은 [파일 시스템 권한](#)에서 확인하세요.

## 배포 내에서 Windows 명령 실행 실패

Microsoft Store에서 제공하는 일부 배포판은 아직 완전히 호환되지 않아서 Windows 명령을 바로 실행할 수 없습니다. `powershell.exe /c start`. 또는 다른 Windows 명령을 실행할 때 `-bash: powershell.exe: command not found` 오류가 발생하면 다음 단계를 수행하여 해결할 수 있습니다.

1. WSL 배포에서 `echo $PATH`를 실행합니다.

`/mnt/c/Windows/system32`를 포함하지 않는 경우 무언가 표준 PATH 변수를 다시 정의하는 것입니다.

2. `cat /etc/profile`을 사용하여 프로필 설정을 확인하세요.

PATH 변수 할당을 포함하는 경우에는 파일을 편집하여 # 문자로 PATH 할당 블록을 주석 처리합니다.

3. `wsl.conf`가 있는지 확인하고(`cat /etc/wsl.conf`) `appendWindowsPath=false`가 없는지 확인합니다. 있는 경우에는 주석 처리합니다.

4. `wsl -t` 다음에 배포 이름을 입력하여 배포를 다시 시작하거나 cmd 또는 PowerShell에서 `wsl --shutdown`을 실행합니다.

## WSL 2를 설치한 후 부팅할 수 없음

사용자가 WSL 2를 설치한 후 부팅할 수 없는 문제에 대해 알고 있습니다. 현재 이러한 문제를 철저하게 진단하고 있으며, [버퍼 크기를 변경](#)하거나 [적절한 드라이버를 설치](#)하면 이 문제를 해결하는 데 도움이 된다는 사용자들의 제보가 있었습니다. 이 문제에 대한 최신 업데이트를 보려면 이 [GitHub](#) 문제를 참조하세요.

## ICS를 사용하지 않는 경우 WSL 2 오류

ICS(인터넷 연결 공유)는 WSL 2의 필수 구성 요소입니다. ICS 서비스는 HNS(호스트 네트워크 서비스)에서 WSL 2가 NAT, DNS, DHCP 및 호스트 연결 공유에 의존하는 기본 가상 네트워크를 만드는 데 사용됩니다.

ICS 서비스(SharedAccess)를 사용하지 않거나 그룹 정책을 통해 ICS를 사용하지 않으면 WSL HNS 네트워크가 생성되지 않습니다. 이로 인해 새 WSL 버전 2 이미지를 만들 때 오류가 발생하고 버전 1 이미지를 버전 2로 변환하려고 할 때 다음 오류가 발생합니다.

콘솔

There are no more endpoints available from the endpoint mapper.

WSL 2가 필요한 시스템은 ICS 서비스(SharedAccess)를 기본 시작 상태인 수동(트리거 시작)으로 두어야 하며 ICS를 사용하지 않도록 설정하는 모든 정책을 덮어쓰거나 제거해야 합니다. ICS 서비스를 사용하지 않도록 설정하면 WSL 2가 중단되기 때문에 ICS를 사용하

지 않도록 설정하는 것을 권장하지 않지만, 이러한 지침을 사용하여 ICS의 일부를 사용하지 않도록 설정하지 않을 수 있습니다.

## 이전 버전의 Windows 및 WSL 사용

Windows 10 크리에이터 업데이트(2017년 10월, 빌드 16299) 또는 1주년 업데이트(2016년 8월, 빌드 14393)와 같은 이전 버전의 Windows 및 WSL을 실행하는 경우 주목해야 하는 여러 가지 차이점이 있습니다. 최신 Windows 버전으로 업데이트하는 것이 좋지만, 업데이트가 불가능한 경우를 대비하여 아래에 몇 가지 차이점을 설명해 놓았습니다.

상호 운용성 명령의 차이점:

- `bash.exe` 가 `ws1.exe`로 바뀌었습니다. Linux 명령은 Windows 명령 프롬프트 또는 PowerShell에서 실행할 수 있지만, 초기 Windows 버전의 경우 `bash` 명령을 사용해야 합니다. 예: `C:\temp> bash -c "ls -la"` `bash -c`에 전달된 WSL 명령은 수정되지 않고 WSL 프로세스에 전달됩니다. 파일 경로는 WSL 형식으로 지정해야 하며, 관련 문자를 이스케이프 방식으로 처리해야 합니다. 예를 들어 `C:\temp> bash -c "ls -la /proc/cpuinfo"` 또는 `C:\temp> bash -c "ls -la \"/mnt/c/Program Files\""` 입니다.
- 특정 배포에 사용할 수 있는 명령을 확인하려면 `[distro.exe] /?`를 실행합니다. 예를 들어 Ubuntu에서는 `C:\> ubuntu.exe /?`를 실행합니다.
- Windows 경로가 WSL `$PATH`에 포함됩니다.
- 이전 버전의 Windows 10에서 WSL 배포를 통해 Windows 도구를 호출하는 경우 디렉터리 경로를 지정해야 합니다. 예를 들어 WSL 명령줄에서 Windows 메모장 앱을 호출하려면 `/mnt/c/Windows/System32/notepad.exe`를 입력합니다.
- 기본 사용자를 `root`로 변경하려면 PowerShell에서 `c:\> 1xrun /setdefaultuser root` 명령을 사용한 다음, `c:\> bash.exe` 명령으로 Bash.exe를 실행하여 로그인합니다. 배포판 암호 명령 `$ passwd username`을 사용하여 암호를 다시 설정한 다음, `$ exit` 명령을 사용하여 Linux 명령줄을 닫습니다. Windows 명령 프롬프트 또는 Powershell에서 `c:\> 1xrun.exe /setdefaultuser username` 명령을 사용하여 기본 사용자를 다시 일반 Linux 사용자 계정으로 설정합니다.

## 기존 버전의 WSL 제거

크리에이터 업데이트(2017년 10월, 빌드 16299) 이전 버전의 Windows 10에 WSL을 설치한 경우 설치된 이전 Linux 배포판에서 꼭 필요한 파일, 데이터 등을 Microsoft Store를 통해 설치한 최신 배포판으로 마이그레이션하는 것이 좋습니다. 마신에서 기존 배포판을 제거하려면 명령줄 또는 PowerShell 인스턴스에서 `ws1 --unregister Legacy` 명령을 실행합니다. Windows 파일 탐색기 또는 PowerShell에서 `rm -Recurse $env:localappdata/lxss/`

명령으로 `%localappdata%\lcss\` 폴더(및 모든 하위 콘텐츠)를 삭제하여 이전 배포판을 수동으로 제거하는 방법도 있습니다.

# Linux용 Windows 하위 시스템의 릴리스 정보

아티클 • 2023. 03. 21. • 읽는 데 138분 걸림

## 빌드 21364

빌드 21364에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- 이제 GUI 앱을 사용할 수 있습니다. 자세한 내용은 [이 블로그 게시물](#)을 참조하세요.
- \\wsl.localhost\를 통해 파일에 액세스할 때 발생하는 오류를 해결합니다.
- LxssManager 서비스의 잠재적 교착 상태가 수정되었습니다.

## 빌드 21354

빌드 21354에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- 네트워크에 "wsl"이라는 머신이 있을 때 문제가 발생하지 않도록 \wsl 접두사를 \wsl.localhost로 전환합니다. \wsl\$은 계속 작동합니다.
- wow 프로세스에 대한 Linux 빠른 액세스 아이콘을 사용합니다.
- 버전 2가 항상 wslapi RegisterDistribution을 통해 전달되는 문제를 업데이트합니다.
- /usr/lib/wsl/lib 디렉터리의 fmask를 222로 변경하여 파일이 실행 가능한 것으로 표시되도록 합니다. [GH 3847]
- Virtual Machine 플랫폼을 사용하지 않는 경우 wsl 서비스 충돌을 수정합니다.

## 빌드 21286

빌드 21286에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- 명령의 현재 작업 디렉터리를 설정하는 wsl.exe --cd 명령이 도입되었습니다.
- Linux 오류 코드와 NTSTATUS의 매핑이 개선되었습니다. [GH 6063]
- wsl.exe --mount 오류 보고가 개선되었습니다.
- /etc/wsl.conf에 시작 명령을 사용하도록 옵션을 추가했습니다.

콘솔

```
[boot]
command=<string>
```

## 빌드 20226

빌드 20226에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- LxssManager 서비스의 충돌이 해결되었습니다. [GH 5902]

## 빌드 20211

빌드 20211에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- 실제 또는 가상 디스크를 탑재하는 `wsl.exe --mount`를 도입했습니다. 자세한 내용은 [Windows 및 WSL 2에서 Linux 파일 시스템 액세스](#)를 참조하세요.
- VM이 유휴 상태인지 확인할 때 LxssManager 서비스의 작동 중단이 수정되었습니다. [GH 5768]
- 압축된 VHD 파일을 지원합니다. [GH 4103]
- c:\windows\system32\lxss\lib에 설치된 Linux 사용자 모드 라이브러리가 OS 업그레이드에서 유지되도록 합니다. [GH 5848]
- `wsl --install --list-distributions`를 사용하여 설치할 수 있는 사용 가능한 배포를 나열하는 기능이 추가되었습니다.
- 이제 사용자가 로그오프하면 WSL 인스턴스가 종료됩니다.

## 빌드 20190

빌드 20190에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- WSL1 인스턴스가 시작되지 않도록 하는 버그가 수정되었습니다. [GH 5633]
- Windows 프로세스 출력을 리디렉션하는 경우 발생하는 중단 문제가 해결되었습니다. [GH 5648]
- VM 유휴 시간 제한(`wsl2.vmlidleTimeout=<time_in_ms>`)을 제어하는 `%userprofile%\wslconfig` 옵션이 추가되었습니다.
- WSL에서 앱 실행 별칭 시작을 지원합니다.
- WSL2 커널 및 배포를 `wsl.exe --install`에 설치하기 위한 지원이 추가되었습니다.

## 빌드 20175

빌드 20175에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- WSL2 VM의 기본 메모리 할당을 호스트 메모리의 50% 또는 8GB 중 [GH 4166]보다 더 적은 값으로 조정합니다.
- URI 구문 분석을 지원하기 위해 `\wsl$` 접두사를 `\wsl`로 변경합니다. 이전 `\wsl$` 경로도 여전히 지원됩니다.

- amd64에서 WSL2에 대해 중첩된 가상화를 기본적으로 사용하도록 설정합니다. %userprofile%\.wslconfig([wsl2] nestedVirtualization=false)를 통해 이 기능을 사용하지 않도록 설정할 수 있습니다.
- wsl.exe --update demand가 Microsoft Update를 시작하도록 합니다.
- DrvFs에서 읽기 전용 파일의 이름 변경을 지원합니다.
- 오류 메시지가 항상 올바른 코드 페이지에 인쇄되어 있는지 확인합니다.

## 빌드 20150

빌드 20150에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- WSL2 GPU 컴퓨팅에 대한 자세한 내용은 [Windows 블로그](#)를 참조하세요.
- WSL을 쉽게 설치할 수 있도록 wsl.exe --install 명령줄 옵션이 도입되었습니다.
- WSL2 커널 업데이트를 관리하는 wsl.exe --update 명령줄 옵션이 도입되었습니다.
- WSL2를 기본값으로 설정합니다.
- WSL2 VM 정상 종료 시간 제한이 늘어났습니다.
- 디바이스 메모리를 매핑할 때 발생하는 virtio-9p 경합 상태를 수정했습니다.
- UAC를 사용하지 않도록 설정한 경우에는 관리자 권한 9p 서버를 실행하지 마세요.

## 빌드 19640

빌드 19640에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] virtio-9p(drivfs)의 안정성이 향상되었습니다.

## 빌드 19555

빌드 19555에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 메모리 cgroup을 사용하여 설치 및 변환 작업에 사용되는 메모리 양 제한 [GH 4669]
- 기능 검색 기능을 향상시키기 위해 Linux용 Windows 하위 시스템 선택적 구성 요소를 사용할 수 없는 경우 wsl.exe가 표시되도록 합니다.
- WSL 선택적 구성 요소가 설치되지 않은 경우 도움말 텍스트를 인쇄하도록 wsl.exe 변경
- 인스턴스를 만들 때 경합 상태 수정
- 모든 명령줄 기능이 포함된 wslclient.dll 생성
- LxssManagerUser 서비스 중지 중 충돌 방지
- distroName 매개 변수가 NULL인 경우 wslapi.dll 빠른 실패 수정

## 빌드 19041

빌드 19041에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 프로세스를 시작하기 전에 신호 마스크 지우기
- [WSL2] Linux 커널을 4.19.84로 업데이트
- symlink가 비 상대적일 때 /etc/resolv.conf symlink 생성 처리

## 빌드 19028

빌드 19028에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] Linux 커널을 4.19.81로 업데이트
- [WSL2] /dev/net/tun의 기본 권한을 0666으로 변경 [GH 4629]
- [WSL2] Linux VM에 할당된 기본 메모리 양을 호스트 메모리의 80%로 조정
- [WSL2] 잘못된 호출자가 서버를 중지할 수 없게 시간 제한이 있는 요청을 처리하도록 interop 서버 수정

## 빌드 19018

빌드 19018에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 9p 탑재의 기본값으로 cache=mmap을 사용하여 dotnet 앱 수정
- [WSL2] localhost 릴레이 수정 [GH 4340]
- [WSL2] 크로스 배포판 공유 tmpfs 탑재를 도입하여 배포판 사이에서 상태 공유
- \\wsl\$에 대한 영구 네트워크 드라이브 복원 수정

## 빌드 19013

빌드 19013에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] WSL 유틸리티 VM의 메모리 성능을 개선합니다. 더 이상 사용하지 않는 메모리는 호스트로 다시 해제됩니다.
- [WSL2] 커널 버전을 4.19.79로 업데이트합니다. (CONFIG\_HIGH\_RES\_TIMERS, CONFIG\_TASK\_XACCT, CONFIG\_TASK\_IO\_ACCOUNTING, CONFIG\_SCHED\_HRTICK 및 CONFIG\_BRIDGE\_VLAN\_FILTERING 추가).
- [WSL2] stdin이 닫혀 있지 않은 파일 핸들인 경우를 처리하기 위해 입력 릴레이를 수정합니다.[GH 4424]
- \\wsl\$ 대/소문자를 구분하지 않는지 확인합니다.

```
[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).
```

## 빌드 19002

빌드 19002에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL] 일부 유니코드 문자 처리 이슈 해결:  
<https://github.com/microsoft/terminal/issues/2770>
- [WSL] 빌드 간 업그레이드 후 바로 시작할 경우 드물게 배포판이 등록 취소되는 사례를 수정합니다.
- [WSL] 인스턴스 유휴 타이머가 취소되지 않은 wsl.exe --shutdown의 사소한 이슈를 해결합니다.

## 빌드 18995

빌드 18995에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 작업이 중단된 후 DrvFs 탑재가 중지되는 문제 해결(예: ctrl-c) [GH 4377]
- [WSL2] 매우 큰 hvsocket 메시지 처리 문제 해결 [GH 4105]
- [WSL2] stdin이 파일인 경우 interop 문제 해결 [GH 4475]
- [WSL2] 예기치 않은 네트워크 상태가 발생한 경우 서비스 충돌 문제 해결 [GH 4474]
- [WSL2] 현재 프로세스에 환경 변수가 없는 경우 interop 서버에서 배포 이름 쿼리
- [WSL2] stdin이 파일인 경우 interop 문제 해결
- [WSL2] Linux 커널 버전을 4.19.72로 업데이트
- [WSL2] .wslconfig를 통해 추가 커널 명령줄 매개 변수를 지정하는 기능 추가

```
[ws12]
kernelCommandLine = <string> # Additional kernel command line arguments
```

## 빌드 18990

빌드 18990에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- \\wsl\$의 디렉터리 목록의 성능 개선
- [WSL2] 추가 부팅 엔트로피를 주입합니다.[GH 4416]

- [WSL2] su/sudo를 사용할 때 Windows 인터럽트를 수정합니다.[GH 4465]

## 빌드 18980

빌드 18980에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- FILE\_READ\_DATA를 거부하는 읽기 symlink를 수정했습니다. 여기에는 "C:\Document and Settings"처럼 이전 버전과의 호환성을 위해 Windows에서 만드는 모든 symlink와 사용자 프로필 디렉터리의 symlink가 포함됩니다.
- 예기치 않은 파일 시스템 상태를 심각하지 않은 상태로 표시합니다. [GH 4334, 4305]
- [WSL2] CPU/펌웨어가 가상화를 지원하는 경우 arm64에 대한 지원이 추가됩니다.
- [WSL2] 권한 없는 사용자가 커널 로그를 볼 수 있도록 허용합니다.
- [WSL2] stdout/stderr 소켓이 닫혀 있을 때 발생하는 출력 릴레이를 수정했습니다. [GH 4375]
- [WSL2] 배터리 및 AC 어댑터 통과를 지원합니다.
- [WSL2] Linux 커널을 4.19.67로 업데이트했습니다.
- 다음과 같이 /etc/wsl.conf에서 기본 사용자 이름을 설정하는 기능이 추가되었습니다.

```
[user]
default=<string>
```

## 빌드 18975

빌드 18975에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 여러 localhost 안정성 이슈를 해결했습니다. [GH 4340]

## 빌드 18970

빌드 18970에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

- [WSL2] 시스템이 절전 모드에서 다시 시작될 때 시간을 호스트 시간과 동기화합니다. [GH 4245]
- [WSL2] 가능한 경우 Windows 볼륨에 NT symlink를 만듭니다.
- [WSL2] UTS, IPC, PID 및 탑재 네임스페이스에서 배포판을 만듭니다.
- [WSL2] 서버가 localhost에 직접 바인딩할 때 localhost 포트 릴레이를 수정합니다. [GH 4353]

- [WSL2] 출력이 리디렉션될 때 interop를 수정합니다. [GH 4337]
- [WSL2] 절대 NT symlink 변환을 지원합니다.
- [WSL2] 커널을 4.19.59로 업데이트합니다.
- [WSL2] eth0에 대한 서브넷 마스크를 적절히 설정합니다.
- [WSL2] 종료 이벤트가 신호를 받으면 콘솔 작업자 루프를 중단하도록 논리를 변경합니다.
- [WSL2] 배포판이 실행되고 있지 않으면 배포용 vhd를 배출합니다.
- [WSL2] 빈 값을 올바르게 처리하도록 구성 구문 분석 라이브러리를 수정합니다.
- [WSL2] 교차 배포판 탑재를 만들어 Docker 데스크톱을 지원합니다. 다음 줄을 /etc/wsl.conf 파일에 추가하여 배포판에서 이 동작을 옵트인할 수 있습니다.

```
[automount]
crossDistro = true
```

## 빌드 18945

빌드 18945에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- [WSL2] localhost:port를 사용하여 호스트에서 액세스할 수 있도록 WSL2에서 수신 대기 tcp 소켓을 허용합니다.
- [WSL2] 향후 이슈를 추적하도록 설치/변환 오류 및 추가 진단을 수정합니다. [GH 4105]
- [WSL2] WSL2 네트워크 이슈 진단 기능을 개선합니다.
- [WSL2] 커널 버전을 4.19.55로 업데이트합니다.
- [WSL2] docker에 필요한 구성 옵션으로 커널을 업데이트합니다. [GH 4165]
- [WSL2] 경량 유틸리티 VM에 할당된 CPU 수를 호스트와 동일하게 늘립니다(이전에는 커널 구성의 CONFIG\_NR\_CPUS에 의해 8로 제한). [GH 4137]
- [WSL2] WSL2 경량 VM에 대한 스왑 파일을 만듭니다.
- [WSL2] 사용자가 \\wsl\$\\distro를 통해 탑재를 볼 수 있도록 허용합니다(예: sshfs). [GH 4172]
- [WSL2] 9p 파일 시스템 성능을 개선합니다.
- [WSL2] vhd ACL이 무한으로 확장되지 않도록 보장합니다. [GH 4126]
- [WSL2] squashfs 및 xt\_conntrack을 지원하도록 커널 구성을 업데이트합니다. [GH 4107, 4123]
- [WSL2] interop.enabled /etc/wsl.conf 옵션을 수정합니다. [GH 4140]
- [WSL2] 파일 시스템에서 EA를 지원하지 않는 경우 ENOTSUP를 반환합니다.

- [WSL2] \\wsl\$을 사용하여 CopyFile 중단 수정
- 기본 umask를 0022로 전환하고 /etc/wsl.conf에 filesystem.umask 설정을 추가합니다.
- symlink를 적절하게 확인하도록 wslpath를 수정합니다. 이것은 19h1에서 회귀되었습니다. [GH 4078]
- WSL2 설정을 수정하기 위한 %UserProfile%\wslconfig 파일 도입

```
[ws12]
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
```

## 빌드 18917

빌드 18917에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 이제 WSL 2를 사용할 수 있습니다! 자세한 내용은 [블로그](#)를 참조하세요.
- symlink를 통해 Windows 프로세스를 시작하면 올바르게 작동하지 않는 성능 문제를 해결합니다. [GH 3999]
- wsl.exe에 wsl.exe --list --verbose, wsl.exe --list --quiet 및 wsl.exe --import --version 옵션을 추가합니다.
- wsl.exe --shutdown 옵션을 추가합니다.
- 플랜 9: 쓰기가 성공하도록 디렉터리 열기를 허용합니다.

## 빌드 18890

빌드 18890에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 비차단 소켓 유실 [GH 2913]
- 터미널의 EOF 입력이 후속 읽기를 차단할 수 있습니다. [GH 3421]
- wsl.conf를 참조하도록 resolv.conf 헤더를 업데이트합니다. [GH 3928에서 설명]
- epoll 삭제 코드의 교착 상태 [GH 3922]
- --import 및 --export에 대한 인수의 공백을 처리합니다. [GH 3932]
- mmap'd 파일을 확장하면 올바르게 작동하지 않습니다. [GH 3939]
- ARM64 \\wsl\$ 액세스가 올바르게 작동하지 않는 문제 해결
- wsl.exe에 대한 개선된 기본 아이콘을 추가합니다.

## 빌드 18342

빌드 18342에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 사용자가 Windows에서 WSL 배포판의 Linux 파일에 액세스할 수 있는 기능을 추가했습니다. 이러한 파일은 명령줄을 통해 액세스할 수 있으며 파일 탐색기, VSCode 등의 Windows 앱에서도 이러한 파일과 상호 작용할 수 있습니다. \\wsl\\<distro\_name>으로 이동하여 파일에 액세스하거나, \\wsl\$로 이동하여 실행 중인 배포판 목록을 확인하세요.
- 추가 CPU 정보 태그를 추가하고 Cpus\_allowed[\_list] 값을 수정합니다. [GH 2234]
- 리더가 아닌 스레드의 exec를 지원합니다. [GH 3800]
- 구성 업데이트 실패를 심각하지 않은 오류로 처리합니다. [GH 3785]
- 오프셋을 적절하게 처리하도록 binfmt를 업데이트합니다. [GH 3768]
- 플랜 9에 매핑 네트워크 드라이브를 사용합니다. [GH 3854]
- 바인딩 마운트를 위한 Windows -> Linux 및 Linux -> Windows 경로 변환을 지원합니다.
- 읽기 전용으로 열린 파일의 매핑에 대한 읽기 전용 섹션을 만듭니다.

## 빌드 18334

빌드 18334에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- Windows 표준 시간대가 Linux 표준 시간대에 매핑되는 방식을 다시 디자인합니다. [GH 3747]
- 메모리 누수를 해결하고 새 문자열 변환 함수를 추가합니다. [GH 3746]

- 스레드 없는 threadgroup의 SIGCONT는 no-op입니다. [GH 3741]
- /proc/self/fd에 소켓 및 epoll 파일 설명자를 올바르게 표시합니다.

## 빌드 18305

빌드 18305에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 주 스레드가 종료되면 pthread가 파일에 액세스할 수 없습니다. [GH 3589]
- TIOCSCTTY는 꼭 필요한 경우 외에는 "force" 매개 변수를 무시해야 합니다. [GH 3652]
- wsl.exe 명령줄 기능이 개선되고 가져오기/내보내기 함수가 추가되었습니다.

```
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.
```

## 빌드 18277

빌드 18277에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 빌드 18272에서 발생하는 "해당 인터페이스를 지원하지 않습니다" 오류를 해결합니다. [GH 3645]
- umount syscall에 대한 MNT\_FORCE 플래그를 무시합니다. [GH 3605]
- 공식 CreatePseudoConsole API를 사용하도록 WSL interop를 전환합니다.
- FUTEX\_WAIT가 다시 시작할 때 시간 제한 값을 유지하지 않습니다.

## 빌드 18272

빌드 18272에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- **경고:** 이 빌드에는 WSL이 작동하지 않는 문제가 있습니다. 배포를 시작하려고 하면 "해당 인터페이스를 지원하지 않습니다" 오류가 표시됩니다. 이 이슈는 해결되었으며 다음 주의 초기 참가자 빌드에 포함될 예정입니다. 이 빌드를 설치한 경우 설정->

업데이트 & 보안->복구에서 "이전 버전의 Windows 10으로 되돌리기"를 사용하여 이전 Windows 빌드로 롤백할 수 있습니다.

## 빌드 18267

빌드 18267에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 좀비 프로세스가 제거되지 않고 무기한으로 유지되는 문제를 해결합니다.
- 오류 메시지가 최대 길이를 초과하는 경우 WslRegisterDistribution이 중지됩니다. [GH 3592]
- DrvFs에서 읽기 전용 파일에 대해 fsync가 성공하도록 허용합니다. [GH 3556]
- symlink를 만들기 전에 /bin 및 /sbin 디렉터리가 존재하는지 확인합니다. [GH 3584]
- WSL 인스턴스에 대한 인스턴스 종료 시간 제한 메커니즘이 추가되었습니다. 현재 시간 제한은 15초로 설정되어 있습니다. 즉, 마지막 WSL 프로세스가 종료된 후 15초 후에 인스턴스가 종료됩니다. 배포를 즉시 종료하려면 다음을 사용합니다.

```
wslconfig.exe /terminate <DistributionName>
```

## 빌드 17763(1809)

빌드 17763에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 동일한 스레드 우선 순위를 변경하기 위한 Setpriority syscall 권한 검사가 너무 엄격 합니다. [GH 1838]
- clock\_gettime(CLOCK\_BOOTTIME)에 대한 음수 값이 반환되지 않도록, 부팅 시간에 비편향 인터럽트 시간을 사용하도록 확인합니다. [GH 3434]
- WSL binfmt 인터프리터의 symlink를 처리합니다. [GH 3424]
- threadgroup 리더 파일 설명자 정리를 보다 효율적으로 처리합니다.
- 오버플로를 방지하기 위해 KeQueryPerformanceCounter 대신 KeQueryInterruptTimePrecise를 사용하도록 WSL을 전환합니다. [GH 3252]
- Ptrace attach로 인해 시스템 호출에서 잘못된 값이 반환될 수 있습니다. [GH 1731]
- 여러 AF\_UNIX 관련 이슈를 해결합니다. [GH 3371]
- 현재 작업 디렉터리 길이가 5자 미만인 경우 WSL interop가 실패할 수 있는 이슈를 해결합니다. [GH 3379]

- 존재하지 않는 포트에 대한 루프백 연결이 실패하지 않도록 1초 지연을 방지합니다. [GH 3286]
- /proc/sys/fs/file-max 스텝 파일을 추가합니다. [GH 2893]
- 보다 정확한 IPV6 범위 정보입니다.
- PR\_SET\_PTRACER를 지원합니다. [GH 3053]
- 파이프 파일 시스템이 의도치 않게 에지에서 트리거한 epoll 이벤트를 삭제합니다. [GH 3276]
- NTFS symlink를 통해 시작된 Win32 실행 파일이 symlink 이름을 준수하지 않습니다. [GH 2909]
- 좀비 지원이 개선되었습니다. [GH 1353]
- Windows interop 동작을 제어하기 위한 wsl.conf 항목을 추가합니다. [GH 1493]

[interop]

```
enabled=false # enable launch of Windows binaries; default is true

appendWindowsPath=false # append Windows path to $PATH variable;
default is true
```

- getsockname이 경우에 따라 UNIX 소켓 패밀리 유형을 반환하지 않는 문제를 해결합니다. [GH 1774]
- TIOCSTI 지원이 추가됩니다. [GH 1863]
- 연결 중인 비차단 소켓은 쓰기 시도에 대한 EAGAIN을 반환해야 합니다. [GH 2846]
- 탑재된 VHD에서 interop를 지원합니다. [GH 3246, 3291]
- 루트 폴더에 대한 권한 확인 이슈를 해결합니다. [GH 3304]
- TTY 키보드 ioctl KDGKBTYPE, KDGKBMODE 및 KDSKBMODE에 대한 지원이 제한됩니다.
- Windows UI 앱은 백그라운드에서 시작된 경우에도 실행되어야 합니다.
- wsl -u 또는 --user 옵션이 추가됩니다. [GH 1203]
- 빠른 시작을 사용하는 경우 WSL 시작 이슈를 해결합니다. [GH 2576]
- Unix 소켓은 연결 해제된 피어 자격 증명을 유지해야 합니다. [GH 3183]
- 비차단 Unix 소켓이 EAGAIN과 함께 무기한 실패합니다. [GH 3191]
- case=off는 새로운 기본 drvfs 탑재 유형입니다. [GH 2937, 3212, 3328]
  - 자세한 내용은 [블로그](#)를 참조하세요.
- 실행 중인 배포를 중지하려면 wslconfig /terminate를 추가합니다.
- 공백이 포함된 경로를 올바르게 처리하지 않는 WSL 셸 팝업 메뉴 항목의 이슈를 해결합니다.
- 디렉터리별 대/소문자 구분을 확장 특성으로 노출합니다.
- ARM64: 캐시 유지 관리 작업을 에뮬레이트합니다. [dotnet 이슈](#)를 해결합니다.
- DrvFs: 프라이빗 범위에서 이스케이프된 문자에 해당하는 문자만 unescape합니다.
- ELF 파서 인터프리터 길이 유효성 검사의 OB1 오류를 수정합니다. [GH 3154]

- 과거 시간이 포함된 WSL 절대 타이머가 시작되지 않습니다. [GH 3091]
- 새로 만든 재분석 지점이 부모 디렉터리에 표시되는지 확인합니다.
- DrvFs에 대/소문자를 구분하는 디렉터리를 원자 단위로 만듭니다.
- 파일이 있어도 다중 스레드 작업에서 ENOENT를 반환할 수 있는 추가 이슈를 해결했습니다. [GH 2712]
- UMCI를 사용하는 경우 WSL 시작이 실패하는 문제가 해결되었습니다. [GH 3020]
- WSL을 시작하는 탐색기 팝업 메뉴를 추가합니다. [GH 437, 603, 1836]. 사용하려면 탐색기 창에서 Shift 키를 누른 상태로 마우스 오른쪽 단추를 클릭합니다.
- Unix 소켓 비차단 동작을 수정합니다. [GH 2822, 3100]
- GH 2026에 보고된 것처럼 NETLINK 명령이 중지되는 문제를 해결합니다.
- 탑재 전파 플래그에 대한 지원이 추가됩니다. [GH 2911]
- 자르기가 Inotify 이벤트를 발생시키지 않는 이슈를 해결합니다. [GH 2978]
- wsl.exe가 셸 없이 단일 이진 파일을 호출하는 --exec 옵션이 추가됩니다.
- wsl.exe가 특정 배포판을 선택하는 --distribution 옵션이 추가됩니다.
- dmesg에 대한 지원이 제한됩니다. 이제 애플리케이션에서 dmesg에 기록할 수 있습니다. WSL 드라이버는 제한된 정보를 dmesg에 기록합니다. 이후에 드라이버의 다른 정보/진단을 수행하도록 이 기능이 확장될 수 있습니다.
  - 참고: dmesg는 현재 `/dev/kmsg` 디바이스 인터페이스를 통해 지원됩니다. `syslog` syscall 인터페이스는 아직 지원되지 않습니다. 따라서 `-s`, `-c` 같은 `dmesg` 명령줄 옵션 중 일부는 작동하지 않습니다.
- 네이티브와 일치하도록 직렬 디바이스의 기본 그리드 및 모드를 변경합니다. [GH 3042]
- DrvFs는 이제 확장된 특성을 지원합니다.
  - 참고: DrvFs는 확장 특성의 이름에 대한 몇 가지 제한이 있습니다. 일부 문자(예: '/', ':' 및 '\*')는 허용되지 않으며, 확장 특성 이름은 DrvFs에서 대/소문자를 구분하지 않습니다.

## 빌드 18252(앞으로 건너뛰기)

빌드 18252에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- lssmmanager.dll에서 init 및 bsdtar 이진 파일을 별도의 도구 폴더로 이동합니다.
- CLONE\_FILES를 사용할 때 파일 설명자를 닫는 동안 경합을 해결합니다.
- DrvFs 경로를 변환할 때 /proc/pid/mountinfo의 선택적 필드를 처리합니다.
- S\_IFREG에 대한 메타데이터 지원 없이 DrvFs mknod가 성공하도록 허용합니다.
- DrvFs에서 만든 읽기 전용 파일에는 readonly 특성 세트가 있어야 합니다. [GH 3411]
- DrvFs 탑재를 처리하는 /sbin/mountdrvfs 도우미가 추가됩니다.

- DrvFs에서 POSIX rename을 사용합니다.
- 볼륨 GUID 없이 볼륨의 경로 변환을 허용합니다.

## 빌드 17738(패스트)

빌드 17738에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 동일한 스레드 우선 순위를 변경하기 위한 Setpriority syscall 권한 검사가 너무 엄격합니다. [GH 1838]
- clock\_gettime(CLOCK\_BOOTTIME)에 대한 음수 값이 반환되지 않도록, 부팅 시간에 비편향 인터럽트 시간을 사용하도록 확인합니다. [GH 3434]
- WSL binfmt 인터프리터의 symlink를 처리합니다. [GH 3424]
- threadgroup 리더 파일 설명자 정리를 보다 효율적으로 처리합니다.

## 빌드 17728(패스트)

빌드 17728에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 오버플로를 방지하기 위해 KeQueryPerformanceCounter 대신 KeQueryInterruptTimePrecise를 사용하도록 WSL을 전환합니다. [GH 3252]
- Ptrace attach로 인해 시스템 호출에서 잘못된 값이 반환될 수 있습니다. [GH 1731]
- 여러 AF\_UNIX 관련 이슈를 해결합니다. [GH 3371]
- 현재 작업 디렉터리 길이가 5자 미만인 경우 WSL interop가 실패할 수 있는 이슈를 해결합니다. [GH 3379]

## 빌드 18204(앞으로 건너뛰기)

빌드 18204에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 파이프 파일 시스템이 의도치 않게 에지에서 트리거한 epoll 이벤트를 삭제합니다. [GH 3276]
- NTFS symlink를 통해 시작된 Win32 실행 파일이 symlink 이름을 준수하지 않습니다. [GH 2909]

# 빌드 17723(패스트)

빌드 17723에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 존재하지 않는 포트에 대한 루프백 연결이 실패하지 않도록 1초 지연을 방지합니다. [GH 3286]
- /proc/sys/fs/file-max 스텝 파일을 추가합니다. [GH 2893]
- 보다 정확한 IPV6 범위 정보입니다.
- PR\_SET\_PTRACER를 지원합니다. [GH 3053]
- 파이프 파일 시스템이 의도치 않게 에지에서 트리거한 epoll 이벤트를 삭제합니다. [GH 3276]
- NTFS symlink를 통해 시작된 Win32 실행 파일이 symlink 이름을 준수하지 않습니다. [GH 2909]

# 빌드 17713

빌드 17713에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 좀비 지원이 개선되었습니다. [GH 1353]
- Windows interop 동작을 제어하기 위한 wsl.conf 항목을 추가합니다. [GH 1493]

```
[interop]

enabled=false # enable launch of Windows binaries; default is true

appendWindowsPath=false # append Windows path to $PATH variable;
default is true
```

- getsockname이 경우에 따라 UNIX 소켓 패밀리 유형을 반환하지 않는 문제를 해결합니다. [GH 1774]
- TIOCSTI 지원이 추가됩니다. [GH 1863]
- 연결 중인 비차단 소켓은 쓰기 시도에 대한 EAGAIN을 반환해야 합니다. [GH 2846]
- 탑재된 VHD에서 interop를 지원합니다. [GH 3246, 3291]
- 루트 폴더에 대한 권한 확인 이슈를 해결합니다. [GH 3304]
- TTY 키보드 ioctl KDGKBTYP, KDGKBMODE 및 KDSKBMODE에 대한 지원이 제한됩니다.

- Windows UI 앱은 백그라운드에서 시작된 경우에도 실행되어야 합니다.

## 빌드 17704

빌드 17704에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- wsl -u 또는 --user 옵션이 추가됩니다. [GH 1203]
- 빠른 시작을 사용하는 경우 WSL 시작 이슈를 해결합니다. [GH 2576]
- Unix 소켓은 연결 해제된 피어 자격 증명을 유지해야 합니다. [GH 3183]
- 비차단 Unix 소켓이 EAGAIN과 함께 무기한 실패합니다. [GH 3191]
- case=off는 새로운 기본 drvfs 탑재 유형입니다. [GH 2937, 3212, 3328]
  - 자세한 내용은 [블로그](#)를 참조하세요.
- 실행 중인 배포를 중지하려면 wslconfig /terminate를 추가합니다.

## 빌드 17692

빌드 17692에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 공백이 포함된 경로를 올바르게 처리하지 않는 WSL 셸 팝업 메뉴 항목의 이슈를 해결합니다.
- 디렉터리별 대/소문자 구분을 확장 특성으로 노출합니다.
- ARM64: 캐시 유지 관리 작업을 에뮬레이트합니다. [dotnet 이슈](#)를 해결합니다.
- DrvFs: 프라이빗 범위에서 이스케이프된 문자에 해당하는 문자만 unescape합니다.

## 빌드 17686

빌드 17686에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- ELF 파서 인터프리터 길이 유효성 검사의 OB1 오류를 수정합니다. [GH 3154]
- 과거 시간이 포함된 WSL 절대 타이머가 시작되지 않습니다. [GH 3091]
- 새로 만든 재분석 지점이 부모 디렉터리에 표시되는지 확인합니다.
- DrvFs에 대/소문자를 구분하는 디렉터리를 원자 단위로 만듭니다.

## 빌드 17677

빌드 17677에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 파일이 있어도 다중 스레드 작업에서 ENOENT를 반환할 수 있는 추가 이슈를 해결했습니다. [GH 2712]
- UMCI를 사용하는 경우 WSL 시작이 실패하는 문제가 해결되었습니다. [GH 3020]

## 빌드 17666

빌드 17666에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

**경고: 일부 AMD 칩셋에서 WSL이 실행되지 않는 이슈가 있습니다 [GH 3134]. 해결 방법이 준비되어 있으며 참가자 빌드 분기에 적용 중입니다.**

- WSL을 시작하는 탐색기 팝업 메뉴를 추가합니다. [GH 437, 603, 1836]. 사용하려면 탐색기 창에서 Shift 키를 누른 상태로 마우스 오른쪽 단추를 클릭합니다.
- Unix 소켓 비차단 동작을 수정합니다. [GH 2822, 3100]
- GH 2026에 보고된 것처럼 NETLINK 명령이 중지되는 문제를 해결합니다.
- 탑재 전파 플래그에 대한 지원이 추가됩니다. [GH 2911]
- 자르기가 Inotify 이벤트를 발생시키지 않는 이슈를 해결합니다. [GH 2978]
- wsl.exe가 셸 없이 단일 이진 파일을 호출하는 --exec 옵션이 추가됩니다.
- wsl.exe가 특정 배포판을 선택하는 --distribution 옵션이 추가됩니다.

## 빌드 17655(앞으로 건너뛰기)

빌드 17655에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- dmesg에 대한 지원이 제한됩니다. 이제 애플리케이션에서 dmesg에 기록할 수 있습니다. WSL 드라이버는 제한된 정보를 dmesg에 기록합니다. 이후에 드라이버의 다른 정보/진단을 수행하도록 이 기능이 확장될 수 있습니다.

- 참고: dmesg는 현재 `/dev/kmsg` 디바이스 인터페이스를 통해 지원됩니다. `syslog` syscall 인터페이스는 아직 지원되지 않습니다. 따라서 `-s`, `-c` 같은 `dmesg` 명령줄 옵션 중 일부는 작동하지 않습니다.
- 파일이 있어도 다중 스레드 작업에서 ENOENT를 반환할 수 있는 이슈를 해결했습니다. [GH 2712]

## 빌드 17639(앞으로 건너뛰기)

빌드 17639에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 네이티브와 일치하도록 직렬 디바이스의 기본 그리드 및 모드를 변경합니다. [GH 3042]
- DrvFs는 이제 확장된 특성을 지원합니다.
  - 참고: DrvFs는 확장 특성의 이름에 대한 몇 가지 제한이 있습니다. 특히 일부 문자 (예: '/', ':' 및 '\*')는 허용되지 않으며, 확장 특성 이름은 DrvFs에서 대/소문자를 구분하지 않습니다.

## 빌드 17133(패스트)

빌드 17133에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- WSL의 중단 문제를 해결합니다. [GH 3039, 3034]

## 빌드 17128(패스트)

빌드 17128에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### WSL

- 없음

## 빌드 17627(앞으로 건너뛰기)

빌드 17627에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- futex pi 인식 작업에 대한 지원이 추가됩니다. [GH 1006]
  - 우선 순위는 현재 지원되는 WSL 기능이 아니므로 제한이 있지만, 표준 사용의 차단을 해제해야 합니다.
- Windows 방화벽은 WSL 프로세스를 지원합니다. [GH 1852]
  - 예를 들어 WSL python 프로세스가 모든 포트에서 수신 대기하도록 허용하려면 관리자 권한 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` 를 사용합니다.
  - 방화벽 규칙을 추가하는 방법에 대한 자세한 내용은 [링크](#) 를 참조하세요.
- wsl.exe를 사용할 때 사용자의 기본 셸을 준수합니다. [GH 2372]
- 모든 네트워크 인터페이스를 이더넷으로 보고합니다. [GH 2996]
- 손상된 /etc/passwd 파일을 보다 효율적으로 처리합니다. [GH 3001]

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17618(앞으로 건너뛰기)

빌드 17618에 대한 일반적인 Windows 정보는 [Windows 블로그](#) 를 참조하세요.

## WSL

- NT interop에 대한 pseudoconsole 기능을 도입합니다. [GH 988, 1366, 1433, 1542, 2370, 2406]
- 레거시 설치 메커니즘(lxrun.exe)은 더 이상 사용되지 않습니다. 배포판 설치를 위한 메커니즘은 Microsoft Store를 통해 지원됩니다.

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17110

빌드 17110에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- Windows에서 /init를 종료할 수 있습니다. [GH 2928]
- 이제 DrvFs는 기본적으로 디렉터리 단위 대/소문자 구분을 사용합니다("case=dir" 탑재 옵션과 동일).
  - "case=force"(이전 동작)를 사용하려면 레지스트리 키를 설정해야 합니다.  
"case=force"를 사용해야 하는 경우 reg add  
HKLM\SYSTEM\CurrentControlSet\Services\lxss /v  
DrvFsAllowForceCaseSensitivity /t REG\_DWORD /d 1 명령을 실행하여  
"case=force"를 설정합니다.
  - 이전 버전의 Windows에서 WSL을 사용하여 만든 기존 디렉터리가 있고 대/소문자를 구분해야 하는 경우 다음 fsutil.exe를 사용하여 대/소문자를 구분하도록 설정합니다. fsutil.exe file setcasesensitiveinfo <path> enable
- NULL은 uname syscall에서 반환된 문자열을 종료합니다.

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17107

빌드 17107에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- 마스터 pty 엔드포인트에서 TCSETSF 및 TCSETSW를 지원합니다. [GH 2552]
- 동시 interop 프로세스를 시작하면 EINVAL이 발생할 수 있습니다. [GH 2813]

- /proc/pid/status에서 적절한 추적 상태를 표시하도록 PTRACE\_ATTACH를 수정합니다.
- CLEARTID 및 SETTID 플래그를 모두 사용하여 복제된 단기 프로세스가 TID 주소를 지우지 않고 종료될 수 있는 경합을 수정합니다.
- 17093 이전 빌드에서 이동할 때 Linux 파일 시스템 디렉터리를 업그레이드하는 동안 메시지를 표시합니다. 17093 파일 시스템 변경 내용에 대한 자세한 내용은 [17093](#) 릴리스 정보를 참조하세요.

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17101

빌드 17101에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- signalfd를 지원합니다. [GH 129]
- 지원되지 않는 NTFS 문자가 포함된 파일 이름을 프라이빗 유니코드 문자로 인코딩하여 지원합니다. [GH 1514]
- 쓰기가 지원되지 않는 경우 자동 탑재는 읽기 전용으로 대체됩니다. [GH 2603]
- 유니코드 서로게이트 쌍(예:이모지 문자)을 붙여넣을 수 있습니다. [GH 2765]
- /proc 및 /sys의 의사 파일은 select, poll, epoll 등에서 read and write ready를 반환해야 합니다. [GH 2838]
- 레지스트리가 변조되거나 손상된 경우 서비스가 무한 루프에 빠질 수 있는 이슈를 해결합니다.
- 최신 버전의 iproute2(업스트림 4.14)를 사용하도록 netlink 메시지를 수정합니다.

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17093

빌드 17093에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 중요:

이 빌드로 업그레이드한 후 처음으로 WSL을 시작할 때 Linux 파일 시스템 디렉터리를 업그레이드하는 작업을 수행해야 합니다. 이 작업은 몇 분 정도 걸릴 수 있으므로 WSL이 느리게 시작되는 것처럼 보일 수 있습니다. 이 작업은 스토어에서 설치한 배포판마다 한 번씩만 수행됩니다.

- DrvFs의 대/소문자 구분 지원이 개선되었습니다.
  - 이제 DrvFs는 디렉터리 단위 대/소문자 구분을 지원합니다. 이 새로운 플래그를 디렉터리에 설정하여 해당 디렉터리의 모든 작업을 대/소문자 구분으로 나타낼 수 있으며, 이렇게 하면 Windows 애플리케이션에서도 대/소문자만 다른 파일을 올바르게 열 수 있습니다.
  - DrvFs에는 디렉터리 단위로 대/소문자 구분을 제어하는 다음과 같은 새로운 탑재 옵션이 있습니다.
    - case=force: 모든 디렉터리가 대/소문자 구분으로 처리됩니다(드라이브 루트 제외). WSL에서 만든 새 디렉터리는 대/소문자를 구분하는 것으로 표시됩니다. 새 디렉터리를 대/소문자 구분으로 표시하는 것만 제외하면 레거시 동작입니다.
    - case=dir: 디렉터리 단위 대/소문자 구분 플래그가 있는 디렉터리만 대/소문자 구분으로 처리되고, 다른 디렉터리는 대/소문자를 구분하지 않습니다. WSL에서 만든 새 디렉터리는 대/소문자를 구분하는 것으로 표시됩니다.
    - case=off: 디렉터리 단위 대/소문자 구분 플래그가 있는 디렉터리만 대/소문자 구분으로 처리되고, 다른 디렉터리는 대/소문자를 구분하지 않습니다. WSL로 만든 새 디렉터리는 대/소문자를 구분하지 않는 것으로 표시됩니다.
  - 참고: 이전 릴리스의 WSL에서 만든 디렉터리는 이 플래그가 설정되지 않았으므로 "case=dir" 옵션을 사용하는 경우 대/소문자를 구분하는 것으로 간주되지 않습니다. 기존 디렉터리에 이 플래그를 설정하는 방법이 곧 제공될 예정입니다.
  - 이러한 옵션을 사용하는 탑재 예제로는 sudo mount -t drvfs C: /mnt/c -o case=dir이 있습니다(기존 드라이브의 경우 먼저 탑재 해제해야만 다른 옵션으로 탑재 가능).
  - 현재는 case=force가 여전히 기본 옵션입니다. 향후 case=dir로 변경될 것입니다.
- 이제 DrvFs를 탑재할 때 Windows 경로에 슬래시를 사용할 수 있습니다(예: sudo mount -t drvfs //server/share /mnt/share).
- 이제 인스턴스를 시작하는 동안 WSL에서 /etc/fstab 파일을 처리합니다. [GH 2636]
  - 이 작업은 DrvFs 드라이브를 자동으로 탑재하기 전에 수행됩니다. fstab을 통해 이미 탑재된 드라이브는 자동으로 다시 탑재되지 않으므로 특정 드라이브의 탑재 지점을 변경할 수 있습니다.

- 이 동작은 wsl.conf를 사용하여 해제할 수 있습니다.
- /proc의 mount, mountinfo 및 mountstats 파일은 백슬래시 및 공백과 같은 특수 문자를 적절히 이스케이프합니다. [GH 2799]
- DrvFs를 사용하여 만든 특수 파일(예: WSL 기호화된 링크) 또는 메타데이터를 사용하는 경우 fifos 및 소켓을 이제 Windows에서 복사하여 이동할 수 있습니다.

## wsl.conf를 사용하여 보다 유연하게 WSL 구성 가능

하위 시스템을 시작할 때마다 적용되는 WSL의 특정 기능을 자동으로 구성하는 방법이 추가되었습니다. 여기에는 자동 탑재 옵션 및 네트워크 구성이 포함됩니다. 블로그 게시물 <https://aka.ms/wslconf> 에서 자세히 알아보세요.

## AF\_UNIX를 사용하면 WSL의 Linux 프로세스와 Windows 네이티브 프로세스 간에 소켓 연결이 가능합니다.

WSL 및 Windows 애플리케이션은 이제 Unix 소켓을 통해 서로 통신할 수 있습니다. Windows에서 서비스를 실행하고 Windows 및 WSL 앱 모두에서 이 서비스를 제공하려 한다고 가정해 보겠습니다. 이제 Unix 소켓을 사용하면 가능합니다. 블로그 게시물 <https://aka.ms/afunixinterop> 에서 자세히 알아보세요.

## WSL

- MAP\_NORESERVE를 통해 mmap() 지원 [GH 121, 2784]
- CLONE\_PTRACE 및 CLONE\_UNTRACED 지원 [GH 121, 2781]
- 복제 시 비-SIGCHLD 종료 신호 처리 [GH 121, 2781]
- /proc/sys/fs/inotify/max\_user\_instances 및 /proc/sys/fs/inotify/max\_user\_watches 를 스럽습니다. [GH 1705]
- 0이 아닌 오프셋이 있는 로드 헤더가 포함된 ELF 이진 파일을 로드하는 동안 오류 발생 [GH 1884]
- 이미지를 로드할 때 후행 페이지 바이트 삭제
- execve가 자동으로 프로세스를 종료하는 사례 감소

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17083

빌드 17083에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- epoll 관련 버그 검사 수정 [GH 2798, 2801, 2857]
- ASLR을 끄면 중단되는 문제 해결 [GH 1185, 2870]
- mmap 작업이 원자성으로 표시되는지 확인 [GH 2732]

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17074

빌드 17074에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- DrvFs 메타데이터의 스토리지 형식이 수정되었습니다. [GH 2777]  
**중요:** 이 빌드 전에 생성된 DrvFs 메타데이터가 잘못 표시되거나 전혀 표시되지 않습니다. 영향을 받는 파일을 수정하려면 chmod 및 chown을 사용하여 메타데이터를 다시 적용합니다.
- 여러 신호 및 다시 시작 가능한 syscall 관련 이슈 해결

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17063

빌드 17063에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## WSL

- DrvFs는 추가 Linux 메타데이터를 지원합니다. 따라서 chmod/chown을 사용하여 파일의 소유자와 모드를 설정할 수 있으며, fifos, unix 소켓, 디바이스 파일 등의 특수 파일을 만들 수 있습니다. 이 기능은 아직 실험 중이므로 지금은 기본적으로 사용되지 않습니다. 참고: DrvFs에서 사용하는 메타데이터 형식의 버그를 수정했습니다. 메타데이터는 이 빌드에서 실험에 사용되지만, 향후 빌드는 이 빌드에서 생성된 메타데이터를 올바르게 읽을 수 없습니다. 수정된 파일의 소유자를 수동으로 업데이트하고 사용자 지정 디바이스 ID를 사용하는 디바이스를 다시 만들어야 할 수도 있습니다.

DrvFs를 사용하려면 다음과 같이 메타데이터 옵션으로 DrvFs를 탑재합니다(기존 탑재에서 DrvFs를 사용하려면 먼저 DrvFs를 분리해야 함).

Bash

```
mount -t drvfs C: /mnt/c -o metadata
```

Linux 권한은 파일에 추가 메타데이터로 추가되며, Windows 권한에 영향을 주지 않습니다. Windows 편집기를 사용하여 파일을 편집하면 메타데이터가 제거될 수 있습니다. 이 경우 파일이 기본 권한으로 되돌아갑니다.

- 메타데이터를 사용하지 않고 파일을 제어하는 탑재 옵션이 DrvFs에 추가되었습니다.
  - uid: 모든 파일의 소유자에게 사용되는 사용자 ID입니다.
  - gid: 모든 파일의 소유자에게 사용되는 그룹 ID입니다.
  - umask: 모든 파일과 디렉터리에서 제외할 권한의 8진수 마스크입니다.
  - fmask: 모든 일반 파일에서 제외할 권한의 8진수 마스크입니다.
  - dmask: 모든 디렉터리에서 제외할 권한의 8진수 마스크입니다.

예:

```
mount -t drvfs C: /mnt/c -o uid=1000,gid=1000,umask=22,fmask=111
```

메타데이터 없이 파일에 대한 기본 권한을 지정하려면 메타데이터 옵션과 결합합니다.

- WSL과 Win32 간에 환경 변수가 어떻게 흐르는지 구성하는 새 환경 변수 `WSLENV` 가 도입되었습니다.

예:

Bash

```
WSLENV=GOPATH/1:USERPROFILE/pu:DISPLAY
```

`WSLENV`는 WSL의 Win32 또는 Win32 프로세스에서 WSL 프로세스를 시작할 때 포함 할 수 있는 콜론으로 구분된 환경 변수 목록입니다. 각 변수에 슬래시를 접미사로 붙이고 그 뒤에 플래그를 사용하여 변환 방식을 지정할 수 있습니다.

- p: 이 값은 WSL 경로와 Win32 경로 간에 변환해야 하는 경로입니다.
- l: 이 값은 경로 목록입니다. WSL에서는 콜론으로 구분된 목록입니다. Win32에서 는 세미콜론으로 구분된 목록입니다.
- u: 이 값은 Win32에서 WSL을 호출할 때만 포함되어야 합니다.
- w: 이 값은 WSL에서 Win32를 호출할 때만 포함되어야 합니다.

.bashrc 또는 사용자 지정 Windows 환경에서 사용자에 대한 `WSLENV`를 설정할 수 있 습니다.

- drvfs 탑재는 tar, cp-p의 타임스탬프를 올바르게 유지합니다. (GH 1939)
- drvfs symlink는 올바른 크기를 보고합니다. (GH 2641)
- 매우 큰 IO 크기에 대한 읽기/쓰기가 작동합니다. (GH 2653)
- waitpid는 프로세스 그룹 ID와 함께 작동합니다. (GH 2534)
- 큰 예약 영역에 대한 mmap 성능이 크게 향상되었습니다. ghc 성능이 향상됩니다. (GH 1671)
- 퍼스널리티에서 READ\_IMPLIES\_EXEC를 지원합니다. maxima 및 clisp를 수정합니다. (GH 1185)
- mprotect에서 PROT\_GROWSDOWN을 지원합니다. clisp를 수정합니다. (GH 1128)
- 오버커밋 모드의 페이지 오류를 수정합니다. sbcl을 수정합니다. (GH 1128)
- 복제에서 더 많은 플래그 조합을 지원합니다.
- epoll 파일의 select/epoll을 지원합니다(이전에는 no-op).
- ptrace에 구현되지 않은 syscall을 알립니다.
- resolv.conf 이름 서버를 생성할 때 작동하지 않는 인터페이스를 무시합니다. [GH 2694]
- 실제 주소가 없는 네트워크 인터페이스를 열거합니다. [GH 2685]

- 추가로 버그를 수정하고 기능을 개선했습니다.

## Windows 기반 개발자가 사용할 수 있는 Linux 도구

- Windows 명령줄 도구 체인에는 bsdtar(tar) 및 curl이 포함되어 있습니다. 이 [블로그](#)를 읽고 새로 추가된 이 두 가지 도구에 대해 자세히 살펴보고 두 도구가 Windows에서 어떤 개발자 환경을 제공하는지 알아보세요.
- AF\_UNIX는 Windows 참가자 SDK(17061+)에서 사용할 수 있습니다. 이 [블로그](#)를 읽고 AF\_UNIX에 대해 자세히 살펴보고 Windows 기반의 개발자가 어떻게 사용할 수 있는지 알아보세요.

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17046

빌드 17046에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

## WSL

- 활성 터미널 없이 프로세스를 실행할 수 있습니다. [GH 709, 1007, 1511, 2252, 2391 등]
- CLONE\_VFORK 및 CLONE\_VM에 대한 지원이 향상되었습니다. [GH 1878, 2615]
- WSL 네트워킹 작업을 위한 TDI 필터 드라이버를 건너뜁니다. [GH 1554]
- 특정 조건이 충족되면 DrvFs에서 NT symlink를 만듭니다. [GH 353, 1475, 2602]
  - 연결 대상은 상대 경로여야 하고, 탐색 지점이나 symlink와 교차해서는 안 되며, 존재해야 합니다.
  - 개발자 모드가 켜져 있지 않으면 사용자에게 SE\_CREATE\_SYMBOLIC\_LINK\_PRIVILEGE가 있어야 합니다(일반적으로 관리자 권한으로 wsl.exe를 실행해야 함).
  - 그 외의 상황에서는 DrvFs가 여전히 WSL symlink를 만듭니다.
- 관리자 권한 및 비관리자 권한 WSL 인스턴스를 동시에 실행할 수 있습니다.

- /proc/sys/kernel/yama/ptrace\_scope를 지원합니다.
- WSL<->Windows 경로를 변환하는 wslpath가 추가됩니다. [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'
```

## 콘솔

- 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17040

빌드 17040에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

## WSL

- 17035 이후로는 수정 사항이 없습니다.

## 콘솔

- 17035 이후로는 수정 사항이 없습니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17035

빌드 17035에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- DrvFs의 파일에 액세스할 때 가끔 EINVAL과 함께 액세스가 실패할 수 있습니다. [GH 2448]

### 콘솔

- VT 모드에서 줄을 삽입/삭제할 때 일부 색이 손실됩니다.

## LTP 결과:

테스트를 진행 중입니다.

# 빌드 17025

빌드 17025에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- 새 프로그램 프로세스 그룹에서 초기 프로세스를 시작합니다. [GH 1653, 2510]
- SIGHUP 전송 픽스 [GH 2496]
- 제공된 이름이 없으면 기본 가상 브리지 이름을 생성합니다. [GH 2497]
- /proc/sys/kernel/random/boot\_id를 구현합니다. [GH 2518]
- 더 많은 interop stdout/stderr 파이프 픽스를 제공합니다.
- syncfs 시스템 호출을 스텁합니다.

### 콘솔

- 타사 콘솔에 대한 입력 VT 변환을 수정합니다. [GH 111]

## LTP 결과:

테스트를 진행 중입니다.

## 빌드 17017

빌드 17017에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

#### WSL

- 빈 ELF 프로그램 헤더를 무시합니다. [GH 330]
- LxssManager가 비 대화형 사용자에 대한 WSL 인스턴스를 만들도록 허용합니다(ssh 및 예약된 작업 지원). [GH 777, 1602]
- WSL->Win32->WSL("개시") 시나리오를 지원합니다. [GH 1228]
- Interop를 통해 호출되는 콘솔 앱의 종료를 제한적으로 지원합니다. [GH 1614]
- devpts에 대한 탑재 옵션을 지원합니다. [GH 1948]
- Ptrace가 자식 시작을 차단합니다. [GH 2333]
- EPOLLET에 일부 이벤트가 없습니다. [GH 2462]
- PTRACE\_GETSIGINFO에 대한 더 많은 데이터를 반환합니다.
- lseek를 사용하는 Getdents가 잘못된 결과를 제공합니다.
- 일부 Win32 interop 앱이 중단되고, 더 이상 데이터가 없는 파이프에서 입력을 기다립니다.
- tty/pty 파일에 O\_ASYNC가 지원됩니다.
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

#### 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

### LTP 결과:

테스트를 진행 중입니다.

## Fall Creators Update

## 빌드 16288

빌드 16288에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- 소켓 파일 설명자에 대한 uid, gid 및 모드를 올바르게 초기화하고 보고합니다. [GH 2490]
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

### 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

### LTP 결과:

16273 이후로 변경된 내용이 없습니다.

## 빌드 16278

빌드 162738에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- LX MM 상태를 해제할 때 파일 지원 섹션의 매핑된 보기의 명시적 매핑을 해제합니다. [GH 2415]
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

### 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

### LTP 결과:

16273 이후로 변경된 내용이 없습니다.

## 빌드 16275

빌드 162735에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- 이 릴리스에는 WSL과 관련된 변경 내용이 없습니다.

### 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

### LTP 결과:

16273 이후로 변경된 내용이 없습니다.

## 빌드 16273

빌드 16273에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

### WSL

- DrvFs에서 가끔 디렉터리의 파일 형식을 잘못 보고하는 이슈가 해결되었습니다. [GH 2392]
- uevent를 사용하는 프로그램의 차단을 해제하는 NETLINK\_KOBJECT\_UEVENT 소켓을 만들 수 있습니다. [GH 1121, 2293, 2242, 2295, 2235, 648, 637]
- 비차단 연결에 대한 지원이 추가되었습니다. [GH 903, 1391, 1584, 1585, 1829, 2290, 2314]
- CLONE\_FS 복제 시스템 호출 플래그를 구현했습니다. [GH 2242]
- NT interop에서 탭 또는 따옴표를 올바르게 처리하지 않는 이슈를 해결했습니다. [GH 1625, 2164]
- WSL 인스턴스를 다시 시작하려고 할 때 발생하는 액세스 거부 오류를 해결했습니다. [GH 651, 2095]
- futex FUTEX\_REQUEUE 및 FUTEX\_CMP\_REQUEUE 작업을 구현했습니다. [GH 2242]
- 다양한 SysFs 파일에 대한 권한을 수정했습니다. [GH 2214]
- 설치하는 동안 Haskell 스택이 중단되는 오류를 수정했습니다. [GH 2290]
- binfmt\_misc 'C' 'O' 및 'P' 플래그를 구현했습니다. [GH 2103]
- /proc/sys/kernel /shmmax /shmmni & /threads-max 추가 [GH 1753]
- ioprio\_set 시스템 호출에 대한 부분 지원을 추가했습니다. [GH 498]

- SO\_REUSEPORT 스텁 & netlink 소켓에 대한 SO\_PASSCRED 지원 추가 [GH 69]
- 현재 배포판을 설치 중이거나 제거 중일 때 RegisterDistribuiton에서 다른 오류 코드를 반환합니다.
- wslconfig.exe를 통해 부분적으로 설치된 WSL 배포판을 등록 취소할 수 있습니다.
- udp::msg\_peek에서 python 소켓 테스트가 중단되는 오류를 수정했습니다.
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

## LTP 결과:

총 테스트: 1904

건너뛴 총 테스트: 209

총 실패 횟수: 229

## 빌드 16257

빌드 16257에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

## WSL

- prlimit64 시스템 호출을 구현했습니다.
- ulimit -n에 대한 지원이 추가되었습니다(setrlimit RLIMIT\_NOFILE). [GH 1688]
- TCP 소켓에 대한 MSG\_MORE를 스텁합니다. [GH 2351]
- 잘못된 AT\_EXECFN 보조 벡터 동작을 수정합니다. [GH 2133]
- 콘솔/tty의 복사/붙여넣기 동작을 수정하고, 보다 효율적인 전체 버퍼 처리를 추가합니다. [GH 2204, 2131]
- set-user-ID 및 set-group-ID 프로그램의 보조 벡터에서 AT\_SECURE를 설정합니다. [GH 2031]
- Psuedo-terminal 마스터 엔드포인트에서 TIOCPGRP를 처리하지 않습니다. [GH 1063]
- lseek가 LxFs의 디렉터리를 되감는 오류를 수정했습니다. [GH 2310]
- 사용량이 많은 후 /dev/ptmx가 잠깁니다. [GH 1882]
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 콘솔

- 모든 곳에서 가로 선/밑줄을 수정했습니다. [GH 2168]
- 프로세스 순서가 변경되면 NPM을 닫기 어려워지는 문제를 수정했습니다. [GH 2170]
- 새로운 색 구성표를 추가했습니다  
(<https://blogs.msdn.microsoft.com/commandline/2017/08/02/updating-the-windows-console-colors/> ).

## LTP 결과:

16251 이후로 변경된 내용이 없습니다.

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

`prlimit64`

## 알려진 문제

### [GitHub 이슈 2392: WSL에서 Windows 폴더를 인식할 수 없음...](#)

빌드 16257의 경우 WSL에서 `/mnt/c/...` 명령을 통해 Windows 파일/폴더를 열거할 때 이슈가 있습니다. 이 이슈가 해결되었으며, 2017년 8월 14일부터 일주일 동안 참가자 빌드에서 릴리스해야 합니다.

## 빌드 16251

빌드 16251에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

## WSL

- WSL 선택적 구성 요소에서 베타 태그를 제거했습니다. 자세한 내용은 [블로그 게시물](#)을 참조하세요.

- exec의 set-user-ID 및 set-group-ID 이진 파일에 대한 saved-set uid 및 gid를 올바르게 초기화합니다. [GH 962, 1415, 2072]
- ptrace PTTRACE\_O\_TRACEEXIT에 대한 지원이 추가되었습니다. [GH 555]
- NT\_FPREGSET를 사용하는 ptrace PTRACE\_GETFPREGS 및 PTRACE\_GETREGSET에 대한 지원이 추가되었습니다. [GH 555]
- 무시된 신호에서 중지하도록 ptrace를 수정했습니다.
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 콘솔

- 이 릴리스에는 콘솔과 관련된 변경 내용이 없습니다.

## LTP 결과:

통과한 테스트 수: 768

실패한 테스트 수: 244

건너뛴 테스트 수: 96

## 빌드 16241

빌드 16241에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

## WSL

- 이 릴리스에는 WSL과 관련된 변경 내용이 없습니다.

## 콘솔

- 교차 줄 DEC에 대해 잘못된 문자를 출력하는 문제가 해결되었으며, 이 문제는 [여기](#)에 처음 보고되었습니다.
- 코드 페이지 65001(utf8)에 출력 텍스트가 표시되지 않는 문제를 해결했습니다.
- 선택 변경 시 한 색의 RGB 값에 적용한 변경 내용을 색상표의 다른 부분으로 전송하지 마세요. 이렇게 하면 콘솔 속성 시트를 훨씬 쉽게 사용할 수 있습니다.
- Ctrl+S가 제대로 작동하지 않는 것 같습니다.
- ANSI 이스케이프 코드에서 Un-Bold/-Dim이 완전히 없어졌습니다. [GH 2174]
- 콘솔이 Vim 컬러 테마를 올바르게 지원하지 않습니다. [GH 1706]

- 특정 문자를 붙여넣을 수 없습니다. [GH 2149]
- 항목이 편집/명령줄에 있는 경우 재배치 크기 조정이 bash 창 크기 조정과 이상하게 상호 작용합니다. [GH ConEmu 1123]
- Ctrl-L 키를 누르면 화면이 더티 상태로 유지됩니다. [GH 1978]
- HDPI에서 VT를 표시할 때 콘솔 렌더링 버그가 있습니다. [GH 1907]
- 유니코드 문자 U+30FB를 사용할 때 일본어 문자가 이상하게 보입니다. [GH 2146]
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 빌드 16237

빌드 16237에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- lxfs에서 EA가 없는 파일에는 기본 특성을 사용합니다(root, root, 0000).
- 확장된 특성을 사용하는 배포에 대한 지원이 추가되었습니다.
- getdents 및 getdents64에서 반환된 항목의 여백이 수정되었습니다.
- shmctl SHM\_STAT 시스템 호출에 대한 권한 확인이 수정되었습니다. [GH 2068]
- ttys에 대한 잘못된 초기 epoll 상태가 수정되었습니다. [GH 2231]
- DrvFs readdir이 일부 항목을 반환하지 않는 문제가 수정되었습니다. [GH 2077]
- 파일의 연결이 끊어질 때 발생하는 LxFs readdir 문제가 수정되었습니다. [GH 2077]
- 연결되지 않은 drvfs 파일을 procfs를 통해 다시 열 수 있습니다.
- WSL 기능을 사용하지 않도록 설정하기 위한 글로벌 레지스트리 키 재정의가 추가 되었습니다(interop/드라이브 탑재).
- DrvFs(및 LxFs)에 대한 "통계"의 잘못된 블록 수를 수정했습니다. [GH 1894]
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 빌드 16232

빌드 16232에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 이 릴리스에는 WSL과 관련된 변경 내용이 없습니다.

## 빌드 16226

빌드 16226에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- xattr 관련 syscall 지원이 추가되었습니다(getxattr, setxattr, listxattr, removexattr).
- security.capability xattr 지원이 추가되었습니다.
- 비-MS SMB 서버를 비롯한 특정 파일 시스템 및 필터와의 호환성이 향상되었습니다. [GH #1952]
- OneDrive 자리 표시자, GVFS 자리 표시자 및 컴팩트 OS 압축 파일에 대한 지원이 향상되었습니다.
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 빌드 16215

빌드 16215에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- WSL에서 더 이상 개발자 모드를 요구하지 않습니다.
- drvfs에서 디렉터리 연결을 지원합니다.
- WSL 배포 appx 패키지의 제거를 처리합니다.
- 프라이빗 및 공유 매핑을 표시하도록 procfs를 업데이트합니다.
- 부분적으로 설치 또는 제거된 배포판을 정리하는 wslconfig.exe 기능을 추가합니다.
- TCP 소켓에 대한 IP\_MTU\_DISCOVER 지원이 추가되었습니다. [GH 1639, 2115, 2205]
- AF\_INADDR 경로에 대한 프로토콜 패밀리를 유추합니다.
- 직렬 디바이스가 개선되었습니다. [GH 1929]

## 빌드 16199

빌드 16199에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 이 릴리스에는 WSL과 관련된 변경 내용이 없습니다.

## 빌드 16193

빌드 16193에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- SIGCONT 전송과 threadgroup 종료 간의 경합 상태 [GH 1973]
- FILE\_DEVICE\_CONSOLE 대신 FILE\_DEVICE\_NAMED\_PIPE를 보고하도록 tty 및 pty 디바이스를 변경합니다. [GH 1840]
- IP\_OPTIONS에 대한 SSH 픽스
- DrvFs 탑재를 Init daemon으로 이동했습니다. [GH 1862, 1968, 1767, 1933]
- DrvFs에 다음 NT symlink에 대한 지원이 추가되었습니다.

## 빌드 16184

빌드 16184에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- apt 패키지 유지 관리 작업(lxrun.exe/update)을 제거했습니다.
- node.js의 Windows 프로세스에서 출력이 표시되지 않는 문제를 해결했습니다. [GH 1840]
- lxcore의 맞춤 요구 사항이 완화됩니다. [GH 1794]
- 여러 시스템 호출에서 AT\_EMPTY\_PATH 플래그 처리가 수정되었습니다.
- 열린 핸들로 DrvFs 파일을 삭제하면 파일이 정의되지 않은 동작을 보이는 이슈가 수정되었습니다. [GH 544,966,1357,1535,1615]
- 이제 /etc/hosts는 Windows 호스트 파일(%windir%\system32\drivers\etc\hosts)의 항목을 상속합니다. [GH 1495]

## 빌드 16179

빌드 16179에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 이번 주에는 WSL이 변경되지 않았습니다.

## 빌드 16176

빌드 16176에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- [직렬 지원을 사용합니다.](#)
- IP 소켓 옵션 IP\_OPTIONS가 추가되었습니다. [GH 1116]
- pwritev 함수를 구현했습니다(파일을 nginx/PHP-FPM으로 업로드하는 동안). [GH 1506]
- IP 소켓 옵션 IP\_MULTICAST\_IF & IPV6\_MULTICAST\_IF가 추가되었습니다. [GH 990]
- 소켓 옵션 IP\_MULTICAST\_LOOP & IPV6\_MULTICAST\_LOOP에 대한 지원 [GH 1678]
- 앱 노드, traceroute, dig, nslookup, host에 대한 IP(V6)\_MTU 소켓 옵션이 추가되었습니다.
- IP 소켓 옵션 IPV6\_UNICAST\_HOPS가 추가되었습니다.
- [파일 시스템이 향상되었습니다.](#)
  - UNC 경로 탐색 허용
  - drvfs에서 CDFS 지원 사용
  - drvfs의 네트워크 파일 시스템에 대한 권한을 올바르게 처리
  - drvfs에 원격 드라이브에 대한 지원 추가
  - drvfs에서 FAT 지원 사용
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

### LTP 결과

15042 이후에는 변경된 내용이 없습니다.

## 빌드 16170

빌드 16170에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

WSL 테스트에 대해 설명하는 새로운 [블로그 게시물을](#) 발표했습니다.

### 고정

- 소켓 옵션 IP\_ADD\_MEMBERSHIP & IPV6\_ADD\_MEMBERSHIP 지원 [GH 1678]
- PTRACE\_OLDSETOPTIONS에 대한 지원이 추가되었습니다. [GH 1692]
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과

15042 이후에는 변경된 내용이 없습니다.

## 빌드 15046에서 Windows 10 크리에이터스 업데이트로 변경

Windows 10 크리에이터스 업데이트에 포함하기로 계획된 WSL 수정 사항 또는 기능이 더 이상 없습니다. WSL 릴리스 정보는 그 다음 주요 Windows 업데이트를 대상으로 하는 추가 기능을 위해 향후 몇 주 이내에 다시 시작될 예정입니다. 빌드 15046 및 향후 참가자 릴리스에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 빌드 15042

빌드 15042에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- "..."로 끝나는 경로를 제거할 때 발생하는 교착 상태를 해결했습니다.
- 성공 시 FIONBIO에서 0을 반환하지 않는 이슈를 해결했습니다. [GH 1683]
- inet 데이터 그램 소켓의 길이가 0인 읽기와 관련된 이슈를 해결했습니다.
- drvfs inode lookup의 경합 상태 때문에 발생 가능한 교착 상태를 수정했습니다. [GH 1675]
- unix 소켓 보조 데이터에 대한 지원이 확장되었습니다(SCM\_CREDENTIALS 및 SCM\_RIGHTS). [GH 514, 613, 1326]
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 737

전달되지 않는 수(실패, 건너뜀 등): 255

# 빌드 15031

빌드 15031에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- 시간(2)이 우발적으로 오작동하는 버그를 수정했습니다.
- \*SIGPROCMASK syscall이 신호 마스크를 손상시킬 수 있는 이슈를 해결했습니다.
- 이제 WSL 프로세스 만들기 알림에서 전체 명령줄 길이를 반환합니다. [GH 1632]
- 이제 WSL은 GDB 정지에 대해 ptrace를 통해 스레드 종료를 보고합니다. [GH 1196]
- 많은 tmux IO 후에 ptys가 중단되는 버그를 수정했습니다. [GH 1358]
- 여러 시스템 호출(futex, semtimedop, ppoll, sigtimedwait, itimer, timer\_create)의 시간 제한 유효성 검사를 수정했습니다.
- eventfd EFD\_SEMAPHORE 지원이 추가되었습니다. [GH 452]
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 737

전달되지 않는 수(실패, 건너뜀 등...): 255

# 빌드 15025

빌드 15025에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- grep 2.27을 손상시킨 버그를 수정했습니다. [GH 1578]
- eventfd2 syscall에 대한 EFD\_SEMAPHORE 플래그를 구현했습니다. [GH 452]
- /proc/[pid]/net/ipv6\_route를 구현했습니다. [GH 1608]
- unix 스트림 소켓에 대한 신호 구동 IO 지원이 추가되었습니다. [GH 393, 68]
- F\_GETPIPE\_SZ 및 F\_SETPIPE\_SZ를 지원합니다. [GH 1012]
- recvmmmsg() syscall을 구현했습니다. [GH 1531]
- epoll\_wait()가 대기하지 않는 버그를 수정했습니다. [GH 1609]
- /proc/version\_signature를 구현했습니다.
- 두 파일 설명자가 같은 파이프를 참조하는 경우 이제 syscall에서 오류를 반환합니다.
- Unix 소켓에 대한 SO\_PEERCRED의 올바른 동작을 구현했습니다.
- tkill syscall 잘못된 매개 변수 처리를 수정했습니다.

- drvfs의 성능을 높이기 위한 변경 작업을 수행했습니다.
- Ruby IO 차단에 대한 사소한 수정 사항이 있습니다.
- recvmsg()에서 inet 소켓의 MSG\_DONTWAIT 플래그에 대해 EINVAL을 반환하는 문제를 수정했습니다. [GH 1296]
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 732  
전달되지 않는 수(실패, 건너뜀 등...): 255

## 빌드 15019

빌드 15019에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- htop 같은 도구에 대한 procfs에서 CPU 사용량을 잘못 보고하는 버그가 수정되었습니다. (GH 823, 945, 971)
- 기존 파일에서 O\_TRUNC를 사용하여 open()을 호출하면 이제 inotify가 IN\_MODIFY를 IN\_OPEN보다 먼저 생성합니다.
- postgres를 사용하도록 unix 소켓 getsockopt SO\_ERROR가 수정되었습니다. [GH 61, 1354]
- GO 언어에 대한 /proc/sys/net/core/somaxconn이 구현되었습니다.
- 이제 Apt-get 패키지 업데이트 백그라운드 작업이 보기에서 숨겨진 상태로 실행됩니다.
- ipv6 localhost의 범위를 지웁니다(Spring-Framework(Java) 오류).
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 714  
전달되지 않는 수(실패, 건너뜀 등...): 249

## 빌드 15014

빌드 15014에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- 이제 Ctrl+C가 의도한 대로 작동합니다.
- 이제 htop 및 ps auxw가 올바른 리소스 사용률을 표시합니다. (GH #516)
- 기본적으로 NT 예외를 신호로 변환합니다. (GH #513)
- 이제 공간이 부족하면 fallocate가 EINVAL 대신 ENOSPC와 함께 실패합니다. (GH #1571)
- /proc/sys/kernel/sem이 추가되었습니다.
- semop 및 semtimedop 시스템 호출이 구현되었습니다.
- IP\_RECVTOS & IPV6\_RECVTCLASS 소켓 옵션을 사용하여 nslookup 오류를 수정했습니다. (GH 69)
- 소켓 옵션 IP\_RECVTTL 및 IPV6\_RECVHOPLIMIT에 대한 지원이 추가되었습니다.
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 709

전달되지 않는 수(실패, 건너뜀 등...): 255

## Syscall 요약

총 Syscall: 384

총 구현: 235

총 스텝: 22

총 미구현: 127

## 빌드 15007

빌드 15007에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 알려진 이슈

- 콘솔에서 일부 Ctrl + <key> 입력을 인식하지 못하는 알려진 버그가 있습니다. 여기에는 일반적인 'c' 키 누름 역할을 하는 ctrl-c 명령이 포함됩니다.
  - 해결 방법: 대체 키를 Ctrl+C에 매핑합니다. 예를 들어 Ctrl+K를 Ctrl+C에 매핑하려면 `stty intr ^k`를 사용합니다. 이 매핑은 터미널 단위로 수행되며 bash가 시작될 때마다 수행해야 합니다. 사용자는 옵션을 살펴보고 `.bashrc`에 포함시킬 수 있습니다.

## 고정

- WSL을 실행하면 CPU 코어를 100% 사용하는 이슈를 해결했습니다.
- 소켓 옵션 IP\_PKTINFO, IPV6\_RECV\_PKTINFO가 이제 지원됩니다. (GH #851, 987)
- Ixcore에서 네트워크 인터페이스 실제 주소를 16바이트로 자릅니다. (GH #1452, 1414, 1343, 468, 308)
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 709

전달되지 않는 수(실패, 건너뜀 등...): 255

## 빌드 15002

빌드 15002에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 알려진 이슈

알려진 두 가지 이슈:

- 콘솔에서 일부 Ctrl + <key> 입력을 인식하지 못하는 알려진 버그가 있습니다. 여기에는 일반적인 'c' 키누름 역할을 하는 ctrl-c 명령이 포함됩니다.
  - 해결 방법: 대체 키를 Ctrl+C에 매핑합니다. 예를 들어 Ctrl+K를 Ctrl+C에 매핑하려면 `stty intr ^k`를 사용합니다. 이 매핑은 터미널 단위로 수행되며 bash가 시작될 때마다 수행해야 합니다. 사용자는 옵션을 살펴보고 `.bashrc`에 포함시킬 수 있습니다.
- WSL이 실행되는 동안 시스템 스레드에서 CPU 코어를 100% 사용합니다. 내부적으로 근본 원인을 찾아 해결했습니다.

## 고정

- 이제 모든 bash 세션을 동일한 권한 수준에서 만들어야 합니다. 다른 수준에서 세션을 시작하려고 하면 차단됩니다. 즉, 관리자 콘솔과 비관리자 콘솔을 동시에 실행할 수 없습니다. (GH #626)
- 다음 NETLINK\_ROUTE 메시지를 구현했습니다(Windows 관리자 필요).
  - RTM\_NEWADDR(`ip addr add` 지원)
  - RTM\_NEWRROUTE(`ip route add` 지원)

- RTM\_DELADDR(`ip addr del` 지원)
  - RTM\_DELROUTE(`ip route del` 지원)
- 업데이트 할 패키지의 예약된 작업 확인이 더 이상 요금제 연결에서 실행되지 않습니다. (GH #1371)
- 파일럿이 중단되는 오류(예: `bash -c "ls -alR /" | bash -c "cat"`)를 수정했습니다. (GH #1214)
- TCP\_KEEPCNT 소켓 옵션을 구현했습니다. (GH #843)
- IP\_MTU\_DISCOVER INET 소켓 옵션을 구현했습니다. (GH #720, 717, 170, 69)
- NT 경로 조회를 사용하여 init에서 NT 이진 파일을 실행하는 래거시 기능을 제거했습니다. (GH #1325)
- 그룹/기타 읽기 액세스(0644)를 허용하도록 /dev/kmsg 모드를 수정했습니다. (GH #1321)
- /proc/sys/kernel/random/uuid를 구현했습니다. (GH #1092)
- 프로세스 시작 시간이 2432년으로 표시되는 오류를 수정했습니다. (GH #974)
- 기본 TERM 환경 변수를 xterm-256color로 전환했습니다. (GH #1446)
- 프로세스 포크 중에 프로세스 커밋이 계산되는 방식이 수정되었습니다. (GH #1286)
- /proc/sys/vm/overcommit\_memory를 구현했습니다. (GH #1286)
- /proc/net/route 파일을 구현했습니다. (GH #69)
- 바로 가기 이름이 잘못 번역된 오류를 수정했습니다. (GH #696)
- 프로그램 헤더의 유효성을 잘못 검사하는 elf 구문 분석 논리는 PATH\_MAX보다 작거나 같아야 하도록 수정했습니다. (GH #1048)
- procfs, sysfs, cgroupfs 및 binfmtfs에 대한 statfs 콜백을 구현했습니다. (GH #1378)
- 닫히지 않는 AptPackageIndexUpdate 창을 수정했습니다. (GH #1184, GH #1193에서도 설명)
- ASLR 퍼스널리티 ADDR\_NO\_RANDOMIZE 지원이 추가되었습니다. (GH #1148, 1128)
- AV 중에 gdb 스택을 적절하게 추적하도록 PTRACE\_GETSIGINFO, SIGSEGV를 개선했습니다. (GH #875)
- patchelf 이진 파일에 대한 Elf 구문 분석이 더 이상 실패하지 않습니다. (GH #471)
- VPN DNS가 /etc/resolv.conf로 전파되었습니다. (GH #416, 1350)
- 보다 안정적인 데이터 전송을 위해 TCP 닫기가 향상되었습니다. (GH #610, 616, 1025, 1335)
- 이제 파일을 너무 많이 열었을 때 올바른 오류 코드가 반환됩니다(EMFILE). (GH #1126, 2090)
- 이제 Windows 감사 로그가 프로세스 만들기 감사에서 이미지 이름을 보고합니다.
- 이제 bash 창 내에서 bash.exe를 시작할 때 정상적으로 실패합니다.
- interop가 LxFs 아래의 작업 디렉터리(예: notepad.exe .bashrc)에 액세스할 수 없는 경우에 표시되는 오류 메시지가 추가되었습니다.
- WSL에서 Windows 경로가 잘리는 문제를 해결했습니다.
- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 690  
전달되지 않는 수(실패, 건너뜀 등...): 274

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

```
shmctl
shmget
shmdt
shmat
```

## 빌드 14986

빌드 14986에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Netlink 및 Pty IOCTL의 버그 검사를 수정했습니다.
- 이제 커널 버전은 Xenial과의 일관성 때문에 4.4.0-43을 보고합니다.
- 이제 입력이 'nul:'로 전달되면 Bash.exe가 시작됩니다. (GH #1259)
- 이제 스레드 ID가 procfs에서 올바르게 보고됩니다. (GH #967)
- 이제 inotify\_add\_watch()에서 IN\_UNMOUNT | IN\_Q\_OVERFLOW | IN\_IGNORED | IN\_ISDIR 플래그가 지원됩니다. (GH #1280)
- timer\_create 및 관련 시스템 호출이 구현되었습니다. 따라서 GHC가 지원됩니다. (GH #307)
- ping이 0.000ms 시간을 반환하는 이슈를 해결했습니다. (GH #1296)
- 파일을 너무 많이 열었을 때 올바른 오류 코드가 반환됩니다.
- 인터페이스의 하드웨어 주소가 32바이트인 경우(예: Teredo 인터페이스) 네트워크 인터페이스 데이터에 대한 Netlink 요청이 EINVAL과 함께 실패하는 WSL 이슈를 해결했습니다.
  - Linux "ip" 유ти리티에는 WSL에서 32바이트 하드웨어 주소를 보고하면 충돌하는 버그가 있습니다. 이것은 WSL이 아니라 "ip"의 버그입니다. "ip" 유ти리티는 하드

웨어 주소를 인쇄하는 데 사용되는 문자열 버퍼의 길이를 하드 코딩하는데, 이 버퍼가 너무 작아서 32바이트 하드웨어 주소를 인쇄할 수 없습니다.

- 그 외에도 여러 문제를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 669

전달되지 않는 수(실패, 건너뜀 등...): 258

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

```
timer_create  
timer_delete  
timer_gettime  
timer_settime
```

## 빌드 14971

빌드 14971에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Microsoft에서 통제할 수 없는 상황으로 인해 이 빌드에는 Linux용 Windows 하위 시스템에 대한 업데이트가 없습니다. 정기적으로 예약된 업데이트는 다음 릴리스에서 다시 시작됩니다.

## LTP 결과:

14965에서 변경되지 않음

통과한 테스트 수: 664

전달되지 않는 수(실패, 건너뜀 등...): 263

# 빌드 14965

빌드 14965에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Netlink 소켓 NETLINK\_ROUTE 프로토콜의 RTM\_GETLINK 및 RTM\_GETADDR에 대한 지원이 추가되었습니다. (GH #468)
  - 네트워크 열거에 ifconfig 및 ip 명령을 사용할 수 있습니다.
- /sbin은 이제 기본적으로 사용자의 경로에 있습니다.
- 이제 NT 사용자 경로가 기본적으로 WSL 경로에 추가됩니다. 즉, Linux 경로에 System32를 추가하지 않고 notepad.exe를 입력할 수 있습니다.
- /proc/sys/kernel/cap\_last\_cap에 대한 지원이 추가되었습니다.
- 현재 작업 디렉터리에 비 ansi 문자가 포함되어 있으면 이제 WSL에서 NT 이진 파일을 시작할 수 있습니다. (GH #1254)
- 연결되지 않은 unix 스트림 소켓에서 종료할 수 있습니다.
- PR\_GET\_PDEATHSIG에 대한 지원이 추가되었습니다.
- CLONE\_PARENT에 대한 지원이 추가되었습니다.
- 파일이 중단되는 오류(예: bash -c "ls -alR /" | bash -c "cat")를 수정했습니다. (GH #1214)
- 현재 터미널에 연결하라는 요청을 처리합니다.
- /proc/<pid>/oom\_score\_adj를 쓰기 가능으로 표시합니다.
- /sys/fs/cgroup 폴더를 추가합니다.
- sched\_setaffinity는 선호도 비트 마스크 수를 반환해야 합니다.
- 인터프리터 경로를 잘못 가정하는 ELF 유효성 검사 논리는 64자 미만이어야 하도록 수정했습니다. (GH #743)
- Open file 설명자는 콘솔 창을 열어 둘 수 있습니다. (GH #1187)
- rename()이 대상 이름에 후행 슬래시를 사용하여 실패하는 오류를 수정했습니다. (GH #1008)
- /proc/net/dev 파일을 구현했습니다.

- 타이머 해상도로 인한 0.000ms ping을 수정했습니다.
- /proc/self/environ을 구현했습니다. (GH #730)
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 664

전달되지 않는 수(실패, 건너뜀 등...): 263

## 빌드 14959

빌드 14959에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Windows에 대한 Pico 프로세스 알림이 개선되었습니다. 추가 정보는 [WSL 블로그](#)에서 찾을 수 있습니다.
- Windows 상호 운용성 덕분에 안정성이 향상되었습니다.
- EDP(엔터프라이즈 데이터 보호)를 사용하는 경우 bash.exe를 시작할 때 발생하는 0x80070057 오류를 해결했습니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 665

전달되지 않는 수(실패, 건너뜀 등...): 263

## 빌드 14955

빌드 14955에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Microsoft에서 통제할 수 없는 상황으로 인해 이 빌드에는 Linux용 Windows 하위 시스템에 대한 업데이트가 없습니다. 정기적으로 예약된 업데이트는 다음 릴리스에서 다시 시작됩니다.

## LTP 결과:

통과한 테스트 수: 665  
전달되지 않는 수(실패, 건너뜀 등...): 263

## 빌드 14951

빌드 14951에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 새 기능: Windows/Ubuntu 상호 운용성

이제 WSL 명령줄에서 직접 Windows 이진 파일을 호출할 수 있습니다. 덕분에 사용자는 이전에는 불가능하던 방식으로 Windows 환경 및 시스템과 상호 작용할 수 있게 되었습니다. 간단한 예제로, 이제 사용자가 다음 명령을 실행할 수 있습니다.

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
```

자세한 내용은 아래에서 찾을 수 있습니다.

- [Interop에 대한 WSL 팀 블로그](#)
- [WSL 파일 시스템 설명서](#)

## 고정

- 이제 모든 새 WSL 인스턴스에 대해 Ubuntu 16.04(Xenial)가 설치됩니다. 기존 14.04(Trusty) 인스턴스를 사용하는 사용자는 자동으로 업그레이드되지 않습니다.
- 이제 설치 중에 설정된 로캘이 표시됩니다.
- WSL 프로세스를 파일로 리디렉션하는 기능이 가끔 작동하지 않는 버그를 포함하여 터미널이 개선되었습니다.
- 콘솔 수명은 bash.exe 수명에 연결되어야 합니다.
- 콘솔 창 크기는 버퍼 크기가 아닌 가시 크기를 사용해야 합니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 665

전달되지 않는 수(실패, 건너뜀 등...): 263

## 빌드 14946

빌드 14946에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- NT 사용자 이름에 공백이나 따옴표가 포함된 사용자에 대한 WSL 사용자 계정을 만들 수 없는 이슈를 해결했습니다.
- 통계에서 디렉터리의 링크 수로 0을 반환하도록 VolFs 및 DrvFs를 변경합니다.
- IPV6\_MULTICAST\_HOPS 소켓 옵션을 지원합니다.
- tty당 단일 콘솔 I/O 루프로 제한합니다. 예: 다음 명령을 사용할 수 있습니다.
  - bash -c "echo data" | bash -c "ssh user@example.com 'cat > foo.txt'"
- 공백을 /proc/cpuinfo의 탭으로 바꿉니다. (GH #1115)
- 이제 DrvFs는 탑재된 Windows 볼륨과 일치하는 이름으로 mountinfo에 표시됩니다.
- 이제 /home 및 /root가 올바른 이름으로 mountinfo에 표시됩니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

### LTP 결과:

통과한 테스트 수: 665

전달되지 않는 수(실패, 건너뜀 등...): 263

## 빌드 14942

빌드 14942에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- SSH를 차단하는 "ATTEMPTED EXECUTE OF NOEXECUTE MEMORY" 네트워킹 충돌을 포함하여 여러 버그 검사가 수정되었습니다.

- 현재 DrvFs가 있는 Windows 애플리케이션에서 생성된 알림에 inotify가 지원됩니다.
- mongod에 대한 TCP\_KEEPIDLE 및 TCP\_KEEPINTVL이 구현되었습니다. (GH #695)
- pivot\_root 시스템 호출이 구현되었습니다.
- SO\_DONTROUTE에 대한 소켓 옵션이 구현되었습니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 665

전달되지 않는 수(실패, 건너뜀 등...): 263

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

pivot\_root

## 빌드 14936

빌드 14936에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

참고: 향후 릴리스에서 WSL은 Ubuntu 14.04(Trusty) 대신 Ubuntu 16.04(Xenial)를 설치할 것입니다. 이 변경 내용은 새 인스턴스를 설치하는(lxrun.exe /install 또는 bash.exe를 처음으로 실행) 참가자에게 적용됩니다. Trusty를 사용하는 기존 인스턴스는 자동으로 업그레이드되지 않습니다. 사용자는 do-release-upgrade 명령을 사용하여 Trusty 이미지를 Xenial로 업그레이드할 수 있습니다.

## 알려진 이슈

WSL에서 일부 소켓 구현에 이슈가 있습니다. 버그 검사는 "ATTEMPTED EXECUTE OF NOEXECUTE MEMORY" 오류와 함께 자신을 충돌로 나타냅니다. 이 이슈의 가장 일반적인 징후는 ssh를 사용할 때 발생하는 충돌입니다. 근본 원인은 내부 빌드에서 수정되었으며, 기회가 되는 대로 참가자에게 푸시할 예정입니다.

## 고정

- chroot 시스템 호출이 구현되었습니다.
- 현재 DrvFs의 Windows 애플리케이션에서 생성된 알림에 대한 지원을 포함하여 inotify가 개선되었습니다.
  - 수정 사항: Windows 애플리케이션에서 시작된 변경 내용에 대한 Inotify 지원은 현재 제공되지 않습니다.
- IPV6::<port n>에 대한 소켓 바인딩은 이제 IPV6\_V6ONLY(GH #68, #157, #393, #460, #674, #740, #982, #996)를 지원합니다.
- waitid systemcall에 대한 WNOWAIT 동작이 구현되었습니다. (GH #638)
- IP 소켓 옵션 IP\_HDRINCL 및 IP\_TTL에 대한 지원이 추가되었습니다.
- 길이가 0인 read()는 즉시 반환되어야 합니다. (GH #975).
- .tar 파일에 NULL 종결자가 포함되지 않은 파일 이름 및 파일 이름 접두사를 올바르게 처리합니다.
- /dev/null에 epoll이 지원됩니다.
- /dev/alarm 시간 원본이 수정되었습니다.
- 이제 Bash -c가 파일로 리디렉션할 수 있습니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## LTP 결과:

통과한 테스트 수: 664

전달되지 않는 수(실패, 건너뜀 등...): 264

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

chroot

## 빌드 14931

빌드 14931에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- Microsoft에서 통제할 수 없는 상황으로 인해 이 빌드에는 Linux용 Windows 하위 시스템에 대한 업데이트가 없습니다. 정기적으로 예약된 업데이트는 다음 릴리스에서 다시 시작됩니다.

## 빌드 14926

빌드 14926에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 이제 Ping은 관리자 권한이 없는 콘솔에서 작동합니다.
- 이제 Ping6가 지원되며, 관리자 권한이 없습니다.
- WSL을 통해 수정된 파일에 Inotify가 지원됩니다. (GH #216)
  - 지원되는 플래그:
    - inotify\_init1: LX\_O\_CLOEXEC, LX\_O\_NONBLOCK
    - inotify\_add\_watch 이벤트: 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 특성: LX\_IN\_DONT\_FOLLOW, LX\_IN\_EXCL\_UNLINK, LX\_IN\_MASK\_ADD, LX\_IN\_ONESHOT, LX\_IN\_ONLYDIR
    - 출력 읽기: LX\_IN\_ISDIR, LX\_IN\_IGNORED
  - 알려진 이슈: Windows 애플리케이션의 파일을 수정해도 이벤트가 생성되지 않습니다.
- 이제 Unix 소켓이 SCM\_CREDENTIALS를 지원합니다.

### LTP 결과:

통과한 테스트 수: 651

전달되지 않는 수(실패, 건너뜀 등...): 258

## 빌드 14915

빌드 14915에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- unix 데이터 그램 소켓용 Socketpair가 제공됩니다. (GH #262)
- SO\_REUSEADDR에 Unix 소켓이 지원됩니다.
- SO\_BROADCAST에 UNIX 소켓이 지원됩니다. (GH #568)
- SOCK\_SEQPACKET에 Unix 소켓이 지원됩니다. (GH #758, #546)

- unix 데이터 그램 전송, 수신 및 종료에 대한 지원이 추가되었습니다.
- 고정되지 않은 주소에 대한 잘못된 mmap 매개 변수 유효성 검사로 인한 버그 검사를 수정했습니다. (GH #847)
- 터미널 상태의 일시 중단/다시 시작을 지원합니다.
- 화면 유ти리티의 차단을 해제하는 TIOCPKT ioctl을 지원합니다. (GH #774)
  - 알려진 이슈: 기능 키가 작동하지 않습니다.
- 해제된 멤버 'ReaderReady'에 LxpTimerFdWorkerRoutine이 액세스할 수 있게 되는 원인인 TimerFd의 경합을 수정했습니다. (GH #814)
- futex, poll 및 clock\_nanosleep에 대한 다시 시작 가능한 시스템 호출을 지원합니다.
- 바인드 탑재 지원이 추가되었습니다.
- 탑재 네임스페이스 지원에 대한 공유가 해제되었습니다.
  - 알려진 이슈: `unshare(CLONE_NEWNS)`를 사용하여 새 탑재 네임스페이스를 만들 때 현재 작업 디렉터리는 이전 네임스페이스를 계속 가리킵니다.
- 추가적으로 기능이 개선되고 버그가 수정되었습니다.

## 빌드 14905

빌드 14905에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 다시 시작 가능한 시스템 호출이 이제 지원됩니다. (GH #349, GH #520).
- /로 끝나는 디렉터리에 대한 Symlink가 이제 작동합니다. (GH #650)
- /dev/random에 대한 RNDGETENTCNT ioctl이 구현되었습니다.
- /proc/[pid]/mounts, /proc/[pid]/mountinfo 및 /proc/[pid]/mountstats 파일이 구현되었습니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## 빌드 14901

Windows 10 1주년 업데이트 이후에 출시된 첫 번째 참가자 빌드입니다.

빌드 14901에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- 후행 슬래시 이슈를 수정했습니다.

- 이제 `$ mv a/c/ a/b/` 같은 명령이 작동합니다.
- 이제 설치 시 Ubuntu 로캘을 Windows 로캘로 설정해야 하는지 여부를 묻는 메시지가 표시됩니다.
- ns 폴더에 procfs가 지원됩니다.
- tmpfs, procfs 및 sysfs 파일 시스템에 탑재 및 탑재 해제가 추가되었습니다.
- mknod[at] 32비트 ABI 서명이 수정되었습니다.
- Unix 소켓이 디스패치 모델로 전환되었습니다.
- setsockopt를 사용하여 설정된 INET 소켓 수신 버퍼 크기를 준수해야 합니다.
- MSG\_CMSG\_CLOEXEC unix 소켓 수신 메시지 플래그가 구현되었습니다.
- Linux 프로세스 stdin/stdout 파일 리디렉션 (GH #2)
  - CMD에서 bash -c 명령을 파일할 수 있습니다. 예: >dir | bash -c "grep foo"
- 이제 여러 개의 페이지 파일이 있는 시스템에 Bash를 설치할 수 있습니다. (GH #538, #358).
- 기본 INET 소켓 버퍼 크기가 기본 Ubuntu 설정의 크기와 일치해야 합니다.
- xattr syscall을 listxattr에 맞춥니다.
- SIOCGIFCONF의 유효한 IPv4 주소가 있는 인터페이스만 반환합니다.
- ptrace에서 삽입할 때 신호 기본 작업을 수정합니다.
- /proc/sys/vm/min\_free\_kbytes가 구현되었습니다.
- sigreturn에서 컨텍스트를 복원할 때 머신 컨텍스트 레지스터 값을 사용합니다.
  - 이렇게 하면 일부 사용자의 java 및 javac가 중단되는 이슈를 해결할 수 있습니다.
- /proc/sys/kernel/hostname이 구현되었습니다.

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

`waitid`

`epoll_pwait`

## 빌드 14388에서 Windows 10 1주년 업데이트로 전환

빌드 14388에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- 8월 2일에 Windows 10 1주년 업데이트를 준비하도록 수정되었습니다.

- 1주년 업데이트의 WSL에 대한 자세한 내용은 [블로그](#)에서 찾을 수 있습니다.

## 빌드 14376

빌드 14376에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- apt-get이 중단되는 일부 인스턴스를 제거했습니다. (GH #493)
- 빈 탑재가 올바르게 처리되지 않는 이슈를 해결했습니다.
- ramdisks가 올바르게 탑재되지 않는 이슈를 해결했습니다.
- 플래그를 지원하도록 unix 소켓 수용이 변경되었습니다. (부분 GH #451)
- 일반적인 네트워크 관련 블루스크린이 해결되었습니다.
- /proc/[pid]/task에 액세스할 때 블루스크린이 발생하는 문제를 해결했습니다. (GH #523)
- 일부 pty 시나리오에서 CPU 사용률이 높은 문제를 해결했습니다. (GH #488, #504)
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## 빌드 14371

빌드 14371에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- ptrace를 사용할 때 SIGCHLD 및 wait()를 사용하여 타이밍 경합을 수정했습니다.
- 경로에 후행 /가 있을 때 발생하는 몇 가지 동작을 수정했습니다. (GH #432)
- 자식에 대한 열린 핸들로 인해 이름 바꾸기/연결 해제가 실패하는 이슈를 해결했습니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## 빌드 14366

빌드 14366에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- symlink를 통한 파일 만들기를 수정했습니다.
- Python에 대한 listxattr을 추가했습니다. (GH 385)
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## Syscall 지원

- 아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

listxattr

## 빌드 14361

빌드 14361에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- 이제 DrvFs는 Windows 기반의 Ubuntu에서 Bash로 실행될 때 대/소문자를 구분합니다.
  - 사용자는 /mnt/c 드라이브에서 case.txt 및 CASE.TXT를 입력할 수 있습니다.
  - 대/소문자 구분은 Windows 기반 Ubuntu의 Bash 내에서만 지원됩니다. Bash 외부에 있는 경우 NTFS가 파일을 올바르게 보고하지만, 예기치 않은 동작이 발생하여 Windows의 파일과 상호 작용할 수 있습니다.
  - 각 볼륨의 루트(/mnt/c)는 대/소문자를 구분하지 않습니다.
  - Windows에서 이러한 파일을 처리하는 방법에 대한 자세한 내용은 [여기](#)에서 확인할 수 있습니다.
- pty/tty 지원이 대폭 개선되었습니다. 이제 TMUX 같은 애플리케이션이 지원됩니다. (GH #40)
- 가끔 사용자 계정이 만들어지지 않는 이슈를 해결했습니다.
- 매우 긴 인수 목록을 허용하도록 명령줄 인수 구조를 최적화했습니다. (GH #153)
- 이제 DrvFs의 read\_only 파일을 삭제하고 chmod할 수 있습니다.
- 연결이 끊어질 때 터미널이 중단되는 일부 인스턴스를 수정했습니다. (GH #43)
- 이제 chmod 및 chown이 tty 디바이스에서 작동합니다.
- 0\.0.0.0 및 ::에 localhost로 연결할 수 있습니다. (GH #388)
- 이제 Sendmsg/recvmsg는 >1의 IO 벡터 길이를 처리합니다. (부분 GH #376)
- 이제 사용자는 자동 생성된 호스트 파일을 옵트아웃할 수 있습니다. (GH #398)

- 설치하는 동안 자동으로 Linux 로캘을 NT 로캘에 연결합니다. (GH #11)
- /proc/sys/vm/swappiness 파일이 추가되었습니다. (GH #306)
- 이제 strace가 올바르게 종료됩니다.
- /proc/self/fd를 통해 파일을 다시 열 수 있습니다. (GH #222)
- DrvFs에서 %LOCALAPPDATA%\lxss 아래에 디렉터리를 숨깁니다. (GH #270)
- bash.exe ~를 보다 효율적으로 처리합니다. 이제 "bash ~ -c ls" 같은 명령이 지원됩니다. (GH #467)
- 이제 종료하는 동안 소켓이 epoll 읽기를 사용할 수 있음을 알립니다. (GH #271).
- lxrun /uninstall이 파일 및 폴더 삭제를 보다 효율적으로 수행합니다.
- ps -f를 수정했습니다. (GH #246)
- xEmacs 같은 x11 앱에 대한 지원이 개선되었습니다. (GH #481)
- 기본 Ubuntu 설정과 일치하고 get\_rlimit syscall에 올바른 크기를 보고하도록 초기 스레드 스택 크기를 업데이트했습니다. (GH #172, #258)
- pico 프로세스 이미지 이름의 보고 기능을 개선했습니다(예: 감사).
- df 명령에 대한 /proc/mountinfo를 구현했습니다.
- 자식 이름에 대한 symlink 오류 코드를 수정했습니다. 그리고 ...
- 버그 및 기능이 추가로 개선되었습니다.

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

GETTIMEER

MKNODAT

RENAMEAT

SENDFILE

SENDFILE64

SYNC\_FILE\_RANGE

## 빌드 14352

빌드 14352에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

## 고정

- 대용량 파일이 올바르게 다운로드/생성되지 않는 이슈를 해결했습니다. 이제 npm 및 기타 시나리오의 차단을 해제해도 됩니다. (GH #3, GH #313)

- 소켓이 중단되는 일부 인스턴스를 제거했습니다.
- 일부 ptrace 오류를 수정했습니다.
- 255자보다 긴 파일 이름을 허용하는 WSL 이슈를 수정했습니다.
- 영어가 아닌 문자에 대한 지원이 개선되었습니다.
- 현재 Windows 표준 시간대 데이터를 추가하고 기본값으로 설정했습니다.
- 각 탑재 지점에 대한 고유한 디바이스 id(jre 픽스 – GH #49)
- ":" 및 ".."를 포함하는 경로에 대한 문제 해결
- Fifo 지원이 추가되었습니다. (GH #71)
- 네이티브 Ubuntu 형식과 일치하도록 resolv.conf 형식을 업데이트했습니다.
- 일부 procfs를 정리했습니다.
- 관리자 콘솔에 ping을 사용합니다. (GH #18)

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

FALLOCATE

EXECVE

LGETXATTR

FGETXATTR

## 빌드 14342

빌드 14342에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

VolFs 및 DriveFs에 대한 정보는 [WSL 블로그](#)에서 찾을 수 있습니다.

## 고정

- Windows 사용자의 사용자 이름에 유니코드 문자가 있을 때 발생하는 설치 이슈를 해결했습니다.
- 이제 FAQ의 apt-get update udev 해결 방법이 첫 번째 실행 시 기본적으로 제공됩니다.
- DriveFs(/mnt/<drive>) 디렉터리에서 symlink를 사용합니다.
- 이제 symlink가 DriveFs와 VolFs 간에 작동합니다.
- 최상위 경로 구문 분석 이슈를 해결했습니다. 이제 ls ./가 예상대로 작동합니다.
- 이제 DriveFs의 npm 설치와 -g 옵션이 작동합니다.

- PHP 서버가 시작되지 않는 이슈를 해결했습니다.
- 네이티브 Ubuntu와 보다 가깝게 일치하도록 \$PATH 같은 기본 환경 변수를 업데이트했습니다.
- apt 패키지 캐시를 업데이트하기 위한 주간 유지 관리 작업이 Windows에 추가되었습니다.
- ELF 헤더 유효성 검사와 관련된 이슈를 해결했습니다. 이제 WSL은 모든 Melkor 옵션을 지원합니다.
- Zsh 셸이 작동합니다.
- 미리 컴파일된 Go 이진 파일이 이제 지원됩니다.
- Bash.exe 첫 실행에 대한 메시지가 이제 올바르게 번역되었습니다.
- 이제 /proc/meminfo에서 올바른 정보를 반환합니다.
- 이제 VFS에서 소켓이 지원됩니다.
- 이제 /dev가 tmpfs로 탑재됩니다.
- 이제 Fifo가 지원됩니다.
- 이제 다중 코어 시스템이 /proc/cpuinfo에 올바르게 표시됩니다.
- 첫 번째 실행에서 다운로드와 관련된 기능이 개선되고 오류 메시지가 추가되었습니다.
- Syscall 및 버그 픽스가 개선되었습니다. 아래 syscall 목록이 지원됩니다.
- 그 외에도 여러 버그를 수정하고 기능을 개선했습니다.

## 알려진 문제

- `..` 확인하지 않음 올바르게 확인되지 않습니다.

## Syscall 지원

아래는 WSL에서 일부가 구현된 새 syscall 또는 향상된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

FCHOWNAT

GETEUID

GETGID

GETRESUID

GETXATTR

PTRACE

SETGID

SETGROUPS

SETHOSTNAME

## 빌드 14332

빌드 14332에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 고정

- DNS 항목 우선 순위 지정을 포함하여 resolv.conf 생성이 개선되었습니다.
- /mnt 드라이브와 비-/mnt 드라이브 간에 파일 및 디렉터리를 이동할 때 발생하는 이슈
- 이제 symlink를 사용하여 Tar 파일을 만들 수 있습니다.
- 인스턴스를 만들 때 기본 /run/lock 디렉터리가 추가됩니다.
- 적절한 통계 정보를 반환하도록 /dev/null이 업데이트되었습니다.
- 처음 실행하는 동안 다운로드할 때 추가 오류가 발생합니다.
- Syscall 및 버그 픽스가 개선되었습니다. 아래 syscall 목록이 지원됩니다.
- 버그 및 기능이 추가로 개선되었습니다.

### Syscall 지원

다음은 WSL에서 일부가 구현된 새 syscall입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

## 빌드 14328

빌드 14332에 대한 일반적인 Windows 정보는 [Windows 블로그](#)를 참조하세요.

### 새로운 기능

- 이제 Linux 사용자를 지원합니다. Windows 기반의 Ubuntu에 Bash를 설치하면 Linux 사용자를 만들라는 메시지가 표시됩니다. 자세한 내용은 <https://aka.ms/wslusers>에서 확인할 수 있습니다.
- 이제 호스트 이름이 Windows 컴퓨터 이름으로 설정됩니다. 더 이상 @localhost는 없습니다.
- 빌드 14328에 대한 자세한 내용은 <https://aka.ms/wip14328>에서 확인할 수 있습니다.

# 고정

- 비 `/mnt/<drive>` 파일에 대한 Symlink가 개선되었습니다.
  - 이제 npm 설치가 작동합니다.
  - 이제 [여기](#)에 제공된 지침에 따라 jdk/jre를 설치할 수 있습니다.
  - 알려진 이슈: symlink가 Windows 탑재에 대해 작동하지 않습니다. 이 기능은 이 후 빌드에서 사용할 수 있습니다.
- 이제 top 및 htop이 표시됩니다.
- 일부 설치 오류에 대한 오류 메시지가 추가되었습니다.
- Syscall 및 버그 픽스가 개선되었습니다. 아래 syscall 목록이 지원됩니다.
- 버그 및 기능이 추가로 개선되었습니다.

## Syscall 지원

아래는 WSL에서 일부가 구현된 syscall 목록입니다. 이 목록의 syscall은 하나 이상의 시나리오에서 지원되지만, 현재 지원되는 매개 변수 중 일부가 없을 수도 있습니다.

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  
SETHOOKNAME  
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

# Linux용 Windows 하위 시스템 커널의 릴리스 정보

아티클 • 2023. 03. 21. • 읽는 데 6분 걸림

전체 Linux 커널을 사용하는 ↗ WSL 2 배포판에 대한 지원이 추가되었습니다. 이 Linux 커널은 오픈 소스이며, 소스 코드는 [WSL2-Linux-Kernel](#) ↗ 리포지토리에 제공됩니다. 이 Linux 커널은 Microsoft 업데이트를 통해 머신에 제공되며, Windows 이미지의 일부로 제공되는 Linux용 Windows 하위 시스템에 대한 별도의 릴리스 일정을 따릅니다.

## 5.15.57.1

출시 날짜: 시험판 2022/08/02

공식 Github 릴리스 링크 ↗

- v5.15 커널 시리즈를 기반으로 하는 WSL2 커널의 초기 릴리스
- rolling-lts/wsl/5.15.57.1 릴리스
- 안정적인 커널 버전 v5.15.57로 업데이트
- x86\_64 빌드에서 Retbleed 완화 사용
- nftables 및 트래픽 제어 사용
- VGEM 드라이버 사용
- 마지막 v5.10 WSL2 커널 이후 9p 파일 시스템 회귀 수정
- PTP(Precision Time Protocol) 클록 디바이스에 대한 지원 사용
- Landlock LSM(Linux 보안 모듈) 사용
  - <https://landlock.io/> ↗
- 기타 CGroup(제어 그룹) 사용
  - <https://www.kernel.org/doc/html/v5.15/admin-guide/cgroup-v2.html#misc> ↗
- Ceph 분산 파일 시스템에 대한 지원 사용 안 함

## 5.10.102.1

출시 날짜: 시험판 2022/05/09

공식 Github 릴리스 링크 ↗

- rolling-lts/wsl/5.10.102.1 릴리스
- 안정적인 업스트림 커널 릴리스 5.10.102로 업데이트
- 기본적으로 권한 없는 BPF 사용 안 함
- kernel.unprivileged\_bpf\_disabled sysctl을 0으로 설정하여 다시 사용하도록 설정할 수 있습니다.

- Dxgkrnl 버전을 2216으로 업데이트
- ioctls[]에 대한 범위를 벗어난 배열 액세스 수정
- 호스트에 대한 동기 호출을 기다리는 프로세스를 종료할 수 있도록 동기화 VM 버스 메시지를 "killable"로 구현합니다.
- 게스트 프로세스가 종료되는 경우 디바이스를 플러시하여 게스트가 소멸될 때 교착 상태를 방지합니다.

## 5.10.93.2

출시 날짜: 시험판 2022/02/08

[공식 Github 릴리스 링크 ↗](#)

- rolling-lts/wsl/5.10.93.2 릴리스
- 안정적인 업스트림 커널 릴리스 5.10.93으로 업데이트
- CH341 및 CP210X USB 직렬 드라이버 사용
- pahole에 대한 드워프 종속성을 포함하도록 README.md 빌드 지침 수정
- Dxgkrnl 버전을 2111로 전환
- 기존 및 총 sysmem 할당 제한을 제거했습니다.
- 프로세스 정리 중 종료를 위해 디바이스를 올바르게 플러시합니다.
- d3dkmthk.h에 대한 SPDX-License-Identifier 수정

## 5.10.81.1

출시 날짜: 시험판 2022/02/01

[공식 Github 릴리스 링크 ↗](#)

- rolling-lts/wsl/5.10.81.1 릴리스
- 안정적인 업스트림 커널 릴리스 5.10.81로 업데이트
- arm64에서 누락된 옵션을 사용하도록 설정하여 커널 구성 통합
- 비 arch 특정 ACPI 옵션 사용
- 디바이스 매퍼 RAID와 관련된 옵션 사용
- Btrfs 사용
- LZO 및 ZSTD 압축 사용

## 5.10.74.3

출시 날짜: 시험판 2021/11/10

[공식 Github 릴리스 링크 ↗](#)

- rolling-lts/wsl/5.10.74.3 릴리스
- 안정적인 업스트림 커널 릴리스 5.10.74로 업데이트
- eBPF 도구(microsoft/WSL#7437)에서 사용할 BPF 형식(CONFIG\_DEBUG\_INFO\_BTF) 활성화
- Dxgkrnl 버전을 2110으로 업데이트함
- Dxgkrnl 사용을 위해 버퍼 공유 및 동기화 파일 프레임워크 (CONFIG\_DMA\_SHARED\_BUFFER, CONFIG\_SYNC\_FILE) 활성화
- 8.1 이전 버전의 GCC에서 Dxgkrnl 빌드 실패 수정(microsoft/WSL#7558)

## 5.10.60.1

릴리스 날짜: 2021/11/02(시험판 2021/10/05)

[공식 Github 릴리스 링크 ↗](#)

- rolling-lts/wsl/5.10.60.1 릴리스
- 안정적인 업스트림 커널 릴리스 5.10.60으로 업데이트
- PCI BAR 상대 주소 지원을 통해 virtio-pmem 사용
- arm64용 Hyper-V에서 vPCI 지원 사용
- io\_uring 지원 사용
- IP를 통해 USB 지원 사용
- x86\_64용 paravirtualized spinlock 지원 사용
- dxgkrnl 드라이버를 새로 고쳐 버그 수정 및 코드 정리 선택
- NFSv4.1용 NFS 클라이언트 지원 사용
- USB를 통해 Arduino와 상호 작용하기 위한 USB 커널 구성 옵션 사용
- WSL2 관련 README.md 제공

## 5.10.43.3

출시 날짜: 시험판 2021/07/12

[공식 Github 릴리스 링크 ↗](#)

- 버전 rolling-lts/wsl/5.10.43.3
- 업스트림 안정적인 커널 릴리스 5.10.43으로 업데이트
- 향상된 dxgkrnl 드라이버
- Hyper-V 시리즈(v9)에서 arm64 Linux의 새 수정 버전
- 모든 버전의 Windows에서 실행되도록 하기 위해 arm64 게스트에서 Hyper-V Hypercall 인터페이스 항상 사용

## 5.10.16.3

릴리스 날짜: 2021/07/20(시험판 2021/04/16)

[공식 Github 릴리스 링크 ↗](#)

- [GH 5324 ↗](#) 수정
- wsl --mount를 사용하여 LUKS 암호화 디스크에 대한 지원 추가

## 5.4.91

출시 날짜: 시험판 2021/02/22

[공식 Github 릴리스 링크 ↗](#)

## 5.4.72

출시 날짜: 2021/01/21

[공식 Github 릴리스 링크 ↗](#)

- 5\.4.72에 대한 구성 수정

## 5.4.51-microsoft-standard

출시 날짜: 시험판 - 2020/10/22

[공식 Github 릴리스 링크 ↗](#)입니다.

- 안정적인 5.4.51 릴리스

## 4.19.128-microsoft-standard

출시 날짜: 2020/09/15

[공식 Github 릴리스 링크 ↗](#)입니다.

- 안정적인 4.19.128 릴리스입니다.
- dxgkrnl 드라이버 IOCTL 메모리 손상 수정

## 4.19.121-microsoft-standard

출시 날짜: 시험판

[공식 Github 릴리스 링크 ↗](#)입니다.

- Drivers: hv: vmbus: hook up dxgkrnl
- GPU 컴퓨팅에 대한 지원 추가

## 4.19.104-microsoft-standard

출시 날짜: 2020/06/09

공식 [Github 릴리스 링크](#)입니다.

- 4.19.104에 대한 WSL 구성 업데이트

## 4.19.84-microsoft-standard

출시 날짜: 2019/12/11

공식 [Github 릴리스 링크](#)입니다.

- 안정적인 4.19.84 릴리스입니다.

# Microsoft Store의 Linux용 Windows 하위 시스템에 대한 릴리스 정보

아티클 • 2023. 03. 21. • 읽는 데 2분 걸림

Microsoft Store 내의 WSL<sup>↗</sup>에 대한 릴리스 정보는 Microsoft/WSL Github 리포지토리 릴리스 페이지<sup>↗</sup>에서 찾을 수 있습니다. 최신 업데이트는 해당 목록을 참조하세요.

## 알려진 문제:

- 세션 0에서 Linux용 Windows 하위 시스템을 시작하는 작업이 현재 작동하지 않습니다(예: ssh 연결에서).