

SDE – TP1 : Le Shell Unix - scripts

Durée encadrée prévue : 4 heures

Compte-rendu à rendre en fin de séance

Introduction et but du TP

Le but de ce TP est de vous familiariser avec le développement de scripts en shell Unix.
Chaque commande est assujettie d'exemples et / ou exercices qu'il est recommandé de faire.

Plan du document

Compte-rendu à rendre en fin de séance.....	1
1. Notion de script.....	2
2. Notion de commande.....	2
3. Dialogue avec l'utilisateur : "echo" et "read"	3
4. Aide en ligne.....	3
5. Fichiers et répertoires.....	3
6. Processus.....	4
7. Redirections et filtres.....	4
8. Commande "grep"	5
9. Commande "test".....	6
9.1. Tests sur les fichiers et les répertoires.....	6
9.2. Tests sur les chaînes de caractères.....	6
9.3. Tests numériques.....	6
10. Commande "cut"	7
10.1. Sélection par colonnes.....	7
10.2. Sélection par champs.....	7
11. Commande "expr"	8
12. Choix simple (if) et choix multiple (case).....	8
13. Boucles (for, while, until) et interruptions de boucles.....	10
14. Processus en shell sous Unix.....	12
15. Processus en shell.....	13
15.1. Processus en interactif et tâches de fond.....	13
15.2. Traitement en tâche de fond.....	13
15.3. Lister les processus actifs.....	13
15.3.1. Processus en interactif.....	13
15.3.2. Processus en tâche de fond.....	14
15.4. Programmer le lancement d'un processus.....	14
15.5. Arrêter un processus.....	14
15.6. Protection contre un signal : la commande trap.....	15
15.7. La commande nice.....	15
15.8. La commande TOP : un utilitaire supplémentaire.....	15
16. Exercice: Communiquons en Morse !.....	16
17. Annexe: en résumé.....	16
17.1. Aide en ligne.....	16
17.2. Fichiers et répertoires.....	17
17.3. Processus.....	17
17.4. Redirections et filtres.....	18

1. NOTION DE SCRIPT

Le Shell offre un langage de programmation interprété qui lui apporte souplesse et rapidité. Tout fichier exécutable comportant un ensemble de commandes est appelé script.

Lorsque vous lancez un script (c'est-à-dire tapez le nom du fichier qui contient une succession d'instructions shell), le Shell interprète les instructions, puis les fait exécuter par le système, sauf si celles-ci sont précédées du caractère # (permet de mettre des commentaires : les caractères suivant le dièse sont ignorés jusqu'au retour à la ligne).


Tous les scripts qui seront désormais créés devront résider dans le répertoire `$HOME/<nobinome>/bin`. Créez-le s'il n'existe pas

Vérifiez également que ce répertoire fait bien partie de votre PATH, ce qui permettra de lancer vos scripts d'où que vous soyez dans l'arborescence. Si besoin, modifiez votre PATH dans votre `.bashrc`!

Exemple de script :

Créer dans votre répertoire, le script nommé **test1** qui comporte les instructions :

```
echo script test1
pwd
ls -l
```

Donner à ce fichier une autorisation d'exécution : `chmod u+x test1`  et le lancer. Vérifier son fonctionnement.

2. NOTION DE COMMANDE


Toute commande que vous donnez au shell a la forme suivante :

`<commande> [param1] [param2] ... [paramn]`

A chaque paramètre argument, on associe un chiffre représentant sa position. Une variable est créée, contenant la valeur de chaque mot de la commande. Le nom de la variable est sa position dans la commande :

`$0` : nom de la commande.
`$1` : valeur de param1.
`$2` : valeur de param2.
...

Une commande retourne toujours un code (0 si OK, autre valeur sinon). Ce code n'est pas affiché, il est stocké dans la variable `$?` (cf. plus bas).

Exemple : lorsqu'on tape la commande `echo encore du C` , les variables suivantes sont créées :

`$0` contient `echo`
`$1` contient `encore`
`$2` contient `du`
`$3` contient `C`

D'autres variables sont également automatiquement créées par le système d'exploitation. Ce sont :

`$#` : nombre de paramètres
`$*` : paramètres sous forme d'une liste de mots

\$? : contient à la sortie d'une exécution le code de retour (cette valeur est 0 par défaut, et peut être modifiée grâce à la ligne `exit x`, où `x` est un entier et sera la valeur de \$?)

Exercice : écrire un script noté `la` qui affiche tous les fichiers du répertoire fourni en argument. Le tester sur votre répertoire de travail.

3. DIALOGUE AVEC L'UTILISATEUR : "ECHO" ET "READ"

On peut établir un dialogue de type question-réponse avec un script en utilisant les deux commandes suivantes :

- `read` pour la lecture d'informations sur l'entrée standard (au clavier).
- `echo` pour l'affichage sur la sortie standard (écran).

Pour l'instruction `read`, les données sont enregistrées dans des variables du shell dont les noms sont transmis en paramètre à la fonction `read`.





```
read <var1> [<var2> ...]
```

Les affectations sont respectives (première valeur dans la première variable, ...). Dans le cas d'une entrée vide (touche return seule), la variable est définie mais avec un contenu vide.

Exercice : Ecrire le script saisie suivant :

```
echo "saisie : "
read a b
echo "a=$a"
echo "b=$b"
```

Le tester dans les cas suivants :

- 1 pas d'entrée : .
- 1 entrée : toto .
- 1 entrée : toto titi .
- 1 entrée : toto titi tata .

4. AIDE EN LIGNE

Toutes les commandes Unix sont documentées dans le système. Le programme `man` permet de consulter les pages du manuel standard. Par exemple, pour avoir la page du manuel correspondant au programme `ls`, il suffit de taper `man ls` dans une fenêtre shell.

aide en ligne	man commande
connaître son identité	whoami
connaître le répertoire courant	pwd

- `man ls`, `man ps`, `man man`

5. FICHIERS ET RÉPERTOIRES

voir le contenu du répertoire courant	ls
voir le contenu d'un autre répertoire	ls répertoire
voir le contenu de manière détaillée	option -l
voir les fichiers cachés	option -a
affichage du chemin du répertoire courant	pwd
changer de répertoire	cd répertoire

répertoire père	cd ..
répertoire précédent	cd -
affichage du contenu d'un fichier	cat fichier
affichage du contenu d'un fichier page par page	more fichier
créer un fichier vide	touch fichier
créer un répertoire	mkdir chemin
copie de fichier	cp fichier copie
créer un lien symbolique	ln -s fichier lien
déplacer ou renommer un fichier	mv départ arrivée
détruire un fichier	rm fichier
détruire un répertoire	rmdir répertoire
détruire un répertoire ainsi que son contenu	rm -r répertoire
changement des droits d'accès	chmod droits fichier
changement de propriétaire d'un fichier	chown nom fichier
comparaison de deux fichiers	cmp fichier1 fichier2
trouver et afficher les différences entre deux fichiers textes	diff fichier1 fichier2
identification de fichiers par leur contenu	file fichier
recherche de fichiers par leur nom, type, dates d'accès, ...	find chemin critère
voir l'espace utilisé par un répertoire	du -hs répertoire
voir l'utilisation des différents systèmes de fichiers de la machine	df -h
voir l'espace qui reste dans son répertoire d'accueil	quota -v

6. PROCESSUS

lancer une commande en tâche de fond	commande &
retrouver la ligne de commande (oublie précédent du &)	C.z + bg
parenthèses () : exécution séquentiellement dans un shell fils.	(ls -l ;pwd ;whoami)
accolades { } : exécution séquentielle dans le shell actif.	{ ls -l ;pwd ; }
chaînage && : la commande2 n'est exécutée que si la commande1 a été exécutée avec succès	commande1&&commande2
chaînage : commande2 exécutée uniquement si commande1 échoue	commande1 commande2
voir la liste des processus de la machine	ps [axu]
suivre la liste des processus	top
envoi d'un signal à un processus	kill signal pid
lancement d'un processus en faible priorité	nice commande
lancement d'un processus avec obtention de temps d'exécution	time commande

7. REDIRECTIONS ET FILTRES

redirection de la sortie d'un programme dans un fichier	commande > fichier
redirection de l'entrée d'un programme depuis un fichier	commande < fichier
redirection d'un programme vers un autre (pipe)	comm1 comm2
recherche de motifs	grep motif fichiers
affichage du début d'un fichier	head
affichage de la fin d'un fichier	tail
tri	sort
affichage page par page	more

calculer la taille de l'entrée (car, lignes)	wc
remplacements de caractères	tr
remplacement de motifs	sed

8. COMMANDE "GREP"

La commande grep recherche les occurrences d'une chaîne de caractères, ou des expressions plus complexes dites "expressions régulières", soit à partir de l'entrée standard (par défaut), soit à partir d'un fichier. C'est une commande d'extraction de type filtre.

Grep veut dire : Global Regular Expressions Printer ou "affichage des expressions régulières"

```
grep [options] <critere> [fichier(s)]
```

Quelques valeurs pour :

l [options] :

- v toutes les lignes ne contenant pas le critère.
- i majuscules et minuscules indifférenciées.
- n numérote les lignes trouvées.
- l ne fournit que les noms de fichier contenant les lignes du critère.
- s masquage des messages d'erreur.

l <critere> :

- [...] représente un intervalle de caractères.
- [^...] complément de la plage indiquée.
- .
- *
- ^ en début de ligne pour la recherche.
- \$ en fin de ligne pour la recherche.

l valeur de retour :

- 0 : recherche fructueuse.
- 1 : aucune occurrence du critère n'a été trouvée.
- 2 : erreur dans la recherche (par ex le fichier n'existe pas).

Exemples

- Rechercher toutes les lignes qui commencent par "#" dans le fichier stdlib.h

```
grep '^#' /usr/include/stdlib.h
```
- Rechercher toutes les lignes qui se terminent par « ; » dans le fichier « stdlib.h »

```
grep ';$' /usr/include/stdlib.h
```
- La combinaison de ^ et de \$ permet de spécifier toute la chaîne à rechercher :

```
grep '^#endif$' /usr/include/stdlib.h
```
- Une application particulière consiste à repérer les lignes vides dans un fichier :

```
grep -n '^$' /usr/include/stdlib.h
```

Exercice

- Rechercher les fichiers contenus dans le répertoire /usr/bin dont le nom contient au moins 4 caractères dont le dernier est r.
- Rechercher les fichiers dans le répertoire /usr/bin dont le nom a une longueur de 4 caractères et se termine par un r.
- Rechercher les fichiers dans le répertoire /usr/bin dont le nom comprend 4 caractères, commence par a, b ou c et se termine par r.

- Rechercher les fichiers dans le répertoire /usr/bin dont les permissions pour le groupe sont 'r-x'.

NB : la commande ls et les pipes peuvent vous être très utiles pour cet exercice...

9. COMMANDE "TEST"

Cette commande, très utilisée dans les scripts, évalue l'expression fournie en argument. Le code de retour est 0 si l'expression est vraie, différent de zéro si elle est fausse. Les expressions concernent des valeurs numériques, des chaînes de caractères ou des fichiers.

9.1. TESTS SUR LES FICHIERS ET LES RÉPERTOIRES

```
test -[option] <nomfich> ↵
```

Options possibles :

- f le fichier est un fichier normal.
- d le fichier est un répertoire.
- p le fichier est un pipe.
- b le fichier est un fichier périphérique bloc.
- c le fichier est un fichier périphérique caractère.
- s le fichier est un fichier de taille non nulle.
- r le fichier est accessible en lecture.
- w le fichier est accessible en écriture.
- x le fichier est exécutable.

Exemples

```
test -d .bash_profile ↵
echo $? ↵ Renvoie 1 car .bash_profile n'est pas un répertoire
test -d /etc ↵
echo $? ↵ Renvoie 0 car /etc est un répertoire.
test -x .bash_profile ↵
test -w /usr ↵
```

9.2. TESTS SUR LES CHAÎNES DE CARACTÈRES

```
test -z chaîne vrai si la chaîne a une longueur nulle.
test -n chaîne vrai si la chaîne n'est pas vide.

test chaîne1 = chaîne2 ↵ vrai si les 2 chaînes sont égales
test chaîne1 != chaîne2 ↵ vrai si les 2 chaînes diffèrent
```

9.3. TESTS NUMÉRIQUES

Les expressions sont converties en valeur numérique avant comparaison.

```
test <n1> -[option] <n2>
```


1 options :

- eq vrai si n1=n2.
- ne vrai si n1!=n2.
- gt vrai si n1>n2.
- ge vrai si n1>=n2.

-lt vrai si $n1 < n2$.
 -le vrai si $n1 \leq n2$.


1 opérateurs logiques :

-a ET
 -o OU
 ! NON

Exemple : `test <expression1> -a <expression2>` 

La valeur retournée est nulle si le test des deux expressions est vrai.

Forme abrégée : Etant donné la fréquence élevée d'utilisation de cette commande, une syntaxe plus légère sera utilisée, avec les crochets. on a l'équivalence :

`test -r fich1 ou [-r fich1]` 

On placera un espace entre les crochets et le texte de la commande.

Exercice : Créer un fichier de script noté `montre` qui reçoit un nom de fichier en paramètre. Ce script fournira le type du fichier, ses autorisations d'accès pour l'utilisateur du script et la taille du fichier (nulle ou non). (utiliser `if` décrit au §3.7)

10.COMMANDE "CUT"

Contrairement à la commande `grep` qui sélectionne horizontalement (par lignes), la commande `cut` procède à une action verticale (par colonne ou par champs).

10.1. SÉLECTION PAR COLONNES

`cut -c<intervalle de colonnes> nom_fichier`

Exemples

<code>cut -c5 fic</code>	colonne 5
<code>cut -c2-4 fic</code>	colonnes 2 à 4
<code>cut -c2, 3 fic</code>	colonnes 2 et 3
<code>cut -c1-3,5 fic</code>	colonnes 1, 2, 3 et 5


10.2. SÉLECTION PAR CHAMPS


Pour définir des champs, il faut préciser le séparateur de ces champs, la valeur par défaut étant la tabulation `cut [-dséparateur] -f<sélect. de champs> nom_fichier`

La sélection de champ obéit aux mêmes règles que celles de la sélection de colonnes.

Exemple : contenu du fichier `fic` :

```
Toto/1250/1:12:94/3
bébert/100/15:10:94/2
dudu/500/11:11:94/3
manu/1560/21:9:94/4
```

commande `cut -d/ -f2 fic` 
 1250
 100
 500
 1560

Exemple : `cut -d: -f1 /etc/passwd` 

Exercice : Ecrire un script noté `match`, qui admet un paramètre (nom d'utilisateur) et qui vérifie si cet utilisateur est connu par le système. On affichera un message donnant le résultat de la recherche.

NB : pour cet exercice, utilisez une copie du fichier `/etc/passwd` : ce fichier contient la déclaration de tous les utilisateurs qui peuvent se connecter sur la machine, leur groupe, leur HOME...

11.COMMANDE "EXPR"


Toutes les variables contiennent du texte. Outre la commande `test` qui convertit les chaînes en valeurs numériques, il existe une commande appelée `expr` qui manipule les nombres.

Syntaxe : `expr <var1> op <var2>`, où `op` représente l'un des opérateurs suivants :

Opérateur :	Fonction :
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la div)

Exemple

```
expr 4 + 3
7
```



La commande `expr` peut également effectuer des comparaisons de valeurs numériques avec les opérateurs suivants :

Opérateur :	Fonction :
=	égal
!=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

La comparaison retourne le résultat de la comparaison sur la sortie standard :

```
Comparaison vraie 1 1
Comparaison fausse 1 0
```

12.CHOIX SIMPLE (IF) ET CHOIX MULTIPLE (CASE)

La fonction du choix simple est assurée par l'instruction `if`. La syntaxe est la suivante :

```
if <commande>
then
    <instructions>
[else
    <instructions>]
fi
```


L'alternative `else ... instructions` est facultative (et c'est pour exprimer cela qu'il y a des crochets...)

Principe de fonctionnement : <commande> est exécutée en premier, et dans l'hypothèse où le code de retour vaut 0, les instructions suivant le `then` sont exécutées. Si le code de retour est différent de zéro, on exécute les instructions suivant le `else` s'il existe, sinon on passe à la suite du programme.

Exemple :

```
if ls $HOME/bin | grep "^test"
then
    echo "trouvé!!!"
else
    echo "pas trouvé!!!"
fi
```

Ce script recherche dans le répertoire `$HOME/bin` les fichiers commençant par `test`, et affiche un message correspondant au résultat de la recherche.

Remarque : il est possible d'emboîter plusieurs `if` (on utilise le `elif` au lieu de `else if`), ainsi que le montre l'exemple suivant :

```
if ls /bin | grep more > /dev/null
then
    echo "more est un fichier de /bin"
elif ls /usr/bin | grep more > /dev/null
then
    echo "more est un fichier de /usr/bin"
else
    echo "more est ni dans /bin ni dans /usr/bin"
fi
```

Le script recherche si la commande `more` se trouve dans le répertoire `/bin` ou dans le répertoire `/usr/bin`.

Attention! avant le mot-clé `"then"`, il faut soit un saut de ligne soit un ;

```
if ls /bin | grep more > /dev/null ; then
    echo "more est un fichier de /bin"
fi
```

Exercice : Ecrire un script noté `existe` qui admet en premier paramètre un nom de fichier et en second paramètre un nom de répertoire. L'appel se fait donc sous la forme `existe a b`. Ce script vérifie si le fichier `a` se trouve dans le répertoire `b` fourni. On utilisera tous les tests utiles (nombre de paramètres, résultat de la recherche) et on n'oubliera pas les commentaires !!!

L'instruction `case` permet de vérifier le contenu d'une variable par rapport à des valeurs.

Syntaxe :

```
case <valeur> in
    <cas1>) <suite_de commandes_1>;
    <cas2>) <suite_de commandes_2>;
    <cas3>) <suite_de commandes_3>;
    ...
    <casn>) <suite_de commandes_n>;
esac
```

Les mots clé `case` et `esac` doivent toujours être les premiers d'une ligne ou d'une commande. Le double point virgule `;;` termine chaque choix du `case`.

La variable notée `<valeur>` est comparée à plusieurs expressions pouvant être

- une suite de caractères.
- un chiffre.
- des métacaractères.
- toute combinaison des cas précédents.

Les métacaractères utilisables sont les suivants :

- `| *` : chaîne de caractères de longueur variable.
- `| ?` : caractère quelconque.
- `| [...]` : plage de caractères.
- `| [!...]` : négation d'une plage de caractères.
- `| |` : combinaison de cas avec OU logique.

Exemple 1 :

```
case $# in
  0)      echo "erreur : pas de parametres ..."
        exit 1 ;;
  1)      critere=$1
        fichier=test1 ;;
  2)      critere=$1
        fichier=$2 ;;
  *)      echo "erreur : trop de parametres ..."
        exit 1 ;;
esac
```

Exemple 2 :

```
case $choix in
  [yYoO]*) echo "OK, bonne reponse" ;;
  [nN]*)   echo "NON, mauvaise reponse" ;;
  *)       echo "ERREUR, saisie invalide" ;;
esac
```

Exercice : Utiliser l'instruction `case` pour écrire un script qui recherche un critère dans un fichier, l'appel se présentant sous la forme `trouve critere fichier`. Dans le cas où le second paramètre est absent, on recherchera le critère dans le fichier `/etc/hosts` par défaut. On affichera également les messages classiques à l'issue de la recherche. On s'attachera à envisager tous les cas de figure au niveau des paramètres.

13. BOUCLES (FOR, WHILE, UNTIL) ET INTERRUPTIONS DE BOUCLES

La boucle `for` nécessite une liste de valeurs dans laquelle elle puise les valeurs successives de la variable de contrôle. Les commandes à exécuter à chaque itération sont encadrées par les mots clé `do` et `done`.

Syntaxe :

```
for <variable> in <liste>
do
    <suite_de commandes>
done
```

La liste de valeurs peut être explicite ou implicite. On peut également utiliser les métacaractères et les substitutions de commandes.

Exemple :

```
for i in *
do
    if test -d $i
    then echo "$i : REPERTOIRE"
    else echo "$i : FICHIER"
    fi
done
```

Dans cet exemple, * représente l'ensemble des fichiers du répertoire courant.

Remarque : en omettant le `in` et la liste, les valeurs sont implicitement constituées par les valeurs des paramètres d'entrée (\$1 à \$n).

Exercice : Rechercher plusieurs numéros d'utilisateurs dans le fichier `/etc/passwd`. Si aucun paramètre n'est fourni, on affiche la liste des numéros d'utilisateur. La syntaxe doit être la suivante :

```
utilisateur [numero]
```

La boucle `while` est employée lorsque l'on veut piloter l'exécution d'une commande en fonction de la valeur de retour d'une autre. Tant que la dernière commande d'une liste placée derrière le `while` retourne 0, la boucle est exécutée.

Syntaxe :

```
while <suite_de_commandes>
do
    <commandes>
done
```

Exemple :

```
while
    echo "saisir un nom de fichier :\c"
    read nomfichier
    [ -z "$nomfichier" ]
do
    echo "ERREUR : pas de saisie"
done
```

Il est possible de réaliser des boucles infinies en utilisant la commande "ne rien faire" matérialisée par les deux points `..`.

```
while :
do
    ...
done
```

Il existe également une instruction complémentaire de `while` notée `until`. Cette instruction étant relativement peu employée, nous ne donnerons que sa syntaxe. L'instruction `until` est exécutée jusqu'à ce que la commande précédent le `do` soit vraie.

Syntaxe :

```
until <suite_de_commandes>
do
    <commandes>
done
```

Remarque : il est possible d'utiliser également les constantes `false` et `true`.

Exercice : Ecrire un script `efface` normal comportant une boucle infinie, qui efface un fichier s'il est normal, en demandant une confirmation. Le nom du fichier est passé en paramètre.

Exercice : Ecrire un script appelé `question` qui reçoit en paramètre le texte d'une question pour laquelle la réponse ne peut être que oui ou non. Le script affiche la question à l'écran et attend la réponse. Si la réponse est oui, le script retourne 0 (cf. `exit`), et 1 si la réponse est non. Si la réponse est autre, le script réitère la question. Vérifiez le code de retour de la dernière instruction avec l'instruction `echo $?`

Les interruptions de boucle

Deux commandes permettent de sortir des boucles avant que celles-ci ne s'achèvent. Ce sont les commandes `break` et `continue`.

- `break` termine définitivement la boucle, qu'il s'agisse d'un `for`, d'un `while` ou de `until`. Il renvoie à la commande suivant la fin de boucle.

- `continue` provoque le passage direct à l'itération suivante. Il fait passer à l'instruction `done` marquant la fin de l'itération.

Remarque : on peut doter la commande `break` d'un paramètre numérique représentant le nombre d'itérations à sauter, mais cette possibilité reste peu employée par suite du manque de lisibilité qu'elle induit.

Exemple :

```
echo "saisir le texte (q = quit, n = continuer) "
while :
do
    echo "Nom : \c"
    read name
    case $name in
        q) break;;
        n) continue;;
    esac
    echo "Rue : \c"
    read rue
    case $rue in
        q) break;;
        n) continue;;
    esac
    ...
done
```

Exercice

Ecrire un script noté `erase` qui correspond à la syntaxe `erase <nom_fich>`, ou `<nom_fich>` peut représenter une liste non vide de noms de fichiers. Ce script utilise le script `question` pour le dialogue avec le programmeur. Dans le cas où `<nom_fich>` est un fichier normal, on demandera confirmation, et dans le cas où `<nom_fich>` est un répertoire, on demandera s'il faut supprimer toute son arborescence.

14.PROCESSUS EN SHELL SOUS UNIX

Le système Unix est un système multitâches car il exécute plusieurs programmes simultanément. Le processus est un concept fondamental des systèmes d'exploitation. C'est un programme en cours d'exécution qui est composé de deux grandes parties :

- Une partie "code" (ou programme),
- Une partie "environnement d'exécution".

15. PROCESSUS EN SHELL

15.1. PROCESSUS EN INTERACTIF ET TÂCHES DE FOND

A la connexion sur une machine Unix, un processus devient tout de suite actif : c'est le shell qui est en attente des instructions que vous allez lui donner par l'intermédiaire de votre terminal. L'entrée et la sortie standards de ce shell sont représentées par le terminal, et la première tâche du shell est d'afficher le prompt (\$) par défaut). Toute commande qui lui est transmise est exécutée par un processus fils particulier et différent du shell de départ. Le shell de connexion attend le signal de fin du processus fils avant d'afficher à nouveau le prompt.

Il existe deux façons de faire exécuter un travail par un shell :

- En *interactif*, c'est le mode standard et c'est lui que nous avons utilisé jusqu'à maintenant : fonctionnement sous forme d'un dialogue questions/réponses ,
- En *tâche de fond*, ce qui signifie que vous donnez une instruction au shell et que vous voulez immédiatement continuer à travailler sans attendre le résultat.

15.2. TRAITEMENT EN TÂCHE DE FOND

Pour exécuter une commande en tâche de fond, il suffit de la terminer par le caractère & . Une fois la commande lancée, le shell retourne le numéro du processus qui exécute cette tâche.

Il est également possible de mettre le dernier processus interactif lancée en pause (^Z) et de le relancer en tâche de fond (bg).

Remarques

- Le processus en tâche de fond ne doit pas attendre de saisie au clavier .
- Il est préférable que les processus en tâche de fond ne retournent pas leurs résultats sur le terminal initial.
- Les processus en tâche de fond dépendent du shell. Si vous quittez le shell par la commande exit ou ^d, vous forcez également la fin des processus en tâche de fond.

Exemple

```
ls -lR / >listetout 2>/dev/null &
[1] 5844          //ls est le 1er processus en tâche de fond
                  //il a le pid 5844

find / -name core -print >lescore 2>/dev/null
                  //find lancé en interactif
^Z                //find est mis en pause
bg                //find se poursuit en tâche de fond
```

15.3. LISTER LES PROCESSUS ACTIFS

15.3.1. PROCESSUS EN INTERACTIF

La commande ps (process status) permet d'informer l'utilisateur sur ses processus en cours d'exécution en interactif par le shell (ou ses fils) :

Exemple :

```
ps
  PID  TTY  STAT  TIME  COMMAND
```

--- --- --- ---- -----

Chaque ligne représente un processus, et pour chaque processus on a les informations suivantes :

PID : Numéro du processus,

TTY : Terminal sur lequel le processus a été initialisé,

STAT : Etat du processus (eg. R pour runnable, S pour sleeping...)

TIME : Temps d'exécution depuis le début.

COMMAND : Commande correspondante.

- cette commande a un grand nombre d'options, dépendant du shell utilisé. En bash, les options les plus courantes sont :

a qui donne également la liste des processus des autres utilisateurs,

u donne le nom de l'utilisateur qui a lancé le processus

l format long, donnant des informations complémentaires dont :

PPID : identifiant du processus père

PRI et NICE : priorité et facteur nice du processus

RSS : taille mémoire prise par le programme, en Ko

...

- La commande ps dispose d'autres options, dont le rôle est le plus souvent d'effectuer une sélection des processus en cours.
 - -uUtilisateur [,Utilisateur,...]
 - -gNomdegroupe[,Nomdegroupe]
 - -pPID[,PID]

15.3.2. PROCESSUS EN TÂCHE DE FOND

Avec ps, les processus lancés en tâche de fond n'apparaissent pas. L'instruction jobs en donne la liste. De plus, l'instruction fg n° ramène processus de numéro n° en tâche "normale".

15.4. PROGRAMMER LE LANCEMENT D'UN PROCESSUS

Il est possible de programmer le lancement d'un processus avec la commande at

```
at 14:00 nov 21 ↵
at> <commande> ↵
at> ^D
at -l ↵ //liste les programmes en attente d'exécution
atrm n° ↵ //supprime le programme numéro n° de la liste
```

15.5. ARRÊTER UN PROCESSUS

La commande kill permet d'envoyer un signal à un processus. Elle a besoin de deux informations :

1. un numéro de signal,
2. un numéro de processus auquel ce signal doit être appliqué.

```
kill [-Numerosignal] PID [PID...]
```

Les signaux les plus couramment employés sont :

- 1 : il est envoyé par le processus parent à tous ses enfant lorsqu'il meurt.

- 2 : Signal d'interruption d'un processus. Il est envoyé par l'activation de la touche `suppr` ou `^C`.
- 3 : Il n'est pas différent du 2, il stocke par contre dans le répertoire courant l'état de la mémoire (core dump).
- 9 : Il ne peut être ignoré par aucun processus et entraîne fatalement sa mort,
- 15 : C'est le signal par défaut de la commande `kill`.

La réception d'un signal par un processus peut entraîner 3 actions :

- le processus s'est protégé du signal, rien ne se passe (on dit qu'il ignore le signal)
- le processus a indiqué auparavant qu'à la réception d'un numéro de signal, il faudra lancer telle fonction - cette fonction est donc lancée
- le processus n'a rien indiqué : il meurt

Trois signaux particuliers : `SIGSTOP` et `SIGTSTP` (`^z`) gèlent l'avancée du processus sans le tuer, `SIGCONT` dégèle le processus.

Exemple :

```
ls -lR / >contenu 2>/dev/null &
ps -u votre_login //n° du processus ls : n°_de_ls
kill n°_de_ls
```

15.6. PROTECTION CONTRE UN SIGNAL : LA COMMANDE `trap`

Un processus peut se protéger contre un signal, en donnant une instruction à faire à la place de se tuer. En shell, cette instruction est la commande `trap`.

Exemple :

```
trap 'ls -il' 2 //protège le processus en cours du signal
2 : A la réception du signal 2, faire ls -il

trap " 2 //remettre le comportement par défaut pour le
signal 2 (=mourir)
```

Exercice

Ecrire un script `pas2pas3` qui se protège contre les signaux 2 et 3 et boucle à l'infini sur aucune action. La réception du signal 2 doit afficher « signal 2 ». La réception du signal 3 doit afficher « signal 3 ». Testez votre script en lui envoyant plusieurs fois ces signaux. Tuez ce processus.

15.7. LA COMMANDE `NICE`

Cette commande permet de rabaisser la priorité d'un processus. On parle de facteur nice d'un processus.

```
nice -valeur commande
```

où valeur peut être entre -19 et 20 (les valeurs négatives sont réservées au root)

Le facteur nice modifié est ensuite affiché par la commande `ps` avec l'option `-l`, dans la colonne de titre NI. La valeur normale est en général égale à 20. Plus la valeur est basse et plus le processus sera exécuté rapidement. Plus la valeur est haute et plus le processus sera lent dans son traitement.

```
nice -10 ls -lR / >liste 2>/dev/null &
ps -l
```

15.8. LA COMMANDE `TOP` : UN UTILITAIRE SUPPLÉMENTAIRE

Cette commande affiche une liste ordonnée des processus en train d'utiliser le temps CPU. Cet affichage est mis à jour à chaque intervalle de temps. Par défaut l'intervalle est de 5 secondes.

```
top [-d intervalle]
```

Exemple :

```
top 
```

Les informations données par top portent sur :

- Nom de l'utilisateur
- identificateur des processus
- identificateur des processus du groupe
- Utilisation du Cpu
-

Exercices:

- Essayer de voir les valeurs d'identification de votre processus shell, en format long, par la commande ps. Vous ne devrez afficher que vos propres processus.
- Que se passe-t-il si vous envoyez à votre shell le signal 15 ? Essayer également avec les signaux 1, 2 et 9.

16. EXERCICE: COMMUNIQUONS EN MORSE !

On se propose d'écrire un programme qui permettra de communiquer en Morse avec notre ordinateur,

grâce aux signaux.
Le signal 2 aura la valeur du tiret
Le signal 3 aura la valeur du point

Le programme est en attente perpétuelle de signaux.

Quand il reconnaît un caractère valide, il l'affiche. Il n'est pas demandé pas de gérer tout l'alphabet morse! Commencez, par exemple, par les caractères suivants:

A: .-
O: ---
S: ...

17. ANNEXE: EN RÉSUMÉ....

17.1. AIDE EN LIGNE

Toutes les commandes Unix sont documentées dans le système. Le programme man permet de consulter les pages du manuel standard. Par exemple, pour avoir la page du manuel correspondant au programme ls, il suffit de taper man ls dans une fenêtre shell.

aide en ligne	man commande
connaître son identité	whoami
connaître le répertoire courant	pwd

- man ls, man ps, man man

17.2. FICHIERS ET RÉPERTOIRES

voir le contenu du répertoire courant	ls
voir le contenu d'un autre répertoire	ls répertoire
voir le contenu de manière détaillée	option -l
voir les fichiers cachés	option -a
affichage du chemin du répertoire courant	pwd
changer de répertoire	cd répertoire
répertoire père	cd ..
répertoire précédent	cd -
affichage du contenu d'un fichier	cat fichier
affichage du contenu d'un fichier page par page	more fichier
créer un fichier vide	touch fichier
créer un répertoire	mkdir chemin
copie de fichier	cp fichier copie
créer un lien symbolique	ln -s fichier lien
déplacer ou renommer un fichier	mv départ arrivée
détruire un fichier	rm fichier
détruire un répertoire	rmdir répertoire
détruire un répertoire ainsi que son contenu	rm -r répertoire
changement des droits d'accès	chmod droits fichier
changement de propriétaire d'un fichier	chown nom fichier
comparaison de deux fichiers	cmp fichier1 fichier2
trouver et afficher les différences entre deux fichiers textes	diff fichier1 fichier2
identification de fichiers par leur contenu	file fichier
recherche de fichiers par leur nom, type, dates d'accès, ...	find chemin critère
voir l'espace utilisé par un répertoire	du -hs répertoire
voir l'utilisation des différents systèmes de fichiers de la machine	df -h
voir l'espace qui reste dans son répertoire d'accueil	quota -v

17.3. PROCESSUS

lancer une commande en tâche de fond	commande &
retrouver la ligne de commande (oublie précédent du &)	C.z + bg
parenthèses () : exécution séquentiellement dans un shell fils.	(ls -l ;pwd ;whoami)
accolades { } : exécution séquentielle dans le shell actif.	{ ls -l ;pwd ; }
chaînage && : la commande2 n'est exécutée que si la commande1 a été exécutée avec succès	commande1&&commande2
chaînage : commande2 exécutée uniquement si commande1 échoue	commande1 commande2
voir la liste des processus de la machine	ps [axu]
suivre la liste des processus	top
envoi d'un signal à un processus	kill signal pid
lancement d'un processus en faible priorité	nice commande
lancement d'un processus avec obtention de temps d'exécution	time commande

17.4. REDIRECTIONS ET FILTRES

redirection de la sortie d'un programme dans un fichier	commande > fichier
redirection de l'entrée d'un programme depuis un fichier	commande < fichier
redirection d'un programme vers un autre (pipe)	comm1 comm2
recherche de motifs	grep motif fichiers
affichage du début d'un fichier	head
affichage de la fin d'un fichier	tail
tri	sort
affichage page par page	more
calculer la taille de l'entrée (car, lignes)	wc
remplacements de caractères	tr
remplacement de motifs	sed