Séance No 6

- Cours :
  - substitution de variables
  - entrées/sorties
  - signaux
  - analyse de paramètres de la ligne de commande
  - commandes de base: find, xargs, tee

option du shell: -u

- set -u: l'utilisation de la valeur d'une variable non définie est considérée comme une erreur.
- L'utilisation de ce paramètre supprime de nombreuses erreurs dues à des fautes de frappe
- Exemple:

```
$ echo $aaa
-bash: a: unbound variable
$ aaa=12
$ echo $aaa
12
```

de la bonne utilisation de la commande test et des valeurs des variables

- "$var" vs $var
- les bonnes pratiques concernant l'utilisation de la commande test :
  - [-n "$a"] && ["$a" = "$b"] au lieu de [-n "$a" -a "$a" = "$b"]
  - toujours mettre les variables entre apostrophes ": "$var" et pas $var
  - utiliser les operateurs arithmetiques pour les nombres et literaux pour les chaines
  - utiliser ["x$A" = "aexpression"] pour comparer la valeur de variables (pour éviter des problèmes si $A est vide)
  - Les operateurs -e, -nt, -ot, -N, -L, -h, >, <, ne sont pas portables. -e, -h et -L sont POSIX toutefois. -h et -L sont raisonablement portables

substitution de variables

- attribuer une valeur par défaut aux variables:

	valeur si la variable		
expression	est non nulle	est nulle	effet de bord
${variable:-valeur}	$variable	valeur	
${variable:=valeur}	$variable	valeur	la variable prend la valeur valeur
${variable:+valeur}	valeur	$variable	

- afficher un message d'erreur si la variable est vide:
  - ${variable:?message}: si la variable est vide, le shell affiche le message et provoque la terminaison du script shell. Si la variable n'est pas vide, l'expression vaut $variable

substitution de variables (2)

- longueur de la valeur d'une variable: ${#variable}
- suppression de fragments :

à gauche			
	#	plus petit	${variable#modele}
	##	plus grand	${variable##modele}
à droite			
	%	plus petit	${variable%modele}
	%%	plus grand	${variable%%modele}

- le modèle doit être conforme à ceux utilisés pour la substitution de noms de fichiers du shell utilisé
- avec bash, il faut positionner l'option extglob par « shopt -s extglob » pour pouvoir utiliser les expressions complexes dans les modèles.

substitution de variables : exemples

```
f=/usr/local/Photos/reveDeVacancesderevealamon
  tagne.jpg
$ echo ${f#/}
usr/local/Photos/reveDeVacancesderevealamontag
  ne.jpg
$# suppression du premier / du nom
$ echo ${f#*/}
usr/local/Photos/reveDeVacancesderevealamontag
  ne.jpg
$ # suppression du chemin
$ g=${f##*/}
$ echo $g
reveDeVacancesderevealamontagne.jpg
```

substitution de variables : exemples (2)

```
$# la variable f n'a pas été modifiée
$ echo $f
/usr/local/Photos/reveDeVacancesderevealamonta
  gne.jpg
$# pour supprimer l'extension d'un fichier
$ echo ${g%.*}
reveDeVacancesderevealamontagne
$#obtenir le chemin
$ echo ${f%/*}
/usr/local/Photos
$ echo ${g%%ve*}
re
$ echo ${g%ve*}
reveDeVacancesdere
```

break et continue

- break: permet de forcer la sortie d'une boucle for, while ou until. Il peut prendre un argument strictement supérieur à 1 afin de spécifier le nombre de boucles imbriquées dont il faut sortir. Si la valeur spécifiée est supérieure au nombre de boucles imbriquées, on sort de toutes les boucles imbriquées.
- continue: Il permet de passer `a la prochaine it´eration d'une boucle for, while ou until. Le mot clef continue peut aussi prendre un argument strictement supérieur à 1 afin de spécifier le nombre d'itérations à "sauter". Si la valeur spécifiée est supérieure au nombre d'itérations possibles, la dernière itération est toutefois exécutée.

initialisation des paramètres positionnels avec set

- set: suivie d'arguments permet d'affecter ces arguments aux paramètres positionnels
- les caractères présents dans la variable IFS séparent les arguments
- Exemples:

```
$ set a b c
$ echo $1
a
$set $(date)
$echo $2
dèc
```

tableaux

- non traités dans ce cours

gestion avancée des entrées/sorties

- rappel:
  - &1, &2 désignent respectivement la valeur courante de l'entrée standard et de la sortie en erreur
  - exemple d'utilisation: affichage sur la sortie en erreur:
    - echo mon message >2&
- redirection de l'entrée standard
  - « exec 0<fichier » :l'entrée standard de toutes les commandes suivantes du script liront leurs données depuis fichier

redirections avancées: suite

- redirection de la sortie standard
  - « exec 1>fichier » :la sortie standard de toutes les commandes suivantes du script ira dans le fichier fichier
- redirection de la sortie en erreur
  - « exec 2>fichier » :la sortie en erreur de toutes les commandes suivantes du script ira dans le fichier fichier
- redirection des deux sorties d'un seul coup:
  - « exec 1>fichier 2>&1 »

gestion de fichiers

- manipulation de fichiers associés aux descripteurs 3 à 9
- ouverture de fichier
  - en lecture : exec descripteur<fichier
  - en écriture: exec descripteur>fichier
- fermeture d'un fichier:
  - fichier ouvert en lecture : exec descripteur<&-
  - fichier ouvert en écriture : exec descripteur>&-
- lecture : commande <&descripteur
- écriture: commande >&descripteur

traitement d'un fichier

- commande read
- le traitement d'un fichier associe while et read dans une boucle de la forme
  - while read var1 var2 var3 <descripteur
  - do
  - commandes
  - done

traitement de fichers structurés

- rappel: IFS :
  - variable contenant les caractères séparant les mots entre eux
  - par défaut: espace, tabulation et saut de ligne
- si les lignes du fichier f contiennent des champs séparés par un caractère c, il suffit d'affecter c comme valeur à IFS pour que read en tienne compte

traitement de fichers structurés: exemple

```
$ cat traitementFichierStructure.sh
#! /bin/bash

IFS=":"
exec 3</etc/passwd
while read a b c d e f g h i j k l m <&3
do
  echo "$a a l'uid $c"
done
exec 3<&-
 $ ./traitementFichierStructure.sh |head
root a l'uid 0
daemon a l'uid 1
bin a l'uid 2
sys a l'uid 3
...
```

gestion de fichiers: exemples

```
$ cat /tmp/nombres.txt
1 3
6 4
2 -5
$ exec 3</tmp/nombres.txt
$ while read a b<&3
> do
>   echo $((a+b))
> done
4
10
-3
$ exec 3<&-
```

commande eval

- permet de réaliser une double évaluation
  - première passe: les caractères spéciaux sont traités
  - seconde passe: la ligne résultante est évaluée
- exemple:

```
a=123
$ b=a
$ eval echo \$$b
$ a=ls
$ eval $a
[liste des fichiers suite à
 l'exécution de ls]
```

Signaux: principaux signaux

nom du signal	No	cas usuel	comportement par défaut d'un processus le recevant	comportement modifiable ?
HUP	1	perte de connexion	Mort	Oui
INT	2	Ctrl-C sur le clavier	Mort	Oui
KILL	9	processus qui ne répond pas à kill -TERM	Mort	Non
TERM	15	sus en lui laissant la chance de faire le ménage	Mort	Oui
STOP		suspend un processus (C-trl-Z au clavier)	suspend	Non

gestion des signaux: TRAP

- TRAP: permet d'ignorer un signal ou de modifier le comportement associé à un signal
- utile pour garantir qu'un script fera le ménage avant de mourir élégamment
- Syntaxe:
  - trap 'liste de commandes' signal1 [signal2 [signal3 ...]]
- Exemples:
  - ignorer un signal: trap '' signal
  - gérer un signal: trap 'liste de commandes » signal
  - remettre le traitement par défaut: trap - signal

TRAP: exemple

```
#! /bin/bash
fichierTemp=/tmp/$0.$$
#en cas de signal, on fait le ménage:
trap 'rm -f "$fichierTemp";exit 1'\
  SIGHUP SIGINT SIGTERM
#reste du script qui peut
#potentiellement modifier fichierTemp
commande1
commande2
```

analyse des options de ligne de commande

- 2 méthodes:
  - utiliser la commande interne getopts qui permet de gérer des options d'une lettre
  - à la main : boucle while/case/shift (avantage: on peut utiliser des noms longs pour les options)

getopt

- une option est composée d'une lettre précédée d'un + ou d'un -
- une option peut être suivie d'un argument
- exemple: getopt "abc:d" var
  - les options possibles sont -a, -b, -c, -d
  - le « : » après le « c » indique que l'option -c attend un paramètre
  - var est le nom de la variable dans laquelle sera mis le nom de l'option lue
  - l'argument éventuel sera mis dans la variable réservée OPTARG
  - getopt s'utilise dans une boucle while

getopt

- Exemple:

```
while getopt "ac:" option
do
  case "$option" in
    a) echo option a
       ;;
    c) echo option a avec l'argument
 "$OPTARG"
       ;;
    \?) echo option invalide
       ;;
  esac
done
```

getopt: gestion des erreurs

- lorsque getopt rencontre une option invalide:
  - la variable var contient le caractère ?
  - un message est affiché à l'écran
- si : est en première position dans la liste des options:
  - aucun message ne sera affiché à l'écran
  - OPTARG contiendra la valeur de l'option fautive
  - exemple: getopt `":abc:d" var`
- paramètre manquant:
  - var contient : et OPTARG contient le caractère de

gestion artisanale des arguments

```
while [ $# -gt 0 ]; do
  case "$1" in
    -Prefix) shift
             [ $# -gt 0 ] || { affErr; exit 1; }
             Prefix="$1"
             shift
             ;;
    -remoteComputer) shift
            [ $# -gt 0 ] || { affErr; exit 1; }
            remoteComputer="$1"
            shift
            ;;
    *) repASauver="$1"
       shift
       ;;
  esac
done
```

gestion d'un processus en arrière plan: wait

- WAIT p: attend la terminaison du processus dont le pid est p
- WAIT: attend la terminaison de tous les processus lancés en arrière plan à partir du shell courant
- Exemple: pour profiter d'une machine bipro:

```
gzip -9 fichier1&
gzip -9 fichier2&
wait
mv fichier1 fichier2 /archive/
```

commande find

- find permet de chercher récursivement les fichiers vérifiant une ou plusieurs conditions
- outre les expression simples, l'expression que doit vérifier un fichier peut être de la forme (par priorité décroissante) :
  - (expression)
  - ! expression
  - expression1 -a expression2 : ET logique
  - expression1 expression2: ET logique
  - expression1 -o expression2: OU logique

commande find

- expressions élémentaires à argument numérique:
  - +n: toutes les valeurs supérieures ou égales à n
  - -n: toutes les valeurs inférieures ou égales à n
  - n: n exactement
  - par la suite, partout où on verra un argument numérique n, on pourra utiliser +n, n ou -n
  - exemples:
    - -size 1024k: les fichiers de taille égale à 1024 Ko
    - -size -1024k: les fichiers de taille inférieure égale à 1024 Ko
    - -size +1024k: les fichiers de taille supérieur ou égale à 1024 Ko

commande find: quelques expressions élémentaires

- quelques expressions élémentaires:
  - -name motifProtégé: les fichiers vérifiant le motif
  - -size n: les fichiers de taille n
  - -mtime n, -ctime n, -atime n
  - -perm p avec p ayant la forme numérique ou symbolique des arguments de chmod
  - -type c avec c=b,c,d (dossier),l (lien symbolique),p,f (fichier ordinaire),s
  - -user u
  - -group g
  - -link n : nombre de liens physique sur le fichier
  - -print: provoque l'affichage des noms des fichiers vérifiant l'expression (par défaut sur le Gnu find)
  - ...

commande find: -exec

- -exec commande;
  - pour chaque fichier trouvé, la commande est exécutée.
  - si {}apparaît parmi les arguments de la commande, il est remplacé par le nom du fichier trouvé
- -exec commande arguments {} +
  - syntaxe POSIX/SUSv3 (standard mais pas disponible sur toutes les plateformes)
  - les noms des fichiers trouvés sont accumulés dans une liste l
  - la commande est exécutée une seule fois, à la fin de la recherche et {}est remplacé par la liste des arguments

commande find : exemples

- fichiers ordinaire nommés core de plus de 1024Ko
  - find -size +1024k -type f -name core -print
- fichier ordinaires de l'utilisateur petit ou fichiers ordinaires de taille supérieure à 1024 Ko et de nom core
  - find -type f \(-user petit -o \(-size +1024k -name core \) \) -print
- fichier dont le nom commence par C
  - find -name c* -print

commande find: -exec et les espaces (&Co)

- rm est une commande qui ne lit pas sur son entrée standard. 3 méthodes pour effacer un ensemble de fichiers sélectionnés par find :
  - on lance un rm par fichier (lourd)
    - find -name *.bak -exec rm -f {} \;
  - syntaxe POSIX: un seul rm global est lancé :
    - find -name *.bak -exec rm -f {} \+
  - une solution avec les options spécifiques de Gnu find:
    - find -name *.bak -print0 |xargs -0 rm -f
  - ne marche pas avec les noms de fichiers contenant des caractères à problème (espace, saut de ligne, ', etc.):
    - find -name *.bak -print |xargs rm -f

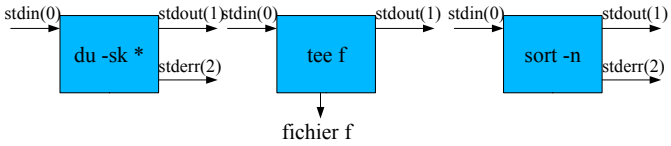
commande xargs

- xargs [options] commande
- xargs rassemble ce qu'elle reçoit sur son entrée standard dans une liste l et exécute « commande l »
- xargs s'utilise avec des commandes qui n'acceptent pas de données sur leur entrée standard
- Exemple:
  - grep -l perso * | xargs chmod 700

xargs : options utiles

- -p: prompt mode. une confirmation est demandée à l'utilisateur pour chaque invocation de la commande
- si le nombre ou la taille des arguments transmis via l'entrée standard est important, il est possible d'indiquer à xargs d'exécuter plusieurs fois la commande avec une liste limitée:
  - -n nombre: la commande est invoquée plusieurs fois et chaque invocation a au plus nombre arguments
  - -s taille: la commande est invoquée plusieurs fois et chaque invocation fait au plus s octets

commande tee



- la commande tee envoie simultanément son entrée standard vers un fichier et vers sa sortie standard.
- options:
  - -a: ajoute au fichier
  - -i : ignore le signal