Unix : shell scripts

ASR1 - Systèmes d'exploitation

Semestre 1, année 2009-2010

Département d'informatique HCM
IUT Bordeaux 1

Janvier 2010

Première partie I

Premiers pas, utilisation interactive

Vous connaissez déjà...

- emacs/vi, g++,
- cp, rm, mv, ls
- cat, less, more
- mkdir, rmdir, cd, pwd
- echo, clear
- man
- ...

Redirection

Les commandes produisent du texte sur leur sortie standard

Exemples de commandes

```
echo "Bonjour"
ls
date "+%H:%M"
```

Ce texte peut être redirigé vers un fichier

Exemples : redirections sortie standard

```
echo "bonjour"  > message.txt
date "+%H:%M"  >> message.txt
```


Redirections sortie standard

- Remplacement d'un fichier : `>`

Exemple

```
echo "bonjour"   >  message.txt
```

- Extension d'un fichier : `>>`

Exemple

```
date >> fichier
```


Sortie d'erreur

- Certains messages sont produits sur la sortie d'erreur

Mise en évidence

```
$ g++ essai.cc > resultat.log
essai.cc:1: error: ISO C++ forbids declaration of
'exemple' with no type
$
```

- Redirection sortie d'erreurs `2 >` `2 >>`

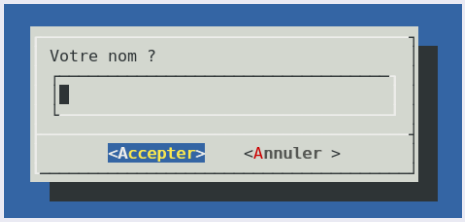
Exemple

```
$ g++ mon-programme.cc 2> erreurs.txt
$
```


Autres usages de la sortie d'erreur

Exemple

```
$ dialog --inputbox "Votre nom ?" 8 40  2> /tmp/nom
```



Redirection de l'entrée standard

- Depuis un fichier : `<`

Exemple

```
tr '[a-z]' '[A-Z]'   <  texte.txt
```

- "here document" : `<<`

Exemple

```
$ tr '[a-z]' '[A-Z]'   <<XXX
ceci Est un
exemple
XXX
```


Redirection entre deux commandes

L'opérateur `|` (*pipe*) redirige la sortie standard d'une commande vers l'entrée standard d'une autre commande

Exemple

```
w | cat -n
```

On peut constituer un pipeline de plusieurs commandes

Exemple : redimensionner une image

```
anytopnm dscn3214.jpg |
    pnmscale -width 100 |
        pnmtopng > statue-hcm-100px.png
```


Droits d'accès

La commande `ls -l` montre les droits d'accès

Exemple

```
billaud@feathers:~/Essais/C++$ ls -l
total 64
drwxr-xr-x 2 billaud profs  4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs  6495 nov  2 21:19 initTab
-rw-r--r-- 1 billaud profs   236 nov  2 21:19 initTab.cc
...
```

Premier caractère

- `d` pour les répertoires (*directory*)
- `-` pour les fichiers

Droits d'accès : suite

Exemple

```
drwxr-xr-x 2 billaud profs  4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs  6495 nov  2 21:19 initTab
-rw-r--r-- 1 billaud profs   236 nov  2 21:19 initTab.cc
```

Lettres suivantes : indiquent les droits d'accès Présentation par groupe de trois :

`-` `rwx` `r-x` `r-x`

- `rwx` pour le propriétaire du fichier (`billaud`)
- `r-x` les utilisateurs du groupe `profs`
- `r-x` pour les autres

Les droits d'accès

Les lettres indiquent les droits d'accès (ou mode, ou permissions)

- `r` pour read (droit de lecture)
- `w` pour write (droit d'écriture, modification)
- `x` pour
  - execute (droit d'éxécution) pour les fichiers,
  - x=cross (droit de traverser) pour les répertoires.

Exemples :

Exemple

```
drwxr-xr-x 2 billaud profs  4096 août 29 12:22 Heap
-rwxr-xr-x 1 billaud profs  6495 nov  2 21:19 initTab
-rw-r--r-- 1 billaud profs   236 nov  2 21:19 initTab.cc
```

- `iniTab.cc` peut être lu et modifié par son propriétaire (rwx), lu par les membres du groupe (r) et par les autres (r).
- `a.out` peut être lu, modifié et exécuté par son propriétaire (rwx), lu et exécuté par les utilisateurs du groupe (rx) ainsi que par les autres (rx).
- le répertoire `Heap` peut être lu, modifié et traversé par le propriétaire (rwx), lu et traversé par les membres du groupe (rx) et les autres.

chmod : changer les droits d'accès

La commande "`chmod`" change les droits d'accès (CHange MODe)

Notation octale

Exemple : `chmod 750 mon-fichier`

- chaque groupe de trois bits est codé en octal, avec r=4, w=2, x=1.
- Donc `chmod 750 ...` donne les droits `rwx` `r-x` `---` au fichier

Exercices

Exercice : notation octale

Complétez la table d'équivalence

octal	droits	octal	droits
0	- - -	4	
1		5	r - x
2		6	
3		7	r w x

Exercices

Exercice : droits sur les fichiers

1. Tapez
   `ls -l > mon-fichier`
   `chmod 777 mon-fichier`
   - pouvez-vous lire le fichier (`cat mon-fichier`) ?
   - le modifier (`echo >> mon-fichier`) ?
2. même question après
   `chmod 666 mon-fichier`
3. ...

droits	lire ?	modifier ?
000	non	non
111		
222		
333		
444		
555		
666		
777	oui	oui

Exercices

Exercice : droits

1. Tapez
   `ls -l > mon-fichier`
   `chmod 777 mon-fichier`
   - pouvez-vous lire le fichier ?
   - le modifier ?
2. Changez les droits
   `chmod 077 mon-fichier`
   - pouvez-vous le lire ?
   - le modifier ?
3. Conclusions ?

chmod : notation symbolique

Exemple

```
chmod u=rwx,g=rx,o= mon-fichier
```

- u user (propriétaire)
- g groupe
- o others (autres)

chmod : modification des droits

Sous forme symbolique, permet de modifier *certains* droits.

- + ajouter des droits
- - enlever

Exemple

`chmod go-w f`
enlève le droit `w` au groupe (g) et aux autres (o), sans changer les autres permissions.

Exemple

`chmod +x f`
ajoute les droits "x"

Quelques commandes

- grep : sélection de lignes
- cut : sélections de colonnes
- sort : tri
- join : jointure

la commande grep

Sélectionne des lignes d'un texte, qui contiennent un certain motif

exemple

- Soit le fichier `villes.txt`

```
1 france:paris
2 vietnam:ho chi minh
3 italie:roma
4 france:bordeaux
5 vietnam:hanoi
6 inde:delhi
```

- la commande
  `grep italie villes.txt`
  affiche les lignes du fichier qui contiennent `italie`

grep

villes.txt

```
1 france:paris
2 vietnam:ho chi minh
3 italie:roma
4 france:bordeaux
5 vietnam:hanoi
6 inde:delhi
```

Essayer

```
grep 'i'  villes.txt
grep ':h' villes.txt
```


grep : ancrage

Les caractères ^ et $ servent à "ancrer" le motif de recherche

- au début d'une ligne ^
- ou à la fin de la ligne $

Essayer

```
grep '^i' villes.txt
grep 'i$' villes.txt
```

La commande cut

La commande cut sélectionne une *colonne* de données.

Essayer

```
cut -c 1-3 villes.txt
cut -d: -f1
grep vietnam villes.txt | cut -d: -f2
```


la commande sort

Ordonne les lignes selon un critère

```
sort villes.txt
sort -t: -k2 villes.txt
```


Exercices "sort"

Exercice

- Créer un fichier `villes-pays.txt` (villes ordonnées par pays)
- Créer un fichier `continents-pays.txt` (continents ordonnées par pays) à partir de `continents.txt`

```
1 europe:france
2 europe:italie
3 asie:vietnam
4 asie:chine
```


Commande "join"

Rapproche deux fichiers sur une clé commune.
Les fichiers doivent être triés

Exemple

```
join -t: -1 2 -2 1 continents-pays.txt villes-pays.txt
```

- `-t :` : délimiteur de champs
- `-1 2` : clé du premier fichier = second champ
- `-2 1` : clé du second fichier = premier champ

Deuxième partie II

Shell-scripts : introduction

Qu'est ce qu'un *shell*

"shell" : programme qui

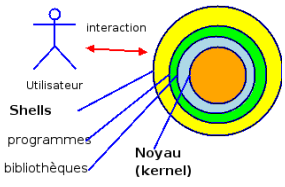
- lit des lignes de commandes
  - tapées par l'utilisateur
  - ou prises dans un fichier
- les fait exécuter

Autre nom : interprète de commandes

Les shells (suite)

Les shells

- ne font pas partie du noyau du système,
- utilisent le noyau pour exécuter des applications, créer des fichiers etc.



Les shells

De nombreux shells sont disponibles sous Unix/Linux,

- sh
- bash (Bourne again shell),
- CSH (C Shell),
- KSH (KORN Shell),
- TCSH
- ...

Ils

- jouent le même rôle,
- ont des syntaxes différentes,
- fournissent des fonctions prédéfinies différentes.

En pratique

- Le fichier `/etc/shells` contient la liste des shells disponibles
  `$ cat /etc/shells`
- pour savoir quel shell vous utilisez, tapez
  `$ echo $SHELL`
- Pour connaaître votre shell par défaut
  `$ finger -l`

Usage interactif / scripts

Usage interactif

1. vous tapez une commande
2. le *shell* l'interprète
3. ...

Scripts

1. vous tapez des commandes dans un fichier texte (script)
2. vous demandez l'exécution de ce script

Shell script

Définition : Shell-script = fichier texte qui contient une suite de commandes.

Exemple : fichier "premier.sh"

```
1 #
2 # Mon premier essai
3 #
4 clear
5 echo -n "Nous sommes le "
6 date
7 echo "et c'est mon premier script"
```

Note : le caractère `#` indique un commentaire

Shell script

Exemple : fichier "premier.sh"

```
1 #
2 # Mon premier essai
3 #
4 clear
5 echo -n "Nous sommes le "
6 date
7 echo "et c'est mon premier script"
```

Exécution par

`$ bash premier.sh`

ou

```
$ chmod +x premier.sh
$ ./premier.sh
```


Comment écrire un script

1. utiliser un éditeur de textes pour écrire le script
   Remarque
   - le suffixe .sh est recommandé
2. le rendre exécutable
   - `chmod +x mon-fichier.sh`
   - `chmod 755 mon-fichier.sh`

Démonstration

1. on tape le script suivant

```
1 #!/bin/bash
2 #
3 # fichier premier.sh
4 #
5 clear
6 echo "les scripts, c'est facile"
```

2. exécution par
   `bash ./first.sh`

Recommandation : la première ligne commençant par `#!` indique quel *shell* il faut utiliser.

Suite

1. lancement par
   `./first.sh`
   ne marche pas, parce que le script n'est pas exécutable
2. Rendre le fichier exécutable
   `$ chmod +x first`
   `$ ./first`

Pourquoi écrire des scripts ?

Intérêt

- Permettent de réutiliser des suites de commandes sans risque d'erreur
- Gagner du temps
- Automatiser les tâches fréquentes
- ...

Applications

Applications des scripts

- Créer ses propres commandes à partir de commandes existantes
- Scripts d'installation
- Fonctionnement du système d'exploitation (ex : lancement de script au démarrage)
- Aide à l'administration du système (tâche répétitives) exemple :surveillance des quotas
- ...

Troisième partie III

Variables, paramètres, expressions...

Variables

Les variables du *shell* mémorisent des chaînes de caractères.

- la liste des variables est affichée par set
- Variables système définies automatiquement : HOME PWD SHELL USERNAME PATH LANG etc.

Affectation / Expansion

- affectation de variable : `NOM=`[CHAINE]
- expansion par "`$NOM`" ou "`${NOM}`"

Exemple

```
1 message="Bienvenue parmi nous"
2 echo $message
```


Variables : exemple

```
1  #! /bin/bash
2
3  echo "Bonjour  $USER"
4  echo -n "aujourd'hui "
5  date
6
7  france="Europe/Paris"
8  vietnam="Asia/Ho_Chi_Minh"
9
10 echo -n "heure Vietnam = "
11 TZ=$vietnam date
12
13 echo -n "heure France = "
14 TZ=$paris" date
15 exit 0
```


Variables système

Essayez

Avec la commande set, trouvez les variables qui indiquent

- le nom de votre poste de travail,
- son type,
- la version du système d'exploitation ?

Environnement, export

- un processus s'exécute dans un environnement (valeurs de variables)
  commande **printenv**
- les variables système sont transmises (exportées) automatiquement lors d'un appel de script
- les autres doivent être exportées explicitement (= ajoutées à l'environnement)
  commande **export** NOM[=VALEUR]
- liste des variables exportées : **export**

Tableaux

À la différence de sh, l'interprète bash possède des tableaux

Exemple

```
1 declare -a pays
2 pays[0]=Australie
3 pays[1]=Vietnam
4 pays[2]=France
5 pays[3]=Allemagne
6
7 i=3
8 j=1
9 echo football : ${pays[$i]} contre ${pays[$j]}
```


Paramètres positionnels

Invocation d'un script

Un script peut être invoqué avec des paramètres, exemple :
`./mon-script Hanoi Paris Bordeaux "Ho Chi Minh City"`

Pendant l'exécution

- `$1="Hanoi"`,
- `$2="Paris"`,
- `$3="Bordeaux"`,
- `$4="Ho Chi Minh City"`

Paramètres positionnels (suite)

Autres variables utiles

- `$# = 4` – le nombre de paramètres
- `$* = Hanoi Paris Bordeaux "Ho Chi Minh City"'`
- `$0 = "./mon-script"` – le nom du script

Exercice

Ecrire un script à un paramètre qui indique les villes d'un pays.

```
$ villes.sh  france
paris
bordeaux
...
$
```

Utiliser le fichier de données précédent.

Affectation

Met une valeur dans une variable.

Noms de variable

- lettres, des chiffres, blancs soulignés
- ne commence pas par un chiffre
- MAJUSCULES/minuscules différenciées

Affectation (suite)

Deux formes

- NOM=CHAINE
  affectation simple
- **let** NOM=EXPRESSION
  affectation du résultat d'un calcul arithmétique

`let` est une commande interne du *bash* (`help let`)

Affectations (suite)

Essayez

```
1 a=12
2 b=42
3
4 c=a+b
5 echo $c
6
7 d=$a+$b
8 echo $d
9
10 let e=a+b
11 echo $e
```

Lecture de variables

La commande

read v1 v2 ...

lit une ligne au terminal, et affecte les mots dans les variables citées.

Exemple

```
read nom prenom
```


Essais avec read

Essayez

- `read nom prenom`
- que se passe-t-il si on tape plus de mots qu'il n'y a de variables ?

Exercice read

Exercice

Écrire un script qui

- demande l'année de naissance,
- imprime l'âge.

Exemple d'exécution

```
$ ./quel_age
Votre année de naissance ?
1984
Vous êtes né en 1984, vous avez donc 25 ans.
```


Séparateur

La variable IFS

indique le séparateur reconnu par `read` (input field separator)

```
$ export IFS=,
$ read NOM PRENOM
einstein,albert
$ echo $PRENOM
albert
```

Alternative à "export"

Affectation temporaire :

```
IFS=, read NOM PRENOM
```

valide pendant la durée d'exécution du `read`

Expansion

Définition

Expansion : remplacement d'une expression par sa valeur

Exemples

- expansion de variables :

  ```
  echo bonjour $USER
  ```

- expansion numérique :

  ```
  echo périmetre = $((2*(longueur+hauteur)))
  ```

- expansion du résultat d'une commande :

  ```
  echo il y a $(who | wc -l) connexions
  ```


Expansion

(suite)

Autre notation

Historiquement, sh utilisait des "anti-quotes" pour `$(...)` :

```
echo il y a `who | wc -l` connexions
```

Moins lisible, risque de confusion avec les apostrophes

```
echo il y a 'who | wc -l' connexions
```


Exercices

Exercice simple

Dans le script qui calcule l'âge, remplacez la constante 2009 par un appel à `date +%Y`

Plus compliqué

- Écrire un script qui affiche le nombre de processus qui vous appartiennent.
  Exécution :

  ```
  Sur tuba, adupont a 45 processus
  ```

  Indication : comptez les lignes qui commencent par votre nom dans le résultat de "`ps axu`".
- Améliorez en prenant le nom en clair "`Albert Dupont (etd1)`" fourni par `finger adupont` ou `getent passwd adupont`.

Exercices (suite)

Ecrire un script qui indique les 5 plus gros sous-répertoires d'un répertoire donné.

Exécution

```
1 $ plus-gros.sh  ~/Essais
2 196     /home/billaud/Essais/LATEX
3 152     /home/billaud/Essais/C++
4 96      /home/billaud/Essais/Python
5 36      /home/billaud/Essais/PHP
6 32      /home/billaud/Essais/PyUNO
```

Indications

- script à 1 paramètre
- `du -s repertoire/*`
- tri *numérique*
- commande `tail -n nombre`

Application : carnet de téléphone

Sous forme de trois commandes

- `tel-ajouter numero nom`
- `tel-chercher nom`
- `tel-afficher`

qui agissent sur un fichier de données `telephones.dat`
Format : un numéro et un nom par ligne

exemple

```
01234578 PUF
98765444 Charlie
```


application (suite)

```
#
# tel-afficher
#
nomFichier="telephones.dat"
cat $nomFichier
```

```
#
# tel-ajouter numero nom
#
nomFichier="telephone.dat"
echo $* >> $nomFichier
```

```
#
# tel-chercher nom
#
nomFichier="telephone.dat"
grep $1 $nomFichier
```


Exercice, suite

Améliorez la présentation avec la commande `dialog`

- `dialog --infobox message hauteur largeur`
- `dialog --textbox nomfichier hauteur largeur`
- `dialog --inputbox message hauteur largeur`

Attention : Avec une "inputbox", le résultat va sur la sortie d'erreur.

Commande et variable

- Ne pas confondre

v=date	affectation de la chaine "date"
date > f	redirection de la sortie vers un fichier
v=$(date)	affectation de la sortie dans une variable

- Exercice : que fait ceci

  `$cmd > $f`

  ?

(suite)

Exercice : que fait ceci

`$cmd > $f`

?

Exemple

```
#
format="%Y-%M-%d"
cmd="date +$format"
f=/tmp/resultat

$cmd > $f
```


Chaînes et expansion

L'expansion

- se fait dans les chaînes délimitées par `"..."`
- pas dans les chaines délimitées par `'...'`

Exemple

```
echo 'la variable $USER ' "contient $USER"
```


Quatrième partie IV

Fonctions

Fonctions

Un script peut comporter des fonctions, avec des paramètres positionnels

Syntaxe

```
function nom-de-fonction
{
   commande
   commande
   ...
}
```


Fonctions : exemple

Exemple

```
#
function archiver
{
  tar -czf /var/svgd/$1.tgz $2
}

archiver photos /home/billaud/photos
archiver musique /home/billaud/musique
```


Fonctions : avantages

Avantages :

- découpage logique,
- code plus facile à lire
- fonctions réutilisables

Fonctions : exemple

Par défaut, les variables sont communes (globales)

Exemple

```
#
destination=/var/svgd

function archiver
{
  tar -czf $destination/$1.tgz $2
}

archiver photos /home/billaud/photos
archiver musique /home/billaud/musique
```


Variables locales

On peut déclarer des variables locales dans une fonction

Exemple

```
#
destination=/var/svgd

function archiver
{
  local nom=$(basename $1)
  tar czf $destination/$nom.tgz
}

archiver /home/billaud/photos
archiver /home/billaud/musique
```

Cinquième partie V

Arithmétique

Let : affectation arithmétique

Syntaxe

`let VARIABLE=EXPRESSION`

Exemple à essayer

```
#! /bin/bash

let somme=$1+$2
echo $somme
```

Comparer

- `let somme=$1+$2`
- `somme=$1+$2`

Note

Dans une affectation arithmétique, l'expansion des variables est automatique

```
let c=a+b
let c=$a+$b
```


Exercice 1

Ecrire un script qui

- demande l'année de naissance
- affiche l'age

scenario

```
$ exercice1.sh
Vous êtes né en quelle année ?
1990
Vous avez donc 20 ans
$
```


Exercice 2

Convertir une heure (donnée sous la forme HHMM) en nombre de minutes

scenario

```
$ exercice2.sh 1015
615
$
```


Exercice

Ecrire un script qui calcule la durée d'un trajet, à partir des heures de départ et d'arrivée sous la forme HHMM

Scénario

```
$ ./duree.sh 630 2215
1545
```


Expansion arithmétique

A la place de

```
let surface=hauteur*largeur
echo la surface du rectangle est $surface m2
```

On peut écrire

```
let surface=hauteur*largeur
echo la surface du rectangle est $((largeur*hauteur)) m
```


Sixième partie VI

Structure de contrôle : case

Structure de contrôle "case"

Choisit les commandes à exécuter en fonction d'un sélecteur

- semblable au'`switch`' de C++
- le sélecteur est une chaîne de caractères

Exemple sérieux

```
#! /bin/bash
# Usage : archiver nom-de-répertoire

echo "Format = normal gz ?''
read format
case "$format" in
gz)
        option=z ;  suffixe=tgz  ;;
normal |  "" )
        option= ;   suffixe=tar  ;;
*)
      echo "format '$format' non reconnu" >&2
      exit 1
      ;;
esac
prefixe=$(basename $1)
tar -c${option}f $prefixe.$suffixe $1
```


case

Exemple

```
#
# usage :
# tel ajouter num nom
# tel chercher nom
# tel voir
#
nomFichier="telephone.dat"

case "$1" in
ajouter)
   shift
   echo $* >> $nomFichier
   ;;
chercher)
   grep "$2" $nomFichier
   ;;
voir)
   cat $nomFichier
   ;;
*)
   echo "Erreur"
   exit 1
   ;;
esac
```


Motifs d'un case

Plusieurs motifs pour un même cas

```
case $response of
oui | o )
    echo "d'accord"
    ;;
non | n )
    echo "tant pis"
    ;;
esac
```


Motifs d'un case jokers

Utilisation des "jokers" de bash

```
case $response of
[oO][Uu][iI] | [oO] )
    echo "d'accord"
    ;;
[nN][oO] )
    echo "tant pis"
    ;;
*)
    echo "quoi ?"
    ;;
esac
```


Septième partie VII

Processus

Définitions

Un processus = un programme "qui tourne"
Un programe lancé depuis le shell peut

- tourner en "avant plan" (foreground) : il faut attendre sa fin pour lancer une autre commande
- tourner en "arrière-plan" (background)
- être stoppé

Pour

lancer une commande en avant-plan	`xclock`
stopper la commande en avant-plan	CTRL-Z
relancer la commande stoppée en avant-plan	`fg`
relancer la commande stoppée en arrière plan	`bg`
lancer une commande en arrière-plan	`xclock &`

Note : Si il y a plusieurs commandes en arrière-plan, commandes

- `jobs`,
- `fg %n`,
- etc.

La table des processus

On peut la voir par la commande ps, ou top, ...

Exemple

```
$ ps
  PID TTY          TIME CMD
 4056 pts/1    00:00:00 bash
 4236 pts/1    00:00:07 xpdf.bin
 4243 pts/1    00:00:10 emacs
 4471 pts/1    00:00:00 xterm
 4613 pts/1    00:00:00 ps
```

- ps sans option montre les processus issus du shell
- options intéressantes : `axule...`
- voir aussi `pstree`

La commande "kill"

- Syntaxe : `kill` [-signal] num-processus ...
- Rôle : envoie un signal à des processus

Exemple

```
$ xclock  &
[6] 4734

$ ps
  PID TTY          TIME CMD
 4056 pts/1    00:00:00 bash
 4236 pts/1    00:00:07 xpdf.bin
 4734 pts/1    00:00:00 xclock
 4739 pts/1    00:00:00 ps

$ kill -TERM 4734
```


La commande "kill" (suite)

- Par défaut, utilise le signal TERM (9) qui termine le programme.
- le signal STOP arrête un processus
- le signal CONT le relance
- `kill -l` affiche la liste des signaux

Pilotage des processus

- commande `&` lance une commande en arrière-plan
- la variable `$!` contient son numéro de processus
- la variable `$$` = numero du shell courant

Exemple

```
mplayer funny-music.mp3 >/dev/null &
music=$!

# sauvegardes
tar czf .....

# arrêter la musique à la fin
kill -9 $music
```


Wait

`wait nnn` attend un processus

Exemple

```
mplayer funny-music.mp3 >/dev/null &
music=$!

# sauvegardes en parallèle
tar czf archive1.tar ..... &
svgd1=$!
tar czf archive2.tar ..... &
svgd2=$!

wait $svgd1
wait $svgd2

kill -9 $music  # arrête la musique
```


Un exemple de service

Une commande pour faire apparaître / disparaître une pendule sur le bureau

Usage

- `./pendule.sh start`
- `./pendule.sh stop`
- `./pendule.sh usage`
- `./pendule.sh restart`

Le code principal

Code 2/2

```
...
case "$1" in
start)
    do_start ;;
stop)
    do_stop ;;
restart)
    do_stop
    do_start ;;
usage)
    print_usage ;;
*)
    print_usage
    exit 1
esac
```


Constantes et Fonctions

Code 1/2

```
prog=/usr/bin/xclock
pid_file=/tmp/$USER.pid

function do_start {
  $prog &
  echo $! > $pid_file
}

function do_stop {
  kill -9 $(cat $pid_file))
}

function usage {
  echo "usage: pendule {start|stop|restart|usage}"
}
```

Huitième partie VIII

Boucle for

Boucle for

Forme générale

```
for VAR in LISTE
do
   COMMANDE
   COMMANDE
   ....
done
```

Boucle avec une variable qui parcourt une liste de mots.

Boucle for, exemples simples

```
for f in *.cc
do
   astyle --syle=gnu $f
done

for f in *.cc
do
   echo "le fichier $f contient $(wc -l $f) lignes"
done
```


Boucle for, exercice

Ecrire une commande qui calcule la somme de ses paramètres (en nombre illimité)

```
$ somme 100 3 20
123
```


Exercice (seq)

Analyser le script

```
r=1
for i in $(seq 1 $1)
do
   let r*=i
done
```


Exercice (find)

Analyser le script

```
for f in $(find ~ -name '*.cc' -ctime -7)
do
  ls -l "$f"
done
```

- rôle de `find ~`
- rôle de `find ~ -name '*.cc'`
- rôle de `find ~ -name '*.cc' -ctime -7r`
- pourquoi les guillemets dans `ls` ?

Neuvième partie IX

Décisions

Code de retour (exit status)

- Le code de retour d'un programme indique si il s'est bien terminé.
- Le code de retour de la dernière commande est dans la variable $?
- Par convention : 0 = OK.

code de retour

Exemple

```
$ ls -ld /tmp
drwxrwxrwt 10 root root 12288 déc 22 17:32 /tmp
$ echo code de retour = $?
code de retour = 0

$ ls -l qsdsqd
ls: ne peut accéder qsdsqd: Aucun fichier ou
répertoire de ce type
$ echo code de retour = $?
code de retour = 2
```


Codes de retour et documentation

Les codes de retour des commandes sont décrits dans le manuel

Exemple : man ls

```
...
   Exit status is 0 if OK, 1 if minor problems,
   2 if serious trouble.
```


exit

- Par défaut, un script returne le code de sa dernière commande
- Il peut retourner un code spécifique par
  `exit [code]`

Exemple

```
1 #!/bin/bash
2 echo "Something strange happened"
3 exit 42
```


La structure de contrôle `if then else fi` utilise le code de retour d'une commande

Exemple

```
1  #
2  # usage:
3  #   compiler.sh prefixe
4  #
5  if g++ -o $1 $1.cc
6  then
7    echo "la compilation s'est bien passée"
8  else
9    echo "il y a eu un problème"
10 fi
```


Test

Le code de retour du programme `test` dépend d'une condition.
Exemple :

`test -f nomFichier`

indique si le fichier existe.

Application

fichier "compiler.sh"

```
1  #!/bin/bash
2
3  if test -f $1.cc
4  then
5    g++ -Wall -o $1 $1.cc
6    echo Compilation terminée
7  else
8    echo "Erreur: pas de fichier $1.cc"
9    exit 1
10 fi
```

Autre notation : [condition]
Autres tests :

```
[ -d nom ]                  # nom désigne un répertoire
[ chaine1 = chaine2 ]       # comparaison de chaines
[ chaine1 != chaine2 ]
[ chaine1 \< chaine2 ]
[ chaine1 \> chaine2 ]
[ nombre1 -eq nombre2 ]     # comparaison de nombres
[ nombre1 -ne nombre2 ]
[ nombre1 -le nombre2 ]     # less or equal
[ nombre2 -ge nombre2 ]     # greater or equal
                            # voir aussi -lt et -gt
```

Unix : shell scripts
└La commande test

Exercice

Ecrire une commande qui affiche le maximum de deux paramètres.

Scénario

```
$ max.sh 37 421
421
```


Enchaînement conditionnel

Syntaxe

```
COMMANDE1 && COMMANDE2
COMMANDE1 || COMMANDE2
```

Exécute la seconde commande seulement si la première a réussi (&&) ou échoué (||)

Exemple

```
g++ prog.cc && echo OK
```

Unix : shell scripts
└Enchaînement conditionnel

Application

```
1 #
2 max=$1
3 [ $2 -ge $max ] && max=$2
4 echo $max
```


if-then-elif-else, forme général

Syntaxe

```
  if COMMANDES
  then
    COMMANDES
[ elif COMMANDES
  then
    COMMANDES
]...
[ else
   COMMANDES
]
fi
```

Unix : shell scripts

Dixième partie X

Boucle while

Boucle while

```
while COMMANDE
do
   COMMANDES
done
```

```
#
# fac nombre
# calcule la factorielle d'un nombre
#
# exemple:    fac 5
i=1
f=1
while [ $i -le $1 ]
do
    let f=f*i
    let i++
done
echo $f
```


Boucle while-read

Tournure habituelle pour exploiter le contenu d'un fichier, ligne par ligne.

```
while read numero nom
do
   printf "| %12s | %-30s |\n" $numero $nom
done < $nomFichier
```


L'instruction break

permet de sortir d'une boucle

```
while true
do
   ....

   echo "voulez-vous arrêter ?"
   read reponse
   [ $reponse = oui ] && break

   ...
done
```

Onzième partie XI

Structures de Contrôle

Il y en a d'autres

- select
- until ...