Linux y HPC para big data

Alex Di Genova

01/09/2025

Linux

Working with text files line by line

- Viewing and Creating Files: *cat, tac, more, less, echo, touch*

- Editing Files: *nano, vi, vim*

- File Content Manipulation:

  - *grep:* Searches for patterns within files; can be used to find specific text.

  - *sed:* Stream editor for filtering and transforming text in a file or input from a pipeline.

  - *awk:* A programming language designed for text processing and manipulation.

  - *paste:* Merges lines of files horizontally.

  - *cut, sort, uniq, tr, wc, head, tail.*

Linux

Working with text files line by line

- Comparing Files:

  - **diff :** Compares files line by line.

- Regular Expressions and Pattern Matching: ***grep, egrep, fgrep, sed, awk***

- Text Processing:

  - *join:* Joins lines of two files on a common field.

  - *split:* Splits a file into fixed-size pieces.

  - *nl* (add line numbers), ***expand*** (tab to spaces)***, unexpand*** (spaces to tab)***, fold y fmt*** (text format).

- File Encoding and Character Conversion: ***dos2unix, unix2dos***

Linux

Working with text files line by line

```
Country,City,AccentCity,Region,Population,Latitude,Longitude
ad,aixas,Aixàs,06,,42.4833333,1.4666667
ad,aixirivali,Aixirivali,06,,42.4666667,1.5
ad,aixirivall,Aixirivall,06,,42.4666667,1.5
ad,aixirvall,Aixirvall,06,,42.4666667,1.5
ad,aixovall,Aixovall,06,,42.4666667,1.4833333
ad,andorra,Andorra,07,,42.5,1.5166667
ad,andorra la vella,Andorra la Vella,07,20430,42.5,1.5166667
ad,andorra-vieille,Andorra-Vieille,07,,42.5,1.5166667
ad,andorre,Andorre,07,,42.5,1.5166667
```

- head worldcitiespop.csv

- wc -l worldcitiespop.csv

- head -100 worldcitiespop.csv > sample_worldcitiespop.csv

- cut -d',' -f1,5 worldcitiespop.csv > country_population.csv

- tail -n +2 worldcitiespop.csv | sort -t',' -k5 -nr | head -10

- cut -d',' -f1 worldcitiespop.csv | sort | uniq | tail -n +2

- awk -F',' 'NR>1 && $5 > 1000000' worldcitiespop.csv

- awk -F',' 'NF > 1 {pop[$1]+=$5} END{ for(c in pop){print c" "pop[c]}}' worldcitiespop.csv | sort -rn -k2,2 | head

- Add country name using join..

Linux

Sed and Awk —- Power tools for editing files

- The basic function of awk is to search files for lines (or other units of text) that contain certain patterns.

    - A Pattern-Matching Programming Language

    - awk 'program' input-file1 input-file2 ...

    - View a text file as a textual database made up of records and fields.

    - Use variables to manipulate the database.

    - Use arithmetic and string operators.

    - Use common programming constructs such as loops and conditionals.

    - Generate formatted reports.

input

input line ← script

output
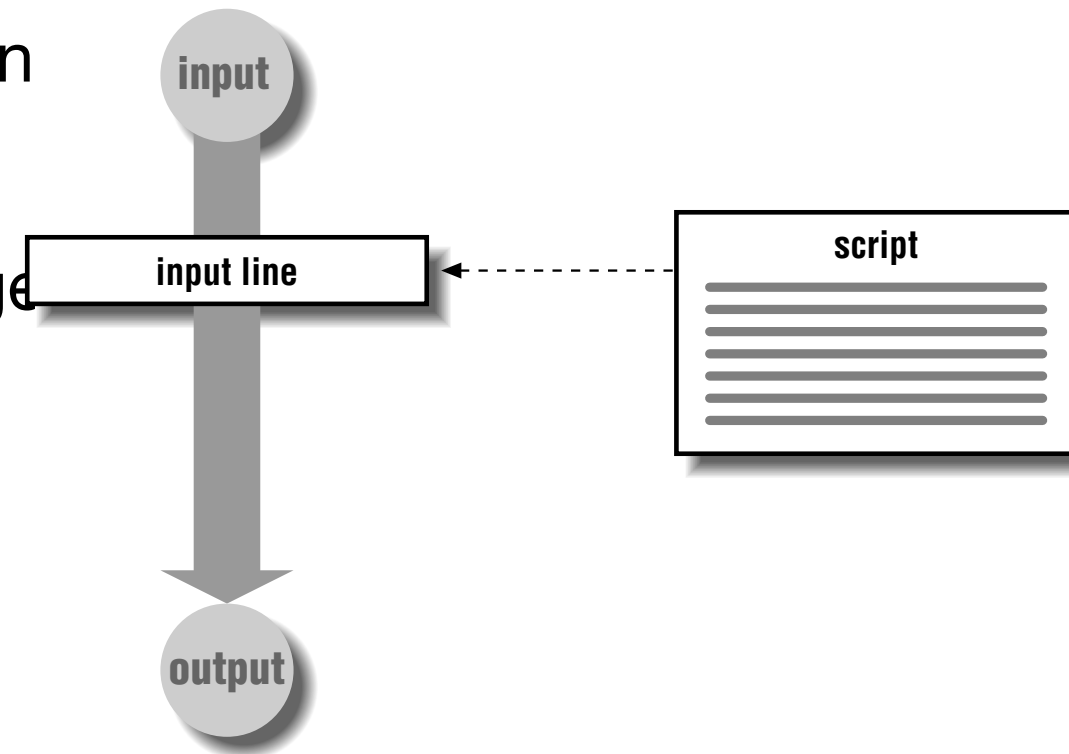
Linux

Sed and Awk —- Power tools for editing files

- Sed

  - Sed is a "non-interactive" stream-oriented editor.

  - To automate editing actions to be performed on one or more files.

  - To simplify the task of performing the same edits on multiple files.
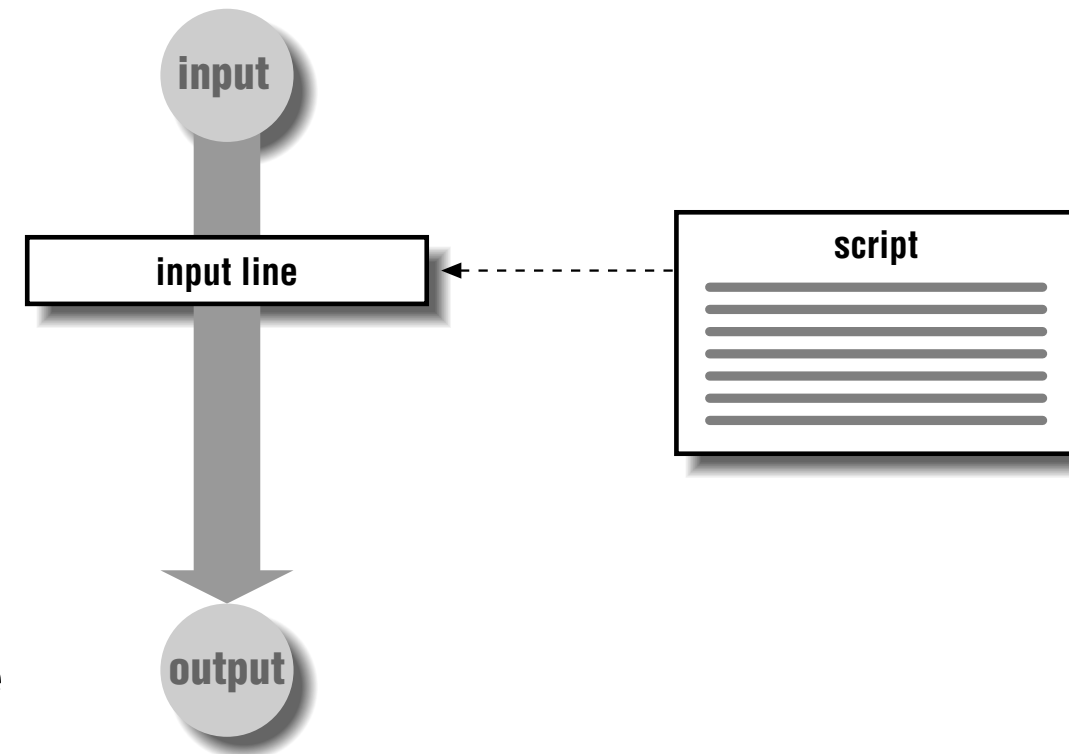
  - To write conversion programs.

input

input line ← script

output

```
sed 's/2022/2024/g' file.txt
sed 's/2022/2024/g ; s/casa/home/g' file.txt

awk '{print $1}' file.txt
awk '{print $1}' *.txt
awk '/perro/ {print $0}' *.txt
```
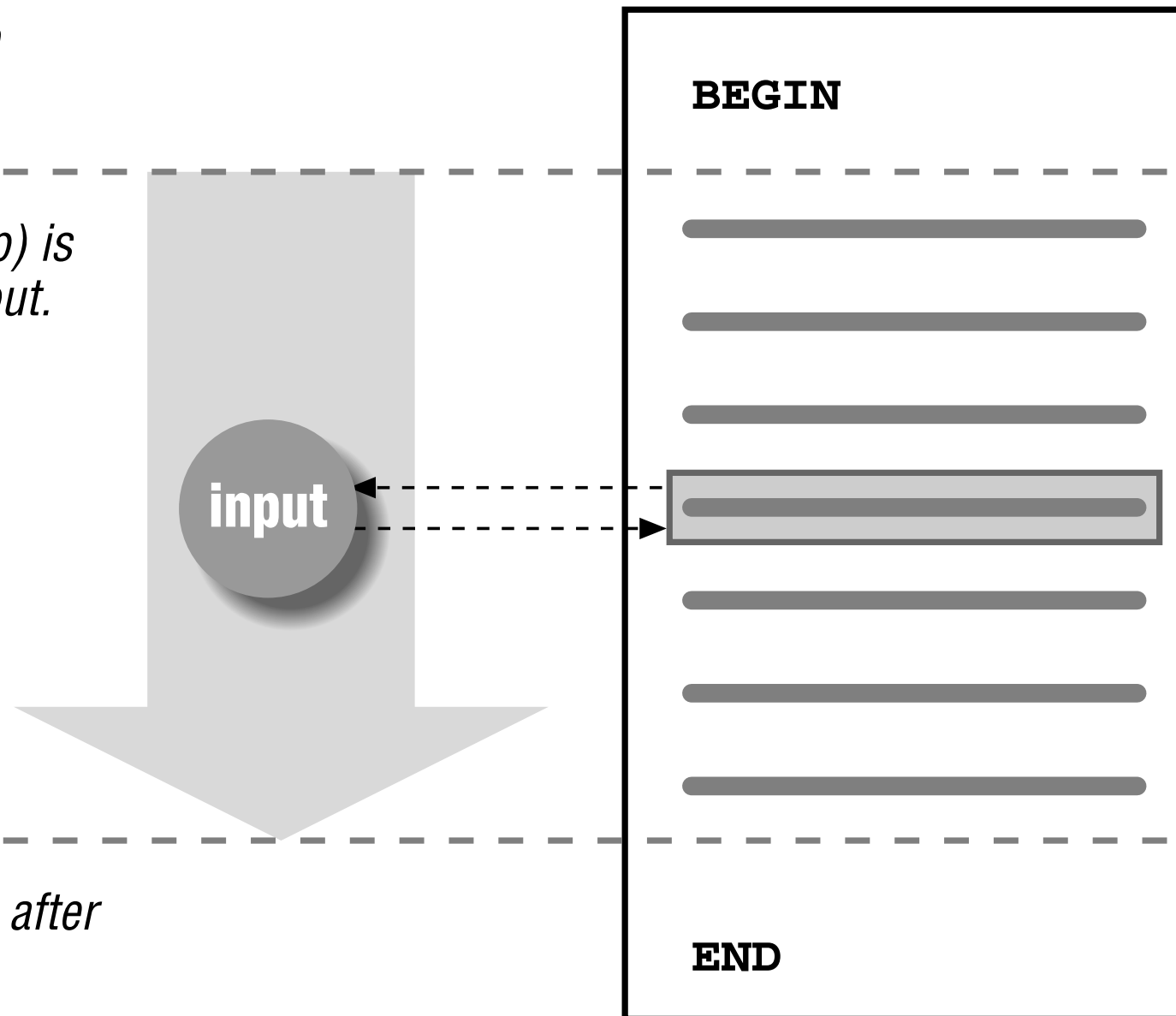
Linux

Awk

1 *1st routine is executed once before any input is read.*

2 *2nd routine (main input loop) is executed for each line of input.*

input

BEGIN

3 *3rd routine is executed once after all input is read.*

END

Linux

Awk

• if (*expression*) *action1*; [else *action2*]

expr ? *action1* : *action2*

grade = (avg >= 65) ? "Pass" : "Fail"

array[*subscript*] = *value*

for (*variable* in *array*)
    do something with array[*variable*

while (*condition*)

    action

for (*set_counter* ; *test_counter* ; *increment_counter*)

    action

Awk Function	Description
cos(x)	Returns cosine of x (x is in radians).
exp(x)	Returns e to the power x.
int(x)	Returns truncated value of x.
log(x)	Returns natural logarithm (base-e) of x.
sin(x)	Returns sine of x (x is in radians).
sqrt(x)	Returns square root of x.
atan2(y,x)	Returns arctangent of y/x in the range -π to π.
rand( )	Returns pseudo-random number r, where $0 <= r < 1$.
srand(x)	Establishes new seed for **rand**( ). If no seed is specified, uses time of day. Returns the old seed.

Linux

Awk

Awk Function	Description
gsub(r,s,t)	Globally substitutes s for each match of the regular expression r in the string t. Returns the number of substitutions. If t is not supplied, defaults to **$0**.
index(s, t)	Returns position of substring t in string s or zero if not present.
length(s)	Returns length of string s or length of **$0** if no string is supplied.
match(s,r)	Returns either the position in s where the regular expression r begins, or 0 if no occurrences are found. Sets the values of **RSTART** and **RLENGTH**.
split(s, a, sep)	Parses string s into elements of array a using field separator sep; returns number of elements. If sep is not supplied, **FS** is used. Array splitting works the same way as field splitting.
sprintf("fmt", $expr$)	Uses **printf** format specification for **expr**.
sub(r,s,t)	Substitutes s for first match of the regular expression r in the string t. Returns 1 if successful; 0 otherwise. If t is not supplied, defaults to **$0**.
substr(s, p, n)	Returns substring of string s at beginning position p up to a maximum length of n. If n is not supplied, the rest of the string from p is used.
tolower(s)	Translates all uppercase characters in string s to lowercase and returns the new string.
toupper(s)	Translates all lowercase characters in string s to uppercase and returns the new string.

Awk

Examples

```
$ cat table.txt
brown bread mat hair 42
blue cake mug shirt -7
yellow banana window shoes 3.14

# print the second field of each input line
$ awk '{print $2}' table.txt
bread
cake
banana

# print lines only if the last field is a negative number
# recall that the default action is to print the contents of $0
$ awk '$NF<0' table.txt
blue cake mug shirt -7

# change 'b' to 'B' only for the first field
$ awk '{gsub(/b/, "B", $1)} 1' table.txt
Brown bread mat hair 42
Blue cake mug shirt -7
yellow banana window shoes 3.14
```

```
awk 'cond1{action1} cond2{action2} ... condN{actionN}'
```

https://learnbyexample.github.io/learn_gnuawk/

Linux
Sed

- Substitutions

  - /Pattern/replacement/flags

    - Flags : n <replace the n-matchin pattern>,g<global>,I <insensitive case>,p <print pattern>

- Transform

  - s/abc/xyz/

  - y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/

- Regular expressions with Sed

  - sed 's/\(.*\):\(.*\)-\(.*\)/\3:\2:\1/' test.txt

```
1:101-201        201:101:1
2:102-202        202:102:2
3:103-203        203:103:3
4:104-204  ───→  204:104:4
5:105-205        205:105:5
6:106-206        206:106:6
```

Linux

Regular expressions

Basic Syntax

- /.../: Start and end
- l: Alternation
- (): Grouping

Character Classes

- \s: Whitespace
- \S: Not whitespace
- \w: Word
- \W: Not word
- \d: Digit
- \D: Not digit
- \x: Hexadecimal digit
- \O: Octal digit

Groups and Ranges

- .: Any character except newline (\n)
- (alb): a or b
- (…): Group
- (?:…): Passive (non-capturing) group
- [abc]: a, b or c
- [^abc]: Not a, b or c
- [a-z]: Letters from a to z
- [A-Z]: Uppercase letters fro
- [0-9]: Digits from 0 to 9

Quantifiers

- *: 0 or more
- +: 1 or more
- ?: 0 or 1
- {3}: Exactly 3
- {3,}: 3 or more
- {3,5}: 3, 4 or 5

Position Matching

- ^: Start of string or start
- $: End of string or end o
- \b: Word boundary
- \B: Not word boundary

Expression

/\b[eSR]\w+/g

Text   Tests NEW

RegExr was created by gskinner.com.

Edit the Expression & Text to see matches. Roll over matches or the expression
flavors of RegEx are supported. Validate your expression with Tests mode.

The side bar includes a Cheatsheet, full Reference, and Help. You can also Save
w patterns you create or favorite in My Patterns.

https://regexr.com/

Linux

Sed & regex examples

```
$ echo 'ac abc abbc abbbc abbbbbbbbc' | sed -E 's/ab{1,4}c/X/g'
ac X X X abbbbbbbbc

$ echo 'ac abc abbc abbbc abbbbbbbbc' | sed -E 's/ab{3,}c/X/g'
ac abc abbc X X

$ echo 'ac abc abbc abbbc abbbbbbbbc' | sed -E 's/ab{,2}c/X/g'
X X X abbbc abbbbbbbbc

$ echo 'ac abc abbc abbbc abbbbbbbbc' | sed -E 's/ab{3}c/X/g'
ac abc abbc X abbbbbbbbc
```

```
$ cat anchors.txt
sub par
spar
apparent effort
two spare computers
cart part tart mart
```

```
# words starting with 'par'
$ sed -n '/\bpar/p' anchors.txt
sub par
cart part tart mart
```

```
# words ending with 'par'
$ sed -n '/par\b/p' anchors.txt
sub par
spar
```

```
$ echo 'fd fed fod fe:d feeeeder' | sed 's/fe*d/X/g'
X X fod fe:d Xer
```

```
# zero or more of '1' followed by '2'
$ echo '31111111111125111142' | sed 's/1*2/-/g'
3-511114-
```

```
# from the first 'b' to the last 't' in the line
$ echo 'car bat cod map scat dot abacus' | sed 's/b.*t/-/'
car - abacus
```

```
$ sed -nE '/two|sub/p' anchors.txt
sub par
two spare computers
```

```
# from the first 'b' to the last 'at' in the line
$ echo 'car bat cod map scat dot abacus' | sed 's/b.*at/-/'
car - dot abacus
```

```
# match 'cat' or 'dog' or 'fox'
# note the use of 'g' flag for mul
$ echo 'cats dog bee parrot foxed' | sed -E 's/cat|dog|fox/--/g'
--s -- bee parrot --ed
```

https://learnbyexample.github.io/learn_gnused/cover.html