Shell

Le *Shell* est l'interprète de ligne commande fourni par Unix a votre login :

- Il peut lancer des commandes
- Exécuter des boucles
- Effectuer des calculs
- Etc…

Langage dédié pour le lancement de commandes ET interpréteur interactif.

- Travail en interprétant des lignes.
- Le premier mot est la commande, les autres des paramètres.
- Les commandes peuvent être :
  - des commandes internes (choses que le *shell* sait faire)
  - des programmes externes.

Le cycle de traitement d'une commande shell

- Le développement du tilde (substitution de ~nom et ~), le remplacement des paramètres et des variables (calcul sur les variables, *parameter expansion*), la substitution de commandes (`...` ou $(...)), l'évaluation arithmétique *arithmetic expansion*.

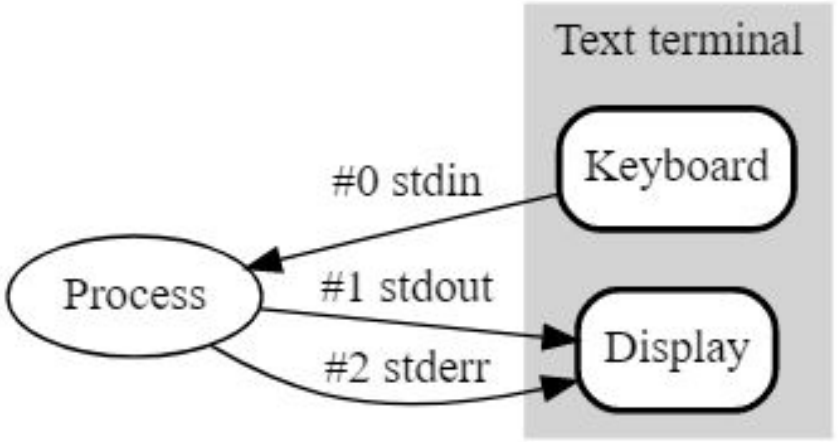
- Découpage des mots sur le résultat de l'étape 1, sauf si IFS est vide.
  - IFS, ou *input field separators* (ou *internal field separators*), est une variable de *shell* Unix définissant les séparateurs de champ reconnus par l'interpréteur. Les IFS « de base sont » l'espace, la tabulation (\t), le saut de ligne (\n).

- Le développement des noms de fichiers (*Pathname / filename expansion*)

- *Quote removal* (suppression des \n, ' et " non littéraux).

Les flux

- flux d’entrée standard: numéro 0
- flux de sortie standard: numéro 1
- flux de sortie d’erreur: numéro 2



Source : Wikipedia

Les redirections de flux

- Redirection du flux d'entrée : *commande < fichier*
- Redirection du flux de sortie standard :
  - écrasante : *commande > fichier*
  - additive : *commande >> fichier*
  - *Nota : > est la version simplifiée de 1>*
- Redirection du flux de sortie d'erreur :
  - Écrasante : *commande 2> fichier*
  - additive : *commande 2>> fichier*
- le flux de sortie standard d'une commande peut être redirigé comme le flux d'entrée d'une autre commande : *commande1* | *commande2 (tube ou pipe)*

Les redirections de flux

- Redirection des deux flux de sortie dans un même fichier :
  - *commande > fichier 2>&1*

- attention l'ordre des deux redirections importe !!
  - Ex. - écrire sur la sortie d'erreur : *echo message 1>&2*

Les redirections de flux : exemples

- Quelles redirections sont équivalente à cp f1 f2
  - cat f1 > f2
  - cat < f1 > f2
  - tee f2 < f1 > /dev/null
    - Tee lit l'entrée standard et l'écrit à la fois dans le résultat standard et dans un ou plusieurs fichiers
  - cat f1 | tee f2 > /dev/null

Les variables

- Positionner une variable : *VARIABLE=valeur*
  - Cette variable ne sera disponible que dans le *Shell* d'éxecution
- Appeler une variable : *echo $VARIABLE* ou *echo ${VARIABLE}*
- Il est positionner une variable dans l'environnement Système :
  - export *VARIABLE=valeur*
- Affectation conditionnelle : echo ${VARIABLE:-valeur}
  - si « VARIABLE » non définie alors VARIABLE = valeur

Affectations conditionnelles

Expression	est remplacée par
${variable}	la variable.
${variable:-mot}	la variable si elle existe sinon par le mot.
${variable:=mot}	la variable si elle existe sinon par mot qui devient la valeur associée à variable.
${variable:?mot}	la variable si elle existe, sinon affiche mot et le processus s'arrête.
${variable:+mot}	le mot si la variable existe, sinon rien.
${variable#forme}	la variable après troncature de son début par la plus petite séquence correspondant à la forme.
${variable##forme}	la variable après troncature de son début par la plus grande séquence correspondant à la forme.
${variable%forme}	la variable après troncature de la fin par la plus petite séquence correspondant à la forme.
${variable%%forme}	la variable après troncature de la fin par la plus grande séquence correspondant à la forme.
$(commande)	le résultat de l'exécution de commande.
'commande'	Pareil que $(commande).

Les variables

- Obtenir la longueur de la chaîne : *${#parameter}*
- Il est possible d'inhiber l'expansion des variables en les mettant entre des simples quotes (').
  - echo '$VARIABLE'
- On peut demander l'expansion mais empêcher le découpage en mots, on utilise des caractères double quotes (").
  - for i in "e e"; do echo $i; done

Les variables

- Les extensions de variable sont des blocs introduits par des accolades
  - Pour créer des séquences : *echo a{1..10}*
  - Pour créer des ensembles *: echo a.{tex,toto,truc,directory}*

- Variables spéciales :
  - $? : code de retour de la dernière commande exécutée.
  - $! : *Process IDentifier* du dernier processus lancé en parallèle.
  - $$: PID du processus en cours.
  - $PWD : le *print working directory*
  - $LANG : langue par défaut
  - …

Les patterns

- Le *shell* comprend un langage de motifs :
  - ? désigne un caractère quelconque.
  - * désigne une chaîne quelconque.
  - [ ] désigne un bloc crochet.

- Un traitement spécial est fait lorsqu'on est dans le contexte des noms de fichier :
  - ? et * ne reconnaissent plus un point initial ni un /
  - le motif doit reconnaître un fichier existant pour être remplacé sinon il est laissé tel quel.

Les boucles conditionnelles

- **for** var **in** suite_de_jetons; **do** commandes; **done**
- **while** condition; **do** commandes; **done**
- **until** condition; **do** commandes; **done**
- **Exemples**
  - for i in f* ; do echo $i ; done
  - x=0; while [$x -lt 10]; do echo $((x++)); done
  - until false; do echo "Counter = $count"; ((count++)); sleep 2; done

Les conditions

- **if** commande renvoyant un code de retour; **then** instructions; **fi**
  - si la condition que vous spécifiez est évaluée à une valeur de sortie nulle (vraie), le *shell* exécute les commandes dans la section *then* du code.
- **elif** commande2 renvoyant un code de retour **then** instructions
- **else** instructions
- **case** word **in**

  pattern1) command ;;

  esac
  - permet d'orienter la suite du programme en fonction d'un choix de différentes valeurs.

Les conditions - exemple

```
case $1 in
        "--create")
        echo "Creating new file $2«
        touch $2
        ;;
        "--delete")
        echo "Deleting file $2"
        rm $2
        ;;
        *)
        echo "Not a valid argument"
        ;;
esac
```

Les tests

- [op1 test op2]
- [test op2]
- Composition possible avec –a (ET logique) et -o (OU logique)
- ! avant une expression signifie la négation
- -eq, -neq, -ge, -gt, -le.... comparaison arithmétique
- -n, -z, =, != comparaison de chaînes de caractères
- -f, -x, -nt, -d, -e, ... tests sur les fichiers.

Les scripts

- La première ligne est la ligne du *shebang*.
  - #!/bin/sh
- contient le chemin absolu vers l'interpréteur du script.
- l'interpréteur désigné dans la ligne du *shebang* sera lancé pour interpréter le script.
- Code de retour nul = tout va bien !
- Code de retour non nul = un problème a eu lieu.

Le Script : un contexte d'exécution particulier

- Les paramètres positionnels :
  - $0 : nom de la commande
  - $1 : premier paramètre
  - $2 : second paramètre…
- $* : tous les paramètres vus comme un seul mot
- $@ : tous les paramètres vus comme des mots séparés
- $# : nombre de paramètres sur la ligne de commande
- shift n : décaler de n à gauche les paramètres positionnels.
  - Le paramètre positionnel x reçoit alors la valeur de x-n

Les fonctions

- Nom défini par l'utilisateur qui permet d'utiliser comme une commande atomique une commande composée, avec de nouveaux paramètres positionnels.
- nom_de_fonction(argument1 argument2 …)

```
{
    commande
    commande2
}
```

- Appel : nom_de_fonction argument1 argument2 …

Les fonctions

- La commande composée sera de l'une des deux formes suivantes :
  - (instructions) □ exécution dans un *sous-shell*, modifications de l'environnement invisibles à l'extérieur
  - {instructions ;} □ exécution dans le shell courant, modifications de l'environnement visibles à l'extérieur

Exercice

- Créer un script qui demande à l'utilisateur de saisir une note et qui affiche un message en fonction de cette note :
  - « très bien » si la note est entre 16 et 20 ;
  - « bien » lorsqu'elle est entre 14 et 16 ;
  - « assez bien » si la note est entre 12 et 14 ;
  - « moyen » si la note est entre 10 et 12 ;
  - « insuffisant » si la note est inférieur à 10.

Exercice

- Créer un script qui affiche « Hello World ! »

Corrigé

```
#!/bin/bash
 echo « Hello World !"
```

Exercice

- Créer un script qui demande à l'utilisateur de saisir une note et qui affiche un message en fonction de cette note :
  - « très bien » si la note est entre 16 et 20 ;
  - « bien » lorsqu'elle est entre 14 et 16 ;
  - « assez bien » si la note est entre 12 et 14 ;
  - « moyen » si la note est entre 10 et 12 ;
  - « insuffisant » si la note est inférieur à 10.

Corrigé

```
#!/bin/bash
echo "Entrez votre note :"
read -r note

if [ "$note" -ge 16 ]; then
    echo "très bien"
elif [ "$note" -ge 14 ]; then
    echo "bien"
elif [ "$note" -ge 12 ]; then
    echo "assez bien"
elif [ "$note" -ge 10 ]; then
    echo "moyen"
else
    echo "insuffisant"
fi
```