Welcome to the Windows Terminal, Console and Command-Line repo

This repository contains the source code for:

- Windows Terminal
- Windows Terminal Preview
- The Windows console host (`conhost.exe`)
- Components shared between the two projects
- ColorTool
- Sample projects that show how to consume the Windows Console APIs

Related repositories include:

- Windows Terminal Documentation (Repo: Contribute to the docs)
- Console API Documentation
- Cascadia Code Font

Installing and running Windows Terminal

> *Note: Windows Terminal requires Windows 10 2004 (build 19041) or later*

Microsoft Store [Recommended]

Install the Windows Terminal from the Microsoft Store. This allows you to always be on the latest version when we release new builds with automatic upgrades.

This is our preferred method.

Other install methods

Via GitHub

For users who are unable to install Windows Terminal from the Microsoft Store, released builds can be manually downloaded from this repository's Releases page.

Download the `Microsoft.WindowsTerminal_<versionNumber>.msixbundle` file from the **Assets** section. To install the app, you can simply double-click on the `.msixbundle` file, and the app installer should automatically run. If that fails for any reason, you can try the following command at a PowerShell prompt:

```
# NOTE: If you are using PowerShell 7+, please run
# Import-Module Appx -UseWindowsPowerShell
# before using Add-AppxPackage.

Add-AppxPackage Microsoft.WindowsTerminal_<versionNumber>.msixbundle
```

> *Note: If you install Terminal manually:*
>
> - *Terminal will not auto-update when new builds are released so you will need to regularly install the latest Terminal release to receive all the latest fixes and improvements!*

Via Windows Package Manager CLI (aka winget)

winget users can download and install the latest Terminal release by installing the `Microsoft.WindowsTerminal` package:

```
winget install --id=Microsoft.WindowsTerminal -e
```

Via Chocolatey (unofficial)

Chocolatey users can download and install the latest Terminal release by installing the `microsoft-windows-terminal` package:

```
choco install microsoft-windows-terminal
```

To upgrade Windows Terminal using Chocolatey, run the following:

```
choco upgrade microsoft-windows-terminal
```

If you have any issues when installing/upgrading the package please go to the Windows Terminal package page and follow the Chocolatey triage process

Via Scoop (unofficial)

Scoop users can download and install the latest Terminal release by installing the `windows-terminal` package:

```
scoop bucket add extras
scoop install windows-terminal
```

To update Windows Terminal using Scoop, run the following:

```
scoop update windows-terminal
```

If you have any issues when installing/updating the package, please search for or report the same on the issues page of Scoop Extras bucket repository.

Windows Terminal 2.0 Roadmap

The plan for delivering Windows Terminal 2.0 is described here and will be updated as the project proceeds.

Project Build Status

Project	Build Status
Terminal	
ColorTool	

Terminal & Console Overview

Please take a few minutes to review the overview below before diving into the code:

Windows Terminal

Windows Terminal is a new, modern, feature-rich, productive terminal application for command-line users. It includes many of the features most frequently requested by the Windows command-line community including support for tabs, rich text, globalization, configurability, theming & styling, and more.

The Terminal will also need to meet our goals and measures to ensure it remains fast and efficient, and doesn't consume vast amounts of memory or power.

The Windows Console Host

The Windows Console host, `conhost.exe`, is Windows' original command-line user experience. It also hosts Windows' command-line infrastructure and the Windows Console API server, input engine, rendering engine, user preferences, etc. The console host code in this repository is the actual source from which the `conhost.exe` in Windows itself is built.

Since taking ownership of the Windows command-line in 2014, the team added several new features to the Console, including background transparency, line-based selection, support for ANSI / Virtual Terminal sequences, 24-bit color, a Pseudoconsole ("ConPTY"), and more.

However, because Windows Console's primary goal is to maintain backward compatibility, we have been unable to add many of the features the community (and the team) have been wanting for the last several years including tabs, unicode text, and emoji.

These limitations led us to create the new Windows Terminal.

> *You can read more about the evolution of the command-line in general, and the Windows command-line specifically in this accompanying series of blog posts on the Command-Line team's blog.*

Shared Components

While overhauling Windows Console, we modernized its codebase considerably, cleanly separating logical entities into modules and classes, introduced some key extensibility points, replaced several old, home-grown collections and containers with safer, more efficient STL containers, and made the code simpler and safer by using Microsoft's Windows Implementation Libraries - WIL.

This overhaul resulted in several of Console's key components being available for re-use in any terminal implementation on Windows. These components include a new DirectWrite-based text layout and rendering engine,

a text buffer capable of storing both UTF-16 and UTF-8, a VT parser/emitter, and more.

Creating the new Windows Terminal

When we started planning the new Windows Terminal application, we explored and evaluated several approaches and technology stacks. We ultimately decided that our goals would be best met by continuing our investment in our C++ codebase, which would allow us to reuse several of the aforementioned modernized components in both the existing Console and the new Terminal. Further, we realized that this would allow us to build much of the Terminal's core itself as a reusable UI control that others can incorporate into their own applications.

The result of this work is contained within this repo and delivered as the Windows Terminal application you can download from the Microsoft Store, or directly from this repo's releases.

Resources

For more information about Windows Terminal, you may find some of these resources useful and interesting:

- Command-Line Blog
- Command-Line Backgrounder Blog Series
- Windows Terminal Launch: Terminal "Sizzle Video"
- Windows Terminal Launch: Build 2019 Session
- Run As Radio: Show 645 - Windows Terminal with Richard Turner
- Azure Devops Podcast: Episode 54 - Kayla Cinnamon and Rich Turner on DevOps on the Windows Terminal
- Microsoft Ignite 2019 Session: The Modern Windows Command Line: Windows Terminal - BRK3321

FAQ

I built and ran the new Terminal, but it looks just like the old console

Cause: You're launching the incorrect solution in Visual Studio.

Solution: Make sure you're building & deploying the `CascadiaPackage` project in Visual Studio.

> ⚠ *Note:* `OpenConsole.exe` *is just a locally-built* `conhost.exe`*, the classic Windows Console that hosts Windows' command-line infrastructure. OpenConsole is used by Windows Terminal to connect to and communicate with command-line applications (via ConPty).*

Documentation

All project documentation is located at aka.ms/terminal-docs. If you would like to contribute to the documentation, please submit a pull request on the Windows Terminal Documentation repo.

Contributing

We are excited to work alongside you, our amazing community, to build and enhance Windows Terminal!

BEFORE you start work on a feature/fix, please read & follow our Contributor's Guide to help avoid any wasted or duplicate effort.

Communicating with the Team

The easiest way to communicate with the team is via GitHub issues.

Please file new issues, feature requests and suggestions, but **DO search for similar open/closed pre-existing issues before creating a new issue.**

If you would like to ask a question that you feel doesn't warrant an issue (yet), please reach out to us via Twitter:

- Kayla Cinnamon, Program Manager: @cinnamon_msft
- Dustin Howett, Engineering Lead: @dhowett
- Mike Griese, Senior Developer: @zadjii
- Carlos Zamora, Developer: @cazamor_msft
- Pankaj Bhojwani, Developer
- Leonard Hecker, Developer: @LeonardHecker

Developer Guidance

Prerequisites

- You must be running Windows 10 2004 (build >= 10.0.19041.0) or later to run Windows Terminal
- You must enable Developer Mode in the Windows Settings app to locally install and run Windows Terminal
- You must have PowerShell 7 or later installed
- You must have the Windows 11 (10.0.22000.0) SDK installed
- You must have at least VS 2019 installed
- You must install the following Workloads via the VS Installer. Note: Opening the solution in VS 2019 will prompt you to install missing components automatically:
  - Desktop Development with C++
  - Universal Windows Platform Development
  - **The following Individual Components**
    - C++ (v142) Universal Windows Platform Tools

Building the Code

This repository uses git submodules for some of its dependencies. To make sure submodules are restored or updated, be sure to run the following prior to building:

```
git submodule update --init --recursive
```

OpenConsole.sln may be built from within Visual Studio or from the command-line using a set of convenience scripts & tools in the **/tools** directory:

Building in PowerShell

```
Import-Module .\tools\OpenConsole.psm1
Set-MsBuildDevEnvironment
Invoke-OpenConsoleBuild
```

Building in Cmd

```
.\tools\razzle.cmd
bcz
```

Running & Debugging

To debug the Windows Terminal in VS, right click on `CascadiaPackage` (in the Solution Explorer) and go to properties. In the Debug menu, change "Application process" and "Background task process" to "Native Only".

You should then be able to build & debug the Terminal project by hitting `F5`. Make sure to select either the "x64" or the "x86" platform - the Terminal doesn't build for "Any Cpu" (because the Terminal is a C++ application, not a C# one).

> 👉 *You will not be able to launch the Terminal directly by running the WindowsTerminal.exe. For more details on why, see #926, #4043*

Coding Guidance

Please review these brief docs below about our coding practices.

> 👉 *If you find something missing from these docs, feel free to contribute to any of our documentation files anywhere in the repository (or write some new ones!)*

This is a work in progress as we learn what we'll need to provide people in order to be effective contributors to our project.

- Coding Style
- Code Organization
- Exceptions in our legacy codebase
- Helpful smart pointers and macros for interfacing with Windows in WIL

Code of Conduct

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.