




[Home](#) [About](#)

[Login](#)

 SSH - Cheat Sheet - header image

SSH - Cheat Sheet

Last updated on July 14, 2022 - [0 comments](#)

Quick Links

[SSH-Keygen](#)

[SSH with Keys](#)

[Authorized_Keys](#)

[SCP](#)

[SSH-Agent](#)

[SSH Config](#)

[Git & Windows OpenSSH](#)

[Exit Dead SSH Sessions](#)

[Multiple GitHub Keypairs](#)

[SSH Agent Forwarding](#)

[SSH Agent Forwarding: Windows to WSL](#)

[SSH Tunnels](#)

[Password Managers & SSH Agents](#)

[Video](#)

[↑ Top](#)

A cheat sheet for popular SSH commands, key generation, SSH agents that I'm using a lot.

SSH-Keygen

Nowadays, most platforms recommend you to generate keys with the ed25519 algorithm.

```
ssh-keygen -t ed25519 -C "your@email.com"
```

If you prefer to go with RSA for compatibility reasons, use the following:

```
ssh-keygen -t rsa -b 4096 -C "your@email.com"
```

The `-C` file simply puts a comment on your public key, like below, so you can e.g. easily make out which public key belongs to which email address, in a busy [Authorized Keys](#) file.

```
ssh-ed25519 KLAJSDLKSAJKLSJD90182980p1+++ your@email.com
```

Note: When generating SSH keys, make sure to protect your private key with a passphrase.

SSH with Keys

To use a specific private key to connect to a server, use:

```
ssh -i mykeyfile user@remotehost.com
```

Instead of specifying your key files manually with `-i`, use an [SSH-Agent](#).

Authorized Keys

Any remote host or service, like GitHub, that you want to use your SSH keys with, needs the *public key* of your SSH keypair.

For servers, you simply need to append your public key to the file `~/.ssh/authorized_keys`.

Use one of the following commands to do that:

- `cat ~/.ssh/id_rsa.pub | ssh USER@HOST "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"`
- <https://askubuntu.com/a/262074>
- `ssh-copy-id user@host` - <https://askubuntu.com/a/46427>

With services like GitHub, AWS etc. you would either use the UI to upload your public key, or, if available, command-line tools.

SCP

To upload a file to a remote server:

```
scp myfile.txt user@dest:/path
```

To recursively upload a local folder to a remote server:

```
scp -rp sourcedirectory user@dest:/path
```

To download a file from a remote server:

```
scp user@dest:/path/myfile.txt localpath
```

To recursively download a local folder to a remote server:

```
scp -rp user@dest:/remotedir localpath
```

Hint: When doing things recursively via SCP, you might want to consider rsync, which also runs over SSH and has [a couple of advantages over SCP](#).

Hint 2: [SCP has been deprecated](#) and you should consider switching to (the less user-friendly) SFTP. [The scp command uses the SFTP protocol since OpenSSH 9](#).

SSH-Agent

With a running OpenSSH agent (automatically available out of the box on most Linux distributions and macOS) simply use:

```
ssh-add privatekeyfile
```

To enable the OpenSSH agent on Windows, you'll need to [execute the following commands](#):

```
# By default the ssh-agent service is disabled. Allow it to be manually started for the next
# Make sure you're running as an Administrator.
Get-Service ssh-agent | Set-Service -StartupType Automatic
```

```
# Start the service
Start-Service ssh-agent
```

Note: On Windows/Linux adding a key to your ssh-agent once, even with a password, will make sure that the key gets associated with your 'login'. Meaning: When you restart your PC and log in again, you'll have your identity automatically available again.

To get the same behavior on macOS, you'll need to follow these instructions [on StackExchange](#).

SSH Config

Create a file `~/.ssh/config` to manage your SSH hosts. Example:

```
Host dev-meta*
  User ec2-user
  IdentityFile ~/.ssh/johnsnow.pem
```

```
Host dev-meta-facebook
  Hostname 192.168.178.1
```

```
Host dev-meta-whatsapp
    Hostname 192.168.178.2
```

```
Host api.google.com
    User googleUser
    IdentityFile ~/.ssh/targaryen.key
```

Note:

The *Host* directive can either

- be a pattern (matching multiple follow-up *Hosts*)
- refer to a made-up hostname (*dev-facebook*)
- be a real hostname.

If it's a made-up hostname, you'll need to specify an additional *Hostname* directive, otherwise, you can leave it out. And to add to the overall confusion, a *Host* line can actually contain multiple patterns.

With the config file above, you could do a:

```
ssh dev-meta-facebook
```

Which would effectively do a *ssh -i ~/.ssh/johnsnow.pem ec2-user@192.168.178.1* for you.

For a full overview of all available options, look at [this article](#).

Git & Windows OpenSSH

To make Git use Windows's OpenSSH (and not the one it bundles), execute the following command:

```
git config --global core.sshcommand "C:/Windows/System32/OpenSSH/ssh.exe"
```

Exit Dead SSH Sessions

To kill an unresponsive SSH session, hit, subsequently.

Enter, ~, .

Multiple GitHub Keypairs

Trying to clone different private GitHub repositories, which have different SSH keypairs associated with them, doesn't work out of the box.

Add this to your *.ssh/config* (this example assumes you have two GitHub keypairs, one for your work account and one for your personal account)

```
Host github-work.com
  Hostname github.com
  IdentityFile ~/.ssh/id_work
```

```
Host github-personal.com
  Hostname github.com
  IdentityFile ~/.ssh/id_personal
```

Then instead of cloning from *github.com*.

```
git clone git@github.com:marcobehlerjetbrains/buildpipelines.git
```

Clone from either *github-work.com* or *github-personal.com*.

```
git clone git@github-work.com:marcobehlerjetbrains/buildpipelines.git
```

SSH Agent Forwarding

Ever wanted to use your local SSH keys on a remote server, without copying your keys to that server? For example to *git clone* a private repository via SSH on a remote server?

Agent forwarding to the rescue. Edit your local *.ssh/config* file like so:

```
Host yourremoteserver.com
  ForwardAgent yes
```

Then simply *ssh* to your server and execute an *_ssh-add -L*. The server's SSH agent should have all local SSH identities available and you can start cloning away!

SSH Agent Forwarding: Windows to WSL

If you want to use the Windows OpenSSH agent with all its identities from WSL, do the following:

1. Install *socat*, e.g. on your WSL Distribution: e.g. *apt install socat* for Ubuntu/Debian.
2. Download a build of [npiperelay](#) and put it somewhere on your (Windows) PATH.
3. Put the following into your WSL *~/.bash_profile* or *~/.bashrc*.

```
# Configure ssh forwarding
export SSH_AUTH_SOCKET=$HOME/.ssh/agent.sock
# need `ps -ww` to get non-truncated command for matching
# use square brackets to generate a regex match for the process we want but that doesn't match
ALREADY_RUNNING=$(ps -auxww | grep -q "[n]piperelay.exe -ei -s //./pipe/openssh-ssh-agent"; echo $?)
if [[ $ALREADY_RUNNING != "0" ]]; then
  if [[ -S $SSH_AUTH_SOCKET ]]; then
    # not expecting the socket to exist as the forwarding command isn't running (http://w
    echo "removing previous socket..."
```

```
rm $SSH_AUTH_SOCKET
fi
echo "Starting SSH-Agent relay..."
# setsid to force new session to keep running
# set socat to listen on $SSH_AUTH_SOCKET and forward to npiperelay which then forwards to
(setsid socat UNIX-LISTEN:$SSH_AUTH_SOCKET,fork EXEC:"npiperelay.exe -ei -s ../../pipe/openss
fi
```

Enjoy!

Major thanks to Stuart Leeks, who I blatantly stole this code from - he did all the work @ <https://stuartleeks.com/posts/wsl-ssh-key-forward-to-windows/>.

Check out his [WSL Book](#) for more such tricks!

SSH Tunnels

Want to connect to a server that is hidden from the outside world, but accessible from a box you have SSH access to? Like an Amazon RDS database, which is only reachable from inside an AWS network?

Use SSH forwarding

```
ssh username@jumphost -N -f -L localport:targethost:targetport
```

The following command establishes an SSH tunnel between my *local machine (@port 3307)* and an *RDS database (@port 3306)*, via an *EC2 jump host (18.11.11.11)*.

```
ssh ec2-user@18.11.11.11 -N -f -L 3307:marcotestme.12345.eu-central-1.rds.amazonaws.com:3306
```

You could now, for example, use the mysql client to connect to *localhost:3307*, which will be transparently tunneled to RDS for you.

```
mysql -h localhost -P 3307
```

Note: A lot of tools/IDEs like [IntelliJ IDEA](#), support opening up SSH tunnels by just clicking a checkbox in the UI.

Password Managers & SSH Agents

Password Managers like [1Password](#) or [Keepass](#) can not only store your SSH keys, but they also come with their own *ssh-agent*, replacing your system's ssh-agent.

This means, whenever you unlock your password manager on any machine that you have it installed on, you'll have all your SSH identities instantly available.

Super useful!

Video

If you prefer video, you can see the guide in action on YouTube.

There's more where that came from

I'll send you an update when I publish new guides. Absolutely no spam, ever. Unsubscribe anytime.

[I want more!](#)

Share



Comments

This domain is not registered with Commento.

