Comandes bàsiques i complexes en Linux

Ll. A.

22 de novembre de 2017

1 Petit manual de comandes de consola...

Comandes:

- `pwd` mostra el directori on estem situats.
- `ls <directori>` mostra el contingut del directori `<directori>` (`ls` mostra el contingut del directori on som en aquell moment). `ls <rang-de-fitxers>` llista el nom de tots els fitxers del rang que existeixin.
- `mkdir <directori>` crea el directori `<directori>`.
- `cd <directori>` canvia de directori (`cd` ens situa al directori `HOME` – `/home/usuari`).
- `cat <fitxer>` mostra el contingut del fitxer per pantalla.
- `less <fitxer>` mostra el contingut del fitxer per pantalla de manera que podem "navegar pel fitxer" endavant i endarrere: podem usar les tecles `[fletxa amunt]`; `[fletxa avall]`; `[fletxa esquerra]`; `[fletxa dreta]`; `[RePág]`; `[AvPág]` i `[q]` per sortir – "quit".
- `cp <fitxer1> <fitxer2>` copia el `<fitxer1>` al `<fitxer2>`
- `mv <fitxer1> <fitxer2>` mou (canvia el nom) el `<fitxer1>` (desapareix) al `<fitxer2>`.
- `rm <fitxer>` borra el fitxer.
- `man <comanda>` mostra el manual de la comanda per pantalla usant el programa `less`.
- `[Ctrl]+[C]` "mata" el programa o comanda que estem executant (per exemple `man` o `less`).

Canviant de directori:

- `.` és el directori actual.
- `..` és el directori a sobre del que estem ara.
- `~` és el directori `HOME`: `/home/usuari` (el caràcter "~" es "fabrica" prement simultàniament les tecles `[Alt Gr]` i `[4]`).

Exemple 1: Si esteu al vostre directori `HOME`: `/home/usuari` (useu `cd` per anar-hi i `pwd` per a comprovar que, efectivament hi sou), les comandes

`cd Desktop` i `cd /home/usuari/Desktop`

tenen el mateix resultat: ens situen al directori `Desktop` mentre que les comandes

`cd`, `cd .`, `cd ~`, `cd Desktop/..` i
`cd /home/usuari/Desktop/..`

no fan res: ens deixen al directori on estàvem (`HOME` – `/home/usuari`).

Exemple 2:

```
cd .
pwd
cd
pwd
cd Treball/prac1/../..
pwd
cd /home/USUARI/Treball/prac1 ; pwd
cd /home/USUARI/Treball/prac1/../.. ; pwd
cd ~ ; pwd
cd $HOME ; pwd
```

1: Que passa quan es posen dues (o més) instruccions a la mateixa línia separades per ';' com al final del bloc d'instruccions anterior?

2: Perquè la primera i la cinquena instruccions són inútils?

3: Trobeu les quatre instruccions `cd` del bloc anterior que són equivalents, *independentment del directori on estigueu situats*. Que fan?

Llistant el contingut dels directoris: Si esteu al vostre directori: `/home/usuari/` useu diverses vegades les comandes useu les comandes

```
ls /home/usuari/Desktop
ls $HOME/Desktop
ls ~/Desktop
```

per llistar el contingut del vostre escriptori.

Variables d'entorn:

- `set` mostra les variables d'entorn que defineixen la nostra sessió, i el seu valor.
- `set | less` mostra les variables d'entorn filtrades pel programa `less` (més fàcils de veure!)
- `echo $<variable>` mostra el valor de la variable `<variable>`.
  Exemples: `echo $HOME`; `echo $PATH`; `echo $USER`

Exemple 3:

```
cd
ls
ls Treball/..
```

```
ls .
ls ..
ls /home/USUARI/Desktop
ls ~/Desktop
ls $HOME/Desktop
ls /home/USUARI/Treball/prac1
ls ~/Treball/prac1
ls $HOME/Treball/prac1
ls /
ls /home
```

Trobeu tots els conjunts d'instruccions equivalents entre les comandes `ls` anteriors, tenint en compte que esteu situats al vostre directori *Home* per la comanda `cd` inicial (pista: n'hi ha 4).

2 Entrada i sortida. Les pipes

Gairebé tots els programes de consola de Linux necessiten unes dades d'entrada i donen algun resultat com a sortida. Per exemple, el programa `ls` no té cap entrada, i dóna com a sortida un llistat de fitxers, i el programa `tar` té com entrada el contingut d'uns fitxers i com sortida un arxiu.

Però hi ha una diferència, ja que `ls` escriu a la pantalla, mentre que `tar` escriu a un fitxer. Aquesta diferència només és aparent, ja que amb les opcions adequades `tar` també donarà la seva sortida a la pantalla (si no posem la opció `f`), però millor no fer-ho, ja que no hi podrem llegir cap cosa interessant.

Com ja s'ha dit a teoria, el programes de consola Linux (gairebé) *sempre* tenen una entrada estàndard (`standard input`—`stdin`), que està connectada al teclat, i una sortida estàndard (`standard output`—`stdout`), que està connectada a la pantalla.

Hi ha una forma comú de connectar la sortida d'un programa a l'entrada d'un altre, fent servir el connector "pipa": `|`. D'aquesta manera es formen unes "pipes" de dades, que passen per l'acció de diferents "motors" (programes).
Nota: el caràcter "`|`" es "fabrica" prement simultàniament les tecles `[Alt Gr]` i `[1]`.

Un primer exemple senzill d'us de pipes: Llisteu el contingut del directori `/usr/bin` amb el programa ls. Hi ha massa fitxers per que els pugueu veure tots! Aquests fitxers són tots programes de Linux: cada un fa coses diferents!! Com podem veure el llistat complet amb calma?

Escriviu la següent ordre: `ls /usr/bin | less`

Una altra forma d'obtenir el mateix resultat, hauria sigut:

```
ls /usr/bin > auxiliar.txt
less auxiliar.txt
rm auxiliar.txt
```

Però hi ha una diferencia fonamental: la segona estratègia, més complicada que la primera, només té sentit si ens interessa tenir un fitxer que contingui la sortida de la primera instrucció. En la primera estratègia no hi ha cap fitxer auxiliar en joc, i llavors no l'hem d'esborrar.

Treballarem diversos exemples perquè pugueu veure la importància de les "pipes". Necessitem un fitxer llarg que podem crear amb la comanda `ls -l /usr/bin > llarg.txt`. Podem tenir el fitxer obert amb la comanda `less` per a veure el seu contingut.

1. Executeu `cat llarg.txt`.
2. Executeu `cat llarg.txt | tac`. Aquesta comanda imprimeix l'entrada a l'invers, des de la última línia a la primera.
3. Executeu `cat llarg.txt | nl`; aquesta comanda re-numera les línies.

4. Executeu `cat llarg.txt | nl | tac` i després `cat llarg.txt | tac | nl`. Com veieu, invertint l'ordre dels factors (els motors) canvia el resultat!

5. Executeu `cat llarg.txt | fmt -40`. A que és interessant? Hem agafat un text i l'hem reescrit en una amplada menor!

6. Executeu `cat llarg.txt | fmt -36 | pr -2`. D'aquesta forma podem formatar un text en dues columnes!!!

Torneu a executar les comandes anteriors afegint `| less` al final, per poder desplaçar-vos més fàcilment pel text de sortida de la comanda.

No només existeixen aquestes comandes, n'hi ha moltes més. A un Linux basat en Debian (per exemple, l'Ubuntu) podeu trobar una llista de comandes bàsiques executant `info coreutils`. Tots aquests programes es poden encadenar i les dades passaran d'un a l'altre com en una cadena de muntatge.

Un altre programa típic que es fa servir amb "pipes", generalment al final, és el `wc` (de *word count*). Executar `man awk | wc` donarà un recompte del nombre de línies, paraules i caràcters que componen el manual de l'`awk`.

No us explicarem totes les comandes possibles, si no que intentarem donar les eines necessàries per a poder connectar programes. Solament explicarem en detall el funcionament de tres programes: el `grep`, el `sort` i l'`awk`.

Una altra forma de "pipa" és la que deriva la sortida d'un programa cap a un fitxer, fent servir el connector "`>`" (o "`>>`" si volem afegir les dades al final del fitxer). També es pot derivar el contingut d'un fitxer cap a l'entrada d'un programa (amb *programa* `<` *fitxer*), encara que és totalment equivalent a `cat` *fitxer* `|` *programa*.

Aplicació:

```
cat <fitxer1> <fitxer2> <fitxer3> > <fitxer4>
```

enganxa el contingut dels tres fitxers `<fitxer1>`, `<fitxer2>` i `<fitxer4>`; i el posa al `<fitxer4>`.

Una altra eina important quan estem fent servir "pipes", és la "T", que és un "canal secundari" pel qual podem tenir una còpia de les dades que estan passant per la "pipa". Com sempre al Linux, el nom d'aquesta eina és senzill, no és més que el nom anglès de la T, `tee`. Fent
...comandes `|` `tee` *UnFitxer* `|` *més comandes*
obtindrem una còpia de les dades, passant per la "pipa", al fitxer que es diu `Un Fitxer` sense afectar de cap forma al funcionament de les altres comandes. Per exemple
`man awk | tee awk.txt | wc`
dóna, com abans, un recompte del nombre de línies, paraules i caràcters que componen el manual de l'`awk`, però a més tindrem una còpia del manual de l'`awk` al fitxer `awk.txt`.

3 Manipulació de dades estil "cutre"

Volem extreure les columnes 1, 4 i 10 del fitxer `Decatlon.dat`. El resultat el volem guardar a un fitxer que s'ha de dir `carreres.dat`. Ho podem fer de diverses maneres usant comandes de consola.

```
head -n 37 Decatlon.dat | tail -n 20 > Dades.dat
cut -d ' ' -f 1 Dades.dat > 1
cut -d ' ' -f 4 Dades.dat > 4
cut -d ' ' -f 10 Dades.dat > 10
paste 1 4 10 > carreres.dat
```

Proveu ara:

```
cut -d ' ' -f 1,4,10 Dades.dat > carreres2.dat
```

Visualitzeu els fitxers `Dades.dat`, `carreres.dat` i `carreres2.dat`. Complementeu la informació sobre les comandes vistes amb l'ajuda (`man head`; `man tail`; `man cut` i `man paste`– també `man split` i `man join`). Interpreteu el que fan les comandes anteriors.

4 Manipulant els permisos:

Es tracta de canviar el permisos del fitxer `guialin.pdf` de manera que els *únics* permisos activats siguin els de lectura i escriptura (esborrat) per l'usuari i el grup (el món no té cap permís i ningú té permís d'execució).

```
chmod a-rwx guialin.pdf
ls -l
chmod u=rw,g=rw guialin.pdf
ls -l
```

Nota: La primera comanda és per modificar el permisos "arreglats" a l'entorn gràfic. Les comandes `ls -l` són per comprovar els canvis de permisos (per obtenir ajuda feu `man ls`).

5 Comandes massives

Convertir n fotos a Grisos:

```
for ff in *.jpg ; do convert $ff -colorspace Gray ${ff%*.jpg}-BN.jpg ; done
```

Reduir la mida d'n fotos:

```
for ff in *.jpg ; do convert $ff -scale 25%x25% ${ff%*.jpg}-Red.jpg ; done
```

Re-nombrat amb re-numeració (la primera instrucció és per saber que no fem un desastre):

```
NuM=0; for ff in *.jpg ; do NuM=$(( $NuM + 1 )) ; echo ${ff%*.jpg}-${NuM}.jpg ; done
NuM=0; for ff in *.jpg ; do NuM=$(( $NuM + 1 )) ; mv -v $ff ${ff%*.jpg}-${NuM}.jpg ; done
NuM=0; for ff in *.jpg ; do NuM=$(( $NuM + 1 )) ; mv -v $ff NomGeneric-${NuM}.jpg ; done
```

6 `sort`, ordenació de dades

A vegades necessitem ordenar unes dades per algun ordre (alfabètic, numèric, etc.). Si aquestes dades es troben a un fitxer de text podem fer servir la comanda de consola `sort`, que vol dir "ordena".

Normalment, `sort` apareix com a filtre (és a dir com a component d'una "pipa" — o bé al final o bé al mig).

L'ús usual de `sort` és el següent:

```
...| sort opcions clau1 clau2 ...| ...
```

on cada *clau* indica la columna o columnes que volem fer servir per ordenar, i les opcions indiquen quin tipus d'ordenació volem. Per exemple, per ordenar un fitxer alfabèticament, però en el qual hi ha una primera columna (per exemple el NIU de una persona) seguida del nom, haurem de fer
`cat fitxer | sort -k2`
Això vol dir que volem una ordenació alfabètica però fent servir de la columna 2 endavant. Per una ordenació numèrica, o bé afegim la opció "`-n` o bé afegim la lletra n després de la clau. Per exemple, per ordenar per la columna 2 numèricament i, quan el valor de la columna coincideix, ordenar per la columna 4, alfabèticament, es pot fer servir
`...| sort -k2,2n -k4,4`

Les lletres per indicar la forma d'ordenar són les següents:

- `n` ordena de forma numèrica. Per exemple, alfabèticament "10" va abans que "9" però numèricament va després.
- `r` ordena a l'invers, per exemple de gran a petit, de desembre a gener o de "Z" a "A", segons les altres lletres posades.
- `b` indica que no s'han de considerar espais en blanc. D'aquesta forma les columnes només contenen les dades, sense els espais que les delimiten.
- `d` indica que s'ha d'ordenar com si fos un diccionari, tenint en compte només lletres, nombres i espais. Altrament, tot caràcter es pren en consideració.
- `f` indica que no s'han de considerar majúscules i minúscules com diferents. Aquest tipus d'ordenació té problemes en algunes instal·lacions de Linux, ja que de vegades no es reconeix que "ç" és la versió minúscula de "Ç".
- `u` `unique`: elimina línies repetides.

Tota lletra pot servir tant com a "opció" per una clau, o com a "opció general", si li afegim un guió abans.

Exemples:
```
ls -l /usr/bin | sort | less
cat Decatlon.dat | grep ^[0123456789] | sort
grep ^[0123456789] Decatlon.dat > filt.dat ; sort filt.dat > ord.dat; less ord.dat
ls -l /usr/bin | grep zip | sort -k 5,5 -r -n
ls -l /usr/bin | grep zip | sort -k 8,8 -r
grep ^[0123456789] Decatlon.dat | sort -k 2,2 -n
grep ^[0-9] Decatlon.dat | sort -k 2,2
```

Exercici: Quines diferències hi ha entre les dues darreres instruccions? Quin significat té l'expressió `[0-9]` a la última instrucció? Quin significat tindria l'expressió `[4-7]`? I l'expressió `[a-z]`?

Pregunta 6.1 *Quin tipus d'ordenació representa la comanda*
`sort -k4,4nr -k2,2f`
i a quin tipus de fitxer de dades pot ser interessant aplicar-la?

Per ordenar un fitxer, en lloc de les dades d'una "pipa", només cal afegir al final de tot el nom del mateix fitxer. De tota forma, el mateix resultat s'obté afegint "`cat fitxer |`" abans de la comanda d'ordenació. D'altra banda, si volem que les dades ordenades vagin cap a un fitxer, ho hem de fer explícitament, amb un "pipejat" cap a un fitxer (`>`).

Pregunta 6.2 *Ordeneu el fitxer d'observacions de lleons respecte el pes del primer exemplar observat (heu d'usar tant un* `grep` *com un* `sort` *a dintre d'una "pipa":*
`cat observ.txt | grep ... | sort ....`

Escriviu quina comanda s'ha de fer servir.

7 `grep` i expressions regulars

`grep` és un programa que agafa un fitxer o bé la sortida del programa anterior (connectat amb l'operador pipa) i només mostra les línies que satisfan una condició, que està donada com a paràmetre.

En primer lloc estudiarem el programa `grep` per a processar la sortida estàndard d'una comanda anterior (és a dir *com a filtre*). Per exemple, podem llistar tots els fitxers de l'ordinador que continguin la cadena "`zip`" al seu nom ho podem fer amb la comanda
`find / | grep zip | less`
(intenteu executar també `find / | grep --color=auto zip`).

Mireu-vos com està fet el fitxer `Decatlon.dat`, per exemple amb `less`, i observeu el resultat de les següents instruccions:

```
ls -l ~ | grep ^d
ls -1 /usr/bin | grep zip
cat Decatlon.dat | grep %
grep % Decatlon.dat
cat Decatlon.dat | grep -v %
grep -v % Decatlon.dat
```

Nota: la opció de la primera instrucció `ls` és una lletra ℓ, mentre que en la segona hi ha un número 1.

La cadena "`*txt`" és un cas particular del que es diu *expressió regular*, que indica diferents paraules o frases. Per exemple, "`*txt`" vol dir "qualsevol cosa que acabi amb `txt`" o, més precisament, "qualsevol cosa seguit de '`txt`'".

La "paraula" buscada pel `grep` també és una expressió regular. Per tant, tot seguit farem un resum de les components d'una expressió regular.

- *•:* Cada caràcter (per exemple `a`) es representa ell mateix.
- *Rangs:* `[acds]` representa qualsevol caràcter en el rang escrit, en aquest cas representa o bé una `a`, o una `c` o una `d` o una `s`. Els rangs seguits es poden indicar amb un guió (`f-m` és el mateix que `fghijklm` Per exemple `[0-9]` representa qualsevol dígit i `[A-La-l]` representa qualsevol caràcter entre `a` i `l`, tant majúscula com minúscula.
  Executeu `find / | grep [w-z]ip` per veure el resultat.
  Si afegim un accent circumflex al principi del rang (`[^a-z]`) representa que NO volem els caràcters del rang.
  A dintre d'un rang, es poden donar conjunts preestablerts, per exemple `[:alpha:]` que és igual a `a-zA-Z`, però altres que no es podrien escriure d'altra manera, com ara `[:space:]`: els espais i tabulacions, o `[:print:]`: tots els caràcters que es poden veure a pantalla. Per exemple, `a[[:space:][:punct:]]a` representa dues lletres `a` separades per un espai o un signe de puntuació.
- *Especials:* Per indicar l'inici de la línia es fa servir `^` (fora d'un rang!) mentre que `$` representa el final de la línia. `\<` i `\>` representen l'inici i el final d'una paraula. Executeu
  `find / | grep zip$`
  per trobar els fitxers que acaben en "zip", i
  `find / | grep "\<zip\>"`
  per trobar els fitxers que es diguin *exactament* "zip".

Repeticions: `?` indica que l'objecte anterior (si cal, entre parèntesis) pot aparèixer o no, però no més d'una vegada. Per exemple `^(go)?ogle` correspon a les línies que comencen per `google` i per `ogle`.
`+` indica que l'objecte anterior ha d'aparèixer una o més vegades. Exemple: `(go)+ogle` correspon tant a `gogoogle` com a `google`, però no a `pirgle`.
`*` indica que l'objecte anterior pot aparèixer o no, i un nombre arbitrari de vegades. Per exemple `g(o)*gle` correspon a `ggle` com a `gooooooooogle`, però no a `gaagle`.
Podem expressar fins i tot la quantitat exacta de vegades que l'objecte ha d'aparèixer. `{5}` volem 5 repeticions. Si hi posem `{5,8}` vol dir que pot aparèixer de 5 a 8 vegades, mentre si posem `{7,}` vol dir que ha d'aparèixer al menys 7 vegades.

Exemple:

- `grep "^cadena"` correspon a l'existència de `cadena` al principi de la línia;
- `grep "cadena$"` correspon a `cadena` al final de la línia;
- `grep "[12b47sdfrq9]"` correspon a qualsevol ocurrència d'un dels caràcters donats;
- `grep "expressio1|expressio2"` correspon a les ocurrències de *al menys una* de les dues expressions;
- `grep -v "expressio"` efectua un filtre al revés, escrivint a la sortida les línies que *no corresponen* al filtre.

Pregunta 7.1 *Què vol dir la següent expressió regular?*

```
^([[:alnum:]]+[[:space:]]*){3}[[:alnum:]][.]?zip\>[[:print:]].*\<(des)?fer
```

Feu exemples de frases que corresponen a l'expressió i que no hi corresponen. En el segon cas, justifiqueu-en la manca de correspondència.

L'explicació completa de les expressions regulars es troba al manual de `grep` *(*`man grep`*).*

Perquè `grep` entengui les expressions regulars en tota la seva potència, és necessari fer servir la opció `-E`, i també les cometes al voltant de l'expressió regular.

L'altra opció important del `grep` és `-v`, que serveix per seleccionar les línies que NO es corresponen amb l'expressió donada. Per exemple `ls | grep -v [aA]` dóna tots els fitxers sense la lletra "a" en el seu nom. Això també ho podeu fer sense fer servir expressions regulars amb la comanda
`ls | grep -vi a`.

Practicant amb `grep`: Pels següents exercicis necessitareu el fitxer `observ.txt`. Creeu un directori anomenat `pract2` al vostre directori d'usuari permanent. Seguidament, baixeu-vos el fitxer auxiliar de la pràctica (`practica2-aux.tar.gz`) des de la plana web de l'assignatura
(`http://mat.uab.es/~infbiotec/`)
i guardeu-lo i descomprimiu-lo al directori que acabeu de crear. Hi trobareu el fitxer `observ.txt` que conté una sèrie d'observacions d'uns investigadors.

1. Llisteu tots els arxius del directori `/usr/bin`, seleccionant i mostrant només els que contenen la paraula `zip` al seu nom. La comanda és `ls /usr/bin | grep --color=auto zip` .
2. El programa `cal` ens mostra el calendari d'un mes o un any arbitrari. Executeu `cal 6 2006` per veure el calendari de juny.
   Seleccioneu la setmana que conté el vostre aniversari del 2030. Per fer-ho, heu d'executar `cal 7 2030 | grep "\<25\>"` si el vostre aniversari és el 25 de juliol.

3. Seleccioneu les línies del fitxer `observ.txt` que contenen la paraula "Lleó" al principi. Això es pot fer de tres formes:
   ```
   cat observ.txt | grep "^Lleó"
   grep "^Lleó" < observ.txt
   grep "^Lleó" observ.txt
   ```
   Nota: A la última forma `grep` no entra en una pipa. Treballa directament sobre un fitxer. De fet és l'ús més habitual del `grep`.

4. De les línies anteriors, seleccioneu les que la segona columna (dia d'observació) és 2:
   ```
   cat observ.txt | grep "^Lleó" | grep -E "[[:alpha:]]+[[:space:]]+2"
   ```
   o bé
   ```
   cat observ.txt | grep -E "^Lleó[[:space:]]+2"
   ```

5. A la primera pràctica dèiem (sense concretar) que el `tar` té més de 150 capítols. Executeu, intentant entendre què està passant, les següents comandes:
   ```
   info tar -o -
   info tar -o - | grep -E "^[*][[:alnum:][:space:]]*:"
   info tar -o - | grep -E "^[*][[:alnum:][:space:]]*:" | wc -l
   ```
   Si en lloc de "`info tar -o -`" feu servir la comanda "`info -f tar.info.gz`" (fent servir el fitxer auxiliar de la Pràctica 1) veureu quina diferència hi ha!

Pregunta 7.2 *Construïu una expressió regular que, aplicada a la comanda*
`ls -l | grep` ***expressió***
doni tots els fitxers que tenen data del mes actual, però de qualsevol any. Executeu aquesta comanda aplicada a qualsevol directori de l'ordinador i bolqueu el seu resultat al fitxer anomenat `llfitxers.txt`*.*

8 `sed` i `tr`

In what follows `standardinput` means (in one line)

```
echo -e "¿Dónde vas, Alfonso XII,\ndónde vas triste de ti?\n\n
Voy en busca de Mercedes\n\nque ayer tarde no la vi."

standardinput
standardinput | tr 'aeiou' 'pqrst'
standardinput | tr 'aeiou' 'pq'
standardinput | sed '{/^$/d; s/vas/vienes/; s/vi/encontré/;}'
standardinput | sed '{/^$/d; s/vas/vienes/; 4s/vi/encontré/;}'
standardinput | sed '{/^$/d; s/vas/vienes/; 4s/vi/encontré/;}' | tr 'aeiou' 'pqrst'
standardinput | sed '{3d; 5d;}'
standardinput | sed 2,3d
standardinput | sed 's!dónde \(.*\) triste de \(.*\)!dónde \2 triste de \1!'
standardinput | tr 'rste' 'i'
standardinput | tr -s 'rste' 'i'
standardinput | tr -d 'rste'
echo "dir1.sbd/dir2.sbd/dir3.sbd/folder" | sed '{ s/ //g; s!\.sbd/!_!g;}' ## For Thunderbird storage
```

Nota: `standardinput` ara fa el paper de l'standard input. `sed` pot actuar com a filtre i pot llegir fitxers com a paràmetre. `tr` solament actua com a filtre.

```
for ff in *.tex ; do
   cp $ff ${ff}.bak ;
```

```
   sed '{s/continuous/discontinuous/g; s/continuity/smooth/g; }' ${ff}.bak > "$ff" ;
done
```

In one line:

```
sed '{ s/January/ Gener /g; s/February/ Febrer /g; s/March/ Març/g;
s/April/Abril/g; s/May/Maig/g; s/June/Juny/g; s/July/Juliol/g;
s/August/ Agost/g; s/September/ Setembre/g;
s/October/Octubre/g; s/November/Novembre/g;
s/December/Desembre/g; s/Mo/Dl/g;
s/Tu/Dt/g; s/We/Dc/g; s/Th/Dj/g;
s/Fr/Dv/g; s/Sa/Ds/g; s/Su/Dg/g }'
```

```
cat acc2tex | iconv -f latin1 -t utf8 | tr '\n\r' '  ' | sed 's! *1,\$!; !g;'
```

```
sed '{ s/á/\\'a/g; s/à/\\`a/g; s/é/\\'e/g; s/è/\\`e/g; s/í/\\'{\\i}/g;
       s/ì/\\`{\\i}/g; s/ï/\\"{\\i}/g; s/ó/\\'o/g; s/ò/\\`o/g;
       s/ú/\\'u/g; s/ù/\\`u/g; s/ü/\\"u/g; s/Á/\\'A/g; s/À/\\`A/g;
       s/É/\\'E/g; s/È/\\`E/g; s/Í/\\'{\\I}/g; s/Ì/\\`{\\I}/g;
       s/Ï/\\"{\\I}/g; s/Ó/\\'O/g; s/Ò/\\`O/g; s/Ú/\\'U/g; s/Ù/\\`U/g;
       s/Ü/\\"U/g; s/ç/\\c{c}/g; s/Ç/\\c{C}/g; s/ñ/\\~n/g;
       s/Ñ/\\~N/g; s/l·l/{\\lgem}/g; }
```

9 awk i manipulació de dades

L'objectiu de l'awk és el d'executar una acció per cada línia de text d'un fitxer (o de l'entrada estàndard) que compleix una condició prefixada. Si no s'especifica cap acció s'imprimeix la línia llegida. Com a conseqüència d'això l'awk *emula* el grep:
awk *'/expressió/'* fa el mateix que grep *expressió*.

Nota: Segons quina instal·lació de Linux s'utilitzi, hi ha diverses versions de l'awk instal·lades. Usualment s'anomenen "awk", "mawk" o "gawk". No hi ha gran diferència entre les diferents versions, a part de que els manuals s'obtenen mitjançant info mawk o amb info gawk.

Recordem l'exemple de les dades del Decatlon.dat Ara volem extreure les columnes 1, 4 i 10 del fitxer Decatlon.dat de manera que quedin ordenades en sentit decreixent respecte de la columna 4 (2ona del resultat) i afegir una quarta columna que doni la suma de les tres columnes anteriors ponderades per pesos 1, 0.6 i 0.83, respectivament. El resultat el volem guardar a un fitxer que s'ha de dir Decatlon-nou.dat. Ho podem fer de diverses maneres usant comandes de consola.

Nota: Cada un dels mètodes que es donen a continuació és una única comanda que cal escriure sense salts de línia. Aquesta única línia (excessivament llarga per l'amplada de la consola) es formata automàticament ocupant aparentment diverses línies. El punt important és que *no s'ha de prémer* Enter fins el final de la comanda.

```
grep "^[0-9]" Decatlon.dat | sort -nr -k 4 | awk '{printf("%d %d %d Suma pondera
da = %f\n",$1,$4,$10,$1+0.6*$4+0.83*$10)}'
```

Nota: Useu man per a obtenir informació sobre grep, sort i awk a la comanda anterior.
Nota 2: Aquí ens hem "oblidat" de crear el fitxer Decatlon-nou.dat.

Mètode 2. Una variant menor que, a més, ja crea el fitxer.

```
cat Decatlon.dat | grep "^[0-9]" | awk '{printf("%d %d %d Suma pondera
da = %f\n",$1,$4,$10,$1+0.6*$4+0.83*$10)}' | sort -nr -k 2 > Decatlon-nou.dat
```

Nota: Useu `cat Decatlon-nou.dat` o bé `less Decatlon-nou.dat` per a visualitzar el resultat.

Mètode 3. Bastant més simple.

```
awk '/^[0-9]/{printf("%d %d %d Suma ponderada = %f\n",$1,$4,$10,$1+0.6*
$4+0.83*$10)}' Decatlon.dat | sort -nr -k 2 > Decatlon-nou.dat
```

Mètode 4. Resultat millor formatat.

```
echo -e "Columnes 1, 4 i 10 de Decatlon.dat\nc1  c4  c10 Suma Ponderada" > Decat
lon-nou.dat ; awk '/^[0-9]/{printf("%d %d %d %.2f\n",$1,$4,$10,$1+0.6*
$4+0.83*$10)}' Decatlon.dat | sort -nr -k 2 >> Decatlon-nou.dat
```

Mètode 5. Una variant mes simple però incorrecta.

```
awk 'BEGIN{printf("Columnes 1, 4 i 10 de Decatlon.dat\nc1  c4  c10 Suma Pondera
da\n");} /^[0-9]/{printf("%d %d %d %.2f\n",$1,$4,$10,$1+0.6*
$4+0.83*$10)}' Decatlon.dat | sort -nr -k 2 > Decatlon-nou.dat
```

Pregunta 5: Perquè la comanda anterior no funciona? Com es pot arreglar?

Nota: La següent comanda, com podeu comprovar, inspeccionant el fitxer `Dades-hor.dat` passa les dades del fitxer `Dades.dat` de columnes a files.

```
for i in $(seq 1 10); do awk -v col=$i '{printf("%d ",$col)}' Dades.dat;
   echo ; done > Dades-hor.dat
```

Consulteu el manual de la comanda `seq` i interpreteu el funcionament de la comanda anterior.

Ara volem escriure, de les línies del fitxer d'observacions, només les que es refereixen a lleons, però només volem saber les dates en que s'han fet les observacions (2na i 3ra columna). Per fer això hem d'escriure

```
awk '/Lleó/ {print $2, $3}' observ.txt
```

Aquest exemple ens mostra el patró general d'una instrucció per l'`awk`, que és:

'/expressió regular/{instruccions}'

i aquest patró general es pot repetir. Per exemple, la següent comanda:

```
awk '/^Lleó/ {print "Lleons:",$3} /^Tigre/ {print "Tigres:",$4}' observ.txt
```

imprimirà per pantalla només les línies que comencin amb "Lleó" o "Tigre" i d'aquestes, respectivament, només la columna 3 o la 4. Com a cas especial, la "columna 0" (`$0` indica tota la línia llegida.

En general, el funcionament de l'`awk` és el següent:

1. Agafa una línia de l'entrada (tan d'una "pipa" com d'un fitxer).
2. Revisa si aquesta entrada coincideix amb alguna de les expressions.
3. Per a cada expressió amb la qual l'entrada coincideix, executa les instruccions corresponents.

Hi ha dues expressions, especials que corresponen a l'inici i el final de l'entrada. Són, respectivament, `BEGIN` i `END`. Aquestes expressions no s'han d'incloure entre signes "`/`" ja que no són expressions regulars.

9.1 Paràmetres

`-f` `program-file`

`--file` `program-file`
Llegeix el programa AWK de `program-file` en comptes de la línia de comandes.

`-F` `fs`

`--field-separator` `fs`
Usa `fs` com a separador de camp d'entrada (determinat per la variable `FS`).

`-v` `var=val`

`--assign` `var=val`
Assigna el valor `val` a la variable `var` abans de començar l'execució del programa.

Exemple: Avantatge: `NFS` i `FD` són variables de `BASH` (i aquesta és la manera de fer-les entrar a un programa `AWK`).

```
awk -v nfs="$NFS" -v FD="$FD" -F '\t'
```

9.2 Expressions de l'awk

A part de la busca d'una expressió regular, l'`awk` pot comprovar condicions molt més complexes. Per exemple, pot veure si una certa expressió regular coincideix amb una certa columna de l'entrada. La comanda
`cat observ.txt | awk '$5 ~ /20/ {print}'`
imprimeix les línies que contenen les lletres "20" a la cinquena columna. A més, es poden fer comparacions, com ara "`$1=="Tigre"`" o "`$4<5`".

També es poden donar condicions sobre el nombre de línies que ja s'han mirat (una mena de comptador) amb la variable `NR` (Number of Rows) i sobre el nombre de columnes de la línia actual, amb la variable `NF` (Number of Fields). L'expressió `$i` fa referència a la columna i del fitxer mentre que `$0` fa referencia a tota la línia (registre) llegida.

Executeu `awk '{print NR, $0}' observ.txt`. Com veieu, `NR` es pot utilitzar a dintre d'una instrucció. Executeu ara `cat observ.txt | awk 'NR==6 {print}'`, veureu que només s'imprimeix la línia 6 (compareu amb la sortida de la comanda anterior).

Magical Mystery: *arrays* associatius
Exemples (per ser explicats detalladament):

```
awk '!seen[$0]++' filename > cleanfilename
awk 'c[$0]++{} END{ for ( r in c ) printf("%s surt %d vegades\n", r, c[r]); }'
```

9.3 Instruccions de l'awk

Per a cada expressió que coincideix amb la línia llegida, s'han de posar una o més instruccions que s'han d'executar. La més senzills és `print`, que simplement imprimeix a la pantalla la línia. També es pot "pipejar" l'escriptura, fent servir els operadors `|`, `>` i `>>` com ja heu fet abans. Per exemple, la

instrucció
`awk '{print $1 >> "primeracolumna.txt"; print $2 >> "segona.txt"}' observ.txt`
crea el fitxer `primeracolumna.txt` que conté la primera columna del `observ.txt` i el `segona.txt` que conté la segona.

Es pot instruir `awk` per que faci moltes més coses. Per exemple, la següent instrucció
`awk 'BEGIN {c=0;p=0} {c=c+length($0);p=p+NF;} END {print c, p, NR}' observ.txt`
escriu el nombre de caràcters, paraules i línies de l'entrada. És a dir, és equivalent a `wc observ.txt`.

En aquest exemple hem fet servir la suma. Podem utilitzar les quatre operacions bàsiques de l'aritmètica (`+ - * /`), i a més el "mòdul" `%`, que dóna la resta de la divisió entera, i la potència `^`. A més, hem introduït les variables "`c`" i "`p`", que anem canviant a cada lectura. A l'hora d'imprimir, una variable s'imprimeix com qualsevol altra cosa, posant-la després d'un `print`. També es poden imprimir cadenes de caràcters posant-les entre cometes.

Hi ha també una sèrie de funcions predefinides que es poden utilitzar. Per exemple:

length: Dóna la llargada del seu argument. Exemples: `var = length($0)` dóna a `var` el nombre de caràcters de la línia actual i `length($3)` dóna la llargada de la tercera paraula de la línia actual.

split: Separa l'argument en trossos amb el separador escollit. Exemple: `split($1,trossos,",")` guarda en `trossos[1]`...`trossos[5]` les diferents parts de `$1` separades pel caràcter ",". Més precisament, si `$1` és `1,34,hola,t1,5666666`, llavors `trossos[1]=1`, `trossos[2]=34`, `trossos[3]="hola"`, `trossos[4]="t1"`, `trossos[5]=5666666`.
`split($0,columna)` ja sabem que fa: aïlla els trossos de `$0`, que són `$1`, `$2`,..., però amb l'avantatge que després podem fer servir la notació `columna[1]` en lloc de `$1`.

Com veieu, un "programa" `awk` es pot allargar i complicar bastant. Tenim la possibilitat d'*automatitzar* l'`awk` fent que llegeixi les instruccions d'un fitxer, de forma que no cal escriure-les cada vegada. Això es fa amb la opció `-f`. Per exemple, feu servir la instrucció de l'`awk` que trobeu al material auxiliar: `awk -f filtre.awk observ.txt` on `filtre.awk` és:

```
#!/bin/awk
# aquest és un comentari!
# però la primera línia és important, perque ajuda a l'editor a reconeixer el tipus de fitxer
# i colorejar-ho bé!!!!
# sempre que hi hagi un caràcter  # el text següent no conta.
BEGIN   { print "Resultat del filtratge del fitxer"; eleconta=0; tigrconta=0; }
/^[^-]/ {                       # selecciona les  línies que no comencen per "-"
         suma=$5+$6+$7+$8+$9+$10; # calcula la suma de les columnes de 5 a 10
         print "El pes total dels exemplars de la espècie " $1 " avui és " suma; # escriu coses
        }
/^Ele/  {
          eleconta=eleconta+NF-4; # afegeix al nombre d'elefants el de columnes
                                  # (menys les 4 primeres que no conten)
          print eleconta " elefantes se balanceaban sobre la tela de una araña..." ; # canta una cançó
          print "y como veían que no se caían fueron a llamar a otro elefante!!";
        }
/^Tigr/ { tigrconta=tigrconta+1; } # conta totes les vegades que observem tigres
END     { print "S'han observat Tigres en " tigrconta " dies diferents!" }
```

Obriu el fitxer d'instruccions `filtre.awk` amb l'editor.

Pregunta 9.1 *Quin efecte creieu que té l'execució d'aquestes instruccions sobre el fitxer de dades? Comenteu amplament el fitxer.*

Com veieu: múltiples instruccions per al mateix bloc s'han de dividir mitjançant signes `;` i es poden posar comentaris, iniciant la línia amb un caràcter `#`.

Pregunta 9.2 *Modifiqueu el fitxer* `filtre.awk` *per tal que imprimeixi la mitjana dels pesos dels exemplars observats per a cada grup. Guardeu el resultat com a* `mitjanes.awk`*. Podeu suposar que no hi ha més de 10 columnes.*

Suggeriment: *el nombre d'exemplars, per a cada línia, es troba a partir de la variable* `NF`*, restant-li el nombre de columnes que no contenen dades d'observació, el nom de la espècie i la data.)*

Pregunta 9.3 *Modifiqueu* `filtre.awk` *per que imprimeixi al final el nombre d'exemplars observats per a cada espècie.*

Suggeriment: *les variables "indexades" són molt útils en aquest cas. Feu servir una variable* `nombre[]`*, i afegiu el nombre d'observacions per a cada línia, a l'índex* `nombre[$1]`*. Per imprimir tota una variable indexada, es fa servir la construcció* `for (i in nombre) print i ": " nombre[i]` *(el fet de separar índex i valor amb* `": "` *és arbitrari, podeu imprimir el que voleu, per exemple* `print "tipus " i " - quantitat: " nombre[i]`*).*

Guardeu el resultat com a `totals.awk` *i la sortida de* `cat observ.txt | awk -f totals.awk` *a* `totals.txt`

Finalment, es poden fer servir estructures de control, que executen instruccions una o més vegades segons unes condicions. La més senzilla és l'`if`, que executa la instrucció només si la condició és certa:
`if ($4>$5) print $1`
imprimeix la primera columna si la quarta té un valor més gran que la cinquena.
`if (a==b) signe=0 else if (a>b) signe=1 else signe=-1` assigna a la variable `signe` un valor diferent, segons el resultat de comparar `a` i `b`.

Un altra estructura, que és com un `if` amb repetició, es el `while`. Amb aquesta estructura es pot repetir moltes vegades un conjunt d'instruccions:
`split($0,columnes); i=5; while (i<=NF) {pes=pes+columnes[i]; i=i+1};`
suma els pesos del fitxer d'observacions, sense importar quantes columnes d'observació hi hagi.

Finalment, el `for`, com acabem de veure, serveix per recórrer una variable indexada, i executar instruccions par a cada índex.