Janvier 2021

MU4BM748

TP : Les filtres grep/sed/awk

Notions de cours (Aide mémoire)

Un **filtre** est une commande unix capable de lire un flux d'entrée (fichier par exemple), d'effectuer des actions sur ce flux (sans modifier le fichier d'origine) et d'afficher en sortie (flux de sortie) le résultat de ces actions.

Dans du texte, un **motif** peut être représenté par une **expression régulière (RE ou ER).** Les expressions régulières **basiques** (BRE) fonctionnent dans tous les cas, les expressions régulières **étendues** (ERE) dans certains cas seulement. Ainsi les BRE sont la syntaxe par défaut des filtres *sed* et *grep* alors que pour utiliser les ERE avec sed et grep, il faut utiliser l'option -E.

Remarque : si vous utilisez macOSX, le sed installé n'est pas celui de GNU. On peut donc constater quelques différences (minimes) avec le GNU sed.

https://www.shellunix.com/regexp.html

http://www.grymoire.com/Unix/Regular.html#uh-1

Regular Expression	Class	Type	Meaning
.	all	Character Set	A single character (except newline)
^	all	Anchor	Beginning of line
$	all	Anchor	End of line
[ ] et [-] [^]	all	Character Set Exclusion	Range of characters Exclusion
*	all	Modifier	zero or more duplicates
\<	Basic	Anchor	Beginning of word
\>	Basic	Anchor	End of word
\(..\)	Basic	Backreference	Remembers pattern
\1..\9 et &	Basic	Reference	Recalls pattern
+	Extended	Modifier	One or more duplicates
?	Extended	Modifier	Zero or one duplicate
\{M,N\}	Extended	Modifier	M to N Duplicates
(...|...)	Extended	Anchor	Shows alteration

Quelques exemples :

```
[A-Za-z0-9] # majuscule ou minuscule ou chiffre
[a-d5-8X-Z] # a, b, c, d, 5, 6, 7, 8, X, Y ou Z
[a-d5-8X-Z]* # 0 à n occurrences de a, b, c, d, 5, 6, 7, 8, X, Y ou Z
[a-d5-8X-Z]+ # (ERE) 1 à n occurrences de a, b, c, d, 5, 6, 7, 8, X, Y ou Z
```

```
^[0-9][0-9]*$ # ligne qui commence et se termine par au moins 1 chiffre = ne contient que des chiffres et au moins 1 (non vide)

.* # 0 à n occurrences de n'importe quel caractère = n'importe quelle chaîne

(toto|titi) # (ERE) chaîne toto ou chaîne titi

[A-Z]{5,9} # une chaîne constituée uniquement de 5 à 9 majuscules

[^A-Z]{5,9} # une chaîne constituée de 5 à 9 caractères qui ne sont pas des majuscules
```

Remarques importantes :

- Pour éviter l'interprétation des RE par le shell (caractères `*`, `.`, `>`, `<`), on notera les RE entre quotes simples `''` dans les filtres.
- Pour empêcher l'interprétation d'un caractère spécial dans une RE (par exemple si on cherche un . ou un ^), on le précède d'un `\`.

Le filtre *`grep`* permet de rechercher des lignes contenant un motif (utiliser l'option -E pour les ERE). Consulter le manuel de grep (*`man grep`*) pour les options utiles telles que -c,-C,-A,-B,-v,-i,-f, -h,-n, etc.

Le filtre *`sed`* permet essentiellement de supprimer des lignes (commande d) contenant un motif ou de substituer des motifs dans des lignes (commande s). Les ER permettent aussi d'adresser les lignes à supprimer ou à substituer. L'option -n associé au drapeau p permet de n'afficher que les lignes modifiées.

Le filtre *`awk`* permet de traiter la ligne par champ (colonne) et de réaliser des programmes pour les traiter (variables, tests, boucles, etc.)

Quelques commandes utiles au TP:

`wc -l test.txt` compte le nombre de lignes du fichier *test.txt*

`cat test.txt` affiche le contenu du fichier *test.txt* à l 'écran (utiliser *`more`* ou *`less`* si les fichiers sont longs)

`*` désigne tous les fichiers du répertoire courant (Attention : signification différente dans le cadre d'une ER)

`tr -d "A" < test.txt` supprime tous les caractères « A » du fichier *test.txt*

`tr "A" "a" < test.txt` remplace tous les caractères « A » du fichier *test.txt* par « a »

Autres informations utiles :

L'encodage des fichiers texte traité par les filtres est en ASCII. Tous les caractères sont encodés mêmes ceux qui ne sont pas visibles. Ainsi un saut de ligne est un « \n », une tabulation est un « \t », un espace est un « vrai » blanc, etc. Pour visualiser l'encodage de ces caractères « invisibles », vous pouvez utiliser la commande *`od -c test.txt`* .

Certains filtres acceptent l'usage de *`\t`* et/ou *`\n`* dans leurs ER d'autres pas (voir à l'usage). Certains problèmes ont parfois lieu avec l'usage de la touche « tabulation » qui est surchargé par le terminal pour enclencher la complétion. *`ctrl+v+tab`* permet de alors taper une « vraie » tabulation sous Linux.

Exercices

Les fichiers nécessaires aux exercices sont sur le site Moodle du module.

Janvier 2021

MU4BM748

1ère partie

Trouver les commandes permettant de répondre aux questions suivantes en utilisant `grep` ou `sed`.

1. Comment peut-on caractériser une ligne vide par une expression régulière ? Proposer une commande qui compte combien il y a de lignes vides dans le fichier *allseq.fasta* ? Tester votre commande sur les fichiers *allseq2.fasta* et *allseq3.fasta*. Que constatez-vous ? Utiliser la commande `od -c` pour visualiser les caractères non visibles.
   NB : on peut appliquer un filtre sur plusieurs fichiers à la fois. Par exemple `grep 'A' *.txt` *recherche la lettre A dans tous les fichiers .txt*

```
grep -c '^$' allseq.fasta # trouve les lignes vides = contenant un saut de ligne
grep -c '^[ ]*$' allseq.fasta # trouve aussi les lignes contenant des espaces
grep -c '^[     ]*$' allseq.fasta # trouve aussi les lignes contenant des
tabulations
grep -c '^[      ]*$' allseq*.fasta # trouve aussi les lignes contenant des
tabulations et/ou des espaces
grep -c '^\s*$' allseq*.fasta # équivalent à si dessus
```

2. Proposer une commande qui supprime les lignes vides des fichiers *allseq.fasta*.

```
sed '/^[         ]*$/d' allseq*.fasta
sed '/^\s*$/d' allseq*.fasta # uniquement si GNU sed (pas sur macOSX qui ne
reconnaît pas \s)
```

3. (a) Proposer une commande qui compte combien il y a de séquences dans les fichiers *allseq.fasta* ? Que se passe-t-il si on oublie les quotes ? (b) Que pensez-vous de la commande suivante :

```
grep -v '^[A-Z]*$' allseq*.fasta
```

```
grep '>' allseq*.fasta | wc -l
grep -c '>' allseq*.fasta
# si on oublie les quotes, le caractères > est interprété par le shell or la
redirection est antérieure à la commande, le fichier est donc écrasé

grep -v '^[A-Z]*$' allseq*.fasta identifie aussi les lignes contenant des blancs
ou des tabulations
allseq.fasta:3
allseq2.fasta:4
allseq3.fasta:5
```

4. Remplacer tous les acides aminés « G » par « g ». Proposer une commande `sed` et une commande `tr`.

```
tr "G" "g" < allseq.fasta > test1.fasta
sed 's/G/g/g' allseq.fasta > test2.fasta
cmp test1.fasta test2.fasta
```

5. Refaire la commande pour l'acide aminé « Q » qu'on souhaite remplacer par « q ». Que constatez-vous ? Adapter votre commande en conséquences.

```
sed 's/Q/q/g' allseq.fasta  # remplace aussi le «Q» de SEQ
# il faut adresser les lignes dans lesquelles on souhaite faire la substitution
(impossible avec tr)
```

```
sed '/^[A-Z]/s/Q/q/g' allseq.fasta
sed '/^>/! s/Q/q/g' allseq.fasta
# impossible avec tr d'adresser les lignes à modifier
```

6. Encadrer, dans les séquences, les acides aminés hydrophobes (P,G,A,V,L,I,M,F,W) par des étoiles.

```
sed 's/[PGAVLIMFW]/*&*/g' allseq.fasta
# impossible avec tr de rappeler la lettre captée
```

7. Dans le fichier `nucleolar-IPI-719.txt`, rechercher (en plus de la ligne contenant l'identifiant) les 2 lignes qui précèdent « IPI00216158 », puis les 2 lignes qui suivent et enfin à la fois les 2 qui précèdent et les 2 qui suivent. *Indication : utiliser les options -A,-B et -C de grep.*

```
grep -B 2 'IPI00216158' nucleolar-IPI-719.txt
grep -A 2 'IPI00216158' nucleolar-IPI-719.txt
grep -C 2 'IPI00216158' nucleolar-IPI-719.txt
```

2ème partie

Le fichier *Taxonomy.liste* contient les taxonomies associées à différentes espèces (ou souches) ainsi que l'identifiant associé selon la nomenclature du NCBI.

<u>En utilisant sed :</u>

1. Afficher les lignes 10 à 20 du fichier. Vérifier que vous n'affichez bien qu'une fois chaque ligne.

```
sed -n '10,20p' Taxonomy.liste
Rq : sed '10,20p' Taxonomy.liste imprimerait 2 fois les lignes concernées.
```

2. Remplacer **toutes** les occurrences (il y en a parfois plusieurs par ligne) du mot « bacteria » ou « Bacteria » par « BACTERIA » même si ce mot fait partie d'un mot plus grand. Faire afficher uniquement les lignes qui ont été modifiées.

```
sed -n 's/[Bb]acteria/BACTERIA/pg' Taxonomy.liste
```

3. Avec une seule commande sed, supprimer à la fois les taxonomies concernant les Eukaryotes et celles concernant les Virus. Supprimer aussi du même coup la ligne d'entête « #TaxID Taxonomy ». *Indication : utiliser l'option -e de sed pour chaîner les commandes sed*

```
sed -e '/Eukaryota/d' -e '/Viruses/d' -e '/^#/d' Taxonomy.liste
sed '/Bacteria/! d' Taxonomy.liste
```

<u>En utilisant grep :</u>

1. Identifier les séquences qui contiennent un « . » ? Vous pouvez utiliser l'option `-o` de `grep` pour visualiser ce que votre motif capte dans la ligne.

```
grep '.' Taxonomy.liste  donne toutes les séquences car « . » est un caractère spécial.
grep -c '\.' Taxonomy.list # il y en a 12
grep -o  Taxonomy.list pour visualiser uniquement le motif recherché
# Attention : Le « . » est aussi un caractère spécial du shell bash (indique le répertoire courant)
```

2. Afficher toutes les lignes ne contenant pas « bacteria » avec leur numéro de ligne dans le fichier d'origine ? *Indication : utiliser les options -v et -n de grep.*

```
grep -nv 'bacteria' Taxonomy.liste
```

3. (a) Parmi **tous les fichiers** du TP, afficher les lignes des fichiers contenant les terme « bacteria », supprimer les noms de fichiers. (b) Afficher la liste des fichiers dans lesquels au moins une ligne contient le terme « bacteria ». *Indication : utiliser les option -l et -h de grep.*

```
grep 'bacteria' *  # affiche fichier:ligne
grep -h 'bacteria' * # affiche lignes
grep -l 'bacteria' * # affiche fichier
```

En utilisant awk :

1. Le filtre *awk* peut diviser les lignes en champs. Quel est le séparateur par défaut de *awk* ? Afficher uniquement pour chaque ligne le domaine du vivant concerné (Viruses, Eukaryota ou Bacteria). Penser à ne pas traiter la 1ère ligne. Supprimer a posteriori le « ;» en chaînant une autre commande (avec un « | »). Si vous êtes curieux, essayer de trouver une commande *cut* pour faire la même chose.

```
# Le séparateur par défaut est tabulation et espace
awk '$0 !~ /^#/ {print $2}' Taxonomy.liste | sed 's/;//'
awk '$0 !~ /^#/ {print $2}' Taxonomy.liste | tr -d ";"
tail -n+2 Taxonomy.liste | cut -f 2 | cut -d';' -f 1
# cut utilise la tabulation comme séparateur par défaut
# tail -n+2 supprime la 1ère ligne
```

3ème partie

Le fichier *PLCplasmidiques.fasta* contient des séquences protéiques de Phospholipases C qui ont 4 origines différentes : sp (pour swissprot), tr (pour trembl), protein:plasmid pour plasmid_db et uniprot. On souhaite comptabiliser le nombre de séquences provenant de chacune de ces 4 banques. On obtiendra une sortie du type :

#banque nombre
sp 10
uniprot 1
tr 10
plasmid_db 2

1. Afficher les séquences dont les descriptions contiennent « plasmid ». Est-ce une bonne manière d'identifier les séquences provenant de la base protéique « *plasmid* » (autre que *sp,tr* ou *uniprot*) ?

```
grep 'plasmid' PLCplasmidiques.fasta # donne aussi des séquences uniprot
(contient « megaplasmid »
grep '\<plasmid\>' PLCplasmidiques.fasta
grep '>protein:plasmid' PLCplasmidiques.fasta
```

2. Pour ces séquences, remplacer l'entête « >protein:plasmid: » par « >plasmid_db| » pour rendre le format plus homogène. Rediriger le résultat obtenu dans un nouveau fichier.

```
sed '/^>/s/protein:plasmid:/plasmid_db|/' PLCplasmidiques.fasta >
PLCnoplasmides.fasta
```

3. Avec awk et un test, afficher puis compter le nombre de séquences issues de la base « sp ».

```
# comptage avec wc
awk -F'|' '{if($1==">sp"){print $0}}' PLCplasmidiques.fasta | wc -l
# comptage avec variable awk
awk -F'|' '{if($1==">sp"){sp++}}END{print sp,"séquences sp"}'
PLCplasmidiques.fasta
```

4. (a) Avec awk et les dictionnaires, comptabiliser les séquences pour chaque banque. Rediriger ce résultat dans un fichier comme décrit ci-dessus. (b) Proposer une commande quasi-équivalente avec `sed`, `cut, sort` et `uniq -c`

```
awk -F"|" '/^>/ {database[$1]++} END {for (i in database) {print
i,database[i]}}' PLCnoplasmides.fasta | tr -d ">"

awk -F'[|>]' '/^>/ {database[$2]++} END {for (i in database) {print
i,database[i]}}' PLCnoplasmides.fasta

awk 'BEGIN {FS="[|>]"; print "#banque","nombre"} /^>/ {database[$2]++} END {for
(i in database) {print i,database[i]}}' PLCnoplasmides.fasta > effBanque.txt

sed -n '/^>/p' PLCnoplasmides.fasta | cut -d '|' -f 1 | tr -d ">" | sort | uniq
-c
```

5. Utiliser awk sur le fichier obtenu à la question précédente afin de compter le nombre total de séquences.

```
awk 'BEGIN{som=0} {if($0 !~ /^#/) {som+=$2}} END {print "Total :
",som,"sequences"}' effBanque.txt
```

Facultatif

Toujours sur ce même fichier de séquences, on souhaite remplacer toutes les zones de faible complexité (suite de D et de E) par des « @@@ ».

1. *Il faut d'abord mettre chaque séquence sur une seule ligne. Attention à la gestion de la première et de la dernière séquence*

Indication : dans awk, s=st concatène les chaînes s et t

```
awk 'BEGIN {s=""} /^>/ {if (s!="") {print s; s=""}; print $0} !/^>/ {s=s$0} END
{print s}' PLCplasmidiques.fasta
```

2. *Puis remplacer une suite d'au moins 3 E ou D par « @@@ »*

Indication : utiliser la fonction gsub() de awk dans la commande précédente ou une commande sed

```
# on utilise grep --color '@@@' pour visualiser les modifications

awk 'BEGIN {s=""} /^>/ {if (s!="") {print s; s=""}; print $0} /^[A-Za-z]/
{gsub("[ED][ED][ED]+","@@@",$0);s=s$0} END {print s}'  PLCplasmidiques.fasta |
grep --color '@@@'

awk 'BEGIN {s=""} /^>/ {if (s!="") {print s; s=""}; print $0} !/^>/ {s=s$0} END
{print s}'  PLCplasmidiques.fasta | sed -E 's/[ED][ED][ED]+/@@@/g'| grep --color
'@@@'
```

4ième partie : Traitement du fichier Genbank U00089.gbk (Facultatif)

Après avoir pris le temps d'observer et de comprendre le format de ce fichier, trouver une commande qui permet d'extraire pour chaque CDS les coordonnées de celui-ci sur le chromosome (début et fin) ainsi que son sens (1 pour « forward », 2 pour « reverse »). Notez qu'il y a 5 espaces avant le nom de chaque feature (dont la feature CDS).

On devra obtenir un fichier du type :

692 1834 2
1838 2767 2
2869 4821 2
4821 7340 2
7312 8574 2
...

Indication : utiliser les fonctions gsub() et split() de awk

```
cat U00089.gbk | egrep '^     CDS' | awk '{if ($2~"complement") {s=1} else
{s=2}; {print $2,s}}'

cat U00089.gbk | egrep '^     CDS' | awk '{if ($2~"complement") {s=1} else
{s=2}; gsub("[a-z()]","",$2);split($2,t,".");{print t[1],t[3],s}}'
```

Compter les différentes « features » du fichier Genbank. On obtient le résultat suivant :
misc_RNA 4
rRNA 3
source 1
CDS 688
tRNA 37
gene 694
repeat_region 40

```
# le pb est de gérer la séquence après  le mot clé ORIGIN qui débute par un
chiffre après des blancs
# il y a aussi le pb de source avant le mot clé FEATURES

 awk 'BEGIN {ft=0;} /^     [^ ]/ && (ft == 1) {table[$1]++} /^FEATURES/ {ft =
1;} /^ORIGIN/ {ft=0;} END {for (i in table) print i,table[i];}' U00089.gbk
```

Janvier 2021

MU4BM748