

**BEGINNER'S GUIDE TO OPENBSD**  
**or**  
**HOW I LEARNED TO STOP WORRYING AND LOVE THE PUFFY**

by kisom <[kisom@devio.us](mailto:kisom@devio.us)>



## **0x00 - Introduction**

Welcome.

You now have a shell on the arguably most secure operating system in the world. OpenBSD is renowned for its strength as an enterprise class operating system and excels in many areas, including as a network operating system in routers and firewalls. I personally use it as a primary operating system on my laptop.

OpenBSD is built around the premises of strong security, including cryptography (being based in Canada, it can bypass many of the cryptographic export restrictions placed on operating systems based in other countries), correctness as a local property, and standardization (the operating system does everything consistently).

This guide will cover using a shell from your personal system; I won't go into installing OpenBSD or using it as a primary laptop OS. Find me in IRC or somewhere else on the internet if you need help with that. The guide is generally slanted towards Windows users. Where appropriate, I will try to point out some tips for OS X and Linux users.

Some very useful resources are covered in the resources chapter at the end. Don't be afraid to poke your head in the IRC channel at some point or post on the forums, either.

Ready? Let's get started.



*Figure 1: No big deal guys, just getting ready to log onto the shell server.*

## 0x01 – Connecting to the Server

The first step in using the shell is connecting to the system. If you're familiar with SSH, you can safely skip this section. The rest of you: listen up.

If you are on Windows, grab a copy of Simon Chatham's PuTTY (available at <http://www.chiark.greenend.org.uk/~sgtatham/putty/>). Install at the minimum PuTTY; if you want the entire suite you can grab the full installer. I recommend the full installer so later on you already have the software to generate your own SSH login keys to do passwordless logins. Once it's installed, open the program. PuTTY will default to SSH (which is good) and the right default port, so just type "devio.us" in the host name box. Hit enter or click "Open".

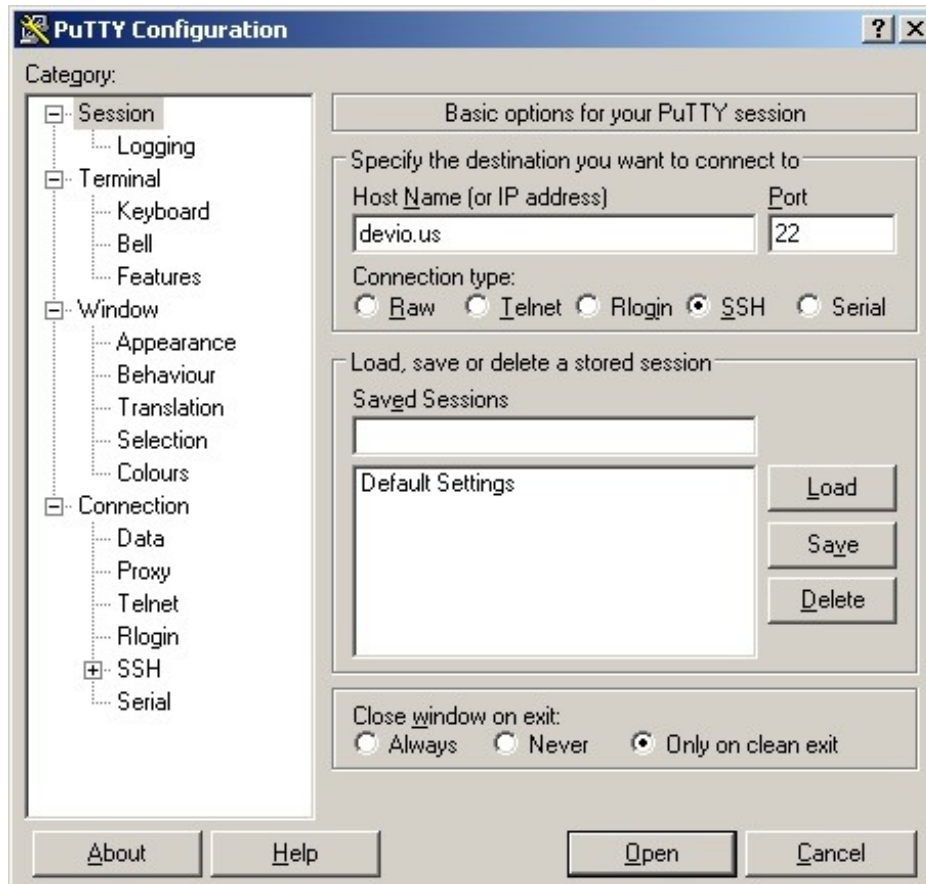


Figure 2: The PuTTY configuration page

If this is your first time connecting, you will get a security alert. Calm down, this is normal. (If needed, refer to the title image.)

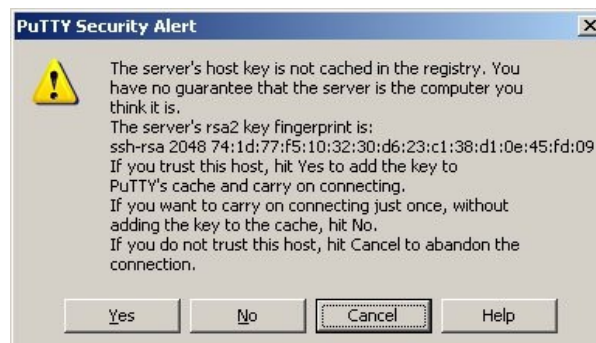
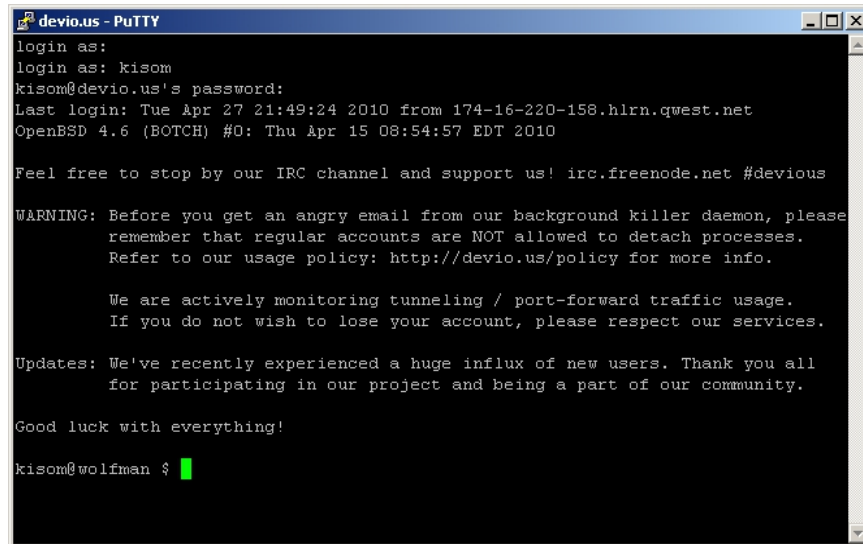


Figure 3: SSH host key security dialog

You should click yes to avoid this window in the future (google SSH host keys if you want to know more). If you click No, you will get the box again in the future. If you click Cancel, the connection will be terminated. As will you. Seriously, we know who you are. If you know what's good for you, click Yes.

A black box will pop up asking for your username; once it connects to the server, it will ask for your password. If you enter the right password you get a cookie, or a shell prompt, whichever the server has available. My guess is you'll get the shell prompt. Cookies over IP is still a draft RFC. If you manage to get cookies, send me an email so I can rejoice with you and maybe figure out where you live and steal your computer so I can continue to get cookies. I digress.



```
devio.us - PuTTY
login as:
login as: kison
kison@devio.us's password:
Last login: Tue Apr 27 21:49:24 2010 from 174-16-220-158.hlrn.qwest.net
OpenBSD 4.6 (BOTCH) #0: Thu Apr 15 08:54:57 EDT 2010

Feel free to stop by our IRC channel and support us! irc.freenode.net #devious

WARNING: Before you get an angry email from our background killer daemon, please
remember that regular accounts are NOT allowed to detach processes.
Refer to our usage policy: http://devio.us/policy for more info.

We are actively monitoring tunneling / port-forward traffic usage.
If you do not wish to lose your account, please respect our services.

Updates: We've recently experienced a huge influx of new users. Thank you all
for participating in our project and being a part of our community.

Good luck with everything!

kison@wolfman $ █
```

*Figure 4: Login screen and shell prompt. Stare at the \$\_. You know you want to. It's shiny.*

As you can see, I am now logged into wolfman, the devio.us shell server as of this writing.

For Mac and Linux users, it is even easier to connect because you don't have to install any software. Mac users, open the your hard drive and open up applications. Under the utilities folder, you will find the Terminal application. Open this up (maybe drag it to your dock for easier access in the future). Linux users should have a terminal or console program in their menu somewhere. From here, type

`ssh user@devio.us`

Make sure to hit enter at the end of the line. You will get a security warning as well, type “yes” and hit enter. Now it should ask for your password – go ahead and type it in. You should be presented with a similar shell prompt.

Take a minute (or hour) to stare at the shell prompt. It is a powerful thing and not to be trifled with.

Get a hold of yourself! Bigger, better, and shinier things are coming. It's time to move on.

## 0x02 – First Steps in the Shell

There are a number of shells you can use – bash is by far the most popular; the typical OpenBSD install defaults to ksh, and there are a number of others available including my favorite, zsh. You can see what shell you are using by typing

```
echo $SHELL
```

on the command line (make sure to press enter after every command). You will see a response like

```
/usr/local/bin/ksh
```

which shows you the path to your shell. Paths are an important thing to learn; basically, the root of the hard drive is '/'; files and folders (directories) are organized under /. /usr is the folder 'usr' in the root directory of the hard drive. /usr/local/bin/ksh is the file ksh in the directory bin which is in the directory local which is in the directory usr which is on the root of the hard drive. The command

```
pwd
```

will “print working directory,” that is, show you what directory you are in now. At this point, you should be in /home/user where user is your username. If you want to see what files are in the current directory, use the command

```
ls
```

or “list”. ls has a number of so-called flags (command line options) that can be passed to it. For example,

```
ls -lh
```

will show you the sizes of all the files in the directory. Some files will be hidden; these are files with a '.' in front of the name, ex. .profile which is the configuration file for certain shells. By default, devio.us has aliased the command 'ls' to automatically show hidden files, so you will likely see a lot of files with dots in front. Now is a good time to introduce the 'man' (manual) command. I mentioned that ls has a number of flags; if you want to see what flags are available, you can pull up the man page for ls, by using

```
man ls
```

This will bring up the manual page for ls. You can scroll up and down using page up and page down, or move one line at a time using the up and down arrow keys. When you've read to your heart's content, you can type 'q' to exit the man page. OpenBSD is famous for having excellent documentation, so just about anything you care to look up can be found via the man pages. You can learn everything you need to know by reading man pages; unfortunately this can have the side effect of causing your eyes to bleed. You can also read the OpenBSD man pages online at <http://www.openbsd.org/cgi-bin/man.cgi>.

What if you can't remember the exact command name? You can use the apropos command; for example, if you couldn't remember what command shows you what directory you are in, you could type

```
apropos directory
```

Holy Batman, Batman! You probably just got a bunch of crap scrolling past your screen. You could use shift+page up / page down to view it, but let's try a different way. First, clear the screen using the command

```
clear
```

Now we are going to use a very powerful feature of UNIX shells: piping. There is a program to read file on OpenBSD called 'less'. You used it once before when you used the man page, you just didn't know it. If you have a command that outputs a lot of stuff and you want to scroll through and read it, you can use the pipe to send it to less. Pipes are a lot like their namesakes in plumbing: you send data down a pathway to another program. They also, like their namesakes, tend to collect crud. C'est la vie. The pipe is the '|' key (should be shift + \ on a standard US keyboard). You use it like so:

```
apropos directory | less
```

You can use the same keys you used in the man page to scroll through the list. You might recall that the command you're looking for returns the current working directory. Let's search for the word `working`; that might give us a faster way to find the command we're looking for. To search, type `/` followed by your search word, then hit enter. Any instances of the search term in less will be highlighted. You can use `n` to go to the next instance, and similarly, `p` to go the previous instance. `q` will exit out of the search. After hitting next a few times, you should see the `'pwd'` command, which is what we are looking for. Hit `q` to exit the search, then `q` to exit less.

So far we've learned all this without leaving our current directory. Let's get adventurous and go into another folder.

On devio.us, your homepage is stored in the directory `/home/user/public_html` – let's take a quick visit, create a test PHP page.

### 0x03 – Creating a Quick Homepage

Later, we'll go over how to upload stuff to your account. For now, we'll write a quick test page..

To change to the `public_html` directory, use the command

```
cd public_html
```

I'll go off on little bit of a rabbit trail here and talk a little bit more about paths. There are two types, absolute paths and relative paths. When we looked at the path to our shell earlier, that was an example of an absolute path: it started with the root directory and listed the entire path to the shell. The example above uses a relative path, i.e. the path is relative to the current working directory. If there was a directory in `public_html` called `cgi-bin`, and that's where we had wanted to go, we could have typed

```
cd public_html/cgi-bin
```

instead. If we were already in `public_html`, then we would just have typed

```
cd cgi-bin
```

There are two special files in every directory you should know about - `'.'` and `'..'`. These are very handy - `'.'` is a link to the current directory and `'..'` is the directory one level up. Say we were in `cgi-bin` and needed to get into the directory under `public_html` called "images". From `cgi-bin`, we could run the command

```
cd ../images
```

Here we can even use a neat trick that some shells have called tab completion. Basically, you type part of the name, hit tab, and the shell tries to autocomplete it as much as possible. It works with commands and file with varying results of usefulness. Right now, the `images` folder doesn't exist. You can create it by using the command

```
mkdir images
```

Remember that this is a relative path, so there is a caveat you have to pay attention to: just `mkdir <dir>` will only create the directory if the entire path leading to `<dir>` exists, i.e. you can't do `mkdir images/vacation` if `images` doesn't already exist. You could, however, use

```
mkdir -p images/vacation
```

because the `-p` flag creates all the necessary parent directories.

Your home directory has a special name that can be used as well - `'~'`. If you type

```
cd ~
```

or

```
cd
```

the shell will return you to your home directory. Likewise, if you had a folder called `'notes'` in your home directory, and you were still in `public_html`, you could use

```
cd ~/notes
```

instead of

```
cd ../notes
```

Alternatively, you could do

```
cd
cd notes
```

Okay, so let's write this web page, mmkay? Find your way back to the public\_html directory. Go ahead, I'm already there and waiting for you. Hurry up.

In case you forgot, you can run the command

```
cd ~/public_html
```

to get there.

We'll create our test page using the editor nano, an easy to use text editor. It's nowhere near as handy as vi/vim but that is a book all in itself. When you get the hang of things, you should definitely look it up and learn to use it. Stay away from emacs, it whispers terrible ideas in your childrens' ears and influences them to do bad things. Quite the nasty character, but do what you must.

So...

```
nano test.php
```

And type the following:

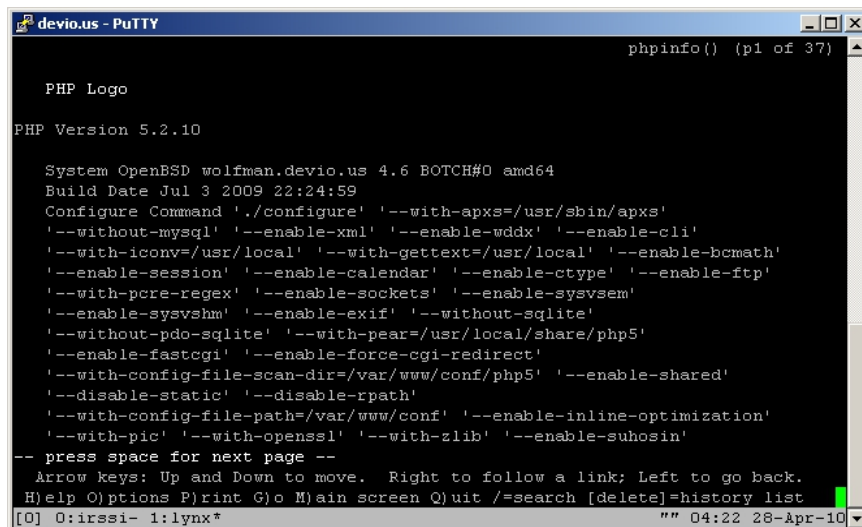
```
<?php phpinfo(); ?>
```

Now, to save the file, you need to hit control + X. At the bottom of the screen, you'll see a bunch of letters with a caret in front of them (such as ^X). The ^ means control +. So ^X means control + X; you can see that means Exit. When you hit it, it will ask you to save. Notice the options for yes and no do not have carets in front of them – you just have to hit the appropriate letter. Hit enter to accept the file name.

Now we'll see one of the other joys of shells: text only web browsing. That's right, it is possible.

```
lynx http://devio.us/~user/test.php
```

where again, user is replaced by your username. lynx is the name of a text-only web browser, which does not, however, support frames. If you need frames, use links.



```
devio.us - PuTTY
phpinfo() (p1 of 37)

  PHP Logo

PHP Version 5.2.10

System OpenBSD wolfman.devio.us 4.6 BOTCH#0 amd64
Build Date Jul 3 2009 22:24:59
Configure Command './configure' '--with-apxs=/usr/sbin/apxs'
'--without-mysql' '--enable-xml' '--enable-wddx' '--enable-cli'
'--with-iconv=/usr/local' '--with-gettext=/usr/local' '--enable-bcmath'
'--enable-session' '--enable-calendar' '--enable-ctype' '--enable-ftp'
'--with-pcre-regex' '--enable-sockets' '--enable-sysvsem'
'--enable-sysvshm' '--enable-exif' '--without-sqlite'
'--without-pdo-sqlite' '--with-pear=/usr/local/share/php5'
'--enable-fastcgi' '--enable-force-cgi-redirect'
'--with-config-file-scan-dir=/var/www/conf/php5' '--enable-shared'
'--disable-static' '--disable-rpath'
'--with-config-file-path=/var/www/conf' '--enable-inline-optimization'
'--with-pic' '--with-openssl' '--with-zlib' '--enable-suhosin'
-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
[0] 0:irssi- 1:lynx* " " 04:22 28-Apr-10
```

Figure 5: Browsing the web with lynx.

What's that you say? You don't see that? Ooooh, you named your file tesr.php, not test.php. And they let you type passwords with those hands... Fair enough. Fortunately this is an easy fix. To move files (which is the same thing as renaming the file) use the 'mv' (move) command:

```
mv tesr.php test.php
```

And voila! Eez fixed.

Ok so now you have a test.php and a folder called ~/notes, and you don't want either of them. Time to learn how to remove files! Unlike Windows, UNIX does not have a recycle bin, so be careful what you delete. The command to delete files is called 'rm,' for "remove." Let's try removing test.php:

```
rm test.php
```

You'll notice I used a relative path – this will only work if you are in the same folder as test.php. Since you know about paths, if you're not in public\_html, I'm sure you can figure out how to use the path to remove it. Now try to remove the folder notes:

```
rm notes
```

The shell spits out:

```
rm: notes is a directory
```

What we need to do is recursively delete notes and anything in it. You do this by passing the -r (recursive) flag to rm:

```
rm -r notes
```

Let's say you had a bunch of test PHP scripts in public\_html named test1.php, test2.php, etc..., and you want to delete all of them. Sure, you could delete each one by hand; you could also do

```
rm test1.php test2.php
```

A faster way is to type

```
rm test*.php
```

The star expands to include anything in between test and php – so if you had a file called testme.php it would also be deleted. There is a good discussion of these so called globs at <http://www.faqs.org/docs/abs/HTML/globberref.html> in the Advanced Bash Scripting guide.

And that should be the majority of the basic commands you need to interact with the shell. Let's move on.

## 0x04 – tmux, the terminal multiplexer, and irssi, the best console IRC client EVAR

So you've gotten this far, but now you have some questions that you need answered. Let's hop on the devio.us IRC channel and ask away. What's this you say, you want to keep working in the shell, but you want to talk on IRC at the same time? Luxury! We used to have to find two systems to log in from, convince the system administrator to let us log on twice, hand toggle in the boot loader on both systems, and do this getting up at four in the morning, two hours before we woke up. But try telling the new generation that and they won't believe you.

Fortunately for you, there's tmux, which is a terminal multiplexer and which now is shipped with OpenBSD. To start with, use the command

```
tmux new <shell>
```

For example, if you are using bash, it would be

```
tmux new bash
```

or if you're using ksh,

```
tmux new ksh
```

You get the idea. From here, you can create new windows and switch between windows. To create a new window (for example for irssi), type control + b then 'c'. If you look at the bottom of the window you'll see a window list. If you look at my screenshot using lynx, you'll see that I had irssi and lynx open.

Now that you have a new window, let's run irssi and get on IRC. Soon you'll understand the true power of a remote shell like devio.us is the ability to lurk more in IRC. To begin, issue the command

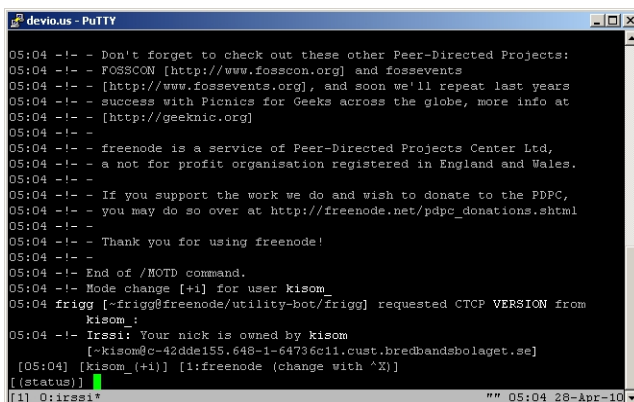
```
irssi
```

Once the window loads, you should see the window list at the bottom change to reflect the fact that you are using irssi in one of the windows. Now type

```
/server irc.freenode.net
```

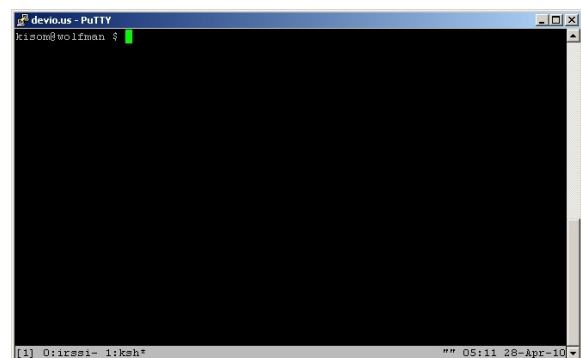
Once you get on the server (see Figure 6 for a screenshot), you can join the devio.us channel by typing

```
/join #devious
```



```
devio.us - PuTTY
05:04 !- - Don't forget to check out these other Peer-Directed Projects:
05:04 !- - FOSCON [http://www.foscon.org] and fossevents
05:04 !- - [http://www.fossevents.org], and soon we'll repeat last years
05:04 !- - success with Picnics for Geeks across the globe, more info at
05:04 !- - [http://geeknic.org]
05:04 !- -
05:04 !- - freenode is a service of Peer-Directed Projects Center Ltd,
05:04 !- - a not for profit organisation registered in England and Wales.
05:04 !- -
05:04 !- - If you support the work we do and wish to donate to the PDPFC,
05:04 !- - you may do so over at http://freenode.net/pdpc_donations.shtml
05:04 !- -
05:04 !- - Thank you for using freenode!
05:04 !- -
05:04 !- - End of /MOTD command.
05:04 !- - Mode change [+i] for user kismom
05:04 frigg [-frigg@freenode/utility-bot/frigg] requested CTCP VERSION from
kismom:
05:04 !- - irssi: Your nick is owned by kismom
[-kismom@42dde155.646-1-64736c11.cust.bredbandsbolaget.se]
[05:04] [kismom (+)] [1:freenode (change with ^X)]
[(status)]
[1] Q:irssi* "" 05:04 28-Apr-10
```

Figure 6: Done logging into freenode, time to join #devious and lurk more.



```
devio.us - PuTTY
kismom@soliman ~$
[1] Q:irssi- 1:ksh* "" 05:11 28-Apr-10
```

Figure 7: Example of tmux with two windows open.

From here, anything you type will be sent as a message. Now, how to get back to the other window? There are two ways: one is to scroll through windows using control + b, 'n' and control + b, 'p', to move to the 'n'ext and 'p'previous windows. The

window you are in will have an asterisk appending it in the window list. The other option that becomes really useful when you have several windows open is to hit control + b, <window number>. For example, if I wanted to switch back to irssi in Figure 7, I could use control + b, '0'.

To close a window, close out of any software you are using and log out of the shell – the window will then close. To exit irssi, you would use

```
/quit [optional quit message]
```

Just typing /quit will exit with a default quit message. Typing

```
/quit hey guys, don't mind me, I'm just following kison's beginner's guide...
```

would display “hey guys, don't mind me, I'm just following kison's beginner's guide...” as your reason for leaving.

## 0x05 – Installing Software into Your Home Directory

You might have a certain piece of software you can't live without, but if you are new to UNIX and shells, that probably isn't the case. Either way, you should know what you don't always have to have software installed globally. You can run most software from your home directory. To demonstrate, we'll install a piece of software called 'blazeblogger,' which will let you set up a blog from your web space. In the devio.us forums (devio.us/forums) the admin blue wrote an article on installing blazeblogger in your home space, including providing a patch to get it working. We'll follow his guidelines to get this set up and running.

To get the software, we need to use the wget program, "the non-interactive network downloader."

```
wget http://blazeblogger.googlecode.com/files/blazeblogger-1.0.0.tar.gz
```

which when it finishes, will give you a shiny "tarball" in your home folder. A tarball is the UNIX equivalent of a .zip or .rar file. Tarballs are created and decompressed with the 'tar' command. To extract blazeblogger, you would use

```
tar xzf blazeblogger-1.0.0.tar.gz
```

Here's where tab completion will come in handy: type through the bla in blazeblogger and hit tab. The shell should automatically fill in the rest of the name for you! Don't worry about how expensive this service is, the cost will just get added to your tab.

I'll take yet another rabbit trail and explain the basics of tar real quick, because it is a very useful tool that you will likely use a lot in your UNIX time. It is actually two programs - 'tar,' the "tape archiver," and a compression program, either gzip or bzip2. Tarballs can generally be opened on windows machines using either WinRAR or 7-zip, or you can get the command line tools to run in either the command prompt or cygwin environment.

There is a really simple syntax for tar:

```
tar <flags> <tarball> [files to be added to tarball]
```

The basic flags for tar are:

-x	extract
-c	compress
-v	verbose
-z	use gzip compression
-j	use bzip2 compression
-f	use the file I am specifying

It is worth noting that flags in tar do not require a dash before them.

So, to add the directory public\_html to a bzip2'd tarball called "website.tar.bz2", you would use

```
tar xjf website.tar.bz2 public_html
```

The paths are preserved relative to where the tarball is being created. This means that in the example above, if I extract the files from website.tar.bz2, I will get a folder called "public\_html" and all the files in it. If I had been in root (i.e. /) and done

```
tar cjf ~/website.tar.bz2 ~/public_html/
```

wherever I extracted the tarball to I would get the folder home/user/public\_html.

Similarly, to extract the tarball website.tar.gz in my home directory, I would use

```
tar xzf website.tar.gz
```

Here's something else you should know about tar: if our website.tar.gz contains the folder public\_html and public\_html already exists in the current directory, then tar will put that folder in the existing public\_html. Wait, what? Yeah, that might have been a bit confusing. Basically, you will end up with public\_html/public\_html because tar will not overwrite the already existing public\_html.

There's more to tar, and guess what? The documentation on it is pretty good (because it's OpenBSD, and that's what

OpenBSD does, remember?) If you want to know more, man tar will give you more than you wanted to know.

Back to blazeblogger...

Where were we? Oh, right, that whole bit about tar. So it should be extracted; now we need to get blue's patch. First, we need to enter the directory blazeblogger extracted to:

```
cd blazeblogger-1.0.0
```

Hopefully you're using tab completion now, instead of typing in these long names by hand. Once in the folder, get the patch:

```
wget http://devio.us/~blue/code/patches/blazeblogger/blazeblogger.patch
```

and then apply the patch:

```
patch -p1 -i blazeblogger.patch
```

Most UNIX software can be built with a sequence of

```
./configure  
make  
make install
```

If you look in the folder, you will see no configure script. Therefore, we only need to type

```
make install
```

This will install blazeblogger under ~/bin . The folder bin is usually a folder where binaries (compiled programs) reside; all the system binaries are located under /bin and /usr/bin. Any binaries requiring superuser privileges reside under /sbin and /usr/sbin. The system will automatically search for binaries in your shell path. You can view your shell path using the command

```
echo $PATH
```

At this point, you should see an entry for /home/user/bin – this is good, the system will automatically search for binaries in your home bin folder. I will discuss shell variables (like \$PATH) in a later section, so don't worry if it doesn't make sense right now.

There is documentation for blazeblogger at <http://blaze.blackened.cz/documentation/>, including a handy reference card. If you look at the documentation, you will see the command blaze-init. Let's set up the blog in our web directory.

The basic syntax is

```
blaze-init                create it in the current directory  
blaze-init --blogdir ~/dir create it in the selected directory  
blaze-init --verbose      list files as they are created
```

I'd prefer to set up my blog at <http://devio.us/~kisom/blog/> but many users set theirs up as their home page. If you want to do it like I do it,

```
blaze-init --verbose --blogdir ~/public_html/blog
```

To set it up in your public\_html folder (i.e. as your homepage) from inside your public\_html folder, run

```
blaze-init
```

Alternatively,

```
blaze-init --blogdir ~/public_html
```

Review the blog configuration commands in the documentation and set up your blog accordingly. Don't forget the blaze-make command!

Let's write our first blog entry:

```
cd ~/<directory to blog>  
blaze-add
```

Write your heartfelt message to the world, then exit out (if you are in vi, hit escape, :wq<enter>). Alternatively, writing a post in a text file and posting the text file is also supported. Learning how to use blaze blogger is outside the scope of this guide, but there is a lot of documentation on the blaze blogger homepage. Essentially, you will need to run

```
blaze-init      (in some form or another)  
blaze-config blog.title This Is An AWESOME Blog  
blaze-add  
blaze-make
```

If you aren't running the commands from your blog directory, you need to use the flag -b <path\_to\_blog>.

You can test it by using lynx (who wants to view pictures anyways?)

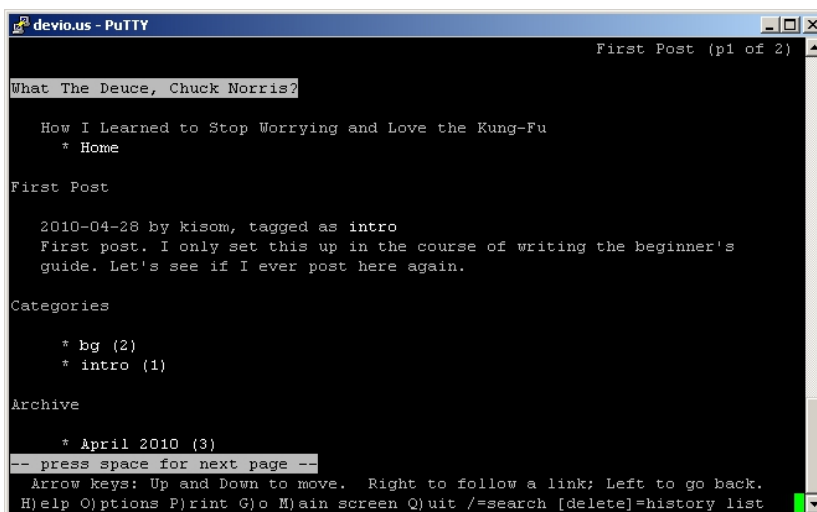
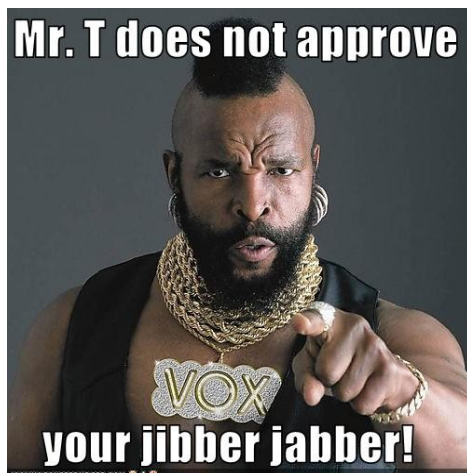


Figure 8: Text only blog. Enough jibber jabber fool!



## 0x06 – Transferring Files to Your Account

To copy files to your shell account, you will need to use an sftp or scp program. Two popular programs on windows are Filezilla and WinSCP; in linux, you can use gftp; OS X users can check out Cyberduck. For this guide, I will be using WinSCP but the principles are the same for any client, except for those clients that are different.

First, download and install WinSCP from <http://winscp.net/eng/index.php>. Once it's installed, run it. You should get a screen like in Figure 8.

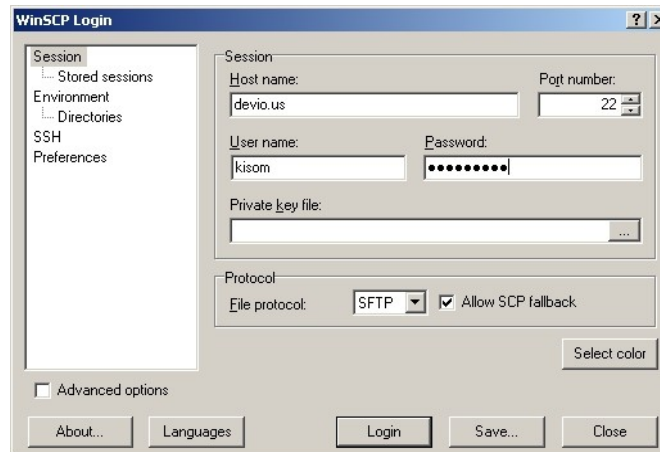
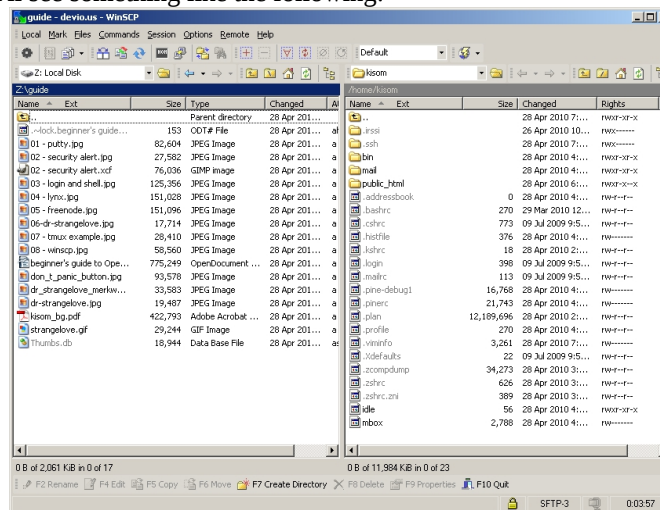


Figure 9: WinSCP login screen

The three relevant fields are host name, user name, and password; all of these should be pretty self explanatory. Hit enter or click login; you will probably get a host key security dialog as with PuTTY.

Once the login completes, you'll see something like the following:



The files on your hard drive are on the left, while the ones on the server are on the right. To upload files, select them in the left hand column, then hit F5 or the Copy button on the bottom. The files should transfer up after a certain amount of time. To copy files from the server, select them in the right hand column and either hit F5 or Copy. So very easy, this is without a doubt going to be the shortest chapter in the guide.

If you somehow have problems or are using another piece of software, you can always ask for help in IRC or the forums. Yeah, you might get your head bit off by a rabid administrator, but on the plus side you only have one head, so it can only be bit off one time. Right?

Right.

## 0x07 – Passwordless Login Using SSH Keys

There are many reasons to want to log in without a password – the admin blue summed up a number of them in his forum post on the topic of SSH keys. This is also an interesting area of discussion in the area of secure key storage... but that's the subject of numerous other books.

Hey, Windows user? Remember when you installed PuTTY, and I told you it would be a good idea to install the full suite? Well, now you will need PuTTYgen. If you don't have it, go get it. I'll give the Mac and Linux users a quick talk while you go do that.

Mac and Linux users... this is really easy. Open up your terminal (but don't log into the shell), and run

```
ssh-keygen -b 2048 -t dsa
```

Just hit enter for which files to save. This generates a DSA keypair with a size of 2048 bits. If you are curious, you should look up public key cryptography; if you do, don't blame me if you start chanting "Bruce Schneier is the new Chuck Norris!" You have only yourself to blame. You have two files, one named `id_dsa` and one named `id_dsa.pub`. `id_dsa` is your secret key; you should protect this file. It should never be uploaded to the shell server! `id_dsa.pub`, however, is your public key, which does not need to be secured at all. You could write it on a bathroom stall in a public restroom to no ill effect, except that would make you more than slightly weird. Now, run

```
scp ~/.ssh/id_dsa.pub user@devio.us:.ssh/authorized_keys
```

We can do this because you shouldn't have any authorized keys yet. If you do, and are following this guide, why are you bothering? Go lift rocks with your mind, or fight giant bears with only a small banana, or something! For the rest of you, enter your password, and wait for it to finish.

Okay, Windows users, come on back over here. So you've installed puttyGen. Open it, and generate a key. In the key box, there should be some stuff like

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAIBk5vZEbI4GbcE+nPBCpVcLLFtFRJ7QT2Uk8qauJF1/H90H
bed5FBScCHNEU9Qa30Kcf0yY1/gS3ZYWtsFsateXcynIV+lLe4CQvEcJiYzFavWdsQbajKa5iw0DzNHN
6jAK1S0bkoKbcS8US76vdbXi7LyNjKaXPvU/zeXPKYmXQ== rsa-key-20100428
```

This is your public key. The easiest way to add this to your account is to copy all of the text, and log into the shell using PuTTY. Now type

```
nano ~/.ssh/authorized_keys
```

and right click into the window (for some reason, PuTTY uses right click as the paste button – most UNIX-based systems use the middle mouse button). It should spit it out into the text file. Save and exit. If this doesn't work, you can paste the public key into notepad and upload this to the shell. Copy it into `~/.ssh/authorized_keys`.

No matter what operating system you are using, now you should try logging into `devio.us` using `ssh`. It shouldn't ask you for a password. If it did, try the following:

```
ls -l ~/.ssh/authorized_keys
```

You should see something like:

```
kisom@wolfman $ ls -l ~/.ssh/authorized_keys
-rw----- 1 kisom devious 977 Apr 28 20:40 ~/.ssh/authorized_keys
kisom@wolfman $
```

An explanation is in order I suppose. The first bit is the permissions for the file; 'kisom' is my username; 'devious' is the group I belong to; the size and timestamp from the last time the file was accessed follow; and last is the file name. The `-rw-----` are the permissions for the file `authorized_keys`, which contains the public keys corresponding to private keys allowed to log in without a password. Permissions are fundamental to UNIX systems. There are three levels of basic UNIX permissions: user, group and world; represented by 'u', 'g', and 'a'. Each level has three basic access rights: read,

write, and execute; these are represented by 'r', 'w', and 'x' while a '-' represents no permissions. Okay, what does that actually mean? Well, the permission set is organized like this

```
  u   g   w
- - - - -
```

Compare this with the permissions above for `authorized_keys`:

```
  u   g   w
- rw- - - -
```

So you see, my `authorized_keys` file is readable and writeable only by me. You might be curious what groups are. Groups are collections of people organized together by some underlying theme. Oh, you meant in UNIX. Right. A group is a set of users, nothing more and nothing less. Groups have a group ID number (a GID), just like users have a user ID number (UID) mapped to a human-friendly name. It may be more fun to tell people you're in group 3888, but that won't mean nearly as much as if you tell them you're in the devious group. If you want to see your UID, what groups you are in, along with their GIDs, you can type the command 'id':

```
kisom@wolfman $ id
uid=1990(kisom) gid=3888(devious) groups=3888(devious)
```

Back to permissions, and how to fix them: permissions can be expressed as a series of numbers like such:

```
  r   4
  w   2
  x   1
```

You add the values for each level of access, so above, my permissions would be 600 – r+w for user, and no access by the group devious or the world. Keep in mind group permissions only apply to the group that owns the file, not to whatever groups you are in. So if you wrote this nifty perl program, and you want your group to be able to read, but not write or execute it, and you don't want the whole world to have anything to do with it, but you want to be able to read, write and execute it, you would need a permission set of 740. That's pretty nifty and all, but you probably have two questions: how would I actually apply new permissions to a file? And how would I change the group, if I wanted the file to actually be owned by a different group I belong to? Whoa there, slow down! One question at a time, please!

To change permissions is easy enough using the `chmod` (change file mode) command:

```
chmod <permission set> <file>
```

If your `authorized_keys` has permissions other than 600, you need to change it like so:

```
chmod 600 ~/.ssh/authorized_keys
```

If you want to change an entire directory (along with all of its subdirectories), you need first of all to understand how permissions affect directories:

- read is the ability to list the contents of the directory
- write is the ability to write new files or delete files in the directory
- execute is the ability to access files in the directory

If a directory does not have execute privileges, you won't be able to do much in it. You have been forewarned.

To have `chmod` change the permissions across the directory, you will need to use the recursive flag, `-R`. For example,

```
chmod -R 740 myfiles
```

You can also change multiple files using the glob we talked about earlier:

```
chmod 644 public_html/*.*
```

The `*.*` ensures only files with extensions get matched – no directories will be matched which will save you from the horror

of non-executable directories. Your `public_html` folder probably only has files with extensions in it.

This brings us to an important point: permissions are not to be taken lightly. Just like any other security factor, you need to consider who needs access and what access they *\*really\** need. Apply permissions accordingly. You can also set a default permission mask on files – look up ‘`umask`’ in the manual pages.

Alright, so how do we change ownership of a file (i.e. user and group)? Similar to the `chmod` command is the `chown` (change ownership) command:

```
chmod <user>:[group] file
```

`chmod` also uses the `-R` recursive flag as well. If you just wanted to change the user, you could try

```
chmod joeblow sekret_stuff.txt
```

But if joe is in the group `coolkids` (and obviously you are not), you would use it like so:

```
chmod joeblow:coolkids sekret_stuff.txt
```

You get the idea.