**SSH Port Forwarding Example**

**Contents**

[**What Is SSH Port Forwarding, aka SSH Tunneling?**](https://www.ssh.com/ssh/tunneling/example#what-is-ssh-port-forwarding,-aka-ssh-tunneling?)[**Local Forwarding**](https://www.ssh.com/ssh/tunneling/example#local-forwarding)[**Remote Forwarding**](https://www.ssh.com/ssh/tunneling/example#remote-forwarding)[**Opening Backdoors into the Enterprise**](https://www.ssh.com/ssh/tunneling/example#opening-backdoors-into-the-enterprise)[**Server-Side Configuration**](https://www.ssh.com/ssh/tunneling/example#server-side-configuration)[**How to Prevent SSH Port Forwarding from Circumventing Firewalls**](https://www.ssh.com/ssh/tunneling/example#how-to-prevent-ssh-port-forwarding-from-circumventing-firewalls)[**Further Information**](https://www.ssh.com/ssh/tunneling/example#further-information)

**What Is SSH Port Forwarding, aka SSH Tunneling?**

SSH port forwarding is a mechanism in [**SSH**](https://www.ssh.com/ssh/) for tunneling application ports from the client machine to the server machine, or vice versa. It can be used for *adding encryption to legacy applications*, *going through firewalls*, and some system administrators and IT professionals use it for *opening backdoors* into the internal network from their home machines. It can also be abused by hackers and malware to open access from the Internet to the internal network. See the [**SSH tunneling**](https://www.ssh.com/ssh/tunneling/) page for a broader overview.

**Local Forwarding**

Local forwarding is used to forward a port from the client machine to the server machine. Basically, the [**SSH client**](https://www.ssh.com/ssh/client) listens for connections on a configured port, and when it receives a connection, it tunnels the connection to an [**SSH server**](https://www.ssh.com/ssh/server). The server connects to a configurated destination port, possibly on a different machine than the SSH server.

Typical uses for local port forwarding include:

* Tunneling sessions and file transfers through [**jump servers**](https://www.ssh.com/iam/jump-server)
* Connecting to a service on an internal network from the outside
* Connecting to a remote file share over the Internet

Quite a few organizations for all incoming SSH access through a single [**jump server**](https://www.ssh.com/iam/jump-server). The server may be a standard Linux/Unix box, usually with some extra hardening, intrusion detection, and/or logging, or it may be a commercial jump server solution.

Many jump servers allow incoming port forwarding, once the connection has been authenticated. Such port forwarding is convenient, because it allows tech-savvy users to use internal resources quite transparently. For example, they may forward a port on their local machine to the corporate intranet web server, to an internal mail server's [**IMAP**](https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) port, to a local file server's 445 and 139 ports, to a printer, to a version control repository, or to almost any other system on the internal network. Frequently, the port is tunneled to an SSH port on an internal machine.

In [**OpenSSH**](https://www.ssh.com/ssh/openssh/), local port forwarding is configured using the -L option:

ssh -L 80:intra.example.com:80 gw.example.com

This example opens a connection to the gw.example.com jump server, and forwards any connection to port 80 on the local machine to port 80 on intra.example.com.

By default, anyone (even on different machines) can connect to the specified port on the SSH client machine. However, this can be restricted to programs on the same host by supplying a *bind address*:

ssh -L 127.0.0.1:80:intra.example.com:80 gw.example.com

The LocalForward option in the [**OpenSSH client configuration file**](https://www.ssh.com/ssh/config/) can be used to configure forwarding without having to specify it on command line.

**Remote Forwarding**

In OpenSSH, remote SSH port forwardings are specified using the -R option. For example:

ssh -R 8080:localhost:80 public.example.com

This allows anyone on the remote server to connect to TCP port 8080 on the remote server. The connection will then be tunneled back to the client host, and the client then makes a TCP connection to port 80 on localhost. Any other host name or IP address could be used instead of localhost to specify the host to connect to.

This particular example would be useful for giving someone on the outside access to an internal web server. Or exposing an internal web application to the public Internet. This could be done by an employee working from home, or by an attacker.

By default, OpenSSH only allows connecting to remote forwarded ports from the server host. However, the GatewayPorts option in the server configuration file [**sshd\_config**](https://www.ssh.com/ssh/sshd_config/) can be used to control this. The following alternatives are possible:

GatewayPorts no

This prevents connecting to forwarded ports from outside the server computer.

GatewayPorts yes

This allows anyone to connect to the forwarded ports. If the server is on the public Internet, anyone on the Internet can connect to the port.

GatewayPorts clientspecified

This means that the client can specify an IP address from which connections to the port are allowed. The syntax for this is:

ssh -R 52.194.1.73:8080:localhost:80 host147.aws.example.com

In this example, only connections from the IP address 52.194.1.73 to port 8080 are allowed.

OpenSSH also allows the forwarded remote port to specified as 0. In this case, the server will dynamically allocate a port and report it to the client. When used with the -O forward option, the client will print the allocated port number to standard output.

**Opening Backdoors into the Enterprise**

Remote SSH port forwarding is commonly used by employees to open backdoors into the enterprise. For example, the employee may set get a [**free-tier server from Amazon AWS**](https://aws.amazon.com/free/), and log in from the office to that server, specifying remote forwarding from a port on the server to some server or application on the internal enterprise network. Multiple remote forwards may be specified to open access to more than one application.

The employee would also set GatewayPorts yes on the server (most employees do not have fixed IP addresses at home, so they cannot restrict the IP address).

For example, the following command opens access to an internal Postgres database at port 5432 and an internal SSH port at port 2222.

ssh -R 2222:d76767.nyc.example.com:22 -R 5432:postgres3.nyc.example.com:5432 aws4.mydomain.net

**Server-Side Configuration**

The AllowTcpForwarding option in the [**OpenSSH server configuration file**](https://www.ssh.com/ssh/sshd_config/) must be enabled on the server to allow port forwarding. By default, forwarding is allowed. Possible values for this option are yes or all to allow all TCP forwarding, no to prevent all TCP forwarding, local to allow local forwardings, and remote to allow remote forwardings.

Another option of interest is AllowStreamLocalForwarding, which can be used to forward Unix domain sockets. It allows the same values as AllowTcpForwarding. The default is yes.

For example:

AllowTcpForwarding remote

AllowStreamLocalForwarding no

The GatewayPorts configuration option as described above also affects remote port forwardings. Possible values were no (only local connections from server host allowed; default), yes (anyone on the Internet can connect to remote forwarded ports), and clientspecified (client can specify an IP address that can connect, anyone can if not specified).

**How to Prevent SSH Port Forwarding from Circumventing Firewalls**

We recommend that port forwarding be expressly disabled when not needed. Leaving port forwarding enabled can expose the organization to security risks and backdoors. For example, if a server intended to only provide [**SFTP**](https://www.ssh.com/ssh/sftp/) file transfers allows port forwardings, those forwardings might be used to gain unintended access into the internal network from the Intranet.

The problem is that port forwarding can in practice only be prevented by a server or firewall. An enterprise cannot control all servers on the Internet. Firewall-based control can also be tricky, as most organizations have servers in Amazon AWS and other cloud services, and those servers are usually accessed using SSH.

**Further Information**

[SSH](https://help.ubuntu.com/community/SSH)/[OpenSSH](https://help.ubuntu.com/community/SSH/OpenSSH)/[PortForwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding)

Parent page: [Internet and Networking](https://help.ubuntu.com/community/InternetAndNetworking) >> [SSH](https://help.ubuntu.com/community/SSH)

|  |
| --- |
| Contents   1. [Introduction](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Introduction) 2. [Types of Port Forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Types_of_Port_Forwarding)    1. [Local Port Forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Local_Port_Forwarding)    2. [Remote Port Forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Remote_Port_Forwarding)    3. [Dynamic Port Forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Dynamic_Port_Forwarding) 3. [Forwarding GUI Programs](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Forwarding_GUI_Programs)    1. [Single Applications](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Single_Applications) 4. [Port Forwarding Explained](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Port_Forwarding_Explained) 5. [Troubleshooting](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Troubleshooting) |

Introduction

[Port forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Port_Forwarding_Explained) via SSH (**SSH tunneling**) creates a secure connection between a local computer and a remote machine through which services can be relayed. Because the connection is encrypted, SSH tunneling is useful for transmitting information that uses an unencrypted protocol, such as IMAP, VNC, or IRC.

Types of Port Forwarding

SSH's port forwarding feature can smuggle various types of Internet traffic into or out of a network. This can be used to avoid network monitoring or sniffers, or bypass badly configured routers on the Internet. Note: You might also need to change the settings in other programs (like your web browser) in order to circumvent these filters.

|  |  |
| --- | --- |
|  | **Warning**: Filtering and monitoring is usually implemented for a reason. Even if you don't agree with that reason, your IT department might not take kindly to you flouting their rules. |

There are three types of port forwarding with SSH:

1. **Local port forwarding**: connections *from the SSH client* are forwarded *via the SSH server*, then *to a destination server*
2. **Remote port forwarding**: connections *from the SSH server* are forwarded *via the SSH client*, then *to a destination server*
3. **Dynamic port forwarding**: connections from *various programs* are forwarded *via the SSH client*, then *via the SSH server*, and finally *to several destination servers*

Local port forwarding is the most common type. For example, local port forwarding lets you bypass a company firewall that blocks Wikipedia.

Remote port forwarding is less common. For example, remote port forwarding lets you connect from your SSH server to a computer on your company's intranet.

Dynamic port forwarding is rarely used. For example, dynamic port forwarding lets you bypass a company firewall that blocks web access altogether. Although this is very powerful, it takes a lot of work to set up, and it's usually easier to use local port forwarding for the specific sites you want to access.

Port-forwarding is a widely supported technique and a feature found in all major SSH clients and servers, although not all clients do it the same way. For help on using a specific client, consult the client's documentation. For example, the PuTTY manual has a [section on port forwarding in PuTTY](http://the.earth.li/~sgtatham/putty/0.60/htmldoc/Chapter3.html#using-port%20forwarding).

To use port forwarding, you need to [make sure port forwarding is enabled in your server](https://help.ubuntu.com/community/SSH/OpenSSH/Configuring#Forwarding). You also need to tell your client the source and destination *port numbers* to use. If you're using local or remote forwarding, you need to tell your client the *destination server*. If you're using dynamic port forwarding, you need to configure your programs to use a SOCKS proxy server. Again, exactly how to do this depends on which SSH client you use, so you may need to consult your documentation.

Local Port Forwarding

Local port forwarding lets you connect from your *local* computer to another server. To use local port forwarding, you need to know your destination server, and two port numbers. You should already know your destination server, and for basic uses of port forwarding, you can usually use the port numbers in Wikipedia's [list of TCP and UDP port numbers](http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers).

For example, say you wanted to connect from your laptop to [http://www.ubuntuforums.org](http://www.ubuntuforums.org/) using an SSH tunnel. You would use source port number 8080 (the alternate *http* port), destination port 80 (the *http* port), and destination server *www.ubuntuforums.org*. :

ssh -L 8080:www.ubuntuforums.org:80 <host>

Where <host> should be replaced by the name of *your laptop*. The -L option specifies *local* port forwarding. For the duration of the SSH session, pointing your browser at <http://localhost:8080/> would send you to <http://www.ubuntuforums.org/>.

In the above example, we used port 8080 for the source port. Ports numbers less than 1024 or greater than 49151 are reserved for the system, and some programs will only work with specific source ports, but otherwise you can use any source port number. For example, you could do:

ssh -L 8080:www.ubuntuforums.org:80 -L 12345:ubuntu.com:80 <host>

This would forward two connections, one to [www.ubuntuforums.org](http://www.ubuntuforums.org/), the other to [www.ubuntu.com](http://www.ubuntu.com/). Pointing your browser at <http://localhost:8080/> would download pages from [www.ubuntuforums.org](http://www.ubuntuforums.org/), and pointing your browser to <http://localhost:12345/> would download pages from [www.ubuntu.com](http://www.ubuntu.com/).

The destination server can even be the same as the SSH server. For example, you could do:

ssh -L 5900:localhost:5900 <host>

This would forward connections to the [shared desktop](https://help.ubuntu.com/community/VNC) on your SSH server (if one had been set up). Connecting an SSH client to *localhost* port 5900 would show the desktop for that computer. The word "localhost" is the computer equivalent of the word "yourself", so the SSH server on your laptop will understand what you mean, whatever the computer's actual name.

Remote Port Forwarding

Remote port forwarding lets you connect from the *remote* SSH server to another server. To use remote port forwarding, you need to know your destination server, and two port numbers. You should already know your destination server, and for basic uses of port forwarding, you can usually use the port numbers in Wikipedia's [list of TCP and UDP port numbers](http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers).

For example, say you wanted to let a friend access your remote desktop, using the command-line SSH client. You would use port number 5900 (the first *VNC* port), and destination server *localhost*:

ssh -R 5900:localhost:5900 guest@joes-pc

The -R option specifies *remote* port forwarding. For the duration of the SSH session, Joe would be able to access your desktop by connecting a VNC client to port 5900 on his computer (if you had set up a shared desktop).

Dynamic Port Forwarding

Dynamic port forwarding turns your SSH client into a *SOCKS proxy server*. *SOCKS* is a little-known but widely-implemented protocol for programs to request any Internet connection through a *proxy server*. Each program that uses the proxy server needs to be configured specifically, and reconfigured when you stop using the proxy server.

For example, say you wanted Firefox to connect to every web page through your SSH server. First you would use dynamic port forwarding with the default SOCKS port:

ssh -C -D 1080 laptop

The -D option specifies *dynamic* port forwarding. 1080 is the standard SOCKS port. Although you can use any port number, some programs will only work if you use 1080. -C enables compression, which [speeds the tunnel up](https://calomel.org/firefox_ssh_proxy.html) when proxying mainly text-based information (like web browsing), but can slow it down when proxying binary information (like downloading files).

Next you would tell Firefox to use your proxy:

1. go to Edit -> Preferences -> Advanced -> Network -> Connection -> Settings...
2. check "Manual proxy configuration"
3. make sure "Use this proxy server for all protocols" is cleared
4. clear "HTTP Proxy", "SSL Proxy", "FTP Proxy", and "Gopher Proxy" fields
5. enter "127.0.0.1" for "SOCKS Host"
6. enter "1080" (or whatever port you chose) for Port.

You can also set Firefox to use the DNS through that proxy, so even your DNS lookups are secure:

1. Type in about:config in the Firefox address bar
2. Find the key called "network.proxy.socks\_remote\_dns" and set it to true

The SOCKS proxy will stop working when you close your SSH session. You will need to change these settings back to normal in order for Firefox to work again.

To make other programs use your SSH proxy server, you will need to configure each program in a similar way.

Forwarding GUI Programs

SSH can also forward graphical applications over a network, although it can take some work and extra software to forward programs to Windows or Mac OS.

Single Applications

If you are logging in from a Unix-like operating system, you can forward single applications over SSH very easily, because all Unix-like systems share a common graphics layer called X11. This even works under Mac OS X, although you will need to install and start the [X11 server](http://www.apple.com/support/downloads/x11formacosx.html) before using SSH.

To forward single applications, connect to your system using the command-line, but add the **-X** option to forward X11 connections:

ssh -X laptop

Once the connection is made, type the name of your GUI program on the SSH command-line:

firefox &

Your program will start as normal, although you might find it's a little slower than it would be if it were running locally. The trailing & means that the program should run in "background mode", so you can start typing new commands in straight away, rather than waiting for your program to finish.

If you only want to run a single command, you can log in like this:

ssh -f -T -X laptop firefox

That will run Firefox, then exit when it finishes. See [the SSH manual page](http://manpages.ubuntu.com/manpages/karmic/en/man1/ssh.1.html) for information about **-f** and **-T**.

If you start an application and it complains that it cannot find the display, try installing the **xauth** package from the Main repository ([click here to install xauth](apt:xauth)). Xauth is installed by default with desktop installations but not server installations.

If you suspect that programs are running slowly because of a lack of bandwith, you can turn SSH compression on with the **-C** option:

ssh -fTXC joe@laptop firefox

Using **-fTXC** here is identical to **-f -T -X -C**.

Nested Windows

Xephyr is a program that gives you an X server within your current server. It's available in the **xserver-xephyr** package in the Main repository ([click here to install xserver-xephyr](apt:xserver-xephyr)).

Two ssh forwarded desktops on dual monitors, click to enlarge

Setting up Xephyr was explained briefly in the [Ubuntu forums](http://ubuntuforums.org/showthread.php?t=620003).

Port Forwarding Explained

To get the most out of port forwarding, it's helpful to know a bit about how the Internet works.

The Internet assigns computers virtual "ports", a bit like the USB ports on the back of your computer:

To let a digital camera share pictures with your PC, you connect the USB port on the camera to any USB port on the PC. The computer then talks to the camera about your photos, and shows you the result.

To let a web server share pages with your PC, you connect the web server port on the server to any Internet port on the PC. The computer then talks to the server about your page, and shows you the result.

Unlike a USB port, there is no physical component to an Internet port. There's no actual wire, or actual hole on the back of your computer. It's all just messages being sent over the Internet. Like other "virtual" computer concepts, Internet ports are just an analogy that help to explain what your computer is doing. Sometimes, that analogy breaks down:

There are two types of Internet port: normal "TCP" ports and strange "UDP" ports (which won't be covered here).

Unlike USB ports, every computer has exactly 65,535 numbered TCP ports, some of which have a special purpose. For example, port number 80 is your web server port, so your web browser knows it should connect to port number 80 in order to download a web page.

Connections between Internet ports can be patched together, so a connection from computer A to computer B on port 12,345 could be patched through to port number 80 on computer C. This is known as **port forwarding**.

Troubleshooting

If you get a message like this when you try to forward a port:

bind: Address already in use

channel\_setup\_fwd\_listener: cannot listen to port: <port number>

Could not request local forwarding.

then someone is already listening on that port number. You won't be able to listen on that port until the other person has finished with it.

If forwarding doesn't seem to work, even though you didn't get a warning message, then your SSH server might have disabled forwarding. To check, do the following:

grep Forwarding /etc/ssh/sshd\_config

If you see something like this:

X11Forwarding no

AllowTcpForwarding no

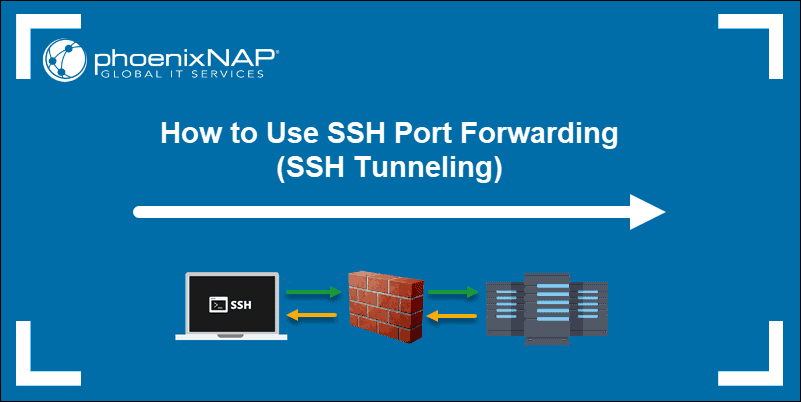
then forwarding is disabled on your server. See [the SSH configuration page](https://help.ubuntu.com/community/SSH/OpenSSH/Configuring#Forwarding) for more information.

**Introduction**

SSH port forwarding is often referred to as SSH tunneling, and the two terms are used interchangeably.

The encrypted SSH ‘tunnel’ serves as a vessel to transfer assorted data and deliver it safely to the remote system. This method is regularly used to circumvent standard firewall security protocols. Port-forwarding is a component of most SSH client and server programs.

Find out how to use [OpenSSH for Linux](https://phoenixnap.com/kb/openssl-tutorial-ssl-certificates-private-keys-csrs), and the Windows PuTTY client to **enable local, remote, or dynamic SSH port forwarding.**



**Prerequisites**

* An SSH client/server of your choice (OpenSSH or PuTTY)
* A reachable IP address or name of the remote/local server

**How to Use Local Port Forwarding**

Valuable network resources do not generally allow remote SSH access. This would be a severe limitation in a modern distributed environment. Organizations usually solve this issue by setting up an intermediary SSH ‘jump’ server to accept remote SSH connections.

Your local SSH client establishes a connection with the [remote SSH server](https://phoenixnap.com/kb/ssh-to-connect-to-remote-server-linux-or-windows). The connection is then forwarded to a resource within the trusted internal network. SSH connections are established, and security efforts can concentrate on the intermediary SSH server rather than individual resources in a network.

**Local Port Forwarding with OpenSSH**

To use SSH tunneling in Linux, you need to provide your client with the source and destination port numbers, as well as the location of the destination server. The location can either be an **IP address** or a **hostname**.

The basic syntax for a**local port forward** command is straightforward:

ssh -L local\_port:destination\_server\_ip:remote\_port ssh\_server\_hostname

⦁    **ssh** – Starts the SSH client program on the local machine and establishes a secure connection to the remote SSH server.

⦁    **-L local\_port:destination\_server\_ip:remote\_port** – The local port on the local client is being forwarded to the port of the destination remote server.

⦁    **ssh\_server\_hostname** – This element of the syntax represents the hostname or IP address of the remote SSH server.

A practical example of an SSH port forwarding command would take the following form:

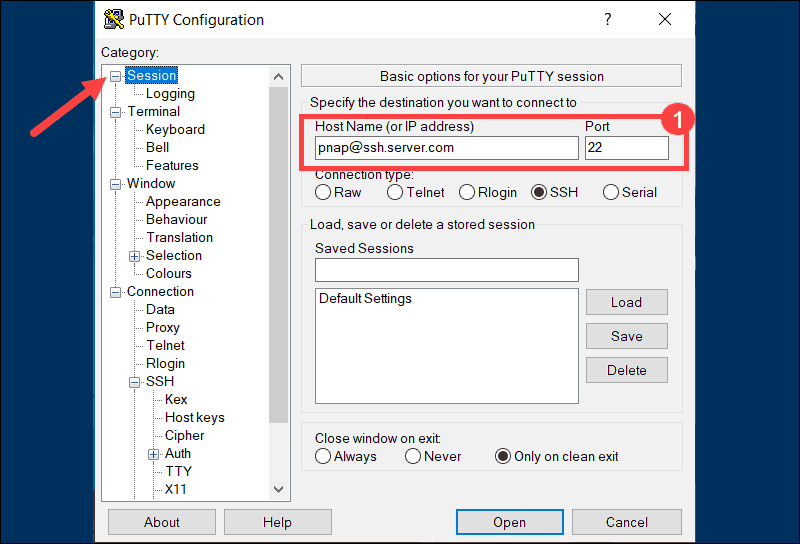
ssh –L 5901:188.17.0.5:4492 pnap@ssh.server.com

In the example above, all traffic sent to port **5901** on your local host is being forwarded to port **4492** on the remote server located at **188.17.0.5.**

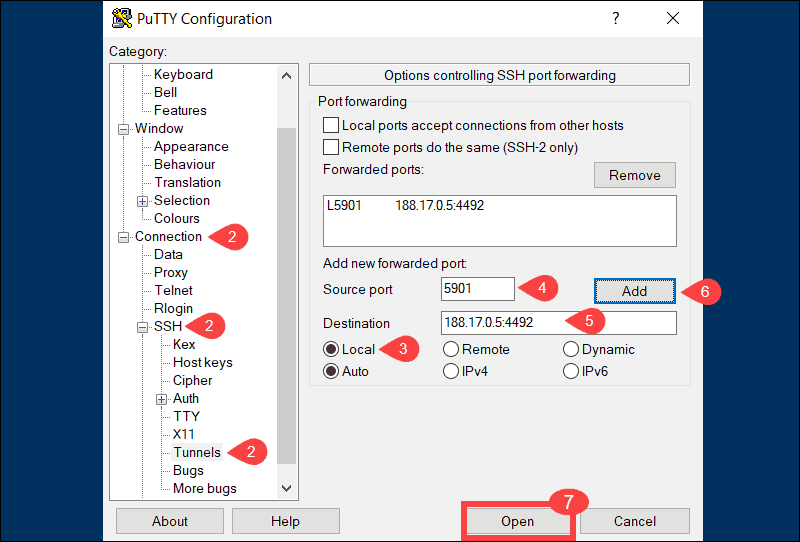
**Local Port Forwarding with PuTTY**

PuTTY is a user-friendly SSH client for Windows. Configure local SSH tunneling by following these steps:

1. When you start the PuTTY application, the **Sessions** screen allows you to enter the hostname or IP address and port number of the destination SSH server (Example: *pnap@ssh.server.com* and port number *22).*



1. Use the **Category** list on the left to navigate to **Connection > SSH > Tunnels**.
2. Select **Local** to define the type of SSH port forward.



1. Enter your local port number in the **Source port** field.
2. Type the destination address and port number in the **Destination** field. Use the following format: **destination\_server\_ip:remote\_port.**(e.g., Source port: 5901**,** Destination: 188.17.0.5:4492)
3. Once you verify that the information you entered is correct, select **Add**.
4. The parameters for the connection are now all set. The **Open** option initiates the local SSH port-forward.

**How to Use Remote Port Forwarding**

The purpose of remote forwarding is to allow a remote server to access resources on your local machine. Remote forwarding represents an inversion of the local forwarding process we explored previously.

**Remote Port Forwarding with OpenSSH**

The basic syntax for a **remote port forward** command is as follows:

ssh -R remote\_port:localhost:local\_port ssh\_server\_hostname

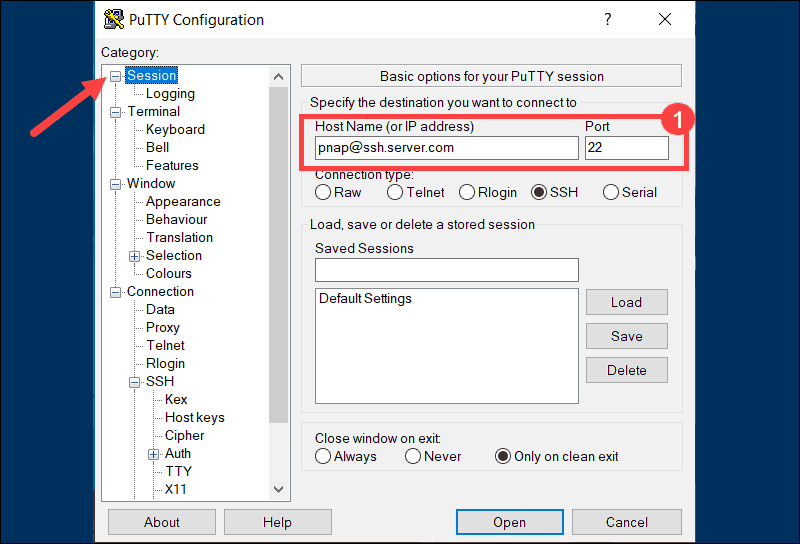
In this example, we have instructed the remote server **ssh.server.com** to forward any connections directed at port **8080** to the local resource listening on port **5534**.

ssh –R 8080:localhost:5534 pnap@ssh.server.com

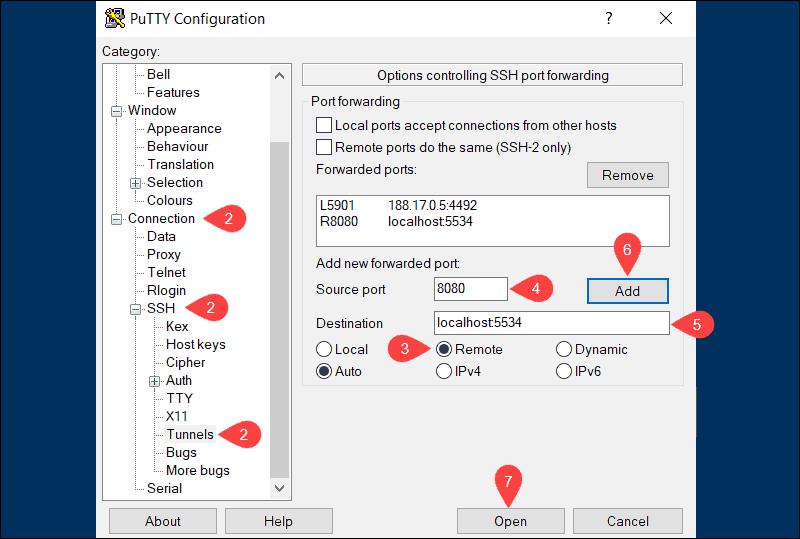
Users with access to the SSH server are now able to access resources on your local machine.

**Remote Port Forwarding with PuTTY**

1. Start the PuTTY application. The initial **Sessions** screen allows you to specify the IP address and port number of the destination SSH server.



1. Use the **Category** list on the left to navigate to **Connection > SSH > Tunnels**.
2. Select **Remote** to define the type of SSH port forward.



1. Enter the remote port number in the **Source port** field (e.g., *8080*).
2. Type the destination address and port number in the **Destination** field using the following format **localhost:local\_port** (e.g., *localhost:5534*).
3. Once you verify that the information you entered is correct, click **Add**.
4. Select **Open** to establish a remote SSH port-forward.

In the example above, users and applications can connect to port 8080 on the remote SSH server and then access the local machine using port 5534.

**How To Use Dynamic Port Forwarding**

Dynamic port forwarding is not as well-known as the previous two options. Most users find it easier to use local port forwarding to achieve similar results. Dynamic port forwarding can provide more flexibility as you no longer need to use a predefined remote port and server.

**Note:**When using dynamic port forwarding, it’s necessary to configure individual programs to use the SOCKS proxy server.

**Dynamic Port Forwarding with OpenSSH**

By using the **ssh** command and the **–D** argument, you can use your SSH client to create a SOCKS proxy on your local machine.:

ssh –D local\_port ssh\_server\_hostname

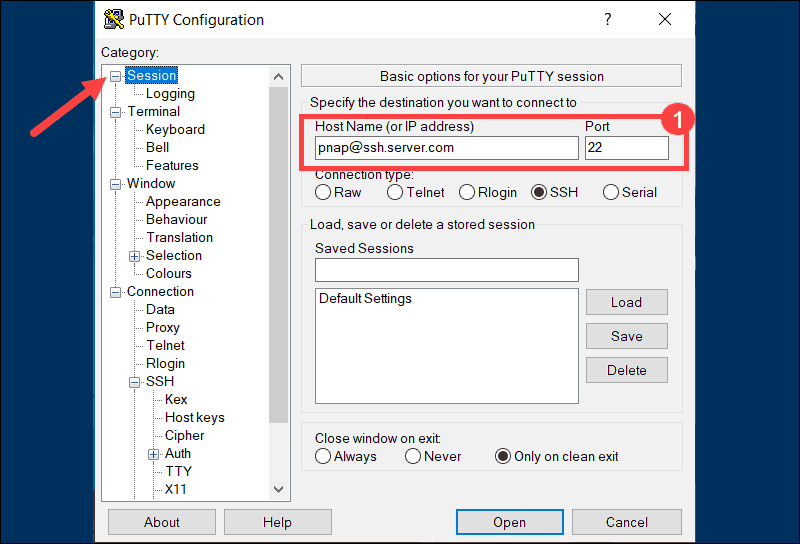
The following command opens a SOCKS proxy at port 5534 on your local machine:

ssh –D 5534 pnap@ssh.server.com

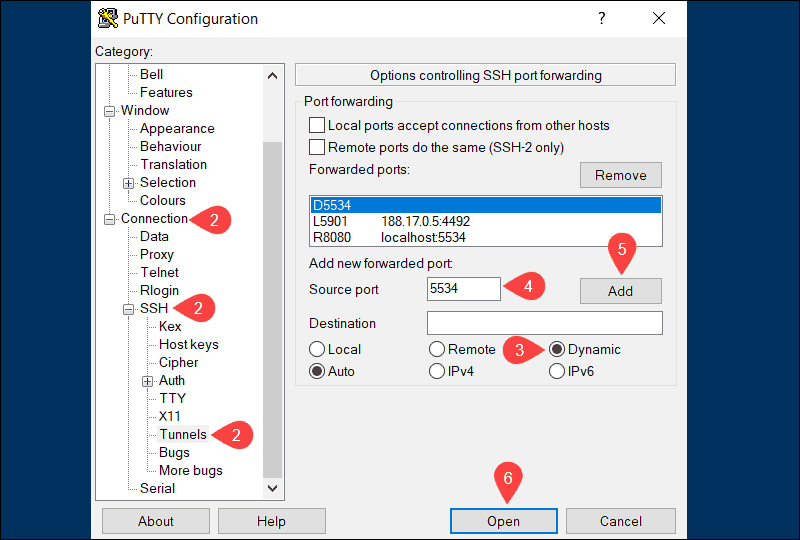
You are now able to configure a local resource, like a browser, to use port 5534. All traffic originating from that resource is directed through the SSH connections established for the defined port.

**Dynamic Port Forwarding with PuTTY**

1. Enter the hostname or IP address and port number of the destination SSH server on the main PuTTY **Sessions** screen.



1. Use the **Category** list to navigate to **Connection > SSH > Tunnels**.
2. Select **Dynamic** to define the type of SSH port forward.



1. Enter the dynamic port number in the **Source port** field (e.g., *5534*). The SOCKS proxy server on your local machine is going to use this port to dynamically forward traffic.
2. Once you are sure that the information you entered is correct, click **Add**.
3. The parameters for the connection are now all set. Select the **Open** option to start dynamic SSH port-forwarding.

For dynamic forwarding to work, you would need to configure and enable each application for the SOCKS proxy server.

**Conclusion**

This tutorial presented three different types of SSH port forwarding, the commands you need to use, and how to implement them on your system.

SSH port forwarding is a powerful feature, and it needs to be carefully administered. High-security standards should be maintained throughout the entire process.

SSH tunneling or SSH port forwarding is a method of creating an encrypted SSH connection between a client and a server machine through which services ports can be relayed.

SSH forwarding is useful for transporting network data of services that uses an unencrypted protocol, such as VNC or [FTP](https://linuxize.com/post/how-to-use-linux-ftp-command-to-transfer-files/) , accessing geo-restricted content or bypassing intermediate firewalls. Basically, you can forward any TCP port and tunnel the traffic over a secure SSH connection.

There are three types of SSH port forwarding:

* Local Port Forwarding. - Forwards a connection from the client host to the SSH server host and then to the destination host port.
* Remote Port Forwarding. - Forwards a port from the server host to the client host and then to the destination host port.
* Dynamic Port Forwarding. - Creates SOCKS proxy server which allows communication across a range of ports.

In this article, we will talk about how to set up local, remote, and dynamic encrypted SSH tunnels.

**Local Port Forwarding**

Local port forwarding allows you to forward a port on the local (ssh client) machine to a port on the remote (ssh server) machine, which is then forwarded to a port on the destination machine.

In this type of forwarding the SSH client listens on a given port and tunnels any connection to that port to the specified port on the remote SSH server, which then connects to a port on the destination machine. The destination machine can be the remote SSH server or any other machine.

Local port forwarding is mostly used to connect to a remote service on an internal network such as a database or VNC server.

In Linux, macOS and other Unix systems to create a local port forwarding pass the -L option to the ssh client:

ssh -L [LOCAL\_IP:]LOCAL\_PORT:DESTINATION:DESTINATION\_PORT [USER@]SSH\_SERVER

The options used are as follows:

* [LOCAL\_IP:]LOCAL\_PORT - The local machine ip and port number. When LOCAL\_IP is omitted the ssh client binds on localhost.
* DESTINATION:DESTINATION\_PORT - The IP or hostname and the port of the destination machine.
* [USER@]SERVER\_IP - The remote SSH user and server IP address.

You can use any port number greater than 1024 as a LOCAL\_PORT. Ports numbers less than 1024 are privileged ports and can be used only by root. If your SSH server is listening on a [port other than 22](https://linuxize.com/post/how-to-change-ssh-port-in-linux/) (the default) use the -p [PORT\_NUMBER] option.

The destination hostname must be resolvable from the SSH server.

Let’s say you have a MySQL database server running on machine db001.host on an internal (private) network, on port 3306 which is accessible from the machine pub001.host and you want to connect using your local machine mysql client to the database server. To do so you can forward the connection like so:

ssh -L 3336:db001.host:3306 user@pub001.host

Once you run the command, you’ll be prompted to enter the remote SSH user password. After entering it, you will be logged in to the remote server and the SSH tunnel will be established. It is a good idea to [set up an SSH key-based authentication](https://linuxize.com/post/how-to-setup-passwordless-ssh-login/) and connect to the server without entering a password.

Now if you point your local machine database client to 127.0.0.1:3336, the connection will be forwarded to the db001.host:3306 MySQL server through the pub001.host machine which will act as an intermediate server.

You can forward multiple ports to multiple destinations in a single ssh command. For example, you have another MySQL database server running on machine db002.host and you want to connect to both servers from your local client you would run:

ssh -L 3336:db001.host:3306 3337:db002.host:3306 user@pub001.host

Copy

To connect to the second server you would use 127.0.0.1:3337.

When the destination host is the same as the SSH server instead of specifying the destination host IP or hostname you can use localhost.

Say you need to connect to a remote machine through VNC which runs on the same server and it is not accessible from the outside. The command you would use is:

ssh -L 5901:127.0.0.1:5901 -N -f user@remote.host

The -f option tells the ssh command to run in the background and -N not to execute a remote command. We are using localhost because the VNC and the SSH server are running on the same host.

If you are having trouble setting up tunneling check your remote SSH server configuration and make sure AllowTcpForwarding is not set to no. By default, forwarding is allowed.

**Remote Port Forwarding**

Remote port forwarding is the opposite of local port forwarding. It allows you to forward a port on the remote (ssh server) machine to a port on the local (ssh client) machine, which is then forwarded to a port on the destination machine.

In this type of forwarding the SSH server listens on a given port and tunnels any connection to that port to the specified port on the local SSH client, which then connects to a port on the destination machine. The destination machine can be the local or any other machine.

In Linux, macOS and other Unix systems to create a remote port forwarding pass the -R option to the [ssh](https://linuxize.com/post/ssh-command-in-linux/) client:

ssh -R [REMOTE:]REMOTE\_PORT:DESTINATION:DESTINATION\_PORT [USER@]SSH\_SERVER

Copy

The options used are as follows:

* [REMOTE:]REMOTE\_PORT - The IP and the port number on the remote SSH server. An empty REMOTE means that the remote SSH server will bind on all interfaces.
* DESTINATION:DESTINATION\_PORT - The IP or hostname and the port of the destination machine.
* [USER@]SERVER\_IP - The remote SSH user and server IP address.

Remote port forwarding is mostly used to give access to an internal service to someone from the outside.

Let’s say you are developing a web application on your local machine and you want to show a preview to your fellow developer. You do not have a public IP so the other developer can’t access the application via the Internet.

If you have access to a remote SSH server you can set up a remote port forwarding as follows:

ssh -R 8080:127.0.0.1:3000 -N -f user@remote.host

The command above will make ssh server to listen on port 8080 and tunnel all traffic from this port to your local machine on port 3000.

Now your fellow developer can type the\_ssh\_server\_ip:8080 in his/her browser and preview your awesome application.

If you are having trouble setting up remote port forwarding make sure GatewayPorts is set to yes in the remote SSH server configuration.

**Dynamic Port Forwarding**

Dynamic port forwarding allows you to create a socket on the local (ssh client) machine which acts as a SOCKS proxy server. When a client connects to this port the connection is forwarded to the remote (ssh server) machine, which is then forwarded to a dynamic port on the destination machine.

This way, all the applications using the SOCKS proxy will connect to the SSH server and the server will forward all the traffic to its actual destination.

In Linux, macOS and other Unix systems to create a dynamic port forwarding (SOCKS) pass the -D option to the ssh client:

ssh -D [LOCAL\_IP:]LOCAL\_PORT [USER@]SSH\_SERVER

Copy

The options used are as follows:

* [LOCAL\_IP:]LOCAL\_PORT - The local machine ip and port number. When LOCAL\_IP is omitted the ssh client binds on localhost.
* [USER@]SERVER\_IP - The remote SSH user and server IP address.

A typical example of a dynamic port forwarding is to tunnel the web browser traffic through an SSH server.

The following command will create a SOCKS tunnel on port 9090:

ssh -D 9090 -N -f user@remote.host

Once the tunneling is established you can configure your application to use it. [This article](https://linuxize.com/post/how-to-setup-ssh-socks-tunnel-for-private-browsing/) explains how to configure Firefox and Google Chrome browser to use the SOCKS proxy.

The port forwarding has to be separately configured for each application that you want to tunnel the traffic thought it.

**Set up SSH Tunneling in Windows**

Windows users can create SSH tunnels using the PuTTY SSH client. You can download PuTTY [here](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) .

1. Launch Putty and enter the SSH server IP Address in the Host name (or IP address) field.
2. Under the Connection menu, expand SSH and select Tunnels. Check the Local radio button to setup local, Remote for remote, and Dynamic for dynamic port forwarding.
   * If setting up local forwarding enter the local forwarding port in the Source Port field and in Destination enter the destination host and IP, for example, localhost:5901.
   * For remote port forwarding enter the remote SSH server forwarding port in the Source Port field and in Destination enter the destination host and IP, for example, localhost:3000.
   * If setting up dynamic forwarding enter only the local SOCKS port in the Source Port field.
3. Click on the Add button as shown in the image below.
4. Go back to the Session page to save the settings so that you do not need to enter them each time. Enter the session name in the Saved Session field and click on the Save button.
5. Select the saved session and log in to the remote server by clicking on the Open button.

A new window asking for your username and password will show up. Once you enter your username and password you will be logged in to your server and the SSH tunnel will be started.

Setting up [public key authentication](https://linuxize.com/post/generate-ssh-keys-on-windows-with-puttygen/) will allow you to connect to your server without entering a password.

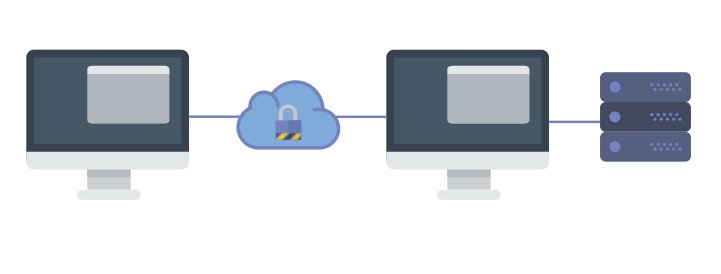
**Conclusion**

We have shown you how to set up SSH tunnels and forward the traffic through a secure SSH connection. For ease of use, you can define the SSH tunnel in your [SSH config file](https://linuxize.com/post/using-the-ssh-config-file/) or create a [Bash alias](https://linuxize.com/post/how-to-create-bash-aliases/) that will set up the SSH tunnel.

If you hit a problem or have feedback, leave a comment below.

A Guide to SSH Port Forwarding/Tunnelling

BY [SUPRIYO BISWAS](https://www.booleanworld.com/author/supriyo_biswas/)



SSH is a widely used protocol for system administration and file transfer. In addition, it has a feature called SSH tunnelling  (or SSH port forwarding). It creates an encrypted connection between a local computer and a remote computer through which you can relay traffic. It is very useful, and you can use it to securely access unencrypted protocols such as VNC or firewalled resources like database servers. You can also use it as a form of proxy/VPN and get around restrictive, firewalled networks.

In this article, we’re going to take a look at using SSH port forwarding.

Prerequisites

Before you can begin, you need to check if forwarding is allowed on the SSH server you’ll connect to.

If you’re using the OpenSSH server, open /etc/ssh/sshd\_config in a text editor. If you find AllowTcpForwarding is set to No, change them to Yes. In addition, if you’re going to use remote port forwarding (discussed later in this article), you also have to set GatewayPorts to Yes. Then, you need to restart the server for the change to take effect.

If you’re on Linux, depending upon the init system used by your distribution, run:

sudo systemctl restart sshd

sudo service sshd restart

Again, depending on your distribution, you may find that the service is named ssh instead of sshd.

If you’re on a Mac, you can restart the server like so:

sudo launchctl unload /System/Library/LaunchDaemons/ssh.plist

sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist

If you’re on Windows and want to set up a SSH server, have a look at [MSYS2](https://www.booleanworld.com/get-unix-linux-environment-windows-msys2/) or [Cygwin](https://www.cygwin.com/).

You also need to have a SSH client on the computer you’re working on. On most Unix-like systems, it’s already installed by default. If you’re on Windows 10 and you use Bash on Windows, you can install OpenSSH in it the way you would on a regular Ubuntu system. However, you may need to add the -4 switch, as IPv6 is not supported properly there.

For other versions of Windows, you can use the OpenSSH package from MSYS2 or Cygwin. If you’re not willing to bring an entire Unix-like system on your computer, try [Putty](http://www.chiark.greenend.org.uk/~sgtatham/putty/).

(If you’re using Putty as your SSH client, simply replace the ssh command with putty in the examples below.)

Local port forwarding

Local port forwarding allows you to forward traffic on a port of your local computer to the SSH server, which is forwarded to a destination server.

That was quite a mouthful, so let’s look at a few examples. Say, you want to connect to a database server running at port 3306 on your server. The port for the database server is firewalled to protect it from external attackers. Although you can’t directly access the database server, you can do through SSH.

On your local computer, type in:

ssh -L 4000:127.0.0.1:3306 user@example.com

Now, SSH will bind to port 4000 on your computer. Any traffic that comes to this port is sent to the SSH server. Then, the traffic received is sent to port 3306 of 127.0.0.1, which is the server itself.

Now, you can connect to the database. You simply need to set the database client to use 127.0.0.1 as the host and 4000 as the port.

Moreover, you can forward multiple sets of ports in a single ssh command:

ssh -L 5901:127.0.0.1:5901 -L 4000:127.0.0.1:3306 user@example.com

In addition to forwarding the local port 4000 to 127.0.0.1:3306, we’ve also forwarded the local port 5901 to 127.0.0.1:5901.

Note that the destination to which the SSH server forwards the traffic is from the perspective of the server itself. This has some interesting implications. Say for example, there’s a single server having public SSH access in your company. You want to connect to the database server, which is running on server003.local in the internal network, on port 3306. Now, you can forward ports like so:

ssh -L 4000:server003.local:3306 user@example.com

Now, point your database client to 127.0.0.1:3306 and bam — you’re connected to the database server!

Remote port forwarding

Remote port forwarding is the exact opposite of local port forwarding. It forwards traffic coming to a port on your server to your local computer, and then it is sent to a destination. Again, let’s take a look at an example.

Suppose, you’re developing a web application that’s running on port 8000 of your local computer. Other people can’t access it directly because you’re sitting behind a NAT network without a public IP. Now, you would like to show a customer how the application looks like. Fortunately, remote forwarding can help you with this. Type in:

ssh -R 7000:127.0.0.1:8000 user@example.com

When you run this command, the SSH server binds to the 7000 port on example.com. Any traffic that it receives on this port is sent to the SSH client on your local computer, which in turn forwards it to port 8000 on 127.0.0.1. Now, the customer can open http://example.com:7000 in a browser, and they can use your application!

Again, as with remote port forwarding, you can also use a different destination. Say for example, you want a friend to help you with configuring a router. Since they can’t access your router directly, you can use remote port forwarding like so:

ssh -R 8080:192.168.100.1:8000 user@example.com

Dynamic port forwarding

So far, we’ve seen how to forward local ports with SSH. Although it comes very close to acting like a proxy, but you can’t use it as-is. This is because, you’ll need to specify different ports for every destination and every service you want to access, which simply isn’t practical.

Fortunately, there’s another type of forwarding called “dynamic port forwarding” which can be used for this purpose. In order to set up dynamic port forwarding, type the following command:

ssh -D 4000 user@example.com

The SSH client creates a SOCKS proxy at port 4000 on your local computer. Any traffic sent to this port is sent to its destination through the SSH server.

Next, you’ll need to configure your applications to use this server. The “Settings” section of most web browsers allow you to use a SOCKS proxy. Moreover, there are some extensions like [Proxy Helper](https://chrome.google.com/webstore/detail/mnloefcpaepkpmhaoipjkpikbnkmbnic)(Chrome) and [FoxyProxy Standard](https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/)(Firefox) that allow you to set up and switch proxies quickly.

For other applications, the Arch Linux wiki has some great instructions on [using SOCKS proxies](https://wiki.archlinux.org/index.php/proxy_settings#Using_a_SOCKS_proxy).

A few tips

If you’ve tried the above examples, you may have noticed that running the command also opens up a shell. If you don’t need the shell, you can disable it with the -N switch:

ssh -N ... user@example.com

If you want SSH to go into the background, you can do so with the -f flag.

ssh -f -N ... user@example.com

If you’re using SSH as a proxy, it’s best not to run a HTTP server on the remote machine. This is because many services will refuse registrations and transactions under the assumption that you’re using a web proxy.

Conclusion

In this article, we’ve seen how to use SSH forwarding, and how its useful in a variety of situations. If you have questions, feel free to ask in the comments below!

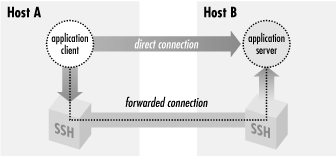
(BTW, if you’re using any of this at your workplace, you should first see if it’s okay with management — they might not be cool with it, after all.)

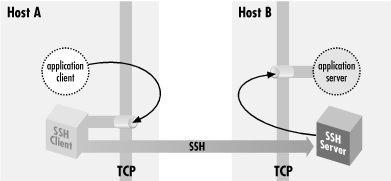
**If you liked this post, please share it 🙂**

**The How To Of Port Forwarding With SSH**

[#devops](https://dev.to/t/devops) [#security](https://dev.to/t/security) [#linux](https://dev.to/t/linux) [#ssh](https://dev.to/t/ssh)

[samuyi profile imageSamuyi](https://dev.to/samuyi)19 Nov 2018 ・5 min read

[](https://res.cloudinary.com/practicaldev/image/fetch/s--y_5i62YS--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/qpawdpvr1du4fwbknq50.gif)  
Port forwarding is a type of interaction between two applications, usually TCP/IP applications, that talk to each other using an SSH connection. SSH intercepts a service request from a client application on a host, creates an SSH session carrying the request to the other side of the SSH connection. The other side decrypts the request before sending it to the application server on the remote host. Port forwarding can be used to secure communications between applications that aren’t secured traditionally. They can also be used for communications that aren’t possible, for instance IT administrators block certain ports on hosts from external access with firewalls to improve security, with port forwarding it becomes possible to access those applications running on the remote machine. In a previous [post](https://dev.to/samuyi/ssh-agents-in-depth-4116) we talked about a different type of forwarding called ssh-agent forwarding. This lets us create SSH connections from one computer, through a remote host, to a third remote host using public-key authentication without the need to have your private keys on the second remote host. Port forwarding is sometimes refereed to as “tunneling” because it provides a means for which you can secure TCP/IP connections through SSH.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--X7qVeNxx--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/la8u765h1lxutfmvonsx.gif)

Local Port Forwarding

Suppose you have an IMAP server running on a remote host and you want to access the server using an email client on your home machine. Suppose also that the administrators of the remote host are super paranoid and they block all external access except only on port 80, 433 and 22. Unfortunately since IMAP runs on port 143 you can’t access it from your home. All you need to do is to tunnel through to the IMAP server using SSH. The command you need to run on your local machine is:

$ ssh -L2001:localhost:143 remote.net

Lets break down the above command. The -L switch specifies local forwarding; this essentially says the TCP client ( your email client) , is on your local machine. The 2001 represents the port on your local machine you want your email client to connect to. The localhost means the source sockets of the connection on the server appear to come from localhost. This means that the above command could be written as:

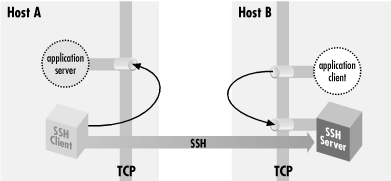
$ ssh -L2001:remote.net:143 remote.net

And the source packets would appear as coming from remote.net address. This maybe trivial in some TCP applications but, some servers are configured to do access control and block connections from the loopback address. Or it might be running on a multi-homed host, and have bound only a subset of the addresses the host has, possibly not including the loopback address. It’s generally better to use the first command. Finally 143 represents the port number the IMAP server is running on the remote server.

By default your SSH client only listens to connections from your local machine. That is it only accepts connections from applications running locally on your machine. Any attempt from external applications to attempt to connect over the wire to your SSH client will fail by default. To enable this you need to tweak the command above like so:

$ ssh -g -L2001:localhost:143 remote.net

The -g switch represents the GateWayPorts option in the client configuration file; if set it to yes then there’s no need for the -g option.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--6ht2e_RB--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_66%2Cw_880/https:/thepracticaldev.s3.amazonaws.com/i/xl0u5wy98adbgna4zr2a.gif)

Remote Port Forwarding

An example of remote forwarding is, suppose we have an email client running on a remote server shell.isp.net and we want to access an IMAP server on another remote server remote.host.net with an SSH server installed then remote forwarding is the way to go in this case. The command to establish remote forwarding is:

$ ssh -R2001:localhost:143 remote.host.net

The syntax is similar to local port forwarding with the only difference being the -R switch instead of the -L in local port forwarding.

There are subtle differences between remote and local forwarding. The main difference being that in local forwarding the SSH client listens for communication from the application client and therefore usually resides on the same box as the application client. In remote forwarding the SSH server listens for communication from the application client, the SSH server and application client reside on the same host.

Dynamic Port Forwarding

You may ask is it possible to tunnel HTTP connections on port 80 to access an insecure website using local forwarding? The answer is yes but there are many restrictions that come with it. Such as, since the browser is running only on localhost then following an absolute URL such as “[http://inseure/web/now.html”](http://inseure/web/now.html%E2%80%9D) would not work since the browser only knows localhost also trying to proxy the request with your browser won’t work with any connections other than port 80. What would work for us in this case is dynamic port forwarding. [SOCKS](https://en.wikipedia.org/wiki/SOCKS) is a dynamic forwarding protocol, it is used by popular browsers such as [tor](https://www.torproject.org/) to proxy requests and tunnel through to websites censored in certain countries and protect the users privacy. A SOCKS client connects via TCP, and indicates via the protocol the remote socket it wants to reach; the SOCKS server makes the connection, then gets out of the way, transparently passing data back and forth. The command to enable this with SSH is run like so:

$ ssh -D 1080 remote.host.net

When a user types an absolute URL into the browser including any port such as “[http://myweb:1890”](http://myweb:1890%E2%80%9D/). The browser connects to SSH socks proxy on port 1080 and asks for connection to myweb:1890. The SSH client associates the browser’s connection with a new SSH session and then connects to the SSH server. The SSH client and server essentially get out of the way and the browser directly connects to the web server. Each new connection to a different web site gets assigned a new socket by SSH.

Conclusion

Port forwarding is a general TCP proxying feature that tunnels TCP connections through an SSH session. It is useful for securing protocols that may not be secure and tunneling through to TCP connections that may be blocked by firewalls. However there are some TCP protocols that may not work properly with port forwarding such as FTP. It so happens that FTP opens random ports on a client after authentication. This makes port forwarding unnecessarily complex. By and large majority of the TCP protocols do work with port forwarding.

An SSH client connects to a [Secure Shell server](https://www.howtogeek.com/114812/5-cool-things-you-can-do-with-an-ssh-server/), which allows you to run terminal commands as if you were sitting in front of another computer. But an SSH client also allows you to “tunnel” a port between your local system and a remote SSH server.

There are three different types of SSH tunneling, and they’re all used for different purposes. Each involves using an SSH server to redirect traffic from one network port to another. The traffic is sent over the encrypted SSH connection, so it can’t be monitored or modified in transit.

You can do this with the ssh command included on Linux, macOS, and other [UNIX-like](https://www.howtogeek.com/182649/htg-explains-what-is-unix/) operating systems. On Windows, which doesn’t include a built-in ssh command, we recommend the free tool [PuTTY](http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) to connect to SSH servers. It supports SSH tunneling, too.

**Local Port Forwarding: Make Remote Resources Accessible on Your Local System**

“Local port forwarding” allows you to access local network resources that aren’t exposed to the Internet. For example, let’s say you want to access a database server at your office from your home. For security reasons, that database server is only configured to accept connections from the local office network. But if you have access to an SSH server at the office, and that SSH server allows connections from outside the office network, then you can connect to that SSH server from home and access the database server as if you were in the office. This is often the case, as it’s easier to secure a single SSH server against attacks than to secure a variety of different network resources.

To do this, you establish an SSH connection with the SSH server and tell the client to forward traffic from a specific port from your local PC—for example, port 1234—to the address of the database’s server and its port on the office network. So, when you attempt to access the database server at port 1234 your current PC, “localhost”, that traffic is automatically “tunneled” over the SSH connection and sent to the database server. The SSH server sits in the middle, forwarding traffic back and forth. You can use any command line or graphical tool to access the database server as if it was running on your local PC.

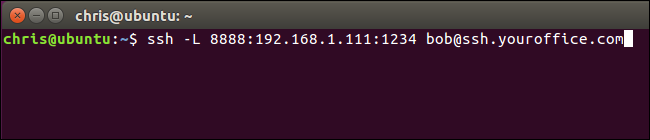
To use local forwarding, connect to the SSH server normally, but also supply the -L argument. The syntax is:

ssh -L local\_port:remote\_address:remote\_port username@server.com

For example, let’s say the database server at your office is located at 192.168.1.111 on the office network. You have access to the office’s SSH server at ssh.youroffice.com , and your user account on the SSH server is bob . In that case, your command would look like this:

ssh -L 8888:192.168.1.111:1234 bob@ssh.youroffice.com

After running that command, you’d be able to access the database server at port 8888 at localhost. So, if the database server offered web access, you could plug http://localhost:8888 into your web browser to access it. If you had a command line tool that needs the network address of a database, you’d point it at localhost:8888. All traffic sent to port 8888 on your PC will be tunneled to 192.168.1.111:1234 on your office network.

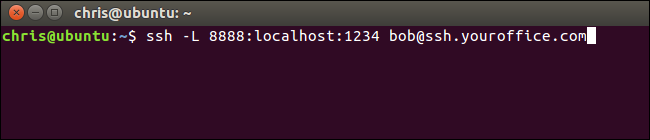


It’s a little more confusing if you want to connect to a server application running on the same system as the SSH server itself. For example, let’s say you have an SSH server running at port 22 on your office computer, but you also have a database server running at port 1234 on the same system at the same address. You want to access the database server from home, but the system is only accepting SSH connections on port 22 and its firewall doesn’t allow any other external connections.

In this case, you could run a command like the following one:

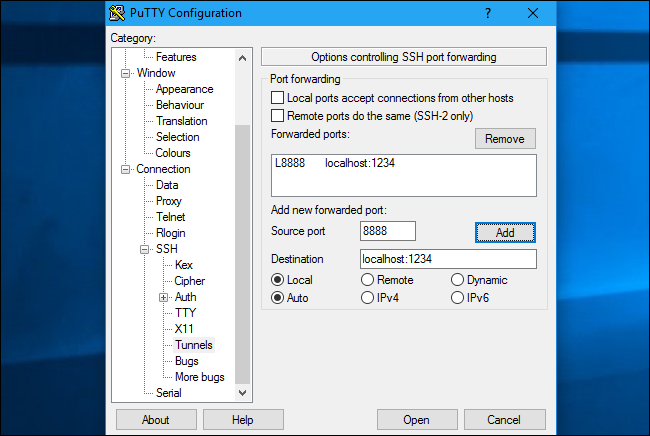
ssh -L 8888:localhost:1234 bob@ssh.youroffice.com

When you attempt to access the database server at port 8888 on your current PC, the traffic will be sent over the SSH connection. When it arrives on the system running the SSH server, the SSH server will send it to port 1234 on “localhost”, which is the same PC running the SSH server itself. So the “localhost” in the command above means “localhost” from the perspective of the remote server.



To do this in the PuTTY application on Windows, select Connection > SSH > Tunnels. Select the “Local” option. For “Source Port”, enter the local port. For “Destination”, enter the destination address and port in the form remote\_address:remote\_port.

For example, if you wanted to set up the same SSH tunnel as above, you’d enter 8888 as the source port and localhost:1234 as the destination. Click “Add” afterwards and then click “Open” to open the SSH connection. You will also need to enter the address and port of the SSH server itself on the main “Session” screen before connecting, of course.



**Remote Port Forwarding: Make Local Resources Accessible on a Remote System**

“Remote port forwarding” is the opposite of local forwarding, and isn’t used as frequently. It allows you to make a resource on your local PC available on the SSH server. For example, let’s say you’re running a web server on the local PC you’re sitting in front of. But your PC is behind a firewall that doesn’t allow incoming traffic to the server software.

Assuming you can access a remote SSH server, you can connect to that SSH server and use remote port forwarding. Your SSH client will tell the server to forward a specific port—say, port 1234—on the SSH server to a specific address and port on your current PC or local network. When someone accesses the port 1234 on the SSH server, that traffic will automatically be “tunneled” over the SSH connection. Anyone with access to the SSH server will be able to access the web server running on your PC. This is effectively a way to tunnel through firewalls.

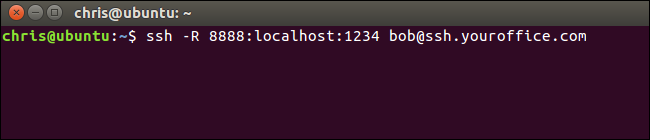
To use remote forwarding, use the ssh command with the -R argument. The syntax is largely the same as with local forwarding:

ssh -R remote\_port:local\_address:local\_port username@server.com

Let’s say you want to make a server application listening at port 1234 on your local PC available at port 8888 on the remote SSH server. The SSH server’s address is ssh.youroffice.com and your username on the SSH server is *bob*. You’d run the following command:

ssh -R 8888:localhost:1234 bob@ssh.youroffice.com

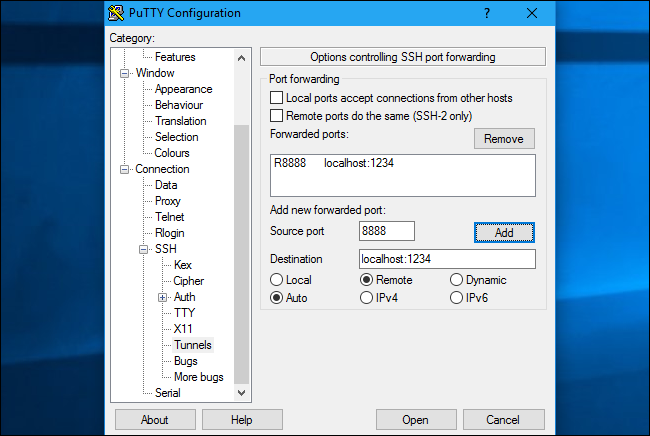
Someone could then connect to the SSH server at port 8888 and that connection would be tunneled to the server application running at port 1234 on the local PC you established the connection from.



To do this in PuTTY on Windows, select Connection > SSH > Tunnels. Select the “Remote” option. For “Source Port”, enter the remote port. For “Destination”, enter the destination address and port in the form local\_address:local\_port.

For example, if you wanted to set up the example above, you’d enter 8888 as the source port and localhost:1234 as the destination. Click “Add” afterwards and then click “Open” to open the SSH connection. You will also need to enter the address and port of the SSH server itself on the main “Session” screen before connecting, of course.

People could then connect to port 8888 on the SSH server and their traffic would be tunneled to port 1234 on your local system.



By default, the remote SSH server will only listen to connections from the same host. In other words, only people on the same system as the SSH server itself will be able to connect. This is for security reasons. You’ll need to enable the “GatewayPorts” option in [sshd\_config](http://manpages.ubuntu.com/manpages/zesty/en/man5/sshd_config.5.html) on the remote SSH server if you want to override this behavior.

**Dynamic Port Forwarding: Use Your SSH Server as a Proxy**

**RELATED:** [***What's the Difference Between a VPN and a Proxy?***](https://www.howtogeek.com/247190/whats-the-difference-between-a-vpn-and-a-proxy/)

There’s also “dynamic port forwarding”, which works similarly to a proxy or VPN. The SSH client will create a [SOCKS proxy](https://www.howtogeek.com/247190/whats-the-difference-between-a-vpn-and-a-proxy/) you can configure applications to use. All the traffic sent through the proxy would be sent through the SSH server. This is similar to local forwarding—it takes local traffic sent to a specific port on your PC and sends it over the SSH connection to a remote location.

**RELATED:** [***Why Using a Public Wi-Fi Network Can Be Dangerous, Even When Accessing Encrypted Websites***](https://www.howtogeek.com/178696/why-using-a-public-wi-fi-network-can-be-dangerous-even-when-accessing-encrypted-websites/)

For example, let’s say you’re using a public Wi-Fi network. You want to [browse securely without being snooped on](https://www.howtogeek.com/178696/why-using-a-public-wi-fi-network-can-be-dangerous-even-when-accessing-encrypted-websites/). If you have access to an SSH server at home, you could connect to it and use dynamic port forwarding. The SSH client will create a SOCKS proxy on your PC. All traffic sent to that proxy will be sent over the SSH server connection. No one monitoring the public Wi-Fi network will be able to monitor your browsing or censor the websites you can access. From the perspective of any websites you visit, it will be as if you were sitting in front of your PC at home. This also means you could use this trick to access US-only websites while outside of the USA—assuming you have access to an SSH server in the USA, of course.

As an another example, you may want to access a media server application you have on your home network. For security reasons, you may only have an SSH server exposed to the Internet. You don’t allow incoming connections from the Internet to your media server application. You could set up dynamic port forwarding, configure a web browser to use the SOCKS proxy, and then access servers running on your home network through the web browser as if you were sitting in front of your SSH system at home. For example, if your media server is located at port 192.168.1.123 on your home network, you could plug the address 192.168.1.123 into any application using the SOCKS proxy and you’d access the media server as if you were on your home network.

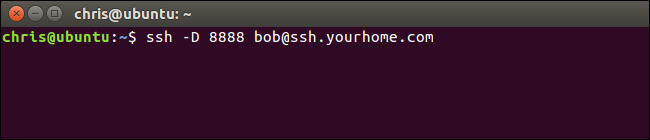
To use dynamic forwarding, run the ssh command with the -D argument, like so:

ssh -D local\_port username@server.com

For example, let’s say you have access to an SSH server at ssh.yourhome.com and your username on the SSH server is bob . You want to use dynamic forwarding to open a SOCKS proxy at port 8888 on the current PC. You’d run the following command:

ssh -D 8888 bob@ssh.yourhome.com

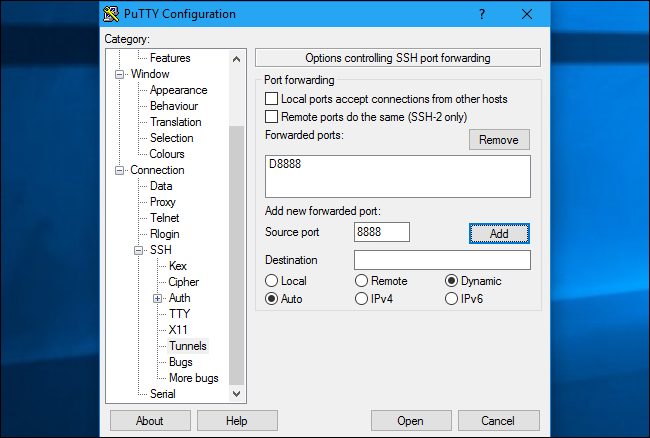
You could then configure a web browser or another application to use your local IP address (127.0.01) and port 8888. All traffic from that application would be redirected through the tunnel.



To do this in PuTTY on Windows, select Connection > SSH > Tunnels. Select the “Dynamic” option. For “Source Port”, enter the local port.

For example, if you wanted to create a SOCKS proxy on port 8888, you’d enter 8888 as the source port. Click “Add” afterwards and then click “Open” to open the SSH connection. You will also need to enter the address and port of the SSH server itself on the main “Session” screen before connecting, of course.

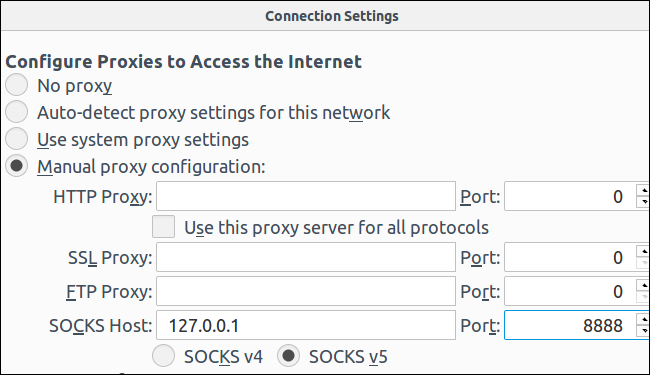
You could then configure an application to access the SOCKS proxy on your local PC (that is, IP address 127.0.0.1, which points to your local PC) and specify the correct port.



**RELATED:** [***How to Configure a Proxy Server in Firefox***](https://www.howtogeek.com/293213/how-to-configure-a-proxy-server-in-firefox/)

For example, you can [configure Firefox to use the SOCKS proxy](https://www.howtogeek.com/293213/how-to-configure-a-proxy-server-in-firefox/). This is particularly useful because Firefox can have its own proxy settings and doesn’t have to use system-wide proxy settings. Firefox will send its traffic through the SSH tunnel, while other applications will use your Internet connection normally.

When doing this in Firefox, select “Manual proxy configuration”, enter “127.0.0.1” into the SOCKS host box, and enter the dynamic port into the “Port” box. Leave the HTTP Proxy, SSL Proxy, and FTP Proxy boxes empty.



The tunnel will remain active and open for as long as you have the SSH session connection open. When you end your SSH session and disconnect from a server, the tunnel will also be closed. Just reconnect with the appropriate command (or the appropriate options in PuTTY) to reopen the tunnel.