

The bare minum to know for a shell user

Author: Daniele Pizzolli
Contact: ors@tovel.it
Copyright: 2010 Some Rights Reserved <http://creativecommons.org/licenses/by-sa/3.0/>
Location: Villa Sant'Ignazio — Trento
Version: Draft
Note: Please send your feedback.

write and wall

```
write username  
some text  
^D  
  
wall < message_file  
  
mesg off  
mesg on
```

```
write (1)          - send a message to another user  
wall (1)           - write a message to users  
mesg (1)           - control write access to your terminal
```

File System Commands

```
cd                Change the shell working directory.  
pwd (1)           - print name of current/working directory  
mkdir (1)         - make directories  
rmdir (1)         - remove empty directories  
ls (1)            - list directory contents
```

User and Permissions Commands

```
adduser (8)       - add a user or group to the system  
addgroup (8)     - add a user or group to the system  
usermod (8)      - modify a user account  
chown (1)        - change file owner and group  
chmod (1)        - change file mode bits  
chattr (1)       - change file attributes on a Linux file system  
stat (1)         - display file or file system status
```

More on filesystem

```
rm (1)          - remove files or directories
cp (1)          - copy files and directories
mv (1)          - move (rename) files
ln (1)          - make links between files
more (1)        - file perusal filter for crt viewing
less (1)        - opposite of more
df (1)          - report file system disk space usage
du (1)          - estimate file space usage
```

Homework

1. Read more about Environment Variables from “The Art of Unix Programming”:

<http://www.catb.org/~esr/writings/taoup/html/ch10s04.html>

2. Create a directory. Create some files inside the directory using various methods:

1. Using `echo` and output redirection
2. Using `cat` and output redirection
3. Using `cat` and output redirection to emulate `cp`.

4. Using a CLI editor (try one from the following list: `nano`, `mcedit`, `vi(m)`) or search yourself:

```
debtags search 'works-with::text && use::editing && interface::text-mode && ! interface::x11'
```

3. Create a file with the name of the current logged user.
4. Create a file with the name of the current logged user that contains the current working directory.
5. (Optional) Read, for fun and profit, The chapter 2 of “The UNIX-HATERS Handbook”: “Welcome, New User!” and try yourself the command line examples.

BIG WARNING: create a new user, and login using this newly created user if you want to try commands like `rm` and keep your data safe.

You can download a copy form:

<http://www.cs.washington.edu/homes/weise/uhh-download.html>

6. (Optional) Read more about Command-Line Options from “The Art of Unix Programming”:

<http://www.catb.org/~esr/writings/taoup/html/ch10s05.html>

Process Commands

```
ps (1)          - report a snapshot of the current processes.
nice (1)        - run a program with modified scheduling priority
renice (1)      - alter priority of running processes
kill (1)        - send a signal to a process
jobs            Display status of jobs.
fg             Move job to the foreground.
bg             Move jobs to the background.
&             nothing appropriate.
^Z            nothing appropriate.
```

Shell Variables

```
KDE_MULTIHEAD=false
SSH_AGENT_PID=1909
DM_CONTROL=/var/run/xdmctl
GPG_AGENT_INFO=/tmp/gpg-u5tvYF/S.gpg-agent:1910:1
DESKTOP_STARTUP_ID=
SHELL=/bin/bash
TERM=xterm
XDM_MANAGED=method=classic
XDG_SESSION_COOKIE=096476d61561447b4e1belca4b099918-1269286508.699873-4308932...
MAKEFLAGS=
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/ors/.gtkrc-2.0:/home/ors/.gtkrc-2.0-kd...
KONSOLE_DBUS_SERVICE=:1.59
GTK_RC_FILES=/etc/gtk/gtkrc:/home/ors/.gtkrc::/home/ors/.kde/share/config/gtk...
GS_LIB=/home/ors/.fonts
WINDOWID=54525977
KDE_FULL_SESSION=true
USER=ors
LS_COLORS=rs=0:di=01;34:ln=01;36:hl=44;37:pi=40;33:so=01;35:do=01;35:bd=40;33...
SSH_AUTH_SOCK=/tmp/ssh-MmaByk1867/agent.1867
SESSION_MANAGER=local/aaa:@/tmp/.ICE-unix/1999,unix/aaa:/tmp/.ICE-unix/1999
MAKELEVEL=1
MFLAGS=
DESKTOP_SESSION=default
PATH=/home/ors/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b...
_=/usr/bin/env
PWD=/home/ors/git/linux_cmdline/2010/presentation
LANG=it_IT.UTF-8
KDE_SESSION_UID=1000
KONSOLE_DBUS_SESSION=/Sessions/11
HOME=/home/ors
SHLVL=3
COLORFGBG=15;0
KDE_SESSION_VERSION=4
LANGUAGE=it_IT:it:en_GB:en
XCURSOR_THEME=oxy-white
LOGNAME=ors
XDG_DATA_DIRS=/usr/share:/usr/share:/usr/local/share
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-26XBpvAQqy,guid=abf9ba3b6aed...
LESSOPEN=| /usr/bin/lesspipe %s
WINDOWPATH=7
PROFILEHOME=
DISPLAY=:0.0
QT_PLUGIN_PATH=/home/ors/.kde/lib/kde4/plugins:/usr/lib/kde4/plugins/
LESSCLOSE=/usr/bin/lesspipe %s %s
```

Various

TODO:

- caratteri speciali
- escape
- ed elenco in bash
- pipes
- wildcards

- `${$,0,*,#,.?!,_,@}`
- heredoc
- expr
- exit status
- test

Shell Grammar

Simple Commands

A simple **command** is a sequence of optional variable assignments followed by a pipeline.

A pipeline is a sequence of one or more commands separated by one of the following operators: `&&`, `&`, `|`, `&|`, `&&|`.

A list is a sequence of one or more pipelines separated by one of the following operators: `&&`, `&`, `|`, `&|`, `&&|`.

A compound **command** is one of the following:

Coprocesses

A coprocess is a shell **command** preceded by the `coproc` reserved word. A shell function definition is a shell **command** preceded by the `function` reserved word.

Shell Function Definitions

A shell **function** is an object that is called like a simple **command** and

Shelly syntax: built-in I/O

- read
- echo
- printf

```
printf "Insert two var: ";
read VAR1 VAR2;
printf "Printing two var: '%s' '%s'\n" "${VAR1}" "${VAR2}";
echo "Don't use echo because behaves differently on different implementation"
```

Try yourself:

- Insert only one string
- Insert more than two strings
- Insert the following line: 'test test' "prova prova" tell which is which...

Shell syntax: if

if elif then else:

```
if true;
then
    print "OK";
elif false;
then
    print "KO";
else
    print "Never Here";
fi
```

Shell syntax: case

```
case "${1}" in
  start)
    do_start
    ;;
  restart|reload|force-reload)
    echo "Error: argument '$1' not supported" >&2
    exit 3
    ;;
  stop)
    # No-op
    ;;
  \*)
    echo "Usage: $0 start|stop" >&2
    exit 3
    ;;
esac
```

Shell syntax: while

Infinite loop:

```
while true;
do
  date +%s;
  sleep 1;
done;
```

More useful example:

```
touch /tmp/touch
# Dont' ask too much about the following line, explanation later
(sleep 10; rm /tmp/touch) &
# Poll for file deletion
while [ -f "/tmp/touch" ];
do
  printf "Waiting for file deletion"
  sleep 1;
done;
printf "File /tmp/touch deleted"
```

Shell syntax: until

```
i="10";
until [ "5" -gt "${i}" ];
do
  echo "Loop number: ${i}";
  i=$((i-1));
done
```

Questions:

- What is the value of "\${i}"?
- What is the last printed number?

Shell syntax: for

```
for VAR in "1 2 3 4";
do
    printf "%i\n" ${VAR}
done;
```

Other commands

```
unset      Unset values and attributes of shell variables and functions.
type      Display information about command type.
command   Execute a simple command or display information about commands.
shopt     Set and unset shell options.
alias     Define or display aliases.
unalias   Remove each NAME from the list of defined aliases.
dirs      Display directory stack.
pushd    Add directories to stack.
popd     Remove directories from stack.
$DIRSTACK
history (3readline) - GNU History Library
true (1) - do nothing, successfully
false (1) - do nothing, unsuccessfully
```

Text Command

```
cat (1) - concatenate files and print on the standard output
tac (1) - concatenate and print files in reverse
rev (1) - reverse lines of a file or files
head (1) - output the first part of files
HEAD (1p) - Simple command line user agent
tail (1) - output the last part of files
cut (1) - remove sections from each line of files
expand (1) - convert tabs to spaces
unexpand (1) - convert spaces to tabs
uniq (1) - report or omit repeated lines
sort (1) - sort lines of text files
wc (1) - print newline, word, and byte counts for each file
tr (1) - translate or delete characters
cmp (1) - compare two files byte by byte
diff (1) - compare files line by line
patch (1) - apply a diff file to an original
dos2unix TODO
```

File System Related Commands

```
find (1)          - search for files in a directory hierarchy
xargs (1)         - build and execute command lines from standard input
locate (1)        - find files by name
updatedb (8)      - update a database for mlocate
file (1)          - determine file type
slocate TODO
```

Compression and archives

```
gzip (1)          - compress or expand files
gunzip (1)        - compress or expand files
bzip2 (1)         - a block-sorting file compressor, v1.0.4
tar (1)           - The GNU version of the tar archiving utility
tar (5)           - format of tape archive files
```

```
debtags search 'interface::commandline && use::compressing' | wc -l
67
```

Time and date

```
time (1)          - run programs and summarize system resource usage
time (7)          - overview of time and timers
date (1)          - print or set the system date and time
touch (1)         - change file timestamps
```

Unix Utilities

TODO:

```
bin/unix_utilities.sh
```

Update alternatives

```
$ update-alternatives --get-selection
$ update-alternatives --list editor
$ update-alternatives --set editor /usr/bin/vim.tiny
```

(lr)regular ex(cept|press)ion

The regular expression concept is pretty broad, and usually you have to specify what type of you are talking about.

- glob(s)
- regular expression(s)
- (extended) regular expression
- perl-compatible regular expression

There is (a lot) to say here.

The notable book on the subject is [Mastering Regular Expression](#) by Jeffrey ? that tell a lot of this particular *programming* language and show useful comparison between various implementation, that are different...

You could also use `txt2regex` both to learn regular exception between different regular expression implementation and the advanced (perverted?) use of the bash shell. Note that `txt2regex` does not implement all possible regex.

A Little Insight Into Shell Scripting: Function Definition

```
function_name() {
localvar="${1}"
command1 "${localvar}";
command2;
return 0;
}

function_name argument1 argument2
```

Homework

1. Read the "Capitolo 19. Introduzione a Unix e ai sistemi GNU in particolare" http://a2.pluto.it/introduzione_a_unix_e_ai_sistemi_gnu_in_particolare.htm You should be familiar with most of the concepts.
2. Find a (simple) script from <http://www.commandlinefu.com/>. Try to learn how it works. You will explain the script to yours colleagues during the next lesson