Unix Fundamentals – Basic Shell Commands

Marek Kozłowski
Faculty of Mathematics and Information Sciences
Warsaw University of Technology

Never use copy&paste for the following exercises. Retype all commands manually!
Don't just read those exercises and examples. Do them, please!

1. **Manual**

   (a) Display the man page on '`man`':
   `$ man man`
   Are the man pages grouped into sections specified on the slide #6? (see: DESCRIPTION)
   Does this man follow the scheme from the slide #5?

   (b) Check man pages on commands used during previous labs:
   `$ man ls`
   `$ man sort`
   `$ man touch`
   `$ man rm`
   `$ man cd`
   `$ man pwd`
   Yes, all those man pages follow the same scheme.

   (c) As you remember the '`echo`' command is implemented twice. The man page:
   `$ man echo`
   contains information on the program. Remember that bash built-ins take precedence over external programs. Run:
   `$ man bash`
   and search (use searching!) for help the '`echo`' built-in.

   (d) Display the man page on '`/etc/passwd`':
   `$ man 5 passwd`
   (remember: specify the section and omit the '`/etc/`' prefix!). What happens if you don't specify a man section?

   (e) Are you unsure what is the C '`printf()`' syntax? Don't worry, there are man pages on all standard C functions:
   `$ man 3 printf`
   (remember: a section number and no parenthesis!)

   (f) Whenever possible you should use POSIX manuals (1P, 3P). If unsure – search all man pages:
   `$ man -a kill`

2. **Session utilities**

   (a) Do you need a FLOSS SSH client for Windows? Visit: https://putty.org . Fortunately under Linux and Unix you have a client.

   (b) Start a remote shell session with the '*ssh*' server:
   `$ ssh your_name@ssh.mini.pw.edu.pl`
   Enter '*yes*' when prompted (only during the first connection with this host name).
   You have a local interface (terminal) but you run all commands on a remote computer.

You may connect to this host from home for practicing.
Finally terminate your ssh session by executing:
`$ exit`
or just closing the terminal (it kills all processes running in it).

(c) As your local user name and domain suffix are the default values, you may safely omit them when connecting to our server from our labs:
`$ ssh ssh`
(the first '*ssh*' is a command, the second one – a host name; yes, it is confusing!)
Enter '*yes*' when prompted (the host is the same but its name you've specified is different).
Terminate your ssh session by executing:
`$ exit`
or just closing the terminal.

(d) Open a terminal. Start some task in it:
`$ top`
(it is some kind of a task manager; we can see it is actively running)
When you close/detach a terminal processes in it are immediately killed.

(e) Open a terminal. Start a protected session:
`$ screen`
then run '`top`' inside it:
`$ top`
You may close the terminal or detach the screen session by pressing '`Ctrl-a d`' – processes in it remain active. Open a different terminal and reattach the screen session:
`$ screen -r`
Yes, we can see the process in it is still running. This works for both local and remotely connected (*ssh*) terminals.
Quit '`top`' by pressing '`q`', exit the screen:
`$ exit`

(f) Start a new terminal typescript:
`$ script`
Run some commands, for example:
`$ ls /etc`
`$ date`
End the script:
`$ exit`
Open a new terminal emulator window (you'll see no difference otherwise) and run:
`$ cat typescript`
Note that typescripts may contain non-ASCII characters for control sequences, colors, etc.

3. **User and system information**

(a) What time is it?
`$ date`
Check the man page for other date and/or time formats (unfortunately, those are not standardized).

(b) What are the dates of our next five labs?
`$ cal -3`

(c) Check if your computer runs Linux:
`$ uname`
Yes, it does. Get all available information:
`$ uname -a`
Check the man page for other available options.

(d) Under Linux you can examine your hardware with those commands:
`$ lscpu`

```
$ lspci
$ lsusb
```
You may use the '`-v`' option for increased verbosity of the last two ones (are you sure it's more clear to you? ;-)).

(e) Check memory usage in human-readable units:
```
$ free -h
```

(f) Print your identity and group information:
```
$ id
```
Only a user name:
```
$ id -un
```
... and only group names:
```
$ id -Gn
```

(g) Connect to *ssh*:
```
$ ssh ssh
```
and check who is logged in:
```
$ w
$ who
```

4. **File viewing and editing**
File viewers can be used as pipeline filters so there are more exercises on them planned for the next labs. What we have below is just an introduction.

(a) '`cat`' can be used for displaying very short files. During previous labs we checked shells installed on our workstations. There is a short file which contains a list of all valid login shells. Let's display it:
```
$ cat /etc/shells
```

(b) '`echo`' allows checking what our shell *sees* when reading a command. Use it if you are unsure; for example:
```
$ echo "\"\\'\$"
```
It is also quite often used for examining variables:
```
$ echo $PATH
```

(c) '`more`' is a very simple viewer that allows paging. Use it for files containing a few dozens lines:
```
$ more /etc/passwd
```

(d) '`less`' allows real navigation instead of just paging – practice searching forward and backward. Let's take a long file for this purpose (the file used in this exercise is not an important one to us but it is long):
```
$ less /etc/services
```
Note: Characters other that letters may require escaping. We'll discuss this more thoroughly the next week.

(e) Most Unix systems use '`less`' for displaying man pages. Instead of exercises on '`less`' you may as well practice on any man page.

(f) '`head`' and '`tail`' simple examples:
```
$ head /etc/services
$ head -n5 /etc/services
$ tail /etc/services
```

(g) Practice using '`nano`'. As it is a part of the GNU project most Linux/GNU distros provide it as the default text editor.

5. **File management**

(a) Switching between two directories:
```
$ cd /etc; pwd
```
Repeat the following command several times:

```
$ cd -; pwd
```
Finally let's go home:
```
$ cd
```

(b) Instead of using the '`pwd`' command you can examine the '`PWD`' variable:
```
$ pwd
$ echo $PWD
```
Working directory change results in introducing an additional variable:
```
$ echo $OLDPWD
```

(c) How many files are created?
```
$ touch a b c; ls
$ touch "d e f"; ls
```
The same (variable number of file names) applies to numerous commands. Delete those four files:
```
$ rm a b c "d e f"
```
Be aware that '`rm`' doesn't move files to your desktop's trash bin. It removes files permanently!

(d) For '`cp`' and '`mv`' the second argument can be a name or a path. The following two commands do the same:
```
$ cp /etc/passwd passwd
$ cp /etc/passwd .
```
Note that '`cp`' has no default arguments so the following command will not work:
```
$ cp /etc/passwd
```

(e) *Renaming* a file is just moving it to a new name:
```
$ touch fiele5; mv fiele5 file5e
```

(f) Can symlinks point to non existing files? Yes, in case of so-called broken symlinks. In such case shell's messages may be a little bit confusing. Let's check:
```
$ nano file6f
```
(add some content)
```
$ ln -s file6f symlink6f; ls -l
$ cat symlink6f
$ nano symlink6f
```
(modify the content)
```
$ cat file6f
$ rm file6f; ls -l
$ cat symlink6f
```
Finally do some cleanup:
```
$ rm symlink6f
```

(g) Remember: first test '`rsync`' in the *dry-run* mode! Why? A trailing slash (added by auto-completion at the end of a path) can result in different directories used for synchronization (for options explained see the slide #34):
```
$ mkdir A B; touch A/a A/b A/c
$ rsync -avzn --delete A/ B
$ rsync -avzn --delete A/ B/
$ rsync -avzn --delete A B
$ rsync -avzn --delete A B/
```
The difference is similar to the difference between:
```
$ cp -R A/* B/
```
and
```
$ cp -R A B/
```
However keep in mind that '`rsync`' and '`cp`' don't do the same job! The example with '`cp`' is just intended for explaining the difference in specifying the source.
Cleanup: delete both directories:
```
$ rm -r A B
```

(h) If necessary copy the file '`/etc/passwd`' to your home directory (see the exercise 5d. above shall the need arise). Change the local copy: delete some lines, add some lines

and modify some lines. Maximize your terminal emulator window by pressing '`<F11>`' and compare both files:
`$ diff -y passwd /etc/passwd`
In short:
`$ diff -y --suppress-common-lines passwd /etc/passwd`
Less human readable output:
`$ diff passwd /etc/passwd`
can be saved as a *patch file* – what modifications turn the first file into the second one (we'll explain the '`>`' operator used in this exercise more in detail the next week):
`$ diff passwd /etc/passwd > file.patch`
Patches can be then applied:
`$ patch passwd file.patch; diff passwd /etc/passwd`
or reverted:
`$ patch -R passwd file.patch; diff passwd /etc/passwd`
Remember: most updates for FLOSS software are distributed as patch files!
Do some final cleanup:
`$ rm passwd file.patch`

6. **Dealing with archives**

   (a) What happens if we try gzipping a few files?
   `$ touch a b c`
   `$ gzip a b c; ls`
   Let's uncompres them:
   `$ gunzip ?.gz; ls`
   (you remember this wildcard, don't you?)

   (b) Compress files with one character long names like a Unix guru:
   `$ tar -cvzf archive.tar.gz ?`
   Remember: Since '`-f`' is an option with a value (archive name) it must be the last option in a group and directly precede the value (the archive name).
   You may now safely remove files added to an archive:
   `$ ls`
   `$ rm ? ; ls`
   Now extract them:
   `$ tar -xvzf archive.tar.gz`
   and delete an unneeded archive:
   `$ rm archive.tar.gz`

   (c) Repeat above examples with *bzip2* compression instead of *gzip* (refer to the slide #40).

   (d) Repeat above examples with *xz* compression instead of *gzip* (refer to the slide #40).

   (e) Remove files used in this exercise:
   `$ ls`
   `$ rm ? ; ls`

7. **File finding**

   (a) Find in '`/etc`' all files that end with '`.conf`' modified more than 60 days ago:
   `$ find /etc -name "*.conf" -mtime +60 2>/dev/null`
   Use quotation marks for wildcards. Those are intended for '`find`' rather than a shell.
   The suffix: '`2>/dev/null`' eliminates *Permission denied* error messages (generally: any error messages of any commands). We'll explain it during the next labs.

   (b) Update timestamps of all files modified the last week:
   `$ find ~ -mtime 7 -exec touch {} \;`

   (c) Create a directory in your home:
   `$ mkdir Somedir`
   Find all regular files in '`/etc`' of the size more than 100kB modified less than 60 days ago and copy them to this directory:

```
$ find /etc/ -size +100k -type f -mtime -60 -exec cp {} Somedir \; 2>/dev/null
$ ls Somedir
```

Remove the directory:

```
$ rm -rf Somedir
```