

Bash: shell commands and scripts

A shell (sometimes referred to as a “terminal window”) is an environment to execute operating system commands and programs. Shell scripts are a text files with a line by line listing of commands that could be executed at the command line as well. The script now acts as new command and executes all the commands stated (line by line) in the text file. Unix shell scripts additionally allow to use some control structures. In combination with powerful system and shell commands scripts for very sophisticated tasks are possible.

They are used for e.g. saving time by executing a bunch of often used commands in one step or to perform operations on so many files an individual user couldn't accomplish. Further common tasks performed by shell scripts are interactive or automatic software configuration and starting programs with different parameter settings.

The Bash shell is considered here as probably the most common unix shell.

Language Characteristics:

Shell scripts mostly contain a linear sequence of commands to be performed. Control structures similar to those in other (higher) programming languages exist, but mostly in simpler forms. However, they are sufficient to structure such scripts and to perform more complicated tasks. Such structures are loops (perform one or more operations more than one time), conditionals (performing one or more operations under certain conditions) and functions (grouping commands under by name).

System Characteristics:

Shell scripts are immediately interpreted and executed by the shell, i.e. they don't need to be compiled or pre-processed. All commands used in a shell script, including the control structures, can be executed as single commands at the command prompt as well. Thus, shell scripts are not only operating system dependent, but also shell dependent. Other shells might contain other commands or use a different syntax.

To understand and to use shell scripts, a little Unix knowledge is necessary:

- ? The Unix filesystem differs from DOS/Windows in many aspects. Two most obvious are: There are no device letters (as e.g. the `c:` in `c:\home\mydirectory`) and the backslashes (“\”) are replaced by slashes (“/”). Thus the directory above would be `/home/mydirectoy` in a Unix system.
- ? To get help for the shell commands and other topics the command `man` (for manual) can be used with the command in question as parameter. For instance `man ls` shows a screen with information on the shell command `ls` (see also below).
- ? To each file and each directory are individual file permissions assigned. These permissions determine which kind of user (all users, members of a specific group or only the user owning the file) is allowed to perform which kind of operation (read, write, execute). The current mode of the file permissions can be seen with the command `ls -l` and the mode can be changed (depending on the user's rights regarding the file) with the command `chmod`.

Examples:

```
chmod a+x file    (gives all users the right to execute the file file)
```

```
chmod g-w file    (forbids all group members assigned to the file file to write into it)
```

Details should be checked in the manual pages or other documentations. In this context it is important to ensure that the file containing the shell script should be (at least for the user) executable.

- ? Parameters in Unix commands are often specified by a hyphen (“-“), as e.g. in `ls -l` above. But this can vary from shell to shell and from command to command.
- ? Some important basic commands are:
 - o `ls`: Lists a directory (as `dir` in DOS/Windows). In the form `ls -l` it prints a detailed listing (including file sizes, file dates, file permissions etc.)
 - o `pwd`: Prints the current working directory.
 - o `cd`: Change directory. Basically the same usage as in DOS/Windows.
 - o `rm`: Remove files and directories. `rm -r dirname` deletes the directory `dirname` and all subdirectories it contains. Generally `rm` is similar to the DOS command `del`. Attention! Depending on the system's configuration there might be no security question and no undelete possibility. Files then are really removed!
 - o `mv`: Moves a file (`mv from_dir to_dir`).
Can also be used to rename files (`mv fileold filenew`).
 - o `cp`: Copies a fie (`cp source_file target_dir`). Similar to DOS command `copy`.

Bash: shell commands and scripts

- ? Some helpful command line tools (with simple examples):
- o `grep`: Searches in a (text) file for a given string.
Example: `grep "mystring" myfile`
Explanation: `grep` lists all the lines of `myfile` containing the string `mystring`
 - o `diff`: Prints the different lines of two files or nothing if they are equal.
Example: `diff file1 file2`
 - o `cut`: Cuts one or more columns separated by a tab stops or another delimiter.
Example: `cut -f1,3 filein > fileout`
Explanation: The columns 1 and 3 (separated by tabulators) are cut of from the file `filein` and written to the file `fileout` (instead of printing it on the screen). The output file will either be created or overwritten. All output that would be normally written to the screen (standard output) can be alternatively written to a file with the operator `>`.
 - o `sort`: Sorts a file.
Example: `sort file`
Explanation: Sorts the file `file` in ascending order of the first character in each line and prints the result to the standard output (screen). Other possibilities would be to change the order, to sort by other columns in the file or to suppress all multiple lines.
 - o `cat`: Prints the content of a file to the standard or another output
Examples:
`cat filein > fileout`
`cat filein >> fileout`
Explanation: The first one reads each line of `filein` and copies it to `fileout`. The output file is therefore newly created. In the second line all lines of the first file are appended to the end of the output file. So two files can be concatenated.
- ? Pipelines:
Pipelines are used to connect different commands and to "pipe" the output data of one command as input to the other command. The pipe symbol is `|`. More than one pipe can be used.
Example: `cut -f2 -d " " file | sort`
Explanation: The column 2 from the file `file` is taken, sorted and printed on the screen. Note that the delimiter between the columns of the input file here is the space character (" ").
Example: `grep "codon" dna | sort -u | cut -f1-4,8`
Explanation: Here are first all lines of the file `dna` searched that contain the string "codon". These lines are sorted uniquely (i.e. all repeats are thrown away) by the first column in ascending order. The final results are from the sorted list all columns from 1 to 4 and column number 8. The delimiter here is the default one (tabulator).

Hello-World Example:

```
#!/bin/bash  
echo "Hello World"
```

The second line with the `echo` command (which produces the output) would be enough for this task. But every shell script should contain a line like `#!/bin/bash` as first line. This ensures that even if the script is started from another shell with a different syntax the bash shell will be used for executing the script. The hash symbol (`#`) usually means a comment which has no influence on the execution. But in this case, followed by the "!" it means the assignment to the shell in the path following after the "!". Therefore the bash shell program should be located in the specified path (which is the case for every "usual" Unix system). Also this line in the text file has often the effect that it is stored as executable so that the file mode has not to be changed (see above) for executing the script.

Download:

<http://www.gnu.org/software/bash/bash.html> - Bash shell and documentation download.
<http://www.gnu.org> - Other GNU tools, such as `grep`, `sort` etc. are here available.
<http://www.cygwin.com/> - Free software that simulates a Linux shell on a Windows system.

Literature:

<http://www.gnu.org/manual/bash-2.05a> - Official Bash manual
<http://www.tldp.org/LDP/abs/html> - Advanced Bash-Scripting Guide. Many explained examples.
Newham, C. and Rosenblatt, B. 1998. Learning the bash shell, 2nd Edition. O'Reilly.