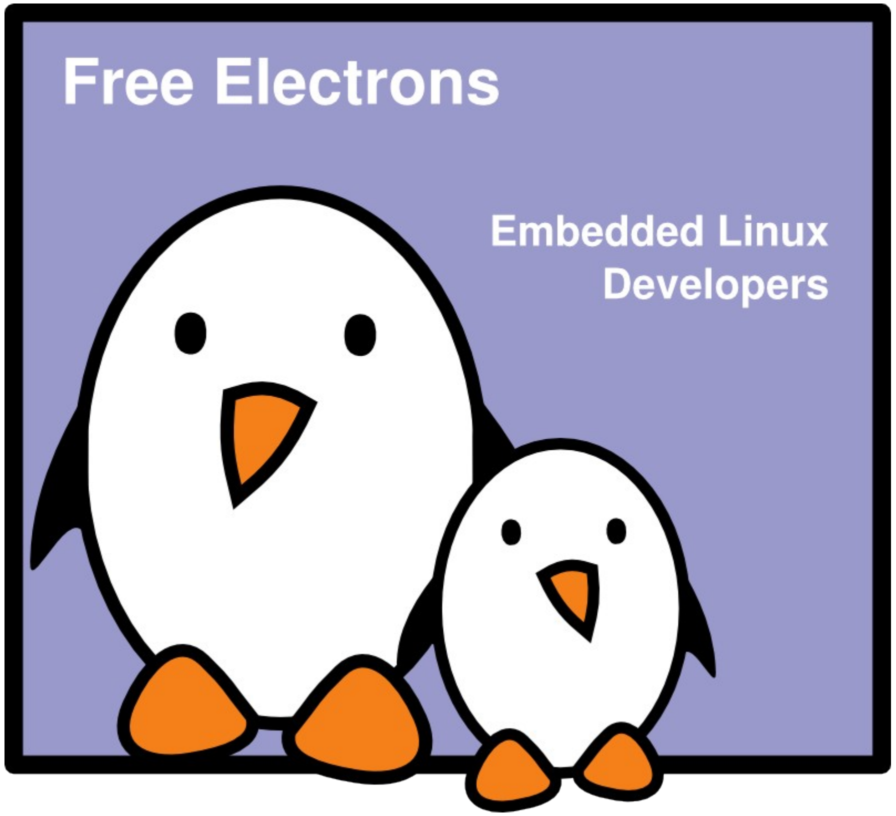
The Unix and GNU/Linux command line

Michael Opdenacker
Thomas Petazzoni
Free Electrons



Abridged for ELE209 Lab 3
By Tim Toolan

© Copyright 2009, Free Electrons.
Creative Commons BY-SA 3.0 license
Latest update: Feb 18, 2013,
Document sources, updates and translations:
http://free-electrons.com/docs/command-line
Corrections, suggestions, contributions and translations are welcome!

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`.

The sort command

- `sort <file>`
  Sorts the lines in the given file in character order and outputs them.
- `sort -r <file>`
  Same, but in reverse order.
- `sort -ru <file>`
  `u`: unique. Same, but just outputs identical lines once.
- More possibilities described later!

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.
- `cat obiwan_kenobi.txt >! starwars_biographies.txt`
  When the file exists, the exclamation point can be used with redirection to indicate that the file should be overwritten.

Standard input

More about command input

- Lots of commands, when not given input arguments, can take their input from *standard input*.

-
  ```
  sort
  windows
  linux
  [Ctrl][D]
  linux
  windows
  ```
  `sort` takes its input from the standard input: in this case, what you type in the terminal (ended by `[Ctrl][D]`)
- `sort < participants.txt`
  The standard input of sort is taken from the given file.

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!

Special devices (1)

Device files with a special behavior or contents

- `/dev/null`
  The data sink! Discards all data written to this file.
  Useful to get rid of unwanted output, typically log information:
  `mplayer black_adder_4th.avi &> /dev/null`
- `/dev/zero`
  Reads from this file always return `\0` characters
  Useful to create a file filled with zeros:
  `dd if=/dev/zero of=disk.img bs=1k count=2048`

See `man null` or `man zero` for details

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 -ux`
  Lists all the processes belonging to the current user
- `ps -aux` (Note: `ps -edf` on System V systems)
  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

```
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.

Miscellaneous

Various commands

Compiling simple applications

- The compiler used for all Linux systems is GCC
  http://gcc.gnu.org
- To compile a single-file application, developed in C :
  `gcc -o test test.c`
  - Will generate a test binary, from the test.c source file
- For C++ :
  `g++ -o test test.cc`
- The -Wall option enables more warnings
- To compile sources files to object files and link the application :

```
gcc -c test1.c
gcc -c test2.c
gcc -o test test1.o test2.o
```

- `gcc` automatically calls the linker `ld`

Remote Shells

- `ssh ele.uri.edu`

  Logon to ele.uri.edu with a remote shell
- `ssh toolan@ele.uri.edu`

  Logon to ele.uri.edu as user `toolan` with a remote shell
- `scp toolan@ele.uri.edu:rfile lfile`

  Copy `rfile` from ele.uri.edu to `lfile` on local system
- `scp -r ldir toolan@ele.uri.edu:/home/rdir`

  Copy local directory `rdir` and all of its contents to remote system ele.uri.edu and put in `/home/rdir`
- `scp` is just like cp, but the source and/or destination can be on a remote system