GNU/Linux

WTPC 2019

GNU/Linux

¿Por qué usar la línea de comandos?

¿Por qué usar la línea de comandos?

Porque no tenemos entorno gráfico

Porque queremos información rápida

Porque no necesitamos información gŕafica

Porque queremos hacer muchas cosas a la vez

Porque nos gusta

Porque la interfaz gráfica la usa cualquiera, y nosotros somos mejores

¿Por qué usar la línea de comandos?

¿Qué hace a linux LINUX?

La “terminal”: herramientas GNU.

Algunas características de la terminal:

- Autocompleta con tab hasta que se repita algo
- Mantiene una historia de comandos usados: con la flecha para arriba los recorren; con ctrl+R pueden buscar comandos anteriores
- Muy fácil cambiar de usuario (no sólo a root, a cualquier otro)
- $man <comando> [mucho mejor que apretar F1 sin querer]
- Podemos redirigir el output y el input

Terminal: find

```
$ find ~/FeF -name “*.pdf”
/home/rlugones/FeF/libros/Machine-Learning-A-Z-Q-A.pdf
/home/rlugones/FeF/charlas/python.pdf
/home/rlugones/FeF/charlas/git.pdf
/home/rlugones/FeF/charlas/workflow.pdf
```

Terminal: ver texto

```
$ cat alumnos.txt
Apellido, Nombre
Lugones, Rodrigo
Dolina, Alejandro
Singer, Paul
```

Terminal: ver texto

```
$ more cuadernillo.tex
\documentclass[10pt]{article}
\usepackage[utf8]{inputenc}
\usepackage[inner=0.5in, outer=1in]{geometry}
\renewcommand{\familydefault}{\sfdefault}
\begin{document}
\section*{Lista de Participantes}
\begin{tabular}{|l | c | c | r|}
--More--(17%)
```

Terminal: ver texto

```
$ less cuadernillo.tex
\documentclass[10pt]{article}
\usepackage[utf8]{inputenc}
\usepackage[inner=0.5in, outer=1in]{geometry}
\renewcommand{\familydefault}{\sfdefault}
\begin{document}
\section*{Lista de Participantes}
\begin{tabular}{|l | c | c | r|}
:
```

Terminal: ver texto

cat: simplemente pone todo el archivo en la terminal

less y more: más complejos, se puede buscar.
less es más potente

Terminal: crear texto

Guardar la lista que sale de find en un archivo

```
$ find ~/FeF -name “*.pdf” > lista_archivos
```

Terminal: crear texto

Guardar la lista que sale de find en un archivo

```
$ find ~/FeF -name “*.pdf” > lista_archivos
$ ls
lista_archivos
```

Terminal: crear texto

Guardar la lista que sale de find en un archivo

```
$ find ~/FeF -name “*.pdf” > lista_archivos
$ ls
lista_archivos
$ cat lista_archivos
/home/rlugones/FeF/libros/Machine-Learning-A-Z-Q-A.pdf
/home/rlugones/FeF/charlas/python.pdf
/home/rlugones/FeF/charlas/git.pdf
/home/rlugones/FeF/charlas/workflow.pdf
```

Terminal: buscar texto

Busquemos en la lista de archivos las charlas

```
$ grep charlas lista_archivos
/home/rlugones/FeF/charlas/python.pdf
/home/rlugones/FeF/charlas/git.pdf
/home/rlugones/FeF/charlas/workflow.pdf
```

Terminal: buscar texto

Busquemos en la lista de archivos las charlas

```
$ grep charlas lista_archivos
/home/rlugones/FeF/charlas/python.pdf
/home/rlugones/FeF/charlas/git.pdf
/home/rlugones/FeF/charlas/workflow.pdf
```

```
$ grep -e <expresion regular>
$ grep -r recursivo: en un directorio
```

Terminal: editar texto

Cambiemos en la lista de archivos los rlugones por palcain

```
$ sed 's/rlugones/palcain/g' lista_archivos
/home/palcain/FeF/libros/Machine-Learning-A-Z-Q-A.pdf
/home/palcain/FeF/charlas/python.pdf
/home/palcain/FeF/charlas/git.pdf
/home/palcain/FeF/charlas/workflow.pdf
```

Terminal: editar texto

Cambiemos en la lista de archivos los rlugones por palcain

```
$ sed 's/rlugones/palcain/g' lista_archivos
/home/palcain/FeF/libros/Machine-Learning-A-Z-Q-A.pdf
/home/palcain/FeF/charlas/python.pdf
/home/palcain/FeF/charlas/git.pdf
/home/palcain/FeF/charlas/workflow.pdf
```

```
$ sed -i modifica el archivo “in place”
```

Claves: pipe

Cambiemos en la lista de archivos los rlugones por palcain

```
$ find ~/FeF -name “*.pdf” > lista_archivos
redirige el standard output a un archivo
```

Claves: pipe

Cambiemos en la lista de archivos los rlugones por palcain

```
$ find ~/FeF -name “*.pdf” > lista_archivos
redirige el standard output a un archivo
```

```
$ ./test.x < archivo_input
redirige el archivo al standard input
```

Claves: pipe

Cambiemos en la lista de archivos los rlugones por palcain

```
$ find ~/FeF -name “*.pdf” > lista_archivos
redirige el standard output a un archivo
```

```
$ ./test.x < archivo_input
redirige el archivo al standard input
```

```
$ find ~/FeF -name “.pdf” | grep linking
la salida de la izq de | es el arg de la der
```

Claves: pipe

Cambiemos en la lista de archivos los rlugones por palcain

```
$ find ~/FeF -name “*.pdf” > lista_archivos
redirige el standard output a un archivo
```

```
$ ./test.x < archivo_input
redirige el archivo al standard input
```

```
$ find ~/FeF -name “.pdf” | grep linking
la salida de la izq de | es el arg de la der
```

```
$ ./un_programa 2>&1 output_y_errores
redirige el standard output y error a un archivo
```

Variables de entorno

Generalmente en mayúscula, se accede con $

```
$ echo $PATH
/home/rlugones/bin
$ export PATH=$PATH:/home/rlugones/proyecto/bin
$ echo $PATH
/home/rlugones/bin:/home/rlugones/proyecto/bin
```

Editando ~/.bashrc pueden setear con export las variables por defecto para su usuario

Bash: lenguaje de programación

Bash Shell Scripting

```
$ cat hello_world
echo 'Hello, world!'
$ bash hello_world
Hello, world!
```

Bash: lenguaje de programación

Bash Shell Scripting

```
$ cat hello_name
echo 'Hello, my name is' $1
$ bash hello_world rodrigo
Hello, my name is rodrigo
```

Bash: lenguaje de programación

Bash Shell Scripting

```
$ cat count
if [ $1 -le 10 ]
then echo $(seq 0 $1)
else echo 'Sorry, I can count upto 10 only'
fi
$ bash count 9
0 1 2 3 4 5 6 7 8 9
$ bash count 31
Sorry, I can count upto 10 only
```

Bash: ejecutando programas

Modificar los permisos con chmod

```
$ ./count
bash: ./count: Permission denied
```

Bash: ejecutando programas

Modificar los permisos con chmod

```
$ ./count
bash: ./count: Permission denied
$ ls -l count
-rw-r--r-- 1 rlugones rlugones ... hello_world
```

Bash: ejecutando programas

Modificar los permisos con chmod

```
$ ./count
bash: ./count: Permission denied
$ ls -l count
-rw-r--r-- 1 rlugones rlugones ... hello_world
$ chmod u+x count
-rwxr--r-- 1 rlugones rlugones ... hello_world
$ ./count 4
0 1 2 3 4
```

Ojo con los directorios! Son siempre “ejecutables”

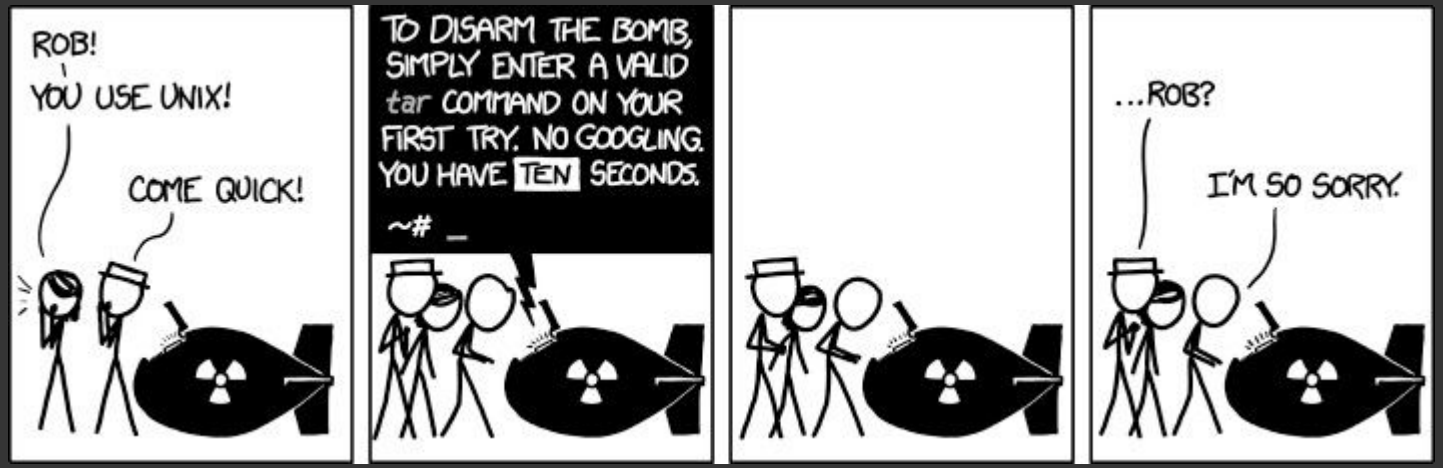
Bash: ejecutando programas

Para ser correctos, el shebang (#!) indica el intérprete

```
$ cat count
#!/bin/bash
if [ $1 -le 10 ]
then echo $(seq 0 $1)
else echo 'Sorry, I can count up to 10 only'
fi
```

Bash: otras herramientas

- ln: crea links (accesos “directos”)
- tar: empaqueta archivos (y comprime)



Ejecutando python con shebang

```
$ cat count.py
#!/usr/bin/env python3
import sys
n = int(sys.argv[1])
if n <= 10:
  for i in range(1, n+1):
    print(i, end=' ')
else:
  print('Sorry, I can count up to 10 only')

$ chmod +x count.py
$ ./count.py 7
1 2 3 4 5 6 7
```

Makefile

Makefiles

Herramienta que controla “compilados” desde source files

Mucho más sofisticado que hacer un pequeño script de bash...
¡Y no tan difícil!

Basado en reglas:

target : dependencies ...
commands
...

Makefiles

Por ejemplo, compilar hello_world.c a hello_world.e

```
$ cat Makefile
hello_world.e: hello_world.c
    gcc hello_world.c -o hello_world.e

$ make
gcc hello_world.c -o hello_world.e

$ make
make: 'hello_world.e' is up to date.
```

Makefiles

Un ejemplo más real: dos archivos .c en un solo ejecutable

```
$ cat Makefile
programa.e: calculadora.c trig.c
    gcc calculadora.c trig.c -o programa.e

$ make
gcc calculadora.c trig.c -o programa.e

$ make
make: 'programa.e' is up to date.
```

Makefiles

Un ejemplo más real: dos archivos .c en un solo ejecutable

```
$ cat Makefile
programa.e: calculadora.c trig.c
    gcc calculadora.c trig.c -o programa.e

$ make
gcc calculadora.c trig.c -o programa.e

$ make
make: 'programa.e' is up to date.

$ vi trig.c          #Actualizamos trig.c

$ make               #make detecta que trig.c cambió
gcc calculadora.c trig.c -o programa.e
```

Makefiles: Más que un script

```
$ cat Makefile
programa.e: calculadora.o trig.o
    gcc calculadora.o trig.o -o programa.e
calculadora.o: calculadora.c
    gcc -c calculadora.c
trig.o: trig.c   #trig.o depende de trig.c
    gcc -c trig.c
$ make
gcc -c calculadora.c
gcc -c trig.c
gcc calculadora.o trig.o -o programa.e
$ vi trig.c      #Actualizamos trig.c
$ make           #make detecta que trig.c cambió => trig.o cambió
gcc -c trig.c    #genero nuevo trig.o
gcc calculadora.o trig.o -o programa.e
```

Makefiles: variables automáticas

```
$ cat Makefile
programa.e: calculadora.o trig.o
    gcc $^ -o $@

calculadora.o: calculadora.h calculadora.c
    gcc -c $<

trig.o: trig.h trig.c
    gcc -c $<
```

$^: Las dependencias
$@: El target
$<: La última dependencia

Makefiles: reglas implícitas

```
$ cat Makefile
programa.e: calculadora.o trig.o
	gcc $^ -o $@

%.o: %.h %.c
	gcc -c $<
```

$^: Las dependencias
$@: El target
$<: La última dependencia

Makefiles: variables

```
$ cat Makefile
CC = gcc
CFLAGS = -O3
LDFLAGS = -lm
programa.e: calculadora.o trig.o
	$(CC) $^ $(LDFLAGS) -o $@

%.o: %.h %.c
	$(CC) $(CFLAGS) -c $<
```

$^: Las dependencias
$@: El target
$<: La última dependencia

Makefiles: obtener los archivos

```
$ cat Makefile
CC = gcc
CFLAGS = -O3
LDFLAGS = -lm
SRC = $(wildcard *.c)
OBJ = $(patsubst %.c, %.o, $(SRC))
programa.e: $(OBJ)
    $(CC) $^ $(LDFLAGS) -o $@

%.o: %.h %.c
    $(CC) $(CFLAGS) -c $<
```

$^: Las dependencias
$@: El target
$<: La última dependencia

Makefiles: algunos detalles

```
$ cat Makefile
...
.PHONY: all clean
all: programa.e

programa.e: $(OBJ)
    $(CC) $^ $(LDFLAGS) -o $@

%.o: %.h %.c
    $(CC) $(CFLAGS) -c $<

clean:
    rm -rfv $(OBJ) programa.e
```

no son archivos
la primera regla
clean (por las dudas)