Christophe Blaess

Fonctionnement du shell

Ecriture de scripts shell

Ce document a été réalisé sur système Linux avec les logiciels libres :

- **Formation+** pour préparer le cours complet,
- **LaTeX** pour produire les transparents et le support de cours imprimé,
- **Xfig** pour les graphiques vectoriels,
- **The Gimp** pour les images et les photos.

Support de cours : version 15

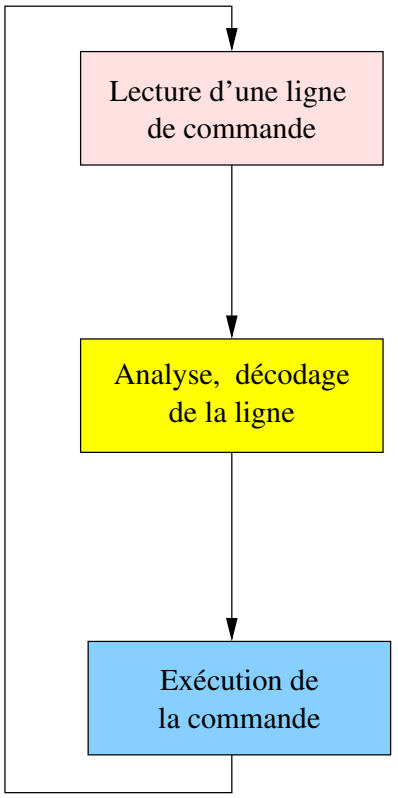
Ecriture de scripts shell

Fonctionnement du shell

Christophe Blaess

Interprétation d'une commande

Boucle générale du shell



Affichage du symbole d'invite
`$ # % >`
sur la sortie d'erreur
Lecture d'une ligne sur l'entrée standard
ou
Lecture d'une ligne de texte depuis le script

Expansion des accolades
Développement du tilde
Remplacement des paramètres
Substitution des commandes
Evaluation arithmétique
Découpage des mots
Développement des noms de fichiers

Commande externe
(processus fils)
ou
Commande interne
(meme processus)

Expansion des accolades

A vous. . .

Que donnent les lignes suivantes :

```
$ echo image{05,06,07}.gif
```

```
$ echo image{01}.gif
```

`prefixe_{nom1,nom2,nom3}_suffixe`
est développé ainsi :
`prefixe_nom1_suffixe prefixe_nom2_suffixe prefixe_nom3_suffixe`

Si le shell ne trouve pas de virgule entre les accolades, il n'effectue pas de modification.

Ne pas confondre ces accolades et celles qui servent à regrouper les commandes composées que nous verrons plus loin.

NB : Le shell Bourne original n'effectue pas l'expansion des accolades, ni les versions du shell Korn antérieures à *Ksh 93*.

Développement du tilde

A vous...

Qu'affichent les lignes suivantes :

```
$ echo ~
```

```
$ echo ~user02/bin
```

```
$ echo ~inexistant
```

```
$ echo a~
```

`~nom/suite/du/chemin`
est remplacé par
`/home/nom/suite/du/chemin`

La chaîne `~/` est remplacée par le contenu de la variable d'environnement `HOME` si elle existe (et le répertoire personnel de l'utilisateur sinon).

Le répertoire personnel est déterminé en consultant `/etc/passwd`.

Si le nom d'utilisateur n'existe pas, la chaîne est inchangée.

Le remplacement n'a lieu que si le `~` est en début de chaîne.

Remplacement des paramètres

A vous...

Qu'affichent les lignes suivantes :

```
$ a = azert  (avec des espaces autour du =)
```

```
$ a=azert (sans espaces)
$ echo $a
```

```
$ echo $ay
```

```
$ echo ${a}y
```

`${nom}`
est remplacé par la valeur de la variable ou du paramètre `nom`.

Lorsqu'il n'y a pas d'ambiguïté possible, on peut utiliser `$nom` à la place de `${nom}`.

Lors d'une affectation de variable ne jamais mettre d'espace autour du signe `=`.

A vous. . .

```
$ a=azert
$ echo ${a-défaut}
```

```
$ echo ${inexistante-défaut}
```

```
$ vide=""
$ echo ${vide-défaut}
```

```
$ echo ${vide:-défaut}
```

```
$ echo ${vide:=défaut}
$ echo $vide
```

Le `$` est un véritable opérateur qui offre de multiples possibilités.

`${variable-défaut}`

est remplacé par `défaut` si la `variable` n'existe pas.

`${variable:-défaut}`

effectue aussi le remplacement par `défaut` si la variable contient une chaîne vide.

`${variable=défaut}`

remplit la `variable` avec la valeur par `défaut` si elle n'existait pas ; fournit le contenu de la variable.

`${variable:=défaut}`

remplit aussi la `variable` si elle contient une chaîne vide.

Il existe aussi des opérateurs `${var+val}`, `${var:+val}`, `${var?val}` et `${var:?val}` rarement utilisés.

A vous. . .

```
$ a=azertyuiop
$ echo ${#a}
```

```
$ echo ${a#aze}
```

```
$ echo ${a%iop}
```

```
$ echo ${a:3:4}
```

```
$ echo ${a:4}
```

`${#variable}`

est remplacé par le nombre de caractères contenus dans la `variable`.

`${variable#prefixe}`

est remplacé par le contenu de la `variable` une fois qu'on a éliminé le `prefixe`.

`${variable%suffixe}`

est remplacé par le contenu de la `variable` une fois qu'on a éliminé le `suffixe`.

`${variable:debut:longueur}`

est remplacé par la sous-chaîne commençant à la position `debut` (comptée à partir de zéro) et ayant la `longueur` indiquée. Sans `longueur` on va jusqu'à la fin de la chaîne.

Les shells Korn antérieurs à *Ksh 93* n'offrent pas l'extraction de sous-chaîne.

Utilisation avancée

Le préfixe ou le suffixe éliminés peuvent contenir des caractères génériques du shell, comme `?` ou `*`.

`${variable#motif}`

supprime le **préfixe** le plus **court** correspondant au `motif`.

`${variable##motif}`

supprime le **préfixe** le plus **long** correspondant au `motif`.

`${variable%motif}`

supprime le **suffixe** le plus **court** correspondant au `motif`.

`${variable%%motif}`

supprime le **suffixe** le plus **long** correspondant au `motif`.

Substitution de commandes

Qu'affichent les lignes suivantes :

```
$ a=$(date)
$ echo $a
```

```
$ a=$(date +%Y)
$ echo $a
```

```
$ a=$(ls)
$ echo $a
```

```
$ a=$(ls -l)
$ echo $a
$ echo "$a"
```

Ancienne construction :
`‘commande‘`
Nouvelle construction :
`$(commande)`

La `commande` est exécutée, et la construction `$(...)` est remplacée par le résultat de sa sortie standard.

L'éventuel saut de ligne final de la sortie standard est éliminé lors de la substitution de commande

L'ancienne construction est plutôt déconseillée pour deux raisons :
- risque de confusion entre `‘commande‘` et les apostrophes encadrant une `’chaîne’`,
- difficulté d'imbriquer des appels (utiliser des *backslashes* \).

Le shell Bourne original n'accepte que la construction `‘commande‘`.

Evaluation arithmétique

Ancienne construction (obsolète) : `$[expression]`
Nouvelle construction : `$((expression))`

Le contenu de l'expression est évalué selon les règles de l'arithmétique entière, et la construction est remplacée par le résultat du calcul.

A vous...

```
$ echo $((12 + 3*(5+3)*5))
```

```
$ a=10
$ a=$((a+1))
$ echo $a
```

```
$ echo $((3.5))
$ echo $((010))
$ echo $((08))
```

L'évaluation est limitée à l'arithmétique entière. Pour des calculs avec des nombres réels, voir l'utilitaire `bc`(1).

Dans les expressions arithmétiques, les variables peuvent être préfixées de l'opérateur `$`, mais ce n'est pas obligatoire.

Les parenthèses ajoutent des priorités dans les expressions.

Les constantes sont :
- décimales par défaut : `9`, `45`, `13600`...
- octales si elles commencent par zéro : `0644`, `0755`...
- hexadécimales si elles commencent par `0x` : `0x4C`...
- en base n si elles commencent par n`#` : `2#10101010`, `10#014`...

NB : Des affectations de variables sont possibles au sein de l'expression arithmétiques, mais on perd de la lisibilité : `i=$((j=5))`.

Le shell Bourne original n'offre pas d'arithmétique, il faut invoquer la commande externe `expr`(1).

Opérateurs arithmétiques

`+ -`	plus et moins unaires
`! ~`	négations logique et binaire
`* / %`	multiplication, division, reste
`+ -`	addition, soustraction
`< < >>`	décalages binaires
`<= >= < >`	comparaisons
`== !=`	égalité, différence
`&`	ET binaire
`^`	OU exclusif binaire
`|`	OU binaire
`&&`	ET logique
`||`	OU logique
`= *= /= %= += -= <<= >>= &= ^= |=`	affectations

- Le shell Korn 93 accepte les valeurs réelles (extension par rapport au standard SUSv3).
- Le shell Korn 88 arrondit les valeurs réelles à l'entier inférieur.
- Les shells Korn (88 et 93) refusent les valeurs héxadécimales, et ne reconnaissent pas les constantes octales. C'est un écart par rapport à SUSv3.

Protections et découpage en mots

A vous...

```
$ a=azerty
$ echo $a
$ echo "$a"
$ echo '$a'
```

```
$ echo "$(ls)"
$ echo '$(ls)'
```

```
$ echo "$((1+2))"
$ echo '$((1+2))'
```

```
$ echo "ab{0,1}cd"
$ echo "~/bin"
```

Les chaînes entre apostrophes simples ' ne sont jamais modifiées.

Dans une chaîne entre guillemets droits ", les modifications suivantes ont lieu :
- remplacement des paramètres et variables
- évaluation arithmétique
- substitution de commande

mais ni l'expansion des accolades, ni le développement du tilde.

Les guillemets vont servir à préserver l'intégrité d'une chaîne sans qu'elle soit découpée en autant d'arguments qu'elle comporte de mots.

A vous...

```
$ touch "un fichier"
$ ls -l un fichier
$ ls -l "un fichier"
$ ls -l un\ fichier
```

```
$ a=un fichier
```

```
$ a="un fichier"
$ ls -l $a
$ ls -l "$a"
```

Les mots de la ligne sont séparés en utilisant comme délimiteurs les caractères contenus dans la variable d'environnement `IFS`.

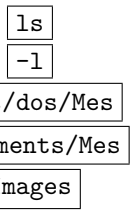
Par défaut, `IFS` contient les caractères :
- espace,
- tabulation,
- saut de ligne.

Les caractères séparateurs précédés par un *backslash* \ ne sont pas pris en compte, mais le *backslash* est supprimé.

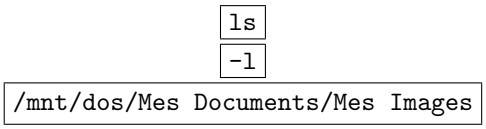
Les portions de ligne encadrés par des guillemets ou des apostrophes ne sont pas découpées.

En raison de diverses problèmes de sécurité, la variable `IFS` est réinitialisée à chaque fois qu'un shell démarre.

```
ls -l /mnt/dos/Mes Documents/Mes Images/
```




```
ls -l /mnt/dos/Mes\ Documents/Mes\ Images/
ls -l "/mnt/dos/Mes Documents/Mes Images/"
ls -l '/mnt/dos/Mes Documents/Mes Images/'
```



Développement des noms de fichiers

A vous. . .

```
$ echo /dev/tty?
```

```
$ echo /dev/tty?0
```

```
$ echo /dev/tty*
```

```
$ echo "/dev/tty*"
```

```
$ echo /dev/ttyZZ*
```

```
$ ls /dev/ttyZZ*
```

```
$ ls "/dev/tty*"
```

Les mots contenant des caractères génériques `*`, `? [ ]` sont considérés comme des motifs, et remplacés par la liste des fichiers correspondants.

- `?` : n'importe quel caractère (sauf le *slash* `/` et le point initial) ;
- `*` : n'importe quelle chaîne – éventuellement vide – de caractères (sauf `/` et `.` initial) ;
- `[ ]` : liste de caractères.

Si aucun nom de fichier ne correspond au motif, le mot n'est pas modifié.

Le développement des noms de fichiers n'a pas lieu dans les chaînes entre guillemets ou apostrophes.

Travaux pratiques

Exercice numéro 1

Manipulation des variables

Exercice numéro 2

Syntaxe d'affectation de variable (1)

Exercice numéro 3

Syntaxe d'affectation de variable (2)

Exercice numéro 4

Protection des expressions

Exercice numéro 5

Protection du contenu d'une variable

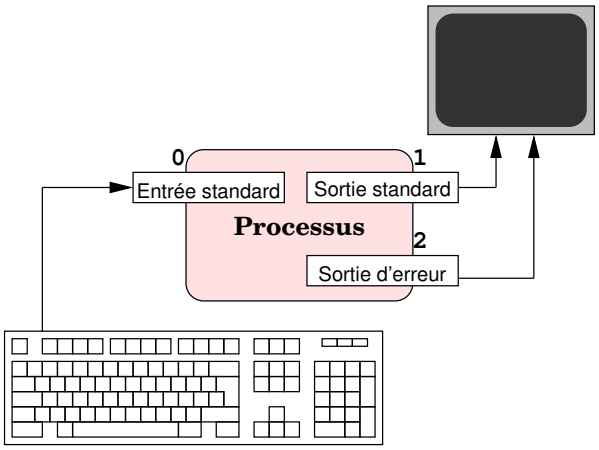
Exercice numéro 6

Développement des caractères génériques

Exercice numéro 7

Invocation de commande et évaluation arithmétique

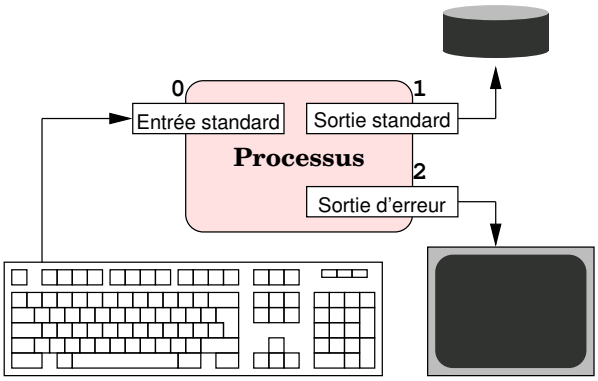
Entrées-sorties standards d'un processus



Ces canaux d'entrée-sortie sont mis en place par le shell avant de lancer une commande.

Redirections de la sortie standard

Avec la notation `commande > fichier` la sortie standard du processus est redirigée vers le fichier plutôt que sur le terminal.

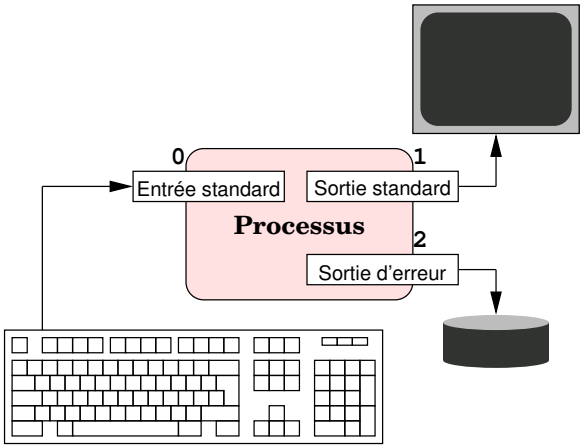


La redirection `commande >> fichier` fait un **ajout** en fin de fichier sans écrasement des données précédentes.

```
$ grep root /etc/*
$ grep root /etc/* > standard
$ grep root /etc/* >> standard
```

Redirections de la sortie d'erreur

La notation `commande 2> fichier` redirige la sortie d'erreur du processus. Et `commande 2>> fichier` ajoute en fin de fichier.



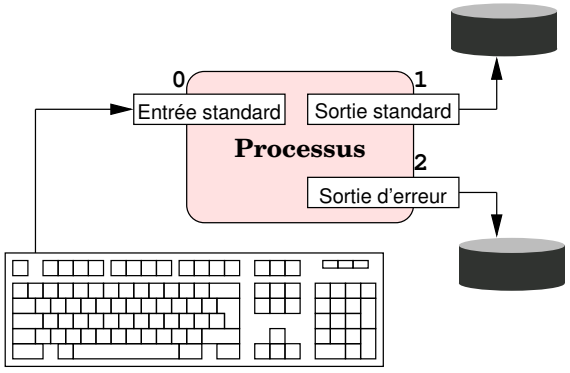
On utilise souvent `commande 2>/dev/null`

```
$ grep root /etc/*
$ grep root /etc/* 2> erreur

$ grep root /etc/* 2> /dev/null
```

Redirections des deux sorties

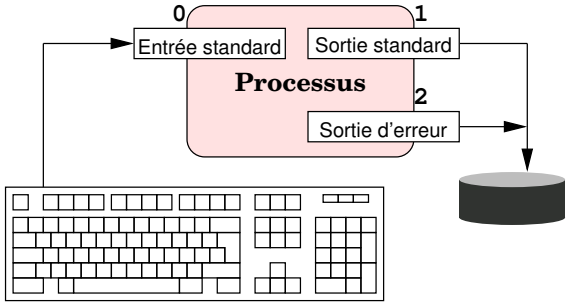
Les sorties standard et d'erreur peuvent être redirigées simultanément avec `commande > fichier 2> erreur`.




```
$ grep root /etc/* > standard 2> erreur
```

Regroupement des deux sorties

La notation `commande > fichier 2>&1` redirige la sortie standard puis recopie sa configuration sur celle de la sortie d'erreur.



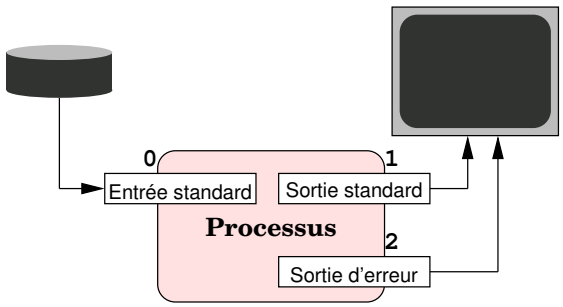
Il n'est plus possible de distinguer les deux sorties une fois le regroupement mis en place.

On peut réaliser le regroupement inverse `1>&2` comme dans `echo "Message d'erreur" >&2`

```
$ grep root /etc/* > sorties 2>&1
```

Redirections de l'entrée standard

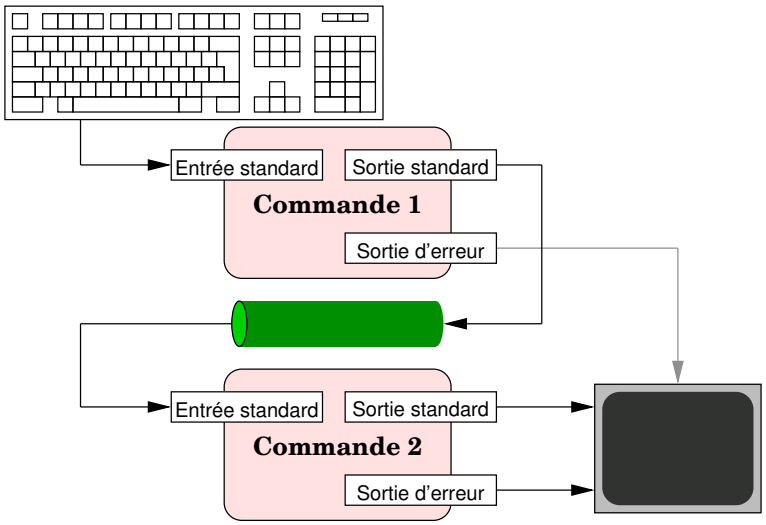
Avec la notation `commande < fichier` l'entrée standard est lue dans le fichier.



La redirection est transparente pour le processus, il ne voit pas de différence par rapport au terminal.

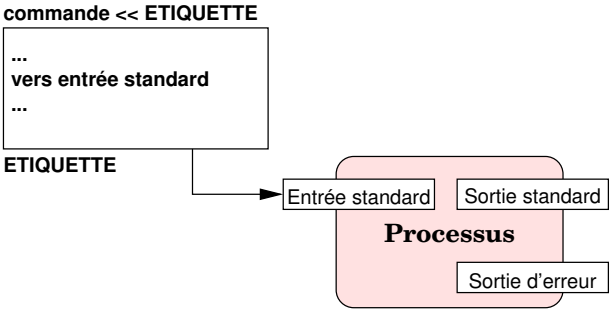
Tubes entre processus

Le symbole *pipe* dans `commande_1 | commande_2` crée un tube entre la sortie standard de la première commande et l'entrée standard de la seconde.



Document en ligne

La redirection des *documents en ligne* permet d'automatiser des tâches interactives.



Avec la redirection `commande <<- ETIQUETTE` les tabulations en début de ligne sont ignorées dans le document en ligne.

Variables du shell

Affectation d'une variable avec la notation
`variable=valeur`

Pas d'espace autour du signe égal !

Lors d'une affectation encadrer les chaînes contenant des espaces par des guillemets ou des apostrophes.

Tableaux

Il existe des tableaux dont les cases sont remplies avec
`tableau[$i]=valeur`
et consultées avec l'expression
`${tableau[$i]}`

Le rang *zéro* correspond la variable sans indice :
`$tableau` est équivalent à `${tableau[0]}`.

L'expression `${#tableau[@]}` correspond au nombre de cases dans le tableau.

NB : les tableaux n'existent que dans les shells modernes (Bash 2, Ksh 93).

Paramètres non-modifiables

`$0`	Nom du script ou du shell
`$1 $2... $9 ${10}...`	Paramètres positionnels (arguments ligne de commande)
`$#`	Nombre de paramètres sauf `$0`
`$* $@`	Tous les paramètres sauf $0
`$$`	PID du shell en cours
`$!`	PID du dernier processus lancé à l'arrière-plan
`$?`	Code de retour de la dernière commande

`$*` et `$@` correspondent tout deux à `$1 $2 ... $n`

`"$*"` correspond à `"$1 $2 ... $n"`

`"$@"` correspond à `"$1" "$2" ... "$n"`

En conclusion, il faut toujours utiliser `"$@"`.

Paramètres utilisables en lecture

`PWD`	Répertoire en cours
`REPLY`	Chaîne de réponse de la commande `read`
`RANDOM`	Valeur aléatoire
`SECONDS`	Durée d'exécution du shell
`OPTARG`	Dernier argument traité par `getopts`
`OPTIND`	Prochain argument à traiter par `getopts`
`HOSTNAME`	Nom de la machine hôte
	...

Paramètres de configuration

`IFS`	Séparateurs de mots
`PATH`	Chemin de recherche des commandes
`HOME`	Répertoire personnel de l'utilisateur
`PS1`	Symbole d'invite principal
`PS2, PS3, PS4`	Symboles d'invite suplémentaires
`LANG`	Langue de l'utilisateur
`LC_ALL`	Localisation générale
	...

Environnement d'un processus

A vous...

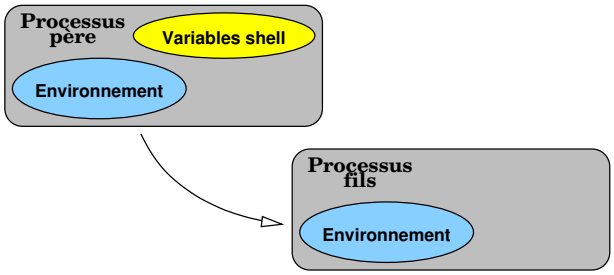
```
$ variable=abc
$ /bin/sh
[sh]$ echo $variable
```

```
[sh]$ exit
$ echo $variable
```

```
$ export variable
$ /bin/sh
[sh]$ echo $variable
```

```
[sh]$ variable=def
[sh]$ echo $variable
[sh]$ exit
$ echo $variable
```

Pour qu'une variable du shell soit visible par ses futurs processus fils, il faut *exporter* la variable dans l'*environnement* du père.



Le processus fils reçoit une *copie* de l'environnement de son père. Il ne peut en aucun cas modifier l'environnement de son père.

La commande `env`(1) sans argument affiche l'environnement du processus.

Travaux pratiques

Exercice numéro 8

Redirection des sorties standard et d'erreur

Exercice numéro 9

Aspects avancés des redirections (1)

Exercice numéro 10

Aspects avancés des redirections (2)

Exercice numéro 11

Aspects avancés des redirections (3)

Exercice numéro 12

Aspects avancés des redirections (4)

Exercice numéro 13

Paramètres de la ligne de commande

Exercice numéro 14

Extraction de préfixes et suffixes

Exercice numéro 15

Développement des arguments en ligne de commande

Index