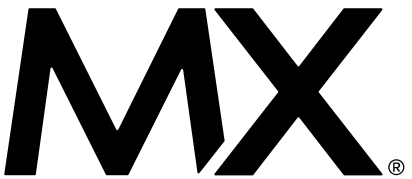
SSH

Intro & Tricks

A brownbag presentation at



by Seth House @whiteinge

Basics

`ssh` – secure shell

Replacement for telnet and other plain-text protocols.

`ssh` – secure shell

Replacement for telnet and other plain-text protocols.

Year	Release
1995	Released as freeware; then made proprietary
1999	OpenSSH forked
2006	Adopted as standard by IETF (Version 2)

A suite of CLI utilities and daemons

```
SSH(1)                BSD General Commands Manual               SSH(1)

[...snip...]

SEE ALSO
     scp(1), sftp(1), ssh-add(1), ssh-agent(1), ssh-keygen(1),
     ssh-keyscan(1), tun(4), ssh_config(5), ssh-keysign(8), sshd(8)
```

https://en.wikipedia.org/wiki/Secure_Shell#Uses

Uses [edit]

SSH is a protocol that can be used for many applications across many platforms including most Unix variants (Linux, the BSDs including Apple's macOS, and Solaris), as well as 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 is possible, but presently only with the OpenSSH server and client implementation.

- For login to a shell on a remote host (replacing Telnet and rlogin)
- For executing a single command on a remote host (replacing rsh)
- For setting up automatic (passwordless) login to a remote server (for example, using OpenSSH[27])
- In combination with rsync to back up, copy and mirror files efficiently and securely
- For forwarding a port
- For tunneling (not to be confused with a VPN, which routes packets between different networks, or bridges two broadcast domains into one).
- For using as a full-fledged encrypted VPN. Note that only OpenSSH server and client supports this feature.
- For forwarding X from a remote host (possible through multiple intermediate hosts)
- For browsing the web through an encrypted proxy connection with SSH clients that support the SOCKS protocol.
- For securely mounting a directory on a remote server as a filesystem on a local computer using 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 [edit]

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

- Secure copy (SCP), which evolved from RCP protocol over SSH
- rsync, intended to be more efficient than SCP. Generally runs over an SSH connection.
- SSH File Transfer Protocol (SFTP), a secure alternative to FTP (not to be confused with FTP over SSH or FTPS)
- Files transferred over shell protocol (a.k.a. FISH), released in 1998, which evolved from Unix shell commands over SSH
- Fast and Secure Protocol (FASP), aka *Aspera*, uses SSH for control and UDP ports for data transfer.

Basic usage

```
ssh <hostname>
ssh <user>@<hostname>
ssh <user>@<hostname> -p <port>
```

Basic usage

```
ssh <hostname>
ssh <user>@<hostname>
ssh <user>@<hostname> -p <port>
```

```
ssh <alias>
```

Password authentication

```
ssh <host>
```

Public/private keypairs

```
ssh-keygen
```

Public/private keypairs

```
ssh-keygen
```

Ok to share `.pub` file:

- `~/.ssh/authorized_keys`
- https://github.com/whiteinge.keys

Public/private keypairs

```
ssh-keygen
```

Ok to share `.pub` file:

- `~/.ssh/authorized_keys`
- https://github.com/whiteinge.keys

Ok to have many identities.

```
% tree ~/.ssh
|-- id_rsa-personal
|-- id_rsa-personal.pub
|-- id_rsa-work
|-- id_rsa-work.pub
|-- id_rsa-misc
`-- id_rsa-misc.pub
```

Public/private keypairs

```
ssh-keygen
```

Ok to share `.pub` file:

- `~/.ssh/authorized_keys`
- https://github.com/whiteinge.keys

Ok to have many identities.

```
% tree ~/.ssh
|-- id_rsa-personal
|-- id_rsa-personal.pub
|-- id_rsa-work
|-- id_rsa-work.pub
|-- id_rsa-misc
`-- id_rsa-misc.pub
```

Ok to have device-specific identities (work laptop, home desktop, phone, etc).

Public/private keypairs

```
ssh-keygen
```

Ok to share `.pub` file:

- `~/.ssh/authorized_keys`
- https://github.com/whiteinge.keys

Ok to have many identities.

```
% tree ~/.ssh
|-- id_rsa-personal
|-- id_rsa-personal.pub
|-- id_rsa-work
|-- id_rsa-work.pub
|-- id_rsa-misc
`-- id_rsa-misc.pub
```

Ok to have device-specific identities (work laptop, home desktop, phone, etc).

Not ok to omit a passphrase!

Certificates

Managing individual user keys is awful at scale.

Key formats and exchange algorithms

Client and server negotiate acceptable mechanisms.

```
ssh -v <host>
```

Run a command

```
ssh <host> 'whoami'
ssh <host> 'hostname; whoami; uptime'
```

Run a command

```
ssh <host> 'whoami'
ssh <host> 'hostname; whoami; uptime'
```

```
printf '%s\n' server-{1..24} | xargs -P6 -I{} ssh {} \
    'echo $(hostname) $(cat /srv/app/current/REVISION)'
```

Transfer a file

```
scp <src> <dest>
scp -r <src> <dest>
```

Transfer a file

```
scp <src> <dest>
scp -r <src> <dest>
```

```
scp somefile user@host:/path/to/dest
scp somefile user@host:~
scp somefile user@host:~/dest

scp user@host:/path/to/src .
scp user@host:/path/to/src ./somedir/dest
scp user@host:/path/to/src ~/dest
```

Transfer a file

```
scp <src> <dest>
scp -r <src> <dest>
```

```
scp somefile user@host:/path/to/dest
scp somefile user@host:~
scp somefile user@host:~/dest

scp user@host:/path/to/src .
scp user@host:/path/to/src ./somedir/dest
scp user@host:/path/to/src ~/dest
```

```
scp -P <port> somefile user@host:~
```

(Capital `-P` !)

sftp & rsync

Drop-in replacement for an FTP server.

sftp & rsync

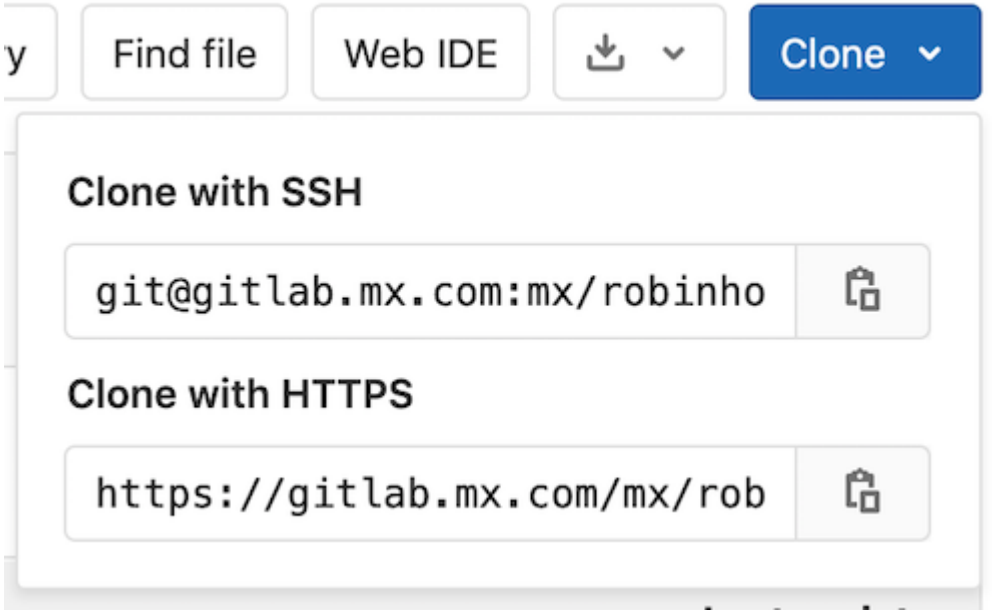
Drop-in replacement for an FTP server.

Works out-of-box with rsync.

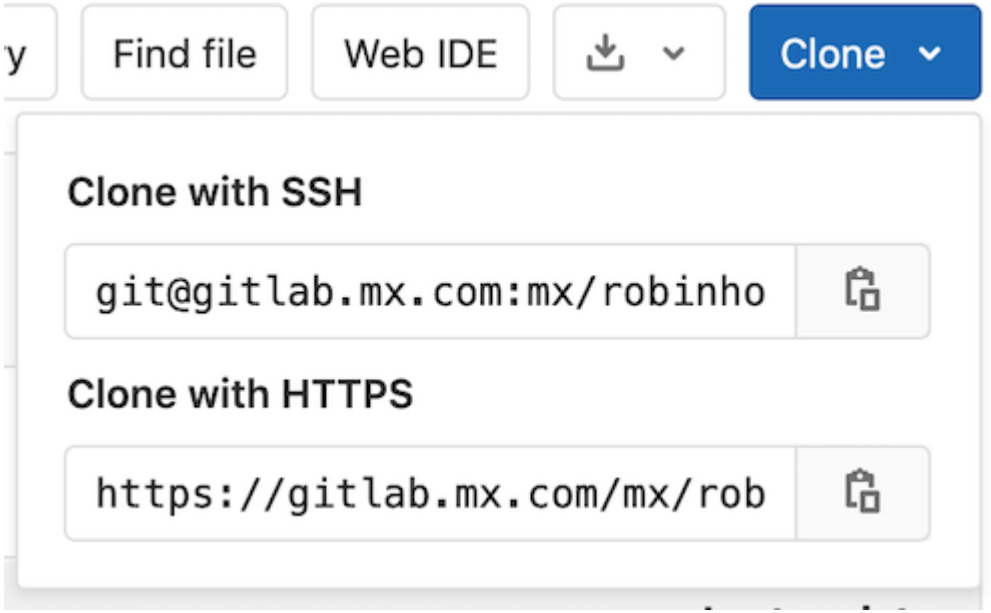
```
rsync -avhz --progress --append-verify \
    ~/some/src \
    user@host:~/some/dest
```

(rsync is magic.)

Git


y
Find file
Web IDE
Clone
Clone with SSH
git@gitlab.mx.com:mx/robinho
Clone with HTTPS
https://gitlab.mx.com/mx/rob

Git




```
GIT_SSH_COMMAND='ssh -v' git fetch
```

Configuration

Configuration file

```
~/.ssh/config
```

Configuration file

```
~/.ssh/config
```

(May need to create.)

```
mkdir ~/.ssh
chmod 700 ~/.ssh

# or

chmod u+rwX,g-rwx,o-rwx ~/.ssh
```

Host, Hostname, & aliases

```
Host 192.168.0.10
```

Host, Hostname, & aliases

```
Host 192.168.0.10
```

```
Host raspberrypi
    Hostname 192.168.0.10
```

Host, Hostname, & aliases

```
Host 192.168.0.10
```

```
Host raspberrypi
    Hostname 192.168.0.10
```

```
Host 192.168.0.10 raspberrypi dlnaserver frank
    Hostname 192.168.0.10
```

User & Port

```
Host raspberrypi
    Hostname 192.168.0.10
    User pi
    Port 2222
```

Host Patterns

```
Host east-web-* east-db-*
    User user-east

Host west-web-* west-db-*
    User user-west
```

Host Patterns

```
Host east-web-* east-db-*
    User user-east

Host west-web-* west-db-*
    User user-west
```

```
Host *
    <global settings>
```

Specify an identity

(Or just let ssh try available options.)

```
Host east-web-* east-db-*
    IdentityFile ~/.ssh/id-prod-rsa

Host sand-web-*
    IdentityFile ~/.ssh/id-sand-rsa
```

Known hosts

`~/.ssh/known_hosts`

Known hosts

`~/.ssh/known_hosts`

(Caveat emptor.)

```
HashKnownHosts no
StrictHostKeyChecking no
UserKnownHostsFile /dev/null
```

Known hosts

`~/.ssh/known_hosts`

(Caveat emptor.)

```
HashKnownHosts no
StrictHostKeyChecking no
UserKnownHostsFile /dev/null
```

```
Host virtualbox 127.0.0.1
    Hostname 127.0.0.1
    Port 2222
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
```

Agent

- A long-running processes (usually started on login by your OS).
- Holds unlocked private keys in memory.

Agent

- A long-running processes (usually started on login by your OS).
- Holds unlocked private keys in memory.

Start manually:

```
ssh-agent tmux
```

Agent

- A long-running processes (usually started on login by your OS).
- Holds unlocked private keys in memory.

Start manually:

```
ssh-agent tmux
```

Add identities:

```
ssh-add
ssh-add ~/.ssh/some-key-rsa
```

Agent

- A long-running processes (usually started on login by your OS).
- Holds unlocked private keys in memory.

Start manually:

```
ssh-agent tmux
```

Add identities:

```
ssh-add
ssh-add ~/.ssh/some-key-rsa
```

List loaded identities:

```
ssh-add -l
```

Agent

- A long-running processes (usually started on login by your OS).
- Holds unlocked private keys in memory.

Start manually:

```
ssh-agent tmux
```

Add identities:

```
ssh-add
ssh-add ~/.ssh/some-key-rsa
```

List loaded identities:

```
ssh-add -l
```

Remove identities:

```
ssh-add -d ~/.ssh/some-key-rsa
ssh-add -D
```

Automatically add identities on use

```
AddKeysToAgent yes
```

(Don't be shy to clear identities from your agent frequently.)

Allow remote hosts to use identities in your local agent

```
Host prod-web-*
    ForwardAgent yes
```

(Convenient but dangerous if the remote host is compromised.)

Pipelining

- Share a single SSH connection with other terminals.

Pipelining

- Share a single SSH connection with other terminals.

```
Host *
    ControlMaster auto
    ControlPersist 1m
    ControlPath ~/.ssh/master-%r@%h:%p
```

Pipelining

- Share a single SSH connection with other terminals.

```
Host *
    ControlMaster auto
    ControlPersist 1m
    ControlPath ~/.ssh/master-%r@%h:%p
```

Send `check`, `exit`, or `stop` signals to the socket:

```
ssh -S ~/.ssh/master-user@host:port -O exit ''
```

sshd (server) configuration

`/etc/ssh/sshd_config`:

```
PasswordAuthentication no
PermitRootLogin no
AuthorizedKeysFile      .ssh/authorized_keys
AllowUsers alice bob
```

Tricks

SOCKS5 proxy

```
Host myvps.example.com
    DynamicForward 8009
```

Port forwarding

- View remote web UIs locally.
- Access remote services as though they were local.

```
Host myvps.example.com
    LocalForward 3000 localhost:3000
    LocalForward 3001 localhost:3001
```

Through a bastion host

(SSH through one machine to access another behind it.)

```
Host inaccessible-host
    Hostname inaccessible-host
    User someuser-on-inaccessible-host
    ServerAliveInterval 30
    ProxyCommand ssh accessible-host -W %h:%p
```

Run in the background

Flag	Description
`-f`	Requests ssh to go to background just before command execution.
`-N`	Do not execute a remote command. Useful for just forwarding ports.

Reverse tunnel

```
a=`ps -ef | grep 19999 | grep -v grep`
if [ ! "$a" ]; then
    ssh -fN -R 19999:localhost:<remoteport> <remoteuse>@<remotehost>
fi
```

```
 * * * * * /path/to/your/script.sh
```

Read from stdin or write to stdout

```
cat somefile | ssh user@host 'cat > somefile-copy'

ssh user@host 'cat somefile' > somefile-copy

dd if=/dev/sda2 | ssh user@host 'cat > backup.img'
```

Use tar without intermediary files

- Great for transfering a directory of files while keeping ownership, permissions, and other attributes intact.

```
tar -C /path/to -cf mydir | ssh user@host 'cat > mydir.tar'
tar -C /path/to -cf mydir | ssh user@host 'tar -C /path/to -xf -'
```

sshfs

- FUSE (filesystem in user space)

```
sshfs user@host:~/path/to/remote/dir /path/to/local/dir
```

LocalCommand

```
Host prod-east-datacenter:
    PermitLocalCommand yes
    LocalCommand printf '\e[1;31mProduction! Be careful.\n\e[0;m'
```