Additional Exercises: Post FreeBSD Install: ccTLD Workshop Nairobi

September 12, 2005

Exercises

For Those who Want More Practice

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.) 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.

2.) 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
```

The options to the `ps` command mean:

a: Display information about all user processes, including your own.
u: Display the following process information: user, pid, cpu, mem, vsz, rss, tt, state, start, time, and command used.
x: Modifies the "a" option to include information for detached processes as well.
w: Use 132 columns to display information vs. just your window size. Critical to include actual command that was used to start the process, particularly when using `ps` with the `grep` command. Use "w" twice to display entire width regardless of size.

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.

3.) 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.

4.) 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.

5.) 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.

6.) 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.

Additionally, it's possible to configure your prompt to indicate what directory you are in at all times. This is dependent on what shell you are using.

7.) 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
```

8.) 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.

9.) 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 equivalent	Description
-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
```

A UNIX Permissions "Gotcha"

If a directory has the World or Group write flag set, and contains a file that is only writeable by the owner, then a member of either the Group or World (everyone) can still make changes to the file. Here's an example of how (become root to do this):

```
# mkdir /tmp/test
# echo "example text" > /tmp/test/example.txt
# chmod 644 /tmp/test/example.txt
# chmod a+w /tmp/test
# su - userid
$ cp /tmp/test/example.txt .
$ echo "add more text to file" >> example.txt
$ mv example.txt /tmp/test/example.txt
```

You will receive the following prompt:

```
override rw-r--r--  root/wheel for /tmp/test/example.txt? (y/n [n])
```

If you press "y" and ENTER, then your version of the file will now overwrite the *read only* version of the file owned by root in the /tmp/test directory. This is because write permission has been enabled for World on the /tmp/test directory. Thus, you have permission to mv (i.e. rename) a file of the same name to this directory. This result may seem surprising. If you do:

```
$ ls -al /tmp/test
```

You will see that your *userid* now owns the file. The root user no longer owns the file. So, using this trick is pretty obvious, unless, of course, you set things back to the way they were using the `chown` command.

10.) 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 a couple of items. First, we'll 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.

Finally, let's change your prompt to show what directory you are in as you move around your system. Assuming you are using the Bash shell for your userid, then we'll make this change in the file ".bashrc" in your home directory. You can make these changes in the .profile file, but Bash will not always pay attention to .profile, particularly if you are starting terminal sessions from within a GUI environment like Gnome or KDE. A nice discussion on how to set the Bash prompt can be found here:

http://www.64-bit.de/dokumentationen/howto/en/html/Bash-Prompt-HOWTO-2.html

Let's set our Bash prompt to display our username, hostname, and the directory we are in. To do this do:

```
cd
$ vi .bashrc
```

It's likely that this file did not exist on your system as Bash is not supported by FreeBSD by default due to licensing issues. Thus, an initial .bashrc file is not included in the /usr/share/skel directory. By typing `vi .bashrc` the file will be created when we save our changes. So, at the end of the file (if it already exists) add a line, otherwise, just press "i" to insert, and add the following:

```
PS1="[\u@\h:\w]\$ "
```

Note the space after the '$' - this is important. Now, exit and save by typing:

```
:wq
```

and now run:

```
$ . .bashrc
```

If you are just in your home directory you'll see:

```
userid@hostname:~]$
```

as the tilde ('~') represents your home directory. Try moving to another directory to see the difference:

```
$ cd /usr/local
```

And you should see:

```
userid@hostname:/usr/local]$
```

There are many ways you can set your prompt, including making use of colors for terminal programs that support this. Read the Bash Prompt HOWTO page listed above for more details.

[Top]

Hervey Allen

Last modified: Thu Sep 8 19:36:24 ART 2005