

TOPICS Tips and Tricks

(https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/remote/troubleshooting.md)

Remote Development Tips and Tricks

This article covers troubleshooting tips and tricks for each of the Visual Studio Code Remote Development (<https://aka.ms/vscode-remote/download/extension>) extensions. See the SSH (</docs/remote/ssh>), Containers (</docs/remote/containers>), and WSL (</docs/remote/wsl>) articles for details on setting up and working with each specific extension. Or try the introductory Tutorials (</docs/remote/ssh-tutorial>) to help get you running quickly in a remote environment.

For tips and questions about GitHub Codespaces (<https://github.com/features/codespaces>), see the GitHub Codespaces documentation (<https://docs.github.com/github/developing-online-with-codespaces>).

SSH tips

SSH is powerful and flexible, but this also adds some setup complexity. This section includes some tips and tricks for getting the Remote - SSH extension up and running in different environments.

Configuring key based authentication

SSH public key authentication (<https://www.ssh.com/ssh/public-key-authentication>) is a convenient, high security authentication method that combines a local "private" key with a "public" key that you associate with your user account on an SSH host. This section will walk you through how to generate these keys and add them to a host.

Tip: PuTTY for Windows is not a supported client, but you can convert your PuTTYGen keys.

Quick start: Using SSH keys

To set up SSH key based authentication for your remote host. First we'll create a key pair and then copy the public key to the host.

Create your local SSH key pair

Check to see if you already have an SSH key on your **local** machine. This is typically located at `~/.ssh/id_ed25519.pub` on macOS / Linux, and the `.ssh` directory in your user profile folder on Windows (for example `C:\Users\your-user\.ssh\id_ed25519.pub`).

If you do not have a key, run the following command in a **local** terminal / PowerShell to generate an SSH key pair:

```
ssh-keygen -t rsa -b 4096
```

Tip: Don't have `ssh-keygen` ? Install a supported SSH client.

Authorize your macOS or Linux machine to connect

Run one of the following commands, in a **local terminal window** replacing user and host name as appropriate to copy your local public key to the SSH host.

- Connecting to a **macOS or Linux** SSH host:

```
export USER_AT_HOST="your-user-name-on-host@hostname"
export PUBKEYPATH="$HOME/.ssh/id_ed25519.pub"

ssh-copy-id -i "$PUBKEYPATH" "$USER_AT_HOST"
```

- Connecting to a **Windows** SSH host:

```
export USER_AT_HOST="your-user-name-on-host@hostname"
export PUBKEYPATH="$HOME/.ssh/id_ed25519.pub"

ssh $USER_AT_HOST "powershell New-Item -Force -ItemType Directory -Path \"\$HOME\\.ssh\"; Add-Content -Force -Path \"\$HOME\\.ssh\\authorized_keys\" -Value '$(tr -d '\n\r' < \"$PUBKEYPATH\")'"
```

You may want to validate that the `authorized_key` file in the `.ssh` folder for your **remote user on the SSH host** is owned by you and no other user has permission to access it. See the OpenSSH wiki (https://github.com/PowerShell/Win32-OpenSSH/wiki/Security-protection-of-various-files-in-Win32-OpenSSH#authorized_keys) for details.

Authorize your Windows machine to connect

Run one of the following commands, in a **local PowerShell** window replacing user and host name as appropriate to copy your local public key to the SSH host.

- Connecting to a **macOS or Linux** SSH host:

```
$USER_AT_HOST="your-user-name-on-host@hostname"
$PUBKEYPATH="$HOME\.ssh\id_ed25519.pub"

$pubKey=(Get-Content "$PUBKEYPATH" | Out-String); ssh "$USER_AT_HOST" "mkdir -p ~/.ssh && chmod 700 ~/.ssh && echo '${pubKey}' >> ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys"
```

- Connecting to a **Windows** SSH host:

```
$USER_AT_HOST="your-user-name-on-host@hostname"
$PUBKEYPATH="$HOME\.ssh\id_ed25519.pub"

Get-Content "$PUBKEYPATH" | Out-String | ssh $USER_AT_HOST "powershell `\"New-Item -Force -ItemType Directory -Path `\"$HOME\.ssh\"; Add-Content -Force -Path `\"$HOME\.ssh\authorized_keys\" `\"`\""
```

Validate that the `authorized_key` file in the `.ssh` folder for your **remote user on the SSH host** is owned by you and no other user has permission to access it. See the OpenSSH wiki (https://github.com/PowerShell/Win32-OpenSSH/wiki/Security-protection-of-various-files-in-Win32-OpenSSH#authorized_keys) for details.

Improving your security with a dedicated key #

While using a single SSH key across all your SSH hosts can be convenient, if anyone gains access to your private key, they will have access to all of your hosts as well. You can prevent this by creating a separate SSH key for your development hosts. Just follow these steps:

1. Generate a separate SSH key in a different file.

macOS / Linux: Run the following command in a **local terminal**:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519-remote-ssh
```

Windows: Run the following command in a **local PowerShell**:

```
ssh-keygen -t ed25519 -f "$HOME\.ssh\id_ed25519-remote-ssh"
```

2. Follow the same steps in the quick start to authorize the key on the SSH host, but set the `PUBKEYPATH` to the `id_ed25519-remote-ssh.pub` file instead.
3. In VS Code, run **Remote-SSH: Open Configuration File...** in the Command Palette (`F1`), select an SSH config file, and add (or modify) a host entry as follows:

```
Host name-of-ssh-host-here
  User your-user-name-on-host
  HostName host-fqdn-or-ip-goes-here
  IdentityFile ~/.ssh/id_ed25519-remote-ssh
```

Tip: You can use `/` for Windows paths as well. If you use `\` you will need to use two slashes. For example, `C:\path\to\my\id_ed25519` .

Reusing a key generated in PuTTYGen #

If you used PuTTYGen to set up SSH public key authentication for the host you are connecting to, you need to convert your private key so that other SSH clients can use it. To do this:

1. Open PuTTYGen **locally** and load the private key you want to convert.
2. Select **Conversions > Export OpenSSH key** from the application menu. Save the converted key to a **local** location under the `.ssh` directory in your user profile folder (for example `C:\Users\youruser\.ssh`).
3. Validate that this new **local** file is owned by you and no other user has permissions to access it.
4. In VS Code, run **Remote-SSH: Open Configuration File...** in the Command Palette (`F1`), select the SSH config file you want to change, and add (or modify) a host entry in the config file as follows to point to the file:

```
Host name-of-ssh-host-here
  User your-user-name-on-host
  HostName host-fqdn-or-ip-goes-here
  IdentityFile ~/.ssh/exported-keyfile-from-putty
```

Improving security on multi-user servers #

The Remote - SSH extension installs and maintains the "VS Code Server". The server is started with a randomly generated key, and any new connection to the server needs to provide the key. The key is stored on the remote's disk, readable only by the current user. There is one HTTP path that is available without authentication at `/version` .

By default, the server listens to `localhost` on a random TCP port that is then forwarded to your local machine. If you are connecting to a **Linux or macOS** host, you can switch to using Unix sockets that are locked down to a particular user. This socket is then forwarded instead of the port.

Note: This setting **disables connection multiplexing** so configuring public key authentication is recommended.

To configure it:

1. Ensure you have a **local OpenSSH 6.7+ SSH client** on Windows, macOS, or Linux and an **OpenSSH 6.7+ Linux or macOS Host** (Windows does not support this mode).
2. Switch Remote - SSH into socket mode by enabling **Remote.SSH: Remote Server Listen On Socket** in your **local** VS Code User settings (`/docs/getstarted/settings`).

Remote.SSH: Remote Server Listen On Socket

☐ When true, the remote VS Code server will listen on a socket path instead of opening a port. Only valid for Linux and macOS remotes. After toggling this setting, run the command "Kill VS Code Server on Host..." for it to take effect. Requires OpenSSH 6.7. Disables dynamic port forwarding and "local server" mode. Requires **AllowStreamLocalForwarding** to be enabled for the SSH server.

3. If you've already connected to the SSH Host, select **Remote-SSH: Kill VS Code Server on Host...** from the Command Palette (`F1`) so the setting takes effect.

If you encounter an error when connecting, you may need to enable socket forwarding on your SSH Host's `sshd` config (https://www.ssh.com/ssh/sshd_config/). To do so:

1. Open `/etc/ssh/sshd_config` in a text editor (like `vi`, `nano`, or `pico`) on the **SSH host** (not locally).
2. Add the setting `AllowStreamLocalForwarding yes`.
3. Restart the SSH server. (On Ubuntu, run `sudo systemctl restart sshd`.)
4. Retry.

Troubleshooting hanging or failing connections

If you are running into problems with VS Code hanging while trying to connect (and potentially timing out), there are a few things you can do to try to resolve the issue.

General troubleshooting: Remove the server

One command helpful to troubleshoot a variety of Remote-SSH issues is **Remote-SSH: Kill VS Code Server on Host**. This will remove the server, which can fix a wide range of issues and error messages you may see, such as "Could not establish connection to `server_name`: The VS Code Server failed to start."

See if VS Code is waiting on a prompt

Enable the `remote.SSH.showLoginTerminal` setting (`/docs/getstarted/settings`) in VS Code and retry. If you are prompted to input a password or token, see [Enabling alternate SSH authentication methods](#) for details on reducing the frequency of prompts.

If you are still having trouble, set the following properties in `settings.json` and retry:

```
"remote.SSH.showLoginTerminal": true,
"remote.SSH.useLocalServer": false
```

Work around a bug with some versions of Windows OpenSSH server

Due to a bug in certain versions of OpenSSH server for Windows, the default check to determine if the host is running Windows may not work properly. This does not occur with OpenSSH server that ships with Windows 1909 and below.

Fortunately, you can work around this problem by specifically telling VS Code if your SSH host is running Windows by adding the following to `settings.json`:

```
"remote.SSH.useLocalServer": false
```

You can also force VS Code to identify a particular host as Windows using the following property:

```
"remote.SSH.remotePlatform": {
  "host-in-ssh-config-or-fqdn": "windows"
}
```

A fix has been merged so this problem should be resolved in a version of the server greater than 8.1.0.0.

Enable TCP Forwarding on the remote host

Remote - SSH extension makes use of an SSH tunnel to facilitate communication with the host. In some cases, this may be disabled on your SSH server. To see if this is the problem, open the **Remote - SSH** category in the output window and check for the following message:

```
open failed: administratively prohibited: open failed
```

If you do see that message, follow these steps to update your SSH server's `sshd` config (https://www.ssh.com/ssh/sshd_config/):

1. Open `/etc/ssh/sshd_config` or `C:\ProgramData\ssh\sshd_config` in a text editor (like `Vim`, `nano`, `Pico`, or `Notepad`) on the **SSH host** (not locally).
2. Add the setting `AllowTcpForwarding yes`.
3. Restart the SSH server. (On Ubuntu, run `sudo systemctl restart sshd`. On Windows, in an admin PowerShell run, `Restart-Service sshd`.)
4. Retry.

Set the ProxyCommand parameter in your SSH config file

If you are behind a proxy and are unable to connect to your SSH host, you may need to use the `ProxyCommand` parameter for your host in a **local** SSH config file (https://linux.die.net/man/5/ssh_config). You can read this SSH ProxyCommand article (<https://www.cyberciti.biz/faq/linux-unix-ssh-proxycommand-passing-through-one-host-gateway-server/>) for an example of its use.

Ensure the remote machine has internet access

The remote machine must have internet access to be able to download the VS Code Server and extensions from the Marketplace. See the FAQ for details (/docs/remote/faq#_what-are-the-connectivity-requirements-for-vs-code-server) on connectivity requirements.

Set HTTP_PROXY / HTTPS_PROXY on the remote host

If your remote host is behind a proxy, you may need to set the HTTP_PROXY or HTTPS_PROXY environment variable on the **SSH host**. Open your `~/.bashrc` file add the following (replacing `proxy.fqdn.or.ip:3128` with the appropriate hostname / IP and port):

```
export HTTP_PROXY=http://proxy.fqdn.or.ip:3128
export HTTPS_PROXY=$HTTP_PROXY

# Or if an authenticated proxy
export HTTP_PROXY=http://username:password@proxy.fqdn.or.ip:3128
export HTTPS_PROXY=$HTTP_PROXY
```

Work around /tmp mounted with noexec

Some remote servers are set up to disallow executing scripts from `/tmp`. VS Code writes its install script to the system temp directory and tries to execute it from there. You can work with your system administrator to determine whether this can be worked around.

Check whether a different shell is launched during install

Some users launch a different shell from their `.bash_profile` or other startup script on their **SSH host** because they want to use a different shell than the default. This can break VS Code's remote server install script and isn't recommended. Instead, use `chsh` to change your default shell on the remote machine.

Connecting to systems that dynamically assign machines per connection

Some systems will dynamically route an SSH connection to one node from a cluster each time an SSH connection is made. This is an issue for VS Code because it makes two connections to open a remote window: the first to install or start the VS Code Server (or find an already running instance) and the second to create the SSH port tunnel that VS Code uses to talk to the server. If VS Code is routed to a different machine when it creates the second connection, it won't be able to talk to the VS Code server.

One workaround for this is to use the `ControlMaster` option in OpenSSH (macOS/Linux clients only), described in [Enabling alternate SSH authentication methods](#), so that VS Code's two connections will be multiplexed through a single SSH connection to the same node.

Contact your system administrator for configuration help

SSH is a very flexible protocol and supports many configurations. If you see other errors, in either the login terminal or the **Remote-SSH** output window, they could be due to a missing setting.

Contact your system administrator for information about the required settings for your SSH host and client. Specific command-line arguments for connecting to your SSH host can be added to an SSH config file (https://linux.die.net/man/5/ssh_config).

To access your config file, run **Remote-SSH: Open Configuration File...** in the Command Palette (`F1`). You can then work with your admin to add the necessary settings.

Enabling alternate SSH authentication methods

If you are connecting to an SSH remote host and are either:

- Connecting with two-factor authentication
- Using password authentication
- Using an SSH key with a passphrase when the SSH Agent is not running or accessible

then VS Code should automatically prompt you to enter needed information. If you do not see the prompt, enable the `remote.SSH.showLoginTerminal` setting (</docs/getstarted/settings>) in VS Code. This setting displays the terminal whenever VS Code runs an SSH command. You can then enter your authentication code, password, or passphrase when the terminal appears.

If you are still having trouble, you may need to the following properties in `settings.json` and retry:

```
"remote.SSH.showLoginTerminal": true,
"remote.SSH.useLocalServer": false
```

If you are on macOS and Linux and want to reduce how often you have to enter a password or token, you can enable the `ControlMaster` feature on your **local machine** so that OpenSSH runs multiple SSH sessions over a single connection.

To enable `ControlMaster` :

1. Add an entry like this to your SSH config file:

```
Host *
  ControlMaster auto
  ControlPath ~/.ssh/sockets/%r@%h-%p
  ControlPersist 600
```

2. Then run `mkdir -p ~/.ssh/sockets` to create the sockets folder.

Setting up the SSH Agent

If you are connecting to an SSH host using a key with a passphrase, you should ensure that the SSH Agent (<https://www.ssh.com/ssh/agent>) is running **locally**. VS Code will automatically add your key to the agent so you don't have to enter your passphrase every time you open a remote VS Code window.

To verify that the agent is running and is reachable from VS Code's environment, run `ssh-add -l` in the terminal of a local VS Code window. You should see a listing of the keys in the agent (or a message that it has no keys). If the agent is not running, follow these instructions to start it. After starting the agent, be sure to restart VS Code.

Windows:

To enable SSH Agent automatically on Windows, start a **local Administrator PowerShell** and run the following commands:

```
# Make sure you're running as an Administrator
Set-Service ssh-agent -StartupType Automatic
Start-Service ssh-agent
Get-Service ssh-agent
```

Now the agent will be started automatically on login.

Linux:

To start the SSH Agent in the background, run:

```
eval "$(ssh-agent -s)"
```

To start the SSH Agent automatically on login, add these lines to your `~/.bash_profile`:

```
if [ -z "$SSH_AUTH_SOCK" ]; then
  # Check for a currently running instance of the agent
  RUNNING_AGENT="`ps -ax | grep 'ssh-agent -s' | grep -v grep | wc -l | tr -d '[:space:]'`"
  if [ "$RUNNING_AGENT" = "0" ]; then
    # Launch a new instance of the agent
    ssh-agent -s &> ~/.ssh/ssh-agent
  fi
  eval `cat ~/.ssh/ssh-agent`
fi
```

macOS:

The agent should be running by default on macOS.

Making local SSH Agent available on the remote

An SSH Agent on your local machine allows the Remote - SSH extension to connect to your chosen remote system without repeatedly prompting for a passphrase, but tools like Git that run on the remote, don't have access to your locally-unlocked private keys.

You can see this by opening the integrated terminal on the remote and running `ssh-add -l`. The command should list the unlocked keys, but instead reports an error about not being able to connect to the authentication agent. Setting `ForwardAgent yes` makes the local SSH Agent available in the remote environment, solving this problem.

You can do this by editing your `.ssh/config` file (or whatever `Remote.SSH.configFile` is set to - use the **Remote-SSH: Open SSH Configuration File...** command to be sure) and adding:

```
Host *
  ForwardAgent yes
```

Note that you might want to be more restrictive and only set the option for particular named hosts.

Fixing SSH file permission errors

SSH can be strict about file permissions and if they are set incorrectly, you may see errors such as "WARNING: UNPROTECTED PRIVATE KEY FILE!". There are several ways to update file permissions in order to fix this, which are described in the sections below.

Local SSH file and folder permissions

macOS / Linux:

On your local machine, make sure the following permissions are set:

Folder / File	Permissions
.ssh in your user folder	chmod 700 ~/.ssh
.ssh/config in your user folder	chmod 600 ~/.ssh/config
.ssh/id_ed25519.pub in your user folder	chmod 600 ~/.ssh/id_ed25519.pub
Any other key file	chmod 600 /path/to/key/file

Windows:

The specific expected permissions can vary depending on the exact SSH implementation you are using. We recommend using the out of box Windows 10 OpenSSH Client (https://docs.microsoft.com/windows-server/administration/openssh/openssh_overview).

In this case, make sure that all of the files in the .ssh folder for your remote user on the SSH host is owned by you and no other user has permissions to access it. See the Windows OpenSSH wiki (<https://github.com/PowerShell/Win32-OpenSSH/wiki/Security-protection-of-various-files-in-Win32-OpenSSH>) for details.

For all other clients, consult your client's documentation for what the implementation expects.

Server SSH file and folder permissions #

macOS / Linux:

On the remote machine you are connecting to, make sure the following permissions are set:

Folder / File	Linux / macOS Permissions
.ssh in your user folder on the server	chmod 700 ~/.ssh
.ssh/authorized_keys in your user folder on the server	chmod 600 ~/.ssh/authorized_keys

Note that only Linux hosts are currently supported, which is why permissions for macOS and Windows 10 have been omitted.

Windows:

See the Windows OpenSSH wiki (<https://github.com/PowerShell/Win32-OpenSSH/wiki/Security-protection-of-various-files-in-Win32-OpenSSH>) for details on setting the appropriate file permissions for the Windows OpenSSH server.

Installing a supported SSH client #

OS	Instructions
Windows 10 1803+ / Server 2016/2019 1803+	Install the Windows OpenSSH Client (https://docs.microsoft.com/windows-server/administration/openssh/openssh_install_firstuse).
Earlier Windows	Install Git for Windows (https://git-scm.com/download/win).
macOS	Comes pre-installed.
Debian/Ubuntu	Run <code>sudo apt-get install openssh-client</code>
RHEL / Fedora / CentOS	Run <code>sudo yum install openssh-clients</code>

VS Code will look for the ssh command in the PATH. Failing that, on Windows it will attempt to find ssh.exe in the default Git for Windows install path. You can also specifically tell VS Code where to find the SSH client by adding the remote.SSH.path property to settings.json .

Installing a supported SSH server #

OS	Instructions	Details
Debian 8+ / Ubuntu 16.04+	Run <code>sudo apt-get install openssh-server</code>	See the Ubuntu SSH (https://help.ubuntu.com/community/SSH?action=show) documentation for details.
RHEL / CentOS 7+	Run <code>sudo yum install openssh-server && sudo systemctl start sshd.service && sudo systemctl enable sshd.service</code>	See the RedHat SSH (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/ch-openssh) documentation for details.
SuSE 12+ / openSUSE 42.3+	In Yast, go to Services Manager, select "sshd" in the list, and click Enable . Next go to Firewall, select the Permanent configuration, and under services check sshd .	See the SuSE SSH (https://en.opensuse.org/OpenSSH) documentation for details.

OS	Instructions	Details
Windows 10 1803+ / Server 2016/2019 1803+	Install the Windows OpenSSH Server (https://docs.microsoft.com/windows-server/administration/openssh/openssh_install_firstuse).	
macOS 10.14+ (Mojave)	Enable Remote Login (https://support.apple.com/guide/mac-help/allow-a-remote-computer-to-access-your-mac-mchlp1066/mac).	

Resolving hangs when doing a Git push or sync on an SSH host #

If you clone a Git repository using SSH and your SSH key has a passphrase, VS Code's pull and sync features may hang when running remotely.

Either use an SSH key without a passphrase, clone using HTTPS, or run `git push` from the command line to work around the issue.

Using SSHFS to access files on your remote host #

SSHFS (<https://en.wikipedia.org/wiki/SSHFS>) is a secure remote filesystem access protocol that builds up from SFTP. It provides advantages over something like a CIFS / Samba share in that all that is required is SSH access to the machine.

Note: For performance reasons, SSHFS is best used for single file edits and uploading/downloading content. If you need to use an application that bulk reads/write to many files at once (like a local source control tool), `rsync` is a better choice.

macOS / Linux:

On Linux, you can use your distribution's package manager to install SSHFS. For Debian/Ubuntu: `sudo apt-get install sshfs`

Note: WSL 1 does not support FUSE or SSHFS, so the instructions differ for Windows currently. **WSL 2 does include FUSE and SSHFS support**, so this will change soon.

On macOS, you can install SSHFS using Homebrew (<https://brew.sh/>):

```
brew install --cask macfuse
brew install gromgit/fuse/sshfs-mac
brew link --overwrite sshfs-mac
```

In addition, if you would prefer not to use the command line to mount the remote filesystem, you can also install SSHFS GUI (<https://github.com/dstuecken/sshfs-gui>).

To use the command line, run the following commands from a local terminal (replacing `user@hostname` with the remote user and hostname / IP):

```
export USER_AT_HOST=user@hostname
# Make the directory where the remote filesystem will be mounted
mkdir -p "$HOME/sshfs/$USER_AT_HOST"
# Mount the remote filesystem
sshfs "$USER_AT_HOST:" "$HOME/sshfs/$USER_AT_HOST" -ovolname="$USER_AT_HOST" -p 22 \
-o workaround=nonodelay -o transform_symlinks -o idmap=user -C
```

This will make your home folder on the remote machine available under the `~/sshfs`. When you are done, you can unmount it using your OS's Finder / file explorer or by using the command line:

```
umount "$HOME/sshfs/$USER_AT_HOST"
```

Windows:

Follow these steps:

1. On Linux, add `.gitattributes` file to your project to **force consistent line endings** between Linux and Windows to avoid unexpected issues due to CRLF/LF differences between the two operating systems. See [Resolving Git line ending issues](#) for details.
2. Next, install SSHFS-Win (<https://github.com/billziss-gh/sshfs-win>) using Chocolatey (<https://chocolatey.org/>): `choco install sshfs`
3. Once you've installed SSHFS for Windows, you can use the File Explorer's **Map Network Drive...** option with the path `\\sshfs\user@hostname`, where `user@hostname` is your remote user and hostname / IP. You can script this using the command prompt as follows: `net use /PERSISTENT:NO X: \\sshfs\user@hostname`
4. Once done, disconnect by right-clicking on the drive in the File Explorer and selecting **Disconnect**.

Connect to a remote host from the terminal #

Once a host has been configured, you can connect to it directly from the terminal by passing a remote URI.

For example, to connect to `remote_server` and open the `/code/my_project` folder, run:

```
code --remote ssh-remote+remote_server /code/my_project
```

We need to do some guessing on whether the input path is a file or a folder. If it has a file extension, it is considered a file.

To force that a folder is opened, add slash to the path or use:

```
code --folder-uri vscode-remote://ssh-remote+remote_server/code/folder.with.dot
```

To force that a file is opened, add `--goto` or use:

```
code --file-uri vscode-remote://ssh-remote+remote_server/code/fileWithoutExtension
```

Using `rsync` to maintain a local copy of your source code #

An alternative to using SSHFS to access remote files is to use `rsync` (<https://rsync.samba.org/>) to copy the entire contents of a folder on remote host to your local machine. The `rsync` command will determine which files need to be updated each time it is run, which is far more efficient and convenient than using something like `scp` or `sftp`. This is primarily something to consider if you really need to use multi-file or performance intensive local tools.

The `rsync` command is available out of box on macOS and can be installed using Linux package managers (for example `sudo apt-get install rsync` on Debian/Ubuntu). For Windows, you'll need to either use WSL (<https://docs.microsoft.com/windows/wsl/install>) or Cygwin (<https://www.cygwin.com/>) to access the command.

To use the command, navigate to the folder you want to store the synched contents and run the following replacing `user@hostname` with the remote user and hostname / IP and `/remote/source/code/path` with the remote source code location.

On macOS, Linux, or inside WSL:

```
rsync -rlptzv --progress --delete --exclude=.git "user@hostname:/remote/source/code/path" .
```

Or using WSL from PowerShell on Windows:

```
wsl rsync -rlptzv --progress --delete --exclude=.git "user@hostname:/remote/source/code/path" "`$(wslpath -a '$PWD')"
```

You can rerun this command each time you want to get the latest copy of your files and only updates will be transferred. The `.git` folder is intentionally excluded both for performance reasons and so you can use local Git tools without worrying about the state on the remote host.

To push content, reverse the source and target parameters in the command. However, **on Windows** you should add a `.gitattributes` file to your project to **force consistent line endings** before doing so. See [Resolving Git line ending issues](#) for details.

```
rsync -rlptzv --progress --delete --exclude=.git . "user@hostname:/remote/source/code/path"
```

Cleaning up the VS Code Server on the remote #

The SSH extension provides a command for cleaning up the VS Code Server from the remote machine, **Remote-SSH: Uninstall VS Code Server from Host...** The command does two things: it kills any running VS Code Server processes and it deletes the folder where the server was installed.

If you want to run these steps manually, or if the command isn't working for you, you can run a script like this:

```
# Kill server processes
kill -9 `ps aux | \grep vscode-server | \grep USER | \grep -v grep | awk '{print $2}'`
# Delete related files and folder
rm -rf $HOME/.vscode-server # Or ~/.vscode-server-insiders
```

The VS Code Server was previously installed under `~/.vscode-remote` so you can check that location too.

SSH into a remote WSL 2 host #

You may want to use SSH to connect to a WSL distro running on your remote machine. Check out this guide (<https://www.hanselman.com/blog/the-easy-way-how-to-ssh-into-bash-and-wsl2-on-windows-10-from-an-external-machine>) to learn how to SSH into Bash and WSL 2 on Windows 10 from an external machine.

Container tips #

This section includes some tips and tricks for getting the Remote - Containers extension up and running in different environments.

If you are running into Docker issues or would prefer not to run Docker locally, you may want to try the preview of GitHub Codespaces managed cloud-based environments (<https://github.com/features/codespaces>). Over time this service will support an increasing number of `devcontainer.json` properties and you can also use its browser-based editor in addition to VS Code.

Docker Desktop for Windows tips #

Docker Desktop (<https://www.docker.com/products/docker-desktop>) for Windows works well in most setups, but there are a few "gotchas" that can cause problems. Here are some tips on avoiding them:

1. **Consider using the new Docker WSL 2 back-end on Windows 10 (2004+).** If you are using Docker Desktop's WSL 2 back-end (<https://aka.ms/vscode-remote/containers/docker-wsl2>), you can now open folders inside WSL as well as locally. Containers are also shared between Windows and inside WSL and this new engine is less susceptible to file sharing issues. See the quick start (/docs/remote/containers#_open-a-wsl-2-folder-in-a-container-on-windows) for details.
2. **Switch out of "Linux Containers on Windows (LCOW)" mode.** While disabled by default, recent versions of Docker support Linux Containers on Windows (LCOW) (<https://docs.microsoft.com/virtualization/windowscontainers/deploy-containers/linux-containers>) that can allow you to use both Windows and Linux containers at the same time. However, this is a new feature, so you may encounter issues and the Remote - Containers extension only supports Linux containers currently. You can switch out of LCOW mode at any time by right-clicking on the Docker task bar item and selecting **Switch to Linux Containers...** from the context menu.
3. **Make sure your firewall allows Docker to set up a shared drive.** Docker only needs to connect between two machine local IPs, but some firewall software may still block any drive sharing or the needed ports. See this Docker KB article (<https://success.docker.com/article/error-a-firewall-is-blocking-file-sharing-between-windows-and-the-containers>) for next steps on resolving this problem.

Here are some tips that applied to older versions of Docker for Windows but should now be resolved. If you run into strange behaviors due to a possible regression, these tips have solved problems in the past.

1. **Use an AD domain account or local administrator account when sharing drives. Do not use an AAD (email-based) account.** AAD (email-based) accounts have well-known issues, as documented in Docker issue #132 (<https://github.com/docker/for-win/issues/132>) and issue #1352 (<https://github.com/docker/for-win/issues/1352>). If you must use an AAD account, create a separate local administrator account on your machine that you use purely for the purpose of sharing drives. Follow the steps in this blog post (<https://blogs.msdn.microsoft.com/stevlasker/2016/06/14/configuring-docker-for-windows-volumes/>) to get everything set up.
2. **Stick with alphanumeric passwords to avoid drive sharing problems.** When asked to share your drives on Windows, you will be prompted for the username and password of an account with admin privileges on the machine. If you are warned about an incorrect username or password, this may be due to special characters in the password. For example, `!`, `[` and `]` are known to cause issues. Change your password to alphanumeric characters to resolve. See this issue about Docker volume mounting problems (<https://github.com/moby/moby/issues/23992#issuecomment-234979036>) for details.
3. **Use your Docker ID to sign in to Docker (not your email).** The Docker CLI only supports using your Docker ID, so using your email can cause problems. See Docker issue #935 (<https://github.com/docker/hub-feedback/issues/935#issuecomment-300361781>) for details.

If you are still having trouble, see the Docker Desktop for Windows troubleshooting guide (<https://docs.docker.com/docker-for-windows/troubleshoot/#volumes>).

Enabling file sharing in Docker Desktop #

The VS Code Remote - Containers (<https://aka.ms/vscode-remote/download/containers>) extension can only automatically mount your source code into a container if your code is in a folder or drive shared with Docker. If you open a dev container from a non-shared location, the container will successfully start but the workspace will be empty.

Note that this step is **not required** with Docker Desktop's WSL 2 engine (<https://aka.ms/vscode-remote/containers/docker-wsl2>).

To change Docker's drive and folder sharing settings:

Windows:

1. Right-click on the Docker task bar item and select **Settings**.
2. Go to **Resources > File Sharing** and check the drive(s) where your source code is located.
3. If you see a message about your local firewall blocking the sharing action, see this Docker KB article (<https://success.docker.com/article/error-a-firewall-is-blocking-file-sharing-between-windows-and-the-containers>) for next steps.

macOS:

1. Click on the Docker menu bar item and select **Preferences**.
2. Go to **Resources > File Sharing**. Confirm that the folder containing your source code is under one of the shared folders listed.

Resolving Git line ending issues in containers (resulting in many modified files) #

Since Windows and Linux use different default line endings, Git may report a large number of modified files that have no differences aside from their line endings. To prevent this from happening, you can disable line ending conversion using a `.gitattributes` file or globally on the Windows side.

Typically adding or modifying a `.gitattributes` file in your repository is the most reliable way to solve this problem. Committing this file to source control will help others and allows you to vary behaviors by repository as appropriate. For example, adding the following to `.gitattributes` file to the root of your repository will force everything to be LF, except for Windows batch files that require CRLF:

```
* text=auto eol=lf
*.{cmd,[cC][mM][dD]} text eol=crlf
*.{bat,[bB][aA][tT]} text eol=crlf
```

Note that this works in **Git v2.10+**, so if you are running into problems, be sure you've got a recent Git client installed. You can add other file types in your repository that require CRLF to this same file.

If you would prefer to still always upload Unix-style line endings (LF), you can use the `input` option.

```
git config --global core.autocrlf input
```

If you'd prefer to disable line-ending conversion entirely, run the following instead:

```
git config --global core.autocrlf false
```

Finally, you may need to clone the repository again for these settings to take effect.

Avoid setting up Git in a container when using Docker Compose #

See [Sharing Git credentials with your container \(/docs/remote/containers#_sharing-git-credentials-with-your-container\)](#) in the main containers article for information on resolving this issue.

Resolving hangs when doing a Git push or sync from a Container #

If you clone a Git repository using SSH and your SSH key has a passphrase, VS Code's pull and sync features may hang when running remotely.

Either use an SSH key without a passphrase, clone using HTTPS, or run `git push` from the command line to work around the issue.

Resolving errors about missing Linux dependencies #

Some extensions rely on libraries not found in the certain Docker images. See the [Containers \(/docs/remote/create-dev-container#_install-additional-software\)](#) article for a few options on resolving this issue.

Speeding up containers in Docker Desktop #

By default, Docker Desktop only gives containers a fraction of your machine capacity. In most cases, this is enough, but if you are doing something that requires more capacity, you can increase memory, CPU, or disk use.

First, try stopping any running containers ([/docs/remote/containers#_managing-containers](#)) you are no longer using.

If this doesn't solve your problem, you may want to see if CPU usage is actually the issue or if there is something else going on. An easy way to check this is to install the Resource Monitor extension (<https://marketplace.visualstudio.com/items?itemName=mutantdino.resourcemonitor&ssr=false#overview>). When installed in a container, it provides information about capacity for your containers in the Status bar.



If you'd like this extension to always be installed, add this to your `settings.json`:

```
"remote.containers.defaultExtensions": [
  "mutantdino.resourcemonitor"
]
```

If you determine that you need to give your container more of your machine's capacity, follow these steps:

1. Right-click on the Docker task bar item and select **Settings / Preferences**.
2. Go to **Advanced** to increase CPU, Memory, or Swap.
3. On macOS, go to **Disk** to increase the amount of disk Docker is allowed to consume on your machine. On Windows, this is located under Advanced with the other settings.

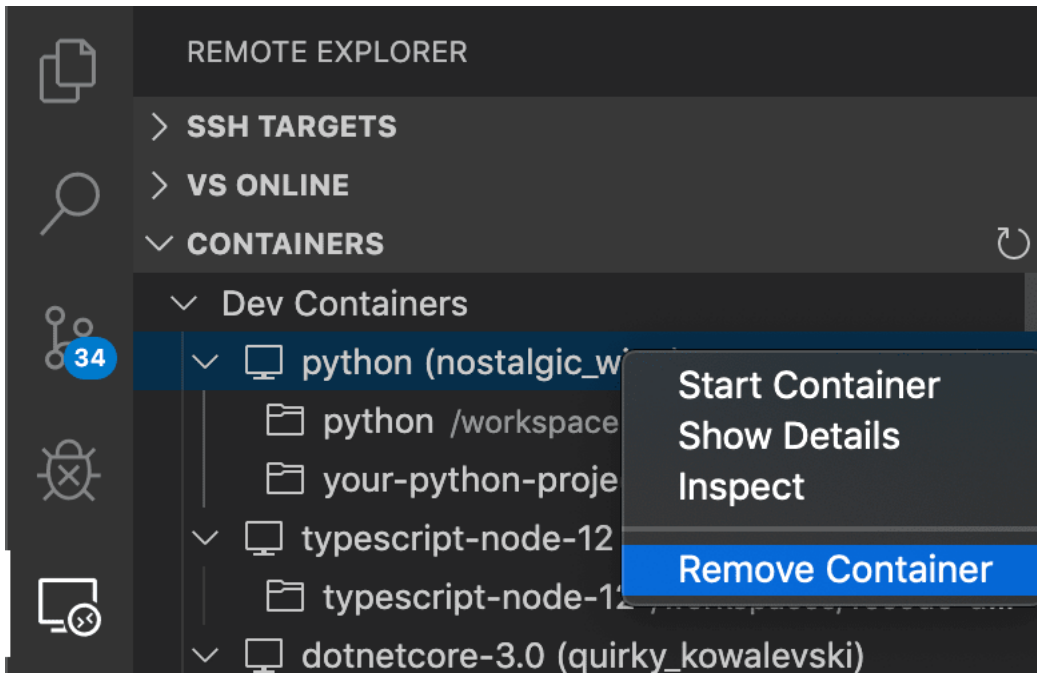
Finally, if your container is **doing disk intensive** operations or you are just looking for faster response times, see [Improving container disk performance \(/remote/advancedcontainers/improve-performance\)](#) for tips. VS Code's defaults optimize for convenience and universal support, but can be optimized.

Cleaning out unused containers and images #

If you see an error from Docker reporting that you are out of disk space, you can typically resolve this by cleaning out unused containers and images. There are a few ways to do this:

Option 1: Use the Remote Explorer

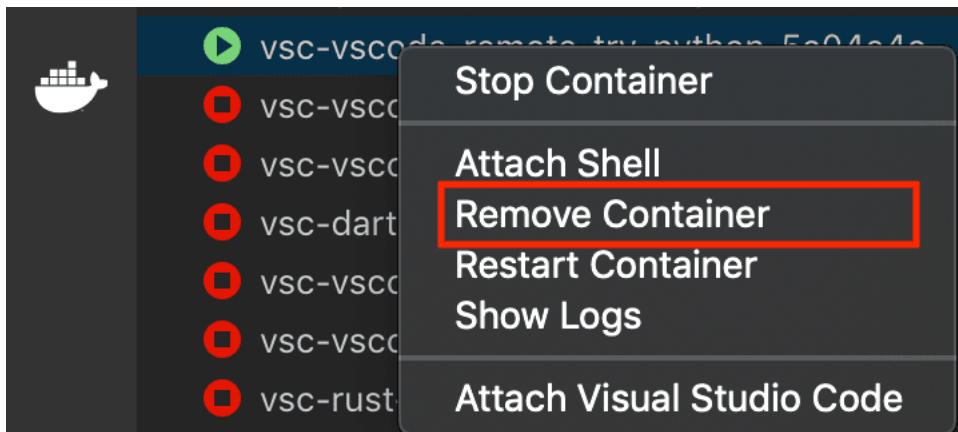
You can delete containers by selecting the **Remote Explorer**, right-click on the container you want to remove, and select **Remove Container**.



However, this does not clean up any images you may have downloaded, which can clutter up your system.

Option 2: Use the Docker extension

1. Open a **local** window in VS Code (**File > New Window**).
2. Install the Docker extension (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker>) from the Extensions view if not already present.
3. You can then go to the Docker view and expand the **Containers** or **Images** node, right-click, and select **Remove Container / Image**.



Option 3: Use the Docker CLI to pick containers to delete

1. Open a **local** terminal/command prompt (or use a local window in VS Code).
2. Type `docker ps -a` to see a list of all containers.
3. Type `docker rm <Container ID>` from this list to remove a container.
4. Type `docker image prune` to remove any unused images.

If `docker ps` does not provide enough information to identify the container you want to delete, the following command will list all development containers managed by VS Code and the folder used to generate them.

```
docker ps -a --filter="label=vsch.quality" --format "table {{.ID}}\t{{.Status}}\t{{.Image}}\tvscodex-{{.Label \"vsch.quality\"}}\t{{.Label \"vsch.local.folder\"}}"
```

Option 4: Use Docker Compose

1. Open a **local** terminal/command prompt (or use a local window in VS Code).
2. Go to the directory with your `docker-compose.yml` file.
3. Type `docker-compose down` to stop and delete the containers. If you have more than one Docker Compose file, you can specify additional Docker Compose files with the `-f` argument.

Option 4: Delete all containers and images that are not running:

1. Open a **local** terminal/command prompt (or use a local window in VS Code).
2. Type `docker system prune --all`.

Resolving Dockerfile build failures for images using Debian 8

When building containers that use images based on Debian 8/Jessie — such as older versions of the `node:8` image — you may encounter the following error:

```
...
W: Failed to fetch http://deb.debian.org/debian/dists/jessie-updates/InRelease Unable to find expected entry 'main/binary-amd64/Packages' in Release file (Wrong sources.list entry or malformed file)
E: Some index files failed to download. They have been ignored, or old ones used instead.
...
```

This is a well known issue (<https://github.com/debuerreotype/docker-debian-artifacts/issues/66>) caused by the Debian 8 being "archived". More recent versions of images typically resolve this problem, often by upgrading to Debian 9/Stretch.

There are two ways to resolve this error:

- **Option 1:** Remove any containers that depend on the image, remove the image, and then try building again. This should download an updated image that is not affected by the problem. See cleaning out unused containers and images for details.
- **Option 2:** If you don't want to delete your containers or images, add this line into your Dockerfile before any `apt` or `apt-get` command. It adds the needed source lists for Jessie:

```
# Add archived sources to source list if base image uses Debian 8 / Jessie
RUN cat /etc/*-release | grep -q jessie && printf "deb http://archive.debian.org/debian/ jessie main\ndeb-src http://archive.debian.org/debian/ jessie main\ndeb http://security.debian.org jessie/updates main\ndeb-src http://security.debian.org jessie/updates main" > /etc/apt/sources.list
```

Resolving Docker Hub sign in errors when an email is used

The Docker CLI only supports using your Docker ID, so using your email to sign in can cause problems. See Docker issue #935 (<https://github.com/docker/hub-feedback/issues/935#issuecomment-300361781>) for details.

As a workaround, use your Docker ID to sign in to Docker rather than your email.

High CPU utilization of Hyperkit on macOS

There is known issue with Docker for Mac (<https://github.com/docker/for-mac/issues/1759>) that can drive high CPU spikes. In particular, high CPU usage occurring when watching files and building. If you see high CPU usage for `com.docker.hyperkit` in Activity Monitor while very little is going on in your dev container, you are likely hitting this issue. Follow the Docker issue (<https://github.com/docker/for-mac/issues/1759>) for updates and fixes.

Using an SSH tunnel to connect to a remote Docker host

The Develop inside a container on a remote Docker Machine or SSH host ([/remote/advancedcontainers/develop-remote-host](https://remote.advancedcontainers.com/develop-remote-host)) article covers how to setup VS Code when working with a remote Docker host. This is often as simple as setting the Docker extension (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker>) `docker.host` property in `settings.json` or the `DOCKER_HOST` environment variable to a `ssh://` or `tcp://` URI.

However, you may run into situations where this does not work in your environment due to SSH configuration complexity or other limitations. In this case, an SSH tunnel can be used as a fallback.

Using an SSH tunnel as a fallback option

You can set up an SSH tunnel and forward the Docker socket from your remote host to your local machine.

Follow these steps:

1. Install an OpenSSH compatible SSH client ([/docs/remote/troubleshooting#_installing-a-supported-ssh-client](https://docs.remote.troubleshooting#_installing-a-supported-ssh-client)).
2. Update the Docker extension (<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker>) `docker.host` property in your user or workspace `settings.json` as follows:

```
"docker.host": "tcp://localhost:23750"
```

3. Run the following command from a local terminal / PowerShell (replacing `user@hostname` with the remote user and hostname / IP for your server):

```
ssh -NL localhost:23750:/var/run/docker.sock user@hostname
```

VS Code will now be able to attach to any running container ([/docs/remote/attach-container](https://docs.remote/attach-container)) on the remote host. You can also use specialized, local `devcontainer.json` files to create / connect to a remote dev container ([/remote/advancedcontainers/develop-remote-host#_converting-an-existing-or-predefined-devcontainerjson](https://remote.advancedcontainers.com/develop-remote-host#_converting-an-existing-or-predefined-devcontainerjson)).

Once you are done, press `Ctrl+C` in the terminal / PowerShell to close the tunnel.

Note: If the `ssh` command fails, you may need to `AllowStreamLocalForwarding` on your SSH host.

1. Open `/etc/ssh/sshd_config` in an editor (like Vim, nano, or Pico) on the **SSH host** (not locally).
2. Add the setting `AllowStreamLocalForwarding yes`.
3. Restart the SSH server (on Ubuntu, run `sudo systemctl restart sshd`).
4. Retry.

Persisting user profile #

You can use the `mounts` property to persist the user profile (to keep things like shell history) in your dev container across rebuilds.

```
"mounts": [
  "source=profile,target=/root,type=volume",
  "target=/root/.vscode-server,type=volume"
],
```

The above code first creates a named volume called `profile` mounted to `/root`, which will survive a rebuild. It next creates an anonymous volume mounted to `/root/.vscode-server` that gets destroyed on rebuild, which allows VS Code to reinstall extensions and dotfiles.

Advanced container configuration tips #

See the Advanced container configuration (</remote/advancedcontainers/overview>) articles for information on the following topics:

- Adding environment variables (</remote/advancedcontainers/environment-variables>)
- Adding another local file mount (</remote/advancedcontainers/add-local-file-mount>)
- Changing or removing the default source code mount (</remote/advancedcontainers/change-default-source-mount>)
- Improving container disk performance (</remote/advancedcontainers/improve-performance>)
- Adding a non-root user to your dev container (</remote/advancedcontainers/add-nonroot-user>)
- Avoiding extension reinstalls on container rebuild (</remote/advancedcontainers/avoid-extension-reinstalls>)
- Setting the project name for Docker Compose (</remote/advancedcontainers/set-docker-compose-project-name>)
- Using Docker or Kubernetes from inside a container (</remote/advancedcontainers/use-docker-kubernetes>)
- Connecting to multiple containers at once (</remote/advancedcontainers/connect-multiple-containers>)
- Developing inside a container on a remote Docker Machine or SSH host (</remote/advancedcontainers/develop-remote-host>)
- Reducing Dockerfile build warnings (</remote/advancedcontainers/reduce-docker-warnings>)

WSL tips #

First time start: VS Code Server prerequisites #

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

DEBIAN AND UBUNTU

Open the Debian or Ubuntu WSL shell to add `wget` and `ca-certificates`:

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

ALPINE

Open the Alpine WSL shell as root (`wsl -d Alpine -u root`) to add `libstdc++`:

```
apk update && apk add libstdc++
```

On Windows 10 April 2018 Update (build 1803) and older, `/bin/bash` is required:

```
apk update && apk add bash
```

Selecting the distribution used by Remote - WSL #

Remote-WSL: New Window will open the WSL distro registered as default.

To open a non-default distro, run `code .` from the WSL shell of the distro to use or use **Remote-WSL: New Window using Distro**.

With WSL versions older than Windows 10, May 2019 Update (version 1903), the WSL command can only use the **default distro**. For this reason, the Remote- WSL might prompt you if you agree to change the default distro.

You can always use `wslconfig.exe` (<https://docs.microsoft.com/windows/wsl/wsl-config>) to change your default.

For example:

```
wslconfig /setdefault Ubuntu
```

You can see which distributions you have installed by running:

```
wslconfig /l
```

Configure the environment for the server startup #

When the Remote WSL extension starts the VS Code server in WSL, it does not run any shell configuration scripts. This was done to avoid that custom configuration scripts can prevent the startup.

If you need to configure the startup environment, you can use the environment setup script as described here (/docs/remote/wsl#_advanced-environment-setup-script).

Configure the environment for the remote extension host #

The environment for the remote extension host and terminal are based on the default shell's configuration scripts. To evaluate the environment variables for the remote extension host process, the server creates an instance of the default shell as an **interactive login shell**. It probes the environment variables from it and uses them as the initial environment for the remote extension host process. The values of environment variables therefore depend on what shell is configured as the default and the content of the configuration scripts for that shell.

See Unix shell initialization (<https://github.com/rbenv/rbenv/wiki/unix-shell-initialization>) for an overview of each shell's configuration scripts. Most WSL distributions have `/bin/bash` configured as the default shell. `/bin/bash` will look for startup files under `/etc/profile` first and for any startup files under `~/.bash_profile`, `~/.bash_login`, `~/.profile`.

To change the default shell of a WSL distro, follow the instructions of this blog post (<https://medium.com/@vinhp/set-and-use-zsh-as-default-shell-in-wsl-on-windows-10-the-right-way-4f30ed9592dc>).

Fixing problems with the code command not working #

If typing `code` from a WSL terminal on Windows does not work because `code` cannot be found, you may be missing some key locations from your PATH in WSL.

Check by opening a WSL terminal and typing `echo $PATH`. You should see VS Code install path listed. By default, this would be:

```
/mnt/c/Users/Your Username/AppData/Local/Programs/Microsoft VS Code/bin
```

But, if you used the **System Installer**, the install path is:

```
/mnt/c/Program Files/Microsoft VS Code/bin
```

...Or...

```
/mnt/c/Program Files (x86)/Microsoft VS Code/bin
```

It's a feature of WSL that paths are inherited from the PATH variable in Windows. To change the Windows PATH variable, use the **Edit environment variables for your account** command from the start menu in Windows.

If you have disabled the path sharing feature, edit your `.bashrc`, add the following, and start a new terminal:

```
WINDOWS_USERNAME="Your Windows Alias"

export PATH="$PATH:/mnt/c/Windows/System32:/mnt/c/Users/${WINDOWS_USERNAME}/AppData/Local/Programs/Microsoft VS Code/bin"
# or...
# export PATH="$PATH:/mnt/c/Program Files/Microsoft VS Code/bin"
# or...
# export PATH="$PATH:/mnt/c/Program Files (x86)/Microsoft VS Code/bin"
```

Note: Be sure to quote or escape space characters in the directory names.

Finding problems with the 'code' command #

If typing `code` from a Windows command prompt does not launch VS Code, you can help us diagnose the problem by running `VSCODE_WSL_DEBUG_INFO=true code .`

Please file an issue and attach the full output.

Finding problems starting or connected to the server #

When the WSL window fails to connect to the remote server, you can get more information in the WSL log. When filing an issue, it is important to always send the full content of the WSL log.

Open the WSL log by running the command **Remote-WSL: Open Log**. The log will show in the terminal view under the WSL tab.

```

[2020-09-10 15:56:47.971] + export VSCODE_AGENT_FOLDER=/root/.vscode-server-insiders
[2020-09-10 15:56:47.971] + /root/.vscode-server-insiders/bin/e790b931385d72cf5669fcef51cdf65990efa5
d/server.sh --port=0 --use-host-proxy --enable-remote-auto-shutdown
[2020-09-10 15:56:47.971] NodeSegfaultHandlerNative: about to dereference NULL (will cause a SIGSEGV)
[2020-09-10 15:56:47.971] Segmentation fault
[2020-09-10 15:56:47.971] VS Code Server for WSL closed unexpectedly.
[2020-09-10 15:56:47.971] For help with startup problems, go to
[2020-09-10 15:56:47.971] https://code.visualstudio.com/docs/remote/troubleshooting#_wsl-tips
[2020-09-10 15:56:47.973] C:\WINDOWS\System32\wsl.exe -d Ubuntu -e kill 7575
[2020-09-10 15:56:48.095] WSL Daemon exited with code 0
  
```

Disconnected from WSL: Ubuntu 0 0

To get even more verbose logging, enable the setting `remote.WSL.debug` in the user settings.

The server fails to start with a segmentation fault #

You can help us investigate this problem by sending us the core dump file. To get the core dump file, follow these steps:

In a Windows command prompt:

- Run `code --locate-extension ms-vscode-remote.remote-wsl` to determine the Remote-WSL extension folder.
- `cd` to the path that is returned.
- Open the `wslServer.sh` script with VS Code, `code .\scripts\wslServer.sh`.
- On the 3rd last line (before `export VSCODE_AGENT_FOLDER="$HOME/$DATAFOLDER"`), add `ulimit -C unlimited`.
- Start the Remote-WSL window running the remote server and wait for the segmentation fault.

The core file will be in the Remote-WSL extension folder from above.

I see EACCESS: permission denied error trying to rename a folder in the open workspace #

This is a known problem with the WSL file system implementation (Microsoft/WSL#3395 (<https://github.com/microsoft/WSL/issues/3395>), Microsoft/WSL#1956 (<https://github.com/microsoft/WSL/issues/1956>)) caused by the file watcher active by VS Code. The issue will only be fixed in WSL 2.

To avoid the issue, set `remote.WSL.fileWatcher.polling` to `true`. However, polling based has a performance impact for large workspaces.

For large workspace you may want to increase the polling interval, `remote.WSL.fileWatcher.pollingInterval`, and control the folders that are watched with `files.watcherExclude`.

WSL 2 (<https://docs.microsoft.com/windows/wsl/wsl2-index>) does not have that file watcher problem and is not affected by the new setting.

Resolving Git line ending issues in WSL (resulting in many modified files) #

Since Windows and Linux use different default line endings, Git may report a large number of modified files that have no differences aside from their line endings. To prevent this from happening, you can disable line-ending conversion using a `.gitattributes` file or globally on the Windows side.

Typically adding or modifying a `.gitattributes` file in your repository is the most reliable way to solve this problem. Committing this file to source control will help others and allows you to vary behaviors by repository as appropriate. For example, adding the following to `.gitattributes` file to the root of your repository will force everything to be LF, except for Windows batch files that require CRLF:

```

* text=auto eol=lf
*.{cmd,[cC][mM][dD]} text eol=crlf
*.{bat,[bB][aA][tT]} text eol=crlf
  
```

Note that this works in **Git v2.10+**, so if you are running into problems, be sure you've got a recent Git client installed. You can add other file types in your repository that require CRLF to this same file.

If you would prefer to still always upload Unix-style line endings (LF), you can use the `input` option.

```
git config --global core.autocrlf input
```

If you'd prefer to disable line-ending conversion entirely, run the following instead:

```
git config --global core.autocrlf false
```

Finally, you may need to clone the repository again for these settings to take effect.

Sharing Git credentials between Windows and WSL #

If you use HTTPS to clone your repositories and **have a credential helper configured** (<https://help.github.com/articles/caching-your-github-password-in-git>) in Windows, you can share this with WSL so that passwords you enter are persisted on both sides. (Note that this does not apply to using SSH keys.)

Just follow these steps:

1. Configure the credential manager on Windows by running the following in a **Windows command prompt** or **PowerShell**:

```
git config --global credential.helper wincred
```

2. Configure WSL to use the same credential helper, but running the following in a **WSL terminal**:

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

Any password you enter when working with Git on the Windows side will now be available to WSL and vice versa.

Resolving hangs when doing a Git push or sync from WSL #

If you clone a Git repository using SSH and your SSH key has a passphrase, VS Code's pull and sync features may hang when running remotely.

Either use an SSH key without a passphrase, clone using HTTPS, or run `git push` from the command line to work around the issue.

GitHub Codespaces tips #

For tips and questions about GitHub Codespaces (<https://github.com/features/codespaces>), see the GitHub Codespaces documentation (<https://docs.github.com/github/developing-online-with-codespaces>). You can also check out the known web limitations and adaptations ([/docs/remote/codespaces#_known-limitations-and-adaptations](https://docs.github.com/remote/codespaces#_known-limitations-and-adaptations)) that may impact your Codespaces.

Extension tips #

While many extensions will work unmodified, there are a few issues that can prevent certain features from working as expected. In some cases, you can use another command to work around the issue, while in others, the extension may need to be modified. This section provides a quick reference for common issues and tips on resolving them. You can also refer to the main extension article on Supporting Remote Development (</api/advanced-topics/remote-extensions>) for an in-depth guide on modifying extensions to support remote extension hosts.

Resolving errors about missing dependencies #

Some extensions rely on libraries not found in the basic install of certain WSL Linux distributions. You can add additional libraries into your Linux distribution by using its package manager. For Ubuntu and Debian based distributions, run `sudo apt-get install <package>` to install the needed libraries. Check the documentation for your extension or the runtime that is mentioned in the error message for additional installation details.

Local absolute path settings fail when applied remotely #

VS Code's local user settings are reused when you connect to a remote endpoint. While this keeps your user experience consistent, you may need to vary absolute path settings between your local machine and each host / container / WSL since the target locations are different.

Resolution: You can set endpoint-specific settings after you connect to a remote endpoint by running the **Preferences: Open Remote Settings** command from the Command Palette (`F1`) or by selecting the **Remote** tab in the Settings editor. These settings will override any local settings you have in place whenever you connect.

Need to install local VSIX on remote endpoint #

Sometimes you want to install a local VSIX on a remote machine, either during development or when an extension author asks you to try out a fix.

Resolution: Once you have connected to an SSH host, container, or WSL, you can install the VSIX the same way you would locally. Run the **Extensions: Install from VSIX...** command from the Command Palette (`F1`). You may also want to add `"extensions.autoUpdate": false` to `settings.json` to prevent auto-updating to the latest Marketplace version. See Supporting Remote Development (</api/advanced-topics/remote-extensions>) for more information on developing and testing extensions in a remote environment.

Browser does not open locally #

Some extensions use external node modules or custom code to launch a browser window. Unfortunately, this may cause the extension to launch the browser remotely instead of locally.

Resolution: The extension can use the `vscode.env.openExternal` API to resolve this problem. See the extension author's guide (</api/advanced-topics/remote-extensions#opening-something-in-a-local-browser-or-application>) for details.

Clipboard does not work #

Some extensions use node modules like `clipboardy` to integrate with the clipboard. Unfortunately, this may cause the extension to incorrectly integrate with the clipboard on the remote side.

Resolution: The extension can switch to the VS Code clipboard API to resolve the problem. See the extension author's guide (/api/advanced-topics/remote-extensions#using-the-clipboard) for details.

Cannot access local web server from browser or application #

When working inside a container, SSH host, or through GitHub Codespaces, the port the browser is connecting to may be blocked.

Resolution: Extensions can use the `vscode.env.openExternal` or `vscode.env.asExternalUri` APIs (which automatically forwards localhost ports) to resolve this problem. See the extension author's guide (/api/advanced-topics/remote-extensions#opening-something-in-a-local-browser-or-application) for details. As a workaround, use the **Forward a Port** command to do so manually.

Webview contents do not appear #

If the extension's webview content uses an `iframe` to connect to a local web server, the port the webview is connecting to may be blocked. In addition, if the extension hard codes `vscode-resource://` URLs instead of using `asWebviewUri`, content may not appear in the Codespaces browser editor.

Resolution: The extension can use the `webview.asWebviewUri` to resolve issues with `vscode-resource://` URLs.

If ports are being blocked, the best approach is to instead use the webview message passing (/api/extension-guides/webview#scripts-and-message-passing) API. As a workaround, `vscode.env.asExternalUri` can be used allow the webview to connect to spawned localhost web servers from VS Code. However, this is currently blocked for the Codespaces browser-based editor (only) by MicrosoftDocs/vscode-spaces#11 (<https://github.com/MicrosoftDocs/vscode-spaces/issues/11>). See the extension author's guide (/api/advanced-topics/remote-extensions#workarounds-for-using-localhost-from-a-webview) for details on the workaround.

Blocked localhost ports #

If you are trying to connect to a localhost port from an external application, the port may be blocked.

Resolution: VS Code 1.40 introduced a new `vscode.env.asExternalUri` API for extensions to programmatically forward arbitrary ports. See the extension author's guide (/api/advanced-topics/remote-extensions#forwarding-localhost) for details. As a workaround, you can use the **Forward a Port** command to do so manually.

Errors storing extension data #

Extensions may try to persist global data by looking for the `~/.config/Code` folder on Linux. This folder may not exist, which can cause the extension to throw errors like `ENOENT: no such file or directory, open '/root/.config/Code/User/filename-goes-here'`.

Resolution: Extensions can use the `context.globalStoragePath` or `context.storagePath` property to resolve this problem. See the extension author's guide (/api/advanced-topics/remote-extensions#persisting-extension-data-or-state) for details.

Cannot sign in / have to sign in each time I connect to a new endpoint #

Extensions that require sign in may persist secrets using their own code. This code can fail due to missing dependencies. Even if it succeeds, the secrets will be stored remotely, which means you have to sign in for every new endpoint.

Resolution: Extensions can use the `keytar` node module to solve this problem. See the extension author's guide (/api/advanced-topics/remote-extensions#persisting-secrets) for details.

An incompatible extension prevents VS Code from connecting #

If an incompatible extension has been installed on a remote host, container, or in WSL, we have seen instances where the VS Code Server hangs or crashes due to the incompatibility. If the extension activates right away, this can prevent you from connecting and being able to uninstall the extension.

Resolution: Manually delete the remote extensions folder by following these steps:

1. For containers, ensure your `devcontainer.json` no longer includes a reference to the faulty extension.
2. Next, use a separate terminal / command prompt to connect to the remote host, container, or WSL.
 - If SSH or WSL, connect to the environment accordingly (run `ssh` to connect to the server or open WSL terminal).
 - If using a container, identify the container ID by calling `docker ps -a` and looking through the list for an image with the correct name. If the container is stopped, run `docker run -it <id> /bin/sh`. If it is running, run `docker exec -it <id> /bin/sh`.
3. Once you are connected, run `rm -rf ~/.vscode-server/extensions` for VS Code stable and/or `rm -rf ~/.vscode-server-insiders/extensions` for VS Code Insiders to remove all extensions.

Extensions that ship or acquire pre-built native modules fail #

Native modules bundled with (or dynamically acquired for) a VS Code extension must be recompiled using Electron's `electron-rebuild` (<https://electronjs.org/docs/tutorial/using-native-node-modules>). However, VS Code Server runs a standard (non-Electron) version of Node.js, which can cause binaries to fail when used remotely.

Resolution: Extensions need to be modified to solve this problem. They will need to include (or dynamically acquire) both sets of binaries (Electron and standard Node.js) for the "modules" version in Node.js that VS Code ships and then check to see if `context.executionContext === vscode.ExtensionExecutionContext.Remote` in their activation function to set up the correct binaries. See the extension author's guide (</api/advanced-topics/remote-extensions#using-native-node.js-modules>) for details.

Extension only fails on non-x86_64 hosts or Alpine Linux #

If an extension works on Debian 9+, Ubuntu 16.04+, or RHEL / CentOS 7+ remote SSH hosts, containers, or WSL, but fails on supported non-x86_64 hosts (for example, ARMv7l) or Alpine Linux containers, the extension may only include native code or runtimes that do not support these platforms. For example, the extensions may only include x86_64 compiled versions of native modules or runtimes. For Alpine Linux, the included native code or runtimes may not work due to fundamental differences (<https://wiki.musl-libc.org/functional-differences-from-glibc.html>) between how `libc` is implemented in Alpine Linux (`musl`) and other distributions (`glibc`).

Resolution: Extensions will need to opt-in to supporting these platforms by compiling / including binaries for these additional targets. It is important to note that some third-party npm modules may also include native code that can cause this problem. So, in some cases you may need to work with the npm module author to add additional compilation targets. See the extension author's guide (</api/advanced-topics/remote-extensions#supporting-nonx8664-hosts-or-alpine-linux-containers>) for details.

Extensions fail due to missing modules #

Extensions that rely on Electron or VS Code base modules (not exposed by the extension API) without providing a fallback can fail when running remotely. You may see errors in the Developer Tools console like `original-fs` not being found.

Resolution: Remove the dependency on an Electron module or provide a fallback. See the extension author's guide (</api/advanced-topics/remote-extensions#avoid-using-electron-modules>) for details.

Cannot access / transfer remote workspace files to local machines #

Extensions that open workspace files in external applications may encounter errors because the external application cannot directly access the remote files.

Resolution: If you create a "UI" extension designed to run locally, you can use the `vscode.workspace.fs` API to interact with the remote workspace filesystem. You can then make this a dependency of your "Workspace" extension and invoke it using a command as needed. See the extension author's guide (</api/advanced-topics/remote-extensions>) for details on different types of extensions and how to use commands to communicate between them.

Cannot access attached device from extension #

Extensions that access locally attached devices will be unable to connect to them when running remotely.

Resolution: None currently. We are investigating the best approach to solve this problem.

Questions and feedback #

Reporting issues #

If you run into an issue with one of the remote development extensions, it's important to collect the correct logs so that we'll be able to help diagnose your issue (<https://aka.ms/vscode-remote/issues/new>).

Each remote extension has a command to view its logs.

You can get the Remote - SSH extension logs with **Remote-SSH: Show Log** from the Command Palette (`F1`). When reporting Remote - SSH issues, please also verify if you're able to SSH into your machine from an external terminal (not using Remote - SSH).

Similarly, you can get the Remote - Containers extension logs with **Remote-Containers: Show Log**.

Like the two above, you can get the Remote - WSL logs with **Remote WSL: Show Log**. Also check whether your issue is being tracked upstream in the WSL repo (<https://github.com/microsoft/WSL/issues>) (and is not due to the Remote - WSL extension).

If you're experiencing issues using other extensions remotely (for example, other extensions aren't loading or installing properly in a remote context), it's helpful to grab the log from the **Remote Extension Host** output channel (**Output: Focus on Output View**), and select **Log (Remote Extension Host)** from the dropdown.

Note: If you only see **Log (Extension Host)**, this is the local extension host, and the remote extension host didn't launch. This is because the log channel is created only after the log file is created, so if the remote extension host does not launch, the remote extension host log file was not created and is not shown in the Output view. This is still helpful information to include in your issue.

Remote question and feedback resources #

We have a variety of other remote resources:

- See Remote Development FAQ (</docs/remote/faq>).
- Search on Stack Overflow (<https://stackoverflow.com/questions/tagged/vscode-remote>).
- Add a feature request (<https://aka.ms/vscode-remote/feature-requests>) or report a problem (<https://aka.ms/vscode-remote/issues/new>).

Was this documentation helpful?

Yes

No

9/1/2022

IN THIS ARTICLE

{ SSH tips


Container tips

WSL tips


GitHub Codespaces tips


Extension tips

Questions and feedback


 [this link](https://twitter.com/intent/tweet?original_referer=https://code.visualstudio.com/docs/remote/troubleshooting&ref_src=twsrc%5Etfw&text=Visual%20Studio%20Code%20Remote%20Development%20Ti) (https://twitter.com/intent/tweet?original_referer=https://code.visualstudio.com/docs/remote/troubleshooting&ref_src=twsrc%5Etfw&text=Visual%20Studio%20Code%20Remote%20Development%20Ti


 [Subscribe\(/feed.xml\)](#)

 [Ask questions\(https://stackoverflow.com/questions/tagged/vscode\)](https://stackoverflow.com/questions/tagged/vscode)

 [Follow @code\(https://go.microsoft.com/fwlink/?LinkID=533687\)](https://go.microsoft.com/fwlink/?LinkID=533687)

 [Request features\(https://go.microsoft.com/fwlink/?LinkID=533482\)](https://go.microsoft.com/fwlink/?LinkID=533482)

 [Report issues\(https://www.github.com/Microsoft/vscode/issues\)](https://www.github.com/Microsoft/vscode/issues)

 [Watch videos\(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w\)](https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

Hello from Seattle. Follow @code (https://go.microsoft.com/fwlink/?LinkID=533687)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d) Privacy (https://privacy.microsoft.com/privacystatement)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

 Microsoft (https://www.microsoft.com)

© 2022 Microsoft