

# Scripting Shell — Annexes

U-Classroom

## 1 Manipulation de chaînes de caractères

Le shell permet certaines manipulations sur les variables. En voici une liste.

### 1.1 Calculer la longueur d'une chaîne

*Syntaxe* : `${#var}`

```
$ echo $PWD
/home/gauvain
$ echo ${#PWD}
15
```

### 1.2 Supprimer la plus courte sous-chaîne à gauche

*Syntaxe* : `${var#modele}`

```
$ echo $PWD
/home/gauvain
$ echo ${PWD#*/}
home/gauvain
```

### 1.3 Supprimer la plus longue sous-chaîne à gauche

*Syntaxe* : `${var##modele}`

```
$ echo $PWD
/home/gauvain
$ echo ${PWD##*/}
gauvain
```

### 1.4 Supprimer la plus courte sous-chaîne à droite

*Syntaxe* : `${var%modele}`

```
$ var="valeur1,,valeur3,valeur4"
$ echo ${var%,*}
valeur1,,valeur3
```

```
$ nom_fichier=fichier.tar.gz
$ echo ${nom_fichier%.*}
fichier.tar
```

### 1.5 Supprimer la plus longue sous-chaîne à droite

*Syntaxe* : `${var%%modele}`

```
$ var="valeur1,,valeur3,valeur4"
$ echo ${var%%,*}
valeur1
```

```
$ nom_fichier=fichier.tar.gz
$ echo ${nom_fichier%.*}
fichier
```

## 1.6 Extraire nb caractères à partir de l'indice ind

*Syntaxe* : `${var:ind:nb}`

```
$ var="abcdefghijklmn"
$ echo ${var:4:3}
efg
```

```
$ echo ${var:4} # si nb est omis, sélectionne jusqu'à la fin
efghijklmn
```

## 1.7 Remplacer la première plus longue sous-chaîne mod par rempl

*Syntaxe* : `${var/mod/rempl}`

```
$ var="do re mi fa sol la si do"
$ echo ${var/do/UT}
UT re mi fa sol la si do
```

```
# si deux possibilités existent, c'est toujours la chaîne la plus longue qui
# est remplacée :
$ var="do re mi fa sol la si do"
$ echo ${var/do*i/UT}
UT do
```

## 1.8 Supprimer la première sous-chaîne mod en début de chaîne

*Syntaxe* : `${var/#mod/}`

```
$ var="do re mi fa sol la si do"
$ echo ${var//do/}
re mi fa sol la si do
```

## 1.9 Supprimer la première sous-chaîne mod en fin de chaîne

*Syntaxe* : `${var/%mod/}`

```
$ var="do re mi fa sol la si do"
$ echo ${var//do/}
do re mi fa sol la si
```

## 1.10 Remplacer toutes les sous-chaînes mod par rempl

*Syntaxe* : `${var//mod/rempl}`

```
$ var="do re mi fa sol la si do"
$ echo ${var//do/UT}
UT re mi fa sol la si UT
```

## 1.11 Supprimer toutes les sous-chaînes mod

*Syntaxe* : `${var//mod/}`

```
$ var="do re mi fa sol la si do"
$ echo ${var//do/}
re mi fa sol la si
```

## 2 Les outils standards

L'utilisation du shell nécessite assez peu de connaissances. Mais manipuler des données peut vite devenir un exercice difficile sans connaître les outils disponibles. Les outils que nous allons présenter ici sont pour la plupart des outils GNU sous Linux, parfois des réimplémentations sous d'autres OS (\*BSD).

Nous n'aborderons pas toutes les possibilités de ces outils, et nous vous conseillons de lire leurs pages man ou info.

### 2.1 cut

cut extrait des portions de chaque ligne d'un fichier suivant un délimiteur donné.

#### Options courantes

**-d delimitier** définit le caractère à utiliser comme délimiteur

**-f section(s)** définit quelles éléments garder

#### Exemples

```
# "découpe" suivant ',' et affiche les éléments 2 et 3
$ echo x,y,z | cut -d, -f2-3
y,z
```

```
# n'affiche que les 2ème, 3ème et 4ème caractères
$ echo x,y,z | cut -b 2-4
,y,
```

### 2.2 sort

sort effectue un tri sur une liste.

#### Options courantes

**-r** reverse, trie dans l'autre sens

#### Exemples

```
$ cut -d: -f1 /etc/passwd # liste les utilisateurs dans l'ordre d'ajout
root
daemon
bin
...
```

```
$ cut -d: -f1 /etc/passwd | sort
apt-mirror
backup
bin
daemon
davfs2
...
```

### 2.3 uniq

uniq permet de supprimer les duplicats d'une liste.

#### Options courantes

**-c** affiche le nombre d'occurrences

## Exemples

```
$ ps au | cut -d' ' -f1
root
root
root
gauvain
gauvain
gauvain
...
```

```
$ ps au | cut -d' ' -f | uniq
root
gauvain
```

```
$ ps au | cut -d' ' -f | uniq -c
7 root
10 gauvain
```

## 2.4 paste

paste lit 2 fichiers ligne à ligne et combine les 2 valeurs sur une ligne. Par défaut \t (tabulation) est utilisé comme séparateur.

### Options courantes

**-d séparateur** permet de définir un autre séparateur

```
\subsubsection*{Exemples}
```

```
$ cat alpha
```

```
a
b
c
```

```
$ cat num
```

```
1
2
3
```

```
$ paste -d, alpha num
```

```
a,1
b,2
c,3
```

## 2.5 split

split découpe un fichier en plusieurs autres fichiers.

### Options courantes

**-l nombre** définit le nombre de ligne par fichier

## Exemples

```
$ cat fichier
```

```
ligne1
ligne2
ligne3
ligne4
```

```

ligne5

# les noms de fichiers seront de la forme 'ficXY'
$ split -l 2 fichier fic

$ cat ficaa
ligne1
ligne2

$ cat ficab
ligne3
ligne4

$ cat ficac
ligne5

```

## 2.6 wc

wc est un compteur. Il sait compter le nombre de caractères, de mots ou de lignes d'une chaîne de caractères.

### Options courantes

- c compte le nombre de caractères
- l compte le nombre de lignes
- w compte le nombre de mots

### Exemples

```

# nombre de lignes dans un fichier
$ wc -l /etc/X11/xorg.conf
41 /etc/X11/xorg.conf

# nombre de caractères (caractère de fin de ligne compris)
$ echo 1234 | wc -c
5

# le caractère de fin de ligne n'est pas envoyé par echo
$ echo -n 1234 | wc -c
4

$ echo "Je vais compter les mots de cette phrase." | wc -w
8

```

## 2.7 tr

tr permet de convertir ou d'éliminer des caractères depuis son entrée standard.

### Options courantes

- d **caractères** supprime les caractères, ne les remplace pas
- s **caractères** remplace une séquence de caractères par ce seul caractère

## Exemples

```
$ echo 'abcd' | tr bd YZ
aYcZ
```

```
$ echo 'CamelCase' | tr '[:lower:]' '[:upper:]'
CAMELCASE
```

```
$ echo 'aabbbaacdd' | tr -d a
bbcdd
```

```
$ echo 'aabbbaacdd' | tr -s ab
abacdd
```

## 2.8 cat

cat permet d'afficher le contenu d'un fichier, mais également de concaténer le contenu de plusieurs fichiers en un seul.

### Exemples

```
$ cat fichier1
contenu du 1er fichier
```

```
$ cat fichier2
contenu du 2ème fichier
```

```
$ cat fichier1 fichier2 > fichier_concat
$ cat fichier_concat
contenu du 1er fichier
contenu du 2ème fichier
```

## 2.9 tac

tac permet d'afficher le contenu d'un fichier à l'envers.

### Exemples

```
$ cat fichier
ligne1
ligne2
ligne3
ligne4
ligne5
```

```
$ tac fichier
ligne5
ligne4
ligne3
ligne2
ligne1
```

## 2.10 grep

grep est une commande qui recherche un motif dans les lignes des fichiers. Par défaut, il affiche les lignes contenant le motif cherché. On peut également lui faire afficher le contexte (lignes précédentes et suivantes), ou les lignes ne contenant pas le motif. grep permet des recherches simples, mais aussi des recherches utilisant des expressions régulières.

### Options courantes

- v cherche les lignes ne contenant pas le motif
- i ignore la casse
- n affiche le numéro de la ligne trouvée
- l n'affiche que les noms de fichiers correspondant
- R cherche récursivement (si un dossier est donnée en argument)

### Exemples

```
$ cat fichier
blabla ligne1 blabla
blabla ligne2 blabla
blabla ligne3 blabla
blabla ligne4 blabla
blabla ligne5 blabla
```

```
$ grep li.*3 fichier
blabla ligne3 blabla
```

```
$ grep -v li.*3 fichier
blabla ligne1 blabla
blabla ligne2 blabla
blabla ligne4 blabla
blabla ligne5 blabla
```

```
$ grep -in LIGNE3 fichier
3:blabla ligne3 blabla
```

## 2.11 dog

dog permet d'afficher le code source HTML d'une URL distante et d'en extraire certains éléments (liens, images).

### Options courantes

- links extrait les liens
- images extrait les images

### Exemples

```
$ dog http://www.monsite.fr
<html>
<head><title>Ma jolie page</title></head>
<body>
<a href="http://u-classroom.net">Lien vers u-classroom</a><br />
<br />
<br />
</body>
</html>
```

```
$ dog --links http://www.monsite.fr
http://u-classroom.net
```

```
$ dog --images http://www.monsite.fr
http://www.monsite.fr/img/image1.jpg
```

`http://www.monsite.fr/img/image2.png`

## 2.12 wget

wget permet de télécharger des ressources sur le web.

### Options courantes

- O `dest` enregistre le lien dans le fichier `dest`
- c continue un téléchargement déjà commencé
- r récupère les données récursivement
- `spider` teste si l'URL pointe sur quelque chose

### Exemples

```
$ wget http://www.monsite.fr/img/image1.jpg
100%[=====>] 19 742      104K/s   in 0,2s
2009-04-29 00:24:45 (104 KB/s) - 'image1.jpg' saved [19742/19742]
```

```
# On peut par exemple l'utiliser couplé avec dog pour télécharger certaines
# ressources d'une URL.
```

```
$ dog --images http://www.monsite.fr | grep jpg > temp; wget -i temp
100%[=====>] 19 742      104K/s   in 0,2s
2009-04-29 00:24:45 (104 KB/s) - 'image1.jpg' saved [19742/19742]
```

## 2.13 basename

basename permet d'extraire le nom d'un fichier en éliminant le chemin. Si l'extension du fichier est fournie en argument, elle est éliminée.

### Exemples

```
$ nom_fic=/home/gauvain/image.jpg
$ basename $nom_fic
image.jpg
```

```
$ basename $nom_fic .jpg
image
```

## 2.14 dirname

dirname permet d'extraire le chemin en éliminant le nom d'un fichier.

### Exemples

```
$ nom_fic=/home/gauvain/image.jpg
$ basename $nom_fic
/home/gauvain
```

## 2.15 file

file permet de retourner le type du fichier passé en paramètre.

## Exemples

```
$ file dir1
dir1: directory

# création d'un lien symbolique
$ ln -s image.jpg link1
$ file link1
link1: symbolic link to 'image.jpg'

$ file image.jpg
image.jpg: JPEG image data, EXIF standard 2.21
```

## 2.16 head

head affiche le début d'un fichier avec le nombre de lignes spécifié (par défaut les 10 premières lignes).

### Options courantes

**-n nb** affiche les nb premières lignes

## Exemples

```
$ cat fichier.txt
ligne1
ligne2
ligne3
ligne4
ligne5

$ head -n 2 fichier.txt
ligne1
ligne2
```

## 2.17 tail

tail affiche la fin d'un fichier avec le nombre de lignes spécifié.

### Options courantes

**-n nb** affiche les nb dernières lignes

**-f** affiche les lignes au fur et à mesure qu'elles sont ajoutées au fichier

## Exemples

```
$ cat fichier.txt
ligne1
ligne2
ligne3
ligne4
ligne5

$ tail -n 2 fichier.txt
ligne4
ligne5

$ sudo tail -f /var/log/syslog
```

## 2.18 touch

`touch` permet de mettre à jour la date et l'heure d'accès et de modification d'un fichier avec la date actuelle. Si le fichier n'existe pas, il est créé.

### Options courantes

`-d date` utilise `date` au lieu de l'heure actuelle

`-r fichier` utilise la date de `fichier` au lieu de l'heure actuelle

### Exemples

```
$ ls -l
-rw-r----- 1 gouvain gouvain 23297 2008-04-29 12:04 toto
-rw-r----- 1 gouvain gouvain   666 2009-01-31 09:33 tata
$ touch toto tata
$ ls -l
-rw-r----- 1 gouvain gouvain 23297 2009-05-02 02:13 toto
-rw-r----- 1 gouvain gouvain   666 2009-05-02 02:13 tata

$ touch tutu
$ ls -l
-rw-r----- 1 gouvain gouvain 23297 2008-05-02 02:13 toto
-rw-r----- 1 gouvain gouvain   666 2009-05-02 02:13 tata
-rw-r----- 1 gouvain gouvain     0 2009-05-02 02:14 tutu
```

## 2.19 find

`find` est une des commandes les plus complètes des utilitaires GNU. Elle permet de rechercher récursivement des fichiers à partir de conditions sur leur nom, leur type, leur date de création, etc...

Elle permet également d'exécuter n'importe quelle autre commande sur l'ensemble des réponses qui correspondent aux critères.

### Exemples

**Recherche sur le nom de fichier :** La recherche est toujours récursive. Une option `-depth` permet de gérer la profondeur de cette recherche dans les dossiers et sous-dossiers.

```
$ ls -l
-rw-r----- 1 gouvain gouvain 112007 2009-05-02 02:18 tata
-rw-r----- 1 gouvain gouvain   251 2009-05-02 02:09 toto
drwxr-x--- 2 gouvain gouvain   4096 2009-05-02 02:19 toto-dir

$ ls -l toto-dir
-rw-r----- 1 gouvain gouvain 12807 2009-05-02 02:21 toto
-rw-r----- 1 gouvain gouvain  5467 2009-05-02 02:19 toto2
-rw-r----- 1 gouvain gouvain   789 2009-05-02 02:19 tutu

$ find . -name toto
./toto-dir/toto
./toto
```

**Recherche avec une expression régulière :**

```
$ find . -regex .*toto.*
./toto-dir
./toto-dir/tutu
./toto-dir/toto
./toto-dir/toto2
./toto
```

Remarquez que le fichier "tutu" est renvoyé car le mot "toto" se trouve dans son chemin (dans le nom du répertoire "toto-dir" qui le contient).

**Recherche sur le type de fichier :** L'option `-type` prend en argument une lettre définissant le type de fichier (liste non exhaustive) :

`f` définit un fichier

`d` définit un dossier

`l` définit un lien symbolique

```
$ find . -regex .*toto.* -type d
./toto-dir
```

**Recherche sur la taille :** L'option `-size` permet de définir selon quel critère de taille la recherche doit être faite. Son argument commence par `+` ou `texttt-` (plus ou moins grand que), est suivi de la taille (`c=octets`, `k=Ko`, `M=Mo`, `G=Go`).

```
$ find . -regex .*t.*t.* -size +12000c
./toto-dir/toto
./tata
```

**Exécuter une commande sur la liste des réponses :**

```
$ find . -name toto -type f -exec mv {} {}_NEW \;
-rw-r----- 1 gouvain gouvain 112007 2009-05-02 02:18 tata
-rw-r----- 1 gouvain gouvain    251 2009-05-02 02:09 toto_NEW
drwxr-x---  2 gouvain gouvain   4096 2009-05-02 02:19 toto-dir
$ ls -l toto-dir
-rw-r----- 1 gouvain gouvain 12807 2009-05-02 02:21 toto_NEW
-rw-r----- 1 gouvain gouvain  5467 2009-05-02 02:19 toto2
-rw-r----- 1 gouvain gouvain   789 2009-05-02 02:19 tutu
```

Pour chaque réponse qui correspond aux critères, l'option `-exec` permet de lancer la commande `mv`. Dans cette sous-commande, `{}` remplace le nom du fichier renvoyé par `find`. Il faut terminer la sous-commande par `\;`.

# Contents

<b>1 Manipulation de chaînes de caractères</b>	<b>1</b>
1.1 Calculer la longueur d'une chaîne . . . . .	1
1.2 Supprimer la plus courte sous-chaîne à gauche . . . . .	1
1.3 Supprimer la plus longue sous-chaîne à gauche . . . . .	1
1.4 Supprimer la plus courte sous-chaîne à droite . . . . .	1
1.5 Supprimer la plus longue sous-chaîne à droite . . . . .	1
1.6 Extraire nb caractères à partir de l'indice <code>ind</code> . . . . .	2
1.7 Remplacer la première plus longue sous-chaîne <code>mod</code> par <code>repl</code> . . . . .	2
1.8 Supprimer la première sous-chaîne <code>mod</code> en début de chaîne . . . . .	2
1.9 Supprimer la première sous-chaîne <code>mod</code> en fin de chaîne . . . . .	2
1.10 Remplacer toutes les sous-chaînes <code>mod</code> par <code>repl</code> . . . . .	2
1.11 Supprimer toutes les sous-chaînes <code>mod</code> . . . . .	2
<b>2 Les outils standards</b>	<b>3</b>
2.1 <code>cut</code> . . . . .	3
2.2 <code>sort</code> . . . . .	3
2.3 <code>uniq</code> . . . . .	3
2.4 <code>paste</code> . . . . .	4
2.5 <code>split</code> . . . . .	4
2.6 <code>wc</code> . . . . .	5
2.7 <code>tr</code> . . . . .	5
2.8 <code>cat</code> . . . . .	6
2.9 <code>tac</code> . . . . .	6
2.10 <code>grep</code> . . . . .	6
2.11 <code>dog</code> . . . . .	7
2.12 <code>wget</code> . . . . .	8
2.13 <code>basename</code> . . . . .	8
2.14 <code>dirname</code> . . . . .	8
2.15 <code>file</code> . . . . .	8
2.16 <code>head</code> . . . . .	9
2.17 <code>tail</code> . . . . .	9
2.18 <code>touch</code> . . . . .	10
2.19 <code>find</code> . . . . .	10