# TP 3 — Réseaux, traitement de caractères et scripts de *shell*

César Rodríguez et Hedi Benzina
rodriguez@lsv.ens-cachan.fr
benzina@lsv.ens-cachan.fr

12 décembre 2011

## 1 La commande `wget`

La commande `wget` permet le téléchargement de fichiers sur l'Internet. La syntaxe d'invocation est très simple : `wget [OPTIONS] URL`.

### 1.1 Exemples

1. `$ wget "http://www.ens-cachan.fr/robots.txt"`
   Télécharger le fichier `robots.txt` au répertoire actuel. Il est important de *protéger* la URL avec des guillemet afin que certains caractères (comme `&` ou `?`) ne soient pas interprétés par la *shell*.
2. `$ wget -O r.txt "http://www.ens-cachan.fr/robots.txt"`
   Stocker le fichier sur `r.txt` au lieu de `robots.txt` [1].
3. `$ wget -O - "http://www.ens-cachan.fr/robots.txt"`
   Ne pas stocker le fichier mais l'écrire sur la sortie standard.

### 1.2 Exercices

1. Utiliser l'option `-O` pour stocker sur le fichier `f.csv` le contenu de

   `http://www.lsv.ens-cachan.fr/~rodriguez/teaching/linux/freq.csv`

   *Solution*:

## 2 Afficher du texte avec `head`, `tail` et `less`

La commande `head -n N` affiche sur sa sortie standard les premières `N` lignes de texte qu'elle lit sur son entrée standard ; `tail` fait de même avec les dernières `N` lignes. La commande `less` permet de visualiser interactivement son entrée. Taper espace pour avancer une page, utiliser les flèches pour avancer ou rétrocéder une ligne et taper `q` pour finir.

### 2.1 Exemples

1. `$ head < f.csv`, ou bien `cat f.csv | head`.
   Afficher les premières 10 lignes de `f.csv`.
2. `$ head -n 6 < f.csv`
   Afficher les premières 6 lignes de `f.csv`.
3. `$ cat f.csv | head -n 3 | tail -n 2`
   Afficher la deuxième et troisième ligne de `f.csv`.
4. `$ cat f.csv | head -n 1000 | less`
   Visualiser interactivement les premières 1000 lignes. Taper `q` pour sortir.

1. Observez que, dans `-O`, `O` est un grand *o*, et pas zéro.

### 2.2 Exercices

1. Afficher la quatrième ligne du fichier `http://www.ens-cachan.fr/robots.txt`, sans le télécharger vers un fichier sur le disque dur.
   *Solution*:
2. Visualiser avec `less` les 100 mots les plus fréquents de la langue française [2] (il faut éviter la première ligne du fichier).
   *Solution*:

## 3 Filtrage avec `grep`

Nous avons déjà étudié la commande `grep`. Voici quelques exemples

1. `$ cat f.csv | grep ",so" | head -n 20`
   Afficher les 20 mots les plus fréquents qui commencent par *so*.
2. `$ cat f.csv | grep ",so" | head -n 20 | grep "l,subst"`
   Parmi les précédents, afficher les substantives qui finissent par *l*.

### 3.1 Exercices

1. Afficher les mots qui commencent par *sou*.
   *Solution*:
2. Parmi les précédents, afficher seulement ceux qui sont substantives.
   *Solution*:
3. Que est-ce qu'il faut ajouter aux commandes précédents pour afficher le *nombre de mots* au lieu de la liste de mots ?
   *Solution*:

### 3.2 Expressions régulières

La commande `grep` sélectionne toutes les lignes qui satisfont une expression régulière. Une *expression régulière* est une chaine de caractères qui décrit un ensemble de chaînes de caractères, à l'aide des correspondances suivantes [3] :

| | | |
|---|---|---|
| caractère `a` – `z`, `A` – `Z` | représente | le même caractère |
| `.` | représente | n'importe quel caractère |
| `^` | représente | aucun caractère au début d'une ligne |
| `$` | représente | aucun caractère à la fin d'une ligne |
| `X*` | représente | zéro, une ou plus répétitions du caractère `X` |

Voici quelques exemples :

| | | |
|---|---|---|
| `ab` | représente | la chaîne `ab` |
| `a.` | représente | `a` suivi de n'importe quel caractère |
| `^abc` | représente | une ligne qui commence par `abc` |
| `^a*b$` | représente | une ligne qui consiste à zéro ou plus `a` suivi de `b` |
| `a.*a` | représente | une chaine qui contienne deux fois `a`, n'importe où dans la chaîne |

2. Chaque ligne du fichier que vous avez télécharge dans l'exercice 1.2.1 stocke la rangée d'une table de trois colonnes. Dans chaque ligne, à l'exception de la première, qui contient un rappel des noms des colonnes, les colonnes sont séparées par des virgules. Les trois colonnes sont :
   (a) *Fréquence*. Fréquence du mot dans la langue française.
   (b) *Mot*. Le mot en question.
   (c) *Type*. Un de *dét*, *prép*, *verbe*, *pron*, *conj*, *adv*, *adj*, *subst*, *numér* ou *interj*.

3. Cette table est loin d'être complète, voir `http://fr.wikipedia.org/wiki/Expression_rationnelle` pour plus d'information.

### 3.3 Exemples avec expressions régulières

Plus particulièrement, la commande `grep` sélectionne toutes les lignes *contenant au moins une sous-chaîne de caractères représenté par l'expression régulière* :

1. `cat f.csv | grep ",a.*z,"`
   Afficher tous les mots qui commencent par `a` et qui finissent par `z`.
2. `cat f.csv | grep ",pron$"`
   Afficher tous les pronoms (il faut remarquer que l'expression `,pron` n'aurait été suffisant, à cause des mots comme *prononcer*).
3. `cat f.csv | grep "^58..,"`
   Afficher les mots avec une fréquence compris entre 5800 et 5899.

### 3.4 Exercices

1. Combien de mots commençant par *con* y a-t-il dans la liste ?
   *Solution*:
2. Parmi les précédents, combien ont-ils une fréquence compris entre 900 et 999 ?
   *Solution*:
3. Combien des substantives commençant par *ve* y a-t-il dans la liste ?
   *Solution*:

## 4 Sélection de colonnes avec `cut`

Tel que nous l'avons déjà étudié, la commande `cut` permet de sélectionner certaines colonnes d'une information tabulaire. Voici quelques exemples :

```
$ cat f.csv | grep ",printemps,"
795,printemps,subst
$ cat f.csv | grep ",printemps," | cut -d , -f 1
795
$ cat f.csv | grep ",printemps," | cut -d , -f 3
subst
$ cat f.csv | grep ",printemps," | cut -d , -f 1,2
795,printemps
```

### 4.1 Exercices

1. Afficher la fréquence et le mot des mots commençant par *ta*.
   *Solution*:
2. Afficher la fréquence du substantif le plus fréquent (elle est *33202*).
   *Solution*:
3. Afficher la liste de types de mot des mots qui commencent par *sou* (chaque type doit apparaitre dans au plus une ligne) [4].
   *Solution*:

## 5 Transformation de lignes avec `sed`

La commande `sed` est un *éditeur non interactif*. Un éditeur non interactif lit les lignes d'un fichier, normalement son entrée standard, leur applique un certain nombre de transformations et renvoi le résultat vers sa sortie standard. Les transformations à appliquer peuvent être décrites dans l'un des arguments d'invocation.

L'une des usages le plus répandu de `sed` est la substitution d'un sous-chaîne de caractères par une autre, sur tous les lignes d'un fichier. La syntaxe est la suivante :

```
cat entree.txt | sed "s/EXPR/NOUV/OPT" > sortie.txt
```

où `EXPR` est une expression régulière décrivant la sous-chaîne à substituer, `NOUV` est la nouvelle chaîne et `OPT` est, pour nos besoins, soit rien, soit la lettre d'option `g`. Sans l'option `g`, `sed` remplace seulement la première sous-chaîne concordante avec `EXPR` ; avec `g` il remplace tous les occurrences dans la ligne d'une telle sous-chaîne.

4. Indice : option `-u` de la commande `sort`, que nous n'avons pas étudié.

### 5.1 Exemples

1. `$ cat f.csv | grep ",printemps," | sed "s/,/XX/"`
   `795XXprintemps,subst`
   `$ cat f.csv | grep ",printemps," | sed "s/,/YY/g"`
   `795YYprintempsYYsubst`
   Seulement la première virgule a été remplacée par `XX` ; les deux virgules ont été remplacées par `YY` (à cause de l'option `g` sur la deuxième invocation).
2. `$ cat f.csv | grep ",printemps," | sed "s/,.*,/AAA\nBBB/g"`
   `795AAA`
   `BBBsubst`
   La nouvelle chaîne peut contenir `\n`, qui produit un *retour chariot*.
3. `$ cat f.csv | grep ",printemps," | sed "s/^/Ligne: /"`
   `Ligne: 795,printemps,subst`

### 5.2 Exercices

1. Afficher, pour les mots qui commencent par *comm*, la fréquence et le mot séparés par un espace.
   *Solution*:
2. Afficher, pour les mots qui commencent par *comm*, la fréquence et le mot de la manière suivante :
   ```
   Freq: 59902
   Mot: comme
   Freq: 5783
   Mot: commencer
   ...
   ```
   *Solution*:

## 6 Variables

Une *variable* est une correspondance entre un nombre fixe (celui de la variable) et un valeur qui peut être modifié. Pour affecter la variable `VAR` au valeur `f.csv` on utilise la syntaxe

```
$ VAR="f.csv"
```

Pour récupérer la valeur de la variable on utilise la syntaxe `$VAR` :

```
$ ls -l "$VAR"
-rw-r--r-- 1 cesar cesar 26736 2011-12-07 19:56 f.csv
```

Le valeur d'une variable peut être utilisé lors de la définition d'une autre variable :

```
$ A="head -n 3 $VAR"
$ echo "$A"
head -n 3 f.csv
```

Dans les exemples précédents, les guillemets (`"`) délimitent un chaîne de caractères à être affecté à la variable. Il est aussi possible d'utiliser l'apostrophe inversée (`‘`) afin de délimiter un *commande dont son sortie (standard)* sera affecté à la variable :

```
$ B=‘head -n 3 $A‘
$ echo "$B"
# fréquence, mot, nature
1050561,le,dét
862100,de,prép
```

### 6.1 Exercices

1. Affecter la variable `A` au valeur *avenir*.
   *Solution*:
2. Affecter la variable `B` à la fréquence du mot `$A`.
   *Solution*:

# 7 Exécution conditionnelle : `if`

A la fin de l'exécution, chaque commande renvoie à la *shell* un *état de sortie*. Il s'agit d'un nombre naturel, non négatif, qui est souvent 0 en cas de succès et différent de 0 autrement. La variable `$?` stocke l'état de sortie de la dernière commande exécutée [5] :

```
$ ls
$ echo "$?"
0
$ ls fjdskfjsdlf
ls: impossible d'accéder à fjdskfjsdlf: Aucun fichier ou dossier de ce type
$ echo "$?"
2
```

Les commandes `true` et `false` renvoient toujours un état de sortie 0 et 1, respectivement :

```
$ true
$ A=$?
$ false
$ echo "$A $?"
0 1
```

La commande `if` a la syntaxe suivante :

```
if cmd
then
   cmd-1
else
   cmd-2
fi
```

ou `cmd`, `cmd-1` et `cmd-2` sont des commandes. Si l'état de sortie de `cmd` est 0, elle exécute `cmd-1` ; sinon elle execute `cmd-2`.

## 7.1 Exercices

1. A l'aide du éditeur `gedit`, éditer le fichier `cond.sh` pour qu'il lise :

   ```
   #!/bin/bash
   A="true"
   if $A
   then
      echo "oui: $A"
   else
      echo "non: $A"
   fi
   ```

   Modifier l'affectation de `A` avec
   (a) `false`
   (b) `ls jfsldfjlfsd`
   (c) `test 'abc' == 'abc'`
   (d) `test 'abc' != 'abc'`
   (e) `test 'abc' != 'xyz'`
   (f) `test -n 'abc'`
   (g) `test -n ''`
   (h) `test -z ''`
   (i) `test -z 'abc'`
   (j) `test -f 'f.csv'`
   (k) `test -f 'jfsldfjlfsd'`

   et décrire en chaque cas le résultat. Quelle est la vérification réalisé par les opérateurs `==`, `!=`, `-n`, `-z` et `-f` de la commande `test` ?
   *Solution*:

5. On assume, dans cet exemple, que le fichier `fjdskfjsdlf` n'existe pas sur le répertoire actuel.

# 8 Scripts de *shell*

Dans un script, on dispose de variables définies automatiquement par la *shell* qui nous permettent de récupérer les arguments d'invocation du script :

| | |
|---|---|
| `$#` | Nombre d'arguments passés au script. |
| `$*` | La liste de tous les arguments. |
| `$1` | Le première argument. |
| `$2` | Le deuxième argument. |
| ... | |

Voici un script très simple qui montre le nombre d'arguments passés et la valeur des trois premières :

```
#!/bin/bash
echo "nombre: $#"
echo "trois premieres: _$1_ _$2_ _$3_"
```

## 8.1 Exercices

Au cours des exercices suivantes nous allons construire un script qui utilise le fichier `f.csv` afin de répondre quelques questions très simples. Notre script, sera capable de montrer la *fréquence* et le type d'un mot, donné comme argument d'invocation après l'option `-f` :

```
$ freq.sh -f soleil
Mot: soleil
Frequence: 6692
Type: subst
```

Il disposera aussi de une option, `-i`, permettant de afficher la fréquence des 5 mots les plus fréquents qui contiennent un sous-chaîne de caractères donné (`-i` pour *infixe*) :

```
./freq.sh -i lar
2546 larme
2226 vieillard
1792 large
1544 déclarer
```

1. Créer un script qui affiche un message d'erreur si le nombre de paramètres d'invocation est différent à deux. Appelez ce script `freq.sh`, vous pouvez vous baser sur l'exemple de la section précédent. En particulier, assurez-vous que la *première* ligne soit `#!/bin/bash`.
2. Modifiez votre script pour que, une fois il a rejeté une invocation avec un nombre incorrecte d'arguments, il vérifie si le fichier `f.csv` est présent. En cas négatif, votre script devra télécharger le fichier.
3. Modifiez votre script pour que, après d'avoir assuré qu'il a accès au fichier `f.csv`, il termine avec un message d'erreur si le première argument n'est pas `-f` ou `-p`.
4. Ajoutez les commandes nécessaires pour afficher :

   ```
   Mot: x
   Frequence: y
   Type: z
   ```

   si les arguments d'invocation de votre script sont `-f x` et que la ligne `y,x,z` est présente dans `f.csv`, ou le message

   ```
   Le mot x n'est pas dans la liste.
   ```

   si `y,x,z` n'est pas présente.
5. Ajoutez les commandes nécessaires pour gérer le cas ou votre script a été appelé avec `-i x` comme arguments. En ce cas, vous devez afficher les fréquences des 5 mots les plus fréquents, chaque fréquence sur une ligne, suivis d'un espace et du mot associé.

# A Rappels des TPs 1 et 2

1. `.`
   Répertoire actuel.
2. `..`
   Répertoire “père” du répertoire actuel.
3. `~`
   Répertoire maison. Alias de `$HOME`.
4. `*`
   Tous les fichiers et répertoires.
5. `cd DIR`
   Changer le répertoire actuel à `DIR`.
6. `pwd`
   Afficher sur la sortie standard le répertoire actuel.
7. `ls [-l -a -R] DIR1 DIR2 ... FILE1 FIlE2 ...`
   Affiche le contenu de répertoires et information administrative sur les fichiers.
8. `cat FILE1 FILE2 ...`
   Afficher, un par un, les fichiers `FIlE1`, `FILE2`, . . .sur la sortie standard.
9. `rm FILE` ou bien `rm -R DIR`
   Effacer des fichiers ou des répertoires.
10. `mkdir DIR`
    Créer des répertoires.
11. `cp -v -R SOURCE DESTINATION`
    Copier des fichiers et des répertoires (`-R` pour récursif).
12. `mv SOURCE DESTINATION`
    Déplacer ou renommer des fichiers ou répertoires.
13. `chmod OUGO FILE DIR...`
    Modifier les droits d’accès pour des fichiers ou des répertoires. Exemple : `0644` : `rw-` pour l’`U`tilisateur, `r--` pour le `G`roup et `r--` pour les `O`utres.
14. `echo Bonjour monde`
    ou `echo "Bonjour monde"` Aficher un ligne de texte sur la sortie standard.
15. `wc [-l -w -c] SOURCE DESTINATION`
    Afficher le nombre de lignes, de mots et d’octets d’un fichier.
16. `grep EXPRESION FILE`
    Afficher sur la sortie standard les lignes du fichier `FILE` qui correspond au motif `EXPRESION`. Si `FILE` est omis, lire de l’entrée standard.
17. `find DIR [-name EXPR -user USER -type [d f]]`
    Rechercher des fichiers dans une hiérarchie de répertoires.
18. `cut -d SEP -f N1,N2,...`
    Supprimer une partie de chaque ligne d’un fichier.
19. `man COMMAND`
    Consulter le manuel de référence de la commande `COMMAND`.
20. `who`
    Montrer qui est connecté sur la même machine.
21. `gedit FILE`
    Éditeur le texte du fichier `FILE`.
22. `ln [-s] TARGET SOURCE`
    Créer des liens (symboliques) entre fichiers.
23. `tar czvf FILE.tgz DIR`
    Archiver dans le fichier `FILE.tgz` le répertoire `DIR` et tous ses sous-répertoires.
24. `tar xzvf FILE.tgz`
    Extraire sur le répertoire actuel le contenu du fichier `FILE.tgz`.
25. `ps [-ef]`
    Présenter un cliché instantané des processus en cours.
26. `kill [-15 -9] PID`
    Touer (avec la signal 15 ou 9) un processus identifié par son `PID`.