Working in the shell

October 23, 2014

Contents

1 A shell primer

In the following we have put together a small and by no ways comprehensive collection of shell commands, such as emphless, chmod, sed, grep, head, tail, tr, sort, comm, uniq, that come in handy for solving our later exercises. If you have already experiences with unix/linux operating systems you may also skip this section or may just have brief look at the introduced functions.

1.1 Data structure

1.1.1 File system

The Linux file system is organized in a hierarchical manner where directories contain either other directories or files. Sometimes this organization is referred to as a 'directory tree'. The top node in this tree is called the *root* and is represented by a '/'. An example is shown in figure 1.

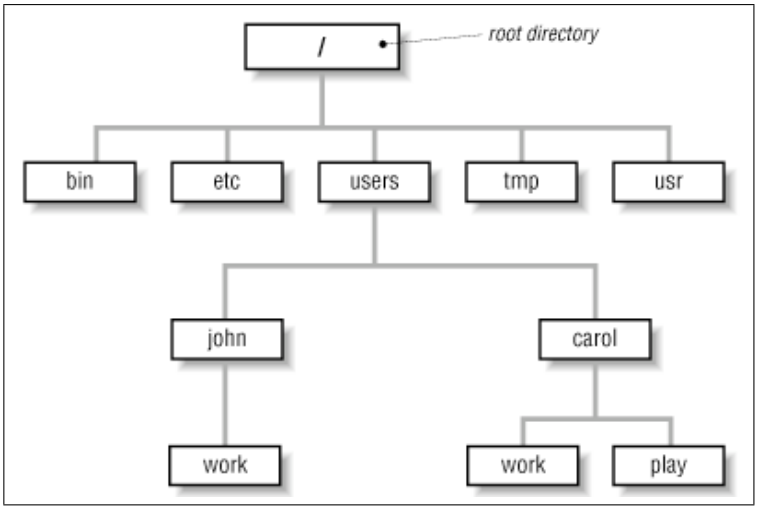

Figure 1: The file structure in Linux

***ls* – listing the contents of a directory** .

Examples:

- *ls* – list the contents of a directory
- *ls -a* – list the contents of a directory including 'hidden files' that start with a '.'
- *ls -l* – list the contents of a directory displaying more information. The output looks like that:

  -rwxr-xr– 1 ingo staff 1377 Dec 7 11:48 exercises-2009.aux

  The individual informations are the following:

  – The first position in the output denotes whether the name represents a file (-) or a directory (d). In the example it is a file.

  – The next nine characters (rwxr−xr−−) define the privacy settings for that file. r denotes read-permission, w denotes write permission (you are allowed to modify the file), and x denotes executable permission. The first three positions represent the settings for the owner of the file. In our case, *rwx* indicates that the owner is allowed to read, change and execute the file. The next block of three gives the

settings for the *group*. In this example, the group has *read* and *write* permissions for the file. The last block of three gives the settings for all *other* users. In the example, every user can only read the file. The privacy settings can be adjusted using the command *chmod*.

  - The next number gives the number of links to that file. In our case 1.
  - The next two strings gives the user and his group to whom this file belongs. In this case 'ingo' in the group 'staff'. These settings can be changed with the command *chown* – change owner.
  - The next position gives the file size in Bytes.
  - The date gives then the time the file has been last modified.
  - The last position gives the file name.

- *ls -ltr* – list the contents of a directory ordering them by the time they have been last modified.

***chmod* – Privacy settings** In principle the file system under linux allows you to see any directory and any file of any user. However, the system allows to adjust the privacy settings for every file and every directory using the comman *chmod*. To see the current privacy settings for a file use *ls -l* (see above). To modify them proceed as following:

- *chmod a+x* make a file executable for *owner*, *group* and *other*.
- *chmod a-x* revoke the executable rights from *owner*, *group* and *other*.
- *chmod u+x* grant execution permissions to *owner*.
- *chmod u-x* revoke execution permissions from *owner*.
- *chmod g+x* grant execution permissions to *group*.
- *chmod g-x* revoke execution permissions from *group*.
- *chmod o+x* grant execution permissions to *other*.
- *chmod o-x* revoke execution permissions from *other*.

Changing read and write permissions works analogous.

***cd* – Moving in the file system** The linux function *cd* (change directory) is used to move around in the file system. To do so efficiently you need to internalize the principle of *paths* in the file system. In figure 1 the path from one directory to the next is given by the edges connecting the directories. Changes between directories can only be done by following the *path* that connects the directory you are currently in with the one you want to change to. The directories along this path are separated by a '/' (please note the difference to the '/' representing the root!) The path is then given as argument to the function *cd*. You need to have *read* permissions for a directory in order to move into it. In the following you will see some examples. There are three short forms that should be remembered:

- . represents the directory you are currently in.
- .. represents the parent directory of the one you are currently in.
- ~ represents your home directory.
- / represents the root directory
- all other directories must be addressed by their name!

Examples:

- *cd .* – changes to the current directory. Thus, nothing happens.
- *cd /* – changes to the root directory
- *cd* – changes to your home directory
- *cd ..* – changes to the parent directory of the one you are currently in
- the following examples now assume that you are in the directory *john* in Figure 1:
  - *cd work* – brings you to the directory *work*
  - *cd ..* – brings you back to the directory *john*
  - *cd ../../* – brings you to the root directory.
  - *cd users/carol/work* – brings you from the root directory to the *work* directory of user *carol*

- *cd ../../john* – brings you from carol's work directory back to john.
- *cd /users/carol/carol/play* – brings you from any place in the tree to carol's *play* directory

Please note the difference between a *relative path* to a directory that depends where in the directory tree you currently reside and an *absolute path* that always starts at the root node.

***pwd* – determine where in the file system you are** When you issue the command *pwd* you will get the absolute path of the directory you are currently in. For example, issuing *pwd* in the directory *john* will result in

/users/john

1.1.2 *mkdir* and *rmdir* – Modifying the directory tree

***mkdir* – creating new directories** Please note, that you need to have write permissions in order to create a sub-directory. Again, the examples assume that you are in */users/john*

Examples:

- *mkdir play* – creates the sub-directory *play* in the directory /users/john
- *mkdir ../carol/guest* – creates the sub-directory *guest* in the directory /users/carol/.
- *mkdir /users/carol/guest* – synonym to *mkdir ../carol/guest* but using the absolute rather than the relative path.
- *rmdir play* – deletes the sub-directory *play* in the directory /users/john. This only works when the directory is empty.

Note that both commands work with relative and absolute paths.

1.1.3 Working with files

Generating files There are many ways to create a file. Either you use your favorite text editor, or you download it from the web or from any other file source. Alternatively, you can generate an empty file using the function *touch* and subsequently edit it.

Examples:

- *touch test.txt* – generates the empty file *test.txt* in the current directory.

***mv* – moving and renaming files** Use the command *mv* to either move a file (or a directory) to a new position in the file system or to rename it. For the examples we have generated a file named *test.txt* in the current directory /users/john/. The absolute path to this file is thus */users/john/test.txt*.

Examples:

- *mv test.txt test_neu.txt* – renames file *test.txt* to *test_neu.txt* without changing the location of the file.
- *mv test_neu.txt work* – moves the file *test_neu.txt* into the sub-directory *work*. Its absolute path is now */users/john/work/test_neu.txt*.
- *mv work/test_neu.txt ./test.txt* – move the file *test_neu.txt* from the sub-directory *work* to the current directory (note, we are still in /users/john) and rename it to *test.txt*.

Note, *mv* works with both relative and absolute paths.

***cp* – copying files** The command *cp* is used to copy files. The copy can have the same name as the original but has then to be in a different directory. Alternatively, you can give the copy a different name. Then the location does not play a role.

Examples:

- *cp test.txt test.txt.cp* – generate a copy of *test.txt* in the same directory and name it *test.txt.cp*.
- *cp test.txt work/test.txt* – generate a copy of *test.txt* in the sub-directory *work*.

***ln* – linking directories or files** The command *ln* is used to link individual files or entire directories rather than copying them. A link can be interpreted as a shortcut in the path leading to a file. If the option *-s* is chosen, a soft link will be generated that just points to the location of the original file or directory. This helps to safe disk space especially in the case of large files. Be careful, if you modify a soft-linked file, you will modify the original!

Examples:

- *ln -s /usr/local/bin ./global_bin*

***rm* – deleting files** The command *rm* is used to delete files from the file system. Be careful, there is no way to undo an accidental deletion of a file!

Examples:

- *rm test.txt* – removes the file *test.txt*
- *rm* $*$ – removes all files in a directory. Be careful!
- *rm -rf work* – removes the directory *work* together with all its contents. Be even more careful!!

***wc* – getting information about files** Use *wc* (word count) to count lines, words and characters in a file.

Examples:

- *wc test.txt* – returns the number of lines, words and characters in the file *test.txt*.
- *wc -l test.txt* – returns just the number of lines in the file *test.txt*. Use *-w* and *-c* to count only words and characters, respectively.

***less* – Looking into files** With *less* you can display the contents of a file directly in your shell. Use the arrow keys or the tab key to navigate up and down in the file. Type *q* to return to the shell.

***head* – Display the first *n* lines of a file** **Examples:**

- *head test.txt* – displays the first 10 lines (default) of the file *test.txt*. If the the file has less than 10 lines, all lines will be displayed.
- *head -n 15 test.txt* – displays the first 15 lines of the file *test.txt*.

***tail* – Display the last *n* lines of a file** **Examples:**

- *tail test.txt* – displays the last 10 lines (default) of the file *test.txt*. If the the file has less than 10 lines, all lines will be displayed.
- *tail -n 15 test.txt* – displays the last 15 lines of the file *test.txt*.

***grep* – pattern search in files** Using the function *grep* you can search for patterns in a file and extract the matching lines. The general syntax for the search is *grep 'searchpattern' file_to_search_in*.

Examples:

- *grep 'and' test.txt* – extract all lines in the file *test.txt* that contain the pattern 'and' either as a word as in "*patterns* **and** *expressions*" or within a word as in "*r***and***om*".
- *grep '^and' test.txt* – extract all lines in the file *test.txt* that start with the string 'and'.
- *grep 'and$' test.txt* – extract all lines in the file *test.txt* that end with the string 'and'.

Note, *grep* is case sensitive. If you want it to ignore the case then add the option *-i* to the function call. *grep* finds it complete expression when it is used with *regular expressions*.

***sed* – editing a file's content** Use *sed* if you want to change the contents in you file, *e.g.*, by removing a certain character from all lines. *sed* is an extremely powerful but unfortunately also pretty cryptic way to alter file contents. Therefore, I just list some simple example applications that can help a lot in certain times. **Examples:**

- *sed -e 's/>//' test.txt* – removes the first occurrence of the character '>' in all lines of the file *test.txt* and prints the modified output to the screen. The file *test.txt* itself remains unaltered.
- *sed -i -e 's/>//' test.txt* – removes the first occurrence of the character '>' in all lines of the file *test.txt* and alters the file. Watch out, there is no undo option!
- *sed -i -e 's/>//g' test.txt* – removes all occurrences of the character '>' in all l ines of the file *test.txt* and alters the file. Watch out, there is no undo option!
- *sed -i -e 's/>/</' test.txt* – changes the first occurrence of the character '>' into a '<' in all lines of the file *test.txt* and alters the file. Watch out, there is no undo option!

Note, by adjusting -e 's/*search_pattern*/*replacement_pattern*/' to your needs, you can make easily use of *sed*. Similar to *grep*, also *sed* finds its complete expression when used with *regular expressions*.

1.1.4 Loops

For a full documentation on how to write loops in a bash script, please refer to one of the many online documentations on shell scripting. Here I will just give a simple example of the basic syntax.

Examples:

- for i in ‘ls‘

  do

  echo ”item: $i”

  done

Obviously, this loop iterates over all return values from the *ls* command, i.e., over all items located in the directory you are currently in. The name of each item will be written into the variable $i and will subsequently printed on the screen using the command *echo*. By replacing ‘ls‘ with, e.g., *grep '>'* you can loop through each line of output that *grep* returns.

1.1.5 Regular expressions

An overview of regular expressions can be found, e.g., here

http://www.regular-expressions.info/

1.1.6 Calculation in a shell

Floating point calculations in the bash can be done by passing the arguments to the function *bf*, and by determining the precision using *scale*.

Examples:

- Division: echo ”scale=3; 7 / 4” |bc
- Multiplication: echo ”scale=3; 3.3 * 5” |bc

1.1.7 Redirecting the output of functions

Saving the output of a function into a file Many unix function produce output that is directly printed onto the screen. *ls* for example lists the directory contents and prints the output on the screen. Saving the output in a file instead is trivial. Just place a > filname after the command.

Examples:

- *ls >dircontent.txt* – saves the output of *ls* into the file dircontent.txt. If the file already exists, its content will be overwritten.

- *ls >>dircontent.txt* – saves the output of *ls* into the file dircontent.txt. If the file already exists, the output will be appended to its content.

Using the output of one command as input for another program To use the output of one function as input for a second function, just connect the functions with a '|' (pipe).

Examples:

- *ls | wc -l* – The output of the *ls* command serves as input for the *wc* (word count) function. Thus, *ls |wc -l* counts the number of files and sub-directories in the current directory.

- *grep '>' test.txt | wc -l* – *grep* selects all lines from the file *test.txt* that contain a > symbol and the output serves as input for the *wc* function. Thus, *grep '>' test.txt | wc -l* counts the number of lines in *test.txt* that contain a > symbol.

Note, using the '|' symbol you can concatenate many different functions and there is virtually no limit. If you eventually want to save the output to a file, just add the '> *filename*' after the last function call.

1.1.8 Final remarks

The number of different functions that are available in a shell is hard to guess. Most of the times you stumble over one that can help you to complete a particular task more or less by chance. If you don't know how to use a certain function, try typing *man* followed by the function name.