Lowell Vanderpool

TECH SAVVY PRODUCTIONS

SSH for Windows

**CONTACT US:**

[mrvanderpool@techsavvyproductions.com](mailto:mrvanderpool@techsavvyproductions.com)

**SUPPORT US:**

If you would like to support the channel, Join our channel membership, it’s $2.99/month (less than a Starbucks coffee); see the “Join” button on our channel homepage.



**OR**

Subscribe to the channel as it helps our channel perform better on YouTube’s algorithm.

**SOCIAL MEDIA AND WEBSITE:**

Check out our **Website**: <https://www.techsavvyproductions.com>

Mr.V **Linkedin**: https://www.linkedin.com/in/lowell-vanderpool-57970623/

**** We translate subtitles on our videos into many languages:

Two free ways to support our channel, like the video if it helped you better understand technology or the topic, and subscribe.

A picture containing graphical user interface

Description automatically generated A picture containing text

Description automatically generated

[OpenSSH for Windows overview](#_OpenSSH_for_Windows)

[OpenSSH Server configuration for Windows Server and Windows](#_OpenSSH_Server_configuration)

[Key-based authentication in OpenSSH for Windows](#_Key-based_authentication_in_1)

[Extensive SSH resource online](https://docstore.mik.ua/orelly/networking_2ndEd/ssh/index.htm)

[Manuals for OpenSSH components](#_Manuals_for_OpenSSH)

[Install Win32-OpenSSH using MSI](#_Install_Win32-OpenSSH_using)

[Configure OpenSSH Server service using “sshd\_config” file](#_Configure_OpenSSH_Server)

[How to Enable and Configure SSH Server on Windows with OpenSSH?](#_How_to_Enable)

[Configuring SFTP (SSH FTP) Server on older versions of Windows/Server](#_Configuring_SFTP_(SSH)

[Configuring SSH Public Key Authentication on Windows](#_Configuring_SSH_Public) very detailed

[The Ultimate Guide to Installing OpenSSH on Windows](#_The_Ultimate_Guide)

[PowerShell remoting over SSH](#_PowerShell_remoting_over)

[Just Enough Administration](#_Just_Enough_Administration)

[How to Use PuTTY on Windows](#_How_to_Use)

[Chapter 12. Troubleshooting and FAQ Contents:](#_Chapter_12._Troubleshooting)

[Excellent online book on SSH and the entire Chapter 12 on Troubleshooting](https://docstore.mik.ua/orelly/networking_2ndEd/ssh/index.htm)

[Secure Shell Wiki](#_Secure_Shell_Wiki)

# OpenSSH for Windows overview

Applies to Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)

OpenSSH is the open-source version of the Secure Shell (SSH) tools used by administrators of Linux and other non-Windows for cross-platform management of remote systems. OpenSSH has been added to Windows (as of autumn 2018), and is included in Windows Server and Windows client.

SSH is based on a client-server architecture where the system the user is working on is the client and the remote system being managed is the server. OpenSSH includes a range of components and tools designed to provide a secure and straightforward approach to remote system administration.

Graphical user interface, diagram

Description automatically generated

**OpenSSH for Windows has the below commands built in.**

* ssh is the SSH client component that runs on the user's local system
* sshd is the SSH server component that must be running on the system being managed remotely
* ssh-keygen generates, manages and converts authentication keys for SSH
* ssh-agent stores private keys used for public key authentication
* ssh-add adds private keys to the list allowed by the server
* ssh-keyscan aids in collecting the public SSH host keys from hosts
* sftp is the service that provides the Secure File Transfer Protocol, and runs over SSH
* scp is a file copy utility that runs on SSH

**Tip**

The documentation focuses on how OpenSSH is used on Windows, including:

* installation
* Windows-specific configuration
* Commands
* and use cases.

Additional detailed documentation for common OpenSSH features is available online at [OpenSSH.com](https://www.openssh.com/manual.html).

Feedback on Windows OpenSSH is welcomed and can be provided by creating GitHub issues in our [OpenSSH GitHub repo](https://github.com/PowerShell/openssh-portable). The [OpenSSH open source project](https://www.openssh.com) is managed by developers at the OpenBSD Project. The Microsoft fork of this project is in [GitHub](https://github.com/PowerShell/openssh-portable).

Of making many books there is no end, and much study wearies the body.

Now all has been heard;  
    here is the conclusion of the matter:  
Fear God and keep his commandments,  
    for this is the duty of all mankind.  
For God will bring every deed into judgment,  
    including every hidden thing,  
    whether it is good or evil.

# OpenSSH Server configuration for Windows Server and Windows

Applies to Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)

This article covers the Windows-specific configuration for OpenSSH Server (sshd).

OpenSSH maintains detailed documentation for configuration options online at [OpenSSH.com](https://www.openssh.com/manual.html), which isn't duplicated in this documentation set.

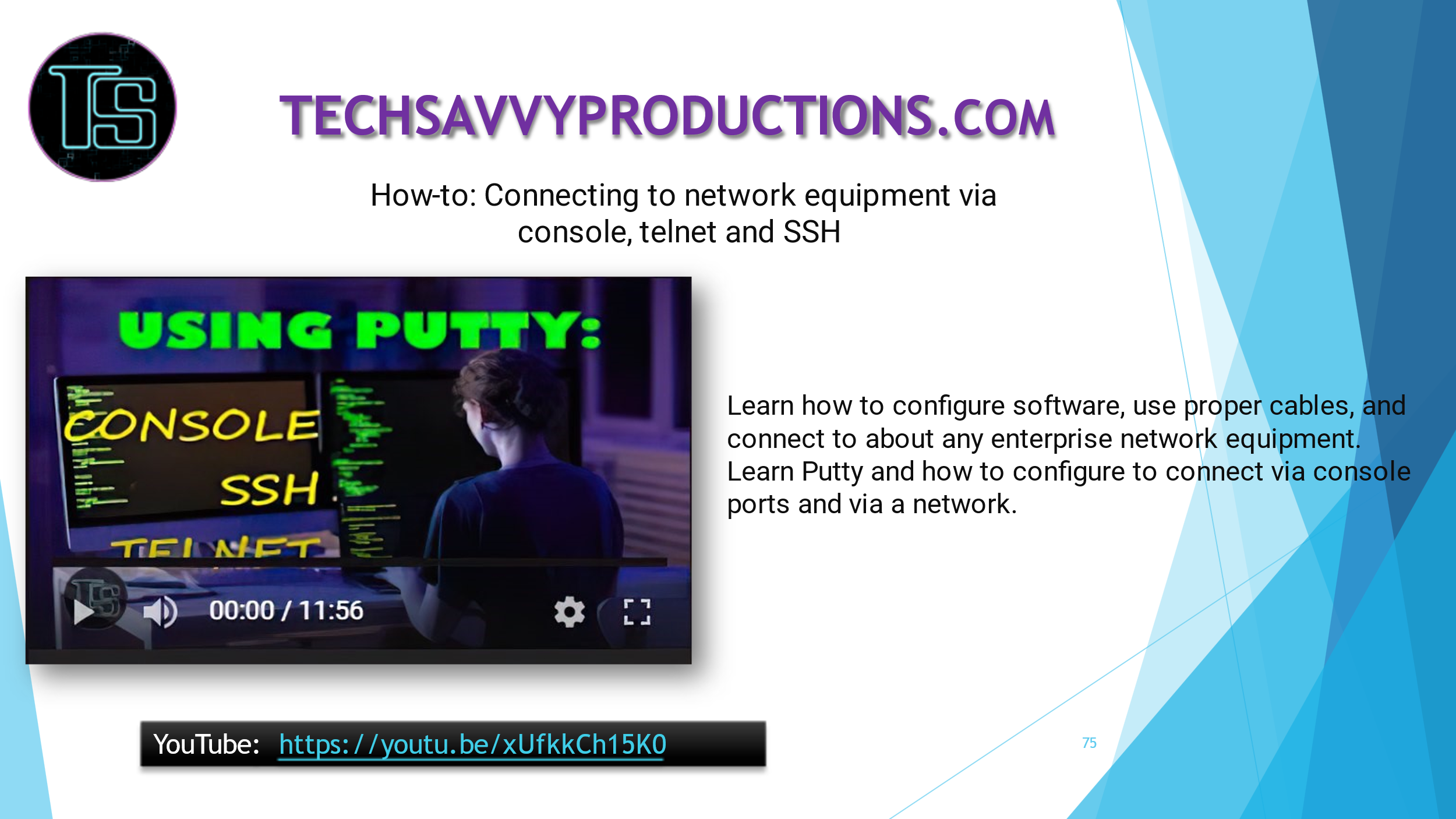
## OpenSSH configuration files

OpenSSH has configuration files for both server and client settings. OpenSSH is open-source and is added to Windows Server and Windows Client operating systems, starting with Windows Server 2019 and Windows 10 (build 1809).

* As a result, open-source documentation for OpenSSH configuration files isn't repeated here.
* Client configuration files and can be found on the [ssh\_config manual page](https://man.openbsd.org/ssh_config)
* and for OpenSSH Server configuration files can be found on the [sshd\_config manual page](https://man.openbsd.org/sshd_config).

Open SSH Server (sshd) reads configuration data from %programdata%\ssh\sshd\_config by default,

* or a different configuration file may be specified by launching sshd.exe with the -f parameter.
* If the file is absent, sshd generates one with the default configuration when the service is started.



<https://youtu.be/xUfkkCh15K0>

In Windows, the OpenSSH Client (ssh) reads configuration data from a configuration file in the following order:

1. By launching ssh.exe with the -F parameter, specifying a path to a configuration file and an entry name from that file.
2. A user's configuration file at %userprofile%\.ssh\config.
3. The system-wide configuration file at %programdata%\ssh\ssh\_config.

## Configuring the default shell for OpenSSH in Windows

The default command shell provides the experience a user sees when connecting to the server using SSH. The initial default Windows is the Windows Command shell (cmd.exe). Windows also includes PowerShell, and third-party command shells are also available for Windows and may be configured as the default shell for a server.

To set the default command shell, first confirm that the OpenSSH installation folder is on the system path.

For Windows, the default installation folder is %systemdrive%\Windows\System32\openssh. The following command shows the current path setting, and adds the default OpenSSH installation folder to it.

| **Command shell** | **Command to use** |
| --- | --- |
| Command | path |
| PowerShell | $env:path |

Configuring the default ssh shell is done in the Windows registry by adding the full path to the shell executable to HKEY\_LOCAL\_MACHINE\SOFTWARE\OpenSSH in the string value DefaultShell.

As an example, the following elevated PowerShell command sets the default shell to be powershell.exe:

PowerShell

New-ItemProperty -Path "HKLM:\SOFTWARE\OpenSSH" -Name DefaultShell -Value "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -PropertyType String -Force

## 

## <https://youtu.be/8lBOokP2ggw>

## Windows Configurations in sshd\_config

In Windows, sshd reads configuration data from %programdata%\ssh\sshd\_config by default,

* or a different configuration file may be specified by launching sshd.exe with the -f parameter.
* If the file is absent, sshd generates one with the default configuration when the service is started.

The elements listed below provide Windows-specific configuration possible through entries in sshd\_config.

There are other configuration settings possible that aren't listed here, as they're covered in detail in the online [Win32 OpenSSH documentation](https://github.com/powershell/win32-openssh/wiki).

**Tip**

The OpenSSH Server (sshd) reads the configuration file when the service starts. Any changes to the configuration file requires the service to be restarted.

### AllowGroups, AllowUsers, DenyGroups, DenyUsers

Controlling which users and groups can connect to the server is done using the:

* AllowGroups,
* AllowUsers,
* DenyGroups,
* and DenyUsers directives.

The allow/deny directives are processed in the following order:

* DenyUsers,
* AllowUsers,
* DenyGroups,
* and finally AllowGroups.

All account names must be specified in lower case.

* For more information about PATTERNS and wildcard in the ssh\_config, see the [sshd\_config OpenBSD manual page](https://man.openbsd.org/ssh_config#PATTERNS).
* When configuring user/group based rules with a domain user or group, use the following format: user?domain\*.
* Windows allows multiple formats for specifying domain principals, but many conflict with standard Linux patterns.
* For that reason, \* is added to cover FQDNs. Also, this approach uses "?", instead of @, to avoid conflicts with the username@host format.
* Work group users/groups and internet-connected accounts are always resolved to their local account name (no domain part, similar to standard Unix names).
* Domain users and groups are strictly resolved to [NameSamCompatible](https://learn.microsoft.com/en-us/windows/desktop/api/secext/ne-secext-extended_name_format) format - domain\_short\_name\user\_name. All user/group based configuration rules need to adhere to this format.

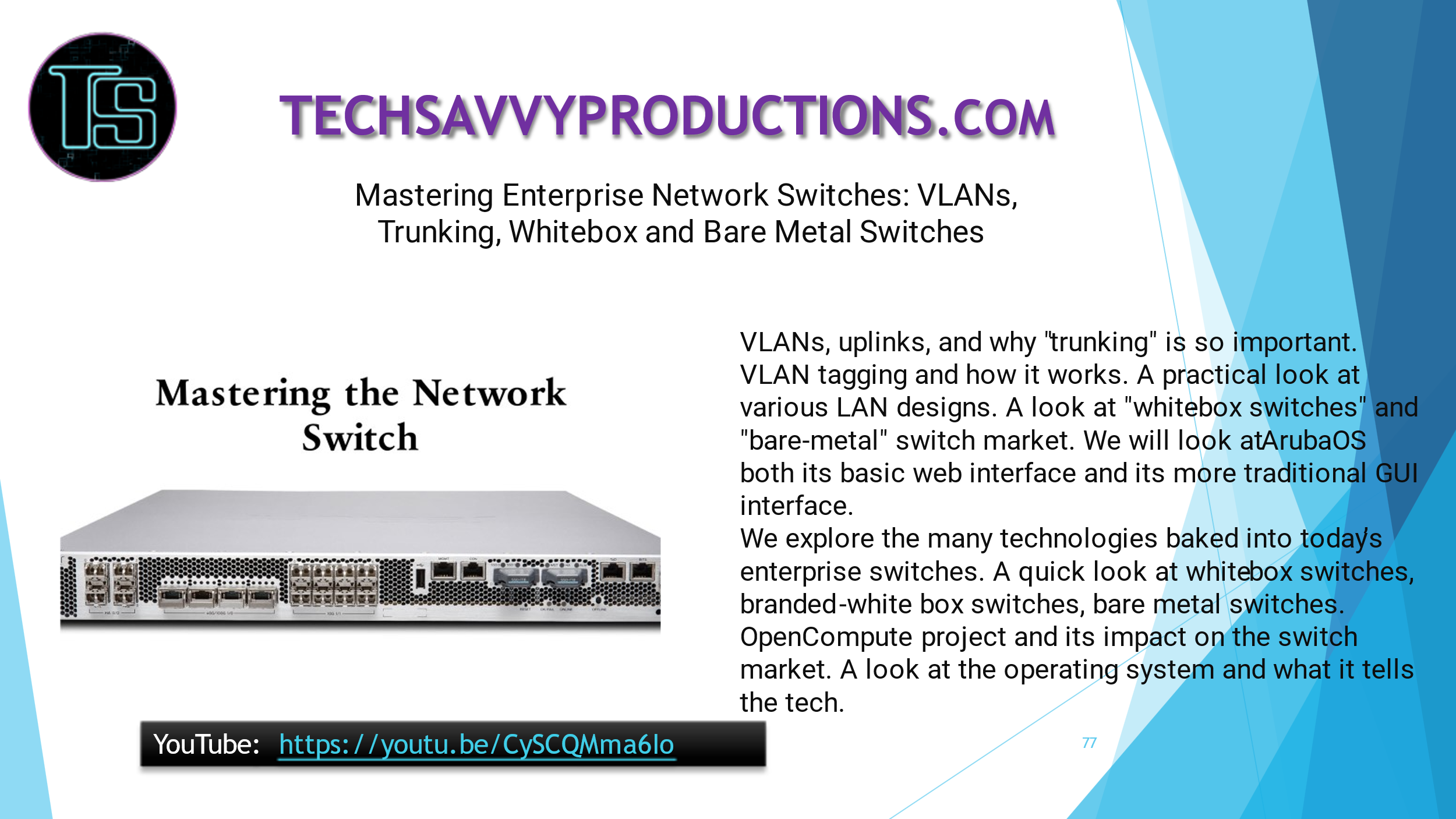
The following example (shown below) denies contoso\admin from the host 192.168.2.23, and blocks all users from contoso domain. It also allows users who are a member of the contoso\sshusers and contoso\serveroperators groups.

sshd\_config

DenyUsers contoso\admin@192.168.2.23

DenyUsers contoso\\*

AllowGroups contoso\sshusers contoso\serveroperators



<https://youtu.be/CySCQMma6Io>

The example below allow the user localusers to sign-in from the host 192.168.2.23 and allows members of the group sshusers.

sshd\_config

AllowUsers localuser@192.168.2.23

AllowGroups sshusers

### AuthenticationMethods

For **Windows OpenSSH**, the only available authentication methods are password and publickey.

Important

* Authentication using an Azure AD account is not currently supported.

### AuthorizedKeysFile

The default is .ssh/authorized\_keys.

If the path isn't absolute, it's taken relative to user's home directory (or profile image path), for example, C:\Users\username.

If the user belongs to the administrator group, %programdata%/ssh/administrators\_authorized\_keys is used instead.

Tip

The administrators\_authorized\_keys file must:

* only have permission entries for the NT Authority\SYSTEM account (full control)
* and BUILTIN\Administrators security group.

The NT Authority\SYSTEM account must be granted full control. The BUILTIN\Administrators security group is required for administrators to manage the authorized keys, you can choose the required access. To grant permissions you can open an elevated PowerShell prompt, and running the command

icacls.exe "C:\ProgramData\ssh\administrators\_authorized\_keys" /inheritance:r /grant "Administrators:F" /grant "SYSTEM:F"

### ChrootDirectory (Support added in v7.7.0.0)

This directive is only supported with sftp sessions. A remote session into cmd.exe wouldn't honor the ChrootDirectory. To set up a sftp-only chroot server, set ForceCommand to internal-sftp. You may also set up scp with chroot, by implementing a custom shell that would only allow scp and sftp.

### **GSSAPIAuthentication**

The GSSAPIAuthentication configuration argument specifies whether GSSAPI based user authentication is allowed. The default for GSSAPIAuthentication is no.

GSSAPI authentication also requires the use of the -K switch specifying the hostname when using the OpenSSH client. Alternatively, you can create a corresponding entry in the SSH client configuration. In Windows, the OpenSSH client reads configuration data from %userprofile%.ssh\config by default.

You can see an example GSSAPI OpenSSH client configuration below.

config

# Specify a set of configuration arguments for a host matching the pattern SERVER01.contoso.com

# Patterns are case sensitive

Host SERVER01.contoso.com

# Enables GSSAPI authentication

GSSAPIAuthentication yes

# Forward (delegate) credentials to the server.

GSSAPIDelegateCredentials yes

Important

GSSAPI is only available starting in Windows Server 2022, Windows 11, and Windows 10 v1803.

### HostKey

The defaults are:

sshd\_config

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_rsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_dsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_ecdsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_ed25519\_key

If the defaults aren't present, sshd automatically generates them on a service start.

### Match

Matches conditions using one or more criteria. Upon a match, the subsequent configuration arguments are applied. Matches uses the pattern rules covered in the [AllowGroups, AllowUsers, DenyGroups, DenyUsers](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_server_configuration#allowgroups-allowusers-denygroups-denyusers) section. User and group names should be in lower case.

### PermitRootLogin

Not applicable in Windows. To prevent administrators signing in, use Administrators with DenyGroups directive.

### SyslogFacility

If you need file based logging, use LOCAL0. Logs are generated under %programdata%\ssh\logs. For any other value, including the default value, AUTH directs logging to ETW. For more info, see [Logging Facilities in Windows](https://github.com/PowerShell/Win32-OpenSSH/wiki/Logging-Facilities).

### Configuration arguments

The following configuration argument is available starting in Windows Server 2022, Windows 11, and Windows 10 v1803

* GSSAPIAuthentication

The following configuration arguments aren't available in the OpenSSH version that ships in Windows Server and the Windows client:

* AcceptEnv
* AllowStreamLocalForwarding
* AuthorizedKeysCommand
* AuthorizedKeysCommandUser
* AuthorizedPrincipalsCommand
* AuthorizedPrincipalsCommandUser
* Compression
* ExposeAuthInfo
* GSSAPICleanupCredentials
* GSSAPIStrictAcceptorCheck
* HostbasedAcceptedKeyTypes
* HostbasedAuthentication
* HostbasedUsesNameFromPacketOnly
* IgnoreRhosts
* IgnoreUserKnownHosts
* KbdInteractiveAuthentication
* KerberosAuthentication
* KerberosGetAFSToken
* KerberosOrLocalPasswd
* KerberosTicketCleanup
* PermitTunnel
* PermitUserEnvironment
* PermitUserRC
* PidFile
* PrintLastLog
* PrintMotd
* RDomain
* StreamLocalBindMask
* StreamLocalBindUnlink
* StrictModes
* X11DisplayOffset
* X11Forwarding
* X11UseLocalhost
* XAuthLocation

Products

[Web Hosting](https://www.hostwinds.com/hosting/shared)

# 

# eKey-based authentication in OpenSSH for Windows

Applies to Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)

Most authentication in Windows environments is done with a username-password pair, which works well for systems that share a common domain. When working across domains, such as between on-premises and cloud-hosted systems, it becomes vulnerable to brute force intrusions.

By comparison, Linux environments commonly use public-key/private-key pairs to drive authentication that doesn't require the use of guessable passwords.

OpenSSH client includes tools to help support key based authentication, specifically:

* **ssh-keygen** for generating secure keys
* **ssh-agent** and **ssh-add** for securely storing private keys
* **scp** and **sftp** to securely copy public key files during initial use of a server

Graphical user interface, text, application

Description automatically generated

This document provides an overview of how to use these tools on Windows to begin using key-based authentication with SSH. If you're unfamiliar with SSH key management, we strongly recommend you review [NIST document IR 7966](http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf) titled "Security of Interactive and Automated Access Management Using Secure Shell (SSH)".

**About key pairs**

Key pairs refer to the public and private key files that are used by certain authentication protocols.

SSH public key authentication uses asymmetric cryptographic algorithms to generate two key files – one "private" and the other "public".

The private key files are the equivalent of a password, and should stay protected under all circumstances. If someone acquires your private key, they can sign in as you to any SSH server you have access to.

The public key is what is placed on the SSH server, and may be shared without compromising the private key.

Key based authentication enables the SSH server and client to compare the public key for a user name provided against the private key. If the server-side public key can't be validated against the client-side private key, authentication fails.

Multi-factor authentication may be implemented with key pairs by entering a passphrase when the key pair is generated (see [user key generation](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_keymanagement#user-key-generation) below). The user will be prompted for the passphrase during authentication. The passphrase is used along with the presence of the private key on the SSH client to authenticate the user.

**Important**

A remote session opened via key based authentication does not have associated user credentials and hence is not capable of outbound authentication as the user, this is by design.

He has shown you, O mortal, what is good.

And what does the Lord require of you?

To act justly and to love mercy

and to walk humbly with your God.

**Host key generation**

Public keys have specific ACL requirements that, on Windows, equate to only allowing access to administrators and System.

On first use of sshd, the key pair for the host will be automatically generated. See below:

Graphical user interface, application

Description automatically generated

**Important**

You need to have OpenSSH Server installed first. Please see [Getting started with OpenSSH](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse).

By default the sshd service is set to start manually. To start it each time the server is rebooted, run the following commands from an elevated PowerShell prompt on your server:

**PowerShell**

# Set the sshd service to be started automatically

Get-Service -Name sshd | Set-Service -StartupType Automatic

# Now start the sshd service

Start-Service sshd

Since there's no user associated with the sshd service, the host keys are stored under C:\ProgramData\ssh.

**User key generation**

To use key-based authentication, you first need to generate public/private key pairs for your client. ssh-keygen.exe is used to generate key files and the algorithms DSA, RSA, ECDSA, or Ed25519 can be specified. If no algorithm is specified, RSA is used. A strong algorithm and key length should be used, such as Ed25519 in this example.

To generate key files using the Ed25519 algorithm, run the following command from a PowerShell or cmd prompt on your client:

PowerShell

ssh-keygen -t ed25519

The output from the command should display the following output (where "username" is replaced by your username):

Output

Generating public/private ed25519 key pair.

Enter file in which to save the key (C:\Users\username/.ssh/id\_ed25519):

You can press Enter to accept the default, or specify a path and/or filename where you would like your keys to be generated. At this point, you'll be prompted to use a passphrase to encrypt your private key files. The passphrase can be empty but it's not recommended. The passphrase works with the key file to provide two-factor authentication. For this example, we're leaving the passphrase empty.

Output

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in C:\Users\username/.ssh/id\_ed25519.

Your public key has been saved in C:\Users\username/.ssh/id\_ed25519.pub.

The key fingerprint is:

SHA256:OIzc1yE7joL2Bzy8!gS0j8eGK7bYaH1FmF3sDuMeSj8 username@LOCAL-HOSTNAME

The key's randomart image is:

+--[ED25519 256]--+

| . |

| o |

| . + + . |

| o B \* = . |

| o= B S . |

| .=B O o |

| + =+% o |

| \*oo.O.E |

|+.o+=o. . |

+----[SHA256]-----+

Now you have a public/private ed25519 key pair in the location specified. The .pub files are public keys, and files without an extension are private keys:

Output

Mode LastWriteTime Length Name

---- ------------- ------ ----

-a---- 6/3/2021 2:55 PM 464 ed25519

-a---- 6/3/2021 2:55 PM 103 ed25519.pub

Remember that private key files are the equivalent of a password should be protected the same way you protect your password.

Use ssh-agent to securely store the private keys within a Windows security context, associated with your Windows account.

To start the ssh-agent service each time your computer is rebooted, and use ssh-add to store the private key run the following commands from an elevated PowerShell prompt on your server:

PowerShell

# By default the ssh-agent service is disabled. Configure it to start automatically.

# Make sure you're running as an Administrator.

Get-Service ssh-agent | Set-Service -StartupType Automatic

# Start the service

Start-Service ssh-agent

# This should return a status of Running

Get-Service ssh-agent

Text

Description automatically generated

# Now load your key files into ssh-agent

ssh-add $env:USERPROFILE\.ssh\id\_ed25519

Text

Description automatically generated

Once you've added the key to the ssh-agent on your client, the ssh-agent will automatically retrieve the local private key and pass it to your SSH client.

**Important**

It is strongly recommended that you back up your private key to a secure location, then delete it from the local system, *after* adding it to ssh-agent. The private key cannot be retrieved from the agent providing a strong algorithm has been used, such as Ed25519 in this example. If you lose access to the private key, you will have to create a new key pair and update the public key on all systems you interact with.

**Deploying the public key**

To use the user key that was created above, the contents of your public key *\.ssh\id\_ed25519.pub* needs to be placed on the server into a text file. The name and location of the file depends on whether the user account is a member of the local administrators group or a standard user account. The following sections cover both standard and administrative users.

**Standard user**

The contents of your public key *\.ssh\id\_ed25519.pub* needs to be placed on the server into a text file called authorized\_keys in *C:\Users\username\.ssh\*

You can copy your public key using the OpenSSH scp secure file-transfer utility, or using a PowerShell to write the key to the file.

The example below copies the public key to the server (where "username" is replaced by your username). You'll need to use the password for the user account for the server initially.

PowerShell

# Get the public key file generated previously on your client

$authorizedKey = Get-Content -Path $env:USERPROFILE\.ssh\id\_ed25519.pub

# Generate the PowerShell to be run remote that will copy the public key file generated previously on your client to the authorized\_keys file on your server

$remotePowershell = "powershell New-Item -Force -ItemType Directory -Path $env:USERPROFILE\.ssh; Add-Content -Force -Path $env:USERPROFILE\.ssh\authorized\_keys -Value '$authorizedKey'"

# Connect to your server and run the PowerShell using the $remotePowerShell variable

ssh username@domain1@contoso.com $remotePowershell

Text

Description automatically generated

**Administrative user**

The contents of your public key *\.ssh\id\_ed25519.pub* needs to be placed on the server into a text file called administrators\_authorized\_keys in *C:\ProgramData\ssh\*.

You can copy your public key using the OpenSSH scp secure file-transfer utility, or using a PowerShell to write the key to the file.

The ACL on this file needs to be configured to only allow access to administrators and System.

The example below copies the public key to the server and configures the ACL (where "username" is replaced by your user name). You'll need to use the password for the user account for the server initially.

**Note**

This example shows the steps for creating the administrators\_authorized\_keys file. This only applies to administrator accounts and must be user instead of the per user file within the user's profile location.

Graphical user interface, application

Description automatically generated

**PowerShell**

# Get the public key file generated previously on your client

$authorizedKey = Get-Content -Path $env:USERPROFILE\.ssh\id\_ed25519.pub

# Generate the PowerShell to be run remote that will copy the public key file generated previously on your client to the authorized\_keys file on your server

$remotePowershell = "powershell Add-Content -Force -Path $env:ProgramData\ssh\administrators\_authorized\_keys -Value '$authorizedKey';icacls.exe ""$env:ProgramData\ssh\administrators\_authorized\_keys"" /inheritance:r /grant ""Administrators:F"" /grant ""SYSTEM:F"""

# Connect to your server and run the PowerShell using the $remotePowerShell variable

ssh username@domain1@contoso.com $remotePowershell

These steps complete the configuration required to use key-based authentication with OpenSSH on Windows. Once the example PowerShell commands have been run, the user can connect to the sshd host from any client that has the private key.

Text

Description automatically generated

Note: 1st logon you will still use your username password but after this session and you try again and create a new session “no password” will be required.

**Recommended content**

**[OpenSSH Server configuration for Windows](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_server_configuration?source=recommendations)**

Learn about the Windows-specific configuration options for OpenSSH Server on Windows Server and Windows.

**[Windows Terminal SSH](https://learn.microsoft.com/en-us/windows/terminal/tutorials/ssh?source=recommendations)**

In this tutorial, learn how to set up an SSH connection in Windows Terminal.

**[Get started with OpenSSH for Windows](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?source=recommendations)**

Learn how to install and connect to remote machines using the OpenSSH Client and Server for Windows.

**[OpenSSH for Windows overview](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview?source=recommendations)**

Overview about the OpenSSH tools used by Windows administrators for cross-platform management of remote systems.

# Manuals for OpenSSH components

* [sshd\_config](https://github.com/PowerShell/Win32-OpenSSH/wiki/sshd_config)
* [sshd](https://github.com/PowerShell/Win32-OpenSSH/wiki/sshd)
* [ssh\_config](https://man.openbsd.org/ssh_config)
* [ssh](https://man.openbsd.org/ssh)
* [ssh-keygen](https://man.openbsd.org/ssh-keygen)
* [ssh-keyscan](https://man.openbsd.org/ssh-keyscan)
* [ssh-add](https://man.openbsd.org/ssh-add)

# 

# Install Win32-OpenSSH using MSI

## Installation

# [MSI Install Instructions](https://github.com/PowerShell/Win32-OpenSSH/wiki/Install-Win32-OpenSSH-Using-MSI)

## 1. Run MSI Installer

The MSI must be run in any command prompt (cmd.exe & pwsh.exe both work), as it does not yet have a UI (coming soon).  
The MSI will install OpenSSH to the ProgramFiles\OpenSSH folder.  
The commands to run, are as follows:

* To install both the SSH Client & the SSH Server (default behavior)  
  msiexec /i <path to openssh.msi>
* To install only the SSH Client  
  msiexec /i <path to openssh.msi> ADDLOCAL=Client
* To install only the SSH Server  
  msiexec /i <path to openssh.msi> ADDLOCAL=Server
* To uninstall only the SSH Client  
  msiexec /i <path to openssh.msi> REMOVE=Client
* To uninstall only the SSH Server  
  msiexec /i <path to openssh.msi> REMOVE=Server

### Examples:

* Installing SSH Client & openssh.msi is in the working directory:  
  msiexec /i openssh.msi ADDLOCAL=Client
* Installing SSH Server & openssh.msi is in C:\users\public\downloads:  
  msiexec /i C:\users\public\downloads\openssh.msi ADDLOCAL=Server
* Uninstalling SSH Client & openssh.msi is in the working directory:  
  msiexec /i openssh.msi REMOVE=Client
* Uninstalling SSH Server & openssh.msi is in C:\users\public\downloads:  
  msiexec /i C:\users\public\downloads\openssh.msi REMOVE=Server

## 2. Update SYSTEM PATH (Required for SCP and SFTP)

Append the Win32-OpenSSH install directory to the system path, by running the following command in an elevated PowerShell session:

[Environment]::SetEnvironmentVariable("Path", [Environment]::GetEnvironmentVariable("Path",[System.EnvironmentVariableTarget]::Machine) + ';' + ${Env:ProgramFiles} + '\OpenSSH', [System.EnvironmentVariableTarget]::Machine)

To verify that the System Path variable was modified properly, the Environment Variables can be viewed in Control Panel, under the Advanced tab.

## 3. Verify OpenSSH Install

Check the status of the SSH Service.  
In PowerShell, run:

Get-Service -Name ssh\*

# Uninstall Win32-OpenSSH using MSI

Similarly, the command to uninstall Win32-OpenSSH is as follows:  
msiexec /x <path to openssh.msi>

# Additional Documentation on msiexec

Further information on msi command-line options can be found [here](https://docs.microsoft.com/en-us/windows/win32/msi/command-line-options)

# Video recording

<https://user-images.githubusercontent.com/23668037/159090480-fb40882f-bf52-4dd7-a6b2-f3536128ab82.mp4>

* [Script Install Instructions](https://github.com/PowerShell/Win32-OpenSSH/wiki/Install-Win32-OpenSSH)
* [Alternative installation using the universal installer](https://github.com/PowerShell/Win32-OpenSSH/wiki/%5BDeprecated%5D-Win32-OpenSSH-Automated-Install-and-Upgrade-using-Chocolatey)
* [Retrieving download links for the latest packages](https://github.com/PowerShell/Win32-OpenSSH/wiki/How-to-retrieve-links-to-latest-packages)

## Usage

* [SSH Usage Examples](https://github.com/PowerShell/Win32-OpenSSH/wiki/ssh.exe-examples)

**Login With Password**

1. Workgroup users
   * ssh user@host
2. Domain users: Prior to v7.7.0.0, domain needs to be explicitly specified. Any of the following formats work
   * ssh -l user@domain host
   * ssh domain\user@host
   * ssh user@domain@host
   * ssh user@host (works from v7.7.0.0 onwards provided user has no conflicts otherwise - ex. user exists both on local account data base and on domain)

**Login With SSH Keys**

**Usage from client-side (ssh)**

1. Generate a key pair on the client (preferably with a passphrase, this provide 2-factor):
   * ssh-keygen -t rsa -f id\_rsa
2. Register private key with ssh-agent (optional, for single sign-on experience)
   * net start ssh-agent
   * ssh-add id\_rsa
3. Login using private key
   * ssh -i .\id\_rsa user@host (workgroup user)
   * ssh -i .\id\_rsa -l user@domain host (domain user)

**Setup server-side (sshd)**

1. Append contents of id\_rsa.pub (client's public key) to the following file in corresponding user's directory %systemdrive%\Users\<user>\.ssh\authorized\_keys (create one if needed).
2. Double check access permissions on authorized\_keys (only System, Administrators and owner can have access). icacls %systemdrive%\Users\<user>\.ssh\authorized\_keys

**Login using Kerberos Authentication**

**Setup server-side**

1. On a domain joined server, set GSSAPIAuthentication to yes in sshd\_config
2. If sshd\_config is modified then restart the sshd service
   * net stop sshd
   * net start sshd

**Usage on a domain joined Windows client logged in as domain user**

* ssh -K host

**Please note you have to use the hostname instead of the username.**

* [SFTP Usage Examples](https://github.com/PowerShell/Win32-OpenSSH/wiki/sftp.exe-examples)
* [Using Certificate Authentication](https://github.com/PowerShell/Win32-OpenSSH/wiki/Certificate-Authentication)
* [Fix SSH file permissions](https://github.com/PowerShell/Win32-OpenSSH/wiki/OpenSSH-utility-scripts-to-fix-file-permissions)
* [Info on SSH remote sessions on Windows](https://github.com/PowerShell/Win32-OpenSSH/wiki/SSH-remote-sessions-on-Windows)
* [TTY PTY support](https://github.com/PowerShell/Win32-OpenSSH/wiki/TTY-PTY-support-in-Windows-OpenSSH)
* [Troubleshooting](https://github.com/PowerShell/Win32-OpenSSH/wiki/Troubleshooting-Steps)

# Configure OpenSSH Server service using “sshd\_config” file

# sshd\_config

Listed here are Windows specific details that supplement or override the original sshd configuration manual documented in [OpenBSD manual](https://man.openbsd.org/sshd_config). If you don't see a configuration entry here, the original man page reference holds true.

In Windows, [sshd](https://man.openbsd.org/sshd) reads configuration data from %programdata%\ssh\sshd\_config (or the file specified with -f on the command line). If this file is absent, sshd will generate one with the default configuration on a service start.

Graphical user interface, text, application

Description automatically generated

To override the default shell (cmd.exe) used for command invocations, follow steps [here](https://github.com/PowerShell/Win32-OpenSSH/wiki/DefaultShell)

#### [AllowGroups](https://man.openbsd.org/sshd_config#AllowGroups), [AllowUsers](https://man.openbsd.org/sshd_config" \l "AllowUsers), [DenyGroups](https://man.openbsd.org/sshd_config" \l "DenyGroups), [DenyUsers](https://man.openbsd.org/sshd_config" \l "DenyUsers)

The allow/deny directives are processed in the following order: **DenyUsers**, **AllowUsers**, **DenyGroups**, and finally **AllowGroups**.

See PATTERNS in [ssh\_config](http://man.openbsd.org/ssh_config.5" \l "PATTERNS) for more information on patterns.

User and group names are case insensitive in Windows (unlike in Unix). You **should** always use **lower** case while specifying these irrespective of their original case.

**Note** the following for domain accounts:

Prior to v7.7.0.0, there was no well defined way to specify domain principals (users and groups). To account for a domain principal in [various forms](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724268(v=vs.85).aspx), it is recommended to use the following format while configuring user/group based rules - user?domain\* - note the ? instead of @ to avoid conflict with username@host format and \* added to cover FQDNs.

From v7.7.0.0 on wards, work group users/groups and internet-connected accounts are strictly resolved to their local account name (no domain part, similar to standard Unix names). Domain users and groups are strictly resolved to NameSamCompatible format - domain\_short\_name\user\_name. All user/group based configuration rules need to adhere to this format.

* Ex. for domain users and groups
  + DenyUsers contoso\admin@192.168.2.23 : blocks contoso\admin from 192.168.2.23
  + DenyUsers contoso\\* : blocks all users from contoso domain
  + AllowGroups contoso\sshusers : only allow users from contoso\sshusers group
  + AllowGroups "contoso\ssh users" : only allow users from "contoso\ssh users" group
* Ex. for local users and groups
  + AllowUsers localuser@192.168.2.23
  + AllowGroups sshusers

**Note that user and group names are in lower case**

#### [AuthenticationMethods](https://man.openbsd.org/sshd_config#AuthenticationMethods)

Available authentication methods are "password" and "publickey".

#### [AuthorizedKeysFile](https://man.openbsd.org/sshd_config#AuthorizedKeysFile)

The default is “.ssh/authorized\_keys .ssh/authorized\_keys2”. If the path is not absolute, it is taken relative to user's home directory (or profile image path). Ex. c:\users\user.

From v7.7.2.2 on wards, following is the default location of AuthorizedKeysFile for all users in Administrators group

%programdata%/ssh/administrators\_authorized\_keys

#### [ChrootDirectory](https://man.openbsd.org/sshd_config#ChrootDirectory)

Support added in v7.7.0.0

This directive is only supported with sftp sessions. A remote session into cmd.exe wouldn't honor this. To setup a sftp-only chroot server, set ForceCommand to internal-sftp. You may also set up scp with chroot, by implementing a custom shell that would only allow scp and sftp.

#### [ForceCommand](https://man.openbsd.org/sshd_config#ForceCommand)

Enforced only on non-PTY sessions. To block PTY access, use [PermitTTY="no"](https://man.openbsd.org/sshd_config" \l "PermitTTY) directive. For certificate based ForceCommand, use [no-pty](https://man.openbsd.org/ssh-keygen#no-pty) option.

#### [GSSAPIAuthentication](https://man.openbsd.org/sshd_config#GSSAPIAuthentication)

Support for Kerberos authentication via GSSAPI is added in v7.9.0.0.

#### [HostKey](https://man.openbsd.org/sshd_config#HostKey)

The defaults are %programdata%/ssh/ssh\_host\_ecdsa\_key, %programdata%/ssh/ssh\_host\_ed25519\_key and %programdata%/ssh/ssh\_host\_rsa\_key. If the defaults are not present, sshd will automatically generate these on a service start.

#### [Match](https://man.openbsd.org/sshd_config#Match)

Note that pattern rules in [this](https://github.com/PowerShell/Win32-OpenSSH/wiki/sshd_config#allowgroups-allowusers-denygroups-denyusers) section. User and group names should be in **lower** case.

#### [PermitRootLogin](https://man.openbsd.org/sshd_config#PermitRootLogin)

Not applicable in Windows. To prevent administrator login, use Administrators with DenyGroups directive.

#### [SyslogFacility](https://man.openbsd.org/sshd_config#SyslogFacility)

If you need file based logging, use LOCAL0. Logs will be generated under %programdata%\ssh\logs.

Any other value, including the default value AUTH directs logging to ETW. For more info see [Logging Facilities in Windows](https://github.com/PowerShell/Win32-OpenSSH/wiki/Logging-Facilities).

# How to Enable and Configure SSH Server on Windows with OpenSSH?

# Win11ENTVM12 on Intel-i-9

**SSH-server** based on the OpenSSH package is part of the operating system in all modern versions of Windows. In this article, we’ll show you how to install and configure the OpenSSH server on Windows 10/11 and Windows Server 2022/2019 and connect to it remotely via a secure SSH connection (just like in Linux 🙂).

<http://woshub.com/connect-to-windows-via-ssh/>

**Contents:**

* [How to Install OpenSSH Server on Windows?](http://woshub.com/connect-to-windows-via-ssh/#h2_1)
* [Configuring SSH Server on Windows](http://woshub.com/connect-to-windows-via-ssh/#h2_2)
* [Sshd\_config: OpenSSH Server Configuration File](http://woshub.com/connect-to-windows-via-ssh/#h2_3)
* [How to Connect to a Remote Windows Computer via SSH?](http://woshub.com/connect-to-windows-via-ssh/#h2_4)
* [Checking SSH Connection Logs in Windows](http://woshub.com/connect-to-windows-via-ssh/#h2_5)

## How to Install OpenSSH Server on Windows?

The OpenSSH Server package is a part of all modern versions of Windows 10 (starting with 1803), Windows 11, and Windows Server 2022/2019 as a **Feature on Demand** (FoD). To install the OpenSSH server, open the elevated PowerShell prompt and run the command:

Graphical user interface, text, application, email

Description automatically generated

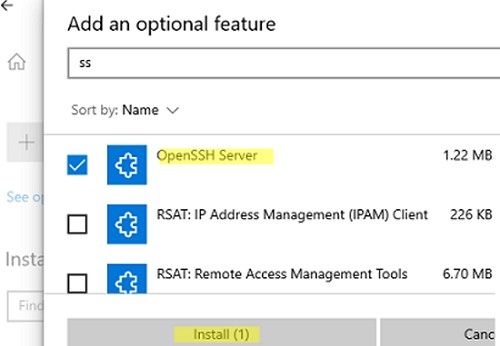
Figure 1 here both are installed

Or using DISM:

dism /Online /Add-Capability /CapabilityName:OpenSSH.Server~~~~0.0.1.0

If your computer is directly connected to the Internet, the OpenSSH.Server package will be downloaded and installed on Windows.

You can also install OpenSSH on Windows 10/11 through the modern **Settings** panel (Settings -> Apps and features -> Optional features -> Add a feature). Find **Open SSH Server** in the list and click **Install**.

[](http://woshub.com/wp-content/uploads/2020/06/install-openssh-server-windows10-settings.jpg)

You can also [install the Remote Server Administration Tools (RSAT)](http://woshub.com/install-rsat-feature-windows-10-powershell/) from here.

On computers in disconnected (offline) environments, you can install the OpenSSH Server from the **Feature on Demand ISO** image (available in your account on the Microsoft websites: MSDN or my.visualstudio.com).

Download the ISO and extract its contents to the E:\FOD folder (you can only extract the file OpenSSH-Server-Package~31bf3856ad364e35~amd64~~.cab) and install the Windows feature from the local repository:

Add-WindowsCapability -Name OpenSSH.Server~~~~0.0.1.0 -Online -Source E:\FOD

An MSI installer for OpenSSH for Windows is also available in the official Microsoft repository on GitHub (<https://github.com/PowerShell/Win32-OpenSSH/releases/>). For example, for Windows 10 x64, you need to download and install the **OpenSSH-Win64-v8.9.1.0.msi** package.

The following PowerShell command will download the MSI file and install the OpenSSH client and server on your computer:

[Invoke-WebRequest](http://woshub.com/parsing-html-webpages-with-powershell/) https://github.com/PowerShell/Win32-OpenSSH/releases/download/v8.9.1.0p1-Beta/OpenSSH-Win64-v8.9.1.0.msi -OutFile $HOME\Downloads\OpenSSH-Win64-v8.9.1.0.msi -UseBasicParsing  
msiexec /i $HOME\Downloads\OpenSSH-Win64-v8.9.1.0.msi

[](http://woshub.com/wp-content/uploads/2020/06/openssh-msi-installer-windows.jpg)

You can install an OpenSSH server in previous Windows versions as well (Windows 8.1, Windows Server 2016/2012R2/2012). Check the example on how to install and configure **Win32-OpenSSH** in the article “[How to Configure SFTP Server (SSH FTP) on Windows?](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/)”.

To make sure the OpenSSH server has been installed, run the command:

Get-WindowsCapability -Online | ? Name -like 'OpenSSH.Ser\*'

State : Installed

check openssh server feature installed on windows Get-WindowsCapability

## Configuring SSH Server on Windows

After installing the OpenSSH server on Windows, two services are added:

* **ssh-agent** (OpenSSH Authentication Agent) – can be used to manage private keys if you have configured SSH key authentication;
* **sshd** (OpenSSH SSH Server).

You need to change the startup type of the sshd service to automatic and [start the service using PowerShell](http://woshub.com/manage-windows-services-powershell/):

Set-Service -Name sshd -StartupType 'Automatic'  
Start-Service sshd

start sshd service on windows 10

Use the netstat command to make sure that the SSH server is running and waiting for the connections on TCP port 22:

netstat -na| find ":22"

check ssh tcp port 22 listening on windows 10

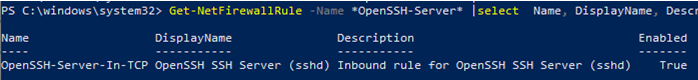
Make sure that Windows Defender Firewall allows inbound connections to Windows through TCP port 22:

Get-NetFirewallRule -Name \*OpenSSH-Server\* |select Name, DisplayName, Description, Enabled

Name DisplayName Description Enabled

---- ----------- ----------- -------

OpenSSH-Server-In-TCP OpenSSH SSH Server (sshd) Inbound rule for OpenSSH SSH Server (sshd) True



If the rule is disabled (Enabled=False) or missing, you can [create a new inbound rule using the New-NetFirewallRule cmdlet](http://woshub.com/manage-windows-firewall-powershell/):

New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22

By default, key OpenSSH components are located in these folders:

* OpenSSH Server executables: C:\Windows\System32\OpenSSH\(sshd.exe, ssh.exe, ssh-keygen.exe, sftp.exe, etc.)
* The **sshd\_config** file (created after the first service start of the service): C:\ProgramData\ssh
* The authorized\_keys file and keys can be stored in the user profile folder: %USERPROFILE%\.ssh\

## Sshd\_config: OpenSSH Server Configuration File

You can change your OpenSSH server settings in the config file: %programdata%\ssh\**sshd\_config**. This is a plain text file with a set of directives. You can use any text editor for editing:

start-process notepad C:\Programdata\ssh\sshd\_config

For example, to deny SSH connection for a specific domain user account (or all users in the specified domain), add these directives to the end of the file:

DenyUsers woshub\admin@192.168.1.10

DenyUsers corp\\*

To allow SSH connection to the specific domain security group only:

AllowGroups woshub\sshadmins

You can allow access to a local user group:

AllowGroups sshadmins

By default, all Windows users can connect to OpenSSH. Directives in the sshd\_config files are processed in the following order: DenyUsers, AllowUsers, DenyGroups, AllowGroups.

You can deny SSH login for the accounts with administrator privileges. In this case, if you need to perform any privileged actions in your SSH session, you will have to use [runas](http://woshub.com/run-program-as-different-user-windows/).

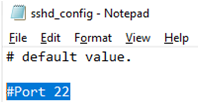
DenyGroups Administrators

The following directives allow you to [access Windows using SSH private keys](http://woshub.com/using-ssh-key-based-authentication-on-windows/) or a password.

PubkeyAuthentication yes

PasswordAuthentication yes

You can change the default TCP/22 port on which OpenSSH Server connections are accepted in the sshd\_config configuration file using the **Port** directive.



After making any changes to the sshd\_config file, you need to restart the sshd service

restart-service sshd

## SSHD\_Config: OpenSSH Server Config text file C:\programdata\ssh

# This is the sshd server system-wide configuration file. See

# sshd\_config(5) for more information.

# The strategy used for options in the default sshd\_config shipped with

# OpenSSH is to specify options with their default value where

# possible, but leave them commented. Uncommented options override the

# default value.

#Port 22

#AddressFamily any

#ListenAddress 0.0.0.0

#ListenAddress ::

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_rsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_dsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_ecdsa\_key

#HostKey \_\_PROGRAMDATA\_\_/ssh/ssh\_host\_ed25519\_key

# Ciphers and keying

#RekeyLimit default none

# Logging

#SyslogFacility AUTH

#LogLevel INFO

# Authentication:

#LoginGraceTime 2m

#PermitRootLogin prohibit-password

#StrictModes yes

#MaxAuthTries 6

#MaxSessions 10

#PubkeyAuthentication yes

# The default is to check both .ssh/authorized\_keys and .ssh/authorized\_keys2

# but this is overridden so installations will only check .ssh/authorized\_keys

AuthorizedKeysFile .ssh/authorized\_keys

#AuthorizedPrincipalsFile none

# For this to work you will also need host keys in %programData%/ssh/ssh\_known\_hosts

#HostbasedAuthentication no

# Change to yes if you don't trust ~/.ssh/known\_hosts for

# HostbasedAuthentication

#IgnoreUserKnownHosts no

# Don't read the user's ~/.rhosts and ~/.shosts files

#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!

#PasswordAuthentication yes

#PermitEmptyPasswords no

# GSSAPI options

#GSSAPIAuthentication no

#AllowAgentForwarding yes

#AllowTcpForwarding yes

#GatewayPorts no

#PermitTTY yes

#PrintMotd yes

#PrintLastLog yes

#TCPKeepAlive yes

#UseLogin no

#PermitUserEnvironment no

#ClientAliveInterval 0

#ClientAliveCountMax 3

#UseDNS no

#PidFile /var/run/sshd.pid

#MaxStartups 10:30:100

#PermitTunnel no

#ChrootDirectory none

#VersionAddendum none

# no default banner path

#Banner none

# override default of no subsystems

Subsystem sftp sftp-server.exe

# Example of overriding settings on a per-user basis

#Match User anoncvs

# AllowTcpForwarding no

# PermitTTY no

# ForceCommand cvs server

Match Group administrators

AuthorizedKeysFile \_\_PROGRAMDATA\_\_/ssh/administrators\_authorized\_keys

## How to Connect to a Remote Windows Computer via SSH?

Now you can try to connect to your Windows 10 computer using the SSH client (I’m using putty in this example).

You can [use the built-in Windows SSH client](http://woshub.com/using-native-ssh-client-windows/) to connect to a remote host. To do this, open the command prompt and run the following command:

ssh [max@192.168.13.12](mailto:max@192.168.13.12)

In this example, max is the username on the remote Windows computer, and 192.168.13.12 is the IP address or DNS name of the computer.

Note that you can use the following username formats when connecting to Windows via SSH:

* max@server1 – [local Windows user](http://woshub.com/manage-local-users-groups-powershell/)
* max@woshub.com@server1 – Active Directory user or Microsoft/Azure account (use the [UserPrincipalName](http://woshub.com/configure-userprincipalname-upn-suffixes-active-directory/) format)
* woshub\max@server1 – NetBIOS name format

In an Active Directory domain, you can use Kerberos authentication in SSH. To do this, you need to enable the following directive in sshd\_config:

GSSAPIAuthentication yes

You can now transparently connect to an SSH server from a domain-joined Windows machine with a domain user session. In this case, the user’s password will not be requested, and SSO authentication via Kerberos will be performed:

ssh -K server1

The first time you connect, you will be prompted to add the host to the list of known SSH hosts (C:\Users\your\_user\.ssh\known\_hosts).



If you are using SSH to connect to a server for the first time, you will probably see a message looking something like this:

The host key is not cached for this server:

ssh.example.com (port 22)

You have no guarantee that the server is the computer you think it is.

The server's ssh-ed25519 key fingerprint is:

ssh-ed25519 255 SHA256:TddlQk20DVs4LRcAsIfDN9pInKpY06D+h4kSHwWAj4w

If you trust this host, press "Accept" to add the key to PuTTY's

cache and carry on connecting.

If you want to carry on connecting just once, without adding the key

to the cache, press "Connect Once".

If you do not trust this host, press "Cancel" to abandon the connection.

This is a feature of the SSH protocol. It is designed to protect you against a network attack known as spoofing: secretly redirecting your connection to a different computer, so that you send your password to the wrong machine. Using this technique, an attacker would be able to learn the password that guards your login account, and could then log in as if they were you and use the account for their own purposes.

**To prevent this attack**, each server has a unique identifying code, called a host key. These keys are created in a way that prevents one server from forging another server's key. So if you connect to a server and it sends you a different host key from the one you were expecting, PuTTY can warn you that the server may have been switched and that a spoofing attack might be in progress.

PuTTY records the host key for each server you connect to, in the Windows Registry. Every time you connect to a server, it checks that the host key presented by the server is the same host key as it was the last time you connected. If it is not, you will see a stronger warning, and you will have the chance to abandon your connection before you type any private information (such as a password) into it. (See [section 10.2](file:///\\tsclient\E\AA-%20YouTube\YouTube%20stuff\AA-Active%20Projects\Video%20SSH\Research\puttydoc\Chapter10.html#errors-hostkey-wrong) for what that looks like.)

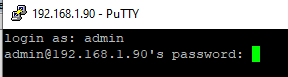
However, when you connect to a server you have not connected to before, PuTTY has no way of telling whether the host key is the right one or not. So it gives the warning shown above, and asks you whether you want to trust this host key or not.

Whether or not to trust the host key is your choice. If you are connecting within a company network, you might feel that all the network users are on the same side and spoofing attacks are unlikely, so you might choose to trust the key without checking it. If you are connecting across a hostile network (such as the Internet), you should check with your system administrator, perhaps by telephone or in person. (When verifying the fingerprint, be careful with letters and numbers that can be confused with each other: 0/O, 1/I/l, and so on.)

Many servers have more than one host key. If the system administrator sends you more than one fingerprint, you should make sure the one PuTTY shows you is on the list, but it doesn't matter which one it is.

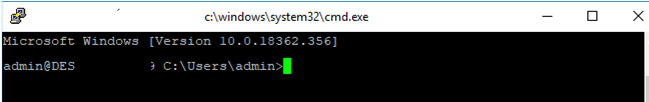
If you don't have any fingerprints that look like the example (SHA256: followed by a long string of characters), but instead have pairs of characters separated by colons like a4:db:96:a7:..., try pressing the ‘More info...’ button and see if you have a fingerprint matching the ‘MD5 fingerprint’ there. This is an older and less secure way to summarise the same underlying host key; it's possible for an attacker to create their own host key with the same fingerprint; so you should avoid relying on this fingerprint format unless you have no choice. The ‘More info...’ dialog box also shows the full host public key, in case that is easier to compare than a fingerprint.

Click Yes, and login under your Windows user account.



If the SSH connection is successful, you will see the cmd.exe shell prompt.

admin@win10pc C:\Users\admin>



Graphical user interface

Description automatically generated

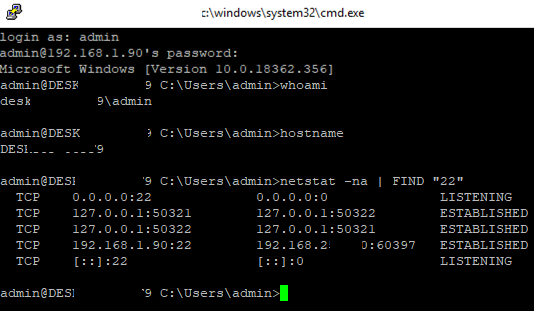
Graphical user interface

Description automatically generated

Graphical user interface

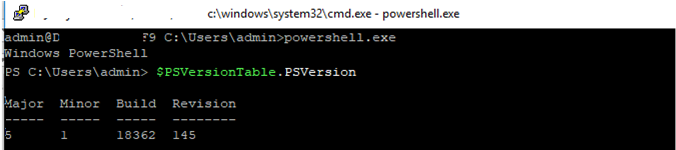
Description automatically generated

You can run different commands, scripts, and apps in the SSH command prompt.



I prefer working in the PowerShell console. To start it, run:

powershell.exe

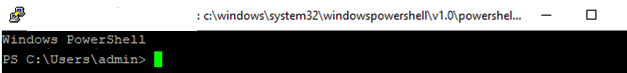


In order to change the default cmd.exe shell in OpenSSH to PowerShell, make [changes to the registry using the following PowerShell command](http://woshub.com/how-to-access-and-manage-windows-registry-with-powershell/):

New-ItemProperty -Path "HKLM:\SOFTWARE\OpenSSH" -Name DefaultShell -Value "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -PropertyType String –Force

New-ItemProperty replacing ssh shell from cmd.exe to powershell.exe

Restart your SSH connection and make sure that PowerShell is now used as a default SSH shell (this is indicated by the prompt PS C:\Users\admin>).



The PowerShell prompt has been started in my SSH session, where the usual functions work: tab autocomplete, PSReadLine syntax highlighting, [command history](http://woshub.com/powershell-commands-history/), etc. If the current user is a member of the local administrators’ group, all session commands are executed elevated even if [UAC is enabled](http://woshub.com/user-account-control-slider-and-group-policy-settings/).

OpenSSH server on Windows can be used in various [SSH tunneling](https://woshub.com/ssh-tunnel-port-forward-windows/) scenarios.

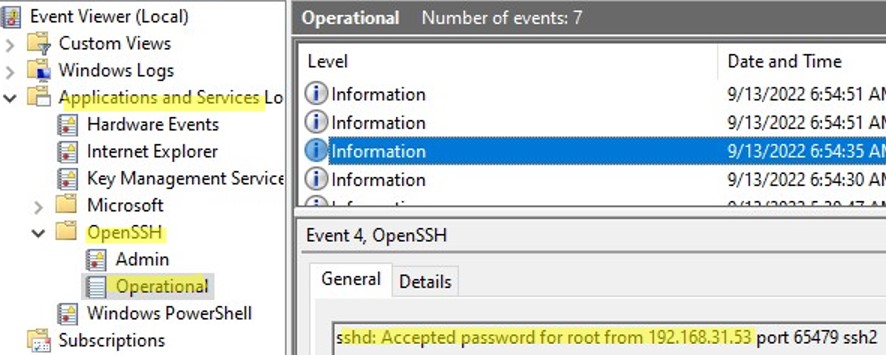
## Checking SSH Connection Logs in Windows

By default in Windows SSH server connection logs are written not to text files, but to a separate event log via Event Tracing for Windows (ETW). Open the Event Viewer console (eventvwr.msc ) and navigate to **Application and services logs** -> **OpenSSH** -> **Operational**.

If you successfully connect to the SSH server using a password, an event will appear in the log:

EventID: 4

sshd: Accepted password for root from 192.168.1.53 port 65749 ssh2

[](http://woshub.com/wp-content/uploads/2020/06/sshd-connection-logs-event-viewer.jpg)

If SSH key authentication was performed, you will see the following event:

sshd: Accepted publickey for locadm from 192.168.1.53 port 61426

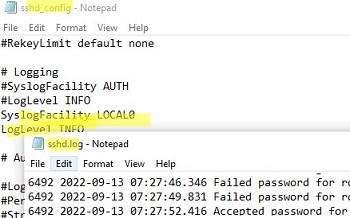
ssh2: ED25519 SHA256:FEHDEC/G42FS23209C2KMb4335923pigN31s3qMK322lGibD

If you want the SSH connection logs to be written to a local text file, you need to enable the following parameters in the **sshd\_config** file:

SyslogFacility LOCAL0

LogLevel INFO

Restart the sshd service and make sure that the SSH server logs are now written to a plain text file **C:\ProgramData\ssh\logs\sshd.log**

[](http://woshub.com/wp-content/uploads/2020/06/sshd-log-file-openssh-server.jpg)

You can [use a secure SSH connection with PSRemoting](http://woshub.com/powershell-remoting-over-ssh/) to manage remote computers via [WinRM](http://woshub.com/enable-winrm-management-gpo/).

##### **[How to Remove RD Session Host from Remote Desktop Services Deployment?](http://woshub.com/remove-rds-host/)**

#### **RELATED READING**

### [**How to Install and Configure Free Hyper-V Server...**](http://woshub.com/install-configure-free-hyper-v-server/)

November 22, 2022

### [**How to Find the Source of Account Lockouts...**](http://woshub.com/troubleshooting-identify-source-of-active-directory-account-lockouts/)

November 22, 2022

### [**Using Process Tracking Audit Policy in Windows**](http://woshub.com/process-tracking-audit-policy/)

November 23, 2022

# Configuring SFTP (SSH FTP) Server on older versions of Windows/Server

You can use the built-in OpenSSH package in Windows to easily enable secure file transfers between the client and Windows server using the **SFTP** (**Secure FTP**) protocol. In this article, we will show how to install and configure an SFTP server on Windows 10 or Windows Server 2022/2019/2016/2012R2.

**Contents:**

* [How to Install OpenSSH on Windows?](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/#h2_1)
* [How to Configure SFTP Server on Windows Using OpenSSH?](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/#h2_2)
* [Connecting to SFTP Server Using WinSCP or PowerShell](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/#h2_3)
* [Configuring SFTP Public Key Authentication](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/#h2_4)

**SFTP** (**Secure** **File Transfer Protocol**, **Secure FTP**, or **SSH FTP**) is the extension of SSH protocol, which is the standard in the world of UNIX/Linux systems. From the user’s point of view, it is similar to FTP, but in fact, it is a completely different protocol, having nothing in common with [FTP](http://woshub.com/ftp-site-with-user-isolation-on-windows-server-2012-r2/). Data between the client and the server is transmitted on port 22 through an [SSH tunnel](http://woshub.com/ssh-tunnel-port-forward-windows/) (TCP port 22).

**The main advantages of SFTP:**

* Files and commands are transferred within a secure SSH session;
* One connection is used to send both files and commands;
* Symbolic links, interrupt/resume the transfer, file delete functions, etc. are supported;
* SFTP connection is much faster and more reliable on WAN links where FTP is slow or intermittent;
* Possibility to authenticate using SSH keys.

**Do not confuse SFTP and FTPS protocols.**

* [FTPS](http://woshub.com/ftp-over-ssl-ftps-windows-server-2012-r2/) is essentially just a simple FTP with an SSL certificate, and
* SFTP is the protocol to transfer the FTP data and commands inside the SSH session.

The win32 port of OpenSSH (**Win32-OpenSSH**) is built into all modern versions of Windows by default. You can use it to configure a secure SFTP server instead of using third-party products like Core FTP, FileZilla, CYGWIN, FTP Shell, IPSwitch, etc.

**How to Install OpenSSH on Windows?**

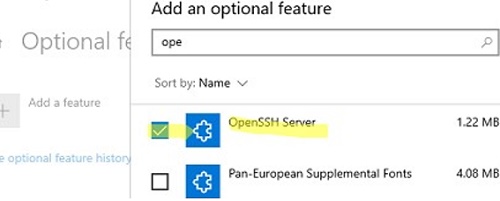
The OpenSSH package is a part of the operating system in modern builds of Windows 10 (starting from 1803), Windows 11, and Windows Server 2022/2019 as a Feature on Demand (like [RSAT](http://woshub.com/install-rsat-feature-windows-10-powershell/)).

On these versions of [Windows, you can install the OpenSSH server](http://woshub.com/connect-to-windows-via-ssh/) using PowerShell:

Add-WindowsCapability -Online -Name OpenSSH.Server\*

Or using DISM:  
dism /Online /Add-Capability /CapabilityName:OpenSSH.Server~~~~0.0.1.0

Also, you can install OpenSSH server from Windows 10 GUI (**Settings** -> **Apps** -> **Optional Features** -> **Add a feature** -> **Open SSH Server** -> **Install**).

[](http://woshub.com/wp-content/uploads/2019/10/install-openssh-server-windows.jpg)

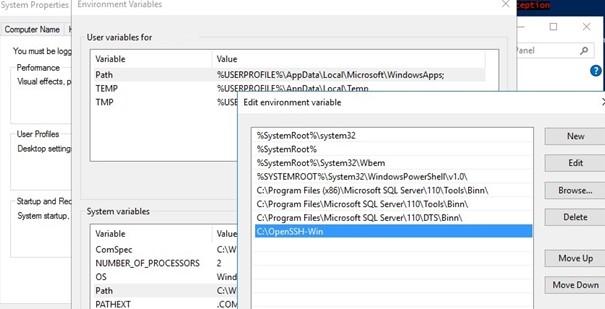
To check if a package is installed:

Get-WindowsCapability -Online | ? Name -like 'OpenSSH\*'

* OpenSSH executables are located in the directory: c:\windows\system32\OpenSSH\;
* The**sshd\_config** configuration file is located in  C:\ProgramData\ssh (this directory is created after the first start of the sshd service);
* Log file: c:\windows\system32\OpenSSH\logs\sshd.log;
* The authorized\_keys file and keys are stored in a directory: %USERPROFILE%\.ssh\.

On the previous earlier builds of Windows 10, Windows 8.1, and on Windows Server 2016/2012 R2, you will have to download Win32-OpenSSH for Windows from GitHub and install it manually

(<https://github.com/PowerShell/Win32-OpenSSH/releases>). We need a version for Windows x64: **OpenSSH-Win64.zip** (4,15 MB).

1. Extract the archive to the target folder: C:\OpenSSH-Win;
2. Open an elevated PowerShell prompt and switch to the OpenSSH folder: Cd C:\OpenSSH-Win
3. Add the path to the OpenSSH directory to the **Path** environment variable (System Properties -> Advanced tab -> Environment Variables -> Select and edit the **Path** system variable -> Add the path to the OpenSSH folder); [](http://woshub.com/wp-content/uploads/2016/07/openssh-folder-in-path-system-variable.jpg)
4. Install the OpenSSH server: .\install-sshd.ps1 (a green message should appear “*sshd and ssh-agent services successfully installed*”);

If running PowerShell scripts on your computer is blocked by your [PowerShell Execution Policy](http://woshub.com/configure-powershell-script-execution-policy/), you can run the script with this command: powershell.exe -ExecutionPolicy Bypass -File install-sshd.ps1

[install openssh (sshd service) with powershell script ](http://woshub.com/wp-content/uploads/2016/07/install-sshd-powershell-script-in-openssh.jpg)

**How to Configure SFTP Server on Windows Using OpenSSH?**

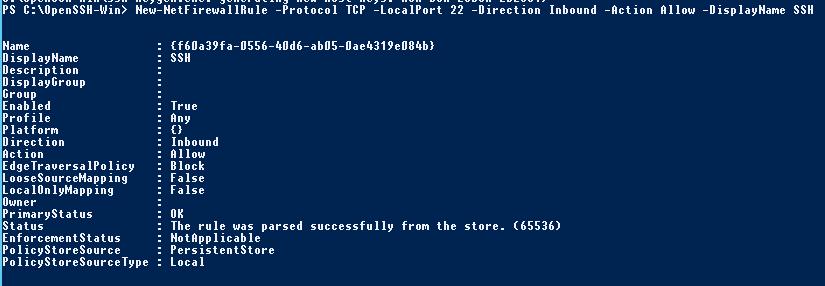
Now you need to configure OpenSSH on Windows for SFTP mode.

Enable autostart for the SSHD service and start it using the following [PowerShell service management commands](http://woshub.com/manage-windows-services-powershell/):

Set-Service -Name sshd -StartupType 'Automatic'  
Start-Service sshd

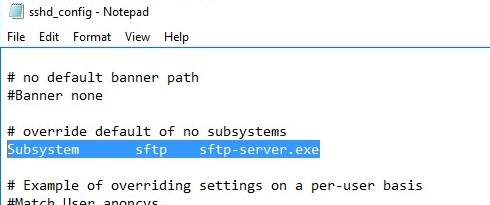
Use the PowerShell to open TCP port 22 in the Windows Firewall for incoming SSH traffic:

[New-NetFirewallRule](http://woshub.com/manage-windows-firewall-powershell/) -Protocol TCP -LocalPort 22 -Direction Inbound -Action Allow -DisplayName SSH

[](http://woshub.com/wp-content/uploads/2016/07/New-NetFirewallRule-port-22.jpg)

**Note**. The previous command won’t work in old desktop Windows versions. In this case, another command is used: netsh advfirewall firewall add rule name='SSH Port' dir=in action=allow protocol=TCP localport=22

Open the SSHD configuration file (C:\ProgramData\SSH\sshd\_config) in any text editor.  Find and check the value of the **Subsystem sftp** directive. The **sftp-server.exe** file should be specified here.

[](http://woshub.com/wp-content/uploads/2016/07/openssh-sshd-config-file.jpg)

You can additionally configure the following parameters in the sshd\_config configuration file:

# only allow users in this domain group to connect to OpenSSH

AllowGroups corp\sftp\_users

# enable password authentication (SSH keys cannot be used)

AuthenticationMethods password

#default (chrooot) directory for SFTP users  (by default, the user connects to the directory with his profile in the C:\users\username folder)

ChrootDirectory C:\SFTP

ForceCommand internal-sftp

#You can set an individual chrootdirectory for each user:

Match User abrown

ChrootDirectory c:\SFTP\abrown

ForceCommand internal-sftp

X11Forwarding no

AllowTcpForwarding no

Match User jsmith

ChrootDirectory c:\SFTP\jsmith

ForceCommand internal-sftp

X11Forwarding no

AllowTcpForwarding no

**Connecting to SFTP Server Using WinSCP or PowerShell**

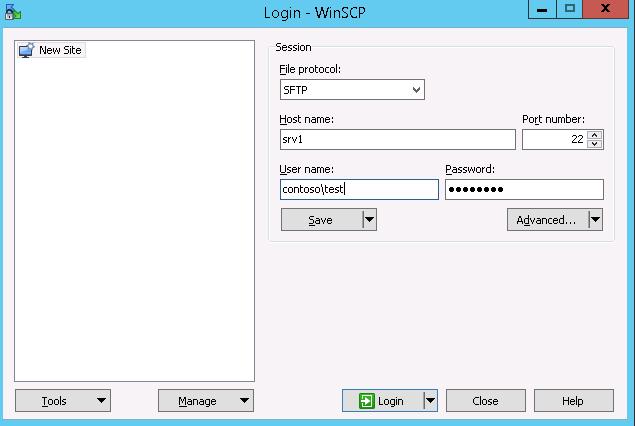
Now you can connect to your Windows SSH server using the SFTP protocol. Next, we’ll show you how to connect to an SFTP server using the free **WinSCP** client, the **PowerShell** console, and the built-in **sftp.exe** tool.

<https://winscp.net/eng/index.php> Free Award-Winning File Manager

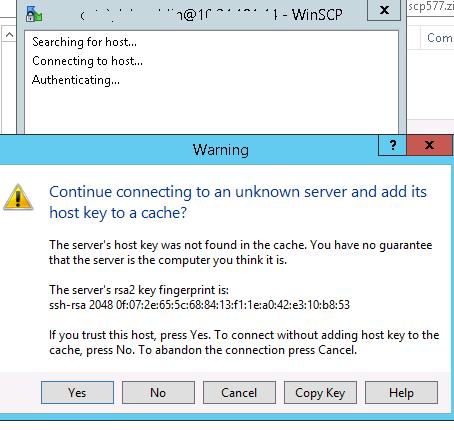
A picture containing graphical user interface

Description automatically generated

In the connection configuration window, select the **SFTP** as the file transfer protocol, specify the server name and the credentials of the Windows account (use the user@domain format for domain users), which is used for connection (it is also possible to configure public key authentication).

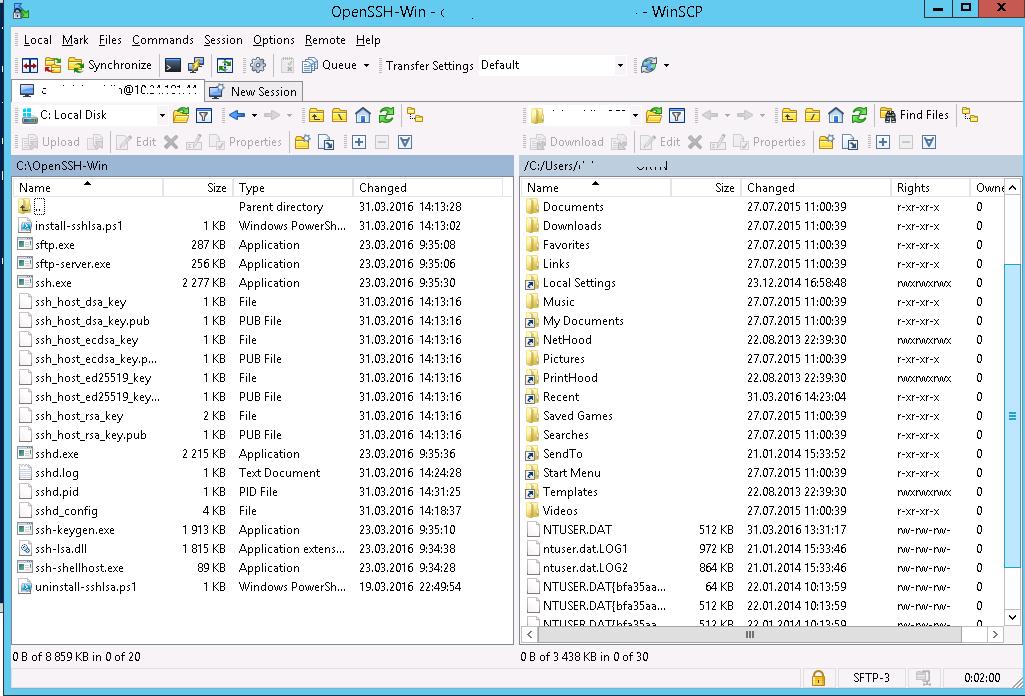
[](http://woshub.com/wp-content/uploads/2016/07/winscp-test-sftp-server.jpg)

When you try to connect for the first time, the following notification of the host key not found in the local cache appears.

[](http://woshub.com/wp-content/uploads/2016/07/rsa2-key-warning.jpg)

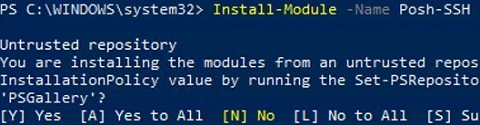
If everything is configured correctly, a client should connect to the SFTP server and display the list of files in the user’s home chroot directory (by default, it is the user’s profile directory).

Using the familiar file manager interface (*like Total Commander*), you can copy files between the server and the client using the secure SFTP protocol.

[](http://woshub.com/wp-content/uploads/2016/07/connect-openssh-on-windows-using-winscp.jpg)

You can use the **Posh-SSH** module to connect to an SFTP server from PowerShell. You can download and install the module from the PowerShell Gallery or [offline](http://woshub.com/install-powershell-module-offline/):

Install-Module -Name Posh-SSH

[](http://woshub.com/wp-content/uploads/2019/10/install-posh-ssh-module.jpg)

To connect to the SFTP server using a password, you need to get the username and password via Get-Credential:

$usrCreds= Get-Credential

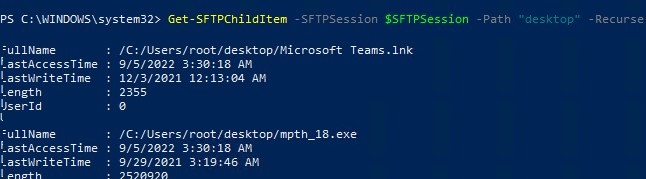
You can use the [PowerShell module SecretManagement](http://woshub.com/powershell-secretmanagement-module/) to securely retrieve a saved password from a [Windows Credential Manager](http://woshub.com/saved-passwords-windows-credential-manager/) or external vault (KeePass, HashiCorp Vault, Azure Key Vault, Bitwarden, etc.).

Now you can connect to your SFTP server:

$SFTPSession = New-SFTPSession -ComputerName 192.168.3.20 -Credential $usrCreds

Now you can list the files in the remote directory on the SFTP server. In this example, I will get a list of files on the user’s Desktop (the user’s profile will be the root user folder/chroot in this case)

Get-SFTPChildItem -SFTPSession $SFTPSession -Path "desktop" –Recurse

[](http://woshub.com/wp-content/uploads/2019/10/powershell-list-files-in-sftp-folder.jpg)

Download a file from a remote SFTP server:

Get-SFTPItem -SessionId $SFTPSession.SessionId -Path "desktop/OpenVPNScript.log" -Destination c:\PS

To upload a file from your computer to a remote SFTP host:

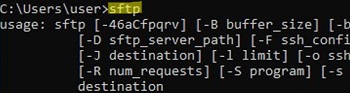
Set-SFTPItem -SessionId $SFTPSession.SessionId -Path C:\PS\mytestfile.log -Destination "desktop"

[upload files to sftp with powershell](http://woshub.com/wp-content/uploads/2019/10/upload-files-to-sftp-powershell.jpg)

Close the SFTP session:

Remove-SFTPSession -SFTPSession $SFTPSession

On Windows, you can use the built-in **sftp.exe** console command (installed with the OpenSSH client) to connect to an SFTP server.

[](http://woshub.com/wp-content/uploads/2019/10/sftp-exe-tool-in-windows.jpg)

Connect to sftp server:

sftp user1@192.168.3.20

Connect using ssh private key:

sftp -i .ssh/id\_rsa user1@192.168.3.20

List files in a remote directory:

pwd

Download the file from SFTP to a local directory on your computer:

get download\_this\_file\_from\_sftp.txt

Upload a file from your computer to an SFTP server:

put file\_to\_uplodad.txt

Close session:

exit

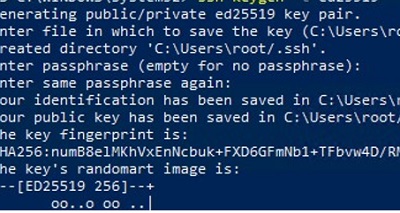
Configuring SFTP Public Key Authentication

You can enable SFTP key-based authentication in Windows. In this case, you can authenticate to the SFTP server without entering a password.

For more information on [how to set up SSH key-based authentication](http://woshub.com/using-ssh-key-based-authentication-on-windows/), check this article.

1. Create SSH keys on your computer (SFTP client) for the user under which you will connect to the server: ssh-keygen -t ed25519

Text

Description automatically generated[](http://woshub.com/wp-content/uploads/2019/10/generate-ssh-keys-sftp.jpg)

1. The ssh-keygen tool will generate two files **id\_ed25519** (private key for the client computer) and **id\_ed25519.pub** (public key for the SFTP server);
2. Now you need to add your SSH key to the Windows server (SFTP host). Copy the file id\_ed25519.pub (or id\_rsa.pub depending on the key type) to the .ssh directory of the user profile under which you will connect to the SFTP. Rename the file to **authorized\_keys** (for example, the following key file is used for the user max1: C:\Users\max1\.ssh\authorized\_keys )

Graphical user interface, application

Description automatically generated

Figure 2 private keys are stored in the user profile

Graphical user interface, text, application

Description automatically generated

Figure 3 Here the private key is stored as Homeboss.pub

Now you can use the id\_ed25519 file to authenticate on the SFTP server. You can set your key in the WinSCP settings (Advanced -> to SSH connection settings > Authentication page -> Private key file).

If you want to use an SSH key when connecting to SFTP from PowerShell, use the following command:

New-SFTPSession -ComputerName 192.168.3.20 -Credential remoteuser1 -KeyFile C:\Users\max1\.ssh\id\_ed25519" -Verbose

# Configuring SSH Public Key Authentication on Windows

<http://woshub.com/using-ssh-key-based-authentication-on-windows/>

In this article, we will show how to configure SSH authentication in Windows using RSA or EdDSA keys.  Let’s see how to generate public and private key pair on Windows and configure an OpenSSH server on Windows 10/11 or Windows Server 2019/2022 for key-based authentication (without passwords).

SSH key-based authentication is widely used in the Linux world, but in Windows, it has appeared quite recently. The idea is that the client’s public key is added to the SSH server, and when a client tries to connect to it, the server checks if the client has the corresponding private key. This way a remote user can authenticate in Windows without entering a password.

**Contents:**

* [Generating an SSH Key Pair on Windows](http://woshub.com/using-ssh-key-based-authentication-on-windows/#h2_1)
* [OpenSSH: Configuring Key-Based Authentication with Public Key on Windows](http://woshub.com/using-ssh-key-based-authentication-on-windows/#h2_2)
* [Logging Windows with SSH Key Under Administrative User](http://woshub.com/using-ssh-key-based-authentication-on-windows/#h2_3)

## Generating an SSH Key Pair on Windows

You must generate two SSH keys (public and private) on the client computer that you will use to connect to the remote Windows host running OpenSSH.

* A private key is stored on a client side (keep the key safe and don’t share it with anyone!),
* and a public key is added to the **authorized\_keys** file on the SSH server. To generate RSA keys on a Windows client, you must install the OpenSSH server.

On Windows 10/11 and Windows Server 2019/2022, the OpenSSH client is installed as an optional Windows feature using PowerShell:

Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

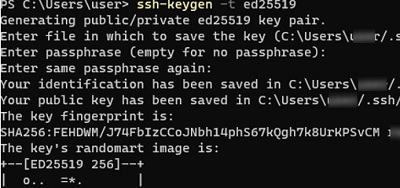
On previous Windows versions, you can install the **Win32-OpenSSH** port from GitHub (see the example in the article about [setting up an SFTP (SSH FTP) server on Windows](http://woshub.com/installing-sftp-ssh-ftp-server-on-windows-server-2012-r2/)).

Open a standard (non-elevated) PowerShell session and generate a pair of ED25519 keys using the command:

ssh-keygen -t ed25519

By default, the ssh-keygen tool generates RSA 2048 keys. Currently, it is recommended to use ED25519 instead of RSA keys.

You will be prompted to provide a password to protect the private key. If you specify the password, you will have to enter it each time you use this key for SSH authentication. I did not enter a passphrase (not recommended).

[](http://woshub.com/wp-content/uploads/2020/07/ssh-keygen-ed25519-ssh-key-windows.jpg)

Generating public/private ed25519 key pair.

Enter file in which to save the key (C:\Users\myuser/.ssh/id\_ed25519):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in C:\Users\myuser/.ssh/id\_ed25519.

Your public key has been saved in C:\Users\myuser/.ssh/id\_ed25519.pub.

The key fingerprint is: SHA256:xxxxxxxx myuser@computername

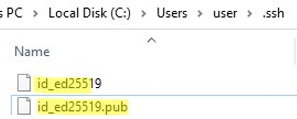
The key's randomart image is:

+--[ED25519 256]--+

+----[SHA256]-----+

Ssh-keygen will create the **.ssh** directory in the profile of a current Windows user (%USERPROFILE%\.ssh) and generate 2 files:

* id\_ed25519 – private key (if you generated an RSA key, the file will be named id\_rsa )
* id\_ed25519.pub – public key (a similar RSA key is called id\_rsa.pub

[](http://woshub.com/wp-content/uploads/2020/07/ssh-public-private-keys-windows.jpg)

After the SSH keys are generated, you can add your private key to the SSH Agent service, which allows you to conveniently manage private keys and use them for authentication.

The **SSH Agent** service can store your private keys and provide them in the security context of the current user. Run the ssh-agent service and configure it to start automatically using the [PowerShell service management commands](http://woshub.com/manage-windows-services-powershell/):

set-service ssh-agent StartupType ‘Automatic’  
Start-Service ssh-agent

Add your private key to the ssh-agent database:

ssh-add "C:\Users\youruser\.ssh\id\_ed25519"

Identity added: C:\Users\youruser\.ssh\id\_ed25519 (youruser@computername)

[add ssh private key to ssh-agent in windows](http://woshub.com/wp-content/uploads/2020/07/add-ssh-private-key-ssh-agent.jpg)

Or as follows:

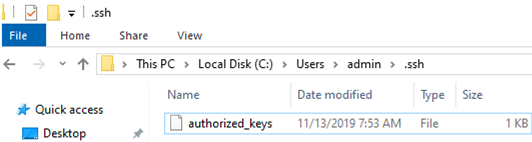
ssh-add.exe $ENV:UserProfile\.ssh\id\_rsa

## OpenSSH: Configuring Key-Based Authentication with Public Key on Windows

Now you need to copy your SSH public key to the SSH server. The SSH server in this example is a remote Windows 11 machine that has the OpenSSH service installed and configured.

You can check out the full guide “[How to configure an OpenSSH server on Windows?](http://woshub.com/connect-to-windows-via-ssh/)”.

Copy the **id\_ed25519.pub**file to the .**ssh** directory in the profile of the user you will use to connect to the SSH server. For example, I have an user1 account on my remote Windows 11 device, so I need to copy the key to C:\Users\user1\.ssh\authorized\_keys.



You can copy the public key to the SSH server from the client using SCP:

scp C:\Users\youruser\.ssh\id\_rsa.pub admin@192.168.1.15:c:\users\admin\.ssh\authorized\_keys

You can add multiple public keys to a single authorized\_keys file.

Public key authentication is disabled by default in the OpenSSH server on Windows. You can check this in the sshd\_config. The easiest way to get a list of allowed authentication methods in OpenSSH is to use the following PowerShell command ([Select-String is used as an analog of grep in PowerShell](http://woshub.com/grep-powershell-select-string-cmdlet/)):

cat "C:\ProgramData\ssh\sshd\_config"| Select-String "Authentication"

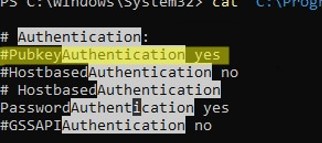
#PubkeyAuthentication yes

#HostbasedAuthentication no

#HostbasedAuthentication

PasswordAuthentication yes

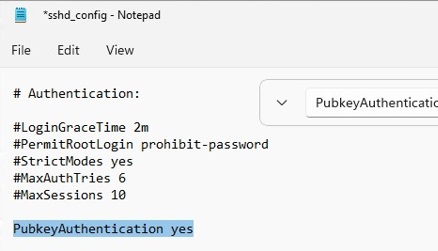
#GSSAPIAuthentication no

[](http://woshub.com/wp-content/uploads/2020/07/openssh-enable-public-key-authetication.jpg)

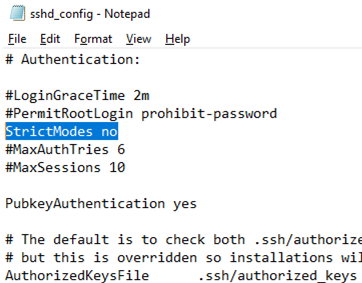
In this example, the **PubkeyAuthentication** line is commented out, which means that this authentication method is disabled. Open the sshd\_config file with notepad and uncomment the line:

Notepad C:\ProgramData\ssh\sshd\_config

PubkeyAuthentication yes

[](http://woshub.com/wp-content/uploads/2020/07/windows-enable-publickey-authentication-windows-sshd_config.jpg)

Also, you will have to disable the **StrictModes** option in the sshd\_config configuration file. By default, this mode is enabled and prevents SSH key-based authentication if private and public keys are not properly protected. Uncomment the line #StrictModes yes  and change it to StrictModes no

  
Now you can connect to your Windows SSH server without a password. If you have not set a password (passphrase) for the private key, you will automatically connect to your remote Windows host.

To connect to a remote host using a [native SSH client](http://woshub.com/using-native-ssh-client-windows/), use the following command:

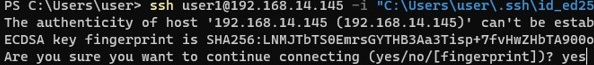
ssh (username)@(SSH server name or IP address)

For example:

ssh admin@192.168.1.15

It means that you want to connect to a remote SSH server with the IP address 192.168.1.15 under the user1 account. SSH Agent service will automatically try to use your private key to authenticate on a remote host.

* If you do not want to use the ssh-agent service to manage SSH keys, you can specify the path to the private key file to be used for the SSH authentication: ssh user1@192.168.1.15 -i "C:\Users\youuser\.ssh\id\_ed25519"
* To connect SSH host using a user account from an Active Directory domain, use the following format: ssh jsmith@woshub.com@192.168.1.15 -i <private\_key\_absolute\_path>

[](http://woshub.com/wp-content/uploads/2020/07/add-key-fingerprint-trusted-windows.jpg)

When connecting for the first time, you need to add the fingerprint of the SSH server key to the trusted list. Type yes -> Enter.

The authenticity of host '192.168.1.15 (192.168.1.15)' can't be established.

ECDSA key fingerprint is SHA256:xxxxxxx.

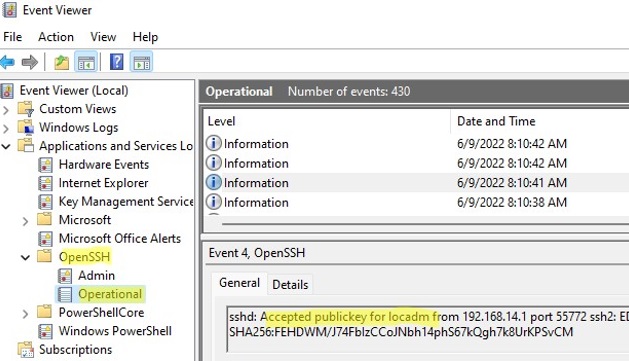
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

ETW logging is used in Windows OpenSSH to store SSH logs instead of plain text files. You can check the SSH key-based authentication logs in the Windows Event Viewer (Application and Services Logs -> OpenSSH -> Operational).

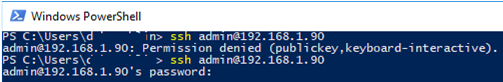
If the SSH connection with the private key is successful, the following event will appear in the OpenSSH log:

EventID 4

sshd: Accepted publickey for locadm from 192.168.15.20 port 55772 ssh2: ED25519 SHA256:xxxxxxx

[](http://woshub.com/wp-content/uploads/2020/07/windows-publickey-based-auth-event-viewer.jpg)

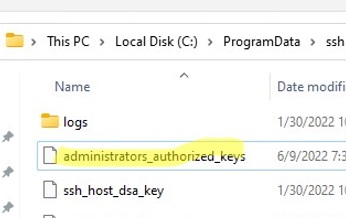
If you were not able to connect to your SSH server using your private key and you are still prompted to enter a password, it is likely that the user account you are trying to connect to is a member of the local Windows administrators group (the [group SID](http://woshub.com/hot-to-convert-sid-to-username-and-vice-versa/) is S-1-5-32-544). We will discuss it later.



## Logging Windows with SSH Key Under Administrative User

OpenSSH uses special key-based authentication settings for admin user accounts on Windows.

You need to use the C:\ProgramData\ssh\**administrators\_authorized\_keys** file instead of the **authorized\_keys** key in the user profile. Add your public SSH key to this text file (for security reasons, only the Administrators and SYSTEM groups should have permission to read this file).

[](http://woshub.com/wp-content/uploads/2020/07/administrators_authorized_keys-file-in-windows.jpg)

You can change the NTFS permissions on a file with:

* The [icacls](http://woshub.com/how-to-backup-and-restore-ntfs-permissions-using-icacls/) tool: icacls.exe "C:\ProgramData\ssh\administrators\_authorized\_keys" /inheritance:r /grant "Administrators:F" /grant "SYSTEM:F"
* or using the [Get-Acl and Set-Acl PowerShell cmdlets](http://woshub.com/manage-ntfs-permissions-powershell/): get-acl "$env:programdata\ssh\ssh\_host\_rsa\_key" | set-acl "$env:programdata\ssh\administrators\_authorized\_keys"

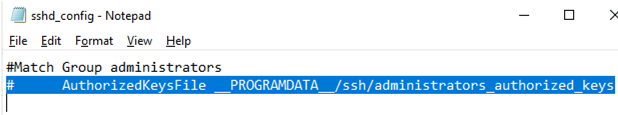
[change permissions on administrators_authorized_keys file in windows](http://woshub.com/wp-content/uploads/2020/07/change-permissions-administrators_authorized_keys-file-windows.jpg)

After that, SSH key authentication works even if the StrictModes is disabled.

In order to use the authorized\_keys file from a user profile and not to move the public key info to the administrators\_authorized\_keys file, you can comment out a line in the OpenSSH configuration file (C:\ProgramData\ssh\**sshd\_config**).

#Match Group administrators

# AuthorizedKeysFile \_\_PROGRAMDATA\_\_/ssh/administrators\_authorized\_keys



Additionally, you can disable SSH password login in the sshd\_config:

PasswordAuthentication no

Don’t forget to restart the sshd service after making the changes in the sshd\_config.

restart-service sshd

If you set PasswordAuthentication no, and configure SSH key authentication incorrectly, then an error will appear when connecting via ssh:

user1@192.168.13.15: Permission denied (publickey,keyboard-interactive).

[windows ssh login error: Permission denied publickey keyboard interactive](http://woshub.com/wp-content/uploads/2020/07/windows-Permission-denied-publickey-keyboard-interactive.jpg)

You can use the **PermitRootLogin** option in OpenSSH on Linux to restrict SSH root login. This directive is not applicable in Windows OpenSSH, and you must use the **DenyGroups** parameter to deny ssh login under admin accounts:

DenyGroups Administrators

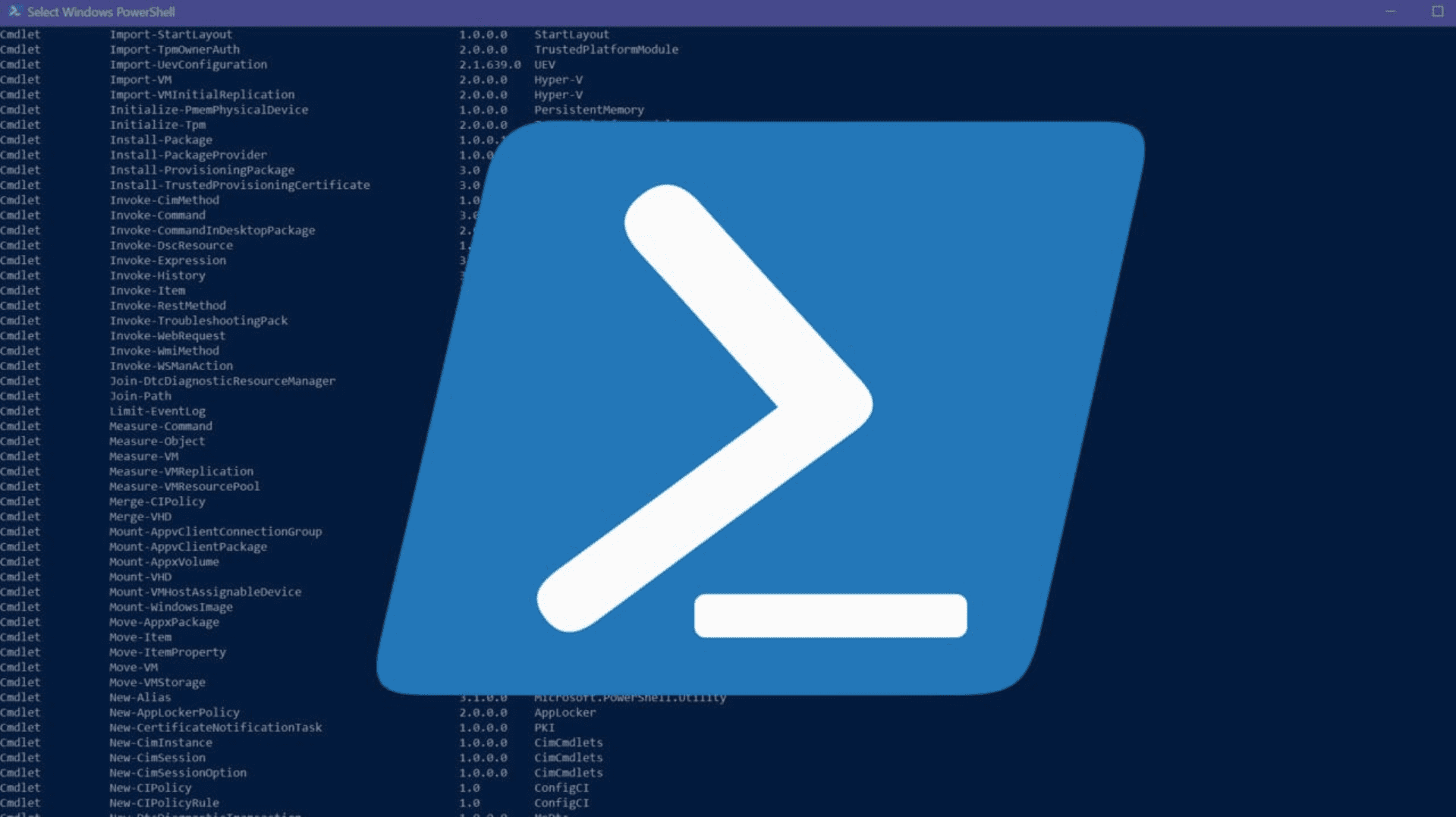
So, you have configured SSH authentication in Windows using a key pair.  Now you can use this authentication method to securely access remote servers, automatically [forward ports in the SSH tunnel](http://woshub.com/ssh-tunnel-port-forward-windows/), run scripts, and perform other automation tasks.

# The Ultimate Guide to Installing OpenSSH on Windows

[Michael Reinders](https://petri.com/author/mdreinders)

|

NOV 12, 2021



As a seasoned, or even new IT Pro, you’re likely an avid user of [Putty](https://www.putty.org/), using secure shell (SSH) to connect to Unix/Linux servers, computers, and even [Windows](https://petri.com/category/windows-10) machines for an efficient and secure remote command-line experience. Well, did you know Windows 10, Windows 11, and [Windows Server 2019](https://petri.com/category/windows-server) (and Windows Server 2022) include an open-source implementation of SSH?

**Table of Contents**

* [**How is SSH implemented in Windows?**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#How_is_SSH_implemented_in_Windows)
* [**Install OpenSSH using Windows Settings**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Install_OpenSSH_using_Windows_Settings)
* [**Install OpenSSH using PowerShell**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Install_OpenSSH_using_PowerShell)
* [**Start and configure OpenSSH Server**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Start_and_configure_OpenSSH_Server)
* [**Using SSH in Windows Terminal**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Using_SSH_in_Windows_Terminal)
* [**Connect to OpenSSH Server**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Connect_to_OpenSSH_Server)
* [**Uninstall OpenSSH using Windows Settings**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Uninstall_OpenSSH_using_Windows_Settings)
* [**Uninstall OpenSSH using PowerShell**](https://petri.com/the-ultimate-guide-to-installing-openssh-on-windows/#Uninstall_OpenSSH_using_PowerShell)

In this mega ‘how-to’ guide, you’ll learn how to install and configure [OpenSSH](https://petri.com/openssh-client-comes-out-of-beta-in-windows-april-2018-update) on Windows. Find out how to connect remotely to Linux, Unix, Oracle, Windows, Windows Server, and other operating systems via the secure command line.

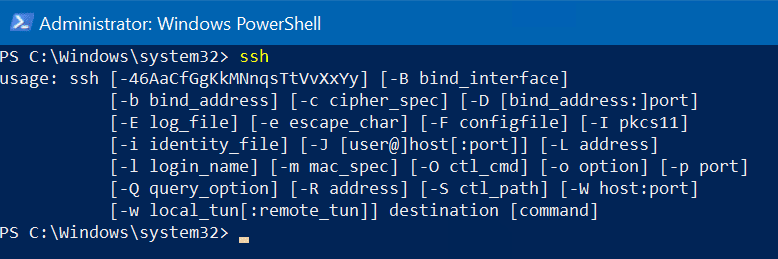
## How is SSH implemented in Windows?

There are two separate components of [OpenSSH](https://www.openssh.com/) in Windows – an SSH client and an SSH server. Microsoft implemented both in Windows using **OpenSSH Client** and **OpenSSH Server**respectively.

And there are also two main methods to install and uninstall these components in Windows. The OpenSSH Client feature is installed by default in higher-end versions of Windows 10 -11.

The Client is like the functionality of Putty. It allows you to make ‘client’ connections to other servers and devices using various secure protocols.

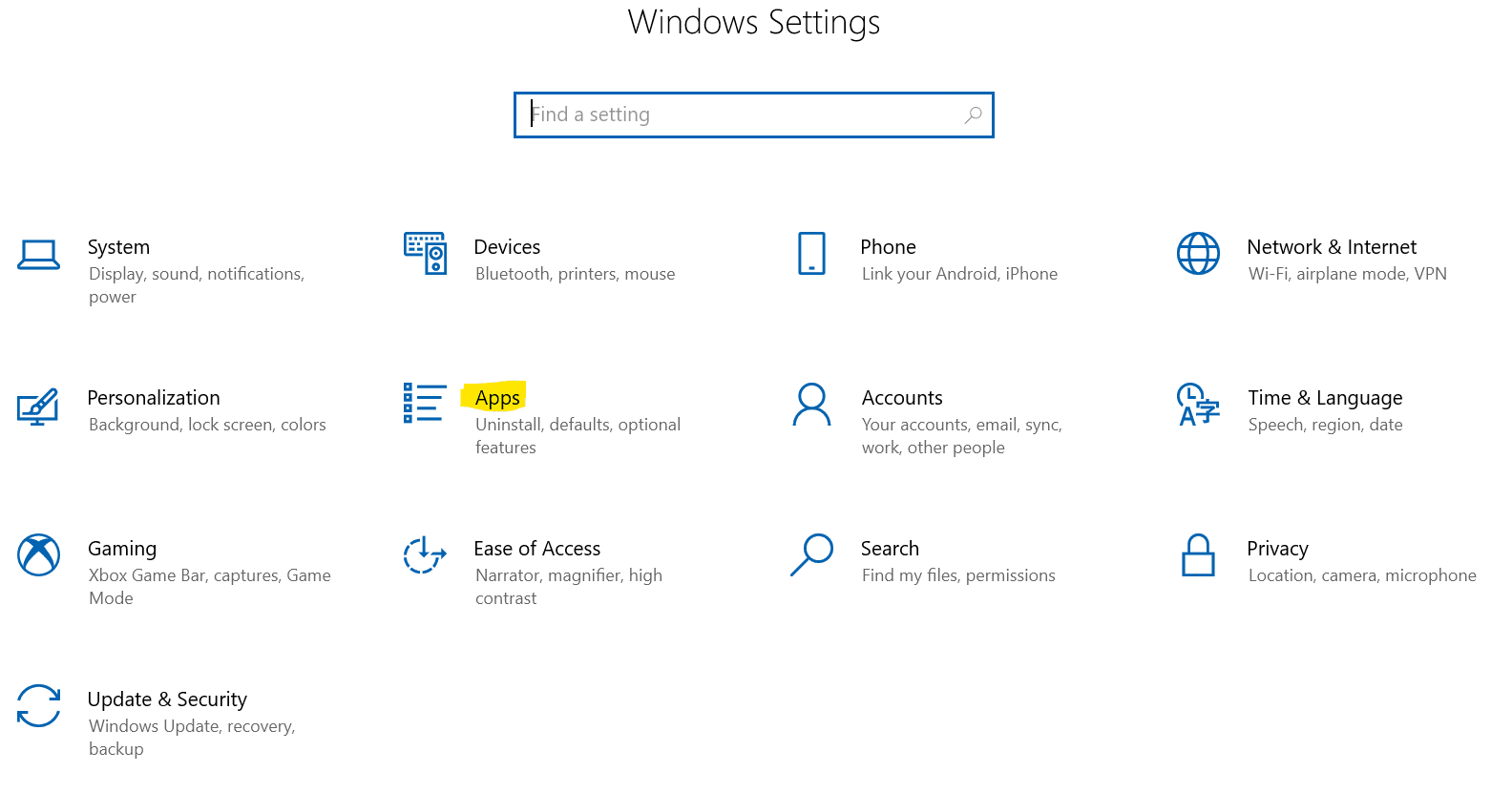
You can confirm if you have the client installed by opening a command prompt or PowerShell prompt and typing ‘ssh’ and hitting Enter. You will be provided with an overview of how to use the [ssh command](https://docs.microsoft.com/en-us/system-center/orchestrator/standard-activities/run-ssh-command?view=sc-orch-2019) if it is already installed.

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-134530.png)OpenSSH common output

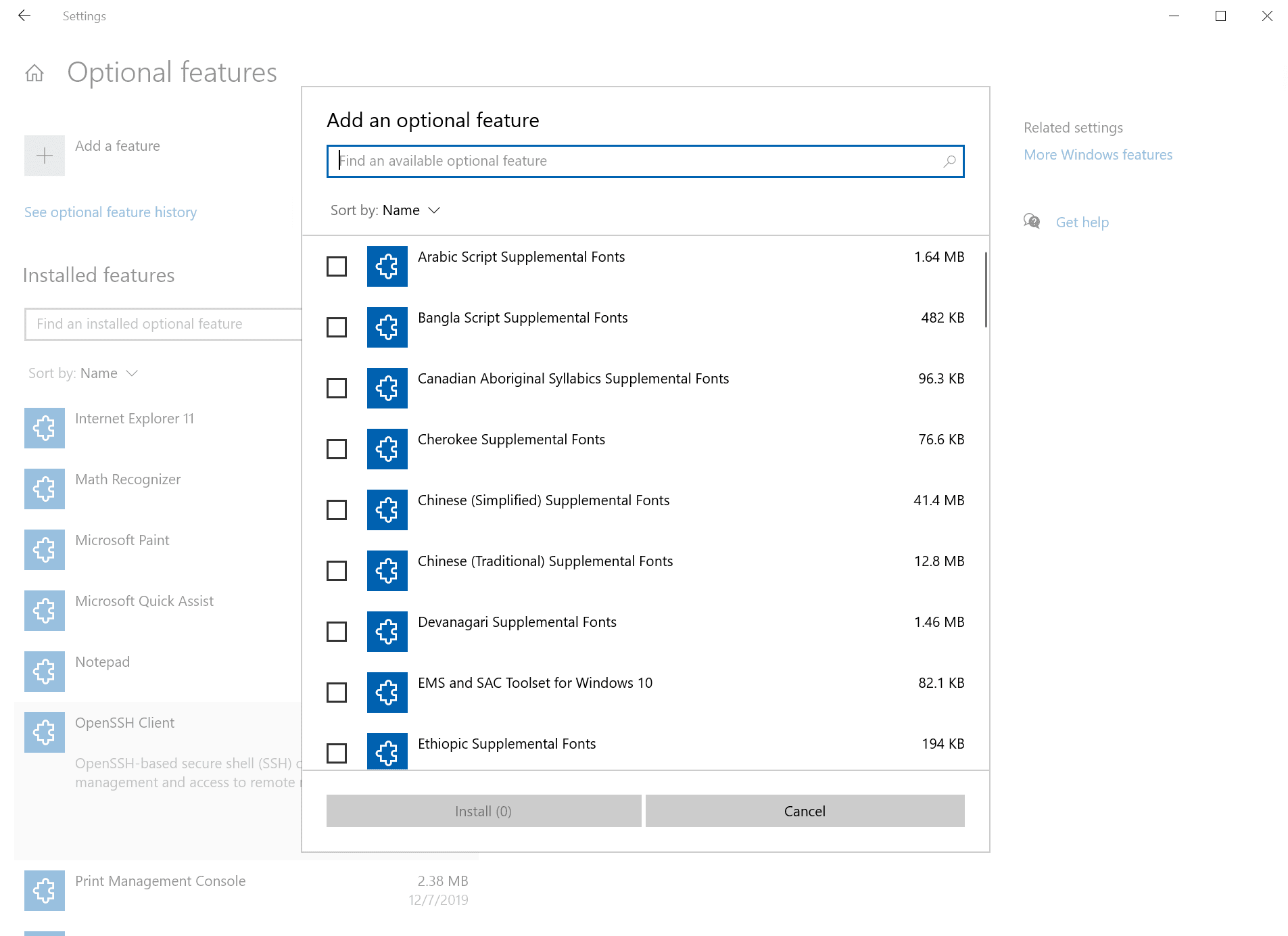
## Install OpenSSH using Windows Settings

To install **OpenSSH Client**, let’s first use the more modern approach – **Windows Settings**.

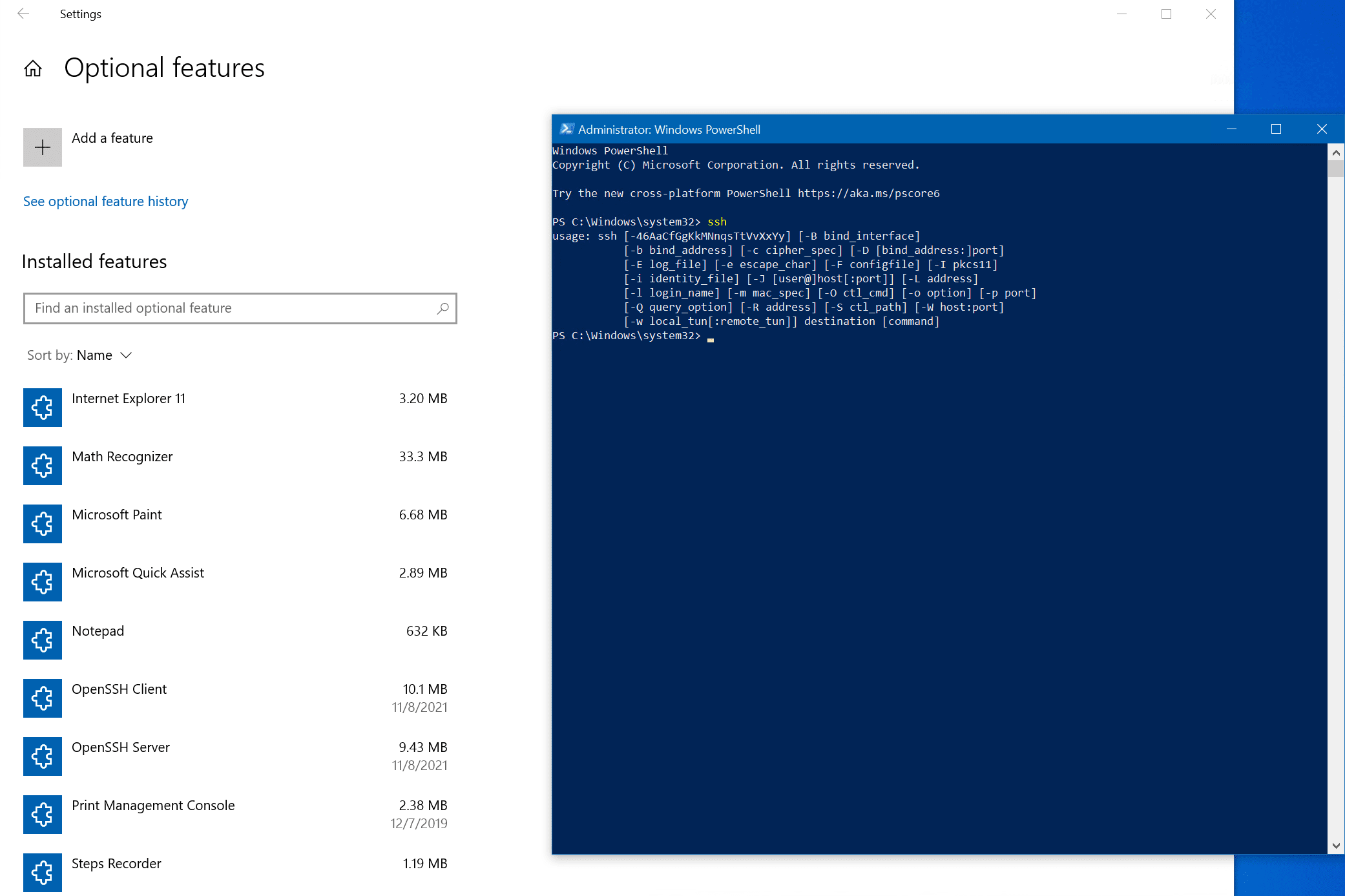
First, click the **Start** button, then click **Settings.** Next, click the ‘**Apps**‘ category.

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-134002.png)Windows Settings

Click the ‘**Add a feature’** ‘**+**‘ at the top of the **‘Optional features’**window.



Scroll down to ‘**OpenSSH Client’**, place a checkmark next to it and click the **‘Install’** button. Wait a few moments, and we’re good!

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-144549.png)OpenSSH Client Installed!

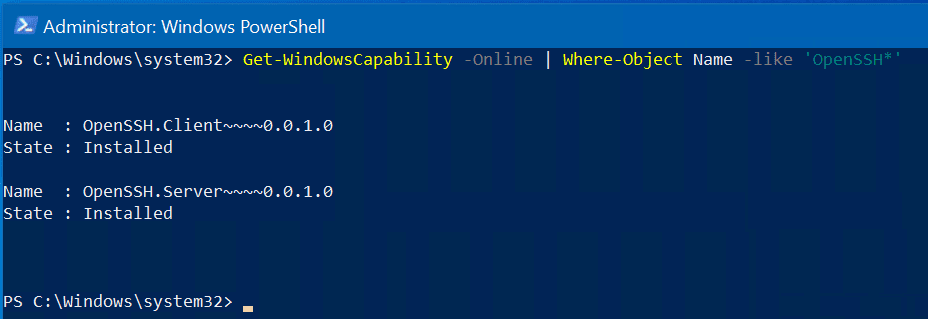
## Install OpenSSH using PowerShell

The other core method to installing OpenSSH is using PowerShell. Fire up an administrative PowerShell prompt and type in this command to install the ‘**OpenSSH Client’**feature.

Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

You can run this command to confirm the feature is installed.

Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH\*'

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-145443.png)OpenSSH Client (and Server) installed

## Start and configure OpenSSH Server

As you may have noticed, you can install **OpenSSH Client** and **OpenSSH Server** on Windows 10 and Windows Server 2019/2022 (You need at least Windows Server 2019 to host OpenSSH Server). I will now switch to one of my Windows Server 2022 servers and demonstrate how to start up the ‘Server’ part of the implementation and test connections from Windows 10.

Fire up another administrative PowerShell prompt and run these commands.

# Start the sshd service

Start-Service sshd

# OPTIONAL but recommended:

Set-Service -Name sshd -StartupType 'Automatic'

# Confirm the Firewall rule is configured. It should be created automatically by setup. Run the following to verify

**if** (!(Get-NetFirewallRule -Name "OpenSSH-Server-In-TCP" -ErrorAction SilentlyContinue | Select-Object Name, Enabled)) {

Write-Output "Firewall Rule 'OpenSSH-Server-In-TCP' does not exist, creating it..."

New-NetFirewallRule -Name 'OpenSSH-Server-In-TCP' -DisplayName 'OpenSSH Server (sshd)' -Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22

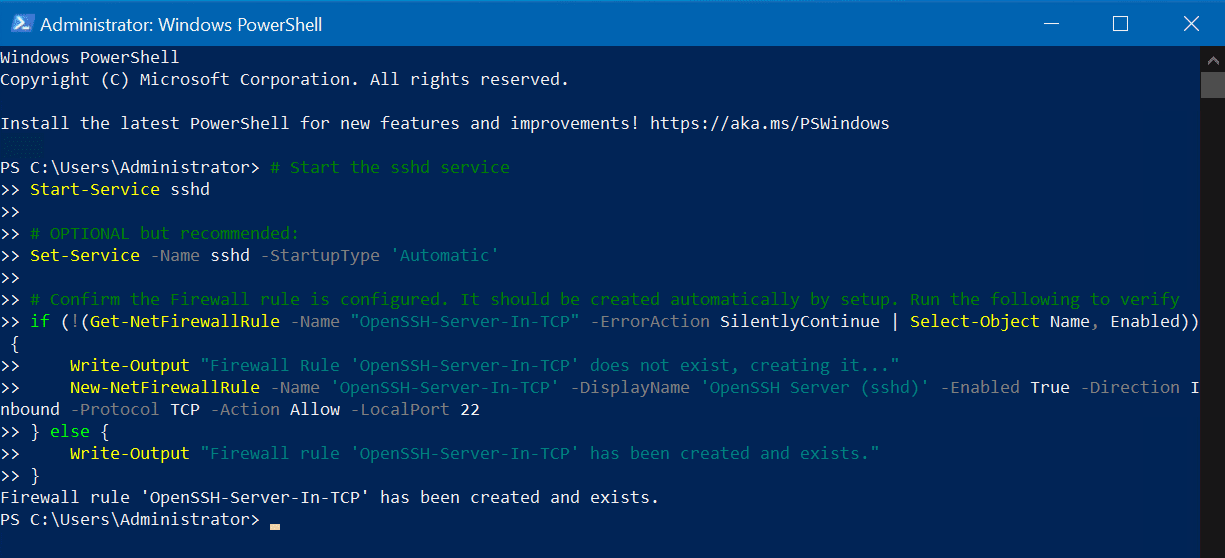
} **else** {

Write-Output "Firewall rule 'OpenSSH-Server-In-TCP' has been created and exists."

}

This will start the secure SSH service (Server), set its service settings to **‘Automatic’** so it runs every time the server boots, and verify all the appropriate Windows Firewall rules are in place to allow client connections on TCP localport 22 through Windows Server’s built-in Windows Defender software-based firewall.

Success!

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-150242.png)OpenSSH Started and Configured

# Tutorial: SSH in Windows Terminal

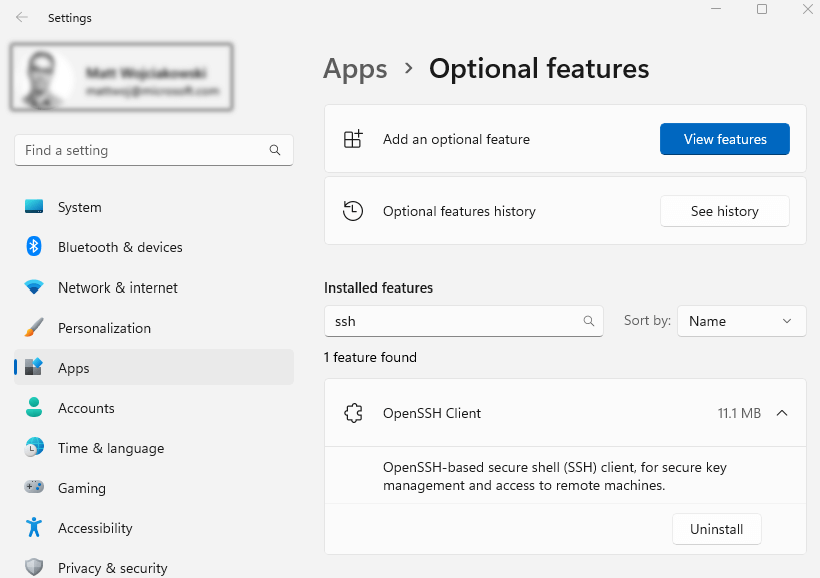
Graphical user interface, application

Description automatically generatedWindows has a built-in SSH client that you can use in Windows Terminal. In this tutorial, you'll learn how to set up a profile in Windows Terminal that uses SSH.

**Access Windows SSH Client**

The latest builds of Windows 10 and Windows 11 include a built-in SSH server and client that are based on OpenSSH, a connectivity tool for remote sign-in that uses the SSH protocol. OpenSSH encrypts all traffic between client and server to eliminate eavesdropping, connection hijacking, and other attacks.

By default, the OpenSSH client will be located in the directory: C:\Windows\System32\OpenSSH. You can also check that it is installed in Windows Settings > Apps > Optional features, then search for "OpenSSH" in your installed features.



**Note**

Windows Terminal version 1.XX+ can dynamically generate profiles to connect to the SSH hosts within your [**OpenSSH config file**](https://man.openbsd.org/ssh_config).

**Create a profile**

You can start an SSH session in your command prompt by executing ssh user@machine and you will be prompted to enter your password. You can create a Windows Terminal profile that does this on startup by adding the command line setting to a profile in your [settings.json file](https://learn.microsoft.com/en-us/windows/terminal/install" \l "settings-json-file) inside the list of profile objects.

JSONCopy

{

"name": "user@machine ssh profile",

"commandline": "ssh user@machine"

}

For more information, see:

* [Windows Terminal Profile - General settings](https://learn.microsoft.com/en-us/windows/terminal/customize-settings/profile-general)

**Specify starting directory**

To specify the starting directory for a ssh session invoked by Windows Terminal, you can use this command:

JSONCopy

{

"commandline": "ssh -t bob@foo \"cd /data/bob && exec bash -l\""

}

The -t flag forces pseudo-terminal allocation. This can be used to execute arbitrary screen-based programs on a remote machine, e.g. when implementing menu services. You will need to use escaped double quotes as bourne shell derivatives don't do any additional parsing for a string in single quotes.

For more information, see:

* [GH Issue: How to specify the starting directory for a ssh session?](https://github.com/MicrosoftDocs/terminal/issues/25)
* [StackOverflow: How can I ssh directly to a particular directory?](https://stackoverflow.com/questions/626533/how-can-i-ssh-directly-to-a-particular-directory)

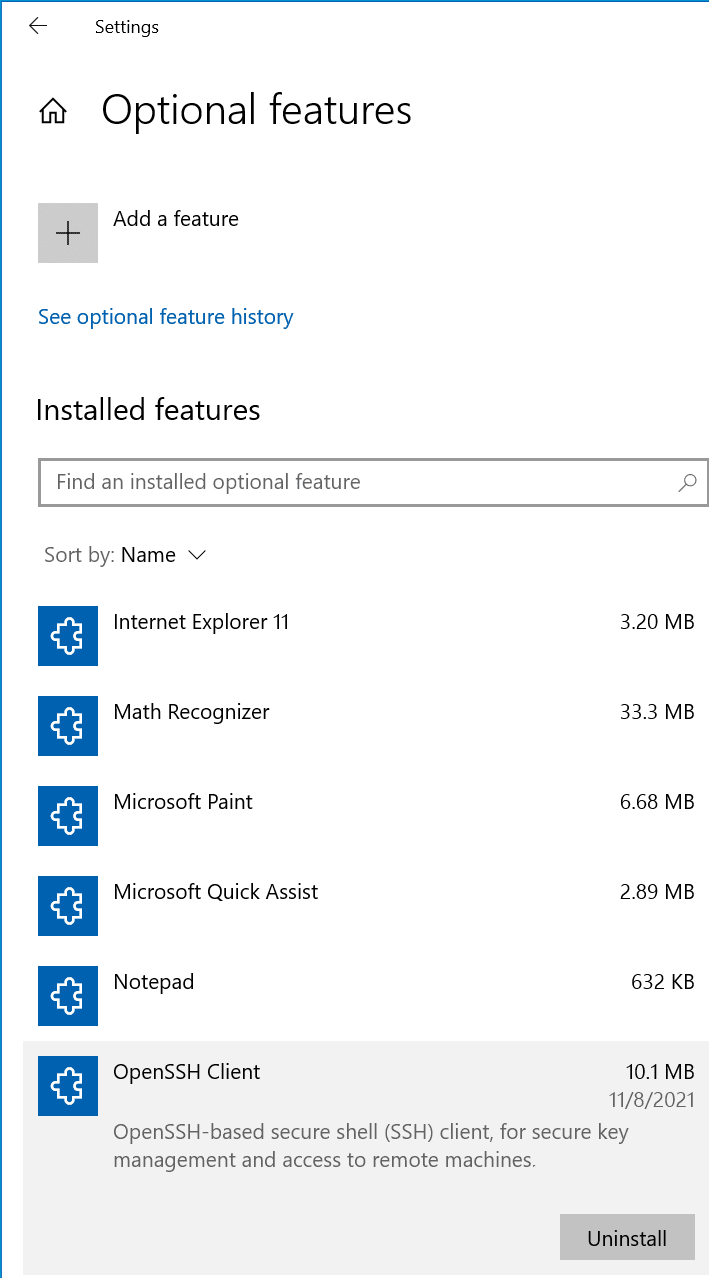
**Resources**

* [How to Enable and Use Windows 10’s New Built-in SSH Commands](https://www.howtogeek.com/336775/how-to-enable-and-use-windows-10s-built-in-ssh-commands/)

[Business Hosting](https://www.hostwinds.com/hosting/business)

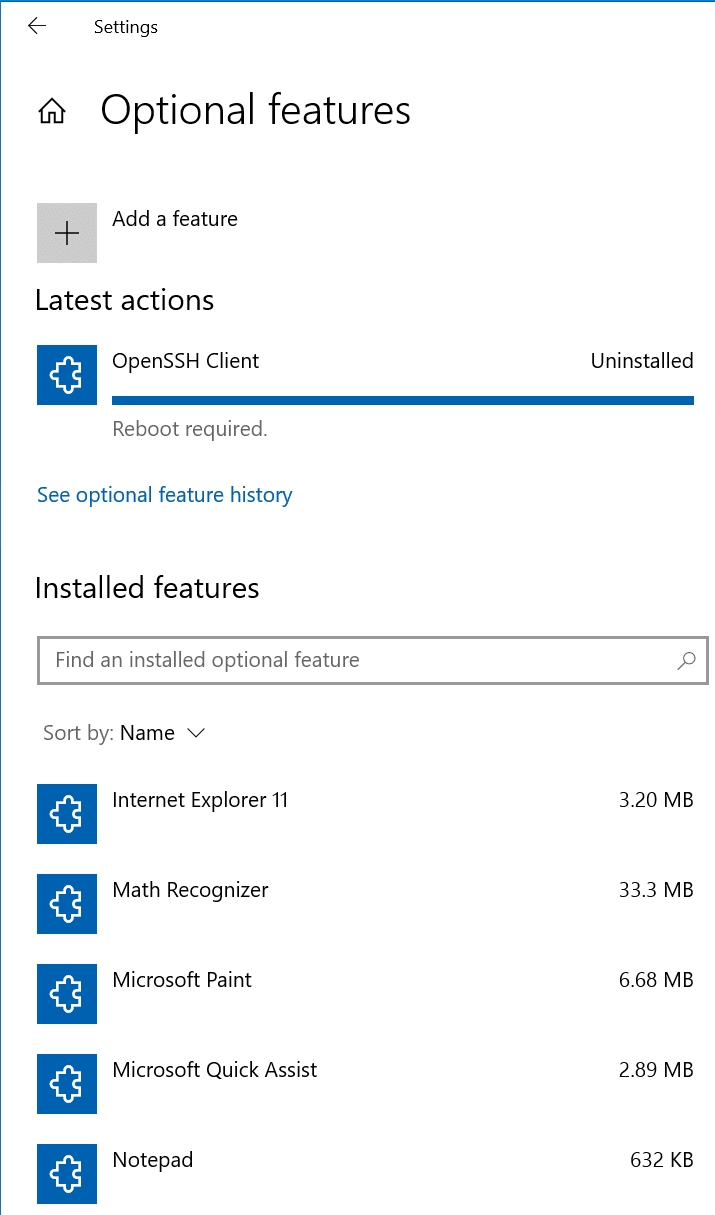
## Uninstall OpenSSH using Windows Settings

If you ever need to uninstall OpenSSH components for security, compliance, or any other reason, it’s straightforward via Windows Settings. Let’s walk you through.

First, click the **Start** button, and click on **Settings.**Click the **Apps** category heading, then **Optional Features**.[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-160842.png)

Ready to Uninstall OpenSSH Client

Click ‘**OpenSSH Client**‘ and click the **Uninstall** button.

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-161040.png)

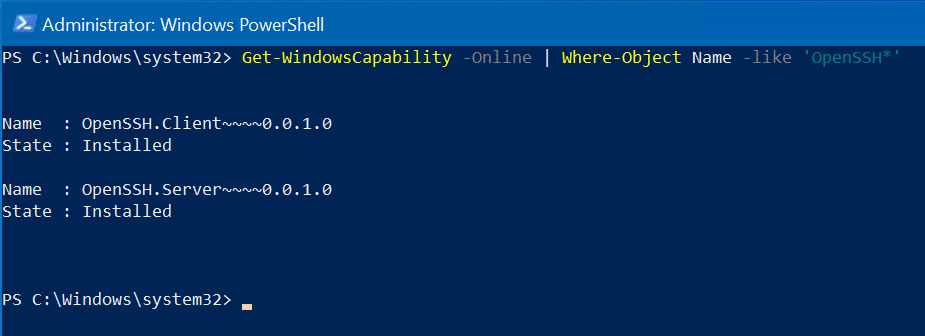
OpenSSH Uninstalled

Go ahead and reboot your computer if it prompts you to (assuming you can, should, and no one will yell at you for **Rebooting the Exchange Server!!!**) One of my favorite online IT Pro videos to watch from many years ago. Some of you will definitely resonate… (The Website is Down #1)

## Uninstall OpenSSH using PowerShell

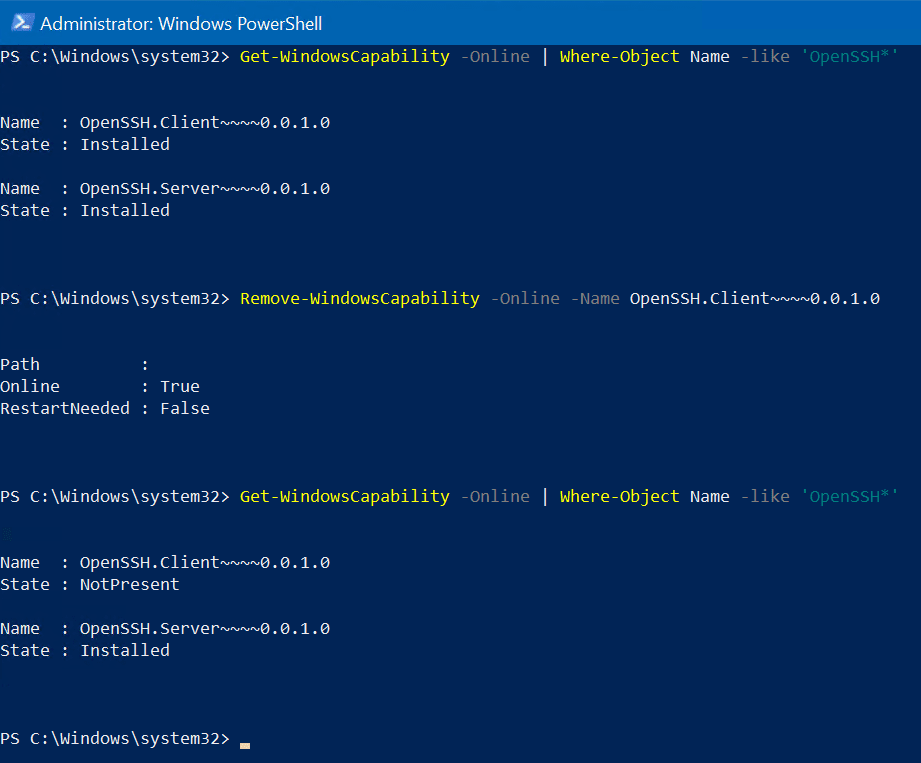
There are strikingly similar PowerShell commands to run to uninstall OpenSSH features in Windows compared to Installing them. I know, right? Mesmerizing. Go ahead and run this command to validate which OpenSSH components are installed on your system.

Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH\*'

[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-161903.png)List of which OpenSSH components are installed

Run the following command to uninstall OpenSSH Client from your computer.

Remove-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

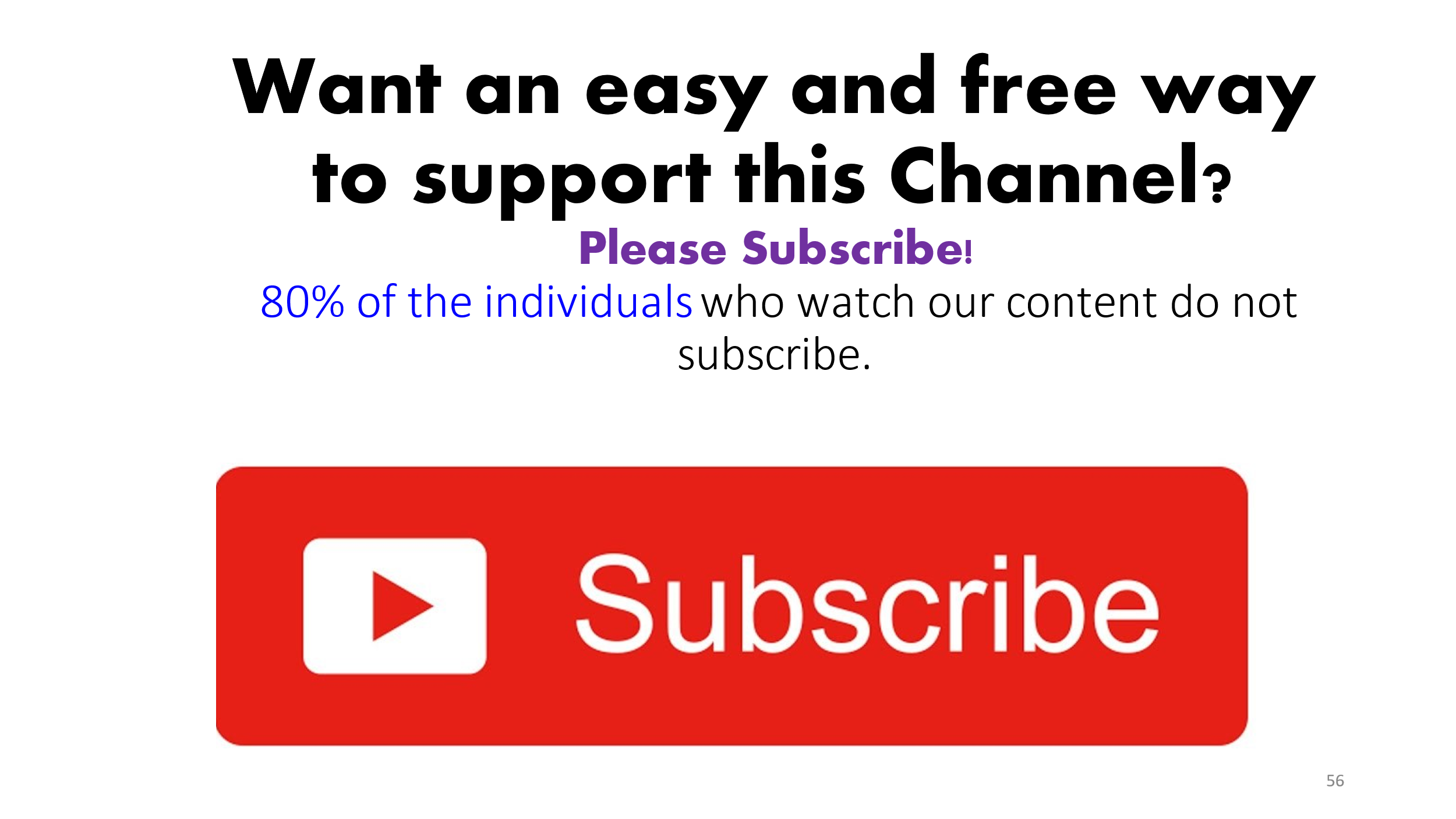
[](https://petri-media.s3.amazonaws.com/2021/11/Screenshot-2021-11-08-162404.png)After uninstall of OpenSSH Client and confirmation

As you can see, I ran the ‘**Get-WindowsCapability’** command again after the feature was uninstalled to confirm. All looks good!

There, that wasn’t so bad. Honestly, it’s pretty straightforward to get up and running fast with OpenSSH in Windows.

# PowerShell remoting over SSH

**Overview**



PowerShell remoting normally uses WinRM for connection negotiation and data transport.

SSH is now available for Linux and Windows platforms and allows true multiplatform PowerShell remoting.

WinRM provides a robust hosting model for PowerShell remote sessions. SSH-based remoting doesn't currently support remote endpoint configuration and **Just Enough Administration (JEA).**

SSH remoting lets you do basic PowerShell session remoting between Windows and Linux computers. SSH remoting creates a PowerShell host process on the target computer as an SSH subsystem.

Eventually we'll implement a general hosting model, similar to WinRM, to support endpoint configuration and JEA.

The New-PSSession, Enter-PSSession, and Invoke-Command cmdlets now have a new parameter set to support this new remoting connection.

[-HostName <string>] [-UserName <string>] [-KeyFilePath <string>]

To create a remote session, you specify the target computer with the **HostName** parameter and provide the user name with **UserName**. When running the cmdlets interactively, you're prompted for a password. You can also use SSH key authentication using a private key file with the **KeyFilePath** parameter. **Creating keys for SSH authentication varies by platform.**

**General setup information**

PowerShell 6 or higher, and SSH must be installed on all computers.

* Install both the SSH client (ssh.exe)
* and server (sshd.exe) so that you can remote to and from the computers.
* OpenSSH for Windows is now available in Windows 10 build 1809 and Windows Server 2019. For more information, see [Manage Windows with OpenSSH](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview). For Linux, install SSH, including sshd server, that's appropriate for your platform.
* You also need to install PowerShell from GitHub to get the SSH remoting feature.
* The SSH server must be configured to create an SSH subsystem to host a PowerShell process on the remote computer. And, you must enable **password** or **key-based** authentication.

**Install the SSH service on a Windows computer**

1. Install the latest version of PowerShell. For more information, see [Installing PowerShell on Windows](https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3#msi).

You can confirm that PowerShell has SSH remoting support by listing the New-PSSession parameter sets. You'll notice there are parameter set names that begin with **SSH**. Those parameter sets include **SSH** parameters.

(Get-Command New-PSSession).ParameterSets.Name

OutputCopy

Name

----

SSHHost

SSHHostHashParam

1. Install the latest Win32 OpenSSH. For installation instructions, see [Getting started with OpenSSH](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse).

**Note**

If you want to set PowerShell as the default shell for OpenSSH, see [**Configuring Windows for OpenSSH**](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_server_configuration).

1. Edit the sshd\_config file located at $env:ProgramData\ssh.

Make sure password authentication is enabled:

Copy

PasswordAuthentication yes

Create the SSH subsystem that hosts a PowerShell process on the remote computer:

Copy

Subsystem powershell c:/progra~1/powershell/7/pwsh.exe -sshs -nologo

**Note**

Starting in PowerShell 7.3, you no longer need to use the -nologo parameter when running PowerShell in SSH server mode.

**Note**

The default location of the PowerShell executable is c:/progra~1/powershell/7/pwsh.exe. The location can vary depending on how you installed PowerShell.

You must use the 8.3 short name for any file paths that contain spaces. There's a bug in OpenSSH for Windows that prevents spaces from working in subsystem executable paths. For more information, see this [**GitHub issue**](https://github.com/PowerShell/Win32-OpenSSH/issues/784).

The 8.3 short name for the Program Files folder in Windows is usually Progra~1. However, you can use the following command to make sure:

PowerShellCopy

Get-CimInstance Win32\_Directory -Filter 'Name="C:\\Program Files"' |

Select-Object EightDotThreeFileName

OutputCopy

EightDotThreeFileName

---------------------

c:\progra~1

Optionally, enable key authentication:

Copy

PubkeyAuthentication yes

For more information, see [Managing OpenSSH Keys](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_keymanagement).

1. Restart the **sshd** service.

PowerShellCopy

Restart-Service sshd

1. Add the path where OpenSSH is installed to your Path environment variable. For example, C:\Program Files\OpenSSH\. This entry allows for the ssh.exe to be found.

**Install the SSH service on an Ubuntu Linux computer**

1. Install the latest version of PowerShell, see [Installing PowerShell on Ubuntu](https://learn.microsoft.com/en-us/powershell/scripting/install/install-ubuntu?view=powershell-7.3).
2. Install [Ubuntu OpenSSH Server](https://ubuntu.com/server/docs/service-openssh).

BashCopy

sudo apt install openssh-client

sudo apt install openssh-server

1. Edit the sshd\_config file at location /etc/ssh.

Make sure password authentication is enabled:

Copy

PasswordAuthentication yes

Optionally, enable key authentication:

Copy

PubkeyAuthentication yes

For more information about creating SSH keys on Ubuntu, see the manpage for [ssh-keygen](http://manpages.ubuntu.com/manpages/xenial/man1/ssh-keygen.1.html).

Add a PowerShell subsystem entry:

Copy

Subsystem powershell /usr/bin/pwsh -sshs -nologo

**Note**

The default location of the PowerShell executable is /usr/bin/pwsh. The location can vary depending on how you installed PowerShell.

**Note**

Starting in PowerShell 7.3, you no longer need to use the -nologo parameter when running PowerShell in SSH server mode.

1. Restart the **ssh** service.

BashCopy

sudo systemctl restart sshd.service

**Install the SSH service on a macOS computer**

1. Install the latest version of PowerShell. For more information, [Installing PowerShell on macOS](https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-macos?view=powershell-7.3).

Make sure SSH Remoting is enabled by following these steps:

* 1. Open System Preferences.
  2. Click on Sharing.
  3. Check Remote Login to set Remote Login: On.
  4. Allow access to the appropriate users.

1. Edit the sshd\_config file at location /private/etc/ssh/sshd\_config.

Use a text editor such as **nano**:

BashCopy

sudo nano /private/etc/ssh/sshd\_config

Make sure password authentication is enabled:

Copy

PasswordAuthentication yes

Add a PowerShell subsystem entry:

Copy

Subsystem powershell /usr/local/bin/pwsh -sshs -nologo

**Note**

The default location of the PowerShell executable is /usr/local/bin/pwsh. The location can vary depending on how you installed PowerShell.

**Note**

Starting in PowerShell 7.3, you no longer need to use the -nologo parameter when running PowerShell in SSH server mode.

Optionally, enable key authentication:

Copy

PubkeyAuthentication yes

1. Restart the **sshd** service.

BashCopy

sudo launchctl stop com.openssh.sshd

sudo launchctl start com.openssh.sshd

**Authentication**

PowerShell remoting over SSH relies on the authentication exchange between the SSH client and SSH service and doesn't implement any authentication schemes itself. The result is that any configured authentication schemes including multi-factor authentication are handled by SSH and independent of PowerShell. For example, you can configure the SSH service to require public key authentication and a one-time password for added security. Configuration of multi-factor authentication is outside the scope of this documentation. Refer to documentation for SSH on how to correctly configure multi-factor authentication and validate it works outside of PowerShell before attempting to use it with PowerShell remoting.

**Note**

Users retain the same privileges in remote sessions. Meaning, Administrators have access to an elevated shell, and normal users will not.

**PowerShell remoting example**

The easiest way to test remoting is to try it on a single computer. In this example, we create a remote session back to the same Linux computer. We're using PowerShell cmdlets interactively so we see prompts from SSH asking to verify the host computer and prompting for a password. You can do the same thing on a Windows computer to ensure remoting is working. Then, remote between computers by changing the host name.

**Linux to Linux**

PowerShellCopy

$session = New-PSSession -HostName UbuntuVM1 -UserName TestUser

OutputCopy

The authenticity of host 'UbuntuVM1 (9.129.17.107)' can't be established.

ECDSA key fingerprint is SHA256:2kCbnhT2dUE6WCGgVJ8Hyfu1z2wE4lifaJXLO7QJy0Y.

Are you sure you want to continue connecting (yes/no)?

TestUser@UbuntuVM1s password:

PowerShellCopy

$session

OutputCopy

Id Name ComputerName ComputerType State ConfigurationName Availability

-- ---- ------------ ------------ ----- ----------------- ------------

1 SSH1 UbuntuVM1 RemoteMachine Opened DefaultShell Available

PowerShellCopy

Enter-PSSession $session

OutputCopy

[UbuntuVM1]: PS /home/TestUser> uname -a

Linux TestUser-UbuntuVM1 4.2.0-42-generic 49~16.04.1-Ubuntu SMP Wed Jun 29 20:22:11 UTC 2016 x86\_64 x86\_64 x86\_64 GNU/Linux

[UbuntuVM1]: PS /home/TestUser> Exit-PSSession

PowerShellCopy

Invoke-Command $session -ScriptBlock { Get-Process powershell }

OutputCopy

Handles NPM(K) PM(K) WS(K) CPU(s) Id SI ProcessName PSComputerName

------- ------ ----- ----- ------ -- -- ----------- --------------

0 0 0 19 3.23 10635 635 powershell UbuntuVM1

0 0 0 21 4.92 11033 017 powershell UbuntuVM1

0 0 0 20 3.07 11076 076 powershell UbuntuVM1

**Linux to Windows**

PowerShellCopy

Enter-PSSession -HostName WinVM1 -UserName PTestName

Copy

PTestName@WinVM1s password:

PowerShellCopy

[WinVM1]: PS C:\Users\PTestName\Documents> cmd /c ver

OutputCopy

Microsoft Windows [Version 10.0.10586]

**Windows to Windows**

PowerShellCopy

C:\Users\PSUser\Documents>pwsh.exe

OutputCopy

PowerShell

Copyright (c) Microsoft Corporation. All rights reserved.

PowerShellCopy

$session = New-PSSession -HostName WinVM2 -UserName PSRemoteUser

OutputCopy

The authenticity of host 'WinVM2 (10.13.37.3)' can't be established.

ECDSA key fingerprint is SHA256:kSU6slAROyQVMEynVIXAdxSiZpwDBigpAF/TXjjWjmw.

Are you sure you want to continue connecting (yes/no)?

Warning: Permanently added 'WinVM2,10.13.37.3' (ECDSA) to the list of known hosts.

PSRemoteUser@WinVM2's password:

PowerShellCopy

$session

OutputCopy

Id Name ComputerName ComputerType State ConfigurationName Availability

-- ---- ------------ ------------ ----- ----------------- ------------

1 SSH1 WinVM2 RemoteMachine Opened DefaultShell Available

PowerShellCopy

Enter-PSSession -Session $session

OutputCopy

[WinVM2]: PS C:\Users\PSRemoteUser\Documents> $PSVersionTable

Name Value

---- -----

PSEdition Core

PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}

SerializationVersion 1.1.0.1

BuildVersion 3.0.0.0

CLRVersion

PSVersion 6.0.0-alpha

WSManStackVersion 3.0

PSRemotingProtocolVersion 2.3

GitCommitId v6.0.0-alpha.17

[WinVM2]: PS C:\Users\PSRemoteUser\Documents>

**Limitations**

* The **sudo** command doesn't work in a remote session to a Linux computer.
* PSRemoting over SSH doesn't support Profiles and doesn't have access to $PROFILE. Once in a session, you can load a profile by dot sourcing the profile with the full filepath. This isn't related to SSH profiles. You can configure the SSH server to use PowerShell as the default shell and to load a profile through SSH. See the SSH documentation for more information.
* Prior to PowerShell 7.1, remoting over SSH didn't support second-hop remote sessions. This capability was limited to sessions using WinRM. PowerShell 7.1 allows Enter-PSSession and Enter-PSHostProcess to work from within any interactive remote session.

**See also**

* [Installing PowerShell on Linux](https://learn.microsoft.com/en-us/powershell/scripting/install/install-ubuntu?view=powershell-7.3)
* [Installing PowerShell on macOS](https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-macos?view=powershell-7.3)
* [Installing PowerShell on Windows](https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3#msi)
* [Manage Windows with OpenSSH](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview)
* [Managing OpenSSH Keys](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_keymanagement)
* [Ubuntu SSH](https://ubuntu.com/server/docs/service-openssh)

**Recommended content**

**[WS-Management (WSMan) Remoting in PowerShell - PowerShell](https://learn.microsoft.com/en-us/powershell/scripting/learn/remoting/wsman-remoting-in-powershell-core?source=recommendations)**

Remoting in PowerShell using WSMan

**[Clear-History (Microsoft.PowerShell.Core) - PowerShell](https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/clear-history?source=recommendations)**

Clear-History deletes the command history from a PowerShell session. Each PowerShell session has its own command history. To display the command history, use the Get-History cmdlet. By default, Clear-History deletes the entire command history from a PowerShell session. You can use parameters with Clear-History to delete selected commands. Clear-History does not clear the PSReadLine command history file. The PSReadLine module stores a history file that contains every PowerShell command from every PowerShell

**[Get-History (Microsoft.PowerShell.Core) - PowerShell](https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-history?source=recommendations)**

The Get-History cmdlet gets the session history, that is, the list of commands entered during the current session. PowerShell automatically maintains a history of each session. The number of entries in the session history is determined by the value of the $MaximumHistoryCount preference variable. Beginning in Windows PowerShell 3.0, the default value is 4096. By default, history files are saved in the home directory, but you can save the file in any location. For more information about the history features

**[Installing PowerShell on Red Hat Enterprise Linux (RHEL) - PowerShell](https://learn.microsoft.com/en-us/powershell/scripting/install/install-rhel?source=recommendations)**

Information about installing PowerShell on Red Hat Enterprise Linux (RHEL)

Whoever of you loves life

and desires to see many good days,

keep your tongue from evil

and your lips from telling lies.

Turn from evil and do good;

seek peace and pursue it.

# 

# Just Enough Administration

Just Enough Administration (JEA) is a security technology that enables delegated administration for anything managed by PowerShell. With JEA, you can:

* **Reduce the number of administrators on your machines** using virtual accounts or group-managed service accounts to perform privileged actions on behalf of regular users.
* **Limit what users can do** by specifying which cmdlets, functions, and external commands they can run.
* **Better understand what your users are doing** with transcripts and logs that show you exactly which commands a user executed during their session.

**Why is JEA important?**

Highly privileged accounts used to administer your servers pose a serious security risk. Should an attacker compromise one of these accounts, they could launch [lateral attacks](https://aka.ms/pth) across your organization. Each compromised account gives an attacker access to even more accounts and resources, and puts them one step closer to stealing company secrets, launching a denial-of-service attack, and more.

It's not always easy to remove administrative privileges, either. Consider the common scenario where the DNS role is installed on the same machine as your Active Directory Domain Controller. Your DNS administrators require local administrator privileges to fix issues with the DNS server. But to do so, you must make them members of the highly privileged **Domain Admins** security group. This approach effectively gives DNS Administrators control over your whole domain and access to all resources on that machine.

JEA addresses this problem through the principle of **Least Privilege**. With JEA, you can configure a management endpoint for DNS administrators that gives them access only to the PowerShell commands they need to get their job done. This means you can provide the appropriate access to repair a poisoned DNS cache or restart the DNS server without unintentionally giving them rights to Active Directory, or to browse the file system, or run potentially dangerous scripts. Better yet, when the JEA session is configured to use temporary privileged virtual accounts, your DNS administrators can connect to the server using **non-admin** credentials and still run commands that typically require admin privileges. JEA enables you to remove users from widely privileged local/domain administrator roles and carefully control what they can do on each machine.

# How to Use PuTTY on Windows

This page is about the **PuTTY SSH client on Windows**. For information about PuTTY on Mac, see the [**PuTTY Mac page**](https://www.ssh.com/ssh/putty/mac/). For PuTTY on Linux, see the [**PuTTY Linux page**](https://www.ssh.com/ssh/putty/linux/).

This page explains how to use the PuTTY terminal window on Windows. How to configure PuTTY, how to create and save profiles, and what configuration options to change. Advanced topics, such as configuring public key authentication, are also addressed.

## Getting and installing

You can download a copy of the software for the Windows platform from the [**download page**](https://www.ssh.com/ssh/putty/download). Detailed installation instructions are provided on the [**installation instructions**](https://www.ssh.com/ssh/putty/windows/install) page.

## Running PuTTY and connecting to a server

If you selected to create a desktop icon during installation, you can start the software simply by (double-)clicking on the icon. Otherwise, open the software from the Windows **Start** menu.

When the software starts, a window titled **PuTTY Configuration** should open. This window has a configuration pane on the left, a **Host Name (or IP address)** field and other options in the middle, and a pane for saving session profiles in the lower right area.

For simple use, all you need to do is to enter the domain name or IP address of the host you want to connect to in the **Host Name** field and click **Open** (or press Enter). A domain name looks like students.example.edu. An IP address looks something like 78.99.129.32.

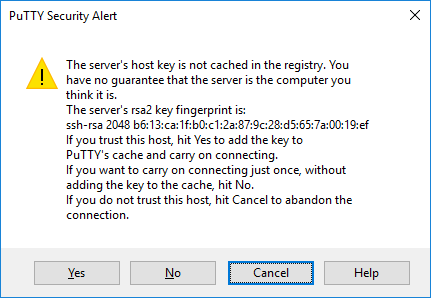
## What if you don't have a server

If you don't have a server to connect to, you can try **[Tectia SSH](https://www.ssh.com/products/tectia-ssh/)** on Windows or [**OpenSSH**](https://www.ssh.com/ssh/openssh/) on Linux.

## Security alert dialog box

When you connect to a server for the first time, you are likely to see a PuTTY Security Alert dialog about the server's host key not being cached in the registry. This is normal when you are connecting to a server for the first time. If you ever get this with a server, it could mean that someone is trying to attack your connection and steal your password using a [**man-in-the-middle attack**](https://www.ssh.com/attack/man-in-the-middle).

But as said, the first time you connect, this is normal, and you should just click **Yes**. If you want to be fancy, you can check the displayed key fingerprint and make sure it is the same that is used by the server. In real life, almost nobody does that. It is more secure to use a proper [**SSH key management solution**](https://www.ssh.com/products/universal-ssh-key-manager/) anyway.



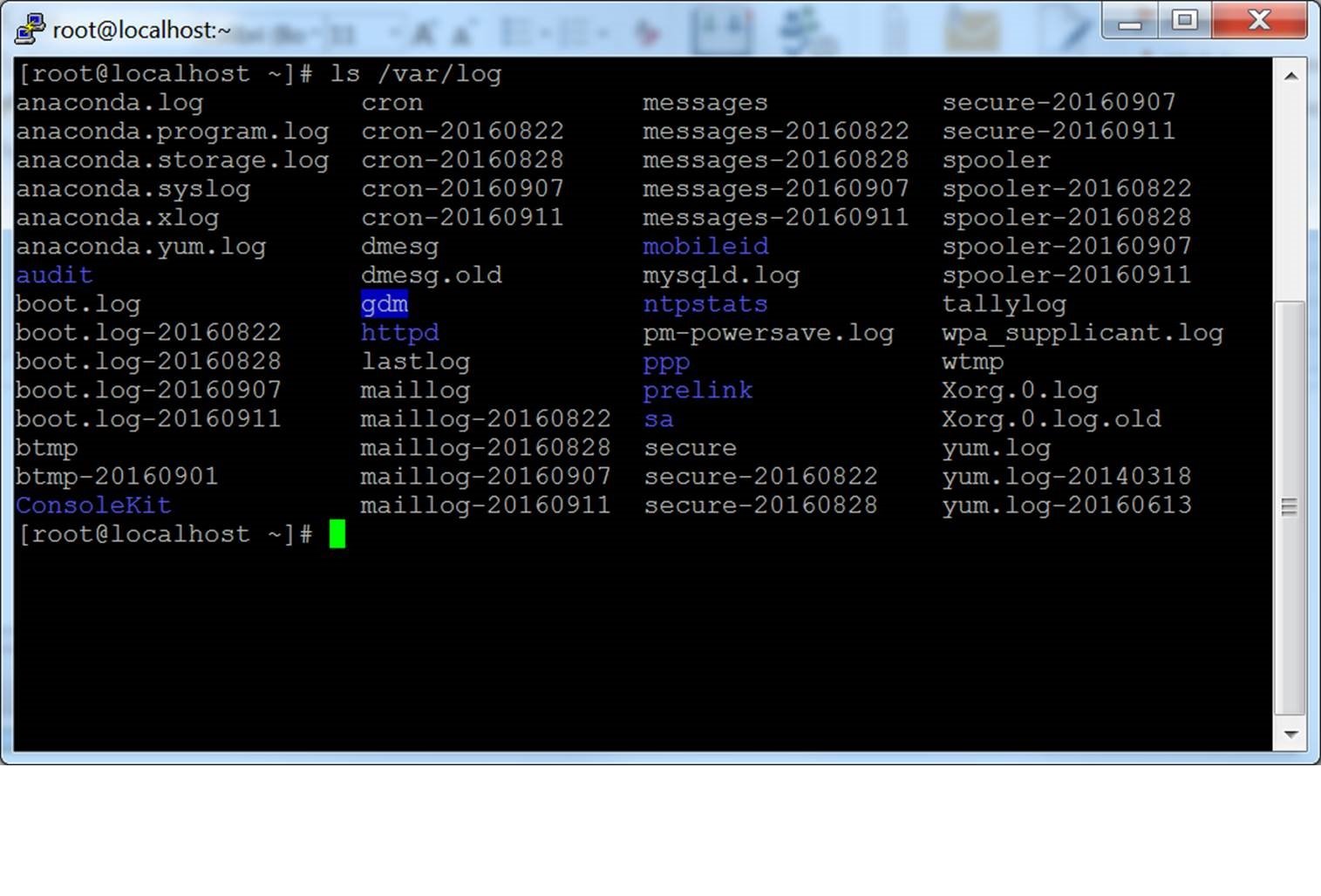
## Terminal window and login credentials

After the security alert, you should get a terminal window. By default, this is a black, very bland window. It should first ask for your user name and then password. After these, you should get a command line on the server.

You can then type into the terminal Window. You are now connected to the server, and anything you type in the Window is sent to the server. Server's responses are displayed in the Window. You can run any text-based applications on the server using the window.

The session terminates:

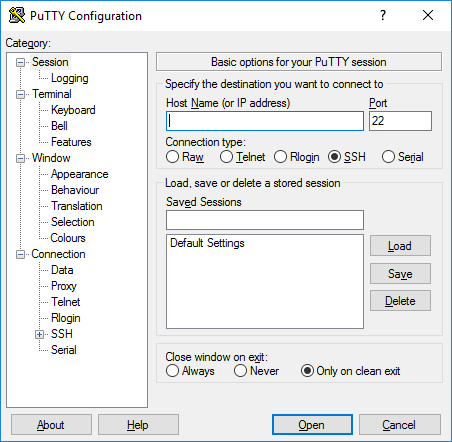
* when you exit the command-line shell on the server (typically by typing exit) to the command line
* or pressing Control-D.
* Alternatively, you can forcibly terminate the session by closing the terminal window.



## 

## Configuration options and saved profiles

The initial configuration window contains a lot of options. Most of them are not needed in normal use.



### **Port**

The **port** field specifies the TCP/IP port to connect. For SSH, this is the port on which the SSH server runs. Normally it can be left to 22. If for some reason you need to connect to a different port number, just change the value. Usually only developers would change this to a different value, but some enterprises are known to run SSH servers in non-standard ports or to run multiple SSH servers on the same server at different ports.

### **Connection type**

The **Connection type** selection almost never needs to be touched. Just leave it as **SSH**. SSH is a secure, encrypted communications protocol designed to ensure your password and data are maximally protected.

**Raw connections** might be used for developers to connect a TCP/IP socket for testing (e.g., when developing a network application that listens on a TCP/IP port).

[**Telnet**](https://www.ssh.com/ssh/telnet) is an old legacy protocol that is almost never used, unless you manage equipment that is more than 10 years old. Telnet is not secure. Passwords are sent in the clear on the network. Attackers can easily eavesdrop on plaintext communications and steal user names and passwords. **Rlogin** is another legacy protocol with similar woes.

**Serial** refers to a serial port, another legacy communications mechanism for connecting computers to peripheral devices. Most PCs these days no longer have serial ports, but they are still sometimes used for controlling physical equipment, instrumentation, machinery, or communications devices. Another use for serial ports is debugging operating systems or embedded software.

### **Load, save, or delete a stored session**

This section allows you to save your settings as named profiles. Just write the name of your new profile in the **Saved Sessions** box and click **Save** to create a new profile. The host name and your other settings are saved in the profile.

Saved profiles appear in the larger box below it. Initially it will contain just **Default Settings**. Profiles you save will be included there. Select a profile and click **Load** to use a previously saved profile. Select a profile and click **Delete** to delete a profile that is no longer needed.

### **Close window on exit**

Finally, the **Close window on exit** setting specifies whether the terminal window should be automatically closed when the connection is terminated. There is rarely any need to change it from the default value of **Only on clean exit**.

## Left pane configuration options

More options can be found in the left pane titled **Category**. Select a category from the tree, and the right pane will change to show configuration options for that category. The initally shown options belong to the **Session** category.

Only the more relevant options are described here. There are lots of options, and most of them would never be used.

### Terminal options

The options in this category influence terminal emulation and keyboard mappings. They are largely self-explanatory, and will not be covered here. Very few people need to touch these. Some people may change how the **bell** character is handled; people using exotic operating systems might change what is sent by the **backspace** or **delete** character.

### Window options

The window options influence the appearance and behavior of the terminal window. It can also specify how characters are translated on output and to select fonts and colors for the window.

## Connection options

Of the connection options, the **Data** options can be useful. The **Auto-login user name** specifies the user to log in as, so that the name will not have to be entered every time.

The **Proxy** options are rarely useful for home users, but may be needed in enterprises that do not allow outgoing Internet connections without using a SOCKS proxy or other similar mechanisms. Don't worry if you don't know what a SOCKS proxy is; just stay out of that section.

The **Telnet**, **Rlogin**, and **Serial** categories only contain options for those protocols, and very few people would ever use them.

The **SSH** options, however, are important and useful for some people. The ordinary user or student need not worry about them. But if you want to use public key authentication, then they are needed. Note that **you need to open the SSH options subtree** by clicking on the small [+] symbol. Otherwise you won't see all the options.

### **Key exchange, host keys, and cipher options**

You almost never want to touch the Kex (key exchange), Host Keys, or Cipher options. They all have reasonable default values, and most people don't know enough about [**cryptography**](https://www.ssh.com/cryptography/) to select any better values. Thus just skip these options, unless you know what you are doing.

### **Authentication options - public key authentication**

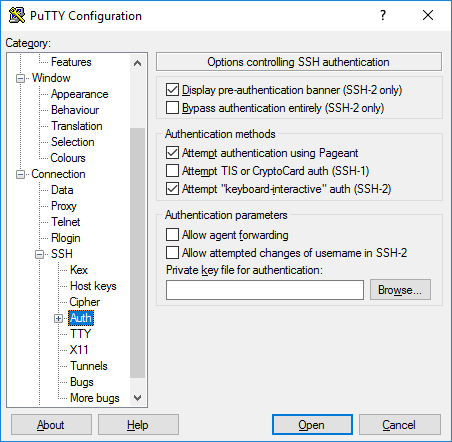
The **Auth** subtree contains some options that may be useful. When **Auth** is clicked, it shows a pane titled **Options controlling SSH authentication**.

To enable public key authentication, you just [**generate an SSH key**](https://www.ssh.com/ssh/putty/windows/puttygen) and then click the **Browse** button in the **Authentication parameters** box in the middle right area of this configuration pane.

For more information, see also [**configuring public key authentication for PuTTY**](https://www.ssh.com/ssh/putty/public-key-authentication). Advanced users may also want to check the **Allow agent forwarding** checkbox to use key-based single sign-on.

Most users have no need to generate SSH keys and need not know what public key authentication is.

System administrators, however, should learn it and should also familiarize themselves with [**SSH key management**](https://www.ssh.com/iam/ssh-key-management) and ensure their organization implements proper provisioning and termination processes and audits for SSH keys.



### **Active Directory authentication (GSSAPI / Kerberos)**

One of the interesting features of PuTTY is support for [**Active Directory**](https://www.ssh.com/iam/active-directory/) single sign-on. Technically it uses the Kerberos protocol via a programming interface called [**GSSAPI**](https://tools.ietf.org/rfc/rfc2743.txt).

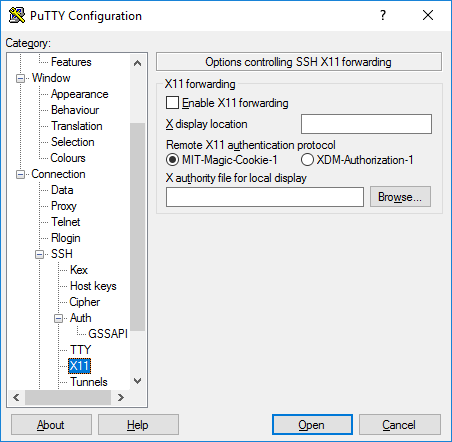
In the SSH protocol, the mechanism is called GSSAPI authentication.

Enterprise users using Kerberos authentication (e.g., via the Centrify or Quest Authentication Services aka Vintela) may want to take advantage of the single-sign-on capability. Other users don't need to care. The settings for GSSAPI authentication can be found under the **SSH** / **Auth** section. Note that you must again expand the **Auth** section by clicking on the [+] symbol to see the GSSAPI options.

### **X11 forwarding options**

[**X11**](https://en.wikipedia.org/wiki/X_Window_System) is a protocol and system for running graphical applications on Unix and Linux. It supports running graphical applications remotely over a network out-of-the-box.

PuTTY does not implement an X11 server (the display side), but it can work with some other product that implements X server functionality on Windows. A popular free alternative is **[XMing](http://www.straightrunning.com/XmingNotes/" \t "_target)**.

To use an X11 server, you need to check the **Enable X11 forwarding** box and enter **localhost:0.0** in the **X display location** box. The other settings need not be touched. Detailed instructions can be found, e.g., [**here**](http://www.geo.mtu.edu/geoschem/docs/putty_install.html).

### **Tunneling options**

The final category of configuration options we'll discuss is **Tunnels**. They are used for configuring SSH tunneling, also called SSH port forwarding. This panel can be used for defining forwardings for the connection. Forwardings are saved in the profile.

To add a **local forwarding** (i.e., TCP/IP port on local machine forwarded to a port on the remote machine or to a machine reachable from the remote machine), write the source port in the **Source port** field, the destination host and port (e.g., www.dest.com:80) in the **Destination** field, and select **Local**. The click **Add**.

To add a **remote forwarding** (i.e., a TCP/IP port on the remote machine forwarded to a port on the local machine or to a machine reachable from the local machine), specify **Source port** on the destination machine and **Destination** that is reachable from the local machine (your desktop).

Normally you need not check **Local ports accept connections from other hosts** or the same for remote ports. However, if the connection to the forwarded port is from a over a network instead of from localhost, then you need to check these. There is a small security risk, but usually it is not a problem in the cases where SSH tunneling is used. However you should understand that anyone who can connect to the respective computer can then connect to the forwarded port. In some cases port forwarding can be used to traverse firewalls. We suggest you read our article on [**the risks of SSH port forwarding**](https://www.ssh.com/ssh/tunneling).

# 

Love is patient, love is kind. It does not envy, it does not boast, it is not proud. It does not dishonor others, it is not self-seeking, it is not easily angered, it keeps no record of wrongs. Love does not delight in evil but rejoices with the truth. It always protects, always trusts, always hopes, always perseveres.

# Chapter 12. Troubleshooting and FAQ Contents:

Debug Messages: Your First Line of Defense Problems and Solutions Other SSH Resources Reporting Bugs

* SSH1, SSH2, and OpenSSH are complex products.
* When a problem occurs, your plan of action should be, in order:
  + Run the client and server in debug mode.
  + Consult archives of questions and answers to see if anyone else has encountered and solved this problem.
  + Seek help.

Many people jump immediately to Step 3, posting questions in public forums and waiting hours or days for a reply, when a simple ssh -v or FAQ can clarify the problem in moments.

Be a smart and efficient technologist, and use your available resources before seeking help from the community. (Although the SSH community is eager to help if you've done your homework.)

12.1. Debug Messages: Your First Line of Defense SSH1/SSH2 clients and servers have debugging built-in.

When invoked with appropriate options, these programs emit messages about their progress and failures.

You can use these messages to isolate problems.

12.1.1. Client Debugging Most clients print debug messages when invoked with the -v (verbose mode) option: [Section 7.4.15, "Logging and Debugging"]

* $ ssh -v server.example.com
* $ scp -v myfile server.example.com:otherfile

So many problems can be identified in verbose mode. This should be your first instinct whenever you encounter a problem.

TIP: Please take a deep breath and repeat after us: "ssh -v is my friend..." "ssh -v is my friend..." "ssh -v is my friend..."

12.1.2. Server Debugging The SSH1, SSH2, and OpenSSH servers also print debug messages when asked: # SSH1, OpenSSH $ sshd -d # SSH2 only $ sshd2 -v In either case, the server enters a special debugging mode.

It accepts a single connection, operates normally until the connection terminates, and then exits.

It doesn't go into the background or create a child process to handle the connection, and it prints information on its progress to the screen (that is, to the standard error stream).

* SSH2 has a more complicated system for debugging: numeric debugging levels, specified with the -d option, where a higher number means more information.
* [Section 5.8.2, "Logging and SSH2"] In fact, -v for verbose mode is actually just a shorthand for -d2.
* At higher debug levels, the output is so huge that only SSH developers will likely find it of use in tracking down obscure problems.
* But you may need to crank up the level beyond 2 to see the information you need.
* For example, to have it report which algorithms are negotiated for a connection, use -d3.
* If you get the error message "TCP/IP Failure", turning up to -d5 shows the more specific OS-level error message returned from the connection attempt. When debugging a server, remember to avoid port conflicts with any other running SSH server.

Either terminate the other server or use an alternative port number for debugging: $ sshd1 -d -p 54321 Use the -p option in the client when testing this debugging instance of the server: $ ssh -p 54321 localhost This way, you don't interrupt or affect another sshd in use.

The Top Ten SSH Questions

* How do I install my public key file on the remote host the first time? Connect by password authentication and use your terminal program's copy and paste feature. [Section 12.2.2.4, "Public-key authentication"]
* I put my SSH public key file mykey.pub into my remote SSH directory, but public-key authentication doesn't work. The public key must be referenced in your remote authorization file. [Section 12.2.2.4, "Public-key authentication"]
* Public-key authentication isn't working. Use ssh -v, and check your keys, files, and permissions. [Section 12.2.2.4, "Public-key authentication"]
* Password authentication isn't working. Use ssh -v. There are a variety of possible causes. [Section 12.2.2.2, "Password authentication"]
* Trusted-host authentication isn't working (SSH1 RhostsRSA, SSH2 hostbased). Use ssh -v. Check your four control files, hostnames, and setuid status of the SSH client program or ssh-signer2. [Section 12.2.2.3, "Trusted-host authentication"]
* How do I authenticate without typing a password or passphrase? ssh-agent, unencrypted keys, trusted-host authentication, or Kerberos. [Section 12.2.2.1, "General authentication problems"]
* How do I secure FTP with port forwarding? Forward a local port to port 21 on the FTP server for the control connection; the data connection is much harder. [Section 12.2.5.6, "Port forwarding"]
* X forwarding isn't working. Don't set your remote DISPLAY variable manually. (And there are other things to check.) [Section 12.2.5.6, "Port forwarding"]
* Why don't wildcards or shell variables work on the scp command line? Your local shell expands them before scp runs. Escape the special characters. [Section 12.2.5.4, "scp"]
* A feature of ssh or scp isn't working, but I'm sure I'm using it correctly. Use ssh -v. Also the system configuration may be overriding your settings. docstore.mik.ua/orelly/networking\_2ndEd/ssh/ch12\_01.htm

Graphical user interface, text, application, Word, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated with medium confidence

More troubleshooting information like you see above: <https://docstore.mik.ua/orelly/networking_2ndEd/ssh/index.htm>

# 

# Secure Shell Wiki

*.*

|  |  |
| --- | --- |
| **Secure Shell** | |
| [Protocol stack](https://en.wikipedia.org/wiki/Protocol_stack) | |
| **Purpose** | secure connection, remote access |
| **Developer(s)** | Tatu Ylönen, [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) |
| **Introduction** | 1995 |
| [**OSI layer**](https://en.wikipedia.org/wiki/OSI_model) | [Transport layer](https://en.wikipedia.org/wiki/Transport_layer) through [application layer](https://en.wikipedia.org/wiki/Application_layer) |
| [**Port(s)**](https://en.wikipedia.org/wiki/Port_(computer_networking)) | 22 |
| [**RFC(s)**](https://en.wikipedia.org/wiki/Request_for_Comments) | RFC 4250, RFC 4251, RFC 4252, RFC 4253, RFC 4254 |

The **Secure Shell Protocol** (**SSH**) is a [cryptographic](https://en.wikipedia.org/wiki/Cryptography) [network protocol](https://en.wikipedia.org/wiki/Network_protocol) for operating [network services](https://en.wikipedia.org/wiki/Network_service) securely over an unsecured network.[[1]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4251-1) Its most notable applications are remote [login](https://en.wikipedia.org/wiki/Login) and [command-line](https://en.wikipedia.org/wiki/Command-line_interface) execution.

SSH applications are based on a [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server_model) architecture, connecting an [SSH client](https://en.wikipedia.org/wiki/SSH_client) instance with an [SSH server](https://en.wikipedia.org/wiki/SSH_server).[[2]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4252-2) SSH operates as a layered protocol suite comprising three principal hierarchical components: the *transport layer* provides server authentication, confidentiality, and integrity; the *user authentication protocol* validates the user to the server; and the *connection protocol* multiplexes the encrypted tunnel into multiple logical communication channels.[[1]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4251-1)

SSH was designed on [Unix-like](https://en.wikipedia.org/wiki/Unix-like) operating systems, as a replacement for [Telnet](https://en.wikipedia.org/wiki/Telnet) and for [unsecured](https://en.wikipedia.org/wiki/Computer_security) remote [Unix shell](https://en.wikipedia.org/wiki/Unix_shell) protocols, such as the Berkeley [Remote Shell](https://en.wikipedia.org/wiki/Remote_Shell) (rsh) and the related [rlogin](https://en.wikipedia.org/wiki/Rlogin) and [rexec](https://en.wikipedia.org/wiki/Remote_Process_Execution" \o "Remote Process Execution) protocols, which all use insecure, [plaintext](https://en.wikipedia.org/wiki/Plaintext) transmission of authentication tokens.

SSH was first designed in 1995 by Finnish computer scientist Tatu Ylönen. Subsequent development of the protocol suite proceeded in several developer groups, producing several variants of implementation. The protocol specification distinguishes two major versions, referred to as SSH-1 and SSH-2. The most commonly implemented software stack is [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH), released in 1999 as open-source software by the [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD) developers. Implementations are distributed for all types of operating systems in common use, including embedded systems.



## Contents

* [1Definition](https://en.wikipedia.org/wiki/Secure_Shell#Definition)
* [2Authentication: OpenSSH key management](https://en.wikipedia.org/wiki/Secure_Shell#Authentication:_OpenSSH_key_management)
* [3Use](https://en.wikipedia.org/wiki/Secure_Shell#Use)
* [4Historical development](https://en.wikipedia.org/wiki/Secure_Shell#Historical_development)
  + [4.1Version 1](https://en.wikipedia.org/wiki/Secure_Shell#Version_1)
  + [4.2Version 2](https://en.wikipedia.org/wiki/Secure_Shell#Version_2)
  + [4.3Version 1.99](https://en.wikipedia.org/wiki/Secure_Shell#Version_1.99)
  + [4.4OpenSSH and OSSH](https://en.wikipedia.org/wiki/Secure_Shell#OpenSSH_and_OSSH)
* [5Uses](https://en.wikipedia.org/wiki/Secure_Shell#Uses)
  + [5.1File transfer protocols](https://en.wikipedia.org/wiki/Secure_Shell#File_transfer_protocols)
* [6Architecture](https://en.wikipedia.org/wiki/Secure_Shell#Architecture)
* [7Algorithms](https://en.wikipedia.org/wiki/Secure_Shell#Algorithms)
* [8Vulnerabilities](https://en.wikipedia.org/wiki/Secure_Shell#Vulnerabilities)
  + [8.1SSH-1](https://en.wikipedia.org/wiki/Secure_Shell#SSH-1)
  + [8.2CBC plaintext recovery](https://en.wikipedia.org/wiki/Secure_Shell#CBC_plaintext_recovery)
  + [8.3Suspected decryption by NSA](https://en.wikipedia.org/wiki/Secure_Shell#Suspected_decryption_by_NSA)
* [9Standards documentation](https://en.wikipedia.org/wiki/Secure_Shell#Standards_documentation)
* [10See also](https://en.wikipedia.org/wiki/Secure_Shell#See_also)
* [11References](https://en.wikipedia.org/wiki/Secure_Shell#References)
* [12Further reading](https://en.wikipedia.org/wiki/Secure_Shell#Further_reading)
* [13External links](https://en.wikipedia.org/wiki/Secure_Shell#External_links)

## Definition

SSH uses [public-key cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography) to [authenticate](https://en.wikipedia.org/wiki/Authentication) the remote computer and allow it to authenticate the user, if necessary.[[2]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4252-2)

SSH may be used in several methodologies. In the simplest manner, both ends of a communication channel use automatically generated public-private key pairs to encrypt a network connection, and then use a [password](https://en.wikipedia.org/wiki/Password) to authenticate the user.

When the public-private key pair is generated by the user manually, the authentication is essentially performed when the key pair is created, and a session may then be opened automatically without a password prompt. In this scenario, the public key is placed on all computers that must allow access to the owner of the matching private key, which the owner keeps private. While authentication is based on the private key, the key is never transferred through the network during authentication. SSH only verifies that the same person offering the public key also owns the matching private key.

In all versions of SSH it is important to verify unknown [public keys](https://en.wikipedia.org/wiki/Public_key), i.e. [associate the public keys with identities](https://en.wikipedia.org/wiki/Public-key_cryptography#Associating_public_keys_with_identities), before accepting them as valid. Accepting an attacker's public key without validation will authorize an unauthorized attacker as a valid user.

## Authentication: OpenSSH key management

On [Unix-like](https://en.wikipedia.org/wiki/Unix-like) systems, the list of authorized public keys is typically stored in the home directory of the user that is allowed to log in remotely, in the file ~/.ssh/authorized\_keys.[[3]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-3) This file is respected by SSH only if it is not writable by anything apart from the owner and root. When the public key is present on the remote end and the matching private key is present on the local end, typing in the password is no longer required. However, for additional security the private key itself can be locked with a passphrase.

The private key can also be looked for in standard places, and its full path can be specified as a command line setting (the option -i for ssh). The [ssh-keygen](https://en.wikipedia.org/wiki/Ssh-keygen) utility produces the public and private keys, always in pairs.

SSH also supports password-based authentication that is encrypted by automatically generated keys. In this case, the attacker could imitate the legitimate server side, ask for the password, and obtain it ([man-in-the-middle attack](https://en.wikipedia.org/wiki/Man-in-the-middle_attack)). However, this is possible only if the two sides have never authenticated before, as SSH remembers the key that the server side previously used. The SSH client raises a warning before accepting the key of a new, previously unknown server. Password authentication can be disabled from the server side.

## Use

SSH is typically used to log into a remote machine and execute commands, but it also supports [tunneling](https://en.wikipedia.org/wiki/Tunneling_protocol), [forwarding](https://en.wikipedia.org/wiki/Port_forwarding) [TCP ports](https://en.wikipedia.org/wiki/TCP_and_UDP_port) and [X11](https://en.wikipedia.org/wiki/X11) connections; it can transfer files using the associated [SSH file transfer](https://en.wikipedia.org/wiki/SSH_file_transfer_protocol) (SFTP) or [secure copy](https://en.wikipedia.org/wiki/Secure_copy) (SCP) protocols.[[2]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-rfc4252-2) SSH uses the [client–server model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model).

An SSH [client](https://en.wikipedia.org/wiki/Client_(computing)) program is typically used for establishing connections to an SSH [daemon](https://en.wikipedia.org/wiki/Daemon_(computer_software)), such as sshd, accepting remote connections. Both are commonly present on most modern [operating systems](https://en.wikipedia.org/wiki/Operating_systems), including [macOS](https://en.wikipedia.org/wiki/MacOS), most distributions of [Linux](https://en.wikipedia.org/wiki/Linux), [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD), [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD), [NetBSD](https://en.wikipedia.org/wiki/NetBSD), [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)) and [OpenVMS](https://en.wikipedia.org/wiki/OpenVMS).

Notably, versions of [Windows](https://en.wikipedia.org/wiki/Windows) prior to Windows 10 version 1709 do not include SSH by default. [Proprietary](https://en.wikipedia.org/wiki/Proprietary_software), [freeware](https://en.wikipedia.org/wiki/Freeware) and [open source](https://en.wikipedia.org/wiki/Open-source_software) (e.g. [PuTTY](https://en.wikipedia.org/wiki/PuTTY),[[4]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-4) and the version of [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) which is part of [Cygwin](https://en.wikipedia.org/wiki/Cygwin)[[5]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-5)) versions of various levels of complexity and completeness exist.

File managers for UNIX-like systems (e.g. [Konqueror](https://en.wikipedia.org/wiki/Konqueror" \o "Konqueror)) can use the [FISH](https://en.wikipedia.org/wiki/Files_transferred_over_shell_protocol) protocol to provide a split-pane GUI with drag-and-drop. The open source Windows program [WinSCP](https://en.wikipedia.org/wiki/WinSCP)[[6]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-6) provides similar file management (synchronization, copy, remote delete) capability using PuTTY as a back-end.

Both WinSCP[[7]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-7) and PuTTY[[8]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-8) are available packaged to run directly off a USB drive, without requiring installation on the client machine. Setting up an SSH server in Windows typically involves enabling a feature in Settings app. In [Windows 10 version 1709](https://en.wikipedia.org/wiki/Windows_10_version_history#Version_1709_(Fall_Creators_Update)), an official Win32 port of OpenSSH is available.

SSH is important in [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) to solve connectivity problems, avoiding the security issues of exposing a cloud-based virtual machine directly on the Internet. An SSH tunnel can provide a secure path over the Internet, through a firewall to a virtual machine.[[9]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-9)

The [IANA](https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority) has assigned [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) [port](https://en.wikipedia.org/wiki/Port_(computer_networking)) 22, [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) port 22 and [SCTP](https://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol) port 22 for this protocol.[[10]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-10) IANA had listed the standard TCP port 22 for SSH servers as one of the [well-known ports](https://en.wikipedia.org/wiki/List_of_well-known_ports_(computing)) as early as 2001.[[11]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-11) SSH can also be run using [SCTP](https://en.wikipedia.org/wiki/SCTP) rather than TCP as the connection oriented transport layer protocol.[[12]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-12)

## Historical development

### Version 1

In 1995, **Tatu Ylönen**, a researcher at [Helsinki University of Technology](https://en.wikipedia.org/wiki/Helsinki_University_of_Technology), Finland, designed the first version of the protocol (now called **SSH-1**) prompted by a password-[sniffing](https://en.wikipedia.org/wiki/Packet_analyzer) attack at his [university network](https://en.wikipedia.org/wiki/University_network).[[13]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-13) The goal of SSH was to replace the earlier [rlogin](https://en.wikipedia.org/wiki/Rlogin), [TELNET](https://en.wikipedia.org/wiki/TELNET), [FTP](https://en.wikipedia.org/wiki/FTP)[[14]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-14) and [rsh](https://en.wikipedia.org/wiki/Remote_Shell" \o "Remote Shell) protocols, which did not provide strong authentication nor guarantee confidentiality. Ylönen released his implementation as [freeware](https://en.wikipedia.org/wiki/Freeware) in July 1995, and the tool quickly gained in popularity. Towards the end of 1995, the SSH user base had grown to 20,000 users in fifty countries.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

In December 1995, Ylönen founded SSH Communications Security to market and develop SSH. The original version of the SSH software used various pieces of [free software](https://en.wikipedia.org/wiki/Free_software), such as [GNU libgmp](https://en.wikipedia.org/wiki/GNU_Multi-Precision_Library), but later versions released by SSH Communications Security evolved into increasingly [proprietary software](https://en.wikipedia.org/wiki/Proprietary_software).

It was estimated that by 2000 the number of users had grown to 2 million.[[15]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-Nicholas_Rosasco_and_David_Larochelle-15)

### Version 2

"Secsh" was the official [Internet Engineering Task Force's](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) name for the IETF working group responsible for version 2 of the SSH protocol.[[16]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-16) In 2006, a revised version of the protocol, **SSH-2**, was adopted as a standard. This version is incompatible with SSH-1. SSH-2 features both security and feature improvements over SSH-1. Better security, for example, comes through [Diffie–Hellman key exchange](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange) and strong [integrity](https://en.wikipedia.org/wiki/Integrity) checking via [message authentication codes](https://en.wikipedia.org/wiki/Message_authentication_code). New features of SSH-2 include the ability to run any number of [shell](https://en.wikipedia.org/wiki/Unix_shell) sessions over a single SSH connection.[[17]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-17) Due to SSH-2's superiority and popularity over SSH-1, some implementations such as libssh (v0.8.0+),[[18]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-18) [Lsh](https://en.wikipedia.org/wiki/Lsh" \o "Lsh)[[19]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-19) and [Dropbear](https://en.wikipedia.org/wiki/Dropbear_(software)" \o "Dropbear (software))[[20]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-20) support only the SSH-2 protocol.

### Version 1.99

In January 2006, well after version 2.1 was established, [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4253](https://datatracker.ietf.org/doc/html/rfc4253) specified that an SSH server supporting 2.0 as well as prior versions should identify its protocol version as 1.99.[[21]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-21) This version number does not reflect a historical software revision, but a method to identify [backward compatibility](https://en.wikipedia.org/wiki/Backward_compatibility).

### OpenSSH and OSSH

In 1999, developers, desiring availability of a free software version, restarted software development from the 1.2.12 release of the original SSH program, which was the last released under an [open source license](https://en.wikipedia.org/wiki/Open_source_license).[[22]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-22) This served as a code base for Björn Grönvall's OSSH software.[[23]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-23) Shortly thereafter, [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD) developers [forked](https://en.wikipedia.org/wiki/Fork_(software_development)) Grönvall's code and created [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH), which shipped with Release 2.6 of OpenBSD. From this version, a "portability" branch was formed to port OpenSSH to other operating systems.[[24]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-24)

As of 2005, [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) was the single most popular SSH implementation, being the default version in a large number of operating system distributions. OSSH meanwhile has become obsolete.[[25]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-25) OpenSSH continues to be maintained and supports the SSH-2 protocol, having expunged SSH-1 support from the codebase in the OpenSSH 7.6 release.

## Uses

Graphical user interface

Description automatically generated

Example of tunneling an [X11](https://en.wikipedia.org/wiki/X11) application over SSH: the user 'josh' has "SSHed" from the local machine 'foofighter' to the remote machine 'tengwar' to run [xeyes](https://en.wikipedia.org/wiki/Xeyes" \o "Xeyes).

Text

Description automatically generated

Logging into [OpenWrt](https://en.wikipedia.org/wiki/OpenWrt" \o "OpenWrt) via SSH using [PuTTY](https://en.wikipedia.org/wiki/PuTTY) running on [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows).

SSH is a protocol that can be used for many applications across many platforms including most [Unix](https://en.wikipedia.org/wiki/Unix) variants ([Linux](https://en.wikipedia.org/wiki/Linux), the [BSDs](https://en.wikipedia.org/wiki/BSD) including [Apple's](https://en.wikipedia.org/wiki/Apple_Inc) [macOS](https://en.wikipedia.org/wiki/MacOS), and [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system))), as well as [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows).

Some of the applications below may require features that are only available or compatible with specific SSH clients or servers. For example, using the SSH protocol to implement a [VPN](https://en.wikipedia.org/wiki/VPN) is possible, but presently only with the [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) server and client implementation.

* For login to a shell on a remote host (replacing [Telnet](https://en.wikipedia.org/wiki/Telnet) and [rlogin](https://en.wikipedia.org/wiki/Rlogin))
* For executing a single command on a remote host (replacing [rsh](https://en.wikipedia.org/wiki/Remote_shell" \o "Remote shell))
* For setting up automatic (passwordless) login to a remote server (for example, using [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH)[[26]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-26))
* In combination with [rsync](https://en.wikipedia.org/wiki/Rsync" \o "Rsync) to back up, copy and mirror files efficiently and securely
* For [forwarding](https://en.wikipedia.org/wiki/Port_forwarding) a port
* For [tunneling](https://en.wikipedia.org/wiki/Tunneling_protocol) (not to be confused with a [VPN](https://en.wikipedia.org/wiki/VPN), which [routes](https://en.wikipedia.org/wiki/VPN#Routing) packets between different networks, or [bridges](https://en.wikipedia.org/wiki/VPN#OSI_Layer_1_services) two [broadcast domains](https://en.wikipedia.org/wiki/Broadcast_domain) into one).
* For using as a full-fledged encrypted VPN. Note that only [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) server and client supports this feature.
* For forwarding [X](https://en.wikipedia.org/wiki/X_Window_System) from a remote [host](https://en.wikipedia.org/wiki/Host_(network)) (possible through multiple intermediate hosts)
* For browsing the web through an encrypted proxy connection with SSH clients that support the [SOCKS protocol](https://en.wikipedia.org/wiki/SOCKS).
* For securely mounting a directory on a remote server as a [filesystem](https://en.wikipedia.org/wiki/File_system) on a local computer using [SSHFS](https://en.wikipedia.org/wiki/SSHFS).
* For automated remote monitoring and management of servers through one or more of the mechanisms discussed above.
* For development on a mobile or embedded device that supports SSH.
* For securing file transfer protocols.

### File transfer protocols

The Secure Shell protocols are used in several file transfer mechanisms.

* [Secure copy](https://en.wikipedia.org/wiki/Secure_copy) (SCP), which evolved from [RCP](https://en.wikipedia.org/wiki/Rcp_(Unix)) protocol over SSH
* [rsync](https://en.wikipedia.org/wiki/Rsync), intended to be more efficient than SCP. Generally runs over an SSH connection.
* [SSH File Transfer Protocol](https://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol) (SFTP), a secure alternative to [FTP](https://en.wikipedia.org/wiki/File_Transfer_Protocol) (not to be confused with [FTP over SSH](https://en.wikipedia.org/wiki/FTP_over_SSH) or [FTPS](https://en.wikipedia.org/wiki/FTPS))
* [Files transferred over shell protocol](https://en.wikipedia.org/wiki/Files_transferred_over_shell_protocol) (FISH), released in 1998, which evolved from [Unix shell](https://en.wikipedia.org/wiki/Unix_shell) commands over SSH
* [Fast and Secure Protocol](https://en.wikipedia.org/wiki/Fast_and_Secure_Protocol) (FASP), aka *Aspera*, uses SSH for control and UDP ports for data transfer.

## Architecture

Diagram

Description automatically generated

Diagram of the SSH-2 binary packet.

The SSH protocol has a layered architecture with three separate components:

* The *transport layer* ([RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4253](https://datatracker.ietf.org/doc/html/rfc4253)) typically uses the [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) of [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP), reserving [port number](https://en.wikipedia.org/wiki/Port_number) 22 as a server listening port. This layer handles initial key exchange as well as server authentication, and sets up encryption, compression, and integrity verification. It exposes to the upper layer an interface for sending and receiving plaintext packets with a size of up to 32,768 bytes each, but more can be allowed by each implementation. The transport layer also arranges for key re-exchange, usually after 1 GB of data has been transferred or after one hour has passed, whichever occurs first.
* The *user authentication layer* ([RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4252](https://datatracker.ietf.org/doc/html/rfc4252)) handles client authentication, and provides a suite of authentication algorithms. Authentication is *client-driven*: when one is prompted for a password, it may be the SSH client prompting, not the server. The server merely responds to the client's authentication requests. Widely used user-authentication methods include the following:
  + *password*: a method for straightforward password authentication, including a facility allowing a password to be changed. Not all programs implement this method.
  + *publickey*: a method for [public-key-based authentication](https://en.wikipedia.org/wiki/Public-key_cryptography), usually supporting at least [DSA](https://en.wikipedia.org/wiki/Digital_Signature_Algorithm), [ECDSA](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm) or [RSA](https://en.wikipedia.org/wiki/RSA_(algorithm)) keypairs, with other implementations also supporting [X.509](https://en.wikipedia.org/wiki/X.509) certificates.
  + *keyboard-interactive* ([RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4256](https://datatracker.ietf.org/doc/html/rfc4256)): a versatile method where the server sends one or more prompts to enter information and the client displays them and sends back responses keyed-in by the user. Used to provide [one-time password](https://en.wikipedia.org/wiki/One-time_password) authentication such as [S/Key](https://en.wikipedia.org/wiki/S/Key) or [SecurID](https://en.wikipedia.org/wiki/SecurID). Used by some OpenSSH configurations when [PAM](https://en.wikipedia.org/wiki/Pluggable_authentication_modules) is the underlying host-authentication provider to effectively provide password authentication, sometimes leading to inability to log in with a client that supports just the plain *password* authentication method.
  + [GSSAPI](https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface) authentication methods which provide an extensible scheme to perform SSH authentication using external mechanisms such as [Kerberos 5](https://en.wikipedia.org/wiki/Kerberos_(protocol)) or [NTLM](https://en.wikipedia.org/wiki/NTLM), providing [single sign-on](https://en.wikipedia.org/wiki/Single_sign-on) capability to SSH sessions. These methods are usually implemented by commercial SSH implementations for use in organizations, though OpenSSH does have a working GSSAPI implementation.
* The *connection layer* ([RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4254](https://datatracker.ietf.org/doc/html/rfc4254)) defines the concept of channels, channel requests, and global requests, which define the SSH services provided. A single SSH connection can be multiplexed into multiple logical channels simultaneously, each transferring data bidirectionally. Channel requests are used to relay out-of-band channel-specific data, such as the changed size of a terminal window, or the exit code of a server-side process. Additionally, each channel performs its own flow control using the receive window size. The SSH client requests a server-side port to be forwarded using a global request. Standard channel types include:
  + *shell* for terminal shells, SFTP and exec requests (including SCP transfers)
  + *direct-tcpip* for client-to-server forwarded connections
  + *forwarded-tcpip* for server-to-client forwarded connections
* The [SSHFP](https://en.wikipedia.org/wiki/SSHFP) DNS record (RFC 4255) provides the public host key fingerprints in order to aid in verifying the authenticity of the host.

This open architecture provides considerable flexibility, allowing the use of SSH for a variety of purposes beyond a secure shell. The functionality of the transport layer alone is comparable to [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) (TLS); the user-authentication layer is highly extensible with custom authentication methods; and the connection layer provides the ability to multiplex many secondary sessions into a single SSH connection, a feature comparable to [BEEP](https://en.wikipedia.org/wiki/BEEP) and not available in TLS.

## Algorithms

* [EdDSA](https://en.wikipedia.org/wiki/EdDSA),[[27]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-27) [ECDSA](https://en.wikipedia.org/wiki/ECDSA), [RSA](https://en.wikipedia.org/wiki/RSA_(cryptosystem)) and [DSA](https://en.wikipedia.org/wiki/Digital_Signature_Algorithm) for [public-key cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography).[[28]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-RFC5656-28)
* [ECDH](https://en.wikipedia.org/wiki/ECDH) and [Diffie–Hellman](https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman) for [key exchange](https://en.wikipedia.org/wiki/Key_exchange).[[28]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-RFC5656-28)
* [HMAC](https://en.wikipedia.org/wiki/HMAC), [AEAD](https://en.wikipedia.org/wiki/AEAD) and [UMAC](https://en.wikipedia.org/wiki/UMAC) for [MAC](https://en.wikipedia.org/wiki/Message_authentication_code).[[29]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-29)
* [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) (and deprecated [RC4](https://en.wikipedia.org/wiki/RC4), [3DES](https://en.wikipedia.org/wiki/3DES), [DES](https://en.wikipedia.org/wiki/Data_Encryption_Standard)[[30]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-30)) for [symmetric encryption](https://en.wikipedia.org/wiki/Symmetric_encryption).
* [AES-GCM](https://en.wikipedia.org/wiki/AES-GCM)[[31]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-31) and [ChaCha20-Poly1305](https://en.wikipedia.org/wiki/ChaCha20-Poly1305) for [AEAD](https://en.wikipedia.org/wiki/AEAD) encryption.
* [SHA](https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) (and deprecated [MD5](https://en.wikipedia.org/wiki/MD5)) for [key fingerprint](https://en.wikipedia.org/wiki/Key_fingerprint).

## Vulnerabilities

### SSH-1

In 1998, a vulnerability was described in SSH 1.5 which allowed the unauthorized insertion of content into an encrypted SSH stream due to insufficient data integrity protection from [CRC-32](https://en.wikipedia.org/wiki/CRC-32) used in this version of the protocol.[[32]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-32)[[33]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-33) A fix known as SSH Compensation Attack Detector[[34]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-34) was introduced into most implementations. Many of these updated implementations contained a new [integer overflow](https://en.wikipedia.org/wiki/Integer_overflow) vulnerability[[35]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-35) that allowed attackers to execute arbitrary code with the privileges of the SSH daemon, typically root.

In January 2001 a vulnerability was discovered that allows attackers to modify the last block of an [IDEA](https://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm)-encrypted session.[[36]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-36) The same month, another vulnerability was discovered that allowed a malicious server to forward a client authentication to another server.[[37]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-cert1-37)

Since SSH-1 has inherent design flaws which make it vulnerable, it is now generally considered obsolete and should be avoided by explicitly disabling fallback to SSH-1.[[37]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-cert1-37) Most modern servers and clients support SSH-2.[[38]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-38)

### CBC plaintext recovery

In November 2008, a theoretical vulnerability was discovered for all versions of SSH which allowed recovery of up to 32 bits of plaintext from a block of ciphertext that was encrypted using what was then the standard default encryption mode, [CBC](https://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_(CBC)).[[39]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-SSH_CBC_vulnerability-39) The most straightforward solution is to use [CTR](https://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Counter_(CTR)), counter mode, instead of CBC mode, since this renders SSH resistant to the attack.[[39]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-SSH_CBC_vulnerability-39)

### Suspected decryption by NSA

On December 28, 2014 [*Der Spiegel*](https://en.wikipedia.org/wiki/Der_Spiegel) published classified information[[40]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-Spiegel2014-40) leaked by whistleblower [Edward Snowden](https://en.wikipedia.org/wiki/Edward_Snowden) which suggests that the [National Security Agency](https://en.wikipedia.org/wiki/National_Security_Agency) may be able to decrypt some SSH traffic. The technical details associated with such a process were not disclosed. A 2017 analysis of the [CIA](https://en.wikipedia.org/wiki/CIA) hacking tools *BothanSpy* and *Gyrfalcon* suggested that the SSH protocol was not compromised.[[41]](https://en.wikipedia.org/wiki/Secure_Shell#cite_note-41)

In addition, the [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) project includes several vendor protocol specifications/extensions:

* [OpenSSH PROTOCOL overview](http://cvsweb.openbsd.org/cgi-bin/cvsweb/~checkout~/src/usr.bin/ssh/PROTOCOL?content-type=text/plain)
* [OpenSSH certificate/key overview](http://cvsweb.openbsd.org/cgi-bin/cvsweb/~checkout~/src/usr.bin/ssh/PROTOCOL.certkeys?content-type=text/plain)
* [draft-miller-ssh-agent-04](https://tools.ietf.org/html/draft-miller-ssh-agent-04) - SSH Agent Protocol (December 2019)

## See also

* [Brute-force attack](https://en.wikipedia.org/wiki/Brute-force_attack)
* [Comparison of SSH clients](https://en.wikipedia.org/wiki/Comparison_of_SSH_clients)
* [Comparison of SSH servers](https://en.wikipedia.org/wiki/Comparison_of_SSH_servers)
* [Corkscrew](https://en.wikipedia.org/wiki/Corkscrew_(program))
* [Ident](https://en.wikipedia.org/wiki/Ident_protocol)
* [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH)
* [Secure Shell tunneling](https://en.wikipedia.org/wiki/Secure_Shell_tunneling)
* [Web-based SSH](https://en.wikipedia.org/wiki/Web-based_SSH)