Bienvenue dans le cours de script shell linux

1 Quelques bases du terminal

Avant de commencer, essayons de connaître notre terminal. Sachez qu'il existe plusieurs types de terminal. Lorsque vous êtes débutant, le terminal que vous utilisez n'est pas un vrai terminal, désolé de vous décevoir. Le terminal que vous utilisez princialement quand vous prenez en main linux est généralement le terminal "Konsole". Il s'agit d'un émulateur de terminal, c'est à dire une interface graphique utilisant les mêmes programmes qu'un vrai terminal mais avec un visuel plus pratique et plus "jolie". Pour voir de vrai terminal, cherchez l'installation d'une ArchLinux.

"Konsole" est donc logiciel émulateur parmi tant d'autre de diffèrents types (alacritty, kitty) programmé en diffèrents languages (Rust pour l'émulateur Wezterm). Attention, ne confondez pas le language de programmation utilisé pour faire fonctionner votre terminal émulé et le langage interprété utilisé par votre terminal. Le language interprété par votre terminal est le langage entre vous et votre machine, ce sont ce qu'on appelle des languages shells. Ces languages sont assez universel dans le sens où si votre OS (système d'exploitation) est cassé, vous pouvez toujours essayer de le récuperer en accédant au terminal (au vrai !) de votre machine, l'installation d'ArchLinux se fait par exemple purement avec le language interpreté Bash. Bash , le language interprété de base sur tout le système linux. Il en exite tout une famille, chacun intégrant ses propres fonctionnalités, d'autres en reprenant certaines fonctionnalités d'un langage de base et en apportant une touche personnel (oui, de la même façon que vous pouvez créer votre propre distibution linux, vous pouvez aussi créer votre propre language !). On retrouve notamment le ksh, le zsh, le csh ou encore le tcsh. Pour vous faire une idée de la diffèrence, voyez le csh comme une version de Bash codé en C, et tcsh comme une version de csh avec des fonctions supplémentaires.

Vous pouvez donc "parler à votre shell" de diffèrentes façons, le plus important c'est que vous sachiez dans quelle langue vous échanger, c'est plus pratique !

Connaitre le type de language utilisé par votre terminal émulé

```
$ cat /etc/passwd | grep "clement"
        clement:x:1000:1000:clement,,,:/home/clement:/bin/bash
```

Ce que vous voyez, n'est qu'une partie infime du fichier passwd. Ce fichier se situe dans le répertoire /etc/ , il comprend toute les informations concernant les utilisateurs, leur dernière connexion, ainsi que beaucoup de lignes illisibles (voyez par vous même en tapant simplement " cat /etc/passwd ") . La sortie de la commande cat est envoyée (via le pipe |) en entré de la commande grep. Voyez le grep comme une sorte de filtre ou de harçon qui cherchera tout les occurences que vous lui passerez en paramètre. Ainsi, la commande ci dessus vous ressort LA ligne possèdent l'occurence clement dans le fichier passwd. Cette ligne est assez laide, mais simple à comprendre. Elle est composé de diffèrent champs séparés par des ":" . Le premier champs vous indique le nom d'utilisateur. En deuxième position vous retrouvez le mode passe de l'utilisateur (caché par "x"), les 2 nombres qui suivent correspondent respectivement à votre uid ainsi que votre gid (vos identifiant vu par l'ordinateur en tant qu'utilisateur et le groupe auquel vous appartenez) . Le 5ème champs correspond à votre groupe d'appartenance (vous pouvez faire partie de différents groupes). Ensuite, vous retrouvez votre dossier d'utilisateur et enfin le language utilisé par votre shell.

Les diffèrents langages shell à votre disposition sont stockés dans le fichier /etc/shells

```
$ cat /etc/shells

# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
/usr/bin/sh
/bin/ksh93
/usr/bin/ksh93
/bin/rksh93
/usr/bin/rksh93
```

Vous pouvez changer votre shell en utilisant la commande "chsh" . Celle ci vous demandera votre mot de base et ensuite le chemin de votre nouveau shell (doit figuré parmi la liste du fichier shells).

1.1 Quelques fonctions de base

Avant de parler de fonctions essentielles pour discuter avec votre ordinateur, il faut faire une diffèrence entre les commandes internes et externes.

Les commandes internes sont les commandes propres à votre shell, elles ont directement été integré lors de la progrommation de votre terminal.

Les commandes externes sont les commandes utilisable par tout type de shell, que ca soit bash, ksh ...ect. Elles correspondent à des fonction utilisable par tout les programmes que ca soit terminal ou encore un navigateur web. Elles sont généralement installer lors de l'installation de Linux, vous pouvez donc toujours en installer pour les utiliser dans votre Shell.

Pour savoir à quel type appartiennent les fonctions vous pouvez utilisez la commande "type"

```
$ type cat

        cat est haché (/usr/bin/cat)
```

Comme vous pouvez le voir la commande cat (permettant d'afficher un fichier ou un resultat) est une fonction externe, elle se situe dans le dossier /bin (le dossier où se situe tous vos éxecutables).

```
$ type pwd

        pwd est une primitive du shell
```

"pwd" qui vous permet de savoir où vous vous situez est une commande interne au shell

```
$ type ll

        ll est un alias vers « ls -alF »
```

ll est un alias qui fait référence à la commande ls, nous verrons dans la suite l'utilité des alias.

En cas de doute sur une quelconque commande, vous pouvez acceder à la documentation grace à la commande "man". Si man ne marche pas verifiez si les commandes ont une option "- - help" (tiré collé) ou juste "-h"

echo echo [paramètre] [argument]

echo permet d'afficher du texte. il possède diffèrents paramètres :

- "-n" permet de ne pas sauter de ligne jusqu'au prochain echo
- "-e" permet d'utiliser les caractères ayant une action particulière (\t pour tabuler \n pour passer à la ligne ...ect)

Les caractères de substitutions Ils permettent de faire référence à des fichiers ou des objets qu'on ne sait pas entièrement nommer.

caractère	rôle
*	remplace une chaine de longueur variable voir vide
?	remplace un caractère unique quelconque
[. . .]	une série ou une plage de caractère
[a-b]	un caractère parmi la plage indiqué
[! . . .]	inversion de la recherche
[^ . . .]	inversion de la recherche
.	désigne un caractère quelconque
^	désigne un début de ligne
$	désigne une fin de ligne
\	protège le caractère suivant
{nbr}	désigne une répétition du caractère ou de la substition précedente

- fichiers commencants par a : a*

- fichiers de 4 caractères commencant par a : a???
- fichier d'au moins 3 caractères commencant par b : b??*
- fichier finissant par 1 ou 2 : *[12]
- fichier commencant par un caractère dans la plage a-e possèdant au moins un caractère avant la terminaison de 1 ou 2 : [a-e]?*[12]
- fichier ne finissant pas par 3 : *[!3]
- .* : ligne quelconque
- ^$: ligne vide
- ^[123] : ligne commencant par 1,2 ou 3
- *.\\ : ligne contenant un caractère * (protégé), suivi d'un caractère quelconque, suivi d'un caractère \ (protégé)
- $[A-Z]{2} : ligne finissant par 2 majuscules

On peut utiliser ces méta-caractère pour faire de la conversion de caractères :

- conversion minuscule $\Longrightarrow$ MAJUSCULE
  - seulement la première lettre :

```
$ chaine="linux c'est bien"
$ echo ${chaine^}
        Linux c'est bien
```

  - toutes les lettres

```
$ chaine="linux c'est bien"
$ echo ${chaine^^}
        LINUX C'EST BIEN
```

- conversion MAJUSCULE $\Longrightarrow$ minuscules
  - seulement la premiere lettre

```
$ chaine="LINUX C'EST BIEN"
$ echo ${chaine,}
        lINUX C'EST BIEN
```

  - toutes les lettres

```
$ chaine="LINUX C'EST BIEN"
$ echo ${chaine,,}
        linux c'est bien
```

substitution / remplacement de motif On peut remplacer une occurence dans des variables avec "/"

```
$ chaine="une simple chaine"
$ echo ${chaine/simple/longue}
        une longue chaine
$ echo ${chaine/simple/\/}  # attention caractère protégé "/"
        une / chaine
$ echo ${chaine/a*e/aise}
        une simple chaise
```

les caractères protégés correspondent à des caractères qui peuvent être interprétés diffèremment que ce que vous pensez, par exemple si vous voulez recherchez les fichiers pdf , une première approche serai de lister avec la commande “ls “*.pdf”. Cependant le shell va comprendre le point comme un caractère quelconque, par exemple si un fichier s’appele “fichierpdf” , le shell va vous lister le fichier alors que ce n’est pas un fichier avec la bonne terminaison .pdf. La commande “ls “*\.pdf” sera donc plus appropriée (on veut absolument que ce soit le caractère “.” à cet emplacement, en le protégeant).

Si on veut remplacer un même motif plusieurs fois, on utilise //

```
$ chaine="mon premier script"
$ echo ${chaine//r/\?}  #caractère protégé "?"
       mon p?emie? s?ipt
```

find find comme son nom l’indique permet de chercher l’emplacement de fichier ou répertoire.

option	commande
-print	affiche le resultat (le resultat n’est pas afficher à l’utililsateur de base
-exec	execute une action (commande) sur les fichiers trouvés
-name	recherche suivant la chaine
-mtime	recherche date de derniere modif
–atime	recherche date du dernier accées
-ctime	recherche sur la date de création
-newer	recherche les objet plus récent qu’un fichier
-size	recherche critère de taille (find -size +100k : fichier de plus de 100 kilo-octet)
-type	recherche sur le type de l’objet (d pour dossier, f fichier ordinaire...) d’autre existe
-ok	pareil que -exec mais demande la permission
-ls	affiche des info détaillé sur les objets trouvés
-perm	recherche suivant la permission d’accès au fichier
-user, -group	recherche suivant l’appartenance du fichier

-mtime 1 : modif hier, -mtime +2 : modif il y a + de 2j, -mtime -3 ; modif il y a moins de 3 jours

On peut imaginer toute sorte de combinaison pour trouver des fichiers

```
$ find \(-type d -o -type f \) \(-name "*[!5]" -a -name "Ex*\) -size +500k -mtime -10
```

cette commande recherche dans l’ordre :

- l’objet est soit un fichier, soit un repertoire
- le nom de l’objet ne finit pas par 5 et commence par Ex
- sa taille est supérieur à 500 kilo-octets
- la date de modification du fichier est inférieur à 10 jours

Pour executer des actions sur certains fichiers, il faut terminer la commande -exec ou -ok par un “;” et pour que celui-ci soit correctement interprété par le shell, il faut le protégé en mettant un “\”. Pour exécuter la commande sur le fichier trouvé, on substitue par “{}”

```
$ touch .test .exemple .bin
$ find ‘pwd‘ -name ".*" -ok rm {} \;
```

Cette commande va, après avoir demandé la permission, supprimer les fichiers cachés.

Les Filtres

Il existe diffèrents filtres dans linux, nous verrons les plus utiles.

grep grep [option] [fichier1, fichier2...]
Nous avons déja utilisé cette commande, elle permet de chercher toute les occurences d'une chaine passer en paramètre. Elle possède diffèrentes options :

- "-v" effectue la recherche inverse : toutes les lignes ne correspondant pas aux critères passés en paramètre
- "-c" ne retourne que le nombre de lignes trouvées
- "-i" ne différencie pas les minuscules et les majuscules
- "-n" indique le numero de ligne
- "-l" dans le cas de fichier multiple, indique le fichier où apparait l'occurence

```
$ grep -l -v "?*" [fichier2 fichier10]
```

cette ligne recherche les occurences de plus d'un caractère dans les fichier 2 et 10 (attention a la négation) et indique les fichiers.

wc wc [option] [fichier1, fichier2...]
word count, vous compte le nombre de ligne, le nombre de mot ainsi que le nombre de caractère dans un fichier. Vous pouvez bien sûr spécifier si vous voulez un seul de ces résultats.

- "-l" indique seulement le nombre de lignes
- "-w" indique le nombre de mots
- "-m" indique le nombre de caractères

sort sort permet de trier un fichier qui se trouve sous la forme d'un tableau.
sort [option] [-k pos1, pos2...] [fichier1,fichier2...]

option	rôle
-d	tri dictionnaire
-n	tri numérique(pour les valeur)
-b	ignore les espaces en début de champs
-f	aucune diffèrence entre majuscule et minuscule
-r	tri ordre décroissant
-tc	change le délimiteur des champs

```
$ sort -n -r -tc ";" -k 2 fichier1
```

cette commande tri les valeur de la colonne 2 dans l'ordre décroissant du fichier 1, les valeurs étant délimité par le caractère ;

cut dans un fichier, un caractère représente une colonne.
cut [-c pos1, pos2 ..] fichier

- -c2 coupe seulement la colonne 2
- -c1-10 coupe les colonne de 1 à 10
- -c1,3,8 coupe les colonnes 1, 3 et 8

Cependant, cut permet aussi (et surtout) de selectionner un ou des champs

option	role
-d	correspont au délimiteur utilisé
-f	champs à selectionner

```
$ cut -d ":" -f 1,4 etc/passwd
```

cette commande vous affiche les colonnes 1 et 4 du fichier passwd (les champs sont séparé par ":", vérifiez le !) .

uniq supprime les doublons

```
$ cut -d: -f4 fichier1 | sort -n | uniq
```

Dans cette commande les actions suivantes sont faites dans l'ordre :

- selection de la colonne 4 de fichier1 dont les champs sont séparés par :
- la sortie est envoyée à sort qui trie les valeurs numériques dans l'ordre croissant
- la sortie est envoyée dans la fonction uniq qui affiche le résultat sans doublon

paste Permet de concatener un fichier ligne à ligne

```
$ cat fichier1

l1
l2
l3

$ cat fichier2

l4
l5
l6

$ paste -d: fichier1 fichier2
l1:l4
l2:l5
l3:l6
```

Formatage de fichier

tr Permet de substituer des caractères. Cette commande est pratique pour convertir des minuscules en majuscules ou inversement.

```
$ cat liste | tr "[a-z]" "[A-Z]"
```

Tous les caractères minuscules du fichier liste est transformé en majuscule.

expand unexpand Certaines commandes s'attendent à obtenir des tabulations comme séparateur de champs (comme cut par exemple). On a 2 commande à disposition : la commande expand convertit les tabulation en espace et unexpand fait le contraire.

head, tail Pour afficher le début d'un fichier on utilise la commande tail, pour voir la fin, la commande tail.

```
$ head -4 fichier
```

Cette commande affiche les 4 premières lignes de fichier

```
$ tail -4 fichier
```

Cette commande affiche les 4 dernières lignes de fichier

Stockage

du Pour connaître l'espace utilisé par un repertoire, on utilise la commande "du". Elles possèdent 2 options principalement : -k pour afficher la taille en kilo-octet et -s pour faire la somme.

```
$ du -sk *
```

cette commande affiche la taille de chaque repertoire et fichier contenu dans le repertoire à l'emplacement où est exécutée la commande

```
$ du -sk .
```

cette commande affiche la taille total du repertoire.

2 Script Bash, ksh, csh

Variable l'affectation d'une variable se fait avec le signe "=" sans séparation d'espace avant et après le signe. Pour accéder au contenu d'une variable, on place le signe "$" devant. Si on veut affecter une chaine de caractère à une variable, on la place entre guillemet ou entre apostrophe. La différence entre les guillemets et les apostrophes est l'interprétation des variables et des substitutions

```
$ a=Jules
$ b=Cesar
$ c="$a $b a conquis le Gaule"
$ d='$a $b a conquis la Gaule'
$ echo $c
        Jules Cesar a conquis la Gaule
$ echo $d
        $a $b a conquis la Gaule
```

On peut retirer l'affectation d'une variable avec la commande "unset" .

On peut affecter le résultat d'une commande à une variable, par exemple si on prend la cmomande date quiressort la date et qu'on le stocke dans une variable nommée D, on utilise les ` ` (altgr + 7). On peut aussi utiliser $(..).

```
$ D=`date`    #ou D=$(date)
$ cat $D
        19 mars 2023 18h51
```

On peut éliminer ce qui se trouve

- avant un premier motif dans une chaîne avec "#*" et en utilisant les accolades !!
- après le dernier motif, on utilise "%"
- avant la dernière apparition d'un motif on utilise "##"
- après la première apparition d'un motif, on utilise "%%"

```
$ chaine="j'aime linux"
$ echo ${chaine#*' '}
        linux
$ echo ${chaine%i*}
        j'aime l
$ echo ${chaine##*i}
        nux
$ echo ${chaine%%i*}
        j'a
```

Pour récupérer une certaine partie d'un éléments, on utilise les ":".

```
$ chaine="j'aime linux"
$ echo ${chaine:2:10}
        aime linux
$ echo ${chaine:2}
        aime linux
$ echo ${chaine::2}
        j'
```

Accolades et remplacement conditionnel Selon la présence ou non d'une variable, il est possible de remplacer sa valeur par une autre :

remplacement	signification
{x:-texte}	si la variable x est vide ou inexistante, texte prendra la place de x
{x:=texte}	si x est vide ou inexistante alors x prendra la valeur de texte
{x:+texte}	si x est définit et non vide, texte prendra sa place sinon une chaine vide prend sa place
{x:?texte}	si x est vide ou inexistant, le sript est interrompu et le message texte s'affiche

Longueur d'une chaîne On obtient la longueur d'une chaîne avec le caractère "#".

```
$ a=Jules
$ echo "longueur de $a : ${#a}"
        longueur de Jules : 5
```

Tableau et champs

- Cas du BASH

Deux moyen sont disponibles pour déclarer un tableau, l'un avec l'utilisation des crochets [], l'autre avec la création globale. Le premier élement est 0, le dernier 1023. Pour accéder au contenu du tableau, il faut mettre la varible ET l'elément entre accolade {}.

```
$ nom[0]=Jules
$ nom[1]=Romain
$ echo ${nom[0]}
        Jules
```

ou

```
$ nom=(Jules Romain)
$ echo ${nom[1]}
        Romain
```

Pour lister tous les élements, on utilise "*"

```
$ echo ${nom[*]}
        Jules Romain
```

Pour connaitre le nombre d'élements :

```
$ echo ${#nom[*]}
        2
```

Si l'index est une variable, on ne met pas le $ devant celui-ci (pratique pour faire des boucles sur les élements d'un tableau).

```
$ index=0
$ echo ${nom[index]}
        Jules
```

Pour ajouter un élément, on ajoute comme une simple variable avec +=(..) , ne pas oublier les parenthèses.

- cas du KSH

Avec le KSh, on déclare un tableau avec set -A

```
#!/bin/ksh

set -A tab un deux trois
print tab[2]  #on peut utiliser print dans ksh et csh
        deux
```

calcul Les variables peuvent être typée en entier avec la commande typset -i. la commande let ou ((..)) permet des calculs sur les variables.

```
$ resultat=6*7
$ add=5
$ let add+=add resultat*=add
$ echo $resultat
        630
```

On peut effectuer les opérations entre double parenthèse

Argument d'un script Lorsque l'on execute un script, on peut lui passer certains paramètres pour faire leur faire un traitement.

variable	contenu
$0	nom de la commande (du script)
$1-9	paramètre passé en script
$#	nombre total de paramètre passé
$*	liste de tous les paramètres
$@	liste des paramètres sous forme d'élement distinct "$1", "$2",...

```
$ echo $0
$ echo liste $*
$ echo element $@

./Monprog un "deux trois"
        Monprog
        un deux trois
        un deux trois
```

la diffèrence entre $@ et $* est subtile "deux trois" est pris comme un élement simple dans $* alors qu'il est découpé en 2 dans $@.

Test La commande test permet d'effectuer des test de conditions. Le résultat est récupérable par la variable $?

- Test sur des chaînes :
  - test -z "variable" : zero, retour OK si la variable est vide
  - test -n "variable" non zero, retour OK si la variable est non vide
  - test "variable" = chaine : OK si les 2 chaines sont identiques
  - test "variable" ! = chaine : OK si les 2 chaines sont diffèrentes

```
$ a=
$ test -z "$a"; echo $?
        0
$ test -n "$a"; echo $?
        1
$ a=Jules
$ test "$a" = Jules ; echo $?
        0
```

Attention à bien placer les variables contenant du texte entre guillemet. Dans le cas contraire un bug se produira si la variable est vide.

```
$ a=
$ b=toto
$ [ $a = $b ] && echo "ok"
        bash: [: = : unary operator expected
```

Alors que

```
$ [ "$a" = "$b" ] && echo "ok"
```

ne produira pas d'erreur.

- Test sur les variables numériques :

option	role
-eq	equal
-ne	not equal
-lt	less than
-gt	greater than
-le	less or equal
-ge	greater or equal

- Test sur les fichiers :" $ test option nom_fichier"

option	Role
-f	est un fichier normal
-d	est un repertoire
-c	est un fichier en mode caractère
-r	autorisation en lecture
-w	autorisation en écriture
-x	autorisation en execution
-s	fichier non vide
-e	le fichier existe
-L	le fichier est un lien symbolique

saisie de l'utilisateur La commande read permet à l'utilisateur de saisir une chaine et de la placer dans une ou plusieurs variable. La saisie est validé par entré.

option	role
-p	affiche du texte avant la saisie
-n	stop la saisie après n caractère
-t	stop la saisie après t secondes

```
$ read -p "entre la valeur" x
```

Pour faire une saisie dans un tableau non initialisé :

```
$ read -p "entrez les valeurs séparer par un caractère" -a Tab
```

Lecture d'un fichier ligne à ligne On lis un fichier ligne par ligne c'est à dire tant qu'il n'y a pas le caractère retour à la ligne.

```
cat toto.txt | while read ligne
do
        echo $ligne
done
```

ou encore

```
while read ligne
do
        echo $ligne
done < toto.txt
```

if...then...else Si une condition se réalise, on effectue la commande, il faut faire attention à la syntaxe du bash :

```
if [ $# -ne 0 ]    #attention à l'espacement
then
        echo aucun paramètre passé
else
        echo "$# paramètre passé"
fi
```

choix multiple case

```
  case valeur in
   model1) commande ;;
   model2) commande ;;
   *) action par défaut ;;
esac
```

```
case $1 in
        a*) echo "le mot commence par a" ;;
        b*) echo "le mot commence par b ;;
        fic[123]) echo "fic1 , fic2 ou fic3" ;;
        *) echo  "commence par n'importe quoi" ;;
esac
```

```
$ monProg aurevoir
        le mot commence par a
$ monProg fic2
        fic1 , fic2 ou fic3
```

boucle for la boucle for (en bash) ne se base pas sur une quelconque incrémentation numérique mais sur une liste de valeurs, de fichiers.

```
for var in liste
do
commande à executer
done
```

- Avec une variable “$ ./MonProg fichier1 fichier2 ...

```
for param in $@
do
        echo "$param"
done
```

le script réécrit les variables passé en paramètre

- Avec une liste implicite

Si vous ne précisez aucune liste dans la boucle for, alors c'est la liste des paramètres qui est implicite. Ainsi, le script précedent aurait pu ressembler à

```
for param
do
        echo "$param"
done
```

- Avec une liste explicite

Les élement situés après le “in” seront utilisés pour faire la boucle for

```
for param in liste1 liste2 liste3
do
        echo "$param"
done
```

- Critère de recherche dans une arborescence

Si le script se situe dans un repertoire alors le caractère * listera tous les fichiers dans le répertoire.

```
for fic in *
do
        ls -l $fic
done
```

- Avec un interval de valeur

Il existe 2 méthodes pour compter de 1 à n avec une boucle for. La première consiste à utiliser une substitution de commade avec la commande “seq”

```
$ seq(5)
        1
        2
        3
        4
        5

for i in $(seq(5))
do
        echo $i
done
```

La seconde méthode consiste à utiliser la syntaxe proche du C

```
for ((a=1 ; a<=5 ; a++))
do
        echo $a
done
```

Une méthode un peu plus barbare (mais beaucoup plus simple !) consiste à faire une plage de valeur :

```
for i in {0..20}
do
        commande
done
```

Boucle while La boucle while est similaire à la boucle for, en version indeterministe.

```
while condition
do
commande
done
```

boucle select la commande select permet de créer des menus simples avec séléction par énumération. la saisie s’effectue au clavier avec le prompt de la variable PS3.

```
select variable in liste_contenu
do
traitement
done
```

```
PS3="votre choix"
echo "quelle reponse ?"

select reponse in jules romain quitte
do
        if [[ "$reponse" == "$quitte" ]]
        then
                break
        fi
        echo "vous avez choisi $reponse"
done
echo "au revoir"
```

```
$ ./monProg
$ quelle reponse ?
$ 1) jules
$ 2) romain
$ 3) quitte
$ votre choix : 3
$ au revoir
```

Les fonctions

- En bash :

Les fonctions sont des bouts de script nommés, directement appelée par leur nom, pouvant accepter des paramètres et retourner des valeurs

```
nom_fonction ()
{
commandes
return
}
```

- En KSH

on doit preceder le nom de la fonction par “function”