Department of Informatics

LINUX SCRIPTING AND FILE PERMISSIONS

Network Security

Diego Sempreboni

1 UNDERSTANDING BASIC LINUX COMMANDS

Welcome to the first lab of Network Security. Even if you think you are familiar with scripting and bash, I would recommend to follow this lab carefully because sometimes being able to protect a network is a timing matter. During the whole course you will have to use the bash substantially, so this is a good opportunity to become familiar with this environment.

a. Print the current date

```
$ date
Sun 1 Feb 2015 21:44:51 GMT
```

b. Read the manual pages and info pages for the commands, head, tail and touch.
hints: Note that the command "man" may use the system pager (less or more).
To quit the man page type q. To go to next page use space bar. You can also use the arrow keys usually.

c. Explain the purpose of the following commands in 3 sentences or less each: head, tail, touch, more, less, man (use "man man" from the command line for this).

```
use "man head", etc. Or google for the answer!
```

d. If you wanted to print the first 3 lines of a file, and the last 5 lines of a file explain how you would do this using the commands, head, and tail. Print the first 3 lines and the last 3 lines of /etc/passwd.

```
First three lines of a file: head -3 /etc/passwd
last 5 lines of a file: tail -5 /etc/passwd.
```

e. Type "ls" to list your home directory contents.
type "touch file.txt" in your home directory
Now type ls again. Do you see the new file you just created?

2 COMPOSING BASIC LINUX COMMANDS

a. Page through /etc/passwd (Hint: use "less /etc/passwd" or "more /etc/passwd" from the command line. Identify the name of some user (the first field on any line. Recall that each field is separated by the ":" character). Do you see your own username in this list? Why or why not?
(*Hint*: google /etc/shadow for some hints on alternate possibilities to /etc/passwd)

```
On some systems, passwords may actally be stored in
/etc/shadow, which is not readable by ordinary users (so
less /etc/shadow does not work). In many systems, this
and other information can be configured to be provided
from a number of sources, including a network information
service or nis. View /etc/nsswitch.conf for details [and
for details about nsswitch, you can do "man
nsswitch.conf".
For further details, see:
http://en.wikipedia.org/wiki/Network_Information_Service
http://superuser.com/questions/580148/users-not-found-in-
etc-passwd
```

b. The last field of each line is the default shell for a user. Use the "cut" command to print out all the shells being used by users. Use "man" to identify how to use cut for this purpose.

```
cut -f7 -d':' /etc/passwd
```

c. You will find that the output of "cut" has many duplicates. Try to remove these duplicates using the command "uniq". To do this you will have to pipe the output of cut to the command uniq.

```
cut -f7 -d':' /etc/passwd | uniq
```

d. You may find that even after uniq, there are still some duplicates. This is because uniq requires its inputs to be sorted. (each Unix command does one job, and relies on others in the suite of tools to do other jobs. It is your job to compose them together properly). Now, to completely remove all duplicates, enhance the above pipeline, to sort the output of cut, before passing to uniq. The unix command for sorting a list of lines is, predictably, called sort. List all the unique shells you see now.

```
cut -f7 -d':' /etc/passwd | sort | uniq
```

e. Pick a file name which does not exist (e.g., */etc/password*), and try to list it ("ls <filename>") Notice that an error message gets printed on screen. This is an **error** message and gets printed on the standard **error** stream, rather than on the standard **output** stream. How do we tell this?

   Try to do the ls again, except, now, ***redirect*** the standard error to a file.

```
ls /etc/password 2>~/logfile
```

Read the contents of the file “logfile” created in your home directory
(hint: use “cat ~/logfile”).

Now repeat the procedure with a file which exists:

```
ls /etc/passwd 2>~/logfile
```

Now read logfile.
Notice that there is nothing in the standard error stream now. But something did get printed on the standard output, which is also output to the terminal.

```
This question is just trying out a series of commands. All
information is in the question itself. Please follow the
instructions on redirections, to understand how standard error
streams are used.
```

f. You can also redirect standard output. Instead of using 2> symbol, you can use “1>” in the above command. use this to list both /etc/password and /etc/passwd.
write down the contents of the log file in each case.
(in the case of standard output, you may omit the 1, and use “>” instead of “1>”. try this and confirm it works).

```
Please repeat the above commands from 2e for 2f. as shown
below (in the below, $ is the command prompt. User input is
all the text after the prompt.)
$ ls /etc/passwd 1>logfile
$ cat logfile
/etc/passwd
$ ls /etc/passwd >logfile
#Note >logfile== 1>logfile
$ cat logfile
/etc/passwd
$ ls /etc/password >logfile
ls: cannot access /etc/password: No such file or directory
(notice in the above that error prints to terminal even if
standard output has been redirectedd to logfile. In fact, if
you see logfile there is nothing in it:
$ cat logfile
$
```

g. The error messages are for humans to read, when composing commands in a shell script, the author of a script is less interested in reading the message and more in doing different things depending on the output of a previous command.

h. Use the || command to echo “sorry, could not find file” if a file cannot be listed. Redirect standard error to a logfile and standard output from ls to an “output.log” Try this to list /etc/passwd and /etc/password
hint: the format to follow would be “ls <filename>” || echo “sorry,...” [you need to add the output redirection syntax to this basic skeleton].

```
ls /etc/password || echo “sorry, could not find the file”
```

i. Before using a command such as cut, you may want to make sure that the file exists. Use the && command to ensure that the file exists, and then use the cut command to output the names of the shells of the users of your system.
hint: test that the file exists, before using the pipeline developed in 2.d

```
ls /etc/passwd && cut -f7 -d’:’ /etc/passwd
```

j. Develop a small shell script, using the ‘if’ construct. This script should output the contents of the file if it exists, and echo “sorry, cant find file” if the file does not exist. Test it on some arbitrary file that you create yourself, or on /etc/passwd and /etc/password.
hint: See 3.b, but only after you have tried this yourself.

```
The shell script looks as follows:
#!/bin/sh
if [ -f $1 ]
then
      cat $1
else
      echo "Sorry, cant find file"
fi
(type this in a file by invoking an editor. For instance, call
“nano myshellscript.sh” type in the following, and use Ctrl-X
to exit the file.
Now you may test your script by calling it as:
$ sh myshellscript.sh /etc/password
AND as
$ sh myshellscript.sh /etc/passwd
```

3 UNDERSTANDING FILE PERMISSIONS

a. Make sure you have a file called file.txt in your home directory (see 1.e). Now change directory to your home directory by typing “cd” at the command prompt. Now type “ls -l file.txt”.
What are the file permissions set for the new file you just created?

Explain each field in the file permission set.

```
ls -l file.txt
-rw-r--r-- 1 ns staff 0 12 Feb 17:44 file.txt
The file permissions I see are: -rw-r—r--
-rw-r—r— This first bit indicates a plain file (-)
-rw-r—r-- The next three bits are my permissions. I can
read and write but not execute file.
-rw-r--r-- The next three bits are the group permissions.
my group members can ready but can’t write. Similarly,
the last three bits are for others who are not in my own
group.
```

b. Create a directory testdir in your home directory using “mkdir testdir”. Explain the file permissions you see with ls -ld testdir

```
The permissions are exactly same as above, except that instead
of - (which indicates plain file), the type is d for
directory.
```

Why did you have to use the additional option “d”. Use man ls to find out. Also try without the -d option, i.e., ls -l testdir

```
Without using -d option, the contents of testdir will be
displayed. This can be a bit confusing especially when testdir
is empty. The below will create a file in the directory so
that you can see it is the contents which are displayed.
```

If the output above is confusing, try the following:
create a file in testdir, by typing “`touch testdir/file`”.
Now, try `ls -l testdir` again.

c. Open file.txt and type in the shell script below (note that the spaces around [and] are important!). Then, save and quit your editor. Now, at the command prompt, type in “`sh file.txt <filename>`” (use some file name which exists, and then use a filename which does not exist)”. Explain what you see.

```
#!/bin/sh
if [ -f $1 ]
then
    cat $1
else
    echo "Sorry, cant find file"
fi
```

```
cat <filename> will be executed if file exists; otherwise the
error message is echoed. See slides for how the program '['
works.
```

d. Observe that you have just created a text file and executed it by passing it as an input to the shell ("sh")! Now try to execute the file by itself, by typing

```
./file.txt <filename>
```

Did that work? Why or why not? Explain by looking at the file permissions for file.txt using the output of "ls -l file.txt" to see the permissions.

```
You can only execute a file if the x bit is set (in rwx of
permissions)
```

Now add execute permissions with chmod u+x file.txt.

Test that you can now execute file.txt with

```
./file.txt <filename>
```

e. Move file.txt to testdir using "`mv file.txt testdir`". (Make sure you are in your home directory before doing this!). Now list the testdir (`ls testdir`).

You can still execute file.txt. Except you have type in `(./testdir/file.txt <filename>)`. Check this works. The execute permissions are associated with the file rather than the directory.

f. Recall that file permissions work differently for directories.

g. Remove ls permissions from testdir by typing the following from the command prompt (make sure you are in home directory)

```
chmod u-r testdir
```

h. Now try list the testdir (`ls testdir`) and report the results.

i. Of course, the 'r' bit is different from the 'x' bit, or the 'w'.
For instance, if 'x' bit is set (it should be), you should still be able to traverse into testdir, and read its files or even execute them. Ensure this is true, by typing in:
(./testdir/file.txt <filename>)
You have just executed a file from a directory that you cannot read. So effectively, you cannot tell if the file file.txt is in testdir, but if it is there, you can execute it!

j. You can even remove files in testdir (because you have 'w' permission).

```
rm testdir/file.txt
```

k. You can even remove testdir itself, Ensure this with:

```
rmdir testdir
```

(Note you can only do this if testdir is empty. If it is not empty you can use "rm -rf testdir" to remove "testdir" and all its contents including other sub-directories. rm -rf is a dangerous command, be careful where you use it, so you don't lose files!)

l. How does Linux ensure you have the permissions to remove testdir? Where is this permission stored? Explain your answer with a simple experiment. (hint: each directory can be considered as equivalent to a simple file, which maintains the mapping filename—> inode for each file in the dir.)

```
suma-2:play ns$ mkdir testdir
suma-2:play ns$ mkdir testdir/subdir
suma-2:play ns$ mkdir testdir/subdir-nonremovable
suma-2:play ns$ chmod u-w testdir/subdir
suma-2:play ns$ touch testdir/subdir/file.txt
touch: testdir/subdir/file.txt: Permission denied
suma-2:play ns$ echo "cant create files in subdir without
write permissions"
cant create files in subdir without write permissions
suma-2:play ns$ echo "but we can still remove subdir" && rmdir
testdir/ subdir
but we can still remove subdir
suma-2:play ns$ ls testdir #see that it does not have subdir
subdir-nonremovable
suma-2:play ns$ #now, we change permissions on *testdir*
suma-2:play ns$ chmod u-w testdir
suma-2:play ns$ #after this, we cannot remove any files in
testdir, such as subdir-nonremovable
suma-2:play ns$ rmdir testdir/subdir-nonremovable/ || echo
"cant remove" rmdir: testdir/subdir-nonremovable/: Permission
denied
cant remove
```