Les caractères suivants possèdent une signification :

`;`	séparateur de commande
`&`	exécution en arrière plan
`( )`	regroupement de commande
`{ }`	bloc de commandes (pour les fonctions)
`|`	tube (pipe en anglais)
`> <`	symboles de redirections
`?*[ ]~ !`	caractères génériques pour les noms de fichiers
`" ' \`	chaînes de caractères
`$`	substitution

Pour les chaînes de caractères

- Les caractères compris entre deux ' (apostrophes) sont traités tels quels, sans signification particulière
- Les caractères compris entre deux " (guillemets) sont traités tels quels, sauf pour $ et pour les autre ".

Définition de motifs de paramètres fichiers :

`*`	zéro ou plusieurs caractères
`?`	exactement un caractère
`[abc...]`	ensemble de caractères. On peut définir un ensemble en donnant les bornes séparées par un tiret, par exemple [a-z0-0] signifie « tout caractère alphabétique minuscule ou chiffre ».
`[!abc...]`	caractère qui ne sont pas dans l'ensemble.
`{motif1,... motif2,...}`	choix multiple de motifs
`~nom`	répertoire personnel de l'utilisateur `nom`

Exemples de motifs :

`?a*`	tous les mots ayant 'a' en deuxième caractère.
`[!a-e]*`	tous les mots dont le premier caractère n'est pas a,b,c,d ou e.
`*{ab,cd}`	tous les mots finissant par "ab" ou "cd".

Enchaînement de plusieurs commandes :

`cmd1 ; cmd2`	exécute cmd1 puis cmd2
`cmd1 | cmd2`	tube, la sortie standard de cmd1 est redirigée dans l'entrée de cmd2
`cmd1 $(cmd2)`	la sortie standard de cmd2 est l'argument (option ou paramètre) donné à cmd1
`cmd1 && cmd2`	exécute cmd2 si et seulement si cmd1 s'est déroulé correctement (valeur de retour == 0)
`cmd1 || cmd2`	n'exécute cmd2 que si cmd1 échoue (valeur de retour différente de zéro)

Exemples d'enchaînement de commandes

`date ; pwd`	exécute date puis affiche le répertoire courant
`date --date ="@$ (cat /proc/stat | grep btime | cut -d ' ' -f 2)"`	affiche la date du dernier boot.
`cat fichier.txt | wc -l`	affiche le nombre de lignes de fichier.txt
`test -f fichier && more fichier`	affiche le fichier si il existe
`test -f fichier || echo "fichier inconnu"`	affiche « fichier incconnu » si il n'existe pas
`test -f fichier && more fichier || echo "fichier inconnu".`	affiche le fichier si il existe, sinon affiche « ficher inconnnu »

Redirection de flux standards

`cmd > fichier`	La sortie standard de cmd est stockée dans fichier
`cmd < fichier`	fichier est envoyé dans l'entrée standard de cmd
`cmd >> fichier`	La sortie standard de cmd est rajoutée à fichier
`cmd 2> fichier`	L'erreur standard est stockée dans fichier
`cmd 2>&1`	Redirige l'erreur standard sur la sortie standard
`Cmd > fichier 2>&1`	Sortie et erreur standard sont stockées dans fichier
`cmd << texte`	Lit l'entrée standard jusqu'à une ligne égale à texte

Variables

`var="chaine"`	Affectation de la valeur « chaine » à la variable var. attention pas d'espace autour du"="
var2=$var	utilisation de la variable $var affectée à var2

Exemple d'utilisation de variables

`x=10 ; echo "la variable \$x contient $x"`	affiche "la variable $x contient 10"

Variables prédéfinies

`$0`	le nom sous lequel le script est exécuté.
`$1, ... $9`	les neufs premiers paramètres de ligne de commande.
`$@`	liste séparée par des espaces de tous les paramètres.
`$#`	nombre de paramètres.
`$?`	code de retour de la dernière commande exécutée.
`$$`	identifiant du processus.

`$!`	identifiant de la dernière commande exécutée en arrière plan.
`$UID`	identifiant de l'utilisateur qui a lancé le script
`$HOME`	le répertoire personnel de l'utilisateur
`$PWD`	répertoire courant

... de nombreuses autres (`$PATH, $TERM, ...`)

Structures de contrôle

`if test ;`
`then`
`  commandes`
`fi`
	`case chaine in`
	`  motif1)`
	`commandes`
	`  ;;`
	`  motif2)`
	`commandes`
	`  ;;`
	`  * )`
	`commandes`
	`  ;;`
	`esac`
	la chaîne est comparée aux motifs du choix multiple. La dernière est le motif par défaut.
		`while test ;`
		`do`
		`  commandes`
		`done`

`if test ;`
`then`
`  commandes`
`else`
`commandes`
`fi`
		`until test ;`
		`do`
		`  commandes`
		`done`

`if test1 ;`
`then`
`  commandes`
`elif test2 ;`
`then`
`  commandes`
`else`
`  commandes`
`fi`
		`for i in liste ; do`
		`  commandes`
		`done`
		
		liste est une suite de mots séparée par des espaces ou des retours à la ligne ("\n")

Calcul d'expressions arithmétiques

Bash permet d'évaluer certaines expressions arithmétique tout en utilisant des variables avec la commande `let` ou en entourant l'expression dans `$((...))`. Dans l'expression, les variables peuvent être utilisées sans le $ devant le nom. Les opérations les plus communes sont les suivantes, pour plus de détails voir "`help let`".

`-,+, *, /, %, **`	Respectivement les opérations moins (unaire et binaire), plus multiplication, division, modulo et puissance
`&, ^, |`	ET, XOR et OR binaires
<, >, ==, = !	Inférieur, supérieur, égal, différent.
=	affectation
<<, >>	Décalage binaire vers la gauche ou la droite
!, ~	Négation logique et binaire
`b#<nombre>`	Expression d'un nombre en base b

Exemple

`a=$((2#011^2#100))`	La variable a est égale à 7

La commande test

La commande test est indispensable pour l'exécution de conditionnelle et de boucles. Son code de retour dépends de la comparaison de chaînes de caractères, de nombres ou bien de l'existence de fichiers. Elle est souvent abrégée par des [] .Pour plus de détails voir "`help test`"

`[ -d fichier ]`	le fichier est un répertoire
`[ -e fichier ]`	le fichier existe
`[ -f fichier ]`	le fichier est un ficher régulier
`[ -r fichier ]`	le fichier existe et est lisible
`[ s1 = s2 ]`	la chaîne s1 est égale à la chaîne s2
`[ s1 != s2 ]`	la chaîne s1 est différente de la chaîne s2
`[ n1 -eq n2 ]`	le nombre n1 est égal à n2
`[ n1 -ge n2 ]`	le nombre n1 est supérieur ou égal à n2
`[ n1 -gt n2 ]`	le nombre n1 est supérieur à n2
`[ n1 -ne n2 ]`	le nombre n1 est différent de n2
`[ c1 ] && [ c2 ]`	condition c1 et condition c2
`[ c1 ] || [ c2 ]`	condition c1 ou condition c2

Les fonctions

Les fonctions se comportent comme une commande externe, et reconnaissent les arguments positionnels ($1, $2,...) comme les arguments des fonctions lors de leur appel. Les fonctions bash ne retourne jamais de valeur, à part peut-être par la commande echo.

Exemple la fonction fact() calcule la factorielle d'un nombre passé en paramètre :

```
function fact
{
    let n=$1
    let res=1
    while [ $n  -gt 1 ]; do
      let res=res*n
      let n=n-1
    done
    echo $res
}
fact 3
```