bash

GNU Bourne-Again Shell

Bash est un interpreteur de commande compatible sh qui exécute les commandes lues depuis l'entrée standard ou un fichier. Bash imcorpore également de nombreuses fonctionnalités des shells Korn et C. Toutes les options simple caractère documentés dans la description de la commande intégré **set** peuvent être utilisés comme option quand le shell est invoqué. En plus, bash interprète les options suivantes :

OPTIONS

-c string Les commandes sont lus depuis string. S'il y'a des arguments après string, ils sont assignés aux paramètres positionnels en commençant par $0

-i Mode interactif

-l Agit comme s'il avait été invoqué comme login shell

-r Mode shell restreins

-s Sans arguments après les options, les commandes sont lues depuis l'entrée standard

-D Une liste de toutes les chaînes précédés par $ est affiché sur la sortie standard. implique -n. Aucune commande n'est exécutée

[-+]O [shopt_option] shopt_option est une option accepté par la commande intégré **shopt**. Si présent, -O définis la valeur de cette option. +O la reset. Si shopt_option n'est pas fournis, les noms et leur valeurs acceptés par **shopt** sont affichés. Avec +O, la sortie est affichée dans un format qui peut être ré-utilisé en entrée.

– Signal la fin des options et désactive le traitement des options qui suivent. Tout ce qui suit est traité comme noms de fichier et arguments.

Bash interprète également des options multi-caractères. Ces options doivent apparaître sur la ligne de commande avant les options simple caractère pour être reconnus.

–debugger le profile debugger est exécuté avant que le shell démarre.

–dump-po-strings Équivalent à **-D**, mais la sortie est au format po GNU gettext

–dump-strings Équivalent à **-D**

–init-file file Éxecute les commandes depuis le fichier spécifié au lieu du fichier système /etc/bash.bashrc et le fichier personnel ~/.bashrc

–rcfile file Éxecute les commandes depuis le fichier spécifié au lieu du fichier système /etc/bash.bashrc et le fichier personnel ~/.bashrc

–login Équivalent à **-l**

–noediting N'utilise pas GNU readline pour lire les lignes de commandes quand le shell est interactif

–noprofile Ne lis ni /etc/profile, ni les fichiers d'initialisation personnel ~/.bash_profile, ~/.bash_login, ou ~/.profile.

–norc Ne lis pas le fichier d'initialisation /etc/bash.bashrc et le fichier d'initialisation personnel ~/.bashrc si le shell est interactif. (par défaut si le shell est invoqué en tant que **sh**)

–posix Force bash à être conforme posix

–restricted Mode restreins

–verbose Équivalent à **-v**

Arguments

Si des arguments restent après le traitement des options, et ni **-c** ni **-s** ne sont présents, le premier argument est supposé être un nom de fichier contenant des commandes shell. Si bash est lancé de cette manière, **$0** est définis au nom de ce fichier, et les paramètres positionnels sont définis avec les arguments restant. Bash lit et exécute les commandes de ce fichier puis se termine. Le code de sortie de Bash est le code de sortie de la dernière commande exécutée dans le script.

Invocation

Un login shell est un shell dont le premier caractère de l'argument zéro est un **-**, ou lancé avec **–login**.

Un shell interactif est un shell lancé sans arguments non-option et sans **-c** et dont l'entrée et la sortie standard sont connectés aux terminaux (comme déterminé par isatty(3)), ou lancé avec **-i**. **PS1** est définis et **$-** inclus **i** si bash est interactif, permettant à un script shell ou un fichier de démarrage de tester cet état.

Fichiers de démarrage

Si un des fichier existe mais ne peut pas être lu, Bash reporte une erreur. Quand Bash est invoqué, il lit et exécute **/etc/profile** s'il existe. Puis il lit et exécute **~/.bash_profile**, **~/.bash_login**, et **~/.profile**, dans cet ordre. L'option **–noprofile** peut être utilisé pour empêcher ce mode. Quand un login shell existe, Bash lit et exécute **~/.bash_logout**. Quand un shell qui n'est pas un login shell est lancé, bash lit et exécute les commandes depuis **/etc/bash.bashrc** et **~/.bashrc**. Peut être inhibé avec **–norc**. **–rcfile** va forcer bash à lire et exécuter les commandes depuis le fichier spécifié au lieu de **/etc/bash.bashrc** et **~/.bashrc**

Quand bash est lancé non intéractivement, pour lancer un script shell, par exemple, il cherche la variable **BASH_ENV**, étend ses valeurs, et utilise la valeur étendue comme nom de fichier à exécuter.

Bash fonctionne comme cette commande au lancement, sauf que PATH n'est pas utilisé :

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

sh

Si bash est invoqué avec le nom **sh**, il tente de se comporter comme la version historique de **sh** et le standard POSIX. Invoqué comme login shell, il tente d'exécuter **/etc/profile** et **~/.profile**, dans cet ordre. Peut être inhibé avec **–noprofile**. Invoqué comme shell interactif, il recherche la variable ENV, l'étend et l'utilise comme nom de fichier à exécuter. Invoqué avec le nom **sh**, il ne tente pas de lire d'autre fichiers. l'option **–rcfile** n'a pas d'effet. Invoqué en shell non interactif, il ne tente pas de lire d'autres fichiers. Il entre en mode POSIX une fois les fichiers de démarrage lancé.

Quand bash est lancé en mode posix, comme avec l'option **–posix**, il suit le standard POSIX pour les fichiers de démarrage. Dans ce mode, les shells interactif étendent la variable **ENV**. Aucun fichier de démarrage n'est exécuté.

Bash tente de déterminer quand il est lancé avec son entrée standard connecté à une connexion réseaux, comme lorsqu'il est exécuté par un service de shell distant, généralement **rshd** ou **sshd**. Si bash détermine qu'il est dans ce mode, il lit et exécute les commandes depuis **~/etc/bashrc** et **~/.bashrc**. Il ne le fera pas s'il est invoqué en **sh**. L'option **–rcfile** peut être utilisé pour forcer à lire un autre fichier, et **–norc** pour inhiber ce mode.

Si le shell est lancé avec l'id utilisateur/group effectif différent de l'id utilisateur/group réel, et que l'option **-p** n'est pas fournis, aucun fichier de démarrage n'est lu, les fonctions shell d'héritent pas le l'environnement, les variables **SHELLOPTS**, **BASHOPTS**, **CDPATH**, et **GLOBIGNORE** sont ignorées, et l'id effectif est définis comme id réel. Si l'option **-p** est fournie, le démarrage est le même, mais l'id effectif n'est pas réinitialisé.

Définitions

Les définitions suivantes sont utilisée dans le reste de ce document :

blank Un espace ou une tabulation

word Une séquence de caractères considérés comme une simple unité par le shell. Également connu comme un **token**

name Un word consistant uniquement de caractères alphanumérique et "_", et commençant avec un caractère alphabétique ou un "_". Également appelé un **identifier**

metacharacter Un caractère que, sans guillemets, sépare les mots. un des suivant : **| & ; ( ) < > space tab**

control operator Un token qui exécute une fonction de contrôle. un des suivant : **|| & && ; ; ( ) | |& <newline>**

Mots Réservés

Les mot réservés sont des mots qui ont une signification pour le shell :

! case do done elif else esac fi for function if in select then until while { } time [[ ]]

Commandes simple

Une commande simple est une séquence d'assignement de variable optionnel, suivi par des mots et redirection, terminés par un opérateur de contrôle. Le premier mot spécifie la commande à exécuter, et est passé comme argument 0. Les autres mots sont passés en argument à la commande invoqué. La valeur de retour est sont code de sortie, ou **128+n** si la commande est terminée par le signal **n**

Pipelines

Un pipeline est une séquence d'une ou plusieurs commandes séparés par un des opérateurs de contrôle **|** ou **|&**. Le format est :
[time [-p]] [!] command [[|||&] command2 ...]
La sortie standard de **command** est connectée via un pipe à l'entrée standard de **command2**. Cette connexion est effectuée avant toute redirection spécifiée par le commande. Si **|&** est utilisé, l'erreur standard de **command** est connectée à l'entrée standard de **command2**, est est un raccourci pour **2>&1 |**. Cette redirection implicite de l'erreur standard est effectuée après toute redirection spécifiée par la commande. Chaque commande dans un pipeline est exécuté comme processus séparé.

Le status de retour d'un pipeline est le status de sortie de la dernière commande, sauf si l'option **pipefail** est spécifiée. Si **pipefail** est activé, le status de retour du pipeline est la valeur commande qui se termine avec un status non-zéro, ou 0 si toutes les commandes ont réussit. Si le mot réservé **!** précède un pipeline, de code de sortie de ce pipeline est la négation logique du status de sortie.

Si **time** est spécifié, le temps système, utilisateur et d'exécution sont reportés quand le pipeline se termine. l'option **-p** change le format de sortie pour être conforme POSIX. En mode posix, le mot **time** n'est pas reconnus si le token suivant commence avec un **-**. La variable **TIMEFORMAT** peut être utilisé pour le format de temps. Quand le shell est en mode posix, **time** peut être suivi d'un newline. Dans ce cas, le shell affiche le temps total consommé par le shell et ses enfants.

Listes

Une liste est une séquence d'un ou plusieurs pipelines, séparés par un ou plusieurs opérateurs **;**, **&**, **&&**, ou **||**, et optionnellement terminé par **;**, **&**, ou **<newline>**.De cette liste d'opérateurs, **&&** et **||** ont une précédence égale, suivi par **;** et **&** qui ont une précédence égale. Une séquence d'une ou plusieurs newlines peut apparaître dans une liste au lieu de **;** pour délimiter les commandes.

Si une commande est terminée par l'opérateur de contrôle **&**, le shell exécute la commande dans un sous shell en tâche de fond. Le shell n'attend pas la fin de l'exécution de la commande et retourne le status 0. Les commandes séparées par **;** sont exécutée séquenciellement ; le shell attend que chaque commande se termine. Les listes **AND** et **OR** sont des séquences d'un ou plusieurs pipelines, séparés par les opérateurs de contrôle **&&** et ||, respectivement.

Une liste AND a la forme :
command1 && command2
command2 est exécuté si command1 retourne un status de sortie de 0
Un liste OR a la forme :
command1 || command2
command2 est exécutée si command1 retourne un status de sortie autre que 0

Commandes composées

Une commande composées est une des suivante :

(list) **list** est exécuté dans un environnement sous-shell. Les assignements de variable et le commandes intégrée qui affectent l'environnement du shell n'ont plus d'effet une fois la commande complétée.

{ list ; } **list** est simplement exécuté dans l'environnement du shell courant. **list** doit être terminé avec un newline ou un point-virgule. Le status de retour est le code de sortie de **list**.

((expression)) L'expression est évaluée en accord avec les règles décrites dans **Évaluation arithmétique**. Si la valeur de l'expression est non-zéro, le status de retour et 0 ; sinon 1. C'est exactement équivalent à **let "expression"**

[[expression]] Retourne un status 0 ou 1 en fonction de l'évaluation de l'expression conditionnelle. Les expressions sont composées de primaires, décrites dans **Expressions conditionnelles**. le splitting des mots et l'expansion des path ne sont pas effectués. Utilisé avec **[[**, les opérateurs **<** et **>** trient lexicographiquement en utilisant les locales en cours.

Quand les opérateurs **==** et **!=** sont utilisés, la chaîne à la droite de l'opérateur est considéré être un motif et évalué en accord avec les règles décrites plus bas dans **Pattern Matching**. Si l'option shell **nocasematch** est présent, le match est effectué sans regarder la casse. La valeur de retour et 0 si la chaîne match (**==**) ou ne match pas (**!=**), sinon 1.

Un opérateur additionnel est disponible : **=~**, avec la même précédence. Quand il est utilisé, la chaîne à la droite de l'opérateur est considéré comme expression régulière étendue. la valeur de retour est 0 si elle match, 1 sinon, ou 2 si la syntaxe de l'expression est incorrect. Si l'option **nocasematch** est présente, le match est insensible à la casse. Les sous-chaînes matchés par les sous-expressions sous parenthèses dans l'expression sont sauvée dans la variable tableau **BASH_REMATCH**.

Les expressions peuvent être combinés en utilisant les opérateurs suivant, listés dans l'ordre décroissant de précédence :

(expression) Retourne la valeur de l'expression. Peut être utilisé pour forcer la précédence normal des opérateurs

! expression Vrai si l'expression est fausse

expression1 && expression2 Vrai si les 2 expressions sont vrai

expression1 || expression2 Vrai si une et une seul expression est vrai

Les opérateurs **&&** et || n'évaluent pas expression2 si la valeur de l'expression1 est suffisante pour déterminer la valeur de retour.

for name [[in [word ...]] ;] do list ; done La liste de mots suivant **in** est étendue, générant un liste d'éléments. la variable **name** est définis à chaque élément de cette liste, et **list** est exécuté chaque fois. Si **in** est omis, **for** exécute **liste** une fois pour chaque paramètres positionnels. Le status de retour est le code de sortie de la dernière commande exécutée. Si l'expansion de **list** résulte en une liste vide, aucune commande n'est exécutée et le code de sortie est 0.

for ((expr1 ; expr2 ; expr3)) ; do list ; done **expr1** est évalué. **expr2** est ensuite évaluée en boucle tant qu'elle n'est pas évaluée à 0. À chaque itération de **expr2**, **list** est exécuté et **expr3** est évalué. Si une des expression est manquante, elle est évaluée à 1.

select name [in word] ; do list ; done La liste de mots après **in** est étendue, générant une liste d'éléments. Le jeu étendu est affiché sur l'erreur standard, chacun précédé par un nombre. Si ces mots sont omis, les paramètres positionnels sont affichés. la prompt **PS3** est ensuite affiché avec un line read de l'entrée standard. Si la ligne consiste d'un nombre correspondant à un des mots affiché, la valeur de **name** est définie à ce mot. Si la ligne est vide, les mots et le prompt sont ré-affichés. Si EOF est lu, la commande se termine. Toute autre valeur lue définis **name** à NULL. Le line read est placé dans **REPLY**. La liste est exécutée aprés chaque séléction jusqu'à un **break**

case word in [[(] pattern [| pattern] ...) list ; ;] ... esac Étend les mots, et tente de correspondre avec chaque **pattern**. Quand un match est trouvé, la **list** correspondante est exécutée. Si l'opérateur **; ;** est utilisé, aucun autre match n'est tenté aprés le premier match. Utiliser **;&** à la place force à continuer les matchs.

if list ; then list ; [elif list ; then list ;] ... [else list ;] fi **list** de **if** est exécuté. Si son code de sortie est 0, **list** de **then** est exécuté, sinon **list** de **elif** est exécuté, et si son code de sortie est 0, **list** de **then** est exécuté. Sinon **list** de **else** est exécuté.

while list-1 ; do list-2 ; done Exécute **list-2** tant que la dernière commande de **list-1** retourne un code 0.

until list-1 ; do list-2 ; done Identique à **while** mais le test est inversé (tant que **list-1** ne retourne pas 0)

Coprocesses

Une commande shell précédée par **coproc** est exécutée de manière asynchrone dans un sous-shell, comme si elle avait été terminée par **&**. Le format est

coproc [NAME] command [redirections]

Cela créé un coprocessus nommé **NAME**. Si **NAME** n'est pas fournis, il est nommé **COPROC**. **NAME** ne doit pas être fournie pour une commande simple, sinon il est interprété comme le premier mot d'une commande simple. Quand le coprocessus est exécuté, le shell créé un tableau de variable nommé **NAME** dans le contexte du shell exécutant. La sortie standard de la commande est connectée via un pipe à un descripteur de fichier dans de shell exécutant, et ce descripteur est assigné à **NAME[0]**.

L'entrée standard de la commande est connectée via un pipe à un descripteur de fichier dans le shell exécutant, et ce descripteur est assigné à **NAME[1]**. Ce pipe est établis avant toute redirection spécifiée par la commande. Les descripteurs de fichier peuvent être utilisé comme arguments aux commandes shells est aux redirections en utilisant l'expansion de mots standard. Le process ID du shell est disponible comme valeur de la variable **NAME_PID**. La commande **wait** peut être utilisé pour attendre que le coprocessus se termine. Le status de retour d'un coprocess est le code de sortie de la commande.

Définitions des fonctions shell

Une fonction shell est un objet qui est appelé comme une commande simple est exécute une commande composée avec un nouveau jeu de paramètres positionnels. Les fonctions shell sont déclarées

name () **compound-command [redirection]**

function name [()] **compound-command [redirection]**

Cela définis une fonction nommé **name**. Le mot réservé **function** est optionnel. Si le mot **function** est spécifié, les parenthèses sont optionnels. Le corp de la fonction est la commande composée. Cette commande est généralement une listes de commandes entre **{** est **}**, mais peut être tout type de commande composé. Toute redirection spécifié quand une fonction est définie sont effectués quand la fonction est exécutée.

Le status de sortie d'une définition de fonction est 0 sauf si une erreur de syntaxe de produit ou une fonction lecture seule avec le même nom existe déjà. Une fois exécuté, le code de sortie d'une fonction est le code de la dernière commande exécuté.

Commentaires

Dans un shell non-interactif, ou un shell interactif dans lequel l'option **interactive_comments** de la commande **shopt** est activé, **#** et tout ce qui suit sur la ligne est un commentaire.

Quoting

Le quoting est utilisé pour supprimer la signification spéciale de certains caractères ou mots. Chacun des méta-caractères listé dans **Définitions** a une signification spéciale pour le shell et doit être quoté pour qu'il ne soit pas être interprété par le shell. Quand les expansions d'historique de commande sont utilisées, le caractère d'expansion, généralement **!**, doit être quoté pour empêcher l'expansion de l'historique.

Il y a 3 mécanismes de quoting : le caractère d'échappement, les guillemets simples, et les guillemets double.

Le **** est le caractère d'échappement. Il préserve la valeur littérale du caractère suivant.

Les caractères entre guillemets simples préservent la valeur littérale de chaque caractères.

Les caractères entre guillemets double préservent la valeur littérale de tous les caractères, à l'exception de **$**, **‘**, ****, et, quand l'expansion d'historique est activé, **!**. Les caractères **$** et **‘** conservent leur signification. Le **** conserve sa signification spéciale uniquement lorsqu'il est suivi par **$**, **‘**, **"**, **** ou **<newline**.

Les paramètres spéciaux ***** et **@** ont une signification spéciale dans les guillemets double. Les mots sous la forme **$'string'** sont traités spécialement. L'extension de string, avec les caractères échappés remplacés comme spécifié par le standard ANSI C. Les caractères d'échappement, si présents, sont décodés comme suit :

\a alert (bell)
\b BACKSPACE
\e Un caractère échappe
\e Un caractère échappe
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab
**** backslash
\' single quote
\" double quote
\nnn Le caractère 8bits dont la valeur est nnn en octal
\xHH Le caractère 8bits dont la valeur est HH en en hexadécimal
\uHHHH Le caractère unicode (ISO/IEC 10646) dont la valeur est HHHH en hexadécimal
\UHHHHHHHH Le caractère unicode (ISO/IEC 10646) dont la valeur est HHHHHHHH en hexadécimal
\cx Le caractère control-x

Paramètres

Un paramètre est une entité qui stocke une valeur. Il peut être un nom, un nombre, ou un des caractères spéciaux listés dans **Paramètres Spéciaux**. Une variable est un paramètre dénoté par un nom. Une variable a une valeur et 0 ou plusieurs attributs. Les attributs sont assignés en utilisant la commande **declare**. Un paramètre est définis si une valeur lui a été assignée. La chaîne null est une valeur valide. Une fois une valeur définie, **unset** permet de l'enlever.

Une variable peut être assignée par une déclaration sous la forme :

```
name=[value]
```

Si **value** n'est pas donnée, la variable est assigné à la chaîne null. Toutes les valeurs sont soumises à l'expansion du tilde, expansion de paramètres et de variables, substitution de commande, expansion arithmétique, et suppression de quotes. Si la variable a son attribut **integer**

défini, la valeur est évaluée comme expression arithmétique même si l'expansion **$((...))** n'est pas utilisé. Le word splitting n'est pas effectué, à l'exception de **$@**. l'expansion de chemin n'est pas effectué. La déclaration d'assignement peut aussi apparaître comme argument des commandes **alias**, **declare**, **typeset**, **export**, **readonly**, et **local**.

Dans le context de déclaration d'assignement d'une valeur à une variable shell ou un index de tableau, l'opérateur **+=** peut être utilisé pour attacher ou ajouter à la valeur précédente. Pour une variable entière, La valeur est ajoutée. Pour une variable tableau, les nouvelles valeurs sont assigné au tableau en commençant par l'index maximum + 1 (pour les tableaux indexés), ou ajoutées en paires de clé additionnelles dans un tableau associatif.

Paramètres positionnels

Un paramètre positionnel est un paramètre dénoté par un ou plusieurs chiffres, autre que 0. Ils sont assignés depuis les arguments du shell quand il est invoqué, et peuvent être ré-assignés en utilisant **set**. Les paramètres positionnels ne peuvent pas être assignés avec des déclaration d'assignement. Ils sont temporairement remplacés quand un fonction shell est exécutée. Quand un paramètre positionnel consistant de plus d'un chiffre est étendus, il doit être mis entre accolades.

Paramètres spéciaux

Le shell traite de nombreux paramètres de manière spéciales, ces paramètres peuvent seulement être référencés, leur assignement n'est pas permis.

***** Étend les paramètres positionnels, commençant à 1. Quand l'expansion se produit dans des guillemets doubles, il étend un simple mot avec la valeur de chaque paramètre séparés par le premier caractère de **IFS**, donc **"$*"** est équivalent à **"$1c$2c..."** où c est le premier caractère de **IFS**.

@ Étend les paramètres positionnels, commençant à 1. Quand l'expansion se produit dans des guillemets doubles, Chaque paramètre est étendu à un mot séparé, donc**"$@"** est équivalent à **"$1" "$2" ...**. Si l'expansion double-quoted se produit dans un mot, l'expansion du premier paramètre est join avec le début du mot original, et l'expansion du dernier paramètre est join à la fin du mot original.

Étend le nombre de paramètres positionnels en décimal

? Étend le status de sortie du pipeline le plus récemment exécuté

- Étend les flags d'options courants comme spécifiés à l'invocation, par la commande **set** ou ceux définis par le shell lui-même.

$ Étend le process ID du shell ou du script shell.

! Étend le process ID de la commande la plus récemment exécutée

0 Étend le nom du shell ou du script

_ Au démarrage du shell, définis le chemin absolu utilisé pour invoquer le shell ou le script à exécuter tel que passé dans l'environnement ou la liste d'arguments. Définis également le chemin complet de chaque commande exécutée et placé dans l'environnement exporté à cette commande. En vérifiant les mail, maintient le nom et le fichier mail actuellement vérifié.

Variables Shell

BASH Étend au nom complet utilisé pour invoquer cette instance de bash

BASHOPTS Liste valide d'options pour -s, séparés par de ",". Sont activés avant de lire les fichiers de démarrage

BASHPID Process ID du shell courant

BASH_ALIASES Une variable tableau associatif dont les membres correspondent à la liste interne d'aliase tel que maintenu par la commande alias.

BASH_ARGC Un tableau de variables dont les valeurs sont le nombre de paramètre dans chaque frame de la pile d'appel d'exécution du shell courant. Le nombre de paramètres de la sous-routine courante est en haut de la pile

BASH_ARGV Une variable tableau contenant tous les paramètres dans la pile d'appel du bash courant. Le paramètre final de l'appel de la dernière sous-routine est en haut de la pile.

BASH_CMDS Une variable tableau associatif dont les membres correspondent à la table de hash interne de commandes tel que maintenu par la commande hash.

BASH_COMMAND La commande actuellement exécutée ou à exécuter, sauf si le shell exécute une commande commme résultat d'un trap, dans ce cas c'est la commande exécutée au moment du trap.

BASH_EXECUTION_STRING L'argument de commande pour l'option d'invocation -c

BASH_LINENO Un tableau dont les membres sont les numéros de ligne dans les fichiers sources pour chaque membre de FUNCNAME invoqué. ${BASH_LINENO[$i]} est le numéro de ligne dans le fichier source (${BASH_SOURCE[$i+1]} où ${FUNCNAME[$i]} a été appelé (ou ${BASH_LINENO[$i-1]} si référencé dans une autre fonction.

BASH_REMATCH tableau dont les membres sont assignés par l'opérateur binaire **=~** à la commande **[[**. variable read-only

BASH_SOURCE Tableau dont les membres sont les noms de fichier source où les nom de fonction shell correspondant dans le tableau **FUNCNAME** sont définis. La fonction shell ${FUNCNAME[$i]} est définis dans le fichier ${BASH_SOURCE[$i]} et appelée depuis ${BASH_SOURCE[$i+1]}

BASH_SUBSHELL Incrémenté de 1 chaque fois qu'un sous-shell ou un environnement sous-shell est généré.

BASH_VERSINFO Variable read-only dont les membres maintiennent les informations de version de cette instance de bash. les valeurs assignées sont les suivantes

BASH_VERSINFO[0] Numéro de version majeur

BASH_VERSINFO[1] Numéro de version mineur

BASH_VERSINFO[2] Niveau de patch

BASH_VERSINFO[3] Version de build

BASH_VERSINFO[4] Status de la release

BASH_VERSINFO[5] La valeur de **MACHTYPE**

BASH_VERSION Étend la chaîne décrivant la version de cette instance de bash

COMP_CWORD Un index dans ${COMP_WORDS} du mot contenant la position courante du curseur. disponible uniquement dans les fonctions shell invoqués par les completion programmables

COMP_KEY La touche (ou la touche final d'un séquence de touche) utilisé pour invoquer la fonction de completion courante.

COMP_LINE Ligne de commande courante. Disponible seulement dans les fonctions shell et les commandes externes invoqués par la completion programmable.

COMP_POINT l'index de la position du curseur courant relative au début de la commande courante. Si la position courante est à la fin de la commande courante, la valeur est égale à ${#COMP_LINE}. Disponible seulement dans les fonctions shell et les commandes externes invoqués par la completion programmable.

COMP_TYPE Définis une valeur entière correspondant au type de completion tenté qui a causé la fonction de completion à être appelée : **TAB**, pour une completion normale, **?** pour une completion de listing après des tabs successifs, **!** pour des listing alternatifs sur des completion de mot partiels, **@** pour des completions de liste si le mot n'est pas modifié, ou **%** pour une completion de menu. Disponible seulement dans les fonctions shell et les commandes externes invoqués par la completion programmable.

COMP_WORDBREAKS Le jeu de caractères que readline traite comme séparateur de mots en effectuant la completion de mot.

COMP_WORDS Un tableau consistant de mots individuels dans la ligne de commande courante. La ligne est splité en mots comme le fait readline, en utilisant **COMP_WORDBREAKS** Disponible seulement dans les fonctions shell et les commandes externes invoqués par la completion programmable.

COPROC Tableau créé pour maintenir les descripteurs de fichier pour les sorties et entrée des co-processus non nommés.

DIRSTACK Un tableau contenant le contenu courant de la pile de répertoire. Les répertoires apparaissent dans la pile dans l'ordre qu'ils sont affichés par la commande dirs. pushd et popd doivent être utilisés pour ajouter ou supprimer des répertoires.

EUID Étend l'ID utilisateur effectif de l'utilisateur courant, initialisé au démarrage du shell. variable read-only

FUNCNAME Tableau contenant les noms de toutes les fonctions shell actuellement dans la pile d'appel d'exécution. L'élément d'index 0 est le nom de la fonction actuellement exécutée, l'élément d'index le plus élevé est la fonction main. Cette variable peut être utilisée avec **BASH_LINENO** et **BASH_SOURCE**. Chaque élément de **FUNCNAME** a des éléments correspondant dans ces 2 variables pour décrire la pile. ${FUNCNAME[$i]} a été appelé depuis le fichier ${BASH_SOURCE[$i+1]} à la ligne numéro ${BASH_LINENO[$i]}. La commande caller affiche la pile d'appel courante utilisant cette information.

GROUPS Un tableau contenant la liste de groupes dont l'utilisateur est membre.

HISTCMD Le numéro d'historique, ou index dans la liste d'historique, de la commande courante

HOSTNAME Définis automatiquement au nom de l'hôte courant

HOSTTYPE Définis automatiquement à une chaîne que décris de manière unique le type de machine sur lequel bash est exécuté.

LINENO Chaque fois que ce paramètre est référencé, le shell substitue un nombre décimal représentant le numéro de ligne séquentiel courant dans un script ou fonction. Quand ce n'est pas un script ou une fonction, la valeur substituée n'a pas de signification garantie.

MACHTYPE Définis automatiquement à une chaîne qui décris de type de système sur lequel bash est exécuté dans de format GNU standard.

MAPFILE Un tableau créé pour maintenir le texte lu par la fonction mapfile quand aucun nom de variable n'est fournis

OLDPWD Le répertoire de travail précédent

OPTARG La valeur du dernier argument traité par la commande getopts

OPTIND d'index du prochain argument à traiter par la commande getopts

OSTYPE Définis automatiquement à une chaîne qui décris le système d'exploitation sur lequel bash est exécuté

PIPESTATUS Un tableau contenant une liste de status de sortie des processus dans le pipeline le plus récemment exécuté

PPID Le process ID du shell parent. read-only

PWD Le répertoire de travail courant, définis par cd

RANDOM Chaque fois que ce paramètre est référencé, un entier entre 0 et 32767 est généré. La séquence de nombres aléatoire peut être initialisée en assignant une valeur à RANDOM.

READLINE_LINE Le contenu du tampon de ligne de readline, à utiliser avec bind -x

READLINE_POINT La position du point d'insertion dans le tampon de ligne de readline, à utiliser avec bind -x

REPLY Définis à la ligne de l'entrée lus par la commande read quand aucun argument n'est fournis

SECONDS Chaque fois que ce paramètre est référencé, le nombre de secondes depuis l'invocation du shell est retourné. Si une valeur est assigné à SECONDS, retourne le nombre de seconde depuis l'assignement plus la valeur assigné.

SHELLOPTS Une liste séparée par des virgules d'options du shell. Chaque mot dans la liste est un argument valide pour l'option **-o** de la commande **set**. Cette variable est lu avant tout fichier de démarrage.

SHLVL Incrémenté de 1 chaque fois qu'une instance de bash est lancée

UID Étend l'ID utilisateur, initialisé au démarrage du shell, read-only

Les variables suivantes sont utilisés par le shell, Dans certain cas, bash assigne une valeur par défaut.

BASH_ENV Si ce paramètre est définis quand bash exécute un script, sa valeur est interprété comme nom de fichier contenant des commandes pour initialiser le shell, comme dans ~/.bashrc. Sa valeur est sujet à l'expansion de paramètre, substitution de commande et expansion arithmétique avant d'être interprété comme nom de fichier. PATH n'est pas utilisé pour rechercher le fichier

BASH_XTRACEFD Si définis à un entier correspondant à un descripteur de fichier valide, bash écrit les donnée générées par set -x dans ce descripteur de fichier. Ce descripteur est fermé quand BASH_XTRACEFD est ré-initialisé ou assigné à une autre valeur.

CDPATH Le chemin de recherche pour la commande cd. Liste séparé par des virgules de répertoires dans lequel le shell recherche les répertoires de destination spécifiés par la commande cd. ex **". :~ :/usr"**

COLUMNS Utilisé par select pour déterminer la largeur du terminal en affichant la liste de sélection. Automatiquement définis à la réception de SIGWINCH

COMPREPLY Un tableau dans lequel bash lis les completions possibles générées par un fonction shell invoqué par la completion programmable.

EMACS Si bash trouve cette variable au démarrage et qu'elle commence par "t", il assume que le shell fonctionne comme un shell Emacs et désactive l'édition de ligne.

ENV Similaire à BASH_ENV, utilisé quand le shell est invoqué en mode posix

FCEDIT L'éditeur par défaut pour la commande fc

FIGNORE Une liste séparée par des virgules de suffixes à ignorer lors des completion de nom de fichier. ex : **".o :~"**

FUNCNEST Si définis à une valeur numérique supérieur à 0, définis le niveau maximum de niveau imbriqué.

GLOBIGNORE Une liste séparée par des virgules de patterns définissant le jeu de noms de fichier à ignorer par l'expansion de nom de fichier.

HISTCONTROL Une liste séparée par des virgules de valeur contrôlant la manière dont les commandes sont sauvées dans la liste d'historique. Si **HISTCONTROL** n'est pas définis ou n'inclus pas de valeur valide, toutes les lignes sont sauvegardées dans l'historique, sujet à **HISTIGNORE**. Les lignes d'une commande composée multi-ligne excepté la première ne sont pas testées et sont ajoutés dans l'historique sans regarder la valeur de HISTCONTROL. La liste peut inclure les mots clé suivant :

ignorespace les lignes commençant par un espace ne sont pas sauvés.

ignoredups ne sauve pas les lignes qui matche en entrée précédente dans l'historique.

ignoreboth raccourci pour ignorespace et ignoredups

erasedups supprime toutes les précédente lignes matchant la ligne courant avant de sauver la ligne.

HISTFILE Nom du fichier dans lequel l'historique des commandes est sauvegardé. Défaut : ~/.bash_history. Si non définis, l'historique n'est pas sauvegardé.

HISTFILESIZE Nombre maximum de lignes contenus dans le fichier d'historique.

HISTIGNORE Liste de patterns utilisés pour décider quelles lignes de commandes devraient être sauvés dans l'historique. Chaque pattern doit matcher la ligne complète. **&** match la ligne précédente dans l'historique. Les lignes d'une commande composée multi-ligne excepté la première ne sont pas testées

HISTSIZE Nombre de commandes à mémoriser dans l'historique de commandes. Défaut : 500

HISTTIMEFORMAT Utilisée pour formater les timestamp via strftime(3) dans le fichier d'historique.

HOME Le répertoire home de l'utilisateur courant. argument par défaut de la commande cd

HOSTFILE Contient le nom d'un fichier dans le même format que /etc/hosts qui devrait être utilisé quand le shell doit compléter un nom d'hôte. La liste peut être changé pendant l'exécution du shell. Si non définis, utilise /etc/hosts.

IFS Internal Field Separator utilisé pour séparer les mots après l'expansion et pour séparer les lignes en mots avec la commande read. Défaut : **"<space><tab><newline>"**

IGNOREEOF Contrôle l'action d'un shell interactif à la réception du caractère EOF. Si définis, la valeur est le nombre de caractères EOF consécutifs qui doivent être tapés avant que bash se termine. Défaut : 10 si la variable est définis mais vide. Si elle n'existe pas, EOF signifie la fin de l'entrée du shell.

INPUTRC Nom du fichier de démarrage pour readline. Défaut : ~/.inputrc

LANG Utilisé pour déterminer la catégorie de la locale

LC_ALL Remplace LANG et tout autre variable LC_ spécifiant une catégorie de locale

LC_COLLATE Détermine l'ordre de classement utilisé lors du trie des résultat de l'expansion de nom de fichier, et détermine la méthode de plage d'expression, d'équivalence de classe, et de classement de séquences dans l'expansion de nom de fichier et de recherche de pattern.

LC_CTYPE Détermine l'interprétation des caractères et le mode de classes de caractères dans l'expansion de chemin et de pattern matching

LC_MESSAGES Détermine la locale utilisée pour traduire les chaînes entre guillemets double précédés par un $

LC_NUMERIC Détermine la locale utilisée pour formatter les nombres

LINES Utilisé par select pour déterminer la longueur de colonne pour afficher la liste de sélection. Définis automatiquement à la réception d'un SIGWINCH

MAIL Si définis à un fichier ou un répertoire et que MAILPATH n'est pas définis, bash informe l'utilisateur de l'arrivée de mails dans fichier/répertoire spécifié.

MAILCHECK Spécifie la fréquence en secondes que bash vérifis les mails. le message peut être spécifié en séparane le nom du fichier du message par un "?". quand utilisé dans le texte du message $_ étend au nom du fichier. (ex : **MAILPATH='/var/mail/bfox?"You have mail":~/shell-mail?"$_ has mail!"'**)

OPTERR À 1, bash affiche les messages d'erreur générés par la commande getopts. OPTERR est initialisé à 1 à chaque fois que le shell est invoqué ou qu'un script shell est exécuté

PATH Les chemins de recherche pour les commandes

POSIXLY_CORRECT Si cette variable est dans l'environnement lorsque bash démarre, le shell entre en mode posix avant de lire les fichiers de démarrage.

PROMPT_COMMAND Si définis, la valeur est exécutée comme commande avant de fournir chaque prompt primaire.

PROMPT_DIRTRIM Si définis à un nombre supérieur à 0, la valeur est utilisée comme nombre de composants de répertoire à conserver en étendant la chaîne \w et \W.

PS1 La valeur de ce paramètre est étendu et utilisé comme prompt primaire. Défaut : **\s-\v\$**

PS2 La valeur de ce paramètre est étendu et utilisé comme prompt secondaire. Défaut : **>**

PS3 La valeur de ce paramètre est étendu et utilisé comme prompt pour la commande select

PS4 La valeur de ce paramètre est étendu et utilisé comme PS1 et la valeur est affichée avant chaque commande que bash affiche en mode trace. Le premier caractère est répliqué plusieurs fois si nécessaire, pour indiquer plusieurs niveaux d'indirection. Défaut : **+**

SHELL Le chemin complet du shell est conserver dans cette variable.

TIMEFORMAT La valeur de ce paramètre est utilisé pour spécifier comment les information de temps pour les pipe, préfixeés avec le mot réservé time sont affichés. Défaut : **$'\nreal\t%3lR\nuser\t%3lU\nsys%3lS'**. **%** introduit une séquence échappée.

%% un % littéral

%[p][l]R Le temps passé en secondes

%[p][l]U Le nombres de secondes CPU passé en mode utilisateur

%[p][l]S Le nombres de secondes CPU passé en mode système

%P Le pourcentage de CPU, calculé avec **(%U + %S) / %R**

p spécifie la précision (de 0 à 3). **l** spécifie un format plus long sous la forme **MMmSS.FFs**

TMOUT Si définis à une valeur supérieure à 0, est traité comme timeout par défaut pour les commandes read et select. Dans un shell interactif, la valeur est interprétée comme nombre de secondes à attendre une entrée après avoir fournis le prompt primaire.

TMPDIR Spécifie un répertoire dans lequel bash créé les fichiers temporaires

auto_resume Contrôle comment le shell interagis avec l'utilisateur et le contrôle de job. Si la valeur est définis, une commande simple mot sans redirection est traitée comme candidat à la reprise d'un job stoppé. Il n'y a pas ambiguïté permise, si plus d'un job commence avec la chaîne tapée, le job le plus récent est sélectionné. Le nom d'un job, dans ce contexte, est la ligne de commande utilisée pour la lancer. Si définis à exact, la chaîne fournie doit matcher exactement le nom d'un job stoppé. Si définis à substring, la valeur fournie est analogue à %?. Si définis à une autre valeur, la chaîne fournie doit être préfixeé par le nom d'un job stoppé, analogue à %string.

histchars Les 2 ou 3 caractères qui contrôlent l'expansion d'historique et la tokenisation. Le premier caractère est le caractère l'expansion d'historique (généralement !). Le second est le caractère quick substitution, qui est utilisé comme raccourci pour relancer la commande précédente (généralement ^). Le 3ème caractère, optionnel, est le caractère qui indique que le reste de la ligne est un commentaire quand il est trouvé comme premier caractère d'un mot, généralement #. Il permet à la substitution d'historique de sauter les mots restants sur la ligne.

tableaux

bash fournis des variable tableaux indexés et associatifs à une dimension. Toute variable peut être utilisée comme un tableau indexé ; la commande declare déclare implicitement un tableau. Il n'y a pas de limite sur la taille d'un tableau, ni ne requière que ses membres soient indexés ou assignés en continus. Les tableaux indexés sont référencés en utilisant des entiers (incluant des expressions mathématiques). Les tableaux associatifs sont référencés en utilisant des chaînes arbitraires.

Un tableau indexé est crée automatiquement si une variable est assignée en utilisant la syntaxe **name[subscript]=value**. subscript est traité comme expression arithmétique que doit évaluer un nombre. si subscript évalue un nombre inférieur à 0, il est utilisé comme offset depuis le plus grand élément du tableau (ex : -1 réfère au dernier élément du tableau).

les tableaux indexé peuvent être créés implicitement avec **declare -a**, les tableaux associatifs avec **declare -A**. Les attributs peuvent être spécifiés en utilisant declare et readonly. Chaque attribut s'applique à tout le tableau.

Les tableaux sont assignés en utilisant les assignements composés sous la forme **name=(value1 ... valuen)**, où chaque valeur est sous la forme **[subscript]=string**. En assignant un tableau associatif, le subscript est requis. La syntaxe est accepté par la commande declare. Les éléments individuels peuvent être assignés avec **name[subscript]=value**

Tout élément dans un tableau peut être référencé en utilisant **${name[subscript]}**. Si subscript est @ ou *, le mot s'étend à tous les membres de **name**. Ces subscripts diffèrent seulement quand le mot apparaît entre guillemets double. Si le mot est entre guillemets double, **${name[*]}** étend à un simple mot avec la valeur de chaque membre du tableau, et **${name[@]}** étend chaque élément de **name** à un mot séparé. **${#name[subscript]}** étend à la longueur de **${name[subscript]}**. Si subscript est * ou @, l'expansion est le nombre d'éléments dans le tableau. Référence un tableau sans subscript est équivalent à référencer le tableau avec un subscript de 0.

Un tableau est considéré définis si un subscript à été assigné. Une chaîne null est une valeur valide. **unset** permet de détruire un tableau. **unset name[subscript]** détruit l'élément du tableau. Les commandes **declare**, **local**, et **readonly** acceptent chacun l'option **-a** pour spécifier un tableau indexé, et **-A** pour spécifier un tableau associatif. **read** acceptent **-a** pour assigner une liste de mots lus depuis l'entrée standard dans un tableau. **set** et **declare** affichent les valeurs de tableau de manière à ce qu'ils puissent être réutilisés comme assignement.

Expansion

L'expansion est effectuée sur la ligne de commande après qu'elle ait été splitté en mots. Il y a 7 types d'expansion : expansion d'accolades, expansion de tilde, expansion de paramètre et de variable, La substitution de commande, l'expansion arithmétique, le word splitting, et l'expansion de nom de chemin, dans l'ordre d'expansion. Sur les systèmes qui le supporte, il y a une expansion supplémentaire : la substitution de processus.

Seul l'expansion d'accolades, le word splitting et l'expansion de nom de chemin peuvent changer le nombre de mots de l'expansion ; les autres étendent un simple mot en un simple mot. Les seuls exceptions sont les expansions de **$@** et **${name[@]}**.

Expansion d'accolades

L'expansion l'accolades est un mécanisme par lequel des chaînes arbitraires peuvent être générées. Il est similaire à l'expansion de chemin, mais les noms de fichiers n'ont pas besoin d'exister. Les motifs à étendre prennent la forme d'un préambule optionnel, suivi par soit une série de chaînes séparées par de virgules, soit une séquence d'expressions entre une paire d'accolades, suivi par un postscript optionnel. Le préambule est préfixeé à chaque chaîne, est le postscript est ajouté à chaque chaîne résultante.

Les expansions d'accolades peuvent être imbriquées. Le résultat de chaque chaîne étendue n'est pas trié ; l'ordre de gauche à droite est préservé. Par exemple, **a{d,c,b}e** s'étend à **'ade ace abe**. Une séquence d'expression prend la forme **{x..y[..incr]}**, où x et y sont soit des entiers soit un simple caractère, et incr, un incrément optionnel, un entier. Quand des entiers sont fournis, l'expression étend à chaque nombre entre x et y, inclusif. Les entiers fournis peuvent être préfixeés avec des 0 pour forcer chaque terme à avoir la même largeur. Quand des caractères sont fournis, l'expression étend à chaque caractère lexicographiquement entre x et y, inclusif. x et y doivent être de même type. Quand l'incrément est fournis, il est utilsié comme différence entre chaque terme. l'incrément par défaut est 1 ou -1.

l'expansion d'accolades est effectuée avant tout autre expansion, et tout caractère spécial pour d'autres expansions sont préservés dans le résultat. bash n'applique aucune interprétation syntaxique au contexte de l'expansion ou au texte entre les accolades.

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

Expansion de tilde

Si un mot commence avec un tilde non quoté, tous les caractères précédant le premier "/" non quotés (ou tous les caractères, s'il n'y a aucun / non quoté) sont considérés comme préfixe de tilde. Si aucun des caractères dans le préfixe de tilde ne sont quotés, les caractères dans le préfixe de tilde suivant le tilde sont traités comme un nom de login. Si le nom de login est une chaîne null, le tilde est remplacé avec la valeur du paramètre HOME. Si HOME n'est pas définis, le répertoire de l'utilisateur courant est utilisé. Sinon, le préfixe de tilde est remplacé avec le home associé avec le nom de login spécifié.

Si le préfixe de tilde est **~+**, La valeur de PWD remplace le préfixe de tilde. Si le préfixe de tilde est **~-**, la valeur de OLDPWD, si définis, est substitué. Si le caractère suivant le tilde dans le préfixe de tilde consiste d'un nombre N, optionnellement préfixeé par un **+** ou un **-**, le préfixe est remplacé avec l'élément correspondant depuis la pile de répertoire, comme si c'était affiché par dirs invoqué avec le préfixe de tilde comme argument. Si les caractères suivant le tilde dans le préfixe de tilde consiste d'un nombre sans commencer par un **+** ou **-**, **+** est assumé.

Si le nom de login est invalide, ou l'expansion de tildes échoue, de mot est inchangé. Chaque assignement de variable est vérifié pour des préfixe de tilde suivant immédiatement un **:** ou le premier **=**. Dans ce cas, l'expansion de tilde est aussi effectué. En conséquence, on peut utiliser des noms de fichiers avec des tildes en assignement à PATH, MAILPATH, et CDPATH, et le shell assigne la valeur étendue.

Expansion de paramètres

Le caractère **$** introduit l'expansion de paramètre, la substitution de commande, ou l'expansion arithmétique. le nom de paramètre ou symbole à étendre peut être entre accolades, qui sont optionnels mais servent à protéger la variable de l'extension depuis le caractère immédiatement après, qui peuvent être une partie du nom.

${parameter} La valeur de parameter est substitué. Les accolades sont requises quand parameter est un paramètre positionnel avec plus d'un chiffre, ou quand il est suivi par un caractère qui ne doit pas être interprété comme partie de son nom.

Si le premier caractère de parameter est un **!**, un niveau d'indirection de variable est introduit. bash utilise la valeur de la variable formée depuis le reste de parameter comme nom de variable. Cette variable est ensuite étendue et cette valeur est utilisée dans le reste de la substitution au lieu de la valeur de parameter elle-même. C'est l'expansion d'indirection. Les exceptions sont les expansions de **${!prefix*}** et **${!name[@]}**.

Dans chacun de ces cas, **word** est sujet à l'expansion du tilde, l'expansion de paramètre, la substitution de commande et l'expansion arithmétique.

${parameter :-word} Si parameter est non définis ou null, l'expansion de word est substitué. Sinon, la valeur de parameter est substitué

${parameter :=word} Si parameter est non définis ou null, l'expansion de word est assigné à parameter. la valeur de parameter est ensuite substitué. les paramètres positionnels et paramètres spéciaux ne peuvent pas être assignés de cette manière.

${parameter :?word} Si parameter est non définis ou null, l'expansion de word (ou un message à cet effet si word n'est pas présent) est affiché sur stderr et le shell, s'il n'est pas interactif, quitte. Sinon, la valeur de parameter est substitué.

${parameter :+word} Si parameter est non définis ou null, rien n'est substitué, sinon l'expansion de word est substitué.

${parameter :offset}

${parameter :offset :length} Étend jusqu'à length caractères de parameter, en commençant au caractère spécifié par offset. Si length est omis, étend à la sous-chaîne de paramètre en commençant au caractère spécifié par offset. length et offset sont des expressions arithmétique. Si offset évalue à un nombre inférieur à 0, la valeur est utilisée comme offset depuis la fin de la valeur de paramètre. Les expressions arithmétique commenfant avec un - doivent être séparés par un espace blanc du précédent.

Si length évalue à un nombre inférieur à 0, et parameter n'est pas @ ni un tableau, il est interprété comme offset depuis la fin de la valeur de parameter. Si parameter est un nom de tableau indexé, le résultat sont les length membres du tableau commençant avec **${parameter[offset]}**. Si offset est 0 et que les paramètres positionnels sont utilisés, $0 est préfixé à la liste.

${!prefix*}

${!prefix@} Étends aux noms de variables dont les noms commencent avec prefix, séparés par le premier caractère de IFS. Quand @ est utilisé et que l'expansion apparaît dans des guillemets doubles, chaque nom de variable s'étends à un mot séparé.

${!name[@]}

${!name[*]} Si name est un tableau, étends à la liste d'indices assignés dans name. Si name n'est pas un tableau, étends à 0 si name est définis et null sinon. Quand @ est utilisé et que l'expansion apparaît dans des guillemets doubles, chaque indice d'étends à un mot séparé

${#parameter} Subsitute à la longueur en caractères de la valeur de parameter. Si parameter est * ou @, la valeur substitué est le nombre de paramètres positionnel. Si parameter est un tableau, la valeur substitué est le nombres d'éléments dans le tableau

${parameter#word}

${parameter##word} word est étendu pour produire un pattern comme dans une expansion de pathname. Is le pattern match le début de la valeur de parameter, le résultat est la valeur de parameter raccourci du pattern matchant le plus court (#), ou le plus long (##). Si parameter est @ ou * , l'opération de suppression est appliqué à chaque paramètre positionnel, et l'expansion est la liste résultante. Si parameter est un tableau, l'opération est appliquée à chaque membre du tableau.

${parameter%word}

${parameter%%word} L'expansion est identique mais l'opération de suppression de pattern se produit en partant de la fin de parameter.

${parameter/pattern/string} pattern est étendu pour produire un un pattern comme dans l'expansion de pathname. parameter est étendu et le plus grand match de pattern est remplacé avec string. Si pattern commence avec /, tous les matches de pattern sont remplacés avec string. Normalement, seul le premier match est remplacé. Si pattern commence avec #, il doit matcher au début de la valeur étendu de paramètre, et à la fin si pattern commence avec %. Si string est null, les matchs sont supprimés. Si parameter est @ ou * , l'opération de substitution est appliquée à chaque paramètre positionnel. Si parameter est un tableau, la substitution est appliquée à chaque membre.

${parameter^pattern}

${parameter^^pattern}

${parameter,pattern}

${parameter,,pattern} Cette expansion modifie la casse des caractères alphabétique dans parameter. **^** convertis en majuscule le premier pattern qui match (^^tous les patterns qui matchent), **,** convertis en minuscule.

Substitution de commande

La substitution de commande permet à la sortie d'un commande de remplace de nom de commande. Il y a 2 forme

$(command)

‘command‘

Bash effectue l'expansion en exécutant **command** et en remplaçant la substitution avec la sortie standard de la commande. La substitution de commande **$(cat file)** peut être remplacé par l'équivalent plus rapide **$(< file)**. Quand l'ancien style **‘** est utilisé, le backslash garde sa signification litérale excepté quand il est suivi par **$**, **‘** ou ****. Le premier **‘** non précédé par un **** termine la substitution de commande. En utilisant la forme **$(command)**, tous les caractères entre les parenthèses font la commande, aucun n'est traité spécialement.

Les substitutions de commande peuvent être imbriqués. Pour imbriquer en utilisant la forme **“**, échapper les **‘** à l'intérieur avec des ****. Si la substitution de commande apparaît dans des guillemets doubles, le word splitting et l'expansion de pathname ne sont pas effectués.

Expansion arithmétique

L'expansion arithmétique permet l'évaluation d'une expression arithmétique et la substitution du résultat. Le format est :

$((expression))

L'expression est traitée comme si elle était entre guillemets double, mais un guillemet double dans les parenthèses ne sont pas traités spécialement. tous les tokens dans l'expression subissent l'expansion de paramètre, l'expansion de chaîne, la substitution de commande, et la suppression de quote. Les expressions arithmétique peuvent être imbriqués. L'évaluation est effectué en accord avec les règles listé dans **Évaluation arithmétique**.

Substitution de processus

La substitution est supportée sur les systèmes qui supportent les pipes nommés (FIFO) ou la méthode de fichiers **/dev/fd**. Elle prend la forme **<(list)** ou **>(list)**. list avec sa sortie ou sa sortie connecté à un FIFO. Le nom de ce fichier est passé en argument à la commande courante en résultat de l'expansion. En utilisant **>(list)**, écrire dans le fichier fourni une entrée pour list. En utilisant **>(list)**, le fichier passé en argument devrait être lu pour obtenir la sortie de list. Quand disponible, la substitution est effectué simultanément avec l'expansion de paramètre et de variable, la substitution de commande et l'expansion arithmétique.

Découpage de mots

Le shell traite chaque caractère de IFS comme délimiteur, et découpe le résultat d'autre expansions en mots. Si IFS n'est pas définis, sa valeur est exactement **<space><tab><newline>**, et les séquences **<space>**, **<tab>** et **<newline>** au début et à la fin du résultat de l'expansion précédente sont ignorés, et toute autre séquence de caractère de IFS sert à délimiter les mots. Si IFS a une valeur autre que son défaut, alors les séquences **space** et **tab** sont ignorés au début et à la fin du mot, tant que le caractère whitespace est dans la valeur d'IFS.

Tout caractère dans IFS qui n'est pas un whitespace, délimite un champ. Une séquence de whitspace d'IFS est aussi traité comme un délimiteur. Si la valeur d'IFS est null, aucun découpage ne se produit.

Les arguments null explicites **""** et **''** sont retenus. Les arguments null implicite non quotés, résultant de l'expansion de paramètre qui n'a pas de valeur, sont supprimés. Si un paramètre sans valeur est étendu dans des guillemets double, il en résulte un argument null et est retenus. Noter que si aucune expansion ne se produit, aucun découpage n'est effectué.

Expansion de chemin

Après le word splitting, à moins que l'option **-f** n'ait été définis, bash scanne chaque mot pour le caractère *****, **?**, et **[**. Si un de ces caractères apparaît, alors de mot est lu comme pattern, et remplacé avec une liste triée alphabétiquement de nom de fichiers correspondant au pattern. Si aucun nom de fichier matchant n'est trouvé et que l'option **nullgrob** n'est pas activé, le mot reste inchangé. Si **nullglob** est définis, et qu'aucun match n'est trouvé, le mot est supprimé. Si l'option shell **failglob** est définis et qu'aucun matche n'est trouvé, un message d'erreur est affiché et la commande n'est pas exécutée. Si l'option **nocaseglob** est activée, le match est effectué sans regarder la casse des caractères.

Noter qu'en utilisant des expression de plage comme [a-z], les lettres de l'autre casse peuvent être incluent en fonction de **LC_COLLATE**. Quand un pattern est utilisé pour l'expansion de pathname, le caractère **.** au début d'un nom ou suivant immédiatement un **/** doit être matché explicitement sauf si l'option **dotglob** est définis. En matchant un pathname, le **/** doit toujours être matché explicitement. Dans les autres cas, **.** n'est pas traité spécialement.

La variable **GLOBIGNORE** peut être utilisé pour restreindre le jeu de noms de fichier matchant un pattern. Si elle est définie, chaque nom de fichier matchant qui match également les patterns dans GLOBIGNORE sont supprimés de la liste des matchs. **GLOBIGNORE** définis à une valeur null active l'option **dotglob**, donc tous les noms de fichiers commençant par **.** vont matcher. pour ignorer les noms de fichier commençant par un **.**, créer un pattern **.*** dans **GLOBIGNORE**, l'option **dotglob** est désactivé quand **GLOBIGNORE** est définis.

Motifs de correspondance

Tout caractère que apparaît dans un pattern, autre que les caractères de pattern spéciaux, se matche lui-même. Le caractère NUL ne peut pas exister dans un pattern. Un '\' échappe les caractère suivant. Les caractères de pattern doivent être quotés s'ils doivent matcher littéralement. Les caractères de pattern spéciaux ont la signification suivante :

***** Match toutes chaîne, incluant une chaîne null. Quand l'option **globstar** est activée, et que ***** est utilisé dans l'expansion de pathname, 2 ***** adjacent utilisés dans un simple pattern vont matcher tous les fichiers et ou plusieurs répertoire et sous-répertoires. Si suivi par un **/**, 2 ***** adjacents vont matcher uniquement les répertoires et sous-répertoires.

? Matche un simple caractère.

[...] Matche un des caractères entre les crochets. Une paire de caractères séparés par un "-" dénote une plage ; tous caractères qui se trouvent entre ces 2 caractères sont matchés. Si le premier caractère après **[** est un **!** ou un **^** alors tous caractères qui n'est pas dans les crochets matchent. L'ordre de trie des caractères dans une plage est déterminé par la locale courant et la valeur de **LC_COLLATE**, si définis. Un **-** peut être matché en l'incluant comme premier ou dernier caractère dans les crochets. Un **]** peut être matché en l'incluant comme premier caractère.

- Dans [et], les classes de caractères peuvent être spécifiés en utilisant la syntaxe **[:class :]**, où class peut être **alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit**
- Dans [et], une équivalence de classe peut être spécifiée en utilisant la syntaxe **[=c=]**, qui matche tous les caractères de la même classe que le caractère c.
- Dans [et], la syntaxe **[.symbol.]** matche la même classe de symbole.

Si **extglob** est activé, de nombreux opérateurs de correspondance de pattern étendus sont reconnus. Dans les descriptions suivante, pattern-list est une liste d'un ou plusieurs pattern séparés par un |. Les patterns composites peuvent être formés en utilisant un ou plusieurs sous-pattern :

?(pattern-list) Matche 0 ou une occurence des patterns donnés

***(pattern-list)** Matche 0 ou plusieurs occurences des patterns donnés

+(pattern-list) Matche 1 ou plusieurs occurences des patterns donnés

@(pattern-list) Matche 1 des patterns donnés

!(pattern-list) Matche tout sauf un des patterns donnés

Suppression de quote

Après les précédentes expansions, toutes les occurences non-quotés des caractères ****, **'** et **"** qui ne résultent pas d'une des précédentes expansions sont supprimés.

Redirection

Avant qu'une commande soit exécutée, son entrée et sa sortie peuvent être redirigées en utilisant une notation interprétée par le shell. La redirection peut être également utilisée pour ouvrir et fermer des fichiers pour l'environnement d'exécution du shell courant. Les redirections sont traitées dans l'ordre qu'elles apparaissent.

Chaque redirection qui peut être précédée par un descripteur de fichier peut, à la place, être précédée par un mot sous la forme **{varname}**. Dans ce cas, pour chaque opérateur de redirection excepté **>&-** et **<&-**, le shell va allouer un descripteur de fichier supérieur à 10 et l'assigner à varname. Si **>&-** ou **<&-** est précédé par **{varname}**, la valeur de varname définis le descripteur de fichier à fermer.

Dans les descriptions suivantes, si le descripteur de fichier est omis, et que le premier caractère de l'opérateur de redirection est **<**, la redirection réfère à l'entrée standard (descripteur de fichier 0). Si le premier caractère de l'opérateur de redirection est **>**, la redirection réfère à la sortie standard (descripteur de fichier 1).

Le mot suivant l'opérateur de redirection dans les descriptions suivante, sauf mention, est sujet à l'expansion d'accolades, de tilde, et de word splitting. S'il s'étend à plus d'un mot, bash reporte une erreur. Noter que l'ordre des redirections est significatif.

par exemple, la commande
ls > dirlist 2>&1
Dirige la sortie standard et l'erreur standard vers le fichier dirlist, alors que la commande
ls 2>&1 > dirlist
dirige seulement la sortie standard vers de ficher dirlist, parce que l'erreur standard a été dupliquée depuis la sortie standard avant que la sortie standard ai été redirigée vers dirlist.
bash gère de nombreux noms de fichiers spécialement quand ils sont utilisés dans les redirections :

/dev/fd/fd Si fd est un entier valide, le descripteur fd set dupliqué

/dev/stdin Le descripteur de fichier 0 est dupliqué

/dev/stdout Le descripteur de fichier 1 est dupliqué

/dev/stderr Le descripteur de fichier 2 est dupliqué

/dev/tcp/host/port Si host est un nom d'hôte valide ou une adresse internet, et port est un numéro de port ou un nom de service, bash tente d'ouvrir une connexion TCP sur le socket correspondant.

/dev/udp/host/port Si host est un nom d'hôte valide ou une adresse internet, et port est un numéro de port ou un nom de service, bash tente d'ouvrir une connexion UDP sur le socket correspondant.

Une impossibilité d'ouvrir ou créer un fichier échoue la redirection. Les redirections utilisant les descripteurs de fichier supérieur à 9 devraient être utilisés avec précaution vu qu'ils peuvent être en conflit avec les descripteur de fichiers que le shell utilise en interne. Noter que la commande **exec** peut créer des redirections qui prennent effet dans le shell courant.

Redirection d'entrée

La redirection d'entrée ouvre le fichier résultant de word en lecture sur le descripteur de fichier n, ou l'entrée standard si non spécifié. Le format général est :

[n]<word

Redirection de sortie

La redirection de sortie ouvre le fichier résultant de word en écriture sur le descripteur de fichier n, ou la sortie standard si non spécifié. Si le fichier n'existe pas il est créé ; s'il existe ta taille est tronqué à 0. Le format général est :

[n]>word
Si l'opérateur de redirection est **>**, et l'option **noclobber** est définie, la redirection échoue si le fichier dont le nom résulte de l'expansion de word existe et est un fichier régulier. Si l'opérateur de redirection est **>|**, ou **>** et l'option **noclobber** n'est pas activé, la redirection est tentée même si le fichier nommé par word existe.

Ajouter à la sortie redirigée

La redirection de la sortie de cette manière ouvre le fichier résultant de l'expansion de word à être ouvert sur le descripteur de fichier n pour ajout. Si le fichier n'existe pas il est créé. Le format général est :

[n]»word

Rediriger Stdout et Stderr

Cette constuction permet à la sortie standard et l'erreur standard d'être redirigés vers de fichier dont le nom est l'expansion de word. Il y a 2 formes :

&>word
et
>&word
Des 2 formes, la première est préféré. C'est équivant à
>word 2>&1

Ajouter Stdout et Stderr

Cette construction permet à la sortie standard est l'erreur standard d'être ajoutés au fichier dont le nom est l'expansion de word.

&»word
C'est équivalent à
»word 2>&1

Here Documents

Ce type de redirection instruit de shell de lire l'entrée depuis la source courante jusqu'à ce qu'une ligne contenant seulement delimiter (sans espace blanc après) soit vu. Toutes les lignes lues jusqu'à ce point sont utilisés comme entrée standard pour une commande.

«[-]word
here-document
delimiter
Aucune expansion de paramètre, substitution de commande, expansion arithmétique, expansion de chemin n'est effectué dans word. Si des caractères dans word sont quotés, le delimiter est le résultat de la suppression des quote dans word et les lignes dans le here-document ne sont pas étendues. Si word est non quoté, toutes les ligne du here-document sont sujet à l'expansion de paramètre, substitution de commande et expansion arithmétique. Dans ce dernier cas, la séquence **\<newline>** est ignorée, et **** doit être utilisé pour quoter les caractères ****, **$** et **‘**. Si l'opérateur de redirection est **«-**, tous les caractères de tabulation en début de ligne sont supprimés.

Here String

Une variante de here-document. word est étendu et fournis à la commande sur son entrée standard. le format est :

«<word

Dupliquer les descripteurs de fichier

L'opérateur de redirection

[n]<&word

est utilisé pour dupliquer les descripteurs de fichier d'entrée. Si word s'étend à un ou plusieurs chiffres, le descripteur de fichier dénoté par n devient une copie de ce descripteur de fichier. Si word évalue à **-**, le descripteur n est fermé. Si n n'est pas spécifié, l'entrée standard est utilisée.

[n]>&word

est utilisé pour dupliquer les descripteurs de fichier de sortie. Si n n'est pas spécifié, la sortie standard est utilisée. Un cas spécial, si n est omis, et word ne s'étend pas à un ou plusieurs chiffres, l'entrée standard et l'erreur standard sont redirigés comme décris précédemment.

Déplacer les descripteurs de fichier

L'opérateur de redirection

[n]<&digit-

déplace le descripteur de fichier digit au descripteur de fichier n, ou l'entrée standard si n n'est pas spécifié. digit est fermé après avoir été dupliqué à n. Similairement, l'opérateur de redirection

[n]>&digit-

déplace le descripteur de fichier digit au descripteur de fichier n, ou la sortie standard si n n'est pas spécifié.

Ouvrir des descripteurs de fichier en lecture-écriture

L'opérateur de redirection

[n]<>word

ouvre le fichier en lecture et écriture sur le descripteur de fichier n ou le descripteur de fichier 0 si n n'est pas spécifié. Si le fichier n'existe pas, il est créé.

Alias

Les alias permettent à une chaîne d'être substituée pour un mot quand il est utilisé comme premier mot d'une simple commande. Le shell maintient une liste d'alias qui peuvent être définis et supprimés avec les commande alias et unalias. Le premier mot de chaque commande simple, si non quoté, est vérifié pour voir si elle a un alias. Si c'est le cas, ce mot est remplacé par le texte de l'alias. Les caractères **/**, **$**, **‘**, et **=** et tout autre métacaractère shell ou caractère de quote ne doivent pas apparaître dans un nom d'alias.

Le texte de remplacement peut contenir une entrée shell valide, incluant des métacaractères. Le premier mot du remplacement est testé pour un alias, mais un mot qui est identique à un alias qui est étendu n'est pas étendu une seconde fois. Cela signifie que l'on peut créer un alias **ls** à **ls -F**. Si le dernier caractère de la valeur de l'alias est un blanc, alors le prochain mot suivant l'alias est aussi vérifié pour l'expansion de l'alias.

Il n'y a pas de mécanisme pour utiliser des arguments dans le texte de remplacement. Si les arguments sont nécessaires, une fonction shell devrait être utilisée. Les alias ne sont pas étendus quand le shell n'est pas interactif, sauf si l'option **expand_aliases** est définis.

Bash lit toujours au moins une ligne complète d'entrée avant d'exécuter les commandes sur la ligne. Les alias sont étendus quand une commande est lue, pas quand elle est exécutée, Cependant, une définition d'alias apparaissant sur la même ligne qu'une autre commande ne prend effet qu'à la ligne suivante, Le problème est similaire avec les fonctions.

Fonctions

Une fonction shell stocke une série de commande pour une exécution ultérieur. Quand le nom d'une fonction est utilisé comme nom de commande simple, la liste des commandes associée avec cette fonction est exécutée. Les fonctions sont exécutées dans le contexte du shell courant ; aucun nouveau processus n'est créé pour les interpréter. Quand une fonction est exécutée, les arguments de cette fonction deviennent les paramètres positionnels durant son exécution. Le paramètre spécial **#** est mis à jour pour refléter ce changement et **0** est inchangé. Le premier élément de **FUNCNAME** est définis au nom de la fonction durant son exécution.

Tous les autres aspects de l'environnement d'exécution du shell sont identiques entre une fonction et son appelant à l'exception des traps **DEBUG** et **RETURN** qui ne sont pas hérités sauf si la fonction a été donnée par l'attribut **trace** ou l'option **-o functrace** est activée. Le trap **ERR** n'est pas hérité sauf si **-o errtrace** est activé.

- Les variables locales aux fonctions peuvent être déclarées avec la commande **local**. Généralement, les variables et leur valeurs sont partagées entre la fonction et son appelant.
- La variable **FUNCNEST**, si définie à une valeur numérique supérieur à 0, définis le niveau d'imbrication de fonction maximum. Les invocations de fonction qui excèdent cette limite sont annulées.
- Si la commande **return** est exécutée dans une fonction, la fonction se termine et retourne à l'appelant. Toute commande associée avant le trap **RETURN** est exécutée avant que l'exécution se termine. Quand une fonction se termine, les valeurs des paramètres positionels et le paramètre spécial **#** sont restaurés aux valeurs qu'ils avaient avant l'exécution de la fonction.
- Les fonctions peuvent être récursives, La variable **FUNCNEST** peut être utilisée pour limiter la profondeur d'appel et restreindre le nombre d'invocation.

Les noms et définitions de fonction peuvent être listés avec l'option **-f** des commandes declare ou typeset. L'option **-F** de declare ou typeset va lister les noms de fonction uniquement (et optionnellement le fichier source et le numéro de ligne, si **extdebug** est activé). Les fonctions peuvent être exportées pour que les sous-shells les aient automatiquement definies avec l'option -f de la commande export. Une définition de fonction peut être supprimée avec unset -f. Noter que les fonctions shell et le variables de même nom peuvent résulter en plusieurs entrées nommées identiquement dans l'environnement passé aux enfants du shell, cela peut poser problème.

Évaluation Arithmétique

Le shell permet d'évaluer des expressions arithmétiques, sous certaines circonstances. L'évaluation est faite avec des entiers à largeur fixe sans vérification de débordement. Les opérateurs et leur précédence, l'associativité et les valeurs sont les même que dans le langage C. La liste suivant d'opérateurs est groupé en niveaux de précédence équivalant. Les niveaux sont listés dans l'ordre décroissant de précédence.

id++ id– post-incrément et post-décrément de variable

++id –id pré-incrément pré-décrément de variable

- + moins et plus unaire

! ~ négation logique et au niveau des bits

****** Exponentialité

*** / %** multiplication, division et reste

+ - addition et soustraction

« » décalages de bit

<= >= < > Comparaison

== != égalité et inégalité

& ET bit à bit

^ OU exclusif bit à bit

| OU bit à bit

&& AND logique

|| OU logique

expr ?expr :expr opérateur conditionnel

= *= /= %= += -= «= »= &= ^= |= assignements

expr1 , expr2 virgule

Les variables shell sont des opérandes permises ; l'expansion de paramètre est effectué avant que l'expression soit évalué. Dans l'expression, les variables shell peuvent aussi être référencées par mon sans utiliser la syntaxe d'expansion de paramètre. Une variable shell qui est null ou indéfinie évalue à 0 quand elle référencée par nom sans utiliser la syntaxe d'expansion de paramètre. La valeur d'une variable est évaluée comme expression arithmétique quand elle est référencée, ou quand une variable a l'attribut entier. Une valeur null évalue à 0. Une variable shell n'a pas besoin de l'attribut entier pour être utilisé dans une expression

Les constantes commençant avec un 0 sont interprétées en octal, en celles commençant avec 0x ou 0X sont interprétés en héxadécimal. Sinon, les nombres prennent la forme **[base#]n**, où base est entre 2 et 64 et représente la base arithmétique, et n est un nombre de cette base. Si base est omis, utilise de système décimal. En spécifiant n, les chiffres supérieur à 9 sont représenté par les lettres minuscule, les lettres majuscules, @, et _, dans cet ordre. Si base est inférieur ou égal à 36, les lettres minuscules et majuscule peuvent être inter-changés. Les opérateurs sont évalués dans l'ordre de précédence. Les sous-expressions entre parenthèses sont évalués en premier et peuvent écraser les règles de précédence.

Expressions conditionnelles

Les expressions conditionnelles sont utilisée par la commande composée **[[** et les commandes **test** et **[** pour tester les attributs de fichier et effectuer des comparaisons arithmétique et de chaîne. Les expressions sont formées depuis les primaires unaire ou binaire suivantes. Si un argument **file** d'un des primaires est de forme **/dev/fd/n**, le descripteur de fichier n est vérifié. Sauf mention, les primaires qui opèrent sur les fichiers suivent les liens symbolique. Quand utilisé avec **[[**, les opérateurs **<** et **>** trient lexicographiquement en utilisant la locale courante. La commande **test** trie en utilisant l'ordonnancement ASCII.

-a file Vrai si file existe

-b file Vrai si file existe et est un fichier spécial block

-c file Vrai si file existe et est un fichier spécial caractère

-d FILE vrai si FILE existe et est un répertoire

-e file Vrai si file existe

-f FILE vrai si FILE existe et est un fichier régulier

-g file Vrai si file existe et est set-groud-id

-h file Vrai si file existe et est un lien symbolique

-k file Vrai si file existe et son sticky bit est mis

-p file Vrai si file existe et est un pipe nommé

-r file Vrai si file existe et est accéssible en lecture

-s FILE vrai si FILE existe et a une taille supérieur à 0

-t fd Vrai si le descripteur de fichier est ouvert et réfère à un terminal

-u file Vrai si file existe et son set-user-id est mis

-w file Vrai si file existe et est accéssible en écriture

-x file Vrai si file existe et est exécutable

-G file Vrai si file existe et est possédé par l'id de groupe effectif

-L file Vrai si file existe et est un lien symbolique

-N file Vrai si file existe et a été modifié depuis le dernier accès en lecture

-O file Vrai si file existe et est possédé par l'id utilisateur effectif

-S FILE vrai si FILE existe et est un socket

file1 -ef file2 Vrai si file1 et file2 réfèrent au même numéro de device et inode

file1 -nt file2 Vrai si file1 est plus récent que file2, ou si file1 existe et non file2

file1 -ot file2 Vrai si file1 est plus ancien que file2, ou si file2 exist en non file1

-o optname Vrai si l'option shell est activée.

-v varname Vrai si la variable shell est définie

-R varname Vrai si la variable shell est définie et est une référence nommée

-Z STRING vrai si la longueur de STRING est 0

string

-n string Vrai si la longueur de string est différente de 0

string1 == string2

string1 = string2 Vrai si les chaînes sont identiques. = devrait être utilisé avec test.

string1 != string2 Vrai si les chaînes ne sont pas identiques

string1 < string2 Vrai si string1 se trie avant string2 lexicographiquement

string1 > string2 Vrai di string1 se trie après string2 lexicographiquement

arg1 OP arg2 OP vaut : **-eq, -ne, -lt, -le, -gt,** ou **-ge**. Ces opérateurs arithmétique retournent vrai si arg1 est égal à, non égal à, inférieur à, inférieur ou égal à, supérieur à, supérieur ou égal à arg2, respectivement.

Expansion de commande simple

Quand une commande simple est exécutée, le shell effectue les expansions, assignements et redirections suivantes, de gauche à droite :

1. Les mots que le parser a marqué comme assignement de variable (ceux précédant le nom de commande) et des redirections sont sauvés pour traitement ultérieur.
2. Les mots qui ne sont pas des assignement de variable ou des redirections sont étendus. Si des mots restent après l'expansion, le premier mot est pris comme nom de commande et les mots restants sont les arguments.
3. Les redirections sont effectuées
4. Le texte après le = dans chaque assignement de variable subit d'expansion de tilde, expansion de paramètres, substitution de commande, expansion arithmétique, et suppression de quote avant l'être assigné à la variable.

S'il ne résulte aucun nom de commande, les assignements de variable affectent le shell courant. Sinon, les variables sont ajoutés à l'environnement de la commande exécutée et n'affecte pas l'environnement du shell courant.

S'il ne résulte aucun nom de commande, les redirections sont effectuées, mais n'affectent pas l'environnement du shell courant.

Exécution de commande

Une fois une commande découpée en mots, s'il en résulte en une simple commande et une liste optionnelle d'arguments, les actions suivantes sont effectuées.

Si le nom de la commande ne contient pas de **/**, le shell tente de la localiser. S'il existe une fonction shell de ce nom, elle est invoquée. Sinon, si le nom correspond à une commande intégrée, elle est invoquée.

Si le nom n'est ni une fonction ni un commande intégrée et ne contient pas de **/**, bash recherche chaque élément de **PATH** à la recherche d'un fichier exécutable de ce nom. Bash utilise une table de hash pour mémoriser les chemins complet des fichiers exécutable. Une recherche complète des répertoires n'est effectuée que si la commande n'est pas trouvée dans la table. Si la recherche ne donne aucun résultat, le shell recherche une fonction shell définie nommée **command_not_found_handle**. Si elle existe, elle est invoquée avec la commande original et ses arguments en argument, et le code de sortie de la fonction devient le code de sortie du shell. Si cette fonction n'est pas définie, le shell affiche une erreur et retourne un code de sortie de 127.

Si la recherche a réussie, ou si le nom de commande contient un ou plusieurs **/**, le shell exécute le programme nommé dans un environnement séparé. L'argument 0 est le nom donnée.

Si le programme est un fichier commençant avec **#!**, le reste de la première ligne spécifie un interpréteur de commande. Le shell exécute l'interpéteur spécifié.

Environnement d'exécution de commande

Le shell a un environnement d'exécution, qui consiste en :

- Les fichier ouverts hérités par le shell à l'invocation, comme modifié par les redirections fournies à la commande **exec**
- Le répertoire de travail courant comme définis par **cd**, **pushd** ou **popd**
- Le masque de mode de création des fichiers comme définis par **umask** ou hérité du shell parent
- Les traps courant définis par **trap**
- Les paramètres shell qui sont définis par l'assignement de variable ou avec **set** ou hérités du shell parent
- Les fonctions shell définies durant l'exécution ou hérités du shell parent
- Les options activées à l'invocation ou définis pas **set**
- les options activée avec **shopt**
- les alias définis avec **alia**
- Divers process ID, incluant ceux des tâche de fond, la valeur de **$$**, et de **PPID**

Quand une commande simple autre qu'un commande intégrée ou une fonction shell est exécutée, elle est invoquée dans un environnement d'exécution séparé qui consiste en :

- Les fichier ouverts du shell, plus toute modification et ajout spécifié par les redirections à la commande
- Le répertoire de travail courant
- Le masque de mode de création de fichier
- Les variables et fonctions shell marqué pour l'export, et les variables exportés pour la commande, passés dans l'environnement

- Les traps capturés par le shell sont réinitialisés aux valeurs hérités du shell parent, et les traps ignorés par le shell sont ignorés.

Une commande invoquée dans un environnement séparés ne peut pas affecter l'environnement d'exécution du shell. La substitution de commande, les commandes groupées avec des parenthèses et les commande asynchrone sont invoquées dans un sous-shell qui est dupliqué depuis l'environnement du shell, excepté que les traps capturés par le shell sont réinitialisés aux valeur que le shell a hérité de son parent à l'invocation. Les commande intégrées qui sont invoquées comme partie d'un pipeline sont aussi exécutés dans un sous-shell. Les changements fait au sous-shell ne peuvent pas affecter l'environnement d'exécution du shell.

Les sous-shell générés pour exécuter des substitutions de commande héritent de la valeur de l'option **-e** du shell parent. Quand il n'est pas en mode posix, bash efface l'option **-e** dans de tels sous-shell. Si une commande est suivie par un **&** et que le contrôle de job n'est pas actif, l'entrée standard par défaut pour la commande est **/dev/null**. Sinon, la commande hérite des descripteurs de fichier du shell appelant comme modifié par les redirections.

Environnement

Quand un programme est invoqué il lui est donné un tableau de chaînes appelé l'environnement. C'est une liste de paire nom=valeur. Le shell fournis de nombreuses manières de manipuler l'environnement. À l'invocation, le shell scanne son propre environnement et créé un paramètre pour chaque nom trouvé, en le marquant automatiquement pour export à ses processus enfant. Les commandes exécutées héritent de l'environnement. Les commandes export et declare -x permettent d'ajouter et de supprimer des fonctions et des variables à l'environnement.L'environnement pour une commande simple ou une fonction peut être augmentée temporairement en la préfixant avec des assignements de paramètre. Ces assignements affectent uniquement l'environnement vu par la commande.

Si l'option -k est définie, tous les assignements de paramètre sont placés dans l'environnement pour une commande, pas seulement ceux qui précèdent le nom de la commande. Quand bash invoque une commande externe, la variable _ est définie au fichier de la commande et passée à cette commande dans son environnement.

Codes de sortie

Le statut de sortie d'une commande exécutée est la valeur retournée par l'appel système **waitpid** ou une fonction équivalente. Les statuts de sortie vont de 0 à 255, bien que le shell utilise les valeurs supérieur à 125 de manière spéciales. Les statuts de sortie des commandes intégrées et des commandes composées sont également limitées à cette plage. Dans certaines circonstances, le shell va utiliser des valeur spéciales pour indiquer des modes d'échec spécifiques.

Une commande qui se termine avec un code de sortie de 0 à réussie. Un autre code indique une erreur. Quand une commande se termine sur un signal fatal N, bash utilise la valeur de 128+N comme code de sortie. Si une commande n'est pas trouvée, le processus enfant créé pour l'exécuter retourne un statut de 127. Si une commande est trouvée mais n'est pas exécutable, le code de retour est 126. Si une commande échoue à cause d'une erreur durant l'expansion ou la redirection, de code de sortie est supérieur à 0. Les commandes intégrées retournent 0 si elles réussissent. Toutes les commandes intégrées retournent 2 pour indiquer une utilisation incorrect. Bash lui-même retourne le code de sortie de la dernière commande exécutée, sauf si une erreur de syntaxe se produit, dans ce cas il se termine avec un code non-zéro.

Signaux

Quand bash est interactif, en l'absence de traps, il ignore **SIGTERM**, et **SIGINT** est géré (donc wait est intérruptible). Dans tous les cas, bash ignore **SIGQUIT**. Si de contrôle de job est effectif, bash ignore **SIGTTIN**, **SIGTTOU**, et **SIGTSTP**.

Les commandes non-intégrées lancées par bash ont des gestionnaires de signaux définis aux valeurs hérités par le shell de son parent. Quand le contrôle de job est effectif, les commandes asynchrone ignorent **SIGINT** et **SIGQUIT** en plus des handlers hérités. Les commandes lancées en résultat d'une substitution de commande ignorent les signaux de contrôle générés au clavier **SIGTTIN**, **SIGTTOU**, et **SIGTSTP**.

Le shell quitte par défaut une fois reçu un **SIGHUP**. Avant de quitter, un shell interactif envoie **SIGHUP** a tous les jobs en cours ou stoppés. Les jobs stoppés reçoivent **SIGCONT** pour s'assurer qu'ils reçoivent bien **SIGHUP**. Pour empêcher le shell d'envoyer le signal à un job particulier, il doit être supprimé de la table des jobs avec **disown** ou marqué pour ne pas recevoir **SIGHUP** en utilisant **disown -h**.

Si l'option **huponexit** a été mis avec **shopt**, bash envoie un **SIGHUP** à tous les jobs quand un login shell interactif se termine. Si bash attend qu'une commande se termine et reçoit un signal pour lequel un trap est définis, le trap ne l'exécutera pas tant que la commande ne s'est pas terminé. Quand bash attend une commande asynchrone via wait, la réception d'un signal pour lequel un trap a été définis va forcer wait à retourner immédiatement avec un code de sortie supérieur à 128, immédiatement après que de trap ait été exécuté.

Contrôle de job

Le contrôle de job réfère à la capacité à sélectivement stopper (suspendre) l'exécution de processus et le continuer leur exécution ultérieurement. Le shell associe un job avec chaque pipeline. Il conserve une table de jobs actuellement exécutés, qui peut être listé avec jobs. Quand bash lance un job asynchrone (en tâche de fond), il affiche un ligne qui ressemble à :

[1] 25647
indiquant que ce job est le job numéro 1 et le process ID du dernier processus dans le pipeline associé à ce job est 25647. Tous les processus dans un seul pipeline sont membre du même job. bash utilise l'abstraction de job comme base pour le contrôle de job.

Pour faciliter l'implémentation de l'interface utilisateur pour contrôler les jobs, le système d'exploitation maintient la notion de **current terminal process group ID**. Les membres de ce groupe de processus (les processus dont le process group ID est égal au process group ID du terminal courant) reçoivent les signaux clavier tel que **SIGINT**. Ces processus sont dit en **foreground**. Les processus **background** sont ceux dont le process group ID diffère de celui du terminal. De tels processus sont immunisés aux signaux clavier. Seul les processus **foreground** sont autorisés à lire ou écrire dans le terminal, si l'utilisateur le spécifie avec stty tostop. Les processus **background** qui tentent de lire ou écrire quand stty tostop est effectif dans le terminal reçoivent **SIGINT (SIGTTOU)** par le pilote terminal du kernel, qui, à moins d'être capturé, suspend le processus.

Si l'OS supporte le contrôle de job, bash contient de facilitées pour l'utiliser. En tapant de caractère suspend (généralement ^Z, Control-Z) pendant qu'un processus fonctionne, celui-ci est stoppé et le contrôle est retourné à bash. En tapant le caractère delayed suspend (généralement ^Y, Control-Y), le processus est stoppé quand il tente de lire l'entrée depuis le terminal. L'utilisateur peut ainsi manipuler l'état de ce job, en utilisant les commandes bg, fg et kill.

Il y a différentes manières de faire référence à un job dans le shell. Le caractère **%** introduit une spécification de job (jobspec). Le job numéro n peut être référencé par **%n**. Un job peut aussi être référencé en utilisant un préfixe du nom utilisé pour le lancer, ou en utilisant une sous-chaîne qui apparaît dans sa ligne de commande. Par exemple, **%ce** fait référence au job nommé **ce** stoppé. Si le préfixe matche plus d'un job, bash reporte une erreur. **% ?ce** fait référence à tout job contenant là chaîne **ce** dans sa ligne de commande. Si la sous-chaîne matche plus d'un job, bash reporte une erreur. Les symboles **%%** et **%+** font référence à la notion de job courant, qui est le dernier job stoppé pendant qu'il étaie en foreground ou lancé en background. e précédent job peut être référencé en utilisant **%-**.

S'il y a seulement un seul job, **%+** et **%-** peuvent être utilisés pour faire référence à ce job. Dans la sortie se rapportant aux jobs (ex : la sortie de la commande jobs), le job courant est toujours flaggé avec un **+**, et le précédent job avec un **-**. Un simple **%** fait également référence au job courant.

Nommer simplement un job peut être utilisé pour le passer en foreground : **%1** est un synonyme pour **fg %1**, et passe le job 1 du background en foreground. Similairement, **%1 &** résume le job 1 en background, équivalent à **bg %1**.

Le shell apprend immédiatement quand un job change de statut. Normalement, bash attend jusqu'à ce qu'il affiche un prompt avant de reporter les changements dans le statut d'un job afin de ne pas interrompre une autre sortie. Si l'option **-b** de la commande set est activée, bash reporte de tels changements immédiatement. Tout trap sur **SIGCHLD** est exécuté pour chaque enfant qui se termine.

Si une tentative de quitter bash est faite pendant que les jobs sont stoppés, le shell affiche un warning, et, si l'option **checkjobs** est activé, liste les jobs et leur statuts. La commande jobs peut être ainsi utilisé pour inspecter leur statut.

Prompting

Exécuté intéractivement, bash affiche le prompt primaire quand il est prêt à lire une commande, et le prompt secontaire quand il demande plus d'entrée pour compléter une commande. les caractères suivants peuvent être utilisés pour personnaliser ces prompt :

\a Le caractère ASCII bell (07)

\d La date au format "Jours Mois" ex : **Tue May 26**

\D{format} Le format est passé à strftime(3)

\e Le caractère ASCII d'échappement (033)

\h Le nom d'hôte jusqu'au premier point

\H Le nom d'hôte

\j Le nombre de jobs actuellement géré par le shell

\l Le basename du nom du périphérique terminal du shell

\n nouvelle ligne

\r retour charriot

\s Le nom du shel, le basename de $0

\t L'heure courante HH :MM :SS au format 24 heures

\T L'heure courante HH :MM :SS au format 12 heures

\@ L'heure courante HH :MM au format 12 heures

\A L'heure courante HH :MM au format 24 heures

\u Le nom de l'utilisateur courant

\v La version de bash

\V La release de bash

\w Le répertoire de travail courant, $HOME est remplacé par un tilde (valeur de PROMPT_DIRTRIM)

\W Le répertoire de travail courant, $HOME est remplacé par un tilde

\! le numéro d'historique de cette commande

\# Le numéro de commande de cette commande

\$ Is l'ID effectif est 0, un # sinon un $

\nnn Le caractère correspondant au numéro octal nnn

**** Le caractère \

\[Commence une séquence de caractère non imprimable, qui peut être utilisé pour embarquer une séquence de contrôle de terminal

\] La fin d'un séquence de caractères non imprimable.

Le numéro de commande et le numéro d'historique sont généralement différent : le numéro d'historique d'une commande est sa position dans la liste d'historique, qui peut inclure des commandes restaurée depuis le fichier d'historique. Le numéro de commande est la position dans la séquence de commandes exécutées durant la session du shell courant.

READLINE

C'est la librairie qui manipule l'entrée avec un shell interactif, sauf si l'option **–noediting** est donné. L'édition de ligne est aussi utilisé avec l'option **-e** de la commande **read**. Par défaut, les commandes d'édition de ligne sont similaires à celles d'emacs. Un style vi est également disponible. L'édition de ligne peut être activé avec **-o emacs** ou **-o vi** de la commande set. Pour la désactiver, **+o emacs** ou **+o vi**.

Notation

Dans cette section le style emacs est utilisé. Les clés de contrôle sont dénotés par C-key et les clé méta sont dénotés M-key. Pour les claviers sans touche meta, utiliser la touche échappe. Les commandes Readline peuvent être donnés en arguments numériques, qui normalement agit comme un compteur de répétition. Parfois, cependant, c'est le signe de l'argument qui est significatif. En passant un argument négatif à une commande qui agit en avant, agit en arrière. Quand une commande est décrite comme killing teste, le teste supprimé est sauvé pour une récupération future (yanking). Le texte tué est sauvé dans un kill ring. Des kills consécutifs accumulent le texte en une unité, qui peut être yanké en une seule fois. Le commandes qui ne kill pas le texte séparent le portions de texte dans le kill ring.

Initialisation de Readline

Readline est personnalisé en plaçant des commandes dans un fichier d'initialisation (le fichier **inputrc**). Le nom de ce fichier est pris de la valeur de **INPUTRC**. Si cette variable est non définis, utilise **~/.inputrc**. Quand un programme utilise la librairie Readline au démarrage, le fichier d'initialisation est lu, et le combinaisons de touche et variables sont définies. Il n'y a que quelques constructions basiques permises dans le fichier d'initialisation.

Exemple, la ligne suivante
M-Control-u : universal-argument
ou
C-Meta-u : universal-argument
Définis une combinaisons de touche pour la commande readline universal-argument
Les noms de caractère symbolique suivant sont reconnus : **RUBOUT, DEL, ESC, LFD, NEWLINE, RET, RETURN, SPC, SPACE, TAB** En plus des noms de commande, Readline permet de lier de touches à une chaîne qui est insérée quand la touche est pressée (une macro).

Combinaisons de touches

La syntaxe pour contrôler les combinaisons de touche dans le fichier inputrc est simple. Il faut le nom de la commande ou le texte d'une macro et une séquence de touches. Le nom peut être spécifié de deux manières : comme nom de touche symbolique, optionnellement préfixeé avec Meta- ou Control-, ou comme séquence de touche. En utilisant la forme **keyname :function-name** ou **macro**, par exemple :

Control-u : universal-argument
Meta-Rubout : backward-kill-word
Control-o : "> output"

Dans cet exemple, **C-u** est lié à la fonction **universal-argument**, **M-DEL** est lié à la fonction **backward-kill-word**, et **C-o** est une macro. (insert "**> output**").

Dans la seconde forme, **"keyseq".function-name** ou **macro**, Une séquence de touche entière peut être spécifiée en plaçant la séquence entre guillemets. Certains échappements de caractère Emacs peuvent être utilisés, comme dans l'exemple suivant, mais les caractères symboliques ne sont pas reconnus.

"\C-u" : universal-argument
"\C-x\C-r" : re-read-init-file
"\e[11~" : "Function Key 1"

Dans cet exemple, **C-u** est lié à la fonction **universal-argument**, **C-x C-r** est lié à la fonction **re-read-init-file** et **ESC [1 1 ~** est une macro qui insert **"Function Key 1"**. Le jeu complet de séquences d'échappement Emacs est :

\C- control prefix
\M- meta prefix
\e an escape character
**** backslash
\" literal "
\' literal '

En plus des séquence Emacs, un autre jeu d'échappement est disponible :

\a alert (bell)

\b BACKSPACE

\d delete

\f form feed

\n NEWLINE

\r retour charriot

\t tabulation horizontal

\v tabulation vertical

\NNN La valeur 9-bits de numéro octal NNN (1 à 3 chiffres)

\xHH La valeur 9-bits de numéro hexa HH (1 à 2 chiffres)

En entrant le texte d'un macro, les guillemets simple ou double doivent être utilisés pour indiquer une définition de macro. Le texte non-quoté est supposé être un nom de fonction. Dans le corps d'un macro, l'échappement "\" est étendu. bash permet de modifier ces liaisons avec la commande bind. Le mode édition peut être utilisé en utilisant l'option -o de la commande set.

Variables ReadLine

Readline a des variables que peuvent être utilisées pour personnaliser sont fonctionnement. Une variable peut être définie dans le fichier inputrc, sous la forme :

set variable-name value

Sauf mention, les variables readline peuvent prendre les valeurs **on** ou **off**. Les noms de variable non reconnues sont ignorés. Quand une variable est lue, les valeurs vide, null, "on" et "1" sont équivalents à **On**. Toutes les autres variables sont équivalentes à **off**. Les valeurs par défauts sont spécifiées entre parenthèses :

bell-style (audible) tente de jouer le bell du terminal, **none** pour le désactiver.

bind-tty-special-chars (On) Tente le lier les caractères de contrôle traités spécialement par le terminal à leur équivalent readline.

colored-stats (Off) À **on**, tente d'afficher les completions possibles en utilisant différentes couleurs pour indiquer le type de fichier. Les définitions de couleurs sont prises depuis la valeur **LS_COLORS**.

comment-begin ("#") La chaîne qui est insérée quand la commande readline **insert-comment** est exécutée. Cette commande est liée à **M-#** en mode emacs, et à **#** en mode vi.

completion-ignore-case (Off) À **on**, readline est insensible à la casse lors des completions et filename matching.

completion-prefix-display-length (0) La longueur en caractères du préfixe commun d'une liste de complétions possible qui est affiché sans modification. à une valeur supérieur à 0, les préfixes communs plus longs sont remplacés avec une ellipse quand les completions possible sont affichés.

completion-query-items (100) Détermine quand demander à l'utilisateur pour voir les completions possibles générées par la commande **possible-completions**.

convert-meta (On) Readline convertit les caractères avec le 8-ème bit mit en une séquence clé ASCII en supprimant ce 8-ème bit et en préfixeant du caractère échappe.

disable-completion (Off) À **On**, readline inhibe la completion de mots. Les completions de caractères seront insérés dans la line comme s'ils avaient été mappés à **self-insert**.

editing-mode (emacs) Contrôle si Readline utilise un jeu de combinaisons de touches similaire à emacs ou vi.

echo-control-characters (On) Readline répète un caractère correspondant à un signal généré depuis le clavier.

enable-keypad (Off) à **on**, readline tente d'activer le keypad quand il est appelé. Certains systèmes en ont besoin pour accéder aux touches fléchées.

enable-meta-key (On) readline tente d'activer une touche méta demandé par le terminal. Dans de nombreux terminaux, la touche méta est utilisée pour envoyer des caractères 8-bits.

expand-tilde (Off) à **on**, l'expansion de tilde est effectuée quand readline tente la completion de mot.

history-preserve-point (Off) à **on**, le code d'historique tente de placer un point au même emplacement dans chaque ligne d'historique récupérée avec **previous-history** or **next-history**.

history-size (0) Définis le nombre maximum d'entrées d'historique sauvegardé. À 0, toutes les entrées sont supprimées et aucune entrée n'est sauvegardée. Par défaut, le nombre d'entrée n'est pas limité.

horizontal-scroll-mode (Off) à **on**, ReadLine utilise une seule ligne pour l'affichage, scroll l'entrée horizontalement dans une seule ligne quand elle devient plus grande que l'écran.

input-meta (Off) À **on**, Readline active l'entrée 8-bits sans regardes si le terminal le supporte. le nom **meta-flag** est un synonyme pour cette variable.

isearch-terminators ("C-[C-J") La chaîne de caractères qui devrait terminer une recherche incrémentale sans exécuter le caractère comme commande. Si cette variable n'a pas de valeur, les caractères ESC est C-J vont terminer une recherche incrémentale.

keymap (emacs) Définis le mappage ReadLine courant. Les jeu de keymap valide est **emacs**, **emacs-standard**, **emacs-meta**, **emacs-ctlx**, **vi**, **vi-command** et **vi-insert**.

keyseq-timeout (500) Spécifies la durée d'attente d'un caractère quand une séquence de touche ambigüe est lue. Si aucune entrée n'est reçue passée ce délai, readline utilise la séquence de touche. La valeur est spécifiée en millisecondes, Si non définis ou à une valeur inférieur ou égal à 0 ou à une valeur non-numérique, readline attend toujours une autre touche pour décider quelles séquence de touche compléter.

mark-directories (On) Les noms de répertoires complétés ont un **/** ajouté.

mark-modified-lines (Off) À **on**, les lignes d'historique qui ont été modifiés sont suffixés avec un (*)

mark-symlinked-directories (Off) À **on**, les noms complétés qui sont des liens symboliques vers des répertoires ont un **/** ajouté

match-hidden-files (On) ReadLine matche les noms de fichiers cachés lors de la completion.

menu-complete-display-prefix (Off) À **on**, la completion de menu affiche de préfixe commun de la liste de completions possible avant de cycler dans la liste.

output-meta (Off) À **on**, readline affiche les caractères avec le 8-ième bis mis directement plutôt que comme séquence d'échappement préfixeé par méta.

page-completions (On) ReadLine utilise un pager style more interne pour afficher un écran complet de completions possible.

print-completions-horizontally (Off) À **on**, ReadLine affiche les completions avec les matches triés horizontalement dans l'ordre alphabétique.

revert-all-at-newline (Off) À **on**, ReadLine annule tout changements dans les lignes d'historique avant de retourner quand **accept-line** est exécuté.

show-all-if-ambiguous (Off) À **on**, les mots qui ont plus d'une completion possible affichent immédiatement les matches au lieu de sonner le bell

show-all-if-unmodified (Off) À **on**, les mots qui on plus d'une completion possible sans completion partielle possible (les completions possible ne partagent pas de prefixe commun) affichent immédiatement les matches au lieu de sonner le bell

show-mode-in-prompt (Off) À **on**, ajoute un caractère au début du prompt indiquant le mode d'édition : emacs (@), vi command (:) ou vi insertion (+)

skip-completed-text (Off) À **on**, Altère la completion en insérant un simple matche dans la ligne. Actif seulement lors de la completion au milieu d'un mot. Si actif, readline n'insert pas de caractères de la completion qui matche les caractère après un point dans le mot à compléter.

visible-stats (Off) À **on**,Un caractère dénotant un type de fichier comme reporté par stat est ajouté au nom de fichier en listant les completions possibles.

Constructions conditionnelles de ReadLine

Readline implémente une facilité similaire aux compilations comditionnel du préprocesseur C qui permet à des raccourcis et variables d'être exécutés en tant que résultat de tests. Il y a 4 directives utilisés.

$if Permet de baser les liaisons sur le mode d'édition, le terminal utilisé, ou l'application utilisant readline.

mode Utilisé pour tester si readline est en mode vi ou emacs.

term Utilisé pour inclure des liaisons spécifique au terminal, par exemple pour lier la sortie des séquences de touche à des fonctions du terminal. Le mot est testé avec le nom complet du terminal et la portion du nom avant le premier **-**.

application Utilisé pour inclure des paramètres spécifiques à une application. Chaque programme utilisant readline définis le nom d'application. La commande suivante ajoute une séquence de touches qui quote le mot courant et le précédant dans bash :

```
$if Bash
# Quote the current or previous word
"\C-xq" : "\eb\"\ef\""
$endif
```

$endif Termine une commande $if

$else Exécuté si la directive $if échoue

$include Fichier où lire des commandes et liaisons.

Recherche

ReadLine fournis des commandes pour rechercher dans l'historique de commandes des lignes contenant une chaîne spécifiée. Il y a 2 modes de recherche : incrémentale et non incrémentale.

Les recherches incrémentales commencent avant que l'utilisateur ait finis de taper la chaîne de recherche. À chaque caractère entré, readline affiche la prochaine entrée de l'historique correspondante. Une recherche incrémentale nécessite uniquement autant de caractères que nécessaire pour trouver l'entrée d'historique désirée. Les caractères présents dans **isearch-terminators** sont utilisés pour terminer une recherche incrémentale. Si cette variable n'a pas de valeur, Escape et Control-J terminent la recherche incrémentale. Contrôle-G annule une recherche incrémentale et restaure la ligne originale. Quand la recherche est terminée, l'entrée d'historique contenant la chaîne recherchée devient la ligne courante.

Pour trouver d'autre entrées correspondantes dans la liste d'historique, utiliser Control-S ou Control-R. Toute autre séquence liée à une commande readline termine la recherche et exécuter la commande. ReadLine mémorise la dernière chaîne de recherche incrémentale. Si 2 Control-R sont tapés sans un caractère définissant une nouvelle recherche, la chaîne mémorisée est utilisée.

Les recherches non-incrémentales lisent toute la chaîne de recherche avant de commencer la recherche.

Noms de commandes ReadLine

La liste suivante contient les noms des commandes et leur séquence de touche par défaut. Dans les descriptions, **point** réfère à la position courant du curseur, et **mark** réfère à la position du curseur sauvée par la commande **set-mark**. Le texte entre le point et mark est la **region**

Commandes de déplacement

beginning-of-line (C-a) Se placer au début de la ligne courante

end-of-line (C-e) Se placer à la fin de la ligne

forward-char (C-f) Se déplacer d'un caractère en avant

backward-char (C-b) Se déplacer d'un caractère en arrière

forward-word (M-f) Se déplacer à la fin du prochain mot

backward-word (M-b) Se déplacer au début du mot courant ou précédent

shell-forward-word Se déplacer à la fin du prochain mot

shell-backward-word Se déplacer au début du mot courant ou précédent

clear-screen (C-l) Effacer l'écran en plaçant la ligne courant à la ligne du haut

redraw-current-line Rafraîchis la ligne courante

Commandes de manipulation d'historique

accept-line (Newline, Return) Accepte la ligne sans regarder où se situe le curseur. Si la ligne n'est pas vide, l'ajoute à la liste d'historique. Si la ligne est une ligne d'historique modifiée, restaure la ligne à son état original

previous-history (C-p) Se déplacer en arrière dans la liste d'historique

next-history (C-n) Se déplacer en avant dans la liste d'historique

beginning-of-history (M-<) Se placer à la première ligne d'historique

end-of-history (M->) Se placer à la fin de l'historique

reverse-search-history (C-r) Rechercher en arrière dans l'historique en commençant à la ligne courante. C'est une recherche incrémentale

forward-search-history (C-s) Rechercher en avant dans l'historique en commençant à la ligne courante. C'est une recherche incrémentale

non-incremental-reverse-search-history (M-p) Recherche en arrière dans l'historique en commençant à la ligne courant en utilisant une recherche non-incrémentale.

non-incremental-forward-search-history (M-n) Recherche en avant dans l'historique en commençant à la ligne courant en utilisant une recherche non-incrémentale.

history-search-forward Recherche en avant dans l'historique la chaîne de caractères entre le début de la ligne courante et le point. C'est une recherche non-incrémentale

history-search-backward Recherche en arrière dans l'historique la chaîne de caractères entre le début de la ligne courante et le point. C'est une recherche non-incrémentale

yank-nth-arg (M-C-y) Insert le premier argument de la commande précédente au point. Avec un argument n, insert le n-ième mot de la commande précédente. Un argument négatif commence à la fin de la commande précédente. Une fois l'argument calculé, l'argument est extrait comme si l'expansion **!n** avait été spécifié.

yank-last-arg (M-., M-_) Insert le dernier argument de la commande précédente. Une fois l'argument calculé, l'argument est extrait comme si l'expansion **!$** avait été spécifié.

shell-expand-line (M-C-e) Étend la ligne comme le shell. Celà effectue une expansion d'alias et d'historique de la même manière que l'expansion de mot du shell.

history-expand-line (M-^) Expansion d'historique sur la ligne courante.

magic-space Expansion d'historique sur la ligne courante et insert un espace

alias-expand-line Expansion l'alias sur la ligne courante

history-and-alias-expand-line Expansion d'historique et d'alias sur la ligne courante.

insert-last-argument (M-., M-_) synonyme pour **yank-last-arg**

operate-and-get-next (C-o) Accepte la ligne courante et remplis la ligne suivante relativement à la ligne courante dan l'historique. Tout argument est ignoré

edit-and-execute-command (C-xC-e) Invoque un éditeur dans la ligne de commande courante, et exécute le résultat en tant que commande shell. bash tente d'invoquer VISUAL, EDITOR, et emacs dans cet ordre.

Commandes pour changer du texte

end-of-file (usually C-d) Le caractère indiquant la fin du fichier.

delete-char (C-d) Supprime le caractère au point.

backward-delete-char (Rubout) Supprime le caractère avane le curseur

forward-backward-delete-char Supprime le caractère sous le curseur

quoted-insert (C-q, C-v) Ajoute le prochain caractère tapé textuellement

tab-insert (C-v TAB) Insert un caractère tabulation

self-insert (a, b, A, 1, !, ...) Insert le caractère tapé

transpose-chars (C-t) Fait glisser le mot avant le point aprés ce point, déplaçant le point après ce mot également. Si le point est à la fin de la ligne, transpose les 2 derniers mot de la ligne.

upcase-word (M-u) Met en majuscule le mot courant, ou le précédant si l'argument est négatif

downcase-word (M-l) Met en minuscule le mot courant, ou le précédant si l'argument est négatif

capitalize-word (M-c) Capitalise le mot courant, ou le précédant si l'argument est négatif

overwrite-mode Passe en mode écarsement. Avec un argument positif explicite, switche en mode overwrite. Avec un argument négatif explicite, switche en mode insert. Chaque appel à readline() commence en mode insert. N'affecte que le mode emacs. En mode overwrite, les caractères liés à **self-insert** remplacent le texte au point au lieu de pousser le texte. Les caractères liés à **backward-delete-char** remplacent le caractère avant le point avec un espace.

Killing et Yanking

kill-line (C-k) Tue le texte depuis le point jusqu'à la fin de la ligne

backward-kill-line (C-x Rubout) Tue le teste au début de la ligne

unix-line-discard (C-u) Tue le teste depuis le point jusqu'au début de la ligne. Ce texte tué est sauvé dans le kill-ring

kill-whole-line Tue toute la ligne

kill-word (M-d) Tue le mot courant

backward-kill-word (M-Rubout) Tue le mot avant le point

shell-kill-word (M-d) Tue depuis le point jusqu'à la fin du mot courant

shell-backward-kill-word (M-Rubout) Tue le mot derrière le point

unix-word-rubout (C-w) Tue le mot derrière le point, en utilisant les espaces blanc comme délimiteur de mot. Le texte tué est sauvé dans le kill-ring

unix-filename-rubout Tue le mot derrière le point, en utilisant les espaces blanc et le slash comme délimiteur de mot. Le texte tué est sauvé dans le kill-ring

delete-horizontal-space (M-\) Supprime tous les espaces et tabulations autour du point

kill-region Tue de texte dans la region courante

copy-region-as-kill Copie de texte dans la region dans le tampon kill

copy-backward-word Copie le mot avant le point dans le tampon kill

copy-forward-word Copie le mot suivant le point dans le tampon kill

yank (C-y) déplace le haut du kill-ring dans le tampon au point

yank-pop (M-y) Rotation du kill-ring, et déplace le nouveau haut de la pile.

Arguments numérique

digit-argument (M-0, M-1, ..., M–) Ajoute ce chiffre à l'argument déjà accumulé, ou commence un nouvel argument. M– commence un argument négatif

universal-argument Autre manière de spécifier un argument. Si cette commande est suivie par un ou plusieurs chiffres, optionnellement avec un signe moins, cet chiffres définissent un argument. Si la commande est suivi par des chiffres, exécuter de nouveau **universal-argument** termine l'argument, mais est ignoré. Un cas spécial, si la commande est immédiatement suivie par un caractère qui n'est ni un chiffre ni le signe moins, le compteur d'arguments pour la prochaine commande est multiplié par 4.

Conpletion

complete (TAB) Tente d'effectuer une completion sur le texte avant le point. Bash tente la completion en traitant le texte comme une variable (si le texte commence avec $), nom d'utilisateur (si le texte commence avec ~), nom d'hôte (si le texte commence avec @), ou une commande (incluant les alias et fonctions). S'il n'y a toujours aucun match, tente une completion de nom de fichier.

possible-completions (M-?) Leste les completions possible du texte avant le point

insert-completions (M-*) Insert toutes les completions du texte avant le point qui aurait été généré par possible-completion

menu-complete Similaire à **complete**, mais remplace le mot à completer avec un seul match dans la liste des completions possible. Une exécution répétée de **menu-complete** passe à l'élément suivant dans la liste des completions possible.

menu-complete-backward Identique à **menu-complete** mais se déplace en arrière dans la liste des completions possible.

delete-char-or-list Supprime le caractère sous le curseur s'il n'est pas au début ou à la fin de la ligne (comme **delete-char**). S'il est au début ou à la fin de la ligne, devient identique à **possible-completions**

complete-filename (M-/) Tente une completion de nom de fichier sur le texte avant le point

possible-filename-completions (C-x /) Liste les completions possible sur le texte avant le point, traité comme nom de fichier

complete-username (M-~) Tente une completion de nom d'utilisateur sur le texte avant le point.

possible-username-completions (C-x ~) Liste les completions possible sur le texte avant le point, traité comme nom d'utilisateur

complete-variable (M-$) Tente une completion de variable sur le texte avant le point.

possible-variable-completions (C-x $) Liste les completions possible sur le texte avant le point, traité comme variable

complete-hostname (M-@) Tente une completion de nom d'hôte sur le texte avant le point.

possible-hostname-completions (C-x @) Liste les completions possible sur le texte avant le point, traité comme nom d'hôte

complete-command (M-!) Tente une completion de commande sur le texte avant le point.

possible-command-completions (C-x !) Liste les completions possible sur le texte avant le point, traité comme commande

dynamic-complete-history (M-TAB) Tente une conpletion d'historique sur le texte avant le point

dabbrev-expand Tente une completion de menu sur le texte avant le point, en comparant le texte avac les lignes de l'historique

complete-into-braces (M-{) Effecture une completion de nom de fichier et insert la liste des completion possible entre accolade pour que la liste sont disponible au shell.

Macros Clavier

start-kbd-macro (C-x () Commence à sauver les caractères tapés dans la macro clavier courante

end-kbd-macro (C-x)) Stop l'enregistrement des caractère tapés dans la macro clavier courante

call-last-kbd-macro (C-x e) Re-exécute la dernière macro définie

print-last-kbd-macro () Affiche la dernière macro définie dans un format utilisable dans inputrc

Divers

re-read-init-file (C-x C-r) recharge le contenu du fichier inputrc,

abort (C-g) Annule la commande d'édition courante.

do-uppercase-version (M-a, M-b, M-x, ...) Si le caractère utilisé avec Méta est en minuscule, lance la commande qui est lié au caractère majuscule correspondant.

prefix-meta (ESC) Le prochain caractère est un caractère méta. par exempre **ESC f** est équivalent à **Meta-f**

undo (C-_, C-x C-u) undo incrémental

revert-line (M-r) Annule tout changements fait sur cette ligne

tilde-expand (M-&) Effecue une expansion de tilde sur le mot courant

set-mark (C-@, M-<space>) Définis le mark sur le point. Si un argument est fournis, définis le mark à cette position

exchange-point-and-mark (C-x C-x) inter-change le point et le mark

character-search (C-]) Un caractère est lu et le point est déplacé à la prochaine occurrence de ce caractère.

character-search-backward (M-C-]) Un caractère est lu est le point est déplacé à la précédente occurrence de ce caractère.

skip-csi-sequence Lit suffisamment de caractère pour consommer une séquence multi-touche telle que celles définies pour des touches comme Home et End. De telles séquences commencent avec un Control Sequence Indicator (CSI), généralement ESC-[. Si la séquence est liée à **\[**, les touches produisant de telles séquences n'auront pas d'effet sauf explicitement liée à une commande readline, au lieu d'insérer les caractères dans le tampon d'édition.

insert-comment (M-#) Sans argument numérique, la valeur de la variable readline **comment-begin** est insérée au début de la ligne courante. Si un argument numérique est fournis, cette commande agit comme un switch : Si les caractères au début de la ligne ne matchent pas la valeur de **comment-begin**, la valeur est insérée, sinon les caractères dans **comment-begin** sont supprimés du début de la ligne. La valeur par défaut de **comment-begin** force cette commande à commenter la ligne courante.

glob-complete-word (M-g) Le mot avant le point est traité comme pattern pour l'expansion de chemin, avec un * ajouté implicitement. Ce pattern est utilisé pour générer une liste de noms de fichiers possible.

glob-expand-word (C-x *) Le mot avant le point est traité comme pattern pour l'expansion de chemin, et la liste de noms de fichiers matchant est inséré, remplaçant le mot. Si un argument numérique est fournis, un * est ajouté implicitement avant l'expansion.

glob-list-expansions (C-x g) La liste des expansions qui auraient été générés par **glob-expand-word** est affichée, et la ligne et redéssinée. Si un argument numérique est fournis, un * est ajouté implicitement avant l'expansion.

dump-functions Affiche toutes les fonctions et leur liaisons dans le flux de sortie de readline. Si un argument numérique est fournis, la sortie est formatée au format inputrc

dump-variables Affiche toutes les variables readline dans le flux de sortie de readline. Si un argument numérique est fournis, la sortie est formatée au format inputrc

dump-macros Affiche toutes les séquences de touche et leur valeurs dans le flux de sortie de readline. Si un argument numérique est fournis, la sortie est formatée au format inputrc

display-shell-version (C-x C-v) Affiche le version de l'instance courante.

Completion Programmable

Quand la completion de mot et tenté par un argument d'une commande pour laquelle une spécification de completion (compspec) a été définie en utilisant la commande **complete**, la fonction de completion programmable est invoquée.

Premièrement, le nom de la commande est identifiée. Si la commande est une chaîne vide, tout compspec définie avec l'option **-E** de la commande complete est utilisée. Si un compspec a été définis pour cette commande, le compspec est utilisé pour générer la liste de completions possible. Si la commande est un chemin complet, un compspec pour ce chemin complet et recherché en premier. Si ces recherches ne réussissent pas, tout compspec définis avec l'option **-D** est utilisé.

Une fois un compspec trouvé, il est utilisé pour générer la liste de mots correspondant. Si un compspec n'est pas trouvé, la completion par défaut de bash est effectué.

D'abord, les actions spécifiées par le compspec sont utilisés. Seul les matchs qui sont préfixeés par le mot à compléter sont retournés. Quand l'option **-f** ou **-d** est utilisé pour la completion de nom de fichier ou de nom de répertoire, la variable shell **GLOBIGNORE** n'est pas utilisée pour filtrer les matchs, mais la variable **FIGNORE** est utilisée.

Ensuite, la chaîne spécifiée comme argument de l'option **-W** est considérée. La chaîne est d'abord splitté en utilisant les caractères dans **IFS**. Le quoting est honoré. Chaque mot est ensuite étendu en utilisant l'expansion d'accolade, l'expansion de tilde, l'expansion de paramètre et de variable, la substitution de commande, et l'expansion arithmétique. Les résultat sont splittés en utilisant les règles de Word Splitting. Les résultats de l'expansion sont préfixe-matchés avec le mot à compléter, et les mots matchés deviennent les completions possible.

Après que ces matchs aient été générés, toute fonction ou commande shell spécifiée avec les options **-F** et **-C** est invoquée, les variables **COMP_LINE**, **COMP_POINT**, **COMP_KEY** et **COMP_TYPE** sont ajustées. Si une fonction shell est invoquée, les variables **COMP_WORDS** et **COMP_CWORD** sont également ajustées. Quand la fonction ou la commande est invoquée, le premier argument (**$1**) est le nom de la commande dont les arguments sont à compléter, le deuxième argument (**$2**) est le mot à compléter, et le troisième argument (**$3**) est le mot précédant le mot à compléter sur la ligne courante.

Toute fonction spécifiée avec **-F** est invoquée en premier. La fonction peut utiliser toute facilité du shell, incluant le compgen, pour générer les matchs. Il dois placer les completions possible dans le tableau **COMPREPLY**.

Ensuite, toute commande spécifiée avec l'option **-C** est invoquée dans un environnement équivalent à la substitution de commande. Elle devrait afficher une liste de completions, une par ligne, sur la sortie standard. un \ peut être utilisé pour échapper un newline, si nécessaire.

Une fois toutes les completions possible générées, tout filtre spécifié avec l'option **-X** est appliqué à la liste. Le filtre est un motif comme utilisé pour l'expansion de chemin ; un **&** dans le pattern est remplacé avec le texte du mot à compléter. Toute completion qui matche le pattern est supprimé de la liste. Un **!** inverse le pattern.

Finallement, tout préfixe et suffixe spécifié avec les options **-P** et **-S** sont ajoutés à chaque membre de la liste de completion, et le résultat est retourné au code de completion de readline.

Si les actions précédemment appliquées ne génèrent aucun match, et que l'option **-o dirnames** de la commande complete est fournie, la completion de nom de répertoire est tentée. Si l'option **-o plusdirs** de la commande complete est fournie, la completion de nom de répertoire est tentée et tout matche est ajouté au résultat des autres actions.

Par défaut, si un compspec est trouvé, tout ce qu'il génère est retourné au code de completion. Les completions bash par défaut ne sont pas effectués, et la completion de nom de fichier par défaut de readline est désactivée. Si l'option **-o bashdefault** est fournie, les completions par défaut de bash sont tentée quand le compspec ne génère aucun match. Si l'option **-o default** est fournie, la completion par défaut de readline est effectuée si le compspec (et, si tenté, les completions de bash) ne génère aucun match.

Quand un compspec indique que la completion de répertoire est désiré, les fonctions de completion programmable forcent readline à ajouter un **/** aux noms complétés qui sont des liens symboliques de répertoire, sujet à la valeur de **mark-directory**, sans regarder la variable **mark-symlinked-directories**.

Il y a un support pour les completions dynamiques. C'est plus utile quand utilisé en combinaison avec une completion par défaut spécifiée avec **complete -D**. Il est possible pour les fonctions shell exécutées comme manipulateurs de completion d'indiquer que la completion devrait être récupérée en retournant un code de sortie 124. Si une fonction shell retourne 124, et les changements du compspec associé avec la commande dans laquelle la completion est tentée, la completion programmable redémarre du début, et tente de trouver un autre compspec pour cette commande. Cela permet à un jeu de completion d'être construit dynamiquement.

Assumons qu'il y a une librairie de compspecs, chacun conservés dans un fichier correspondant au nom de la commande, la fonction de completion par défaut suivante charge les completions dynamiquement :

```
_completion_loader()
{
 . "/etc/bash_completion.d/$1.sh" >/dev/null 2>&1 && return 124
}
complete -D -F _completion_loader -o bashdefault -o default
```

Historique

Quand **set -o history** est activé, le shell fournis un accès à l'historique de commande, la liste des commandes précédemment tapés. La valeur de **HISTSIZE** est utilisé comme nombre de commandes à sauvegarder. Le shell stocke chaque commande avant l'expansion de paramètre et de variables, mais après que l'expansion l'historique ait été effectué, sujet aux valeurs de **HISTIGNORE** et **HISTCONTROL**.

Au démarrage, l'historique est initialisé depuis le fichier nommé par la variable **HISTFILE** (défaut : ~/.bash_history). Ce fichier est tronqué, si nécessaire, pour ne pas contenir plus de lignes que le nombre spécifié par **HISTFILESIZE**. Quand le fichier d'historique est lu, les lignes commençant avec le caractère de commentaire d'historique suivi immédiatement par un chiffre sont interprétés comme timestamps pour la précédente ligne d'historique. Ces timestamp sont optionnellement affichés en fonction de la valeur de **HISTTIMEFORMAT**. Quand un shell avec l'historique activé se termine, les dernière lignes **HISTSIZE** sont copiées de la liste d'historique dans **HISTFILE**.

Si l'option shell **histappend** est activée, les lignes sont ajoutée au fichier d'historique, sinon les lignes écrasent le contenu du fichier. Si **HISTFILE** n'est pas définis, ou si le fichier d'historique n'est pas accessible en écriture, l'historique n'est pas sauvegardé. Si

HISTTIMEFORMAT est définie, les timestamps sont écris dans le fichier d'historique, marqués avec le caractère de commentaire d'historique. Une fois l'historique sauvé, le fichier d'historique est tronqué pour contenir **HISTFILESIZE** lignes. Si **HISTFILESIZE** n'est pas définie, null, une valeur non numérique, ou une valeur inférieur à 0, le fichier d'historique n'est pas tronqué.

Le commande **fc** peut être utilisée pour lister ou éditer et ré-exécuter une portion de la liste d'historique. La commande **history** peut être utilisée pour afficher ou modifier la liste d'historique et manipuler le fichier d'historique. En utilisant l'édition de ligne de commande, les commandes de recherche sont disponible dans chaque mode d'édition qui fournissent un accès au fichier d'historique.

Le shell permet de contrôler quelles commandes sont sauvées dans la liste d'historique. Les variable **HISTCONTROL** et **HISTIGNORE** peuvent être définis pour déterminer ce que le shell sauvegarder dans la liste. La commande **cmdhist**, si activée, force le shell à tenter de sauver chaque ligne d'un commande multi-ligne dans la même entrée d'historique, en ajoutant des **;** où c'est nécessaire. L'option shell **lithist** force le shell à sauvegarder la commande avec les newline au lieu des ;.

Expansion d'historique

Le shell supporte l'expansion d'historique qui est similaire à l'expansion d'historique dans **csh**. Peut être désactivé avec **set +H**. L'expansion d'historique est effectuée immédiatement après qu'une ligne complète soit lue, avant que le shell la sépare en mots. Elle prend place en 2 parties. La première est pour déterminer quelle ligne de la liste d'historique utiliser durant la substitution. La secondes, les portions de cette ligne à inclure dans la courante. La ligne sélectionnée de l'historique est l'event, et les portions de cette ligne sont les mots. Divers modifieurs sont disponible pour manipuler les mots sélectionnés. La ligne est décomposée en mot de la même manière que lorsque l'entrée est lue, et de nombreux mots séparés par les méta-caractères entourés par des quotes sont considéré comme un seul mot. L'expansion d'historique est introduit par la présence du caractère d'expansion d'historique qui est **!** par défaut.

Des caractères inhibent l'expansion d'historique s'ils sont trouvés juste après le caractère d'expansion d'historique, même s'ils ne sont pas quotés : **space**, **tab**, **retour charriot**, et **=**. Si l'option **extglob** est activée, **(** inhibe également l'expansion

Certaine options shell peuvent être utilisées pour modifier le fonctionnement de l'expansion d'historique. Si l'option shell **histverify** est définie, et que readline est utilisé, les substitutions d'historique ne sont par immédiatement passées au parser. À la place, la ligne étendue est rechargée dans le tampon d'édition de readline pour d'autres modifications. Si readline est utilisé, et que l'option shell **histreedit** est activée, une substitution d'historique échouée sera rechargée dans le tampon d'édition de readline pour correction. L'option **-p** de la commande history peut être utilisée pour voir quelle expansion d'historique sera faite avant de l'utiliser. L'option **-s** peut être utilisée pour ajouter des commandes à la fin de la liste d'historique sans les exécuter.

Le shell permet de contrôler les divers caractères utilisés par l'expansion d'historique. Le shell utilise le caractère de commentaire d'historique pour marquer les timestamps d'historique en écrivant le fichier d'historique.

Désignateurs d'évènement

Un event designator est une référence à une ligne de commande dans la liste d'historique. À moins que la référence soit absolue, les events sont relatifs à la position courante dans la liste d'historique.

! Commence une substitution d'historique, avec les exceptions décrites plus haut.

!n Réfère à la ligne de commande n

!-n Réfère à la commande courante - n

!! Réfère à la commande précédente. synonyme de !-1

!string Réfère à la commande la plus récente précédent la position courante commençant avec string

!?string[?] Réfère à la commande la plus récente précédent la position courante commençant avec string, les ? peuvent être omis si string est suivi immédiatement par un newline.

^string1^string2 Substitution rapide. Répète la commande précédante en remplaçant string1 par string2. Équivalent à **!! :s/string1/string2/**

!# Toute la ligne de commande jusqu'à présent.

Désignateurs de mot

Les words designator sont utilisés pour sélectionner les mots désirés depuis l'event. Un **:** sépare la spécification d'event du word designator. Il peut être omis si le word designator commence avec un **^**, **$**, *****, **-**, ou **%**. Les mots sont numérotés du début de la ligne, en commençant à 0. Les mots sont insérés dans la ligne courante séparés par un espace.

0 Le mot numéro 0. Pour le shell, c'est la commande.

n Le nième mot

^ Le premier argument, le mot numéro 1

$ Le dernier mot

% Le mot matché par le plus récent ?string ? recherché

x-y Une plage de mots ; -y est une abréviation de 0-y

***** Tous les mots sauf le premier (0). C'est un synonyme de 1-$.

x* Abréviation de x-$

x- Abréviation de x-$ come x*, mais omet le dernier mot

Si un word designator est fournis sans un event specification, la précédante commande est utilisée comme event.

Modificateurs

Après le word designateur optionnel, il peut apparaître une séquence d'un ou plusieurs modifiers suivants, chacun précédés par un **:** :

h Supprime un composant restant du nom de fichier, ne laissant que le début

t Supprime tous les composant du nom de fichiers, ne laissant que la fin

r Supprime un suffixe sous la forme .xxx, laissant le basename

e Supprime tout sauf le suffixe restant.

p Affiche la nouvelle commande mais ne l'affiche pas

q Quote les mots substitués

x Quote les mots substitués, mais les sépare sur les blancs et les newline.

s/old/new/ Substitue la première occurence de old à new dans la ligne d'event. Tout délimiteur peut être utilisé à la place de /. Le délimiteur final est optionnel si c'est le dernier caractère de la ligne d'event. Si & apparaît dans new, il est remplacé par old. Un simple \ va quoter le &. Si old est null, il est définis au dernier old substitué, ou, si aucune substitution n'a eu lieu, la dernière chaîne dans la recherche **!?string[?]**

& Répète la précédente substitution.

g Les changements sont appliqués sur toute la ligne d'event. C'est utilisé en conjonction avec **:s** (ex : :gs/old/new/), ou **:&**. Si utilisé avec **:s**, tout délimiteur peut être utilisé en place de /, et le délimiteur final est optionnel si c'est le dernier caractère de la ligne d'event. Un **a** peut être utilisé comme synonyme de **g**.

G Applique de modifieur **s** une fois pour chaque mot dans la ligne d'event.

Commandes intégrées au shell

Sauf mention contraire, chaque commande intégrée dans cette section acceptant des options précédés pas **-** acceptent **–** pour signifier la fin des options. Les commandes **:**, **true**, **false**, et **test** n'acceptent pas d'options et ne traitent par **–** spécialement. Les commandes **exit**, **logout**, **break**, **continue**, **let**, et **shift** acceptent et traitent les options commençant par **-** sans nécessiter **–**. Les autres commandes qui acceptent des

arguments mais n'acceptent pas d'options interprètent les arguments commençant avec **-** comme invalide et nécessite **–** pour empêcher leur interprétation.

: [arguments[Pas d'effet ; la commande ne fait rien hormis étendre les arguments et effectue les redirections spécifiées. Un code de sortie 0 est retourné.

. filename [arguments], source filename [arguments] Lis et exécute les commandes depuis **filename** dans l'environnement du shell courant et retourne le code de sortie de la dernière commande de **filename**. Si **filename** ne contient pas de /, **PATH** est utilisé pour trouver le fichier. Le fichier recherché dans **PATH** n'a pas besoin d'être exécutable. Quand bash n'est pas en mode posix, le répertoire courant est recherché si aucun fichier n'est trouvé dans **PATH**. Si l'option **sourcepath** de shopt est désactivé, le **PATH** n'est pas recherché. Retourne 0 si aucune commande n'a été exécutée, et false si **filename** n'a pas été trouvé ou ne peut être lu.

alias [-p] [name[=value] ...] Sans arguments ou avec l'option -p, affiche la liste des alias sur stdout. Les arguments fournis servent à définir des alias. Un espace après une **valeur** force le prochain mot à être vérifié pour une substitution d'alias quand l'alias est étendu. Pour chaque nom fournis sans valeur, le nom et la valeur de l'alias sont affichés. aliar retourne true à moins qu'un nom soit donné pour lequel aucun alias n'a été définis.

bg [jobspec ...] Relance chaque job suspendu spécifié en tâche de fond, comme s'il avait été spécifié avec **&**. Si jobspec n'est pas présent, la notion shell de job courant est utilisé. **bg jobspec** retourne 0 à moins qu'il soit lancé quand le contrôle de job soit désactivé ou si jobspec n'a pas été trouvé ou a été démarré sans contrôle de job.

bind [-m keymap] [-lpsvPSVX]

bind [-m keymap] [-q function] [-u function] [-r keyseq]

bind [-m keymap] -f filename

bind [-m keymap] -x keyseq :shell-command

bind [-m keymap] keyseq :function-name

bind readline-command Affiche les touches/fonctions readline courante, lie une séquence de touche à une fonction readline ou une macro, ou définis une variable readline. Chaque argument non option est une commande au format .inputrc, mais chaque liaison ou commande doit être passée comme argument séparé ; par ex : **'"\C-x\C-r" : re-read-init-file'**. Retourne 0 à moins qu'une option non reconnues soit donnée ou une erreur s'est produite. Les options, si fournies, ont la signification suivante :

-m keymap Affecte keymap au binding suivant. Les noms de keymap acceptable sont **emacs**, **emacs-standard**, **emacs-meta**, **emacs-ctlx**, **vi**, **vi-move**, **vi-command**, et **vi-insert**. vi est équivalent à vi-command ; emacs est équivalent à emacs-standard.

-l Liste les noms de toutes les fonctions readline

-p Affiche les noms des fonctions readline et leur liaisons au format readline

-P Liste les noms des fonctions readline et leur liaisons

-s Affiche les séquences de touche liées au macros et les chaînes qu'elles affichent au format readline.

-S Liste les séquences de touche liées au macros et les chaînes qu'elles affichent

-v Affiche les variables et leur valeurs au format readline

-V Liste les variables et leur valeurs

-f filename Lis les liaisons de touche depuis le fichier

-q function Requête quels touches invoquent la fonction spécifiée

-u function Supprime toutes les touches liées à la fonction spécifiée

-r keyseq Supprime la liaison courant pour keyseq

-x keyseq :shell-command Lie keyseq à la commande. Quand shell-command est exécutée, le shell définis la variable **READLINE_LINE** au contenu du tampon de ligne readline et la variable **READLINE_POINT** à l'emplacement courant du point d'insertion. Si la commande exécutée change la valeur d'une de ces 2 variables, ces nouvelles valeurs vont être reflétées dans l'état d'édition.

-X Liste toutes les séquences liées aux commandes shell au format readline.

break [n] Sort d'une boucle **for**, **while**, **until**, ou **select**. Si n est fournis, sont de n niveau. n doit être supérieur à 1. Si n est supérieur au nombre de boucles imbriquées, sort de toutes les boucles. Retourne 0 sauf si n est inférieur à 1.

builtin shell-builtin [arguments] Exécute la commande intégrée spécifiée, et retourne son code de sortie. Utile quand une fonction à le même nom qu'une commande intégrée. Retourne false si la commande spécifiée n'est pas une commande intégrée.

caller [expr] Retourne le contexte d'un appel à une sous-routine active (une fonction ou un script exécuté avec **.** ou **source**). Sans expr, caller affiche le numéro de ligne et le nom du fichier de l'appel de la sous-routine courante. Si un entier non-négatif est fournis dans expr, caller affiche le numéro de ligne, le nom de la sous-routine, et le fichier source correspondant à la position dans la pile d'appel courante. Cette information peut être utilisée, par exemple, pour afficher un stack trace. La frame courante est la frame 0. La valeur de retour est 0 sauf si le shell n'exécute pas de sous-routine, ou expr ne correspond pas à une position valide dans la pile d'appel.

cd [-L|[-P [-e]] [-@]] [dir] Change de répertoire courant. Si **dir** n'est pas fournis, la valeur de la variable **HOME** est le défaut. Tout argument additionnel après **dir** est ignoré. La variable **CDPATH** définis le chemin de recherche pour le répertoire contenant **dir**. Si dir commance par **/**, **CDPATH** n'est pas utilisé.

-P cd utilis la structure de répertoire physique en résolvant les liens symboliques en traversant les répertoires et avant de traiter les instances de **..** dans dir.

-L Suis les liens symboliques et résolvant le lien après le traitement des instances de **..** dans dir. Si **..** apparaît dans dir, il est traité en supprimant le composant du path précédant.

Avec **-P**, si le répertoire courant ne peut être déterminer après un changement de répertoire, cd va retourner une erreur.

-@ Présente des attributs étendus associés avec un fichier comme répertoire.

Un argument **-** est convertis en **OLDPWD** avant de tenter de changer de répertoire.

command [-pVv] command [arg ...] Lance la commande spécifiée en supprimant la recherche de fonction shell. Seuls les commande intégrée ou les commandes trouvées dans **PATH** sont exécutées. Si **-p** est donné, la recherche de la commande est effectuée en utilisant une valeur par défaut pour **PATH** qui garantit de trouver tous les utilitaires standards. Si **-V** ou **-v** est founis, une description de la commande est affiché. **-v** affiche la commande ou le nom du fichier. **-V** produit une description plus verbeuse. Si **-V** ou **-v** est founis, retourne une code de sortie de 0 si la commande a été trouvée, 1 sinon. Sans ces options, si une erreur s'est produite ou commande n'est pas trouvé, retourne 127. Dans tous les autres cas, retourne le codes de sortie de la commande.

compgen [option] [word] Génère les completions possibles pour le mot en accord avec les options spécifiées, qui peuvent être toutes options acceptées par la commande complete, sauf -p et -r, et écrit les matches sur stdout. En utilisant les options **-F** ou **-C**, les divers variables shell définies par la completion programmable, si disponible, n'auront pas de valeurs utiles. Les matches sont générés de la même manière que la completion programmable. Si un mot est spécifié, seul les completions matchant ce mot sont affichés. La valeur retournée est true sauf si une option est invalide, ou qu'aucun matche n'a été généré.

complete [-abcdefgjksuv] [-o comp-option] [-DE] [-A action] [-G globpat] [-W wordlist] [-F function] [-C command] [-X fil

complete -pr [-DE] [name ...] Spécifie comment les arguments de chaque nom devrait être complétés. Si **-p** est fournis, ou sans options, les spécifications de completion existantes sont affichées dans un format ré-utilisable. L'option **-r** supprime une spécification de completion pour chaque nom, ou, si aucun nom n'est fournis, toutes les spécifications de completion. L'option **-D** indique que les options et actions restantes devraient s'appliquer à la commande de completion par défaut ; c'est à dire la completion tentée sur une commande pour laquelle aucune completion n'a été définie. L'option **-E** indique que les options et actions restantes devraient à la commande de completion vide, c'est à dire, la completion tentée sur une ligne vide.

Les arguments de **-G**, **-W**, **-X**, et les options **-P** et **-S** devraient être quotés pour les protéger des expansions avant que complete soit invoqué.

-o comp-option Contrôle divers aspects du mode de compspec au-delà de la simple générations des completions. comp-option peut être :

bashdefault Effectue le reste des completion bash par défaut si compspec ne génère aucun match.

default Utilise la completion par défaut de readline si compspec ne génère aucun match.

dirnames Effectue une completion de nom de répertoire si compspec ne génère aucun match.

filenames Dit à readline que compspec génère les noms de fichier, pour qu'il puisse effectuer tout traitement spécifique au noms de fichier (comme ajouter un **/** aux noms de répertoire, quoter les caractères spéciaux, ou supprimer les espaces restant). À utiliser avec les fonctions shell.

noquote Dit à readline de ne pas quoter les mots complétés si ils sont des noms de fichier.

nospace Dit à readline de ne pas ajouter un espace aux mots complétés à la fin de la ligne.

plusdirs Après que tous les matches définis par le compspec aient été générés, tente la completion de noms de répertoire et tous les matches sont ajoutés au résultat des autres actions.

-A action L'action peut être une des suivante peut générer la liste des completions possible :

alias Noms d'alias. Peut aussi être spécifié par **-a**

arrayvar Noms de variable tableau

binding Noms de key binding readline

builtin Noms des commandes intégrées. Peut aussi être spécifié par **-b**

command Noms de commandes. Peut aussi être spécifié par **-c**

directory Noms de répertoire. Peut aussi être spécifié par **-d**

disabled Noms des builtins désactivés

enabled Noms des builtins activés

export Noms des variables shell exportés. Peut aussi être spécifié par **-e**

file Noms de fichiers. Peut aussi être spécifié par **-f**

function Noms des fonctions shell

group Noms des groupes. Peut aussi être spécifié par **-g**

helptopic Helptopic tels qu'accèptés par la commande help

hostname Noms d'hôte, pris dans le fichier **HOSTFILE**

job Noms de jobs, si le contrôle de job est actif. Peut aussi être spécifié par **-j**

keyword Mots réservés du shell. Peut aussi être spécifié par **-k**

running Noms des jobs lancés, si le contrôle de job est actif

service Noms des services. Peut aussi être spécifié par **-s**

setopt Arguments valides pour l'option -o de la commande set.

shopt Noms d'options shell acceptés par la commande shopt

signal Noms des signaux

stopped Noms des jobs stoppés, si le contrôle de job est actif.

user Noms des utilisateurs. Peut aussi être spécifié par **-u**

variable Noms des variables shell. Peut aussi être spécifisé par **-v**

-C command La commande est exécutée dans un environnement sous-shell, et sa sortie est utilisée comme completion possible.

-F function La fonction shell est exécutée dans l'environnement du shell courant. Quand la fonction est exécutée, le premier argument est le nom de la commande dons les arguments sont à compléter, le second argument est le mot à compléter, et le troisième argument est le traitement de mot du mot à compléter sur la ligne courante. Une fois terminée, les completions possible sont récupérée depuis la valeur **COMPREPLY**.

-G globpat Le motif d'expanion de pathname **globpat** est étendus pour générer les completions possible.

-P prefix Le préfixe est ajouté au début de chaque completion possible après que toutes les options aient été appliquées.

-W wordlist La liste de mot est splittée en utilisant **IFS**, et chaque mot résultant est étendu. Les completions possible sont les membre de la liste résultante qui matche le mot à compléter.

-X filterpat Est un motif comme utilisé pour l'expansion de pathname. Il est appliqué à la liste des completions possible généré par les options et arguments précédents, et chaque completion matchant filterpat est supprimé de la liste. si filterpat commence par un **!**, le motif est inversé.

La valeur de retour est true sauf si une option est invalide, un option autre que **-p** ou **-r** est fournie sans un argument nommé, une tentative de supprimer une spécification de completion pour un nom pour lequel aucune spécification n'existe, ou une erreur s'est produite en ajoutant une spécification de completion.

compopt [-o option] [-DE] [+o option] [name] Modifie les options de completion pour chaque **name** en accord avec les options, ou pour la completion actuellement exécutée si aucun nom n'est spécifié. Si aucune option n'est donnée, affiche les options de completion pour chaque nom ou la completion courante. Les options sont les même que pour la commande complete. Les options **-D** et **-E** ont la même signification que pour la commande complete. La valeur de retour est vraie sauf si une option est invalide, le nom n'a pas de spécification de completion, ou une erreur s'est produite.

continue [n] Passe à l'itération suivante dans une boucle **for**, **while**, **until** ou **select**. Si **n** est spécifié, passe à à l'itération suivante de la n-ième boucle. n doit être supérieur à 1. Si n est supérieur au nombre de boucles imbriquées, le boucle du haut est utilisée, La valeur retournée est 0 sauf si n n'est pas égal ou supérieur à 1.

declare [-aAfFgilnrtux] [-p] [name[=value] ...]

typeset [-aAfFgilnrtux] [-p] [name[=value] ...] Déclare des variable et/ou leur donne des attributs. Su aucun nom n'est donné, affiche les valeurs des variables.

-p affiche les attributs et valeur de chaque nom. Utilisé avec name, les options additionnelles, autre que **-f** et **-F**, sont ignorés. Utilisé sans nom, il affiche les attributs et valeurs de toutes les variables ayant les attributs spécifiés par les options additionnelles. Si aucune autre option n'est fournis avec **-p**, declare va afficher les attribut et valeurs de toutes les variables shell.

-f restreins d'affichage aux fonctions shell.

-F inhibe l'affichage des définitions de fonction ; seul les noms et attributs des fonctions sont affichés. Si l'option **extdebug** est active, le nom du fichier source le le numéro de ligne où est définis la fonction est affiché également. Implique **-f**

-g Force les variables à être crées ou modifiées dans un scope global, même quand declare est exécuté dans une fonction shell.

Les options suivantes peuvent être utilisée pour restreindre la sortie aux variables avec les attributs spécifiées, ou pour donner des attributs aux variables :

-a Chaque nom est une variable tableau indexé

-A Chaque nom est une variable tableau associatif

-f Utilise les noms de fonction uniquement

-i La variable est traitée comme un entier

-l Convertis tous les caractères majuscule en minuscule lorsque la variable reçoit une valeur.

-n Donne à chaque nom l'attribut nameref, c'est à dire une référence à une autre variable. Toutes référence et assignement au nom, excepté pour changer l'attribut -n, sont effectué sur la variable référencée par la valeur du nom. Ne peut pas être appliquée aux variables tableau.

-r Mode lecture seule. Il n'est plus possible d'assigner de nouvelles valeurs à ces noms

-t Donne à chaque nom l'attribut trace. les fonction tracées héritent des trap **DEBUG** et **RETURN** du shell appelant. N'a pas de signification spéciale pour les variables.

-u Convertis tous les caractères minuscule en majuscule lorsque la variable reçoit une valeur.

-x Exporte les noms

Utiliser **+** au lieu de **-** désactive l'attribut, à l'exception de **+a** qui ne peut pas être utilisé pour détruirs une variable tableau, et **+r** qui ne supprime par l'attribut lecture seule. Utilisé dans une fonction, declare et typeset rendent chaque nom local, comme avec la commande local, sauf si **-g** est fournis. Si un nom de variable est suivis par **=value**, la valeur de la variable est assignée. En utilisant **-a** ou **-A** et la syntaxe d'assignement composée pour créer des variables tableaux, les attributs additionnels ne prennent effet qu'après l'assignement. La valeur de retour est 0 sauf si une option invalide est rencontrée, si une fonction est définie avec **-f foo=bar**, une assignation à une variable lecture seule, une assignation de valeur à une variable tableau sans utiliser l'assignation composée, un noms n'est pas valide, tenter de désactiver l'attribut lecture seul, ou afficher une fonction qui n'existe pas avec **-f**.

dirs [-clpv] [+n] [-n] Sans options, affiche la liste des répertoires courant mémorisés. L'affichage par défaut est sur une seule ligne. Les répertoires sont ajoutés à la liste avec la commande pushd, popd pour les supprimer. La valeur de retour est 0 sauf si une option invalide est rencontrée, ou l'index n est au-delà de la pile de répertoire.

-c Efface la pile de répertoires

-l produit un listing en utilisant les chemin complets.

-p Affiche la pile de répertoire, une entrée par ligne

-v Affiche la pile de répertoire, une entrée par ligne, en préfixeant chaque entrée avec sont index dans la pile.

+n Affiche la n-ième entrée en comptant depuis la gauche de la liste affichée par dirs invoqué sans options, en commençant à 0

-n Affiche la n-ième entrée en comptant depuis la droite de la liste affichée par dirs invoqué sans options, en commençant à 0

disown [-ar] [-h] [jobspec ...] Sans options, supprime chaque jobspec de la table des jobs actifs. Si jobspec n'est pas présent et que **-a** et **-r** ne sont pas fournis, le job courant est utilisé. Si **-h** est donné, chaque jobspec nqest pas supprimé de la table, mais est marqué pour que **SIGHUP** ne soit pas envoyé au job si le shell reçoit ce signal. Si aucun jobspec n'est fournis, **-a** signifie de supprimer ou marquer tous les jobs et **-r** restreins l'opération aux jobs lancés. La valeur de retour est 0 sauf si un jobspec n'est pas un job valide.

echo [-neE] [arg ...] Affiche les arguments, séparés par des espaces, suivis par une nouvelle ligne. Le code de retour est 0 sauf si une erreur d'écriture s'est produite. echo n'interprète par **–**

-n n'ajoute pas de nouvelle ligne à la fin

-e Interprète les caractère échappés

-E Désactive l'interprétation les caractères échappés, même sur les système où ils sont interprétés par défaut. L'option shell **xpg_echo** peut être utilisée pour déterminer dynamiquement si echo étend les caractères échappés par défaut.

echo interprète les séquences échappées suivantes :

\a alerte

\b BACKSPACE

\c n'affiche pas la suite

\e, \E Un caractère échappe

\f form feed

\n NEWLINE

\r retour charriot

\t tabulation horizontale

\v taulation verticale

**** \

\0nnn Le caractère 8-bits dont la valeur est nnn en octal

\xHH Le caractère 8-bits dont la valeur est HH en hexadécimal

\uHHHH Le caractère Unicode (ISO/IEC 10646) dont la valeur est HHHH en hexadécimal.

\UHHHHHHHH Le caractère Unicode (ISO/IEC 10646) dont la valeur est HHHHHHHH en hexadécimal.

enable [-a] [-dnps] [-f filename] [name ...] Les commandes enable et disable. Désactiver une commande intégrée permet à une commande qui a le même nom d'être exécutée sans spécifier un pathname complet. La valeur de retour est 0 sauf si nom n'est pas une commande intégrée, ou s'il y a une erreur au chargement d'un objet partagé.

-n chaque nom est désactivé, sinon, chaque nom est activé.

-f Permet de charger la nouvelle commande intégrée depuis le fichier objet partagé, sur les système qui supportent le chargement dynamique.

-d Supprime une commande intégrée précédemment chargée avec **-f**.

-p Liste les commandes intégrées. Sans autre arguments, les toutes les options actives, avec -n, affiche uniquement les commande désactivées.

-a Liste toutes les commandes intégrées, en spécifiant si elles sont activées ou non.

-ss La sortie est restreinte aux commandes spéciales POSIX.

eval [arg ...] Les arguments sont lus et concaténés en une commande simple. Cette commande est ensuite lue et exécutée par le shell, et son code de sortie est la valeur retournée par eval. S'il n'y a par d'argument, ou seulement un argument null, eval retourne 0.

exec [-cl] [-a name] [command [arguments]] Si la commande est spécifiée, elle remplace le shell. Aucun nouveau processus n'est créé. Les arguments deviennent les arguments de la commande. Si l'option **-l** est spécifiée, le shell place un tiret au début de l'argument 0 passé à la commande. C'est ce que login fait. L'option **-c** exécute la commande avec un environnement vide. **-a** passe le nom en tant qu'argument 0 à la commande. Si la commande ne peut pas être exécutée, un shell non-interactif se termine, sauf si l'option **execfail** est actif. Dans ce cas, exec retourne une erreur. Si la commande n'est pas spécifiée, toute redirection prend effet dans le shell courant, et retourne 0. S'il y une erreur de redirection, retourne 1.

exit [n] Le shell se termine avec le code de sortie n. Si n est omis, le code de sortie est celui de la dernière commande exécutée. Un trap sur **EXIT** est exécuté avant que le shell se termine.

export [-fn] [name[=word]] ...

export -p Les noms fournis sont exportés automatiquement dans l'environnement des commandes exécutées. Si l'option **-f** est fournis, les mons réfèrent à des fonctions. Si aucun nom n'est donné, ou si l'option **-p** est fournis, une liste de toutes les variables exportées est affichée. L'option **-n** supprime la propriété export pour chaque nom. Si un nom de variable est suivi par **=word**, la valeur lui est assignée. export retourne un code de sortie 0 sauf si une option invalide est rencontrée, un des noms n'est pas une variable shell valide, ou **-f** est fournis avec un nom qui n'est pas une fonction.

fc [-e ename] [-lnr] [first] [last]

fc -s [pat=rep] [cmd] La forme **first** sélectionne une plage de commandes de **first** à **last** dans la liste d'historique est l'affiche ou l'édite, et les ré-exécute. **first** et **last** peuvent être spécifiés comme chaîne (pour localiser la dernière commande commençant avec cette chaîne) ou un nombre (un index dans la liste d'historique, où un nombre négatif est utilisé depuis le numéro de commande courant) Si **last** n'est pas spécifié, il est définis à la commande courante (donc **fc -l -10** affiche les 10 dernières commandes) et à **first** sinon. Si **first** n'est pas spécifié il est définis à la commande précédente pour l'édition, et f -16 pour l'affichage.

-n Supprime les numéros de commande lors de l'affichage.

-r Inverse l'ordre des commandes

-l Les commandes sont listées sur stdout.

Sinon l'éditeur donné par **ename** est invoqué sur un fichier contenant les commandes.Si **ename** n'est pas spécifié, utilise dans l'ordre, **FCEDIT** ou **EDITOR** (défaut : vi). Quand l'édition est terminée, les commandes éditées sont exécutées.

Dans la seconde forme, la commande est ré-exécutée après que chaque instance de **pat** ait été remplacée par **rep**. La commande est interprétée de la même manière que **first**. Un alias utle à utiliser avec ça est **r="fr -s"**, donc en taper **r cc**, lance la dernière commande commençant avec **cc** et taper **r** re-exécute la dernière commande.

Si la première forme est utilisée, la valeur de retour est 0 sauf si une option invalide est rencontrée, ou **first** ou **last** spécifient des lignes d'historique hors plage. Si l'option **-e** est fournis, la valeur de retour est la valeur de la dernière commande exécutée ou une erreur si une erreur s'est produite avec le fichiers de commandes temporaire. Si la seconde forme est utilisée, le code de retour est celle de la commande ré-exécutée, sauf si la commande ne spécifie pas une ligne d'historique valide.

fg [jobspec] Met jobspec en tâche courante. Si jobspec n'est pas présent, la notion shell de job courant est utilisé. La valeur de retour est celle de la commande placée en foreground, ou une erreur si le contrôle de job est désactivé ou si jobspec ne spécifie pas de job valide ou un job qui a été lancé sans contrôle de job.

getopts optstring name [args] Utilisé par les procédures shell pour parser les paramètres positionnels. **optstring** contient les caractères d'option à reconnaitre ; si un caractère est suivi par une virgule, l'option est supposé avoir un argument, qui devrait être séparé par un espace. Les caractères **,** et **?** ne peuvent pas être utilisés comme caractère d'option. chaque fois qu'il est invoqué, **getopts** place la prochaine option dans la variable chaîne **name**, et index le prochaine argument à traiter dans la variable **OPTIND**. **OPTIND** est initialisée à 1 chaque fois que le shell ou un script shel est invoqué. Quand une option nécessite un argument, **getopts** place ect argument dans la variable **OPTARG**. Le shell ne réinitialise pas **OPTIND** automatiquement.

Quand la fin des options est rencontrée, **getopts** se termine avec une valeur de retour supérieur à 0. **OPTIND** est définis à l'index du premier argument non-option, et name est définis à **?**. **getopts** parse normalement les paramètres positionnels, mais si plus d'un argument est donné en argument, **getopts** les parse à la place.

getopts peut reporter les erreurs de 2 manières. Si le premier caractère de **optstring** est une virgule, une erreur silencieuse est utilisée. Dans une opération normale, les messages de diagnostique sont affichés quand une option invalide ou manquante est rencontrée. Si la variable **OPTERR** est à 0, aucun messages d'erreur ne sera affiché, même si le premier caractère de **optstring** n'est pas une virgule.

Si une option invalide et trouvée, **getopts** place un **?** dans **name** et, si non silencieux, affiche un message d'erreur et réinitialise **OPTARG**. Si **getopts** est silencieux, le caractère option trouvé est placé dans **OPTARG** et aucun message n'est affiché.

Si un argument requis n'est pas trouvé, et non silencieux, un **?** est placé dans **name**, **OPTARG** est réinitialisé, et un message d'erreur est affiché. Si silencieux, **:** est placé dans **name** ,t **OPTARG** est définis au caractère trouvé.

getopts retourne vrai si une option, spécifiée ou non spécifiée, est trouvée. Retourne false si la fin des options est rencontrée ou une erreur s'est produite.

hash [-lr] [-p filename] [-dt] [name] Chaque fois que **hash** est invoqué, le chemin complet de la commande est déterminée en recherchant les répertoires dans **PATH** et mémorisé. Tout chemin précédemment mémorisé est supprimé. **-p** ne recherche pas les répertoires, name est utilisé comme chemin complet. **-r** oubli tous les emplacements mémorisés. **-d** oublie l'emplacement mémorisé pour chaque nom. **-t** affiche le chemin pour chaque nom fournis. **-l** affiche la sortie dans un format réutilisable. Le statut de retour est true sauf si name n'est pas trouvé, ou une option est invalide.

help [-dms] [pattern] Affiche des information d'aide sur les commandes intégrées. Si pattern est spécifié, donne une aide détaillée sur toutes les commandes qui matchent ce pattern ; sinon affiche l'aide pour toutes les commandes intégrées. Le code de retour est 0 sauf si aucune commande ne matche pattern.

-d Affiche une description courte de chaque pattern

-m Affiche la description de chaque pattern dans un format style manpage

-s Affiche uniquement le synopsis pour chaque pattern

history [n]

history -c

history -d offset

history -anrw [filename]

history -p arg [arg ...]

history -s arg [arg ...] Sans options, affiche la liste numérotée d'historique de commande. Les lignes listées avec un * ont été modifiées. Un argument de **n** liste uniquement les dernières n lignes. Si la variable shell **HISTTIMEFORMAT** est définie et non null, elle est utilisé comme chaîne de formatage pour strftime(3) pour afficher le timestamp associé avec chaque entrée d'historique affichée. Si **filename** est fournis, il est utilisé comme nom de fichier d'historique ; sinon, la valeur de **HISTFILE** est utilisé.

-c Effache la liste d'historique

-d offset Supprime l'entrée d'historique à la position donnée

-a Ajoute les nouvelles lignes d'historique (les lignes entrées depuis le début de la session bash courante) au fichier d'historique.

-n Lit les lignes d'historique pas encore lues du fichier d'historique dans la liste d'historique courante.

-r Lit le contenu du fichier d'historique et l'ajoute à la liste d'historique courante.

-w Écrit le liste d'historique courante dans le fichier d'historique, en écrasant le contenu du fichier d'historique

-p Effectue une substitution d'historique sur les arguments fournis et affiche le résultat. Ne stocke pas le résultat dans la liste d'historique. Chaque argument doit être quoté pour désactiver l'expansion d'historique normal.

-s Stocke les arguments dans la liste d'historique en une seule entrée. La dernière commande dans la liste d'historique est supprimée avant que les arguments soient ajoutés.

Si la variable **HISTTIMEFORMAT** est définie, les informations de timestamp associés avec chaque entrée d'historique est écrit dans le fichier d'historique, marqués avec le caractère de commentaire d'historique. Quand le fichier d'historique est lu, les lignes commançant avec ce caractère suivis immédiatement par un chiffre sont interprétés comme timestamp pour la ligne d'historique précédente. La valeur de retour est 0 sauf si une option invalide est rencontrée, une erreur s'est produite en lisant ou écrivant dans le fichier d'historique, un offset invalide ou l'expansion d'historique fournis avec -p échoue.

jobs [-lnprs] [jobspec ...]

jobs -x command [args ...] La première forme liste les jobs actifs. Si jobspec est donné, la sortie est restreinte aux informations sur ce job. Le code de retour est 0 sauf si une option invalide est rencontrée ou un jobspec invalide est fournis.

-l Liste les ID des processus

-n Affiche des informations uniquement sur les jobs qui ont changés de status depuis la dernière notification de status reçu par l'utilisateur.

-p Liste uniquement les process ID des process group leader des jobs.

-r Affiche uniquement les jobs lancés

-s Affiche uniquement les jobs stoppés

-x Remplace tout jobspec trouvé dans la commande ou les arguments avec les process group ID correspondant, et exécute la commandes, retournant son codes de sortie.

kill [-s sigspec | -n signum | -sigspec] [pid | jobspec] ...

kill -l [sigspec | exit_status] Envoie le signal **sigspec** ou **signum** aux processus nommés par **pid** ou **jobspec**. Si **sigspec** n'est pas présent, envoie **SIGTERM**. **-l** liste les noms des signaux. Si un des argument est donné avec -l, liste les signaux correspondant. Le code de sortie de -l est un nombre décrivant soit le numéro du signal, ou le code de sortie d'un process terminé par un signal. Retourne vrai si au moins un signal a été envoyé avec succès, ou faux si une erreur s'est produite ou une option invalide est rencontrée.

let arg [arg ...] Chaque argument est une expression arithmétique à évaluer. Si le dernier argument évalue à 0, let retourne -1 ; 0 sinon.

local [option] [name[=value] ...] Permet de créer des variables locales. Les options sont les mêmes que celles acceptées par **declare**. Sans opérandes, local affiche la liste des variables locales. Retourne 0 sauf si local est utilisé en dehors d'un fonction, un nom est invalide, ou name est une variable lecture seule.

logout Quitte le login shell

mapfile [-n count] [-O origin] [-s count] [-t] [-u fd] [-C callback] [-c quantum] [array]

readarray [-n count] [-O origin] [-s count] [-t] [-u fd] [-C callback] [-c quantum] [array] Lis les lignes depuis l'entrée standard dans le tableau indexé **array**, ou depuis le descripteur de fichier **fd** si -u est fournis. La variable **MAPFILE** est le tableau par défaut.

-n Copie au maximum **count** ligne. Si count vaut 0, toutes les lignes sont copiées.

-O Commence à assigner le tableau là l'index **origin**. Défaut : 0

-s Ne tient pas compte des **count** premières lignes lues

-t Supprime les newline de fin de chaque ligne lues

-u Lit les lignes depuis le descripteur de fichier **fd** au lieu de l'entrée standard

-C Évalue **callback** chaque fois que **quantum** lignes sont lues.

-c Le **quantum**. Spécifie le nombre de lignes lues entre chaque appel **callback**

Si **-C** est spécifié sans **-c**, le quantum par défaut est 5000. Quand **callback** est évalué, il reçoit l'index du prochain élément du tableau à assigner et la ligne à assigner à cette élément comme argument. **callback** est évalué après que la ligne soit lue, mais avant l'assignation. Si l'origine n'est pas fournis, **mapfile** efface le tableau avant de l'assigner. **mapfile** retourne un succès sauf si une option invalide est rencontrée ou un argument option est fournis, le tableau est invalide ou non assignable, ou si le tableau n'est pas un tableau indexé.

popd [-n] [+n] [-n] Supprime les entrées de la pile de répertoires. Sans arguments, supprime le haut de la pile, et effectue un **cd** dans le nouveau répertoire top. Si la commande réussie, un **dirs** est effectué et le code de retour est 0. popd retourne false si une option invalide est rencontrée, la pile de répertoire est vide, une entrée de pile n'existe pas ou le changement de répertoire échoue.

-n Supprime le changement de répertoire lors de la suppression des répertoires dans la pile, seul la pile est manipulée.

+n Supprime la n-ième entrée dans la liste, en comptant de gauche à droite comme affiché par **dirs**, en commençant à 0

-n Supprime la n-ième entrée dans la liste, en comptant de droite à gauche comme affiché par **dirs**, en commençant à 0

printf [-v var] format [arguments] Écrit les argument formatés en utilisant le format spécifié. L'option **-v** assigne la sortie à la variable au lieu de l'afficher sur stdout. Le format est une chaîne de caractères qui contient 3 types d'objects : Des caractères normaux qui sont simplements copiés sur la sortie, des séquences de caractères échappés, et des spécification de format, chacun affichant l'argument suivant. En plus des spécification de format de printf(1), printf interprète les extensions suivantes :

%b Étend les séquences échappées dans l'argument correspondant, excepté que **\c** termine la sortie, **\'**, **\"** et **\?** ne sont pas supprimés, et les échappement octal commençant avec **\0** peuvent contenir jusqu'à 4 chiffres.

%q Affiche l'argument correspondant dans un format réutilisable en entrée.

%(datefmt)T Affiche la chaîne de date résultante en utilisant **datefmt** comme format de chaîne pour **strftime(3)**. L'argument correspondant est un entier représentant le nombre de seconde depuis l'epoch. 2 arguments spéciaux peuvent être utilisés : -1 représente le temps courant, et -2 représente l'heure d'invocation du shell. Sans arguments, utilise -1.

Les arguments pour des spécifieurs de format non-chaîne sont traités comme des constantes C, excepté qu'un signe + ou - est permis, et si ce caractère est un quote simple ou double, la valeur est la valeur ASCII du caractère suivant. Le format est réutilisé si nécessaire pour consommer tous les arguments. Si le format requière plus d'arguments que fournis, les spécifications restantes se comporte comme si une valeur 0 ou une chaîne null avait été fournis. La valeur de retour est 0 sauf en cas d'erreur.

pushd [-n] [+n] [-n]

pushd [-n] [dir] Ajoute un répertoire au-dessus de la pile de répertoire, ou fait tourner la pile pour que le répertoire de travail courant soit en haut de la pile. Sans arguments, échange les 2 répertoires du haut de la pile et retourne 0, sauf si la pile est vide. Si la première forme est utilisée, pushd retourne 0 sauf si cd échoue. Avec la forme suivante, pushd retourne 0 sauf si la pile est vide, en élément de la pile n'existe pas, ou le changement de répertoire échoue.

-n Ne change pas de répertoire, n'agit que sur la pile.

+n Tourne la pile pour que le n-ième répertoire (en comptant de gauche à droite comme affiché par dirs) soit en haut de la pile.

-n Tourne la pile pour que le n-ième répertoire (en comptant de droite à gauche comme affiché par dirs) soit en haut de la pile.

dir Ajoute le répertoire à la pile, puis effectue un cd sur ce répertoire.

pwd [-LP] Affiche le chemin absolue du répertoire de travail courant. Ce chemin est affiché sans liens symbolique si **-P** est fournis, ou si l'option **-o physical** est activée. L'option **-L** permet d'afficher les chemin avec des liens symboliques. Le status de retour est 0 sauf si une erreur s'est produite en lisant le répertoire courant, ou une option invalide est fournie.

read [-ers] [-a aname] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...] Une ligne est lue depuis l'entrée standard, ou depuis le descripteur de fichier fournis, et le premier mot est assignée au premier nom, le second au deuxième, etc, le dernier nom contient tous les mots restants. S'il à moins de mots que de nom, les noms en trop sont vides. **IFS** est utilisé pour séparer les mots en utilisant les mêmes règles que le shell pour les expansions. Le **** peut être utilisé pour supprimer toute signification spéciale du prochain caractère lu.

-a aname Les mots sont assignés aux indices séquentiels du tableau aname, en commençant à 0. aname est ré-initialisé avant d'assigner les nouvelles valeurs. les autres noms sont ignorés.

-d delim Le premier caractère est utilisé pour terminer la ligne d'entrée, au lieu d'une nouvelle ligne.

-e Si l'entrée standard vient du terminal, readline est utilisé pour obtenir la ligne. readline utilise les paramètre d'édition courants.

-i text Si readline est utilisé pour lire la ligne, le texte est placé dans le tampon d'édition avant que l'édition commence.

-n nchars retourne après que nchars caractères aient été lus, mais honore les délimiteurs

-N nchars retourne après que nchars caractères aient été lus, sauf si EOF est rencontré. N'honore pas les délimiteurs.

-p prompt Affiche le prompt sur l'erreur standard, sans newline final, avant de tenter de lire l'entrée. Le prompt est affiché uniquement si l'entrée provient du terminal.

-r les \ n'agissent pas comme caractère d'échappement.

-s Mode silencieux. Si l'entrée vient du terminal, les caractère ne sont pas répétés.

-t timeout délai en seconde avant que read ne termine et retourne une erreur si l'entrée n'est pas complète. timeout peut être un nombre décimal avec une portion fractionnelle. Effectif uniquement si c'est l'entrée standard, un pipe, ou un fichier spécial ; il n'a pas d'effet en lisant un fichier régulier. Si le timeout est atteind, la ligne d'entrée partielle est sauvée dans lu variable name. Si timeout est 0, retourne immédiatement, sans tenter de lire une donnée. Le statut de sortie est 0 si l'entrée est disponible sur le descripteur de fichier, Un statut de sortie de 128 pour un timeout atteind.

-u fd Lis l'entrée depuis le descripteur donné

Si aucun nom n'est donné, la ligne lue est assignée à la variable **REPLY**. Le code de retour est 0, sauf si EOF est rencontré, le timeout est atteind, une erreur lors de l'assignememt de variable ou un descripteur de fichier est invalide.

readonly [-aAf] [-p] [name[=word] ...] Les noms donnés sont marqués en lecture seule ; les valeurs de ces noms ne peuvent plus être changés. Le statut de retour est 0 sauf si une option invalide est rencontrée, un des noms n'est pas une variable shell, ou -f est fournis avec une variable qui n'est pas une fonction.

-f Les fonctions correspondantes sont également marquées.

-a restreint les variables aux tableaux indexés

-A Restreint es variables aux tableaux associatif. Si -a est également fournis, -A a précédence.

-p Si aucun nom n'est fournis ou si -p est spécifié, liste toutes les variables lecture seule dans un format réutilisable.

return [n] Stop l'exécution d'une fonction et retourne la valeur spécifiée par n à son appelant. Si n est omis, retourne le status de la dernière commande exécutée. Si return est utilisé en dehors d'une fonction, mais durant l'exécution d'un script par la commande **.** (**source**), le shell stop l'exécution du script. Si n est fournis, la valeur de retour sont les 8bits de poid faible de n. Le status de retour est non-zéro si return est founis avec un argument non-numérique, ou est utilisé hors d'une fonction ou d'un script lancé par **.**. Toute commande associée avec le trap **RETURN** est exécuté avant de terminer la fonction ou le script.

set [–abefhkmnptuvxBCEHPT] [-o option-name] [arg ...]

set [+abefhkmnptuvxBCEHPT] [+o option-name] [arg ...] Sans options, le nom et la valeur de chaque variable shell sont affichés dans une format réutilisable. Les variables lecture seul ne peuvent pas être ré-initialisées. En mode posix, seul les variables shell sont listées.a sortie est triée en accord avec la locale courante. Quand des options sont spécifiées, elles définissent ou indéfinissent des attributs du shel. Tous arguments restant après le traitement des options sont traités comme valeurs pour les paramètres positionnels et sont assignés dans l'ordre, de **$1**, **$2**, **...**, **$n**.

-a Marque automatiquement les variables en fonctions qui sont modifiés ou créés en export.

-b Reporte le statut des jobs de fond terminés immédiatement, et non avant le prochain prompt primaire. C'est effectif uniquement si le contrôle de job est actif.

-e Quitte immédiatement si un pipe (que peut consister d'une simple commande), une liste, ou une commande composée fait partie de la liste de commande immédiatement suivant un **while** ou **until**, fait partie du test suivant un **if** ou **elif**, fait partie d'une commande exécutée dans un **&&** ou || excepté la commande suivant le **&&** ou le || final, une commande dans un pipeline sauf le dernier, ou si la valeur de retour de la commande est inversée avec **!**. Si une commande composée autre qu'un sous-shell retourne un statut non-zéro à cause d'une commande échouée alors que **-e** était ignoré, le shell ne se termine pas. Un trap sur **ERR** est exécuté avant que le shell se termine. Cette option s'applique à l'environnement shell et chaque sous-environnement, et peut terminer les sous-shell avant d'exécuter toutes les commandes dans le sous-shell.

Si une commande composée ou une fonction shell est exécutée dans un contexte où **-e** est ignoré, aucune commande exécutée dans la commande composée ou la fonction ne sera affectée par le paramètre **-e**, même si **-e** est définis et qu'une commande retourne un statut d'échec. Si une commande composée ou une fonction shell définie **-e** durant l'exécution dans un contexte où **-e** est ignoré, ce paramètre n'aura aucun effet tant que la commande composée ou la commande contenant l'appel de la fonction ne soit terminée.

-f Désactive l'expansion de pathname

-h Mémorise l'emplacement des commandes comme si elles étaient recherchées pour exécution. Activé par défaut.

-k Tous les arguments dans la forme de déclaration d'assignement sont placés dans l'environnement pour une commande, pas seulement ceux qui précède le nom de la commande.

-m Mode monitor. le contrôle de job est activé. Cette option est activée par défaut pour les shells interactifs sur les systèmes qui le supporte. Tous les processus sont lancés dans un groupe de processus séparés. Quand un job de fond se termine, le shell affiche un ligne contenant son code de sortie.

-n Lit les commandes mais ne les exécute pas. Peut être utilisé pour vérifier les erreurs de syntaxe des scripts shell. Ignorés par les shells interactifs.

-o option-name Si option-name n'est pas fournis, les valeurs des options courantes sont affichées. si **+o** est fournis sans option-name, une série de commandes set pour recréer les paramètres courants est affiché sur stdous. Les options peuvent être les suivantes :

allexport Idem à **-a**

braceexpand Iden à **-B**

emacs Utilise une interface d'édition de ligne de commande de style emacs. Activé par défaut quand le shell est interactif, sauf si le shell est lancé avec l'option **–noediting**. Affecte également l'interface d'édition pour **read -e**

errexit Idem à **-e**

errtrace Idem à **-e**

functrace Idem à **-T**

hashall Idem à **-h**

histexpand Idem à **-H**

history Active l'historique de commandes. Actif par défaut pour les shells interactifs.

ignoreeof Idem à la commande shell **IGNOREEOF=10**

keyword Idem à **-k**

monitor Idem à **-m**

noclobber Idem à **-C**

noexec Idem à **-n**

noglob Idem à **-f**

nolog Actuellement ignoré

notify Idem à **-b**

nounset Idem à **-u**

onecmd Idem à **-t**

physical Idem à **-P**

pipefail Si définis, la valeur de retour d'un pipeline est la valeur de la dernière commande avec un statut de sortie non-zéro, ou 0 si toutes les commandes ont réussies. Désactivé par défaut.

posix bash passe en mode posix

privileged Idem à **-p**

verbose Idem à **-v**

vi Utilise une interface d'édition de ligne de commande style vi. Affecte également l'interface d'édition utilisé pour **read -e**

xtrace Idem à **-x**

-P Passe en mode privilégié. Dans ce mode, les fichiers **ENV** et **BASH_ENV** ne sont pas traités, les fonctions shell n'héritent pas de l'environnement, les variables **SHELLOPTS**, **BASHOPTS**, **CDPATH** et **GLOBIGNORE** sont ignorées. Si le shell est lancé avec le user/group ID effectif différent du user/group id réel, et que l'option **-p** n'est pas fournie, le user id effectif est mis au user id réel. Si l'option **-p** est fournis au lancement, le user id effectif n'est pas réinitialisé. Désactiver cette option cause les user/group id effectifs à être définis comme user/group id réels.

-t Quitte après avoir lu et exécuté une commande.

-u Traite les variables et paramètres non-définis autre que les paramètres spéciaux @ et * comme des erreurs lors de l'expansion de paramètres. Si l'expansion est tenté sur un paramètre ou une variable non-définie, le shell affiche un message d'erreur, et, si non-interactif, quitte avec un code de retour non-zéro.

-v Affiche les lignes lues

-x Après chaque expansion d'une commande simple, d'une commande **for**, **case**, **select**, ou une commande **for** arithmétique, affiche la valeur étendue de **PS4**, suivi par la commande et ses arguments étendus.

-B Le shell effectue l'expansion d'accolade. Activé par défaut.

-C Si définis, bash n'écrase pas les fichiers existants avec **>**, **>&**, et **<>**. On peut toujours écraser les fichiers en créant des fichiers avec **>|** au lieu de **>**.

-E Si définis, un trap sur **ERR** est hérité par les fonctions, substitutions de commandes, et les commandes exécutées dans en environnement sous-shell. le trap **ERR** n'est normalement pas hérités dans de tels cas.

-H Active la subsitution d'historique de commande avec **!**. Activé par défaut pour les shells interactifs.

-P Si définis, le shell ne résoud pas les liens symboliques en exécutant les commandes telles que **cd**. Il utilise la structure de répertoire physique à la place. Par défaut, bash suit les liens.

-T Si définis, un trap sur **DEBUG** et **RETURN** est hérité par les fonctions, substitutions de commandes, et les commandes exécutées dans en environnement sous-shell. Ces trap ne sont normalement pas hérités dans de tels cas.

– Si aucun argument ne suit cette option, les paramètres positionnels sont indéfinis. Sinon, les paramètres positionnels sont définis avec les arguments, même si certain d'entre eux commencent par **-**.

- Signal la fin des options, tous les arguments restants sont assignés aux paramètres positionnels. **-x** et **-v** sont désactivés. S'il n'y a aucun argument, les paramètres positionnels restent inchangés

Les options sont désactivées par défaut, sauf mention. Utiliser **+** au lieu de **-** désactive l'option. Les options peuvent également être spécifiées à l'invocation du shell. Le jeu d'options courant peut être trouvé dans **$-**. Le code de retour est toujours vrai sauf si une option invalide est rencontrée.

shift [n] Les paramètres positionnels de n+1 ... sont renommés en $1 ... Les paramètres représentés par les nombres $# décroissant à **$#-n+1** sont indéfinis. **n** doit être un nombre non-négatif inférieur ou égal à **$#**. Si **n** vaut 0, aucun paramètre n'est changé. Si **n** n'est pas donné, il est assumé 1. Si n est supérieur à **$#**, les paramètres positionnels ne sont pas changés. Le status de retour est supérieur à 0 si n est supésieur à **$#** ou inférieur à 0.

shopt [-pqsu] [-o] [optname ...] Active/désactive les paramètres contrôlant le fonctionnement du shell. Ces paramètres peuvent être ceux listés ci-dessous, ou, si **-o** est donné, ceux disponible avec **set -o**. Sans options ou avec **-p**, affiche la liste de tous les paramètres définissable, indiquant s'il sont actif ou non. **-p** affiche dans un format réutilisable.

-s Active chaque optname

-u Désactive chaque optname

-q Mode silencieux. Le status de retour indique si optname est définis ou non. Si plusieurs optname sont fournis, retourne 0 si tous les optnames sont activés.

-o Restreins les valeurs de optname à celles définis pour la commande **set -o**

Si **-s** ou **-u** est utilisé sans optname, shopt affiche uniquement les options qui sont définies ou non. Sauf mention, les options de shopt sont désactivées par défaut. les options shopt sont :

autocd Un nom de commande qui est le nom d'un repertoire est exécuté comme si elle était un argument de cd. Uniquement pour les shells interactifs.

cdable_vars Un argument de la commande cd qui n'est pas un répertoire est assumé être le nom d'un variable contenant un répertoire.

cdspell Les erreurs mineur dans les arguments de la commande cd sont corrigés. Il s'agit des caractères transposés, un caractère manquant, un caractère en trop. Si une correction est trouvée, le nom du fichier corrigé est affiché, et la commande est traitée. Uniquement pour les shells interactifs.

checkhash Vérifie qu'une commande trouvée dans la table de hash existe avant de tenter de l'exécuter. Si un hash d'une commande n'est pas trouvée, une recherche normale est effectuée.

checkjobs Liste le statut des jobs lancés et stoppés avant de quitter un shell interactif. Si un job est lancé, la sortie est différée jusqu'à un nouveau exit. Le shell rapporte toujours si des jobs sont stoppés.

checkwinsize Vérifie toujours la taille de la fenêtre après chaque commande, et, si nécéssaire, met à jour **LINES** et **COLUMNS**.

cmdhist Tente de sauver toutes les lignes d'un commande sur plusieurs lignes dans la même entrée d'historique.

compat31 Bash change son comportement à la version 3.1 en respectant les arguments quotés de la commande conditionnelle **[[** de l'opérateur **=** et les comparaisons de chaîne spécifique au locale en utilisant les opérateurs **<** et **>** des commandes conditionnelles **[[**. Les versions de bash avant 4.1 utilisent le classement l'assemblage ASCII et strcmp(3) ; bash 4.1 et + utilisent la séquence d'assemblage du locale et strcoll(3).

compat32 Bash change son comportement à la version 3.2 en respectant les comparaisons de chaîne spécifique au locale en utilisant les opérateurs **<** et **>** de la commande conditionnelle **[[**.

compat40 Bash change son comportement à la version 4.0 en respectant les comparaisons de chaîne spécifique au locale en utilisant les opérateurs **<** et **>** de la commande conditionnelle **[[**. et l'effet de l'interruption d'un liste de commande. Bash 4.0 et + interrompt la liste comme si le shell reçoit l'interruption.

compat41 Bash, en mode posix, traite un simple quote dans une expansion de paramètre double-quoté comme un caractère spécial.

compat42 Bash ne traite par les chaînes de remplacement dans la substitution de motif de l'expansion de mot en utilisant la suppression de quote.

complete_fullquote Bash quote tous les métacaractères shell dans les noms de fichier et répertoire en effectuant les completions. Si non définis, bash supprime tous les métacaractères tel que **$** du jeu de caractères qui seront quotés dans les noms de fichier complétés quand ces métacaractères apparaissent dans le références de variable shell dans les mot à compléter. Cela signifie que le signe dollar dans les noms de variable qui s'étendent à des repertoires ne seront pas quotés. Cependant, un signe dollar apparaissant dans les noms de fichier ne seront pas quotés. Activé par défaut.

direxpand Remplace les noms des répertoires avec le résultat de l'expansion de mot lors des completions de pathname. Cange le contenu du tampon d'édition de readline. Non définis, bash tente de préserver ce qui a été tapé.

dirspell Tente de corriger les noms de répertoires durant la completion de mot si le nom du répertoire initial n'existe pas.

dotglob Inclus les noms de fichier commençant avec un **.** dans le résultat de l'expansion de pathname.

execfail Un shell non-interactif ne se terminera pas s'il ne peut pas exécuter le fichier spécifié en argument de la commande exec. Un shell interactif ne qui jamais dans ce cas.

expand_aliases Les alias sont étendus. Activé par défaut pour les shells interactifs

extdebug Utilisé pour les débuggeurs :

1. L'option **-F** de la commande declare affiche le nom du fichier source et le numéro de ligne correspondant à chaque nom de fonction fournis en argument.

2. Si la commande lancé par le trap **DEBUG** retourne une valeur non-zéro, la prochaine commande est sauté et non exécutée.

3. Si la commande lancé par le trap **DEBUG** retourne la valeur 2 et que le shell est exécuté dans une sous-routine (une fonction shell ou un script shell exécuté par . ou source), un appel à **return** est simulé.

4. **BASH_ARGC** et **BASH_ARGV** sont mis à jours.

5. Le tracing de fonction est activé : la substitution de commande, les fonctions shell, et les sous-shell invoqués avec **(** **commande)** héritent des traps **DEBUG** et **RETURN**.

6. Le tracing d'erreur est activé : la substitution de commande, les fonctions shell, et les sous-shell invoqués avec **(** **commande)** héritent du trap **ERR**.

extglob Les fonctionnalités de correspondance de motif étendu décrites dans Expansion de chemin sont activées

extquote **$'string'** et **$"string"** sont effectués dans les expansion ${parameter} enfermés dans les guillemets double. Activé par défaut.

failglob Les motif qui échouent les correspondance de nom de fichier créer une erreur d'expansion.

force_fignore Les suffixes spécifiés par **FIGNORE** forcent les mots à être ignorés lors de l'expansion de mot même si les mots ignorés sont les seules completions possible. Activé par défaut.

globasciiranges Les expressions de plage utilisés dans les expressions entre accolade de pattern matching fonctionnent comme dans la locale traditionnelle C lors des comparaisons. Donc **b** ne sera pas assemblé entre **A** et **B**, et les caractères ASCII minuscules et majuscule ne seront pas assemblé ensemble.

globstar le pattern ****** utilisé dans l'expansion de pathname va matcher tous les fichiers et 0 ou plusieurs répertoires et sous-répertoires. Si le pattern est suivi par un **/**, seul les répertoires et sous-répertoires matchent.

gnu_errfmt Les messages d'erreurs du shell sont écrits au format des messages d'erreurs standard GNU.

histappend La liste d'historique est ajoutée au fichier nommé par **HISTFILE** quand le shell quitte, au lieu d'écraser le fichier.

histreedit Si readline est utilisé, un utilisateur a l'opportunité de ré-éditer une substitution d'historique échouée.

histverify Si readline est utilisé, le résultat de la substitution d'historique n'est pas passé immédiatement au parser du shell. À la place, la ligne résultante est chargée dans le tampon d'édition de readline, permettant d'autres modifications.

hostcomplete Si readline est utilisé, bash tente d'effectuer une completion de nom d'hôte quant un mot contient @. Activé par défaut.

huponexit Tente d'envoyer **SIGHUP** à tous les jobs quand le login shell intéractif se termine.

interactive_comments Permet à un mot commençant avec **#** de forcer ce mot et tous les caractères restant sur la ligne à être ignoré dans le shell intéractif. Activé par défaut.

lastpipe Si le contrôles de job est actif, le shell lance la dernière commande d'un pipeline non exécutée en tâche de fond dans l'environnement du shell courant.

lithist Si l'option **cmdhist** est activé, les commandes multi-ligne sont sauvées dans l'historique avec le newline embraqué au lieu l'utiliser un **;**.

login_shell Le shell définis cette option s'il est lancé comme login shell. Cette valeur ne peut pas être changée.

mailwarn Si bash vérifie un fichier pour les mails reçus, le message "The mail in <mailfile> has been read" est affiché.

no_empty_cmd_completion Si readline est utilisé, bash ne tente pas de rechercher le **PATH** pour les completions possible.

nocaseglob Les matches de noms de fichier lors de l'expansion de pathname sont insensible à la casse.

nocasematch Les matches de pattern dans les commandes conditionnelles **case** et **[[** sont insensible à la casse.

nullglob Bash permet au pattern qui ne matchent aucun fichier de s'étendre à une chaîne nulle, au lieu d'eux-même.

progcomp Les fonctionnalité de completion programmable sont activés, Activé par défaut.

promptvars les chaînes de prompt sont sujet à l'expansion de paramètre, substitution de commande, expansion arithmétique, et suppression de quotes après avoir été étendus. Activé par défaut

restricted_shell Le shell est lancé en mode restreint. Cette valeur ne peut pas être changée.

shift_verbose la commande shift affiche un message d'erreur quand le compteur excède le nombre de paramètres positionnels.

sourcepath La commande source utilise **PATH** pour trouver le répertoire contenant le fichier fournis en argument. Activé par défaut.

xpg_echo La commande echo étend les séquences backslash-excape par défaut.

suspend [-f] Suspend l'exécution du shell jusqu'à ce qu'il reçoive un signal **SIGCONT**. Un login shell ne peut pas être suspendu ; l'option **-f** peut être utilisé pour forcer cette suspension. Le status de retour est 0 sauf si le shell est un login shel et que **-f** n'est pas fournis, ou si le contrôle de job n'est pas actif.

test expr

[expr] Retourne un status de 0 ou 1 en fonction de l'évaluation de l'expression conditionnelles expr. Chaque opérateur et opérande doit être un argument séparé. Les expressions sont composées des primaires décrites sous expressions conditionnelles. test n'accèpte pas d'options, ni n'accèpte et ignore un argument **–**. Les expressions peuvent être combinées en utilisant les opérateurs suivant, listés par ordre de précédence décroissant. L'évaluation dépend du nombre d'arguments. La précédence d'opérateur est utilisé quand il y au moins 5 arguments.

! expr Vrai si expr est faux

(expr) Retourne la valeur de expr. Peut être utilisé pour outrepasser les précédences

expr1 -a expr2 Vrai si expr1 et expr2 sont vrai

expr1 -o expr2 Vrai si expr1 ou expr2 est vrai

test, [évaluent les expressions conditionnelles en utilisant un jeu de règles basées sur le nombre d'arguments :

0 arguments L'expression est fausse

1 argument L'expression est vrai si et seulement si l'argument n'est pas null.

2 arguments Si le premier argument est **!**, l'expression est vrai si et seulement si le second argument est null. Si le premier argument est un des opérateurs conditionnel unaire, l'expression est vrai si le test unaire est vrai. Si le premier argument n'est pas un opérateur conditionnel unaire valide, l'expression est fausse.

3 arguments Les conditions suivantes sont appliquées dans l'ordre listé. Si le second argument est un des opérateurs conditionnel binaires, le résultat de l'expression est le résultat du test binaire une utilisant le premier et le troisième argument comme opérandes. Les opérateur **-a** et **-o** sont considérés comme des opérateurs binaire quand il y a 3 arguments. Si le premier argument est **!**, la valeur est la négation du test à 2 arguments. Si le premier argument est exactement **(** est le troisième exactement **)**, le resultat est le test 1 argument du second argument. Sinon l'expression est fausse.

4 arguments Si le premier argument est **!**, le résultat est la négation de l'expression 3 arguments. Sinon l'expression est parsée est évaluée en accord avec la précédent est utilisant les règles listées plus haut.

5 ou + arguments L'expression est parsée et évaluée en accord avec les précédence en utilisant les règles ci-dessus.

Utilisé avec **test** ou **[**, les opérateurs **<** et **>** trient lexicographiquement en utilisant l'ordre ASCII.

times Affiche le temps système et utilisateur accumulé pour le shell et pour les processus lancés depuis le shell. Le status de retour est 0.

trap [-lp] [[arg] sigspec ...] La commande arg est lue et exécutée quand le shell reçoit le signal sigspec. Si arg est absent (et il y a un simple sigspec) ou **-**, chaque signal spécifié est réinitialisé à sa disposition initiale (la valeur avant d'entrée dans le shell). Si arg est une chaîne null le signal spécifié par chaque sigspec est ignoré par le shell et par la commande qu'il invoque. Si arg n'est pas présent et **-p** est fournis, les commandes de trap associé avec chaque sigspec sont affichés. Si aucun argument n'est fournis ou si seulement **-p** est donné, trap affiche la liste des commandes associées avec chaque signal. L'option **-l** affiche une liste de noms de signaux et leurs numéros correspondant. Chaque sigspec est soit un nom de signal, ou son numéro.

Si un sigspec est **EXIT (0)** la commande arg est exécutée à la sortie du shel. Si un sigspec est **DEBUG**, la commande arg est exécutée avant toute commande simple, for, case, select, et avant la première commande exécutée dans une fonction shell. Si un sigspec est **RETURN**, La commande arg est exécutée chaque fois qu'une fonction shell ou un script est exécutée avec **.** ou **source**.

Si un sigspec est **ERR**, la commande arg est exécutée si un pipeline (qui peut consister d'une commande simple), une liste, ou une commande composée retourne un statut de sortie non-zéro, sujet aux conditions suivante. le trap **ERR** n'est pas exécuté si la commande échouée fait partie de la liste de commande immédiatement après un **while** ou **until**, d'un test dans une déclaration **if**, partie d'une commande exécutée dans un **&&** ou || excepté la commande finale, une commande dans un pipe sauf la dernière, ou si la valeur de retour de la commande est inversée par **!**. Ce sont les même conditions pour l'option **errexit**.

Les signaux ignorés lors de l'entrée dans le shell ne peuvent plus être trappés ou réinitialisés. Les signaux trappés qui ne sont pas ignorés sont ré-initialisés à leur valeur d'origine dans un sous-shel ou un environnement sous-shell quand il est créé. Le statut de retour est false si un sigspec est invalide, sinon trap retourne true.

type [-aftpP] name [name ...] Sans options, indique comment chaque nom serait interprété si utilisé comme nom de commande.

-t Affiche une chaîne qui est un parmis alias, keyword, function, builtin ou file. Si le nom n'est pas trouvé, rien n'est affiché et retourne false.

-p Retourne soit le nom du fichier, ou rien si **type -t name** ne retournerai pas de fichier.

-P Force une recherche **PATH** pour chaque nom, même si **type -t name** ne retournerai pas de fichier. Si une commande est hashée, **-p** et **-P** affichent la valeur hashée, qui n'est pas nécessairement le fichier qui apparaît en premier dans **PATH**.

-a Affiche tous les emplacements qui contiennent un exécutable nommé name. Celà inclus les aliases et les fonctions, si est seulement si l'option **-p** n'est pas utilisée. La table des hash n'est pas consultée avec **-a**.

-f Supprime la recherche de fonctions, comme avec la commande **command**. type retourne true si tous les arguments sont trouvés, false sinon.

ulimit [-HSTabcdefilmnpqrstuvx [limit]] Fournis un contrôle sur les ressources disponibles au shell et aux processus lancés ce dernier. Les options **-H** et **-S** spécifient que les limites hard ou soft sont définies pour la ressource donnée. Une limite hard ne peut pas être augmentée par un utilisateur non-root, une limite soft peut être augmentée jusqu'à la valeur hard. Si ni **-H** ni **-S** ne sont spécifiés, les 2 limites sont définies. La valeur de limit peut être un nombre dans l'unité spécifée pour la ressource ou au valeur spéciale **hard**, **soft** ou **unlimited**, qui spécifie la limite hard courante, limite soft courant, et aucune limite, respectivement. Si limit est omis, la valeur courante de la limite soft de la ressource est affichée, sauf si **-H** est donné. Quand plus d'une ressource est spécifié, le nom de la limite et l'unité sont affichés avant la valeur. Les autres options sont interprétés comme suit :

-a Reporte toutes les limites courante

-b maximum socket buffer size

-c maximum size of core files created

-d maximum size of a process's data segment

-e maximum scheduling priority ("nice")

-f maximum size of files written by the shell and its children

-i maximum number of pending signals

-l maximum size that may be locked into memory

-m maximum resident set size

-n maximum number of open file descriptors

-p pipe size in 512-byte blocks (ne peut pas être définis)

-q maximum number of bytes in POSIX message queues

-r maximum real-time scheduling priority

-s maximum stack size

-t maximum amount of cpu time in seconds

-u maximum number of processes available to a single user

-v maximum amount of virtual memory available to the shell

-x maximum number of file locks

-T maximum number of threads

Si limit est donné sans **-a**, limit est la nouvelle valeur pour la ressource spécifiée. Si aucune option n'est donnée, assume **-f**. Les valeurs sont en incrément de 1024 octets, excepté pour **-t** qui est en secondes, **-p** qui est en unité de blocks de 512 octets, et **-T**, **-b**, **-n** et **-u**. Le statut de retour est 0 sauf si une option ou un argument invalide est rencontré, ou une erreur se produit en définissant une nouvelle limite.

umask [-p] [-S] [mode] Définis le masque de création de fichier de l'utilisateur. Si mode commence avec un chiffre, il est interprété en octal, sinon il est interprété similairement à la commande chmod(1). Si mode est omis, la valeur umask courant est affichée. L'option **-S** affiche le masque sous la forme symbolique, sinon l'affiche en octal. L'option **-p**, sans mode, affiche dans un format réutilisable. Le statut de retour est 0 si le mode a été changé avec succès ou si aucun mode n'a été fournis, false sinon.

unalias [-a] [name ...] Supprime chaque nom fournis de la liste des alias définis. Si **-a** est spécifié, toutes les définitions d'alias sont supprimées. La valeur de retour est true sauf si un nom fournis n'est pas un alias.

unset [-fv] [-n] [name ...] Pour chaque nom, supprime la variable ou la fonction correspondante. Si **-v**, chaque nom réfère à une variable shell, et est supprimée. Les variable lecture seul ne peuvent pas être supprimées. Si **-f**, chaque nom réfère à une fonction et est supprimée. Si **-n** et name est une variable avec l'attribut nameref, le nom sera indéfinis au lieu de sa référence. **-n** n'a pas d'effet si **-f** est fournis. Sans options, chaque nom réfère à une variable, s'il n'y a pas de variable correspondant au nom, toute fonction avec ce nom est supprimée. Chaque variable ou fonction est supprimée de l'environnement passée aux commandes suivantes. Si un de **COMP_WORDBREAKS, RANDOM, SECONDS, LINENO, HISTCMD, FUNCNAME, GROUPS, DIRSTACK** est indéfinis, elles perdent leur propriétés spéciales, même si elle sont ré-initialisée ultérieurement. Le code de sortie est true sauf si un nom est en lecture seul.

wait [-n] [n ...] Attend pour chaque processus enfant spécifié et retourne son code de sortie. Chaque **n** peut être un ID de processus ou une spécification de job. Si un jobspec est donné, attend pour tous les processus dans ce pipeline de job. Si **n** n'est pas fournis, attend les processus enfant courant actif et retourne 0. Si **-n**, attend la fin de tous les jobs et retourne son code de sortie. Si **n** spécifie un processus non-existant, retourne 127, sinon retourne de statut de sortie du dernier job attendu.

Shell restreint

Si bash est lancé avec le nom **rbash**, ou **-r** est fournis à l'invocation, le shell devient restreint. Ce shell est utilisé pour définir un environnement plus contrôlé qu'un shell standard. Les fonctionnalités suivantes sont interdite ou non exécutées :

- Changer de répertoire avec cd
- Définir ou indéfinir les valeurs de **SHELL**, **PATH**, **ENV** ou **BASH_END**
- Spécifier des noms de commandes contenant **/**
- Spécifier un nom de fichier contenant **/** à la commande **.**
- Spécifier un nom de fichier contenan **/** en argument de l'option **-p**
- Importer des définitions de fonction dans l'environnement au démarrage
- Parcourir la valeur de **SHELLOPTS** dans l'environnement au démarrage
- Rediriger la sortie en utilisant **>, >|, <>, >&, &>,** et **»**
- Utiliser la commande exec pour remplace le shell avec une autre commande
- Ajouter ou supprimer les commandes intégrée avec les options **-f** ou **-d** de la commande enable
- Utiliser la commande enable pour activer des commandes intégrées

- Spécifier l'option **-p** à la commande command
- Désactiver le mode restreint avec **set +r** ou **set +o restricted**

Ces restrictions sont active une fois les fichiers de démarrage lus. Quand une commande est un script shell, **rbash** désactive toute restriction dans le shell créé pour exécuter le script.

Fichiers

/bin/bash L'exécutable bash

/etc/profile Le fichier d'initialisation système, exécuté par les login shell

/etc/bash.bashrc Le fichier d'initialisation système, exécuté par les shell interactifs

/etc/bash.bash.logout Le fichier d'initialisation système, exécuté quand le login shell se termine

~/.bash_profile Fichier d'initialisation personnel, exécuté par les login shell

~/.bashrc Fichier d'initialisation personnel, exécuté par les shell interactifs

~/.bash_logout Fichier d'initialisation personnel, exécuté quand le login shell se termine

~/.inputrc Fichier d'initialisation readline personnel.