Version : **2024.01**

Dernière mise-à-jour : 2024/10/21 11:20

RH13401 - Les Scripts Shell

Contenu du Module

LAB #1 - Les Scripts Shell

Le but de la suite de cette unité est de vous amener au point où vous êtes capable de comprendre et de déchiffrer les scripts, notamment les scripts de démarrage ainsi que les scripts de contrôle des services.

Écrire des scripts compliqués est en dehors de la portée de cette unité car il nécessite une approche programmation qui ne peut être adressée que lors d'une formation dédiée à l'écriture des scripts.

1.1 - Exécution

Un script shell est un fichier dont le contenu est lu en entrée standard par le shell. Le contenu du fichier est lu et exécuté d'une manière séquentielle. Afin qu'un script soit exécuté, il suffit qu'il puisse être lu au quel cas le script est exécuté par un shell fils soit en l'appelant en argument à l'appel du shell :

/bin/bash myscript

soit en redirigeant son entrée standard :

/bin/bash < myscript

Dans le cas où le droit d'exécution est positionné sur le fichier script et à condition que celui-ci se trouve dans un répertoire spécifié dans le PATH de l'utilisateur qui le lance, le script peut être lancé en l'appelant simplement par son nom :

myscript

Pour lancer le script sans qu'il soit dans un répertoire du PATH, il convient de se placer dans le répertoire contenant le script et de le lancer ainsi :

./myscript

Dans le cas où le script doit être exécuté par le shell courant, dans les mêmes conditions que l'exemple précédent, et non par un shell fils, il convient de le lancer ainsi :

. myscript

Dans un script il est fortement conseillé d'inclure des commentaires. Les commentaires permettent à d'autres personnes de comprendre le script. Toute ligne de commentaire commence avec le caractère **#**.

Il existe aussi un **pseudo commentaire** qui est placé au début du script. Ce pseudo commentaire permet de stipuler quel shell doit être utilisé pour l'exécution du script. L'exécution du script est ainsi rendu indépendant du shell de l'utilisateur qui le lance. Le pseudo commentaire commence avec les caractères **#!**. Chaque script commence donc par une ligne similaire à celle-ci :

```
#!/bin/sh
```

Puisque un script contient des lignes de commandes qui peuvent être saisies en shell intéractif, il est souvent issu d'une procédure manuelle. Afin de faciliter la création d'un script il existe une commande, **script**, qui permet d'enregistrer les textes sortis sur la sortie standard, y compris le prompt dans un fichier dénommé **typescript**. Afin d'illustrer l'utilisation de cette commande, saisissez la suite de commandes suivante :

```
[trainee@redhat9 ~]$ script
Script started, output log file is 'typescript'.

[trainee@redhat9 ~]$ pwd
/home/trainee
```

```
[trainee@redhat9 ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  typescript  Videos

[trainee@redhat9 ~]$ exit
exit
Script done.

[trainee@redhat9 ~]$ cat typescript
Script started on 2024-10-21 11:57:26+02:00 [TERM="xterm-256color" TTY="/dev/pts/0" COLUMNS="86" LINES="24"]
[trainee@redhat9 ~]$ pwd
/home/trainee
[trainee@redhat9 ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  typescript  Videos
[trainee@redhat9 ~]$ exit
exit

Script done on 2024-10-21 11:57:36+02:00 [COMMAND_EXIT_CODE="0"]
```

Cette procédure peut être utilisée pour enregistrer une suite de commandes longues et compliquées afin d'écrire un script.

Pour illustrer l'écriture et l'exécution d'un script, créez le fichier **myscript** avec **vi** :

```
[trainee@redhat9 ~]$ vi myscript
[trainee@redhat9 ~]$ cat myscript
pwd
ls
```

Sauvegardez votre fichier. Lancez ensuite votre script en passant le nom du fichier en argument à /bin/bash :

```
[trainee@redhat9 ~]$ /bin/bash myscript
/home/trainee
Desktop    Downloads  myscript  Public     typescript
Documents  Music      Pictures  Templates  Videos
```

Lancez ensuite le script en redirigeant son entrée standard :

```
[trainee@redhat9 ~]$ /bin/bash < myscript
/home/trainee
Desktop    Downloads  myscript  Public     typescript
Documents  Music      Pictures  Templates  Videos
```

Pour lancer le script en l'appelant simplement par son nom, son chemin doit être inclus dans votre PATH:

```
[trainee@redhat9 ~]$ echo $PATH
/home/trainee/.local/bin:/home/trainee/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

Dans le cas de RHEL 9, même si PATH contient $HOME/bin, le répertoire n'existe pas :

```
[trainee@redhat9 ~]$ ls
Desktop    Downloads  myscript  Public     typescript
Documents  Music      Pictures  Templates  Videos
```

Créez donc ce répertoire :

```
[trainee@redhat9 ~]$ mkdir bin
```

Ensuite déplacez votre script dans ce répertoire et rendez-le exécutable pour votre utilisateur :

```
[trainee@redhat9 ~]$ mv myscript ~/bin
[trainee@redhat9 ~]$ chmod u+x ~/bin/myscript
```

Exécutez maintenant votre script en l'appelant par son nom à partir du répertoire **/tmp** :

```
[trainee@redhat9 ~]$ cd /tmp

[trainee@redhat9 tmp]$ myscript
/tmp
```

```
dbus-BKNtynzn0b
dbus-G7skg3Wlpv
dbus-pGKMf26gAW
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-colord.service-GbUnUn
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-dbus-broker.service-94ZOZ9
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-kdump.service-waLNMP
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-ModemManager.service-7yMSFI
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-power-profiles-daemon.service-NP07Sj
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-rtkit-daemon.service-FHomNd
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-switcheroo-control.service-QyA5XT
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-systemd-logind.service-S5YYJs
systemd-private-c3bb2399f5814488bf583ccb169e7ffc-upower.service-Cd1DLj
```

Placez-vous dans le répertoire contenant le script et saisissez les commandes suivantes :

```
[trainee@redhat9 tmp]$ cd ~/bin

[trainee@redhat9 bin]$ ./myscript
/home/trainee/bin
myscript

[trainee@redhat9 bin]$ . myscript
/home/trainee/bin
myscript
```

1.2 - La commande read

La commande **read** lit son entrée standard et affecte les mots saisis dans la ou les variable(s) passée(s) en argument. La séparation entre le contenu des variables est l'espace. Par conséquent il est intéressant de noter les exemples suivants :

```
[trainee@redhat9 bin]$ read var1 var2 var3 var4
fenestros edu is great!
```

```
[trainee@redhat9 bin]$ echo $var1
fenestros

[trainee@redhat9 bin]$ echo $var2
edu

[trainee@redhat9 bin]$ echo $var3
is

[trainee@redhat9 bin]$ echo $var4
great!
```

Important: Notez que chaque champs a été placé dans une variable différente. Notez aussi que par convention les variables déclarées par des utilisateurs sont en miniscules afin de les distinguer des variables système qui sont en majuscules.

```
[trainee@redhat9 bin]$ read var1 var2
fenestros edu is great!

[trainee@redhat9 bin]$ echo $var1
fenestros

[trainee@redhat9 bin]$ echo $var2
edu is great!
```

Important : Notez que dans le deuxième cas, le reste de la ligne après le mot *fenestros* est mis dans **$var2**.

Code de retour

La commande **read** renvoie un code de retour de **0** dans le cas où elle ne reçoit pas l'information **fin de fichier** matérialisée par les touches Ctrl+D.
Le contenu de la variable **var** peut être vide et la valeur du code de retour **0** grâce à l'utilisation de la touche Entrée :

```
[trainee@redhat9 bin]$ read var
```

↵ Entrée

```
[trainee@redhat9 bin]$ echo $?
0
[trainee@redhat9 bin]$ echo $var

[trainee@redhat9 bin]$
```

Le contenu de la variable **var1** peut être vide et la valeur du code de retour **autre que 0** grâce à l'utilisation des touches Ctrl+D :

```
[trainee@redhat9 bin]$ read var
```

Ctrl+D

```

[trainee@redhat9 bin]$ echo $?
0
[trainee@redhat9 bin]$ read var

[trainee@redhat9 bin]$ echo $?
1

[trainee@redhat9 bin]$
```

La variable IFS

La variable IFS contient par défaut les caractères Espace, Tab et Entrée :

```
[trainee@redhat9 bin]$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

Important : La commande **od** (*Octal Dump*) renvoie le contenu d'un fichier ou de l'entrée standard au format octal. Ceci est utile afin de visualiser les caractères non-imprimables. L'option **-c** permet de sélectionner des caractères ASCII ou des backslash dans le fichier ou dans le contenu fourni à l'entrée standard.

La valeur de cette variable définit donc le séparateur de mots lors de la saisie des contenus des variables avec la commande **read**. La valeur de la variable **IFS** peut être modifiée :

```
[trainee@redhat9 bin]$ IFS=":"

[trainee@redhat9 bin]$ echo "$IFS" | od -c
0000000   :  \n
0000002
```

De cette façon l'espace redevient un caractère normal :

```
[trainee@redhat9 bin]$ read var1 var2 var3
fenestros:edu is:great!

[trainee@redhat9 bin]$ echo $var1
fenestros
```

```
[trainee@redhat9 bin]$ echo $var2
edu is

[trainee@redhat9 bin]$ echo $var3
great!
```

Restaurez l'ancienne valeur de IFS :

```
[trainee@redhat9 bin]$ unset IFS
```

1.3 - La commande test

La commande **test** peut être utilisée avec deux syntaxes :

test *expression*

ou

[Espace*expression*Espace**]**

Tests de Fichiers

Test	Description
-f fichier	Retourne vrai si fichier est d'un type standard
-d fichier	Retourne vrai si fichier est d'un type répertoire
-r fichier	Retourne vrai si l'utilisateur peut lire fichier
-w fichier	Retourne vrai si l'utilisateur peut modifier fichier
-x fichier	Retourne vrai si l'utilisateur peut exécuter fichier
-e fichier	Retourne vrai si fichier existe
-s fichier	Retourne vrai si fichier n'est pas vide
fichier1 -nt fichier2	Retourne vrai si fichier1 est plus récent que fichier2

Test	Description
fichier1 -ot fichier2	Retourne vrai si fichier1 est plus ancien que fichier2
fichier1 -ef fichier2	Retourne vrai si fichier1 est identique à fichier2

Testez si le fichier **a100** est un fichier ordinaire :

```
[trainee@redhat9 bin]$ mkdir ../training

[trainee@redhat9 bin]$ cd ../training

[trainee@redhat9 training]$ touch a100

[trainee@redhat9 training]$ test -f a100

[trainee@redhat9 training]$ echo $?
0

[trainee@redhat9 training]$ [ -f a100 ]

[trainee@redhat9 training]$ echo $?
0
```

Testez si le fichier a101 existe :

```
[trainee@redhat9 training]$ [ -f a101 ]

[trainee@redhat9 training]$ echo $?
1
```

Testez si /home/trainee/training est un répertoire :

```
[trainee@redhat9 training]$ [ -d /home/trainee/training ]

[trainee@redhat9 training]$ echo $?
```

```
0
```

Tests de chaînes de caractère

Test	Description
-n chaîne	Retourne vrai si chaîne n'est pas de longueur 0
-z chaîne	Retourne vrai si chaîne est de longueur 0
string1 = string2	Retourne vrai si string1 est égale à string2
string1 != string2	Retourne vrai si string1 est différente de string2
string1	Retourne vrai si string1 n'est pas vide

Testez si les deux chaînes sont égales :

```
[trainee@redhat9 training]$ string1="root"

[trainee@redhat9 training]$ string2="fenestros"

[trainee@redhat9 training]$ echo $string1
root

[trainee@redhat9 training]$ echo $string2
fenestros

[trainee@redhat9 training]$ [ $string1 = $string2 ]

[trainee@redhat9 training]$ echo $?
1
```

Testez si la string1 n'a pas de longueur 0 :

```
[trainee@redhat9 training]$ [ -n $string1 ]

[trainee@redhat9 training]$ echo $?
```

```
0
```

Testez si la string1 a une longueur de 0 :

```
[trainee@redhat9 training]$ [ -z $string1 ]

[trainee@redhat9 training]$ echo $?
1
```

Tests sur des nombres

Test	Description
value1 -eq value2	Retourne vrai si value1 est égale à value2
value1 -ne value2	Retourne vrai si value1 n'est pas égale à value2
value1 -lt value2	Retourne vrai si value1 est inférieure à value2
value1 -le value2	Retourne vrai si value1 est inférieur ou égale à value2
value1 -gt value2	Retourne vrai si value1 est supérieure à value2
value1 -ge value2	Retourne vrai si value1 est supérieure ou égale à value2

Comparez les deux nombres **value1** et **value2** :

```
[trainee@redhat9 training]$ read value1
35

[trainee@redhat9 training]$ read value2
23

[trainee@redhat9 training]$ [ $value1 -lt $value2 ]

[trainee@redhat9 training]$ echo $?
1

[trainee@redhat9 training]$ [ $value2 -lt $value1 ]
```

```
[trainee@redhat9 training]$ echo $?
0

[trainee@redhat9 training]$ [ $value2 -eq $value1 ]

[trainee@redhat9 training]$ echo $?
1
```

Les opérateurs

Test	Description
!expression	Retourne vrai si expression est fausse
expression1 -a expression2	Représente un et logique entre expression1 et expression2
expression1 -o expression2	Représente un ou logique entre expression1 et expression2
\(expression\)	Les parenthèses permettent de regrouper des expressions

Testez si $file n'est pas un répertoire :

```
[trainee@redhat9 training]$ file=a100

[trainee@redhat9 training]$ [ ! -d $file ]

[trainee@redhat9 training]$ echo $?
0
```

Testez si $directory est un répertoire **et** si l'utilisateur à le droit de le traverser :

```
[trainee@redhat9 training]$ directory=/usr

[trainee@redhat9 training]$ [ -d $directory -a -x $directory ]

[trainee@redhat9 training]$ echo $?
```

```
0
```

Testez si l'utilisateur peut écrire dans le fichier a100 **et** /usr est un répertoire **ou** /tmp est un répertoire :

```
[trainee@redhat9 training]$ [ -w a100 -a \( -d /usr -o -d /tmp \) ]

[trainee@redhat9 training]$ echo $?
0
```

Tests d'environnement utilisateur

Test	Description
-o option	Retourne vrai si l'option du shell “option” est activée

```
[trainee@redhat9 training]$ [ -o allexport ]

[trainee@redhat9 training]$ echo $?
1
```

1.4 - La commande [[expression]]

La commande **[[**`Espace`**expression**`Espace`**]]** est une amélioration de la commande **test**. Les opérateurs de la commande test sont compatibles avec la commande [[expression]] sauf **-a** et **-o** qui sont remplacés par **&&** et **||** respectivement :

Test	Description
!expression	Retourne vrai si expression est fausse
expression1 && expression2	Représente un **et** logique entre expression1 et expression2
expression1 || expression2	Représente un **ou** logique entre expression1 et expression2
(expression)	Les parenthèses permettent de regrouper des expressions

D'autres opérateurs ont été ajoutés :

Test	Description
string = modele	Retourne vrai si chaîne correspond au modèle
string != modele	Retourne vrai si chaîne ne correspond pas au modèle
string1 < string2	Retourne vrai si string1 est lexicographiquement avant string2
string1 > string2	Retourne vrai si string1 est lexicographiquement après string2

Testez si l'utilisateur peut écrire dans le fichier a100 **et** /usr est un répertoire **ou** /tmp est un répertoire :

```
[trainee@redhat9 training]$ [[ -w a100 && ( -d /usr || -d /tmp ) ]]

[trainee@redhat9 training]$ echo $?
0
```

1.5 - Opérateurs du shell

Opérateur	Description
Commande1 && Commande2	Commande 2 est exécutée si la première commande renvoie un code vrai
Commande1 || Commande2	Commande 2 est exécutée si la première commande renvoie un code faux

```
[trainee@redhat9 training]$ [[ -d /root ]] && echo "The root directory exists"
The root directory exists

[trainee@redhat9 training]$ [[ -d /root ]] || echo "The root directory exists"

[trainee@redhat9 training]$
```

1.6 - L'arithmétique

La commande expr

La commande **expr** prend la forme :

expr Espace value1 Espace *opérateur* Espace value2 Entrée

ou

expr Tab value1 Tab *opérateur* Tab value2 Entrée

ou

expr Espace chaîne Espace : Espace *expression_régulière* Entrée

ou

expr Tab chaîne Tab : Tab *expression_régulière* Entrée

Opérateurs Arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
\(\)	Parenthèses

Opérateurs de Comparaison

Opérateur	Description
\<	Inférieur
\<=	Inférieur ou égal
\>	Supérieur
\>=	Supérieur ou égal
=	égal
!=	inégal

Opérateurs Logiques

Opérateur	Description
|	ou logique
\&	et logique

Ajoutez 2 à la valeur de $x :

```
[trainee@redhat9 training]$ x=2

[trainee@redhat9 training]$ expr $x + 2
4
```

Si les espaces sont retirés, le résultat est tout autre :

```
[trainee@redhat9 training]$ expr $x+2
2+2
```

Les opérateurs doivent être protégés :

```
[trainee@redhat9 training]$ expr $x * 2
expr: syntax error

[trainee@redhat9 training]$ expr $x \* 2
4
```

Mettez le résultat d'un calcul dans une variable :

```
[trainee@redhat9 training]$ resultat=`expr $x + 10`

[trainee@redhat9 training]$ echo $resultat
12
```

La commande let

La commande let est l'équivalent de la commande ((expression)). La commande ((expression)) est une amélioration de la commande **expr** :

- plus grand nombre d'opérateurs
- pas besoin d'espaces ou de tabulations entre les arguments
- pas besoin de préfixer les variables d'un $
- les caractères spéciaux du shell n'ont pas besoin d'être protégés
- les affectations se font dans la commande
- exécution plus rapide

Opérateurs Arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
^	Puissance

Opérateurs de comparaison

Opérateur	Description
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
==	égal
!=	inégal

Opérateurs Logiques

Opérateur	Description
&&	et logique
||	ou logique
!	négation logique

Opérateurs travaillant sur les bits

Opérateur	Description
~	négation binaire
>>	décalage binaire à droite
<<	décalage binaire à gauche
&	et binaire
|	ou binaire
^	ou exclusif binaire

```
[trainee@redhat9 training]$ x=2

[trainee@redhat9 training]$ ((x=$x+10))

[trainee@redhat9 training]$ echo $x
12

[trainee@redhat9 training]$ ((x=$x+20))

[trainee@redhat9 training]$ echo $x
32
```

1.7 - Structures de contrôle

If

La syntaxe de la commande If est la suivante :

```
if condition
then
    commande(s)
else
    commande(s)
fi
```

ou :

```
if condition
then
    commande(s)
    commande(s)
fi
```

ou encore :

```
if condition
then
     commande(s)
elif condition
then
     commande(s)
elif condition
then
     commande(s)
else
    commande(s)
```

```
fi
```

Créez le script **user_check** suivant :

```
[trainee@redhat9 training]$ vi user_check
[trainee@redhat9 training]$ cat user_check
#!/bin/bash
if [ $# -ne 1 ] ; then
  echo "Mauvais nombre d'arguments"
  echo "Usage : $0 nom_utilisateur"
  exit 1
fi
if grep "^$1:" /etc/passwd > /dev/null
then
  echo "Utilisateur $1 est défini sur ce système"
else
  echo "Utilisateur $1 n'est pas défini sur ce système"
fi
exit 0
```

Testez-le :

```
[trainee@redhat9 training]$ chmod 770 user_check

[trainee@redhat9 training]$ ./user_check
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur

[trainee@redhat9 training]$ ./user_check root
Utilisateur root est défini sur ce système

[trainee@redhat9 training]$ ./user_check mickey mouse
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur
```

```
[trainee@redhat9 training]$ ./user_check "mickey mouse"
Utilisateur mickey mouse n'est pas défini sur ce système
```

case

La syntaxe de la commande case est la suivante :

```
case $variable in
modele1) commande
   ...
   ;;
modele2) commande
   ...
   ;;
modele3 | modele4 | modele5 ) commande
   ...
   ;;
esac
```

Exemple

```
  case "$1" in
      start)
          start
          ;;
      stop)
          stop
          ;;
      restart|reload)
          stop
          start
```

```
        ;;
    status)
        status
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status}"
        exit 1
esac
```

Important : L'exemple indique que dans le cas où le premier argument qui suit le nom du script contenant la clause **case** est **start**, la fonction *start* sera exécutée. La fonction *start* n'a pas besoin d'être définie dans **case** et est donc en règle générale définie en début de script. La même logique est appliquée dans le cas où le premier argument est **stop**, **restart** ou **reload** et **status**. Dans tous les autres cas, représentés par une étoile, **case** affichera la ligne **Usage: $0 {start|stop|restart|status}** où $0 est remplacé par le nom du script.

1.8 - Boucles

for

La syntaxe de la commande for est la suivante :

```
for variable in liste_variables
do
    commande(s)
done
```

while

La syntaxe de la commande while est la suivante :

```
while condition
do
    commande(s)
done
```

Exemple

```
U=1
while [ $U -lt $MAX_ACCOUNTS ]
do
useradd fenestros"$U" -c fenestros"$U" -d /home/fenestros"$U" -g staff -G audio,fuse -s /bin/bash 2>/dev/null
useradd fenestros"$U"$ -g machines -s /dev/false -d /dev/null 2>/dev/null
echo "Compte fenestros$U créé"
let U=U+1
done
```

1.9 - Scripts de Démarrage

Quand Bash est appelé en tant que shell de connexion, il exécute des scripts de démarrage dans l'ordre suivant :

- **/etc/profile**,
- **~/.bash_profile** ou **~/.bash_login** ou **~/.profile** selon la distribution,

Dans le cas de RHEL 9, le système exécute le fichier **~/.bash_profile**.

Quand un shell de login se termine, Bash exécute le fichier **~/.bash_logout** si celui-ci existe.

Quand bash est appelé en tant que shell interactif qui n'est pas un shell de connexion, il exécute le script **~/.bashrc**.

A faire : En utilisant vos connaissances acquises dans ce module, expliquez les scripts suivants ligne par ligne.

~/.bash_profile

```
[trainee@redhat9 training]$ cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
```

~/.bashrc

```
[trainee@redhat9 training]$ cat ~/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
```

```
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
if [ -d ~/.bashrc.d ]; then
        for rc in ~/.bashrc.d/*; do
                if [ -f "$rc" ]; then
                        . "$rc"
                fi
        done
fi

unset rc
```

1.10 - Rappel des Expressions Régulières dans Bash

Option	Description
^	Trouver la chaîne au début de la ligne
$	Trouver la chaîne à la fin de la ligne
\	Annuler l'effet spécial du caractère suivant
[ ]	Trouver n'importe quel des caractères entre les crochets
[^]	Exclure les caractères entre crochets
.	Trouver n'importe quel caractère sauf à la fin de la ligne
*	Trouver 0 ou plus du caractère qui précède
\<	Trouver la chaîne au début d'un mot
\>	Trouver la chaîne à la fin d'un mot

Option	Description
?	Trouver 0 ou 1 occurrence de ce qui précède
+	Trouver 1 ou **n** d'occurrences de ce qui précède
{x,y}	Trouver de **x** à **y** occurrences de ce qui précède
{x}	Trouver exactement le nombre **x** d'occurrences de ce qui précède
{x,}	Trouver le nombre **x** ou plus d'occurrences de ce qui précède
{,x}	Trouver le nombre **x** ou moins d'occurrences de ce qui précède
()	Faire un **ET** des expressions régulières entre les paranthèses
|	Faire un **OU** des expressions régulières se trouvant de chaque côté du pipe
[:alnum:]	Caractères alphanumériques : [:alpha :] et [:digit :]; dans la locale 'C' et le codage de caractères ASCII, cette expression est identique à [0-9A-Za-z].
[:alpha:]	Caractères alphabétiques : [:lower :] et [:upper :]; dans les paramètres régionaux 'C' et le codage de caractères ASCII, cette expression est identique à [A-Za-z].
[:blank:]	Caractères vides : espace et tabulation.
[:cntrl:]	Caractères de contrôle. En ASCII, ces caractères ont les codes octaux 000 à 037, et 177 (DEL).
[:digit:]	Chiffres : 0 1 2 3 4 5 6 7 8 9.
[:graph:]	Caractères graphiques : [:alnum :] et [:punct :].
[:lower:]	Lettres minuscules ; dans la locale « C » et le codage de caractères ASCII : a b c d e f g h i j k l m n o p q r s t u v w x y z.
[:print:]	Caractères imprimables : [:alnum :], [:punct :] et espace.
[:punct:]	Caractères de ponctuation ; dans les paramètres régionaux « C » et le codage des caractères ASCII : ! « # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~.
[:space:]	Caractères d'espacement : dans les paramètres régionaux « C », il s'agit de la tabulation, de la nouvelle ligne, de la tabulation verticale, du saut de page, du retour chariot et de l'espacement.
[:upper:]	Lettres majuscules : dans les paramètres régionaux « C » et le codage des caractères ASCII : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.
[:xdigit:]	Chiffres hexadécimaux : 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f.
\b	Faire correspondre la chaîne vide au bord d'un mot.
\B	Correspondre à la chaîne vide à condition qu'elle ne se trouve pas à la périphérie d'un mot.
\<	Correspondre à la chaîne vide au début d'un mot.
\>	Correspondre à la chaîne vide à la fin d'un mot.
\w	Correspondre au mot constituant. Synonyme de [_[:alnum :]].
\W	Correspondre à un constituant non-mot. Synonyme de [^_[:alnum :]].

Option	Description
\s	Correspondre à l'espace blanc. Synonyme de '[[:espace :]`].
\S	Correspondre à un espace non blanc. Synonyme de [^[:espace :]].