Exercises: Post FreeBSD Install: PacNOG I Workshop

June 20, 2005

Please do not change the root password on your machine for any reason!

Your instructor may indicate some updates or changes to these exercises. The initial exercises are to finalize our machine configurations for use later today, Tuesday and Wednesday, as well as to highlight some areas that are different from Linux - particularly methods by which you can install software.

During these exercises we reference the machine named "noc". This is the classroom server box. It's fully qualified name for this workshop is noc.pacnog.school.fj, but just "noc" should work as well.

Exercises

Final Configuration

1. Use pw to create a new userid that you will use instead of root
2. Use /stand/sysinstall and pkg_add to install lynx, bash, sudo and joe
3. Installing two more packages (gmake, unzip)
4. Installation of software using ports
5. Setup sudo for privileged commands
6. Installing/configuring X11 (Xorg), Gnome and gdm

Some Differences from Linux

7. Process startup and the RC system
8. Disk partitions: how to view your disk
9. Getting help using manpages, docs, and the FreeBSD Handbook
10. Shutdown and reboot your system

For Those who Want More Practice

11. Initial login, virtual terminals, command line manipulation
12. General job control (ctrl-c, ctrl-z, bg)
13. Processes and stopping them
14. Use top to see the state of your system
15. Create a file and use vi to edit the file
16. Using vipw to edit a user entry
17. Practice with basic filesystem commands
18. Command pipes and grep
19. Searching for more information about your system
20. File and directory permissions
21. Commands - programs - shell - path

To be Done Last by Everyone

22. Update your /usr/ports collection using cvsup

Note: The "#" and "$" characters before commands represents your system prompt and is not part of the command itself. "#" indicates a command issued as root while "$" indicates a command issued as a normal user.

Note 2: If you install software, update your environment as root and the change is not immediately available try typing `rehash` at the root shell prompt. This is only necessary when running a C shell (e.g., like /bin/csh).

1.) Use pw to create a new userid that you will use instead of root [Top]

First login to your computer if you have not already done so. Login as userid "root" using the password given at the start of class and that you used while installing FreeBSD.

Now that you are root you can create a new user account on your machine. If more than one person is using your machine, then be sure that you create an account for each and every person.

To create or remove a user account you should use the "pw" command. To get a feel for the power and complexity of this command take a look at its man pages:

```
# man pw
```

So, first pick a username that you want to use. For example, use your first name, last name, a combination of both, or whatever you prefer. For purposes of this example we use *username*. So, to add a new user to your system type:

```
# pw useradd username -m -s /usr/local/bin/bash
```

This creates the user *username*. The "-m" says add a new home directory of /usr/home/*username* and copies files from /usr/share/skel to the new user's home directory. The default shell "bash" will be used. Note that at this point the bash shell has not been installed on your machine. We'll be doing this in the upcoming exercises.

Next you need to set the password for the new userid, otherwise you won't be able to use the account properly. To do this use the `passwd` command like this:

```
# passwd username
```

You will be prompted to enter in a new password, then to enter it again to verify. You need to pick a secure password. Note, as root you are allowed to pick any password you want, but you need to pick a secure password for your new account. Here are some quick guidelines to picking a secure password:

- Password should be at least 6 characters in length.
- Don't use any words in a dictionary in any language - forwards or backwards.
- The password should contain a mix of letters, numbers, and upper and lower-case letters.

For example, something like "b00TaN2k5" is OK. Something like "pN0G2k5!" ("PacNOG 2005!") isn't bad either. Please don't pick either of these for this exercise.

So, finally, let's change your new user's password using the command:

```
# passwd username
```

2.) Use /stand/sysinstall and pkg_add to install lynx, bash, and joe [Top]

For this exercise you need to be logged in as root.

FreeBSD has a large collection of preconfigured binary packages for each release that can be installed in a number of ways, including:

1. Using the post-installation configuration utility `/stand/sysinstall` (don't use this now).
2. Directly from the FreeBSD installation CD-ROMs using pkg_add.
3. From a local network collection of FreeBSD packages specifying the network path and protocol using pkg_add.
4. Blindly across the network using the "-r" option of the pkg_add command

The core package utilities available in FreeBSD are:

```
pkg_add
pkg_delete
pkg_info
```

You can read about each of these package commands using the "man" command, i.e.:

```
# man pkg_delete

# man pkg_add
```

Remember, the pkg utilities under FreeBSD are very powerful as they can resolve dependencies, let you test installation before actually installing software, and keep track of what is installed with utilities to query the package database and assocated files.

For our first package we are going to install a text-based web browser called *lynx*. We'll use the /stand/sysinstall program to install lynx from the first disc of the FreeBSD CD-ROM installation set.

First,you can see if lynx is already installed on your system. Use the pkg_info command to do this. To get an idea of what packages are installed and what the default pkg_info output looks like type:

```
# pkg_info | more
```

You'll notice a pause while your machine prepares to show all the packages installed, in alphabetical order. Press space to scroll down the list, or ctrl-c to stop the list.

Now to check for just the lynx package type:

```
# pkg_info | grep lynx
or
# pkg_info lynx\*
```

If you don't understand what "grep" is doing type "man grep". In exercise 18 we go in to detail about the use of "*grep*", and the pipe facility represented above by the "|" character.

If lynx is already installed, for purposes of this exercise, first remove the package from our system (it's good practice!). You can do this by typing:

```
# pkg_delete lynx-2.8.5
```

Now type again:

```
# pkg_info | grep lynx
```

And, assuming that lynx is now not installed you should just get your prompt back.

Now to install lynx using the FreeBSD install CD-ROM do the following:

Insert the FreeBSD Disc 1 Installation CD in your machine's CD-ROM drive.
Be sure that you are logged in as the root user:

Now start the "sysinstall" utility by typing:

```
# /stand/sysinstall
```

At this point you take the following steps in the sysinstall menu system to choose the lynx package and install it:

- Choose "Configure" (you can press "C" and ENTER)
- Arrow down to "Packages" and press ENTER
- For installation media choose "CD/DVD" and press ENTER (note you can use network options here)
- If prompted choose "acd0" for your CD-ROM drive
- Scroll down the list of presented categories (using the down arrow key) until you see "www", and press ENTER
- Scroll down to "lynx-2.8.5" to highlight this then press the spacebar to choose the package, then press ENTER
- Now choose "Install" and press ENTER to install Lynx version 2.8.5 on your system

At this point you will be prompted for disc 2 of the FreeBSD installation set. Most of the

FreeBSD packages have been moved to this disc. Go ahead and follow the on-screen instructions to swap cd-roms.

- Once installation is complete you will be presented with the initial FreeBSD Configuration Menu - Scroll to the top of the screen and choose "X" for "Exit" and press ENTER
- Finally choose "Exit Install" to leave the "sysinstall" utility

After exiting this utility your FreeBSD install CD-ROM may still be mounted. You can type "df" and look for an entry like this:

```
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/acd0      660384   660384        0   100%    /cdrom
```

If you don't see the "/cdrom" entry, then type:

```
# mount /cdrom
```

again to make sure your CD-ROM (FreeBSD disc 2) is correctly mounted.

Installation of bash

Now let's install the bash package using a single command. As your FreeBSD CD-ROM is mounted under /cdrom, and as your instructor knows where the bash package resides on your CD, you could install bash by typing:

```
# pkg_add /cdrom/packages/shells/bash-3.0.16_1.tbz
```

But, rather than doing this, let's do this a bit differently. First use the *cd* (Change Directory) command to change to the initial cd-rom directory. Then we'll change to the directory /cdrom/packages/All. The "All" directory contains a list of all packages available to be installed on the cd-rom. It has the advantage that if you use pkg_add installing from this directory that if there are dependent packages that are needed, then they will be found and installed as well. So, first do:

```
# cd /cdrom

# cd packages/All
```

Now let's look for the bash package. We can use the *ls* command to do this with a wildcard. You'll be practicing and learning much more about these basic system commands in a few more exercises. OK, so do:

```
# ls bash*
```

You should see *bash-3.0.16_1.tbz* on your screen. This is how you know the exact name of the package you want to install. In addition, once you have installed and are using the bash package it will be easy to use *command completion* to figure out complicated filenames like these. You'll see more of this in exercise 7. So, now that you know you want to install the package called *bash-3.0.16_1.tbz* (we're going to use the newer version vs. the 2.05b version that's listed), and you are already in the directory where this file resides, then you can simply type:

```
# pkg_add bash-3.0.16_1.tbz
```

and the *bash* package will be installed. By the way, *bash* is an alternate shell environment that you can use which has more features than the default shell *sh*. Notice the file extension of "tbz" on the file bash-3.0.16_1.tbz, this means that the file is actually a group of files that have been compressed and saved as a single file.

Installation of joe

Some of you may practe the use of the "vi" editor, which is not as friendly as the optional "joe" editor. There are reasons for this, which we'll explain. But, for later exercises let's install "joe" right now, but this time we'll install from our local workshop server. If the server name is different than what is shown below your workshop instructor will let you know.

To install the "joe" editor from the local workshop server you can simply type the following:

```
# pkg_add ftp://noc/pub/FreeBSD/packages/All/joe-2.8_5.tbz
```

This will download and install the joe 2.8_5 package to your machine from the local workshop server. Note that this editor has been added by us. It no longer ships with the default FreeBSD installation.

Do a little bit of exploring about what you have just installed by entering the following commands:

```
# man lynx
```
(we'll discuss *man* a bit later)

```
# man bash

# man joe

# pkg_info lynx-2.8.5

# lynx www.freebsd.org

'q' (to 'q'uit lynx)

# joe

ctrl-c (to abort, you can say "no" to saving data)
```

3.) Installing two more packages (gmake, unzip) [Top]

For this exercise you need to be logged in as root.

During the week we will need two additional packages to do our work. The two packages we need are:

- **gmake**: The GNU Make utility.
- **unzip**: List, test and extract compressed files in a ZIP archive.

Both of these items are packages available on the first FreeBSD CD-ROM. We have placed these files on our classroom server in the /pub/FreeBSD/packages directory. So, to install these three packages simply issue the following three commands

```
# pkg_add ftp://noc/pub/FreeBSD/packages/All/gmake-3.80_2.tbz

# pkg_add uftp://noc/pub/FreeBSD/packages/All/unzip-5.52_1.tbz
```

If you want to learn more about these three packages you can type:

```
# man gmake

# man unzip
```

4.) Installation of software using ports [Top]

For this exercise you need to logged in as root.

The ports collection, which we installed during the FreeBSD install, is larger than the packages collection available on the first FreeBSD install cd-rom. As of June 2004 there were over 13,000 ports available! The FreeBSD Handbook has an excellent discussion of using Ports in section 4.5. In a nutshell these are the key points to remember about ports:

- The ports collection you installed contains descriptive files about how to make a software

package and where to get the actual source files. The actual source files (referred to as "distfiles") are not installed.
- You can tell your machine to find distfiles on a local server, or you can get them from CD-ROM, otherwise when you install a port file the software source will be downloaded from the Internet.
- You can use the cvsup tool to keep your ports collection up-to-date. That is, all the descriptive files of where to find source, version numbers, etc.
- Once you have installed a software package using ports it is possible to "deinstall" the package.
- There is a tool called portupgrade which will find all installed ports and keep them up-to-date.
- If you have a ports distfiles cd-rom it needs to be mounted as /cdrom for the ports utilities to find the distfiles.

The very first thing we are going to do is to update a master configuration file so that when you attempt to install a port it will look on our local server for the distfiles rather than using our single outgoing link to fetch distfiles from the internet.

To do this you should edit the file (use joe or vi) /etc/make.conf and add the following two lines to the end of the file:

```
MASTER_SITE_BACKUP=ftp://noc.pacnog.school.fj/pub/FreeBSD/ports/distfiles/${DIST_SUBDIR}/
MASTER_SITE_FREEBSD=yes
```

Remember to replace "noc" with what we gave you at the start of these exercises. Now that this is done we are going to install the following port:

- **lsof**: LiSt Open Files. Shows information about files opened by processes.

First do the following:

```
# cd /usr/ports
# make search name=lsof
```

This will output something like:

```
Port:   lsof-4.72.2
Path:   /usr/ports/sysutils/lsof
Info:   Lists information about open files (similar to fstat(1))
Maint:  obrien@FreeBSD.org
B-deps:
R-deps:
WWW:    http://people.freebsd.org/~abe/
```

From this you can see that the lsof utility resides in /usr/ports/sysutils/lsof. So, to install lsof on your system you do the following:

```
# cd /usr/ports/sysutils/lsof
# make
# make install
```

Make will download the lsof source (hopefully from our local server!), and then compile the source. "Make install" will place the compiled binary files in the appropriate directories on your system and update the appropriate configuration files if necessary. You can issue the single command "`make install`" instead.

You can now type:

```
# man lsof
```

For more information. Finally, if you want to deinstall a port once it is installed, for instance lsof, you would do the following (please don't do this):

```
# cd /usr/ports/sysutils/lsof
# make deinstall
```

And, if you decided that was a mistake, you can now do the following to reinstall a port (after it's been installed once):

```
# cd /usr/ports/sysutils/lsof
# make reinstall
```

If you do a "make clean" then "make reinstall" will no longer work for that port. If you want to see `lsof` in action you can type:

```
# lsof -i
```

In addition, FreeBSD installs the utilities `fstat`, and `sockstat` that can display the same information.

5.) Using sudo for privileged commands [Top]

When you created your own user account *username* you did not place it in the wheel group. This means that you cannot become the root user from your *username* using the command *su*. This is a useful thing to be able to do if you don't want to have to logout and log back in each time you need to be root. In addition, the change we are going to make to the "sudoers" file, which controls access to the *sudo* command is based on your *username* being in the wheel group. So, the first thing you need to do is be sure you are logged in as root:

```
$ exit

Login: root
```

Now we are going to use the *pw* command to make a change to your *username* account make it a member of the wheel group. The wheel group is a special group in Unix that allows members to become root. To do this type:

```
# pw usermod username -G wheel
```

The "-G" indicates "group list". If your user already belongs to other groups you *must* include all these groups in list format (e.g. "group1,group2,group3,etc.), otherwise your user will be *removed* from the other groups when you issue this command. To see what groups a user belongs to you can type:

```
# groups username
```

Once your user is a member of the wheel group you can use the su (Substitute User identity) command to become another user without having to login and logout of your session.

To practice this you need to login as your *username* account now:

```
# exit

$ Login: username
```

Now you can practice using *su*.

```
$ su - root
```

If you skip the "-" option, then you won't execute the root login scripts. In general you can type "su user", or "su - user", and switch to any user's environment.

Now let's drop back to your standard user account and try to do something that requires privileges:

```
# exit

$ less /etc/master.passwd
```

You should get the message, `"/etc/master.passwd: Permission denied"` - Now try running the same command like this:

```
$ sudo less /etc/master.passwd
```

You noticed that this did not work, right? First we need to add a package called "sudo" to your machine, then we need to configure the file /etc/sudoers to allow you to use this facility. Let's get the package installed first. We'll use pkg_add and we'll get it from our local server:

```
$ su - (you must be root to install this package)

# pkg_add ftp://noc/pub/FreeBSD/packages/All/sudo-1.6.8.7.tbz
```

Now we need to configure the file /etc/sudoers so that you can use the `sudo` command:

```
# visudo
```

This is a special utility to edit the /etc/sudoers file using vi, but to lock the file so that no one else can edit it while you are using `visudo`. So, remember your vi commands and scroll down the screen until you find the following lines in the file (if you need help with vi ask your instructor or one of the classroom helpers):

```
## root and users in group wheel can run anything on any machine as any user
#root           ALL = (ALL) ALL
#%wheel         ALL = (ALL) ALL
```

(Note/hint: there are other ways of finding this line in vi. You could search for something using "/".).

Now, once you are here, just make one, small change. Remove the "#" symbol (it's a comment symbol) from in front of "%wheel". Once you do this save and quit from the file. So, your file should now look like:

```
## root and users in group wheel can run anything on any machine as any user
#root           ALL = (ALL) ALL
%wheel         ALL = (ALL) ALL
```

And, to save and quite it's:

```
:wq
```

That's it. You are now ready to run "sudo" as a general user (or, at least a general user in the wheel group). Make sure you exit from the root shell you are in:

```
# exit
```

And, now as a general user, try issuing the command:

```
$ sudo less /etc/master.passwd
```

You will be prompted for a password, and the first time you use sudo you'll receive a warning. The password you should enter will be your user account's password. This is different from Linux. Type this in, and you should be able to view the /etc/master.passwd file even though you are running as a standard user. Try doing this again ("less /etc/master.passwd") and you'll notice that you are no longer prompted for the root password. You can issue any privileged command (pretty much) without needing a password at this point. Once you login and logout again, then you'll have to enter a password the next time you use sudo.

sudo is very useful to allow you to do system administration tasks on your machine without needing to be logged in as root. This can help to protect you from making mistakes running as root, which can be costly.

6.) Installing/configuring X11 (Xorg), Gnome and gdm [Top]

For this exercise you will need to be logged in as root.

Before we begin there are a few items to understand to put this in perspective:

- A much more complete discussion of using X and Desktops under FreeBSD can be found in Chapter 5, The X Windows System, of the FreeBSD Handbook. I *highly* recommend reading this entire chapter if you wish to fully configure your X Window environment for FreeBSD.
- FreeBSD is a server-based operating system. Configuration of X, display managers, and desktops like Gnome and KDE are not done during installation as this is not the objective of the OS.
- You can configure FreeBSD to autostart in to a GUI environment, but I would not recommend this, except, possibly, for desktop and laptop use.
- To automate the use of X Window you can edit /etc/ttys (see the Handbook), xinitrc, .xinitrc, .xsession, Xsession, and KDE config files to your liking. For our purposes we are going to install Gnome and manually call the Gnome Dislpay Manager (gdm) when we wish to use an X Window environment.
- X11 is the implementation of the X Windows environment. Xorg is the X11 implementation that FreeBSD uses by default. You can still use XFree86 if you wish, but configuring Xorg is easier. Gnome and KDE are Desktop environments that include Window managers and entire sets of desktop applications.
- You can clean up and make your font environment considerably nicer under X, particularly fonts used by OpenOffice, Netscape/Mozilla/Firefox, and The Gimp. This is easier to do using Xorg. Read Chapter 5 of the FreeBSD Handbook for more details.
- The Gnome and KDE installations that come with the FreeBSD 5.4 CD-ROM installer are quite up-to-date (as of June 2005). The Gnome version is 2.10 and KDE is version 3.4. Gnome version 2.10.1 is available via the ports tree, and you can read about Gnome on FreeBSD here:

  http://www.freebsd.org/gnome/

  You can read about KDE on FreeBSD here:

  http://freebsd.kde.org/

- If you plan on installing both Gnome and KDE and/or possibly offering up Desktop environment via your network from a server, then you will most likely use the KDE Display Manager (kdm) or, possibly the X Display Manager (xdm). kdm can offer access to both Gnome and KDE Desktop environments at loging.

For this workshop we are going to install Gnome from your FreeBSD CD-ROM install set and we are going to use gdm manually to start our X Window environment. We have already configured an Xorg configuration file for your particular machines setting the correct monitor refresh rates and setting your desktops to use 24-bit color at a depth of 1024x768. You will be copying this file from our classroom server, noc, to the appropriate location.

During our initial installation of FreeBSD we installed Xorg, but we have not yet configured it to run on our hardware. First, let's install Gnome using the `sysinstall` utility.

Start the `sysinstall` utility (as root) by typing:

```
# /stand/sysinstall
```

At this point you take the following steps in the sysinstall menu system to choose the Gnome packages and install them:

- Choose "Configure" (you can press "C" and ENTER)
- Arrow down to "Packages" and press ENTER
- For installation media choose "CD/DVD" and press ENTER (note you can use network options here)

- If prompted choose "acd0" for your CD-ROM drive
- Scroll down the list of presented categories (using the down arrow key) until you see "gnome", and press ENTER
- Scroll down the list to "gnome2-2.10.0" to highlight this then press the spacebar to choose the package, then press ENTER

  You should have noticed that multiple Gnome packages are automatically chosen as dependencies (a "D" appears in the selection boxes vs. an "X") when you choose just gnome2-2.10.0.

- Now choose "Install" and press ENTER to install Gnome version 2.10.0 on your system

  At this point you will be prompted for disc 2 of the FreeBSD installation set. Most of the FreeBSD packages have been moved to this disc. Go ahead and follow the on-screen instructions to swap cd-roms.

- Once installation is complete you will be presented with the initial FreeBSD Configuration Menu - Scroll to the top of the screen and choose "X" for "Exit" and press ENTER
- Finally choose "Exit Install" to leave the "sysinstall" utility

At this point both Xorg (the X11 X Window system) and Gnome are installed on your system.

Configuration follows.

Some Differences from Linux

7.) Process startup and the RC system [Top]

This is a rather complex topic. But, in a nutshell, under FreeBSD 5.4, when your machine boots processes (that is daemons or services) are configured and/or started like this:

- Items that are configured or referenced to start in the file /etc/defaults/rc.conf contain corresponding script entries in /etc/rc.d/.
- You can override settings in /etc/defaults/rc.conf by placing settings in /etc/rc.conf.
- Every script in /etc/rc.d/ will attempt to run, but if it does not see:
  script_enable="YES"
  in /etc/defaults/rc.conf or /etc/rc.conf then the script will not run to start the corresponding service.
- Some third party software will place entries in /etc/rc.conf and corresponding third party startup script packages in /usr/local/etc/rc.d/.
- Some third party packages will place startup/config scripts in /usr/local/etc/rc.d/ that do not look in any file, but simply execute upon system boot.
- Some older software may still place startup items in /etc/rc.local, but this has been deprecated, even though it is still supported.

To get a better feel for this you should read:

```
$ man rc
```

Then, you should probably read this again...

Now, if you want to start a process each time your machine boot you generally add an item to /etc/rc.conf to indicate this. For instance, if you wanted to enable the ssh daemon (server) each time your machine started then you would add the line:

```
sshd_enable="YES"
```

to /etc/rc.conf. If you look at /etc/defaults/rc.conf you'll see that sshd is not enabled in this file by default. By enabling sshd in /etc/rc.conf this overrides the setting in /etc/defaults/rc.conf. In

addition, if you look in /etc/rc.d/ you'll find an sshd script file that starts this service.

To see how sshd is enabled initially do this:

```
$ grep sshd /etc/defaults/rc.conf
```

You should see something like:

```
sshd_enable="NO"                # Enable sshd
sshd_program="/usr/sbin/sshd"   # path to sshd, if you want a different one.
sshd_flags=""                   # Additional flags for sshd.
```

These are the lines in /etc/defaults/rc.conf that deal with the ssh daemon. When you specified to enable the ssh daemon during installation then in the file /etc/rc.conf the lines that read:

```
sshd_program="/usr/sbin/sshd"   # path to sshd, if you want a different one.
sshd_flags=""                   # Additional flags for sshd.
```

became active. So, if your ssh program was not "/usr/sbin/sshd" for some reason, then in /etc/rc.conf you could add:

```
sshd_program="/new/directory/sshd"   # new path to sshd
```

to override what's in /etc/defaults/rc.conf.

You could just put everything in /etc/defaults/rc.conf, but you don't want to do this. If you upgrade your system it's almost certain that /etc/defaults/rc.conf could be overwritten. In addition, this file is large and it would be hard to see the changes you had made if you were to do them in /etc/defaults/rc.conf.

To start a service manually you can use it's startup script by hand. For instance, try typing:

```
$ su - (you must be root to run these commands)

# /etc/rc.d/sshd
```

What is returned on the screen? It should be something like:

```
Usage: /etc/rc.d/sshd [fast|force|one](start stop restart rcvar keygen reload status poll)
```

So, you could type

```
# /etc/rc.d/sshd status
```

to see if ssh is running. If it is, then try:

```
# /etc/rc.d/sshd stop
```

to start the service. Now type:

```
# /etc/rc.d/sshd start
```

to restart the service. Note the startup script option "reload" - This would let you make changes to the ssh configuration file(s) and then reload the service without actually stopping it so that it reads the new configuration. Note that already connected clients would not see this new configuration change until the logoff and log back in again.

Finally, and this is not obvious, if a startup script in /etc/rc.d has not been enabled in /etc/defaults/rc.conf or /etc/rc.conf, then even if you manually invoke the script it will not run. You will not get any indication of this other than the service not starting (i.e. use "*ps auxw| grep servicename*" and you won't see it started).

At this point take a closer look at /etc/rc.d/

```
# ls /etc/rc.d
```

and use "man" to read about some of the services. If you want to try and start and stop some of these services feel free to do so now, but remember you'll need to add 'servicename_enable="YES"' in /etc/rc.conf for the service to start.

8.) Disk partitions: how to view your disk [Top]

First, make sure you are logged in as your user and not as root.

Now in a terminal lets look at the partitions. Type:

```
$ df
$ df -h
```

What difference did you see between "df" and "df -h". How can you see what your swap contains (note it was not listed using "df")? Use this:

```
$ swapinfo
```

If you want to see more detailed information about your disk slices you can use the "fdisk" command. As a general user you are not allowed to run this program, so you must use sudo. Try this by typing:

```
$ sudo fdisk
```

Be careful with fdisk as you can remove slices, partitions, etc.

If you are interested in how much space files are taking up in a directory or a group of directories you can use the "du" command. Try it out by typing:

```
$ du

$ du -h
```

As usual you can get more information by typing "man du".

Now that you've seen these methods for viewing disk information, you should be able to understand the output of the *disklabel* command a little better. The *disklabel* command is really the proper way to view the status of partitions in your FreeBSD slice at it will show you all your FreeBSD slice information no matter if a partition is mounted or not. In some cases this is critical if you are trying to troubleshoot problems and not all partitions have mounted. So, if FreeBSD resides on the slice /dev/ad0s1 on your disk, then to see disk information you can type (as a normal user): `$ sudo disklabel /dev/ad0s1`

This shows you output along these lines (will be different for this workshop):

```
# /dev/ad0s2:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:   409600        0    4.2BSD     2048 16384 25608
  b:  1433600   409600      swap
  c:  8385930        0    unused        0     0         # "raw" part, don't edit

  d:  1843200  1843200    4.2BSD     2048 16384 28552
  e:  3072000  3686400    4.2BSD     2048 16384 28552
  f:  1627530  6758400    4.2BSD     2048 16384 28552
```

As you can see it includes the "c" partition and "b" or the "swap" partition as well as filetypes.

9.) Getting help using manpages, docs, and the FreeBSD Handbook [Top]

Now that you have FreeBSD up and running you probably want to have a way to figure out how to use it when there are no instructors around, or other FreeBSD-knowledgeable people. First and foremost, make it a habit to read the man pages (MANual pages) for the commands that you use. You might be surprised at some of the things these commands can do! In any case, as you have seen, when in doubt about what a command does, or how it works simply type:

```
$ man command
```

Optionally, you might find some additional information for some commands typing:

```
$ info command
```

And, to get information about both these commands type:

```
$ man man

$ info info
```

After this there is a large amount of documenation available to you in serveral ways. For instance, if you look in:

/usr/share/doc

you will find multiple FreeBSD articles in various languages available to you. In addition, the FreeBSD Handbook is available here under /usr/share/doc/handblook. If you wanted to start reading the FreeBSD Handbook from your local har drive you could either point a web browser to the file /usr/share/doc/handbook/index.html, or you could type the command:

```
$ lynx /usr/share/doc/handbook/index.html
```

Go ahead and type this command now to get a feel for what information is available to you in the FreeBSD Handbook. Now let's look at the FreeBSD FAQ by typing:

```
$ lynx /usr/share/doc/faq/index.html
```

After this, have a look at some of the available articles by doing:

```
$ cd /usr/share/doc/en/articles

$ ls
```

Finally, there are several papers available as well. Try:

```
$ ls /usr/share/doc/papers
```

If you have a network connection, then you can go to `http://www.freebsd.org/docs.html` for even more information. Become accustomed to the idea of using man to get specific information about commands, and then using these additional resources to get an overview of entire sub-systems of the FreeBSD operating system. If you want to read the FreeBSD Handbook online it is available here `http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html`. If you want to understand FreeBSD concepts in more depth, then the FreeBSD Handbook is *really* where you should begin.

10.) Shutdown and reboot your system [Top]

For this exercise you need to be root. It is better to close open files and programs (for example Firefox, vi, etc.), but it is not necessary. Before continuing read the man pages for shutdown, init, halt, and reboot (you'll see they are all connected):

```
# man shutdown
# man init
# man reboot
# man halt
```

Now, in a terminal do the following (save data, etc. as this will immediately reboot your machine!):

```
# shutdown -r now
```

Now your machine is rebooting. The "-r" stand for "reboot" and the "now" meant to take this action "now". This takes a moment. To stop your machine entirely you can use the command:

```
# halt
```

Or, you can also change your run level to run level 0, which is the same as "halt". So, you would write:

```
# init 0
```

And, to reboot this is the same as init 6, or:

```
# init 6
```

If you are running something like gdm for a graphical login prompt on your machine you can usually use provided menu choices to reboot or shutdown. The thinking is that once you have this level of access, then you can simply turn off the machine's power if you wish. At the very least it is much more friendly to use a software interface to shutdown or reboot than pulling the power as processes have a chance to clean up, save data, etc.

Note, sometimes it is useful to bring your machine down to runlevel 1, or "single user mode". For instance, if you are running X Windows and want to shut it down quickly (you'd really only do this on a desktop machine, by the way!), then you can open a terminal window as root (or use "su"), and then type"

```
# init 1
```

This will shut down X Windows, networking, and quite a bit more. Now you are in "single user mode". To get back to "multi-user mode" you simply type:

```
# exit
```

This exits your single-user modem shell and tells the system to go back to multi-user mode. Notice that the "runlevel" concept under FreeBSD is different than under Linux. The major difference are:

- Linux uses runlevels 1, 3, and 5 (single user, network with no GUI, network with GUI [X]).
- FreeBSD uses runlevel 1, single user mode. To get back to network mode with or without GUI you `exit` runlevel 1 (or you can reboot).
- You can use `init c` if you with to block further logins under FreeBSD.
- Both FreeBSD and Linux use runlevels 0 and 6 in the same manner (halt and reboot).

For Those who Want More Practice

11.) Initial login, virtual terminals, command line manipulation [Top]

The first time you login on your system after installation you will be presented with a prompt that looks something like this:

```
FreeBSD/i386 (name.domain) (ttyv0)

login:
```

At this point you can enter in "root" and, when prompted, the password we gave you in class for the root account. Once you are logged in you can work with an additional 7 virtual terminals if you wish. Actually, you can login from any virtual terminal you want at any time. To do this simply press:

ALT-FN

With "FN" being anything from the F1 to the F8 key. By default you are in ttyv0 which corresponds to the

ALT-F1 keyboard sequence. Go ahead and login and then press:

ALT-F2

and login again. Feel free to do this on F3, F4, F8, etc. as you wish. You can cycle through each terminal session easily. This is an extremely useful technique when you wish to do more than one thing at the same time, but you are not using a graphical interface. Since we are loading the mouse daemon (mouse support) you can, also, copy and paste text between your virtual terminals. To do this go back to your initial login terminal by pressing:

ALT-F1

Try typing in the command:

```
$ clear
```

and pressing ENTER. Note that your screen clears and goes to the top. Now, you can easily recover your last command by pressing the UP-ARROW key on your keyboard once. Get used to doing this as it can be very useful if you have entered in a long command and made a mistake. You can press UP-ARROW to get the command back, then you can use the LEFT-ARROW to move your cursor to where the mistake is and correct it, then simply press ENTER to reissue the command. Note, you *do not* need to go back to the end of the command line before pressing ENTER.

In any case, you've pressed UP-ARROW once and should have the command "clear" visible. Now take your mouse and highlight just the command using the left mouse button. Without doing anything else with your mouse now press:

ALT-F2

to get to one of your other terminal sessions. Now just press the *middle* mouse button once. What happened? The text "clear" should have pasted on to your command line. Now you can press ENTER to execute the command. You can use this same trick to copy and paste in to editor windows, long and complex commands, between applications in a graphical environment, etc.

One final useful tip when working in your terminal sessions. As you type each command it is being saved in to a file called your "history file". This has a very useful purpose. Go back to your original login terminal pressing:

ALT-F1

and type the command:

```
$ history
```

You should see a list of commands you have entered in earlier. This list is probably short and it will even include incorrect commands you may have typed in. To quickly and immediately recover and execute a prior command make note of the number in the left-hand column next to the command and then just type:

```
$ !N
```

Where N is the number. So, if "clear" had been the second command, and you typed in:

```
$ !2
```

Then clear would appear on the command line very quickly and immediately execute. During the week you are going to be typing in some long and complex commands. The use of history can save you considerable time. If you press the UP-ARROW key repeatedly you can scroll through the previous commands you have entered beginning with the last command. Give it a try.

One *very useful* trick when using XWindows. If you need to escape your graphical environment and go to a console for some reason you can use the same virtual terminals, except that you press the keyboard combination:

ALT-CTRL-F1 through F8

At this point you will exit your graphical environemnt (KDE, Gnome, etc.) and be presented with a text console login prompt. Your graphical environment is still running as you left it. To get back to your graphical environment press:

ALT-CTRL-F9

Why might you need to do this? Here's one scenario:

You've just installed your favorite MTA (email server/Mail Transport Agent) and you typed something like (don't do this now):

```
pw usermod username -G mail
```

to place your user in the "mail" group. What just happened if your user was also a member of the "wheel" group and this is how you are using `su` and `sudo`? You can no longer become root in your session. To fix this you could go to a virtual console window, log in as root, and reissue the command above, but this time correctly:

```
pw usermod username -G wheel,mail
```

To fix this problem.

12.) General job control (ctrl-c, ctrl-z, bg) [Top]

For this exercise you need to be logged in as root.

When you wish to stop an active process in your shell you can use the keyboard combination "ctrl-c". If this does not work, then you may want to try "ctrl-z". The ctrl-z sequence will suspend the process allowing you to place it in the background using the `bg` (BackGround) command. Generally this command is most useful if you are running in a graphical and have started a service that has locked you out of a shell window. Here are some examples to try to demonstrate these concepts:

```
# cd /
# ls -R (starts to recursively show contents of all directories on your system)
press CTRL-C (aborts the output)
```

In addition, when you type "man command" you can press the "q" key to exit from viewing the man pages. The same goes for "less" as well.

Now let's start a process in your shell to requires you to use ctrl-c to end the process:

```
# tail -f /var/log/messages
```

In this example you are viewing the end of the main log message for your system with data appended to the output as the file grows. That is, if a message is generated and placed at the *end* of the log file you will see this message interactively appear on the screen. This is a very useful tool when you are trying to debug problems on your system. You can open one terminal window (as root, or with an account that can run privileged commands), type "tail -f /var/log/messages", start a process in another terminal window and then view the end results in the window where the *tail* command is running.

Now to end the "tail -f" process press ctrl-c to abort the output. In this case "q" or "ESC" will not work. You need to know about ctrl-c to do this.

13.) Processes and stopping them [Top]

For this exercise please be sure you are logged in as root.

If you would like to see what is running on your system, then you use the "ps" command (ProceSs). For

example, to see everything running on your system for all users, and even items running that are detached type:

```
# ps auxw
```

If you find something that you wish to stop (maybe your web browser session has hung), then you can look for the process ID number, and you can issue a "kill" command to stop the process. This is a *very* powerful feature of UNIX. If you need to kill a process for another user then you must have privileges to do this, usually root. The example we are going to use is to kill an account login in another terminal. We'll be going back and forth from one terminal to another and we'll use tail, /var/log/messages, and ps for this exercise. First, let's start an interactive *tail* command of our /var/log/messages file:

```
# tail -f /var/log/messages
```

Now switch to a second terminal and log in again as root:

```
ALT-F2

Login: root
```

Switch back to your original login terminal where the interactive *tail* command is running by pressing ALT-F1. What do you see. You should see something indicating that root logged in on terminal ttyv1. More specifically the entry will look basically like this:

```
Jan  10 17:21:05 localhost login: ROOT LOGIN (root) ON ttyv1
```

OK, now break out of the *tail -f* process by pressing "CTRL-C". Now let's find the process ID of this new login session. To do this type:

```
# ps auxw| grep login
```

If you had just typed "ps -aux" you would have seen all processes and then you may have figured out that the "login" process of your other root login can be found by looking for the login keyword.

Now we are going to kill the login shell of your other root login on ttyv1. But, how do you know which process to kill. You probably saw something like this when you did "ps auxw| grep login":

```
root   1060  0.0  0.3  1612 1308  v0  Is    17:22PM   0:00.03 login [pam] (login)
root   1061  0.0  0.3  1612 1308  v1  Is    17:22PM   0:00.03 login [pam] (login)
```

These look almost identical, but note the process IDs. One is 1060 and the other is 1061. This should be your clue. Your *first* root login will have a lower process ID number than your next root login. In addition, if you had waited and started other processes before logging in your second root login, then the process ID numbers might be separated by more than 1. So, in this case you will want to stop the process ID 1061. **Note:** Naturally on your system the actual process ID numbers will almost certainly be different. Now, to stop process UD 1061 do this:

```
# kill 1061
```

Now go back to the other terminal and see if the process has been ended. That is press ALT-F2 and see if the "`Login:`" prompt appears. If it does, then you just forced the user off the system. If the "`Login:`" prompt does not appear, then you may need to use a more forceful *kill* command in the form:

```
# kill -9 1061
```

But, *be careful* this shuts down the process without giving it any chance to save data, remove lock files, or generally clean up. I.E. you could lose or corrupt data.

In general, if you do:

```
# ps auxw | more
```

And you see items running that you do not want to have run each time you boot, then you can, generally, edit /etc/rc.conf and override the setting that starts this service (see /etc/defaults/rc.conf). You can stop the service immediately by using the "kill" command. If you are testing a configuration file for a known running service (say the Apache web server) you can, often, tell the service to restart reloading the configuration file, but to

reload using the same parameters it originally used to start. This can be very useful if the service requires a complex set of parameters to restart. The command to do this is:

```
# kill HUP nnnn
```

Try using the ps command. See if there is anything you can stop by using the kill command. Note, you could cause all sorts of interesting behavior if you do this with running services that you need. I suggest doing this exercise as your standard user, not root, and then starting some program, finding it using "ps auxw| grep progname" and then using kill to stop it.

Finally, some programs spawn many processes when running. Web browsers are an example of this. It can be time-consuming to kill each process one-by-one until you have completely shut down the program. An alternate command is "killall" which will kill processes by name. Naturally you have to be a bit careful with this as you could shutdown something unexpectedly, but if you have 20 processes all called "Mozilla" that you want to stop, then you can simply type:

```
# killall mozilla
```

I suggest reading the man page ("man killall") about this command before using it regularly.

14.) Use top to see the state of your system [Top]

If you want to see what processes are using how many resources, then use the command:

```
$ top
```

Notice that not only do you get to see what's running, but you get to see a bunch of system information as well at the top of the screen. For instance, on my laptop running KDE this is some of what I might see:

```
last pid:  2707;  load averages:  0.17,  0.26,  0.19      up 0+01:43:52  01:16:24
47 processes:  1 running, 46 sleeping
CPU states:     % user,     % nice,     % system,     % interrupt,     % idle
Mem: 98M Active, 104M Inact, 55M Wired, 20M Cache, 47M Buf, 91M Free
Swap: 700M Total, 700M Free

  PID USERNAME PRI NICE   SIZE    RES STATE    TIME   WCPU    CPU COMMAND
 2421 root      96    0 27988K 19980K select   0:14  0.24%  0.24% Xorg
 2699 root      96    0 28792K 19388K select   0:01  0.10%  0.10% kdeinit
 2476 root      20  -76 13456K  8680K kserel   0:09  0.00%  0.00% artsd
 2506 root      96    0 36344K 26120K select   0:06  0.00%  0.00% kdeinit
 2504 root      96    0 10444K  8644K select   0:04  0.00%  0.00% emacs
 2488 root      96    0 28724K 20116K select   0:03  0.00%  0.00% kdeinit
 2467 root      96    0 26904K 17772K select   0:03  0.00%  0.00% kdeinit
 2480 root      96    0 32100K 20392K select   0:03  0.00%  0.00% kdeinit
 2499 root      96    0 28792K 19276K select   0:02  0.00%  0.00% kdeinit
 2484 root      96    0 26228K 17576K select   0:02  0.00%  0.00% kdeinit
 2486 root      96    0 27044K 18636K select   0:01  0.00%  0.00% kdeinit
 2267 root      96    0  1232K   688K select   0:01  0.00%  0.00% moused
 2498 root      96    0 27736K 18428K select   0:01  0.00%  0.00% korgac
 2491 root      96    0 25692K 17140K select   0:01  0.00%  0.00% kdeinit
 2494 root      96    0 25160K 16240K select   0:00  0.00%  0.00% kdeinit
 2483 root      96    0 24836K 15896K select   0:00  0.00%  0.00% kdeinit
 2458 root      96    0 24020K 14412K select   0:00  0.00%  0.00% kdeinit
 2461 root      96    0 23408K 13836K select   0:00  0.00%  0.00% kdeinit
 2464 root      96    0 24844K 15132K select   0:00  0.00%  0.00% kdeinit
 2500 root       5    0  2388K  1852K ttyin    0:00  0.00%  0.00% csh
 2057 root      96    0  1784K  1188K select   0:00  0.00%  0.00% dhclient
 2700 root      20    0  2272K  1624K pause    0:00  0.00%  0.00% csh
 2432 root       8    0  1640K  1112K wait     0:00  0.00%  0.00% sh
 2414 root       8    0  1612K  1184K wait     0:00  0.00%  0.00% login
 2100 root      96    0  1312K   812K select   0:00  0.00%  0.00% syslogd
 2416 root       5    0  2276K  1680K ttyin    0:00  0.00%  0.00% csh
 2489 root      96    0 24836K 14780K select   0:00  0.00%  0.00% kdeinit
 2481 root       8    0  1312K   708K nanslp   0:00  0.00%  0.00% kwrapper
 2422 root       8    0  2652K  1640K wait     0:00  0.00%  0.00% kdm
 2233 root       8    0  1356K   964K nanslp   0:00  0.00%  0.00% cron
 2419 root      96    0  2332K  1168K select   0:00  0.00%  0.00% kdm
 2707 root      96    0  2256K  1416K RUN      0:00  0.00%  0.00% top
 2353 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
 2415 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
```

```
 2303 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
 2307 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
 2304 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
 2306 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
 2305 root       5    0  1280K   832K ttyin    0:00  0.00%  0.00% getty
```

That's a lot of information, but top will very quickly give you a feel for your overall system state, what's using most of your resources, whether you have a user who is hogging your cpu, another your memory, etc. Pay special attention to things like load averages and RAM usage.

Load averages are listed for the past 1, 5, and 15 minutes. Numbers from 1.0 to less are generally OK. If you start to see consistent load averages around 2, or above, then your machine is very busy and you probably need to consider tuning one of your subsystems, improving hardware, buying a new machine, or checking for inefficient use of resources.

Several other ways to see load averages and machine usage include:

```
$ w
```

$ uptime

Try using "w" and "uptime" and read the man pages for these commands.

15.) Create a file and use vi to edit the file [Top]

For this exercise please login as your *username*. Type `whoami` or id if you are not sure if you are logged in as root.If you are logged in as root, then type:

```
# exit

$ Login: username
```

Now we are going to open an empty file and write something in it.

The vi editor uses "modes"

This is a critical point. The vi editor has two modes. These are:

- Command mode
- Input mode

To go back and forth between these modes when you are in vi you can press:

- ESCape key (command mode)
- Letter "i" for Input mode, letter "o" for input mode with newline below cursor

Remember this as it is confusing. The easiest thing to do when you get confused in vi is to press the ESCape key a couple of times and start over.

Now let's do the following:

```
$ cd /home/username<
$ touch temp.txt
$ vi temp.txt
```

Now you are in vi. Press the "i" key to switch to input mode.

Type something like, "VI is great! I think I'll be using vi from now on instead of Microsoft Word."

Press ENTER to add lines. Type some more stuff, whatever you like.

Here is a short list of vi commands:

```
Open: vi fn, vi -r fn, vi + fn, vi +n fn, vi +/pat fn
Close: :w, :w!, :wq, :wq!, :q, :q!
Movement: h,j,k,l, w, W, b, B, :n
Editing: i, o, x, D, dd, yy, p, u
Searching: /pattern, ?pattern, n, N
```

OK, let's save the file that you are in. To do this do:

```
Press the ESCape key to get in to command mode
```

Press ":" to get ready to issue a file command

Type "w" and press ENTER to save your file.

Press ":" to get back to the prompt to issue a file command

Press "q" to quite the file

Instead of the multiple steps you could have type ":wq" to write and quite at the same time. If you need to quit a file without saving it *after* you've made changes, then you press **:q!**. For many people this is the most important command to remember in vi :-).

Below is a more complete vi cheat sheet. In addition you will be receive a vi summary book as part of the book package for this workshop.

vi Cheat Sheet

Open:

```
vi filename             (fn=filename)
vi -r filename          Recover a file from a crashed session
vi + filename           Place the cursor on last line of file.
vi +n filename          Place the cursor on line "n" of file.
vi +/pat filename       Place cursor on first occurrence of "pat"tern
```

Close:

```
:w                      Write the file to disk. Don't exit.
:w!                     Write the file to disk even if read/only.
:wq                     Write the file to disk and exit.
:wq!                    Write the file to disk even if read/only and quit.
:q                      Quit the file (only if no changes).
:q!                     Quite the file even if changes.
```

Movement:

```
A                       Move to end of line, change to insert mode.
h                       Move 1 space backwards (back/left arrow).
j                       Move down 1 line (down arrow).
k                       Move up 1 line (up arrow).
l                       Move 1 space forwards (forward/right arrow)
w                       Move cursor to start of next word.
W                       Same as "w".
b                       Move cursor to start of previous word.
B                       Same as "b".
:n                      Go to line number "n" in the file.
```

Editing:

```
i                       Enter in to input mode.
o                       Add a line below cursor and enter in to input mode.
x                       Delete character (del key in some cases).
D                       Delete line from right of cursor to end of line.
dd                      Delete entire line.
u                       Undo last edit or restore current line.
p                       Put yanked text before the cursor.
yy                      Yank current line.
```

Searching:

```
/pattern               Search for "pattern" in the file going forwards.
?pattern               Search for "pattern" in the file going backwards.
n                      Find the next occurrence of pattern found forwards.
N                      Find next occurrence of patter found backwards.
```

```
Copy/Cut and Paste
nyyp                    Copy n lines to buffer, paste below cursor
nyyP                    Copy n lines to buffer, paste above cursor
nddp                    Cut n lines and copy to buffer, paste below cursor
nddP                    Cut n lines and copy to buffer, paste above cursor
```

Now let's go back in to the file you were just editing. To do this type:

```
$ vi temp.txt
```

Play with moving around. Move your cursor to a line with text and see what happens when you go in to command mode (ESCape) and use "w" or "W" or "b" or "B" - remember, to get in to command mode press the ESCape key.

Now press "/" and type a word that is in your document, then press ENTER. What happens?

Do the same, but press the "?" key at first. Use ESCape to start in command again again if necessary.

To save your file press the ":" key and next type "w" and enter . .

To exit and save do:

:wq

To exit and not save anything (lose all changes you have made since the last save) do:

:q!

But, try to save your file for later use. Practice saving, exiting, opening a file in vi again, etc.

16.) Using vipw to edit a user entry [Top]

You need to be logged in as root for this exercise.

When you log in as root, or one of the userids you created on your box you are automatically given a "shell" in which to work. There are many different shell environments under Unix and each one has its own unique set of features and commands. The root user should log in using the "csh" shell (located in /bin/csh). This is to ensure that a shell is always available to this account if there are problems and not all filesystems can be mounted. The general user, however, can take advantage of some more powerful shell programs, such as bash. Bash (under FreeBSD) resides in the /usr/local/bin directory. This directory may not be available if there are problems at system boot time, thus it is not recommended that you switch your root account to use bash.

One of the nicest features of bash is (remember?) "command completion" - This allows you to type some piece of a command then press the tab key to complete the command. If there is more than one possibility you press the tab key twice in rapid succession and all possible completions will be displayed. When we were installing bash from /cdrom/pkgs/editors and the filename was "bash-3.0.16_1.tbz" this feature can be very useful. If you are using the bash shell, then rather than having to type "bash-3.0.16_1.tbz" exactly, you could have simply typed:

```
# pkg_add /cdrom/packages/All/bash
```

and pressed the tab key to complete the command. In addition, you can press tab twice to see all possibilities in a directory, which can be useful when trying to decide if you are in the right place.

Earlier we switched your account to use the bash shell, but let's look at another way to do this. In addition, at this point you could change the name associated with your username, and other items as well. Rather than use the "pw usermod" command we are going to use "`vipw`". This command will open the file /etc/passwd using the vi editor, but with a wrapper process that prevents other users from updating the file while we are editing it. This is important because if two people on your system tried to edit /etc/passwd at the same time, then it could become corrupted, which could cause problems for everyone attempting to log in to your system.

So, let's edit /etc/passwd using the vi editor:

```
# vipw
```

Now you will be in the /etc/passwd file. You can scroll down to the line that contains your username entry to get ready to make some changes. The line will look something like this:

```
username:#1#swefo/qx#IUYRRAD09DAAD.:1001:0::0:0:User &:/home/username:/usr/local/bin/bash
```

In this case notice that the name position does not contain any real meaningful information. We could change this to read:

```
username:#1#swefo/qx#IUYRRAD09DAAD.:1001:0::0:0:New User:/home/username:/usr/local/bin/bash
```

In addition, note that the last entry on this line is ":/usr/local/bin/bash" - this indicates that the bash shell will be used for this account. If you wanted to switch this account back to using the "sh" shell you could change the line to read:

```
username:#1#swefo/qx#IUYRRAD09DAAD.:1001:0::0:0:New User:/home/username:/bin/sh
```

But, I would not recommend this as the bash shell is much nicer.

To exit from the /etc/passwd file issue the vi command:

```
:q (to quit if no changes made)
:wq (to write changes and quite)
:q! (to quit and not save changes if any made)
```

It is your choice whether you wish to use the `vipw` or the `pw` command to manipulate your user's accounts.

Now, assuming you left the shell for your account as /usr/local/bin/bash, log out as the root user and log back in using your userid. You can either type `logout` or `exit` to do this.

Try playing around a bit with bash. For instance, type:

```
$ ls /
```

Then hit the tab key twice quickly to see all the directories under the root or "/" directory. We'll be stressing the use of command completion repeatedly throughout the week. This is one the largest timesavers available to you when using the Unix command-line environment.

17.) Practice with basic filesystem commands [Top]

Be careful in this exercise. Running as root means that you can easily damage your system, so we ask that you log out of your root account and log in as your own user account instead.

If you are not sure of a command ask the instructor or helpers before continuing.

The first command that we are going to use is *man*, this is short for "man"ual. Read about each command to see the range of options that exist. We've already been using *man*, but now we'll practice some more:

Many of the basic commands we'll be practicing are built in as part of your shell environment (that is you won't find a binary file for *cd*). To read about commands like *cp, cd, ls, mv, rm* in more detail you can just type:

```
$ man builtin
```

And, for a command like *ls* you can type:

```
$ man ls
```

And, even for a built-in command you can just type "man commandName", or something like:

```
$ man cd
```

and this will open the "builtin" man page for you.

If you have problems exiting from "man" press the "q" key. Also, you can use the keyboard arrows to move around in the descriptions.

As we move around directories an extremely useful command is *pwd*, which return the working directory name you are in. So, if you get lost just type:

```
$ pwd
```

We'll do this from time to time as we use directory commands.

Now we are ready to practice a bit with the commands:

```
$ cd /
$ pwd
$ ls
$ ls -la
$ cd /tmp
$ cd ..
$ pwd
$ cd tmp
```

What's going on here? If you don't understand, ask.

```
$ cd (take you back to your home directory)
$ pwd
$ touch text.txt
$ cp text.txt new.txt
$ mv text.txt new.txt
```

What's happening now? If prompted to overwrite, respond "y". Note that "userid" is the name of the user account you created in the first exercise.

```
$ cp text.txt /home/username/.
$ cd ../home/username
```

Now play with the use of the tab key. For example, in /home/*username* start to type the first part of the command "cp text.txt text.txt.bak" - then, type:

```
$ cd (to return to our home directory)
$ cp te
$ cp text.txt te
$ cp text.txt text.txt.bak
```

The tab key makes life much easier. Now type:

```
$ cd
$ mkdir tmp
$ mv text.* tmp/.
$ ls
```

Finally, we are going to remove the directory that contains the two archives.

```
$ cd tmp
$ rm *
$ cd ..
$ rmdir tmp
```

You can force this using a command like this:

```
$ rm -rf tmp
```

The use of "rm -rf" is **very dangerous!**, and, naturally, very useful. For example, if you are "root" and you type

"rm -rf /*" this would be the end of your server. This commands says "remove, forcibly and recursively, everything" - Or, if you start in the root directory (/), remove all files and directories *without asking* on the entire server. If you want to use "rm -rf *" always take a deep breath and check where you are first (really, do this!):

```
$ pwd
```

First this says in what directory you are. If you are mistaken, then you have the opportunity to not remove files that you might really need.

18.) Command pipes and grep [Top]

There are a few items that you are going to be using during the week, and which you may have already seen in use, that are important to understand. The first item is the concept of command pipes. The other item is the use of the search facility *grep*. First let's talk about *grep*.

The *grep* command allows you to search for items in files, or in output from another command. For instance, to see all instances of the string "pop" in the file /etc/services (lists services and corresponding tcp/udp ports) you can type the command:

```
$ grep pop /etc/services
```

And you should see each line in the file that contains the string "pop" (including substrings of "pop") displayed. It will look somethig like this:

```
pop3pw          106/tcp    3com-tsmux   #Eudora compatible PW changer
pop2            109/tcp    postoffice   #Post Office Protocol - Version 2
pop2            109/udp    postoffice   #Post Office Protocol - Version 2
pop3            110/tcp    #Post Office Protocol - Version 3
pop3            110/udp    #Post Office Protocol - Version 3
hybrid-pop      473/tcp
hybrid-pop      473/udp
pop3s           995/tcp    spop3        # pop3 protocol over TLS/SSL
pop3s           995/udp    spop3
kpop            1109/tcp   #Unofficial
kpop            1109/udp   #Unofficial
```

If you want to check for all occurrences of the string "ssh" in all files in a directory you could do something like:

```
$ grep ssh /etc/rc.d/*
```

Scroll through the list of output and you'll see that the file /etc/rc.d/sshd (startup script file) is the only file with this string in it. Note that when you do a wildcard search on files, then files that contain the string are indicated in the left-hand column of the output.

Now maybe you are looking for more than one word in a file. Or, perhaps, you figure you need more than one word to make your search unique. For instance, each time you login on your machine you get a welcome message that starts out with the text "Welcome to FreeBSD". Perhaps you are wondering where this message comes from. Well, as much of your initial system configuration is done by files located in /etc/ this might be a reasonable place to look. So, to search for the file with this string you could type:

```
$ grep "Welcome to FreeBSD" /etc/*
```

and you will get back:

```
/etc/motd:Welcome to FreeBSD!
```

and now you know that the string resides in the file /etc/motd. Type:

```
$ man motd
```

to read about this file, known as the Message Of The Day file, or "motd".

Remember when we did "*grep ssh /etc/rc.d/**"? Did you notice that the results filled more than a screen in your

terminal? Well, let's use the concept of the pipe facility to help us out. The pipe facility is the ability to take the output of one command (or stream of information) and "pipe" it to another command. For instance, the output from "*grep ssh /etc/rc.d/**" can be "piped" to the *more* command so that the screen will pause as the results are being shown. The pipe facility is accessed using the symbol "|" (usually above the "\" character). So, if you do:

```
$ grep ssh /etc/rc.d/* | more
```

Now your output will pause and you'll have an easier time seeing what is happening.

Remember when we did the following in the second exercise?:

```
$ pkg_info | grep lynx
```

Now you should understand better what this is doing. But, if you are not sure do this:

```
$ pkg_info | more
```

And, you'll note that there is a long list of packages installed on your machine. The use of the "| more" let's you view them one page at a time. But, if you were just trying to figure out if something by the name "lynx" was installed, then using pipe with grep to seach the output of the "*pkg_info*" command can make your life much easier. So, the command "*pkg_info | grep lynx*" is simply searching the output of the *pkg_info* command for the string "lynx" and only displaying results with "lynx". Thus, when you type this command you should see something like:

```
$ lynx-2.8.5          A non-graphical, text-based World-Wide Web client
```

which is really useful. Now you know the correct name for the lynx package installed is "lynx-2.8.5" and you know what it is.

So, now that you have the correct name you can get the detailed package information quickly by typing:

```
$ pkg_info lynx-2.8.5
```

If you just type "*pkg_info lynx*" this won't work as pkg_info wants either the exact name, or the form "*pkg_info lynx**".

We'll use grep and pipes through the week. For instance you'll see us discussing commands that output lots of data. Getting a handle on this data is often done using pipes and commands like grep, more, etc.

Remember, as usual, if you want to know more type:

```
$ man grep
```

19.) Searching for more information about your system [Top]

If you want to see the contents of a file there are three typical ways to do this:

$ cat
$ less
$ more

Each of these commands has it's own features:

- **cat:** or conCATentate a file. By default to the screen. The cat command will display the contents of almost any file, including files that are "open". In some cases if you use more or less, file contents will not be properly dispaly. In addition, cat displays a file without first clearing your screen - something that can be annoying about less and more if you just want to look in a small file and keep what's on the screen visible.
- **less:** Lets you scroll back and forth while displaying a file. Let's you search (like in vi) while seeing the ocntents of a file. Pauses after each screen of information.
- **more:** is similar to less, but with less functionality. Basically more simply pauses after each screen of

information and you can scroll by pressing the space-bar for each new screenful of information.

the typical saying is to remember that "less is more" when it comes to Unix.

Test this using the three commands using them with an informational file like:

```
$ cd /etc
$ cat motd
$ more services (youcan exit with "q")
$ less services (you can exit with "q")
```

Try looking at some more files, for instance, fstab, rc.conf, termcap, etc. If you don't understand what you are looking at, then use the "man" command. For example, type:

```
$ man fstab
$ man rc.conf
$ man termcap
```

Finally, there is one other useful command for looking inside files. First make sure that you are root:

```
su -
```

Now, as root, type:

```
# tail /var/log/messages
```

This will show you the last few lines of your main logfile on your system. This can be *really* useful if you just want to see what the *last* thing written to a file was. Your file /var/log/messages will get very large over time, so if you used cat, less, or more to view this file for new messages this could become very time-consuming. Now, even more fun is to do the following:

```
# tail -f /var/log/messages
```

Now press ALT-FN (say "F3") to go to one of your virtual terminals. Login as root on this terminal. Now go back to your original terminal where you typed the command `tail -f /var/log/messages`. You should see a message on the screen saying that root just logged in. The "-f" option means, "output appended data as the file grows" - or, you can watch each new item as it's written to the end of a file. This is incredibly useful when you are trying to debug problems and you need to see what happens in your logfiles in real time.

Don't forget to log out of your other terminal window.

If you have any questions ask the instructor or one of the class helpers.

20.) File and directory permissions* [Top]

*Reference: Shah, Steve, "Linux Administration: A Beginner's Guide", 2nd. ed., Osborne press, New York, NY.

If you look at files in a directory using "ls -al" you will see the permissions for each file and directories. Here is an example:

```
drwxrwxr-x    3 hervey   hervey       4096 Feb 25 09:49 directory
-rwxr--r--   12 hervey   hervey       4096 Feb 16 05:02 file
```

The left column is important. You can view it like this:

```
Type User    Group World Links  owner  group  size   date   hour  name
d    rwx     rwx   r-x   3      hervey hervey 4096   Feb 25 09:49 directory
-    rwx     r     r     12     hervey hervey 4096   Feb 16 05:02 file
```

So, the directory has r (read), w (write), x (execute) access for the user and group. For world it has r (read) and x (execute) access. The file has read/write/execute access for the world and read only access for everyone else (group and world).

To change permissions you use the "chmod" command. chmod uses a base eight (octal) system to configure permsissions. Or, you can use an alternate form to specify permissions by column (user/group/world) at a time.

Permissions have values like this:

```
Letter  Permission   Value

R       read         4
W       write        2
X       execute      1
```

Thus you can give permissions to a file using the sum of the values for each permssion you wish to give for each column. Here is an example:

```
Letter  Permission                  Value

---     none                        0
r--     read only                   4
rw-     read and write              6
rwx     read, write, and execute    7
r-x     read and execute            5
--x     Execute                     1
```

This is just one column. Thus, to give all the combinations you have a table like this:

```
Permissions  Numeric      Description
             equivalent

-rw-------   600          Owner has read & execute permission.
-rw-r--r--   644          Owner has read & execute.
                          Group and world has read permission.
-rw-rw-rw-   666          Everyone (owner, group, world) has read & write
                          permission (dangerous?)
-rwx------   700          Onwer has read, write, & execute permission.
-rwxr-xr-x   755          Owner has read, write, & execute permission.
                          Rest of the world has read & execute permission
                          (typical for web pages or 644).
-rwxrwxrwx   777          Everyone has full access (read, write, execute).
-rwx--x--x   711          Owner has read, write, execute permission.
                          Group and world have execute permission.
drwx------   700          Owner only has access to this directory.
                          Directories require execute permission to access.
drwxr-xr-x   755          Owner has full access to directory. Everyone else
                          can see the directory.
drwx--x--x   711          Everyone can list files in the directory, but group
                          and world need to know a filename to do this.
```

Now lets practice changing permissions to see how this really works. As a normal user (i.e. don’t login as root) do the following:

```
$ cd (what does the "cd" command do when you do this?)
$ echo "test file" > read.txt
$ chmod 444 read.txt
```

In spite of the fact that the file does not have write permission for the owner, the owner can still change the file’s permissions so that they can make it possible to write to it:

```
$ chmod 744 read.txt
```

Or, you can do this by using this form of chmod:

```
$ chmod u+w read.txt
```

The forms of chmod, to add permissions, if you don’t use octal numbers are:

$ chmod u+r, chmod u+w, chmod u+x
$ chmod g+r, chmod g+w, chmod g+x
$ chmod a+r, chmod a+w, chmod a+x

Note that "a+r" is for world access. The "a" is for "all", "u" is for "user", and "g" is for "group".

Now, change the file so that the owner cannot read it, but they can write to the file...

```
$ chmod u-r read.txt
```

Or, you can do something like:

```
$ chmod 344 read.txt
```

You probably noticed that you can use the "-" (minus) sign to remove permissions from a file.

Finally, there is a concept that when you execute a file you normally execute it using the permissions of the user who does this. For example, if the user "carla" types "netstat", the netstat programs runs with their privileges. But, if you want netstat to always run with the permissions of the owner or of the group of the netstat program, then you can configure the "SetUID" or "SetGID" bit. You can do this using chmod. However, remember that this can be a bad idea from the viewpoint of security...

To do this add a "4" to the chmod octal to set the SetUID, or a "2" to set the SetGID bit.

As an example, you could do (you must be root):

```
# chmod 4755 /usr/bin/netstat
```

Naturally you would need to be root to do this, or you would have to use the "sudo" command.

And, to set the SetGID bit it would be:

```
# chmod 2755 /usr/bin/netstat
```

After you do "sudo chmod 4755 /usr/bin/netstat" the permissions on the file would look like this:

```
-rwsr-xr-x    1 root     kmem       106344 Feb 23  16:42 /usr/bin/netstat
```

Note the "s" in the owner column.

And, after you issue the command ""sudo chmod 2755 /usr/bin/nestat" the permissions look like this:

```
-rwxr-sr-x    1 root     kmem       106344 Feb 23  16:42 /usr/bin/netstat
```

Note: This is for example only. Setting netstat to run with SetUID or SetGID bit set as shown would be a very bad idea. So, if you have done SetUID or SetGID, then please unset these bits for netstat like this:

```
# chmod 0755 /usr/bin/netstat
```

and now you would see:

```
-rwxr-xr-x    1 root     kmem       106344 Feb 23  16:42 /usr/bin/netstat
```

21.) Commands - programs - shell - path [Top]

For this exercise we want you to run as a user other than root. So, if you are root do this:

```
# su - user
```

What do you think the "-" does? (hint: "man su", and we talked about this earlier)

When you type a command or the name of a program the system looks for something with that name using in the directories specified in your PATH environmental variable. Or, if the command is a built-in shell program (such as "cd", see "man builtin"), then it will execute the command without needing to use the PATH variable. To see your what your PATH is set to, do this:

```
$ printenv PATH
```

The PATH variable is configured when you login to your account in the file ".profile" in your home directory.

To see how this works let's create a shell script that will run a simple command, but which resides in a directory outside your PATH statement ("user" here is the name of your own userid).

```
$ cd /home/user
$ mkdir scripts
$ cd scripts
$ vi hello.sh
```

Now in the file add these lines:

```
#!/bin/sh
#

echo hello
```

Remember to save and exit the file (:wq). And, to ensure that you can execute the file use the command:

```
$ chmod u+x hello.sh
```

Remember that this is using chmod to set the eXecutable bit for the user only.

Now we are going to add the /home/user/scripts directory to our login profile PATH statement.

```
$ cd /home/user
$ vi .profile
```

Look for the line that reads (more or less - could be different on your machine):

PATH=/sbin:/bin:/usr/sbin:....:#HOME/bin; export PATH

and change it so that at the end of the PATH statement you add:

PATH=/sbin:/bin:/usr/sbin:....#HOME/bin:#HOME/scripts; export PATH

Save the file and now do the following:

```
$ hello.sh
$ . .profile
$ hello.sh
```

What just happened? You changed the PATH statement to include /home/user/scripts, but when you tried to run the script in /home/user/scripts it didn't work. This was because you had not actually updated the PATH variable in your shell. When you did ". .profile" you executed your user profile again, which updated the PATH variable with your new PATH variable. You can verify this by typing "printenv PATH" again.

You may have noticed the "#HOME/bin" item in the PATH. As you can see FreeBSD has the concept that you may wish to have your own personal bin directory for executables, so normally the "hello.sh" script might reside in /home/user/bin, but for purposes of this exercise we used /home/user/scripts.

Finally, if you want to change something like the PATH for everyone you can do this in two ways. One, you could update /etc/profile with a new PATH statement. This means that everyone on your system will see this change as soon as they login the next time. Or, you can change /usr/share/skel/dot.profile so that all new accounts have the new PATH, but previous users will not see this change. In both cases changes like this should not be done lightly. When setting up a server with many users you will probably want to think about what directories your users need to have in the PATH from the beginning and update /usr/share/skel/dot.profile before creating initial user accounts.

In addition, you can run the "hello.sh" script by typing "/home/user/scripts/hello.sh" at any time.

To finish up we are going to change how the "rm" command runs to make it "safer" (in my opinion). Here are the steps:

```
$ vi /home/user/.profile
```

Go to the end of the file and type an "o" (add a line after the cursor and put you in to input mode). Then type:

```
alias rm='rm -i';
```

Now exit and save the file (:wq). After that type:

```
$ touch temp.txt
$ rm temp.txt
$ . .profile
$ touch temp.txt
$ rm temp.txt
```

And, what happened? Now the "rm" command asks you before you erase a file if you are sure you want to do this. If you don't like this you can remove the alias in the .profile, re-run .profile, and leave things as they were. Note, you can always just use "rm -f" to force remove files and skip the prompt. My advice is to leave the "rm" command in interactive mode - you are likely to be very thankful for this at some point in the future.

To be Done Last by Everyone

22.) Update your /user/ports collection using cvsup [Top]

As our final exercise we are going to ask you to update your entire ports collection on your machine. We are going to use the cvsup package (not yet installed) to tell your machine to compare all the files in /usr/ports with the files on a cvs server and then download any updates that may have been made. The ports collection is large and if you attempted to do this over our classroom link it would take many hours, if not days, to complete. To avoid this we have installed a local mirror of an up-to-date FreeBSD ports collection and cvs server on our classroom server. We are going to use this machine to update our ports collection.

You need to be root to complete this exercise.

So, to go through this process you first need to install the cvsup-without-gui package. We have downloaded and copied this to our local server. To install cvsup-without-gui do the following:

```
pkg_add ftp://noc/pub/FreeBSD/packages/All/cvsup-without-gui-16.1h_2.tbz
```

Next we need to configure your machine so that it will automatically look in the correct location for the cvs server in our classroom to update your ports collection.

If you completed exercise 4, then this step should already be completed.

To do this first start by editing the file /etc/make.conf:

```
# vi /etc/make.conf
```

And, scroll down to the bottom of the file. Now you should add the following two lines (remember "o" for input mode in vi with a new line):

```
MASTER_SITE_BACKUP=ftp://noc.pacnog.school.fj/pub/FreeBSD/ports/distfiles/${DIST_SUBDIR}/
MASTER_SITE_FREEBSD=yes
```

Remember to replace "noc" with what was indicated at the start of these exercises. These last steps are not technically necessary at this point as we are going to explicitly tell our cvsup process where to look for updated ports file, but we make sure that the variables MASTER_SITE_BACKUP and MASTER_SITE_FREEBSD are set so that when you install future ports using the *make* command the process will look on our local server for files to download, and not go out to the freebsd.org site across our classroom link to the outside world.

If you are interested in how items are configured for doing things like updating your ports files, maybe your FreeBSD source tree, etc. then have a look at the example files in /usr/share/examples/cvsup.

When we run cvsup, in this case, it is going to download updates to descriptive files only. Actually source is not downloaded until you issue a *make*. This has been determined by the configuration in the example cvsup config

file we'll use in our command below, which is /usr/share/examples/cvsup/ports-supfile.

Now that things are setup correctly, let's start the ports package upgrade process using `cvsup`. Once you issue this command you can leave your machine running as this may take some time to complete. The command to issue is:

```
cvsup -g -L 2 -h noc /usr/share/examples/cvsup/ports-supfile
```

Specifics on the command (taken directly from "`man cvsup`"):

- "-g": Disables the use of the graphical user interface. This option is implied if the DISPLAY environment variable is not set.
- "-L" verbosity: Sets the verbosity level for non-GUI output. A level of 0 causes cvsup to be completely silent unless errors occur. A level of 1 (the default) causes each updated file to be listed. A level of 2 provides more detailed information about the updates performed on each file. All messages are directed to the standard output. This option is ignored when the GUI is used.

Hervey Allen

Last modified: Tue Jun 14 04:25:07 CLT 2005