# Using SSH agent forwarding

To simplify deploying to a server, you can set up SSH agent forwarding to securely use local SSH keys.

**In this article**



SSH agent forwarding can be used to make deploying to a server simple. It allows you to use your local SSH keys instead of leaving keys (without passphrases!) sitting on your server.

If you've already set up an SSH key to interact with GitHub, you're probably familiar with `ssh-agent`. It's a program that runs in the background and keeps your key loaded into memory, so that you don't need to enter your passphrase every time you need to use the key. The nifty thing is, you can choose to let servers access your local `ssh-agent` as if they were already running on the server. This is sort of like asking a friend to enter their password so that you can use their computer.

Check out Steve Friedl's Tech Tips guide for a more detailed explanation of SSH agent forwarding.

## Setting up SSH agent forwarding

Ensure that your own SSH key is set up and working. You can use our guide on generating SSH keys if you've not done this yet.

You can test that your local key works by entering `ssh -T git@github.com` in the terminal:

```
$ ssh -T git@github.com
# Attempt to SSH in to github
> Hi username! You've successfully authenticated, but GitHub does not provide
> shell access.
```

We're off to a great start. Let's set up SSH to allow agent forwarding to your server.

1. Using your favorite text editor, open up the file at `~/.ssh/config`. If this file doesn't exist, you can create it by entering `touch ~/.ssh/config` in the terminal.
2. Enter the following text into the file, replacing `example.com` with your server's domain name or IP:

```
Host example.com
  ForwardAgent yes
```

**Warning:** You may be tempted to use a wildcard like `Host *` to just apply this setting to all SSH connections. That's not really a good idea, as you'd be sharing your local SSH keys with *every* server you SSH into. They won't have direct access to the keys, but they will be able to use them *as you* while the connection is established. **You should only add servers you trust and that you intend to use with agent forwarding.**

## Testing SSH agent forwarding

To test that agent forwarding is working with your server, you can SSH into your server and run `ssh -T git@github.com` once more. If all is well, you'll get back the same prompt as you did locally.

If you're unsure if your local key is being used, you can also inspect the `SSH_AUTH_SOCK` variable on your server:

```
$ echo "$SSH_AUTH_SOCK"
# Print out the SSH_AUTH_SOCK variable
> /tmp/ssh-4hNGMk8AZX/agent.79453
```

If the variable is not set, it means that agent forwarding is not working:

```
$ echo "$SSH_AUTH_SOCK"
# Print out the SSH_AUTH_SOCK variable
> [No output]
$ ssh -T git@github.com
# Try to SSH to github
> Permission denied (publickey).
```

## Troubleshooting SSH agent forwarding

Here are some things to look out for when troubleshooting SSH agent forwarding.

### You must be using an SSH URL to check out code

SSH forwarding only works with SSH URLs, not HTTP(s) URLs. Check the `.git/config` file on your server and ensure the URL is an SSH-style URL like below:

```
[remote "origin"]
  url = git@github.com:yourAccount/yourProject.git

  fetch = +refs/heads/*:refs/remotes/origin/*
```

### Your SSH keys must work locally

Before you can make your keys work through agent forwarding, they must work locally first. Our guide on generating SSH keys can help you set up your SSH keys locally.

### Your system must allow SSH agent forwarding

Sometimes, system configurations disallow SSH agent forwarding. You can check if a system configuration file is being used by entering the following command in the terminal:

```
$ ssh -v example.com
# Connect to example.com with verbose debug output
> OpenSSH_8.1p1, LibreSSL 2.7.3
> debug1: Reading configuration data /Users/you/.ssh/config
> debug1: Applying options for example.com
> debug1: Reading configuration data /etc/ssh_config
> debug1: Applying options for *
$ exit
# Returns to your local command prompt
```

In the example above, the file `~/.ssh/config` is loaded first, then `/etc/ssh_config` is read. We can inspect that file to see if it's overriding our options by running the following commands:

```
$ cat /etc/ssh_config
```

```
$ cat /etc/ssh_config
# Print out the /etc/ssh_config file
> Host *
>   SendEnv LANG LC_*
>   ForwardAgent no
```

In this example, our `/etc/ssh_config` file specifically says `ForwardAgent no`, which is a way to block agent forwarding. Deleting this line from the file should get agent forwarding working once more.

## Your server must allow SSH agent forwarding on inbound connections

Agent forwarding may also be blocked on your server. You can check that agent forwarding is permitted by SSHing into the server and running `sshd_config`. The output from this command should indicate that `AllowAgentForwarding` is set.

## Your local `ssh-agent` must be running

On most computers, the operating system automatically launches `ssh-agent` for you. On Windows, however, you need to do this manually. We have a guide on how to start `ssh-agent` whenever you open Git Bash.

To verify that `ssh-agent` is running on your computer, type the following command in the terminal:

```
$ echo "$SSH_AUTH_SOCK"
# Print out the SSH_AUTH_SOCK variable
> /tmp/launch-kNSlgU/Listeners
```

## Your key must be available to `ssh-agent`

You can check that your key is visible to `ssh-agent` by running the following command:

```
ssh-add -L
```

If the command says that no identity is available, you'll need to add your key:

```
$ ssh-add yourkey
```

On macOS, `ssh-agent` will "forget" this key, once it gets restarted during reboots. But you can import your SSH keys into Keychain using this command:

SSH keys into keychain using this command:

```
$ ssh-add -K yourkey
```