

Sed

- La comanda **sed** és un editor orientat a línia no interactiu.
- El funcionament bàsic d'aquesta comanda és la següent:
 - Rep un text d'entrada (des de l'entrada estàndard o des d'un fitxer)
 - Realitza operacions sobre totes o un subconjunt de les línies de text d'entrada, processant una línia en cada moment.
 - El resultat s'envia a la sortida estàndard.
- Sintaxi:
`sed operació [fitxer...]`
- L'especificació de l'operació a realitzar per la comanda sed té el format següent:
`[adreça[, adreça]] comanda`



Especificació d'adreces

- Les adreces decideixen el rang de línies de text sobre les que s'aplicarà la comanda.
- El rang de línies sobre les que es realitzarà el processament es pot especificar de dues formes:
 - Rang d'adreces.
S'especifica amb la línia inicial i la final del rang (ambdues incloses).
Per referenciar el final de línia s'utilitza un \$.
 - Patró de coincidència.
S'utilitza una expressió regular per decidir el conjunt de línies a processar per la comanda **sed**.
- Exemples:
 - > **sed '1,5d' fich1** # Elimina les cinc primeres línies de fich1
 - > **sed '\$d' fich1** # Elimina l'última línia de fich1
 - > **sed '/^[1a]/d' fich1** # Elimina totes les línies que comencen amb 1 o a
/^[1a]/ és una expressió regular



Comandes “sed” més importants

- Comanda d'impressió (`[rang-adreces]/p`)
Imprimeix les línies seleccionades.
- Comanda d'esborrar (`[rang-adreces]/d`)
Les línies seleccionades d'entrada són eliminades, imprimint la resta de línies per pantalla.
- Comanda de substitució (`[rang-adreces]/s/patró1/patró2/`)
Substitueix la primera instància de patró1 per patró2 en cada línia seleccionada.
- Comanda de lectura d'un fitxer (`[rang-adreces]/r nom_fic`)
- Comanda d'escriptura d'un fitxer (`[rang-adreces]/w nom_fic`)



Exemples sed (I)

Eliminar totes les línies en blanc:

> **sed '/^\$/d' fic**

Imprimir totes les línies des del principi fins la primera línia en blanc:

> **sed -n '1,/^\\$/p' fic**

Substituir la primera ocurrència de la paraula Windows per la paraula Linux en cadascuna de las línies del fitxer:

> **sed 's/Windows/Linux/' fic**

Esborrar tots els espais en blanc al final de cada línia:

> **sed 's/*\$//' fic**

Canvia totes les seqüències d'un o més zeros per un únic 0. La g permet múltiples substitucions en una mateixa línia:

> **sed 's/00*/0/g' fic**



Awk

- **Awk** és un llenguatge de cerca i processament de patrons.
 - Cerca línies en fitxers que contenen certs patrons.
 - Quan es troba un patró en una línia, **awk** hi realitza les accions especificades.
 - Es processen totes les línies d'entrada fins al final del fitxer.
- Execució de programes awk:
 - Si el programa és senzill, és més fàcil incloure'l en la mateixa comanda que executa **awk**, de la forma següent:
`awk 'program' input-file1 input-file2 ...`
 - Quan el programa és llarg, es pot posar en un fitxer i executar-lo amb una comanda així:
`awk -f program-file input-file1 input-file2 ...`



Programes awk

- El programa **awk** consisteix en una sèrie de regles. Cada regla especifica un patró a buscar, i una acció a realitzar quan es troba el patró en el registre (línia) d'entrada.

- Sintaxi:

```
BEGIN { acció }  
patró { acció }      patró pot ser una /Expressió Regular/  
.....  
patró { acció }  
END { acció }
```

- Exemple:

```
# Programa awk que imprimeix el nom dels directoris i  
# fitxers de l'usuari francescsolsonatehas.
```

```
ls -la | awk 'BEGIN{ print "Inici processament" }  
/^d/ { print "Directorio "$9 }  
/^-/ && $3 == "francescsolsonatehas" {print "fitxer Francesc: "$9 }  
END { print "Fi processament"}
```



Entrades programa awk

- Cada línia d'entrada es divideix en camps separats per espais. Cada camp es denomina amb \$1, \$2, etc. \$0 indica tota la línia.
- Es permet la modificació del contingut de qualsevol camp.
- Es pot referenciar els camps amb expressions numèriques com \$i o \$(n+1).

Si l'expressió resultant no és entera, es truncarà la part decimal.

- Exemples:

```
# Sumar els camps 2 i 3 en el camp 1, e imprimir el nou  
# registre.
```

```
awk '{ $1 = $2 + $(2+1); print $0 }' arxiu
```

```
# Renombrar els fitxers que compleixen un determinat patró.
```

```
ls -l patró | awk '{print "mv \"$1\" \"$1\".nou"}' | bash
```



Variables predefinides

Existeixen unes variables predefinides, que poden ser utilitzades dins del programa **awk**:

- **FS**: Separador dels camps d'entrada. Es pot especificar amb l'opció `-F fs` en la línia de comandes
- **RS**: Separador del registre d'entrada (normalment salt de línia).
- **OFS**: Separador del camp de sortida.
- **ORS**: Separador del registre de sortida.
- **NF**: Número de camps en el registre actual
- **NR**: Número de registres (línies) processats fins al moment.



Variables usuari

- awk permet l'ús de variables numèriques i cadenes, definides per l'usuari.
- No és necessari declarar les variables.
- Les variables solen inicialitzar-se dins de la clàusula BEGIN.
- Exemple:

El següent programa mostra la forma en que pot
calcular-se la mida mitja i total dels fitxers d'un usuari

#!/bin/bash

```
ls -l | awk 'BEGIN { print "comencem ..."; total=0 }  
            { total=total+$5 }  
            END { mitja=total/NR; print "quantitat total: " total ;  
                print "mitja: " mitja;  
                print "programa executat ..." }'
```

exit 0



Sentències

- **if** (*expressió*) *estament* [**else** *estament*]
- **while** (*expressió*) *estament*
- **for** (*inicialització ; final ; increment*) *estament*
- **for** (*var in array*) *estament*
- **do** *estament* **while** (*expressió*)
- **break**
- **continue**
- **exit** [*expressió*]



Operaciones

- Aritmètiques:

- $+$, $-$ # Suma i resta
- $*$, $/$ # multiplicació i divisió
- $\%$ # Operador mòdul.
- $++$, $--$ # Operador d'increment i decrement
- $+=$, $-=$, $*=$, $/=$, $\%=$ #Assignacions

- Comparació:

- $<$, $<=$, $>$, $>=$ # Menor, menor igual, major, major igual
- $==$, $!=$ # Igual, diferent
- cadena \sim ExpReg # Veritat si cadena encaixa amb
l'expressió regular ExpReg.
- cadena $!\sim$ ExpReg # Veritat si cadena no encaixa amb
l'expressió regular ExpReg.
- $\&\&$, $||$, $!$ # and, or i negació d'expressions lògiques.



Funcions incorporades

- `length(x)`: Retorna la longitud de l'argument
- `match(s,r)`: Retorna la posició de `s` on passa `r`. Si no existeix retorna 0.
- `substr(s,m,n)`: Retorna la subcadena de `s` que comença en la posició `m` i acaba en la `n`.
- `sub(r,t,s)` Substitueix `t` per la primera ocurrència de `r` en la cadena `s`.
- `index(s1,s2)` Retorna la posició de la cadena `s1` on es troba la cadena `s2`.
- `toupper(s)` Retorna la cadena `s` convertida a majúscules.
- `tolower(s)` Retorna la cadena `s` convertida a minúscules.
- `system(cmd)` Executa la comanda UNIX `cmd` especificada i retorna el seu estat de sortida.
- `rand()` Retorna un número aleatori entre 0 i 1
- Funciones matemàtiques: `sqrt(x)`, `log(x)`, `exp(x)`, `int(x)`, `cos(x)` `sin(x)`, `atan(x)`.



Exemples awk (I)

- Shell script per matar tots els processos amb el nom passat com argument de l'usuari actiu

```
#!/bin/bash
```

```
proces=$1
```

```
ps -ef | \
```

```
awk '$1~/ '$USER'/'&&$8~/ '$procés'/'&& $8!~/awk/ {print $2}' | \
```

```
xargs kill -9
```

- Programa awk que treballa amb vectors i sentències for. Utilitza un array anomenat `línia' per llegir un arxiu complet i després l'imprimeix en ordre invers:

```
#!/bin/bash
```

```
awk '{ línia[NR]=$0 }
```

```
END { for(i=NR;i>0;i=i-1) { print línia[i] } }' pr.txt
```

```
exit 0
```



Exemples awk (II)

- Imprimir el número total de camps de totes les línies d'entrada.
`awk '{ num_camps = num_camps + NF }
END { print num_camps }' fitxer`
- Imprimir totes les línies que tinguin més de 30 caràcters.
`awk 'length($0) > 30 {print $0}' fitxer`
- Utilitzar funciones incorporades per imprimir per pantalla 7 números aleatoris de 0 a 100.
`awk 'BEGIN { for (i = 1; i <= 7; i++)
print 100 * rand() }'`
- Utilitzar funciones incorporades per emmagatzemar en "fitxer" 7 números aleatoris de 0 a 100.
`awk 'BEGIN { for (i = 1; i <= 7; i++)
print 100 * rand() }' > fitxer`



Expressions Regulars

- \ suppress the special meaning of a character when matching.
For example: \\$ matches the character ` '\$`.
- ^ matches the beginning of a string. For example: ^chapter matches the `chapter' at the beginning of a string
- \$ matches the end of a string. For example: p\$ matches a record that ends with a `p`.
- . The period, or dot, matches any single character. For example: .P matches any single character followed by a `P' in a string.
- [...] matches any one of the characters that are enclosed in the square brackets. For example: [MVX] matches any one of the characters `M', `V', or `X' in a string. [0-9] matches any digit. [A-Za-z0-9] matches all alphanumeric characters.



Expressions Regulars

- [^ ...] f.e.: [^0-9] matches any string starting with a number.
- | specifies alternatives. For example: ^P|[0-9] matches any string that matches either `^P' or `[0-9]'. This means it matches any string that starts with `P' or contains a digit.
- (...) used for concatenate regular expressions. For example, `@(samp|code)\{[^}]+\}' matches both `@code{foo}' and `@samp{bar}'.
- * the preceding regular expression is to be repeated as many times as necessary to find a match. For example: ph* applies the `*' symbol to the preceding `h' and looks for matches of one `p' followed by any number (≥ 0) of `h's.
- + similar to `*', but the preceding expression must be matched at least once. For example: wh+y match `why' and `whhy'.



Expressions Regulars

$\{n\}$, $\{n,\}$, $\{n,m\}$ interval expression. If there is one number (n) in the braces, the preceding regular expression is repeated n times. If there are two numbers separated by a comma, the preceding regular expression is repeated n to m times. If there is one number (n) followed by a comma, then the preceding regular expression is repeated at least n times.

$wh\{3\}y$ matches `whhhy`

$wh\{3,5\}y$ matches `whhhy` or `whhhhhy` or `whhhhhhy`

$wh\{2,\}y$ matches `whhy` or `whhhy`, and so on.



Expressions Regulars

`[:alnum:]` alphanumeric characters. F.E.: `/[[:alnum:]]/` matches all alphabetic and numeric characters

`[:alpha:]` alphabetic characters.

`[:blank:]` space and tab characters.

`[:cntrl:]` control characters.

`[:digit:]` numeric characters.

`[:lower:]` lower-case alphabetic characters.

`[:print:]` Printable characters.

`[:punct:]` punctuation characters.

`[:space:]` space characters (space, tabs, etc.)

`[:upper:]` upper-case alphabetic characters.

`[:xdigit:]` hexadecimal digits.