Dr. Linux TEST CASE

Do you want to find out what lies behind a command or who has just logged on? Marianne Wacholz shows how all this can be tested with a few inputs on the keyboard

Shell/bash The "mediator program", which accepts your commands, processes them and finally passes them to the kernel for execution. Under Unix operating systems you can choose between diverse shells with varying functionality and operating philosophies. As a rule, on Linux systems the bash ("Bourne Again SHell") is used as the standard shell, but other shells such as the csh ("C-Shell") and/or the tcsh (an extended C-shell with the option of editing the command line) are also frequently installed.

Stop root!

Q I like to interrupt programs, which I start on a **shell**, with the key combination Ctrl+Z, in order to execute other commands in the meantime. Unfortunately this doesn't work when I edit/configure files using **su** as root. Is there any other command to temporarily deactivate the root shell created by *su* and to bring it back again later?

Dr. Linux: There is indeed and the command is *suspend*, which is entered in the bash (or csh/tcsh). *suspend* is a so-called shell built-in, meaning it is a command forming part of the scope of performance of the respective shell. It interrupts the superuser message with *su* in the same way as the key combination Ctrl+Z does with normal programs/processes, i.e. the root shell is put to sleep and instead you get back your original shell, on which you can carry out new commands completely as normal. First you will see the output *Stopped* followed by which command has just been stopped:

```
perle@maxi:~> su
Password: your_root password
maxi:/home/perle # suspend
[1]+Stopped su
perle@maxi:~> [input tasks to be performed as
Perle user]
```

To get back to the root shell, enter the command *fg* (*f*ore*g*round). You leave this in the usual way with the command *exit*:

```
perle@maxi:~> fg
su
maxi:/home/perle # [other commands to be
executed as root]
maxi:/home/perle # exit
exit
perle@maxi:~>
```

Knowing what you're doing

Q I start programs on a console by typing in their names. But basically, I'm often not clear as to whether this is a proper, compiled program or for example a shell script that I'm dealing with. I'd like to get to know my system better so that I know what kind of command I have just put in. How do I do this?

systems are, have some little complaints all of their own. Dr. Linux observes the patients in the Linux newsgroups, issues prescriptions here for the latest problems and proposes alternative healing methods.

Dr. Linux: With the shell function *type* plus the command name, you can see the origin of your selected program:

```
perle@maxi:~> type suspend
suspend is a shell builtin
```

suspend is thus a built-in command of the shell, while...

```
perle@maxi:~> type dir
dir is aliased to 'ls -l'
```

... *dir* is an **alias** for the command *ls -l*.

```
perle@maxi:~> type gimp
gimp is /usr/bin/gimp
```

su This command enables you to change to a different user ID on the command line. When you do this, a new shell with the user identification of the user specified as argument is created. If the latter is missing, it allows *su* to work for a short period as superuser root. Provided you are not already logged on as root, you will need the password for the selected user. You can drop the new identity again by entering the command "exit" on the command line.

If on the other hand, as in the case of *gimp*, this is a real file (or a link) in the filesystem, you can research more deeply with the command *file*. In the same simple way as using *type*, this identifies any unknown file types. For example, this means we can find out that */usr/bin/gimp* is a proper binary program, while the command *groups* is an executable shell script:

```
perle@maxi:~> file /usr/bin/gimp
/usr/bin/gimp: ELF 32-bit LSB executable, Intel
80386, version 1,
dynamically linked (uses shared libs), not
stripped
perle@maxi:~> file /usr/bin/groups
/usr/bin/groups: Bourne shell script text
```

When *file* cannot tell what kind of file a given entry is (or if the file must not be displayed), you will receive in the output the keyword "data". Separate details are provided by *type* with the output:

```
command is hashed (Path/to/command)
```

This means that the corresponding command has already been executed in the current session and has therefore been stored by the bash in the **Hash table**:

```
perle@maxi:~> type gimp
gimp is hashed (/usr/bin/gimp)
```

Checked out

Q Although I do have a bit of experience already with my Linux system, it sometimes happens that I edit a file in an editor, but when I come to save my work I discover that I have no write privilege for it. Is there a neat little command, with which I can query rights in advance, without searching through the long directory lists created by *ls –l*?

Dr. Linux: The command with which you can get rapid and specific information about a file or directory is called simply *test* and is a shell built-in. It is called up as follows:

```
test -option filename_or_directoryname
```

This is how you check, with the following example, whether you have writing rights with your current identity (*–w* stands for *w*rite) for the file */etc/fstab*:

```
perle@maxi:~> test -w /etc/fstab
```

Anyone who is now perplexed and wondering where *test* displays the test result, will probably be disappointed by the answer: it doesn't. This built-in is in fact designed to be able to formulate conditions in shell scripts along the lines of "If the writing rights for */etc/fstab* match, then execute the following", and hence one needs no verbal outputs on the standard output. On the other hand, the return value of the command is interesting.

This is saved for the last respective command in the **Variable** $*?*. If it contains the value *0*, then the test was successful; so it is possible to modify the content of the file. But if the command writes *echo $?*, with which one outputs the content (*$*) of the variable *?* on the command line, a value not equal to *0* on the screen, then the test was not successful; ergo there are no writing rights for the corresponding file. This sounds much more complicated than it is though, just look at the two steps on the command line:

```
perle@maxi:~> test -w /etc/fstab
perle@maxi:~> echo $?
1
```

The output *1* is quite definitely not equal to *0*; which means that the user making the query has no write privileges for the file with the *F*ilesystem *Tab*le.

Further useful options of *test* are:

- -*d*: Is the argument file a directory ("*d*irectory")?
- -*e*: Does the file even *E*xist?
- -*r*: Can I *r*ead it?

Queries with *test* are, by the way, always processed in the following sequence: if you are the owner of the file, *test* checks the rights of the file owner. For non-owners on the other hand the built-in first evaluates the group rights, then the rights for the so-called "rest of the world". If you want to know more about *test*, call up the corresponding documentation with *man test* and/or *info test*.

Who's there?

Q In which file can I find details on who has logged on?

Dr. Linux: Basically, on every Linux system, data is collected on the system start and the log-in activities of the users are kept in various databanks and log files. So there is no general answer to this question. Two commands which are simple to use are *last* and *lastlog*.

Alias A new name or else an abbreviation for a self-defined command. You can find out which alias commands in your system are "pre-defined" if you enter the command "alias" with no further details in the bash.

Hash table: If a command is executed in a bash, then the shell remembers this command in the form of the path specification and saves it in the so-called Hash table. If you start the same program again, the bash reads the path specification from the Hash table, instead of first foraging through the entire search path for the corresponding executable file. This speeds up the execution of the command.

Variable Many functions in the bash (and thus the whole system) are controlled by variables. User-defined and system variables can be modified at any time for the current shell. On the other hand the user has read-only access to the so-called "special variables". The command *set* will tell you which variables are set in the system (in addition to other information such as on shell functions) on the output screen.

Listing 1: Sample outputs from *last*...

```
perle@maxi:~> last
news          tty3                          Mon Jan  7 19:04 - 19:04  (00:00)
web           tty2                          Mon Jan  7 19:04   still logged in
perle         tty5                          Mon Jan  7 18:33 - 18:35  (00:02)
perle         tty1                          Mon Jan  7 11:07   still logged in
trish         pts/7        chekov.linux-mag Mon Jan  7 11:06   still logged in
reboot        system boot  2.4.10-4GB       Mon Jan  7 11:03          (08:00)
[...]
wtmp begins Wed Jan  2 01:23:54 2002
```

You can get your own information from the file */var/log/wtmp*, which stores details of successful log-ins and log-outs by system users. But if you try to look at this file using an editor, you will see nothing but confusing symbols (see Figure 1).

To coax out your details from the file, enter *last* on a command line. You will get a multi-column table, which is relatively simple to understand (Listing 1). In the first column, you will see the user name ("Who is logged in?"). Details of the terminal in use (*tty*, *terminal type*), the host to be used for remote log-ins (*From*), the date and the time ("Logged on since when?") and finally the total duration of the log-in session follow. At the bottom of the table you will find information telling you since when log-ins have been being recorded in this file: Usually the system captures and archives */var/log/wtmp* from time to time, so that the file does not get too big.

Just as simple and clear is the output of *lastlog*. Here all the users the system has recorded are included in the output. From the point of view of security special users generated by the system usually have no home directory and are not even allowed to log in directly. This is the case, for example, with the *games* user. Because he has never logged onto the system as user, in the column headed "Latest log-in" (*Latest*) there is the short but sweet comment ***Never logged in***.***

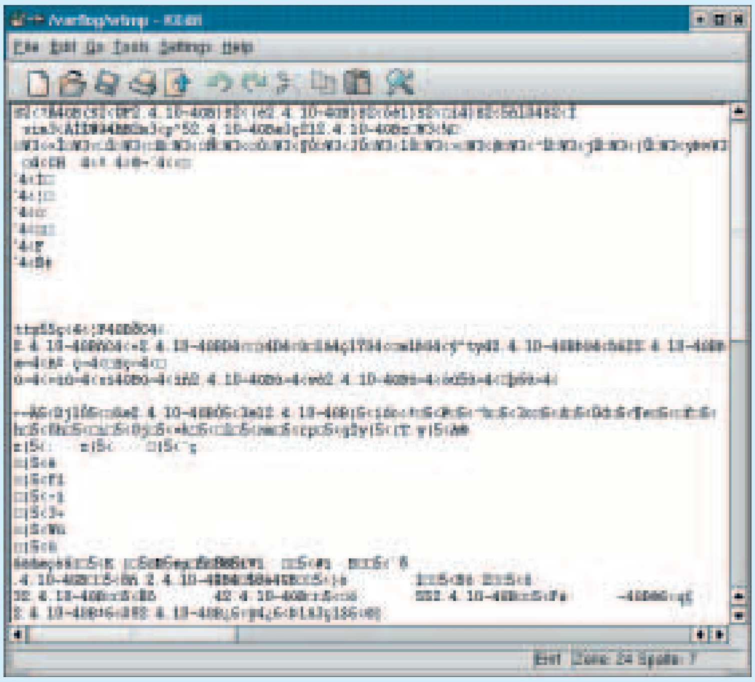

Figure 1: /var/log/wtmp in an editor

Info

Patricia Jung: "Command Line Juggler", Linux Magazine Issue 19. Page 73 provides additional information on shell built-ins and aliases.

Listing 2: ...and *lastlog*

```
root           tty1                       Sun Dec 16 22:06:26 +0100 2001
[...]
lp                                         **Never logged in**
games                                     **Never logged in**
irc                                        **Never logged in**
ftp                                        **Never logged in**
firewall                                  **Never logged in**
postfix                                   **Never logged in**
web            tty2                       Mon Jan  7 19:04:04 +0100 2002
perle          tty5                       Mon Jan  7 18:33:15 +0100 2002
man                                        **Never logged in**
[...]
cz             tty4                       Mon Dec  3 03:13:35 +0100 2001
```