sshx

Release 0.33.5

WqyJh

Jan 18, 2021

CONTENTS:

sshx is a lightweight wrapper for ssh/scp command, which has the following features:

- Remember your ssh accounts safely.
- Connect to your account without typing password.
- Set jump hosts for your connection.
- Create ssh port forwardings without typing password.
- Create socks5 proxy by ssh dynamic port forwarding.
- Enable jump hosts for your port forwardings.
- Copy files from/to your account without typing password.
- Enable jump hosts for your scp connection.
- Execute remote command without typing password.
- Enable jump hosts for executing command.
- Install ssh public keys to remote server.
- Enable jump hosts for public key installation.

CHAPTER

ONE

INSTALLATION

sshx is on PyPI, and can be installed with pip:

```
pip install sshx
```

1.1 Supported platforms

- Linux
- macOS
- WSL/cygwin/msys2 on Windows
- Termux on Android

Attention:

- Windows CMD/PowerShell support was removed.
- Python 2 support was removed.

1.2 Requirements

- Python >= 3.5
- openssh-clients: `ssh`, `scp`, `ssh-keygen`

1.3 Install on Android

First install latest Termux app (tested version 0.92) on your android device.

Then install some requirements on termux shell.

```
pkg update
pkg in python openssh
pip install sshx
```

CHAPTER

TWO

QUICK START

2.1 Install

```
sudo pip3 install sshx
```

2.2 Initialization

Perform only once after you've installed sshx.

```
sshx init
```

2.3 Adding an account

```
sshx add myhost -l test@192.168.9.155
```

This command will ask you to type your password and sshx would store the encrypted password.

2.4 Connect to the account

```
sshx connect myhost
```

CHAPTER

THREE

INITIALIZATION

`sshx init` performs initialization.

```
$ sshx init
$ tree ~/.sshx
~/.sshx
└── .accounts
```

It will create the config file `~/.sshx/.accounts` which stores the accounts info.

If the config files are damaged you'll probably lost all the accounts added, so **DON'T TOUCH IT**.

3.1 Customize config location

You can set environment variable `SSHX_HOME` to customize the location of config files, the default value is `~/.sshx`.

3.2 Config file format

The passwords of accounts and passphrases of identities are stored encrypted.

The `phrase` field is a string randomly generated during initialization to encrypt and decrypt the passwords of accounts and the passphrase of identities.

It's **very important** to keep the config file save, otherwise your passwords may leak.

Want more secure, see *Security Option*

```
{
    "security": false,
    "phrase": "acnhPz21bUfx1PE",
    "accounts": [
        {
            "name": "host1",
            "user": "root",
            "host": "172.17.0.2",
            "port": "22",
            "password": "InJvb3Qi.gM0vblkle9Utv5NrW6Q_WzZEgSg",
            "identity": "",
            "passphrase": "",
            "via": ""
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "name": "host2",
            "user": "root",
            "host": "192.168.0.6",
            "port": "22",
            "password": "InJvb3Qi.gM0vblkle9Utv5NrW6Q_WzZEgSg",
            "identity": "",
            "passphrase": "",
            "via": ""
        }
    ]
}
```

3.3 Force initialization (Dangerous)

Delete the previous config files and perform initialization.

```
sshx init --force
```

If it's previous inited, `sshx init` would failed which protect the config file from being damaged.

Only if you're sure you don't need the existing accounts anymore, you could add this option.

`--force` option can be used with `--security` option.

3.4 Security Option

Enable security option during initialization:

```
sshx init --security
```

This option will ask you to set a `phrase` manually for encryption and decryption. The `phrase` won't be stored in the config file, instead the SHA1 hash of it would be stored for verification.

If the security option was enabled, you would be asked to input the `phrase` from prompt every you execute a command of sshx that need to access the encrypted data.

This option could be changed after initialization.

Enable security option. This will ask you to set a `phrase` and re-encrypt all sensitive data.

```
sshx config --security-on
```

Disable security option. This will ask `phrase` from prompt for verification, generate a new random phrase and re-encrypt all sensitive data.

```
sshx config --security-off
```

Change `phrase`: If security option was disabled, re-generate a random phrase, otherwise it would ask user to input the current `phrase`, verify it and ask user to set a new `phrase`.

```
sshx config --chphrase
```

CHAPTER

FOUR

ADD ACCOUNTS

`sshx add` adds an account.

Usage:

```
Usage: sshx add [OPTIONS] NAME

Add an account and assign a name for it.

Options:
-l TEXT                <user>@<host>[:port]
-H, --host TEXT
-P, --port TEXT
-u, --user TEXT
-p, --password
-i, --identity TEXT  SSH identity file.
-v, --via TEXT       Account name of jump host.
--help               Show this message and exit.
```

Add an account and specify an password for authentication.

```
sshx add myhost -H host -P port -u user -p
```

Add an account in an simple way.

```
sshx add myhost -l user@host:port
```

Add an account and specify an identity file for authentication. This may ask you to input the passphrase from prompt if the identity file has one.

```
sshx add myhost -H host -P port -u user -i identity_file
```

Add an account and specify both password and identity file for authentication. In this situation, only identity file would be used for authentication.

```
sshx add myhost -H host -P port -u user -p -i identity_file
```

4.1 Jump hosts

Jump hosts are intermediate hosts for establishing SSH connections.

Assume you have an server A in an internal network, and you cannot access it directly, but you have an server B, which you can access directly and B can access A, then you can connect server A via server B, while the server B is a jump host.

Add an account and specify an jump host for it.

```
sshx add -l user@host:port -v myhost myhost2
```

After the account with jump host was added, you can connect it by `sshx connect myhost2`, an ssh connection to `myhost2` would be established via the jump host `myhost`.

You can specify multiple jump hosts for an single account, which are seperated by comma characters.

```
sshx add -l user@host:port -v myhost1,myhost2,myhost3 myhost4
```

Jump hosts would be visited sequentially. For example, connect to `myhost1`, then connect to `myhost2` by `myhost1`, then connect to `myhost3` by `myhost2`, finally connect to `myhost4` by `myhost3`.

The jump hosts would be translated to `-J`, `ProxyJump` or `ProxyCommand` options of `ssh` command.

CHAPTER

FIVE

SHOW ACCOUNTS

5.1 List accounts

`sshx list` lists all the accounts in the following format.

```
name                host                        user                via
-----               -----                       -----               -----
host1               192.168.7.1                 root
host2               192.168.7.2                 test                host1
host3               192.168.7.3                 root                host2
```

The outputs can be sorted and reversed.

Usage:

```
Usage: sshx list [OPTIONS]

List all accounts.

Options:
--sort [name|host|user]  Sort by keys.
--reverse
--help                   Show this message and exit.
```

5.2 Show account detail

`sshx show` show details for a specified account.

Show account info (without encrypted data).

```
sshx show host1
```

Show account info with data decrypted. (Need to input `phrase` if *Security Option* was enabled.)

```
sshx show host2 -p
```

CHAPTER

SIX

DELETE ACCOUNTS

`sshx del` deletes an account.

```
sshx del host1
```

CHAPTER

SEVEN

UPDATE ACCOUNTS

`sshx update` updates an account. The supported options are the same with `add` command, all the specified fields will be updated.

Change the host1's `host` field to `domain.com`.

```
sshx update host1 -H domain.com
```

Change the host1's `password`. This would ask password from prompt.

```
sshx update host1 -p
```

Set/Change the host1's identity to `identity2`. This would ask passphrase from prompt if it had.

```
sshx update host1 -i identity2
```

Rename host1 to host2.

```
sshx update host1 -n host2
```

Unset the host1's identity. This would ask user to set and password if it's never been set.

```
sshx update host1 -i ''
```

CHAPTER

EIGHT

CONNECT ACCOUNTS

`sshx connect` connect to an account.

Connect host1 directly.

```
sshx connect host1
```

Connect host1 using host2 as jump host. If the host1 already had an jump host, this would temporarily override it.

```
sshx connect host1 -v host2
```

Connect to host1 using host2 as jump host, while the host2 is using host3 as jump host. This is also a temporary jump host, which won't affect the config files.

```
sshx connect host1 -v host2,host3
```

If the host3 was already had host2 as its jump host, then the following command is equivalent to the above one.

```
sshx connect host1 -v host3
```

You can specify extra arguments for ssh, which would be added into the ssh command line.

For example, add verbose option.

```
sshx connect host1 -e '-v'
```

8.1 X11Forwarding

The ssh server must have the following config on `/etc/ssh/sshd_config`.

```
X11Forwarding yes
X11UseLocalhost no
```

Connect to host1 and enable `X11Forwarding`.

```
sshx connect host1 -e '-X'
```

Then you can execute graphical program like `gedit` on the shell, and the window would show on your local screen.

CHAPTER

NINE

CREATE SOCKS5 PROXIES

`sshx socks` creates socks5 proxies by ssh's `DynamicForward` option. When you connect to an server by ssh with this option, the ssh client would establish an socks5 server locally, just like the `shadowsocks` client, while the connected ssh server would perform the requests for you, just like the `shadowsocks` server.

Why create socks5 proxies with ssh? Because it's very simple and safe.

- Simple: no need for extra softwares, easy to config and use.
- Safe: all traffic would be carried and encrypted by ssh, safer than `shadowsocks`.

Create an socks5 proxy listening on `127.0.0.1:1080`.

```
sshx socks host1
```

Customize the listening address with `--bind` option.

```
sshx socks host1 --bind 0.0.0.0:1081
```

Create socks proxy with jump hosts.

```
sshx socks host1 -v host2,host3
```

Run in background with `-b` option.

```
sshx socks host1 -b
```

9.1 Configuration

The ssh server must enable `AllowTcpForwarding` option which is enabled by default. Therefore, no need to set it manually.

If you could connect to the server but cannot establish an socks5 proxy, then consider to enable this option in `/etc/ssh/sshd_config` on the server.

CHAPTER

TEN

CREATE PORT FORWARDINGS

`sshx forward` creates port fowardings.

```
Usage: sshx forward [OPTIONS] NAME

SSH port forward via specified account.

Options:
-v, --via TEXT            Account name of jump host.
-L, -f, --forward TEXT    [bind_address]:<bind_port>:<remote_address>:<remot
                            e_port> => Forward local bind_address:bind_port to
                            remote_address:remote_port.
-R, -rf, --rforward TEXT  <bind_address>:<bind_port>:<local_address>:<local_
                            port> => Forward remote bind_address:bind_port to
                            local local_address:local_port.
-b, --background          Run in background.
--help                    Show this message and exit.
```

Forward `localhost:80` to `192.168.77.7:80`, while the host1 is the intermedia server, so you must ensure the host1 could connect to `192.168.77.7:80`.

```
sshx forward host1 -L :80:192.168.77.7:80
```

Forward `host1:8000` to `192.168.99.9:8000`. When you access `localhost:8000` on host1, the connection would be forward to `192.168.99.9:8000`, while your computer is working as a intermediate server, so you have to ensure your computer has access to `192.168.99.9:8000`.

```
sshx forward host1 -R :8000:192.168.99.9:8000
```

Specify multiple forwards. The following command sets two local forwards and one remote forward.

```
sshx forward host1 -L :80:192.168.77.7:80 :8080:192.168.77.7:8080 -R :8000:192.168.99.
↪9:8000
```

Forward can also be used with jump hosts.

```
sshx forward host1 -v host2 -L :80:192.168.77.7:80
```

CHAPTER

ELEVEN

COPY FILES

`sshx scp` copy files to/from servers.

Copy local directory `/tmp/dir` to host1's `/tmp`.

```
sshx scp /tmp/src host1:/tmp
```

Copy remote files from host1 to local.

```
sshx scp host1:/tmp/src /tmp
```

Copy local files to host1 with host2 as jump host.

```
sshx scp /tmp/src host1:/tmp -v host2
```

Copy remote files to local, using host2 as jump host and using host3 as host2's jump host.

```
sshx scp host1:/tmp/src /tmp -v host2,host3
```

CHAPTER TWELVE

EXECUTE COMMAND

`sshx exec` execute an remote command. The arguments after `--` is the command line to be executed remotely.

Execute `ls -al` on host1.

```
sshx exec host1 -- ls -al
```

Execute an command with tty.

```
sshx exec host1 --tty -- /bin/bash
```

Execute an command on host1 via host2.

```
sshx exec host1 -v host2 -- ls -al
```

INSTALL PUBLIC KEYS

`sshx copyid` installs public keys (`*.pub`) to remote hosts.

Assume you have an account host1 using password for authentication. Now you want to change it to use public key `id_rsa.pub` for authentication. You can achieve this by two steps.

Step 1: copy public key to the host1.

```
sshx copyid id_rsa.pub host1
```

Step 2: set the corresponding identity to the account.

```
sshx update host1 -i id_rsa
```

You can also install public keys via jump hosts.

```
sshx copyid id_rsa.pub host1 -v host2
```

CHAPTER FOURTEEN

GLOBAL OPTIONS

```
Usage: run.py [OPTIONS] COMMAND [ARGS]...

Options:
--version                       Show the version and exit.
-d, --debug
--interval INTEGER RANGE        ServerAliveInterval for ssh_config.
--countmax INTEGER RANGE        ServerAliveCountMax for ssh_config.
--forever                       Keep ssh connection forever.
--retry RETRY                   Reconnect after connection closed, repeat
                                for retry times. Supported values are
                                "always" or non negative integer. If retry
                                was enabled, --interval must be greater than
                                0.
--retry-interval INTEGER RANGE  Sleep seconds before every retry.
--help                          Show this message and exit.
```

Global options must be specified before sub-commands.

The `--retry` and `--retry-interval` options can only be used for `connect`, `forward`, `socks` and `exec` commands.

Create a socks5 proxy and always reconnect immediately when the connection was closed.

```
sshx --interval 1 --countmax 1 --retry always socks host1
```

Create a socks5 proxy and always reconnect after 5s when the connection was closed.

```
sshx --interval 1 --countmax 1 --retry always --retry-interval 5 socks host1
```

Create a socks5 proxy and reconnect for 5 times when the connection was close.

```
sshx --interval 1 --countmax 1 --retry 5 socks host1
```

Create a ssh connection and set the `ServerAlive` options. The following options make the ssh client sends a keepalive probe to server after no data was transfered for 30s and after probing for 60 times the connection would be closed (idle for 1800s).

```
sshx --interval 30 --countmax 60 connect host1
```

The `--forever` option is an alias for `--interval 60 --countmax 52560000`, which means the ssh connection would be closed after idle for 100 years (long enough :). You can also set a value longer than `--forever`.

Note: The `forever` option is now a default option, which improves user experience.

CHAPTER

FIFTEEN

DEVELOPMENT

Install requirements.

```
pipenv install --dev
```

Then activate the virtual environment created by pipenv.

```
pipenv shell
```

Run unittests.

```
pytest
```

Build packages.

```
python setup.py build bdist_wheel
```

Because of the directory structure defined by `setuptools`, the `sshx.py` cannot be run directly. Generally, we have to run `python setup.py install` before we run `sshx`, which is really inefficient. To solve the problem, I've created an `run.py` script.

```
./run.py --help
```

Commit messages must match the Conventional Commits.

15.1 Releasing (Ignore it)

It is the maintainer's job to release a new version. Therefore this section is wrote for me.

Before release a new version, maintainer must fully test the code to be released (which usually is the latest master branch) on a production environment. The recommended way is to build an `bdist_wheel` and install it on a new docker environment, then test all of the functions.

Version number should follow Semantic Versioning. Currently, I use bumping to calculate the semver from git commit messages.

Step 1: Calculate the currently version, the output is the version value like `0.27.1`.

```
bumping
```

Step 2: Bump to new version and commit.

```
./bump_version.sh <version>
```

Step 3: Generate changelog and commit. Push the commits and wait for ci to be passed.

```
auto-changelog --latest-version <version>
git add -A
git commit -m "docs: udpate CHANGELOG.md"
git push origin master
```

Step 4: Tag for new version.

```
git tag <version>
```

Step 5: push the version tag and the travis-ci would build sshx with tags and upload it to PyPI.

```
git push origin --tags release-<version>
```

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search