Gestion des processus:

ps : permet de connaitre les processus actifs à un moment donné.
ps-aux : obtenir tous les processus y compris ceux utilisés par le système.
ctrl+z : pour stopper un programme (par exemple firefox)
kill -9 (6062) : pour arrêter un processus qui porte le num 6062.
jobs : pour lancer les travaux qui existent.
fg %2 : foreground (réactiver le processus qui porte le numéro 2).
ctrl+c : arrêter un processus la même tâche que kill -9.

Sauvegarde et archivage avec tar :

tar : permet de sauvergarder un ensemble des fichiers sur un seul fichier.
tar -c : créer un archive.
tar -t : visualiser le contenu de l'archive.
tar -x : extraire ce que contient l'archive.
tar -f : spécifier le nom de l'archive.
tar -z : compresser l'archive en utilisant l'utilitaire gzip.
tar -g : compresser l'archive en utilisant l'utilitaire bzip2.
tar -cf
tar -tf
tar -xf

Compression avec gzip/bzip2 :

gzip : compresser.
gzip -d : décompresser.
gzip --1...-9) : dégré de compression.

L'utilitaire rpm (Redhat Package Manager) permet (installer/désinstaller) des packages facilement.
urpmi package (installer)
urpme package (désinstaller)
urpmq package (plus de détails)

Les alias :

Un alia est une substitution d'une commande par un raccourci.
exemple : alias test='ls -l'
unalias : pour annuler les alias.
unalias -a : annuler tous les alias.

Autres commandes :

whereis : l'emplacement de la commande.
whatis : connaitre la fonction de la commande.
sdiff file1 file2 : comparer les fichiers.

Gestion des utilisateurs :

useradd : permet de créer un utlisateur sur le système
useradd +option (-u;-s;-g;....) nomusr
-u Num d'identifiant de cet utilisateur
-g Le num qui identifie son groupe
-c Pour ajouter un commentaire
-d Pour créer son répertoire de base
-e Date d'expiration du compte
-s Pour spécifier le terminal

SU (compte root) mot de passe : PASSROOT
passwd test (Pour affecter un mot de passe à condition d'être le compte root)
passwd -d user : Supprimer le mot de passe
passwd -l user : Pour vérouiller la session
passwd -u user : Déverouiller la session
usermod -u 505 (Pour changer les modes -u -g -c -d -e)
userdel -r : Supprimer le compte avec son repertoire

id : Permet de savoir le uid le gid du utilisateur courant
Tout ce qui concerne la gestion et l'identification se trouve dans le dossier /etc/passwd
Pour décrire un utilisateur (chfn user changer le commentaire et autre)
Les mots de passe cryptés se trouvent dans le dossier /etc/shadow

Gestion des groupes :

groupadd -g + le nom du groupe : Permet d'ajouter un groupe
exemple : groupeadd -g 600 testgroupe

La gestion des groupes est assurée par : /etc/group
vi /etc/group

Les scripts shell :

1. Tester le shell:

Il s'agit d'une interface texte entre l'utilisateur et le système.
-Tout se fait au clavier
-Pas de clic de souris

L'utilisateur tape des commandes qui sont exécutées par le système.
-Le shell est donc un "interpréteur de commandes"
-Chaque commande a une syntaxe particulière
-Il existe des milliers de commandes différentes

Les commandes peuvent aussi prévenir d'un fichier
-Le fichier contient des commandes à exécuter
-L'utilisateur appelle le fichier plutôt que de taper toutes les commandes (Utile pour les tâches répétitives)

Le Shell reste le moyen le plus efficace pour contrôler le système
-C'est aussi le plus utilisé sous Linux/Unix

Shell ou language de programmation?

Le shell est un véritable environnement de programmation.
-Variables, boucles, structures de contrôle if...
-Programmes
Les programmes écrit pour le shell sont interprétés au moment de l'execution
-Aucune compilation préalable n'est utile
-Les performances n'égalent pas un programme en C
Les programmes écrits pour le shell sont des "scripts" :

Exemple :

```
#Test de l'existence du fichier
if (-f $logfile)
then
rm $logfile
echo effacement de ipscan.log effectué
fi
echo scan des ip...
```

Types de shell:

Il existe plusieurs types de shell:
-Bourne shell
-Korn shell
-Bash (Bourne again shell)
-Tsch(Terminal C shell)
-L'interpréteur de commande MS-DOS (sous windows)
-Powershell (windows 2008 server)
Sous linux, on peut choisir son shell
-Le shell bash domine le marché actuellement.

Personnalisation des commandes bash

- /etc/bashrc étant le dernier script d'initialisation du shell bash
Seul le compte root peut y définir des alias globaux pour tous les utilisateurs
- #vi /etc/bashrc
 -alias file =’touch’
 -alias del=’rm -rf’
 -alias rwx=’chmod 777’
 -:wq (pour écrire dans le fichier et quiter vi)

Puis se reloguer (exit) pour que ces nouvelles commandes soient prises en compte par le nouveau shell.

Les variables :
Sous linux il y a trois types de variables, parmi eux les variables d'environnement.

1**. Les variables d'environnement :**

On peut définir des variables dites d'environnement qui sont des variables contenant des chaînes de caractères. Il n'y a pas de déclaration de type
Comme en C, ces variables contiennent des informations utilisables dans de différents programmes, elles sont définies à l'ouverture de session, leurs
Valeurs dépendent de l'utilisateur connecté.
La commande "printenv" donne la liste des variables d'environnement.

Exemples:
PWD contient le répertoire courant de l'utilisateur connecté.
LANG indique le langage du système.
HOME contient le répertoire d'accueil de l'utilisateur.
SHELL indique le nom du Shell utilisé.
PATH définie les différents chemins où chercher les commandes.

Remarque:

Les commandes qui sont dans le dossier /bin sont des commandes externes.
La commande "LS" est une commande externe, son exécutable se trouve dans le dossier /bin/ls.
echo $PATH permet d'obtenir la valeur de path. On peut modifier cette valeur en y ajoutant un nouveau répertoire.

Exemple:

PATH=$PATH:/home/user/mes scripts

Certaines commandes sont internes et directement traitées par le shell.

Exemple:

cd, test, echo...

Elles n'ont pas de fichier exécutable dans un des répertoires bin.
La commande "type" permet de connaitre le type d'une commande (Interne/Externe).

Suppression et protection :

Unset : supprime une variable

Exemple :

```
a=user1
b=user2
echo "$a et $b" --> User1 et User2
unset b
echo "$a et $b" --> User et
```

On peut protéger une variable en écriture et contre sa suppression avec la commande "readonly". Il n'existe aucun moyen de la remplacer en écriture et de la supprimer sauf acquitter le shell.

Exemple:

```
a=user1
b=user2
readonly a
a=user2
unset a
```

Variables de subsitution :

Elles sont définies implicitement et peuvent être utilisées à tout moment dans le script.
Quelques variables utiles :

$0 : Nom du script (Utile lorsqu'on renomme le script)
$1 à $9 : Argument 1 à 9 passés au script
$# : Nombre d'arguments passés au script
$? : Valeur du code de retour de la dernière commande exécutée. (0 si vrai, autre valeur si c'est faux)
$* : Liste des arguments (Paramètres d'appel de la commande)

Exemple:

```
vi nomscript
echo "Script de test des variables de substitution"
echo "Le nom du script est $0"
echo "Le nombre des paramètres est $#"
echo "Les paramètres sont $*"
echo "1=$1;2=$2;3=$3"
echo "Fin de script"
:wq!
chmod u=x
./test
```

sh test : Afficher le contenu du script
echo $?
0

Exercices :

1. Ecrire un script "Premier.sh" qui affiche un message sur la sortie standard.
2. Ecrire un script "Accueil.sh" qui affiche le nom de l'utilisateur, le nom de la machine, et la date du système. Comment faire pour que ces informations soient affichées à chaque lancement du shell.
3. Ecrire un script "A1.sh" qui affiche le nombre et la valeur des arguments de la ligne de commande.
4. Ecrire un script "A2.sh" qui affiche le nom du script , la valeur du 1er et 4ème argument.

Les expressions arithmétiques :

- expr : Effectue des opérations arithmétiques avec des variables shell, le résultat est affecté à une variable.

N.B: Avant et après le signe (+,-...)
Pour la multiplication, il faut protéger l'opérateur * par / car il fait parti des caractères spéciaux.

Exemple :

```
n=2
echo $n
expr $n+3
```

- let : Effectue aussi des opérations arithmétiques.

Exemple:

```
- let b=1+5
echo $b
- let b=($b*2)/4; echo $b
```

Opérateurs arithmétiques :

Le shell permet d'opérer des calculs arithmétiques, même s'il est moins puissant que d'autres langages (Perl, Scheme, C, etc.) pour cela.

Les opérateurs sont les suivants :

- `-eq` (*equal*) : « égal à » (signe « = ») ;
- `-ne` (*not equal*) : « différent de » (signe « ≠ ») ;
- `-gt` (*greater than*) : « strictement supérieur à » (signe « > ») ;
- `-lt` (*lesser than*) : « strictement inférieur à » (signe « < ») ;
- `-ge` (*greater or equal*) : « supérieur ou égal à » (signe « ≥ ») ;
- `-le` (*lesser or equal*) : « inférieur ou égal à » (signe « ≤ ») ;

Opérateurs sur les fichiers

Une grande partie de la puissance du shell se déploie dans sa faculté de manipuler des fichiers.
Les principaux opérateurs disponibles sont :

- nature du fichier :
  - `-e` (*exists*) : vérifie l'existence d'un fichier ;
  - `-f` (*file*) : vérifie l'existence d'un fichier, et le fait qu'il s'agisse bien d'un fichier au sens strict ;
  - `-d` (*directory*) : vérifie l'existence d'un répertoire ;
  - `-L` (*link*) : vérifie si le fichier est un lien symbolique ;
- attributs du fichier :
  - `-s` (*size*) : vérifie qu'un fichier n'est pas vide ;
- droits sur le fichier :
  - `-r` (*readable*) : vérifie si un fichier peut être lu ;
  - `-w` (*writable*) : vérifie si un fichier peut être écrit ou modifié ;
  - `-x` (*writable*) : vérifie si un fichier peut être exécuté ;
- comparaison de fichiers :
  - `-nt` (*newer than*) : vérifie si un fichier est plus récent qu'un autre ;
  - `-ot` (*older than*) : vérifie si un fichier est plus ancien qu'un autre.

Ou bien tout simplement on peut effectuer des opérations arithmétiques à l'aide des (( )).

Exemple:
echo $((3+7)) --> 10
echo $(((3+7)/5) --> 2

Exercice:

Ecrire un script qui permet de faire l'addition, la soustraction, la multiplication et la division de deux nombres entiers.

1. vi cal.sh
2. #!bin/bash
   echo $(($1+$2))
   echo $(($1*$2))
   echo $(($1-$2))
   echo $(($1/$2))
3. chmod u+x cal.sh
4. bash cal.sh 2 6

Exercice1 :

Ecrire un script qui permet de tester la plus grande valeur entre deux variables A et B et affiche un message comme
quoi (A>B) ou bien l'inverse.

```
#!/bin/bash
echo "Ecrire la première valeur : "
read a
echo "Ecrire la deuxième valeur : "
read b
test $a -gt $b && echo "A>B" || echo "A<B"
```

Exercice2 :

Ecire un script qui permet de tester si l'utilisateur connecté est : Iga et son répertoire d'accueil /home/iga et affiche un message en cas d'erreur.

```
#!/bin/bash
test $USER = IGA && test $HOME = /home/iga && echo "Bienvenue Mr IGA" || echo "vous étes un voleur"
```

Exercice3 :

Ecrire un script qui permet de tester si deux variables A et B sont comprises entre 0 et 20 et affiche un message en cas d'erreur.

```
#!/bin/bash
echo "Ecrire la première valeur : "
read a
echo "Ecrire la deuxième valeur : "
read b
test $a -ge 0 && test $a -le 20 && test $b -ge 0 && test $b -le 20 && echo "Correct" || echo "Erreur!!"
```

Les conditions alternatives (Si Alors, Sinon) :

Syntaxe:

```
If Liste des conditions;Then
  Liste des commandes
Else
  Liste des commandes
Fi
```

Exemple:

Ecire un script qui permet de tester si la valeur entrée est comprise entre 0 et 20 en utilisant if.

```
1. vi ex.sh
2. !#/bin/bash
if [ $# -eq 1 ]
        if test $1 -le 20 && test $1 -ge 0 ;then
echo "La valeur entrée est comprise entre 0 et 20."
else
echo "La valeur est incorrecte"
bash ex.sh
fi
3. bash ex
```

Exercice 1 :

Ecrire un script qui réussit s'il y a un seul argument et si la valeur de l'argument entré est comprise entre 0 et 20.

1. vi ex1.sh
2. !#/bin/bash

```
if [ $# -eq 1 ]
        if test $1 -le 20 && test $1 -ge 0 ;then
        echo "La valeur entrée est comprise entre 0 et 20."
        else
        echo "La valeur est incorrecte"
        fi
else
echo "Entrer un seul argument !!!!  "
fi
```

3. bash ex1

Exercice 2 :

1. Ecrire un script "Existfile.sh" qui comporte un paramètre $1 indiquant le fichier dont on veut connaitre l'existence.
2. S'il existe montrer que c'est un fichier ou un répertoire.
3. S'il existe montrer ces droits d'accès.

Exemple 1 :

Ecrire un script qui retourne une salutation selon la langue de l'utilisateur en utilisant case esac.

```
#! /bin/bash
echo "Entrer votre langue : "
read l

case $l in
francais) echo "Salut";;
Anglais) echo "Hello";;
Espagnol) echo "Hola";;
Arabe) echo "Salam Alaykom";;
*) echo "Vous n'avez choisi aucune langue !!";;
esac
```

Exemple 2:

1. Ecrire un script "Existfile.sh" qui comporte un paramètre $1 indiquant le fichier dont on veut connaitre l'existence.
2. S'il existe montrer que c'est un fichier ou un répertoire.
3. S'il existe montrer ces droits d'accès.

```
#! /bin/bash
echo "Entrer le nom du fichier : "
read a
if [-e $a];then
echo "Entrer l'extension du fichier (f:fichier, d:directory, c:fichier spécial, l:link) : "
read t
case $t in
f) echo "Le fichier $a existe et c'est un fichier!"0
d) echo "Le fichier $a existe et c'est un répertoire!"
c) echo "Le fichier $a existe et c'est un fichier spécial!"
l) echo "Le fichier $a existe et c'est un lien!"
esac
fi
```

EXERCICE Précédent :

1. Ecrire un script qui permet d'afficher un message sur la sortie standard.

```
vi 1.sh
echo "Bonjour tout le monde"
```

2. Ecrire un script qui permet de donner le nom d'utilisateur connecté, le nom de la machine et la date du système. Comment faire pour afficher ces messages à chaque démarrage du shell.

```
vi 2.sh
#!bin/basg
echo "L'utilisateur $LOGNAME(ou bien USER) est connecté sur la machine $HOSTNAME à `date +%Hh:%Mm:%Ss`"
echo "Fin du script"

su root
/etc/bashrc
```

3. Ecrire un script qui permet de donner le nom du script, le nombre de paramètres du 1er et 3ème paramètre.

```
echo "Le nom du script est $0"
echo "Le nombre de paramètres est $#
```

4. Ecrire un script qui permet de lister les paramètres passés, de lancer le 1er et le 3ème script.

```
vi 4.sh
echo "Les paramètres sont $*"
bash 1.sh
```

Boucle for :

Syntaxe:
for var in liste des valeurs; do
liste des cmd
done

Exercice 1:

Ecrire un script qui permet d'afficher un calendrier des années 2000, 2005 et 2010.

```
#! /bin/bash
for var in 2000 2005 2010; do
cal $var
done
```

Exercice 2:

Ecrire un script qui permet d'afficher 3 mois saisis par l'ordinateur des années 2000 2005 2010.

```
#! /bin/bash
for i in 1 : 3; do
echo "Entrer le mois : "
read a
for var in 2000 2005 2010; do
cal $a $var
done
done
```

Exercice 3:

Ecrire un script qui permet d'afficher le contenu de chaque fichier du répertoire $1.

```
#!/bin/bash
echo "le scripte $0 liste tous les fichier du repertoire $1"

if [ $# -ne 1 -o ! -d "$1" ];then
        echo "erreur sur le nom de repertoire"

else

for i in $1/*; do
        if [ -f $i ]; then

                echo -e " ======== le fichier : $i ========== "
                cat $i
        else
                echo "$i n'est pas un fichieir"
        fi

done

fi
```