Programmation système

Shell et Commandes UNIX

Tuyêt Trâm DANG NGOC
<dntt@u-cergy.fr>

Université de Cergy-Pontoise

Objectifs du shell

1. Fournir une interface pour la saisie de commande
2. Redirection des entrées/sorties standards
3. Analyser les commandes
   - substitution de noms de fichiers
   - substitution de variables
   - redirection d'entrées/sorties
4. Exécution de commandes
   - mode synchrone
   - mode asynchrone
5. Fournir un langage interprété

Types de Shell

Shell	Nom	Description
Bourne Shell	sh	Shell disponible sur toute plateforme UNIX
C shell	csh	Shell développé par BSD
Korn shell	ksh	Bourne Shell étendu par l'AT&T
Bourne Again Shell	bash	Version améliorée de sh et csh. Fourni le plus souvent avec Linux.
Zero Shell	zsh	shell avec beaucoup de fonctionnalités : typage, substitution et complétion très poussées
Tenex	tcsh	csh étendu
rc	rc	Implémentation pour UNIX du shell de Plan 9
es	es	Extension de rc

Pour l'administrateur système : **/bin/sh**.

Utilisation du shell

Deux modes d'utilisation :

- interactif : en ligne de commande.
  1. Présente une invite (*prompt*) à l'utilisateur et attend que celui-ci tape une commande ;
  2. Exécute* la commande tapée par l'utilisateur
  3. Retour en 1.
- non-interactif : scripts shell, batch
  1. Lit une ligne du fichier
  2. Exécute* les instructions données dans la ligne du fichier
  3. Passe à la ligne suivante
  4. Retour en 1

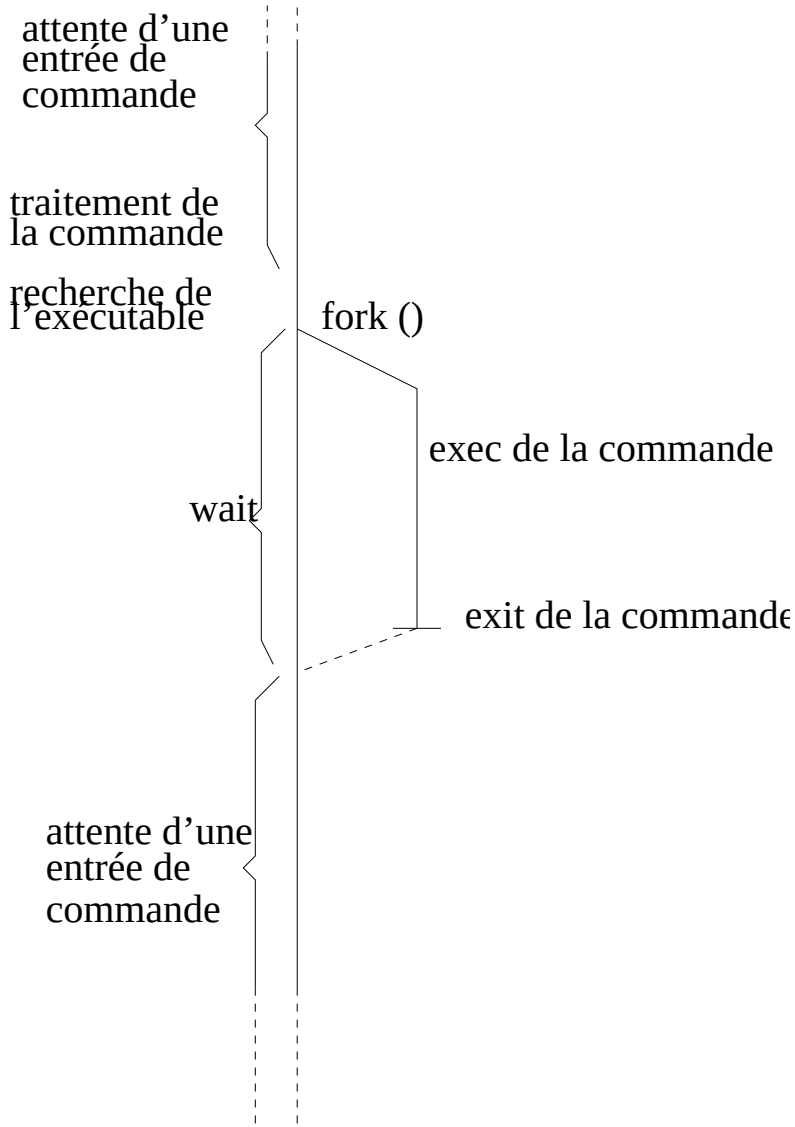
  Le programme s'arrête lorsqu'il n'y a plus de ligne à lire ou lorsqu'un instruction spéciale (**exit** ou **return**) est rencontrée.

Convention : l'invite est :

- $ pour l'utilisateur normal en sh, bsh, bash
- % pour l'utilisateur normal en csh, tcsh
- # pour root dans tous les shells

Exécution d'une commande

1. Attente d'une entrée de commande
2. traitement des caractères spéciaux de la commande
3. recherche de l'exécutable. Si non trouvé, afficher un message d'erreur et revenir en 1.
4. fork () + exec () de la commande à lancer
5. wait de la commande
6. Revenir en 1.



Caractères spéciaux

Caractères	Description
tabulation, espace	Délimiteur de mot
retour chariot	Fin de la commande à exécuter
&	Lance une commande en tâche de fond
; ; ;	Séparateur de commande
*?[][^]	Substitution de noms de fichiers
&& ||!	Opérateurs booléens
' " \	Caractères de quotation
<><<>> ‘ |<><< & > & << − >|	Opérateurs de redirection d'entrées sorties
$	Valeur d'une variable
#	Début de commentaires
(){}	Groupement de commande

Mots réservés

Mots réservés	Signification
case ... in esac	
for ... in ... do ... done	
if ... then ... elif ... else ... fi	
while ... do ... done	
until ... do ...done	
break, continue	
return, exit	

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

$

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
ls is /bin/ls
$
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
ls is /bin/ls
$ type cat
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
ls is /bin/ls
$ type cat
cat is /bin/cat
$
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
ls is /bin/ls
$ type cat
cat is /bin/cat
$ type gcc
```

Commandes internes (builtins)

Les commandes internes sont réalisées de manière interne par le shell lui-même ; c'est-à-dire qu'il n'y a pas de création de processus pour exécuter la commande. Ces commandes ne possèdent pas d'exécutables associés puisqu'elles sont codées en interne au shell. Une méthode pour identifier les builtins est d'utiliser la commande interne **type**.

```
$ type cd
cd is a shell builtin
$ type echo
echo is a shell builtin
$ type ls
ls is /bin/ls
$ type cat
cat is /bin/cat
$ type gcc
gcc is /usr/bin/gcc
$
```

Commandes internes (builtins)

alias	bg	builtin
bind	cd	chdir
command	echo	eval
exec	exit	export
fc	fg	getopts
hash	jobid	jobs
pwd	read	readonly
set	setvar	shift
trap	type	ulimit
umask	unalias	unset
wait		.
	:	

Commandes (externes)

Les commandes qui ne sont pas internes sont des exécutables qui peuvent être trouvés dans la hiérarchie des répertoires :

- soit directement si le chemin complet est spécifié
- soit trouvé par le shell en explorant les répertoires spécifiés dans la variable d'environnement `PATH`.

/bin/cat	/bin/chmod	/bin/cp	/bin/date
/bin/kill	/bin/ln	/bin/ls	/bin/mkdir
/bin/mv	/bin/ps	/bin/pwd	/bin/rmdir
/bin/sleep	/usr/bin/awk	/usr/bin/basename	/usr/bin/bc
/usr/bin/bg	/usr/bin/chgrp	/usr/bin/cmp	/usr/bin/comm
/usr/bin/cut	/usr/bin/diff	/usr/bin/dirname	/usr/bin/find
/usr/bin/grep	/usr/bin/head	/usr/bin/join	/usr/bin/man
/usr/bin/more	/usr/bin/nohup	/usr/bin/paste	/usr/bin/sed
/usr/bin/sort	/usr/bin/tail	/usr/bin/time	/usr/bin/top
/usr/bin/touch	/usr/bin/uniq	/usr/bin/vi	/usr/bin/w
/usr/bin/wc	/usr/bin/xargs	/usr/sbin/chown	

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

$

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
gateau rateau
$
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
gateau rateau
$ ls [^br]ateau
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
gateau rateau
$ ls [^br]ateau
gateau
$
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
gateau rateau
$ ls [^br]ateau
gateau
$ ls [a-c]*
```

Caractères de substitution

Car.	Commande
*	N'importe quelle séquence de caractères
?	N'importe quel caractère
[]	N'importe quel caractère choisi dans les caractères donnés entre crochets
[^]	n'importe quel caractère sauf ceux dans les caractères donnés entre crochets
[–]	n'importe quel caractère dans la plage de caractères donnés entre crochets

```
$ ls *
achat bateau chat chateau cheval chien gateau rateau
$ ls ?ateau
bateau gateau rateau
$ ls *ateau
bateau chateau gateau rateau
$ ls [gr]ate*
gateau rateau
$ ls [^br]ateau
gateau
$ ls [a-c]*
achat bateau chat chateau cheval chien
```

Variables

- Une variable est une suite de caractères alphanumérique associée éventuellement à une valeur.
- Une variable doit être de la forme : $[A-Za-z\_][A-Za-z\_]*$, par exemple, `toto`, `TOTO`, `MA_Valeur`, `t`, sont des variables.
- La notion de typage n'existe pas en shell, toutes les valeurs sont des chaînes de caractères, et par conséquent :
  - il n'y a pas de déclaration de variables.
  - il n'y a pas de valeur numérique.

Variables : assignation et utilisation

$

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

```
$ TITI=abcdefghijk
```

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

```
$ TITI=abcdefghijk
$
```

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

```
$ TITI=abcdefghijk
$ echo $TITI
```

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

```
$ TITI=abcdefghijk
$ echo $TITI
abcdefghijk
$
```

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

```
$ TITI=abcdefghijk
$ echo $TITI
abcdefghijk
$ XYZ=3
```

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

Variables : assignation et utilisation

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

```
$ TITI=abcdefghijk
$ echo $TITI
abcdefghijk
$ XYZ=3
$
```

Variables : assignation et utilisation

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

```
$ TITI=abcdefghijk
$ echo $TITI
abcdefghijk
$ XYZ=3
$ set
```

Variables : assignation et utilisation

- Assignation de variable
  variable=*valeur*
- Consultation de variable :
  $*variable* ou ${*variable*}
- Visualisation de toutes les variables définies :
  set

```
$ TITI=abcdefghijk
$ echo $TITI
abcdefghijk
$ XYZ=3
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=abcdefghijk
USER=dntt
XYZ=3
$
```

Variables : destruction et protection

$

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

```
$ TITI=raton-laveur
```

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

```
$ TITI=raton-laveur
$
```

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

```
$ TITI=raton-laveur
$ unset XYZ
```

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

```
$ TITI=raton-laveur
$ unset XYZ
$
```

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

```
$ TITI=raton-laveur
$ unset XYZ
$ set
```

- Destruction d'une variable :
  unset variable
- Protection d'une variable en écriture :
  readonly variable
- Liste des variables protégées en écriture :
  readonly

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
$
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
$ TITI=koala
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
$ TITI=koala
TITI : is read only
$
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
$ TITI=koala
TITI : is read only
$ readonly
```

Variables : destruction et protection

- Destruction d'une variable : *unset variable*
- Protection d'une variable en écriture : *readonly variable*
- Liste des variables protégées en écriture : *readonly*

```
$ TITI=raton-laveur
$ unset XYZ
$ set
HOME=/users/dntt
LOGNAME=dntt
PATH=/bin :/usr/bin
PS1='$ '
PWD=/users/dntt/A
SHELL=/usr/local/bin/ksh
TERM=xterm
TITI=raton-laveur
USER=dntt
$ readonly TITI
$ TITI=koala
TITI : is read only
$ readonly
TITI
$
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :
$

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que $*nom_variable* :

```
$ MOT=volume
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :

```
$ MOT=volume
$
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :

```
$ MOT=volume
$ echo $MOT42
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :

```
$ MOT=volume
$ echo $MOT42

$
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :

```
$ MOT=volume
$ echo $MOT42

$ echo ${MOT}42
```

Précaution sur les variables

Mieux vaut utiliser ${*nom_variable*} que ${*nom_variable*} :

```
$ MOT=volume
$ echo $MOT42

$ echo ${MOT}42
volume42
$
```

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"
$

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
+ : not found
$
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
+ : not found
$ titi=$titi+1
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
+ : not found
$ titi=$titi+1
$
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
+ : not found
$ titi=$titi+1
$ echo $titi
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Valeurs numériques et variables

Il n'y a pas de type "numérique en shell"

```
$ titi=3
$ echo $titi
3
$ titi=$titi + 1
+ : not found
$ titi=$titi+1
$ echo $titi
3+1
$
```

Il faut utiliser des commandes spécifiques pour « utiliser » les chaînes à valeur numérique (cf. **expr**)

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
Le contenu de la variable TOTO est :
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
Le contenu de la variable TOTO est :
$ export TOTO
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
Le contenu de la variable TOTO est :
$ export TOTO
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
Le contenu de la variable TOTO est :
$ export TOTO
$ ./affiche_toto
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Exportation de variables : **export**

Une variable définie n'est accessible que dans le shell courant. Elle n'est pas transmise aux processus fils. Pour qu'elle soit transmise aux processus fils, il faut utiliser le mot-clef **export** *variable*

Soit le script `affiche_toto` :

```
echo "Le contenu de la variable TOTO est : $TOTO"
```

```
$ TOTO=42
$ echo $TOTO
42
$ ./affiche_toto
Le contenu de la variable TOTO est :
$ export TOTO
$ ./affiche_toto
Le contenu de la variable TOTO est : 42
$
```

Raccourcis : *variable*=*valeur* **export** *variable* $\Leftrightarrow$ **export** *variable*=*valeur*

Une variable exportée est appelée variable d'environnement

Variables d'environnement

Les variables ne servent qu'aux processus qui en ont besoin !

Variables d'environnement courantes utilisées par de nombreuses commandes (ainsi que par le shell).

HOME	Le répertoire « maison » de l'utilisateur
USER	Le login utilisateur
SHELL	Le shell utilisateur
PATH	La liste des chemins dans laquelle le shell cherchera les commandes exécutables qui seront tapés sans indication de chemin
TERM	Le type de terminal
PS1	L'invite (*prompt*)
PWD	Le répertoire courant
LANG	Le langage utilisé

Variables d'environnement : **env**

L'affichage des variables d'environnement se fait par la commande **env**

```
$
```

Variables d'environnement : **env**

L'affichage des variables d'environnement se fait par la commande **env**

```
$ env
```

Variables d'environnement : **env**

L'affichage des variables d'environnement se fait par la commande **env**

```
$ env
 USER=dntt
HOME=/users/dntt
PAGER=more
PS1=[dnttbanzai]
ENV=/users/dntt/.kshrc
VISUAL=vi
ORACLE_BASE=/usr/oracle
LOGNAME=dntt
TERM=xterm
DISPLAY= :0.0
SHELL=/usr/local/bin/ksh
CLASSPATH=/users/dntt/jdklib :.
LD_LIBRARY_PATH=/usr/lib :/usr/oracle/OraHome1/lib :/lib :
CVSROOT= :ext :dntt@persee.prism.uvsq.fr :/home/cvs
JAVA_HOME=/usr/local/linux-sun-jdk1.4.2
PATH=/sbin :/bin :/usr/sbin :/usr/bin :/usr/local/sbin :
/usr/local/bin :/usr/X11R6/bin :/usr/oracle/OraHome1/bin
```

Variables d'environnement : **env**

L'affichage des variables d'environnement se fait par la commande **env**

```
$ env
 USER=dntt
HOME=/users/dntt
PAGER=more
PS1=[dnttbanzai]
ENV=/users/dntt/.kshrc
VISUAL=vi
ORACLE_BASE=/usr/oracle
LOGNAME=dntt
TERM=xterm
DISPLAY= :0.0
SHELL=/usr/local/bin/ksh
CLASSPATH=/users/dntt/jdklib :.
LD_LIBRARY_PATH=/usr/lib :/usr/oracle/OraHome1/lib :/lib :
CVSROOT= :ext :dntt@persee.prism.uvsq.fr :/home/cvs
JAVA_HOME=/usr/local/linux-sun-jdk1.4.2
PATH=/sbin :/bin :/usr/sbin :/usr/bin :/usr/local/sbin :
/usr/local/bin :/usr/X11R6/bin :/usr/oracle/OraHome1/bin
 PRINTER=lasernb
LANG=fr
EDITOR=vi
WINDOWID=33554446
BLOCKSIZE=K
$
```

Substitution de variables

`${variable:-valeur}`	utilise la valeur de la variable si elle est positionnée et est non nulle, sinon utilise la valeur donnée.
`${variable:=valeur}`	utilise la valeur de la variable si elle est positionnée et est non nulle, sinon affecte la valeur à la variable et utilise la valeur donnée.
`${variable:?valeur}`	utilise la valeur de la variable si elle est positionnée et est non nulle, sinon écrit la valeur sur l'erreur standard et fait un exit avec un statut non nul
`${variable:+valeur}`	utilise une chaîne vide si la variable est positionnée et est non nulle, sinon, utilise la valeur.
`${#variable}`	utilise la taille de la valeur de la variable

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

$

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?                                                    $
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

```
$ echo $$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

```
$ echo $$
902
$
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

```
$ echo $$
902
$ echo $!
```

Variables prédéfinies : $?, $$, $!

Syntaxe	Commande
$?	Code de retour de la dernière commande exécutée
$$	PID du shell en train de s'exécuter
$!	PID de la dernière commande lancée en tâche de fond
$-	options passées au shell courant

```
$ echo $?
0
$ ls abcdef
ls : abcdef : No such file or directory
$ echo $?
1
$ ./long_programme &
$ ps -U dntt
 PID TT STAT TIME COMMAND
902 p4 S 0 :00.23 /bin/sh
1114 p4 R 0 :03.14 ./long_programme
1115 p4 R+ 0 :00.00 ps -U dntt
$
```

```
$ echo $$
902
$ echo $!
1114
$
```

Caractères spéciaux

' " \ changent la façon dont le shell interprête les caractères spéciaux

Symbole	Signification
' (single-quote)	le shell ignore tout caractère spéciaux entre deux '
" (double-quote)	le shell ignore tout caractère spéciaux entre deux ", à l'exception de $ et \ et ‘
\ (antislash ou backslash)	le shell ignore le caractère spécial suivant le \
‘ (backquote ou antiquote)	le shell exécute ce qu'il y a entre deux ‘

Caractères spéciaux : exemples

$

Caractères spéciaux : exemples

```
$ ls
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo ‘whoami‘ et le ${TITI} et les c* ; ls
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo `whoami` et le ${TITI} et les c* ; ls
dntt et le raton-laveur et les chat chien
chat chien poisson
$
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo `whoami` et le ${TITI} et les c* ; ls
dntt et le raton-laveur et les chat chien
chat chien poisson
$ echo '`whoami` et le ${TITI} et les c* ; ls'
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo `whoami` et le ${TITI} et les c* ; ls
dntt et le raton-laveur et les chat chien
chat chien poisson
$ echo '`whoami` et le ${TITI} et les c* ; ls'
`whoami` et le ${TITI} et les * ; ls
$
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo ‘whoami‘ et le ${TITI} et les c* ; ls
dntt et le raton-laveur et les chat chien
chat chien poisson
$ echo ’‘whoami‘ et le ${TITI} et les c* ; ls’
‘whoami‘ et le ${TITI} et les * ; ls
$ echo "‘whoami‘ et le ${TITI} et les c* ; ls"
```

Caractères spéciaux : exemples

```
$ ls
chat chien poisson
$ whoami
dntt
$ TITI=raton-laveur
$ echo ‘whoami‘ et le ${TITI} et les c* ; ls
dntt et le raton-laveur et les chat chien
chat chien poisson
$ echo ’‘whoami‘ et le ${TITI} et les c* ; ls’
‘whoami‘ et le ${TITI} et les * ; ls
$ echo "‘whoami‘ et le ${TITI} et les c* ; ls"
dntt et le raton-laveur et les * ; ls
$
```

Ordre d'évaluation de la ligne de commande

Ordre de gauche à droite

1. Redirection des entrées/sorties
2. Substitution des variables
3. Substitution des noms de fichiers

- Automatisation des actions
- Langage de programmation interprêté

Structure d'un script shell

#!	*Sur la première ligne : interprèteur du présent script (#! suivi du chemin complet du shell utilisé plus d'éventuels arguments)*
#commentaires	*Les ligne de commentaire sont précédées de #*
code	*Le code est donné ligne par ligne*

```
 #!/bin/sh
# Ce script shell "progdebut"
# dit bonjour, liste les
# fichiers et processus
# dit au revoir et sort
echo "Bonjour !"
ls
ps
echo "au revoir"
exit 0
```

$

Structure d'un script shell

#!	*Sur la première ligne : interprèteur du présent script (#! suivi du chemin complet du shell utilisé plus d'éventuels arguments)*
#commentaires	*Les ligne de commentaire sont précédées de #*
code	*Le code est donné ligne par ligne*

```
#!/bin/sh
# Ce script shell "progdebut"
# dit bonjour, liste les
# fichiers et processus
# dit au revoir et sort
echo "Bonjour !"
ls
ps
echo "au revoir"
exit 0
```

```
$ ./progdebut
```

Structure d'un script shell

#!	*Sur la première ligne : interprèteur du présent script (#! suivi du chemin complet du shell utilisé plus d'éventuels arguments)*
#commentaires	*Les ligne de commentaire sont précédées de #*
code	*Le code est donné ligne par ligne*

```
 #!/bin/sh
# Ce script shell "progdebut"
# dit bonjour, liste les
# fichiers et processus
# dit au revoir et sort
echo "Bonjour !"
ls
ps
echo "au revoir"
exit 0
```

```
$ ./progdebut
 Bonjour ! chat chateau
cheval chien
PID TTY TIME CMD
6450 pts/3 00 :00 :00 ksh
7152 pts/3 00 :00 :00 ps
au revoir
$
```

Variables de passage de paramètre à un script (variables positionelles)

Syntaxe	Commande
$*	Tous les paramètres passés au script shell sous la forme de mots séparés
$@	Tous les paramètres passés au script shell
$#	Le nombre de paramètres passés au script shell
$0	Nom du script shell
$1, $2, ...	Le premier, deuxième, ... paramètre

Passage de paramètres : exemple

```
 #!/bin/sh
# Mon programme qui affiche les parametres de la ligne de
commande
echo "* Le nom du programme est : $0"
echo "* Le troisieme parametre est : $3"
echo "* Le nombre de parametre est : $#"
echo "* Tous les parametres (mots individuels) : $*"
echo "* Tous les parametres : $@"
exit 0
$
```

Passage de paramètres : exemple

```
 #!/bin/sh
# Mon programme qui affiche les parametres de la ligne de
commande
echo "* Le nom du programme est : $0"
echo "* Le troisieme parametre est : $3"
echo "* Le nombre de parametre est : $#"
echo "* Tous les parametres (mots individuels) : $*"
echo "* Tous les parametres : $@"
exit 0
$ ./script2.sh un "deux" "trois quatre" cinq
```

Passage de paramètres : exemple

```
 #!/bin/sh
# Mon programme qui affiche les parametres de la ligne de
commande
echo "* Le nom du programme est : $0"
echo "* Le troisieme parametre est : $3"
echo "* Le nombre de parametre est : $#"
echo "* Tous les parametres (mots individuels) : $*"
echo "* Tous les parametres : $@"
exit 0
$ ./script2.sh un "deux" "trois quatre" cinq
 * Le nom du programme est : ./script2.sh
* Le troisieme parametre est : trois quatre
* Le nombre de parametre est : 4
* Tous les parametres (mots individuels) : un deux trois
quatre cinq
* Tous les parametres : un deux trois quatre cinq
$
```

Passage de paramètres : précaution

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$
```

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
```

Passage de paramètres : précaution

```
#!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$ ./script3_faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
```

```
#!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
```

Passage de paramètres : précaution

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$ ./script3_faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf un0 un1
un2
$
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
```

Passage de paramètres : précaution

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$ ./script3_faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf un0 un1
un2
$
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
$
```

Passage de paramètres : précaution

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$ ./script3_faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf un0 un1
un2
$
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
$ ./script3_vrai.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
```

Passage de paramètres : précaution

```
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
$ ./script3_faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf un0 un1
un2
$
 #!/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
$ ./script3_vrai.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf dix onze
douze
$
```

Décalage de paramètres : **shift**

```
 #!/bin/sh
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
exit 0
$
```

Décalage de paramètres : **shift**

```
 #!/bin/sh
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
exit 0
$ ./script4.sh un deux
```

Décalage de paramètres : **shift**

```
 #!/bin/sh
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
echo "$# : arg1 = $1 , arg2 = $2 ; total : $"
shift
exit 0
$ ./script4.sh un deux
 2 : arg1 = un , arg2 = deux ; total : un deux
1 : arg1 = deux , arg2 = ; total : deux
0 : arg1 = , arg2 = ; total :
shift : can't shift that many
$
```

Remarque sur **shift**

```
for i in $#
do
echo $1
shift
done
```

$\Leftrightarrow$

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

Commandes de test : **test, [**

test *expression*
[*expression*]

permet d'évaluer une expression.

- Si vrai, renvoie 0, sinon, renvoie 1.
- S'il n'y a pas d'expression, renvoie 1 (false).

test *expression* est équivalent à [*expression*]

-d fic	vrai si le fichier existe et est un répertoire
-f fic	vrai si le fichier existe et est un fichier « ordinaire »
-h fic	vrai si le fichier existe et est un lien symbolique
-x fic	vrai si le fichier existe et est autorisé en exécution
-w fic	vrai si le fichier existe et est autorisé en écriture
-r fic	vrai si le fichier existe et est autorisé en lecture
ch1 = ch2	si les deux chaines sont identiques
ch1 != ch2	si les deux chaines sont différentes

n1 -eq n2	si les deux nombres sont numériquement égaux
n1 -neq n2	si les deux nombres sont numériquement inégaux
n1 -lt n2	si n1 est numériquement inférieur à n2
n1 -gt n2	si n1 est numériquement supérieur à n2
n1 -le n2	si n1 est numériquement inférieur ou égal à n2
n1 -ge n2	si n1 est numériquement supérieur ou égal à n2
! exp1	vrai si l'expression est fausse (et vice-versa)
exp1 -a exp2	vrai si les deux expressions sont vraies (AND)
exp1 -o exp2	vrai si au moins une expression est vraie (OR)

Exemple d'utilisation de **test**

$

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
0
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
0
$ [ 3 -gt 42 -a -x fic-executable ]
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
0
$ [ 3 -gt 42 -a -x fic-executable ]
$
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
0
$ [ 3 -gt 42 -a -x fic-executable ]
$ echo $?
```

Exemple d'utilisation de **test**

```
$ test -f fic-qui-existe
$ echo $?
0
$ test -f fic-qui-n-existe-pas
$ echo $?
1
$ [ 3 -lt 42 -a -x fic-executable ]
$ echo $?
0
$ [ 3 -gt 42 -a -x fic-executable ]
$ echo $?
1
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
$ echo $?
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
$ echo $?
0
$
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
$ echo $?
0
$ feifez
```

Valeur de retour

L'exécution d'une commande modifie la variable de retour $?. Des commandes telles que **ls**, **cd**, **rm**, ... modifieront la variable de retour $? suivant que l'opération se soit bien passée ou non.

Note :

: est une commande interne retournant toujours vrai (0).

```
ls wzwzwzw
ls : zwzwz : No such file or directory
$ echo $?
1
$ ls fichier
fichier
$ echo $?
0
$  :
$ echo $?
0
$ feifez
feifez : not found
```

Listes de commandes

- Une séquence de commande s'écrira :
  `commande1 ; commande 2 ;...`
- ou
  `commande1`
  `commande2`
  `...`
- Attention, chaque nouvelle commande (non interne) est exécutée dans un nouveau processus.
- Sous-shell : `(commande1 ; commande2)`
- Exemple :
  `(cd repertoire ; touch toto)`
  exécute la commande **touch toto** sans changer la valeur du répertoire initial.

Branchement conditionnel : **if-then-elif-else-fi**

if *liste-commandes-1*
then *liste-commandes-2*
elif *liste-commandes-3* *< – – – – – autant de fois que nécessaire*
else *liste-commandes-4* *< – – – – – si nécessaire*
fi

```
 if ls monfichier
then echo "la commande ls monfichier a reussi"
else echo "la commande ls monfichier a echoue"
fi
```

Exemple d'utilisation de **if-then-elif-else-fi**

```
 if [ -d toto ]
then echo "toto est un répertoire"
elif [ -h toto ]
then echo "toto est un lien symbolique"
else echo "faut pousser l'investigation plus loin"
fi
```

Opérateurs logiques du shell : &&, ||

Souvent utilisé pour une forme compacte et élégante.

ET logique :

cmd1 && cmd2	$\Leftrightarrow$	if cmd1 then cmd2 fi

Si la commande 1 réussi, alors faire la commande 2.

`toto.c -o toto && ./toto` OU logique :

cmd1 || cmd2	$\Leftrightarrow$	if ! cmd1 then cmd2 fi

Si la commande 1 a échouée, alors faire la commande 2.

`cat toto || touch toto`

Branchement conditionnel : **case-esac**

case *valeur* **in**
motif **)** *liste-commandes* ; ; $<$ − − − − − *autant de fois que*
esac

Exécute la liste-commandes suivant le motif reconnu.

Le motif à reconnaitre peut s'exprimer sous forme d'expression rationnelle utilisant les métacaractères : $*?[]!-$ |

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
$
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
$ ./question YeS
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$
```

Exemple avec **case-esac**

```
 case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
reponse idiote
$
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
reponse idiote
$ ./question bof
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
reponse idiote
$ ./question bof
Couci-couça
$
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
reponse idiote
$ ./question bof
Couci-couça
$ ./question raton-laveur
```

Exemple avec **case-esac**

```
case $reponse in
[Yy][eE][sS] | [oO][uU][iI] | OK) echo "Tu approuves" ; ;
[Nn][oO]*) echo "Tu désapprouves" ; ;
bof) echo "Couci-couça" ; ;
pfff*) echo "Pas la peine de répondre" ; ;
*) echo "reponse idiote" ; ;
esac
```

```
$ ./question YeS
Tu approuves
$ ./question OK
Tu approuves
$ ./question nO
Tu désapprouves
$ ./question pfffjhfrfe
Pas la peine de répondre
$ ./question areuh
reponse idiote
$ ./question bof
Couci-couça
$ ./question raton-laveur
reponse idiote
$
```

Boucle **for-do-done**

for *variable* **in** *liste de mots*
do
liste-commandes
done

La variable prend successivement les valeurs de la liste de mots, et pour chaque valeur, liste-commandes est exécutée.

Exemple avec **for-do-done**

```
 for i in un deux trois
quatre
do
echo "chapitre $i"
done
```

```
 chapitre un
chapitre deux
chapitre trois
chapitre quatre
```

```
 for i in /etc/f*
do
echo $i
done
```

```
 /etc/fbtab
/etc/fstab
/etc/ftpusers
```

Boucle **while-do-done**

while *liste-commandes-1*
do
liste-commandes-2
done

La valeur testée par la commande **while** est l'état de sortie de la dernière commande de liste-commandes-1. Si l'état de sortie est 0, alors le shell exécute liste-commandes-2 puis recommence la boucle.

Boucle **until-do-done**

until *liste-commandes-1*
do
liste-commandes-2
done

Le shell exécute liste-commandes-2 puis teste l'état de sortie de liste-commandes-1. Si l'état de sortie est 0, alors, la boucle est recommencée.

Contrôle du flux d'exécution : **break, continue**

break ou **break** *n*

permet de sortir d'une boucle **for**, **while** ou **until**. Si n est précisé, il indique le nombre d'imbrication concernée par le **break**.

continue *n*

permet de passer à l'itération suivante d'une boucle **for**, **while** ou **until**. Si n est précisé, il indique le nombre d'imbrication concernée par le **continue**.

Fonction

On peut regrouper les commandes au sein d'une fonction.
Une fonction se définit de la manière suivante :

```
 nom_fonction ()
{
liste-commandes
}
```

Les paramètres au sein de la fonction sont accessibles via $1, $2, ... $@, $#. L'appel d'une fonction se fait de la manière suivante :

```
nom_fonction parametre1 parametre2...
```

Une fonction doit être déclarée avant de pouvoir être exécutée.

Code de retour : return, exit

return *n*

Renvoie une valeur de retour pour la fonction shell.

exit *n*

Provoque l'arrêt du shell courant avec un code retour de n si celui-ci est spécifié. S'il n'est pas spécifié, il s'agira de la valeur de retour de la dernière commande exécutée.

Gestion des signaux : **trap, kill**

Il est possible de faire intercepter des signaux par le shell. Ces signaux sont générés par des évènements lancés par l'utilisateur lors de l'exécution du shell, par exemple :

- 1 : coupure de ligne
- 2 : arrêt (CTRL+C)
- 9 : destruction (NON INTERCEPTABLE)
- 15 : fin de process

trap "*commandes*" *numéro de signal*

Intercepte le signal donné lorsqu'il se présente et effectue la commande. L'exécution se poursuit ensuite en reprenant à l'endroit de l'interruption.

```
trap " rm -f "/tmp/FICTMP ; exit" 2
```

kill -*signal pid*

Envoyer un signal à un processus de pid donné

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

$

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$ tutu=‘expr $tutu + 1‘
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$ tutu=‘expr $tutu + 1‘
$
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$ tutu=‘expr $tutu + 1‘
$ echo $tutu
```

Petits calculs numériques : **expr**

expr *chaine*

évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$ tutu=‘expr $tutu + 1‘
$ echo $tutu
4
$
```

Manipulation d'arguments : **xargs** OPTIONS *commande*

xargs

construire et exécuter des lignes de commandes à partir de l'entrée standard

-n *nbargs* regroupe les arguments par *nbargs*

```
echo "$*"
$
```

Manipulation d'arguments : **xargs** OPTIONS *commande*

xargs

construire et exécuter des lignes de commandes à partir de l'entrée standard

-n *nbargs* regroupe les arguments par *nbargs*

```
 echo "$*"
$ ls /etc | xargs -n2
```

Manipulation d'arguments : **xargs** OPTIONS *commande*

xargs

construire et exécuter des lignes de commandes à partir de l'entrée standard

-n *nbargs* regroupe les arguments par *nbargs*

```
 echo "$*"
$ ls /etc | xargs -n2
 X11 aliases
amd.map apmd.conf
auth.conf bluetooth
crontab csh.cshrc
...
$
```

Manipulation d'arguments : **xargs** OPTIONS *commande*

xargs

construire et exécuter des lignes de commandes à partir de l'entrée standard

-n *nbargs* regroupe les arguments par *nbargs*

```
 echo "$*"
$ ls /etc | xargs -n2
 X11 aliases
amd.map apmd.conf
auth.conf bluetooth
crontab csh.cshrc
...
$
$
```

Manipulation d'arguments : **xargs** OPTIONS *commande*

xargs

construire et exécuter des lignes de commandes à partir de l'entrée standard

-n *nbargs* regroupe les arguments par *nbargs*

```
 echo "$*"
$ ls /etc | xargs -n2
 X11 aliases
amd.map apmd.conf
auth.conf bluetooth
crontab csh.cshrc
...
$
$ ls ${HOME}/src/JAVA | xargs javac
```

- Shell interactif - shell non interactif : Un shell est interactif lorsqu'il demande à l'utilisateur d'entrer au clavier une instruction à exécuter, exécute celle-ci, et ainsi de suit.
  Le shell est non interactif lorsqu'il exécute un fichier contenant une suite de commandes, plus couramment appelé script.
- Shell attaché à un tty : Un shell possède trois descripteurs de fichiers : entrée standard, sortie standard, sortie d'erreur. Chacun de ces descripteurs peut être "relié" soit à un tty (pseudo-terminal), soit à un socket, soit à un fichier, soit à /dev/null.
- Login shell - non-login shell : Tout shell démarré via **login** sur un terminal texte, **rlogin**, **telnet**, **ssh** , ou lors d'un job cron ou at est un login shell.

Fichiers d'initialisation

<table>
<tr><th>Shell</th><th>sh</th><th>ksh</th><th>bash</th><th>csh</th><th>tcsh</th></tr>
<tr><td>À chaque session</td><td colspan="3">/etc/profile</td><td colspan="2">/etc/csh.login</td></tr>
<tr><td>À chaque session (mode login shell)</td><td colspan="2">${HOME}/.profile</td><td>${HOME}/.bash_profile</td><td colspan="2">${HOME}/.login</td></tr>
<tr><td>À chaque déconnexion</td><td colspan="2"></td><td>.bash_logout</td><td colspan="2">.logout</td></tr>
<tr><td>À chaque lancement (hors mode login shell)</td><td></td><td>.kshrc</td><td>.bashrc</td><td>.cshrc</td><td>.tcshrc</td></tr>
</table>

Fichiers d'initialisation I

- bsh (Bourne shell)
  - /etc/profile, si le shell est un login shell ;
  - ${HOME}/.profile, si le shell est un login shell.
- csh (C shell)
  - /etc/csh.cshrc ;
  - /etc/csh.login, si le shell est un login shell ;
  - ${HOME}/.cshrc ;
  - ${HOME}/.login, si le shell est un login shell.
- sh ou ksh (Korn shell)
  - /etc/profile, si le shell est un login shell ;
  - ${HOME}/.profile, si le shell est un login shell ;
  - ${ENV}.
- bash (Bourne Again shell)
  - si le shell est un login shell,
  - /etc/profile ;

Fichiers d'initialisation II

- ${HOME}/.bash_profile, ou à défaut ${HOME}/.bash_login, ou à défaut ${HOME}/.profile ;
- si le shell est un non-login shell interactif,
- ${HOME}/.bashrc ;
- si le shell est un non-login shell non interactif,
- ${BASH_ENV}.

- tcsh (Turbo C shell)
  - /etc/csh.cshrc ;
  - /etc/csh.login, si le shell est un login shell ;
  - ${HOME}/.tcshrc, ou à défaut ${HOME}/.cshrc ;
  - ${HOME}/.login, si le shell est un login shell.