

# Contributing to Visual Studio Code

There are many ways to contribute to the Visual Studio Code project: logging bugs, submitting pull requests, reporting issues, and creating suggestions.

After cloning and building the repo, check out the [issues list](#). Issues labeled [help wanted](#) are good issues to submit a PR for. Issues labeled [good first issue](#) are great candidates to pick up if you are in the code for the first time. If you are contributing significant changes, or if the issue is already assigned to a specific month milestone, please discuss with the assignee of the issue first before starting to work on the issue.

## Prerequisites

In order to download necessary tools, clone the repository, and install dependencies via `yarn`, you need network access.

You'll need the following tools:

- [Git](#)
- [Node.JS](#), **x64**, version `>=14.17.4` and `<17`
- [Yarn](#), follow the [installation guide](#)
- [Python](#) (required for node-gyp; check the [node-gyp readme](#) for the currently supported Python versions)
  - **Note:** Python will be automatically installed for Windows users through installing `windows-build-tools` npm module (see below)
- A C/C++ compiler tool chain for your platform:
  - **Windows 10/11**
    - Install the Windows Build Tools:
      - if you install Node on your system using the Node installer from the [Node.JS](#) page then ensure that you have installed the 'Tools for Native Modules'. Everything should work out of the box then.
      - if you use a node version manager like [nvm](#) or [nvs](#) then follow these steps:
        - Install the current version of Python using the [Microsoft Store Package](#).
        - Install the Visual C++ Build Environment by either installing the [Visual Studio Build Tools](#) or the [Visual Studio Community Edition](#). The minimum workload to install is 'Desktop Development with C++'.
        - open a command prompt and run `npm config set msvs_version 2019`
    - **Warning:** Make sure your profile path only contains ASCII letters, e.g. *John*, otherwise, it can lead to [node-gyp usage problems \(nodejs/node-gyp/issues#297\)](#).
    - **Note:** Building and debugging via the Windows subsystem for Linux (WSL) is currently not supported.
  - **Windows WSL2:** <https://github.com/microsoft/vscode/wiki/Selfhosting-on-Windows-WSL>
  - **macOS**
    - [Xcode](#) and the Command Line Tools, which will install `gcc` and the related toolchain containing `make`
      - Run `xcode-select --install` to install the Command Line Tools
  - **Linux**
    - On Debian-based Linux: `sudo apt-get install build-essential g++ libx11-dev libxkbfile-dev libsecret-1-dev python-is-python3`

- On Red Hat-based Linux: `sudo yum groupinstall "Development Tools" &&`  
`sudo yum install libX11-devel.x86_64 libxkbfile-devel.x86_64`  
`libsecret-devel # or .i686 .`
- Others:
  - `make`
  - [pkg-config](#)
  - [GCC](#) or another compile toolchain
  - Dependencies:
- Building deb and rpm packages requires `fakeroot` and `rpm`; run: `sudo apt-get`  
`install fakeroot rpm`

## Troubleshooting

In case of issues, try deleting the contents of `~/.node-gyp` (alternatively `~/.cache/node-gyp` for Linux, `~/Library/Caches/node-gyp/` for macOS, or `%USERPROFILE%\AppData\Local\node-gyp` for Windows) first and then run `yarn cache clean` and then try again.

*If you are on Windows or Linux 64 bit systems and would like to compile to 32 bit, you'll need to set the `npm_config_arch` environment variable to `ia32` before running `yarn`. This will compile all native node modules for a 32 bit architecture. Similarly, when cross-compiling for ARM, set `npm_config_arch` to `arm`.*

**Note:** For more information on how to install NPM modules globally on UNIX systems without resorting to `sudo`, refer to [this guide](#).

*If you have Visual Studio 2019 installed, you may face issues when using the default version of node-gyp. If you have Visual Studio 2019 installed, you may need to follow the solutions [here](#).*

## Development container

Alternatively, you can avoid local dependency installation as this repository includes a Visual Studio Code Remote - Containers / Codespaces [development container](#).

- For [Remote - Containers](#), use the **Remote-Containers: Open Repository in Container...** command which creates a Docker volume for better disk I/O on macOS and Windows.
- For Codespaces, install the [Visual Studio Codespaces](#) extension in VS Code, and use the **Codespaces: Create New Codespace** command.

Docker / the Codespace should have at least **4 Cores and 6 GB of RAM (8 GB recommended)** to run the full build. See the [development container README](#) for more information.

If you'd like to contribute to the list of available development containers in the Remote - Containers extension, you can check out the [Contributing documentation](#) in the `vscode-dev-containers` repo.

## Enable Commit Signing

If you're a community member, feel free to jump over this step.

Otherwise, if you're a member of the VS Code team, follow the [Commit Signing](#) guide.

## Build and Run

If you want to understand how VS Code works or want to debug an issue, you'll want to get the source, build it, and run the tool locally.

*NOTE: If you need to debug the 32bit version of VS Code on 64bit Windows, follow [the guide on how to do that](#).*

## Getting the sources

First, fork the VS Code repository so that you can make a pull request. Then, clone your fork locally:

```
git clone https://github.com/<<<your-github-account>>>/vscode.git
```

Occasionally you will want to merge changes in the upstream repository (the official code repo) with your fork.

```
cd vscode
git checkout main
git pull https://github.com/microsoft/vscode.git main
```

Manage any merge conflicts, commit them, and then push them to your fork.

**Note:** The `microsoft/vscode` repository contains a collection of GitHub Actions that help us with triaging issues.

As you probably don't want these running on your fork, you can disable Actions for your fork via

```
https://github.com/<<Your Username>>/vscode/settings/actions .
```

## Build

Install and build all of the dependencies using `Yarn` :

```
cd vscode
yarn
```

Then you have two options:

- If you want to build from inside VS Code, you can open the `vscode` folder and start the build task with `Ctrl+Shift+B` (`CMD+Shift+B` on macOS). The build task will stay running in the background even if you close VS Code. If you happen to close VS Code and open it again, just resume the build by pressing `Ctrl+Shift+B` (`CMD+Shift+B`) again. You can kill it by running the `Kill Build VS Code` task or pressing `Ctrl+D` in the task terminal.
- If you want to build from a terminal, run `yarn watch` . This will run both the core watch task and watch-extension tasks in a single terminal.

The incremental builder will do an initial full build and will display a message that includes the phrase "Finished compilation" once the initial build is complete. The builder will watch for file changes and compile those changes incrementally, giving you a fast, iterative coding experience.

### Troubleshooting:

- **Windows:** If you have installed Visual Studio 2017 as your build tool, you need to open **x64 Native Tools Command Prompt for VS 2017**. Do not confuse it with *VS2015 x64 Native Tools Command Prompt*, if installed.
- **Linux:** You may hit a `ENOSPC` error when running the build. To get around this follow instructions in the [Common Questions](#).

### Errors and Warnings

Errors and warnings will show in the console while developing VS Code. If you use VS Code to develop VS Code, errors and warnings are shown in the status bar at the bottom left of the editor. You can view the error list using

```
View | Errors and Warnings
```

 or pressing `Ctrl+P` and then `!` (`CMD+P` and `!` on macOS).

👉 **Tip!** You don't need to stop and restart the development version of VS Code after each change. You can just execute `Reload Window` from the command palette. We like to assign the keyboard shortcut `Ctrl+R` (`Cmd+R` on macOS) to this command.

## Run

To test the changes, you launch a development version of VS Code on the workspace `vscode`, which you are currently editing.

## Desktop

Running on Electron with extensions run in NodeJS:

### macOS and Linux

```
./scripts/code.sh
./scripts/code-cli.sh # for running CLI commands (eg --version)
```

### Windows

```
.\scripts\code.bat
.\scripts\code-cli.bat
```

👉 **Tip!** If you receive an error stating that the app is not a valid Electron app, it probably means you didn't run `yarn watch` first.

## VS Code for the Web

Extensions and UI run in the browser.

👉 Besides `yarn watch` also run `yarn watch-web` to build the web bits for the built-in extensions.

### macOS and Linux

```
./scripts/code-web.sh
```

### Windows

```
.\scripts\code-web.bat
```

## Code Server Web

UI in the browser, extensions run in code server (NodeJS):

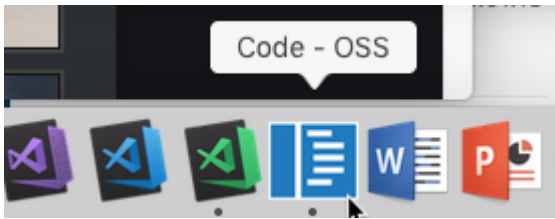
### macOS and Linux

```
./scripts/code-server.sh --launch
```

### Windows

```
.\scripts\code-server.bat --launch
```

You can identify the development version of VS Code ("Code - OSS") by the following icon in the Dock or Taskbar:



## Debugging

VS Code has a multi-process architecture and your code is executed in different processes.

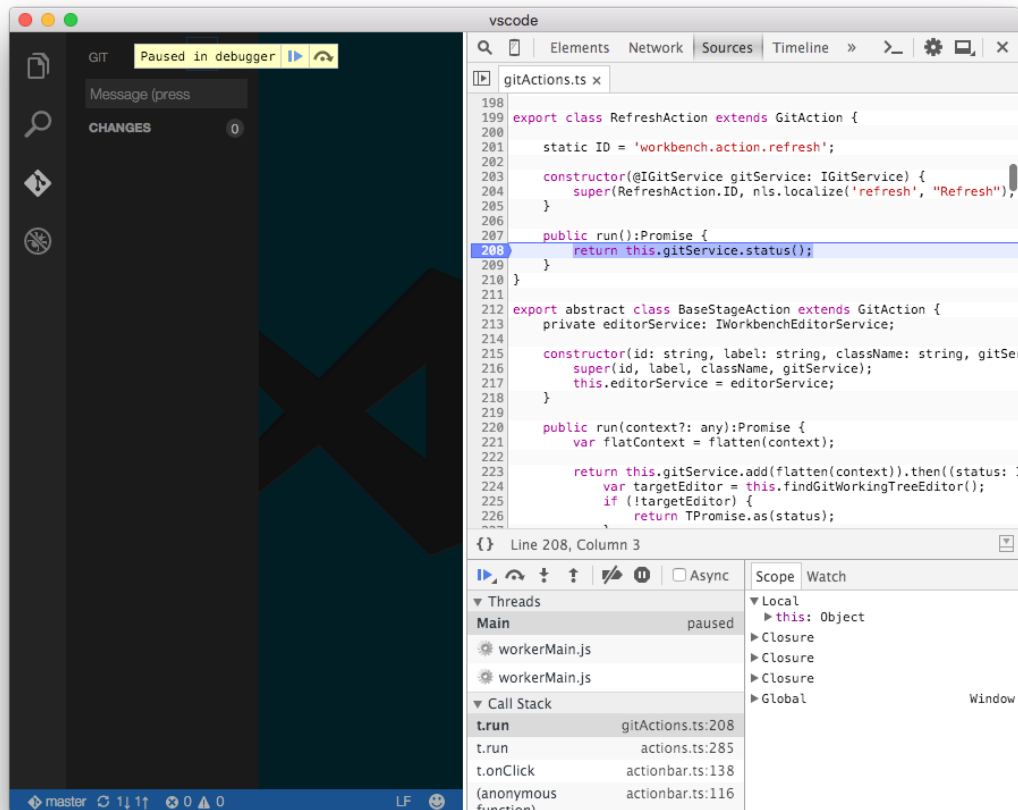
The **render** process runs the UI code inside the Shell window. To debug code running in the **render** you can either use VS Code or the Chrome Developer Tools.

### Using VS Code

- Open the `vscode` repository folder
- Choose the `Launch VS Code` launch configuration from the launch dropdown in the Debug viewlet and press `F5`.

### Using the Chrome Developer Tools

- Run the `Developer: Toggle Developer Tools` command from the Command Palette in your development instance of VS Code to launch the Chrome tools.
- It's also possible to debug the released versions of VS Code, since the sources link to sourcemaps hosted online.



The **extension host** process runs code implemented by a plugin. To debug extensions (including those packaged with VS Code) which run in the extension host process, you can use VS Code itself. Switch to the Debug viewlet, choose the `Attach to Extension Host` configuration, and press `F5`.

The **search** process can be debugged, but must first be started. Before attempting to attach, start a search by pressing `Ctrl+P` (`CMD+P` on macOS), otherwise, attaching will fail and time out.

## Automated Testing

Run the unit tests directly from a terminal by running `./scripts/test.sh` from the `vscode` folder ( `scripts\test` on Windows). The [test README](#) has complete details on how to run and debug tests, as well as how to produce coverage reports.

We also have automated UI tests. The [smoke test README](#) has all the details.

## Unit Testing

Run the tests directly from a terminal by running `./scripts/test.sh` from the `vscode` folder ( `scripts\test` on Windows). The [test README](#) has complete details on how to run and debug tests, as well as how to produce coverage reports.

## Linting

We use [eslint](#) for linting our sources. You can run eslint across the sources by calling `yarn eslint` from a terminal or command prompt. You can also run `yarn eslint` as a VS Code task by pressing `Ctrl+P` (`CMD+P` on macOS) and entering `task eslint`.

To lint the source as you make changes you can install the [eslint extension](#).

## Extensions

The Visual Studio Marketplace is not available from the `vscode` open source builds. If you need to use or debug an extension you can check to see if the extension author publishes builds in their repository (check the `Builds` page) or if it is open source you can clone and build the extension locally. Once you have the `.VSIX`, you can "side load" the extension either through the command line or using **Install from VSIX** command in the Extensions View command drop-down ([see more](#) on command line extension management).

## Work Branches

Even if you have push rights on the Microsoft/vscode repository, you should create a personal fork and create feature branches there when you need them. This keeps the main repository clean and your personal workflow cruft out of sight.

## Pull Requests

Before we can accept a pull request from you, you'll need to sign a [[Contributor License Agreement (CLA)|Contributor-License-Agreement]]. It is an automated process and you only need to do it once.

To enable us to quickly review and accept your pull requests, always create one pull request per issue and [link the issue in the pull request](#). Never merge multiple requests in one unless they have the same root cause. Be sure to follow our [[Coding Guidelines|Coding-Guidelines]] and keep code changes as small as possible. Avoid pure formatting changes to code that has not been modified otherwise. Pull requests should contain tests whenever possible.

## Introducing usage of new Electron API with a PR

A pull request that depends on Electron API that VS Code is currently not using comes with a certain risk and may be rejected. Whenever we update Electron, there is a chance that less popular Electron APIs break and it is very hard to find out upfront. Once a PR lands in VS Code, the role of maintaining the feature moves to the team and as such we have to follow up with upstream components to ensure the feature is still supported. As such, as a rule of thumb:

- avoid Electron APIs and use web standards instead (this also ensures that your feature is supported in our web client)
- if you must use Electron APIs, we require a unit test at <https://github.com/electron/electron> so that we protect against future breakage.

## Where to Contribute

Check out the [full issues list](#) for a list of all potential areas for contributions. Note that just because an issue exists in the repository does not mean we will accept a contribution to the core editor for it. There are several reasons we may not accept a pull request like:

- Performance - One of Visual Studio Code's core values is to deliver a *lightweight* code editor, that means it should perform well in both real and perceived performance.
- User experience - Since we want to deliver a *lightweight* code editor, the UX should feel lightweight as well and not be cluttered. Most changes to the UI should go through the issue owner and/or the UX team.

- Architectural - The team and/or feature owner needs to agree with any architectural impact a change may make. Things like new extension APIs *must* be discussed with and agreed upon by the feature owner.

To improve the chances to get a pull request merged you should select an issue that is labelled with the [help-wanted](#) or [bug](#) labels. If the issue you want to work on is not labelled with `help-wanted` or `bug`, you can start a conversation with the issue owner asking whether an external contribution will be considered.

To avoid multiple pull requests resolving the same issue, let others know you are working on it by saying so in a comment.

## Spell check errors

Pull requests that fix spell check errors in **translatable strings** (strings in `nls.localize(...)` calls) are welcomed but please make sure it doesn't touch multiple [feature areas](#), otherwise it will be difficult to review. Pull requests only fixing spell check errors in source code are **not** recommended.

## Packaging

VS Code can be packaged for the following platforms: `win32-ia32` | `win32-x64` | `darwin-x64` | `darwin-arm64` | `linux-ia32` | `linux-x64` | `linux-arm`

These `gulp` tasks are available:

- `vscode-[platform]` : Builds a packaged version for `[platform]` .
- `vscode-[platform]-min` : Builds a packaged and minified version for `[platform]` .

👉 **Tip!** Run `gulp` via `yarn` to avoid potential out of memory issues, for example `yarn gulp vscode-linux-x64`

See also: [Cross-Compiling for Debian-based Linux](#)

## Suggestions

We're also interested in your feedback for the future of VS Code. You can submit a suggestion or feature request through the issue tracker. To make this process more effective, we're asking that these include more information to help define them more clearly.

## Translations

We accept feedback on translations in language packs via GitHub issues in our [localization repo](#) that contains our currently supported language packs.

## Discussion Etiquette

In order to keep the conversation clear and transparent, please limit discussion to English and keep things on topic with the issue. Be considerate to others and try to be courteous and professional at all times.