Charla para estudiantes que saben usar la computadora bien y quieren aprender a hacer otras cosas bien también

David Alejandro González Márquez
Maximiliano Geier

Charlas para estudiantes que saben usar la computadora bien y quieren aprender a hacer otras cosas bien también

Charla 01 - Procesamiento de texto

Charla 02 - Sistema linux y redes

Charla 03 - Compilación y procesos

Charlas para estudiantes que saben usar la computadora bien y quieren aprender a hacer otras cosas bien también

Charla 01 - Procesamiento de texto

Charla 02 - Sistema linux y redes

Charla 03 - Compilación y procesos

Charla 01 - Procesamiento de texto

Objetivo

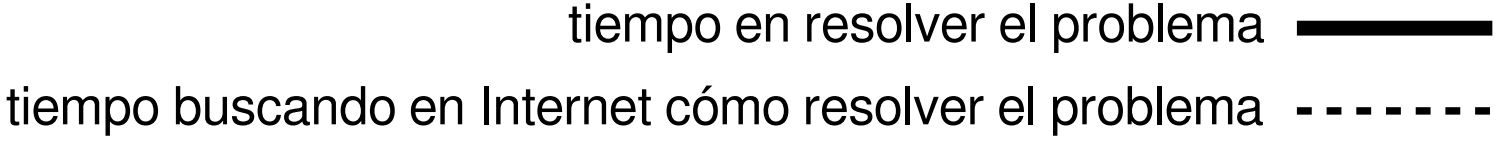
- Enseñar comandos y combinaciones de comandos.
- Luego de la charla se espera que busquen ejemplos.
- Es una referencia *quick and dirty*.

Público

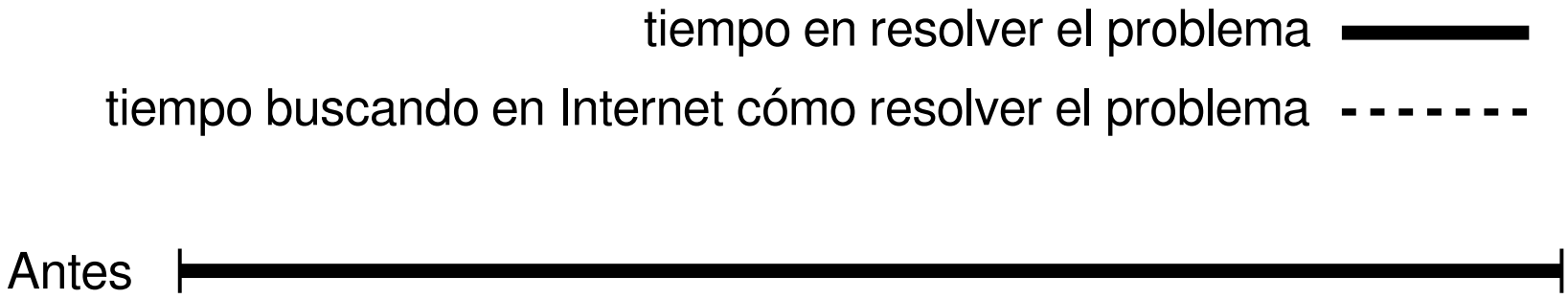
- Cualquiera que quiera aprender a procesar texto eficientemente con herramientas simples.

¿Por qué aprender esto?

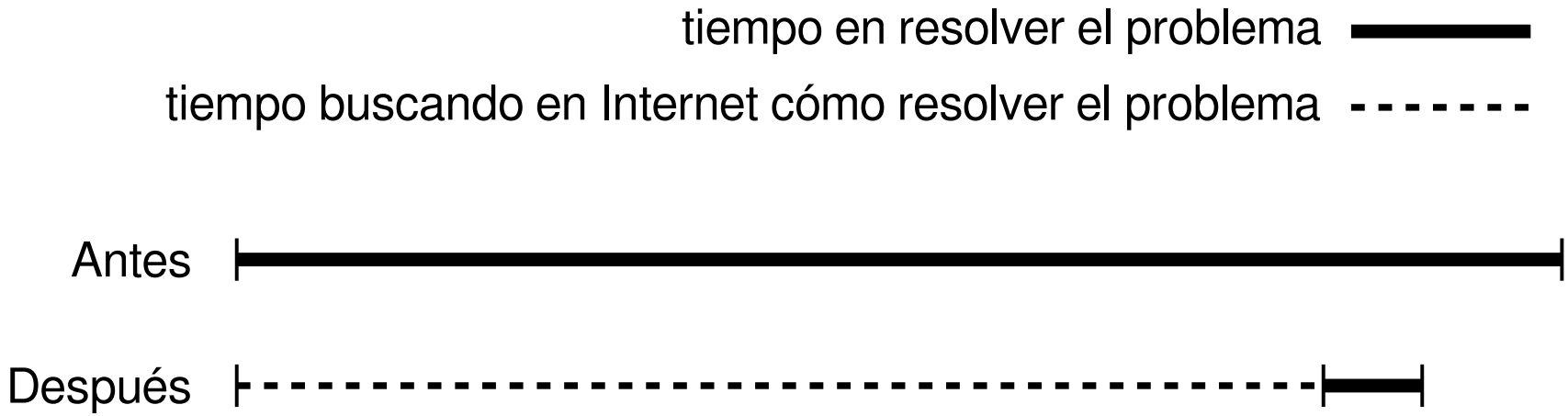
¿Por qué aprender esto?



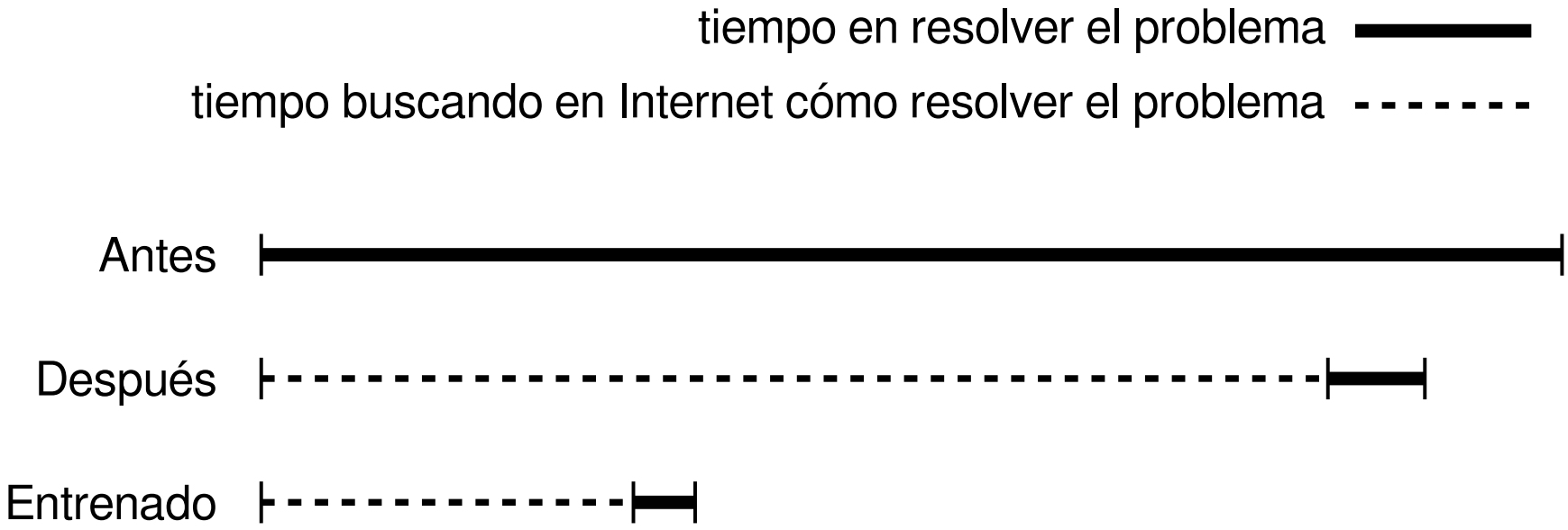
¿Por qué aprender esto?



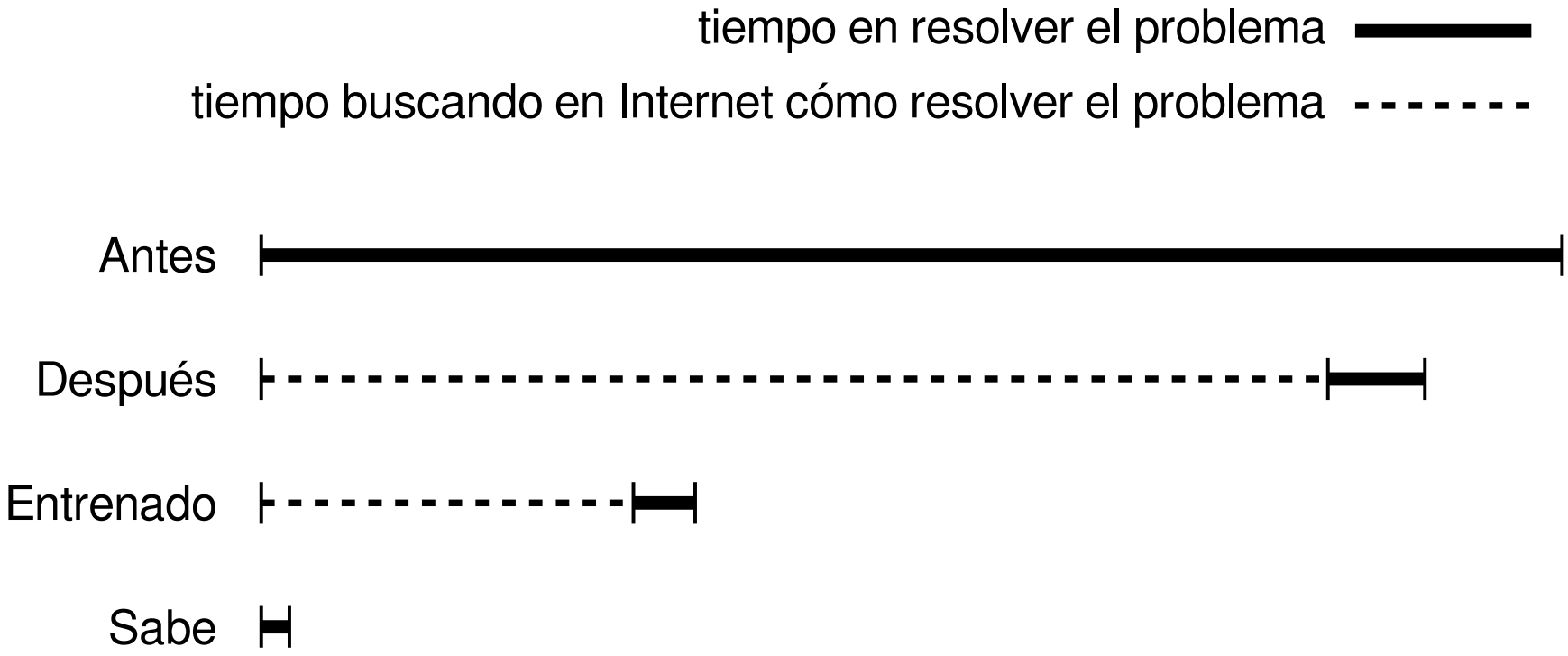
¿Por qué aprender esto?



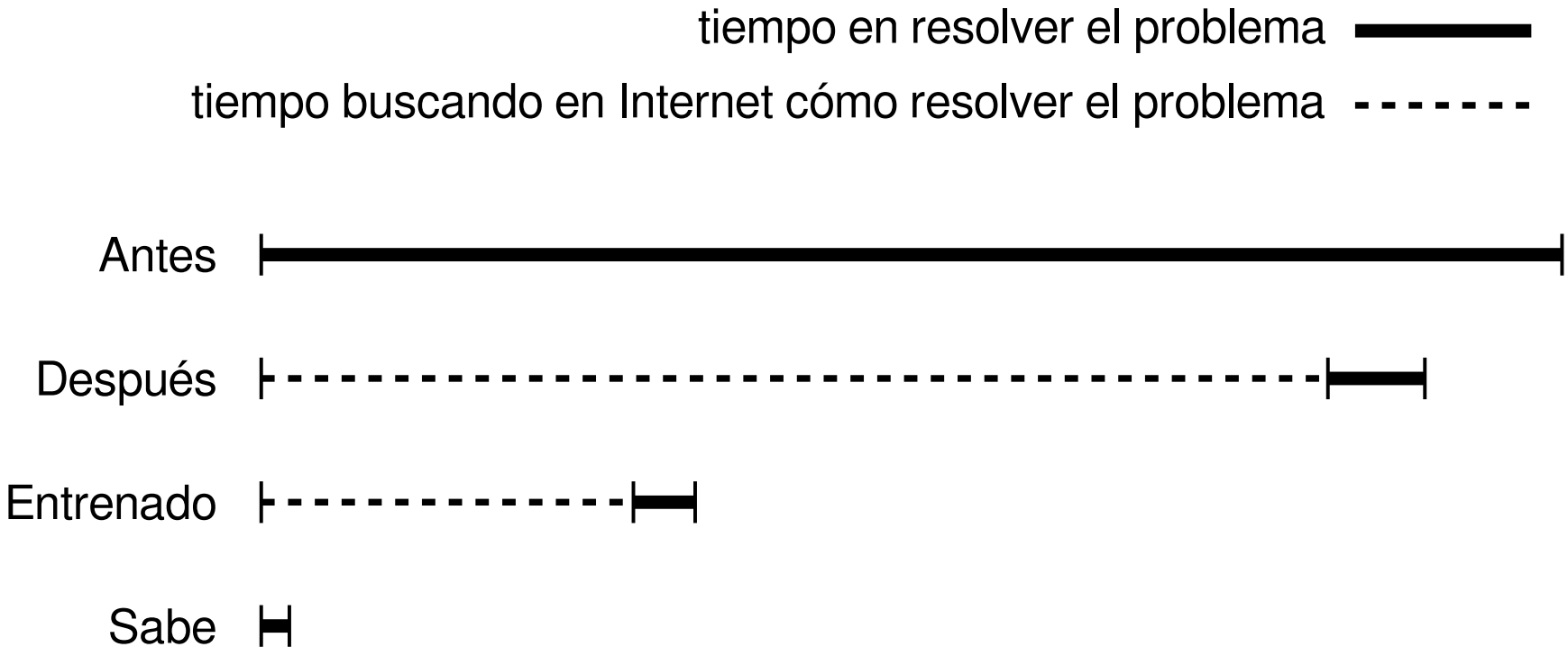
¿Por qué aprender esto?



¿Por qué aprender esto?

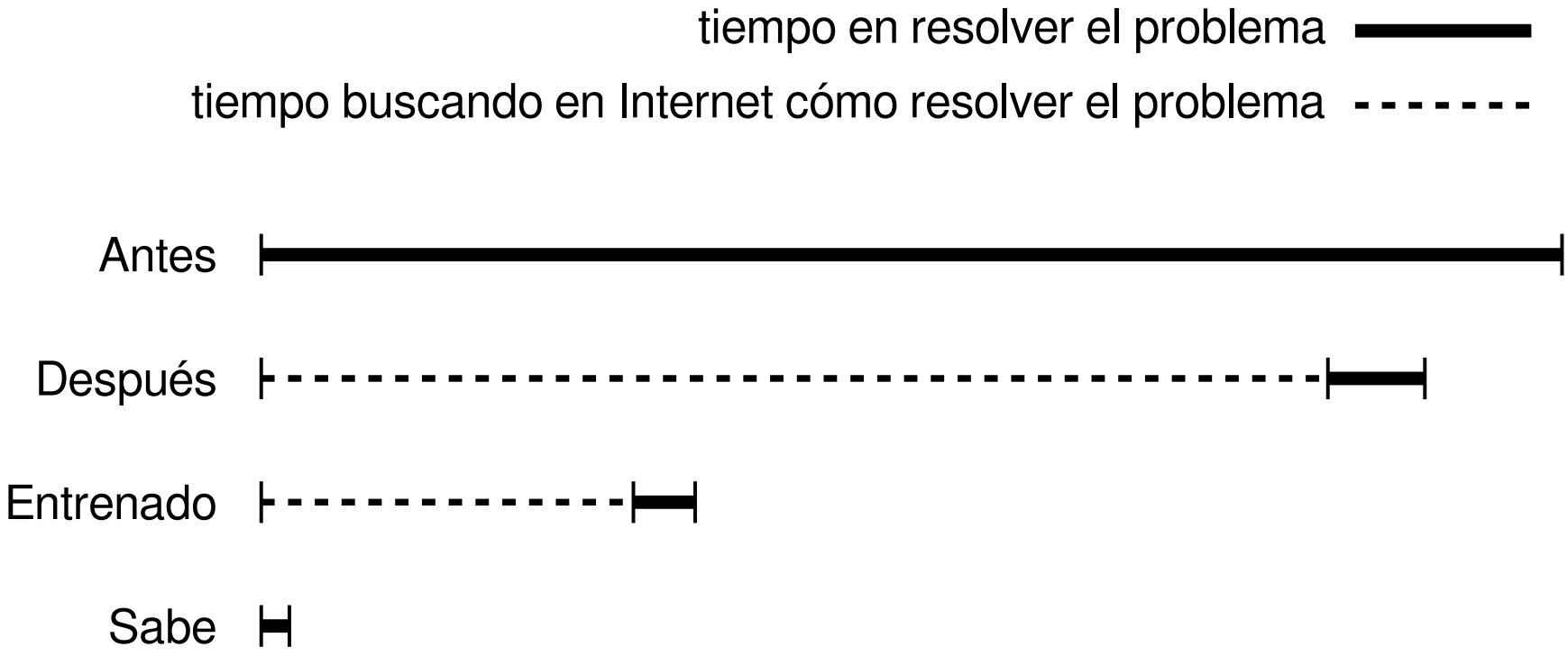


¿Por qué aprender esto?



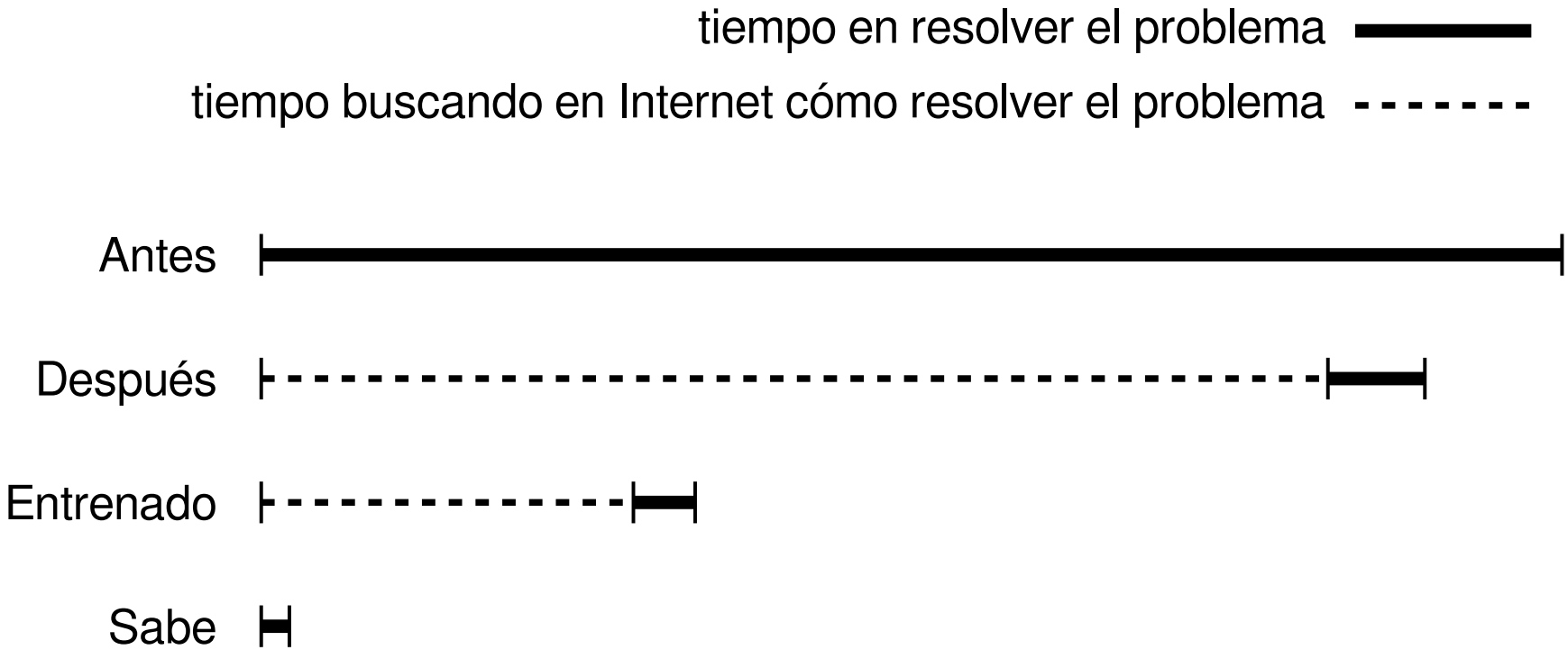
- Usar bien los comandos requiere **entrenamiento** y **práctica**.

¿Por qué aprender esto?



- Usar bien los comandos requiere **entrenamiento** y **práctica**.
- Si después de está charla no aplican lo aprendido en su trabajo · · ·

¿Por qué aprender esto?



- Usar bien los comandos requiere **entrenamiento** y **práctica**.
- Si después de está charla no aplican lo aprendido en su trabajo · · ·

Commando echo

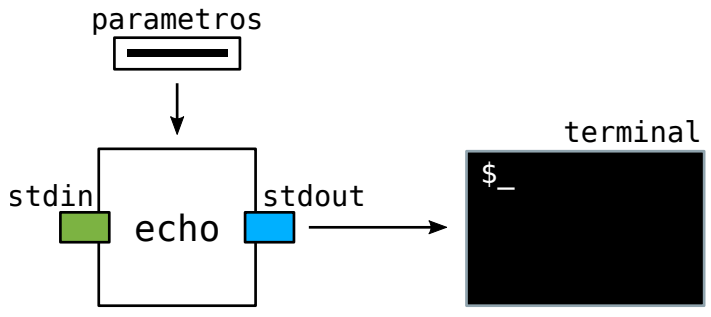
```
$ echo "hello world"
```

- Toma un texto como parámetro (“`hello world`”).
- Imprime en la salida estándar (`stdout`) el parámetro.

Commando echo

```
$ echo "hello world"
```

- Toma un texto como parámetro (“`hello world`”).
- Imprime en la salida estándar (`stdout`) el parámetro.




```
$ echo "hello world"
hello world
```

Commando wc y operador |

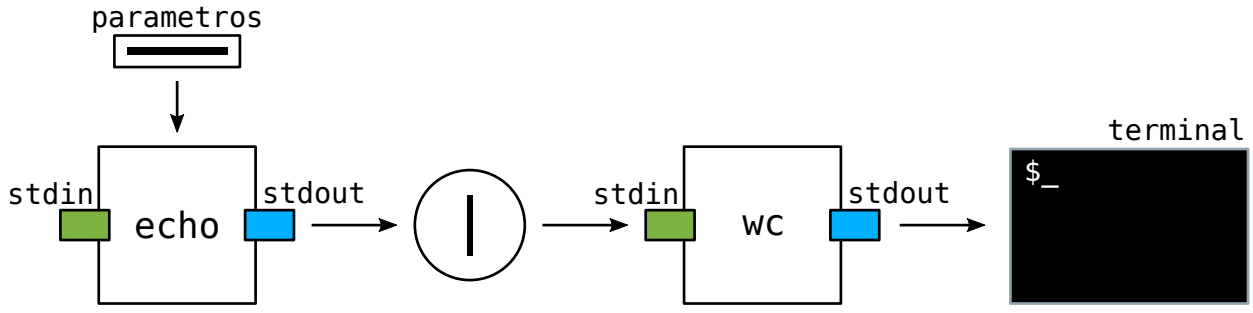
```
$ echo "hello world" | wc
```

- El operador "|" (pipe) conecta una salida con una entrada.
- El programa "wc" cuenta líneas, palabras y caracteres.
- Toma por su entrada estándar (`stdin`) el resultado de "echo".

Commando wc y operador |

```
$ echo "hello world" | wc
```

- El operador "|" (pipe) conecta una salida con una entrada.
- El programa "wc" cuenta líneas, palabras y caracteres.
- Toma por su entrada estándar (`stdin`) el resultado de "echo".




```
$ echo "hello world" | wc
      1       2      12
```

Commando cat

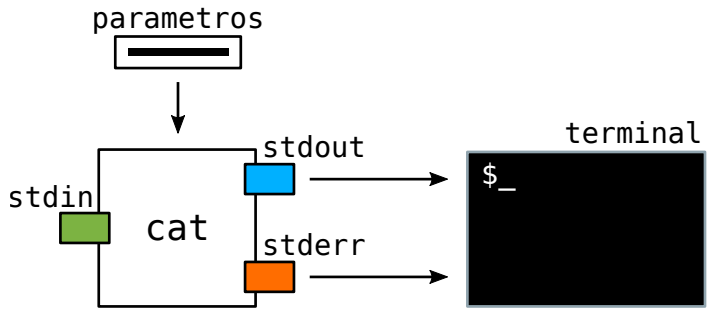
```
$ cat sarasa.txt
```

- El programa “cat” lee un archivo y lo imprime por stdout.
- Además de stdout los programas pueden generar stderr (salida de error).

Commando cat

```
$ cat sarasa.txt
```

- El programa “cat” lee un archivo y lo imprime por stdout.
- Además de stdout los programas pueden generar stderr (salida de error).




```
$ cat sarasa.txt
cat: sarasa.txt: No such file or directory
```

Operador >

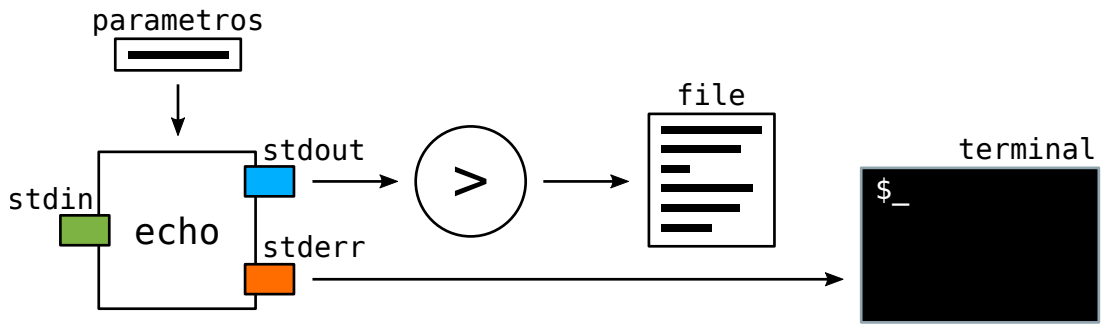
```
$ echo "hello world" > a.txt
```

- El operador “>” permite redireccionar la salida a un archivo.

Operador >

```
$ echo "hello world" > a.txt
```

- El operador ">" permite redireccionar la salida a un archivo.




```
$ echo "hello world" > a.txt

$ cat a.txt
hello world
```

Operador >>

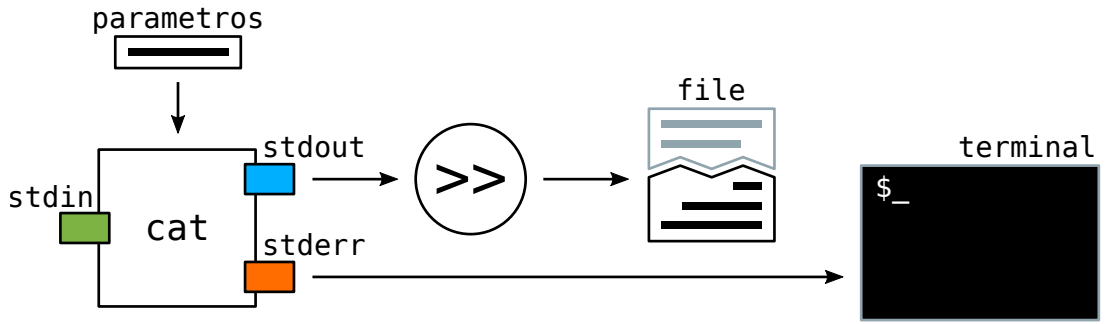
```
$ cat a.txt >> file.txt
```

- El operador “>>” permite redireccionar la salida y la concatena a un archivo.

Operador >>

```
$ cat a.txt >> file.txt
```

- El operador “>>” permite redireccionar la salida y la concatena a un archivo.




```
$ cat file.txt
texto cualquiera

$ cat a.txt >> file.txt
$ cat file.txt
texto cualquiera
hello world
```

Operador &>

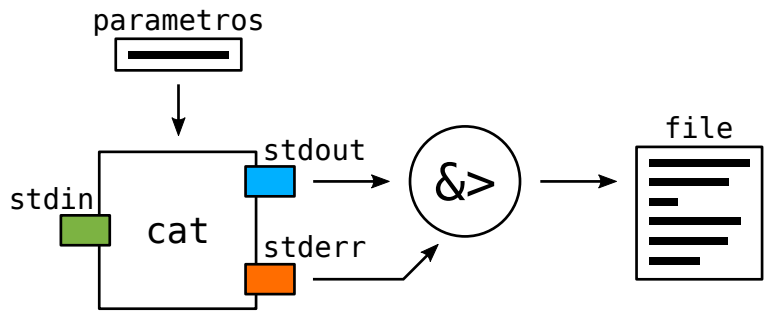
```
$ cat sarasa.txt &> a.txt
```

- El operador “&>” permite redireccionar `stdout` y `stderr` a un archivo.

Operador &>

```
$ cat sarasa.txt &> a.txt
```

- El operador "&>" permite redireccionar `stdout` y `stderr` a un archivo.




```
$ cat sarasa.txt > a.txt
cat: sarasa.txt: No such file or directory

$ cat sarasa.txt &> a.txt
$ cat a.txt
cat: sarasa.txt: No such file or directory
```

Commando grep

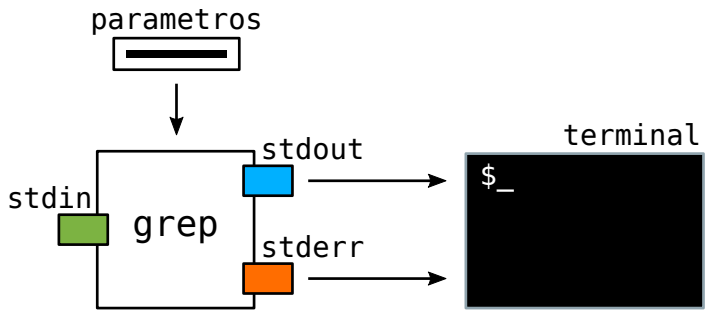
```
$ grep "hello" file.txt
```

- El programa `grep` busca línea por línea el texto `hello` en el archivo `file.txt`.

Commando grep

```
$ grep "hello" file.txt
```

- El programa grep busca línea por línea el texto `hello` en el archivo `file.txt`.




```
$ cat file.txt
red = rojo
hola = hello
cat = gato

$ grep "hello" file.txt
hola = hello
```

Commando grep

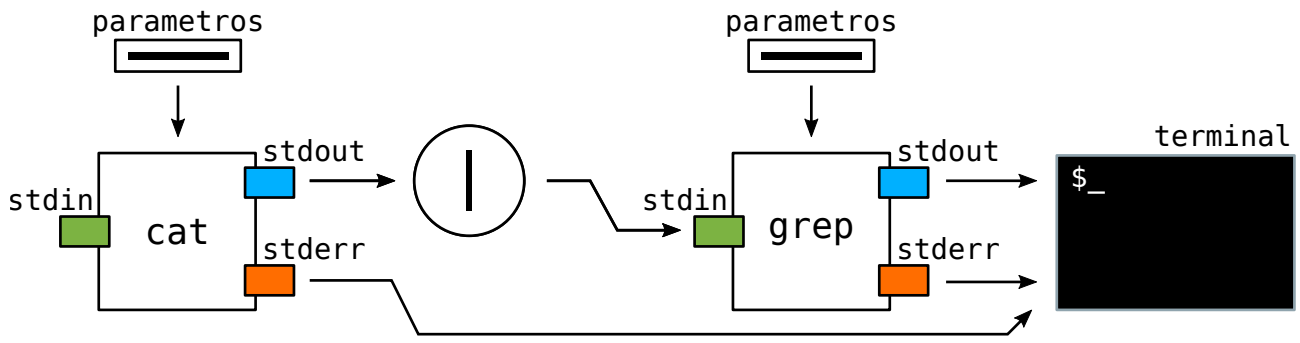
```
$ cat file.txt | grep "hello"
```

- Otra opción es imprimir en stdout y luego redireccionarlo a grep.

Commando grep

```
$ cat file.txt | grep "hello"
```

- Otra opción es imprimir en stdout y luego redireccionarlo a grep.




```
$ cat file.txt
red = rojo
hola = hello
cat = gato

$ cat file.txt | grep "hello"
hola = hello
```

Commando grep

grep además soporta expresiones regulares:

- Ejemplos
  - "^____" = busca al comienzo de la línea.
  - "____$" = busca al final de la línea.
  - "__.__" = un caracter cualquiera.
  - "__.*_" = un conjunto de caracteres cualquiera.

- Otros parámetros:

- -i = insensible.
- -v = inverso.
- -r = recursivo.
- -A *n* = imprime n líneas arriba del encontrado.
- -B *n* = imprime n líneas debajo del encontrado.
- -C *n* = imprime n líneas arriba y abajo del encontrado.

Commando sed

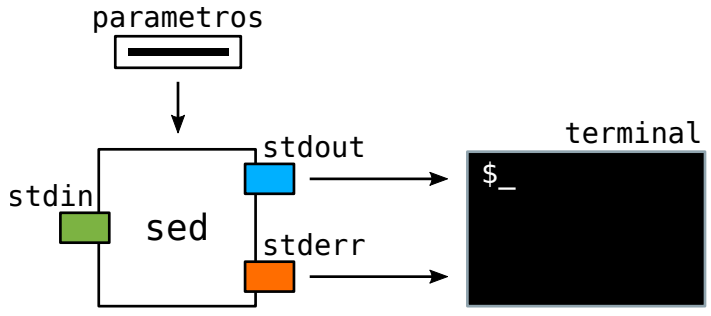
```
$ sed 's/__1__/__2__/g' file.txt
```

- Remplaza todas las apariciones de “__1__” por “__2__” en file.txt e imprime el resultado en la `stdout`.
- “g” indica que remplaza todas las apariciones.

Commando sed

```
$ sed 's/__1__/__2__/g' file.txt
```

- Remplaza todas las apariciones de “__1__” por “__2__” en file.txt e imprime el resultado en la `stdout`.
- “g” indica que remplaza todas las apariciones.




```
$ sed -i 's/__1__/__2__/g' file.txt
```

- La opción “`-i`” modifica el archivo `file.txt`.

Commando awk

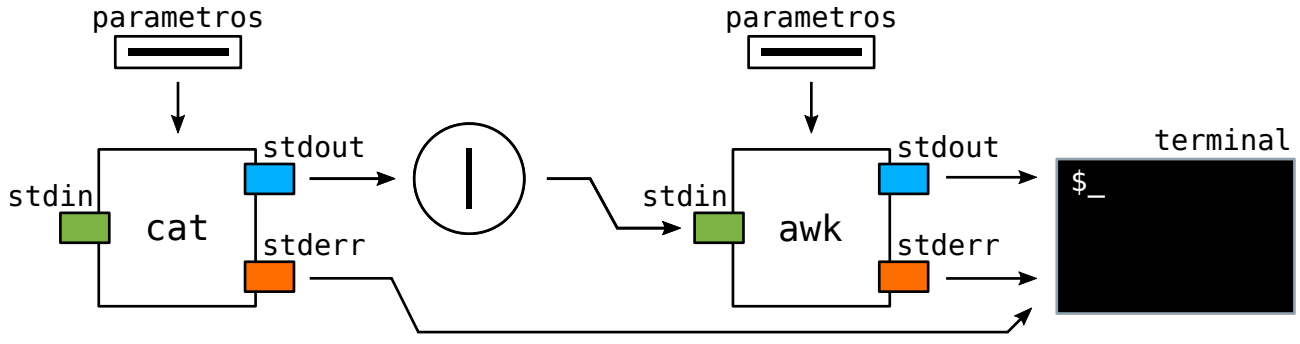
```
$ cat file.txt | awk '{print $3}'
```

- “awk” es un lenguaje de procesamiento de texto.
- *tokeniza* cada línea y ejecuta el programa indicado.
- usa como separador espacios o tabs.
- el operador “`$i`” se remplaza el i-ésimo texto tokenizado.

Commando awk

```
$ cat file.txt | awk '{print $3}'
```

- “awk” es un lenguaje de procesamiento de texto.
- *tokeniza* cada línea y ejecuta el programa indicado.
- usa como separador espacios o tabs.
- el operador “`$i`” se remplaza el i-ésimo texto tokenizado.



Ejemplo:

```
Estúpido y sensual Flanders!
$1 = "Estúpido", $2 = "y" $3 = "sensual", $4 = "Flanders!"
```

Commando awk

```
$ cat file.txt
Wenquan  China    5.000
Potosí   Bolivia  4.090
Oruro    Bolivia  3.706
Lhasa    China    3.650
Cuzco    Perú     3.399
```

```
$ cat file.txt | awk '{print $3}'
5.000
4.090
3.706
3.650
3.399
```

Commando awk

```
$ cat file.txt | awk -F "." '{print $3}'
```

- El parametro “-F” indica el separador utilizado.

Commando awk

```
$ cat file.txt | awk -F "." '{print $3}'
```

- El parametro “-F” indica el separador utilizado.

```
$ cat autobots.txt
Name.Alternate modes.Rank.Function
Optimus Prime.Peterbilt flat-nosed.10.Autobot Leader
Bumblebee.Volkswagen Beetle.7.Espionage
Windcharger.Pontiac Fire Bird Trans Am.5.Warrior
Outback.Land Rover.4.Administrative Assistant

$ cat autobots.txt | awk -F "." '{print $3}'
Rank
10
7
5
4
```

Commando `sort` y `uniq`

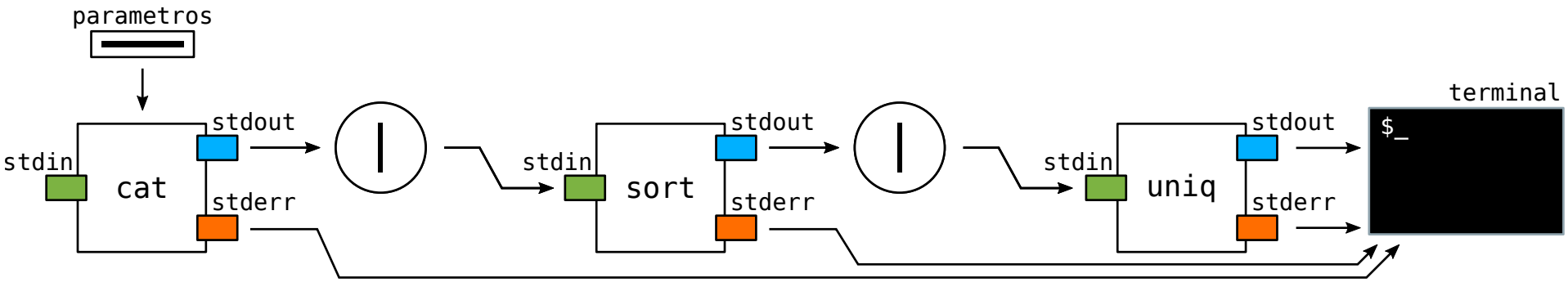
```
$ cat file.txt | sort | uniq
```

- “`sort`” ordena lexicográficamente las líneas.
- “`uniq`” imprime una línea si es distinta a la anterior.

Commando `sort` y `uniq`

```
$ cat file.txt | sort | uniq
```

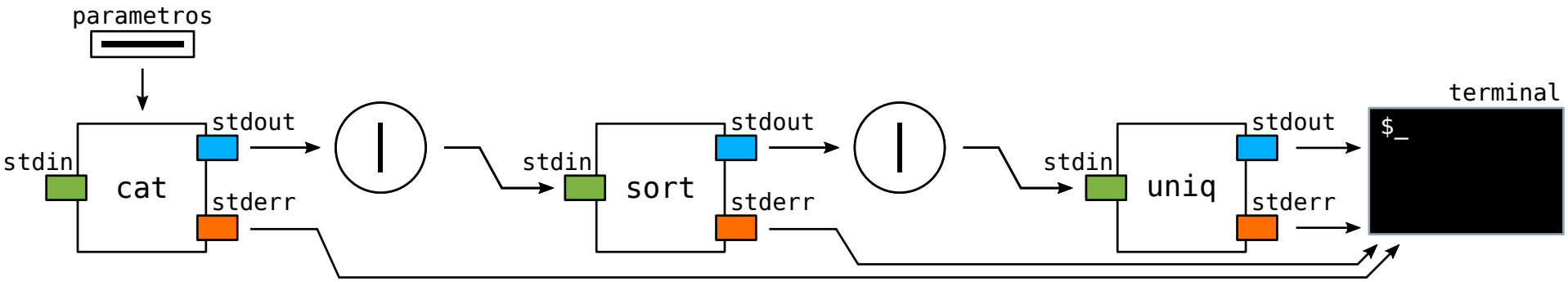
- “`sort`” ordena lexicográficamente las líneas.
- “`uniq`” imprime una línea si es distinta a la anterior.



Commando `sort` y `uniq`

```
$ cat file.txt | sort | uniq
```

- “`sort`” ordena lexicográficamente las líneas.
- “`uniq`” imprime una línea si es distinta a la anterior.



file

```
Japan
Russia
Japan
Egypt
South Korea
Egypt
Bangladesh
South Korea
Mexico
Bangladesh
```

sort

```
Bangladesh
Bangladesh
Egypt
Egypt
Japan
Japan
Mexico
Russia
South Korea
South Korea
```

uniq

```
Bangladesh
Egypt
Japan
Mexico
Russia
South Korea
```

Commando `sort` y `uniq`

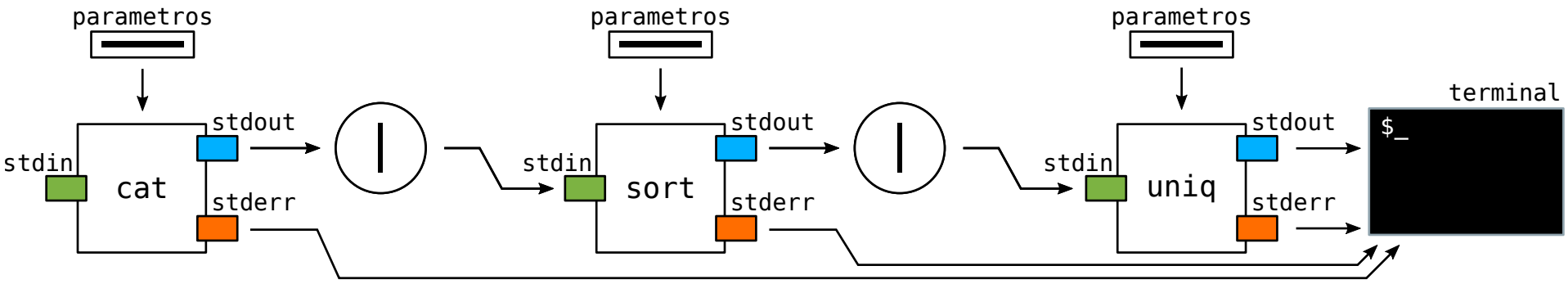
```
$ cat file.txt | sort -n | uniq -c
```

- “-n” ordena numéricamente usando el primer dato como número.
- “-c” indica la cantidad de líneas que fueron no impresas.

Commando `sort` y `uniq`

```
$ cat file.txt | sort -n | uniq -c
```

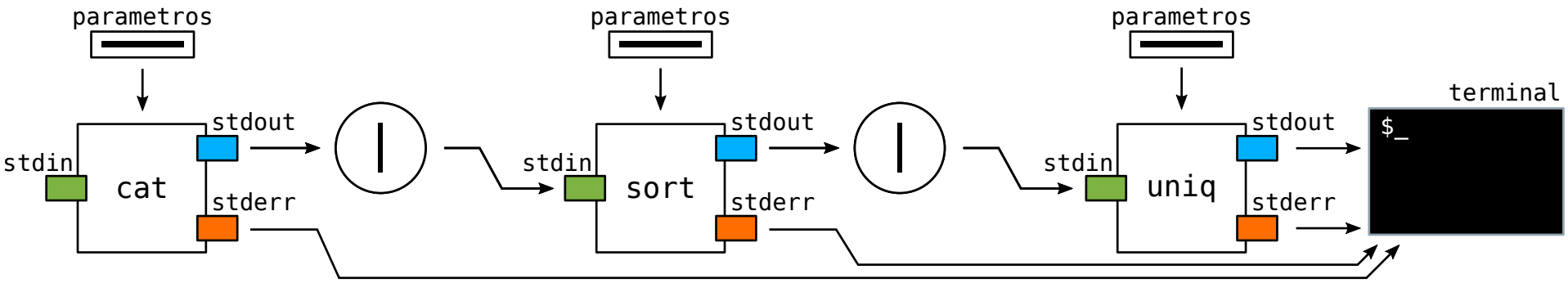
- “-n” ordena numéricamente usando el primer dato como número.
- “-c” indica la cantidad de líneas que fueron no impresas.



Commando sort y uniq

```
$ cat file.txt | sort -n | uniq -c
```

- "-n" ordena numéricamente usando el primer dato como número.
- "-c" indica la cantidad de líneas que fueron no impresas.



file

```
123 sarasa
99
123 SARASA
9999
99
313
123 sarasa
9999
213
9999
```

sort

```
99
99
123 sarasa
123 sarasa
123 SARASA
213
313
9999
9999
9999
```

uniq

```
      2 99
      2 123 sarasa
      1 123 SARASA
      1 213
      1 313
      3 9999
```

Commando `find`

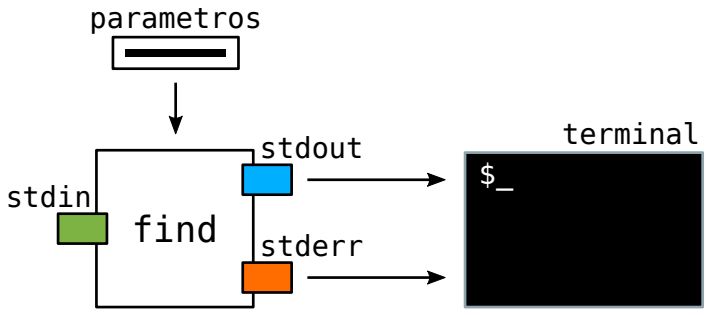
```
$ find
```

- Imprime todos los nombres de archivos/carpetas recursivamente desde la carpeta actual.

Commando `find`

```
$ find
```

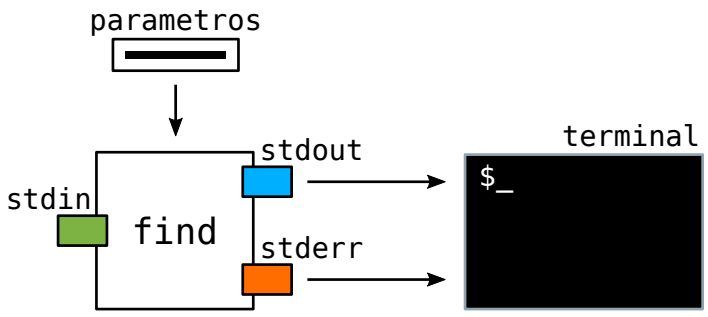
- Imprime todos los nombres de archivos/carpetas recursivamente desde la carpeta actual.



Commando `find`

```
$ find
```

- Imprime todos los nombres de archivos/carpetas recursivamente desde la carpeta actual.

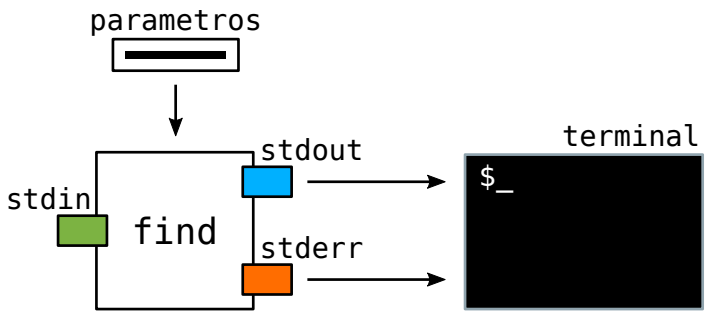



```
.
./results.txt
./files
./files/other
./files/other/file.txt
./files/file.txt
./colums.dat
./country.txt
./sources.list
./numbers.txt
./names.txt
./autobots.txt
./hosts
```

Commando `find`

```
$ find -name '*.txt' -type f
```

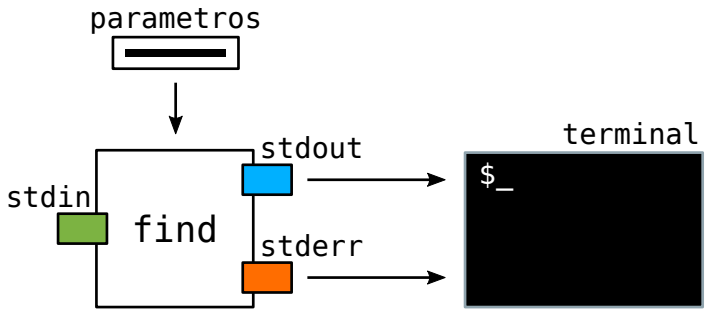
- Imprime solamente los archivos cuyo nombre termina con “`.txt`”.



Commando `find`

```
$ find -name '*.txt' -type f
```

- Imprime solamente los archivos cuyo nombre termina con "`.txt`".



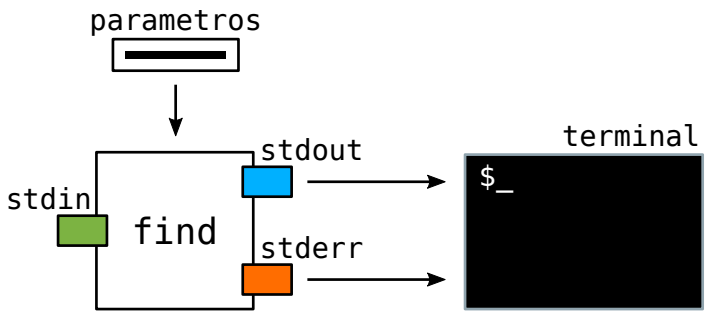
Ejemplo:

```
./results.txt
./files/other/file.txt
./files/file.txt
./country.txt
./numbers.txt
./names.txt
./autobots.txt
```

Commando `find`

```
$ find -name '*.txt' -type f -exec wc -l {} \;
```

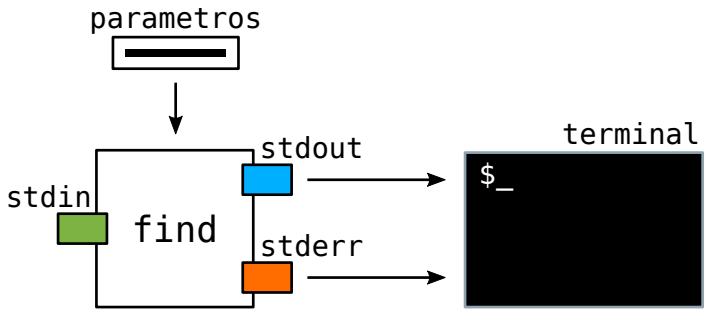
- Por cada archivo terminado en "`.txt`" ejecuta "`wc -l`". Esto cuenta la cantidad líneas de cada archivo.



Commando `find`

```
$ find -name '*.txt' -type f -exec wc -l {} \;
```

- Por cada archivo terminado en "`.txt`" ejecuta "`wc -l`". Esto cuenta la cantidad líneas de cada archivo.



Ejemplo:

```
19 ./results.txt
1 ./files/other/file.txt
1 ./files/file.txt
10 ./country.txt
10 ./numbers.txt
51 ./names.txt
6 ./autobots.txt
```

Commando xargs

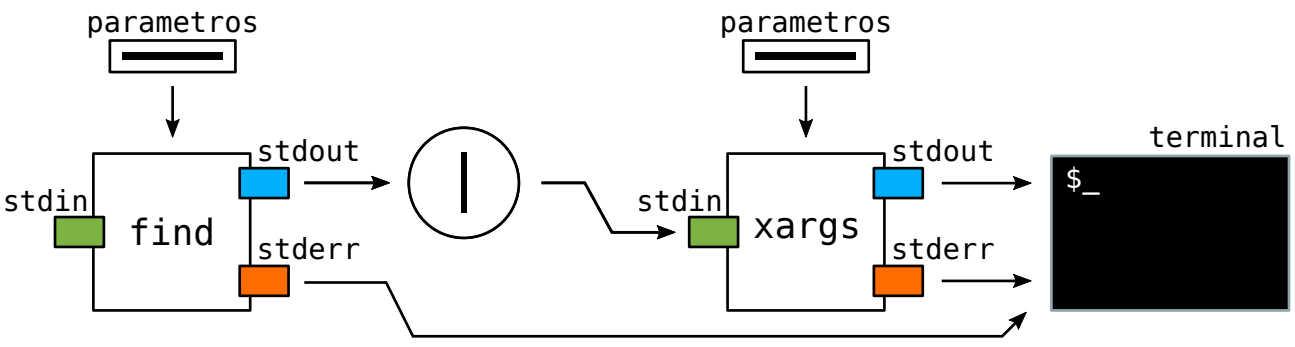
```
$ find -name '*.txt' -type f | xargs wc -l
```

- `xargs` toma varios archivos al mismo tiempo por `stdin` y ejecuta el comando pasado por parámetro.
- Esto concatena todos los archivos, y los pasa todos juntos a `wc -l` (cuenta líneas totales).

Commando xargs

```
$ find -name '*.txt' -type f | xargs wc -l
```

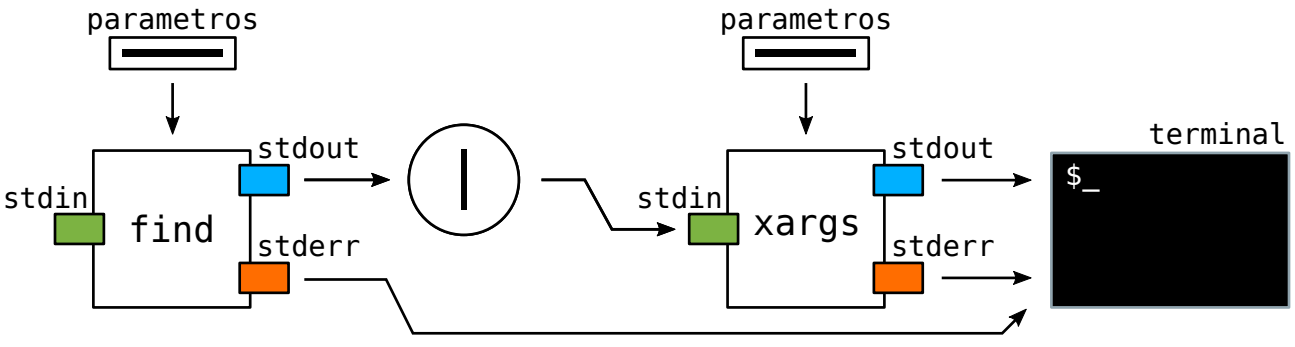
- `xargs` toma varios archivos al mismo tiempo por `stdin` y ejecuta el comando pasado por parámetro.
- Esto concatena todos los archivos, y los pasa todos juntos a `wc -l` (cuenta líneas totales).



Commando xargs

```
$ find -name '*.txt' -type f | xargs wc -l
```

- `xargs` toma varios archivos al mismo tiempo por `stdin` y ejecuta el comando pasado por parámetro.
- Esto concatena todos los archivos, y los pasa todos juntos a `wc -l` (cuenta líneas totales).



Ejemplo:

```
19 ./results.txt
 1 ./files/other/file.txt
 1 ./files/file.txt
10 ./country.txt
10 ./numbers.txt
51 ./names.txt
 6 ./autobots.txt
98 total
```

Commando head

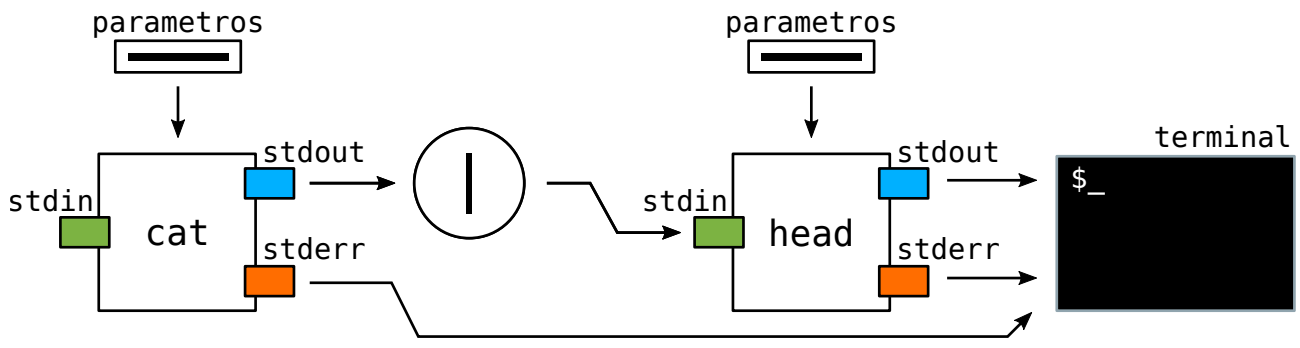
```
$ cat file.txt | head -n 5
```

- Imprime las primeras 5 líneas de un archivo.

Commando head

```
$ cat file.txt | head -n 5
```

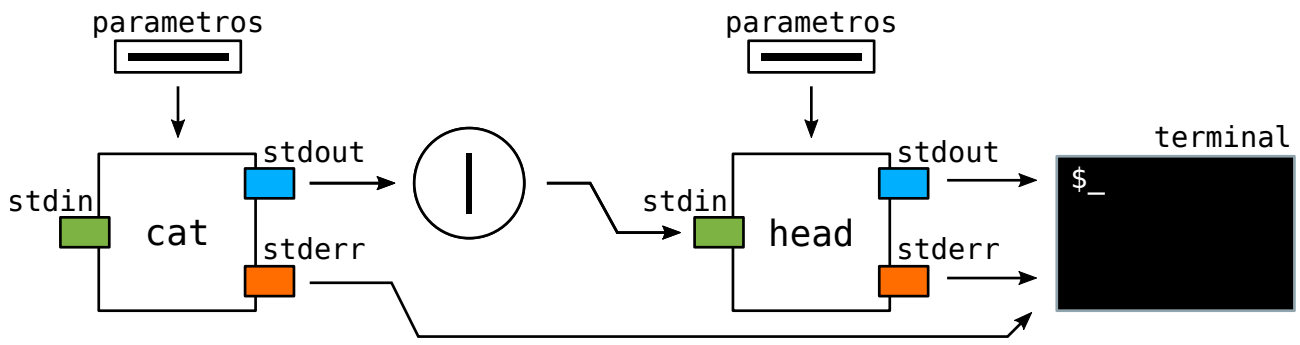
- Imprime las primeras 5 líneas de un archivo.



Commando head

```
$ cat file.txt | head -n 5
```

- Imprime las primeras 5 líneas de un archivo.



Ejemplo:

```
Dots   Boxes
13     2134
4      4325
12     234
23     43
```

Commando `tail`

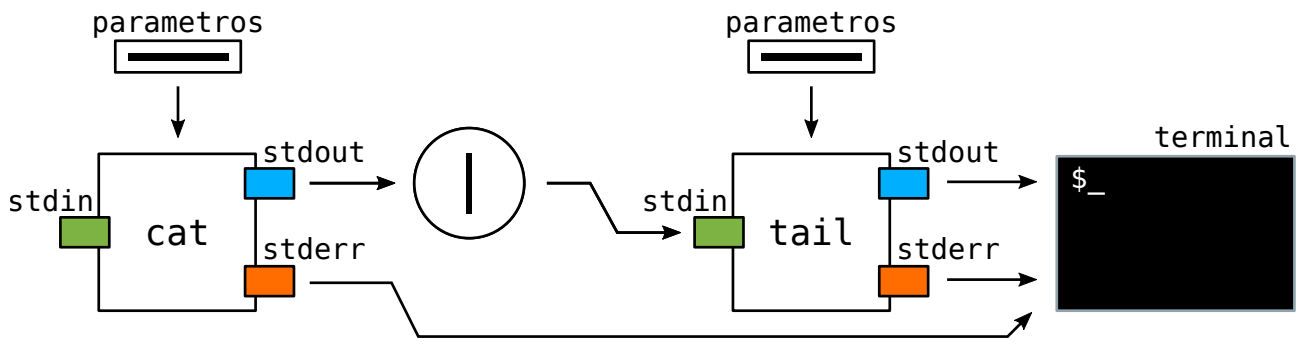
```
$ cat columns.dat | tail -n 5
```

- Imprime las últimas 5 líneas del archivo.

Commando `tail`

```
$ cat columns.dat | tail -n 5
```

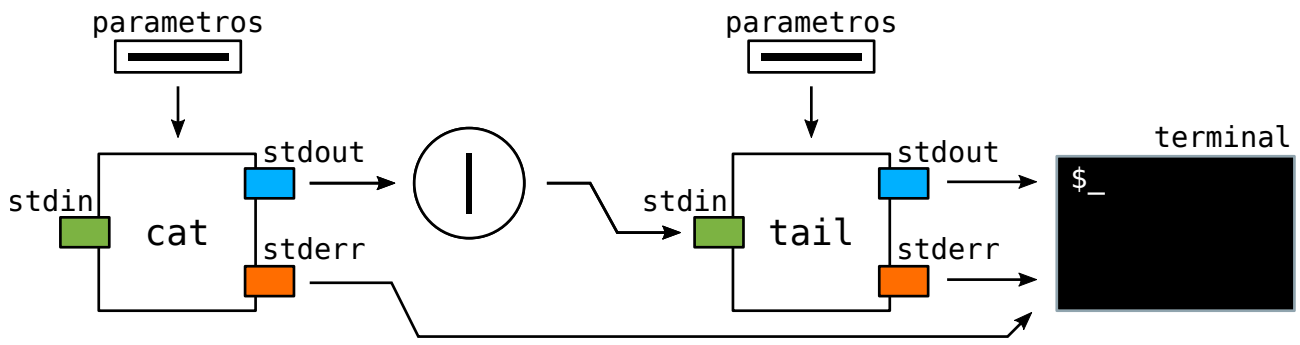
- Imprime las últimas 5 líneas del archivo.



Commando `tail`

```
$ cat columns.dat | tail -n 5
```

- Imprime las últimas 5 líneas del archivo.



Ejemplo:

```
463     56
4       795
75      2
632     690
46      78
```

Commando `tail`

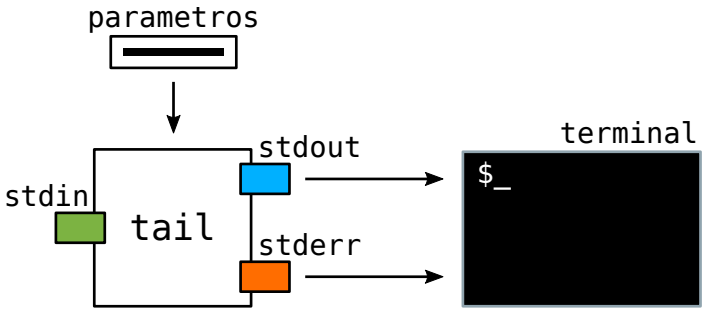
```
$ tail -f /var/log/syslog
```

- Muestra las últimas líneas 10 líneas
- Además, se queda esperando a que otro programa agregue nuevas líneas al mismo archivo para mostrarlas.

Commando `tail`

```
$ tail -f /var/log/syslog
```

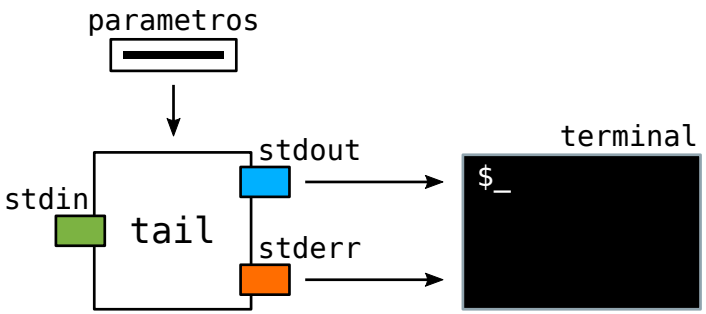
- Muestra las últimas líneas 10 líneas
- Además, se queda esperando a que otro programa agregue nuevas líneas al mismo archivo para mostrarlas.



Commando `tail`

```
$ tail -f /var/log/syslog
```

- Muestra las últimas líneas 10 líneas
- Además, se queda esperando a que otro programa agregue nuevas líneas al mismo archivo para mostrarlas.



Ejemplo:

```
May 12 00:53:54 faraday dhclient[14419]: XMT: Reques
May 12 00:53:54 faraday NetworkManager[853]: <info>
May 12 00:53:54 faraday NetworkManager[853]: <info> [
May 12 00:53:56 faraday whoopsie[1556]: [00:53:56] o
May 12 00:53:56 faraday kdeconnectd.desktop[2511]: k
May 12 00:53:56 faraday kdeconnectd.desktop[2511]: kde
May 12 00:53:56 faraday kdeconnectd.desktop[2511]: kde
May 12 00:53:56 faraday kdeconnectd.desktop[2511]: k
_
```

[tail espera que más caracteres se escriban en el archivo]

Se termina el programa con `Control + C`

Commando tee

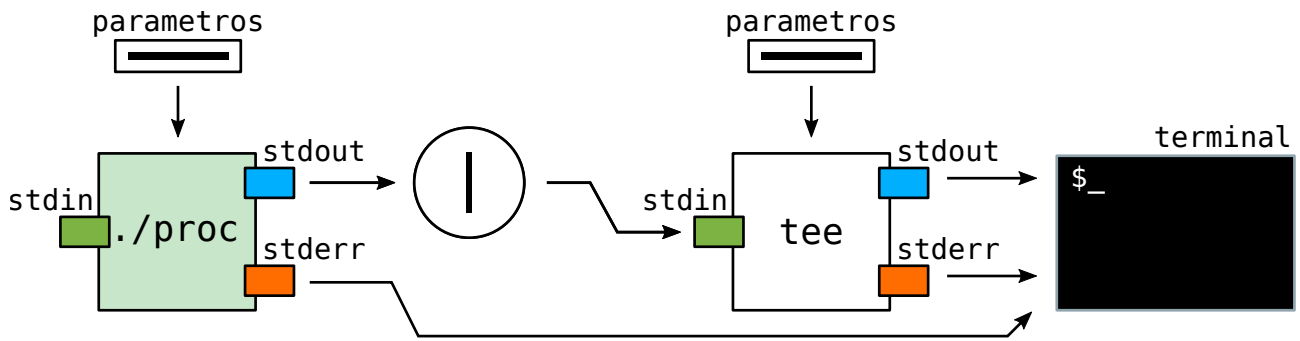
```
$ ./prog | tee log.txt
```

- Imprime la salida “prog” por stdout y además la redirecciona al archivo “log.txt”.

Commando tee

```
$ ./prog | tee log.txt
```

- Imprime la salida “prog” por stdout y además la redirecciona al archivo “log.txt”.



Commando tee

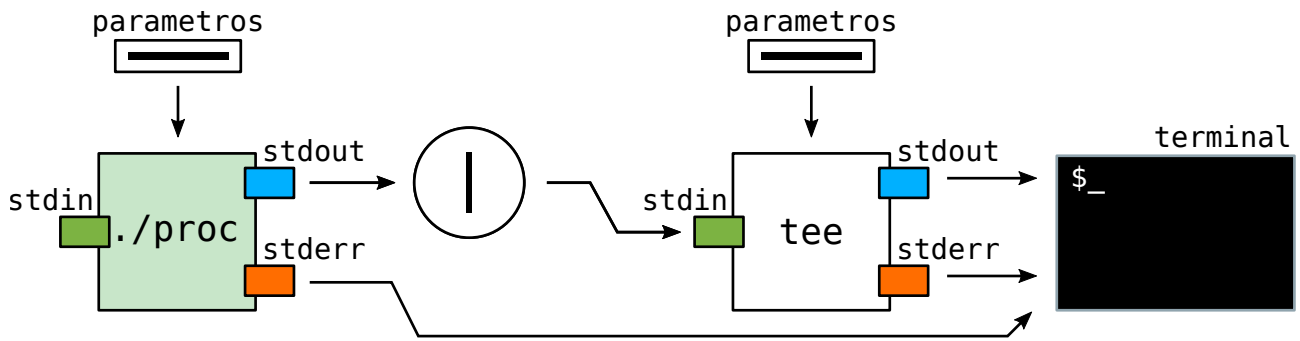
```
$ ./prog | tee -a log.txt
```

- La opción “-a” permite concatenar las líneas al archivo “log.txt”.

Commando tee

```
$ ./prog | tee -a log.txt
```

- La opción “-a” permite concatenar las líneas al archivo “`log.txt`”.



Commando paste

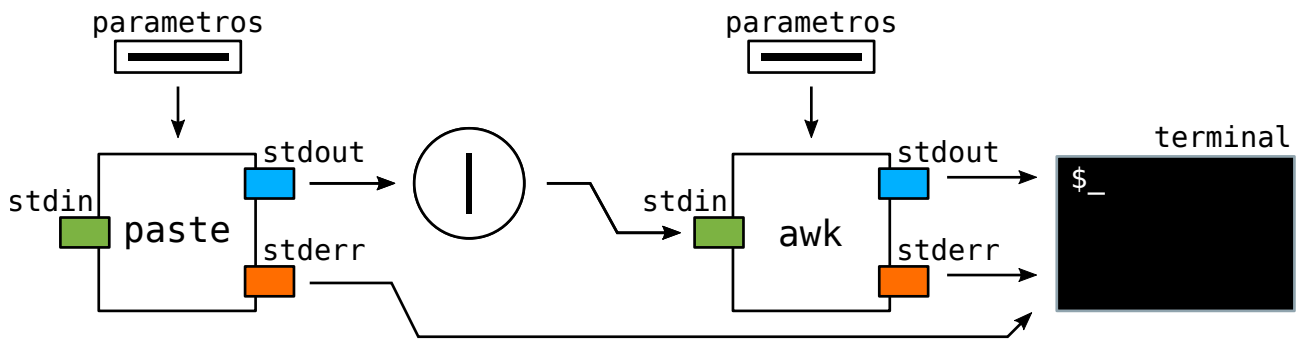
```
$ paste f1.txt f2.txt | awk '{print $1+$2 }'
```

- Concatena línea por línea los archivos pasados por parámetro.
- “awk” toma los dos primeros valores de cada línea y los suma.

Commando paste

```
$ paste f1.txt f2.txt | awk '{print $1+$2 }'
```

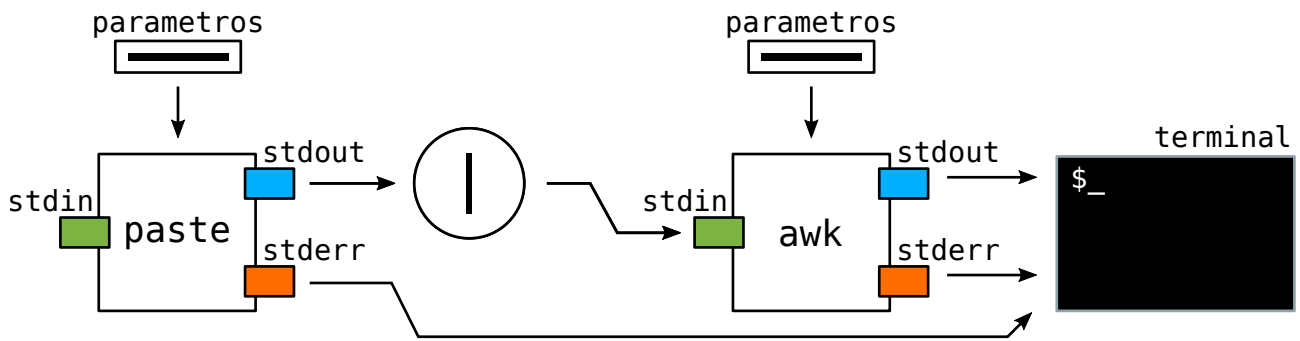
- Concatena línea por línea los archivos pasados por parámetro.
- “awk” toma los dos primeros valores de cada línea y los suma.



Commando paste

```
$ paste f1.txt f2.txt | awk '{print $1+$2 }'
```

- Concatena línea por línea los archivos pasados por parámetro.
- “awk” toma los dos primeros valores de cada línea y los suma.



f1.txt

```
100
201
300
401
500
601
```

f2.txt

```
601
500
401
300
201
100
```

paste

```
100  601
201  500
300  401
401  300
500  201
601  100
```

awk

```
701
701
701
701
701
701
```

Commando `file`

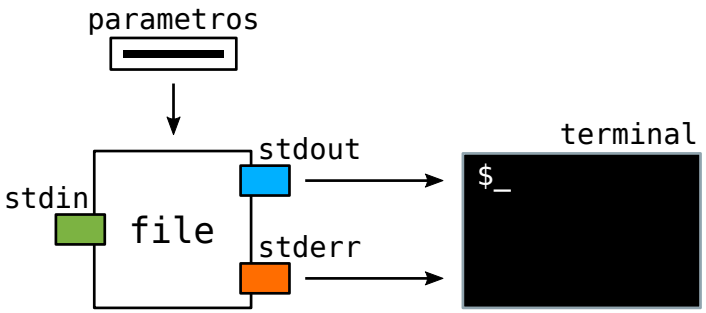
```
$ file sarasa
```

- Detecta el tipo de archivo de “`sarasa`”.

Commando `file`

```
$ file sarasa
```

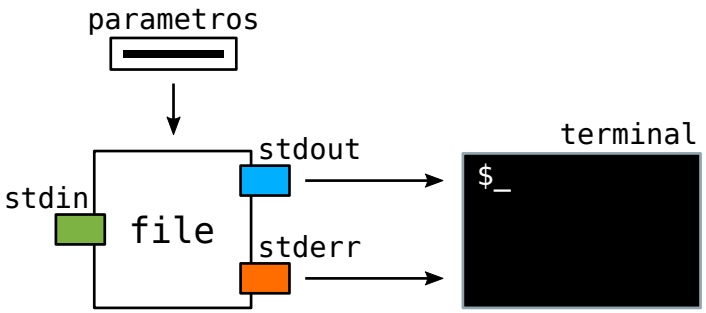
- Detecta el tipo de archivo de “sarasa”.



Commando `file`

```
$ file sarasa
```

- Detecta el tipo de archivo de “sarasa”.




```
sarasa: gzip compressed data, last modified:
Mon Mar  5 16:38:22 2018, from Unix
```

```
sarasa: ASCII text, with very long lines
```

```
sarasa: UTF-8 Unicode text
```

```
sarasa: PNG image data, 400 x 40, 8-bit/color
RGB, non-interlaced
```

```
sarasa: JPEG image data, JFIF standard 1.01,
aspect ratio, density 1x1, segment length 16,
baseline, precision 8, 498x278, frames 3
```

Operadores y comillas

`'...'`	Comillas simples: No interpretado por bash.
`"..."`	Comillas dobles: Es interpretado por bash, los caracteres de comillas no llegan al programa.
`` `...` ``	Comillas de ejecución: Son interpretadas por bash y ejecutadas. El resultado generado por el `stdout` remplaza la expresión.
`$(...)`	Contexto de ejecución: Son interpretadas por bash y ejecutadas. El resultado generado por el `stdout` remplaza la expresión (soportan anidamientos)
`... ; ...`	Separador de comandos: Permite separar un comando de otro. Estos se ejecutan uno a uno.
`... && ...`	Operador AND: Permite separar un comando de otro. Ejecuta comandos uno a uno. Si se genera un error deja de ejecutar los siguientes comandos.

Ejemplos complejos

Remplazar texto en varios archivos en varias carpetas

Remplazar texto en varios archivos en varias carpetas

Remplazar el texto “david” por “maxi”.

```
sed 's/david/maxi/g'
```

Remplazar texto en varios archivos en varias carpetas

Remplazar el texto “david” por “maxi”.

```
sed 's/david/maxi/g'
```

Agrego “-i” para hacerlo sobre los mismos archivos.

```
sed -i 's/david/maxi/g'
```

Remplazar texto en varios archivos en varias carpetas

Remplazar el texto “david” por “maxi”.

```
sed 's/david/maxi/g'
```

Agrego “-i” para hacerlo sobre los mismos archivos.

```
sed -i 's/david/maxi/g'
```

Uso “find” para seleccionar los archivos que quiero.

```
find -name '*.txt'
```

Remplazar texto en varios archivos en varias carpetas

Remplazar el texto “david” por “maxi”.

```
sed 's/david/maxi/g'
```

Agrego “-i” para hacerlo sobre los mismos archivos.

```
sed -i 's/david/maxi/g'
```

Uso “find” para seleccionar los archivos que quiero.

```
find -name '*.txt'
```

Uso “exec” para ejecutar el comando sobre cada archivo.

```
$ find -name '*.txt' -exec sed -i 's/david/maxi/g' "{}" \;
```

Cambiar el orden de columnas

Cambiar el orden de columnas

columns.dat

```
Dots  Boxes
13    2134
4     4325
12    234
23    43
43    5
5234  2
52    534
345   12
46    57
234   7
6     8
24    89
61    6
46    789
```

Cambiar el orden de columnas

columns.dat

```
Dots   Boxes
13     2134
4      4325
12     234
23     43
43     5
5234   2
52     534
345    12
46     57
234    7
6      8
24     89
61     6
46     789
```

Columnas Invertidas

```
Boxes   Dots
2134    13
4325    4
234     12
43      23
5       43
2       5234
534     52
12      345
57      46
7       234
8       6
89      24
6       61
789     46
```

Cambiar el orden de columnas

columns.dat

```
Dots   Boxes
13     2134
4      4325
12     234
23     43
43     5
5234   2
52     534
345    12
46     57
234    7
6      8
24     89
61     6
46     789
```

Columnas Invertidas

```
Boxes   Dots
2134    13
4325    4
234     12
43      23
5       43
2       5234
534     52
12      345
57      46
7       234
8       6
89      24
6       61
789     46
```

```
$ cat columns.dat | awk '{ print $2 "\t" $1 }'
```

Transponer los datos de una columna

Transponer los datos de una columna

Obtengo la columna indicada usando “awk”.

```
$ cat columns.dat | awk '{ print $1 }'
```

Transponer los datos de una columna

Obtengo la columna indicada usando “awk”.

```
$ cat columns.dat | awk '{ print $1 }'
```

Pero viene con el nombre de columna, lo borro usando “`tail`”

```
$ cat columns.dat | tail -n ???
```

Transponer los datos de una columna

Obtengo la columna indicada usando “awk”.

```
$ cat columns.dat | awk '{ print $1 }'
```

Pero viene con el nombre de columna, lo borro usando “`tail`”

```
$ cat columns.dat | tail -n ???
```

Pero, ¿Cómo hago para calcular la cantidad de filas?

```
$ wc columns.dat -l | awk '{print $1-1}'
```

Transponer los datos de una columna

Obtengo la columna indicada usando “awk”.

```
$ cat columns.dat | awk '{ print $1 }'
```

Pero viene con el nombre de columna, lo borro usando “`tail`”

```
$ cat columns.dat | tail -n ???
```

Pero, ¿Cómo hago para calcular la cantidad de filas?

```
$ wc columns.dat -l | awk '{print $1-1}'
```

Uso la expresión anterior en un contexto de ejecución.

```
$ cat columns.dat | tail -n $(wc columns.dat -l | awk '{print $1-1}')
```

Transponer los datos de una columna

Obtengo la columna indicada usando “awk”.

```
$ cat columns.dat | awk '{ print $1 }'
```

Pero viene con el nombre de columna, lo borro usando “`tail`”

```
$ cat columns.dat | tail -n ???
```

Pero, ¿Cómo hago para calcular la cantidad de filas?

```
$ wc columns.dat -l | awk '{print $1-1}'
```

Uso la expresión anterior en un contexto de ejecución.

```
$ cat columns.dat | tail -n $(wc columns.dat -l | awk '{print $1-1}')
```

Resta combinar todo usando “`xargs`”.

```
$cat columns.dat | awk '{ print $1 }'
       | tail -n $(wc columns.dat -l | awk '{print $1-1}')
       | xargs echo
```

Hacer checkout de todos los archivos modificados en un git

Hacer checkout de todos los archivos modificados en un git

Primero obtengo los archivos que fueron modificados.

```
$ git status | grep 'modified:'
```

Hacer checkout de todos los archivos modificados en un git

Primero obtengo los archivos que fueron modificados.

```
$ git status | grep 'modified:'
```

Luego me quedo solamente con los nombres de los archivos.

```
$ git status | grep 'modified:' | awk '{print $2}'
```

Hacer checkout de todos los archivos modificados en un git

Primero obtengo los archivos que fueron modificados.

```
$ git status | grep 'modified:'
```

Luego me quedo solamente con los nombres de los archivos.

```
$ git status | grep 'modified:' | awk '{print $2}'
```

Por último, los recupero a todos juntos usando “xargs”

```
$ git status | grep 'modified:' | awk '{print $2}' | xargs git checkout
```

Obtener la cantidad de datos distintos de un archivo

```
Name: Juan
Action: run
Name: Pedro
Action: jump
Name: Rodrigo
Action: jump
Name: Matias
Action: jump
Name: Ruben
Action: run
Name: Maria
Action: jump
Name: Lucas
Action: jump
Name: Laura
Action: run
Name: Paula
Action: stop
Name: Laura
Action: jump
Name: Rodrigo
Action: run
Name: Matias
Action: jump
Name: Laura
Action: jump
```

Obtener la cantidad de datos distintos de un archivo

Primero obtengo el dato que quiero contar.

```
$ cat names.txt | grep "^Name:"
```

```
Name: Juan
Name: Pedro
Name: Rodrigo
Name: Matias
Name: Ruben
Name: Maria
Name: Lucas
Name: Laura
Name: Paula
Name: Laura
Name: Rodrigo
Name: Matias
Name: Laura
```

Obtener la cantidad de datos distintos de un archivo

Primero obtengo el dato que quiero contar.

```
$ cat names.txt | grep "^Name:"
```

Ordeno las líneas.

```
$ cat names.txt | grep "^Name:" | sort
```

```
Name: Juan
Name: Laura
Name: Laura
Name: Laura
Name: Lucas
Name: Maria
Name: Matias
Name: Matias
Name: Paula
Name: Pedro
Name: Rodrigo
Name: Rodrigo
Name: Ruben
```

Obtener la cantidad de datos distintos de un archivo

Primero obtengo el dato que quiero contar.

```
$ cat names.txt | grep "^Name:"
```

Ordeno las líneas.

```
$ cat names.txt | grep "^Name:" | sort
```

Filtro por líneas únicas.

```
$ cat names.txt | grep "^Name:" | sort | uniq
```

```
Name: Juan
Name: Laura
Name: Lucas
Name: Maria
Name: Matias
Name: Paula
Name: Pedro
Name: Rodrigo
Name: Ruben
```

Obtener la cantidad de datos distintos de un archivo

Primero obtengo el dato que quiero contar.

```
$ cat names.txt | grep "^Name:"
```

Ordeno las líneas.

```
$ cat names.txt | grep "^Name:" | sort
```

Filtro por líneas únicas.

```
$ cat names.txt | grep "^Name:" | sort | uniq
```

Por último, cuento la cantidad de líneas totales.

```
$ cat names.txt | grep "^Name:" | sort | uniq | wc -l
```

Obtener cantidades de datos agrupados por conjuntos

Ejemplo

```
====
result: 123
a=0
a=0
x=231
====
result: 12344
a=0
a=2
a=3
a=5
a=7
x=123
====
result: 12344
a=0
a=2
a=3
x=512
```

Obtener cantidades de datos agrupados por conjuntos

Primero obtengo los datos útiles.

```
$ cat results.txt | grep -E 'a=|===='
```

Ejemplo

```
====
a=0
a=0
====
a=0
a=2
a=3
a=5
a=7
====
a=0
a=2
a=3
```

Obtener cantidades de datos agrupados por conjuntos

Primero obtengo los datos útiles.

```
$ cat results.txt | grep -E 'a=|===='
```

Luego, me quedo con algo igual que pueda contar.

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }'
```

Ejemplo

```
a
a

a
a
a
a
a

a
a
a
```

Obtener cantidades de datos agrupados por conjuntos

Primero obtengo los datos útiles.

```
$ cat results.txt | grep -E 'a=|===='
```

Luego, me quedo con algo igual que pueda contar.

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }'
```

Cuento los valores iguales usando "uniq".

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }' | uniq -c
```

Ejemplo

```
      1
      2 a
      1
      5 a
      1
      3 a
```

Obtener cantidades de datos agrupados por conjuntos

Primero obtengo los datos útiles.

```
$ cat results.txt | grep -E 'a=|===='
```

Luego, me quedo con algo igual que pueda contar.

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }'
```

Cuento los valores iguales usando "uniq".

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }' | uniq -c
```

Por último, filtro y limpio los resultados.

```
$ cat results.txt | grep -E 'a=|===='
                  | awk -F "=" '{ print $1 }' | uniq -c
                  | grep a | awk '{ print $1 }'
```

Ejemplo

```
2
5
3
```

Ordenar un archivo por un valor numerico dentro de cada línea

Ejemplo

```
10.2.0.1      node1
10.2.0.2      node2
10.2.0.6      node6
10.2.0.19     node19
10.2.0.4      node4
10.2.0.3      node3
10.2.0.10     node10
```

Ordenar un archivo por un valor numerico dentro de cada línea

Genero un índice para ordenar.

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
```

Ejemplo

```
1 10.2.0.1 node1
2 10.2.0.2 node2
6 10.2.0.6 node6
19 10.2.0.19 node19
4 10.2.0.4 node4
3 10.2.0.3 node3
10 10.2.0.10 node10
```

Ordenar un archivo por un valor numerico dentro de cada línea

Genero un índice para ordenar.

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
```

Ordeno utilizando "sort" con la opción "-n".

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
            | sort -n
```

Ejemplo

```
1 10.2.0.1 node1
2 10.2.0.2 node2
3 10.2.0.3 node3
4 10.2.0.4 node4
6 10.2.0.6 node6
10 10.2.0.10 node10
19 10.2.0.19 node19
```

Ordenar un archivo por un valor numerico dentro de cada línea

Genero un índice para ordenar.

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
```

Ordeno utilizando "sort" con la opción "-n".

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
            | sort -n
```

Borro el índice.

```
$ cat hosts | awk -F "node" '{print $2 " " $1 " node" $2 }'
            | sort -n
            | awk '{print $2 "\t" $3}'
```

Ejemplo

```
10.2.0.1    node1
10.2.0.2    node2
10.2.0.3    node3
10.2.0.4    node4
10.2.0.6    node6
10.2.0.10   node10
10.2.0.19   node19
```

Borrar comentarios de un archivo

Borrar comentarios de un archivo

Uso “sed” con patrones de sustitución.

```
$ sed 's/\(.*\)\(#.*\)/\1/g' sources.list
```

- El patrón `\(.*\)\(#.*\)`, se lee como: `(.*)(#.*)`
- La primera parte `(.*)` *matchea* con cualquier expresión.
- La segunda parte `(#.*)` *matchea* con cualquier expresión comenzada por #
- Luego, la primera parte será generada en el remplazo como `\1`
- Mientras que la segunda será `\2`

Borrar comentarios de un archivo

Uso “sed” con patrones de sustitución.

```
$ sed 's/\(.*\)\(#.*\)/\1/g' sources.list
```

- El patrón `\(.*\)\(#.*\)`, se lee como: `(.*)(#.*)`
- La primera parte `(.*)` *matchea* con cualquier expresión.
- La segunda parte `(#.*)` *matchea* con cualquier expresión comenzada por #
- Luego, la primera parte será generada en el remplazo como `\1`
- Mientras que la segunda será `\2`

Resta borrar las líneas que quedaron vacías.

```
$ sed 's/\(.*\)\(#.*\)/\1/g' sources.list | grep -v "^$"
```

- La expresión "`^$`" matchea con la línea vacía.
- La opción “`-v`” genera el resultado inverso.

¡Gracias!

¿Preguntas?

Contenido disponible en:

https://github.com/fokerman/linuxTalks