Text Processing and Manipulation

Other text manipulation tools at your disposal

In your day-to-day work with a Linux server, you may find yourself needing to view, manipulate and change text data without "editing" files, per se. In this section we will look at various tools that come with standard Linux distributions for dealing with text data.

Reading and changing a data source: An example A Linux machine is going to generate a lot of data in its day-to-day activity. Most of this is going to be very important for your management of the machine. Let's take an example. If you have the Apache web server installed, it constantly pumps data to two important files, access and errors. The access file logs every visit to your web site. Every time somebody (or something) visits your website, Apache creates an entry in this file with the date, hour and information the file that was requested. The errors file is there to record errors such as server misconfiguration or faulty CGI scripts.

You can find good software out there for processing your access file. It will tell you how many hits you got on your webpage. It will even automatically look for requests for missing or non-existent web pages, known as 404s. Some of this software is very good at generating informative reports of traffic, warnings of missing pages and other website statistics, but it is aimed at website administrators in general and as such presents its data in a general way. At times though, you may want or need to do a bit of hands on evaluation of the data in the file - to go beyond what these web stats packages offer.

I can grep that!

Any information that you might want to get from a file is at your fingertips, thanks to grep. Despite rumors to the contrary that grep is Vulcan forfind this word, grep stands for General Regular Expression Parser. And now that you know that, I'm sure you feel a lot better. Actually the more proficient you are at dealing with regular expressions, the better you will be at systems administration. Let's look at another example from our Apache access file.

Let's say you're interested in looking at the number of visits to your website for the month of June. You would do a simple grep on the access file, like so:

```
grep -c 'Jun/2003' access
```

This will give you the number of files requested during the month of June. If you were interested in this month's requests, we could add this little twist:

```
grep -c `date +%b` access
```

What we've done basically is put a command in the way, so to speak. grep will take the output for that command, which is the name of the month (%b option in date will give you the abbreviation of the name of the month) and look for it in the file.

Commands that are actually options of other commands must be placed inside the backward or grave accent marks, or else they won't work.

You can actually do some pretty cool stuff with your Apache access file using grep plus the date command. You might want to look for just today's hits:

```
grep -c `date +%d/%b` access
```

Maybe you'd like to see what the hits were like f or the same day but other months:

```
grep -c `date +%d/` access
```

Perhaps you were expecting more information? Some Linux distributions will install special scripts that will run a cron job to compress the Apache access files. Even if your old file are compressed, say, each month, you can still grep them for information without having to de-compress them. Grep has a version that can look it files compressed with gzip. It's called, appropriately zgrep

```
zgrep -c `date +%d/` access_062003.gz
```

This will look for the hits on the webserver for this same day, but in June in your gzip'd access file. (unless of course, it's the 31st).
Speaking of gzip, you can actually do the reverse. You can grep files and, based on the results, create a gzip'd file. For example, if you wanted to create a file of the hits this month to date, you could do this:

```
grep `date +%b/` access | gzip -c > access_01-20jul.gz
```

There is really no end to what grep will furnish for you, as long as you some idea of what you're looking for. The truth is that we've been concentrating on the Apache files. There are dozens of files that might interest you at any given moment and all can be "grep'd' to give you important information. For example, you might be interested in how many messages you have in your in-box. Again, grep can provide you with that information.

```
grep -c '^From:' /var/spool/mail/penguin
```

If you're interested in who the mail is from, then just leave off the -c option. You should see something like this:

```
From: fred@tidlywinks.con
From: webmaster@perly.ork
From: "F. Scott Free" < fsfree@freebird.ork>From: "Sarah Doktorindahaus" < sarah@hotmael.con>
```

The caret (^) indicates that grep should look for every line that starts with the expression you want. In this case, the 'From:' header in standard email messages.

Speaking of mail, let's say that somebody sent you an email yesterday with the name of a person that you needed to contact and his/her phone number. You wouldn't even have to open your email client to get this information. You could just look for numbers formatted like phone numbers in your inbox.

```
grep '[0-9]{3}-[0-9]{4}' inbox
```

Phone numbers are usually formatted as the exchange (555 - 3 numbers) and the number itself (6677 - 4 numbers), so grep will look for any numbers grouped in 3 and 4 respectively with a slash (-) between them.

We used the caret (^) to look for lines that began with certain characters. We can also look for lines that ends with specific characters with the dollar ($) sign. For example, if we wanted to know which users on our system use the bash shell, we would do this:

```
grep bash$ /etc/passwd
```

In this case, the shell used is the last thing listed on each line of the /etc/passwd file, so you'd see something like this as output:

```
root:x:0:0:root:/root:/bin/bash
mike:x:500:500:mike:/home/mike:/bin/bash
dave:x:501:501:dave:/home/dave:/bin/bash
laura:x:502:502:laura:/home/laura:/bin/bash
jeff:x:503:503:jeff:/home/jeff:/bin/bash
```

Using grep with other commands

One of the more common uses of grep in administration tasks is to pipe other commands to it. For example, you're curious as to how much of your system's resources you're hogging .. eh hem.. using:

```
ps uax | grep $USER
```

Or how about looking for the files you created during the month of October:

```
ps -l | grep Oct
```

Again, we could go on for a long, long time, but we have other commands to look at. Needless to say, you'll find grep to be one of your most valuable tools.

Feeling a little awk(ward) awk is another one of those tools that will make your hunt for meaningful data a lot easier. awk is actually a programming language designed particularly for text manipulation, but it is widely used as an on-the-spot tool for administration.

For example, let's the return to the example I used earlier with grep. You want to see the processes that you're using. You can to this with awk as well.

```
ps uax | awk '/mike/'
```

does exactly the same as grep. However, the tool ps presents its data in a tabled format so awk is better suited to getting just the parts we want out of it more than grep is. The uax options we used above will present the information like so:

```
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root      3355  0.0  0.1  1344  120 ?        S    Jul27   0:00 crond
```

So using awk, we can get specific information out of each column. For example, we could see what processes root is using and how much memory they're using. That would mean telling awk to get us the information about root's processes from columns 1,2 and 4:

```
ps uax | awk '/root/ {print $1,$2,$4}'
```

This would get us some information like this:

```
root 1 0.3
root 2 0.0
root 3 0.0
root 4 0.0
root 9 0.0
root 5 0.0
root 6 0.0
root 7 0.0
root 8 0.0
root 10 0.0
root 11 0.0
root 3466 0.0
root 3467 0.0
root 3468 0.0
root 3469 0.0
root 3512 0.0
root 3513 7.9
root 14066 0.0
```

You'll notice that PID 3513 shows more memory usage that the others, so you could further use awk to see what's going on there. We'll add the column 11 which will show us the program that's using that memory.

```
ps uax | awk '/3513/ {print $1,$2,$4,$11}
```

And you'll see that X-window is the program in question:

```
root 3513 7.6 /usr/X11R6/bin/X
```

We could even take this one step further. We could use grep to get a total of the memory that our processes are using:

```
ps uax | awk '/^mike/ { x += $4 } END { print "total memory: " x  }'
```

awk is nice enough to accommodate with the total:

```
total memory: 46.8
```

I am using almost half of the machine's memory. Nice to see that awk can do math too!

As it can do math, you can also use it to check out the total bytes of certain files. For example, the total bytes taken up by jpgs in your photo collection:

```
ls -l | awk '/jpg/ { x += $5 } END { print "total bytes: " x }'
```

Column 5 of ls -l shows the bytes of the file, and we just have awk add it all up.

Tengo 'sed'!

sed, though an excellent tool, has been largely replaced by features in Perl which duplicate it somewhat. I say somewhat, because the use of sed in conjunction with pipes is somewhat more comfortable (at least to this author) for substitution of text in files.

Sed is the Spanish word for thirst, Though you may thirst (or hunger, or whatever) for an easier way to do things, its name did not derive from there. It stands for Stream editor. Sed, as you saw in our example, is basically used for altering a text file without actually having to open it up in a traditional editor. This results in an incredible time savings for a system administrator.

One of the most common uses of sed is to alter or eliminate text in a file. Let's say you want to see who last logged into your network. You would run a utility called lastlog. This will show you the last time that every user logged in. The fact is though that we're only interested in real human users, not users that are really daemons or processes or literally "nobody". If you just ran lastlog you'd get a bunch of entries like this:

```
mail                              **Never logged in**
news                              **Never logged in**
uucp                              **Never logged in**
```

You can, of course, make this output more meaningful and succinct by bringing sed into the picture. Just pipe the output of lastlog and have sed eliminate all of the entries that contain the word Never.

```
lastlog | sed '/Never/d' > last_logins
```

In this example, sed will delete (hence the 'd' option) every line that contains this word and send the output to a file called last_logins. This will only contain the real-life users, as you can see:

```
Username        Port     From            Latest
fred           pts/3   s57a.acme.com   Mon Jul 14 08:45:49 +0200 2011
sarah           pts/2   s52b.acme.com   Mon Jul 14 08:01:27 +0200 2011
harry           pts/6   s54d.acme.com   Mon Jul 14 07:56:20 +0200 2011
carol          pts/4   s53e.acme.com   Mon Jul 14 07:57:05 +0200 2011
carlos          pts/5   s54a.acme.com   Mon Jul 14 08:07:41 +0200 2011
```

So you've made a nice little report here of people's last login times for your boss. Well, nice until you realize that you've shown that you (fred) logged in at 08:45, three quarters of an hour after you were supposed to be at work. Not to worry. Sed can fix your late sleeping habit too. All we have to do is substitute 08 for 07 and it looks like you're a real go-getter.

```
cat last_logins | sed 's/08/07/g' > last_logins2
```

What you've done is use the 's' option (for substitute) to change every instance of 08 to 07. So now it looks like you came in at 7:45. What a go-getter you are now! The only problem is that you have made those changes throughout the whole file. That would change the login times for sarah and carlos as well to 7:01 and 7:07 respectively. They look like real early birds! That might raise suspicion, so you need a way to change only the 08 on the line that shows your (fred) login. Again, we sed will come to our rescue.

```
sed '/fred/s/08/07/g'
```

And sed will make sure that only the line containing fred gets changed to 07.

Sed, of course, is more than just a text substitution tool (and a way to hide your tendency to over-sleep on Monday). Let's go back to a previous example of looking for MS Windows-based worms hitting our web server. Sed can actually do a better job of 'grepping' than grep can do. Let's say you wanted to look at all the CodeRed hits for just the month of July, you would do this:

```
cat access | sed '/default.ida/!d; //Jul/!d' > MS_Exploits_July
```

The 'd' option, which we used before to delete text, can be used, with the exclamation point in front of it, to find text.
On the contrary, you may want to get a print out of just normal web traffic, sans all those Windows exploits. That's where just plain 'd' comes in handy again.

```
cat access | sed '/^.{200}/d' > normal_traffic
```

This will eliminate any line of less than 200 characters, which means it will show most requests for normal files, and not CodeRed or Nimda hits which are longer. Now, using reverse logic, we now have a different method CodeRed/Nimda hits by analyzing hits that are more than 200 characters in length. For this, we'll make a few modifications:

```
cat access | sed -n '/^.{220}/p' > MS_exploits
```

The difference with this example is the -n option. This tells sed to limit its action to the other option designated 'p', which means pattern. In other words, limit what you show to those lines matching a pattern of 220 characters in length or more.

There are a lot more ways you can use sed to make changes to files. One of the best ways to find out more about sed is to see how others use it and to learn from their examples. If you go to Google and type: +sed +one liners, you can get all kinds of examples of how sed can be used. It's an excellent way to polish your skills.

Using Uniq Uniq is a good tool for weeding out useless information in files. It's the Unix/Linux way of separating the wheat from the chaff, so to speak. Let's say that you were involved in some sort of experiment where you had to observe sometime of behavior and report on it. Let's say you were observing somebody's sleep habits. You had to watch somebody sleeping and report on it every 10 minutes, or whenever there was a change. You might sit in front of your terminal and issue this command, for example:

```
echo `date +%y-%m-%d_AT_%T`   No changes >> sleep_experiment_43B
```

And when you see some changes, you could issue this command:

```
echo `date +%y-%m-%d_AT_%T`   subject moved right arm >> sleep_experiment_43B
```

You'd end up with a file that looks like this:

```
03-08-09_AT_23:10:16 No change
03-08-09_AT_23:20:24 No change
03-08-09_AT_23:30:29 No change
03-08-09_AT_23:40:31 No change
03-08-09_AT_23:50:33 No change
03-08-09_AT_00:00:34 No change
03-08-09_AT_00:10:35 No change
03-08-09_AT_00:20:37 No change
03-08-09_AT_00:30:05 subject rolled over
03-08-09_AT_00:40:12 No change
03-08-09_AT_00:50:13 No change
03-08-09_AT_01:00:50 subject moved left leg
03-08-09_AT_01:10:17 No change
03-08-09_AT_01:20:18 No change
03-08-09_AT_01:30:19 No change
03-08-09_AT_01:40:20 No change
03-08-09_AT_01:50:47 subject moved right arm
03-08-09_AT_02:00:11 No change
03-08-09_AT_02:10:20 subject scratched nose
```

If this file went on until 07:00, when the subject finally wakes up, you might have a lot of entries in there with 'no change', which is fairly uninteresting.

What if you wanted to see just the changes in sleep behavior? Uniq is your tool. Uniq will show you just the 'unique' lines or lines with different information. You may be thinking: But all the lines are unique, really. That's correct. All the times are different, but we can adjust for that on the command line.

```
uniq -f 1 sleep_experiment_43B
```

This tells uniq to skip the first field, which is the data field and just list the unique fields. You'll end up with something like this:

```
03-08-09_AT_23:10:16    No change
03-08-09_AT_00:30:05    subject rolled over
03-08-09_AT_01:00:50    subject moved left leg
03-08-09_AT_01:50:47    subject moved right arm
03-08-09_AT_02:10:20    subject scratched nose
```

Now you're probably saying: I want to run Linux machines, not conduct sleep experiments. How do I use 'uniq' along these lines?. Well, let's go back to our example of looking for users last login times. If you remember, lastlog got that information for us, but it also listed users that weren't real people. We narrowed it down to real logged in users by invoking sed. The only drawback to that is that we got only users that had logged in. If you want to see all real users whether they have logged in or not you could do this:

```
lastlog | uniq -f 4
```

The fourth field of the output of lastlog is the one that indicates **Never logged in**, so This will find all real users whether or not they have logged in or not. This is good for finding out which users have actually used our system. Some people are given shell accounts and then they never use them. You can then weed these people out.

Sort this out
Another text manipulation tool that will come in handy is sort. This tool takes a text file or the output of another command and "sorts" it out (puts it in some order) according to the options to choose. Using sort without any options will just put the lines of a file in order. Let's imagine that you've got a grocery list that looks something like this:

```
chocolate
ketchup
detergent
cola
chicken
mustard
bleach
ham
rice
bread
croissants
ice-cream
hamburgers
cookies
spaghetti
```

In order to put this in alphabetical order, you'd just type:

```
sort grocery_list
```

That would give you a nice list starting with bleach and ending with spaghetti. But let's say you're smarter than you're average shopper and you have also written down where the items can be found, saving you time. Let's say you're list looks like this:

```
chocolate      aisle 3
ketchup        aisle 9
detergent      aisle 6
cola           aisle 5
chicken        meat dept
mustard        aisle 9
bleach         aisle 6
ham            deli counter
rice           aisle 4
bread          aisle 1
croissants     aisle 1
ice-cream      aisle 2
hamburgers     meat dept
cookies        aisle 3
spaghetti      aisle 4
```

To make your trip to the supermarket faster, you could sort the second column of the list like so:

```
sort +2 grocery_list
```

The +2 (+ [column]) means to sort it according to the second column. Now you'll get everything nicely sorted by section:

```
bread        aisle 1
croissants   aisle 1
ice-cream    aisle 2
chocolate    aisle 3
cookies      aisle 3
rice         aisle 4
spaghetti    aisle 4
cola         aisle 5
bleach       aisle 6
detergent    aisle 6
ketchup      aisle 9
mustard      aisle 9
ham          deli counter
chicken      meat dept
hamburgers   meat dept
```

Again, you're probably saying: Will being a more efficient shopper help me with my system administration tasks? The answer is: Yes, of course! But let's look at another example that has to so with our system.

tail is another one of the more useful commands on a system. tail will show you the last 10 lines of a file. But if you use 'cat' with a pipe to sort and 'more', you get a tail that's sort of interactive (I couldn't resist).

```
cat /var/log/mail.log | sort -r |more
```

The -r option stand for reverse, so here will see the entries in your mail server logs beginning with the most recent. As you push enter, you'll begin to go to the older entries.

Cut to the chase

Sometimes the output of a program gives you too much information. You'll have to cut some of it out. That's where the program cut comes in handy. Earlier we saw how to get only the IP addresses of the visitors to the website. Here's a refresher, with a twist.

```
cat access | cut -c1-16 > IP_visitors
```

This would get us a file with all of the IP addresses. We'll get back to this in a minute.

There are, of course, other practical uses. The other day a client asked me to let a user on the system use Hylafax, a fax server for Linux. One of the requirements of letting a user access Hylafax is knowing his or her user ID or UID. This information is located in /etc/passwd and there is a quick way to get at it:

```
cat /etc/passwd | grep bob | cut -f1,3 -d":"
```

Basically what we've done is to grep the user bob out of the passwd file and pass it to cut and take the first and third (1,3) fields which are separated (delimited -d) by a colon (:). The result is this:

```
bob:1010
```

User 'bob' has a UID of 1010.

Let's look at our Apache weblog example again. We can combine a few of the text manipulation tools we've learned to see how many unique IP addresses have visited our website.

```
cat access | cut -f1-2 -d" " | sort | uniq | wc -l
```

This is a great example of the Unix way of doing things. We've cut out the first field (-f1-2 - shows us only fields 1 to 2 delimited by whitespace -d""). We pipe it to sort, which puts them in numerical order. The uniq tool then shows only the "unique" IP address. Finally, 'wc -l' counts the lines. You really have to ask yourself: What would it cost me to get this info with a normal word processor or text editor?

Using the vi Editor

VI is probably the most popular text editor for Linux. Even if you don't like it, you may end up using it quite often. If you need to make a quick change to a file, you can't beat 'vi'. This is not meant to be an exhaustive guide to vi. This is just meant to show you how to use the most common (and useful) commands.

Sometimes you need vi

I had an unpleasant surprise once. A friend of mine who had installed Linux had somehow changed the default editor from vi to joe. He called to tell me that his crontab entries didn't do anything. One more reason to get to know vi. Crontab is designed for vi and may not work if you use certain alternative editors.

vi basics

Open file with vi

```
vi /etc/hosts.allow
```

Of course, just opening the file gets you nowhere, unless you only want to read it. That can be just as easily done with less, so we'd better look at what to do with 'vi' beyond just opening the file.

Adding text

To add text, you would type the following:

```
ESC + i
```

(i for insert)

And there you have it. Now you can type away. Of course, it doesn't really do us too much good if you're not able to save what you've typed. So let's save it, shall we?

Saving the file

```
ESC + w
```

(w for write)

Closing the file

Now, to finish you would type the following:

```
ESC + q
```

(q for quit)

Of course, there is a lot more that you may need to do. Let's have a look.

Vi for budding power users

Again, my aim is not to do into a treatise on vi, but here are a few more commands that you might need to do a little more heavy lifting.

Removing Lines

You may find that you need to remove an entire line from a file. Just place the cursor at the beginning of the line and type:

```
ESC + d
```

(d for delete)

Changing your mind

Sometimes you wish you hadn't done something. With vi, you can undo what you just did.

```
ESC + u
```

(u for undo)

Changing your mind (again)
Again, you have second thoughts. With vi, there are no regrets.

```
ESC + q!
```

(q! for quit and I *really* mean it!)

The exclamation point (!) is used in vi to override default settings. If you make changes to a file, vi is going to ask you if you want to save them. 'q!' means that you want to quit without saving.

Where did I put that word?

Did you misplace a word in your file? You can find it easily with vi

```
ESC + /word
```

If you're looking for the word nonplussed in your text (as in: 'vi is so easy I am nonplussed') you would type:

```
ESC /nonplussed
```

and it will find every instance of the word nonplussed.

Can I change that word?

Maybe you don't want to use the word nonplussed. Perhaps it's better to use a more well-known word. You can use vi to change the word

First you could use the
`/nonplussed`

to look for that word. When you find it, you would then type

```
ESC : s/nonplussed/amazed/
```

to replace the word on that line.

If you were sure that you wanted to replace all instances of that word in the whole text, you could type this

```
ESC :%s/nonplussed/amazed/g
```

and nonplussed would be changed to amazed throughout the text.

If you want to get some control over what you replace - that is you want to used both nonplussed and amazed, then you would add gc to the end:

```
ESC :%s/nonplussed/amazed/g
```

Vi will now ask you for confirmation.

Vi configuration settings

There are some basic vi configuration setting that you should be aware of if you want to use this text editor comfortably.

Word Wrapping If you don't want your words running off the page into oblivion, you can set the word wrapping feature in vi

```
ESC : set wm=30
```

This is a good wrap for me personally. It sets the wrap to 30 spaces from the right so it makes it tight. This means that the bigger the number, the sooner you get a line break. If you send something via email, then this tight wrap ensures that it will get there without the lines being broken into stair-steps.

Vi as an email editor

What did you say? Vi and email? Yes! You can use vi as an email editor. This is most commonly done with the email client mutt.

More Vi

This just scratches the surface of what you can do with vi. Here I've tried to show you what might be the most useful features and setting of vi. It should be enough to get some basic work done with this ubiquitous editor for Linux.

How to automate tasks on a Linux system

You don't have to do that yourself!One of the main tasks of a system administrator is carrying out maintenance work on the server. Most of these tasks can be automated or programmed to be carried out at certain times without user intervention. In this section we'll talk about the two most widely used programs to carry out tasks in this way.

Use of 'at'

'at' is a program to carry out commands that you intend to do only once. It's mostly used for scheduling specific jobs under specific circumstances. If you had to rotate your company's webserver logs every Saturday, 'at' is not the appropriate tool for the job. That would be done best with 'cron', about which we will talk about shortly. Let say your boss, the CTO, called for a meeting with you at 1:00. He wants to know how frequently your external consultants are logging into the network. This is a prime candidate for 'at'.

First, you'd type:

```
at 12:45
```

which would give you plenty of time to get that information before the meeting. You will see the 'at' prompt:

```
warning: commands will be executed using /bin/sh
at >
```

Now you'd write the commands you want carried out. Here we'll get the output of the command last, which tells us who's logged in to our servers lately, and write it to a file called 'log-ins'. The second command, separated by a semi-colon (;) will then print that file using lp.

```
last > $HOME/log-ins; lp $HOME/log-ins
```

press 'Enter' and then 'Ctl + d' and you will see the following:

```
job 15 at 2003-02-16 12:45
```

Of course, your job number will vary with the number of times you've used 'at'.

There are various ways to indicate at what time you want 'at' to carry out commands. at now + 5 minutes will carry out a command five minutes from when you type it. There's even a provision for at teatime which will carry out commands at 4:00 PM/16:00 hrs. (If you don't believe me, consult 'man at'!). You can cancel these jobs as well. If you type:

```
atrm 15
```

you will remove job 15 from the 'at' queue. To see what is in the 'at' queue, type:

```
atq
```

You can control which users are allowed to use 'at'. By default

```
/etc/at.deny
```

controls who cannot use 'at'. That is to say, the users listed in at.deny cannot use it. You can also create an

```
/etc/at.allow
```

file. Creating at.allow makes the at daemon ignore the

```
/etc/at.deny
```

Therefore, anyone who is not in at.allow cannot use 'at'. The question of using one file or another comes down to a question of your management style. If you prefer to let people use things until the abuse the privilege, then use the default at.deny. When the user 'barney' programs an 'at' job to set off an infernal sounding noise when he's gone to get coffee, scaring the bejeebers out of everybody in the office, then you can add him to the at.deny file. If you're of the opinion that nobody needs to use it, then create an at.allow file with only your personal user account listed. Remember that the root user can always use at.

Use of 'cron'

From a system administrator's point of view, the cron daemon is probably the best thing since sliced bread. You can schedule practically any program (provided that they don't have a graphic user interface since cron is not really designed to run GUI applications) at any time, for any date and at any interval. That is to say, if you want a text dump of the number of times a person with the IP address 64.09.200.12 has logged into your computer and you only want it on February 4th, cron will do this for you.

The jobs that you want to run with cron can be scheduled in various ways. The most common way is to edit a file which is known as your crontab. Normally, each user has his/her own and is able to schedule jobs by editing it. You can add to and delete entries from you crontab by typing:

```
crontab -e
```

But before we go jumping right into scheduling jobs, it's important to point out that cron looks for a particular syntax in your crontab file. You just can't just write:

```
get my mail from mail.mydomain.com
```

and expect it to work. The syntax in your crontab is not easy to master, but it is not excessively difficult to comprehend either. First, there are 5 time periods that cron looks for. You start your crontab entry with these. Here is the order and some examples:

Table 1. Guide to Cron times

Minute	Hour	Day (of the month)	Month	Weekday
15 = quarter past	18 = 6:00 PM	12 = 12th day of the month	4 = April	0 = Sunday
40 = twenty to	2 = 2:00 AM	9 = 9th day of the month	10 = October	6 = Saturday

You will not be able to use all of them at the same time. If you have used the first four, you do not need the last one. This last one, the weekday, is particularly useful because it lets you run jobs once a week. There is also another way of doing that and we'll talk about it shortly. If you don't wish to specify a particular time period, you must substitute an asterisk (*).

Once you have decided when you want a particular command to be run, you add the command itself at the end. Thus, a typical crontab entry will end up looking like this:

```
30 3 * * 0 $HOME/bkup_script
```

which runs a script in your home directory to back up your files at 3:30 AM on Sunday. If you entered this into your crontab, you would simply save the file by pressing ESC + :wq which is a vi command. Vi is normally the editor that crontab uses by default, but you may use a text editor other than vi, by typing export VISUAL=pico, for example, which would allow you to use the pico editor. Every time you want to alter, add or delete an entry, you would first type

```
crontab -e
```

Enter whatever it is that you want to get done and then type

```
ESC + :wq
```

(or the combination of keys used to save a file in your particular text editor of choice). If you're curious about what's in your crontab file and want to list the jobs you have programmed, type:

```
crontab -l
```

If you want to delete your crontab file, type

```
crontab  -r
```

Variations on a theme

Crontab entries don't have to necessarily have just numbers in them. We can combine the numbers with other characters to modify how commands get carried out. For example, I have a USB webcam that doesn't really do what it's supposed to, which is to take a picture every minute and then shut off. It takes the picture all right, but it doesn't shut off. So I wrote a script to shut it off and then I added a crontab entry to call this script every minute. This is what I added:

```
0-59/1 * * * *  $HOME/shutoff_cam >/dev/null 2>&1
```

Let's look at this one part at a time

```
0-59/1
```

basically means that between the 0-59 minutes of every hour, at every 1 minute interval, the camera is to shut off. To show you how useful cron is, I remember seeing a James Bond movie where the perpetual bad-guy, Blofeld, was brainwashing girls to carry out biological attacks from a base in the Swiss Alps. He would play these hypnotic tapes to the girls every evening. There is one scene where you see Blofeld and one of his minions switching the tapes manually. If only they had had a Linux computer! They could have done this:

```
30-45/3 22 * * *  mpg123 /home/esblofeld/brainwash_girls.mp3 >/dev/null 2>&1
```

which would play the brain-washing instructions at 3 minute intervals between 10:30 and 10:45 PM.

Disclaimer: PLEASE DO NOT TRY BRAINWASHING TECHNIQUES AT HOME! ALSO, LINUX ONLINE DOES NOT ENDORSE THE WORLD DOMINATION SCHEMES OF SPECTRE. THIS IS ONLY USED AS AN EXAMPLE. THE ONLY WORLD DOMINATION SCHEME WE ENDORSE IS THAT OF LINUS TORVALDS.

We should also point out something that you've probably already noticed in the two examples above; that they end with

```
command >/dev/null 2>&1
```

We tacked this on the end because cron, by default, mails a "report" to you of the command you carried out. This is so you can either get the output directly in the mail, and/or to see if the command was successful. You may have made a mistake when you added an entry to your crontab (like typing the wrong path or the name of a command wrong). That way, you're notified and even if your job was important and you missed the first one, you can correct it and then you won't miss any others. Again, in the examples above, if we got a mail every time the command was carried out (every minute or couple of minutes), your in-box would quickly fill up with useless mail. Therefore, we tack that on so that cron will send notification of those jobs to /dev/null (ie. the trash).

Here are some other examples of variations:

```
0 0 15,30 * * cat /var/log/mysql.log > $HOME/mysql_use
30 8 * * 1-5 /usr/bin/who
```

The first one makes use of the comma, which means 'and'. In the first example, we see that we will get a report of MySQL use on the 15th and 30th of every month (except February, of course!). The second one will run 'who' which tell us who is logged in, every weekday (1-5) at 8:30 AM. This would be a particularly good one for systems administrators who want to see who's working (or at least who's logged-in) at the time they also start work.

Permissions for cron The ability to use cron is regulated in the same way as 'at'. Those in

```
/etc/cron.deny
```

are not allowed to use cron and all other users are allowed. If you have a

```
/etc/cron.allow
```

file, this supersedes cron.deny (ie, cron.deny is ignored) and allows only those listed in it to use cron.

cron.hourly, cron.daily and cron.monthly

Most Linux distributions have three directories in /etc called cron.hourly, cron.daily andcron.monthly, which, as you may have already guessed, lets the systems administrator run jobs on an hourly, daily or monthly basis. Simply by placing a shell script here, jobs can be carried out at those intervals. There is no need to have a crontab entry for these jobs.

As you can see, the sky is the limit with the things that you can do with cron. It won't get you to the point where you've programmed absolutely everything, letting you pass your working hours at the beach, but it will make your life a whole lot simpler.

Installing programs and upgrading versions

As system administrator, one of the most important tasks is to install programs. This also implies un-installing some as well. The procedure to follow depends mainly on the Linux distribution you're using. There may be other factors as well, like the install scripts provided by developers and companies.

What's your flavor?

It was in vogue a few years ago to refer to different Linux distributions as "flavors" of Linux. Although there is a movement afoot to standardize the installation of Linux, there exist important differences among the major distributions on the market. The less well-known distributions normally pick up their installation procedures from the bigger players. Each has their own "package" install system. Let's look at the best known ones:

Slackware

Slackware (http://www.slackware.com) has traditionally been the purists' distribution of choice. As far as this writer can gauge though, it's being gradually upstaged by Gentoo Linux in that role. (More on Gentoo later) Slackware relies on a system that installs tarballs with a *.tgz extension. Typing:

```
installpkg program.tgz
```

installs a program. Use the option -warn with installpkg if you want to know what new files are going to installed. If you're of the "Doubting Thomas" persuasion, you can appreciate this option. Typing:

```
removepkg program.tgz
```

will remove the application and any other supplementary files, like documentation.

You can upgrade programs to newer versions with this command:

```
upgradepkg new_version.tgz
```

Debian GNU/Linux

Debian (http://www.debian.org) has always been the preferred distributions of another type of purist: the licensing purist. Only truly open source packages go into the official Debian release. That doesn't mean that others can't create non-official packages. They often do. Regardless of its status, if you get a new package, you can install it the traditional way:

```
dpkg -i package.deb
```

That will install a program on a system running Debian GNU/Linux.

Debian is perhaps the best distribution out there if you're interested in keeping your system's packages up-to-date in a fairly painless way. This is accomplished by using the apt-get system. apt stands forAdvanced Package Tool. You can really take advantage of this tool if you've got a broadband or better connection. To set up for using it, type:

```
apt-setup
```

and it will ask you about information about where you want to get the packages (mirrors) and other ones related to keeping your system up-to-date. Now you're free to start using it.

To simply install a program, you would type:

```
apt-get install new_package.deb
```

It will look for the package on a remote server and install that package. What's even better is that it will automatically resolve any dependencies that you might have. We all know how nasty packages can be when you haven't got the right libraries or other program installed that it needs. apt-get rivals aspirin as a headache reliever in these cases. Likewise, you can use this command to uninstall programs:

```
apt-get remove unwanted_package.deb
```

The real advantage to this system is that you can use it to update multiple packages or even update your whole system. By typing:

```
apt-get update
```

and then

```
apt-get upgrade
```

you can ensure that your machine is always running the latest offerings from the folks who develop Debian.

Red Hat and other RPM based distributions

At the time of this writing, the three most popular commercial distributions are Red Hat, SuSE and Mandrake. These distributions use the popular Red Hat Package Management system (RPM for short) to install and upgrade Linux programs.

To install a program on any RPM based distribution, type:

```
rpm -i new_program.rpm
```

(the -i option means install). That should simply install the program on your system and any documentation that accompanies it. If you're not sure whether you have that program installed already, you might want to check out your inventory of installed RPMs using the -qa (query/all) options and piping the output to grep.

Let's say you wanted to look for popular Apache modules you might have. You would type this:

```
rpm -qa | grep mod_
```

You might get output like this:

```
mod_php4-4.0.4pl1-90
mod_perl-1.25-30
mod_python-2.7.2-27
```

This comes in handy if you've heard about some security alert pertaining to certain packages. You can see exactly what packages and versions you've got on your system. If your version number is among those vulnerable, you can pick up a new RPM at the company's FTP site and update it:

```
rpm -F new_improved_package.rpm
```

Something you may want to do to increase your security is to verify the packages. There have been instances of crackers switching packages on FTP servers. Not too long ago, there was a case where a "trojaned" copies of the open source version of Secure Shell, OpenSSH, appeared on some FTP servers as well as cracked versions of Sendmail, the world's most popular email server program. This is particularly nasty, so you would want to avoid something like this happening. We'll go into security in greater detail, but the RPM system provides a way to verify that the packages you download are authentic and haven't been tampered with.

```
rpm -v -checksig some_package.rpm
```

This presupposes a couple of things:
You have gpg, the GNU project's Pretty Good Privacy clone, installedYou have retrieved your Linux distribution company's public key.Check with your vendor for details on getting their public key. These days, there's no such thing as paranoia when it comes to protecting your machines and network. As the saying goes: Caution is the better part of valor.

Windows substitutes - LindowsOS and Xandros

We're just going to mention these distributions briefly. The reason for this is because these new distributions, LindowsOS and Xandros, are meant, in the opinion of this writer, to be Microsoft Windows substitutes. If you consider yourself an intermediate level user of Linux, you would find these distributions too limited to employ them in business, scientific or other technical environments. They are, in fact, ``dumbed-down'' so to speak. Far be it for me to imply that anyone who uses these distributions are dummies, but they are hostile to tweaking in the way that the accomplished Linux users like. They are also geared around the use of WINE, an MS Windows emulator, to run Microsoft programs without a lot of fuss, so ideally, their environment is the home or office workstation.
Both are based on Debian, so the rules for install and updates this distribution apply to them.

Gentoo Linux

In the Gentoo Linux distribution, the installation and update method is known as Portage. The work is done via the emerge command.
To get the latest set of Gentoo packages, type:

```
emerge rsync
```

Then

```
emerge -up system
```

After, you can install new package. It is always best to do it with the -p option. This will check for dependencies.

```
emerge -p PACKAGE-NAME
```

You may update installed packages using the -u option.

```
emerge -u PACKAGE-NAME
```

For more information on administration of Gentoo Linux systems, you can visit the documentation section of their webpage: http://http://www.gentoo.org/main/en/docs.xml
Tarballs: The Universal SolutionAs we mentioned, most distributions have their own ways to install and update packages. There are some developers, however, that only provide tarballs for their applications. This is not anything to be worried about and just adds a few extra steps to the installation procedure you're accustomed to with other distributions.

First, you would download and unzip the tarball in some directory. I prefer to create a subdirectory in /usr . You might want to call this directory /newapps, for example. To be on the safe side, I usually create a subdirectory for the app. If the tarball's correctly made, it should create its own subdirectory, but I've had my share of incorrectly made ones that have littered up my nice /newapps directory. Let's say you've got a tarball called nice_network_monitor_app.tar.gz. You could create a subdirectory called /nnma, for example, and unzip and untar the tarball there.

```
tar -zxvpf nice_network_monitor_app.tar.gz
```

This will unzip and untar the file. Then you should go in and read the INSTALL and/or README files provided with the tarball.

Installation is normally a question of doing the following:

```
./configure
```

This will usually find out if you've got the right libraries and programs on your machine that the application needs to be compiled and to run. A lot of stuff will appear that indicates that it's looking for the things it needs. Don't be alarmed if you see a few messages indicating that it can't find some of what it's looking for. If you don't see any error messages in the end that tell you that the program can't be installed, then there's no problem. At this point, if you haven't seen any fatal error messages, you can type:

```
./make
```

This command actually compiles the program and creates the binary or executable file. Don't be alarmed if your CPU usage goes off the scale. That's the compiler at work. After this, you would then normally type:

```
./make install
```

This installs the program into the path where the author felt it should run from. This is usually /usr/bin or /usr/X11/bin or it might be even put into /bin.

Steps one through three can be done as a normal user (we mentioned this in an earlier section on root privilege sharing). ./make install must be done by root or by someone with root privileges.

Notes:
grep is one of the important commands we talked about in our Getting Started With Linux Course. With grep, you can find words in files or in the output of programs

Back-up early, back-up often

The second most important part of being a system administrator (some might argue the most important), is backing up your system. Getting a computer up and running in the first place precludes data storage, but it's the storing of data that makes computers so important. Practically every business activity in the world today is run by computers or at least relies on computers to be more efficient. Let's put it this way: If you have a business and the data you stored suddenly disappears, you are out of business.

Despite the importance of data storage, backing up data on a Linux box is actually one of the easiest and least labor intensive parts of being a system administrator. After the initial decision as to what data should be backed up, when and to what medium, it can all be automated.

Tarred and feathered

Backing up to another machine

One method of assuring backups is to put your data onto another machine. The most efficient way of doing this is to set up a cron job (cron is a daemon that is used for scheduling jobs to be carried out automatically on a regular basis. Consult: *man cron* for more information) to do everything, ideally, at night while you sleep. For those who sleep during the day, that's OK too. You can set a cron job for any time.

First, you would make a tarball. Create the file mystuff.tar.gz inserting into it all the files in the directory /home/acme-inc/

```
tar -czpf mystuff.tar.gz /home/acme-inc/
```

Now you have a tarball of the entire directory /home/acme-inc/. Now, we need to get that tarball to another machine on your network.

The first way is kind of a fudge way of doing it. I don't mean to imply that it's not a good way. It's perfectly valid, assuming you have a webserver installed on the machine whose files you want to back up. If you were hosting virtual websites with Apache, for example, this solution would be ideal and fairly easy to set up. You could make regular backups of these sites and make sure those backups sit safely on another machine in your network.

You need to have the program wget installed on the target machine. (ie. the machine you're going to store the backups on).wget is a GNU utility to retrieve files via HTTP and FTP protocols. We'll use this because it's fairly easy to retrieve your tarball because you'll know exactly where you put it. Plus wget is non-interactive. You just have to program your cron job with

wget + URL to the machine

and let it do the work.

Now that you've got the tarball made, your best bet is to copy it to the machine's root webserver directory. Usually this is something like /var/www/. Your mileage may vary. You'd just issue the command to copy or move the file there.

```
cp mystuff.tar.gz /var/www/
```

Now the file is in /var/www/. If that machine is called 'hal9000' on your local network, your other machine, would issue this command to retrieve that file:

```
wget http://hal9000/mystuff.tar.gz
```

We could write up a script to do the first part. It would look something like this:

```
#!/bin/sh
```

```
# script to create backup file and
# copy it to webserver directory

# create tarball

tar -czpf mystuff.tar.gz /home/acme-inc/

# move/copy tarball
# change cp to mv if your partial to moving it
# feel free to change /where/it/is to the real place

cp /where/it/is/mystuff.tar.gz /var/www/

# end script
```

Call it backup_script, for example and make it executable with

```
chmod u+x backup_script
```

You can now place this in your crontab to get it executed when you wish. Use

```
crontab -e
```

to edit the file.

```
* 4 * * * /where/it/is/backup_script
```

Some caveats

The previous example is not very secure. It is just meant to introduce the general idea. In order for you to use this scheme, it's quite possible that your tarball gets copied to a public server. If your webserver isn't available to the public, as is the case with many home networks (make sure you're blocking access to port 80) then no problem. But if you have sensitive information in the file, then you don't want to have it in a place that's accessible to the public. Then we'll go to plan B.

Plan B

Plan B is going to be a lot more secure, but a bit more complicated. Such is life. It's easier to leave your front door open than to put on a lock but it's safer to use locks. First, to carry out this plan, you need to have a couple of other things installed on your machines. You'll need OpenSSH server and client. These come with the scp (secure copy) program. You'll also need the package expect. Expect is a program that will talk back to interactive programs - substituting human user intervention. In our previous example, wget was non-interactive. It just got the file. scp is different. It expects a password, so to give it what it's expecting we use expect. I expect you get my meaning ...
You'd create a tarball in just the same way as before, but this time we won't copy it to the webserver. We'll just leave it where it is. scp will retrieve the file where we tell it to. To carry out backup Plan B, we'll need to modify our script and create another one.

```
#!/bin/sh

# script to create backup file and
# scp it to another machine in our network

# create tarball

tar -czpf mystuff.tar.gz /home/acme-inc/
```

```
# call the expect script

./automate_scp mike@merlot:/home/mike PASSWORD

# end script
```

What I've done is use the script called automate_scp to move the file to my home directory on another machine in my network called 'merlot' (the use of names here requires a valid /etc/hosts file with entries in it for the machines in the network). My real password would have gone where you see PASSWORD. This could be a potential security risk, so make sure you chmod this script 0700 (that is, -rwx---) so other people logged into the system can't read it. Now we'll have to create the script automate_scp.

```
#!/usr/bin/expect -f
# tell the script it should be looking for a password
set password [lindex $argv 1]
# use scp to copy our file
spawn /usr/bin/scp mystuff.tar.gz [lindex $argv 0]
# be on the look out for password
expect "password:"
# here comes the password
send "$password
#that's all folks!
expect eof
```

Selecting which data to backup

It's probably best to be selective about what data you back up. In most cases, you don't need to back up everything. For example, when I do backups of my home directory, I leave out the cache files left by Mozilla and other web browsers. To me that's just extra baggage that unnecessarily bloats the tarball. The best way I've found making backups is to create a file of what exactly it is you're interested in getting back when and if there's a problem with your hard drive. First, create a file from a directory listing:

```
ls -a1 > backmeup
```

The

```
-a1
```

options will do two important things. The

```
-a
```

will get the "dot" files - those that begin with (.) and the

```
-1
```

will make sure that the list turns out as a nice single column. It's easier to edit this way. Now, just go into the file and decide what you want and don't want to back up. If you're backing up, let's say, /home/steve/ then you will need to edit this file should you create any new files or directories in /home/steve/. If you create anything in already existing subdirectories, you don't need to worry. To make your tarball, just type the following.

```
tar -zcvpf my_backup.tar.gz `cat backmeup`
```

The program 'cat' feeds the contents of the file backmeup into the tar program and creates the tarball based on its contents.

Incremental BackupsYou can also make periodic backups using a system of checking for changed or new files. These are known as incrementalbackups. Basically, you use 'find' to look for files that have changed within a certain period of time that you determine is right for your needs. You'd start with a "master" file of sorts which would be your first tarball.

```
tar -cvzf acme-complete.tar.gz /home/acme/ > /home/admin/YYMMDD.complete.log
```

Here we've created a tarball of the entire directory /home/acme. Using the -v option with tar gives us verbose output to pipe to a log file. That's optional but it may come in handy, especially if you log your incremental backups as well. You can later run the utility 'diff' on the log files and get an idea of your productivity. Then again, you may not want to do that (if your 'diffs' don't turn out to be too 'different'). Now for the incremental backups:

```
find /home/homedir -mtime  -1 -print | tar -cvzf acme_complete.tar.gz -T - > YYMMDD.bkup.log
```

Using the option -mtime -1 -print, we'll look for files that have been modified or created within the last (1) day. These are then piped to 'tar' and those files are added to our original tarball. If you did backups every week instead of everyday, you would have to change -1 to -7.
You could do the first "master" tarball by hand, but it would probably be a good idea to automate the incremental backups. Here's is a sample script that would do it:

```
#!/bin/sh
# get the date in YYYY-MM-DD format
this_day=`date +%Y-%m-%0e`
# make the backup and create log file for each day
find /home/acme -mtime  -1 -print | tar -cvzf acme_complete.tar.gz -T - > $this_day.backup.log
# end script
```

Then you would now copy this to your favorite location for storing backups.

Backing up to CDs

The use of CD-Read/Write devices, also known as CD "burners" has exploded within the last few years. This is also a suitable and easy to use medium for backups. First we'll use the utility mkisofs to make a "image" of what we want to put on the CD.

```
mkisofs -o acme_bkup.iso -R -v /home/acme/
```

Let's explain some of these options. -o (output) should be put before the image file you're going to create. The -R option (Rock Ridge) will ensure that the file will preserve it's file names as they were originally. This is very important for Linux. You don't want all your files to end up looking like MS-DOS names! Finally, -v is for verbose, just in case you want to know what's going on. Now you can put it on CD. For this, we'll use 'cdrecord'.

```
cdrecord -v speed=2 dev=2,0 acme_bkup.iso
```

This example assumes a couple of things. One is that the device address of your CD-RW device is 2,0. You should run

```
cdrecord -scanbus
```

to find out what device address is valid for your system. The other is that you're going to write at double speed. This, of course, may vary.
These are command line options, so there should be no problem in combining these into a script to automate the process. Leave

a CD in the drive when you go to bed and voil�! - freshly roasted CD in the morning. It really can't compare with the smell of freshly brewed coffee, but it will give you more peace of mind.

How to administer the system's users

Adding users to the system

If you're the systems administrator, regardless of the environment (large, medium or small organization), one of your tasks is going to be to add users to the system. This means giving a person a user account on one or more machines in your network. The main issues that you have to deal with is to decide, based on either your own or the company's criteria, what privileges the user is entitled to.

First, let's look at the main ways to add users to the system. Nowadays, most major distributions have tools to do this graphically and painlessly. If you're just interested in adding users and giving them default privileges, then you can use these tools. Some distributions have a simple command line tool for adding users, named, appropriately enough, adduser. If you type the command

```
adduser
```

you'll be prompted for the login name of a user. It is standard practice to use the first letter of the name and the last name. However, just because it's standard practice, doesn't mean you have to do it. If you're working in a small organization, feel free to use

```
adduser susan
```

to add a user named Susan Johnson. It would make more sense to use

```
adduser sjohnson
```

to take into account the fact that your company could hire another person named Susan at some point. Other things to take into account are:

- You should not include white space in the name
- Avoid hyphens and underscores in the user name
- Try not to use excessively long names (If you have a user named Ralph Inladomakrempandipadic, then you may have to improvise something for good ole Ralph!)

More control

If you're interested in having tighter control over how you create users, you should use the standard command line tool useradd. Let's look at the options available to you. First, if we run the following command:

```
useradd -D
```

You will see how users are added to the system by default.

```
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

Let's dissect this a bit. In this example, any user will be added to a group called users.

```
GROUP=100
```

If you looked at /etc/group, which lists the names of the groups on your system, you'll find number 100 as being assigned to users. This is a default, catch-all group of sorts, that includes all users.

```
HOME=/home
```

As you already know, a user added to the system will be given a directory to keep his or her files. This is created, by default, in /home.

```
INACTIVE=-1
```

refers to the number of days it takes for the system to lock you out when your account expires. As you can see here, you get locked out real soon!

```
EXPIRE=
```

refers to a date when your account is supposed to expire. In our example, there is no date. This is probably fine for regular employees, but if you hired some outside consultant to do work for you, you might want to his or her account to expire at the conclusion of business. The best way to handle temporary workers (and especially fired employees) is to delete the account entirely. But even the best administrators are often over-worked and may forget, so setting an expiration date will ensure that an account cannot be used.

```
SHELL=/bin/bash
```

refers to the shell that users will have to run commands on the system. Bash, or the Bourne Again Shell, is standard on Linux distributions. That doesn't mean that you as a system administrator can't install other shells on the system and make them the default (The Korn shell, for example). The last item is very important for you as an administrator.

```
SKEL=/etc/skel
```

means that the files located in /etc/skel will be copied to the users directory when you create the account.

You may want to add files to this directory or you may want to make changes to them. For example, you may want to add a few lines to .bashrc to make it safer for users.

```
alias cp='cp -p -i'
alias rm='rm -i'
alias mv='mv -i'
```

or you could add this give to give your users a nicer looking login prompt

```
PS1='[?33[45m][?33[1;33m][@]u@h:W >[?33[0m] '
```

Feel free to change the values 45m and 33m if you don't like those colors. Not everybody's into magenta and yellow!

Now you're ready to add users to the system with more selective criteria. The "man" page for useradd is one of the better ones for a Linux command. Here you can see all the possible options with good explanations about what every one does. Let's look at an example using some of the most common options.

```
useradd -c "William Shakespeare - AKA, The Bard" -d /home/wshakespeare  -m -k /etc/skel -g 100 -s /bin/bash wshakespeare
```

Let's see what we've done. First, we've added William Shakespeare as a user to our system. The first option, -c, is a short comment that must come between quotation marks and helps identify our user in the /etc/passwd file. The second option, -d is where you want his personal directory to be created. The -m option creates the home directory and the -k option copies the files in the following directory (in our case, /etc/skel) there. The -g option puts the user in the group whose number follows. The -s option provides the user with a shell to use (in our case, Bash). Finally, the name the user will login with.

What you may have noticed is that we didn't use the -p (password) option. For the time being, Shakespeare is without a password and therefore cannot log in. This is a bit more complicated. Most Linux distributions nowadays use a system of shadow passwords. These are encrypted so that a normal user can't look at the passwords in the system. The problem we have with useradd, is that it needs an already encrypted password to work correctly. Let's say I wanted to give Shakespeare the password 'h4ml3t'. If I added -p h4ml3t, the system would reject his login because it wasn't encrypted. If you looked at the /etc/shadow file where these encrypted passwords are stored, you'd find something like this next to a user's login name:

```
Q37spqpXAsl1Y
```

User jsmith may have the password 'mrT1bbs' but it would appear as something like

```
jsmith:F54spqpRAsl1X:12043:0:99999:7:::
```

in /etc/shadow. This is because the password has been created as an md5 hash, which is an encryption algorithm. So, the bottom line here is that you have two options. An administrator that simply needs to add an account here and there can use the options above, minus the -p and then run

```
passwd user
```

and provide a password. passwd will then create the md5 hash automatically. If you found yourself needing to create many users at the same time, you might want to look into some tools that create these hashes beforehand and then you can provide them with the -p option.

Deleting Users

As an administrator, you will need to delete users as well. People leave and move on to other things. They come for short periods and need temporary accounts. Accounts need to be deleted when they are no longer going to be used. This is of maximum importance from a security point of view. We've all heard stories of disgruntled former employees logging back into their accounts and using the system to either embarrass their former employers by playing pranks or to actually try to do serious damage.

Again, as we stated before, your Linux distribution choice may have tools to do this. Some make this easier by also deleting users home directory (and all the files!) along with the account. The standard tool that comes with all distributions, however, is deluser It is simply run like this:

```
userdel wshakespeare
```

where you substitute the wshakespeare with the username you wish to delete.

In this particular case, we've deleted William Shakespeare's user account, but not his home directory and files. To get rid of any trace of oldWill, then you would use the -r option:

```
userdel -r wshakespeare
```

which would remove everything, including his email. We may not want to do this, however. Many people leave leave their employment but their work remains property of the company because of contractual obligations. In this case, we would obviously want to keep the user's files until they could be looked at, copied to another user's account or stored. So be very careful with the -r option.

Permissions

As you already know, everything that exists on a Linux computer is a file. As we explained in previous section, you hardware is also represented as a series of files (an IDE hard drive is /dev/hda, a modem might be /dev/ttyS1, for example). Seeing that everything is a file, these files have to have a series of permissions assigned to them. These permissions govern the use of and access to a given file. Specifically, they determine whether a file can be read, written (ie, modified, deleted) or, in the case of binary programs, executed. These permissions have to be strictly enforced or you could do a tremendous amount of damage to a system.
As we saw in the first course, you can see the permission information by typing:

```
ls -l todo_list
```

Let's say you've just written a to-do list with your favorite text editor. It's likely that on a normal system, you will have created this file with what are known as 'easy' permissions. The output of the previous command would look like this:

```
-rw-r-r-    1 bob     users         155 Mar 26 12:33 todo_list
```

The first set of permissions (rw) apply to the owner of the file, bob. These are read and write permissions. The group the file belongs to, users, can read, as we see in the second set of permissions (r). The third set refers to others. This file can be read (r) by anyone with access to the system and bob's home directory.
Let's look at another example. Let's say I wanted to create a Bash shell script to backup my work in a directory called /project. Ideally, I could run this as what's known as a 'cron' job. I would then make the script executable. If we looked at the permissions for this file, it would look something like this.

```
-rwxr-r-    1 bob     users  95 Mar 26 12:38 backup_work
```

As you can see, there's an 'x' added to the set of permissions for the owner of the file.

Assigning permissions

So how do you give a file the permissions you want? Well, as a user, you are restricted to giving permissions to only those files that you own. As root, you can give permissions to any file on the system.

Let's take the two aforementioned files as examples. First, if you wanted todo_list to be confidential, so to speak, you could remove read permissions for others. There are two ways to do this. Both use the command chmod

```
chmod o-r todo_list
```

which is the more literal way of doing it. As you can see, we have set others (o) minus (-) read (r) for the file.

We can also use the number scheme. On a Linux system, permissions are assigned number values. Read permissions are assigned a value of 4, write permissions a value of 2 and execute permission a value of 1. Therefore, if I wanted to deny others permission to read my to-do list, then I could also type:

```
chmod 640 todo_list
```

if you used the -c option, you could also get a brief explanation of what you did.

```
chmod -c 640 todo_list
mode of `todo_list' changed to 0640 (rw-r---)
```

First of all, as you can see in the output, there is a 0 before the 640. This refers to directory permissions and is not applicable here. How did we get 640? Well, read (4) + write (2) for the owner and read (4) for the group plus no permissions (0) for others equals 640.

Let's look again at the other file to backup our work. We would add execute permissions for the owner to it. One way is:

```
chmod u+x backup_work
```

which literally adds (u+x) permission to execute for the owner. We could also user our number system:

```
chmod 744 backup_work
```

and assign execute permissions for the owner. That is, read (4) + write (2) + execute (1) equals (7) plus read (4) for the group and read (4) for others.

Directory Permissions

A directory can be assigned permissions just like a file. If you look at the root directory in Linux, you'll find that most of the directories have this set of permissions:drwxr-xr-x. The letter d is to distinguish it as a directory. The rest are like the normal file permissions. There is usually one exception to this and it is the /tmp directory. Here is the normal permissions for the /tmp directory: drwxrwxrwt. A lot of programs need story temporary files (hence, /tmp) in this directory, so there are permissions to write to it by group and others. The letter t at the end is a very important difference here as well. This is known as the sticky bit. This means that the files created there cannot be removed, even if there are write permissions. This is done for security reasons. Under normal circumstances, if I had permissions to write to a directory, then I could logically remove any file in there, regardless of whether I had created it or not. You can't do this with the sticky bitset. Unless you had created the file, you cannot remove it. That is, anyone can create a file in a directory with the t but only the owner of the directory and file can remove it.

To test this, do the following. Create two directories; one called 'jack' and the other called 'jill'. Set the sticky bit for 'jack' like so:

```
chmod 1777 jack
```

Now enter that directory and create a file:

```
touch pail_of_water
```

Now set write permissions for 'jill' for everybody without the sticky bit

```
chmod 0777 jill
```

and create another file:

```
touch up_the_hill
```

If you have created another user account (or you could create one now with what you learned in the previous section), you could log in as that other user, enter the directory 'jill' and remove the file up_the_hill. But now, enter the directory 'jack' and do:

```
rm pail_of_water
```

You get the proverbial door slammed in your face.

```
rm: cannot unlink `pail_of_water': Operation not permitted
```

That file is "stuck" in that directory until only the owner decides to delete it, hence the term sticky. If you've created some directory for others to write to (you're working on project with others, for example), it's always a good idea to set the sticky bit. That way, only you are responsible for making mistakes.

Hard and Symbolic Links

There exist types of files known as hard links and symbolic links. These are created for the purpose of providing easier access to certain files. To use a real-world example, I created a symbolic link to my Mozilla bookmark file so that I can open it up and access my favorite and most frequently used pages when I am using other browsers. Mozilla places its bookmarks in an HTML formatted file in a deep and rather cryptically named subdirectory. This is an ideal opportunity to use a symbolic link. I can't alter where Mozilla places it's bookmark file, but I can create another file that is a link to the actual bookmark file but is easier to get at.

```
ln -s .mozilla/bob/k4til3zu.slt/bookmarks.html mozilla_bookmarks.html
```

The -s option stands for symbolic. The permissions for a symbolic link look like:

```
lrwxrwxrwx  1 bob users 41 Dec 22 16:29 mozilla_bookmarks.html -> .mozilla/bob/k4til3zu.slt/bookmarks.html
```

As you can see, there's an 'l' at the beginning to indicate it is a symbolic link. Also, you'll see that it has all three kinds of permissions. Despite these permissions, this file cannot be modified by anyone other than its owner and, unless it's actually a binary file, it isn't executable either.

If you use ln with no options, you create a hard link, which is essentially a linked copy of the same file instead of a path to the original. Since my intention is to explain the file permissions of these links, we'll save the pros and cons of creating a certain type of link for another section. If you created a hard link to that file, the permissions would be those of a normal file.

File Ownership

Linux, like Unix on which it is based, is a possessive operating system. As we mentioned in the last section, everything in Linux is a file. We should now add that every file must have an owner. There is even a user on most Linux systems called nobody to provide for the fact that even nobody has to be somebody!.

You can assign file ownership with the command chown. If you're the administrator, root, you can transfer ownership of a particular file to another user. Let's say you've created some file doing your routine administrative work- you've dumped some output from a log file, for example. You may want to transfer that file's ownership from root to yourself as a user. You would do the following:

```
chown bob:users log_20030226
```

Now you're the owner of that file. If you hadn't done this, and wish to modify it - write some commentary in the file, for example, you would have had to switch to being user root. Now you can safely modify it logged in as an ordinary user.

Cleaning up the mess

Let's face it. Kids aren't the only ones who are messy. A computer's hard disk can't rival even the worst teenager's room. Anyone who's used the Emacs editor available for Linux can attest to the fact that it will leave around files named .saves-[a number + your hostname] in your directory. When you edit a file, Emacs will also create a file with the same name but with a tilde at the end of it. That's OK if you've messed up and want to be able to get back to the state you were at when you started editing. But what if you're never going to edit that file again? You should probably clean up those files that are cluttering up your home directory, at least periodically. You have the tools at your disposal to do this without hunting around for them.

```
find . -name ".saves*" -print -exec rm -f {} ;
```

This finds any file named '.saves' at the end and passes it to the rm command. Again, as with anything that uses the -f (force) option, you should be extremely careful with this. If you were just looking for tilde files, it would be advisable to do it by directory:

```
find /work -name "*~" -print -exec rm -f {} ;
```

As you can see, you're only one tilde away from disaster! Using the same techniques, you may have to enforce a company policy that may prohibit using P2P file sharing networks. Management may frown upon having MP3 files on their servers and it would fall upon you to get rid of them. Again, you would have to exercise some caution if workers had permission to listen to MP3s from a special company-wide selection.

```
find /home -name "*.mp3" -print -exec rm -f {} ;
```

would only eliminate these files from users' directories in /home.

System Administration- An Overview

All the hoopla

As mentioned before, some people want you to believe that administering a Linux system is like arranging a peace settlement in the Middle East or understanding a Jackson Pollock painting. Some years ago when Linux was really a hobbyist's system it was considered difficult. Now Linux has gone mainstream and there's nothing taxing about running a Linux system. You do not have to be a computer "guru" to use it. Anybody can be a Linux "administrator".

Years back, if you had the title 'system administrator' it was comparable to the role of the arch-angel Michael. (Michael comes from the Hebrew words meaning He who is like God). That usually meant your own parking space at the company and a seat at the executive dining room. There are some system administrators for large corporations that run mainframes who enjoy these privileges (and rightly so). However, if you've successfully installed the Linux operating system then you too can now proudly wear the badge of 'system administrator'. It doesn't matter if you're setting up one computer running Linux or a bunch of computers in your small business or a computer room in your local community center or school, you've now signed on to become the 'big cheese'.

The role of root

Using the 'root' account is like being God on a Linux system. (Hence, my earlier reference to the archangel Michael). That means that you want to be extremely careful when working as root. With something as simple as a wrong keystroke you could do a great deal of damage. Before you actually sit down and work as root for the first time, I would recommend going into the file known as .bashrc (if you're using the bash shell, which is the most popular one) and adding a few aliases in there. An alias is nothing but an entry in that file that says that a certain command that you type can perform additional actions above and beyond its default behavior. For example, if I type:

```
rm unwanted.doc
```

in a terminal, unwanted.doc is going to byte heaven (or hell).

```
rm
```

is for removing or deleting files. There is no undelete that is practical and easy when you're using a shell, so If you didn't want to delete that, you're pretty much out of luck. But if I add an entry in my .bashrc file like this one:

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

it makes sure that I get asked before the file actually gets deleted. You may want to do the same with other potentially dangerous commands.

```
alias cp='cp -iv'
```

makes the copy command interactive (i) and verbose (v) so it will both ask you and then tell you what you did.

```
alias mv='mv -i'
```

also makes the move command (used for moving and renaming files) interactive.

There are people who say that adding these to root's .bashrc is something that 'wussies' do. I always ask them: If a sailor tied a cable on to himself/herself before he/she went out on deck in thirty foot seas to fix something, would that be considered a 'wussie' move? Making a mistake is comparable to encountering a rogue wave on a calm sea. There really isn't anything comparable to being in rough seas sitting in front of your computer, but just as dangerous rogue waves have been known to appear on calm, sunny days and sink boats, silly mistakes have ruined projects. Better to keep a buffer zone between you and your mistakes.

Delegating authority

Back to the nautical motif for a moment; just as one ship generally doesn't have two captains, it is rare that a small organization would have two systems administrators. There's usually not too much benefit in delegating authority in this setting. People are prone to making mistakes. Even a seasoned systems administrator has sometimes deleted files that he/she shouldn't have or messes up the configuration of something. If two heads think better than one, then four hands also might make more mistakes than two.

The use of sudo as an alternative

If you're the head systems administrator (or the only one) you can "deputize" your co-workers by installing and configuring the program sudo. In Unix/Linux speak, the term 'su' means superuser - that is, root. Only root has true administration rights and privileges, so this program allows others to "do" su, hence the name, sudo. Ok, Sheriff, time to fight the bad guys. Let's see what your deputies can do.

su can also stand for switch user. For example, if you had two accounts on a machine - let's say bob and harry - you could log on as 'bob' and do: su harry and then work as harry.

Your distribution should have this very popular program among its packages. If it doesn't, you can go to:http://www.courtesan.com/sudo and get Todd Miller's great application. After you've installed it, you have to create what's called a sudoers file. You do this by typing:

```
visudo
```

as root. This is essentially a special version of the text editor vi just for creating and editing the sudoers file.

Basic Vi commands

- ESC+i = insert text
- ESC+wq = save and quit
- ESC+u = undo

Here is an example sudoers file I have for my home network. It is not really as complicated as most are, but it gives a good basic idea of what you need to do to let other users help you out with some administration tasks.

```
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers man page for the details on how to write a sudoers file.
#

# Host alias specification

# User alias specification

User_Alias TRUSTED = mike, maria

# Cmnd alias specification

Cmnd_Alias INTERNET = /usr/sbin/traceroute, /usr/sbin/ntpdate
Cmnd_Alias KILL = /bin/kill, /usr/bin/killall
Cmnd_Alias TOOLS = /bin/mount, /bin/umount

# User privilege specification
root    ALL=(ALL) ALL
TRUSTED ALL=INTERNET, KILL, TOOLS
```

Let's break this down. First of all, we add the line

```
User_Alias TRUSTED = mike, maria
```

That means that the users mike and maria become the "trusted" users. And what are they trusted with? Jump down to the last line for a second. They are trusted with commands in the group INTERNET, KILL and TOOLS. What are those commands? Jump back up to the section

```
#Cmnd alias
specification
```

These trusted users can use ntpdate, for example, to keep the computer's time correct. More information on that command later. (One of your duties as system administrator will be to make sure your machines keep accurate time and display the correct date. ntp is probably the best package to use to do this.)

I've created a KILL group (sounds like Mafia hit men!) so other users can kill runaway process that can only be shut off by root normally. Some server process may have a problem and you might have to shut down that process. Starting it again is something that's not included here however. It might be best for these deputized users call the "real" system administrator and if that's you, for example, you may want to check out the configuration files for certain servers before you start them again. You may have to mount floppies or other partitions to get data from them, and that's where the TOOLS section comes in handy.
When the user executes a command that's authorized in the sudoers file, he/she first needs to type

```
sudo
```

and the command. For example, if you wanted to update the machines clock to the exact time, you would type:

```
sudo ntpdate atimeserver.nearyou.gov/edu
```

Then you need to type your user password. If you make a mistake, sudo will hurl insults at you (no kidding). Todd Miller has a good sense of humor and the results of botching a keystroke are sometimes hilarious!

You can add more commands and users to your own sudoers file. Whatever you think is prudent in your situation. There is some possibility for abuse. Use your best judgment.

Taking care when working as root

As I mentioned, there's a chance of doing some damage when you work as root. There are other ways to protect yourself besides putting aliases in your .bashrc file. One of them is using the program su.

su lets you work as root when you're logged in as another user. Good practice dictates that you disallow root logins from remote machines, so if you're performing administration tasks remotely, su is irreplaceable. The remote machine lets you log in as user fred, for example, and then you just type:

```
su
```

and type the root password. For all intents and purposes you've got a root terminal open now. That means that you can do anything - just as if you had logged in as root in the first place. You're really running the same risks by working as root, but you've at least eliminated the risk of logging in as root. That's quite important from a security standpoint. The real advantage to using su is the possibility to carry out individual commands. Let's say you've downloaded an application and it's only available as source code in a tarball. You can download the source code to your user directory and as a user you can run the configure and make scripts provided with most tarballs. That way, you minimize the risks of compiling the software. If you're satisfied with the binary file (the application itself), then you can use 'su' to install the application and its accompanying documentation in the usual application and documentation directories. Here's what you'd do:

```
su -c "./make install"
```

You need to add the -c (command) switch and put the actual command in quotes. Then type the root password.

As you see, you can run any command that root can. That, of course, means that you need to know the root password, so this is something that only the system administrator should do. It's not a good idea to be giving your password to other users so they can run commands. They can do that with sudo, as we mentioned earlier.
You can also use su to do things as other users. You're not just restricted to doing things as root. This might come in handy if you've got two accounts on a machine. Instead of having to log in as the other user, you could, for example, read a file that you've restricted permissions for. Let's say you've got a file called my_ideas and you've removed read permissions to it for your group and for others. Instead of logging in, you can type:

```
su fdavis -c "less /home/fdavis/my_ideas"
```

and you will now be asked for fdavis' password. Now you can access files from your other user account without having to log in. Other users may also give you their passwords to access things in their account. I question the wisdom of doing that, but in an emergency it's an option. A co-worker who's ill may give you his/her password to access important files. That's acceptable in that kind of situation it seems, but the ill worker should promptly change his/her password as soon as he/she's back at work.

As you can see, if used correctly, su is a great tool for getting administration tasks done safely. It's also useful as a time and trouble saver. Just make sure you don't abuse it.