

# ***SSH: Using and Securing***

**By Whil Hentzen**

**SSH is one of those typical “Linux mysteries” for the uninitiated. SSH provides a secure mechanism to connect to another machine over a network. This allows you to control a remote computer (such as through the command window) over the Internet without exposing your connection to other people. Here's what SSH does, why you'd use it, how it works, how to use it, and how to secure it.**

## 1. Preface

### 1.1 Copyright

Copyright 2004-7 Whil Hentzen. Some rights reserved. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License, which basically means that you can copy, distribute, and display only unaltered copies of this work, but in return, you must give the original author credit, you may not distribute the work for commercial gain, nor create derivative works based on it without first licensing those rights from the author. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/2.0/>.

### 1.2 Revisions

#### 1.2.1 History

Version	Date	Synopsis	Author
1.0.0	2004/8/19	Original	WH
1.1.0	2006/1/6	Added public/private key usage	WH
2.1.0	2007/08/13	Complete reorganization, added section on securing, added info about -XC, logs	WH

#### 1.2.2 New version

The newest version of this document will be found at [www.hentzenwerke.com](http://www.hentzenwerke.com).

#### 1.2.3 Feedback and corrections

If you have questions, comments, or corrections about this document, please feel free to email me at 'booksales@hentzenwerke.com'. I also welcome suggestions for passages you find unclear.

### 1.3 Acknowledgments

Thanks to MLUG member Joe Baker for a great talk on SSH in 2003, the MLUG regulars at the August '04, December '05, and August '07 meetings, Tom Francis, Aaron Schrab, and Chad Voelker for content and reviews, Ted Roche for his critical eyeballing and nudging me to include scp, and Steve Suehring for some background material.

### 1.4 Disclaimer

No warranty! This material is provided as is, with no warranty of fitness for any particular purpose. Use the concepts, examples and other content at your own risk. There may be errors and inaccuracies that in some configurations may be damaging to your system. The author(s) disavows all liability for the contents of this document.

Before making any changes to your system, ensure that you have backups and other resources to restore the system to its state before making those changes.

All copyrights are held by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark. Naming of particular products or brands should not be seen as endorsements.

### 1.5 Prerequisites

This document was written using Fedora Core 2.0 on the server and clients running Fedora Core 1.0 and SuSE 9.2, and assumes a beginner's familiarity with use of Linux via the GUI and the Command Window. It was updated using Fedora Core 6 on both the server and the client.

## 2. SSH definition - what is it and what does it do?

Connecting to another machine on the Internet is a risky proposition. Some tools, like ftp and telnet, that provide access to another machine transmit the username and password in clear text (meaning that they're not encrypted, and thus readable by anyone who captures them.) In some cases, this isn't a problem. For example, FTP transfers are fine for communication that doesn't have to be secure, such as if you're just downloading a set of RPMs whose authenticity can be verified separately. But this isn't acceptable if you are trying to transfer things securely.

On the other hand, telnetting over a public network into a machine to set a configuration, while a common practice years ago, is now disavowed as a practice by all savvy users. This is because there are all sorts of bad guys out there with tools to capture these clear text usernames and passwords for their own unscrupulous purposes. As a result, the safe way to connect is to use a program that encrypts the communications between the two computers.

SSH is one such program. It is a software utility, included with every mainstream Linux distribution, that provides a method of using a command window (or shell) on a remote machine. Its name is an acronym for "Secure SHell". SSH has to run both on the machine you're using (known as the local machine or the client, running client software) and the machine you're connecting to (known as the remote machine, the host, or the server, running server software.) SSH runs on the remote machine just like any other Linux service, and is initiated by you on the client in order to connect to the remote machine.

The internal behavior of SSH is to establish a secure tunnel and then pass back and forth the contents of a shell session. In fact, it gets better. SSH also will also let you share that secure tunnel with other applications that would not be secure on their own.

The client runs the program 'ssh' manually, while the server runs 'sshd' as a service.

## 3. The configuration used for this discussion

I used the following components for this document.

1. A Web server running Fedora Core 2.0 with an IP address of 169.207.151.113 that is tied to the domain of [www.hidbigo.com](http://www.hidbigo.com). This Web server had port 22 open in order to be able to accept SSH connections.
2. A client running Fedora Core 1.0, running on a separate network and Internet connection (for the initial part), and a client running SuSE 9.2, also running on a separate network and Internet connection (for the public/private key part.)
3. Fedora Core 6 for client and server for ports and other new stuff.

## 4. Basic SSH - Using SSH via Passwords

For this HOWTO, I'm going to assume you've got physical access to the server for configuration purposes. Then you will use the client to connect to the server.

First, SSH (the server) must be running on the remote machine, and you need to know the IP address or URL of the remote machine. The remote machine needs to be able to accept requests on port 22, the port used by SSH. (The number port can be changed, as we'll see.)

### 4.1 Configuring the server

For the time being, we're not going to touch the server. We'll discuss how to configure the server, primarily for securing it, later. You will need to make sure the sshd daemon is running, of course:

```
server root>service sshd start
```

on the server as root.

## 4.2 Using the client to connect to the server

Now let's go to the Linux client - the machine you're working on. Open a command window and issue the command

```
ssh
```

followed by the IP address or URL of the remote machine, like so

```
ssh 169.207.151.113
```

or

```
ssh www.hidbigo.com
```

Upon receipt of a connection attempt from a client, the server will begin a handshaking process between the two machines that begins with the server sending its fingerprint to the client so that the client can verify that the server is indeed who it says it is. The client will compare its own copy of the server's key (the copy is stored on the client) with the copy that the server has sent.

If this is the first time you've connected to the remote machine, you'll see a response displaying the fingerprint of the remote machine and a warning indicating that the client doesn't know if that's the right fingerprint. You'll be prompted to verify that the fingerprint is correct, as shown in **Figure 1**.



**Figure 1.** Upon first connection, you're asked to confirm this is the machine you want to connect to.

The line

```
The authenticity of host 'www.hidbigo.com (169.207.151.113)' can't be established.
```

means that the client doesn't yet have a copy, and thus the client doesn't know if the fingerprint is valid or not. . The line

```
RSA key fingerprint is 8a:f8:6f:e4:1f:9a:ca:50:44:f8:83:3e:11:9e:a1:5e.
```

shows the key that the server purportedly sent to the client. (I kinda think it would be clearer if the prompt said something like "RSA key fingerprint on the server" but once you've used SSH for a while, you'll get used to filling in the missing words for yourself.) It is up to the client user to confirm that this key is indeed the actual key of the server. I say 'purportedly' because it is technically possible that the fingerprint that the server sent was intercepted and modified en route before its display on the client's machine.

If this connection is between machines containing highly sensitive data, you may wish to verify the validity of the key through a second channel, such as the telephone. In other words, for this first contact, you (the person in front of the client) would communicate with the owner of the server via another mechanism to find out what the real fingerprint of the server is, and compare that value with the value that was displayed on your computer screen. If they're the same, you can be sure

that the fingerprint was not intercepted and modified en route. For example, you, sitting in front of the client, would call up someone sitting by the server, and ask, "Hey, I'm looking at a fingerprint value of 8a-f8-6f... Is that correct?" and the person by the server would either confirm or warn you that the fingerprint has been changed (and thus there is a security problem.)

To be pragmatic, in most situations you'll simply assume that the initial value of the fingerprint received from the server is authentic.

In both cases, assuming that the key is good, you'll enter 'yes' to the "Are you sure you want to continue connecting?" prompt, and a copy of the key will be saved on the client computer in a file named "known\_hosts". This is indicated by the "Permanently added 'www.hidbigo.com, 169.207.151.113' (RSA) to the list of known hosts." message in **Figure 2**.



**Figure 2.** SSH asks you to verify that the key the server sent to you, the client, is correct; you'll either want to independently verify the authenticity of the key, or blindly trust on the first connection.

The list of known hosts referred to is in the file (on the client)

`/home/{username}/.ssh/known_hosts`

and looks like this:

```
www.hidbigo.com,169.207.151.113 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEAnAUoD6Jhn4KbJyAiX3oX/nR+u4BIP03ZWxl1+PKLDju1n6mYAIUI6m2LK/I0fPDydwC1X/w13Ip+Y/udAN
/KmFmPrRauJIJgVn/81SBRHLY2AsW7gKXGNnwvPrQ7O3v6+tmSTeJrvnUwSr2022rncQ+gf1Tc1rE3L/d9G+5GnXM=
```

The known\_hosts file will contain a separate entry for every host (server) that the client connects to. The known\_hosts file is stored in the user's home directory, so that there is a separate known\_hosts file for each user on the client. Thus, the hosts that Al connects to won't be confused with the hosts that Barb connects to.

Once the value has been added to the known\_hosts file, the value received from the server will be compared with the value stored on the client during subsequent connections. If the values differ, it could mean that the server has been modified, such as through an update of the ssh program, or that the server has been attacked and compromised. Regardless, in the event of the value has changed, you'll want to investigate why before continuing with the connection.

Upon successful entry of the password, the connection will be closed, as shown in Figure 2, but the client will now have a record of the server. Issue the 'ssh' command again with the IP address or URL, and you'll immediately be prompted for the password this time, as shown in **Figure 3**.



**Figure 3.** After successfully supplying the password to the remote user account, you'll see a command prompt that confirms you're connected remotely.

Upon successful entry of the password this time, you'll get a command prompt for the user on the server, and you can now issue commands to the server just as if you were sitting in front of the machine itself.

### 4.3 Matching accounts

At this point, it's appropriate to discuss the user accounts and passwords being used by ssh more explicitly. When you're logging on to the remote box, you're logging onto an account on that machine. Since you haven't explicitly provided an account when you issued the

```
local> ssh www.hidbigo.com
```

command, ssh is going to use the account that you're using on the client. However, since you're logging onto the server, you're prompted for the password of the client's currently logged on user – but on the server machine.

Suppose that there's an alvin on the server with a password of 'serendipity', and an alvin on the client with a password of 'courageous'. If you're logged in as 'alvin' on the client machine, there will need to be an 'alvin' account on the server as well, in order to simply issue the 'ssh <host>' command. However, when you're prompted for alvin's password by SSH, you'll need to enter the password for the 'alvin' account on the server, i.e. 'serendipity'.

If you don't have an 'alvin' account on the server, you can log on to the server using a different account; that will be discussed shortly.

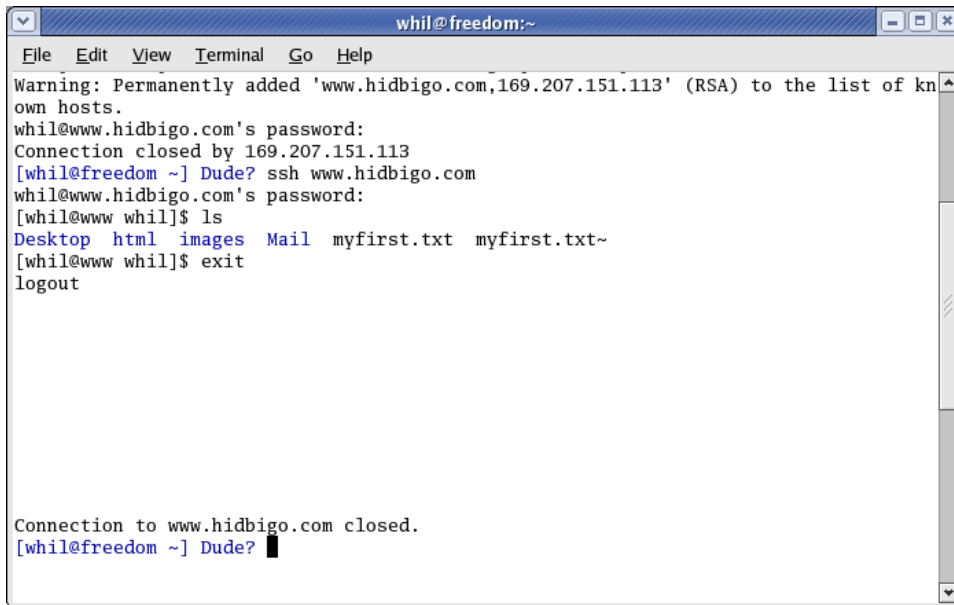
If you, the reader, named 'herman', try to log in to the sample domain here, www.hidbigo.com, you'll be stymied unless you know a user account on the www.hidbigo.com server, and the password associated with that user account, because there isn't a 'herman' account at www.hidbigo.com. Although there probably should be.

### 4.4 Shutting down

Once you're done with your session on the remote machine, type

```
exit
```

in the command window, as shown in **Figure 4**.



```
whil@freedom:~  
File Edit View Terminal Go Help  
Warning: Permanently added 'www.hidbigo.com,169.207.151.113' (RSA) to the list of known hosts.  
whil@www.hidbigo.com's password:  
Connection closed by 169.207.151.113  
[whil@freedom ~] Dude? ssh www.hidbigo.com  
whil@www.hidbigo.com's password:  
[whil@www whil]$ ls  
Desktop html images Mail myfirst.txt myfirst.txt~  
[whil@www whil]$ exit  
logout  
  
Connection to www.hidbigo.com closed.  
[whil@freedom ~] Dude? █
```

**Figure 4.** Logging out of the SSH session restores your command prompt to its previous value.

After a few moments, the server will respond with the logout command and then the client will confirm the closure of the connection as well. The command window on the client will be returned to the currently logged in user on the client.

## 5. Using SSH

Once you've connected to the remote box, you're going to want to do things. Here is how to perform a couple of specific actions.

### 5.1 Become root on the remote machine

You can become root on the remote machine by issuing the 'su' command in the command window once you're connected to the remote machine. You'll need the password of the root user on the remote machine, of course. When you're done with your root work on the remote machine, issue the 'exit' command and you'll be returned to the regular user who was logged onto the remote machine. Let's walk through the steps.

For example, if alvin had logged onto the server, the prompt in the command window would say

```
[alvin@remote alvin]$
```

After alvin then su'd to become root, the prompt would say

```
[root@remote alvin]$
```

or if alvin did 'su -', the prompt would say

```
[root@remote ~]$
```

where the ~ denotes root's home directory, because the '-' incorporates the root user's environment and switches the current directory to the user's home directory, which would be /root in this case.

## 5.2 Logging in as a different account

If you want to log in as a specific user on the remote machine in a minimum number of keystrokes, you can do it in one of two ways. If you wanted to log in as the user bob on the remote machine, for example, you could

```
ssh -l bob www.hidbigo.com
```

or

```
ssh bob@www.hidbigo.com
```

where '-l' is the letter "ell", and where you could use the IP address instead of the hostname if you wished. You don't need to have a corresponding 'bob' account on the client.

## 5.3 Testing SSH on a single machine

If you don't have a pair of machines to experiment with, you can log in to the local machine, using '127.0.0.1' or 'localhost' as the IP address or URL, like so:

```
ssh localhost
```

or

```
ssh 127.0.0.1
```

## 5.4 Restarting SSH remotely

You can restart the SSH server remotely (while you're seated at your client) by logging onto the remote machine via SSH, changing to the root account, and issuing the restart command. You can issue the restart command in a variety of ways, depending on the distribution. Here are two examples:

```
service sshd restart
```

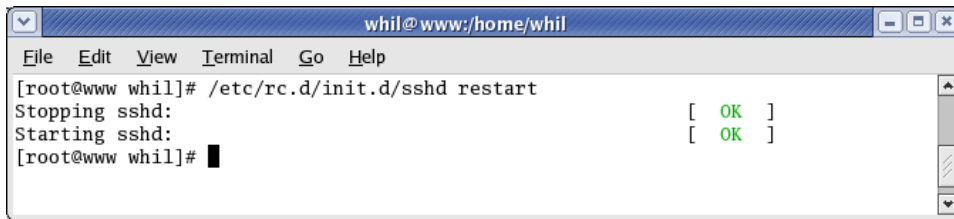
```
/etc/rc.d/init.d/sshd restart
```

If you forget to switch to root first, you'll get a pair of errors, as shown in **Figure 5**.



**Figure 5.** You can't restart SSH on the remote machine unless you're logged in as root.

**Figure 6** shows a successful restart.



**Figure 6.** When you are logged in as root on the remote machine, you can restart SSH.

At this point, it would probably be a good idea to try establishing another (brand new) SSH connection to the server before closing the current (original) connection. This way, if you messed up the configuration (indicated by the new connection attempt failing), you haven't locked yourself out, since you still have the original connection open and available to make further changes.

## 5.5 Using scp (secure copy)

So you can make a secure connection to another machine, and then run programs on that remote machine from the comfort of your own easy chair. What else might you want to be able to do? Copy files back and forth, just as if that remote machine was just another locally accessible share. Why wouldn't you just use the 'cp' command? Because when you're on the remote box, 'cp' no longer knows anything about the client machine. When you're connected to a remote machine via ssh, the command window you're using is for all practical purposes a command window on that remote box, just as if you were sitting in front of it. Only you aren't. And so, since if you were in front of the remote machine, you wouldn't be able to use the cp command to copy files from the client box back on your desk to the remote machine, you can't do it in an SSH session either.

Enter scp, the secure copy command. scp uses the SSH secure tunnel to transmit files in both directions.

scp, however, knows about both the client and the server – about both the local machine and the remote box. Here's how it works. Suppose you are on a local box, and you want to transfer a file from your local box, say, a new HTML page named itemlist.txt, to the remote machine, and keep the same name. Issue the scp command:

```
[whil@freedom ~] Dude? scp ./itemlist.txt whil@www.hidbigo.com:/home/somedir
whil@www.hidbigo.com's password:
itemlist.txt                                100% 191      3.4MB/s   00:00
[whil@freedom ~] Dude?
```

Note that the syntax of scp is similar to cp: command, source, and then target. When you are describing the remote machine, however, you need to include more than just the filename – you need to include the name of the host and the user you're connecting through (just like an ssh command) as well.

You'll be asked for whil's password on the remote machine, and assuming success in that area, voila, the file will be copied through the secure SSH tunnel.

## 5.6 Using graphical tools on the remote box

Up to this point, we've been using a terminal window and issuing text commands. Wouldn't it be nice to use graphical tools? For instance, I hate text-only editors like vi. I'd rather use kate, but kate is a graphical tool. As long as your /etc/hosts file contains a 'localhost' entry, you can issue ssh with the "-XC" flags like so:

```
alvin local> ssh -XC bob@www.hidbigo.com
```

Once in the remote shell, you can issue graphical commands like xclock or kate, and you'll be running those tools on the server.

## 6. Securing SSH

Now that you're comfortable using SSH, it's time to talk about locking down the server. The following sections act as a checklist of actions you should consider in order to make your SSH server more secure.

### 6.1 Restricting root logins

On the server, the first thing you should do is change a setting in `/etc/ssh/sshd_config` the SSH config file to make sure users can't ssh into the server as the user's root user. The default value for

```
PermitRootLogin
```

is (on some distributions)

```
yes
```

With this default, anyone can try to log in to the server as root, and once that's accomplished (either by knowing the root password or by guessing), they own the machine.

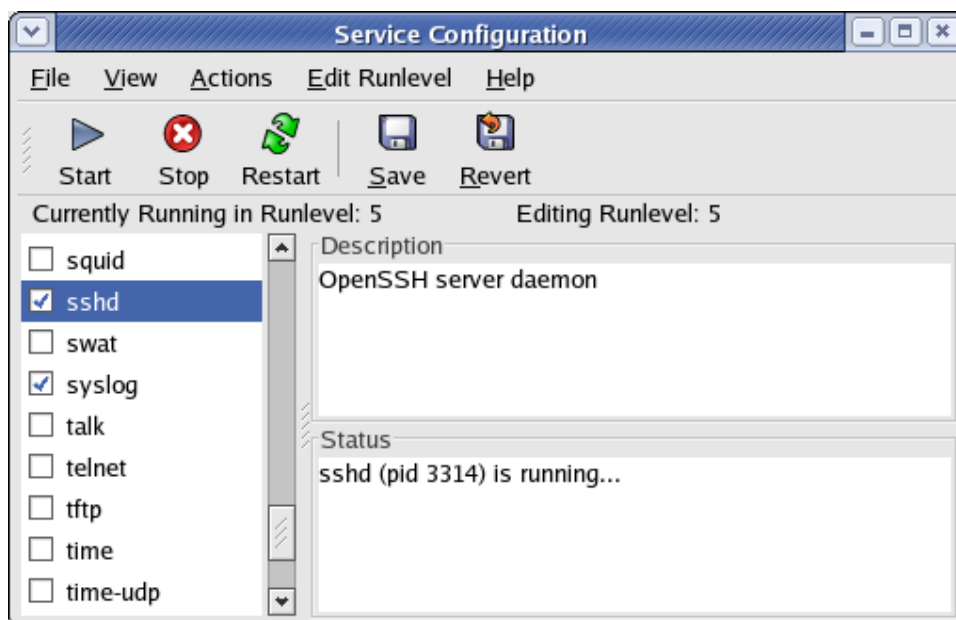
Change this setting to no, like so:

```
PermitRootLogin no
```

Some distributions have this setting commented out, which is equivalent to having the setting set to no. Nonetheless, it's probably a good idea to explicitly set it to no. This 'no' setting prevents someone from SSH'ing into the machine as root. Instead, they would have to SSH into the server using a different user account, and then, as that user, 'su' to become root. This means they have to know not only the root account's password, but also the username and password of another account on the machine for the initial access, so you're made it harder for them to break in and gain control.

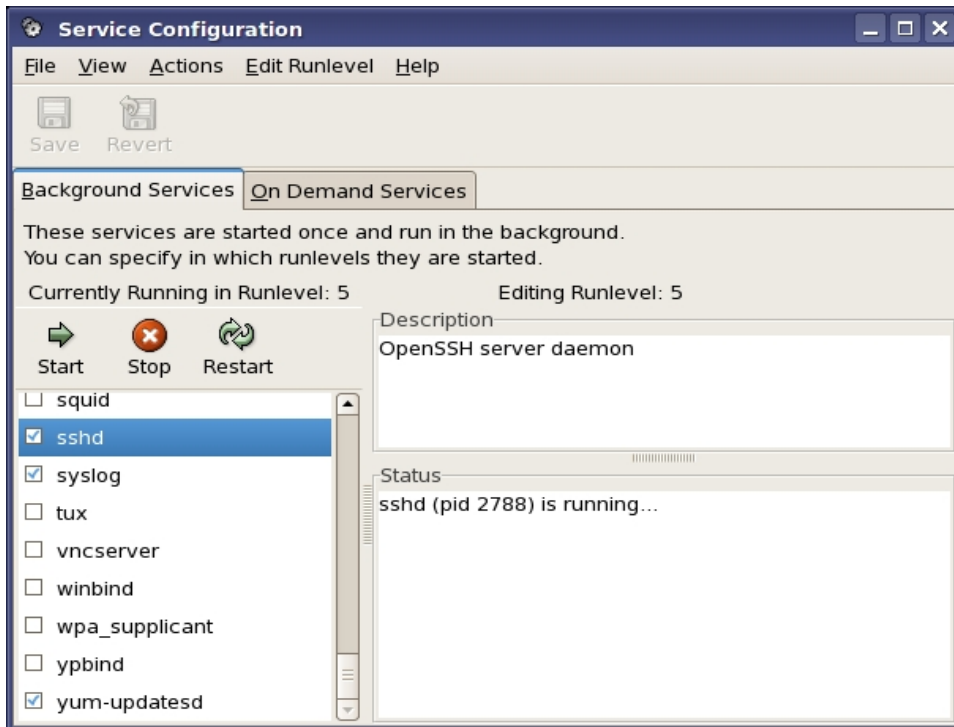
Once you made the change, you'll need to restart the SSH server. This can be done via a command in the command window as discussed earlier, or through the GUI.

In the SUSE GUI, open the Services dialog via the System Settings | Server Settings | Services menu option (you'll need to enter the root password), select the sshd service, and then click the Restart button as shown in **Figure 7**.



**Figure 7.** The Service Configuration dialog in SUSE allows you to restart the SSH service.

In the Fedora Core GUI, open the Services dialog via the System | Services menu option (you'll need to enter the root password), select the `sshd` service, and then click the Restart button as shown in **Figure 8**.



**Figure 8.** The Service Configuration dialog in Fedora Core allows you to restart the SSH service.

## 6.2 Changing the SSH Port

SSH communicates over port 22 by default, and, as a result, attackers frequently direct their attention to port 22. If you configure SSH to listen on a different port, SSH-specific attacks that target port 22 will be stymied. Here's how to configure the SSH server to listen on a different port.

Open up `/etc/ssh/sshd_config` on the server and change the line

**Port 22**

to, say,

**Port 484**

The port number you use depends on what services you are already using on the server. For instance, if you're running a Web server on the server, you wouldn't want to use port 80. Then restart the SSH server. If you then try to connect to the SSH server normally:

```
client> ssh bob@www.hidbigo.com
```

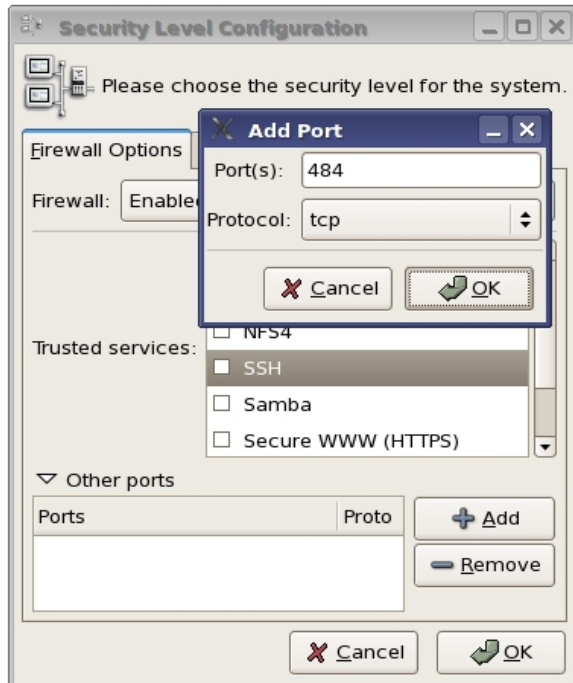
you'll get a message like this:

```
client> ssh: connect to host www.hidbigo.com port 22: Connection refused
```

That's what we're trying to accomplish, by the way. So how DO you connect? You have to specify which port to connect to, like so:

```
client> ssh -p484 bob@www.hidbigo.com
```

If the command just hangs, you probably didn't open the firewall on the server to allow requests to get through port 484. (You can Ctrl-C to get out of the non-responsive ssh attempt.) On Fedora Core, you can access the firewall (called the Security Level Configuration dialog) via the **F | Administration | Security Level and Firewall** menu. Click the "Add" button and add the port you specified in the `sshd_config` file. Adding Port 484 is shown in **Figure 9**.



**Figure 9.** Opening a port in the Fedora Core firewall for SSH to use.

### 6.3 Changing the port for easier external access

This may seem to be counter-intuitive to the heading of this section, but it can be useful to change the port that the SSH server is listening on in order to make access to the SSH server available from an outside connection that is restricted on that end. For example, suppose you're at a public library, and you need to SSH into your server. However, the library has their network locked down so that only HTTP (port 80) and HTTPS (port 443) requests are allowed out. If you know in advance that this will be the case, you could set up your SSH server to listen on port 443, and you could then connect from the library with nary an issue. This would only work, of course, if you're not already using port 443 on the server.

### 6.4 Restricting via the Firewall

In concert with changing the port that SSH is listening on, you can also restrict access to the SSH server by turning off access to the port via the firewall. Figure 9 also shows the SSH server port being turned off (note that the checkbox next to SSH is unchecked.)

## 6.5 Restricting via host.allow and host.deny

Another ring in your defenses can be to use the host.allow and host.deny files in /etc to provide discrete access to specific hosts. A discussion how to configure these is beyond the scope of this article – we're talking about SSH, after all. But depending on your needs for security, this step ought be in your checklist.

## 6.6. Using SSH via Public Key Authentication

While password authentication, described at the beginning of this article, solves the problem of mechanisms that transmit passwords in clear text, it's still susceptible to cracking. The year of 2005 saw a rapid increase in SSH attacks using brute force password guessing tools against common accounts. Hopefully your remote machine doesn't have an account named 'webmaster' with a password of 'god' or 'test123' or 'qwerty' (although if it does, the box has long been compromised.) As a result, it's an opportune time to update this paper with a discussion of an alternative to passwords - authentication using public and private keys.

This method of authentication has an added benefit - you can log into the remote machine without having to type a password. In fact, some folks espouse public key authentication more for this convenience than for the additional security benefits.

### 6.6.1 How public/private keys work

Public key authentication involves a pair of related keys that in this situation take the form of text files. The public key sits on the remote machine while the private key resides on the local (and presumably terribly, terribly safe) machine. When the local machine connects to the remote box, the two keys are matched up by the remote machine. If they match correctly, the authentication passes and you are granted access.

#### 6.6.1.1 An amateur's discussion of public and private keys

You can skip this section if you're not interested in how the keys work. But since you're a Linux geek, you probably are interested.

(For a more in-depth discussion, see Simon Singh's excellent book, *The Code Book* (Anchor Books, ISBN 0-385-49532-3) that addresses this topic as part of complete coverage of cryptography from ancient Egypt to modern day methods.)

The idea behind public and private keys is that a user, Alice, can distribute her public key in many, many places, for everyone to use. Anyone, say, Bob, can use that public key to encrypt a message and send the result to Alice. However, since Alice is the only one who has the matching private key, she's the only person who can decrypt the message.

Singh uses the example of Alice having the only key to a padlock, and then distributing bazillions of copies of this padlock, say, to every post office in the world. Then, when Bob wanted to send a secret message to Alice, he'd go to his local post office, ask for an "Alice padlock", lock up his message, and send it off to her. When she got the locked-up message, she'd use her key to unlock it.

The padlock(s), of course, represent the public key (I think of "public lock"), and Alice's key represents her private key. Anyone can encrypt a message for Alice, but only she, using her private key, can decrypt it. This way, two people can communicate securely without ever having to meet to exchange a mediating mechanism first.

In the realm of computers, the public and private keys are files that contain data, and the locking mechanism is a mathematical algorithm. Alice publishes a public key,  $N$ , which is the sum of two prime numbers, and for which only she knows the two prime factors.

Bob encrypts his message using a special function and Alice's public key, like so:

**EncrMsg = f(UnenMsg, N)**

This function is special because it is a 'one-way' function. You can calculate EncrMsg if you know UnenMsg and N, but you can't reverse engineer the function to determine UnenMsg if you know EncrMsg and N. This is in contrast to a 'two-way' function like multiplication, where, if you know any two of the pieces, you can determine the third.

A simple example of a one-way function is modulus arithmetic. If you know  $i$ , you can easily calculate  $3^i \bmod N$  like so:

$$x = 3^i / N$$

However, if you know  $x$  and  $N$ , and those values are large, it becomes highly laborious (even with a computer) to determine what  $i$  is.

Back to Alice and Bob. Since the special function can only be reverse-engineered using the factors, and only Alice knows the factors, it works. (This, of course, is why the search for very, very large prime numbers is so interesting to security folks.)

### 6.6.2 To implement public key authentication

As with the password section, I'm going to assume you've got access to both machines - you'll use the server for configuration purposes and then will use the client to connect to the server.

First, SSH (the server) must be running on the remote machine, and you need to know the IP address or hostname of the remote machine. The remote machine needs to be able to accept requests on port 22, the port used by SSH.

#### 6.6.2.1 Configuring the server

On the server, the first thing you will need to do is change several settings in `/etc/ssh/sshd_config` the SSH config file. First, change the default value for

```
PermitRootLogin
```

```
to
```

```
no
```

if it hasn't been already done (as described earlier.) Next, make sure the following lines are set (also in the config file) as shown:

```
PubkeyAuthentication yes
```

```
PasswordAuthentication no
```

```
ChallengeResponseAuthentication no
```

```
UsePAM no
```

The first enables public key authentication, while the second and third disable other types of authentication, so that potential intruders can't use an alternative method to authenticate and gain access. The last turns off PAM (Pluggable Authentication Module.)

Once you made these four changes, you'll need to restart the SSH server.

And now you're done with configuring the server (although don't ship the box off to a bunker under the Rocky Mountains just quite yet, since we'll be working with it once more for a moment - in order to move the public key file from the client to the remote box.)

By the way, I'm assuming that you have a non-SSH mechanism for moving the public key file that you create on the local box to the remote box, like, perhaps, a floppy disk or a thumb drive. If you were planning on using SCP (via password authentication) to copy the public key file to the remote box, you'll want to keep "PasswordAuthentication" set to 'yes' until

after you get the file transferred and verify that you can SSH into the box via Public Key Authentication successfully. Don't want to lock yourself out of the box by throwing away the only working key.

### 6.6.2.2 Create the public and private keys on the client

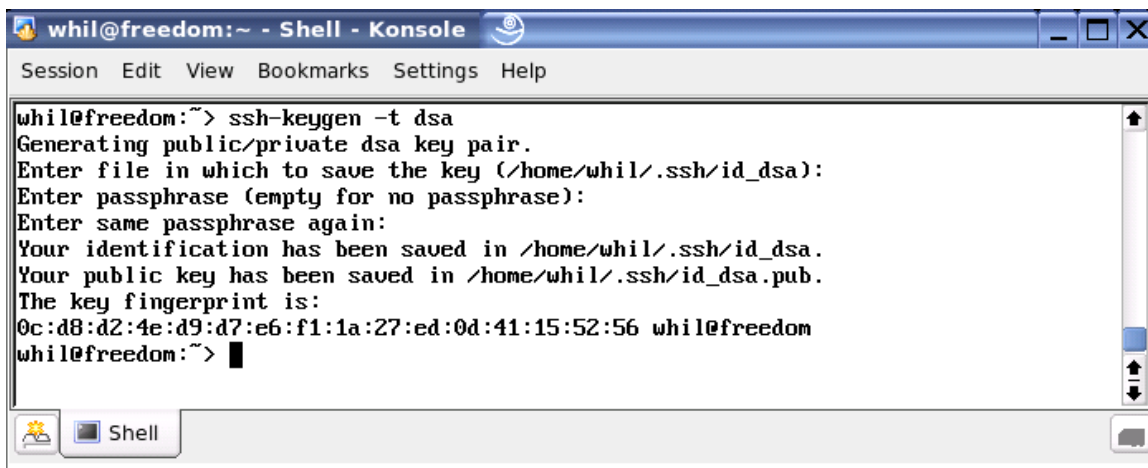
On the client, create the public and private keys via the `ssh-keygen` command. There are two variations of the command, one that uses an RSA public/private keypair, the other that uses DSA. OpenSSH recommends DSA, although it's not hard to find rabid fans for both options. Type

```
ssh-keygen -t rsa
```

or

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

You'll be prompted for a passphrase, and then prompted to verify the passphrase. as shown in **Figure 10**. Use something long but easy to remember (and easy to type!) - and don't use your password!



**Figure 10.** Generating public and private keys via `ssh-keygen` on the client.

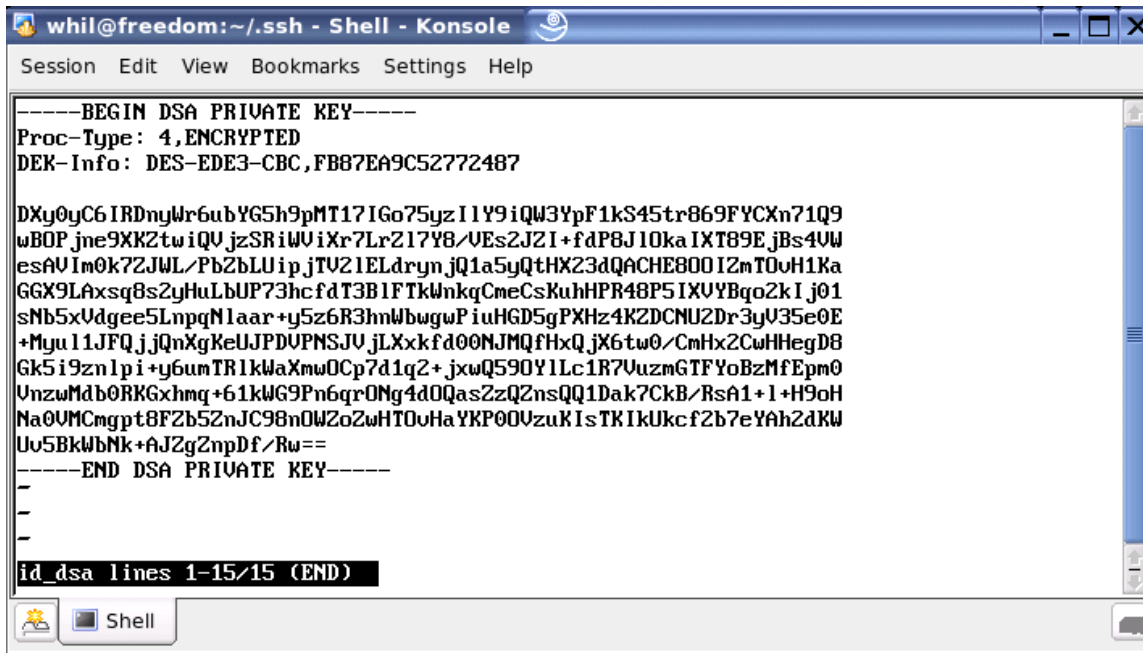
Again, I feel these prompts could be slightly more informative. "Your identification" is the private key. Your public key will be distributed to the ssh server in a moment.

(You may be wondering what the difference between a password and a passphrase is. In the most limited sense, a password is a word, like 'mother', or 'zombiemare' or 'jl44RT\_9', while a passphrase is a full string of (preferably memorable) words, like "send in the clowns", "paul is dead long live the king", or "it was the best of times it was the worst of times". If you're wondering why use a passphrase instead of a password, it's because you're going to be able to use the passphrase to keep your key in memory, while with password authentication, you'll end up typing it in over and over again.)

Once the keys have been generated in memory (it may take five or ten seconds, depending on the horsepower of the box you're on), you'll be prompted for a filename; if you simply hit return, `ssh-keygen` will use the default values of `id_dsa` and `id_dsa.pub` (or `id_rsa` and `id_rsa.pub`), and put them in `$HOME/.ssh`.

After the keys have been written, you'll be reminded what your files are named and where they're located (again, as shown in Figure 10.)

The first file (`id_dsa`) is your private key, and you want to keep it secret, secret, secret. Theoretically, since it's on your local box, it's presumably as safe as anything else on the box. The private key is a simple text file, viewable through any text editor. If you were to open it up, you'd see something like that shown in **Figure 11**.



```

whil@freedom: ~/.ssh - Shell - Konsole
Session Edit View Bookmarks Settings Help

-----BEGIN DSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, FB87EA9C52772487

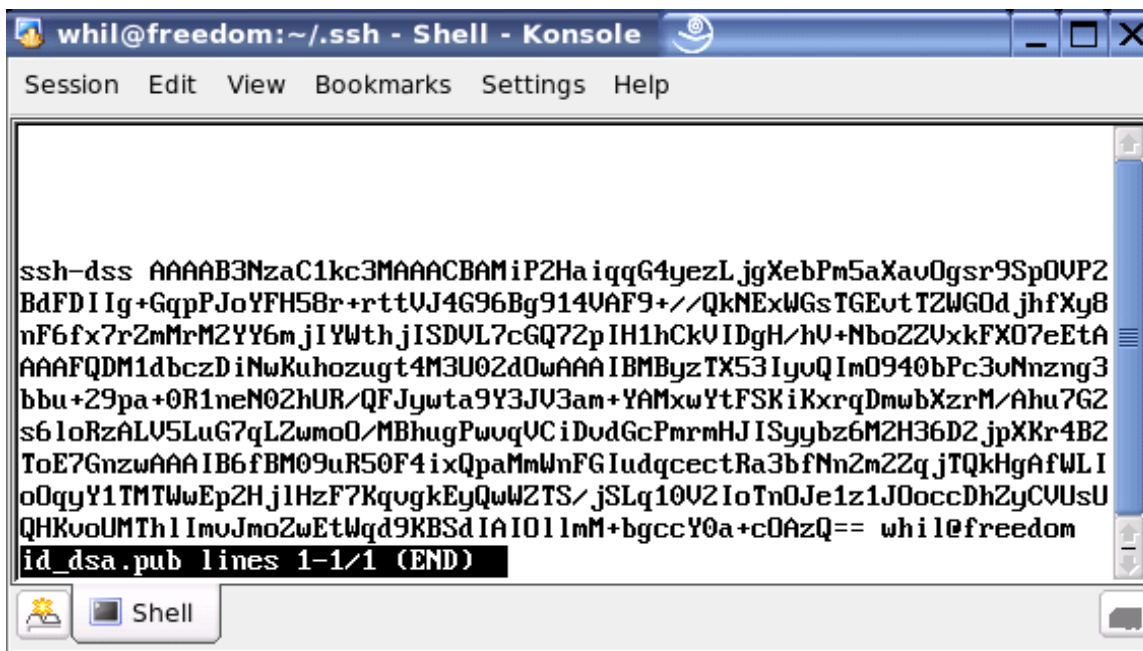
DXy0yC6IRDnyW6ubYg5h9pMT17IGo75yzI1Y9iQW3YpF1kS45tr869FYCXn71Q9
wBOPjne9XKZtwiQVjzSRiWUixr7Lr217Y8/VEs2JZi+fdP8J10kaIXT89EjBs4UW
esAVIm0k72JWL/PbZbLUIpJTU21ELdrynjQ1a5yQtHX23dQACHE800I2mT0uH1Ka
GGX9LXsq8s2yHuLbUP73hcfdT3B1FTkUnkqCmeCsKuhHPR48P5IXUYBqo2kIj01
sNb5xVdgee5LnpqM1aar+y5z6R3hmWbwgWPiuHGd5gPXHz4K2DCNU2Dr3yU35e0E
+Myu11JFQjJqNhgKeUJPDUPNSJUjLXxkf00NMJMQfHxQjX6tw0/CmHx2CwHHegD8
Gk5i9znlpI+y6umTR1kWaXmwOCp7d1q2+jxwQ590Y1Lc1R7UuzmGTFYoBzMfEpm0
UnzwMdb0RK6xhm9+61kWG9Pn6qr0Ng4d0QasZzQZnsQQ1Dak7CkB/RsA1+1+H9oH
Na0VMcngpt8F2b52nJC98n0WZo2wHT0uHaYKP00VzuK1sTK1kUkcf2b7eYAh2dKW
Uu5BkwbMk+AJZgZnpDf/Rw==
-----END DSA PRIVATE KEY-----

id_dsa lines 1-15/15 (END)

```

Figure 11. Contents of a typical private key.

The second file (`id_dsa.pub` – the same as the private key, but with a `.pub` extension) is your public key. Just like the collection of Alice's locks, you can (and are encouraged to) distribute this key to anyone who might want to encrypt something before sending it to you. Some folks place their public key on a public Web page, and then provide a link to that page in their email signature. A quick peek at it through your favorite text editor will look something like **Figure 12**.



```

whil@freedom: ~/.ssh - Shell - Konsole
Session Edit View Bookmarks Settings Help

ssh-dss AAAAB3NzaC1kc3MAAACBAMiP2Ha1qqG4yezLjgXebPm5aXav0gsr9SpOVP2
BdFDIlg+GqppJJoYFH58r+rttUJ4G96Bg914UAF9+//QkNExWGsTGEvtTZWG0djhfxY8
nF6fx7r2mMrM2YY6mjIYWthjISDVL7cGQ72pIH1hCkUIDgH/hU+NboZZVxkFX07eEtA
AAAFQDM1dbczDiNwKuhozugt4M3U02d0wAAAIBMByzTX53IyUQIm0940bPc3vNnzng3
bbu+29pa+0R1neN02hUR/QFJywtA9Y3JU3am+YAMxwYtFSKikXrqDmwbXzrM/Ahu7G2
s6loRzALV5LuG7qLZwmo0/MBhugPwuvQCidvdGcPmrnHJISygbz6M2H36D2jpXKr4B2
ToE7GnzwAAAIB6fBM09uR50F4ixQpaMmWnFGIudqcectRa3bfNn2m2ZqjTQkHgAfWLI
oOqyY1TMTWwEp2Hj1HzF7KqvgkEyQwWZTS/jSLq10V2IoTn0Je1z1J0occDh2yCVUsU
QHKvoUMTh1ImvJmo2wEtWqd9KBSdIAIO1lmM+bgccY0a+c0AzQ== whil@freedom

id_dsa.pub lines 1-1/1 (END)

```

Figure 12. Contents of a typical public key file.

### 6.6.2.3 Put the public key on the server

For this public key to be used with SSH on the remote box, it needs to be put on the remote system into the ".ssh/authorized\_keys" file in the home directory of the account that you'll be SSH-ing into. For example, if you're going to be doing a

```
alvin local> ssh bob@www.example.com
```

command, then you need to put id\_dsa.pub into /home/bob/.ssh/authorized\_keys on the www.example.com box.

You can get the public key file over to the remote box via the SCP command (explained in more detail later in this article) like so:

```
local ~> scp .ssh/id_dsa.pub bob@www.example.com:/home/bob/.ssh
```

replacing 'www.example.com' with the URL or IP address of your remote box. This will put a copy of the public key file in the .ssh directory of the 'bob' user on the remote box.

Next, access the remote box (either via SSH - password authentication, or physical access). Stuff the public key into the authorized\_keys file like so:

```
remote ~/.ssh> cat id_dsa.pub >> authorized_keys
```

(The authorized\_keys file on the remote box can hold more than public key - one for each local system you would be connecting from - so the 'cat' command adds this public key to the file instead of replacing the entire file, hence the use of the '>>' instead of just '>'.)

Finally, you'll want to make sure that the permissions on the .ssh directory and the authorized\_keys file are correct.

```
remote ~> chmod 700 .ssh
```

will make sure the directory is readable only by the user, while

```
remote ~> chmod 644 .ssh/authorized_keys
```

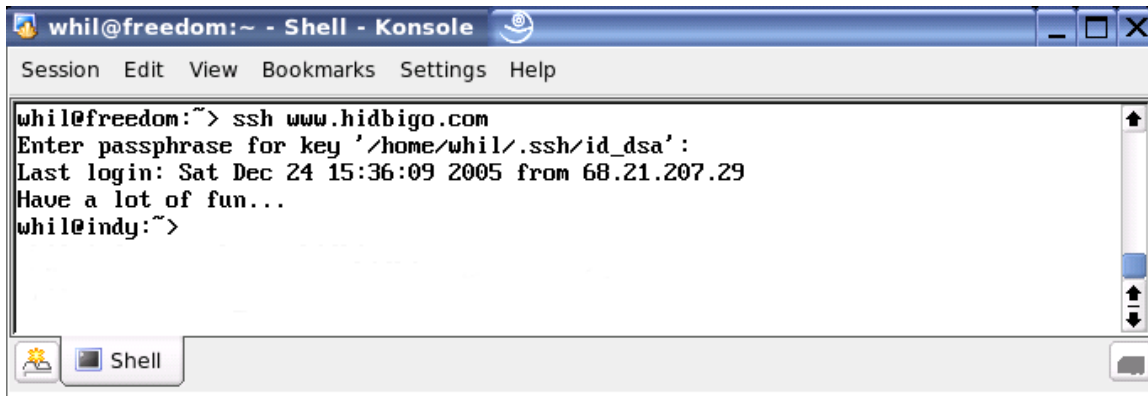
will make sure the authorized\_keys file is read-only by every one else. Now you're all set to give it a shot.

### 6.6.2.4 Using the client to connect to the server

From your local machine, issue the ssh command. Suppose you wanted to log in to the 'bob' account on the 'www.example.com' box, you'd enter:

```
local >ssh bob@example.com
```

You'll be prompted for the passphrase that you entered in Figure 6 earlier. Enter it and you'll be logged into the account on the remote machine, as shown in **Figure 13**.



**Figure 13.** After SSHing into a remote box, you'll be prompted for your passphrase.

Remember those settings in the `sshd_config` file on the remote box that you changed section 5.2.1? If you didn't change the `PasswordAuthentication` setting to `no`, you can still gain access to the remote box using password authentication. If you just hit 'Enter' after being prompted for the passphrase, you'll be prompted for your password next, and if you enter the password for the account on the remote machine, you'll gain access as expected.

As a result, just because you turned "PubkeyAuthentication" to 'yes' doesn't mean you're safe from password crackers - you'll still need to turn `PasswordAuthentication` to 'no' as well.

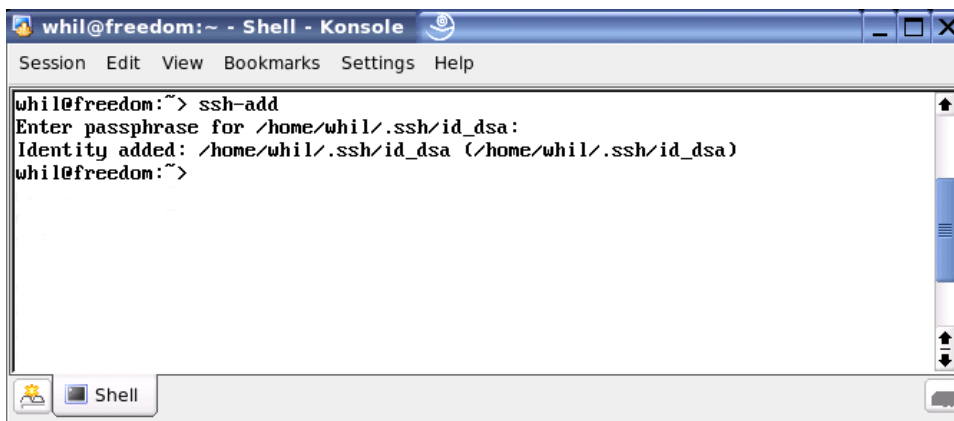
#### 6.6.2.5 Adding your passphrase to your ssh-agent key ring

One of the benefits to using a passphrase is that you don't have to enter it every time you SSH into the remote box. However, if you've already gone through section 5.2.3 and then played around a bit, you'll find that you're prompted for your passphrase every time. What a nuisance!

A program called 'ssh-agent', which is normally started when you log into your GUI, keeps your keys in memory, relieving you from having to type the passphrase in each time. Use the `ssh-add` program to add your private key to the ssh-agent keyring, using the passphrase to unlock it, like so:

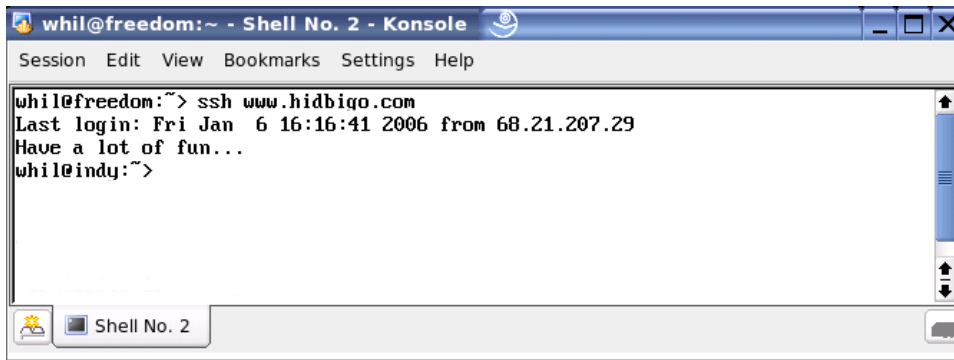
```
local> ssh-add
```

and enter your passphrase when prompted, as shown in **Figure 14**.



**Figure 14.** Running `ssh-add` to add your passphrase to your keyring.

Now, run SSH - you should find that you're not prompted for your passphrase, as shown in **Figure 15**.



**Figure 15.** Once you've added your key to your keyring, you won't be prompted for it any longer.

Your key will stay in memory as long as you stay logged in, until you explicitly remove the key from the keyring, or until you lock ssh-agent (to keep it from handing out keys) via `ssh-add -x`. (You can use `ssh-add -X` to unlock.)

If you don't have the ssh-agent process running, check out the man page for ssh-agent. It's surprisingly readable.

When you log out of the GUI, ssh-agent will terminate and free the memory used to store the keys.

#### 6.6.2.6 Using a private key on a separate device

If you don't want to keep your private key on your local machine, say, because you bounce from machine to machine, you could put it on a removable device, like a USB drive. Suppose your USB flash drive was mounted on

```
/usb
```

and the private key was thus at

```
/usb/id_dsa
```

You could ssh into a remote box like so:

```
local> ssh -i /usb/id_dsa bob@www.example.com
```

## 7. Where to go for more information

The ssh command has both man and info pages (type “man ssh” and “info ssh” in a command window) as part of its inclusion in a Linux distribution. In addition, you may find the following Web sites useful.

<http://www.suso.org/linux/tutorials/ssh.phtml>

[http://wks.uts.ohio-state.edu/sysadm\\_course/html/sysadm-558.html](http://wks.uts.ohio-state.edu/sysadm_course/html/sysadm-558.html)

<http://www.openssh.com/>

## 8. About the author

Whil Hentzen started out life in the early '80's as a custom software developer using dBASE II (he still has the original 8 1/2 x 11 grey binder of documentation, much to the chagrin of his wife), and switched to FoxPro in 1990. Besides billing 15,000 hours in the 90's, he presented more than 70 papers at conferences throughout North America and Europe, edited FoxTalk, Pinnacle Publishing's high end technical journal for 7 years, hosted the Great Lakes Great Database Workshop since 1994. He's written 7 books and published 30 more on a variety of software development topics. He was a Microsoft Most Valuable Professional from 1995 through 2003 for his contributions to the FoxPro development community, and received the first Microsoft Lifetime Achievement Award for Visual FoxPro in 2001.

Whil began using Linux on the desktop when OpenOffice.org became a standard in the mainstream distributions, as it spelled potential for custom application development in the future, and has been a Linux user, developer, and evangelist ever since. His first book on Linux, *Linux Transfer for Windows Power Users*, was published in early 2004.

He is available for new and legacy Visual FoxPro application development as well as Web and desktop development on Linux.

## 9. A word from our sponsor

This free whitepaper is published and distributed by Hentzenwerke Publishing, Inc. We have the largest lists of “Moving to Linux”, OpenOffice.org, and Visual FoxPro books on the planet.

We also have oodles of free whitepapers on our website and more are being added regularly. Our Preferred Customer mailing list gets bi-monthly announcements of new whitepapers (and gets discounts on our books, first crack at special deals, and other stuff as we think of it.)

Click on “Your Account” at [www.hentzenwerke.com](http://www.hentzenwerke.com) to get on our Preferred Customer list.

**If you found this whitepaper helpful, check out these Hentzenwerke Publishing books as well:**

**Linux Transfer for Windows® Network Admins:  
A roadmap for building a Linux file and print server  
Michael Jang**

**Linux Transfer for Windows® Power Users:  
Getting started with Linux for the desktop  
Whil Hentzen**