

Terminal Contributor's Guide

Below is our guidance for how to report issues, propose new features, and submit contributions via Pull Requests (PRs).

Open Development Workflow

The Windows Terminal team is VERY active in this GitHub Repo. In fact, we live in it all day long and carry out all our development in the open!

When the team finds issues we file them in the repo. When we propose new ideas or think-up new features, we file new feature requests. When we work on fixes or features, we create branches and work on those improvements. And when PRs are reviewed, we review in public - including all the good, the bad, and the ugly parts.

The point of doing all this work in public is to ensure that we are holding ourselves to a high degree of transparency, and so that the community sees that we apply the same processes and hold ourselves to the same quality-bar as we do to community-submitted issues and PRs. We also want to make sure that we expose our team culture and "tribal knowledge" that is inherent in any closely-knit team, which often contains considerable value to those new to the project who are trying to figure out "why the heck does this thing look/work like this???"

Repo Bot

The team triages new issues several times a week. During triage, the team uses labels to categorize, manage, and drive the project workflow.

We employ [a bot engine](#) to help us automate common processes within our workflow.

We drive the bot by tagging issues with specific labels which cause the bot engine to close issues, merge branches, etc. This bot engine helps us keep the repo clean by automating the process of notifying appropriate parties if/when information/follow-up is needed, and closing stale issues/PRs after reminders have remained unanswered for several days.


Therefore, if you do file issues, or create PRs, please keep an eye on your GitHub notifications. If you do not respond to requests for information, your issues/PRs may be closed automatically.

Reporting Security Issues

Please do not report security vulnerabilities through public GitHub issues. Instead, please report them to the Microsoft Security Response Center (MSRC). See [SECURITY.md](#) for more information.

Before you start, file an issue

Please follow this simple rule to help us eliminate any unnecessary wasted effort & frustration, and ensure an efficient and effective use of everyone's time - yours, ours, and other community members':

 If you have a question, think you've discovered an issue, would like to propose a new feature, etc., then find/file an issue **BEFORE** starting work to fix/implement it.

Search existing issues first

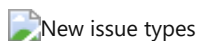
Before filing a new issue, search existing open and closed issues first: This project is moving fast! It is likely someone else has found the problem you're seeing, and someone may be working on or have already contributed a fix!

If no existing item describes your issue/feature, great - please file a new issue:

File a new Issue

- Don't know whether you're reporting an issue or requesting a feature? File an issue
- Have a question that you don't see answered in docs, videos, etc.? File an issue
- Want to know if we're planning on building a particular feature? File an issue
- Got a great idea for a new feature? File an issue/request/idea
- Don't understand how to do something? File an issue
- Found an existing issue that describes yours? Great - upvote and add additional commentary / info / repro-steps / etc.

When you hit "New Issue", select the type of issue closest to what you want to report/ask/request:



Complete the template

Complete the information requested in the issue template, providing as much information as possible. The more information you provide, the more likely your issue/ask will be understood and implemented. Helpful information includes:

- What device you're running (inc. CPU type, memory, disk, etc.)
- What build of Windows your device is running

👉 Tip: Run the following in PowerShell Core

```
C:\> $PSVersionTable.OS  
Microsoft Windows 10.0.18909
```

... or in Windows PowerShell

```
C:\> $PSVersionTable.BuildVersion  
  
Major  Minor  Build  Revision  
-----  
10     0      18912  1001
```

... or Cmd:

```
C:\> ver  
  
Microsoft Windows [Version 10.0.18900.1001]
```

- What tools and apps you're using (e.g. VS 2019, VSCode, etc.)
- Don't assume we're experts in setting up YOUR environment and don't assume we are experts in `<your distro/tool of choice>`. Teach us to help you!
- **We LOVE detailed repro steps!** What steps do we need to take to reproduce the issue? Assume we love to read repro steps. As much detail as you can stand is probably *barely* enough detail for us!

- If you're reporting a particular character/glyph not rendering correctly, the specific Unicode codepoint would be MOST welcome (e.g. U+1F4AF, U+4382)
- Prefer error message text where possible or screenshots of errors if text cannot be captured
- We MUCH prefer text command-line script than screenshots of command-line script.
- **If you intend to implement the fix/feature yourself then say so!** If you do not indicate otherwise we will assume that the issue is ours to solve, or may label the issue as `Help-Wanted`.

DO NOT post "+1" comments

⚠ DO NOT post "+1", "me too", or similar comments - they just add noise to an issue.

If you don't have any additional info/context to add but would like to indicate that you're affected by the issue, upvote the original issue by clicking its [+ 😊] button and hitting 👍 (+1) icon. This way we can actually measure how impactful an issue is.

Contributing fixes / features

If you're able & willing to help fix issues and/or implement features, we'd love your contribution!

The best place to start is the list of ["Easy Starter"](#) issues. These are bugs or tasks that we on the team believe would be easier to implement for someone without any prior experience in the codebase. Once you're feeling more comfortable in the codebase, feel free to just use the ["Help Wanted"](#) label, or just find an issue you're interested in and hop in!

Generally, we categorize issues in the following way, which is largely derived from our old internal work tracking system:

- ["Bugs"](#) are parts of the Terminal & Console that are not quite working the right way. There's code to already support some scenario, but it's not quite working right. Fixing these is generally a matter of debugging the broken functionality and fixing the wrong code.
- ["Tasks"](#) are usually new pieces of functionality that aren't yet implemented for the Terminal/Console. These are usually smaller features, which we believe
 - could be a single, atomic PR
 - Don't require much design consideration, or we've already written the spec for the larger feature they belong to.
- ["Features"](#) are larger pieces of new functionality. These are usually things we believe would require larger discussion of how they should be implemented, or they'll require some complicated new settings. They might just be features that are composed of many individual tasks. Often times, with features, we like to have a spec written before development work is started, to make sure we're all on the same page (see below).

Bugs and tasks are obviously the easiest to get started with, but don't feel afraid of features either! We've had some community members contribute some amazing "feature"-level work to the Terminal (albeit, with lots of discussion 😊).

Often, we like to assign issues that generally belong to somebody's area of expertise to the team member that owns that area. This doesn't mean the community can't jump in -- they should reach out and have a chat with the assignee to see if it'd okay to take. If an issue's been assigned more than a month ago, there's a good chance it's fair game to try yourself.

To Spec or not to Spec

Some issues/features may be quick and simple to describe and understand. For such scenarios, once a team member has agreed with your approach, skip ahead to the section headed "Fork, Branch, and Create your PR", below.

Small issues that do not require a spec will be labelled `Issue-Bug` or `Issue-Task`.

However, some issues/features will require careful thought & formal design before implementation. For these scenarios, we'll request that a spec is written and the associated issue will be labeled `Issue-Feature`. More often than not, we'll add such features to the ["Specification Tracker" project](#).

Specs help collaborators discuss different approaches to solve a problem, describe how the feature will behave, how the feature will impact the user, what happens if something goes wrong, etc. Driving towards agreement in a spec, before any code is written, often results in simpler code, and less wasted effort in the long run.

Specs will be managed in a very similar manner as code contributions so please follow the ["Fork, Branch and Create your PR"](#) section below.

Writing / Contributing-to a Spec

To write/contribute to a spec: fork, branch and commit via PRs, as you would with any code changes.

Specs are written in markdown, stored under the `\doc\specs` folder and named `[issue id] - [spec description].md`.

👉 **It is important to follow the spec templates and complete the requested information.** The available spec templates will help ensure that specs contain the minimum information & decisions necessary to permit development to begin. In particular, specs require you to confirm that you've already discussed the issue/idea with the team in an issue and that you provide the issue ID for reference.

Team members will be happy to help review specs and guide them to completion.

Help Wanted

Once the team has approved an issue/spec, development can proceed. If no developers are immediately available, the spec can be parked ready for a developer to get started. Parked specs' issues will be labeled "Help Wanted". To find a list of development opportunities waiting for developer involvement, visit the Issues and filter on [the Help-Wanted label](#).

Development

Fork, Clone, Branch and Create your PR

Once you've discussed your proposed feature/fix/etc. with a team member, and you've agreed an approach or a spec has been written and approved, it's time to start development:

1. Fork the repo if you haven't already
2. Clone your fork locally
3. Create & push a feature branch
4. Create a [Draft Pull Request \(PR\)](#)
5. Work on your changes
6. Build and see if it works. Consult [How to build OpenConsole](#) if you have problems.

Testing

Testing is a key component in the development workflow. Both Windows Terminal and Windows Console use TAEF(the Test Authoring and Execution Framework) as the main framework for testing.

If your changes affect existing test cases, or you're working on brand new features and also the accompanying test cases, see [TAEF](#) for more information about how to validate your work locally.

Code Review

When you'd like the team to take a look, (even if the work is not yet fully-complete), mark the PR as 'Ready For Review' so that the team can review your work and provide comments, suggestions, and request changes. It may take several cycles, but the end result will be solid, testable, conformant code that is safe for us to merge.

*⚠ Remember: **changes you make may affect both Windows Terminal and Windows Console and may end up being re-incorporated into Windows itself!** Because of this, we will treat community PR's with the same level of scrutiny and rigor as commits submitted to the official Windows source by team members and partners.*

Merge

Once your code has been reviewed and approved by the requisite number of team members, it will be merged into the main branch. Once merged, your PR will be automatically closed.

Thank you

Thank you in advance for your contribution! Now, [what's next on the list?](#) 🤔