


\$ SHELL

Jean-Philippe Eisenbarth

Janvier 2021

 Université de Lorraine - TELECOM Nancy



Document distribué selon les termes de la licence



© Licence Creative Commons version 4.0 (ou ultérieure)

👤 Attribution 🔄 Partage dans les mêmes conditions

<https://creativecommons.org/licenses/by-sa/4.0/>

Remerciements :

- Victorien Elvinger pour son cours [Shell \(2017\)](#)
- Benjamin Ségault pour le poly du cours

cut - remove sections from each line of files

cut OPTION... [FILE]...

Options courantes :

- -c : affiche les caractères aux positions indiquées
- -f : affiche les champs indiqués (par défaut les champs sont considérés séparés par une tabulation)
- -d : spécifie le séparateur de champs à prendre en compte pour l'option -f

Commande cut, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | cut -c 3
b
```

# Commande cut, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | cut -c 3  
b
```

- ```
$ echo "a,b,c,d,e" | cut -c 3-5
b,c
```

# Commande cut, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | cut -c 3  
b
```

- ```
$ echo "a,b,c,d,e" | cut -c 3-5
b,c
```

- ```
$ echo "a,b,c,d,e" | cut -c 3-  
b,c,d,e
```

Commande cut, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | cut -f 1
a,b,c,d,e
```

# Commande cut, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | cut -f 1  
a,b,c,d,e
```

- ```
$ echo "a,b,c,d,e" | cut -d ',' -f 3
c
```



tr - translate or delete characters

tr [OPTION]... SET1 [SET2]

Options courantes :

- -d : supprime les caractères présents dans SET1
- -s : remplace toutes les répétitions de caractères de SET1 par une seule occurrence de ce caractère

tr s'applique sur les caractères que la commande reçoit sur l'entrée standard.

# Commande tr, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | tr -d ','  
abcde
```

Commande tr, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | tr -d ','
abcde
```

- ```
$ echo "a,b,c,d,e" | tr ',' ';'  
a;b;c;d;e
```

Commande tr, exemples

Quelques exemples :

- ```
$ echo "a,b,c,d,e" | tr -d ','
abcde
```

- ```
$ echo "a,b,c,d,e" | tr ',' ';'  
a;b;c;d;e
```

- ```
$ echo "a,,,b,,,c,,,d,,,e" | tr -s ','
a,b,c,d,e
```

Une expression régulière est une séquence de caractères qui, selon une certaine syntaxe, décrit un motif (*pattern*).

En informatique ils existent différentes syntaxes : POSIX, GNU, Perl, Java, .NET, ...

En Shell, nous nous concentrerons sur la syntaxe POSIX et GNU.

Références: <https://www.regular-expressions.info/gnu.html> et <https://www.regular-expressions.info/posix.html>

## Syntaxe d'expression régulière

| (Méta)caractère            | Signification ( <i>le motif match ...</i> )                  |
|----------------------------|--------------------------------------------------------------|
| un caractère<br>a, b, 0... | Le caractère donné                                           |
| .                          | N'importe quel caractère, une fois, hormis un saut de ligne. |
| [aBc0]                     | Tout caractère, une fois, parmi ceux entre crochets          |
| [^ae]                      | Tout caractère, une fois, sauf ceux entre crochets           |
| [a-z]                      | N'importe quel caractère compris entre a et z, une fois.     |

## Remarques générales

- Les caractères sont sensibles à la casse
- Les caractères `.` `[` `]` doivent être échappés (`\.`, `\[`, `\]`) si l'on ne veut pas qu'ils soient considérés comme les métacaractères correspondants
- La syntaxe avec les crochets est appelée *POSIX bracket expression* ou encore *Regular expression character class* (différent de *POSIX character classes* que vous pouvez trouver page 46 (section 5.4) du poly)

## Remarques sur les *POSIX bracket expressions*

- le backslash n'est pas un métacaractère à l'intérieur de crochets
- pour match `]` il faut le placer juste après le crochet de départ ou après le `^` de négation
- pour match `-` il faut le placer juste avant le crochet de fermeture
- pour match `^` il faut le placer avant le dernier littéral - ou avant le crochet de fermeture

Exemple : `[]\d^-]` va match `]`, `\`, `d`, `^` ou `-`



# Syntaxe d'expression régulière - capture des groupements et quantificateurs

Les parenthèses ( ) permettent de capturer des groupements pour les réutiliser (voir exemple après).

Les quantificateurs (quantifiers) sont :

| Métacaractère | Signification ( <i>le motif match ...</i> ) |
|---------------|---------------------------------------------|
| *             | zéro, une fois ou plus                      |
| +             | au moins une fois                           |
| ?             | zéro ou une fois                            |
| {n}           | exactement $n$ fois                         |
| {n,m}         | entre $n$ et $m$ fois                       |
| {n,}          | au moins $n$ fois                           |

# Syntaxe d'expression régulière - exemples

Quelques exemples :

- **ab\*c** match "ac", "abc", "abbc" etc
- **[hc]at** match "hat" et "cat"
- **a{3}** match "aaa"
- **(abc|def)=\1** match "abc=abc" ou "def=def" mais pas "abc=def" ou "def=abc"

## Syntaxe d'expression régulière - d'autres métacaractères

| Métacaractère | Signification ( <i>le motif match ...</i> ) |
|---------------|---------------------------------------------|
| ^             | marque le début de ligne                    |
| \$            | marque la fin de ligne                      |
|               | un "ou" (en anglais : <i>alternation</i> )  |

Quelques exemples :

- `^[hc]at` match "hat" ou "cat" mais seulement si la chaîne est placée en début de ligne
- `[^b]at` match toutes les chaînes ".at" sauf "bat"
- `cat|dog|mouse` match la chaîne "cat" ou la chaîne "dog" ou la chaîne "mouse"

Le standard POSIX propose 2 jeux de normes :

- BRE (Basic Regular Expression)
- ERE (Extended Regular Expression)

**Principale différence aujourd'hui :**

- Avec BRE, il est nécessaire d'échapper les métacaractères quantificateurs (e.g. `\{2,5\}`) ainsi que le métacaractère *alternation* (`\|`)

grep - print lines matching a pattern

grep [OPTIONS] PATTERN [FILE...]

Options courantes :

- -E : mode ERE (mode par défaut BRE)
- -i : la recherche est insensible à la casse
- -v : sélectionne les lignes qui ne correspondent pas
- -q : mode silencieux
- --color=auto : active la coloration pour les correspondances

# Commande grep, exemples

Quelques exemples :

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep "Linux"  
GNU Linux
```

Commande grep, exemples

Quelques exemples :

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep "Linux"
GNU Linux
```

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep -i "gnu"  
GNU Linux
```


Commande grep, exemples

Quelques exemples :

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep "Linux"
GNU Linux
```

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep -i "gnu"  
GNU Linux
```

- ```
$ echo -e "GNU Linux\nWindows\nMacOS" | grep -v "GNU Linux"
Windows
MacOS
```

# Commande grep, exemples

```
■ $ echo -e "hat\ncat\ndog\nbat" | grep "[hc]at"
hat
cat
```

# Commande grep, exemples

```
$ echo -e "hat\ncat\ndog\nbat" | grep "[hc]at"
```

- hat  
cat

```
$ echo -e "hat\nex-équo\nex-aequo\nex-equo\n" | grep -E
↪ "ex-(a?e|æ|é)quo"
```

- ex-équo  
ex-aequo  
ex-equo

Écrire une commande **islog** permettant de tester si un utilisateur est connecté sur la machine hôte, en utilisant la commande **who**.

Par exemple : **islog dupont** affichera **dupont connected** ou **dupont not connected** selon s'il est connecté ou non

# Commande sed

sed - stream editor for filtering and transforming text

sed [OPTION]... script-only-if-no-other-script [input-file]...

Options courantes :

- -E ou -r : mode ERE (mode par défaut BRE)
- -i : Modifie directement le fichier en cours de traitement

Syntaxes pour la substitution :

```
$ sed 's/<searchregex>/<replacement>/' file
$ sed 's/<searchregex>/<replacement>/g' file
```

Syntaxe pour la suppression :

```
$ sed '/<searchregex>/d' file
```

# Commande sed, exemples

```
$ cat os.txt
Linux
Windows
MacOS
```

Quelques exemples :

```
$ sed s/Linux/GNULinux/ os.txt
GNULinux
■ Windows
MacOS
```

```
$ sed /Windows/d os.txt
■ Linux
MacOS
```

## Exercice

Écrire un script shell **sustitution** effectuant la substitution dans le fichier passé en premier argument, de toutes les occurrences de la chaînes de caractères passée en deuxième argument par la chaîne passée en troisième argument.

Le résultat sera rangé dans le fichier désigné par le premier argument.

Restriction: les arguments ne comportent pas d'espaces.

Par exemple : `./sustitution fichier.txt "test" "essai"` remplace toutes les occurrences de "test" par "essai" dans le fichier "fichier.txt"