



SSH教程

——by Wangdoc

目录

1. SSH 基本知识
2. SSH 客户端
3. SSH 密钥登录
4. SSH 服务器
5. 日志
6. 端口转发
7. SSH 证书登录
8. scp 命令
9. rsync 命令
10. sftp 命令
11. Fail2ban 教程

SSH 基本知识

SSH (Secure Shell 的缩写) 是一种网络协议，用于加密两台计算机之间的通信，并且支持各种身份验证机制。

实务中，它主要用于保证远程登录和远程通信的安全，任何网络服务都可以用这个协议来加密。

目录 [隐藏]

- [1. SSH 是什么](#)
- [2. 历史](#)
- [3. SSH 架构](#)

1. SSH 是什么

历史上，网络主机之间的通信是不加密的，属于明文通信。这使得通信很不安全，一个典型的例子就是服务器登录。登录远程服务器的时候，需要将用户输入的密码传给服务器，如果这个过程是明文通信，就意味着传递过程中，线路经过的中间计算机都能看到密码，这是很可怕的。

SSH 就是为了解决这个问题而诞生的，它能够加密计算机之间的通信，保证不被窃听或篡改。它还能对操作者进行认证 (authentication) 和授权 (authorization)。明文的网络协议可以套用在它里面，从而实现加密。

2. 历史

1995年，芬兰赫尔辛基工业大学的研究员 Tatu Ylönen 设计了 SSH 协议的第一个版本（现称为 SSH 1），同时写出了第一个实现（称为 SSH1）。

当时，他所在的大学网络一直发生密码嗅探攻击，他不得不为服务器设计一个更安全的登录方式。写完以后，他就把这个工具公开了，允许其他人免费使用。

SSH 可以替换 rlogin、TELNET、FTP 和 rsh 这些不安全的协议，所以大受欢迎，用户快速增长，1995年底已经发展到五十个国家的20,000个用户。SSH 1 协议也变成 IETF 的标准文档。

1995年12月，由于客服需求越来越大，Tatu Ylönen 就成立了一家公司 SCS，专门销售和开发 SSH。这个软件的后续版本，逐渐从免费软件变成了专有的商业软件。

SSH 1 协议存在一些安全漏洞，所以1996年又提出了 SSH 2 协议（或者称为 SSH 2.0）。这个协议与1.0版不兼容，在1997年进行了标准化，1998年推出了软件实现 SSH2。但是，官方的 SSH2 软件是一个专有软件，不能免费使用，而且 SSH1 的有些功能也没有提供。

1999年，OpenBSD 的开发人员决定写一个 SSH 2 协议的开源实现，这就是 OpenSSH 项目。该项目最初是基于 SSH 1.2.12 版本，那是当时 SSH1 最后一个开源版本。但是，OpenSSH 很快就完全摆脱了原始的官方代码，在许多开发者的参与下，按照自己的路线发展。OpenSSH 随 OpenBSD 2.6 版本一起提供，以后又移植到其他操作系统，成为最流行的 SSH 实现。目前，Linux 的所有发行版几乎都自带 OpenSSH。

现在，SSH-2 有多种实现，既有免费的，也有收费的。本书的内容主要是针对 OpenSSH。

3. SSH 架构

SSH 的软件架构是服务器-客户端模式（Server - Client）。在这个架构中，SSH 软件分成两个部分：向服务器发出请求的部分，称为客户端（client），OpenSSH 的实现为 ssh；接收客户端发出的请求的部分，称为服务器（server），OpenSSH 的实现为 sshd。

本教程约定，大写的 SSH 表示协议，小写的 ssh 表示客户端软件。

另外，OpenSSH 还提供一些辅助工具软件（比如 ssh-keygen、ssh-agent）和专门的客户端工具（比如 scp 和 sftp），这个教程也会予以介绍。

SSH 客户端

目录 [隐藏]

1. 简介
2. 基本用法
3. 连接流程
4. 服务器密钥变更
5. 执行远程命令
6. 加密参数
7. ssh 命令行配置项
8. 客户端配置文件
 - 8.1 位置
 - 8.2 主机设置
 - 8.3 配置命令的语法
 - 8.4 主要配置命令

1. 简介

OpenSSH 的客户端是二进制程序 ssh。它在 Linux/Unix 系统的位置是 `/usr/local/bin/ssh`。

Linux 系统一般都自带 ssh，如果没有就需要安装。

```
# Ubuntu 和 Debian
$ sudo apt install openssh-client

# CentOS 和 Fedora
$ sudo dnf install openssh-clients
```

安装以后，可以使用 `-V` 参数输出版本号，查看一下是否安装成功。

```
$ ssh -V
```

2. 基本用法

ssh 最常见的用途就是登录服务器，这要求服务器安装并正在运行 SSH 服务器软件。

ssh 登录服务器的命令如下。

```
$ ssh hostname
```

上面命令中，`hostname` 是主机名，它可以是域名，也可能是 IP 地址或局域网内部的主机名。不指定用户名的情况下，将使用客户端的当前用户名，作为远程服务器的登录用户名。如果要指定用户名，可以采用下面的语法。

```
$ ssh user@hostname
```

上面的命令中，用户名和主机名写在一起了，之间使用 `@` 分隔。

用户名也可以使用 `ssh` 的 `-l` 参数指定，这样的话，用户名和主机名就不用写在一起了。

```
$ ssh -l username host
```

ssh 默认连接服务器的22端口，`-p` 参数可以指定其他端口。

```
$ ssh -p 8821 foo.com
```

上面命令连接服务器 `foo.com` 的8821端口。

3. 连接流程

ssh 连接远程服务器后，首先有一个验证过程，验证远程服务器是否为陌生地址。

如果是第一次连接某一台服务器，命令行会显示一段文字，表示不认识这台机器，提醒用户确认是否需要连接。

```
The authenticity of host 'foo.com (192.168.121.111)' can't be
established.
ECDSA key fingerprint is
SHA256:Vybt22mVXuNuB5unE++yowF7lgA/9/2bLSi03qmYWBY.
Are you sure you want to continue connecting (yes/no)?
```

上面这段文字告诉用户，`foo.com` 这台服务器的指纹是陌生的，让用户选择是否要继续连接（输入 yes 或 no）。

所谓“服务器指纹”，指的是 SSH 服务器公钥的哈希值。每台 SSH 服务器都有唯一一对密钥，用于跟客户端通信，其中公钥的哈希值就可以用来识别服务器。

下面的命令可以查看某个公钥的指纹。

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub
256 da:24:43:0b:2e:c1:3f:a1:84:13:92:01:52:b4:84:ff      (ECDSA)
```

上面的例子中，`ssh-keygen -l -f` 命令会输出公钥 `/etc/ssh/ssh_host_ecdsa_key.pub` 的指纹。

ssh 会将本机连接过的所有服务器公钥的指纹，都储存在本机的 `~/.ssh/known_hosts` 文件中。每次连接服务器时，通过该文件判断是否为陌生主机（陌生公钥）。

在上面这段文字后面，输入 `yes`，就可以将当前服务器的指纹也储存在本机 `~/.ssh/known_hosts` 文件中，并显示下面的提示。以后再连接的时候，就不会再出现警告了。

```
Warning: Permanently added 'foo.com (192.168.121.111)' (RSA) to the
list of known hosts
```

然后，客户端就会跟服务器建立连接。接着，ssh 就会要求用户输入所要登录账户的密码。用户输入并验证密码正确以后，就能登录远程服务器的 Shell 了。

4. 服务器密钥变更

服务器指纹可以防止有人恶意冒充远程主机。如果服务器的密钥发生变更（比如重装了 SSH 服务器），客户端再次连接时，就会发生公钥指纹不吻合的情况。这时，客户端就会中断连接，并显示一段警告信息。

```
@@@@@@@WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle
attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
77:a5:69:81:9b:eb:40:76:7b:13:04:a9:6c:f4:9c:5d.
Please contact your system administrator.
Add correct host key in /home/me/.ssh/known_hosts to get rid of this
message.

Offending key in /home/me/.ssh/known_hosts:36
```

上面这段文字的意思是，该主机的公钥指纹跟 `~/.ssh/known_hosts` 文件储存的不一样，必须处理以后才能连接。这时，你需要确认是什么原因，使得公钥指纹发生变更，到底是恶意劫持，还是管理员变更了 SSH 服务器公钥。

如果新的公钥确认可以信任，需要继续执行连接，你可以执行下面的命令，将原来的公钥指纹从 `~/.ssh/known_hosts` 文件删除。

```
$ ssh-keygen -R hostname
```

上面命令中，`hostname` 是发生公钥变更的主机名。

除了使用上面的命令，你也可以手工修改 `known_hosts` 文件，将公钥指纹删除。

删除了原来的公钥指纹以后，重新执行 `ssh` 命令连接远程服务器，将新的指纹加入 `known_hosts` 文件，就可以顺利连接了。

5. 执行远程命令

SSH 登录成功后，用户就进入了远程主机的命令行环境，所看到的提示符，就是远程主机的提示符。这时，你就可以输入想要在远程主机执行的命令。

另一种执行远程命令的方法，是将命令直接写在 `ssh` 命令的后面。

```
$ ssh username@hostname command
```

上面的命令会使得 SSH 在登录成功后，立刻在远程主机上执行命令 `command`。

下面是一个例子。

```
$ ssh foo@server.example.com cat /etc/hosts
```

上面的命令会在登录成功后，立即远程执行命令 `cat /etc/hosts`。

采用这种语法执行命令时，`ssh` 客户端不会提供互动式的 Shell 环境，而是直接将远程命令的执行结果输出在命令行。但是，有些命令需要互动式的 Shell 环境，这时就要使用 `-t` 参数。

```
# 报错
$ ssh remote.server.com emacs
emacs: standard input is not a tty

# 不报错
$ ssh -t server.example.com emacs
```

上面代码中，`emacs` 命令需要一个互动式 Shell，所以报错。只有加上 `-t` 参数，`ssh` 才会分配一个互动式 Shell。

6. 加密参数

SSH 连接的握手阶段，客户端必须跟服务端约定加密参数集（cipher suite）。

加密参数集包含了若干不同的加密参数，它们之间使用下划线连接在一起，下面是一个例子。

TLS_RSA_WITH_AES_128_CBC_SHA

它的含义如下。

- TLS：加密通信协议
- RSA：密钥交换算法
- AES：加密算法
- 128：加密算法的强度
- CBC：加密算法的模式
- SHA：数字签名的 Hash 函数

下面是一个例子，客户端向服务器发出的握手信息。

```

Handshake protocol: ClientHello
Version: TLS 1.2
Random
    Client time: May 22, 2030 02:43:46 GMT
    Random bytes:
b76b0e61829557eb4c611adfd2d36eb232dc1332fe29802e321ee871
Session ID: (empty)
Cipher Suites
    Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256"
    Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    Suite: TLS_RSA_WITH_AES_128_GCM_SHA256
    Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
    Suite: TLS_RSA_WITH_AES_128_CBC_SHA
    Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
    Suite: TLS_RSA_WITH_RC4_128_SHA
Compression methods
Method: null
Extensions
Extension: server_name
    Hostname: www.feistyduck.com
Extension: renegotiation_info
Extension: elliptic_curves
    Named curve: secp256r1
    Named curve: secp384r1
Extension: signature_algorithms
    Algorithm: sha1/rsa
    Algorithm: sha256/rsa
    Algorithm: sha1/ecdsa
    Algorithm: sha256/ecdsa"

```

上面的握手信息 (ClientHello) 之中，**Cipher Suites** 字段就是客户端列出可选的加密参数集，服务器在其中选择一个自己支持的参数集。

服务器选择完毕之后，向客户端发出回应。

```

Handshake protocol: ServerHello
Version: TLS 1.2
Random
    Server time: Mar 10, 2059 02:35:57 GMT"
    Random bytes:
8469b09b480c1978182ce1b59290487609f41132312ca22aacaf5012
Session ID:
4cae75c91cf5adf55f93c9fb5dd36d19903b1182029af3d527b7a42ef1c32c80
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
Compression method: null
Extensions
    Extension: server_name
    Extension: renegotiation_info"

```

上面的回应信息（ServerHello）中，`Cipher Suite` 字段就是服务器最终选定的加密参数。

7. ssh 命令行配置项

ssh 命令有很多配置项，修改它的默认行为。

-c

`-c` 参数指定加密算法。

```

$ ssh -c blowfish,3des server.example.com
# 或者
$ ssh -c blowfish -c 3des server.example.com

```

上面命令指定使用加密算法 `blowfish` 或 `3des`。

-C

`-C` 参数表示压缩数据传输。

```
$ ssh -C server.example.com
```

-D

-D 参数指定本机的 Socks 监听端口，该端口收到的请求，都将转发到远程的 SSH 主机，又称动态端口转发，详见《端口转发》一章。

```
$ ssh -D 1080 server
```

上面命令将本机 1080 端口收到的请求，都转发到服务器 `server`。

-f

-f 参数表示 SSH 连接在后台运行。

-F

-F 参数指定配置文件。

```
$ ssh -F /usr/local/ssh/other_config
```

上面命令指定使用配置文件 `other_config`。

-i

-i 参数用于指定私钥，意为“`identity_file`”，默认值为 `~/.ssh/id_dsa` (DSA 算法) 和 `~/.ssh/id_rsa` (RSA 算法)。注意，对应的公钥必须存放到服务器，详见《密钥登录》一章。

```
$ ssh -i my-key server.example.com
```

-J

-J 指定跳转服务器。假定本地无法直接与 SSH 服务器通信，就可以通过 **-J** 指定跳转服务器。

```
$ ssh -J root@J1,root@J2 root@s1
```

上面示例中，本机先通过 J1，再通过 J2，登陆到 S1 服务器。

-l

-l 参数指定远程登录的账户名。

```
$ ssh -l sally server.example.com
# 等同于
$ ssh sally@server.example.com
```

-L

-L 参数设置本地端口转发，详见《端口转发》一章。

```
$ ssh -L 9999:targetServer:80 user@remoteserver
```

上面命令中，所有发向本地 `9999` 端口的请求，都会经过 `remoteserver` 发往 `targetServer` 的 `80` 端口，这就相当于直接连上了 `targetServer` 的 `80` 端口。

-m

-m 参数指定校验数据完整性的算法（message authentication code，简称 MAC）。

```
$ ssh -m hmac-sha1,hmac-md5 server.example.com
```

上面命令指定数据校验算法为 `hmac-sha1` 或 `hmac-md5`。

-N

-N 参数用于端口转发，表示建立的 SSH 只用于端口转发，不能执行远程命令，这样可以提供安全性，详见《端口转发》一章。

-o

-o 参数用来指定一个配置命令。

```
$ ssh -o "Keyword Value"
```

举例来说，配置文件里面有如下内容。

```
User sally
Port 220
```

通过 **-o** 参数，可以把上面两个配置命令从命令行传入。

```
$ ssh -o "User sally" -o "Port 220" server.example.com
```

使用等号时，配置命令可以不用写在引号里面，但是等号前后不能有空格。

```
$ ssh -o User=sally -o Port=220 server.example.com
```

-p

`-p` 参数指定 SSH 客户端连接的服务器端口。

```
$ ssh -p 2035 server.example.com
```

上面命令连接服务器的2035端口。

-q

`-q` 参数表示安静模式（quiet），不向用户输出任何警告信息。

```
$ ssh -q foo.com  
root's password:
```

上面命令使用 `-q` 参数，只输出要求用户输入密码的提示。

-R

`-R` 参数指定远程端口转发，详见《端口转发》一章。

```
$ ssh -R 9999:targetServer:902 local
```

上面命令需在跳板服务器执行，指定本地计算机 `local` 监听自己的 9999 端口，所有发向这个端口的请求，都会转向 `targetServer` 的 902 端口。

-t

`-t` 参数在 ssh 直接运行远端命令时，提供一个互动式 Shell。

```
$ ssh -t server.example.com emacs
```

-v

-v 参数显示详细信息。

```
$ ssh -v server.example.com
```

-v 可以重复多次，表示信息的详细程度，比如 **-vv** 和 **-vvv**。

```
$ ssh -vvv server.example.com
# 或者
$ ssh -v -v -v server.example.com
```

上面命令会输出最详细的连接信息。

-V

-V 参数输出 ssh 客户端的版本。

```
$ ssh -V
ssh: SSH Secure Shell 3.2.3 (non-commercial version) on i686-pc-linux-gnu
```

上面命令输出本机 ssh 客户端版本是 **SSH Secure Shell 3.2.3**。

-X

-X 参数表示打开 X 窗口转发。

```
$ ssh -X server.example.com
```

-1, -2

-1 参数指定使用 SSH 1 协议。

-2 参数指定使用 SSH 2 协议。

```
$ ssh -2 server.example.com
```

-4, -6

-4 指定使用 IPv4 协议，这是默认值。

```
$ ssh -4 server.example.com
```

`-6` 指定使用 IPv6 协议。

```
$ ssh -6 server.example.com
```

8. 客户端配置文件

8.1 位置

SSH 客户端的全局配置文件是 `/etc/ssh/ssh_config`，用户个人的配置文件在 `~/.ssh/config`，优先级高于全局配置文件。

除了配置文件，`~/.ssh` 目录还有一些用户个人的密钥文件和其他文件。下面是其中一些常见的文件。

- `~/.ssh/id_ecdsa`：用户的 ECDSA 私钥。
- `~/.ssh/id_ecdsa.pub`：用户的 ECDSA 公钥。
- `~/.ssh/id_rsa`：用于 SSH 协议版本2 的 RSA 私钥。
- `~/.ssh/id_rsa.pub`：用于 SSH 协议版本2 的 RSA 公钥。
- `~/.ssh/identity`：用于 SSH 协议版本1 的 RSA 私钥。
- `~/.ssh/identity.pub`：用于 SSH 协议版本1 的 RSA 公钥。
- `~/.ssh/known_hosts`：包含 SSH 服务器的公钥指纹。

8.2 主机设置

用户个人的配置文件 `~/.ssh/config`，可以按照不同服务器，列出各自的连接参数，从而不必每一次登录都输入重复的参数。下面是一个例子。

```

Host remoteserver
  HostName remote.example.com
  User neo
  Port 2112

Host *
  Port 2222

```

上面代码中，`Host remoteserver` 表示，下面的设置只对主机 `remoteserver` 生效。`remoteserver` 只是一个别名，具体的主机由 `HostName` 命令指定，`User` 和 `Port` 这两项分别表示用户名和端口。`HostName`、`User`、`Port` 这三项前面的缩进并不是必需的，只是为了视觉上易于识别针对不同主机的设置。

后面的 `Host *` 表示对所有主机生效，`*` 是一个通配符，比如 `Host *.edu` 表示只对一级域名为 `.edu` 的主机生效。这条命令下面的 `Port 2222` 表示所有主机的默认连接端口都是2222，这样就不用在登录时特别指定端口了。

注意，当 `Host *` 与 `Host remoteserver` 下面有同一项设定时（比如两者都有 `Port` 设定），第一个出现的值生效。在本例中，连接 `remoteserver` 时，默认端口将是2112，而不是2222，如果 `Host *` 放在配置文件的顶部，那么默认端口将是2222。

以后，登录 `remote.example.com` 时，只要执行 `ssh remoteserver` 命令，就会自动套用 config 文件里面指定的参数。

```

$ ssh remoteserver
# 等同于
$ ssh -p 2112 neo@remote.example.com

```

8.3 配置命令的语法

`ssh` 客户端配置文件的每一行，就是一个配置命令。配置命令与对应的值之间，可以使用空格，也可以使用等号。

```

Compression yes
# 等同于
Compression = yes

```

`#` 开头的行表示注释，会被忽略。空行等同于注释。

8.4 主要配置命令

下面是 ssh 客户端的一些主要配置命令，以及它们的范例值。

- `AddressFamily inet`：表示只使用 IPv4 协议。如果设为 `inet6`，表示只使用 IPv6 协议。
- `BindAddress 192.168.10.235`：指定本机的 IP 地址（如果本机有多个 IP 地址）。
- `CheckHostIP yes`：检查 SSH 服务器的 IP 地址是否跟公钥数据库吻合。
- `Ciphers blowfish,3des`：指定加密算法。
- `Compression yes`：是否压缩传输信号。
- `ConnectionAttempts 10`：客户端进行连接时，最大的尝试次数。
- `ConnectTimeout 60`：客户端进行连接时，服务器在指定秒数内没有回复，则中断连接尝试。
- `DynamicForward 1080`：指定动态转发端口。
- `GlobalKnownHostsFile /users/smith/.ssh/my_global_hosts_file`：指定全局的公钥数据库文件的位置。
- `Host server.example.com`：指定连接的域名或 IP 地址，也可以是别名，支持通配符。`Host` 命令后面的所有配置，都是针对该主机的，直到下一个 `Host` 命令为止。
- `HostKeyAlgorithms ssh-dss,ssh-rsa`：指定密钥算法，优先级从高到低排列。
- `HostName myserver.example.com`：在 `Host` 命令使用别名的情况下，`HostName` 指定域名或 IP 地址。
- `IdentityFile keyfile`：指定私钥文件。
- `LocalForward 2001 localhost:143`：指定本地端口转发。
- `LogLevel QUIET`：指定日志详细程度。如果设为 `QUIET`，将不输出大部分的警告和提示。
- `MACs hmac-sha1,hmac-md5`：指定数据校验算法。
- `NumberOfPasswordPrompts 2`：密码登录时，用户输错密码的最大尝试次数。
- `PasswordAuthentication no`：指定是否支持密码登录。不过，这里只是客户端禁止，真正的禁止需要在 SSH 服务器设置。
- `Port 2035`：指定客户端连接的 SSH 服务器端口。
- `PreferredAuthentications publickey,hostbased,password`：指定各种登录方法的优先级。
- `Protocol 2`：支持的 SSH 协议版本，多个版本之间使用逗号分隔。
- `PubKeyAuthentication yes`：是否支持密钥登录。这里只是客户端设置，还需要在 SSH 服务器进行相应设置。

- `RemoteForward 2001 server:143` : 指定远程端口转发。
- `SendEnv COLOR` : SSH 客户端向服务器发送的环境变量名，多个环境变量之间使用空格分隔。环境变量的值从客户端当前环境中拷贝。
- `ServerAliveCountMax 3` : 如果没有收到服务器的回应，客户端连续发送多少次 `keepalive` 信号，才断开连接。该项默认值为3。
- `ServerAliveInterval 300` : 客户端建立连接后，如果在给定秒数内，没有收到服务器发来的消息，客户端向服务器发送 `keepalive` 消息。如果不希望客户端发送，这一项设为 `0`，即客户端不会主动断开连接。
- `StrictHostKeyChecking yes` : `yes` 表示严格检查，服务器公钥为未知或发生变化，则拒绝连接。`no` 表示如果服务器公钥未知，则加入客户端公钥数据库，如果公钥发生变化，不改变客户端公钥数据库，输出一条警告，依然允许连接继续进行。`ask` (默认值) 表示询问用户是否继续进行。
- `TCPKeepAlive yes` : 客户端是否定期向服务器发送 `keepalive` 信息。
- `User userName` : 指定远程登录的账户名。
- `UserKnownHostsFile /userssmith/.ssh/my_local_hosts_file` : 指定当前用户的 `known_hosts` 文件 (服务器公钥指纹列表) 的位置。
- `VerifyHostKeyDNS yes` : 是否通过检查 SSH 服务器的 DNS 记录，确认公钥指纹是否与 `known_hosts` 文件保存的一致。

SSH 密钥登录

SSH 默认采用密码登录，这种方法有很多缺点，简单的密码不安全，复杂的密码不容易记忆，每次手动输入也很麻烦。密钥登录是比密码登录更好的解决方案。

目录 [隐藏]

1. 密钥是什么
2. 密钥登录的过程
3. `ssh-keygen` 命令：生成密钥
 - 3.1 基本用法
 - 3.2 配置项
4. 手动上传公钥
5. `ssh-copy-id` 命令：自动上传公钥
6. `ssh-agent` 命令, `ssh-add` 命令
 - 6.1 基本用法
 - 6.2 `ssh-add` 命令
7. 关闭密码登录

1. 密钥是什么

密钥（key）是一个非常大的数字，通过加密算法得到。对称加密只需要一个密钥，非对称加密需要两个密钥成对使用，分为公钥（public key）和私钥（private key）。

SSH 密钥登录采用的是非对称加密，每个用户通过自己的密钥登录。其中，私钥必须私密保存，不能泄漏；公钥则是公开的，可以对外发送。它们的关系是，公钥和私钥是一一对应的，每一个私钥都有且仅有一个对应的公钥，反之亦然。

如果数据使用公钥加密，那么只有使用对应的私钥才能解密，其他密钥都不行；反过来，如果使用私钥加密（这个过程一般称为“签名”），也只有使用对应的公钥解密。

2. 密钥登录的过程

SSH 密钥登录分为以下的步骤。

预备步骤，客户端通过 `ssh-keygen` 生成自己的公钥和私钥。

第一步，手动将客户端的公钥放入远程服务器的指定位置。

第二步，客户端向服务器发起 SSH 登录的请求。

第三步，服务器收到用户 SSH 登录的请求，发送一些随机数据给用户，要求用户证明自己的身份。

第四步，客户端收到服务器发来的数据，使用私钥对数据进行签名，然后再发还给服务器。

第五步，服务器收到客户端发来的加密签名后，使用对应的公钥解密，然后跟原始数据比较。如果一致，就允许用户登录。

3. `ssh-keygen` 命令：生成密钥

3.1 基本用法

密钥登录时，首先需要生成公钥和私钥。OpenSSH 提供了一个工具程序 `ssh-keygen` 命令，用来生成密钥。

直接输入 `ssh-keygen`，程序会询问一系列问题，然后生成密钥。

```
$ ssh-keygen
```

通常做法是使用 `-t` 参数，指定密钥的加密算法。

```
$ ssh-keygen -t dsa
```

上面示例中，`-t` 参数用来指定密钥的加密算法，一般会选择 DSA 算法或 RSA 算法。如果省略该参数，默认使用 RSA 算法。

输入上面的命令以后，`ssh-keygen` 会要求用户回答一些问题。

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_dsa):
press ENTER
Enter passphrase (empty for no passphrase): *****
Enter same passphrase again: *****
Your identification has been saved in /home/username/.ssh/id_dsa.
Your public key has been saved in /home/username/.ssh/id_dsa.pub.
The key fingerprint is:
14:ba:06:98:a8:98:ad:27:b5:ce:55:85:ec:64:37:19 username@shell.isp.com
```

上面示例中，执行 `ssh-keygen` 命令以后，会出现第一个问题，询问密钥保存的文件名，默认是 `~/.ssh/id_dsa` 文件，这个是私钥的文件名，对应的公钥文件 `~/.ssh/id_dsa.pub` 是自动生成的。用户的密钥一般都放在主目录的 `.ssh` 目录里面。

如果选择 `rsa` 算法，生成的密钥文件默认就会是 `~/.ssh/id_rsa`（私钥）和 `~/.ssh/id_rsa.pub`（公钥）。

接着，就会是第二个问题，询问是否要为私钥文件设定密码保护（passphrase）。这样的话，即使入侵者拿到私钥，还是需要破解密码。如果为了方便，不想设定密码保护，可以直接按回车键，密码就会为空。后面还会让你再输入一次密码，两次输入必须一致。注意，这里“密码”的英文单词是 `passphrase`，这是为了避免与 Linux 账户的密码单词 `password` 混淆，表示这不是用户系统账户的密码。

最后，就会生成私钥和公钥，屏幕上还会给出公钥的指纹，以及当前的用户名和主机名作为注释，用来识别密钥的来源。

公钥文件和私钥文件都是文本文件，可以用文本编辑器看一下它们的内容。公钥文件的内容类似下面这样。

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAQIEAvpB4lUbAaEb9u6HLig7amsfywD4fqSzq2ikACIUBn3GyR
weQh702ofXbDydZAKMcDvBJqRhUotQUwqV6HJxqoqPD1PGUUYo8RDIkLUIPRyq
ypZxmK9aCXokFiHoGCXfQ9imUP/w/jfqb9ByDtG97tUJF6nFMP5WzhM=
username@shell.isp.com
```

上面示例中，末尾的 `username@shell.isp.com` 是公钥的注释，用来识别不同的公钥，表示这是哪台主机（`shell.isp.com`）的那个用户（`username`）的公钥，不是必需项。

注意，公钥只有一行。因为它太长了，所以上面分成三行显示。

下面的命令可以列出用户所有的公钥。

```
$ ls -l ~/.ssh/id_*.pub
```

生成密钥以后，建议修改它们的权限，防止其他人读取。

```
$ chmod 600 ~/.ssh/id_rsa
$ chmod 600 ~/.ssh/id_rsa.pub
```

3.2 配置项

`ssh-keygen` 的命令行配置项，主要有下面这些。

(1) `-b`

`-b` 参数指定密钥的二进制位数。这个参数值越大，密钥就越不容易破解，但是加密解密的计算开销也会加大。

一般来说，`-b` 至少应该是 1024，更安全一些可以设为 2048 或者更高。

(2) `-C`

`-C` 参数可以为密钥文件指定新的注释，格式为 `username@host`。

下面命令生成一个4096位 RSA 加密算法的密钥对，并且给出了用户名和主机名。

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@domain.com"
```

(3) `-f`

`-f` 参数指定生成的私钥文件。

```
$ ssh-keygen -t dsa -f mykey
```

上面命令会在当前目录生成私钥文件 `mykey` 和公钥文件 `mykey.pub`。

(4) `-F`

`-F` 参数检查某个主机名是否在 `known_hosts` 文件里面。

```
$ ssh-keygen -F example.com
```

(5) `-N`

`-N` 参数用于指定私钥的密码 (passphrase)。

```
$ ssh-keygen -t dsa -N secretword
```

(6) `-p`

`-p` 参数用于重新指定私钥的密码 (passphrase)。它与 `-N` 的不同之处在于，新密码不在命令中指定，而是执行后再输入。ssh 先要求输入旧密码，然后要求输入两遍新密码。

(7) `-R`

`-R` 参数将指定的主机公钥指纹移出 `known_hosts` 文件。

```
$ ssh-keygen -R example.com
```

(8) `-t`

`-t` 参数用于指定生成密钥的加密算法，一般为 `dsa` 或 `rsa`

4. 手动上传公钥

生成密钥以后，公钥必须上传到服务器，才能使用公钥登录。

OpenSSH 规定，用户公钥保存在服务器的 `~/.ssh/authorized_keys` 文件。你要以哪个用户的身份登录到服务器，密钥就必须保存在该用户主目录的 `~/.ssh/authorized_keys` 文件。只要把公钥添加到这个文件之中，就相当于公钥上传到服务器了。每个公钥占据一行。如果该文件不存在，可以手动创建。

用户可以手动编辑该文件，把公钥粘贴进去，也可以在本机计算机上，执行下面的命令。

```
$ cat ~/.ssh/id_rsa.pub | ssh user@host "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

上面示例中，`user@host` 要替换成你所要登录的用户名和主机名。

注意，`authorized_keys` 文件的权限要设为 `644`，即只有文件所有者才能写。如果权限设置不对，SSH 服务器可能会拒绝读取该文件。

```
$ chmod 644 ~/.ssh/authorized_keys
```

只要公钥上传到服务器，下次登录时，OpenSSH 就会自动采用密钥登录，不再提示输入密码。

```
$ ssh -l username shell.isp.com
Enter passphrase for key '/home/you/.ssh/id_dsa': *****
Last login: Mon Mar 24 02:17:27 2014 from ex.ample.com
shell.isp.com>
```

上面例子中，SSH 客户端使用私钥之前，会要求用户输入密码（passphrase），用来解开私钥。

5. ssh-copy-id 命令：自动上传公钥

OpenSSH 自带一个 `ssh-copy-id` 命令，可以自动将公钥拷贝到远程服务器的 `~/.ssh/authorized_keys` 文件。如果 `~/.ssh/authorized_keys` 文件不存在，`ssh-copy-id` 命令会自动创建该文件。

用户在本地计算机执行下面的命令，就可以把本地的公钥拷贝到服务器。

```
$ ssh-copy-id -i key_file user@host
```

上面命令中，`-i` 参数用来指定公钥文件，`user` 是所要登录的账户名，`host` 是服务器地址。如果省略用户名，默认为当前的本机用户名。执行完该命令，公钥就会拷贝到服务器。

注意，公钥文件可以不指定路径和 `.pub` 后缀名，`ssh-copy-id` 会自动在 `~/.ssh` 目录里面寻找。

```
$ ssh-copy-id -i id_rsa user@host
```

上面命令中，公钥文件会自动匹配到 `~/.ssh/id_rsa.pub`。

`ssh-copy-id` 会采用密码登录，系统会提示输入远程服务器的密码。

注意，`ssh-copy-id` 是直接将公钥添加到 `authorized_keys` 文件的末尾。如果 `authorized_keys` 文件的末尾不是一个换行符，会导致新的公钥添加到前一个公钥的末尾，两个公钥连在一起，使得它们都无法生效。所以，如果 `authorized_keys` 文件已经存在，使用 `ssh-copy-id` 命令之前，务必保证 `authorized_keys` 文件的末尾是换行符（假设该文件已经存在）。

6. ssh-agent 命令，ssh-add 命令

6.1 基本用法

私钥设置了密码以后，每次使用都必须输入密码，有时让人感觉非常麻烦。比如，连续使用 `scp` 命令远程拷贝文件时，每次都要求输入密码。

`ssh-agent` 命令就是为了解决这个问题而设计的，它让用户在整个 Bash 对话 (session) 之中，只在第一次使用 SSH 命令时输入密码，然后将私钥保存在内存中，后面都不需要再输入私钥的密码了。

第一步，使用下面的命令新建一次命令行对话。

```
$ ssh-agent bash
```

上面命令中，如果你使用的命令行环境不是 Bash，可以用其他的 Shell 命令代替。比如 `zsh` 和 `fish`。

如果想在当前对话启用 `ssh-agent`，可以使用下面的命令。

```
$ eval `ssh-agent`
```

上面命令中，`ssh-agent` 会先自动在后台运行，并将需要设置的环境变量输出在屏幕上，类似下面这样。

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-barrett/ssh-22841-agent; export SSH_AUTH_SOCK;
SSH_AGENT_PID=22842; export SSH_AGENT_PID;
echo Agent pid 22842;
```

`eval` 命令的作用，就是运行上面的 `ssh-agent` 命令的输出，设置环境变量。

第二步，在新建的 Shell 对话里面，使用 `ssh-add` 命令添加默认的私钥（比如 `~/.ssh/id_rsa`，或 `~/.ssh/id_dsa`，或 `~/.ssh/id_ecdsa`，或 `~/.ssh/id_ed25519`）。

```
$ ssh-add
Enter passphrase for /home/you/.ssh/id_dsa: *****
Identity added: /home/you/.ssh/id_dsa (/home/you/.ssh/id_dsa)
```

上面例子中，添加私钥时，会要求输入密码。以后，在这个对话里面再使用密钥时，就不需要输入私钥的密码了，因为私钥已经加载到内存里面了。

如果添加的不是默认私钥，`ssh-add` 命令需要显式指定私钥文件。

```
$ ssh-add my-other-key-file
```

上面的命令中，`my-other-key-file` 就是用户指定的私钥文件。

第三步，使用 `ssh` 命令正常登录远程服务器。

```
$ ssh remoteHost
```

上面命令中，`remoteHost` 是远程服务器的地址，`ssh` 使用的是默认的私钥。这时如果私钥设有密码，`ssh` 将不再询问密码，而是直接取出内存里面的私钥。

如果要使用其他私钥登录服务器，需要使用 `ssh` 命令的 `-i` 参数指定私钥文件。

```
$ ssh -i OpenSSHPublicKey remoteHost
```

最后，如果要退出 `ssh-agent`，可以直接退出子 Shell（按下 `Ctrl + d`），也可以使用下面的命令。

```
$ ssh-agent -k
```

6.2 ssh-add 命令

`ssh-add` 命令用来将私钥加入 `ssh-agent`，它有如下的参数。

(1) `-d`

`-d` 参数从内存中删除指定的私钥。

```
$ ssh-add -d name-of-key-file
```

(2) `-D`

`-D` 参数从内存中删除所有已经添加的私钥。

```
$ ssh-add -D
```

(3) `-l`

`-l` 参数列出所有已经添加的私钥。

```
$ ssh-add -l
```

7. 关闭密码登录

为了安全性，启用密钥登录之后，最好关闭服务器的密码登录。

对于 OpenSSH，具体方法就是打开服务器 `sshd` 的配置文件 `/etc/ssh/sshd_config`，将 `PasswordAuthentication` 这一项设为 `no`。

```
PasswordAuthentication no
```

修改配置文件以后，不要忘了重新启动 `sshd`，否则不会生效。

SSH 服务器

目录 [隐藏]

1. 简介
2. sshd 配置文件
3. sshd 密钥
4. sshd 配置项
5. sshd 的命令行配置项

1. 简介

SSH 的架构是服务器/客户端模式，两端运行的软件是不一样的。OpenSSH 的客户端软件是 ssh，服务器软件是 sshd。本章介绍 sshd 的各种知识。

如果没有安装 sshd，可以用下面的命令安装。

```
# Debian
$ sudo aptitude install openssh-server

# Red Hat
$ sudo yum install openssh-server
```

一般来说，sshd 安装后会跟着系统一起启动。如果当前 sshd 没有启动，可以用下面的命令启动。

```
$ sshd
```

上面的命令运行后，如果提示“sshd re-exec requires execution with an absolute path”，就需要使用绝对路径来启动。这是为了防止有人出于各种目的，放置同名软件在 `$PATH` 变量指向的目录中，代替真正的 sshd。

```
# Centos、Ubuntu、OS X
$ /usr/sbin/sshd
```

上面的命令运行以后，sshd 自动进入后台，所以命令后面不需要加上 `&`。

除了直接运行可执行文件，也可以通过 Systemd 启动 sshd。

```
# 启动
$ sudo systemctl start sshd.service

# 停止
$ sudo systemctl stop sshd.service

# 重启
$ sudo systemctl restart sshd.service
```

下面的命令让 sshd 在计算机下次启动时自动运行。

```
$ sudo systemctl enable sshd.service
```

2. sshd 配置文件

sshd 的配置文件在 `/etc/ssh` 目录，主配置文件是 `sshd_config`，此外还有一些安装时生成的密钥。

- `/etc/ssh/sshd_config`：配置文件
- `/etc/ssh/ssh_host_ecdsa_key`：ECDSA 私钥。
- `/etc/ssh/ssh_host_ecdsa_key.pub`：ECDSA 公钥。
- `/etc/ssh/ssh_host_key`：用于 SSH 1 协议版本的 RSA 私钥。
- `/etc/ssh/ssh_host_key.pub`：用于 SSH 1 协议版本的 RSA 公钥。
- `/etc/ssh/ssh_host_rsa_key`：用于 SSH 2 协议版本的 RSA 私钥。
- `/etc/ssh/ssh_host_rsa_key.pub`：用于 SSH 2 协议版本的 RSA 公钥。
- `/etc/pam.d/sshd`：PAM 配置文件。

注意，如果重装 sshd，上面这些密钥都会重新生成，导致客户端重新连接 ssh 服务器时，会跳出警告，拒绝连接。为了避免这种情况，可以在重装 sshd 时，先备

份 `/etc/ssh` 目录，重装后再恢复这个目录。

配置文件 `sshd_config` 的格式是，每个命令占据一行。每行都是配置项和对应的值，配置项的大小写不敏感，与值之间使用空格分隔。

```
Port 2034
```

上面的配置命令指定，配置项 `Port` 的值是 `2034`。`Port` 写成 `port` 也可。

配置文件还有另一种格式，就是配置项与值之间有一个等号，等号前后的空格可选。

```
Port = 2034
```

配置文件里面，`#` 开头的行表示注释。

```
# 这是一行注释
```

注意，注释只能放在一行的开头，不能放在一行的结尾。

```
Port 2034 # 此处不允许注释
```

上面的写法是错误的。

另外，空行等同于注释。

`sshd` 启动时会自动读取默认的配置文件。如果希望使用其他的配置文件，可以用 `sshd` 命令的 `-f` 参数指定。

```
$ sshd -f /usr/local/ssh/my_config
```

上面的命令指定 `sshd` 使用另一个配置文件 `my_config`。

修改配置文件以后，可以用 `sshd` 命令的 `-t` (`test`) 检查有没有语法错误。

```
$ sshd -t
```

配置文件修改以后，并不会自动生效，必须重新启动 `sshd`。

```
$ sudo systemctl restart sshd.service
```

3. sshd 密钥

sshd 有自己的一对或多对密钥。它使用密钥向客户端证明自己的身份。所有密钥都是公钥和私钥成对出现，公钥的文件名一般是私钥文件名加上后缀 `.pub`。

DSA 格式的密钥文件默认为 `/etc/ssh/ssh_host_dsa_key`（公钥为 `ssh_host_dsa_key.pub`），RSA 格式的密钥为 `/etc/ssh/ssh_host_rsa_key`（公钥为 `ssh_host_rsa_key.pub`）。如果需要支持 SSH 1 协议，则必须有密钥 `/etc/ssh/ssh_host_key`。

如果密钥不是默认文件，那么可以通过配置文件 `sshd_config` 的 `HostKey` 配置项指定。默认密钥的 `HostKey` 设置如下。

```
# HostKey for protocol version 1
# HostKey /etc/ssh/ssh_host_key

# HostKeys for protocol version 2
# HostKey /etc/ssh/ssh_host_rsa_key
# HostKey /etc/ssh/ssh_host_dsa_key
```

上面命令前面的 `#` 表示这些行都是注释，因为这是默认值，有没有这几行都一样。

如果要修改密钥，就要去掉行首的 `#`，指定其他密钥。

```
HostKey /usr/local/ssh/my_dsa_key
HostKey /usr/local/ssh/my_rsa_key
HostKey /usr/local/ssh/my_old_ssh1_key
```

4. sshd 配置项

以下是 `/etc/ssh/sshd_config` 文件里面的配置项。

AcceptEnv

`AcceptEnv` 指定允许接受客户端通过 `SendEnv` 命令发来的哪些环境变量，即允许客户端设置服务器的环境变量清单，变量名之间使用空格分隔（`AcceptEnv PATH TERM`）。

AllowGroups

`AllowGroups` 指定允许登录的用户组（`AllowGroups groupName`，多个组之间用空格分隔。如果不使用该项，则允许所有用户组登录。

AllowUsers

`AllowUsers` 指定允许登录的用户，用户名之间使用空格分隔（`AllowUsers user1 user2`），也可以使用多行 `AllowUsers` 命令指定，用户名支持使用通配符。如果不使用该项，则允许所有用户登录。该项也可以使用 `用户名@域名` 的格式（比如 `AllowUsers jones@example.com`）。

AllowTcpForwarding

`AllowTcpForwarding` 指定是否允许端口转发，默认值为 `yes`（`AllowTcpForwarding yes`），`local` 表示只允许本地端口转发，`remote` 表示只允许远程端口转发。

AuthorizedKeysFile

`AuthorizedKeysFile` 指定储存用户公钥的目录，默认是用户主目录的 `ssh/authorized_keys` 目录（`AuthorizedKeysFile .ssh/authorized_keys`）。

Banner

`Banner` 指定用户登录后，`sshd` 向其展示的信息文件（`Banner /usr/local/etc/warning.txt`），默认不展示任何内容。

ChallengeResponseAuthentication

`ChallengeResponseAuthentication` 指定是否使用“键盘交互”身份验证方案，默认值为 `yes`（`ChallengeResponseAuthentication yes`）。

从理论上讲，“键盘交互”身份验证方案可以向用户询问多重问题，但是实践中，通常仅询问用户密码。如果要完全禁用基于密码的身份验证，请将 `PasswordAuthentication` 和 `ChallengeResponseAuthentication` 都设置为 `no`。

Ciphers

`Ciphers` 指定 `sshd` 可以接受的加密算法（`Ciphers 3des-cbc`），多个算法之间使用逗号分隔。

ClientAliveCountMax

`ClientAliveCountMax` 指定建立连接后，客户端失去响应时（超过指定时间，没有收到任何消息），服务器尝试连接（发送消息）的次数（`ClientAliveCountMax 8`）。

ClientAliveInterval

`ClientAliveInterval` 指定允许客户端发呆的时间，单位为秒（`ClientAliveInterval 180`）。超过这个时间，服务器将发送消息以请求客户端的响应。如果为 `0`，表示不向客户端发送消息，即连接不会自动断开。

Compression

`Compression` 指定客户端与服务器之间的数据传输是否压缩。默认值为 `yes`（`Compression yes`）。

DenyGroups

`DenyGroups` 指定不允许登录的用户组（`DenyGroups groupName`）。

DenyUsers

`DenyUsers` 指定不允许登录的用户（`DenyUsers user1`），用户名之间使用空格分隔，也可以使用多行 `DenyUsers` 命令指定。

FascistLogging

SSH 1 版本专用，指定日志输出全部 Debug 信息（`FascistLogging yes`）。

HostKey

`HostKey` 指定 sshd 服务器的密钥，详见前文。

Key_regeneration_interval

`Key_regeneration_interval` 指定 SSH 1 版本的密钥重新生成时间间隔，单位为秒，默认是 3600 秒（`Key_regeneration_interval 3600`）。

ListenAddress

`ListenAddress` 指定 sshd 监听的本机 IP 地址，即 sshd 启用的 IP 地址，默认是 `0.0.0.0`（`ListenAddress 0.0.0.0`）表示在本机所有网络接口启用。可以改成只在某个网络接口启用（比如 `ListenAddress 192.168.10.23`），也可以指定某个域名启用（比如 `ListenAddress server.example.com`）。

如果要监听多个指定的 IP 地址，可以使用多行 `ListenAddress` 命令。

```
ListenAddress 172.16.1.1
ListenAddress 192.168.0.1
```

LoginGraceTime

`LoginGraceTime` 指定允许客户端登录时发呆的最长时间，比如用户迟迟不输入密码，连接就会自动断开，单位为秒（`LoginGraceTime 60`）。如果设为 `0`，就表示没有限制。

LogLevel

`LogLevel` 指定日志的详细程度，可能的值依次为 `QUIET`、`FATAL`、`ERROR`、`INFO`、`VERBOSE`、`DEBUG`、`DEBUG1`、`DEBUG2`、`DEBUG3`，默认为 `INFO`（`LogLevel INFO`）。

MACs

`MACs` 指定sshd 可以接受的数据校验算法（`MACs hmac-sha1`），多个算法之间使用逗号分隔。

MaxAuthTries

`MaxAuthTries` 指定允许 SSH 登录的最大尝试次数（`MaxAuthTries 3`），如果密码输入错误达到指定次数，SSH 连接将关闭。

MaxStartups

`MaxStartups` 指定允许同时并发的 SSH 连接数量（`MaxStartups`）。如果设为 `0`，就表示没有限制。

这个属性也可以设为 `A:B:C` 的形式，比如 `MaxStartups 10:50:20`，表示如果达到10个并发连接，后面的连接将有50%的概率被拒绝；如果达到20个并发连接，则后面的连接将100%被拒绝。

PasswordAuthentication

`PasswordAuthentication` 指定是否允许密码登录，默认值为 `yes`（`PasswordAuthentication yes`），建议改成 `no`（禁止密码登录，只允许密钥登录）。

PermitEmptyPasswords

`PermitEmptyPasswords` 指定是否允许空密码登录，即用户的密码是否可以为空，默认为 `yes`（`PermitEmptyPasswords yes`），建议改成 `no`（禁止无密码登录）。

PermitRootLogin

`PermitRootLogin` 指定是否允许根用户登录，默认为 `yes` (`PermitRootLogin yes`)，建议改成 `no` (禁止根用户登录)。

还有一种写法是写成 `prohibit-password`，表示 root 用户不能用密码登录，但是可以用密钥登录。

```
PermitRootLogin prohibit-password
```

PermitUserEnvironment

`PermitUserEnvironment` 指定是否允许 sshd 加载客户端的 `~/.ssh/environment` 文件和 `~/.ssh/authorized_keys` 文件里面的 `environment= options` 环境变量设置。默认值为 `no` (`PermitUserEnvironment no`)。

Port

`Port` 指定 sshd 监听的端口，即客户端连接的端口，默认是 22 (`Port 22`)。出于安全考虑，可以改掉这个端口 (比如 `Port 8822`)。

配置文件可以使用多个 `Port` 命令，同时监听多个端口。

```
Port 22
Port 80
Port 443
Port 8080
```

上面的示例表示同时监听 4 个端口。

PrintMotd

`PrintMotd` 指定用户登录后，是否向其展示系统的 motd (Message of the day) 的信息文件 `/etc/motd`。该文件用于通知所有用户一些重要事项，比如系统维护时间、安全问题等等。默认值为 `yes` (`PrintMotd yes`)，由于 Shell 一般会展示这个信息文件，所以这里可以改为 `no`。

PrintLastLog

`PrintLastLog` 指定是否打印上一次用户登录时间，默认值为 `yes` (`PrintLastLog yes`)。

Protocol

`Protocol` 指定 sshd 使用的协议。`Protocol 1` 表示使用 SSH 1 协议，建议改成 `Protocol 2`（使用 SSH 2 协议）。`Protocol 2,1` 表示同时支持两个版本的协议。

PubkeyAuthentication

`PubkeyAuthentication` 指定是否允许公钥登录，默认值为 `yes`（`PubkeyAuthentication yes`）。

QuietMode

SSH 1 版本专用，指定日志只输出致命的错误信息（`QuietMode yes`）。

RSAAuthentication

`RSAAuthentication` 指定允许 RSA 认证，默认值为 `yes`（`RSAAuthentication yes`）。

ServerKeyBits

`ServerKeyBits` 指定 SSH 1 版本的密钥重新生成时的位数，默认是 768（`ServerKeyBits 768`）。

StrictModes

`StrictModes` 指定 sshd 是否检查用户的一些重要文件和目录的权限。默认为 `yes`（`StrictModes yes`），即对于用户的 SSH 配置文件、密钥文件和所在目录，SSH 要求拥有者必须是根用户或用户本人，用户组和其他人的写权限必须关闭。

SyslogFacility

`SyslogFacility` 指定 Syslog 如何处理 sshd 的日志，默认是 Auth（`SyslogFacility AUTH`）。

TCPKeepAlive

`TCPKeepAlive` 指定系统是否应向客户端发送 TCP keepalive 消息（`TCPKeepAlive yes`）。

UseDNS

`UseDNS` 指定用户 SSH 登录一个域名时，服务器是否使用 DNS，确认该域名对应的 IP 地址包含本机（`UseDNS yes`）。打开该选项意义不大，而且如果 DNS 更新不及时，还有可能误判，建议关闭。

UseLogin

`UseLogin` 指定用户认证内部是否使用 `/usr/bin/login` 替代 SSH 工具，默认为 `no` (`UseLogin no`)。

UserPrivilegeSeparation

`UserPrivilegeSeparation` 指定用户认证通过以后，使用另一个子线程处理用户权限相关的操作，这样有利于提高安全性。默认值为 `yes` (`UsePrivilegeSeparation yes`)。

VerboseMode

SSH 2 版本专用，指定日志输出详细的 Debug 信息 (`VerboseMode yes`)。

X11Forwarding

`X11Forwarding` 指定是否打开 X window 的转发，默认值为 `no` (`X11Forwarding no`)。

修改配置文件以后，可以使用下面的命令验证，配置文件是否有语法错误。

```
$ sshd -t
```

新的配置文件生效，必须重启 sshd。

```
$ sudo systemctl restart sshd
```

5. sshd 的命令行配置项

sshd 命令有一些配置项。这些配置项在调用时指定，可以覆盖配置文件的设置。

(1) `-d`

`-d` 参数用于显示 debug 信息。

```
$ sshd -d
```

(2) `-D`

`-D` 参数指定 sshd 不作为后台守护进程运行。

```
$ sshd -D
```

(3) `-e`

`-e` 参数将 sshd 写入系统日志 syslog 的内容导向标准错误（standard error）。

(4) `-f`

`-f` 参数指定配置文件的位置。

(5) `-h`

`-h` 参数用于指定密钥。

```
$ sshd -h /usr/local/ssh/my_rsa_key
```

(6) `-o`

`-o` 参数指定配置文件的一个配置项和对应的值。

```
$ sshd -o "Port 2034"
```

配置项和对应值之间，可以使用等号。

```
$ sshd -o "Port = 2034"
```

如果省略等号前后的空格，也可以不使用引号。

```
$ sshd -o Port=2034
```

`-o` 参数可以多个一起使用，用来指定多个配置关键字。

(7) `-p`

`-p` 参数指定 sshd 的服务端口。

```
$ sshd -p 2034
```

上面命令指定 sshd 在 `2034` 端口启动。

`-p` 参数可以指定多个端口。

```
$ sshd -p 2222 -p 3333
```

(8) `-t`

`-t` 参数检查配置文件的语法是否正确。

SSH 日志

SSH 在服务器端可以生成日志，记录登录当前服务器的情况。

SSH 日志是写在系统日志当中的，查看的时候需要从系统日志里面找到跟 SSH 相关的记录。

目录 [隐藏]

1. journalctl 命令
2. 其他命令
3. 日志设置

1. journalctl 命令

如果系统使用 Systemd，可以使用 `journalctl` 命令查看日志。

```
$ journalctl -u ssh

Mar 25 20:25:36 web0 sshd[14144]: Accepted publickey for ubuntu from
10.103.160.144 port 59200 ssh2: RSA
SHA256:l/zFNib1vJ+64nxLB4N9KaVhBEMf8arbWGxHQg01SW8
Mar 25 20:25:36 web0 sshd[14144]: pam_unix(sshd:session): session
opened for user ubuntu by (uid=0)
Mar 25 20:39:12 web0 sshd[14885]: pam_unix(sshd:session): session
closed for user ubuntu
...
```

上面示例中，返回的日志每一行就是一次登录尝试，按照从早到晚的顺序，其中包含了登录失败的尝试。`-u` 参数是 Unit 单元的意思，`-u ssh` 就是查看 SSH 单元，有的发行版需要写成 `-u sshd`。

`-b0` 参数可以查看自从上次登录后的日志。

```
$ journalctl -t ssh -b0
```

`-r` 参数表示逆序输出，最新的在前面。

```
$ journalctl -t ssh -b0 -r
```

`since` 和 `until` 参数可以指定日志的时间范围。

```
$ journalctl -u ssh --since yesterday # 查看昨天的日志  
$ journalctl -u ssh --since -3d --until -2d # 查看三天前的日志  
$ journalctl -u ssh --since -1h # 查看上个小时的日志  
$ journalctl -u ssh --until "2022-03-12 07:00:00" # 查看截至到某个时间点  
的日志
```

下面的命令查看实时日志。

```
$ journalctl -fu ssh
```

2. 其他命令

如果系统没有使用 Systemd，可以在 `/var/log/auth.log` 文件中找到 `sshd` 的日志。

```
$ sudo grep sshd /var/log/auth.log
```

下面的命令查看最后 500 行里面的 `sshd` 条目。

```
$ sudo tail -n 500 /var/log/auth.log | grep sshd
```

`-f` 参数可以实时跟踪日志。

```
$ sudo tail -f -n 500 /var/log/auth.log | grep sshd
```

如果只是想看谁登录了系统，而不是深入查看所有细节，可以使用 `lastlog` 命令。

```
$ lastlog
```

3. 日志设置

sshd 的配置文件 `/etc/ssh/sshd_config`，可以调整日志级别。

```
LogLevel VERBOSE
```

如果为了调试，可以将日志调整为 DEBUG。

```
LogLevel DEBUG
```

SSH 端口转发

目录 [隐藏]

- [1. 简介](#)
- [2. 动态转发](#)
- [3. 本地转发](#)
- [4. 远程转发](#)
- [5. 实例](#)
 - [5.1 简易 VPN](#)
 - [5.2 两级跳板](#)
- [6. 参考链接](#)

1. 简介

SSH 除了登录服务器，还有一大用途，就是作为加密通信的中介，充当两台服务器之间的通信加密跳板，使得原本不加密的通信变成加密通信。这个功能称为端口转发（port forwarding），又称 SSH 隧道（tunnel）。

端口转发有两个主要作用：

- (1) 将不加密的数据放在 SSH 安全连接里面传输，使得原本不安全的网络服务增加了安全性，比如通过端口转发访问 Telnet、FTP 等明文服务，数据传输就都会加密。
- (2) 作为数据通信的加密跳板，绕过网络防火墙。

端口转发有三种使用方法：动态转发，本地转发，远程转发。下面逐一介绍。

2. 动态转发

动态转发指的是，本机与 SSH 服务器之间创建了一个加密连接，然后本机内部针对某个端口的通信，都通过这个加密连接转发。它的一个使用场景就是，访问所有外部网站，都通过 SSH 转发。

动态转发需要把本地端口绑定到 SSH 服务器。至于 SSH 服务器要去访问哪一个网站，完全是动态的，取决于原始通信，所以叫做动态转发。

```
$ ssh -D local-port tunnel-host -N
```

上面命令中，`-D` 表示动态转发，`local-port` 是本地端口，`tunnel-host` 是 SSH 服务器，`-N` 表示这个 SSH 连接只进行端口转发，不登录远程 Shell，不能执行远程命令，只能充当隧道。

举例来说，如果本地端口是 `2121`，那么动态转发的命令就是下面这样。

```
$ ssh -D 2121 tunnel-host -N
```

注意，这种转发采用了 SOCKS5 协议。访问外部网站时，需要把 HTTP 请求转成 SOCKS5 协议，才能把本地端口的请求转发出去。

下面是 SSH 隧道建立后的一个使用实例。

```
$ curl -x socks5://localhost:2121 http://www.example.com
```

上面命令中，curl 的 `-x` 参数指定代理服务器，即通过 SOCKS5 协议的本地 `2121` 端口，访问 `http://www.example.com`。

如果经常使用动态转发，可以将设置写入 SSH 客户端的用户个人配置文件 (`~/.ssh/config`)。

```
DynamicForward tunnel-host:local-port
```

3. 本地转发

本地转发 (local forwarding) 指的是，创建一个本地端口，将发往该端口的所有通信都通过 SSH 服务器，转发到指定的远程服务器的端口。这种情况下，SSH 服务器只是一个作为跳板的中介，用于连接本地计算机无法直接连接的远程服务器。本地转发是在本地计算机建立的转发规则。

它的语法如下，其中会指定本地端口 (local-port)、SSH 服务器 (tunnel-host)、远程服务器 (target-host) 和远程端口 (target-port) 。

```
$ ssh -L -N -f local-port:target-host:target-port tunnel-host
```

上面命令中，有三个配置参数。

- `-L`：转发本地端口。
- `-N`：不发送任何命令，只用来建立连接。没有这个参数，会在 SSH 服务器打开一个 Shell。
- `-f`：将 SSH 连接放到后台。没有这个参数，暂时不用 SSH 连接时，终端会失去响应。

举例来说，现在有一台 SSH 服务器 `tunnel-host`，我们想要通过这台机器，在本地 `2121` 端口与目标网站 `www.example.com` 的 `80` 端口之间建立 SSH 隧道，就可以写成下面这样。

```
$ ssh -L 2121:www.example.com:80 tunnel-host -N
```

然后，访问本机的 `2121` 端口，就是访问 `www.example.com` 的 `80` 端口。

```
$ curl http://localhost:2121
```

注意，本地端口转发采用 HTTP 协议，不用转成 SOCKS5 协议。

另一个例子是加密访问邮件获取协议 POP3。

```
$ ssh -L 1100:mail.example.com:110 mail.example.com
```

上面命令将本机的 `1100` 端口，绑定邮件服务器 `mail.example.com` 的 `110` 端口（POP3 协议的默认端口）。端口转发建立以后，POP3 邮件客户端只需要访问本机的 `1100` 端口，

请求就会通过 SSH 服务器（这里是 `mail.example.com`），自动转发到 `mail.example.com` 的110端口。

上面这种情况有一个前提条件，就是 `mail.example.com` 必须运行 SSH 服务器。否则，就必须通过另一台 SSH 服务器中介，执行的命令要改成下面这样。

```
$ ssh -L 1100:mail.example.com:110 other.example.com
```

上面命令中，本机的1100端口还是绑定 `mail.example.com` 的110端口，但是由于 `mail.example.com` 没有运行 SSH 服务器，所以必须通过 `other.example.com` 中介。本机的 POP3 请求通过1100端口，先发给 `other.example.com` 的22端口（`sshd` 默认端口），再由后者转给 `mail.example.com`，得到数据以后再原路返回。

注意，采用上面的中介方式，只有本机到 `other.example.com` 的这一段是加密的，`other.example.com` 到 `mail.example.com` 的这一段并不加密。

这个命令最好加上 `-N` 参数，表示不在 SSH 跳板机执行远程命令，让 SSH 只充当隧道。另外还有一个 `-f` 参数表示 SSH 连接在后台运行。

如果经常使用本地转发，可以将设置写入 SSH 客户端的用户个人配置文件（`~/.ssh/config`）。

```
Host test.example.com
LocalForward client-IP:client-port server-IP:server-port
```

4. 远程转发

远程转发指的是在远程 SSH 服务器建立的转发规则。

它跟本地转发正好反过来。建立本地计算机到远程 SSH 服务器的隧道以后，本地转发是通过本地计算机访问远程 SSH 服务器，而远程转发则是通过远程 SSH 服务器访问本地计算机。它的命令格式如下。

```
$ ssh -R remote-port:target-host:target-port -N remotehost
```

上面命令中，`-R` 参数表示远程端口转发，`remote-port` 是远程 SSH 服务器的端口，`target-host` 和 `target-port` 是目标服务器及其端口，`remotehost` 是远程 SSH 服务器。

远程转发主要针对内网的情况。下面举两个例子。

第一个例子是内网某台服务器 `localhost` 在 80 端口开了一个服务，可以通过远程转发将这个 80 端口，映射到具有公网 IP 地址的 `my.public.server` 服务器的 8080 端口，使得访问 `my.public.server:8080` 这个地址，就可以访问到那台内网服务器的 80 端口。

```
$ ssh -R 8080:localhost:80 -N my.public.server
```

上面命令是在内网 `localhost` 服务器上执行，建立从 `localhost` 到 `my.public.server` 的 SSH 隧道。运行以后，用户访问 `my.public.server:8080`，就会自动映射到 `localhost:80`。

第二个例子是本地计算机 `local` 在外网，SSH 跳板机和目标服务器 `my.private.server` 都在内网，必须通过 SSH 跳板机才能访问目标服务器。但是，本地计算机 `local` 无法访问内网之中的 SSH 跳板机，而 SSH 跳板机可以访问本机计算机。

由于本机无法访问内网 SSH 跳板机，就无法从外网发起 SSH 隧道，建立端口转发。必须反过来，从 SSH 跳板机发起隧道，建立端口转发，这时就形成了远程端口转发。跳板机执行下面的命令，绑定本地计算机 `local` 的 `2121` 端口，去访问 `my.private.server:80`。

```
$ ssh -R 2121:my.private.server:80 -N local
```

上面命令是在 SSH 跳板机上执行的，建立跳板机到 `local` 的隧道，并且这条隧道的出口映射到 `my.private.server:80`。

显然，远程转发要求本地计算机 `local` 也安装了 SSH 服务器，这样才能接受 SSH 跳板机的远程登录。

执行上面的命令以后，跳板机到 `local` 的隧道已经建立了。然后，就可以从本地计算机访问目标服务器了，即在本机执行下面的命令。

```
$ curl http://localhost:2121
```

本机执行上面的命令以后，就会输出服务器 `my.private.server` 的 80 端口返回的内容。

如果经常执行远程端口转发，可以将设置写入 SSH 客户端的用户个人配置文件 (`~/.ssh/config`)。

```
Host remote-forward
HostName test.example.com
RemoteForward remote-port target-host:target-port
```

完成上面的设置后，执行下面的命令就会建立远程转发。

```
$ ssh -N remote-forward
# 等同于
$ ssh -R remote-port:target-host:target-port -N test.example.com
```

5. 实例

下面看两个端口转发的实例。

5.1 简易 VPN

VPN 用来在外网与内网之间建立一条加密通道。内网的服务器不能从外网直接访问，必须通过一个跳板机，如果本机可以访问跳板机，就可以使用 SSH 本地转发，简单实现一个 VPN。

```
$ ssh -L 2080:corp-server:80 -L 2443:corp-server:443 tunnel-host -N
```

上面命令通过 SSH 跳板机，将本机的 2080 端口绑定内网服务器的 80 端口，本机的 2443 端口绑定内网服务器的 443 端口。

5.2 两级跳板

端口转发可以有多级，比如新建两个 SSH 隧道，第一个隧道转发给第二个隧道，第二个隧道才能访问目标服务器。

首先，在本机新建第一级隧道。

```
$ ssh -L 7999:localhost:2999 tunnell-host
```

上面命令在本地 7999 端口与 tunnell-host 之间建立一条隧道，隧道的出口是 tunnell-host 的 localhost:2999，也就是 tunnell-host 收到本机的请求以后，转发给自己的 2999 端口。

然后，在第一台跳板机（tunnell-host）执行下面的命令，新建第二级隧道。

```
$ ssh -L 2999:target-host:7999 tunnel2-host -N
```

上面命令将第一台跳板机 tunnell-host 的 2999 端口，通过第二台跳板机 tunnel2-host，连接到目标服务器 target-host 的 7999 端口。

最终效果就是，访问本机的 7999 端口，就会转发到 target-host 的 7999 端口。

6. 参考链接

- [An Illustrated Guide to SSH Tunnels](#), Scott Wiersdorf
- [An Excruciatingly Detailed Guide To SSH](#), Graham Helton

SSH 证书登录

SSH 是服务器登录工具，一般情况下都采用密码登录或密钥登录。

但是，SSH 还有第三种登录方法，那就是证书登录。某些情况下，它是更合理、更安全的登录方法，本文就介绍这种登录方法。

目录 [隐藏]

1. 非证书登录的缺点
2. 证书登录是什么？
3. 证书登录的流程
4. 生成 CA 的密钥
5. CA 签发服务器证书
6. CA 签发用户证书
7. 服务器安装证书
8. 服务器安装 CA 公钥
9. 客户端安装证书
10. 客户端安装 CA 公钥
11. 废除证书
12. 参考链接

1. 非证书登录的缺点

密码登录和密钥登录，都有各自的缺点。

密码登录需要输入服务器密码，这非常麻烦，也不安全，存在被暴力破解的风险。

密钥登录需要服务器保存用户的公钥，也需要用户保存服务器公钥的指纹。这对于多用户、多服务器的大型机构很不方便，如果有员工离职，需要将他的公钥从每台服务器删除。

2. 证书登录是什么？

证书登录就是为了解决上面的缺点而设计的。它引入了一个证书颁发机构（Certificate Authority，简称 CA），对信任的服务器颁发服务器证书，对信任的用户颁发用户证书。

登录时，用户和服务器不需要提前知道彼此的公钥，只需要交换各自的证书，验证是否可信即可。

证书登录的主要优点有两个：（1）用户和服务器不用交换公钥，这更容易管理，也具有更好的可扩展性。（2）证书可以设置到期时间，而公钥没有到期时间。针对不同的情况，可以设置有效期很短的证书，进一步提高安全性。

3. 证书登录的流程

SSH 证书登录之前，如果还没有证书，需要生成证书。具体方法是：（1）用户和服务器都将自己的公钥，发给 CA；（2）CA 使用服务器公钥，生成服务器证书，发给服务器；（3）CA 使用用户的公钥，生成用户证书，发给用户。

有了证书以后，用户就可以登录服务器了。整个过程都是 SSH 自动处理，用户无感知。

第一步，用户登录服务器时，SSH 自动将用户证书发给服务器。

第二步，服务器检查用户证书是否有效，以及是否由可信的 CA 颁发。证实以后，就可以信任用户。

第三步，SSH 自动将服务器证书发给用户。

第四步，用户检查服务器证书是否有效，以及是否由信任的 CA 颁发。证实以后，就可以信任服务器。

第五步，双方建立连接，服务器允许用户登录。

4. 生成 CA 的密钥

证书登录的前提是，必须有一个 CA，而 CA 本质上就是一对密钥，跟其他密钥没有不同，CA 就用这对密钥去签发证书。

虽然 CA 可以用同一对密钥签发用户证书和服务器证书，但是出于安全性和灵活性，最好用不同的密钥分别签发。所以，CA 至少需要两对密钥，一对是签发用户证书的密钥，假设叫做 `user_ca`，另一对是签发服务器证书的密钥，假设叫做 `host_ca`。

使用下面的命令，生成 `user_ca`。

```
# 生成 CA 签发用户证书的密钥
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/user_ca -C user_ca
```

上面的命令会在 `~/.ssh` 目录生成一对密钥：`user_ca`（私钥）和 `user_ca.pub`（公钥）。

这个命令的各个参数含义如下。

- `-t rsa`：指定密钥算法 RSA。
- `-b 4096`：指定密钥的位数是4096位。安全性要求不高的场合，这个值可以小一点，但是不应小于1024。
- `-f ~/.ssh/user_ca`：指定生成密钥的位置和文件名。
- `-C user_ca`：指定密钥的识别字符串，相当于注释，可以随意设置。

使用下面的命令，生成 `host_ca`。

```
# 生成 CA 签发服务器证书的密钥
$ ssh-keygen -t rsa -b 4096 -f host_ca -C host_ca
```

上面的命令会在 `~/.ssh` 目录生成一对密钥：`host_ca`（私钥）和 `host_ca.pub`（公钥）。

现在，`~/.ssh` 目录应该至少有四把密钥。

- `~/.ssh/user_ca`
- `~/.ssh/user_ca.pub`
- `~/.ssh/host_ca`
- `~/.ssh/host_ca.pub`

5. CA 签发服务器证书

有了 CA 以后，就可以签发服务器证书了。

签发证书，除了 CA 的密钥以外，还需要服务器的公钥。一般来说，SSH 服务器（通常是 `sshd`）安装时，已经生成密钥 `/etc/ssh/ssh_host_rsa_key` 了。如果没有的话，可以用下面的命令生成。

```
$ sudo ssh-keygen -f /etc/ssh/ssh_host_rsa_key -b 4096 -t rsa
```

上面命令会在 `/etc/ssh` 目录，生成 `ssh_host_rsa_key`（私钥）和 `ssh_host_rsa_key.pub`（公钥）。然后，需要把服务器公钥 `ssh_host_rsa_key.pub`，复制或上传到 CA 所在的服务器。

上传以后，CA 就可以使用密钥 `host_ca` 为服务器的公钥 `ssh_host_rsa_key.pub` 签发服务器证书。

```
$ ssh-keygen -s host_ca -I host.example.com -h -n host.example.com -V +52w ssh_host_rsa_key.pub
```

上面的命令会生成服务器证书 `ssh_host_rsa_key-cert.pub`（服务器公钥名字加后缀 `-cert`）。这个命令各个参数的含义如下。

- `-s`：指定 CA 签发证书的密钥。
- `-I`：身份字符串，可以随便设置，相当于注释，方便区分证书，将来可以使用这个字符串撤销证书。
- `-h`：指定该证书是服务器证书，而不是用户证书。
- `-n host.example.com`：指定服务器的域名，表示证书仅对该域名有效。如果有多个域名，则使用逗号分隔。用户登录该域名服务器时，SSH 通过证书的这个值，分辨应该使用哪张证书发给用户，用来证明服务器的可信性。
- `-V +52w`：指定证书的有效期，这里为52周（一年）。默认情况下，证书是永远有效的。建议使用该参数指定有效期，并且有效期最好短一点，最长不超过52周。
- `ssh_host_rsa_key.pub`：服务器公钥。

生成证书以后，可以使用下面的命令，查看证书的细节。

```
$ ssh-keygen -L -f ssh_host_rsa_key-cert.pub
```

最后，为证书设置权限。

```
$ chmod 600 ssh_host_rsa_key-cert.pub
```

6. CA 签发用户证书

下面，再用 CA 签发用户证书。这时需要用户的公钥，如果没有的话，客户端可以用下面的命令生成一对密钥。

```
$ ssh-keygen -f ~/.ssh/user_key -b 4096 -t rsa
```

上面命令会在 `~/.ssh` 目录，生成 `user_key`（私钥）和 `user_key.pub`（公钥）。

然后，将用户公钥 `user_key.pub`，上传或复制到 CA 服务器。接下来，就可以使用 CA 的密钥 `user_ca` 为用户公钥 `user_key.pub` 签发用户证书。

```
$ ssh-keygen -s user_ca -I user@example.com -n user -V +1d user_key.pub
```

上面的命令会生成用户证书 `user_key-cert.pub`（用户公钥名字加后缀 `-cert`）。这个命令各个参数的含义如下。

- `-s`：指定 CA 签发证书的密钥
- `-I`：身份字符串，可以随便设置，相当于注释，方便区分证书，将来可以使用这个字符串撤销证书。
- `-n user`：指定用户名，表示证书仅对该用户名有效。如果有多个用户名，使用逗号分隔。用户以该用户名登录服务器时，SSH 通过这个值，分辨应该使用哪张证书，证明自己的身份，发给服务器。
- `-V +1d`：指定证书的有效期，这里为1天，强制用户每天都申请一次证书，提高安全性。默认情况下，证书是永远有效的。
- `user_key.pub`：用户公钥。

生成证书以后，可以使用下面的命令，查看证书的细节。

```
$ ssh-keygen -L -f user_key-cert.pub
```

最后，为证书设置权限。

```
$ chmod 600 user_key-cert.pub
```

7. 服务器安装证书

CA 生成服务器证书 `ssh_host_rsa_key-cert.pub` 以后，需要将该证书发回服务器，可以使用下面的 `scp` 命令，将证书拷贝过去。

```
$ scp ~/.ssh/ssh_host_rsa_key-cert.pub root@host.example.com:/etc/ssh/
```

然后，将下面一行添加到服务器配置文件 `/etc/ssh/sshd_config`。

```
HostCertificate /etc/ssh/ssh_host_rsa_key-cert.pub
```

上面的代码告诉 sshd，服务器证书是哪一个文件。

重新启动 sshd。

```
$ sudo systemctl restart sshd.service
# 或者
$ sudo service sshd restart
```

8. 服务器安装 CA 公钥

为了让服务器信任用户证书，必须将 CA 签发用户证书的公钥 `user_ca.pub`，拷贝到服务器。

```
$ scp ~/.ssh/user_ca.pub root@host.example.com:/etc/ssh/
```

上面的命令，将 CA 签发用户证书的公钥 `user_ca.pub`，拷贝到 SSH 服务器的 `/etc/ssh` 目录。

然后，将下面一行添加到服务器配置文件 `/etc/ssh/sshd_config`。

```
TrustedUserCAKeys /etc/ssh/user_ca.pub
```

上面的做法是将 `user_ca.pub` 加到 `/etc/ssh/sshd_config`，这会产生全局效果，即服务器的所有账户都会信任 `user_ca` 签发的所有用户证书。

另一种做法是将 `user_ca.pub` 加到服务器某个账户的 `~/.ssh/authorized_keys` 文件，只让该账户信任 `user_ca` 签发的用户证书。具体方法是打开 `~/.ssh/authorized_keys`，追加一行，开头是 `@cert-authority principals="..."`，然后后面加上 `user_ca.pub` 的内容，大概是下面这个样子。

```
@cert-authority principals="user" ssh-rsa AAAAB3Nz...XNRM1EX2gQ==
```

上面代码中，`principals="user"` 指定用户登录的服务器账户名，一般就是 `authorized_keys` 文件所在的账户。

重新启动 `sshd`。

```
$ sudo systemctl restart sshd.service
# 或者
$ sudo service sshd restart
```

至此，SSH 服务器已配置为信任 `user_ca` 签发的证书。

9. 客户端安装证书

客户端安装用户证书很简单，就是从 CA 将用户证书 `user_key-cert.pub` 复制到客户端，与用户的密钥 `user_key` 保存在同一个目录即可。

10. 客户端安装 CA 公钥

为了让客户端信任服务器证书，必须将 CA 签发服务器证书的公钥 `host_ca.pub`，加到客户端的 `/etc/ssh/ssh_known_hosts` 文件（全局级别）或者 `~/.ssh/known_hosts` 文件（用户级别）。

具体做法是打开 `ssh_known_hosts` 或 `known_hosts` 文件，追加一行，开头为 `@cert-authority *.example.com`，然后将 `host_ca.pub` 文件的内容（即公钥）粘贴在后面，大概是下面这个样子。

```
@cert-authority *.example.com ssh-rsa AAAAB3Nz...XNRM1EX2gQ==
```

上面代码中，`*.example.com` 是域名的模式匹配，表示只要服务器符合该模式的域名，且签发服务器证书的 CA 匹配后面给出的公钥，就都可以信任。如果没有域名限制，这里可以写成`*`。如果有多个域名模式，可以使用逗号分隔；如果服务器没有域名，可以用主机名（比如`host1, host2, host3`）或者 IP 地址（比如`11.12.13.14, 21.22.23.24`）。

然后，就可以使用证书，登录远程服务器了。

```
$ ssh -i ~/.ssh/user_key user@host.example.com
```

上面命令的`-i` 参数用来指定用户的密钥。如果证书与密钥在同一个目录，则连接服务器时将自动使用该证书。

11. 废除证书

废除证书的操作，分成用户证书的废除和服务器证书的废除两种。

服务器证书的废除，用户需要在`known_hosts` 文件里面，修改或删除对应的`@cert-authority` 命令的那一行。

用户证书的废除，需要在服务器新建一个`/etc/ssh/revoked_keys` 文件，然后在配置文件`sshd_config` 添加一行，内容如下。

```
RevokedKeys /etc/ssh/revoked_keys
```

`revoked_keys` 文件保存不再信任的用户公钥，由下面的命令生成。

```
$ ssh-keygen -kf /etc/ssh/revoked_keys -z 1 ~/.ssh/user1_key.pub
```

上面命令中，`-z` 参数用来指定用户公钥保存在`revoked_keys` 文件的哪一行，这个例子是保存在第1行。

如果以后需要废除其他的用户公钥，可以用下面的命令保存在第2行。

```
$ ssh-keygen -ukf /etc/ssh/revoked_keys -z 2 ~/.ssh/user2_key.pub
```

12. 参考链接

- [SSH Emergency Access](#), Carl Tashian
- [Using OpenSSH Certificate Authentication](#), Red Hat Enterprise Linux Deployment Guide
- [How to SSH Properly](#), Gus Luxton

scp 命令

`scp` 是 SSH 提供的一个客户端程序，用来在两台主机之间加密传送文件（即复制文件）。

目录 [隐藏]

1. 简介
2. 基本语法
3. 用法示例
4. 配置项

1. 简介

`scp` 是 secure copy 的缩写，相当于 `cp` 命令 + SSH。它的底层是 SSH 协议，默认端口是22，相当于先使用 `ssh` 命令登录远程主机，然后再执行拷贝操作。

`scp` 主要用于以下三种复制操作。

- 本地复制到远程。
- 远程复制到本地。
- 两个远程系统之间的复制。

使用 `scp` 传输数据时，文件和密码都是加密的，不会泄漏敏感信息。

2. 基本语法

`scp` 的语法类似 `cp` 的语法。

```
$ scp source destination
```

上面命令中，`source` 是文件当前的位置，`destination` 是文件所要复制到的位置。它们都可以包含用户名和主机名。

```
$ scp user@host:foo.txt bar.txt
```

上面命令将远程主机（`user@host`）用户主目录下的 `foo.txt`，复制为本机当前目录的 `bar.txt`。可以看到，主机与文件之间要使用冒号（`:`）分隔。

`scp` 会先用 SSH 登录到远程主机，然后在加密连接之中复制文件。客户端发起连接后，会提示用户输入密码，这部分是跟 SSH 的用法一致的。

用户名和主机名都是可以省略的。用户名的默认值是本机的当前用户名，主机名默认为当前主机。注意，`scp` 会使用 SSH 客户端的配置文件 `.ssh/config`，如果配置文件里面定义了主机的别名，这里也可以使用别名连接。

`scp` 支持一次复制多个文件。

```
$ scp source1 source2 destination
```

上面命令会将 `source1` 和 `source2` 两个文件，复制到 `destination`。

注意，如果所要复制的文件，在目标位置已经存在同名文件，`scp` 会在没有警告的情况下覆盖同名文件。

3. 用法示例

(1) 本地文件复制到远程

复制本机文件到远程系统的用法如下。

```
# 语法
$ scp SourceFile user@host:directory/TargetFile

# 示例
$ scp file.txt remote_username@10.10.0.2:/remote/directory
```

下面是复制整个目录的例子。

```
# 将本机的 documents 目录拷贝到远程主机,
# 会在远程主机创建 documents 目录
$ scp -r documents username@server_ip:/path_to_remote_directory

# 将本机整个目录拷贝到远程目录下
$ scp -r localmachine/path_to_the_directory
username@server_ip:/path_to_remote_directory/

# 将本机目录下的所有内容拷贝到远程目录下
$ scp -r localmachine/path_to_the_directory/*
username@server_ip:/path_to_remote_directory/
```

(2) 远程文件复制到本地

从远程主机复制文件到本地的用法如下。

```
# 语法
$ scp user@host:directory/SourceFile TargetFile

# 示例
$ scp remote_username@10.10.0.2:/remote/file.txt /local/directory
```

下面是复制整个目录的例子。

```
# 拷贝一个远程目录到本机目录下
$ scp -r username@server_ip:/path_to_remote_directory local-
machine/path_to_the_directory/

# 拷贝远程目录下的所有内容, 到本机目录下
$ scp -r username@server_ip:/path_to_remote_directory/* local-
machine/path_to_the_directory/
$ scp -r user@host:directory/SourceFolder TargetFolder
```

(3) 两个远程系统之间的复制

本机发出指令, 从远程主机 A 拷贝到远程主机 B 的用法如下。

```
# 语法
$ scp user@host1:directory/SourceFile user@host2:directory/SourceFile

# 示例
$ scp user1@host1.com:/files/file.txt user2@host2.com:/files
```

系统将提示你输入两个远程帐户的密码。数据将直接从一个远程主机传输到另一个远程主机。

4. 配置项

(1) `-c`

`-c` 参数用来指定文件拷贝数据传输的加密算法。

```
$ scp -c blowfish some_file your_username@remotehost.edu:~
```

上面代码指定加密算法为 `blowfish`。

(2) `-C`

`-C` 参数表示是否在传输时压缩文件。

```
$ scp -c blowfish -C local_file your_username@remotehost.edu:~
```

(3) `-F`

`-F` 参数用来指定 `ssh_config` 文件，供 `ssh` 使用。

```
$ scp -F /home/pungki/proxy_ssh_config Label.pdf root@172.20.10.8:/root
```

(4) `-i`

`-i` 参数用来指定密钥。

```
$ scp -vCq -i private_key.pem ~/test.txt
root@192.168.1.3:/some/path/test.txt
```

(5) -l

-l 参数用来限制传输数据的带宽速率，单位是 Kbit/sec。对于多人分享的带宽，这个参数可以留出一部分带宽供其他人使用。

```
$ scp -l 80 yourusername@yourserver:/home/yourusername/* .
```

上面代码中，scp 命令占用的带宽限制为每秒 80K 比特位，即每秒 10K 字节。

(6) -p

-p 参数用来保留修改时间 (modification time)、访问时间 (access time)、文件状态 (mode) 等原始文件的信息。

```
$ scp -p ~/test.txt root@192.168.1.3:/some/path/test.txt
```

(7) -P

-P 参数用来指定远程主机的 SSH 端口。如果远程主机使用默认端口22，可以不用指定，否则需要用-P 参数在命令中指定。

```
$ scp -P 2222 user@host:directory/SourceFile TargetFile
```

(8) -q

-q 参数用来关闭显示拷贝的进度条。

```
$ scp -q Label.pdf mrianto@202.x.x.x:.
```

(9) -r

-r 参数表示是否以递归方式复制目录。

(10) -v

-v 参数用来显示详细的输出。

```
$ scp -v ~/test.txt root@192.168.1.3:/root/help2356.txt
```


rsync 命令

目录 [隐藏]

- 1. 简介
- 2. 安装
- 3. 基本用法
 - 3.1 `-r` 参数
 - 3.2 `-a` 参数
 - 3.3 `-n` 参数
 - 3.4 `--delete` 参数
- 4. 排除文件
 - 4.1 `--exclude` 参数
 - 4.2 `--include` 参数
- 5. 远程同步
 - 5.1 SSH 协议
 - 5.2 rsync 协议
- 6. 增量备份
- 7. 配置项
- 8. 参考链接

1. 简介

rsync 是一个常用的 Linux 应用程序，用于文件同步。

它可以在本地计算机与远程计算机之间，或者两个本地目录之间同步文件（但不支持两台远程计算机之间的同步）。它也可以当作文件复制工具，替代 `cp` 和 `mv` 命令。

它名称里面的 `r` 指的是 remote, rsync 其实就是“远程同步”(remote sync)的意思。与其他文件传输工具(如 FTP 或 scp)不同, rsync 的最大特点是会检查发送方和接收方已有的文件,仅传输有变动的部分(默认规则是文件大小或修改时间有变动)。

虽然 rsync 不是 SSH 工具集的一部分,但因为也涉及到远程操作,所以放在这里一起介绍。

2. 安装

如果本机或者远程计算机没有安装 rsync,可以用下面的命令安装。

```
# Debian
$ sudo apt-get install rsync

# Red Hat
$ sudo yum install rsync

# Arch Linux
$ sudo pacman -S rsync
```

注意,传输的双方都必须安装 rsync。

3. 基本用法

rsync 可以用于本地计算机的两个目录之间的同步。下面就用本地同步举例,顺便讲解 rsync 几个主要参数的用法。

3.1 `-r` 参数

本机使用 rsync 命令时,可以作为 `cp` 和 `mv` 命令的替代方法,将源目录拷贝到目标目录。

```
$ rsync -r source destination
```

上面命令中，`-r` 表示递归，即包含子目录。注意，`-r` 是必须的，否则 rsync 运行不会成功。`source` 目录表示源目录，`destination` 表示目标目录。上面命令执行以后，目标目录下就会出现 `destination/source` 这个子目录。

如果有多个文件或目录需要同步，可以写成下面这样。

```
$ rsync -r source1 source2 destination
```

上面命令中，`source1`、`source2` 都会被同步到 `destination` 目录。

3.2 `-a` 参数

`-a` 参数可以替代 `-r`，除了可以递归同步以外，还可以同步元信息（比如修改时间、权限等）。由于 rsync 默认使用文件大小和修改时间决定文件是否需要更新，所以 `-a` 比 `-r` 更有用。下面的用法才是常见的写法。

```
$ rsync -a source destination
```

目标目录 `destination` 如果不存在，rsync 会自动创建。执行上面的命令后，源目录 `source` 被完整地复制到了目标目录 `destination` 下面，即形成了 `destination/source` 的目录结构。

如果只想同步源目录 `source` 里面的内容到目标目录 `destination`，则需要在源目录后面加上斜杠。

```
$ rsync -a source/ destination
```

上面命令执行后，`source` 目录里面的内容，就都被复制到了 `destination` 目录里面，并不会在 `destination` 下面创建一个 `source` 子目录。

3.3 `-n` 参数

如果不确定 rsync 执行后会产生什么结果，可以先用 `-n` 或 `--dry-run` 参数模拟执行的结果。

```
$ rsync -anv source/ destination
```

上面命令中，`-n` 参数模拟命令执行的结果，并不真的执行命令。`-v` 参数则是将结果输出到终端，这样就可以看到哪些内容会被同步。

3.4 `--delete` 参数

默认情况下，rsync 只确保源目录的所有内容（明确排除的文件除外）都复制到目标目录。它不会使两个目录保持相同，并且不会删除文件。如果要使得目标目录成为源目录的镜像副本，则必须使用`--delete`参数，这将删除只存在于目标目录、不存在于源目录的文件。

```
$ rsync -av --delete source/ destination
```

上面命令中，`--delete` 参数会使得`destination` 成为`source` 的一个镜像。

4. 排除文件

4.1 `--exclude` 参数

有时，我们希望同步时排除某些文件或目录，这时可以用`--exclude` 参数指定排除模式。

```
$ rsync -av --exclude='*.txt' source/ destination
# 或者
$ rsync -av --exclude '*.txt' source/ destination
```

上面命令排除了所有 TXT 文件。

注意，rsync 会同步以“点”开头的隐藏文件，如果要排除隐藏文件，可以这样写`--exclude="./*"`。

如果要排除某个目录里面的所有文件，但不希望排除目录本身，可以写成下面这样。

```
$ rsync -av --exclude 'dir1/*' source/ destination
```

多个排除模式，可以用多个`--exclude` 参数。

```
$ rsync -av --exclude 'file1.txt' --exclude 'dir1/*' source/ destination
```

多个排除模式也可以利用 Bash 的大扩号的扩展功能，只用一个 `--exclude` 参数。

```
$ rsync -av --exclude={'file1.txt','dir1/*'} source/ destination
```

如果排除模式很多，可以将它们写入一个文件，每个模式一行，然后用 `--exclude-from` 参数指定这个文件。

```
$ rsync -av --exclude-from='exclude-file.txt' source/ destination
```

4.2 `--include` 参数

`--include` 参数用来指定必须同步的文件模式，往往与 `--exclude` 结合使用。

```
$ rsync -av --include="*.txt" --exclude='*' source/ destination
```

上面命令指定同步时，排除所有文件，但是会包括 TXT 文件。

5. 远程同步

5.1 SSH 协议

rsync 除了支持本地两个目录之间的同步，也支持远程同步。它可以将本地内容，同步到远程服务器。

```
$ rsync -av source/ username@remote_host:destination
```

也可以将远程内容同步到本地。

```
$ rsync -av username@remote_host:source/ destination
```

rsync 默认使用 SSH 进行远程登录和数据传输。

由于早期 rsync 不使用 SSH 协议，需要用 `-e` 参数指定协议，后来才改的。所以，下面 `-e ssh` 可以省略。

```
$ rsync -av -e ssh source/ user@remote_host:/destination
```

但是，如果 ssh 命令有附加的参数，则必须使用 `-e` 参数指定所要执行的 SSH 命令。

```
$ rsync -av -e 'ssh -p 2234' source/ user@remote_host:/destination
```

上面命令中，`-e` 参数指定 SSH 使用2234端口。

5.2 rsync 协议

除了使用 SSH，如果另一台服务器安装并运行了 rsync 守护程序，则也可以用 `rsync://` 协议（默认端口873）进行传输。具体写法是服务器与目标目录之间使用双冒号分隔 `::`。

```
$ rsync -av source/ 192.168.122.32::module/destination
```

注意，上面地址中的 `module` 并不是实际路径名，而是 rsync 守护程序指定的一个资源名，由管理员分配。

如果想知道 rsync 守护程序分配的所有 module 列表，可以执行下面命令。

```
$ rsync rsync://192.168.122.32
```

rsync 协议除了使用双冒号，也可以直接用 `rsync://` 协议指定地址。

```
$ rsync -av source/ rsync://192.168.122.32/module/destination
```

6. 增量备份

rsync 的最大特点就是它可以完成增量备份，也就是默认只复制有变动的文件。

除了源目录与目标目录直接比较，rsync 还支持使用基准目录，即将源目录与基准目录之间变动的部分，同步到目标目录。

具体做法是，第一次同步是全量备份，所有文件在基准目录里面同步一份。以后每一次同步都是增量备份，只同步源目录与基准目录之间有变动的部分，将这部分保存在一个新的目标目录。这个新的目标目录之中，也是包含所有文件，但实际上，只有那些变动过的文件是存在于该目录，其他没有变动的文件都是指向基准目录文件的硬链接。

--link-dest 参数用来指定同步时的基准目录。

```
$ rsync -a --delete --link-dest /compare/path /source/path /target/path
```

上面命令中，--link-dest 参数指定基准目录 /compare/path，然后源目录 /source/path 跟基准目录进行比较，找出变动的文件，将它们拷贝到目标目录 /target/path。那些没变动的文件则会生成硬链接。这个命令的第一次备份时是全量备份，后面就都是增量备份了。

下面是一个脚本示例，备份用户的主目录。

```

#!/bin/bash

# A script to perform incremental backups using rsync

set -o errexit
set -o nounset
set -o pipefail

readonly SOURCE_DIR="${HOME}"
readonly BACKUP_DIR="/mnt/data/backups"
readonly DATETIME=$(date '+%Y-%m-%d_%H:%M:%S')
readonly BACKUP_PATH="${BACKUP_DIR}/${DATETIME}"
readonly LATEST_LINK="${BACKUP_DIR}/latest"

mkdir -p "${BACKUP_DIR}"

rsync -av --delete \
"${SOURCE_DIR}/" \
--link-dest "${LATEST_LINK}" \
--exclude=".cache" \
"${BACKUP_PATH}"

rm -rf "${LATEST_LINK}"
ln -s "${BACKUP_PATH}" "${LATEST_LINK}"

```

上面脚本中，每一次同步都会生成一个新目录 `${BACKUP_DIR} / ${DATETIME}`，并将软链接 `${BACKUP_DIR} / latest` 指向这个目录。下一次备份时，就将 `${BACKUP_DIR} / latest` 作为基准目录，生成新的备份目录。最后，再将软链接 `${BACKUP_DIR} / latest` 指向新的备份目录。

7. 配置项

`-a`、`--archive` 参数表示存档模式，保存所有的元数据，比如修改时间 (modification time)、权限、所有者等，并且软链接也会同步过去。

`--append` 参数指定文件接着上次中断的地方，继续传输。

`--append-verify` 参数跟 `--append` 参数类似，但会对传输完成后的文件进行一次校验。如果校验失败，将重新发送整个文件。

`-b`、`--backup` 参数指定在删除或更新目标目录已经存在的文件时，将该文件更名后进行备份，默认行为是删除。更名规则是添加由 `--suffix` 参数指定的文件后缀名，默认是 `~`。

`--backup-dir` 参数指定文件备份时存放的目录，比如 `--backup-dir=/path/to/backups`。

`--bwlimit` 参数指定带宽限制，默认单位是 KB/s，比如 `--bwlimit=100`。

`-c`、`--checksum` 参数改变 `rsync` 的校验方式。默认情况下，`rsync` 只检查文件的大小和最后修改日期是否发生变化，如果发生变化，就重新传输；使用这个参数以后，则通过判断文件内容的校验和，决定是否重新传输。

`--delete` 参数删除只存在于目标目录、不存在于源目录的文件，即保证目标目录是源目录的镜像。

`-e` 参数指定使用 SSH 协议传输数据。

`--exclude` 参数指定排除不进行同步的文件，比如 `--exclude="*.iso"`。

`--exclude-from` 参数指定一个本地文件，里面是需要排除的文件模式，每个模式一行。

`--existing`、`--ignore-non-existing` 参数表示不同步目标目录中不存在的文件和目录。

`-h` 参数表示以人类可读的格式输出。

`-h`、`--help` 参数返回帮助信息。

`-i` 参数表示输出源目录与目标目录之间文件差异的详细情况。

`--ignore-existing` 参数表示只要该文件在目标目录中已经存在，就跳过去，不再同步这些文件。

`--include` 参数指定同步时要包括的文件，一般与 `--exclude` 结合使用。

`--link-dest` 参数指定增量备份的基准目录。

`-m` 参数指定不同步空目录。

`--max-size` 参数设置传输的最大文件的大小限制，比如不超过200KB (`--max-size='200k'`)。

`--min-size` 参数设置传输的最小文件的大小限制，比如不小于10KB（`--min-size=10k`）。

`-n` 参数或 `--dry-run` 参数模拟将要执行的操作，而并不真的执行。配合 `-v` 参数使用，可以看到哪些内容会被同步过去。

`-P` 参数是 `--progress` 和 `--partial` 这两个参数的结合。

`--partial` 参数允许恢复中断的传输。不使用该参数时，`rsync` 会删除传输到一半被打断的文件；使用该参数后，传输到一半的文件也会同步到目标目录，下次同步时再恢复中断的传输。一般需要与 `--append` 或 `--append-verify` 配合使用。

`--partial-dir` 参数指定将传输到一半的文件保存到一个临时目录，比如 `--partial-dir=.rsync-partial`。一般需要与 `--append` 或 `--append-verify` 配合使用。

`--progress` 参数表示显示进展。

`-r` 参数表示递归，即包含子目录。

`--remove-source-files` 参数表示传输成功后，删除发送方的文件。

`--size-only` 参数表示只同步大小有变化的文件，不考虑文件修改时间的差异。

`--suffix` 参数指定文件名备份时，对文件名添加的后缀，默认是 `~`。

`-u`、`--update` 参数表示同步时跳过目标目录中修改时间更新的文件，即不同步这些有更新的时间戳的文件。

`-v` 参数表示输出细节。`-vv` 表示输出更详细的信息，`-vvv` 表示输出最详细的信息。

`--version` 参数返回 `rsync` 的版本。

`-z` 参数指定同步时压缩数据。

8. 参考链接

- [How To Use Rsync to Sync Local and Remote Directories on a VPS, Justin Ellingwood](#)
- [Mirror Your Web Site With rsync, Falko Timme](#)
- [Examples on how to use Rsync, Egidio Docile](#)
- [How to create incremental backups using rsync on Linux, Egidio Docile](#)

sftp 命令

`sftp` 是 SSH 提供的一个客户端应用程序，主要用来安全地访问 FTP。因为 FTP 是不加密协议，很不安全，`sftp` 就相当于将 FTP 放入了 SSH。

下面的命令连接 FTP 主机。

```
$ sftp username@hostname
```

执行上面的命令，会要求输入 FTP 的密码。密码验证成功以后，就会出现 FTP 的提示符 `sftp>`，下面是一个例子。

```
$ sftp USER@penguin.example.com
USER@penguin.example.com's password:
Connected to penguin.example.com.
sftp>
```

FTP 的提示符下面，就可以输入各种 FTP 命令了，这部分完全跟传统的 FTP 用法完全一样。

- `ls [directory]`：列出一个远程目录的内容。如果没有指定目标目录，则默认列出当前目录。
- `cd directory`：从当前目录改到指定目录。
- `mkdir directory`：创建一个远程目录。
- `rmdir path`：删除一个远程目录。
- `put localfile [remotefile]`：本地文件传输到远程主机。
- `get remotefile [localfile]`：远程文件传输到本地。
- `help`：显示帮助信息。
- `bye`：退出 sftp。
- `quit`：退出 sftp。
- `exit`：退出 sftp。

Fail2Ban 教程

目录 [隐藏]

- 1. 简介
- 2. fail2ban-client
- 3. 配置
 - 3.1 主配置文件
 - 3.2 封禁配置
 - 3.3 配置项
 - 3.4 ssh 配置

1. 简介

Fail2Ban 是一个 Linux 系统的应用软件，用来防止系统入侵，主要是防止暴力破解系统密码。它是用 Python 开发的。

它主要通过监控日志文件（比如 `/var/log/auth.log`、`/var/log/apache/access.log` 等）来生效。一旦发现恶意攻击的登录请求，它会封锁对方的 IP 地址，使得对方无法再发起请求。

Fail2Ban 可以防止有人反复尝试 SSH 密码登录，但是如果 SSH 采用的是密钥登录，禁止了密码登录，就不需要 Fail2Ban 来保护。

Fail2Ban 的安装命令如下。

```
# ubuntu & Debian
$ sudo apt install fail2ban

# Fedora
$ sudo dnf install epel-release
$ sudo dnf install fail2ban

# Centos & Red hat
$ yum install fail2ban
```

安装后，使用下面的命令查看 Fail2Ban 的状态。

```
$ systemctl status fail2ban.service
```

如果没有启动，就启动 Fail2Ban。

```
$ sudo systemctl start fail2ban
```

重新启动 Fail2Ban。

```
$ sudo systemctl restart fail2ban
```

设置 Fail2Ban 重启后自动运行。

```
$ sudo systemctl enable fail2ban
```

2. fail2ban-client

Fail2Ban 自带一个客户端 fail2ban-client，用来操作 Fail2Ban。

```
$ fail2ban-client
```

上面的命令会输出 fail2ban-client 所有的用法。

下面的命令查看激活的监控目标。

```
$ fail2ban-client status

Status
|- Number of jail:    1
`- Jail list: sshd
```

下面的命令查看某个监控目标（这里是 sshd）的运行情况。

```
$ sudo fail2ban-client status sshd

Status for the jail: sshd
|- Filter
| |- Currently failed: 1
| |- Total failed:    9
| `-. Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 1
  |- Total banned:     1
  `-. Banned IP list:   0.0.0.0
```

下面的命令输出一个简要的版本，包括所有监控目标被封的 IP 地址。

```
$ sudo fail2ban-client banned
[{'sshd': ['192.168.100.50']}, {'apache-auth': []}]
```

下面的命令可以解封某个 IP 地址。

```
$ sudo fail2ban-client set sshd unbanip 192.168.1.69
```

3. 配置

3.1 主配置文件

Fail2Ban 主配置文件是在 `/etc/fail2ban/fail2ban.conf`，可以新建一份副本 `/etc/fail2ban/fail2ban.local`，修改都针对副本。

```
$ sudo cp /etc/fail2ban/fail2ban.conf /etc/fail2ban/fail2ban.local
```

下面是设置 Fail2Ban 的日志位置。

```
[Definition]
logtarget = /var/log/fail2ban/fail2ban.log
```

修改配置以后，需要重新启动 `fail2ban.service`，让其生效。

3.2 封禁配置

Fail2Ban 封禁行为的配置文件是 `/etc/fail2ban/jail.conf`。为了便于修改，可以把它复制一份 `/etc/fail2ban/jail.local`，后面的修改都针对 `jail.local` 这个文件。

```
$ sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

你也可以在目录 `/etc/fail2ban/jail.d` 里面，新建单独的子配置文件，比如 `/etc/fail2ban/jail.d/sshd.local`。

同样地，修改配置以后，需要重新启动 `fail2ban.service`，让其生效。

配置文件里面，`[DEFAULT]` 标题行表示对于所有封禁目标生效。举例来说，如果封禁时间修改为1天，`/etc/fail2ban/jail.local` 里面可以写成：

```
[DEFAULT]
bantime = 1d
```

如果某人被封时，对站长发送邮件通知，可以如下设置。

```
[DEFAULT]
destemail = yourname@example.com
sender = yourname@example.com

# to ban & send an e-mail with whois report to the destemail.
action = %(action_mw)s

# same as action_mw but also send relevant log lines
#action = %(action_mwl)s
```

如果配置写在其他标题行下，就表示只对该封禁目标生效，比如写在 `[sshd]` 下面，就表示只对 sshd 生效。

默认情况下，Fail2Ban 对各种服务都是关闭的，如果要针对某一项服务开启，需要在配置文件里面声明。

```
[sshd]
enabled = true
```

上面声明表示，Fail2Ban 对 sshd 开启。

3.3 配置项

下面是配置文件 `jail.local` 的配置项含义，所有配置项的格式都是 `key=value`。

(1) bantime

封禁的时间长度，单位 `m` 表示分钟，`d` 表示天，如果不写单位，则表示秒。Fail2Ban 默认封禁10分钟（10m 或 600）。

```
[DEFAULT]
bantime = 10m
```

(2) findtime

登录失败计算的时间长度，单位 `m` 表示分钟，`d` 表示天，如果不写单位，则表示秒。Fail2Ban 默认封禁 10 分钟内登录 5 次失败的客户端。

```
[DEFAULT]
```

```
findtime = 10m
maxretry = 5
```

(3) maxretry

尝试登录的最大失败次数。

(4) destemail

接受通知的邮件地址。

```
[DEFAULT]
```

```
destemail = root@localhost
sender = root@<fq-hostname>
mta = sendmail
```

(5) sendename

通知邮件的“发件人”字段的值。

(6) mta

发送邮件的邮件服务，默认是 `sendmail`。

(7) action

封禁时采取的动作。

```
[DEFAULT]
```

```
action = $(action)_s
```

上面的 `action` 是默认动作，表示拒绝封禁对象的流量，直到封禁期结束。

下面是 Fail2Ban 提供的一些其他动作。

```

# ban & send an e-mail with whois report to the destemail.
action_mw = %(action_)s
    %(mta)s-whois[sender=%(sender)s, dest=%(destemail)s,
protocol=%(protocol)s, chain=%(chain)s]

# ban & send an e-mail with whois report and relevant log lines
# to the destemail.
action_mwl = %(action_)s
    %(mta)s-whois-lines[sender=%(sender)s, dest=%
(destemail)s, logpath=%(logpath)s, chain=%(chain)s]

# See the IMPORTANT note in action.d/xarf-login-attack for when to use
this action
#
# ban & send a xarf e-mail to abuse contact of IP address and include
relevant log lines
# to the destemail.
action_xarf = %(action_)s
    xarf-login-attack[service=%(__name__)s, sender=%
(sender)s, logpath=%(logpath)s, port=%(port)s]

# ban IP on CloudFlare & send an e-mail with whois report and relevant
log lines
# to the destemail.
action_cf_mwl = cloudflare[cfuser=%(cfemail)s, cftoken=%
(cfapikey)s]
    %(mta)s-whois-lines[sender=%(sender)s, dest=%
(destemail)s, logpath=%(logpath)s, chain=%(chain)s]

```

(8) ignoreip

Fail2Ban 可以忽视的可信 IP 地址。多个 IP 地址之间使用空格分隔。

```
ignoreip = 127.0.0.1/8 192.168.1.10 192.168.1.20
```

(9) port

指定要监控的端口。可以设为任何端口号或服务名称，比如 `ssh`、`22`、`2200` 等。

3.4 ssh 配置

下面是 sshd 的设置范例。

```
[sshd]
enabled      = true
port         = ssh
filter       = sshd
banaction   = iptables
backend     = systemd
maxretry    = 5
findtime    = 1d
bantime    = 2w
ignoreip   = 127.0.0.1/8
```

首先需要注意，为了让 Fail2Ban 能够完整发挥作用，最好在 `/etc/ssh/sshd_config` 里面设置 `LogLevel VERBOSE`，保证日志有足够的信息。