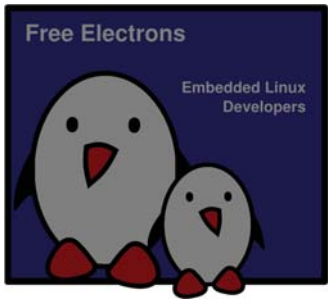
The Unix and GNU/Linux command line

The Unix and GNU/Linux command line



Michael Opdenacker
Thomas Petazzoni
Free Electrons

Latest update: 9/5/2017,
Document sources, updates and translations:
http://free-electrons.com/docs/command-line
Corrections, suggestions, contributions and translations are welcome!

Command memento sheet

It is a useful companion to this presentation.

Examples for the most useful commands are given in just one sheet.

Suggestions for use

Stick this sheet on your wall, use it as desktop wallpaper, make it a mouse mat, print it on clothing, slice it into bookmarks...

Caution

Store away from mice!

Get it on
http://free-electrons.com/doc/legacy/command-line/command_memento.pdf

Training Contents (1)

Shells, filesystem and file handling

Training contents (2)

Standard I/O, redirections, pipes

Training Contents (3)

Task control

Training contents (4)

Miscellaneous

The Unix and GNU / Linux command line

Unix filesystem

Everything is a file

Almost everything in Unix is a file!

- Regular files
- Directories
  Directories are just files listing a set of files
- Symbolic links
  Files referring to the name of another file
- Devices and peripherals
  Read and write from devices as with regular files
- Pipes
  Used to cascade programs
  cat *.log | grep error
- Sockets
  Inter process communication

The Unix and GNU / Linux command line

Shells and file handling

ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the "." character.

- ls -a (all)
  Lists all the files (including .* files)
- ls -l (long)
  Long listing (type, date, size, owner, permissions)
- ls -t (time)
  Lists the most recent files first
- ls -S (size)
  Lists the biggest files first
- ls -r (reverse)
  Reverses the sort order
- ls -ltr (options can be combined)
  Long listing, most recent files at the end

Special directories (1)

./

- The current directory. Useful for commands taking a directory argument. Also sometimes useful to run commands in the current directory (see later).
- So ./readme.txt and readme.txt are equivalent.

../

- The parent (enclosing) directory. Always belongs to the . directory (see ls -a). Only reference to the parent directory.
- Typical usage:
  cd ..

Special directories (2)

~/

- Not a special directory indeed. Shells just substitute it by the home directory of the current user.
- Cannot be used in most programs, as it is not a real directory.

~sydney/

- Similarly, substituted by shells by the home directory of the sydney user.

The cd and pwd commands

- cd <dir>
  Changes the current directory to <dir>.
- cd -
  Gets back to the previous current directory.
- pwd
  Displays the current directory ("working directory").
- Pushd
  Pushes current directory on stack and cd's to new directory.
- popd

The cp command

- cp <source_file> <target_file>
  Copies the source file to the target.
- cp file1 file2 file3 ... dir
  Copies the files to the target directory (last argument).
- cp -i (interactive)
  Asks for user confirmation if the target file already exists
- cp -r <source_dir> <target_dir> (recursive)
  Copies the whole directory.

mv and rm commands

- mv <old_name> <new_name> (move)
  Renames the given file or directory.
- mv -i (interactive)
  If the new file already exits, asks for user confirm
- rm file1 file2 file3 ... (remove)
  Removes the given files.
- rm -i (interactive)
  Always ask for user confirm.
- rm -r dir1 dir2 dir3 (recursive)
  Removes the given directories with all their contents.

Creating and removing directories

- mkdir dir1 dir2 dir3 ... (make dir)
  Creates directories with the given names.
- rmdir dir1 dir2 dir3 ... (remove dir)
  Removes the given directories
  Safe: only works when directories and empty.
  Alternative: rm -r (doesn't need empty directories).

Displaying file contents

Several ways of displaying the contents of files.

- cat file1 file2 file3 ... (concatenate)
  Concatenates and outputs the contents of the given files.
- more file1 file2 file3 ...
  After each page, asks the user to hit a key to continue.
  Can also jump to the first occurrence of a keyword (/ command).
- less file1 file2 file3 ...
  Does more than more with less.
  Doesn't read the whole file before starting.
  Supports backward movement in the file (? command).

The head and tail commands

- head [-<n>] <file>
  Displays the first <n> lines (or 10 by default) of the given file.
  Doesn't have to open the whole file to do this!
- tail [-<n>] <file>
  Displays the last <n> lines (or 10 by default) of the given file.
  No need to load the whole file in RAM! Very useful for huge files.
- tail -f <file> (follow)
  Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
  Very useful to follow the changes in a log file, for example.
- Examples
  head windows_bugs.txt
  tail -f outlook_vulnerabilities.txt

The grep command

- grep <pattern> <files>

Scans the given files and displays the lines which match the given pattern.

- grep error *.log

Displays all the lines containing error in the *.log files

- grep -i error *.log

Same, but case insensitive

- grep -ri error .

Same, but recursively in all the files in . and its subdirectories

- grep -v info *.log

Outputs all the lines in the files except those containing info.

Symbolic links

A symbolic link is a special file which is just a reference to the name of another one (file or directory):

- Useful to reduce disk usage and complexity when 2 files have the same content.
- Example:

anakin_skywalker_biography -> darth_vador_biography

- How to identify symbolic links:
- ls -l displays -> and the linked file name.
- GNU ls displays links with a different color.

Creating symbolic links

- To create a symbolic link (same order as in cp):

ln -s file_name link_name

- To create a link with to a file in another directory, with the same name:

ln -s ../README.txt

- To create multiple links at once in a given directory:

ln -s file1 file2 file3 ... dir

- To remove a link:

rm link_name

Of course, this doesn't remove the linked file!

The Unix and GNU / Linux command line

Command documentation

Command help

Some Unix commands and most GNU / Linux commands offer at least one help argument:

- -h

(- is mostly used to introduce 1-character options)

- --help

(-- is always used to introduce the corresponding “long” option name, which makes scripts easier to understand)

You also often get a short summary of options when you input an invalid argument.

Manual pages

man <keyword>

Displays one or several manual pages for <keyword>

- man man

Most available manual pages are about Unix commands, but some are also about C functions, headers or data structures, or even about system configuration files!

- man stdio.h
- man fstab (for /etc/fstab)

Manual page files are looked for in the directories specified by the MANPATH environment variable.

The Unix and GNU / Linux command line

Users and permissions

File access rights

Use ls -l to check file access rights

3 types of access rights

- Read access (r)
- Write access (w)
- Execute rights (x)

3 types of access levels

- User (u): for the owner of the file
- Group (g): each file also has a “group” attribute, corresponding to a given list of users
- Others (o): for all other users

Access right constraints

- x is sufficient to execute binaries
Both x and r and required for shell scripts.
- Both r and x permissions needed in practice for directories:
r to list the contents, x to access the contents.
- You can't rename, remove, copy files in a directory if you don't have w access to this directory.
- If you have w access to a directory, you CAN remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without w access to it.

Access rights examples

- -rw-r--r--
Readable and writable for file owner, only readable for others
- -rw-r-----
Readable and writable for file owner, only readable for users belonging to the file group.
- drwx------
Directory only accessible by its owner
- -------r-x
File executable by others but neither by your friends nor by yourself.
Nice protections for a trap...

chmod: changing permissions

- chmod <permissions> <files>
2 formats for permissions:
- Octal format (abc):
a,b,c = r*4+w*2+x (r, w, x: booleans)
Example: chmod 644 <file>
(rw for u, r for g and o)
- Or symbolic format. Easy to understand by examples:
chmod go+r: add read permissions to group and others.
chmod u-w: remove write permissions from user.
chmod a-x: (a: all) remove execute permission from all.

Beware of the dark side of root

- root user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting networking,
changing file ownership, package upgrades...
- Even if you have the root password, your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).
- In a training session, it is acceptable to use root.
In real life, you may not even have access to this account, or put your systems and data at risk if you do.

The Unix and GNU / Linux command line

Standard I/O, redirections, pipes

Standard output

More about command output

- All the commands outputting text on your terminal do it by writing to their *standard output*.
- Standard output can be written (redirected) to a file using the > symbol
- Standard output can be appended to an existing file using the >> symbol

Standard output redirection examples

- ls ~saddam/* > ~gwb/weapons_mass_destruction.txt
- cat obiwan_kenobi.txt > starwars_biographies.txt
  cat han_solo.txt >> starwars_biographies.txt
- echo “README: No such file or directory” > README

Useful way of creating a file without a text editor.
Nice Unix joke too in this case.

Pipes

- Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.
- Examples
- cat *.log | grep -i error | sort
- grep -ri error . | grep -v “ignored” | sort -u \ > serious_errors.log
- cat /home/*/homework.txt | grep mark | more
- This one of the most powerful features in Unix shells!

Standard error

- Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- Standard error can be redirected through 2> or 2>>
- Example:
  cat f1 f2 nofile > newfile 2> errfile
- Note: 1 is the descriptor for standard output, so 1> is equivalent to >.
- Can redirect both standard output and standard error to the same file using &> :
  cat f1 f2 nofile &> wholefile

The Unix and GNU / Linux command line

Task control

Full control on tasks

- Since the beginning, Unix supports true preemptive multitasking.
- Ability to run many tasks in parallel, and abort them even if they corrupt their own state and data.
- Ability to choose which programs you run.
- Ability to choose which input your programs takes, and where their output goes.

Processes

"Everything in Unix is a file
Everything in Unix that is not a file is a process"

Processes

- Instances of a running programs
- Several instances of the same program can run at the same time
- Data associated to processes:

Open files, allocated memory, stack, process id, parent, priority, state...

Running jobs in background

Same usage throughout all the shells

- Useful
- For command line jobs which output can be examined later, especially for time consuming ones.
- To start graphical applications from the command line and then continue with the mouse.
- Starting a task: add & at the end of your line:

```
find_prince_charming --cute --clever --rich &
```


Background job control

- jobs

Returns the list of background jobs from the same shell

```
[1]-  Running ~/bin/find_meaning_of_life --without-god &
[2]+  Running make mistakes &
```

- fg

fg %<n>

Puts the last / nth background job in foreground mode

- Moving the current task in background mode:

[Ctrl] Z
bg

- kill %<n>

Aborts the nth job.

Job control example

```
> jobs
[1]-  Running ~/bin/find_meaning_of_life --without-god &
[2]+  Running make mistakes &

> fg
make mistakes

> [Ctrl] Z
[2]+  Stopped make mistakes

> bg
[2]+ make mistakes &

> kill %1
[1]+  Terminated ~/bin/find_meaning_of_life --without-god
```


Listing all processes

... whatever shell, script or process they are started from

- ps -aux

Lists all the processes running on the system

- ps -aux | grep bart | grep bash

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bart      3039  0.0  0.2   5916  1380 pts/2    S    14:35   0:00 /bin/bash
bart      3134  0.0  0.2   5388  1380 pts/3    S    14:36   0:00 /bin/bash
bart      3190  0.0  0.2   6368  1360 pts/4    S    14:37   0:00 /bin/bash
bart      3416  0.0  0.0      0     0 pts/2    RW   15:07   0:00 [bash]
```

- PID: Process id
- VSZ: Virtual process size (code + data + stack)
- RSS: Process resident size: number of KB currently in RAM
- TTY: Terminal
- STAT: Status: R (Runnable), S (Sleep), W (paging), Z (Zombie)...

Live process activity

- top - Displays most important processes, sorted by cpu percentage (**htop**)

```
top - 15:44:33 up  1:11,  5 users,  load average: 0.98, 0.61, 0.59
Tasks:  81 total,   5 running,  76 sleeping,   0 stopped,   0 zombie
Cpu(s): 92.7% us,  5.3% sy,  0.0% ni,  0.0% id,  1.7% wa,  0.3% hi,  0.0% si
Mem:    515344k total,   512384k used,     2960k free,    20464k buffers
Swap:  1044184k total,        0k used,  1044184k free,   277660k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3809 jdoe      25   0  6256 3932 1312 R 93.8  0.8   0:21.49 bunzip2
 2769 root      16   0  157m  80m  90m R  2.7 16.0   5:21.01 X
 3006 jdoe      15   0 30928  15m  27m S  0.3  3.0   0:22.40 kdeinit
 3008 jdoe      16   0  5624  892 4468 S  0.3  0.2   0:06.59 autorun
 3034 jdoe      15   0 26764  12m  24m S  0.3  2.5   0:12.68 kscd
 3810 jdoe      16   0  2892  916 1620 R  0.3  0.2   0:00.06 top
```

- You can change the sorting order by typing M: Memory usage, P: %CPU, T: Time.
- You can kill a task by typing k and the process id.

Killing processes (1)

- kill <pids>

Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first.
Example:
kill 3039 3134 3190 3416

- kill -9 <pids>

Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck (doesn't answer to kill -1).

- kill -9 -1

Kills all the processes of the current user. -1: means all processes.

Killing processes (2)

- killall [-<signal>] <command>

Kills all the jobs running <command>. Example:
killall bash

- xkill

Lets you kill a graphical application by clicking on it!
Very quick! Convenient when you don't know the application command name.

Quoting (1)

Double (") quotes can be used to prevent the shell from interpreting spaces as argument separators, as well as to prevent file name pattern expansion.

```
> echo "Hello World"
  Hello World
> echo "You are logged as $USER"
  You are logged as bgates
> echo *.log
  find_prince_charming.log cosmetic_buys.log
> echo "*.log"
  *.log
```


Quoting (2)

Single quotes bring a similar functionality, but what is between quotes is never substituted

```
> echo 'You are logged as $USER'
  You are logged as $USER
```

Back quotes (`) can be used to call a command within another

```
> cd /lib/modules/`uname -r`; pwd
  /lib/modules/2.6.9-1.6_FC2
```

Back quotes can be used within double quotes

```
> echo "You are using Linux `uname -r`"
  You are using Linux 2.6.9-1.6_FC2
```


Measuring elapsed time

- time find_expensive_housing --near

```
<...command output...>
real    0m2.304s (actual elapsed time)
user    0m0.449s (CPU time running program code)
sys     0m0.106s (CPU time running system calls)
```

real = user + sys + *waiting*
waiting = I/O waiting time + idle time (running other tasks)

~/.bashrc file

- ~/.bashrc

Shell script read each time a bash shell is started

- You can use this file to define
- Your default environment variables (PATH, EDITOR...).
- Your aliases.
- Your prompt (see the bash manual for details).
- A greeting message.

The Unix and GNU / Linux command line

Miscellaneous
Looking for files

The find command

Better explained by a few examples!

- find . -name “*.pdf”

Lists all the *.pdf files in the current (.) directory or subdirectories. You need the double quotes to prevent the shell from expanding the * character.

- find docs -name "*.pdf" -exec xpdf {} ';'

Finds all the *.pdf files in the docs directory and displays one after the other.

- Many more possibilities available! However, the above 2 examples cover most needs.

The locate command

Much faster regular expression search alternative to find

- locate keys

Lists all the files on your system with keys in their name.

- locate “*.pdf”

Lists all the *.pdf files available on the whole machine

- locate “/home/fridge/*beer*”

Lists all the *beer* files in the given directory (absolute path)

- locate is much faster because it indexes all files in a dedicated database, which is updated on a regular basis.
- find is better to search through recently created files.

The Unix and GNU / Linux command line

Miscellaneous
Various commands

The wget command

Instead of downloading files from your browser, just copy and paste their URL and download them with wget!

wget main features

- http and ftp support
- Can resume interrupted downloads
- Can download entire sites or at least check for bad links
- Very useful in scripts or when no graphics are available (system administration, embedded systems)
- Proxy support (http_proxy and ftp_proxy env. variables)

wget examples

- wget -c \ http://microsoft.com/customers/dogs/winxp4dogs.zip

Continues an interrupted download.

- wget -m http://lwn.net/

Mirrors a site.

- wget -r -np http://www.xml.com/ldd/chapter/book/

Recursively downloads an on-line book for off-line access.
-np: "no-parent". Only follows links in the current directory.

Misc commands (1)

- sleep 60

Waits for 60 seconds
(doesn't consume system resources).

- wc report.txt (word count)

```
438  2115 18302 report.txt
```

Counts the number of lines, words and characters in a file or in standard input.