LE LANGAGE DE SCRIPT DE LINUX

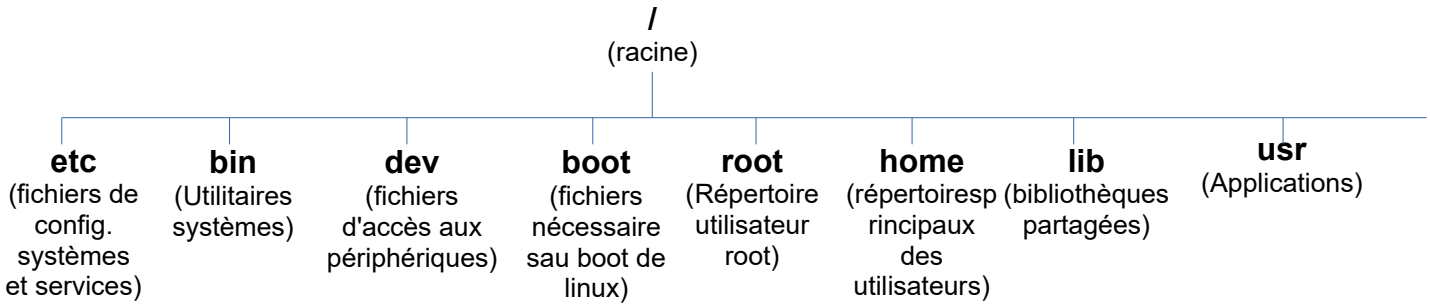
Auteur: Bernard GIACOMONI
Autoentreprise GIACOMONI Bernard

Version	Date	Objet
1.0	10/05/2019	Version initiale

Table des matières

I.RAPPELS:

I.1.ARBORESCENCE DES FICHIERS SOUS LINUX:



Ces répertoires de premier niveau sont utilisés par le système d'exploitation. Ils ne doivent pas être détruits, sous peine de dysfonctionnement.
Rien n'empêche d'ajouter d'autres répertoires à ce niveau ou des sous-répertoires ou fichiers dans les répertoires existants. Il est tout de même recommandé de respecter l'organisation définie.

I.2.MONTAGE D'UN SOUS-SYSTÈME DE FICHIER:

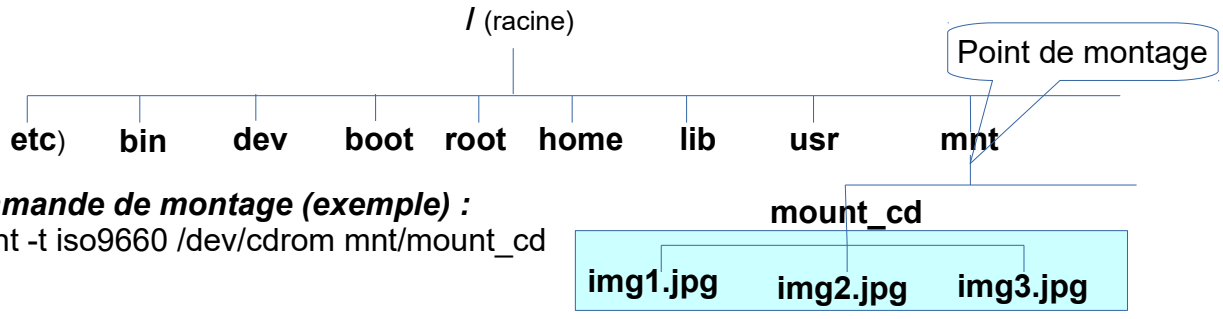
Les systèmes de fichiers contenus par les périphériques de type «mémoire auxiliaire» (tels que les disques durs, les floppy, les cd-roms, les clefs USB...) peuvent être intégrés dans cette arborescence. Il suffit pour cela d'attacher ces systèmes de fichiers internes à un nœud de l'arborescence principale. Cette opération, appelée «**montage d'un système de fichier**», peut s'effectuer à tout moment par la directive système «mount», mais peut aussi être effectuée lors du lancement du système en paramétrant des fichiers de lancement.

REMARQUES:

- *Chaque partition d'un périphériqie constitue une sous-arborescence et peut être montée séparément.*
- *Le montage permet d'intégrer à l'arborescence des systèmes de fichiers de nature différente de celle de l'arborescence principale (ex: partition windows sous linux).*

EXEMPLE:

pour intégrer un CD-ROM, on pourra créer un répertoire mnt sous la racine et un sous répertoire mount_cd puis de monter sur le noeud mont_cd la racine de l'arborescence contenue dans le CD-ROM. Exemple: Cd Rom contenant img1.jpg, img2.jpg et img3.jpg



Commande de montage (exemple) :
mount -t iso9660 /dev/cdrom mnt/mount_cd

I.3.DROITS D'ACCÈS AUX FICHIERS:

A chaque élément de l'arborescence des fichiers (fichier ou répertoire) sont associés 3 files de trois bits qui fixent les droits des différents utilisateurs sur cet élément:

- La première file fixe les droits du possesseur du fichier;
- La deuxième file fixe les droits des membres du groupe auquel appartient le possesseur;
- La deuxième file fixe les droits des autres utilisateurs.

Dans chaque file:

- Le premier bit fixe le droit de lire l'élément;
- Le deuxième bit fixe le droit de modifier (écrire) cet élément;
- Le troisième bit fixe le droit d'exécuter cet élément (s'il s'agit d'un programme exécutable).

Le tableau ci-dessous explicite ces règles:

DÉFINITION DES DROITS D'ACCÈS AU FICHIER «Fic»									
	Droits du possesseur (owner)			**Droits du groupe du possesseur**			**Droits des autres utilisateurs**		
Types de droits	Lecture (R)	Ecriture (W)	Execution (X)	Lecture (R)	Ecriture (W)	Execution (X)	Lecture (R)	Ecriture (W)	Execution (X)
États des bits	1	1	1	1	0	1	1	0	0
Droits des utilisateurs	Le possesseur (owner) de Fic est U1 on lui donne ici le droit de lire, modifier et exécuter Fic			Si U3 et U4 appartiennent au même groupe que le possesseur de Fic on leur donne ici le droit de lire et exécuter Fic (mais non de l'écrire)			U2 n'appartient pas au groupe du possesseur de Fic. Ici, on ne lui donne que le droit de lire Fic.		

NOTA: Il existe un utilisateur privilégie, l'utilisateur «**root**» à qui tous les droits d'accès sont attribués sur toutes les ressources: c'est l'équivalent du compte administrateur de windows.

II.LE LANGAGE DE COMMANDE EN LIGNE DE LINUX:

II.1.AIDE EN LIGNE (man):

> man <commande>

II.2.DATE COURANTE:

> date

II.3.CONTRÔLE DES PROCESSUS:

II.3.1.Lancer un processus en arrière plan (opérateur &)

Placer "&" en fin de commande: ex: >jedit Toto.txt &

II.3.2.Passer un processus d'avant-plan en arrière plan (bg, ctrl-z):

>[ctrl-z] stoppe le processus

>bg le relance en arrière plan

II.3.3.Connaître les processus en arrière plan dans la console (jobs):

>jobs

II.3.4.Repasser un processus en avant-plan (fg):

>fg %<n° processus dans la liste des jobs obtenus par jobs>

II.3.5.Lister les processus en cours (ps):

>ps [-ef]	tous processus
> ps	processus de l'utilisateur
> ps -u <utilisateur>	id ps
> top	liste dynamique des processus

II.3.6.Tuer un processus (ctrl-c, kill):

[ctrl][c]	arrèter le processus lancé en console
kill [-9] <pid1> <pid2>….	-9 pour arréter immédiatement
killall <nom du processus>	Tuer le processus et ses fils

II.3.7.Récupérer le PID d'un processus:

>ps -ef | grep <nom du processus>

Ex: ps -ef | grep jedit

II.3.8.Arrêter et redémarrer le système (halt, reboot, shutdown):

> halt

> reboot

> shutdown

II.3.9.Démarrer et arrêter un DEAMON (ou service):

Présentation:

Dans un système UNIX/LINUX, un DAEMON (souvent prononcé "démon" en Français), désigne un programme informatique qui s'exécute en arrière-plan dans un système d'exploitation. Les daemons, sont le plus souvent démarrés lors du chargement du système d'exploitation et tournent en permanence en attente de requêtes du réseau (ce sont alors des serveurs), de signaux en provenance des couches physiques (ce sont alors des drivers) ou de sollicitations d'autres programmes, sans intervention humaine. Sous Microsoft Windows, les deamons sont appelés «services».

Démarrage, Arrêt, redémarrage d'un deamon:

La syntaxe de démarrage ou d'arrêt manuel d'un deamon est:

> /etc/init.d/<nom du deamon> <start/stop/restart>

Exemple: > /etc/init.d/mysql restart redémarre le serveur MySQL local.

Liste des deamon installés sur une machine:

On peut l'obtenir en listant le répertoire /etc/init.d: > ls -l /etc/init.d. Cette commande permet d'afficher une liste de fichiers exécutables permettant de lancer les traitements afférents à chaque deamon. Ainsi:

- Le fichier "vsftpd" permet de lancer le serveur "verisure file transfert protocol" (serveur ftp sécurisé);
- Le fichier "ssh" permet de lancer le serveur SSH (Sécurite Shell).

Liste des déamons en cours de fonctionnement:

On peut l'obtenir par la commande > ps -A

II.4.UTILISATEURS:

II.4.1.Voir les utilisateurs connectés (w, who, whoami)

>w Lister les utilisateurs connectés

>who Lister les utilisateurs connectés

>whoami voir quel est l'utilisateur courant

II.4.2.Passer en mode super utilisateur

>sudo -s puis saisir le mot de passe super utilisateur

II.4.3.Gérer les utilisateurs (sudo)

> sudo adduser nom_utilisateur Ajouter un utilisateur

> sudo addgroup nom_groupe Ajouter un groupe d'utilisateurs

> sudo deluser nom_utilisateur Supprimer un utilisateur

> sudo delgroup nom_groupe Supprimer un groupe d'utilisateurs

> sudo adduser nom_utilisateur nom_groupe Ajouter un utilisateur à un groupe

Fichier des utilisateurs et mots de passe: /etc/passwd

II.5.FILTRER DES DONNÉES AVEC GREP:

grep [-i] [-n] [-v] [-r] [-E] "chaîne de caractères" <nom du fichier texte>

-i: ne pas tenir compte de la casse
-n: afficher les n° de lignes
-v: inversion (on recherche les lignes qui ne contiennent pas la chaîne)
-r: recherche récursive
-E: recherche d'expression régulière:

.	Caractère quelconque
^	La chaîne qui suit doit être placée en début de ligne
$	La chaîne qui suit doit être placée en fin de ligne
[]	Représente un des caractères entre les crochets
?	L'élément précédent est optionnel (peut être présent 0 ou 1 fois)
*	L'élément précédent peut être présent 0, 1 ou plusieurs fois
+	L'élément précédent doit être présent 1 ou plusieurs fois
|	Ou
()	Groupement d'expressions

Exemples:

Commande	Effet
Grep -E[Aa]lias.bashrc	
Grep -E[0-4].bashrc	
Grep -E[a-zA-Z].bashrc	
Grep -E[a-zA-Z].bashrc	

Trier alphabétiquement des lignes de teste avec sort:

sort [-o] [-r] [-n] <nom du fichier>

-o: fait que le fichier sera modifié par la commande (le résultat du tri est réinjecté dans le fichier)

-r: trier dans l'ordre inverse

-n: trier des valeurs numériques

II.6.RÉSUMÉ DES COMMANDES ESSENTIELLES:

COMMANDES DE GESTION DES RÉPERTOIRES ET DES FICHIERS

Commande	Effet
pwd	affiche le chemin absolu du répertoire courant
ls	affiche les répertoires et les fichiers du répertoire actif) > ls liste le répertoire courant (uniquement les noms) > ls -l liste le répertoire courant (toutes les données) > ls /home/util -l liste le répertoire /home/util (toutes les données)
cd	Change de répertoire > cd .. remonte au répertoire parent > cd /home/util -l Le répertoire courant devient /home/util > cd ~ Remonte au répertoire de base de l'utilisateur
cp	copie des fichiers vers autres fichiers ou répertoire >
mv	déplace et renomme des fichiers > mv Test.txt Test1.txt renomme Test.txt du répertoire courant en Test1.txt dans le répertoire courant. >mv Test.xt .. Déplace Test.txt du répertoire courant dans le répertoire parent. >mv .bat ..\N_.bat Déplace tous les fichiers bat du répertoire courant dans le fichier parent en ajoutant "N_" devant leur nom.
rm	Supprime des fichiers > rm /home/util1/*.exe Supprime tous les fichiers .exe du répertoire /home/util1
mkdir	créer un répertoire
rmdir	supprime un répertoire
grep	Filtre la sortie d'une commande > ls -l | grep "janv" Affiche uniquement les fichiers du répertoire courant créés en janvier.
cat	Liste un fichier de texte
more	affiche page par page sans retour en arrière
diff	Compare des fichiers
read	Lit une variable à partir du clavier
sudo <commande>	Exécute une commande en mode superutilisateur
su <commande>	Permet de changer d'utilisateur > su -s permet de choisir l'utilisateur "root" > su

sudo adduser	ajoute un utilisateur
sudo deluser	supprime un utilisateur
sudo addgroup	Ajoute un groupe
sudo delgroup	supprime un groupe
who	affiche les utilisateurs connectés
whoami	Affiche le nom de l'utilisateur connecté dans la console
ps	liste les processus en cours d'utilisation
chmod	Permet de changer les permissions d'accès aux fichiers Forme: chmod[u g o a][+ - =] [r w x]<nom_du_fichier> > chmod +x Fichier.exe rend le fichier Fichier.exe exécutable pour l'utilisateur, le groupe et tous les autres. > chmod ug-x Fichier.exe rend le fichier Fichier.exe non exécutable pour l'utilisateur et le groupe.
ifconfig	permet de vérifier les paramètres des connexions réseau (adresses ip, masques, passerelles) ou de les configurer.
ping	Teste la réception d'une machine dans le réseau
netstat	Affiche les états des connexions réseau
arp	vérifie let manipule le contenu de la table arp du système

III.SHELL ET SCRIPT SHELLS:

III.1.PRINCIPAUX SHELLS:

- **sh**:*Bourne Shell*. L'ancêtre de tous les shells
- **bash**:*Bourne Again Shell*. Une amélioration du*Bourne Shell*, disponible par défaut sous Linux et Mac OS X
- **ksh**:*Korn Shell*. Un shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.
- **csh**:*C Shell*. Un shell utilisant une syntaxe proche du langage C.
- **tcsh**:*Tenex C Shell*. Amélioration du*C Shell*.
- **zsh**:*Z Shell*. Shell assez récent reprenant les meilleures idées de bash, ksh et tcsh.

Savoir quel shell on utilise: > echo $0

III.2.GENERALITES SUR LES SCRIPT SHELL

III.2.1.Choisi son shell en tête d'un script shell: (#! est appelé sha-bang)

#!/bin/<nom du shell (sh, bash, csh, etc...)>
ex: #!/bin/bash

III.2.2.Insérer des commentaires:

Les commentaires commencent par #

III.2.3.Rendre un script shell exécutable:

> chmod+x <nom du script>.<extension (sh, csh, etc...)>

III.2.4.Exécuter un script comme une commande:

Il suffit de le déplacer ou copier dans un répertoire du PATH (`/bin`,`/usr/bin`ou`/usr/local/bin`, `etc...`).

III.2.5.Debugger un script:

>bash -x <nom du script>
ou bien, utiliser les commandes intégrées à l'intérieur du script:
set -x active mode debug ligne à ligne
set +x stoppe le mode debug

III.2.6.paramètres d'entrée d'un script:

<nom du script> <valeur paramètre n° 1> …. <valeur paramètre n° n>
A l'intérieur du script: valeurs d'un paramètre = $<numero du paramètre>
REMARQUE: $0 est le nom du script.

III.3.VARIABLES DANS UN SHELL:

III.3.1.Déclarer et initialiser une variable:

<nom de variable>=’<valeur>’ (pas d’espace!!!)

ex: VAR=’Adresse’

Pour insérer une apostrophe: utiliser l’antislash: message='c\'estmoi!'

III.3.2.Saisir en ligne la valeur d’une variable:

> read [-p ‘<prompt>’] [-n <nbre caract. Max>] [-s] <nom variable1>….<nom variable n>

-s masque le texte saisi (ex: saisie d’un mot de passe)

EXEMPLE:

> read -p ‘Saisissez votre nom:’ nom

> Saisissez votre nom: ← Saisie de la valeur de nom (ex: Bernard)

> echo"Bonjour$nom!"

Donne: Bonjour Bernard

III.3.3.Afficher ou utiliser la valeur d’une variable:

> message='Bonjour tout le monde'
> echo $message

Résultat: > Bonjour tout le monde

Simples quotes:

> message='Bonjour tout le monde'
> echo 'Le message est: $message'

Résultat: Le message est: $message (le contenu de la variable n’est pas prise en compte)

Doubles quotes:

> message='Bonjour tout le monde'
> echo "Le message est: $message"

Résultat: Le message est: Bonjour tout le monde (la variable est remplacée par sa valeur)

Back quotes: dans le dossier /home/util1:

> message=`pwd`
> echo "Vous êtes dans le dossier $message"

Résultat: vous êtes dans le dossier /home/util1 (la variable message est chargée avec la sortie de la commande pwd).

III.3.4.Substituer une sous-chaîne à une autre dans la valeur textuelle d'une variable:

Il est possible de substituer dans la valeur lue d'une variable textuelle une sous-chaîne à une autre. Cette substitution ne concerne que la valeur lue et non la valeur contenue par la variable. La syntaxe est:

Si on veut remplacer uniquement la première occurrence:

> ${<nom de la variable>/<sous-chaîne à remplacer>/<sous-chaîne remplaçante>}

Si on veut remplacer toutes les occurrence:

> ${<nom de la variable>//<sous-chaîne à remplacer>/<sous-chaîne remplaçante>}

EXEMPLE 1:

```
> Phrase = "Sous le pont Mirabeau coule la Seine"
> echo ${Phrase/Seine/Saône}
```

Affiche: Sous le pont Mirabeau coule la Saône

EXEMPLE 2:

```
> Phrase = "Sous le pont Mirabeau coule la Seine"
> echo ${Phrase//ou/XX}
```

Affiche: SXXs le pont Mirabeau rXXle la Seine

III.3.5.afficher la liste des variables d'environnement:

```
>env
```

III.3.6.Tableaux de variables:

Déclaration:

<nom du tableau>=('valeur N°1' …. 'valeur N°n')

Accès à la valeur i:

${<nom du tableau[i]}

Charger le contenu de la case i:

<nom du tableau>[i] = 'valeur'

Afficher tout un tableau:

${<nom du tableau>[*]}

III.4.OPÉRATIONS MATHÉMATIQUES SUR DES NOMBRES OU DES VARIABLES:

Commande let: let "<expression mathématique avec des nombres ou des variables>"

Exemple:

let "VAR=45"
let "RES=VAR-7"
echo $RES
Donne: 38

III.5.LES CONDITIONS:

III.5.1.Forme générale

If [test] then

    echo "Le premier test a été vérifié"

elif [autre_test] then

    echo "Le second test a été vérifié"

elif [encore_autre_test] then

    echo "Le troisième test a été vérifié"

else

    echo "Aucun des tests précédents n'a été vérifié"

fi

trois types de tests différents :

- Tests sur des chaînes de caractères ;
- Tests sur des nombres ;
- *Tests sur des fichiers.*

Tests sur chaines de caractères (chaînes explicites ou variables) :

if [$1 != $2] then

    echo "Les 2 paramètres sont différents !"

else

    echo "Les 2 paramètres sont identiques !" fi

TABLEAUX DES CONDITIONS:

$chaine1 = $chaine2	Vérifie si les deux chaînes sont identiques. Notez que bash est sensible à la casse : « b » est donc différent de « B ». Il est aussi possible d'écrire « == » pour les habitués du langage C.
$chaine1 !=$chaine2	Vérifie si les deux chaînes sont différentes.
-z $chaine	Vérifie si la chaîne est vide.
-n $chaine	Vérifie si la chaîne est non vide.

If [-z $1] vérifie que $1 est vide

Tests sur des nombres :

$num1 -eq $num2	Vérifie si les nombres sont égaux (**eq***ual*). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères.
$num1 -ne $num2	Vérifie si les nombres sont différents (**n***on***e***qual*). Encore une fois, ne confondez pas avec « != » qui est censé être utilisé sur des chaînes de caractères.
$num1 -lt $num2	Vérifie num1 < num2 (**L**ower **T**han).
$num1 -le $num2	Vérifie num1 ≤ num2 (**L**ower or **E**qual).
$num1 -gt $num2	Vérifie num1 > num2 (**G**reater **T**han).
$num1 -ge $num2	Vérifie num1 ≥ num2 (**G**reater or **E**qual)

Tests sur les fichiers :

-e $nomfichier	Vérifie si le fichier existe.
-d $nomfichier	Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire !
-f $nomfichier	Vérifie si $nomfichier correspond bien à un fichier
-L $nomfichier	Vérifie si le fichier est un lien symbolique (raccourci).
-r $nomfichier	Vérifie si le fichier est lisible (r).
-w $nomfichier	Vérifie si le fichier est modifiable (w).
-x $nomfichier	Vérifie si le fichier est exécutable (x).
$fichier1 -nt $fichier2	Vérifie si fichier1 est plus récent que fichier2 (**N**ewer **T**han).
$fichier1 -ot $fichier2	Vérifie sifichier1est plus vieux que fichier2 (**O**lder**T**han).

Inverser un test :

Le caractère " !" devant la condition inverse celle-ci (exemple:

Instruction conditionnelle à choix multiples :

```
case $1 in
        "Septembre")
                echo "Neuvième mois de l'année"
                ;;
        "Novembre")
                echo "Onzième mois de l'année"
                ;;
        "Décembre")
                echo "Douzième mois de l'année"
                ;;
        *)
                 echo "No de mois inconnu"
                 ;;
esac
```

III.6.BOUCLES DE TRAITEMENT:

III.6.1.boucle tant que ... vraie (while)

```
While [ test ]
do
        <Action a exécuter en boucle>
done
```

(exécutée tant que la condition est vraie)

III.6.2.Boucle tant que … fausse (until)

Until [test]

do

<Action a exécuter en boucle>

done

(exécutée tant que la condition est fausse)

III.6.3.Boucle "Pour" (for)

for variable in 'valeur1' 'valeur2' 'valeur3'

do

<Action à exécuter en boucle>

done

Exemples:

```
for Animal in ‘chien’ ‘chat’ ‘cheval’
do
        echo "la variable vaut: $Animal"
done
```

→ Affiche successivement ‘chien’; ‘chat’ et ‘cheval’

```
forfichierin`ls`
do
        mv *  *-old
done
```

→ transfère tous les fichiers * du répertoire courant dans *-old

III.7.FONCTIONS:

III.7.1.Déclaration: 2 possibilités

```
maFonction ()
{
        bloc d’instructions
}
```

Ou bien:

```
function maFonction
{
        bloc d'instructions
}
```

III.7.2.appel de la fonction:

maFonction <valeur paramètre n° 1> ….<valeur paramètre n° N>

Attention: l'appel à une fonction ne peut se faire qu'après sa déclaration.

IV.LANCEMENT AUTOMATIQUE DE TÂCHES:

IV.1.GÉNÉRALITÉS:

CRON est un outil qui permet de lancer automatiquement des applications (fichiers exécutables *.exe ou *.bat) à des instants définis à l'avance. Cette possibilité peut être utilisée pour lancer des scripts de sauvegarde à des dates et heures précises (périodiquement ou non), pour envoyer des mails à des dates prédéfinies, etc.

CRON fonctionne à partir de fichiers "crontab" qui enregistrent les actions de lancement automatiques.

Un unique fichier crontab est attribué à tout utilisateur qui veut effectuer des activations automatiques de tâches. Ce fichier n'est pas éditable directement.

Le contenu d'un fichier crontab se présente sous la forme de lignes de textes, chacune exprimant les dates et périodicités de lancement d'une action et le chemin d'accès et le nom du fichier exécutable (.bat ou .exe) à lancer.

La commande batch crontab permet de créer, visualiser, éditer ou supprimer le contenu d'un fichier crontab d'un utilisateur et de prendre en compte les actions de lancement.

IV.2.INSTALLATION:

Dans la plupart des linux, CRON est installé par défaut.

NOTA: *Pour s'assurer que le service cron est bien actif dans une machine, il suffit d'exécuter la commande: ps -s | grep "cron"; si le service existe, une ligne contenant la chaîne "cron"doit s'afficher.*
Sinon, l'installation sous UBUNTU s'effectue par la commande bash: apt-get install cron

IV.3.CONFIGURATION:

Les fichiers /etc/cron.allow et /etc/cron.deny permettent de définir les droits des utilisateur vis à vis de CRON:

- Le fichier /etc/cron.allow permet de définir les utilisateurs autorisés à utiliser la commande crontab ;
- Le fichier /etc/cron.deny permet de définir les utilisateurs qui ne sont pas autorisée à utiliser CRON.

Si ces fichiers n'existent pas, seul l'utilisateur ROOT a accès à CRON.

En général, soit ces fichiers n'existent pas, soit un seul de ces fichiers existe (par exemple, seul le fichier /etc/cron.allow existe et celui-ci contient la liste des utilisateurs

habilités à utiliser CRON.

NOTA: *Si un fichier crontab a été attribué à un utilisateur, la suppression de cet utilisateur de /etc/cron.allow n'arrête pas l'exécution de celui-ci. Seule, la suppression du contenu du fichier ou la mise en commentaire des lignes de lancement supprime les activations.*

IV.4.UTILISATION DE LA COMMANDE CRONTAB:

Pour afficher le contenu du fichier crontab de l'utilisateur courant:

>crontab -l

Pour supprimer toutes les actions d'activation de l'utilisateur courant:

>crontab -r

Pour éditer le fichier crontab de l'utilisateur courant:

>crontab -e

Cette commande ouvre le fichier crontab de l'utilisateur avec un éditeur par défaut (en général, c'est vi).

NOTA: *Il est possible de changer cet éditeur par la commande suivante:*

>export EDITOR=<nom de l'éditeur (vi/nedit/gedit/etc.)>

L'éditeur ouvre alors le fichier crontab. Celui-ci peut alors être complété ou modifié. La sauvegarde du contenu déclenche la mise en fonction du nouveau contenu de crontab.

IV.5.SYNTAXE DU FICHIER CRONTAB:

Dans le fichier CRONTAB, chaque action d'activation est décrite sur une seule ligne avec le format suivant:

<minute> <heure> <jour du mois> <mois> <jour de semaine> [<Utilisateur>] <tâche>

Nom du champ	valeurs	Remarques
<minute>	m∈{0-59} ou * ou m-n ou */ ou m,n,p	- Le caractère * signifie "tous/toutes" (toutes les minutes, toutes les heures, tous les mois, tous les jours de la semaine). - m-n signifie "de m à n". - */n signifie toutes les n unités de temps (toutes les n heures par exemple).m,n : pour les unités de temps 2 et 7. - m,n,p, etc.. signifie "pour toutes
<heure>	m∈[0-23] ou * ou m-n ou */ ou m,n,p	
<jour du mois>	m∈[1-31] ou * ou m-n ou */ ou m,n,p	
<mois>	[m∈1-12] ou jan/fe/mar/apr/… ou * ou m-n ou */ ou m,n,p	
<jour de semaine>	m∈[0-6] (0 = sunday) ou m-n ou */ ou m,n,p ou sun/mon/tue/wed/thu/fri/sat ou *	
<Utilisateur>	Nom de l'utilisateur	

		ces valeurs".
<Tâche>	Chemin d'accès au fichier exécutable	

IV.6.DÉMARRAGE ET ARRÊT DU SERVICE:

Par la la commande: > /etc/init.d/cron <start/stop/restart>

IV.7.EXEMPLES D'UTILISATION:

IV.7.1.Exemple 1:

Exécuter tous les vendredi soir à 23h 30 une sauvegarde du système sous utilisateur admin:

30 23 * * wed admin /admin/scripts/sauvegarde.bash

(sauvegarde.bash étant le script qui effectue la sauvegarde).

IV.7.2.Exemple 2:

Exécuter une commande toutes les 5 minutes, mais uniquement le mardi :

*/5 * * * 2 admin /admin/scripts/Signal.bash

IV.7.3.Exemple 3:

Exécuter une commande une fois par an, le premier janvier à 0h 1mn :

1 0 1 1 * /admin/scripts/ScriptBonneAnnee.bash

V.OUVERTURE DE TERMINAUX A DISTANCE:

V.1.LA CONNEXION A DISTANCE SOUS UNIX-LINUX:

Il est possible de définir plusieurs utilisateurs pour un système WINDOWS. Cependant, ces utilisateurs ne peuvent se connecter simultanément à la machine. En effet, WINDOWS est un système MONO-UTILISATEUR: lorsqu'on veut utiliser un système WINDOWS, on commence par se connecter sur l'écran de la machine support (avec un login et un mot de passe). A partir de cet instant, l'utilisateur connecté est le seul à pouvoir exploiter la machine, jusqu'à ce qu'il se déconnecte ou qu'il soit déconnecté par "time-out".

En revanche, les systèmes d'exploitation UNIX et LINUX permettent à plusieurs UTILISATEURS de se connecter SIMULTANÉMENT au système d'exploitation. Ces systèmes sont dits MULTI-UTILISATEURS. La connexion est réalisée par l'ouverture d'une fenêtre particulière appelée TERMINAL SYSTÈME (ou console), soit sur l'écran de la machine, soit sur l'écran d'une machine distante reliée à la première par un média (en général, un réseau local). Ce type de connexion peut être réalisé grâce aux protocoles rlogin, ssh ou telnet et exige que des serveurs correspondant à ces protocoles soient disponibles dans les machines cibles.

REMARQUE: *Il ne faut pas confondre ce mécanisme de connexion d'un utilisateur avec la PRISE DE CONTRÔLE A DISTANCE qui peut être pratiquée entre des machines par des logiciels comme TEAMVIEWER ou le BUREAU A DISTANCE de WINDOWS qui permettent de se substituer à l'utilisateur courant pour réaliser des tâches d'assistance à distance.*

VI.EXEMPLES DE SCRIPT SHELLS:

VI.1.Script créant un tableau de valeurs, puis affichant ce tableau avec une boucle "for"

```
#!/bin/bash
echo $0

TableauJours=( 'lundi' 'mardi' 'mercredi' 'jeudi' 'vendredi'
'samedi' 'dimanche' )

i='0'
while [ ${#TableauJours[$i]} -gt 0 ]
do
          echo  "$i-${TableauJours[$i]}"
          let i=$i+1
done
echo "Terminé"
```

VI.1.1.Script réalisant les 4 opérations arithmétiques de base:

Les variables d'entrées sont: nombre 1, nombre 2 et opération à réaliser (+,-,*,/):

```
#!/bin/bash
echo $0

function Operations
{
     case $3 in
     "+")
          let R=$1+$2
          ;;
     "-")
          let R=$1-$2
          ;;
     ".")
          let R=$1*$2
          ;;
     "/")
          let R=$1/$2
          ;;
     *)
          R='Operande inconnu'
          ;;
     esac
     echo "Le résultat est: $R"
}

CT='o'
while [ "$CT" = "o" ]
do
     read -p 'Saisir le premier nombre:' N1
     read -p 'Saisir le deuxième nombre:' N2
     read -p 'Saisir opérande (+,-,.,/):' OP
     Operations $N1 $N2 $OP
     read -p 'Voulez-vous continuer? (o/n):' CT
done
```