Programmation Shell

Dawan / Pierre Sablé / 2021
Version 3.0

Présentations

Qui suis-je ?

- Mon parcours
- Mes compétences

Exprimer mon besoin

- Le contexte **IT**
- L’environnement de travail
- Les objectifs pour cette formation

Plan de Formation
-1/10-

Présentation

- Qu’est-ce que le Shell ?
- Historique des Shell Unix
- Les différents Shell

Le rappel des commandes

- L’historique
- Édition en mode Emacs
- Édition en mode vi
- La commande fc

Plan de Formation
-2/10-

Configuration de son environnement bash

- Les fichiers de configuration
- Les alias

Utilisation courant du shell

- Les jockers
- Les caractères d’échappement
- Les redirection, les tubes

Atelier : Prise en main vi, tuning environnement utilisateur, commandes avancées

Plan de Formation
-3/10-

Les scripts Shell

- Principes
- Les commentaires
- Exécution d’un script
- Affichage

Atelier : Premier script shell

Plan de Formation
-4/10-

Les variables

- Déclaration d’un variable
- Les variables d’environnement
- L’instruction read
- Affichage

Atelier : Jouons avec les variables, saisie clavier

Plan de Formation
-5/10-

Les instruction de contrôle

- if/else
- for
- case
- Select
- while

Atelier : Scripts avancés avec contrôles

Plan de Formation
-6/10-

Les alias et les fonctions

- Alias
- Fonctions
- Appel d’un script

Les expression régulières

- La commande grep
- Utilisation des expressions régulières avec grep
- La commande egrep

Atelier : structurons nos scripts, exercices

Plan de Formation
-7/10-

Les chaînes de caractères

- Manipulations
- Basename et dirname

La gestion de fichiers

- Création d’un fichier
- Utilisation des redirections
- Séparateur de champs

Atelier : manipulations

Plan de Formation
-8/10-

Gestion de processus

- Lancement/arrêt/reprise/fin de processus
- Les signaux
- Les commandes de contrôle
- Les variables associées

Atelier : Administration des processus

Plan de Formation
-9/10-

Le filtre SED

- Principes
- Commandes de base
- Expression régulières

Atelier : Manipulations, cas courants

Plan de Formation
-10/10-

Le processeur de texte AWK

- Principes
- Structure d’un programme
- Les variables
- Les tableaux
- Les instructions
- Les fonctions

Atelier : Manipulations, cas courants

Objectifs

Théoriques

- Comprendre l’utilité du shell en environnement Linux
- Connaître les bases pour une utilisation courante
- Acquérir les principales fonctionnalités du langage permettant d’écrire des scripts

Pratiques

- Utiliser un système Linux en mode commande
- Réaliser des scripts avec variables, boucles, contrôles
- Utiliser des fonctionnalités avancées et complémentaire :
  - Lecture / écriture / remplacement / fonctions / gestions de paramètres

Présentation

Qu’est ce que le Shell
-1/2-

Shell :

- Interface minimale de communication entre l’utilisateur et le système UNIX
- Exécute les commandes écrites par l’utilisateur
- Affiche du texte correspondant au résultat de l’exécution des commandes
- Propose des fonctions internes par mots clés, caractères spéciaux pour faire de la programmation

Deux modes d’utilisation :

- Interactif : L’utilisateur saisit et exécute ses lignes de commandes une par une dans un terminal
- Non interactif (bash) : le shell lit un ensemble de commandes à partir d’un fichier appelé *shell script*

Historique Shell
-2/2-

Bourne shell (sh) :	1977 Steven Bourne
C shell (csh) :	1978 Bill Joy
Korn shell (ksh) :	1983 David Korn
Bourne-Again shell (bash) :	1988 Brian Fox
Z shell :	1990 Paul Falstad

https://en.wikipedia.org/wiki/Comparison_of_command_shells

Le rappel des commandes

L’invite de commande
-1/7-

Connecté à un terminal, utilisation de l’invite de commande :

```
console

User@NomDeLaMachine~$

~  Dossier courant
$  normal
#  root
```

Rappels :
utilisation de l'invite de commandes
- 2/7 -

Dawan
Training is our way

Exécution d'une commande :

```
console

User@NomDeLaMachine~$ commande paramètres
```

Exemples :

```
console

User@NomDeLaMachine~$ commande -d
User@NomDeLaMachine~$ commande -d -a -U -h
User@NomDeLaMachine~$ commande -daUh
```

Shell Builtin ?
-3/7-

Utilisation de la touche tabulation pour autocomplétion

Commandes internes : Commande dont le code est implanté au sein de l'interpréteur de commande

```
User@NomDeLaMachine~$ type  echo
echo est une primitive du shell
```

Commandes externes : Fichier au format binaire exécutable présent dans l'arborescence

```
User@NomDeLaMachine~$ type  uptime
uptime est /usr/bin/uptime
```

L'aide :

Commandes internes & commandes externes

help : Utilisable pour les commande interne au shell

```
User@NomDeLaMachine~$ help echo | less
echo: echo [-neE] [arg ...]est une primitive du shell
...
```

man : Utilisable pour les commandes externes

```
User@NomDeLaMachine~$ man uptime
UPTIME(1)              User Commands                 UPTIME(1)
NAME
       uptime - Tell how long the system has been running.
...
```

Commandes de base
-5/7-

Se déplacer dans un dossier :	$ cd dossier
Afficher le répertoire courant :	$ pwd
Afficher une ligne	$ echo "texte"
Déplacer ou renommer un fichier ou dossier :	$ mv fichier_source fichier_destination
Créer un répertoire :	$ mkdir rep1
Copier un fichier :	$ cp fichier1 fichier2
Copier un répertoire (et les permissions) :	$ cp -a rep1 rep2
Supprimer un fichier :	$ rm fichier
Supprimer un répertoire :	$ rm -rf repertoire
Lister le contenu d'un répertoire :	$ ls repertoire
Lister le contenu d'un répertoire en détail :	$ ls -alh repertoire
Lister tous les noms de fichier ayant un extension composée d'un caractère	$ ls *. ?
Afficher le contenu d'un fichier :	$ cat fichier
Afficher les 10 dernières lignes d'un fichier :	$ tail fichier
Afficher les 10 premières lignes d'un fichier :	$ head fichier
Stocker la sortie d'une commande dans un fichier (en l'écrasant) :	$ commande >fichier
Ajouter la sortie d'une commande à la fin d'un fichier :	$ commande >>fichier
Stocker la sortie standard et d'erreur d'une commande dans un fichier :	$ commande >fichier 2>&1
Stocker la sortie d'une commande dans un fichier et l'afficher :	$ commande | tee fichier
Afficher le résultat d'une commande par pages :	$ commande | less
Recherche une chaîne de caractères dans un fichier	$ grep "chaine" fichier
Rechercher une chaine de caractères dans le résultat d'une commande	$ cat /etc/passwd | grep -i bob
Compter le nombre lignes renvoyées par une commande	$ who | wc -l

Historique des commandes
-6/7-

Fichier historique :

- **Bash** : ~/.bash_history

Commande interne history :

```
User@NomDeLaMachine~$ history
User@NomDeLaMachine~$ history -c
User@NomDeLaMachine~$ echo $HISTSIZE
```

Recherche dans l'history : CTRL + R

```
(reverse-i-search)`man': man uptime
```

fc : Affiche ou exécute des commandes issues de l' historique

```
User@NomDeLaMachine~$ fc -l
1625    cd dawan
1626    ls -l
```

Modes d’édition
-7/7-

Emacs :

- Plutôt dirigé vers un environnement de travail (dev)
- Extensible : propose dans une même interface web, navigation de fichiers et un shell
- Modules

Vi / Vim :

- Éditeur de texte léger orienté production (presque toujours installé par défaut)
- Excellent support des expressions rationnelles (recherches et remplacements)

Configuration de son environnement bash

Environnement bash

Shell courant :

```
User@NomDeLaMachine~$ env
User@NomDeLaMachine~$ set -o [option]
```

Fichiers d’environnements :

- /etc/profile
- .bash_profile
- .bashrc
- .profile

Alias et alias permanent

Utilisation courante du shell

Les jockers
-1/6-

Caractère joker	Signification
*	N'importe quel caractère, 0 ou n fois
?	N'importe quel caractère, 1 seule fois
[xyz]	Le caractère x ou y ou z
[x-y]	Un caractère entre x et y (exemple : [0-9], [a-z]..)
~	Répertoire HOME de l'utilisateur en cours

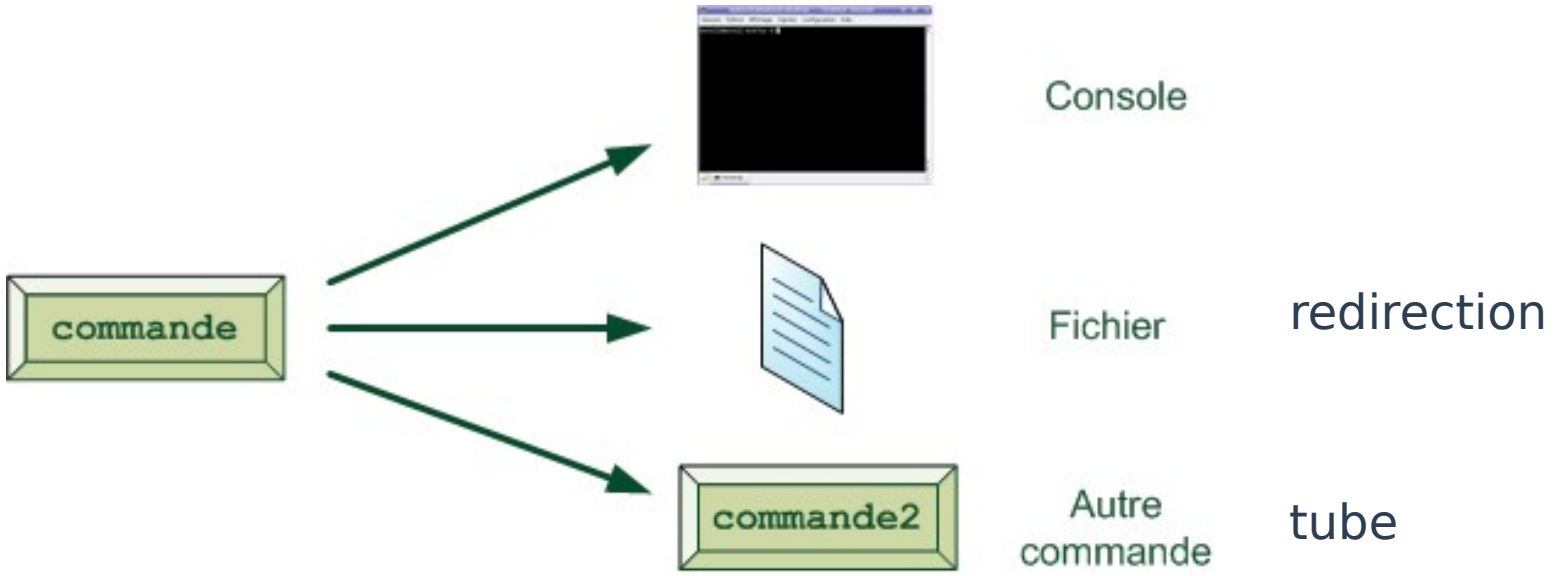
Caractères d'échappement
-2/6-

Exemple echo : utiliser echo -e

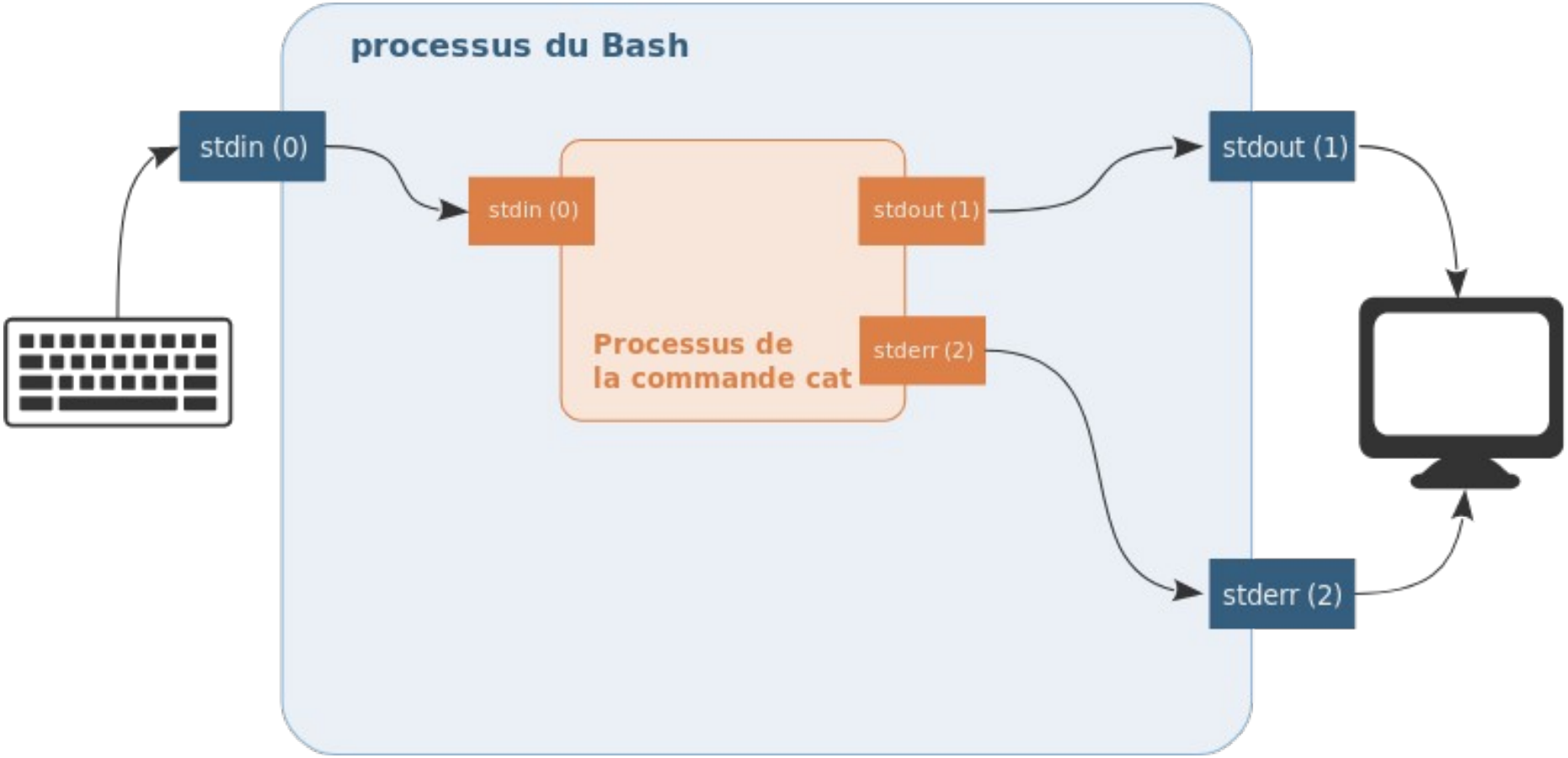
Caractère échappement	Signification
\n	Provoque un saut de ligne
\c	N'importe quel caractère, 1 seule fois. Elimine le saut de ligne naturel de la commande echo
\t	Affiche une tabulation
\	Echape, protège un caractère spécial

Les redirections, tubes
-3/6-

Redirections et tubes :



Les entrées / sorties
-4/6-



Les redirections
-5/6-

Redirections : sortie standard, sortie erreur

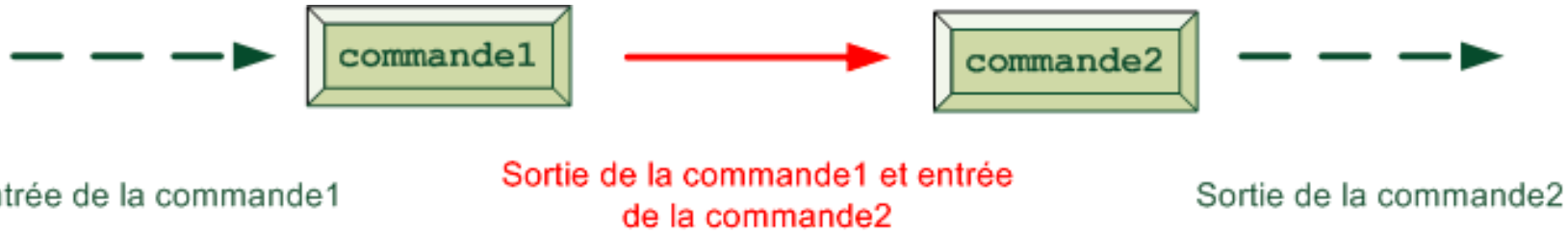
- Simple, standard :
- Double :
- Erreur :
- Standard et Erreur :
- Standard + Erreur :
- Eliminer l'affichage :

```
User@NomDeLaMachine~$ ls /etc/network > list.txt
User@NomDeLaMachine~$ ls /etc/network >> list.txt
User@NomDeLaMachine~$ ls -z 2>err.log
User@NomDeLaMachine~$ ls .bash* .rien* > std.log 2>err.log
User@NomDeLaMachine~$  ls /home /err 1>std2.log  2>&1
User@NomDeLaMachine~$ pstree > /dev/null
```

Redirection : lecture

```
User@NomDeLaMachine~$ cat < std2.log
```

Les tubes
-6/6-



Exemples :

```
User@NomDeLaMachine~$ hostname | cut -c1-3

User@NomDeLaMachine~$ df -h | tee arbo

User@NomDeLaMachine~$ du -sh * | sort -rn
```

Les scripts shell

Premier script shell
-1/2-

En-tête : permet au système d'exécuter un fichier en tant que script bash, perl, python

- shebang

```
#!/bin/bash
```

Commentaires : relecture et portabilité facilités

- #

Affichage :

```
echo
```

Exécution :

```
User@NomDeLaMachine~$ chmod +x script.sh

User@NomDeLaMachine~$ ./script.sh
```

Bonne pratique, debug
-2/2-

Bonnes pratiques :

- Indentation : meilleure lisibilité
- linter : réutilisation, partage, erreurs invisibles
  - shellcheck

Debug :

- Set -x

Les variables

Les variables
-1/4

Définir une variable :

```
User@NomDeLaMachine~$ var1=mot1
User@NomDeLaMachine~$ echo $var1
User@NomDeLaMachine~$ set | grep $var1
```

Retirer une variable :

```
User@NomDeLaMachine~$ unset var1
```

Variable avec espace :

```
User@NomDeLaMachine~$ var2='var avec espace'
```

Isoler le nom d'une variable : bonne pratique

```
User@NomDeLaMachine~$ result=${var1}_${var2}
```

Export d'un variable :

```
User@NomDeLaMachine~$ export MAVARIABLE
```

Les variables réservées
-2/4

Variables réservées au shell :

$0	Nom du script en cours d'exécution tel qu'il a été appelé
$1 $2 $3... $9	Argument passé en paramètre de la commande à la position donnée
$@	Tous les paramètres passés en argument
$*	Tous les paramètres passés en argument, sans conserver les espaces et les guillemets
$#	Nombre d'arguments passés en paramètre
$?	Code de retour du processus précédent
$!	PID du processus précédemment mis en arrière-plan

shift : La commande shift permet de décaler la liste d'arguments d'une ou plusieurs position. Utilisée dans le cas où le premier argument n'a pas a subir le même traitement que les suivants

Variable d'environnement :

```
echo $USERNAME
```

Substitution de commande :

```
list=$(ls)
echo $list
```

L'instruction read
-3/4

Lecture au clavier :

```
User@NomDeLaMachine~$ read var
```

Code retour :

- $? = 0 si une valeur a été saisie

```
User@NomDeLaMachine~$ read societe
Dawan
User@NomDeLaMachine~$ echo $?
0
```

Variable IFS : (Internal field separator) séparateur standard du shell

```
User@NomDeLaMachine~$ set | grep ^IFS=
IFS=$' \t\n'
```

Les tableaux
Manipulation des tableaux à indice (array)
-4/4

Déclarer un tableau à INDICE vide:

```
User@NomDeLaMachine~$ declare -a tab
```

Déclarer un tableau avec valeurs :

```
User@NomDeLaMachine~$ tab2=(apple orange lemon)
```

Attribuer une valeur à un tableau :

```
User@NomDeLaMachine~$ tab[n]="valeur n"
User@NomDeLaMachine~$ declare -a tableau='([0]="val1" [1]="val2")'
```

Afficher tous les éléments d'un tableau sous forme de liste :

```
User@NomDeLaMachine~$ echo ${tab2[@]}
```

Afficher tous les éléments d'un tableau sous forme de chaîne :

```
User@NomDeLaMachine~$ echo ${tab2[*]}
```

Les tableaux

Manipulation des tableaux associatifs (array) -4/4

Déclarer un tableau à ASSOCIATIF vide:

```
User@NomDeLaMachine~$ declare -A tab
```

Attribuer une clé - valeur à un tableau ASSOCIATIF :

```
User@NomDeLaMachine~$ tableau[cle1]=val1
```

Accéder à une valeur :

```
User@NomDeLaMachine~$ echo ${tableau[cle1]}
```

Compter les clés :

```
User@NomDeLaMachine~$ echo ${#tableau[@]}
```

Lister les clés :

```
User@NomDeLaMachine~$ echo ${ !tableau[@]}
```

Les instructions de contrôle

Tests
- 1/6 -

Commande de test :

- [ ] : commande originelle sh, ksh et bash
- [[ ]] : version enrichie, compatible ksh et bash uniquement

Test sur les nombres :

gt	Greater than : plus grand que (if [$nombre -gt $autre]; then)
eq	Equals : égal
ne	Not equal : différent de

Test sur les fichiers :

nt	est plus récent que
d	est un répertoire
f	est un fichier

Combinaisons :

- **&& :** condition ET
- || **:** condition OU
- **! :** inversion de valeur : l'expression doit être fausse

If/Else

Dawan
Training is our way

If / elif / else

- Le **if** teste uniquement le code retour de la commande (0 étant le code de succès)

```
if ( [[ -f $fichier ]] && [[ -r $fichier ]] )
then
  # Traitement si le fichier existe, et s'il est accessible en lecture
elif ( [[ -f $fichier ]] && [[ -x $fichier ]] )
then
  # Traitement si le fichier existe, et s'il est exécutable
else
  # Sortie car aucune condition remplie
fi
```

for
- 2/6 -

for

- La boucle for permet de parcourir une liste de valeurs, elle effectue donc un nombre de tour de boucle qui est connu à l'avance

```
for i in *
do
  echo $i
done
```

- Seconde syntaxe :

```
for ((i=0 ; <=6 ; i++))
do
  echo $i
done
```

```
for ((e1;e2 ; e3))
    do
      echo $i
    done
```

  - Dans laquelle, e1, e2 et e3 sont des expressions arithmétiques. Une telle boucle commence par exécuter l'expression e1, puis tant que l'expression e2 est différente de zéro le bloc d'instructions est exécuté et l'expression e3 évaluée.

case
- 4/6 -

Dawan
Training is our way

Case

- enchaînement de tests pour une variable. Permet d'orienter la suite du programme en fonction d'un choix de différentes valeurs

```
case $variable in
    a)
      echo "variable vaut a"
      ;;
    b)
      echo "variable vaut b"
      ;;
    *)
      echo "variable ne vaut ni a, ni b"
      ;;
esac
```

  - Plus approprié que « if » quand il y a un nombre important de choix

while
- 5/6 -

while

- Teste une condition au début de la boucle et continue **tant que** la condition est vraie

```
while read line
do
    echo "Ligne : $line
done < fichier
```

  - Trouve son utilisation dans des situations où le nombre de répétitions n'est pas connu dès le départ

select
- 6/6 -

select

- Structure de contrôle de type boucle qui permet d'afficher de manière cyclique un menu
- Variables réservées :
  - PS3 : représente le promptutilisé pour la saisie
  - REPLY : contient l'indice de l'item selectionné

select

- 6/6 -

```
PS3="Votre choix : "
select item in "- Sauvegarde -" "- Restauration -" "- Fin -"
do
        echo "Vous avez choisi l'item $REPLY : $item"
        case $REPLY in
        1)
                sauve
                ;;
        2)
                restaure
                ;;
        3)
                echo "Fin du script"
                exit 0
                ;;
        *)
                 echo "Choix incorrect"
                 ;;
        esac
done
```

Les fonctions

Les fonctions

Définition :

```
mafonction() {
        commande1
        commande2
}
```

ou

```
function mafonction {
        commande1
        commande2
}
```

Code retour :

```
$?
Return (0 à 255)
```

Variables :

- Globales
- Locales

Passage d'arguments :

```
$1, $2, …, $*, $@, $#
```

Les expressions régulières
Les chaînes de caractères

Les expressions régulières

Utilisées notamment via grep, egrep :

- Décrivent une recherche textuelle

Caractère	Signification
.	Un caractère quelconque
?	Le caractère précédent, 0 ou 1 fois
*	Le caractère précédent, 0 ou n fois
+	Le caractère précédent, 1 ou n fois
[xy]	Un caractère compris dans la classe (ici, soit x, soit y)
[x-y]	Un caractère compris entre x et y
[^xy]	N'importe quel caractère sauf x ou y
^	Le début de la ligne
$	La fin de la ligne
|	Opérateur « ou »
(..)	Groupement de caractères

Gestion de fichiers

Gestion de fichiers

Création, redirections des entrées/sorties standards :

```
exec 0< fichier1
exec 1> fichier2 2> fichier3
exec 1> fichier4 2>&1
```

Redirection vers descripteurs supplémentaires :

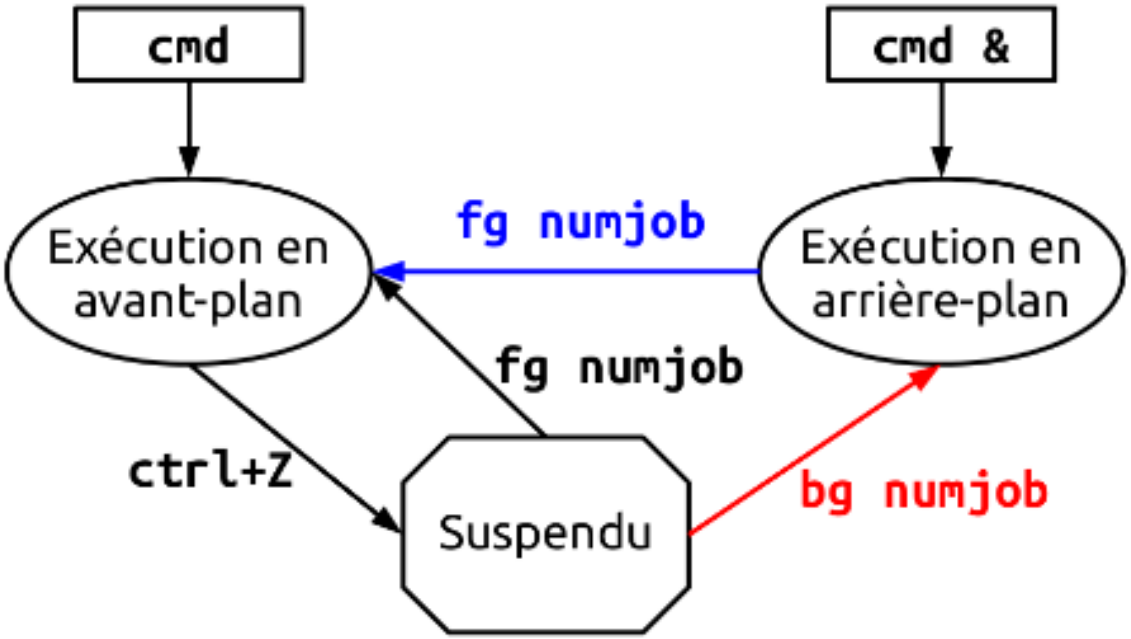
```
exec 3< in.txt
exec 4> out.txt
```

Fermeture fichier :

```
exec 3<&-
exec 4>&-
```

Gestion de processus

Foreground & background



Gestion de processus

Les signaux

- 2/3 -

Signaux :

Nom du signal	N°	Signification
HUP	1	hangup : envoie un signal de réinitialisation
INT	2	Interruption, Généré à partir du clavier. CTRL+C. Tue le processus
TERM	15	Généré via la commande kill. Tue le processus proprement
KILL	9	Généré via la commande kill. Tue le processus brutalement

Commandes de contrôle :

```
ps
pstree
kill
```

Gestion des signaux:

```
trap 'echo "Signal INT recu" ; sleep 3 ; exit 1' INT
```

Le filtre SED

Remplacement rapide : sed

Stream editor : « Éditeur de flux »

- Manipuler des fichiers texte
- Le flux en entrée (fichier ou autre) est traité ligne par ligne

Deux méthodes :

- **« Classique » :** commande sur flux d'entrée et récupération de sortie

```
Sed
```

- **« Directe » :** sed -i, commande appliquée directement sur le fichier passé en entrée

```
Sed
```

Pratique pour faire de la substitution :

```
Sed
```

Processeur de texte AWK

AWK
- 1/2 -

Permet une recherche de chaînes de caractères et l’exécution d’actions sur les lignes sélectionnées

Utile pour :

- Transformer des fichiers
- Récupérer de l’information
- Transformer des flux de sortie

AWK
- 2/2 -

Dawan
Training is our way

Exécution :

```
awk [-Fs] [-v variable] [-f fichier de commandes] 'program' fichier
```

- -F : Spécifie les séparateurs de champ
- -v : Définie une variable utilisée à l'intérieur du programme
- -f : Les commandes awk sont lues à partir d'un fichier

```
awk -F: '{print $1}' /etc/passwd
```

- Séparateur de champ : ':'
- Afficher le premier champ de toutes les lignes du fichier /etc/passwd

```
awk '$2 ~ /[0-9]+/ { print $3 }' agenda.txt
```

- Affiche la 3ème colonne des lignes dont la 2ème colonne est un nombre

Programmation Shell

\(^__-)/

Dawan / Pierre Sablé / 2019-09

Version 1.0