SSH原理与运用（一）：远程登录

作者： [阮一峰](http://www.ruanyifeng.com/)

日期： [2011年12月21日](http://www.ruanyifeng.com/blog/2011/12/)

[](http://www.zhufengpeixun.cn/main/course/index.html?ref=ruanyifeng.com)

SSH是每一台Linux电脑的标准配置。

随着Linux设备从电脑逐渐扩展到手机、外设和家用电器，SSH的使用范围也越来越广。不仅程序员离不开它，很多普通用户也每天使用。

SSH具备多种功能，可以用于很多场合。有些事情，没有它就是办不成。本文是我的学习笔记，总结和解释了SSH的常见用法，希望对大家有用。

虽然本文内容只涉及初级应用，较为简单，但是需要读者具备最基本的"Shell知识"和了解"公钥加密"的概念。如果你对它们不熟悉，我推荐先阅读[《UNIX / Linux 初学者教程》](http://www.ee.surrey.ac.uk/Teaching/Unix/" \t "_blank)和[《数字签名是什么？》](http://www.ruanyifeng.com/blog/2011/08/what_is_a_digital_signature.html)。

=======================================

**SSH原理与运用**

作者：阮一峰



**一、什么是SSH？**

简单说，SSH是一种网络协议，用于计算机之间的加密登录。

如果一个用户从本地计算机，使用SSH协议登录另一台远程计算机，我们就可以认为，这种登录是安全的，即使被中途截获，密码也不会泄露。

最早的时候，互联网通信都是明文通信，一旦被截获，内容就暴露无疑。1995年，芬兰学者Tatu Ylonen设计了SSH协议，将登录信息全部加密，成为互联网安全的一个基本解决方案，迅速在全世界获得推广，目前已经成为Linux系统的标准配置。

需要指出的是，SSH只是一种协议，存在多种实现，既有商业实现，也有开源实现。本文针对的实现是[OpenSSH](http://www.openssh.com/" \t "_blank)，它是自由软件，应用非常广泛。

此外，本文只讨论SSH在Linux Shell中的用法。如果要在Windows系统中使用SSH，会用到另一种软件[PuTTY](http://www.chiark.greenend.org.uk/~sgtatham/putty" \t "_blank)，这需要另文介绍。

**二、最基本的用法**

SSH主要用于远程登录。假定你要以用户名user，登录远程主机host，只要一条简单命令就可以了。

　　$ ssh user@host

如果本地用户名与远程用户名一致，登录时可以省略用户名。

　　$ ssh host

SSH的默认端口是22，也就是说，你的登录请求会送进远程主机的22端口。使用p参数，可以修改这个端口。

　　$ ssh -p 2222 user@host

上面这条命令表示，ssh直接连接远程主机的2222端口。

**三、中间人攻击**

SSH之所以能够保证安全，原因在于它采用了公钥加密。

整个过程是这样的：（1）远程主机收到用户的登录请求，把自己的公钥发给用户。（2）用户使用这个公钥，将登录密码加密后，发送回来。（3）远程主机用自己的私钥，解密登录密码，如果密码正确，就同意用户登录。

这个过程本身是安全的，但是实施的时候存在一个风险：如果有人截获了登录请求，然后冒充远程主机，将伪造的公钥发给用户，那么用户很难辨别真伪。因为不像https协议，SSH协议的公钥是没有证书中心（CA）公证的，也就是说，都是自己签发的。

可以设想，如果攻击者插在用户与远程主机之间（比如在公共的wifi区域），用伪造的公钥，获取用户的登录密码。再用这个密码登录远程主机，那么SSH的安全机制就荡然无存了。这种风险就是著名的["中间人攻击"](http://en.wikipedia.org/wiki/Man-in-the-middle_attack" \t "_blank)（Man-in-the-middle attack）。

SSH协议是如何应对的呢？

**四、口令登录**

如果你是第一次登录对方主机，系统会出现下面的提示：

　　$ ssh user@host

　　The authenticity of host 'host (12.18.429.21)' can't be established.

　　RSA key fingerprint is 98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d.

　　Are you sure you want to continue connecting (yes/no)?

这段话的意思是，无法确认host主机的真实性，只知道它的公钥指纹，问你还想继续连接吗？

所谓"公钥指纹"，是指公钥长度较长（这里采用RSA算法，长达1024位），很难比对，所以对其进行MD5计算，将它变成一个128位的指纹。上例中是98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d，再进行比较，就容易多了。

很自然的一个问题就是，用户怎么知道远程主机的公钥指纹应该是多少？回答是没有好办法，远程主机必须在自己的网站上贴出公钥指纹，以便用户自行核对。

假定经过风险衡量以后，用户决定接受这个远程主机的公钥。

　　Are you sure you want to continue connecting (yes/no)? yes

系统会出现一句提示，表示host主机已经得到认可。

　　Warning: Permanently added 'host,12.18.429.21' (RSA) to the list of known hosts.

然后，会要求输入密码。

　　Password: (enter password)

如果密码正确，就可以登录了。

当远程主机的公钥被接受以后，它就会被保存在文件$HOME/.ssh/known\_hosts之中。下次再连接这台主机，系统就会认出它的公钥已经保存在本地了，从而跳过警告部分，直接提示输入密码。

每个SSH用户都有自己的known\_hosts文件，此外系统也有一个这样的文件，通常是/etc/ssh/ssh\_known\_hosts，保存一些对所有用户都可信赖的远程主机的公钥。

**五、公钥登录**

使用密码登录，每次都必须输入密码，非常麻烦。好在SSH还提供了公钥登录，可以省去输入密码的步骤。

所谓"公钥登录"，原理很简单，就是用户将自己的公钥储存在远程主机上。登录的时候，远程主机会向用户发送一段随机字符串，用户用自己的私钥加密后，再发回来。远程主机用事先储存的公钥进行解密，如果成功，就证明用户是可信的，直接允许登录shell，不再要求密码。

这种方法要求用户必须提供自己的公钥。如果没有现成的，可以直接用ssh-keygen生成一个：

　　$ ssh-keygen

运行上面的命令以后，系统会出现一系列提示，可以一路回车。其中有一个问题是，要不要对私钥设置口令（passphrase），如果担心私钥的安全，这里可以设置一个。

运行结束以后，在$HOME/.ssh/目录下，会新生成两个文件：id\_rsa.pub和id\_rsa。前者是你的公钥，后者是你的私钥。

这时再输入下面的命令，将公钥传送到远程主机host上面：

　　$ ssh-copy-id user@host

好了，从此你再登录，就不需要输入密码了。

如果还是不行，就打开远程主机的/etc/ssh/sshd\_config这个文件，检查下面几行前面"#"注释是否取掉。

　　RSAAuthentication yes  
　　PubkeyAuthentication yes  
　　AuthorizedKeysFile .ssh/authorized\_keys

然后，重启远程主机的ssh服务。

　　// ubuntu系统  
　　service ssh restart

　　// debian系统  
　　/etc/init.d/ssh restart

**六、authorized\_keys文件**

远程主机将用户的公钥，保存在登录后的用户主目录的$HOME/.ssh/authorized\_keys文件中。公钥就是一段字符串，只要把它追加在authorized\_keys文件的末尾就行了。  
  
这里不使用上面的ssh-copy-id命令，改用下面的命令，解释公钥的保存过程：

　　$ ssh user@host 'mkdir -p .ssh && cat >> .ssh/authorized\_keys' < ~/.ssh/id\_rsa.pub

这条命令由多个语句组成，依次分解开来看：（1）"$ ssh user@host"，表示登录远程主机；（2）单引号中的mkdir .ssh && cat >> .ssh/authorized\_keys，表示登录后在远程shell上执行的命令：（3）"$ mkdir -p .ssh"的作用是，如果用户主目录中的.ssh目录不存在，就创建一个；（4）'cat >> .ssh/authorized\_keys' < ~/.ssh/id\_rsa.pub的作用是，将本地的公钥文件~/.ssh/id\_rsa.pub，重定向追加到远程文件authorized\_keys的末尾。

写入authorized\_keys文件后，公钥登录的设置就完成了。

==============================================

关于shell远程登录的部分就写到这里，下一次接着介绍[《远程操作和端口转发》](http://www.ruanyifeng.com/blog/2011/12/ssh_port_forwarding.html" \t "_blank)。

（完）

SSH原理与运用（二）：远程操作与端口转发

作者： [阮一峰](http://www.ruanyifeng.com/)

日期： [2011年12月23日](http://www.ruanyifeng.com/blog/2011/12/)

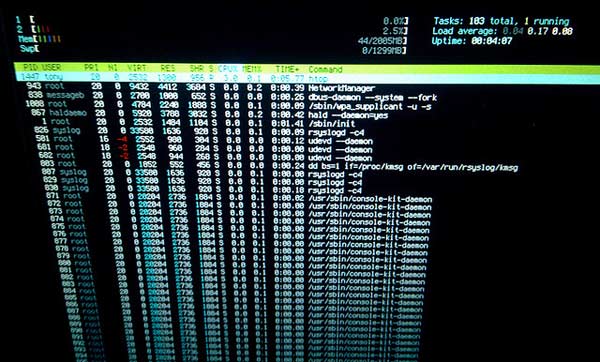
[](http://www.zhufengpeixun.cn/main/course/index.html?ref=ruanyifeng.com)

接着前一次的[文章](http://www.ruanyifeng.com/blog/2011/12/ssh_remote_login.html)，继续介绍SSH的用法。

=======================================

**SSH原理与运用（二）：远程操作与端口转发**

作者：阮一峰



（Image credit: [Tony Narlock](http://www.flickr.com/photos/skiquel/)）

**七、远程操作**

SSH不仅可以用于远程主机登录，还可以直接在远程主机上执行操作。

上一节的操作，就是一个例子：

　　$ ssh user@host 'mkdir -p .ssh && cat >> .ssh/authorized\_keys' < ~/.ssh/id\_rsa.pub

单引号中间的部分，表示在远程主机上执行的操作；后面的输入重定向，表示数据通过SSH传向远程主机。

这就是说，SSH可以在用户和远程主机之间，建立命令和数据的传输通道，因此很多事情都可以通过SSH来完成。

下面看几个例子。

【例1】

将$HOME/src/目录下面的所有文件，复制到远程主机的$HOME/src/目录。

　　$ cd && tar czv src | ssh user@host 'tar xz'

【例2】

将远程主机$HOME/src/目录下面的所有文件，复制到用户的当前目录。

　　$ ssh user@host 'tar cz src' | tar xzv

【例3】

查看远程主机是否运行进程httpd。

　　$ ssh user@host 'ps ax | grep [h]ttpd'

**八、绑定本地端口**

既然SSH可以传送数据，那么我们可以让那些不加密的网络连接，全部改走SSH连接，从而提高安全性。

假定我们要让8080端口的数据，都通过SSH传向远程主机，命令就这样写：

　　$ ssh -D 8080 user@host

SSH会建立一个socket，去监听本地的8080端口。一旦有数据传向那个端口，就自动把它转移到SSH连接上面，发往远程主机。可以想象，如果8080端口原来是一个不加密端口，现在将变成一个加密端口。

**九、本地端口转发**

有时，绑定本地端口还不够，还必须指定数据传送的目标主机，从而形成点对点的"端口转发"。为了区别后文的"远程端口转发"，我们把这种情况称为"本地端口转发"（Local forwarding）。

假定host1是本地主机，host2是远程主机。由于种种原因，这两台主机之间无法连通。但是，另外还有一台host3，可以同时连通前面两台主机。因此，很自然的想法就是，通过host3，将host1连上host2。

我们在host1执行下面的命令：

　　$ ssh -L 2121:host2:21 host3

命令中的L参数一共接受三个值，分别是"本地端口:目标主机:目标主机端口"，它们之间用冒号分隔。这条命令的意思，就是指定SSH绑定本地端口2121，然后指定host3将所有的数据，转发到目标主机host2的21端口（假定host2运行FTP，默认端口为21）。

这样一来，我们只要连接host1的2121端口，就等于连上了host2的21端口。

　　$ ftp localhost:2121

"本地端口转发"使得host1和host3之间仿佛形成一个数据传输的秘密隧道，因此又被称为"SSH隧道"。

下面是一个比较有趣的例子。

　　$ ssh -L 5900:localhost:5900 host3

它表示将本机的5900端口绑定host3的5900端口（这里的localhost指的是host3，因为目标主机是相对host3而言的）。

另一个例子是通过host3的端口转发，ssh登录host2。

　　$ ssh -L 9001:host2:22 host3

这时，只要ssh登录本机的9001端口，就相当于登录host2了。

　　$ ssh -p 9001 localhost

上面的-p参数表示指定登录端口。

**十、远程端口转发**

既然"本地端口转发"是指绑定本地端口的转发，那么"远程端口转发"（remote forwarding）当然是指绑定远程端口的转发。

还是接着看上面那个例子，host1与host2之间无法连通，必须借助host3转发。但是，特殊情况出现了，host3是一台内网机器，它可以连接外网的host1，但是反过来就不行，外网的host1连不上内网的host3。这时，"本地端口转发"就不能用了，怎么办？

解决办法是，既然host3可以连host1，那么就从host3上建立与host1的SSH连接，然后在host1上使用这条连接就可以了。

我们在host3执行下面的命令：

　　$ ssh -R 2121:host2:21 host1

R参数也是接受三个值，分别是"远程主机端口:目标主机:目标主机端口"。这条命令的意思，就是让host1监听它自己的2121端口，然后将所有数据经由host3，转发到host2的21端口。由于对于host3来说，host1是远程主机，所以这种情况就被称为"远程端口绑定"。

绑定之后，我们在host1就可以连接host2了：

　　$ ftp localhost:2121

这里必须指出，"远程端口转发"的前提条件是，host1和host3两台主机都有sshD和ssh客户端。

**十一、SSH的其他参数**

SSH还有一些别的参数，也值得介绍。

N参数，表示只连接远程主机，不打开远程shell；T参数，表示不为这个连接分配TTY。这个两个参数可以放在一起用，代表这个SSH连接只用来传数据，不执行远程操作。

　　$ ssh -NT -D 8080 host

f参数，表示SSH连接成功后，转入后台运行。这样一来，你就可以在不中断SSH连接的情况下，在本地shell中执行其他操作。

　　$ ssh -f -D 8080 host

要关闭这个后台连接，就只有用kill命令去杀掉进程。

**十二、参考文献**

　　\* SSH, The Secure Shell: The Definitive Guide: [2.4. Authentication by Cryptographic Key](http://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch02_04.htm), O'reilly

　　\* SSH, The Secure Shell: The Definitive Guide: [9.2. Port Forwarding](http://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch09_02.htm), O'reilly

　　\* Shebang: [Tips for Remote Unix Work (SSH, screen, and VNC)](http://shebang.brandonmintern.com/tips-for-remote-unix-work-ssh-screen-and-vnc)

　　\* brihatch: [SSH Host Key Protection](http://www.symantec.com/connect/articles/ssh-host-key-protection)

　　\* brihatch: [SSH User Identities](http://www.symantec.com/connect/articles/ssh-user-identities)

　　\* IBM developerWorks: [实战 SSH 端口转发](http://www.ibm.com/developerworks/cn/linux/l-cn-sshforward/)

　　\* Jianing YANG：[ssh隧道技术简介](http://blog.jianingy.com/2009/09/ssh%E9%9A%A7%E9%81%93%E6%8A%80%E6%9C%AF%E7%AE%80%E4%BB%8B/" \t "_blank)

　　\* WikiBooks: [Internet Technologies/SSH](http://en.wikibooks.org/wiki/Internet_Technologies/SSH)

　　\* Buddhika Chamith: [SSH Tunneling Explained](http://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/)

（完）