Universidad de O'Higgins

INSTITUTO DE CIENCIAS DE LA INGENIERÍA

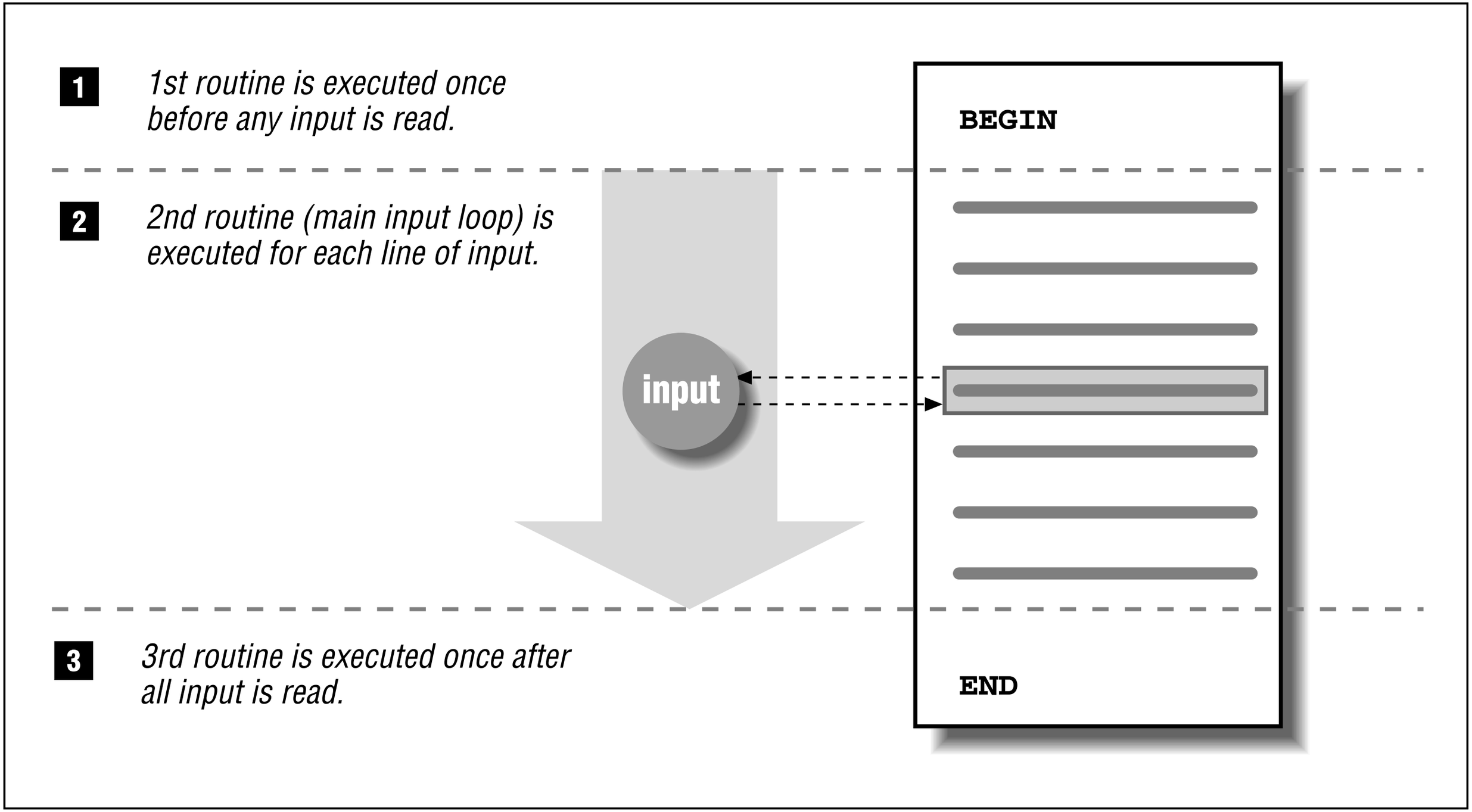
Linux y HPC para big data

Alex Di Genova

01/09/2025

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.
3
3rd routine is executed once after all input is read.
input
BEGIN
END

Linux

Awk

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

expr ? *action1* : *action2*

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

while (*condition*)

action

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

action

array[*subscript*] = *value*

for (*variable* in *array*)

do something with array[*variable*

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.

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
2:102-202
3:103-203
4:104-204
5:105-205
6:106-206
```

→

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

Linux

Regular expressions

Basic Syntax

- /.../: Start and end
- |: 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)
- (a|b): 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

https://regexr.com/

Awk

Operators

Operator	Effect
`lvalue += increment`	Add *`increment`* to the value of *`lvalue`*.
`lvalue -= decrement`	Subtract *`decrement`* from the value of *`lvalue`*.
`lvalue *= coefficient`	Multiply the value of *`lvalue`* by *`coefficient`*.
`lvalue /= divisor`	Divide the value of *`lvalue`* by *`divisor`*.
`lvalue %= modulus`	Set *`lvalue`* to its remainder by *`modulus`*.
`lvalue ^= power`	Raise *`lvalue`* to the power *`power`*.
`lvalue **= power`	Raise *`lvalue`* to the power *`power`*. (c.e.)

Awk

Operators

Expression	Result
x < *y*	True if *x* is less than *y*
x <= *y*	True if *x* is less than or equal to *y*
x > *y*	True if *x* is greater than *y*
x >= *y*	True if *x* is greater than or equal to *y*
x == *y*	True if *x* is equal to *y*
x != *y*	True if *x* is not equal to *y*
x ~ *y*	True if the string *x* matches the regexp denoted by *y*
x !~ *y*	True if the string *x* does not match the regexp denoted by *y*
subscript in *array*	True if the array *array* has an element with the subscript *subscript*

Awk & sed

Problems

1. Count the frequency of words in a text file.
2. Change a hello for bye in the 10-20 lines of a file.
3. Merge two files by a common field.
4. Create a table combining one or more fields from several files.
5. Count 100 most frequent 15-mer of a fasta sequence file.
6. Transpose of a numeric matrix.

Awk and Sed

Solutions

1. **awk** '{for(i=1;i<NF;i++){a[$i]+=1;}}END{for(i in a){print i" "a[i]}}' matrix.txt | sort -nr -k2,2

2. seq 1 100 | sed -n '20,30 p’

3. todo

4. todo

5. **sed** 's/.\{15\}/&\n/g' | **awk** '{if($1 ~/[ACTG]/){a[$1]++}}END{for(i in a) {print i" "a[i]}}' | sort -nr -k2,2 | head -n 30

6. **awk** 'NR == 1 {n=NF; for(I=0;i<NF;i++){row[i]=$i} next;}{if(NF > n) {n=NF} for(i=1; i<NF; i++){row[i]=row[i]" "$i}}END{for(i=1;i<=n; i++) {print row[i]}}' matrix.txt

XSV, Youplot and csvtk

- **xsv** is a command line program for indexing, slicing, analyzing, splitting and joining CSV files.
  - https://github.com/BurntSushi/xsv
- **YouPlot**
  - command line tool that draws plots on the terminal.
- **Csvtk** is convenient for rapid data investigation and easily integrated into analysis pipelines.
  - https://github.com/shenwei356/csvtk