BASH

Section : Manuel de l'utilisateur Linux (1)
Mise à jour de la version anglaise : 28 septembre 2006
Index Menu principal

NOM

bash - Interpréteur de commandes GNU Bourne-Again SHell

SYNOPSIS

bash [options] [fichier]

COPYRIGHT

DESCRIPTION

Bash est un interpréteur de commande (*shell*) compatible **sh** qui exécute les commandes lues depuis l'entrée standard, ou depuis un fichier. **Bash** incorpore également des fonctionnalités provenant des interpréteurs *Korn* et *C*-shell (**ksh** et **csh**).

Bash est conçu pour être conforme à la partie shell et outils des spécifications IEEE POSIX (Norme IEEE 1003.1). **Bash peut être configuré pour être conforme aux spécifications POSIX** par défaut.

OPTIONS

En plus des caractères d'options documentés dans la description de la commande interne **set**, **bash** accepte les arguments suivants lorsqu'on l'invoque :

-c *chaîne*
: Si l'argument **-c** est présent, les commandes sont interprétées depuis la *chaîne* fournie. S'il existe d'autres arguments après la *chaîne*, ils sont transmis comme paramètres positionnels, en commençant par **$0**.

-i
: Si l'option **-i** est présente, l'interpréteur est *interactif*.

-l
: Faire que **bash** se comporte comme un shell de connexion (voir **INVOCATION** plus loin).

-r
: Si l'option **-r** est présente, l'interpréteur est *restreint* (voir **SHELLS RESTREINTS** plus bas).

-s
: Si l'option **-s** est présente, ou s'il n'y a plus d'arguments sur la ligne de commande, après le traitement des options, l'interprétation se fera depuis l'entrée standard. Cette option permet de remplir les paramètres positionnels tout en invoquant un shell interactif.

-D
: Une liste de toutes les chaînes entre « " » et précédées de **$** est affichée sur la sortie standard. Ce sont les chaînes qui sont sujettes à une traduction quand la localisation n'est ni celle du **C**, ni celle des spécifications **POSIX**. Cela nécessite l'option **-n** ; aucune commande ne sera exécutée.

[-+]O [*shopt_option***]**
: *shopt_option* est une des options shell acceptées par la commande interne **shopt** (voir **COMMANDES INTERNES DU SHELL** plus loin). Si *shopt_option* est présente, **-O** positionnera la valeur de cette option ; **+0** la désactivera. Si *shopt_option* n'est pas fournie, les noms et valeurs des options shell acceptées par **shopt** sont affichés sur la sortie standard. Si l'option d'invocation est **+0**, la sortie sera affichée dans un format réutilisable en entrée.

--
: Deux caractères **--** permettent d'indiquer la fin des options, et empêchent le traitement des arguments restants. Ces derniers sont alors traités comme des noms de fichiers et des paramètres. **-** est équivalent à **--**.

Bash reconnaît également plusieurs options multi-caractères. Ces options doivent, pour être reconnues, apparaître sur la ligne de commande avant les options mono-caractères.

--debugger
S'arranger pour que le profil du débogueur soit exécuté avant que le shell démarre. Basculer en mode de débogage étendu (voir la description de l'option **extdebug** de la commande interne **shopt** plus loin) et traçage de fonctions shell (voir la description de l'option **-o functrace** de la commande interne **set** plus loin).
--dump-po-strings
Équivalent à **-D**, mais la sortie est dans le format des fichiers **po** (objets portables) de l'outil GNU *gettext*.
--dump-strings
Équivalent à **-D**.
--help
Affiche un message d'aide sur la sortie standard et se termine avec succès.
--init-file *fichier*
--rcfile *fichier*
Exécuter les commandes contenues dans *fichier* plutôt que celles du classique fichier d'initialisation personnel *~/.bashrc* si le shell est interactif (voir **INVOCATION** plus loin).
--login
Équivalent à **-l**.
--noediting
Ne pas utiliser la bibliothèque GNU *readline* pour la lecture des lignes de commande, lorsque le shell est interactif.
--noprofile
Ne lire ni le fichier de configuration générale */etc/profile* ni les fichiers d'initialisation personnels *~/.bash_profile*, *~/.bash_login*, ou *~/.profile*. Par défaut, **bash** lit ces fichiers lorsqu'il est invoqué comme shell de connexion (voir le paragraphe **INVOCATION** plus bas).
--norc
Ne pas lire ni n'exécuter le fichier de configuration personnalisée *~/.bashrc* lorsque le shell est interactif. Cette option est activée par défaut lorsque **bash** est invoqué sous le nom **sh**.
--posix
Aligner le comportement de **bash** sur la norme POSIX, en ce qui concerne les options dont l'action par défaut diffère de ce standard (*mode posix*).
--restricted
Le shell devient restreint (voir le paragraphe **SHELLS RESTREINTS** plus bas).
--verbose
Équivalent à **-v**
--version
Afficher le numéro de version de **bash** sur la sortie standard et terminer avec succès.

ARGUMENTS

S'il reste des arguments sur la ligne de commande après l'analyse des options, et si ni l'option **-c**, ni l'option **-s** ne sont présentes, le premier argument est supposé être le nom du fichier dans lequel lire les commandes. Lorsque **bash** est invoqué de cette manière, **$0** contient le nom du fichier, et les autres paramètres positionnels contiennent les arguments restants. **Bash** lit et exécute les commandes depuis ce fichier, puis se termine. Le code de retour de **bash** est celui de la dernière commande exécutée dans le fichier script. Si aucune commande n'a été exécutée, le code de retour est 0. Une tentative est faite tout d'abord pour trouver le fichier dans le répertoire courant, et si aucun fichier n'est trouvé, le shell cherche le script dans les répertoires contenus dans **PATH**.

INVOCATION

Un shell est dit *de connexion* (Ndt : login shell) si le premier caractère de son argument numéro zéro est un **-**, ou s'il est invoqué avec l'option **-login**.

Un shell est *interactif* si son entrée standard et sa sortie d'erreur sont toutes deux connectées à un terminal (déterminé par la fonction *isatty*(3)), ou s'il est invoqué avec l'option **-i**. Le paramètre **PS1** est positionné, et le paramètre **$-** contient la lettre **i** si **bash** est interactif, ce qui permet à un script ou à un fichier de démarrage de vérifier l'état du shell.

Le paragraphe suivant décrit comment **bash** exécute ses fichiers d'initialisation. Si l'un de ces fichiers existe mais n'est pas accessible en lecture, **bash** signale une erreur. Les tildes sont remplacées par des noms de fichiers comme décrit dans le paragraphe **Développement du Tilde** de la section **Développement.**

Lorsque **bash** est lancé comme shell de connexion interactif, ou comme shell non-interactif avec l'option **--login**, il lit et exécute tout d'abord les commandes se trouvant dans le fichier */etc/profile* s'il existe. Après lecture de ce fichier, il recherche *~/.bash_profile*, *~/.bash_login*, et *~/.profile*, dans cet ordre, et exécute les commandes se trouvant dans le premier fichier existant et accessible en lecture. L'option **--noprofile** peut être utilisée au démarrage du shell pour empêcher ce comportement.

Lorsqu'un shell de connexion se termine, **bash** lit et exécute les commandes du fichier *~/.bash_logout*, s'il existe.

Quand un shell interactif démarre sans être un shell de connexion, **bash** lit et exécute les commandes se trouvant dans *~/.bashrc* s'il existe. Ce comportement peut être inhibé à l'aide de l'option **--norc**. L'option **--rcfile** *fichier* forcera **bash** à exécuter les commandes dans *fichier* plutôt que dans *~/.bashrc*.

Quand **bash** est démarré de manière non-interactive, pour lancer un script shell par exemple, il recherche la variable **BASH_ENV** dans l'environnement, développe son contenu si elle existe, et considère cette valeur comme le nom d'un fichier à lire et exécuter. **Bash** se comporte comme si la commande suivante se trouvait en début de script :

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

mais la valeur de la variable **PATH** n'est pas utilisée pour rechercher le fichier.

Si **bash** est invoqué sous le nom **sh**, il essaye d'imiter le comportement de démarrage des versions historiques de **sh** autant que possible, tout en restant conforme aux spécifications POSIX. Lorsqu'il est invoqué comme shell de connexion interactif (ou non-interactif avec l'option **--login**), il essaye d'abord d'exécuter les commandes se trouvant dans */etc/profile* et *~/.profile*, dans cet ordre. L'option **--noprofile** peut toujours être utilisée pour désactiver ce comportement. Quand il est invoqué en tant que shell interactif sous le nom **sh**, **bash** consulte la variable **ENV**, développe sa valeur si elle est définie et utilise le résultat en tant que nom de fichier à lire et exécuter. Comme un shell invoqué sous le nom **sh** n'essaye pas d'exécuter d'autre fichier de démarrage, l'option **--rcfile** n'a aucun effet. Un shell non interactif invoqué sous le nom **sh** ne lit aucun autre fichier d'initialisation. Quand il est invoqué sous le nom **sh**, **bash** entre en mode *posix* après avoir lu les fichiers d'initialisation.

Quand **bash** est invoqué en mode *posix*, avec l'option **--posix** sur la ligne de commande, il suit ce standard en ce qui concerne les fichiers de démarrage. Dans ce cas, la variable **ENV** est développée, et le fichier qui en résulte est exécuté. On n'exécute pas d'autre fichier d'initialisation.

Bash tente de déterminer s'il est exécuté par le démon lançant les shells distants (généralement appelé *rshd*). Si **bash** se rend compte qu'il est exécuté par *rshd*, il lit et exécute les commandes de *~/.bashrc* si ce fichier existe et est lisible. Il n'a pas ce comportement lorsqu'il est invoqué sous le nom **sh**. L'option **--norc** peut être utilisé pour interdire ce comportement, et l'option **--rcfile** permet de forcer l'utilisation d'un autre fichier, mais *rshd* n'utilise généralement pas ces options et ne permet pas non plus leur utilisation.

Si le shell est lancé avec un identifiant effectif d'utilisateur (ou de groupe) différent de l'identifiant réel, et si l'option **-p** n'est pas utilisée, aucun fichier d'initialisation n'est lu, les fonctions du shell ne sont pas importées depuis l'environnement, la variable **SHELLOPTS** est ignorée si elle apparaît dans l'environnement, et l'identifiant effectif de l'utilisateur (du groupe) est remplacé par l'ID réel. Si l'option **-p** est fournie, le démarrage est le même mais l'ID effectif n'est pas modifié.

DÉFINITIONS

Les définitions suivantes sont utilisées tout au long de ce document.

blanc
: Une espace ou une tabulation.

mot
: une séquence de caractères considérée comme une unité élémentaire par le shell. On parle également de **token** (jeton).

nom
: un *mot* ne contenant que des caractères alphanumériques ou le soulignement (underscore), commençant par une lettre ou un soulignement. On s'y réfère également sous le terme **identificateur**.

méta-caractère
: Un caractère qui, non protégé, sépare les mots. Un de ceux-ci :

| & ; ( ) < > espace tabulation

opérateur de contrôle
: Un *jeton* ayant une fonction de contrôle. L'un des symboles suivants :

|| & && ; ;; ( ) | <retour-chariot>

MOTS RÉSERVÉS

Les *mots réservés* ont une signification spéciale pour le shell. Les mots suivants sont réservés et interprétés par le shell lorsqu'ils ne sont pas protégés, et s'il s'agit soit du premier mot d'une commande simple, soit du troisième mot d'une commande **case** ou **for** (voir le paragraphe **GRAMMAIRE DU SHELL** ci-dessous).

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

GRAMMAIRE DU SHELL

Commandes simples

Une *commande simple* est une séquence d'affectations de variables facultative, suivie de mots séparés par des *blancs* et des redirections, et terminée par un *opérateur de contrôle*. Le premier mot indique la commande à exécuter et est passée comme étant l'argument zéro. Les mots suivants sont transmis en argument à cette commande.

La valeur de retour d'une *commande simple* est son code de sortie, ou 128+*n* si la commande a été interrompue par le signal *n*.

Pipelines

Un *pipeline* est une séquence d'une ou plusieurs commandes séparées par le caractère |. Le format d'un pipeline est :

[**time** [**-p**]] [!] *commande_1* [| *commande_2* ...]

La sortie standard de la *commande_1* est connectée, via un tuyau (Ndt : pipe) à l'entrée standard de la *commande_2*. Cette connexion est établie avant toute redirection indiquée dans une commande elle-même (voir le paragraphe **REDIRECTION** plus bas).

Le code de retour d'un pipeline est le code de sortie de la dernière commande, à moins que l'option **pipefail** ne soit active. Si cette option est active, le code de retour du pipeline est la valeur de la dernière commande (la plus à droite) à sortir avec un code non nul, ou zéro si toutes les commandes s'achèvent avec succès. Si le mot réservé **!** précède un pipeline, le code de sortie de celui-ci sera la négation logique du code de retour tel qu'on l'a décrit précédemment. L'interpréteur attend la fin de toutes les commandes du pipeline avant de renvoyer une valeur.

Si le mot réservé **time** précède le pipeline, les temps passés par le programme en mode utilisateur et système sont indiqués quand le pipeline se termine. L'option **-p** modifie le format de sortie pour celui spécifié par POSIX. La variable **TIMEFORMAT** peut être affectée avec une chaîne de format indiquant comment les informations horaires doivent être affichées ; consultez la description de **TIMEFORMAT** dans **Variables du Shell** plus bas.

Chaque commande du pipeline est exécutée comme un processus indépendant (c'est-à-dire dans un sous-shell).

Listes

Une *liste* est une séquence d'un ou plusieurs pipelines séparés par l'un des opérateurs **;**, **&**, **&&** ou ||, et terminée éventuellement par **;**, **&** ou **<retour-chariot>**.

Dans cette liste d'opérateurs, **&&** et || ont une précédence identique, suivis par **;** et **&**, qui ont également une précédence identique.

Une séquence d'un ou plusieurs sauts de ligne peut apparaître dans une *liste* à la place d'un point-virgule pour délimiter les commandes.

Si une commande se termine par l'opérateur de contrôle **&**, l'interpréteur l'exécute en *arrière-plan*, dans un sous-shell. L'interpréteur n'attend pas que la commande se termine et retourne un code 0. Les commandes séparées par un **;** sont exécutées successivement, l'interpréteur attend que chaque commande se termine avant de lancer la suivante. Le code de retour est celui de la dernière commande exécutée.

Les opérateurs de contrôle **&&** et || indiquent respectivement une liste liée par un ET, et une liste liée par un OU. Une liste ET a la forme

commande_1 **&&** *commande_2*

commande_2 est exécutée si, et seulement si, *commande_1* renvoie un code de retour nul.

Une liste OU a la forme

commande_1 || *commande_2*

commande_2 est exécutée si, et seulement si *commande_1* renvoie un code de retour non-nul. La valeur de retour des listes ET et OU est celle de la dernière commande exécutée dans la liste.

Commandes composées

Une *commande composée* est l'une des constructions suivantes :

(*liste*)
liste est exécutée dans un sous-shell (voir **ENVIRONNEMENT D'EXÉCUTION DES COMMANDES** plus loin). Les affectations de variables, et les commandes internes qui affectent l'environnement de l'interpréteur n'ont pas d'effet une fois que la commande se termine. Le code de retour est celui de la *liste*.

{ *liste;* }
liste est simplement exécutée avec l'environnement du shell en cours. *liste* doit se terminer par un caractère fin-de-ligne ou un point-virgule. Cette construction est connue sous le nom de *commandes groupées*. Le code de retour est celui de la *liste*. Veuillez noter que contrairement aux méta-caractères **(** et **)**, **{** et **}** sont des *mots réservés* qui ne doivent apparaître que là où un mot réservé peut être reconnu. Puisqu'ils ne provoqueront pas un coupage de mot, ils doivent être séparés de la *liste* par une espace.

((*expression*))
L'*expression* est évaluée selon les règles décrites plus bas au paragraphe **ÉVALUATION**ARITHMÉTIQUE**.** Si la valeur arithmétique de l'expression est non-nulle, le code renvoyé est zéro ; sinon 1 est renvoyé. Cela est strictement identique à **let "***expression*".

[[*expression* **]]**
Renvoie 1 ou 0 selon la valeur de la condition *expression*. Les expressions sont composées d'éléments primaires décrits dans **EXPRESSIONS**CONDITIONNELLES**.** Le coupage des mots et l'expansion des chemins ne sont pas réalisés sur les portions entre **[[** et **]]**; l'expansion des tildes, des paramètres, des variable, des expressions arithmétiques, la substitution des commandes et des processus, ainsi que la disparition des apostrophes sont réalisés. Les opérateurs conditionnels tels que **-f** ne doivent pas être cotés afin d'être reconnus comme primaires. Conditional operators such as **-f** must be unquoted to be recognized

Quand les opérateurs **==** et **!=** sont utilisés, la chaîne placée à droite de l'opérateur est considérée comme étant un motif et est recherchée selon les règles décrites dans **Motifs génériques**. Si l'option shell **nocasematch** est active, la mise en correspondance s'effectue sans tenir compte de la casse (différence majuscule/minuscule). La valeur renvoyée est 0 si les chaînes correspondent (**==**) (ou respectivement ne correspondent pas (**!=**)), et 1 sinon. Toute partie du motif peut être protégée avec des apostrophes pour forcer sa comparaison en tant que chaîne (sans développement).

Un opérateur binaire supplémentaire, **=~**, est disponible, avec la même priorité que **==** et **!=**. Lorsqu'il est utilisé, la chaîne à droite de l'opérateur est considérée comme une expression régulière étendue et est mise en correspondance en conséquence (comme avec **regex**(3)). La valeur renvoyée est 0 si la chaîne correspond au motif, et 1 si elle ne correspond pas. Si l'expression régulière n'est pas syntaxiquement correcte, la valeur de retour de l'expression conditionnelle est 2. Si l'option shell **nocasematch** est active, la mise en correspondance s'effectue sans tenir compte de la casse. Les sous-chaînes mise en correspondance avec des sous-expressions entre parenthèses dans l'expression rationnelle sont enregistrées dans la variable tableau **BASH_REMATCH**. L'élément d'index 0 de BASH_REMATCH est la partie de la chaîne correspondant à la l'expression rationnelle complète. L'élément d'index *n* de BASH_REMATCH est la partie de la chaîne correspondant à la *n*ème sous-expression entre paranthèses.

Les expressions peuvent être combinées en utilisant les opérateurs suivants, par ordre décroissant de priorité :

(*expression* **)**
Retourne la valeur de l'*expression*. Cela peut être utilisé pour outrepasser la priorité normale des opérateurs.

! *expression*
Vraie si *expression* est fausse.

expression1 **&&** *expression2*
Vraie si *expression1* et *expression2* sont toutes les deux vraies.
expression1 || *expression2* Vraie si *expression1* ou *expression2* est vraie.

Les opérateurs **&&** et || n'évaluent pas *expression2* si la valeur de *expression1* suffit à déterminer le code de retour de l'expression conditionnelle entière.

for *nom* [**in** *mot*] ; **do** *liste* ; **done**
La liste de mots à la suite de **in** est développée, créant une liste d'éléments. La variable *nom* prend successivement la valeur de chacun des éléments, et *liste* est exécutée à chaque fois. Si **in** *mot* est omis, la commande **for** exécute la *liste* une fois pour chacun des paramètres positionnels ayant une valeur (voir le paragraphe **PARAMÈTRES** plus bas). Le code de retour est celui de la dernière commande exécutée. Si le développement de ce qui suit **in** est une liste vide, aucune commande n'est exécutée et 0 est renvoyé.

select *nom* [**in** *mot*] ; **do** *liste* ; **done**
La liste de mots à la suite de **in** est développée, créant une liste d'éléments. L'ensemble des mots développés est imprimé sur la sortie d'erreur standard, chacun précédé par un nombre. Si **in** *mot* est omis, les paramètres positionnels sont imprimés (voir le paragraphe **PARAMÈTRES** plus bas). Le symbole d'accueil **PS3** est affiché, et une ligne est lue depuis l'entrée standard. Si la ligne est constituée d'un nombre correspondant à l'un des mots affichés, la variable *nom* est remplie avec ce mot. Si la ligne est vide, les

mots et le symbole d'accueil sont affichés à nouveau. Si une fin de fichier (EOF) est lue, la commande se termine. Pour toutes les autres valeurs, la variable *nom* est vidée. La ligne lue est stockée dans la variable **REPLY**. La *liste* est exécutée après chaque sélection, jusqu'à ce qu'une commande **break** soit atteinte. Le code de retour de **select** est celui de la dernière commande exécutée dans la *liste*, ou zéro si aucune commande n'est exécutée.

case *mot* **in** [*motif* [| *motif*]

Une commande **case** commence d'abord par développer le *mot*, puis essaye de le mettre en correspondance successivement avec chacun des *motifs* en utilisant les mêmes règles que pour les noms de fichiers (voir le paragraphe **Développement des noms de fichiers** plus bas). Le *mot* est développé en utilisant le développement du tilde, le développement des paramètres et des variables, la substitution arithmétique, la substitution de commande, la substitution de processus et la suppression d'apostrophes. Chaque *motif* examiné est développé en utilisant le développement du tilde, le développement des paramètres et des variables, la substitution arithmétique, la substitution de commande et la substitution de processus. Si l'option shell **nocasematch** est active, la mise en correspondance s'effectue sans tenir compte de la casse. Quand une correspondance est trouvée, la *liste* associée est exécutée. Dès qu'un motif correct a été trouvé, il n'y a plus d'autre essais. Le code retour est zéro si aucun motif ne correspond, sinon il s'agit du code de la dernière commande exécutée dans la *liste*.

if *liste* ; **then** *liste* ; [**elif** *liste* ; **then** *liste* ;] ... [**else** *liste* ;] **fi**

La *liste* du **if** est exécutée. Si son code de retour est nul, la *liste* du **then** est exécutée. Sinon, chacune des *listes* des **elif** est exécutée successivement, et si un code de retour est nul, la *liste* du **then** associé est exécutée, et la commande se termine. En dernier ressort, la *liste* du **else** est exécutée. Le code de retour est celui de la dernière commande exécutée, ou zéro si aucune condition n'a été vérifiée.

while *liste* ; **do** *liste* ; **done**

until *liste* ; **do** *liste* ; **done**

La commande **while** répète la *liste* du **do** tant que la dernière commande de la *liste* du **while** renvoie un code de retour nul. La commande **until** agit de même manière, sauf que le test est négatif, et la *liste* du **do** est exécutée tant que la *liste* du **until** renvoie un code non-nul. Le code de retour des commandes **while** et **until** est celui de la dernière commande exécutée dans la liste **do**, ou zéro si aucune commande n'a été exécutée.

Définitions des fonctions shell

Une fonction shell est un objet qui est appelé comme une commande normale et qui exécute une commande composée avec un nouveau jeu de paramètres positionnels. Les fonctions shell sont déclarées de la manière suivante :

[**function**] *nom* () *commande-composée* [*redirection*]

Ceci définit une fonction possédant le *nom* mentionné. Le mot réservé **function** est optionnel. S'il est fourni, les paranthèses sont optionnelles. Le *corps* de la fonction est la *commande-composée* entre { et }, mais peut être toute commande décrite dans le paragraphe **Commandes composées** plus haut. La *commande-composée* est exécutée chaque fois que *nom* est spécifié comme le nom d'une commande normale. Toutes les redirections (voir **REDIRECTION** plus bas) spécifiées lorsqu'une fonction est définie sont effectuées lorsque la fonction est exécutée. Le code de retour d'une définition de fonction est zéro à moins qu'il y ait une erreur de syntaxe ou qu'une fonction en lecture seule, de même nom, existe déjà. Lorsque la fonction est exécutée, Le code de retour est celui de la dernière commande exécutée dans le corps de la fonction. (voir le paragraphe **FONCTIONS** plus bas)

COMMENTAIRES

Dans un shell non-interactif, ou dans un shell interactif avec l'option **-o interactive-comments** activée par la commande interne **shopt** (voir **COMMANDES INTERNES DU SHELL** plus bas), un mot commençant par **#** permettra d'ignorer tous les caractères restants sur la ligne. Un shell interactif sans l'option **interactive-comments** n'autorise pas les commentaires. L'option **interactive_comments** est activée par défaut dans les shells interactifs.

PROTECTION

Les *protections* (quoting) permettent de forcer l'interpréteur à ignorer la signification spéciale de certains caractères ou mots. Les protections peuvent être utilisées pour empêcher le traitement des caractères spéciaux, éviter la reconnaissance des mots-réservés ou empêcher le développement des paramètres.

Tous les *méta-caractères* mentionnés dans le paragraphe **DÉFINITIONS** plus haut ont des significations spéciales pour le shell, et doivent être protégés pour garder leur propre valeur.

Lorsqu'on utilise l'expansion de l'historique des commandes (voir **EXPANSION DE L'HISTORIQUE** plus bas), le caractère de l'*expansion de l'historique* , généralement **!**, doit être mis entre apostrophes pour éviter l'expansion de

l'historique. Il y a trois mécanismes de protection : le caractère d'échappement, les apostrophes (quote) et les guillemets (double-quote).

Un caractère barre oblique inverse (****), quand il n'est pas protégé, représente le *caractère d'échappement*. Il préserve la valeur littérale du caractère qui le suit, à l'exception du <retour-chariot>. Si une séquence ****<retour-chariot> apparaît, et si la barre oblique inverse n'est pas protégée, l'ensemble ****<retour-chariot> est considéré comme une continuation de ligne (autrement dit, il est ignoré).

Encadrer des caractères entre des apostrophes simples préserve la valeur littérale de chacun des caractères. Une apostrophe ne peut pas être placée entre deux apostrophes, même si elle est précédée d'une barre oblique inverse.

Encadrer des caractères entre des guillemets préserve la valeur littérale de chacun des caractères sauf **$**, `, et ****, et **!** quand l'expansion de l'historique est activée. Les caractères **$** et ` conservent leurs significations spéciales, même entre guillemets. La barre oblique inverse ne conserve sa signification que lorsqu'il est suivi par **$**, `, ", ****, ou **<fin-de-ligne>**. Un guillemet peut être protégé entre deux guillemets, à condition de le faire précéder par une barre oblique inverse. Si elle est activée, l'expansion de l'historique sera effectuée à moins qu'un **!** entre guillemets ne soit protégé par une barre oblique inverse. La barre oblique inverse précédant le **!** n'est pas supprimée

Les caractères spéciaux ***** et @ ont des significations spéciales lorsqu'ils se trouvent entre guillemets (voir le paragraphe **PARAMÈTRES** ci-dessous).

Les mots de la forme **$**aq*chaîne*aq sont traités différemment. Le mot est développé en *chaîne* avec les séquences d'échappement remplacées par les séquences standards du C ANSI:

\a
: alerte (cloche alias bip)

\b
: effacement arrière

\e
: un caractère escape

\f
: fin de page

\n
: fin de ligne

\r
: retour chariot

\t
: tabulation horizontale

\v
: tabulation verticale

****
: contre oblique

\aq
: apostrophe

*****nnn*
: le caractère 8 bits dont la valeur en octal est *nnn* (un à trois chiffres)

\x*HH*
: le caractère 8 bits dont la valeur en hexadécimal est *HH* (un à deux caractères hexadécimaux)

\c*x*
: un contrôle-*x* character

Le résultat après traduction est protégé par des apostrophes comme si le symbole dollar n'avait pas été présent.

Une chaîne entre guillemets précédée d'un symbole dollar (**$**) sera traduite selon la localisation en vigueur. Si la locale courante est **C** ou **POSIX**, le symbole dollar est ignoré. Si la chaîne a subi une traduction ou des remplacements, le résultat est protégé par des guillemets.

PARAMÈTRES

Un *paramètre* est une entité permettant de stocker des valeurs, comme les variables dans les langages de programmation courants. Il peut se présenter sous forme d'un *nom*, d'un nombre, ou d'un des caractères spéciaux décrits plus bas, dans **Paramètres Spéciaux**. Au niveau du shell, une *variable* est un paramètre muni d'un *nom*. Une variable possède une *valeur* et zéro ou plusieurs *attributs*. Les attributs sont affectés en utilisant la commande interne **declare** (voir **declare** plus loin dans la section **COMMANDES**INTERNES**DU**SHELL**).**

Un paramètre existe dès qu'on lui attribue une valeur. Une chaîne vide est une valeur valide. Une fois qu'une variable existe, elle ne peut être détruite qu'en utilisant la commande interne **unset** (voir **COMMANDES INTERNES DU SHELL** plus bas).

Une *variable* peut recevoir une valeur par une affectation de la forme

nom=[*valeur*]

Si aucune *valeur* n'est indiquée, la variable reçoit une chaîne vide. Toute *valeur* est soumise aux principes de développement du tilde, des paramètres et des variables, de la substitution de commandes, de l'évaluation arithmétique, et de la suppression des protections (voir **EXPANSION** plus bas). Si une variable a son attribut **-i**(integer) activé, alors la *valeur* est soumise à l'évaluation arithmétique, même si la syntaxe $((...)) n'apparaît pas (voir **ÉVALUATION ARITHMÉTIQUE** plus bas). Les mots ne sont pas tronqués, sauf avec la séquence **"$@"** comme cela est expliqué dans le paragraphe **Paramètres Spéciaux** ci-dessous. Le développement des noms de fichiers n'est pas effectué. Les instructions d'affectation peuvent également apparaître en tant qu'argument des commandes internes suivantes : **alias**, **declare**, **typeset**, **export**, **readonly** et **local.**

Dans le contexte où une instruction d'affectation affecte une valeur à une variable shell ou à un index de tableau, l'opérateur **+=** peut être utilisé pour ajouter ou additionner à la précédente valeur de la variable. Lorsque l'opérateur **+=** est appliqué à une variable dont l'attribut « integer » a été positionné, *valeur* est évaluée comme une expression arithmétique et additionnée à la valeur actuelle de la variable qui est également évaluée. Lorsque l'opérateur **+=** est appliqué à une variable tableau en utilisant l'affectation composée (voir **Tableaux** plus loin), la valeur de la variable n'est pas supprimée (comme elle l'est lorsqu'on utilise =), et les nouvelles valeurs sont ajoutées au tableau, en commençant à l'index maximum du tableau plus un. Lorsque l'opérateur est appliqué à une variable de type chaîne, *valeur* est développée et ajoutée à la valeur de la variable.

Paramètres Positionnels

Un *paramètre positionnel* est un paramètre indiqué par un ou plusieurs chiffres (à l'exception du chiffre 0 seul). Les paramètres positionnels sont remplis avec les arguments du shell lors de son invocation. Ils peuvent être modifiés avec la commande interne **set**. On ne peut pas utiliser d'affectation pour modifier le contenu d'un paramètre positionnel. Les paramètres positionnels sont temporairement modifiés lors de l'exécution d'une fonction du shell (voir le paragraphe **FONCTIONS** plus bas).

Un paramètre positionnel constitué de plusieurs chiffres doit être encadré par des accolades lors de son développement (voir **EXPANSION** plus bas).

Paramètres Spéciaux

Il existe plusieurs paramètres ayant un comportement particulier. Ces paramètres peuvent uniquement être consultés, on ne peut pas les modifier.

*****
: Se développe en l'ensemble des paramètres positionnels, commençant à 1. Quand le développement se produit entre des guillemets, ***** se transforme en un seul mot constitué de la valeur de tous les paramètres positionnels séparés par le premier caractère de la variable spéciale **IFS**. Ceci signifie que « **$*** » est équivalent à « **$1***c***$2***c***...** », dans laquelle *c* est le premier caractère de la valeur de la variable **IFS**. Si **IFS** est inexistante, les paramètres sont séparés par des espaces. Si **IFS** est nulle, les paramètres sont juxtaposés sans séparation.

@
: Se développe en l'ensemble des paramètres positionnels, commençant à 1. Quand le développement a lieu entre guillemets, chaque paramètre se transforme en un mot distinct. Ceci signifie que « **$@** » est équivalent à « **$1** » « **$2** » ... Quand le développement entre guillemets survient dans un mot, le développement du premier paramètre est joint à la première partie du mot d'origine et le développement du dernier paramètre est joint à la dernière partie du mot d'origine. Quand il n'y a pas de paramètres positionnels, « **$@** » et **$@** sont simplement éliminés.

#
: Correspond au nombre de paramètres positionnels, en base décimale.

?
: Fournit le code de retour du dernier pipeline exécuté à l'avant-plan.

-
: Est remplacé par la liste des options du shell indiquées durant l'invocation, configurées avec la commande interne **set** ou positionnées par le shell lui-même (comme avec l'option **-i**).

$
: Se transforme en PID du shell. Dans un sous-shell (), il se transforme en PID du shell, et non pas du sous-shell.

!
: Se transforme en PID de la commande (asynchrone) exécutée en arrière-plan le plus récemment.

0
: Se développe pour donner le nom du shell ou du script. Ce paramètre est rempli lors de l'initialisation de l'interpréteur. Si **bash** est invoqué avec un fichier de commandes, **$0** correspond au nom de ce fichier. Si **bash** est lancé avec l'option **-c**, alors **$0** contient le premier argument, s'il y en a un, après la chaîne de commandes à exécuter. Sinon, ce paramètre contient le chemin d'accès utilisé pour invoquer **bash**, comme il l'a reçu en argument zéro.

_
Au lancement du shell, contient le chemin d'accès absolu utilisé pour invoquer le shell ou du script en cours d'exécution tel qu'il est passé dans l'environnement ou dans la liste d'arguments. Devient le dernier argument de la commande précédente, après développement. Correspond aussi au chemin d'accès complet de chaque commande exécutée, et se retrouve dans l'environnement exporté à cette commande. Lors de la vérification de l'arrivée de courrier, contient le nom du fichier de courrier en cours de vérification [Ndt : généralement */var/mail/nom_de_l_utilisateur*].

Variables du Shell

Les variables suivantes sont remplies par l'interpréteur de commandes :

BASH
Se développe en chemin d'accès complet à l'instance courante de **bash**.

BASH_ARGC
Une variable de type tableau dont les valeurs sont le nombre de paramètres de chaque trame de la pile d'appel d'exécution du shell courant. Le nombre de paramètres de la sous-routine courante (fonction shell ou script exécuté avec **.** ou **source**) est en haut de la pile. Lorsqu'une sous-routine est exécutée, le nombre de paramètres passés est poussé (Ndt : push) dans **BASH_ARGC**. Le shell positionne **BASH_ARGC** seulement dans le mode de débogage étendu (voir la description de l'option **extdebug** de la commande interne **shopt** plus loin).

BASH_ARGV
Une variable de type tableau contenant tous les paramètres de la pile d'appel d'exécution du shell courant. Le dernier paramètre de la dernière sous-routine se situe en haut de la pile ; le premier paramètre du premier appel en bas. Lorsqu'une sous-routine est exécutée, les paramètres fournis sont poussés dans **BASH_ARGV**. Le shell positionne **BASH_ARGV** seulement dans le mode de débogage étendu (voir la description de l'option **extdebug** de la commande interne **shopt** plus loin).

BASH_COMMAND
Contient la commande en cours d'exécution ou sur le point d'être exécutée, à moins que le shell n'exécute une commande résultant d'un « trap », auquel cas il s'agit de la commande s'exécutant au moment du « trap ».

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

BASH_LINENO
Une variable de type tableau dont les membres sont les numéros de ligne des fichiers sources correspondant à chacun des membres de **FUNCNAME**. **${BASH_LINENO[***$i***]}** est le numéro de la ligne dans le fichier source où **${FUNCNAME[***$i***]}** a été appelé. Le nom du fichier source correspondant est **${BASH_SOURCE[***$i***]}**. Utilisez **LINENO** pour obtenir le numéro de ligne courant.

BASH_REMATCH
Une variable de type tableau dont les membres sont affectés avec l'opérateur binaire **=~** dans la commande conditionnelle **[[** L'élément d'index 0 est la partie de la chaîne correspondant à la l'expression rationnelle complète. L'élément d'index *n* est la partie de la chaîne correspondant à la *n*ème sous-expression entre paranthèses. Cette variable est en lecture seule.

BASH_SOURCE
Une variable de type tableau dont les membres sont les noms des fichiers source correspondants aux éléments de la variable tableau **FUNCNAME**.

BASH_SUBSHELL
Incrémenté chaque fois qu'un sous-shell ou un environnement sous-shell est engendré. La valeur initiale est 0.

BASH_VERSINFO
Une variable de type tableau protégée en écriture dont les membres représentent la version de cette instance de **bash**. Cette valeur est renseignée de la manière suivante :

BASH_VERSINFO[0**]**
Le numéro majeur de la version (*release*).
BASH_VERSINFO[1**]**
Le numéro mineur de la version (*version*).
BASH_VERSINFO[2**]**
Le niveau de correctif (Ndt patch).
BASH_VERSINFO[3**]**
Le numéro de compilation.
BASH_VERSINFO[4**]**
Le statut de cette version (p. ex., *beta1*).
BASH_VERSINFO[R5**]**
La valeur de **MACHTYPE**.

BASH_VERSION
Correspond au numéro de version de cette instance de **bash**.

COMP_CWORD
Un index vers **${COMP_WORDS}** du mot contenant la position courant du curseur. Cette variable n'est disponible que dans les fonctions shell invoquées par les fonctionnalites de complétion programmable (voir **Complétion programmable** plus loin).

COMP_LINE
la ligne de commande courante. Cette variable n'est disponible que dans les fonctions shell invoquées par les fonctionnalites de complétion programmable (voir **Complétion programmable** plus loin).

COMP_POINT
L'index de la position courante du curseur relatif au début de la commande courante. Si la position courante du curseur est à la fin de la commande courante, la valeur de cette variable est égale à **${#COMP_LINE}**. Cette variable n'est disponible que dans les fonctions shell et les commandes externes invoquées par les fonctionnalites de complétion programmable (voir **Complétion programmable** plus loin).

COMP_WORDBREAKS
le jeu de caractères que la bibliothèque Readline considèrent comme séparateurs de mots lors de la complétion de mots. Si **COMP_WORDBREAKS** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

COMP_WORDS
Une variable de type tableau (voir **Tableaux** plus loin) consistant en des mots individuels de la ligne de commande courante. Les mots sont fractionnés sur les caractères de remplacement du shell comme l'analyseur du shell les séparerait. Cette variable n'est disponible que dans les fonctions shell invoquées par les fonctionnalites de complétion programmable (voir **Complétion programmable** plus loin).

DIRSTACK
Une variable de type tableau (voir **Tableaux** plus bas) représentant le contenu courant de la pile de répertoires. Les répertoires apparaissent dans la pile dans l'ordre dans lequel la commande interne **dirs** les affiche. Il est possible de modifier directement cette variable mais les commandes internes **pushd** et **popd** doivent être utilisées pour ajouter et enlever des répertoires. Modifier cette variable ne modifiera pas le répertoire courant. Si **DIRSTACK** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

EUID
Contient l'UID effectif de l'utilisateur, initialisé au démarrage du shell. Cette variable est en lecture seule.

FUNCNAME
Une variable de type tableau contenant le nom de toutes les fonctions shell actuellemnt dans la pile d'appel d'exécution. L'élément d'index 0 est le nom de n'importe laquelle des fonctions shell en cours d'exécution. L'élément le plus bas est « main ». Cette variable n'existe que si une fonction shell est en cours d'exécution. Les affectations de **FUNCNAME** n'ont aucun effet et renvoient un code d'erreur (Ndt : cette variable est protégée en écriture). Si **FUNCNAME** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

GROUPS
Une variable de type tableau contenant la liste des groupes dont l'utilisateur est membre. Les affectations de **GROUPS** n'ont aucun effet et renvoient un code d'erreur (Ndt : cette variable est protégée en écriture). Si **GROUPS** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

HISTCMD
Le numéro d'historique, ou le rang dans la liste d'historique, de la commande en cours. Si **HISTCMD** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

HOSTNAME
Automatiquement affectée avec le nom de l'ordinateur.

HOSTTYPE
Automatiquement remplie avec une chaîne qui décrit de manière unique le type de machine sur laquelle **bash** s'exécute. La valeur par défaut dépend du système.

LINENO
Chaque fois que ce paramètre est référencé, le shell le remplace par un nombre décimal représentant le numéro de la ligne actuelle (commençant à 1), au sein du script ou de la fonction. Hors d'un script ou d'une fonction, la valeur n'a pas nécessairement de sens. Si **LINENO** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

MACHTYPE
Automatiquement affectée avec une chaîne décrivant le type du système sur lequel **bash** s'exécute, dans le format standard de GNU *processeur-compagnie-système*. La valeur par défaut dépend du système.

OLDPWD
Le répertoire de travail précédent, configuré par la commande **cd**.

OPTARG
La valeur du dernier argument d'option traité par la commande interne **getopts** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus loin).

OPTIND
Le rang du prochain argument à traiter avec la commande **getopts** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas).

OSTYPE
Automatiquement remplie avec une chaîne qui décrit le système d'exploitation sur lequel **bash** s'exécute. La valeur par défaut dépend du système.

PIPESTATUS

Une variable de type tableau (voir **Tableaux** plus loin) contenant une liste des codes de retour des derniers processus exécutés en avant-plan (éventuellement une seule commande).

PPID
L'ID du processus parent du shell. Cette variable est en lecture seule.

PWD
Le répertoire de travail en cours, configuré par la commande **cd**.

RANDOM
À chaque fois que ce paramètre est référencé, un entier aléatoire entre 0 et 32767 est engendré. Cette séquence de nombres aléatoires peut être initialisée en affectant une valeur à **RANDOM**. Si **RANDOM** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

REPLY
Contient la ligne d'entrée lue par la commande interne **read** lorsqu'elle n'a pas d'argument.

SECONDS
À chaque fois que cette variable est lue, le nombre de secondes écoulées depuis le lancement de l'interpréteur est renvoyé. Si une valeur est affectée à **SECONDS**, elle renverra lors des lectures ultérieures le nombre de secondes écoulées depuis l'affectation, augmenté de la valeur fournie. Si **SECONDS** est détruite, elle perd ses propriétés spéciales, même si elle est recréée par la suite.

SHELLOPTS
Liste des options activées du shell, séparées par des virgules. Chaque mot de la liste est un argument valide pour l'option **-o** de la commande interne **set** (voir **COMMANDES**INTERNES**DU**SHELL**).** Les options apparaissant dans **SHELLOPTS** sont celle indiquées comme *actives* par **-o**. Si cette variable est dans l'environnement au lancement de **bash**, chaque option de la liste est activée avant de lire les fichiers d'initialisation. Cette variable est en lecture seule.

SHLVL
(Ndt : Lire SH LVL - Shell Level - Niveau d'interpréteur) Incrémenté de 1 à chaque invocation d'une instance de **bash**.

UID
Contient l'ID de l'utilisateur, initialisé au démarrage du shell. Cette variable est en lecture seule.

Les variables suivantes sont utilisées par l'interpréteur. Dans certains cas, **bash** affecte des valeurs par défaut aux variables, ces cas sont décrits plus bas.

BASH_ENV
Si ce paramètre est configuré lorsque **bash** exécute un script, sa valeur est considérée comme un nom de fichier, contenant les commandes d'initialisation de l'interpréteur, comme *.bashrc*. La valeur de **BASH_ENV** est soumise au remplacement des paramètres, substitution de commandes, et évaluation arithmétique avant d'être considérée comme nom de fichier. **PATH** n'est pas utilisée pour rechercher le fichier obtenu.

CDPATH
Le chemin de recherche de la commande interne **cd**. Il s'agit d'une liste de répertoires, séparés par des deux-points « : » que l'interpréteur consulte lorsqu'il cherche un sous-répertoire destination de la commande **cd**. Une configuration possible serait « .:~:/usr ».

COLUMNS
Utilisée par la commande interne **select** pour déterminer la taille du terminal lors de l'affichage des listes de sélection. Automatiquement positionnée à la réception d'un signal SIGWINCH.

COMPREPLY
Une variable de type tableau dans lequel **bash** lit les complétions possibles générées par une fonction shell invoquée par la complétion programmable (voir **Complétion programmable** plus loin).

EMACS
Si **bash** trouve cette variable dans l'environnement lorsque le shell démarre avec la valeur « t », il considère que le shell est lancé dans un tampon shell emacs et désactive l'édition de lignes.

FCEDIT
L'éditeur par défaut utilisé par la commande interne **fc**.

FIGNORE
Une liste de suffixes, séparés par des deux-points « : », que **bash** ignorera lors de la complétion des noms de fichiers (voir le paragraphe **READLINE** plus bas). Un nom de fichier dont le suffixe correspond à l'un de ceux mentionnés dans **FIGNORE** est exclu de la liste des fichiers. Par exemple, cette variable peut prendre la valeur « .o:~ ».

GLOBIGNORE
Une liste de motifs séparés par des deux-points, définissant la liste des noms de fichiers à ignorer lors du développement des chemins. Si un nom de fichier correspondant à un motif d'expansion, correspond également à un motif dans **GLOBIGNORE**, il est supprimé de la liste des correspondances.

HISTCONTROL
Une liste de valeurs, séparées par des virgules, controlant la manière dont les commandes sont enregistrées dans l'historique. Si la liste des valeurs contient la valeur *ignorespace*, les lignes commençant par un caractère d'espacement ne seront par mémorisées dans l'historique. Si elle contient la valeur *ignoredups*, une ligne correspondant exactement à la ligne de commande précédente ne sera pas dupliquée dans l'historique. La valeur *ignoreboth* permet de combiner ces deux comportements. Si elle contient la valeur *erasedups*, toutes les lignes précédant la ligne courante et lui correspondant seront retirées de l'historique avant que cette ligne n'y soit enregistrée. Toute valeur qui ne fait pas partie de la liste ci-dessus

est ignorée. Si **HISTCONTROL** est détruite, ou si elle contient une valeur différente des trois mentionnées ci-dessus, tous les lignes lues par l'interpréteur seront mémorisées dans la liste d'historique, sous contrôle de **HISTIGNORE**. La seconde ligne et les suivantes d'une commande multiligne ne sont pas testées, et sont ajoutées dans l'historique quelque soit la valeur de **HISTCONTROL**.

HISTFILE
Le nom du fichier d'historique pour la mémorisation des commandes (voir **HISTORIQUE** plus bas). Par défaut, il s'agit de *~/.bash_history*.*Si*cette*variable*est détruite, l'historique n'est pas enregistré en fin de shell.

HISTFILESIZE
Le nombre maximal de lignes contenues dans le fichier historique. Quand cette variable contient une valeur, le fichier historique est tronqué, si besoin est, en suprimant les entrées les plus anciennes, pour ne contenir que ce nombre de lignes. La valeur par défaut est 500. Ce fichier est tronqué à cette taille après écriture quand un shell interactif se termine.

HISTIGNORE
Une liste de motifs, séparés par des deux-points, déterminant quelles lignes de commandes seront ajoutées à l'historique. Chaque motif est accroché au début de la ligne et doit lui correspondre en entier (il n'y a pas d'ajout de « ***** » implicite). Chaque motif est comparé à chaque ligne après que les vérifications de **HISTCONTROL** aient été appliquées. En plus des motifs génériques normaux du shell, « **&** » correspond à la ligne précédente de l'historique. « **&** » peut être protégé par un barre oblique inverse. Cette dernière est éliminée avant de tenter une comparaison. La seconde ligne et les suivantes d'une commande multiligne ne sont pas testées, et sont ajoutées dans l'historique quelque soit la valeur de **HISTIGNORE**.

HISTSIZE
Le nombre de commandes à mémoriser dans l'historique (voir le paragraphe **HISTORIQUE** plus bas). La valeur par défaut est 500.

HISTTIMEFORMAT
Si cette varaible existe et qu'elle n'est pas vide, sa valeur est utilisée comme une chaîne de caractères par **strftime**(3) afin d'imprimer l'horodatage associé à chaque entrée de l'historique affichée par la commande interne **history**. Si cette variable existe, les horodatages sont écrits dans le fichier historique afin d'être conservés au fur et à mesure des sessions shell.

HOME
Le répertoire personnel de l'utilisateur. C'est l'argument par défaut de la commande interne **cd**. Cette variable sert également lors du développement du tilde.

HOSTFILE
Contient le nom d'un fichier ayant le même format que */etc/hosts* qui sera consulté lorsque le shell devra compléter un nom d'hôte. Ce fichier peut être changé de manière interactive. Lors de la prochaine tentative pour compléter un nom d'hôte, **bash** ajoutera le nouveau fichier à sa base de données déjà existante. Si **HOSTFILE** existe mais est vide, **bash** essayera de lire */etc/hosts* pour obtenir la liste des complétions de noms d'hôte possible. Lorsque **HOSTFILE** est détruite, la liste des noms d'hôte est effacée.

IFS
Le Séparateur de Champs Interne *Internal Field Separator* qui est utilisé pour séparer les mots après les développements, et pour découper les lignes en mots avec la commande interne **read**. La valeur par défaut est « <espace><tabulation><retour-chariot> ».

IGNOREEOF
Cette variable contrôle le comportement du shell lorsqu'il reçoit un caractère **EOF** (fin de fichier) comme unique entrée. Si elle configurée, la valeur de cette variable correspond au nombre de caractères **EOF** consécutifs apparaissant en début de ligne que **bash** ignorera avant de se terminer. Si cette variable existe mais que sa valeur n'est pas numérique, ou si elle n'a pas de valeur, la valeur par défaut est 10. Si elle n'existe pas, **EOF** est considérée comme un terminateur de saisie pour le shell.

INPUTRC
Le nom du fichier de configuration de **readline** prioritaire sur le fichier par défaut *~/.inputrc* (voir **READLINE** plus bas).

LANG
Détermine la localisation pour toutes les catégories non spécifiquement précisées par une variable commençant par **LC_**.

LC_ALL
Cette variable surcharge **LANG** et toute autre variable **LC_** indiquant la localisation.

LC_CTYPE
Cette variable détermine l'interprétation des caractères et le comportement des classes de caractères (Ndt : p. ex. [:alpha:]) lors du développement des chemins et de la recherche de motifs.

LC_MESSAGES
Cette variable détermine le langage des messages et celui utilisé pour traduire les chaînes entre guillemets précédés par un **$**.

LC_NUMERIC
Cette variable détermine la catégorie locale utilisée pour formater les nombres.

LINES
Utilisée par la commande interne **select** pour déterminer la longueur de colonne lors de l'affichage des listes de sélection. Elle est automatiquement raffraichie à la réception d'un signal SIGWINCH.

MAIL
Si ce paramètre correspond à un nom de fichier, et si la variable **MAILPATH** n'est pas configurée, ou positionnée à une valeur qui n'est pas un nombre supérieur ou égal à zéro, **bash** désactivera la vérification

de courrier.

MAILCHECK

Indique une période (en secondes), par défaut 60, avec laquelle **bash** vérifiera si un nouveau courrier est arrivé. Lorsque le délai est écoulé, le shell vérifiera la présence d'un courrier avant d'afficher son symbole d'accueil. Si cette variable est détruite, le shell cessera la vérification.

MAILPATH

Il s'agit d'une liste de fichiers séparés par des deux-points « : », dans lesquels on vérifiera l'arrivée de courrier. Le nom d'un fichier peut être suivi d'un point d'interrogation, puis d'une chaîne de caractères indiquant le message à afficher en cas de courrier. Dans cette chaîne, le paramètre **$_** correspond au nom du fichier. Exemple :

MAILPATH=aq/var/mail/bfox?"Tu as du courrier":~/shell-mail?"$_ a du courrier"aq

Bash fournit une valeur par défaut pour cette variable, mais l'emplacement du fichier boîte à lettres dépend du système (par exemple /usr/spool/mail/**$USER**).

OPTERR

Si cette variable contient la valeur 1, **bash** affiche les messages d'erreurs déclenchés par la commande interne **getopts** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas). **OPTERR** est initialisée avec la valeur 1 à chaque invocation du shell, ou au lancement d'un script.

PATH

Le chemin de recherche des commandes à exécuter. Il s'agit d'une liste de répertoires, séparés par des deux-points « : » que l'interpréteur consulte lorsqu'il recherche une commande (voir le paragraphe **EXÉCUTION DES COMMANDES** plus bas). Le chemin par défaut dépend du système, et de la configuration choisie par la personne installant **bash**. Une configuration fréquente est « /bin:/usr/bin:/usr/local/bin:. ».

POSIXLY_CORRECT

Si cette variable existe dans l'environnement lorsque **bash** démarre, le shell entre dans le *mode posix* avant de lire les fichiers de configuration comme si l'option **--posix** avait été fournie. Si elle est crée pendant que le shell est en cours d'exécution, **bash** active le *mode posix* comme si la commande « set -o posix » était exécutée.

PROMPT_COMMAND

Lorsque cette variable contient une valeur, celle-ci est exécutée en tant que commande, avant l'affichage du symbole d'accueil principal.

PS1

La valeur de ce paramètre est développée puis utilisée comme symbole d'accueil principal (voir le paragraphe **SYMBOLE D'ACCUEIL** ci-dessous). La valeur par défaut est « **\s-\v\$** ».

PS2

La valeur de ce paramètre est développée comme **PS1** puis utilisée comme symbole d'accueil secondaire. Par défaut, il s'agit de « **>** ».

PS3

La valeur de ce paramètre est utilisée comme symbole pour la commande *select* (voir **GRAMMAIRE DU SHELL** plus haut).

PS4

La valeur de ce paramètre est développée comme **PS1** puis affichée entre chaque commande lors d'un suivi d'exécution. Le premier caractère de **PS4** est répété autant de fois que nécessaire pour indiquer le niveau d'imbrication. La valeur par défaut est « **+** ».

SHELL

Le chemin complet vers le shell est conservé dans cette variable d'environnement. Si elle n'existe pas ou si elle est vide lorsque le shell démarre, **bash** lui affectera le chemin complet du shell de connexion courant de l'utilisateur.

TIMEFORMAT

La valeur de ce paramètre est utilisée en tant que chaîne de format pour afficher le temps consommé dans les pipelines préfixés avec le mot réservé **time**. Le caractère **%** introduit une séquence d'échappement qui est développée avec une valeur de temps ou une autre information. Les séquences d'échappement et leurs significations sont les suivantes; les accolades entourent les parties facultatives.

%%
Un **%** seul (non interprété).

%[*p***][l]R**
Le temps écoulé en secondes.

%[*p***][l]U**
Le temps processeur écoulé en mode utilisateur en secondes.

%[*p***][l]S**
Le temps processeur écoulé en mode système en secondes.

%P
Le pourcentage de temps processeur utilisé calculé avec (%U + %S) / %R.

La séquence facultative *p* est un chiffre indiquant la *précision*, le nombre de chiffres après la virgule. Une valeur de 0 provoque l'affichage de nombre entiers. Trois chiffres au plus peuvent être affichés après la virgule; tout chiffre supérieur à 3 sera changé en 3. Si *p* n'est pas indiqué, 3 est utilisé.

La séquence optionnelle **l** spécifie un format étendu, incluant les minutes, de la forme *MM*m*SS*.*FF*s. La valeur de *p* détermine si la partie décimale est affichée ou non.
Si cette variable n'est pas assignée, **bash** réagit comme si la valeur **$aq\nreal\t%3lR\nuser\t%3lU\nsys %3lSaq** avait été affectée. Si la variable est vide, aucune information n'est affichée. Un caractère fin-de-ligne est ajouté à la fin de l'affichage.

TMOUT
Si cette variable contient une valeur supérieure à zéro, celle-ci est considérée comme un délai, en secondes, pour la commande interne **read**. La commande **select** se termine au bout de **TMOUT** s'il n'y a rien sur l'entrée et que celle-ci est un terminal. Dans un shell interactif, la valeur est interprétée comme le nombre de secondes à attendre pour une entrée après avoir émis l'invite de commande primaire. Lorsque ce délai sera écoulé, **bash** se terminera si aucune saisie n'a eu lieu.

TMPDIR
Si cette variable existe, **bash** utilise sa valeur comme le nom d'un répertoire dans lequel il créera les fichiers temporaires nécessaire au shell.

auto_resume
Cette variable configure le comportement du shell interactif vis-à-vis de l'utilisateur et du contrôle de tâches. Si cette variable existe, les commandes simples constituées d'un seul mot, sans redirection, sont considérées comme de possibles relances de travaux arrêtés. Aucune ambiguïté n'est possible, si plusieurs travaux arrêtés commencent par la chaîne saisie, **bash** sélectionne celui qui a été manipulé le plus récemment. Le *nom* de la tâche arrêtée correspond, dans ce contexte, à la ligne de commande utilisée pour le lancer. Si cette variable contient la valeur *exact*, la chaîne fournie doit correspondre exactement au nom d'une tâche arrêtée. Si elle contient la valeur *substring*, la chaîne fournie doit correspondre à une sous-chaîne du nom du job arrêté. La valeur *substring* fournit un comportement analogue à l'identificateur de tâche **%?** (voir **CONTRÔLE DES TÂCHES**
plus bas). Si la variable contient n'importe quelle autre valeur, la chaîne fournie doit correspondre à un préfixe du nom d'un job arrêté. C'est analogue au fonctionnement du job id **%**.

histchars
Cette variable contient deux ou trois caractères contrôlant le développement de l'historique, et le découpage en jetons (voir **DÉVELOPPEMENT DE L'HISTORIQUE** plus bas). Le premier caractère est le *caractère de développement de l'historique*, c'est-à-dire celui qui indique le début d'un développement d'historique (normalement « **!** »). Le second caractère est celui de *substitution rapide*, utilisé comme raccourci pour relancer la commande précédente, en modifiant une partie de la chaîne. Par défaut il s'agit de « **^** ». L'éventuel troisième caractère est celui qui indique, lorsqu'on le trouve en début de mot que le reste de la ligne est un commentaire. Normalement on utilise le caractère « **#** ». Ce caractère de commentaire empêche l'expansion de l'historique pour tous les mots restants sur la ligne. En revanche, il ne s'agit pas obligatoirement du caractère de commentaire pour l'interpréteur.

Tableaux

Bash propose des variables tableaux à une dimension. N'importe quelle variable peut être utilisée comme tableau ; la commande interne **declare** peut servir à déclarer explicitement un tableau. Il n'y a pas de limite maximale à la taille d'un tableau, ni d'obligation que les membres soient indexés ou assignés de manière contigûe. Les tableaux sont indexés par des entiers en commençant à zéro.

Un tableau est créé automatiquement si une variable est remplie en utilisant la syntaxe *nom*[*index*]=*valeur*. l'*index* est traîté comme une expression arithmétique, et s'évaluer en un nombre positif ou nul. Pour déclarer explicitement un tableau, on utilise **declare -a** *nom* (voir **COMMANDES INTERNES DU SHELL** plus bas). **declare -a** *nom*[*index*] est aussi accepté bien que l'*index* soit ignoré. Des attributs peuvent être spécifiés pour une variable tableau en utilisant les commandes internes **declare** et **readonly**. Les attributs s'appliquent à tous les membres du tableau.

Les tableaux sont remplis en utilisant l'affectation composée de la forme *nom*=**(**valeur_*1* ... valeur_*n***)**, où chaque *valeur* est de la forme [*index*]=*chaîne*. Seule la *chaîne* est obligatoire. Si les crochets et les index optionnels sont fournis, les affectations ont lieu en conséquence ; sinon le rang de l'elément rempli est le dernier rang rempli plus un. Les index commencent à zéro. Cette syntaxed est aussi acceptée par la commande interne **declare**. Les éléments individuels du tableaux sont remplis en utilisant la syntaxe *nom*[*index*]=*valeur* présentée plus haut.

Chaque élément d'un tableau peut référencé en utilisant la notation ${*nom*[*index*]}. Les accolades sont indispensables pour éviter les conflits avec le développement des noms de fichiers. Si *index* est **@** ou *****, le développement donne tous les membres de *nom*. Ces deux index diffèrent lorsqu'ils sont employés entre guillemets. Dans ce cas, ${*nom*[*]} correspond à un seul mot contenant les valeurs de tous les membres séparés par le premier caractère de la variable spéciale **IFS**, et ${*nom*[@]} développe chaque membre de *nom* en un mot distinct. Quant il n'y a pas de membre dans le tableau, ${*nom*[@]} n'a pas de valeur. Si le développement entre guillemets survient dans un mot, le développement du premier paramètre est joint à la première partie du mot d'origine et le développement du dernier paramètre est joint à la dernière partie du mot d'origine. Le comportement est analogue à celui des paramètres spéciaux ***** et **@** (voir **Paramètres spéciaux** plus haut). ${*#nom*[*index*]} donne la longueur du membre ${*nom*[*index*]}. Si l'*index* est ***** ou **@**, le développement donne le nombre d'éléments dans le tableau. Référencer une variable sans index est équivalent à en référencer l'élément zéro.

La commande interne **unset** sert à détruire les tableaux. **unset nom**[*index*] détruit l'élément de rang *index*. Un soin particulier doit être pris afin d'éviter des effets de bords non désirés provoqués par la génération des noms de fichiers. **unset** *nom*, où *nom* est un tableau, ou **unset** *nom*[*index*], avec *index* valant ***** ou **@** supprimer le tableau entier.

Les commandes internes **declare**, **local**, et **readonly** acceptent toutes une option **-a** pour spécifier un tableau. La commande interne **read** accepte une option **-a** pour remplir un tableau avec une liste de mots lus depuis l'entrée standard. Les commandes internes **set** et **declare** affiche les valeurs des tableaux d'une manière permettant de les réutiliser pour les affectations.

EXPANSION

Les expansions sont appliquées à la ligne de commande après qu'elle ait été divisée en mots. Il existe sept types de développements : *expansion des accolades*, *développement du tilde*, *remplacement des paramètres et variables*, *substitution de commandes*, *évaluation arithmétique*, *découpage des mots*, et *développement des noms de fichiers*.

L'ordre est : expansion des accolades, du tilde, des paramètres, des variables, des commandes, évaluation arithmétique (selon la méthode gauche-à-droite), découpage des mots et développement des noms de fichiers.

Sur les systèmes qui le supportent, un développement supplémentaire a lieu : la *substitution de processus*.

Seuls l'expansion des accolades, le découpage des mots, et le développement des noms de fichiers peuvent modifier le nombre de mots. Les autres développement transforment un mot unique en un autre mot unique. La seule exception à cette règle est l'expansion de « **$@** » et « **${***nom***[@]}** » comme on l'a vu plus haut (voir le paragraphe **PARAMÈTRES**).

Expansion des accolades

L'*expansion des accolades* est un mécanisme permettant la création de chaînes quelconques. Il est similaire au *développement des noms de fichiers*, mais les noms de fichiers créés n'existent pas nécessairement. Les motifs qui seront développes prennent la forme d'un *préambule* facultatif, suivi par soit une série de chaînes séparées par des virgules, soit une expression de type séquence encadrée par des accolades. Un *postambule* peut éventuellement suivre la série de chaînes. Le préambule est inséré devant chacune des chaînes contenues entre les accolades, et le postambule est ajouté à la fin de chacune des chaînes résultantes, le développement se faisant de gauche à droite.

Plusieurs développements d'accolades peuvent être imbriqués. Les résultats de chaque développement ne sont pas triés, l'ordre gauche à droite est conservé. Par exemple a**{**d,c,b**}**e se développe en « ade ace abe ».

Une expression de type séquence prend la forme **{***x***..***y***}**, où *x* et *y* sont soit des entiers, soit des caractères seuls. Lorsqu'il s'agit d'entiers, l'expression est remplacée par la liste des nombres entre *x* et *y*, *x* et *y* compris. S'il s'agit de caractères, l'expression est remplacée par l'ensemble des caractères situés entre *x* et *y* d'un point de vue lexicographique. Notez que *x* et *y* doivent être du même type.

L'expansion des accolades est effectuée en premier, et tous les caractères ayant une signification spéciale pour les autres développement sont conservés dans le résultat. Il s'agit d'une modification purement littérale. **Bash** n'effectue aucune interprétation syntaxique du texte entre les accolades.

Une formule correcte pour le développement doit contenir des accolades ouvrantes et fermantes non protégées, et au moins une virgule non protégée ou une expression sequence valide. Toute formule incorrecte n'est pas développée et reste inchangée. Un **{** ou un **,** peuvent être protégés par une barre oblique inverse pour éviter d'être considérés comme partie d'une expression entre accolades.

Cette construction est généralement utilisée comme raccourci lorsque le préfixe commun aux différentes chaînes est relativement long :

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

ou

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

Le développement des accolades induit une légère incompatibilité avec les versions traditionnelles de l'interpréteur Bourne **sh**. **sh** n'effectue aucun traitement sur les accolades ouvrantes et fermantes lorsqu'elles apparaissent dans un mot, et les laisse inchangées. **Bash** supprime les accolades dans les mots, après développement. Par exemple, si l'on fournit à **sh** le mot *file{1,2}*, il reste inchangé en sortie. Par contre, il est transformé en *file1 file2* par **bash**. Si une compatibilité stricte avec **sh** est nécessaire, lancez **bash** avec l'option **+B** ou désactivez le développement des accolades avec l'option **+B** de la commande **set** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas).

Développement du Tilde

Si un mot commence avec le caractère tilde (« **~** »), tous les caractères précédant le premier slash non protégé (voire tous les caractères s'il n'y a pas de slash), sont considérés comme un *préfixe tilde*. Si aucun caractère du préfixe tilde n'est protégé, les caractères suivant le tilde sont traités comme un *nom de login* possible. Si ce *nom de login* est une chaîne nulle, le tilde est remplacé par la valeur du paramètre **HOME**. Si **HOME** n'existe pas, le tilde est remplacé par le répertoire de connexion de l'utilisateur exécutant le shell. Si le préfixe tilde est remplacé par le répertoire personnel associé au nom de login en question.

Si le préfixe tilde est « **+** », la valeur du paramètre shell **PWD** le remplace. Si le préfixe tilde est « - », la valeur du paramètre shell **OLDPWD** lui est substitué. Si les caractères à la suite du tilde dans le préfixe tilde représentent un nombre *N* préfixé éventuellement d'un « **+** » ou d'un « - » le préfixe tilde est remplacé par l'élément correspondant de la pile de répertoires telle qu'il serait affiché par la commande interne **dirs** invoquée avec le préfixe tilde en argument. Si les caractères à la suite du tilde dans le préfixe tilde représentent un nombre sans signe, on suppose qu'il s'agit de « **+** ».

Si le nom est invalide, ou si le développement du tilde échoue, le mot est inchangé.

Chaque affectation de variable est soumis au développement du tilde s'il suit immédiatement un **:** ou le premier **=**. On peut donc utiliser des chemins d'accès avec un tilde pour remplir les variables **PATH**, **MAILPATH**, et **CDPATH**, et le shell fournira la valeur correcte.

Remplacement des paramètres

Le caractère « **$** » permet d'introduire le remplacement des paramètres, la substitution de commandes, ou l'expansion arithmétique. Le nom du paramètre ou du symbole à développer peut être encadré par des accolades, afin d'éviter que les caractères suivants ne soient considérés comme appartenant au nom de la variable.

Lorsque les accolades sont utilisées, l'accolade finale est le premier caractère « **}** » non protégé par une barre oblique inverse ni inclus dans une chaîne protégée, ni dans une expresssion arithmétique, une substitution de commande ou un développement de paramètre.

${*paramètre*}
: est remplacé par la valeur du *paramètre*. Les accolades sont nécessaire quand le *paramètre* est un paramètre positionnel ayant plusieurs chiffres, ou si le *paramètre* est suivi de caractères n'appartenant pas à son nom.

Si le premier caractère du *paramètre* est un point d'exclamation, un niveau d'indirection de variable est introduit. **Bash** utilise la valeur de la variable formée par le reste du *paramètre* comme un nom de variable. Cette variable est alors développée et la valeur utilisée pour le reste de la substitution plutôt que la valeur du *paramètre* lui-même. On appelle ce mécanisme *le développement indirect*. Les exceptions à celui-ci sont les développements de ${!*prefix**} et de ${**!***nom*[@]} décrits plus loin. Le point d'exclamation doit immétiatement suivre l'accolade ouvrante afin d'introduire l'indirection.

Dans chacun des exemples suivants, le *mot* est soumis au développement du tilde, au remplacement des paramètres, à la substitution de commandes, et à l'évaluation arithmétique. **Bash** vérifie si un paramètre existe, et s'il n'est pas nul. L'omission du double point ne fournit qu'un test d'existence.

${*paramètre*:-*mot*}
: **Utilisation d'une valeur par défaut**. Si le *paramètre* est inexistant ou nul, on substitue le développement du **mot**. Sinon, c'est la valeur du *paramètre* qui est fournie.

${*paramètre*:=*mot*}
: **Attribution d'une valeur par défaut**. Si le *paramètre* est inexistant ou nul, le développement du *mot* lui est affecté. La valeur du *paramètre* est alors renvoyée. Les paramètres positionnels, et spéciaux ne peuvent pas être affectés de cette façon.

${*paramètre*:?*mot*}
: **Affichage d'une erreur si inexistant ou nul**. Si le *paramètre* est inexistant, ou nul, le développement du *mot* (ou un message approprié si aucun *mot* n'est fourni) est affiché sur la sortie d'erreur standard, et l'interpréteur se termine, s'il n'est pas interactif. Autrement, la valeur du *paramètre* est utilisée.

${*paramètre*:+*mot*}
: **Utilisation d'une valeur différente**. Si le *paramètre* est nul, ou inexistant, rien n'est substitué. Sinon le développement du *mot* est renvoyé.

${*paramètre*:*début*}

${*paramètre*:*début*:*longueur*}
: **Extraction de sous-chaîne** Se développe pour fournir la sous-chaîne de la *longueur* indiquée (en caractères) commençant au *début*.D *offset*. Si la *longueur* est omise, fournit la sous-chaîne commençant au caractère de *début* et s'étendant jusqu'à la fin du *paramètre*. La *longueur* et le *début* sont des expressions arithmétiques (voir **ÉVALUATION ARITHMÉTIQUE** plus bas). La *longueur* doit être positive ou nulle. Si le *début* est négatif, sa valeur est considérée à partir de la fin du contenu du *paramètre*. Si le *paramètre* est **@**, le résultat correspond aux *longueur* paramètres positionnels commençant au *début*. Si le *paramètre* est un nom de tableau indexé par @ ou *, le résultat est les *longueur* membres du tableau commençant à ${*paramètre*[*début*]}. Une valeur négative de *début* est prise relativement à la valeur maximum de l'index du tableau considéré, augmentée de un. Notez qu'une valeur négative de *début* doit être séparée du deux-

points par au moins une espace pour éviter toute confusion avec le développement de « :- ». L'indexation des sous-chaînes débute à zéro, sauf pour les paramètres positionnels qui débute en 1.

${**#***paramètre*}
${**!***paramètre***@**}
Se développe en les noms des variables dont les noms commencent par *paramètre*, séparés par le premier caractère de la variable spéciale **IFS**.

${**!***nom*[*@*]}
${**!***nom*[***]}
Si *nom* est une variable de type tableau, elle se développe en la liste des indices (clés) du tableau affecté à *nom*. Si *nom* n'est pas un tableau, se développe en 0 si *existe* et en rien autrement. Si *@* est utilisé et que le développement apparaît entre guillemets, chaque clé se développe en un mot séparé.

Est remplacé par la longueur, en caractères, de la valeur du *paramètre*.
Si le *paramètre* est ***** ou **@**, la valeur est le nombre de paramètres positionnels. Si le *paramètre* est un nom de tableau indexé par ***** ou **@**, la valeur est le nombre d'éléments dans le tableau.

${*paramètre***#***mot*}
${*paramètre***##***mot*}
Le *mot* est développé pour fournir un motif, comme dans l'expansion des noms de fichiers. Si le motif correspond au début de la valeur du *paramètre*, alors le développement prend la valeur du *paramètre* après suppression du plus petit motif commun (cas « **#** »), ou du plus long motif (cas « **##** »). Si le *paramètre* est **@** ou *****, l'opération de suppression de motif est appliquée à chaque paramètre positionnel successivement, et le développement donne la liste finale. Si le *paramètre* est une variable tableau indexée par **@** ou *****, l'opération de suppression de motif est appliquée à chaque membre du tableau successivement, et le développement donne la liste finale.

${*paramètre***%***mot*}
${*paramètre***%%***mot*}
est développé pour fournir un motif, comme dans l'expansion des noms de fichiers. Si le motif correspond à une portion finale de la valeur du *paramètre*, alors le développement prend la valeur du *paramètre* après suppression du plus petit motif commun (cas « **%** »), ou du plus long motif (cas « **%%** »). Si le *paramètre* est **@** ou *****, l'opération de suppression de motif est appliquée à chaque paramètre positionnel successivement, et le développement donne la liste finale. Si le *paramètre* est une variable tableau indexée par **@** ou *****, l'opération de suppression de motif est appliquée à chaque membre du tableau successivement, et le développement donne la liste finale.

${*paramètre***/***motif***/***chaîne*}
Le *motif* est développé comme dans le traitement des noms de fichiers. Le *paramètre* est développé et la plus longue portion correspondant au *motif* est remplacée par la *chaîneg*. Si le *motif* commence par **/**, toutes les correspondances de *motif* sont remplacés par *chaîne*. Normalement, seule la première correspondance est remplacée. Si le *motif* commence par **#**, il doit correspondre au début de la valeur développée du *paramètre*. Si le *motif* commence par **%**, il doit correspondre à la fin du développement du *paramètre*. Si la *chaîne* est nulle, les portions correspondant au *motif* sont supprimées et le **/** après le *motif* peut être omis. Si le *paramètre* est **@** ou *****, l'opération de substitution est appliquée à chacun des paramètres positionnels successivement, et le résultat est la liste finale. Si le *paramètre* est une variable tableau indexée par **@** ou *****, l'opération de substitution s'applique à chaque membre du tableau successivement, et le résultat est la liste finale.

Substitution de commandes

La *substitution de commandes* permet de remplacer le nom d'une commande par son résultat. Il en existe deux formes :

$(*commande***)**

ou

`` `*commande*` ``

Bash effectue la substitution en exécutant la *commande* et en la remplaçant par sa sortie standard, dont les derniers sauts de lignes sont supprimés. Les sauts de lignes internes ne sont pas supprimés mais peuvent disparaître lors du découpage en mots. La substitution de commande **$(cat** *fichier*) peut être remplacée par l'équivalent plus rapide **$(<** *fichier*).

Quand l'ancienne forme de substitution avec les backquotes « ` » est utilisée, le caractère antislash garde sa signification littérale, sauf s'il est suivi de **$**, `, ou ****. La première backquote non protégée par un antislash termine la substitution de commande. Quand on utilise la forme $(*commande*), tous les caractères entre parenthèses gardent leurs valeurs littérales. Aucun n'est traité spécialement.

Les substitutions de commandes peuvent être imbriquées. Avec l'ancienne forme, il faut protéger les backquotes internes avec une barre oblique inverse.

Si la substitution apparaît entre guillemets, le découpage des mots, et l'expansion des noms de fichiers ne sont pas effectués.

Évaluation Arithmétique

L'évaluation arithmétique permet de remplacer une expression par le résultat de son évaluation. Le format d'évaluation arithmétique est :

$((*expression***))**

L'*expression* est manipulée de la même manière que si elle se trouvait entre guillemets, mais un guillemet se trouvant entre les parenthèses n'est pas traité spécifiquement. Tous les mots de l'expression subissent le développement des paramètres, la substitution des commandes et la suppression des apostrophes et guillemets. Les développement arithmétiques peuvent être imbriquées.

L'évaluation est effectuée en suivant les règles mentionnées dans le paragraphe **CALCUL**ARITHMÉTIQUE**.** Si l'*expression* est invalide, **bash** affiche un message indiquant l'erreur, et aucune substitution n'a lieu.

Substitution de Processus

La *substitution de processus* n'est disponible que sur les systèmes acceptant le mécanisme des tubes nommés (*FIFOs*) ou la méthode **/dev/fd** de noms de fichiers. Elle prend la forme **<(***liste***)** ou **>(***liste***)**. La *liste* de processus est exécutée avec ses entrées et sorties connectée à une *FIFO* ou à un fichier dans **/dev/fd**. Le nom de ce fichier est passé en argument à la commande qui sera exécutée comme résultat de cette substitution. Si on utilise la forme **>(***liste***)**, l'écriture dans le fichier fournira des entrées pour la *liste*. Si la forme **<(***liste***)** est utilisée, le fichier passe en argument devra être lu pour obtenir la sortie de la *liste*.

Sur les systèmes qui le permettent, la *substitution de processus* est effectuée simultanément au remplacement des paramètres et variables, a la substitution de commande, et à l'évaluation arithmétique.

Séparation des mots

Les résultats du remplacement des paramètres, de la substitution de commandes, et de l'évaluation arithmétique, qui ne se trouvent pas entre guillemets sont analysés par le shell afin d'appliquer le *découpage des mots*.

L'interpréteur considère chaque caractère du paramètre **IFS** comme un délimiteur, et redécoupe le résultat des transformations précédentes en fonction de ceux-ci. Si la valeur du paramètre **IFS** est exactement **<espace> <tabulation><retour-chariot>**, (la valeur par défaut), alors toute séquence de caractères **IFS** sert à délimiter les mots. Si **IFS** a une valeur autre que celle par défaut, alors les séquences de caractères blancs **espace** et **tabulation** sont ignorées en début et fin de mot, à condition que ces caractères blancs se trouvent dans la valeur de **IFS** (un caractère de **IFS** blanc). Tout autre caractère de **IFS** se trouvant à côté d'un caractère blanc de **IFS**, sert à délimiter un champ. Une séquence de caractères blancs de **IFS** est également traitée comme un séparateur. Si la valeur de **IFS** est nulle, aucun découpage de mot n'a lieu.

Les arguments nuls explicites ("" ou **aqaq**) sont conservés. Les arguments nuls implicites, résultant du développement des *paramètres* n'ayant pas de valeurs, sont éliminés. Si un paramètre sans valeur est développé entre guillemets, le résultat est un argument nul qui est conservé.

Notez que si aucun développement n'a lieu, le découpage des mots n'est pas effectué.

Développement des noms de fichiers

Après le découpage des mots, et si l'option **-f** n'est pas indiquée, **bash** recherche dans chaque mot les caractères *****, **?**, **(**, et **[**. Si l'un d'eux apparaît, le mot est considéré comme un *motif*, et remplacé par une liste, classée par ordre alphabétique, des noms de fichiers correspondant à ce motif. Si aucun nom de fichiers ne correspond, et si l'option shell **nullglob** n'est pas active, le mot reste inchangé. Si l'option **nullglob** est active, et si aucune correspondance n'est trouvée, le mot est supprimé. Si l'option shell **failglob** est active, et si aucune correspondance n'est trouvée, un message d'erreur est affiché et la commande n'est pas exécutée Si l'option shell **nocaseglob** est activée, la correspondance a lieu sans tenir compte des différences majuscules/minuscules. Quand un motif est utilisé pour créer des chemins d'accès, le caractère « **.** » au début d'un nom, ou immédiatement à la suite d'un slash doit être explicitement mis en correspondance, à moins que l'option shell **dotglob** ne soit active. Le caractère slash doit toujours être mis en correspondance explicitement. Dans tous les autres cas, le caractère « **.** » n'est pas traité spécifiquement. Voyez la description de la commande **shopt** dans **COMMANDES INTERNES DU SHELL** pour une description des options shell **nocaseglob**, **nullglob**, **failglob**, et **dotglob**.

la variable shell **GLOBIGNORE** peut servir à restreindre l'ensemble des noms de fichiers correspondant à un *motif*. Si **GLOBIGNORE** est remplie, chaque nom de fichier qui correspond aussi à un motif de **GLOBIGNORE** est supprimé de la liste des correspondances. Les noms de fichiers « **.** » et « **..** » sont toujours ignorés quand **GLOBIGNORE** existe et n'est pas vide. Toutefois, configurer **GLOBIGNORE** avec une valeur non nulle a pour effet d'activer l'option shell **dotglob**, ainsi tous les autres noms de fichiers commençant par « **.** » correspondront. Pour

obtenir l'ancien comportement ignorant les noms commençant par « **.** », placez « **.*** » dans l'un des motifs de **GLOBIGNORE**. L'option **dotglob** est désactivée si la variable **GLOBIGNORE** n'existe pas.

Motifs génériques

Tout caractère apparaissant dans un motif, hormis les caractères spéciaux décrits ci-après correspond à lui-même. Le caractère NUL ne peut pas se trouver dans un motif. Une barre oblique inverse protège le caractère suivant ; la barre oblique de protection est abandonné si elle correspond. Les caractères spéciaux doivent être protégés si ils doivent se correspondre littéralement.

Les caractères spéciaux ont les significations suivantes :

*****
: Correspond à n'importe quelle chaîne, y compris la chaîne vide.

?
: Correspond à n'importe quel caractère.

[...]
: Correspond à l'un des caractères entre crochets. Une paire de caractères séparés par un trait d'union indique une *expression intervalle*; tout caractère qui correspond à n'importe quel caractère situé entre les deux bornes incluses, en utilisant les paramètres régionaux courant et le jeu de caractères. Si le premier caractère suivant le **[** est un **!** ou un **^** alors la correspondance se fait sur les caractères non-inclus. L'ordre de tri des caractères dans les expressions intervalle est déterminé par les paramètres régionaux courants et par la valeur de la variable shell **LC_COLLATE** si elle existe. Un **-** peut être mis en correspondance en l'incluant en premier ou dernier caractère de l'ensemble. Un **]** peut être mis en correspondance en l'incluant en premier caractère de l'ensemble.

  Entre **[** et **]**, on peut indiquer une *classe de caractère* en utilisant la syntaxe **[:***classe***:]**, où *classe* est l'une des classes suivantes, définies dans la norme POSIX :

  alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit
  Une classe correspond à un caractère quelconque qui s'y trouve. La classe de caractères **word** correspond aux lettres, aux chiffres et au caractère souligné « _ ».

  Entre **[** et **]**, on peut indiquer une *classe d'équivalence* en utilisant la syntaxe **[=***c***=]**, qui correspond à n'importe quel caractère ayant le même ordre (comme indiqué dans la localisation en cours) que le caractère *c*.

  Entre **[** et **]**, la syntaxe **[.***symbole***.]** correspond au symbole de classement *symbole*.

Si l'option **extglob** du shell option est activée par la commande **shopt** plusieurs opérateurs de correspondance étendue sont reconnus. Dans la description suivante, une *liste-motif* est une liste d'un ou plusieurs motifs séparés par des |. Les motifs composés sont formés en utilisant un ou plusieurs sous-motifs comme suit :

?(*liste-motif***)**
: Correspond à zéro ou une occurrence des motifs indiqués

(liste-motif***)**
: Correspond à zéro ou plusieurs occurrences des motifs indiqués

+(*liste-motif***)**
: Correspond à une ou plusieurs occurrences des motifs indiqués

@(*liste-motif***)**
: Correspond à une occurrence exactement des motifs indiqués

!(*liste-motif***)**
: Correspond à tout sauf les motifs indiqués

Suppression des protections

Après les développements précédents, toutes les occurrences non-protégées des caractères ****, **aq**, et **'' qui ne résultent pas d'un développement sont supprimées.

REDIRECTION

Avant qu'une commande ne soit exécutée, il est possible de *rediriger* son entrée et sa sortie en utilisant une notation spéciale interprétée par le shell. Les redirections peuvent également servir à ouvrir ou fermer des fichiers dans l'environnement actuel du shell. Les opérateurs de redirection décrits ci-dessous peuvent apparaître avant, ou au sein d'une *commande simple* ou suivre une *commande*. Les redirections sont traitées dans l'ordre d'apparition de gauche à droite.

Dans les descriptions suivantes, si le numéro de descripteur de fichier est omis, et si le premier caractère de l'opérateur de redirection est **<**, celui-ci correspondra à l'entrée standard (descripteur de fichier 0). Si le premier

caractère de l'opérateur est **>**, la redirection s'appliquera à la sortie standard (descripteur de fichier 1).

Le mot qui suit l'opérateur de redirection dans les descriptions suivantes est soumis à l'expansion des accolades, du tilde, des paramètres, à la substitution de commandes, à l'évaluation arithmétique, à la suppression des protections, et au développement des noms de fichiers. S'il se modifie pour donner plusieurs mots, **bash** détectera une erreur.

Remarquez que l'ordre des redirections est important. Par exemple, la commande

ls **>** liste_répertoires 2**>&**1

redirige a la fois la sortie standard et la sortie d'erreur vers le fichier *liste_répertoires*, alors que la commande

ls 2**>&**1 **>** liste_répertoires

ne redirige que la sortie standard vers le fichier *liste_répertoires*, car la sortie d'erreur a été renvoyée vers la sortie standard avant que celle-ci ne soit redirigée vers *liste_répertoires*.

Bash gère plusieurs nom de fichiers de manière particulière, lorsqu'ils sont utilisés dans des redirections, comme décrite dans la table suivante :

/dev/fd/*fd*
: Si *fd* est un entier valide, le descripteur de fichier *fd* est dupliqué.

/dev/stdin
: Le descripteur de fichier 0 est dédoublé.

/dev/stdout
: Le descripteur de fichier 1 est dédoublé.

/dev/stderr
: Le descripteur de fichier 2 est dédoublé.

/dev/tcp/*host*/*port*
: Si *host* est une adresse Internet ou un nom d'hôte valide, et si *port* est un numéro de port entier ou un nom de service, **bash** tentera d'ouvrir une connexion TCP sur le socket correspondante. If *host* is a valid hostname or Internet address, and *port* is an integer port number or service name, **bash** attempts to open

Une erreur d'ouverture ou de création de fichier peut déclencher un échec.

Les redirections qui utilisent des descripteurs de fichiers supérieurs à 9 doivent être utilisés avec précaution car il peut y avoir des conflits avec les descripteurs de fichiers que le shell utilse en interne.

Redirection d'entrée

Lorsque l'on applique une redirection d'entrée, le fichier dont le nom résulte du développement du *mot* sera ouvert en lecture avec le descripteur de fichier numéro *n*, ou en tant qu'entrée standard (descripteur de fichier 0) si *n* n'est pas mentionné.

Le format général des redirections d'entrée est le suivant :

[*n*]**<***mot*

Redirection de sortie

Lors d'une redirection de sortie, le fichier dont le nom résulte du développement du *mot* est ouvert en écriture, avec le descripteur de fichier *n*, ou en tant que sortie standard (descripteur de fichier 1) si *n* n'est pas mentionné. Si le fichier n'existe pas, il est créé. S'il existait déjà, sa taille est ramenée à 0.

Le format général des redirections de sortie est le suivant :

[*n*]**>***mot*

Si l'opérateur de redirection est **>**, et si l'option **noclobber** de la commande interne **set** est activée, la redirection échouera si le fichier dont le nom résulte du développement du *mot* existe et est un fichier régulier. Si l'opérateur de redirection est **>|**, ou l'opérateur **>** avec l'option **noclobber** de la commande interne **set** n'est pas activée, la redirection sera tentée même si le fichier dont le nom résulte du développement du *mot* existe.

Redirection pour ajout en sortie

Lorsqu'on redirige ainsi la sortie, le fichier dont le nom résulte du développement du *mot* est ouvert pour ajout en fin de fichier, avec le descripteur *n*, ou en tant que sortie standard (descripteur 1) si *n* n'est pas mentionné. Si le fichier n'existe pas, il est créé.

Le format général pour la redirection de sortie avec ajout est :

[*n*]**>>***mot*

Redirection de la sortie standard et de la sortie d'erreur

Bash permet la redirection simultanée de la sortie standard (descripteur 1) et de la sortie d'erreur (descripteur 2), dans un fichier dont le nom est le résultat du développement du *mot* avec cette construction.

Il y a deux formes pour effectuer cette double redirection :

&>*mot*

et

>&*mot*

On préfère généralement la première. Elle est sémantiquement équivalente à

>*mot* 2**>&**1

Document en ligne

Avec ce type de redirection, le shell va lire son entrée standard jusqu'à ce qu'il atteigne une ligne contenant uniquement le *mot* prévu (sans espaces à la suite), nommée *étiquette*. Une fois cette étiquette atteinte, il exécutera la commande demandée en lui fournissant en entrée le texte lu avant l'étiquette, que l'on appelle *document en ligne*.

Le format des documents en ligne est le suivant :

```
<<[-]mot
        document en ligne
étiquette
```

Il n'y a ni remplacement de paramètre, ni substitution de commande, ni développement de chemin d'accès, ni évaluation arithmétique sur le *mot*. Si l'un des caractères du *mot* est protégé, l' étiquette est obtenue après suppression des protections dans le *mot*, et les lignes du document ne sont pas développées. Sinon, toutes les lignes du document sont soumises au remplacement des paramètres, à la substitution de commandes, et à l'évaluation arithmétique. Dans ce dernier cas, les couples **\<retour-chariot>** sont ignorés, et **** doit être utilisé pour protéger les caractères ****, **$**, et **`**.

Si l'opérateur de redirection est **<<-**, alors les tabulations en tête de chaque ligne sont supprimées, y compris dans la ligne contenant *étiquette*. Ceci permet d'indenter de manière naturelle les documents en ligne au sein des scripts.

Chaînes en ligne

Une variante aux documents en ligne, le format est :

<<<*mot*

Le *mot* est développé et fourni à la commande sur son entrée standard.

Dédoublement de descripteur de fichier

L'opérateur de redirection

[*n*]**<&***mot*

permet de dupliquer les descripteurs de fichiers en entrée. Si le *mot* se transforme en un ou plusieurs chiffres, le descripteur de fichier *n* devient une copie de ce descripteur. Si les chiffres du *mot* ne correspondent pas à un descripteur en lecture, une erreur se produit. Si le *mot* prend la forme **-**, le descripteur *n* est fermé. Si *n* n'est pas mentionné, on utilise l'entrée standard (descripteur 0).

L'opérateur

[*n*]**>&***mot*

est utilisé de manière similaire pour dupliquer les descripteurs de sortie. Si *n* n'est pas précisé, on considère la sortie standard (descripteur 1). Si les chiffres du *mot* ne correspondent pas à un descripteur en écriture, une erreur

se produit. Un cas particulier se produit si *n* est omis, et si *mot* ne se développe pas sous forme de chiffres. Alors, les sorties standard et d'erreurs sont toutes deux redirigées comme précédemment.

Déplacement de descripteurs de fichiers

L'opérateur de redirection

[*n*]**<&***chiffre***-**

déplace le descripteur de fichier *chiffre* vers le descripteur de fichier *n*, ou sur l'entrée standard (descripteur de fichier 0) si *n* n'est pas spécifié. *chiffre* est fermé après avoir été dédoublé en IR n .

De la même manière, l'opérateur de redirection

[*n*]**>&***chiffre***-**

déplace le descripteur de fichier *chiffre* vers le descripteur de fichier *n* sour sur la sortie standard (descripteur de fichier 1) si *n* n'est pas spécifié.

Ouverture en Lecture/Écriture d'un descripteur de fichier

L'opérateur de redirection

[*n*]**<>***mot*

ouvre le fichier dont le nom résulte du développement du *mot*, à la fois en lecture et en écriture et lui affecte le descripteur de fichier *n*, ou bien le descripteur 0 si *n* n'est pas mentionné. Si le fichier n'existe pas, il est créé.

ALIAS

Les alias permettent de substituer une chaîne à un mot lorsqu'il est utilisé comme premier mot d'une commande simple. L'interpréteur gère une liste d'*alias* qui peuvent être créés et détruits avec les commandes internes **alias** et **unalias** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas). Le shell vérifie si le premier mot de chaque commande simple (s'il n'est pas protégé) est un alias. Si tel est le cas, ce mot est remplacé par le texte de l'alias. Les caractères **/**, **$**, **`**, et **=** et tout *méta-caractère* ou caractère de citation (Ndt : quoting) listés plus haut ne doivent pas apparaître dans un nom d'alias. Le texte de remplacement peut contenir n'importe quels caractères valides pour le shell, y compris les méta-caractères shell. Le shell vérifie si le premier mot du texte de remplacement est à son tour le nom d'un alias, mais s'il est identique au nom de l'alias il ne sera pas remplacé à nouveau. Ceci permet de créer un alias **ls** valant **ls -F**, par exemple, que **bash** n'essaiera pas de développer récursivement. Si le dernier caractère du texte de remplacement est un *blanc*, alors on vérifiera également si le mot suivant l'alias est lui aussi un alias.

Les alias sont créés et affichés avec la commande **alias**, et supprimés avec la commande **unalias**.

Il n'existe pas de mécanisme permettant d'utiliser des arguments dans le texte de remplacement. Si on en a besoin, il faut utiliser une fonction du shell.

Les alias ne sont développés qu'avec les shells interactifs sauf si l'option **expand_aliases** du shell est activée par la commande **shopt** (voir la description de **shopt** au paragraphe **COMMANDES INTERNES DU SHELL** plus bas)

Les règles concernant la définition et l'utilisation des alias sont un peu confuses. **Bash** lit toujours au moins une ligne d'entrée complètement avant d'exécuter une commande de cette ligne. Les alias sont traités lorsque la commande est lue, et non pas lorsqu'elle est exécutée. Ainsi, une définition d'alias apparaissant sur la même ligne qu'une autre commande ne prend pas effet avant la lecture de la prochaine ligne d'entrée. Autrement dit, une commande placée à la suite d'une définition d'alias, sur la même ligne ne sera pas affectée par cet alias. Ce comportement est également important lors de l'exécution des fonctions. Les alias sont traités lorsque la définition de la fonction est lue, et non pas lors de l'exécution de cette fonction. Ainsi des alias définis dans une fonction ne sont pas actifs avant la fin de l'exécution de la fonction. Pour plus de tranquillité, placez toujours les définitions d'alias sur des lignes isolées, et n'utilisez jamais la commande **alias** dans les commandes composées.

Notez enfin que la plupart du temps les alias sont avantageusement remplacés par des fonctions du shell.

FONCTIONS

Une fonction du shell mémorise une série de commandes pour permettre une exécution ultérieure. Leurs definition est décrite plus haut, dans **GRAMMAIRE**DU**SHELL**. Les fonctions sont exécutées dans le contexte de l'interpréteur en cours. On ne crée pas de nouveau processus pour interpréter une fonction, contrairement à l'exécution d'un script. Les arguments d'une fonction sont placés dans les paramètres positionnels durant son exécution. Le paramètre spécial **#** est mis a jour. Le paramètre spécial 0 n'est pas modifié. Le premier élément de la variable **FUNCNAME** est rempli avec le nom de la fonction lors de l'exécution de celle-ci. Les autres aspects de

l'environnement d'exécution du shell sont identiques entre l'intérieur de la fonction et le corps appelant du programme, à l'exception de la gestion de **DEBUG** et **RETURN** (voir la description de la commande interne **trap** au paragraphe **COMMANDES INTERNES DU SHELL** plus bas) qui n'est pas héritée à moins que la fonction n'ait été fournie avec l'attribut **trace** (voir la description de la commande interne **declare** plus loin) ou que l'option shell **-o functrace** ait été activée avec la commande interne **set** (auquel cas toutes les fonctions hériteront du déroutement (Ndt : trap) de **DEBUG** et **RETURN**.

Les variables locales d'une fonction peuvent être déclarées avec la commande **local**. Autrement, les variables et leurs valeurs sont partagées entre la fonction et son appelant.

Si la commande interne **return** est exécutée dans une fonction, celle-ci se termine et l'exécution reprend avec la commande suivant l'appel de fonction. Toute commande associée au détournement de **RETURN** est exécutée avant que l'exécution s'achève. Quand une fonction se termine, les paramètres positionnels et le paramètre spécial # reprennent les valeurs qu'ils avaient avant l'appel de fonction.

Les noms des fonctions et leurs définitions sont affichés avec l'option **-f** des commandes internes **declare** ou **typeset**. Les options **-F** de **declare** ou **typeset** n'afficheront que les noms des fonctions (et optionnellement le fichier source et le numéro de ligne, si l'option shell **extdebug** est active). Les fonctions peuvent être exportées automatiquement avec l'option **-f** de la commande interne **export**, afin de rester définies dans les sous-shells. Une définition de fonction peut être supprimée en utilisant l'option **-f** de la commande interne **unset**. Notez que les fonctions et variables shell de même noms feront qu'il y aura plusieurs entrées de mêmes noms dans l'envireonnement passé au shell fils. Un soin particulier doit être pris dans ces cas où peuvent survenir des problèmes.

Les fonctions peuvent être récursives. Aucune limite n'est imposée quant au nombre d'appels récursifs.

CALCUL ARITHMÉTIQUE

Le shell permet, dans certaines circonstances, l'évaluation d'expressions arithmétiques (voir les commandes internes **let** et **declare** ainsi que le paragraphe **Évaluation Arithmétique**). L'évaluation est effectuée en utilisant des entiers longs, sans vérification du débordement. Néanmoins, la division par 0 est détectée et renvoie une erreur. La liste des opérateurs ci-dessous, leur priorité, associativité et valeur sont les mêmes que dans le language C. Les niveaux sont classés en ordre de précédence décroissante.

- +
: plus et moins unaire

! ~
: négations logique et binaire

******
: exponentiation

*** / %**
: multiplication, division, reste

+ -
: addition, soustraction

<< >>
: décalage arithmétique à gauche et à droite

<= >= < >
: comparaisons

== !=
: égalité et différence

&
: ET binaire

^
: OU exclusif binaire

|
: OU binaire

&&
: ET logique

||
: OU logique

expr?*expr*:*expr*
: opérateur conditionnel

= *= /= %= += -= <<= >>= &= ^= |=
: assignations

On peut utiliser les variables du shell comme opérandes, le développement des paramètres étant effectué avant l'évaluation arithmétique. Dans une expression, les varaiables shell peuvent également être référencées par leur nom, sans utiliser la syntaxe de développement des paramètres. Une variable shell qui est vide ou qui n'existe plus est évaluée à 0 lorsqu'elle est référencée par son nom sans utiliser la syntaxe de développement des paramètres.

La valeur d'une variable est évaluée comme une expression arithmétique lorqu'elle est référencée, ou lorsque une variable, qui a vu son attribut « entier » positionné en utilisant **declare -i**, se voit affecter une valeur. Une variable shell n'a donc pas besoin d'avoir l'attribut « entier » pour être utilisable dans un calcul.

Les constantes commençant par un 0 sont interprétées comme des nombres octaux. Un entête *0x* ou *0X* indique une valeur hexadécimale. Sinon, les nombres ont la forme [*base#*]n, ou la *base* arithmétique est un nombre décimal entre 2 et 36, et *n* le nombre lui-même exprimé dans cette base. Si la *base* n'est pas mentionnée, le nombre est considéré comme décimal. Les chiffres supérieurs à 9 sont représentés par les lettres minuscules, les lettres majuscules, le @ et le _, dans cet ordre. Si la *base* est inférieure ou égale à 36 les minuscules et les majuscules peuvent être utilisées de manière interchangeable pour représenter les nombres entre 10 et 35.

Les opérateurs sont évalués dans l'ordre de précédence. Les sous-expressions entre parenthèses sont évaluées en premier, ce qui permet de surpasser les règles de précédence.

EXPRESSIONS CONDITIONNELLES

Les expressions conditionnelles sont utilisées par la commande composée **[[** et les commandes internes **test** et **[** pour vérifier l'état d'un fichier et assurer des comparaisons arithmétiques ou sur des chaînes. Les expressions se forment à partir des opérateurs unaires ou binaires suivants. Si un argument *fichier* est de la forme /dev/fd/*n*, alors il représente le descripteur de fichier *n*. Si un argument *fichier* est de la forme */dev/stdin*, */dev/stdout* ou */dev/stderr*, il représente, respectivement, le descripteur de fichier 0, 1 ou 2.

À moins que cela ne soit spécifié autrement, les opérateurs relatifs aux fichiers suivent les liens symboliques et agissent sur la cible du lien plutôt que sur le lien lui-même.

-a *fichier*
: Vrai si le *fichier* existe.

-b *fichier*
: Vrai si le *fichier* existe et est un fichier spécial bloc.

-c *fichier*
: Vrai si le *fichier* existe et est un fichier spécial caractère.

-d *fichier*
: Vrai si le *fichier* existe et est un répertoire

-e *fichier*
: Vrai si le *fichier* existe.

-f *fichier*
: Vrai si le *fichier* existe et est un fichier régulier.

-g *fichier*
: Vrai si le *fichier* existe et a son bit Set-GID positionné.

-h *fichier*
: Vrai si le *fichier* existe et est un lien symbolique.

-k *fichier*
: Vrai si le *fichier* existe et a son bit « sticky » positionné.

-p *fichier*
: Vrai si le *fichier* existe et est un tube nommé (FIFO).

-r *fichier*
: Vrai si le *fichier* existe et est accessible en lecture.

-s *fichier*
: Vrai si le *fichier* existe et a une taille non nulle.

-t *fd*
: Vrai si *fd* est ouvert sur un terminal.

-u *fichier*
: Vrai si le *fichier* existe et a son bit Set-UID positionné.

-w *fichier*
: Vrai si le *fichier* existe et est accessible en écriture.

-x *fichier*
: Vrai si le *fichier* existe et est exécutable.

-O *fichier*
: Vrai si le *fichier* existe et appartient à l'ID effectif de l'utilisateur.

-G *fichier*
: Vrai si le *fichier* existe et appartient au groupe effectif de l'utilisateur.

-L *fichier*
: Vrai si le *fichier* existe et est un lien symbolique.

-S *fichier*
: Vrai si le *fichier* existe et est une socket

-N *fichier*
: Vrai si le *fichier* existe et a été modifié depuis sa dernière lecture.

fichier_1 **-nt** *fichier_2*

Vrai si le *fichier_1* est plus récent que le *fichier_2*, d'après les dates de modification, ou si *fichier_1* existe et pas *fichier_2*.

fichier_1 -**ot** *fichier_2*
Vrai si le *fichier_1* est plus vieux que le fichier_2 ou si *fichier_2* existe et pas *fichier_1*.

fichier_1 **-ef** *fichier_2*
Vrai si le *fichier_1* et le *fichier_2* se réfèrent au même périphérique et ont les mêmes numéros d'i-noeuds.

-o *nom_opt*
Vrai si l'option shell *nom_opt* est active. Cosulter la liste des options dans la description de l'argument **-o** de la commande interne **set** plus bas.

-z *chaîne*
Vrai si la longueur de la *chaîne* est nulle.

chaîne
-n *chaîne*
Vrai si la longueur de la
chaîne est non-nulle.

chaîne_1 **==** *chaîne_2*
Vrai si les deux chaînes sont égales. Le symbole **=** peut servir à remplacer **==** pour une stricte comformité POSIX.

chaîne_1 **!=** *chaîne_2*
Vrai si les deux chaînes sont différentes.

chaîne_1 **<** *chaîne_2*
Vrai si *chaîne_1* se trouve avant *chaîne_2* dans l'ordre lexicographique de la localisation en cours.

chaîne_1 **>** *chaîne_2*
Vrai si *chaîne_1* se trouve après *chaîne_2* dans l'ordre lexicographique de la localisation en cours.

arg1 **OP** arg2
OP est l'un des opérateurs suivants **-eq**, **-ne**, **-lt**, **-le**, **-gt**, ou **-ge**. Ces opérateurs arithmétiques binaires renvoient la valeur vraie si *arg_1* est respectivement égal, différent, inférieur, inférieur ou égal, supérieur, supérieur ou égal à *arg2*. *Arg1* et *arg2* peuvent être des entiers positifs ou négatifs.

DÉVELOPPEMENT DES COMMANDES SIMPLES

Lors de l'exécution d'une commande simple, le shell effectue les développements affectations, et redirections de gauche à droite, suivants.

1. Les mots que l'analyser a repéré comme affectation de variables (ceux qui précèdent le nom de la commande) et les redirections sont mémorisés pour une mise en place ultérieure.
2. Les autres mots sont développés. S'il reste des mots après le développement, le premier est considéré comme le nom d'une commande et les suivants comme ses arguments.
3. Les redirections sont mises en place, comme décrit plus-haut au paragraphe **REDIRECTION**.
4. Le texte suivant le **=** dans chaque affectation est soumis au dévelopement du tilde, des paramètres, à la substitution de commande, à l'évaluation arithmétique et à la suppression des protection avant de remplir la variable.

Si aucun nom de commande ne résulte des précédentes opérations, les assignations de variable modifie l'environnement en cours. Sinon elles sont ajoutées à celui de la commande exécutée et ne modifient pas l'environnement du shell. Si l'une des tentatives d'affectation concerne une variable en lecture seule, une erreur se produit, et la commande se termine sur un code non-nul.

Si aucun nom de commande n'est obtenu, les redirections sont réalisées mais ne modifient pas l'environnement du shell en cours. Une erreur de redirection renvoie un code de retour non-nul.

S'il reste un nom de commande après l'expansion, l'exécution a lieu comme décrit ci-dessous. Sinon la commande se termine. Si l'un des développement contient une substitution de commande, le code de retour est celui de la dernière substitution de commande réalisée. S'il n'y en a pas, la commande se termine avec un code de retour nul.

EXÉCUTION DES COMMANDES

Après le découpage de la ligne de commande en mots, si le résultat est une commande simple suivie d'une éventuelle liste d'arguments, les actions suivantes sont effectuées.

Si le nom de la commande ne contient pas de slash, le shell tente de la trouver. S'il existe une fonction shell de ce nom, elle est appelée comme nous l'avons vu dans le paragraphe **FONCTIONS** plus haut. Si le nom n'est pas celui

d'une fonction, l'interpréteur le recherche dans la liste des fonctions internes. Si une équivalence est trouvée, la fonction interne est invoquée.

Si le nom n'est ni celui d'une fonction shell, ni celui d'une commande interne, et s'il ne contient pas de slash, **bash** va chercher dans chacun des membres de **PATH** un répertoire contenant un fichier exécutable du nom désiré. **Bash** utilise une table de hachage pour mémoriser les noms d'accès complets des fichiers (voir la commande **hash** au paragraphe **COMMANDES INTERNES DU SHELL** plus bas). Une recherche complète dans les répertoires du **PATH** n'est entreprise que si la commande ne se trouve pas dans la table de hachage. Si la recherche est infructueuse, l'interpréteur affiche un message d'erreur et renvoie un code de retour valant 127.

Si la recherche réussit, ou si le nom de la commande contient au moins un slash, le shell exécute le programme voulu dans un environnement distinct. L'argument 0 est rempli avec le nom fourni, et les autres arguments seront éventuellement remplis avec le reste de la ligne de commande.

Si l'exécution échoue parce que le programme n'est pas un exécutable, et si le fichier n'est pas un répertoire, on le considère alors comme un *script shell*, un fichier contenant une série de commandes. Un sous-shell est alors créé pour exécuter ce script. Ce sous-shell se réinitialisera lui-même, comme si un nouveau shell avait été invoqué pour exécuter le script, à la différence qu'il continuera à mémoriser l'emplacement des commandes connues de son parent (voir **hash** dans le paragraphe **COMMANDES INTERNES DU SHELL** plus bas).

Si le programme est un fichier commençant par **#!**, le reste de la première ligne indique un interpréteur pour ce programme. Le shell se chargera d'exécuter cet interpréteur, si le système d'exploitation ne gère pas lui-même ce format d'exécutable. Les arguments de l'interpréteur consistent en un premier argument éventuel fourni sur la première ligne du fichier à la suite du nom de l'interpréteur, suivi du nom du programme, suivi des arguments de la commande s'il y en a.

ENVIRONNEMENT D'EXÉCUTION DES COMMANDES

Le shell fournit un *environnement d'exécution* consistant en l'ensemble des éléments suivants :

* les fichiers ouverts hérités par le shell à son lancement, et modifiés par les redirections de la commande interne **exec** ;
* le répertoire de travail configuré par **cd**, **pushd**, ou **popd**, ou hérité par le shell à son démarrage ;
* le masque de création de fichier fixé par **umask** ou hérité du père du shell ;
* la gestion des signaux configurée avec **trap** ;
* les paramètres du shemm fixés par des affectation de variables ou avec la commande interne **set** ou hérité de l'environnement du père du shell ;
* les fonctions du shell définies lors de l'exécution ou héritées de l'environnement du père du shell ;
* les options fournies au démarrage (par défaut ou sur la ligne de commandes) ou par la commande interne **set** ;
* les options activées par la commande interne **shopt** ;
* les alias du shell définies par la commande **alias** ;
* les ID des divers processus, y compris ceux à l'arrière-plan, la valeur de **$$**, et la valeur de **$PPID**.

Lorsqu'une commande autre qu'une fonction interne ou une fonction du shell doit être exécutée, elle est invoquée dans un environnement d'exécution séparé constitué de ce qui suit. Sauf indication contraire, les valeurs sont héritées de l'environnement du shell.

* les fichier ouverts du shell, auxquels s'ajoutent les modifications indiquées par les redirections apportées à la commande ;
* le répertoire de travail en cours ;
* le masque de création de fichiers ;
* les variables et les fonctions du shell marquées pour l'exportation, ainsi que les variables exportées spécialement pour la commande ;
*

les signaux capturés par le shell reprennent le comportement hérité du père du shell, et les signaux ignorés par le shell restent ignorés.

Une commande invoquée dans un environnement distinct ne peut pas affecter l'environnement d'exécution du shell.

Les substitutions de commandes, les commandes groupées entre paranthèses et les commandes asynchrones sont invoquées dans un environnement de sous-shell qui est une copie de celui du shell, sauf que les signaux capturés par le shell reprennent le comportement qu'ils avaient au lancement du shell. Les fonctions internes invoquées dans un pipeline sont aussi exécutées dans un environnement de sous-shell. Les modifications apportées à l'environnement d'un sous-shell n'affectent aucunement l'environnement du shell.

Si une commande est suivie par un **&** et qu'un contrôle de tâches n'est pas actif, l'entrée standard par défaut de la commande est le fichier vide */dev/null*. Autrement, la commande invoquée hérite des descripteurs de fichiers du shell appelant tels qu'ils ont été modifiés par les redirections.

ENVIRONNEMENT

Quand un programme est invoqué, il reçoit un tableau de chaînes que l'on appelle *environnement*. Il s'agit d'une liste de paires *nom-valeur*, de la forme *nom*=valeur.

Le shell permet de manipuler l'environnement de plusieurs façons. Au démarrage, le shell analyse son propre environnement, et crée un paramètre pour chaque nom trouvé, en le marquant comme *exportable* vers les processus fils. Les commandes exécutées héritent de cet environnement. Les commandes **export** et **declare -x** permettent d'ajouter ou de supprimer des paramètres ou des fonctions de l'environnement. Si la valeur d'un paramètre de l'environnement est modifiée, la nouvelle valeur devient une partie de l'environnement, et elle remplace l'ancienne. L'environnement hérité par les commandes exécutées est l'environnement initial du shell (dont les valeurs peuvent être modifiées), moins les éléments supprimés par la commande **unset**, plus les éléments ajoutés par les commandes **export** et **declare -x**.

L'environnement d'une *commande simple* ou d'une fonction peut être augmenté temporairement en la faisant précéder d'une affectation de paramètre, comme nous l'avons vu au paragraphe **PARAMÈTRES**. Ces affectations ne concernent que l'environnement vu par cette commande ou fonction.

Si l'option **-k** est utilisée (voir la commande interne **set** plus bas), alors *toutes* les affectations de paramètres sont placées dans l'environnement d'une commande exécutée, pas seulement celles qui précèdent son nom.

Quand **bash** invoque une commande externe, la variable **_** contient le chemin d'accès complet à cette commande, et elle est transmise dans l'environnement.

CODE DE RETOUR

Au niveau du shell, une commande qui se termine avec un code de retour nul est considérée comme réussie. Le zéro indique le succès. Un code de retour non-nul indique un échec. Quand une commande se termine à cause d'un signal fatal, **bash** utilise la valeur 128+**signal** comme code de retour.

Si une commande n'est pas trouvée, le processus fils créé pour l'exécuter renvoie la valeur 127. Si la commande est trouvée mais pas exécutable, la valeur renvoyée est 126.

Si une commande échoue à cause d'une erreur survenue lors du développement ou de la mise en place des redirections, le code de retour est supérieur à zéro.

Les fonctions interne renvoient zéro (*vrai*) si elles réussissent et une valeur non nulle (*faux*) si une erreur se produit durant leur exécution. Toutes les fonctions internes renvoient 2 en cas d'erreur de syntaxe.

Bash lui-même renvoie le code de retour de la dernière commande exécutée, à moins qu'une erreur de syntaxe ne se produise, auquel cas il renvoie une valeur non-nulle. Voir également la commande interne **exit** plus bas.

SIGNAUX

Quand **bash** fonctionne de manière interactive, il ignore le signal **SIGTERM** (c'est pourquoi **kill 0** ne tue pas un shell interactif), et le signal **SIGINT** est intercepté et géré (c'est pourquoi la commande interne **wait** n'est pas interruptible). Dans tous les cas, **bash** ignore le signal **SIGQUIT**. Si le contrôle de jobs est disponible, **bash** ignore les signaux **SIGTTIN**, **SIGTTOU**, et **SIGTSTP**.

Les commandes externes lancées par **bash** reçoivent les valeurs des signaux héritées par le shell de leur parent. Quand le contrôle de job n'est pas actif, les jobs en arrière-plan ignorent les signaux **SIGINT** et **SIGQUIT** en plus des gestionnaires hérités. Les commandes exécutées en tant que résultat de la substitution de commandes ignorent les signaux de contrôle de jobs émis par le clavier **SIGTTIN**, **SIGTTOU**, et **SIGTSTP**.

Par défaut, le shell se termine à la réception de **SIGHUP**. Avant de se terminer, un shell interactif renvoie **SIGHUP** à tous les jobs en cours ou arrêtés. Les jobs arrêtés reçoivent aussi **SIGCONT** pour s'assurer qu'il recevront bien le **SIGHUP**. Pour empêcher le shell d'envoyer ce signal à un job particulier, il faut le retirer de la table des jobs à l'aide de la commande interne **disown** (voir **FONCTIONS INTERNES DU SHELL** plus bas) ou le marquer comme exempté de **SIGHUP** avec **disown -h**.

Si l'option du shell **huponexit** a été activée avec la fonction **shopt**, **bash** envoie un signal **SIGHUP** à tous les jobs lorsqu'un shell de login interactif se termine.

Lorsque **bash** attend la fin d'une commande et qu'il reçoit un signal qu'il doit capturer, le gestionnaire de signal ne sera pas exécuté avant que la commande soit finie. Lorsque **bash** attend la fin d'une commande asynchrone avec **wait** la réception d'un signal capturé causera la fin immédiate de la commande **wait** avec un code de retour supérieur à 128, à la suite de quoi le gestionnaire de signal sera exécuté.

CONTRÔLE DES TÂCHES

Le *contrôle des jobs* permet d'arrêter (*suspend*) sélectivement l'exécution d'un processus, et de la reprendre (*resume*) ultérieurement. L'utilisateur bénéficie généralement de cette possibilité grâce à l'action conjointe du pilote de terminal et de **bash**.

L'interpréteur associe un *job* à chaque tube. Il maintient une table des jobs en cours d'exécution, cette table pouvant être affichée avec la commande **jobs**. Quand **bash** lance un job de manière asynchrone (en *arrière-plan*), il affiche une ligne du genre :

[1] 25647

Cette ligne indique que le job est associé au numéro 1, et que le PID du dernier processus dans le tube est 25647. Tous les processus dans un tube simple font partie du même job. **Bash** utilise la notion de *job* comme une abstraction pour aider à les contrôler.

Pour faciliter l'implémentation d'une interface utilisateur pour le contrôle de job, le système introduit la notion d'ID de groupe de processus associés à un terminal. Les membre de ce groupe de processus (ceux dont le GID est égal au Process Group ID du terminal) reçoivent les signaux déclenchés par le clavier, comme **SIGINT**. Ces processus sont dits en *avant-plan*. Les processus en *arrière-plan* sont ceux dont le Process-GID diffère de celui du terminal ; ils ne reçoivent pas les signaux déclenchés par le clavier. Seuls les processus en avant-plan peuvent lire depuis le terminal ou écrire dessus. Les processus en arrière-plan qui tentent une lecture (ou une écriture) sur le terminal reçoivent les signaux **SIGTTIN (SIGTTOU)** émis par le pilote de périphérique du terminal, lesquels signaux suspendent l'exécution du processus (sauf interception du signal).

Si le système d'exploitation sur lequel **bash** fonctionne permet le contrôle des jobs, **bash** vous y donne accès. La frappe d'un caractère *suspension* (généralement **^Z**, Contrôle-Z) pendant l'exécution d'un processus arrête celui-ci, et redonne la main à **bash**. La frappe d'un caractère *suspension différée* (généralement **^Y**, Contrôle-Y) arrêtera le processus au moment où il tentera de lire depuis le terminal, et la main reviendra à **bash**. Il est possible de manipuler ces jobs en utilisant la commande **bg** (background = arrière-plan) pour continuer l'exécution en arrière-plan, **fg** (foreground) pour la reprendre en avant-plan, ou encore la commande **kill** pour tuer le processus. Un **^Z** prend effet immédiatement, et élimine les sorties en attente.

Il y a plusieurs manières de faire référence à un job. Le caractère **%** introduit un nom de job. Le job numéro *n* peut être noté **%n**. On peut également faire référence à un job en utilisant un préfixe du nom utilise pour le lancer, ou une partie de la ligne de commande. Par exemple, **%ce** se réfère à un job arrêté nomme **ce**. Si ce préfixe correspond à plusieurs jobs, **bash** renverra une erreur. D'autre part, **%?ce**, fait référence à n'importe quel job contenant la chaîne **ce** dans sa ligne de commande. Si la sous-chaîne correspond à plusieurs jobs, **bash** renverra une erreur. Les symboles **%%** et **%+** référencent le *job courant* du shell, c'est-à-dire le dernier job arrêté alors qu'il se trouvait en avant-plan. Le *job précédent* peut être mentionné en utilisant **%-**. Dans les affichages se rapportant aux jobs (par exemple ceux de la commande **jobs** le job courant est toujours indiqué par un **+**, et le job précédent avec un **-**. Un simple % (sans spécification de job) indique également le job courant.

Il suffit de nommer un job pour le ramener en avant-plan : **%1** est synonyme de « **fg %1** », et ramène le job 1 depuis l'arrière-plan vers l'avant-plan. De même, « **%1 &** » relance le job 1, mais en arrière-plan cette fois, de manière équivalente à « **bg %1** ».

Le shell est immédiatement prévenu lorsqu'un job change d'état. Normalement **bash** attend d'être prêt à afficher son symbole d'accueil avant de fournir un rapport sur les changements dans les statut des jobs, afin de ne pas perturber les autres affichages. Si l'option **-b** de la commande interne **set** est active, **bash** affichera les modifications immédiatement. Un détournement de **SIGCHLD** est effectué pour chaque enfant qui se termine.

Si vous essayez de quitter **bash** avec des jobs suspendus, le shell affichera un avertissement. La commande **jobs** permet de vérifier leurs états. Une fois cette commande invoquée, ou si vous essayez à nouveau de quitter le shell immédiatement, il n'y aura plus d'avertissement, et les jobs suspendus seront définitivement terminés.

SYMBOLE D'ACCUEIL (PROMPT)

Quand il est exécuté de manière interactive, **bash** affiche le symbole d'accueil principal **PS1** dès qu'il est prêt à lire une commande, et le symbole d'accueil secondaire **PS2** quand il a besoin de plus de données pour exécuter une commande. **Bash** permet de personnaliser ces chaînes d'accueil, avec des séquences d'échappement ayant les significations suivantes :

\a
: le caractère d'alarme ASCII 07

\d
: la date au format « Jour_de_la_semaine Mois Quantième » (p.ex. : « Tue May 26 »)

\D{*format***}**
: le *format* est passé à **strftime**(3) et le résultat est inséré dans la chaîne de l'invite de command ; un *format* vide implique une représentation du temps spécifique aux paramètres régionaux. Les accolades sont nécessaires.

\e
: le caractère d'échappement ASCII 033

\h
: le nom d'hôte de la machine, juqu'au premier point « . »

\H
: le nom d'hôte complet de la machine

\n
: un saut de ligne

\r
: un retour-chariot

\s
: le nom du shell, c'est-à-dire le nom de base de **$0** (la portion suivant le dernier slash)

\t
: l'heure actuelle au format HH:MM:SS sur 24 heures

\T
: l'heure actuelle au format HH:MM:SS sur 12 heures

\@
: l'heure actuelle sur 12 heures au format HH:MM am/pm

\A
: l'heure actuelle au format HH:MM sur 24 heures

\u
: le nom de l'utilisateur

\v
: la version de **bash (par exemple 2.04)**

\V
: le numéro de version complet (avec niveau de correctifs) de **bash, par exemple (2.04.0)**

\w
: le répertoire de travail en cours, avec **$HOME** abbrégé en tilde (« ~ »)

\W
: le nom de base du répertoire de travail en cours, avec **$HOME** abbrégé en tilde (« ~ »)

\!
: le numéro d'historique de la commande

\#
: le numéro de la commande

\$
: **#** si l'UID effectif est 0, **$** sinon

\nnn
: le caractère de code octal **nnn**

****
: le caractère antislash

\[
: débute une série de caractères non-imprimables, qui permettent d'inclure des séquences de contrôle de terminal dans une chaîne d'accueil

\]
: fin d'une série de caractères non-imprimables

Le numéro de commande, et le numéro d'historique sont généralement différents : le numéro d'historique d'une commande correspond à sa position dans la liste d'historique qui peut contenir des commandes relues depuis cette liste (voir le paragraphe **HISTORIQUE** plus bas), alors que le numéro de commande est simplement une position dans la séquence de commandes exécutées depuis le début de la session shell actuelle. Après que la chaîne d'invite ait été décodée, elle est soumise à l'expansion des paramètres, substitution de commandes, évaluation arithmétique et découpage des mots, sous condition de l'option shell **promptvars** (voir la description de **shopt** au paragraphe **FONCTIONS INTERNES DU SHELL** plus bas).

BIBLIOTHÈQUE READLINE

Il s'agit de la bibliothèque qui gère la saisie lors des sessions interactives du shell, à moins que l'option **-nolineediting** n'ait été choisie. Par défaut les commandes d'édition de ligne sont similaires à celles d'emacs. Il est également possible de disposer d'une interface d'édition du style vi. Pour arrêter l'édition de ligne, utiliser l'option **+o emacs** ou **+o vi** de la commande **set** (voir **FONCTIONS INTERNES DU SHELL** plus bas).

Notation Readline

Dans ce paragraphe on utilisera la notation « à la emacs » pour indiquer les séquences de touches. Les touches Contrôle sont notées C-*touche*, par exemple C-n signifie Contrôle-N. De même, les touches *Méta* sont notées par M-*touche*, ainsi M-x signifie Méta-X. (Sur les claviers sans touches *Méta*, M-*x* signifie ESC *x* (sur les claviers français, ESC est noté Échappement), autrement dit presser la touche Échappement puis la touche *x*. On dit alors qu'ESC est le *préfixe méta*. La séquence M-C-*x* signifie ESC-Contrôle-*x*, ou encore Presser la touche Échappement puis maintenez la touche Contrôle enfoncée tout en pressant la touche *x*.

Les commandes readline peuvent recevoir des *arguments* numériques, qui fonctionnent en règle générale comme compteurs de répétitions. Parfois, néanmoins le signe de l'argument est significatif. L'utilisation d'un argument négatif pour une commande agissant dans un sens donné (par ex. **kill-line**) fait agir la commande en sens inverse. Les commandes ayant un comportement différent sont indiqués plus bas.

Lorsqu'une commande est indiquée comme *détruisant* du texte, le texte effacé est en fait disponible pour une récupération ultérieure. Il est sauvé dans un tampon circulaire. Les suppressions successives accumulent les textes en un seul bloc qui peut être récupéré en une fois. Les commandes non destructrices séparent les blocs de textes dans le tampon circulaire.

Initialisation de Readline

L'affectation par défaut des touches peut être modifiée dans un fichier d'initialisation (*inputrc*). Le nom de ce fichier est lu dans la variable shell **INPUTRC**. Si la variable n'existe pas, le nom par défaut est *~/.inputrc*. Quand un programme utilisant la bibliothèque readline démarre, les fichiers d'initialisation sont lus, et les affectations de touches et de variables sont mises en place. Il n'y a que peu d' constructions possibles dans les fichiers d'initialisation de Readline. Les lignes blanches sont ignorées. Les lignes commençant par **#** sont des commentaires. Les lignes commençant par **$** indiquent des constructions conditionnelles. Les autres lignes sont des affectations de touches ou de variables.

Les affectations par défaut des touches peuvent être modifiés par un fichier *inputrc* d'initialisation. D'autres programmes qui utilisent cette bibliothèque peuvent ajouter leurs propres affectations de touches.

Par exemple, en plaçant

> M-Control-u: universal-argument

ou

> C-Meta-u: universal-argument

dans le fichier *~/.inputrc*, on affectera à la séquence de touche M-C-u la commande readline *universal-argument*.

Les caractères symboliques suivants sont acceptés : *RUBOUT*, *DEL*, *ESC*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *SPC*, *SPACE*, et *TAB*. En plus des noms de commandes, readline permet d'affecter à une touche une chaîne de caractères qui sera insérée lors de la pression (*macro*).

Affectation des touches de Readline

La syntaxe des affectations de touches dans le fichier *~/.inputrc* est simple. Les seules choses nécessaires sont le nom d'une commande, ou le texte d'une macro, et la séquence de touches à laquelle il faut l'affecter. Les touches peuvent être indiquées de deux manières : sous forme de nom symbolique, éventuellement précédé de *Meta-* ou *Control-*, ou sous forme de séquence de touches.

Avec la forme **touche**:*nom_de_fonction* ou *macro*, *touche* est le nom de la séquence de touches épelée en Anglais. Par exemple :

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

Dans ces exemples, *C-u* est assigné à la fonction **universal-argument**, *M-DEL* à la fonction **backward-kill-word**, et *C-o* est affecté à la macro exprimée dans la partie gauche (c'est-à-dire insérera « > sortie » sur la ligne).

Avec la seconde forme, "**touches**":*nom_de_fonction* ou *macro*, **touches** n'est plus un nom symbolique, mais décrit une séquence complète de touches entre guillemets. Certaines séquences d'échappement dans le style GNU Emacs peuvent être utilisées, comme dans l'exemple suivant, mais les noms de caractères symboliques ne sont pas reconnus :

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Touche Fonction 1"
```

Dans cet exemple, *C-u* est à nouveau affecté à la fonction **universal-argument**, *C-x C-r* à la fonction **re-read-init-file**, et *ESC [1 1 ~* insérera le texte « Touche Fonction 1 ». L'ensemble complet des séquences d'échappement est le suivant :

\C-
: préfixe Contrôle

\M-
: préfixe Méta

\e
: caractère d'échappement

****
: barre oblique inverse

\"
: un guillemet " littéral

\aq
: une apostrophe aq littéral

En plus des séquences dans le style GNU Emacs, un second jeu de séquences d'échappement est disponible

\a
: alerte (sonnerie)

\b
: backspace

\d
: effacement

\f
: saut de page

\n
: saut de ligne

\r
: retour-chariot

\t
: tabulation horizontale

\v
: tabulation verticale

*****nnn*
: le caractère 8 bits dont la valeur octale est *nnn* (de un à trois chiffres)

\x*HH*
: le caractère 8 bits dont la valeur hexadécimale est*HH* (un à deux symboles hexadécimaux)

Lorsque l'on saisit le texte d'une macro, il faut utiliser des apostrophes ou des guillemets pour indiquer la définition de la macro. Les textes non protégés sont considérés comme des noms de fonctions. La barre oblique inverse protégera n'importe quel caractère dans la définition d'une macro, y compris " et aq.

Bash permet d'afficher ou de modifier les affectations des touches avec la commande interne **bind**. On peut changer de mode d'édition durant une session interactive en utilisant l'option **-o** de la commande interne **set** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas).

Variables de Readline

Readline dispose de variables permettant de personnaliser encore plus son comportement. Une variable peut être configurée dans le fichier *inputrc* avec une déclaration de la forme

set *nom_de_variable valeur*

Sauf contre-indication, les variables de readline prennent les valeurs **On** ou **Off** (la casse étant sans importance) Les noms de variables non reconnues sont ignorés. Lorsqu'une valeur de variable est lue, les variables vides ou de valeur nulle, « on » (insensible à la casse) et « 1 » sont équivalent à **On**. Toutes les autres valeurs sont équivalentes à **Off**. Les variables, et leurs valeurs par défaut, sont :

bell-style (audible)

Cette variable commande readline lorsqu'il faut déclencher le signal sonore du terminal. Si cette variable contient la valeur **none**, readline n'utilise pas de signal, si elle contient **visible**, on utilise un avertisseur visuel, si elle contient **audible**, readline essaie le signal sonore du terminal.

bind-tty-special-chars (On)
Si elle est positionnée à **On**, readline tente de relier les caractères de contrôle traités de manière particulière par le pilote de terminal du noyau pour leurs équilavents readline.

comment-begin (« # »)
Cette chaîne est insérée quand la commande **readline insert-comment** est exécutée. Elle est affectée à **M-#** en mode emacs, et **#** en mode vi.

completion-ignore-case (Off)
Sur **On**, Readline effectue les comparaisons et complétions de noms de fichiers en différenciant les majuscules et minuscules.

completion-query-items (100)
Lors de la complétion de commandes, si le nombre de possibilités est supérieur à cette valeur, on demandera confirmation à l'utilisateur avant d'afficher la liste des possibilités. Le nombre de complétions possibles est déterminé par la commande **possible-completions**. Il peut s'agir de n'importe quelle valeur entière supérieure ou égale à zéro

convert-meta (On)
Si cette variable contient la valeur **On**, readline transformera en séquences ASCII les caractères ayant leur huitième bit à 1. Il supprimera ce huitième bit, et insérera comme préfixe le caractère d'échappement (en utilisant ce caractère d'échappement comme *préfixe méta*).

disable-completion (Off)
Si cette variable contient **On**, Readline désactivera la complétion des mots. Les caractères de complétion seront insérés sur la ligne comme s'ils avaient été associé à **self-insert**.

editing-mode (emacs)
Cette variable indique si readline doit démarrer avec des affectations de touches similaires à *emacs* ou à *vi*. **editing-mode** peut prendre les valeurs **emacs** ou **vi**.

enable-keypad (Off)
Sur **On**, Readline essayera d'activer le clavier de l'application lorsqu'elle sera invoquée. Sur certains systèmes, cela est nécessaire pour utiliser les touches flèchées.

expand-tilde (Off)
Si cette variable contient la valeur **on**, le développement du tilde est effectué lorsque readline tente une complétion de commandes.

history-preserve-point (Off)
Si cette variable contient la valeur **On**, le code de l'historique tente de placer un point au même endroit sur chaque ligne de l'historique avec **previous-history** ou **next-history**.

horizontal-scroll-mode (Off)
Si cette variable contient la valeur **On**, readline utilisera une seule ligne d'affichage, et la fera défiler horizontalement si elle devient plus longue que la largeur de l'écran, plutôt que de basculer sur une nouvelle ligne.

input-meta (Off)
Si cette variable contient la valeur **On**, readline acceptera des saisies sur huit bits (autrement dit il ne supprimera pas le huitième bit), quelque soit le terminal utilisé. Le nom **meta-flag** est un synonyme pour cette variable.

isearch-terminators (« C-[C-J »)
La chaîne de caractères qui doivent terminer une recherche incrémentale sans que le caractère ne soit exécuté comme une commande. Si cette variable n'a pas de valeur, les caractères *ESC* et *C-J* termineront une recherche incrémentale.

keymap (emacs)
Configure l'affectation des touches. Les jeux d'affectations disponibles sont *emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-command*, et *vi-insert*. *vi* est équivalent à *vi-command*; *emacs* est équivalent à *emacs-standard*. La valeur par défaut est *emacs*; notez que la configuration de la variable **editing-mode** modifie aussi l'affectation par défaut.

mark-directories (On)
Sur **On**, les noms de répertoires sont complétés par un slash.

mark-modified-lines (Off)
Sur **On**, les lignes d'historique qui ont été modifiées seront précédées à l'affichage d'un astérisque (*****).

mark-symlinked-directories (Off)
Si cette variable contient la valeur **On**, complète les noms qui sont des liens vers des répertoires qui ont une barre oblique (sujet à la valeur de **mark-directories**).

match-hidden-files (On)
Si cette variable contient la valeur **On**, readline fait correspondre les fichiers dont le nom commence par un point « . » (fichiers cachés) lors de la réalisation de l'auto-complétion, à moins que le « . » de tête ne soit fourni par l'utilisateur dans le nom de fichier à compléter.

output-meta (Off)
Si cette variable contient la valeur **On**, readline affichera directement les caractères ayant leur huitième bit à 1, plutôt que d'utiliser une séquence d'échappement avec le préfixe méta.

page-completions (On)

Si cette variable contient la valeur **On**, readline utilise un visualisateur interne de style *more* pour afficher un plein écran des possibles complétions.

print-completions-horizontally (Off)
Sur **On**, Readline affichera les possibilités de complétions triés horizontalement dans l'ordre alphabétique plutôt que verticalement.

show-all-if-ambiguous (Off)
Cette variable agit sur la complétion des commandes. Lorsqu'elle contient la valeur **on**, toutes les complétions possibles sont affichées immédiatement plutôt que de déclencher l'avertisseur sonore.

show-all-if-unmodified (Off)
Ceci modifie le comportement par défaut des fonctions de complétion de manière similaire à **show-all-if-ambiguous**. Si cette variable contient **on**, les mots qui ont plus d'une complétion possible sans complétion partielle possible (les complétions possibles ne partagent pas de prefix commun) font que les correspondances seront immédiatement affichées au lieu de faire bipper.

visible-stats (Off)
Sur **On**, un caractère indiquant le type de fichier, comme fourni par l'appel-système **stat**(2) sera ajouté aux noms de fichiers lors de l'affichage des complétions possibles.

Constructions conditionnelles de Readine

Readline propose un mécanisme inspiré des possibilités de compilation conditionnelle en C, qui permet d'effectuer certaines assignations de variables, ou affectations de touches en fonction du résultat d'un test. Il y a quatre directives utilisables.

$if
La structure **$if** permet d'affecter les touches en fonction du mode d'édition, du terminal de saisie, ou de l'application qui utilise la bibliothèque readline. Le texte du test se trouve à la suite du **$if**, jusqu'à la fin de la ligne.

mode
La forme **mode=** de la structure **$if** permet de vérifier si readline est en mode emacs ou en mode vi. Ceci peut être utilisé conjointement à la commande **set keymap**, par exemple pour ne configurer les jeux de touches *emacs-standard* et *emacs-ctlx* que si readline démarre en mode emacs.

term
La forme **term=** peut être utilisée pour ajouter des affectations de touches spécifiques au terminal, par exemple sur des touches de fonction. Le mot à droite du signe **=** est comparé au nom complet du terminal, et à la portion du nom qui précède un éventuel signe **-**. Ceci permet, par exemple, de faire correspondre le mot *sun* aussi bien au terminal *sun* qu'au terminal *sun-cmd*.

application
La forme **application** permet d'inclure des configurations spécifiques à l'application. Chaque programme utilisant la bibliothèque readline configure un *nom d'application*, et un fichier d'initialisation peut dépendre de ce nom. Ceci est utile pour des affecter des touches intéressant un programme spécifique. Ainsi, les commandes suivantes ajoutent une séquence de touches protégeant par des guillemets le mot courant ou le mot précédent, sous Bash :

```
$if Bash
# Protéger le mot actuel ou le précédent
"\C-xq": "\eb\"\ef\""
$endif
```

$endif
Cette commande, comme on l'a vu dans l'exemple ci-dessus, termine une commande **$if**.

$else
Permet d'ajouter des commandes qui seront exécutées si un **$if** échoue.

$include
Cette directive prend un simple nom de fichier en argument et y lit les commandes et les affectations. Par exemple, la directive suivante lit le fichier */etc/inputrc*:

```
$include  /etc/inputrc
```

Recherche

Readline propose des commandes de recherche au sein de l'historique de commandes (voir **HISTORIQUE** plus bas) des lignes contenant une chaîne donnée. Il y a deux modes de recherche : *incrémental* et *non-incrémental*.

Les recherches incrémentales commencent avant même que l'utilisateur ait terminé de saisir la chaîne recherchée. À chaque caractère saisi, Readline affiche l'entrée suivante de l'historique qui corresponde à la chaîne déjà entrée. Une recherche incrémentale nécessite seulement le nombre de caractères indispensable pour retrouver l'entrée d'historique désirée. Les caractères se trouvant dans la variable *isearch-terminators* servent à terminer une recherche incrémentale. Si cette variable n'a pas de valeur, les caractères Escape et Contrôle-J terminent une recherche incrémentale. Contrôle-G permet d'interrompre une recherche incrémentale de reprendre la ligne originale. Lorsque la recherche est finie, l'entrée d'historique contenant la chaîne désirée devient la ligne en cours. Poue trouver une autre entrée d'historique correspondant aussi, il faut taper Contrôle-S ou Contrôle-R. Cela recherchera en avant ou en arrière l'entrée d'historique suivante qui corresponde à la chaîne saisie jusque là. Toute séquence de touche affectée à une commande Readline terminera la recherche et exécutera la commande. Par exemple, un *newline* terminera la recherche et validera la ligne, ce qui exécute la commande de la ligne d'historique.

Readline se souvient de la dernière chaîne de recherche incrémentale. Si deux Contrôle-R sont tapés sans caractère pour définir une nouvelle recherche de chaîne, toute chaîne mémorisée sera utilisée.

Les recherches non-incrémentales lisent la chaîne désirée en entier avant de démarrer la recherche de correspondance dans la liste d'historique. La chaîne peut être saisie par l'utilisateur ou se trouver sur la ligne en cours.

Noms des commandes Readline

Voici une liste des noms de commandes, ainsi que les séquences de touches auxquelles elles sont attribuées par défaut. Les commandes sans séquence indiquée n'ont pas d'affectation par défaut.

Commandes de déplacement

beginning-of-line (C-a)
: Saut au début de la ligne en cours.

end-of-line (C-e)
: Saut à la fin de la ligne.

forward-char (C-f)
: Avancer d'un caractère.

backward-char (C-b)
: Reculer d'un caractère.

forward-word (M-f)
: Avancer jusqu'à la fin du mot suivant. Les mots sont composés de caractères alphanumériques (lettres et chiffres).

backward-word (M-b)
: Reculer jusqu'au début du mot courant, ou du précédent. Les mots sont composés de caractères alphanumériques (lettres et chiffres).

clear-screen (C-l)
: Effacer l'écran, en laissant la ligne courante en haut de l'écran. S'il y a un argument, rafraîchir la ligne en cours sans effacer l'écran.

redraw-current-line
: Rafraîchir la ligne en cours. Par défaut pas d'affectation de touche.

Commandes de manipulation de l'historique

accept-line (Retour-Chariot)
: Valider la ligne, sans se soucier de l'emplacement du curseur. Si la ligne n'est pas vide, l'ajouter à la liste d'historique, sous contrôle de la variable **HISTCONTROL**. Si cette ligne est une modification d'une ligne de l'historique, restaurer cette dernière.

previous-history (C-p)
: Récupérer la commande précédente de la liste d'historique.

next-history (C-n)
: Récupérer la commande suivante dans la liste d'historique.

beginning-of-history (M-<)
: Récupérer la première ligne de l'historique.

end-of-history (M->)
: Retourner à la fin de l'historique, c'est-à-dire à la ligne en cours d'édition.

reverse-search-history (C-r)
: Rechercher en arrière dans l'historique, en remontant. C'est une recherche incrémentale.

forward-search-history (C-s)
: Rechercher en avant dans l'historique, en descendant. C'est une recherche incrémentale.

non-incremental-reverse-search-history (M-p)
: Rechercher une chaîne en arrière dans l'historique, en remontant depuis la ligne actuelle, utiliser une recherche non-incrémentale.

non-incremental-forward-search-history (M-n)
Rechercher une chaîne en avant dans l'historique, utiliser une recherche non-incrémentale.
history-search-forward
Rechercher en avant dans l'historique la chaîne comprise entre le début de la ligne actuelle, et la position du curseur. Par défaut cette commande n'est affectée à aucune touche. Cette une recherche non-incrémentale.
history-search-backward
Rechercher en arrière dans l'historique la chaîne comprise entre le début de la ligne actuelle, et la position du curseur. Cette une recherche non-incrémentale.
yank-nth-arg (M-C-y)
Insérer, à l'emplacement du curseur, le premier argument de la commande précédente. Généralement, il s'agit du second mot de la ligne précédente. Avec un argument *n*, insérer le *n*ième mot de la commande précédente (les mots sont numérotés à partir de 0). Un argument négatif insère le *n*ième mot en partant de la fin de la commande précédente. Une fois que l'argument *n* est traité, l'argument est extrait comme si le développement d'historique « !*n* » avait été spécifié.
yank-last-arg (M-., M-_)
Insérer à la position du curseur le dernier argument de la commande précédente (le dernier mot de la ligne précédente). Avec un argument, se comporte exactement comme **yank-nth-arg**. Les appels successifs à **yank-last-arg** reviennent en arrière dans l'historique en insérant le dernier argument de chaque ligne à chaque fois. Les fonctionnalités de développement d'historique sont utilisés pour extraire le dernier argument, comme si le développement d'historique « !$ » avait été spécifié.
shell-expand-line (M-C-e)
Développer, comme le fait le shell, la ligne en cours. Ceci inclut le remplacement des alias et des directives d'historique, aussi bien que l'ensemble des expansions effectuées par le shell. Voir **EXPANSION DE L'HISTORIQUE** plus bas, pour une description détaillée.
history-expand-line (M-^)
Effectuer l'expansion d'historique sur la ligne en cours. Voir le paragraphe **EXPANSION DE L'HISTORIQUE** plus bas, pour une description détaillée.
magic-space
Effectuer l'expansion de l'historique sur la ligne en cours et insérer une espace. Voir **EXPANSION DE L'HISTORIQUE** plus bas, pour une description détaillée.
alias-expand-line
Effectuer l'expansion des alias sur la ligne en cours. Voir **ALIAS** plus bas pour une description des expansions d'alias.
history-and-alias-expand-line
Effectuer l'expansion de l'historique et des alias sur la ligne en cours.
insert-last-argument (M-., M-_)
un synonyme de **yank-last-arg**.
operate-and-get-next (C-o)
Valider la ligne en cours, et récupérer la ligne suivante de l'historique pour l'édition. Tout argument est ignoré.
edit-and-execute-command (C-xC-e)
Invoque un éditeur sur la ligne de commande courante et exécute le résultat comme une commande shell. **Bash** tente d'invoquer **$FCEDIT**, **$EDITOR**, et *emacs* comme éditeur, dans cet ordre.

Commande d'édition de texte

delete-char (C-d)
Effacer le caractère sous le curseur. Si le curseur est au début de la ligne, si celle-ci est vide, et si le dernier caractère frappé n'était pas **delete-char**, alors la valeur renvoyée est **EOF**.
backward-delete-char (Rubout)
Effacer le caractère derrière le curseur. Lorsque un argument est fourni, le texte effacé est sauvegardé dans le tampon circulaire.
forward-backward-delete-char
Effacer le caractère sous le curseur, sauf s'il est en fin de ligne, auquel cas le caractère derrière le curseur est effacé.
quoted-insert (C-q, C-v)
Ajouter littéralement sur la ligne le caractère frappé suivant. Ceci permet d'insérer des caractères comme **C-q**, par exemple.
tab-insert (C-v TAB)
Insérer un caractère de tabulation.
self-insert (a, b, A, 1, !, ...)
Insérer le caractère mentionné.
transpose-chars (C-t)
Échanger les caractères se trouvant devant et sous le curseur. Le curseur est avancé d'un caractère. À la fin d'une ligne, les caractères échangés sont les deux précédents. Les arguments négatifs ne sont pas acceptés.
transpose-words (M-t)

Échanger le mot où se trouve le curseur et le mot précédent, puis avancer le curseur après ce mot. Si le curseur est à la fin de la ligne, les mots échangés sont les deux derniers de la ligne.

upcase-word (M-u)
Transformer le mot en cours (ou le suivant) en majuscules. Avec un argument négatif, agir sur le mot précédent mais ne pas déplacer le curseur.

downcase-word (M-l)
Transformer le mot en cours (ou le suivant) en minuscules. Avec un argument négatif, agir sur le mot précédent mais ne pas déplacer le curseur.

capitalize-word (M-c)
Passer en majuscule l'initiale du mot en cours (ou du suivant). Avec un argument négatif, agir sur le mot précédent mais ne pas déplacer le curseur.

overwrite-mode
Basculer en mode d'écrasement. Avec un argument numérique positif explicite, bascule en mode d'écrasement. Avec un argument numérique non-positif explicite, bascule en mode d'insertion. Cette commande n'affecte que le mode **emacs** ; le mode **vi** écrase de manière différente. Chaque appel à **readline**(3) démarre en mode d'insertion. En mode d'écrasement, les caractères liés à **self-insert** remplacent le texte sous le curseur plutôt que de le déplacer vers la droite. Les caractères liés à **backward-delete-char** remplacent le caractère avant le curseur par une espace. Par défaut, cette commande n'est pas liée.

Effacement et récupération

kill-line (C-k)
Effacer le texte depuis l'emplacement du curseur jusqu'à la fin de la ligne.

backward-kill-line (C-x C-Rubout)
Effacer en arrière jusqu'au début de ligne.

unix-line-discard (C-u)
Effacer le texte entre le début de la ligne et l'emplacement du curseur. Le texte est sauvegardé dans le tampon circulaire.

kill-whole-line
Effacer toute la ligne sans tenir compte de la position du curseur.

kill-word (M-d)
Effacer le texte entre l'emplacement du curseur et la fin du mot en cours, ou jusqu'à la fin du mot suivant si le curseur est entre deux mots. Les séparations de mots sont les mêmes que celles de **forward-word**.

backward-kill-word (M-Rubout)
Effacer le mot sous le curseur. Les séparations de mots sont les mêmes que celles de **backward-word**.

unix-word-rubout (C-w)
Effacer le mot derrière le curseur, en utilisant l'espace blanc comme limite de mot. Cette limite est différente de celle de **backward-kill-word**. Le texte est sauvegardé dans le tampon circulaire.

unix-filename-rubout
Effacer le mot derrière le curseur, en utilisant l'espace blanc et la barre oblique comme limite de mot. Le texte est sauvegardé dans le tampon circulaire.

delete-horizontal-space (M-\)
Effacer tous les espaces blancs et les tabulations autour du curseur.

kill-region
Détruire le texte entre le curseur et la *marque* (position du curseur sauvegardée précédemment). On appelle ceci une *région*.

copy-region-as-kill
Copier le texte de la région vers le tampon circulaire.

copy-backward-word
Copier le mot précédant le curseur dans le tampon circulaire. Les séparations de mots sont les mêmes que celles de **backward-word**.

copy-forward-word
Copier le mot suivant le curseur dans le tampon circulaire. Les séparations de mots sont les mêmes que celles de **backward-word**.

yank (C-y)
Récupérer le texte au sommet du tampon circulaire.

yank-pop (M-y)
Exécuter une rotation du tampon circulaire, et récupérer le nouveau texte situé à son sommet. Ceci ne fonctionne qu'après **yank** ou **yank-pop**.

Arguments numériques

digit-argument (M-0, M-1, ..., M--)
Ajouter le chiffre aux arguments déjà présents, ou créer un nouvel argument. M-- crée un argument négatif.

universal-argument

Il s'agit d'une autre manière d'indiquer un argument. Si cette commande est suivie d'un ou plusieurs chiffres éventuellement précédés d'un signe moins, ces chiffres consituent un argument. Si la commande est suivie de chiffres, une nouvelle exécution de **universal-argument** termine l'argument numérique, mais est ignorée dans les autres cas. De plus, si la commande est suivie immédiatement par un caractère autre qu'un chiffre ou le signe moins, le compteur d'argument de la commande suivante est multiplié par 4. Le compteur d'argument est initialement à un, ainsi la première exécution de cette commande l'amène à quatre, la seconde à seize, et ainsi de suite.

Complétion

complete (TAB)
Essayer d'effectuer une complétion du texte situé avant le curseur. **Bash** effectue la complétion en considérant successivement le texte comme une variable (s'il commence par **$**), un nom d'utilisateur (s'il commence par **~**), un nom d'hôte (s'il commence par **@**), ou une commande (y compris les alias et les fonctions). Si rien ne fonctionne, il essaye la complétion en nom de fichier.

possible-completions (M-?)
Affiche la liste des complétions possibles pour le texte situé avant le curseur.

insert-completions (M-*)
Insérer avant le curseur l'ensemble des complétions qui seraient fournies par **possible-completions**.

menu-complete
Comme **complete**, mais en remplaçant le mot à compléter par une seule correspondance de la liste des complétions possibles. Des exécutions répétées de **menu-complete** parcourrent la liste des complétions possibles, en insérant une correspondance à chaque fois. À la fin de la liste des complétions, l'alarme sonore est déclenchée (en fonction de la configuration de **bell-style**) et le texte initial est récupéré. Un argument *n* déplace de *n* positions dans la liste de correspondances. Un argument négatif peut être utilisé pour revenir en arrière dans la liste. Cette commande est prévue pour être associée à la touche **TAB**, mais n'est pas affectée par défaut.

delete-char-or-list
Effacer le caractère sous le curseur sauf au début ou en fin de ligne (comme **delete-char**). En fin de ligne, se comporte comme **possible-completions**. Cette commande n'est associée à aucune touche par défaut.

complete-filename (M-/)
Essayer la complétion des noms de fichiers sur le texte situé avant le curseur.

possible-filename-completions (C-x /)
Affiche la liste des complétions possibles en noms de fichiers pour le texte situé avant le curseur.

complete-username (M-~)
Essayer la complétion des noms d'utilisateurs sur le texte situé avant le curseur.

possible-username-completions (C-x ~)
Affiche la liste des complétions possibles en noms d'utilisateurs pour le texte situé avant le curseur.

complete-variable (M-$)
Essayer la complétion des noms de variables shell sur le texte situé avant le curseur.

possible-variable-completions (C-x $)
Affiche la liste des complétions possibles en noms de variables shell pour le texte situé avant le curseur.

complete-hostname (M-@)
Essayer la complétion des noms d'hôtes sur le texte situé avant le curseur.

possible-hostname-completions (C-x @)
Affiche la liste des complétions possibles en noms d'hôtes pour le texte situé avant le curseur.

complete-command (M-!)
Essayer la complétion des commandes sur le texte situé avant le curseur. La complétion des commandes inclut les alias, les mots réservés, les fonctions shell, les fonctions internes, et finalement les exécutables, dans cet ordre.

possible-command-completions (C-x !)
Affiche la liste des complétions possibles en noms de commandes pour le texte situé avant le curseur.

dynamic-complete-history (M-TAB)
Essayer la complétion du texte situé avant le curseur, en recherchant dans la liste d'historique.

complete-into-braces (M-{)
Renvoie entre accolades la liste des complétions possibles en noms de fichiers du texte avant le curseur, afin qu'elle soit utilisable par le shell (voir **Expansion des Accolades** plus haut).

Macros Claviers

start-kbd-macro (C-x ()
Démarre la mémorisation dans la macro en cours des caractères tapés.

end-kbd-macro (C-x))
Arrête la mémorisation dans la macro en cours des caractères tapés, et sauvegarde la définition.

call-last-kbd-macro (C-x e)

Ré-exécute la dernière macro définie, en agissant comme si les caractères contenus dans cette macro étaient tapés au clavier.

Divers

re-read-init-file (C-x C-r)
Lire le contenu du fichier d'initialisation personnalisée, en incorporant les assignations de variables, ou les affectations de touches qui s'y trouvent.

abort (C-g)
Abandonner l'édition en cours et déclencher l'avertisseur sonore (soumis à la configuration de la variable **bell-style**).

do-uppercase-version (M-a, M-b, M-*x*, ..)
Si le caractère précédé de Méta *x* est en minuscule, exécuter la commande correspondant aux caractères majuscules indiqués.

prefix-meta (ESC)
Considérer le prochain caractère tapé comme une séquence Meta. **ESC f** est équivalent à **Meta-f**.

undo (C-_, C-x C-u)
Fonction d'annulation incrémentale, chaque ligne étant mémorisée séparément.

revert-line (M-r)
Annuler toutes les modifications sur cette ligne. Ceci est équivalent à répéter **undo** en nombre suffisant pour ramener la ligne à son état initial.

tilde-expand (M-~)
Effectuer le développement du tilde sur le mot en cours.

set-mark (C-@, M-<space>)
Placer une marque à la position actuelle. Si un argument numérique est fourni, la marque est posée à l'emplacement correspondant.

exchange-point-and-mark (C-x C-x)
Échanger le curseur et la marque. Le curseur est placée à la position sauvegardée précédemment, et l'ancien curseur est mémorisé par la marque.

character-search (C-])
Un caractère est lu et le curseur déplacé à l'occurrence suivante de ce caractère. Un argument négatif recherche les occurrences précédentes.

character-search-backward (M-C-])
Un caractère est lu et le curseur déplacé à l'occurrence précédente de ce caractère. Un argument négatif recherche les occurrences suivantes.

insert-comment (M-#)
Sans argument numérique, la valeur de la variable **readline comment-begin** est insérée au début de la ligne en cours. Si un argument numérique est fourni, cette commande agit comme une « touche à bascule » : si les caractères du début de ligne ne correspondent pas à la valeur de **comment-begin**, la valeur est insérée. Autrement, les caractères dans **comment-begin** sont effacés du début de la ligne Dans les autres cas, la ligne est validée comme si on avait tappé Newline. Ceci sert à transformer la ligne en cours en commentaire du shell. Si un argument numérique provoque l'effacement du caractère de commentaire, la ligne sera exécutée par le shell.

glob-complete-word (M-g)
le mot avant le curseur est traité comme un motif pour un développement de chemin, avec un astérique implicitement ajouté. Le motif est utilisé pour générer une liste de noms de fichier correpondants pour de possibles complétions.

glob-expand-word (C-x *)
Le mot précédant le curseur est considéré comme un motif d'expansion de nom de fichier et la liste des possibilités est insérée à sa place. Si un argument numérique est fourni, un astérisque est ajouté avant le développement du nom de chemin.

glob-list-expansions (C-x g)
La liste des développements qui aurait été produite par **glob-expand-word** est affichée et la ligne est rafraîchie. Si un argument numérique est fourni, un astérisque est ajouté avant le développement du nom de chemin.

dump-functions
Afficher toutes les fonctions et leurs assignations de touches sur le flux de sortie de readline. Si un argument numérique est présent, la sortie est mise en forme pour être compatible avec le format d'un fichier *inputrc*.

dump-variables
Afficher toutes les variables Readline modifiables et leurs valeurs sur le flux de sortie. Si un argument numérique est fournie, la sortie est formatée pour pouvoir être acceptée comme contenu d'un fichier *inputrc*.

dump-macros
Afficher toutes les séquences de touches associées à des macros et les chaînes auxquelles elles correspondent. Un argument numérique réclame que la sortie soit mise en forme pour être compatible avec le format d'un fichier *inputrc*.

display-shell-version (C-x C-v)

Afficher des informations sur la version en cours de **bash**.

Complétion programmable

Lorsqu'une complétion est tentée pour l'argument d'une commande pour laquelle une spécification de complétion a été définie en utilisant la commande interne **complete** (voir **COMMANDES INTERNES DU SHELL** pluss loin), les fonctionnalités de complétion programmable sont invoquées.

En premier lieu, le nom de la commande est identifiée. Si une spécification de complétion a été définie pour cette commande, elle est utilisée pour générer la liste des complétions possible pour le mot. Si le mot commande est un chemin complet, on cherche en premier lieu une spécification de complétion pour le chemin complet. Si aucune spécification de complétion n'est trouvée pour le chemin complet, une tentative est effectuée pour trouver une spécification de complétion pour la portion suivant la barre oblique finale.

Une fois que la spécification de complétion a été trouvée, elle est utilisée pour générer la liste des mots correspondants. Si aucune spécification de complétion n'est trouvée, la complétion par défaut de **bash**, telle qu'elle est décrite plus haut dans le paragraphe **Complétion** est effectuée.

Tout d'abord, les actions définies par la spécification de complétion sont utilisées. Seules les correspondances qui sont préfixées par le mot à compléter sont renvoyées. Lorsque l'option **-f** ou **-d** est utilisée pour la complétion des noms de fichiers ou de répertoire, la variable shell **FIGNORE** est utilisée pour filtrer les correspondances.

Toutes les complétions spécifiées par le motif de développement de nom de fichier de l'option **-G** sont à nouveau générées. Il n'est pas nécessaire que les mots générés par le motif correspondent au mot à compléter. La variable shell **GLOBIGNORE** n'est pas utilisée pour filtrer les correspondances, mais la variable **FIGNORE** est utilisée.

Ensuite, la chaîne spécifiée comme argument à l'option **-W** est prise en compte. Cette chaîne est tout d'abord coupée en utilisant les caractères de la variable spéciale **IFS** comme délimiteurs. La citation (Ndt : quoting) shell est prise en compte. Chaque mot est ensuite développé en utilisant les développements d'accolades, du tilde, des paramètres et des variables, de même que la substitution de commande et le développement arithmétique comme décrit plus haut dans le paragraphe **EXPANSION**. Les résultats sont séparés en utilisant les règles décrites plus haut dans le paragraphe **Séparation des mots**. Les résultats du développement sont comparés au mot à compléter en considérant ce dernier comme préfixe et les mots correspondants deviennent les complétions possibles.

Après que ces correspondances aient été générées, toute fonction shell ou commande spécifiées avec les options **-F** et **-C** sont invoquées. Lorsque la commande ou la fonction est invoquée, les variables **COMP_LINE** et **COMP_POINT** se voient affectées de valeurs comme décrit plus haut dans le paragraphe **Variables shell**. Si une fonction shell est invoqué, les variables **COMP_WORDS** et **COMP_CWORD** sont également remplies. Lorsqu'une fonction ou une commande est invoquée, le premier argument est le nom de la commande dont les arguments doivent être complétés, le second argument est le mot à compléter et le troisième argument est le mot précédent celui devant être complété sur la ligne de commande. Aucun filtrage des complétions générées pour le mot à compléter n'est effectué ; la fonction ou la commande à toute liberté dans la génération des correspondances.

Toute fonction invoquée avec **-F** est invoquée en premier. La fonction peut utiliser toute fonctionnalité du shell, incluant la commande insterne **compgen** décrite plus loin, pour générer les correspondances. Elle doit mettre les complétions possibles dans la variable tableau **COMPREPLY**.

Ensuite, toute commande dpécifiée avec l'option **-C est invoquée dans un environnement équivalent pour la substitution de** commandes. Elle devrait afficher une liste des complétieons possibles, une par ligne, sur la sortie standard. Si nécessaire, une barre oblique inverse doit être utilisée pour protéger un caractère retour chariot.

Après que toutes les complétions possibles aient été générées, un filtre, spécifié avec l'option **-X**, est appliqué à la liste. Ce filtre est un motif comme utilisé dans le développement de noms de chemins ; un **B&** dans le motif est remplacé par une barre oblique inverse ; la barre oblique inverse est retirée avant la tentative de mise en correspondance. Un **B&** littéral doit être protégé par une barre oblique inverse ; la barre oblique inverse est retirée avant la tentative de mise en correspondance. Toute complétion qui correspond au motif sera retirée de la liste. Un **B!** en tête du motif correspondant à la négation du motif ; dans ce cas, toute complétion ne correspondant pas au motif sera retirée de la liste.

Finalement, tout prefix et suffixe spécifié avec les options **-P** et **-S** sont ajoutés à chacun des membres de la liste de complétion, et le résultat est renvoyée au code de complétion de readline comme étant la liste des complétions possibles.

Si les opérations précédemment appliquées ne génèrent aucune correspondance, et si l'option **-o dirnames** a été fournie à **complete** lorsque la spécification de complétion a été définie, la complétion des noms de répertoire est tentée.

Si l'option **-o plusdirs** a été fournie à **complete lors de la définition de la spécification de complétion, la** complétion des noms de répertoires est tentée et toute correspondance est ajoutée aux résultats des autres actions.

Par défaut, si une spécification de complétion est trouvée, tout ce qui est généré est renvoyé au code de complétion comme un ensemble complet des complétions possibles. Les complétions par défaut de **bash** ne sont pas tentées et la complétion par defaut de readline ou la complétion de fichiers est déactivée. Si l'option **-o bashdefault** a été fournie a **complete** lors de la définition de la spécification de complétion, les complétions par défaut de **bash** sont tentées si la spécification de complétion ne génère aucune correspondance. Si l'option **-o default** a été fournie à **complete** lors de la définition de la spécification de complétion, la complétion par défaut de readline sera effectuée si la spécification de complétion (et, si tentée, les complétions par défaut de **bash**) ne génère aucune correspondance.

Lorsqu'une spécification de complétion indique qu'une complétion de nom de répertoire est désirée, les fonctions de complétion programmable forcent readline à ajouter une barre oblique pour compléter les noms qui sont des liens symboliques vers des répertoires, en focntion de la valeur de la variable **mark-directories** de readline, sans tenir compte de la variable **mark-symlinked-directories** de readline.

HISTORIQUE

Quand l'option **-o history** de la commande **set** est active, l'interpréteur donne accès à un *historique des commandes*, c'est-à-dire la liste des commandes précédemment utilisées. Le texte des **HISTSIZE** dernières commandes (par défaut 500) est mémorisé dans l'historique. Le shell les enregistre avant expansion des paramètres et variables (voir le paragraphe **EXPANSION** plus haut) mais après avoir effectué l'expansion historique, sous contrôle des variables **HISTIGNORE** et **HISTCONTROL**. Au démarrage, l'historique est initialisé avec le fichier dont le nom est contenu dans la variable **HISTFILE** (par défaut *~/.bash_history*). **HISTFILE** est tronqué si nécessaire, pour ne pas dépasser **HISTFILESIZE** lignes. Lorsqu'un shell interactif se termine, les **HISTSIZE** dernières lignes de l'historique sont copiées dans le fichier **HISTFILE**. Si l'option shell **histappend** (voir la description de **shopt** dans **COMMANDES INTERNES DU SHELL** plus bas) est active, les lignes sont ajoutées au fichier d'historique, sinon le fichier est écrasé. Si la variable **HISTFILE** n'existe pas, ou si le fichier n'est pas accessible en écriture, l'historique n'est pas enregistré. Après la sauvegarder, le fichier d'historique est tronqué pour ne pas contenir plus de **HISTFILESIZE** lignes. Si **HISTFILESIZE** n'est pas configurée, aucune troncature n'a lieu.

La commande interne **fc** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas) permet d'afficher, d'éditer, ou de ré-exécuter une partie de l'historique. La commande interne **history** permet d'afficher l'historique et de manipuler le fichier d'historique. Quelque soit le mode d'édition en ligne de commande, des commandes de recherche sont disponibles pour accéder à l'historique.

Le shell permet de configurer quelles commandes seront sauvées dans la liste d'historique. Les variables **HISTCONTROL** et **HISTIGNORE** peuvent demander au shell de ne sauvegarder qu'une partie des commandes saisies. L'option shell **cmdhist** si elle est active, demande au shell d'essayer de sauver chaque ligne d'une commande multiligne dans la même entrée de la liste, en ajoutant des points-virgules si nécessaire pour préserver l'exactitude de la syntaxe. L'option shell **lithist** demande de sauvegarder les commandes avec des sauts de lignes incorporés plutôt que des points-virgules. Voir la description de la commande **shopt** dans **COMMANDES INTERNES DU SHELL** pour des détails sur la configuration des options shell.

EXPANSION DE L'HISTORIQUE

Le shell propose des possibilités d'expansion de l'historique qui sont semblables à celles de **csh.** Ce paragraphe décrit la syntaxe de ces commandes. Elles sont utilisables par défaut dans les shells interactifs, il est toutefois possible de les désactiver en utilisant l'option **+H** de la commande interne **set** (voir le paragraphe **COMMANDES INTERNES DU SHELL** plus bas). Les shells non-interactifs n'effectuent pas d'expansion d'historique par défaut.

L'expansion de l'historique permet d'insérer des mots de l'historique dans le flux d'entrée, facilitant la répétition de commande, l'insertion des arguments d'une ligne antérieure dans la saisie en cours, ou la correction rapide d'erreur de frappe dans une commande précécente.

Le développement de l'historique est fait immédiatement après la lecture d'une ligne complète, avant que le shell ne fasse la séparation des mots. Il se déroule en deux parties. La première consiste à déterminer la ligne de l'historique à utiliser pour les substitutions, la seconde permet de sélectionner les parties de la lignes à inclure dans la nouvelle ligne. La ligne sélectionnée dans l'historique est l'*événement*, et les parties de cette ligne à réutiliser sont des *mots*. Plusieurs *modificateurs* permettent de manipuler les mots sélectionnés. La ligne est découpée en mots de la même manière que lors d'une saisie, ainsi plusieurs mots séparés par des *méta-caractères*, et protégés par des guillemets sont considérés comme un unique mot. L'expansion de l'historique est introduite par l'apparition du caractère d'expansion, qui se trouve être **!** par défaut. Seule la barre oblique inverse (****) et les apostrophes permettent de protéger le caractère d'échappement de l'historique, par défaut **!**.

Plusieurs caractères inhibent le développement de l'historique s'ils se trouvent immédiatement après le caractère de développement de l'historique, même s'il n'est pas coté : espace, tabulation, nouvelle-ligne, retour-chariot et **=**. Si l'option shell **extglob** est active, **(** inhibera l'expansion.

Plusieurs options shell, configurables aved la commande interne **shopt** permettent d'ajuster le comportement de l'expansion de l'historique. Si l'option shell **histverify** est activée (voir la description de la commande interne **shopt**) et si **readline** est utilisée, les substitutions d'historiques ne sont pas transmises tout de suite à l'analyseur du shell. À la place, la ligne développée est rechargée dans la mémoire tampon de **readline** pour permettre des modifications ultérieures. Si **readline** est utilisée et si l'option **histreedit** est activée, une substitution d'historique qui échoue est rechargée dans la mémoire tampon de **readline** pour correction. L'option **-p** de la commande interne **history** peut servir à examiner ce qu'une substituion donnera avant de l'employer réellement. L'option **-s** de la commande interne **history** permet d'ajouter des commandes à la fin de la liste d'historique sans les les invoquer effectivement, afin qu'elles soient disponibles pour des rappels ultérieurs.

Le shell offre le contrôle des divers caractères utilisés par le mécanisme d'expansion de l'historique (voir la description de **histchars** plus haut dans le paragraphe **Variables du Shell**).

Indicateur d'événement

Un indicateur d'événement est une référence à une ligne de commande dans la liste d'historique.

!
: Débute une substitution de l'historique, sauf s'il est suivi par un **blanc**, nouvelle-linge, retour-chariot, = ou ((lorsque l'option shell **extglob** est active en utilisant la commande interne **shopt**).

!*n*
: Se réfère à la ligne de commande *n*.

!-*n*
: Se réfère à la ligne de commande actuelle moins *n*.

!!
: Est une référence à la commande précédente. C'est un synonyme de « `!-1 ».

!*chaîne*
: Référence la plus récente commande commençant par la *chaîne*.

!?*chaîne***[?]**
: Se réfère à la plus récente commande contenant la *chaîne*. Le **?** final peut être omis si la *chaîne* est suivie immédiatement d'un retour-chariot.

^*chaîne_1***^***chaîne_2***^**
: Substitution rapide. Répéter la dernière commande en remplaçant la *chaîne_1* par la *chaîne_2*. Équivalent à « !!:s/*chaîne_1*/*chaîne_2*/ » (voir **Modificateurs** plus bas).

!#
: La ligne entière tapée jusqu'à présent.

Indicateurs de mots

Ces indicateurs servent à sélectionner les mots désirés dans un événement. Un **:** sépare l'indicateur d'événements de l'indicateur de mot. On peut l'omettre si l'indicateur de mot commence par **^**, **$**, *****, **-**, ou **%**. Les mots sont numérotés depuis le début de la ligne, le premier étant note 0 (zéro). Les mots sont insérés dans la ligne courante, séparés par des espaces.

0 (zéro)
: Le zéro-ième mot. Pour le shell, c'est le nom de la commande.

n
: Le *n*-ième mot.

^
: Le premier argument, c'est-à-dire le mot 1.

$
: Le dernier argument.

%
: Le mot correspondant à la recherche la plus récente « ?*chaîne*? ».

*x***-**y
: Un intervalle de mots, « -*y* » permet d'abréger « 0-*y* ».

*****
: Tous les mots sauf le zéro-ième. C'est un synonyme pour « *1-$* ». Même s'il n'y a qu'un mot, ***** ne déclenche pas d'erreur, simplement la chaîne vide est renvoyée.

x*
: Abréviation de *x-$*.

x-
: Abréviation de *x-$* comme **x***, mais sans le dernier mot.

Si un indicateur de mot est founi sans indication d'événement, la dernière commande est utilisée.

Modificateurs

Après l'éventuel indicateur de mot, vous pouvez ajouter une séquence de modificateurs, chacun précédé par un « : ».

h
: Élimine la fin du chemin d'accès, ne laissant que son début.

t
: Supprimer tout le début d'un chemin d'accès, en ne laissant que la fin.

r
: Supprime un suffixe final de la forme *.xxx*, en ne laissant que le nom de base.

e
: Tout supprimer sauf le suffixe final.

p
: Afficher la nouvelle commande sans l'exécuter.

q
: Protéger les mots substitués, pour éviter les substitutions futures.

x
: Protéger les mots substitués, comme avec **q**, mais en faisant la séparation des mots sur les **blancs** et les retour-chariots.

s/*ancien*/*nouveau*/
: Remplacer par *nouveau* la première occurrence de *ancien* sur la ligne d'événement. N'importe quel délimiteur peut être utilisé à la place de /. Le dernier délimiteur est facultatif si c'est le dernier caractère de la ligne d'événement. Le délimiteur peut être protégé dans *ancien* et *nouveau* par une simple barre oblique inverse. Si & apparaît dans *nouveau*, il est remplacé par *ancien*. Une simple barre oblique inverse protégera le &. Si *ancien* est nul, il est remplacé par le dernier *ancien* substitué, ou si aucune substitution n'a eu lieu précédemment, la dernière *chaîne* dans une recherche **!?***chaîne***[?]**.

&
: Répéter la substitution précédente.

g
: Appliquer les changements sur la ligne d'événement entière. On utilise ceci conjointement à « **:s** » (par exemple « **:gs/***ancien*/*nouveau*/ ») ou « **:&** ». S'il est utilisé avec « **:s** », n'importe quel délimiteur peut être utilisé à la place de « / », et le dernier délimiteur est facultatif si c'est le dernier caractère de la ligne d'événement. Un **a** peut être utilisé en tant que synonyme de **g**.

G
: Appliquer le modificateur « **s** » suivant une fois à chacun des mots dans la ligne d'événements.

COMMANDES INTERNES DU SHELL

Sauf indication contraire, toute commande décrite comme acceptant des options précédées de **-** accepte également la notation **--** pour signifier la fin de la liste d'options. Par exemple, les commandes internes **:**, **true**, **false**, and **test** n'acceptent aucune option. do not accept options.

: [*arguments*]
: Pas d'effet. Cette commande ne fait rien d'autre que l'expansion des *arguments* et la mise en place des redirections. Le code de retour est nul.

. *fichier* [*arguments*]

source *fichier* [*arguments*]
: Lire et exécuter les commandes contenues dans le *fichier* avec l'environnement du shell en cours, puis renvoyer le code de retour de la dernière commande exécutée dans le *fichier*. Si le nom du *fichier* ne contient pas de slash, les chemins d'accès contenus dans **PATH** sont parcourus pour rechercher le répertoire contenant le *fichier*. Lorsqu'il est recherché dans le **PATH** le fichier n'a pas besoin d'être exécutable. Lorsque **bash** n'est pas en *mode posix*, e répertoire de travail en cours est finalement examiné si aucun fichier n'est trouvé dans le **PATH**. Si l'option **sourcepath** de la commande interne **shopt** est désactivée, le **PATH** n'est pas parcouru. Si des *arguments* sont fournis, ils sont transmis dans les paramètres positionnels lorsque le *fichier* est exécuté. Sinon les paramètres positionnel ne sont pas modifiés. Le code de retour est celui de la dernière commande exécutée au sein du script (0 si aucune commande n'est exécutée), et faux si le *fichier* n'est pas trouvé.

alias [**-p**] [*nom*[=*valeur*] ...]
: La commande **alias**, sans argument ou avec l'option **-p** affiche sur la sortie standard la liste des alias actuellement définis, sous la forme *nom=valeur*. Lorsque l'on fournit des arguments, un alias est créé pour chaque *nom* auquel on a donné une *valeur*. Un espace final dans une *valeur* forcera la recherche d'alias dans le mot suivant lors de l'expansion de l'alias. Pour chacun des *noms* pour lesquels aucune *valeur* n'est fournie dans la liste d'arguments, le nom et la valeur de l'alias sont affichés. La commande **Alias** renvoie vrai, sauf si l'on indique un *nom* pour lequel aucun alias n'a été défini.

bg [*job* ...]
Relance en arrière-plan chaque *job* suspendu, comme s'il avait été lancé avec **&**. Si aucun *job* n'est précisé, le job en cours est basculé en arrière-plan. **bg** *job* renvoie 0 si le contrôle des jobs est désactivé, ou si le *job* n'existe pas, ou encore s'il a été lancé sans contrôle de jobs.

bind [**-m** *jeu_de_touches*] [**-lpsvPSV**]
bind [**-m** *jeu_de_touches*] [**-q** *nom*] [**-u** *fonction*] [**-r** *séquence*]
bind [**-m** *jeu_de_touches*] **-f** *fichier*
bind [**-m** *jeu_de_touches*] *séquence*:*fonction*
bind *readline-command*
Affiche les affectations de touches **readline** pour une fonction, ou affecte une séquence de touches à une fonction ou une macro **readline** ou remplit une variable **readline**. Chaque argument qui n'est pas une option est une commande comme elle devrait apparaître dans *.inputrc*, mais chaque affectation ou commande doit être fournie sous forme d'argument indépendant, par exemple « "\C-x\C-r": re-read-init-file ». Les éventuelles options ont les significations suivantes :

-m *jeu_de_touches*
Modifier la configuration du *jeu_de_touches* indiqué. Les noms des *jeux_de_touches* acceptés 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
Afficher la liste des noms de toutes les fonctions **readline**
-p
Affiche les noms des fonctions **readline** et leurs affectations de manière à pouvoir les relire.
-P
Affiche les noms des fonctions **readline** et leurs affectations.
-v
Affiche les noms des variables **readline** et leurs valeurs de manière à pouvoir les relire.
-V
Affiche les noms des variables **readline** et leurs valeurs.
-s
Affiche les séquences de touches **readline** affectées aux macros et les chaînes correspondantes, de manière à pouvoir les relire.
-S
Affiche les séquences de touches **readline** affectées aux macros et les chaînes correspondantes.
-f *fichier*
Lire les affectations dans le *fichier*
-q *fonction*
Rechercher la touche permettant d'invoquer la *fonction* indiquée
-u *fonction*
Désaffecter toutes les touches correspondant à la *fonction*.
-r *séquence*
Supprimer les affectations pour la *séquence* de touches.

La valeur de retour est nulle, sauf si une option invalide est fournie, ou si une erreur se produit.

break [*n*]
Sortir d'une boucle **for**, **while**, ou **until**. Si *n* est fourni, sortir de *n* niveaux de boucles. *n* doit être > 1. Si *n* est supérieur au nombre de boucles imbriquées, on sort de l'ensemble d'entre elles. La valeur de retour est nulle, sauf si le shell n'exécute pas de boucle lorsque le **break** est atteint.

builtin *commande_interne* [*arguments*]
Exécuter la *commande interne* du shell indiquée, en lui passant les *arguments*, puis renvoyer son code de retour. Ceci est utile lorsque l'on veut définir une fonction ayant le même nom qu'une commande interne, mais que l'on a besoin d'invoquer cette commande depuis l'intérieur de la fonction. On redéfinit souvent la commande interne **cd** ainsi. Le code de retour est faux si la *commande_interne* n'existe pas.

cd [**-L**|**-P**] [*rep*]
Remplacer le répertoire de travail en cours par *rép*. La variable **HOME** est la valeur par défaut de *rep*. La variable **CDPATH** définit des chemins de recherche pour trouver le répertoire contenant *rép*. Les répertoires possibles y sont fournis dans une liste séparées par des doubles-points « : ». Un nom de répertoire nul dans **CDPATH** est équivalent au répertoire en cours, c'est-à-dire « **.** ». Si *rép* commence par une barre oblique (*/*), alors **CDPATH** n'est pas utilisé. L'option **-P** demande de n'employer que l'enchaînement réel des répertoires sans suivre les liens symboliques (voir aussi l'option **-P** de la commande interne **set** plus bas). L'option **-L** force le suivi des liens symboliques. Un argument **-** est équivalent à **$OLDPWD**. Si un nom de répertoire non vide de **CDPATH** est utilisé, ou si **-** est le premier argument et si le changement de répertoire réussit, le chemin absolu du nouveau répertoire de travail est écrit sur la sortie standard. La valeur de retour est vraie si l'on a pu changer le répertoire de travail, et fausse sinon.

caller [*expr*]
Renvoie le contexte d'un appel d'une sous-routine active (une fonction shelle ou un script lancé avec les commandes internes **.** ou **source** . Sans *expr*, **caller** affiche le numéro de ligne et le nom de fichier de la sous-routine courante. Si un entier non négatif est fourni comme *expr*, **caller** affiche le numéro de ligne, le

nom de la sous-routine et le nom de fichier source correspondant à la position dans la pile d'appel d'exécution courante. Cette information supplémentaire peut être utilisée, par exemple, pour afficher une trace de la pile. La trame courante est la trame 0. La valeur renvoyée est 0 à moins que le shell n'exécute pas de sous-routine ou que *expr* ne corresponde pas à une position valide dans la pile d'appel.

command [**-pVv**] *commande* [*arguments* ...]

Lancer la *commande* avec les *arguments* indiqués, mais en simplifiant le processus habituel de recherche des commandes. Seules les commandes internes, et les commandes disque trouvées dans le **PATH** seront exécutées. Si l'option **-p** est fournie, la recherche de la *commande* utilisera une valeur par défaut du **PATH** garantissant de trouver tous les utilitaires standards. Si l'option **-V** ou l'option **-v** sont fournies, une description de la *commande* est affichée. L'option **-v** n'affiche qu'un simple mot indiquant la commande interne, ou le chemin d'accès utilisé pour invoquer la *commande*. L'option **-V** fournit une description plus volubile. Si les options **-V** ou **-v** sont fournies, le code de retour est 0 si la *commande* est trouvée, et 1 sinon. Si aucune option n'est fournie, et si une erreur se produit, ou si la *commande* est introuvable, le code de retour est 127. Sinon, le code de retour de cette commande interne est celui de la *commande* invoquée.

compgen [*option*] [*mot*]

Génère les correspondances des complétions possibles pour *mot* en concordance avec les *option*s, qui peuvent être toute option acceptée par la commande interne **complete** à l'exception de **-p** et **-r ,** et écrit le résultat sur la sortie standard. Lorsqu'on utilise les options **-F** ou **-C ,** les diverses variables shell remplies par les fonctionnalités de complétion programmable, lorsqu'elles sont disponibles, n'auront pas de valeurs utiles.

Les correspondances seront générées de la même façon que si le code de complétion les avait générées directement à partir d'une spécification de complétion avec les mêmes attributs. Si *mot* est spécifié, seules les complétions qui correspondent à *mot* sont affichées.

La valeur renvoyée est vrai, à moins qu'une option non valide ait été fournie ou qu'aucune correspondance n'ait été générée.

complete [**-abcdefgjksuv**] [**-o** *option-complétion*] [**-A** *action*] [**-G** *globpat*] [**-W** *liste_de_mots*] [**-P** *prefix*] [**-S** *suffix*]

[**-X** *motif_filtre*] [**-F** *fonction*] [**-C** *commande*] *nom* [*nom* ...]

complete -pr [*nom* ...]

Spécifie comment les arguments de chaque *nom* doivent être complétés. Si l'option **-p** est fournie, ou si aucune option n'est fournie, les spécifications de complétion existantes sont affichés de telle manière qu'elles puissent être réutilisées comme entrée. L'option **-r** retire une spécification de complétion à chaque *nom* fourni, ou, si aucun *nom* n'est fourni, toutes les spécifications de complétions

Le processus d'application de ces spécifications de complétion lorsqu'une complétion de mot est tenté est décrit plus haut dans le paragraphe **Complétion**programmable**.**

Les autres options, si elles sont spécifiées, ont les significations suivantes. Les arguments des options **-G**, **-W** et **-X** (et, si nécessaire, les options **-P** et **-S)** doivent être cotées pour les protéger du développement avant que la commande interne **complete** ne soit invoquée.

-o *option-complétion*

L'*option-complétion* contrôle plusieurs aspects du comportement des spécifications de complétion au-delà de la simple génération de complétion. *option-complétion* peut être l'une des suivantes :

bashdefault

Effectuer le reste des complétions par défaut de **bash** si la spécification de complétion ne génère aucune correspondance.

default

Utiliser la complétion de nom de fichier par defaut de readline si la spécification de complétion ne génère aucune correspondance.

dirnames

Utiliser la complétion de nom de répertoire si la spécification de complétion ne génère aucune correspondance.

filenames

Indiquer à readline que la spécification de complétion génère les noms de fichiers, ainsi il peut effectuer tout traitement spécifique aux noms de fichiers (comme ajouter une barre oblique aux noms de répertoires, supprimer les espaces de fin de ligne). Conçue pour être utiliser avec les fonctions shell.

nospace

Indiquer à de ne pas ajouter d'espace à la fin de la ligne (comportement par défaut) aux mots complétés.

plusdirs

Après que toutes les correspondances définies par la spécification de complétion aient été générées, une tentative de complétion de nom de

répertoire est effectuée et toutes les correspondances sont ajoutées aux résultats des autres actions.

-A *action*

L'*action* peut être l'une des suivantes pour générer une liste des complétions possible :

alias
Noms d'alias. Peut également être spécifiée avec **-a**.

arrayvar
Noms des variables de type tableau.

binding
Nom des raccourcis clavier de **readline**.

builtin
Noms des commandes internes du shell. Peut également être spécifiée avec **-b**.

command
Nom des commandes. Peut également être spécifiée avec **-c**.
Noms des répertoires. Peut également être spécifiée avec **-d**.

disabled
Noms des commandes internes du shell inactives.

enabled
Noms des commandes internes du shell actives.

export
Noms des variables du shell exportées. Peut également être spécifiée avec **-e**.

file
Noms des fichiers. Peut également être spécifiée avec **-f**.

function
Noms des fonctions shell.

group
Noms des groupes. Peut également être spécifiée avec **-g**.

helptopic
Sujets de l'aide acceptés par la commande interne **help**.

hostname
Noms d'hôtes tels qu'ils sont extraits du fichier spécifié par la variable shell **HOSTFILE**.

job
Noms des tâches si le contrôle des tâches est actif. Peut également être spécifiée avec **-j**.

keyword
Mots réservés du shell. Peut également être spécifiée avec **-k**.

running
Noms des tâches en cours d'exécution si le contrôle de tâches est actif.

service
Noms des services. Peut également être spécifiée avec **-s**.

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

shopt
Noms des options shell acceptées par la commande interne **shopt**.

signal
Noms des signaux.

stopped
Noms des tâches arrêtées si le contrôle de tâches est actif.

user
Noms des utilisateurs. User names. Peut également être spécifiée avec **-u**.

variable
Noms de toutes les variables shell. Peut également être spécifiée avec **-v**.

-G *globpat*
Le motif de développement des noms de fichiers *globpat* est développé pour générer les complétions possibles.

-W *liste_de_mots*
La *liste_de_mots* est découpée en utilisant les caractères de la variable spéciale **IFS** comme délimiteurs et chaque mot résultant est développé. Les complétions possibles sont les membres de la liste résultante qui correspondent au mot en cours de complétion.

-C *commande*
La *commande* est exécutée dans un sous-shell et sa sortie est utilisée en tant que complétions possibles.

-F *fonction*
La *fonction* shell est exécutée dans le shell courant. Lorsqu'elle s'achève, les complétions possibles sont récupérées à partir de la valeur de la variable de type tableau **COMPREPLY**.

-X *motif_filtre*
motif_filtre est un motif utilisé pour le développement des noms de fichiers. Il est appliqué à la liste des complétions possibles générée par les options et arguments précédents et chaque complétion correspondant à *motif_filtre* est supprimée de la liste. Un **!** au début du *motif_filtre* inverse le motif ; dans ce cas, toute complétion qui ne correspondant pas à *motif_filtre* est supprimée.
-P *prefix*
prefix est ajouté au début de chaque complétion possible après que toutes les autres options aient été appliquées.
-S *suffix*
suffix est ajouté à chaque complétion possible après que toutes les autres options aient été appliquées.

La valeur de retour est vraie à moins qu'une option non valide ait été fournie, qu'une option comme **-p** ou **-r** ait été fournie sans un *nom* d'argument, qu'une tentative ait été faite pour supprimer une spécification de complétion pour un *nom* pour lequel il n'existe pas de spécification, ou bien qu'une erreur ne soit survenue lors de l'ajout d'une spécification de complétion.

continue [*n*]
Reprendre l'exécution à l'itération suivante d'une boucle **for**, **while**, **until**, ou encore **select**. Si *n* est indiqué, reprendre l'exécution à la *n*-ième boucle imbriquée. *n* doit être > 1. Si *n* est supérieur au nombre de boucles imbriquées, l'exécution reprend à la boucle la plus externe (« top-level »). La valeur de retour est 0 sauf si le shell n'exécutait pas de boucle quand il a rencontré l'instruction **continue**.
declare [**-afFirtx**] [**-p**] [*nom*[=*valeur*] ...]
typeset [**-afFirtx**] [**-p**] [*nome*[=*valeur*] ...]
Déclarer des variables et/ou leur fournir des attributs. Si aucun *nom* n'est mentionné, les valeurs des variables sont affichées. L'option **-p** affichera les attributs et les valeurs de chaque *nom*. Quand **-p** est utilisée, les autres options sont ignorées. L'option **-F** évite l'affichage des définitions de fonctions, seuls les noms des fonctions et leurs attributs sont affichés. Si l'option shell **extdebug** est activé avec **shopt**, le nom du fichier source et le numéro de ligne où apparaît la fonction sont affichés. L'option **-F** implique l'usage de **-f .** Les options suivantes permettent de restreindre l'affichage aux variables dotées de certains attributs, ou de fixer les attributs.

-a
Chaque *nom* est une variable de type tableau (voir le paragraphe **Tableaux** plus haut).
-f
Seulement les noms de fonctions
-i
La variable est traitée comme un entier. L'évaluation arithmétique (voir **ÉVALUATION ARITHMÉTIQUE)** est effectuée lorsque la variable reçoit une valeur.
-r
Rend les *nom*s accessibles en lecture seulement. On ne pourra plus modifier les valeurs de ces noms lors d'assignations ultérieures.
-t
Donne à chaque *nom* l'attribut *trace*. Les fonctions tracées héritent des captures **DEBUG** et **RETURN** du shell appelant. L'attribut *trace* n'a pas de signification spéciale pour ces variables.
-x
Marquer le *nom* pour qu'il soit exporté dans l'environnement.

En utilisant « + » à la place de « - », l'attribut correspondant est désactivé. Dans une fonction, chacun des *noms* devient local, comme avec la commande **local**. Si un nom de variable est suivi par =*valeur*, cette variable est remplie avec *valeur*. La valeur de retour est 0 sauf si une option illégale est rencontrée, si on essaye de définir une fonction en utilisant « -f foo=bar », si on tente d'écrire dans une variable en lecture seule, si on essaye d'affecter une valeur a une variable tableau sans employer la syntaxe composée (voir **Tableaux** plus haut), si l'un des *noms* n'est pas un nom légal pour le shell, si on essaye de supprimer l'attribut lecture-seule d'une variable, si on essaye de supprimer l'attribut tableau d'une variable, ou si on essaye d'afficher avec -f une fonction inexistante.

dirs [-clpv] [**+/-n**]
Affiche la liste des répertoires actuellement mémorisés. L'affichage par défaut se fait sur une seule ligne, les noms de répertoires étant séparés par des espaces. On ajoute des répertoires dans cette liste avec la commande **pushd**, et on peut en supprimer avec la commande **popd**.

+ *n*
affiche le *n*-ième élément de la liste fournie par **dirs** sans options, en comptant à partir de la gauche en commençant à zéro.
- *n*
affiche le *n*-ième élément de la liste fournie par **dirs** sans options, en comptant à partir de la droite en commençant à zéro.
-c
Effacer la liste des répertoires en les supprimant tous.

-l
affiche une liste longue, le format par défaut utilisant le tilde pour représenter le répertoire personnel.
-p
Afficher la liste des répertoires avec un élément par ligne.
-v
Afficher la liste des répertoires avec un élément par ligne, en préfixant chaque entrée avec son numméro d'index dans la liste.

La valeur de retour est 0 sauf si une option illégale a été trouvée, ou si l'index *n* dépasse la longueur de la pile des répertoires.

disown [**-ar**] [**-h**] [*job* ...]
Sans options, chaque *job* est supprimé de la table des jobs actifs. Si l'option **-h** est fournie, chaque *job* n'est pas supprimé mais est marqué pour que le signal **SIGHUP** ne lui soit pas envoyé si le shell reçoit un signal **SIGHUP**. Si aucune *job* n'est indiqué, et si ni l'option **-a** ni l'option **-r** ne sont fournies, le *job en cours* est concerné. Si aucun *job* n'est indiqué, l'option **-a** signifie la suppression ou le marquage de tous les jobs ; l'option **-r** sans spécification de *job* restreint les opérations au job en cours. La valeur renvoyée est zéro sauf si un *job* indiqué n'est pas valide.

echo [**-neE**] [*arguments* ...]
Affiche ses *argumentss*, séparés par des espaces. La valeur de retour est toujours nulle. Avec l'option **-n**, le retour-chariot final est supprimé. Avec l'option **-e** l'interprétation des séquences d'échappement est activée. L'option **-E** désactive l'interprétation des séquences d'échappement, même sur les systèmes où elles sont interprétées par défaut. **echo** n'interprète pas la séquence **--** comme une marque de fin des options. **echo** reconnaît les séquences d'échappement suivantes :

\a
alerte (avertisseur sonore)
\b
backspace
\c
suppression du retour-chariot final
\e
caractère d'échappement
\f
saut de page
\n
nouvelle ligne
\r
retour-chariot
\t
tabulation horizontale
\v
tabulation verticale
****
contre oblique
\0*nnn*
le caractère 8 bits dont la valeur en octal vaut *nnn* (un à trois chiffres)
\x*HH*
le caractère 8 bits dont la valeur en hexadécimal vaut *HHn* (un à deux caractères hexadécimaux)

enable [**-adnps**] [**-f** *fichier*] [*nom* ...]
Active ou inhibe les commandes internes du shell. Ceci permet d'exécuter une commande disque ayant le même nom qu'une commande interne du shell, sans avoir besoin d'indiquer le chemin d'accès complet. Avec l'option **-n**, chaque *nom* indiqué est inhibé, sinon ils sont activés. Par exemple, pour utiliser le programme binaire **test** qui se trouve dans le **PATH** plutôt que la commande interne, utilisez « enable -n test ». L'option **-f** demande de charger la commande interne avec le *nom* indiqué depuis le *fichier* objet partagé sur les systèmes qui le permettent. L'option **-d** permet de supprimer une commande interne chargée précédemment avec **-f**. Si aucun *nom* n'est fourni, ou si l'option **-p** est indiquée, la liste des commandes internes du shell est affichée. Sans autre option, la liste des commandes actives est imprimée. Avec l'option **-n**, seules les commandes désactivées sont affichées. Si l'option **-a** est fournie, la liste de toutes les commandes est affichée, avec une indication des activations et inhibitions. Si on indique **-s**, la sortie est limitée aux commandes internes spéciales POSIX. La valeur de retour est 0, sauf si *nom* n'est pas une commande interne, ou s'il y a un problème avec le chargement d'une nouvelle commande.

eval [*arguments* ...]
Les *arguments* sont lus et regroupés en une seule commande simple. Cette commande est alors lue et exécutée par le shell, et son code de retour est renvoyée comme valeur de la commande **eval**. S'il n'y a pas d'*arguments*, ou uniquement des arguments nuls, **eval** renvoie vrai.

exec [**-cl**] [**-a** *nom*] [*commande* [*arguments*]]

Si une *commande* est indiquée, elle remplace le shell. Aucun nouveau processus n'est créé. Les *arguments* deviennent ceux de la *commande*. Si l'option **-l** est fournie, l'interpréteur place un tiret en guise de zéro-ième argument fourni a la *commande*. C'est ce que fait *login*. L'option **-c** permet d'exécuter la *commande* avec un environnement vide. Si l'option **-a** est indiquée, le shell transmet le *nom* comme zéro-ième argument. Si la *commande* Si la commande ne peut être exécutée, quelqu'en soit la raison, un shell non-interactif se termine, sauf si l'option shell **execfail** est active, auquel cas une erreur est renvoyée. Un shell interactif renvoie l'erreur si le fichier ne peut pas être exécuté. Si aucune *commande* n'est indiquée, les éventuelles redirections sont mises en place dans le shell en cours, et le code de retour est 0. S'il y a un problème de redirection, le code de retour est 1.

exit [*n*]

Termine le shell avec le code de retour *n*. Si *n* est omis, le code de retour sera celui de la dernière commande exécutée. Un éventuel gestionnaire installé sur **EXIT** sera exécuté avant la fin du shell.

export [**-nf**] [*nom*[=*mot*]] ...

export -p

Les *noms* indiqués sont marqués pour être exportés automatiquement dans l'environnement des commandes exécutées par la suite. Si l'option **-f** est fournie, les *noms* sont ceux de fonctions. Si aucun *nom* n'est mentionné, ou si l'option **-p** est fournie, une liste est affichée indiquant l'ensemble des noms exportables par ce shell. L'option **-n** supprime l'attribut d'exportation de chacunes des variables *nom*mées. **export** renvoie un code de retour nul, sauf si une option illégale est rencontrée, ou si l'un des *noms* n'est pas un nom de variable légal, ou si l'option **-f** est fournie avec un *nom* qui ne soit pas une fonction.

fc [**-e** *éditeur*] [**-nlr**] [*premier*] [*dernier*]

fc -s [*motif*=*nouveau*] [*commande*]

(Fix Command). Avec la première forme, un intervalle de commandes, entre la *première* et la *dernière*, est sélectionné dans la liste d'historique. La *première* et la *dernière* peuvent être indiquées sous forme de chaînes (la dernière commande commençant par la chaîne indiquée), ou sous forme numérique (un index dans la liste d'historique, ou un nombre négatif indiquant un décalage par rapport à la position actuelle dans cette liste). Si la *dernière* n'est pas indiquée, on considère qu'il s'agit de la commande en cours si l'on veut afficher la liste (ainsi **fc -l -10** affiche les 10 dernières commandes), et qu'il s'agit de la même que la *première* sinon. Si la *première* n'est pas indiquée, on considère qu'il s'agit de la commande précédente pour une édition, et de la commande -16 pour l'affichage de liste.

L'option **-n** supprime l'affichage des numéros dans la liste. L'option **-r** inverse l'ordre des commandes Avec l'option **-l**, les commandes sont affichées sur la sortie standard. Sinon, l'*éditeur* est invoqué avec un fichier contenant les commandes sélectionnées. Si aucun *éditeur* n'est précisé, on utilise la valeur de la variable **FCEDIT**, ou la valeur de **EDITOR** si **FCEDIT** n'existe pas. Si aucune variable n'existe, *vi* est invoqué. Lorsque l'édition se termine, les commandes sont affichées puis exécutées.

Dans la seconde forme, la *commande* est ré-exécutée après avoir remplacé chaque instance du *motif* par *nouveau*. Un alias utile pour ceci est « r="fc -s" », ainsi en tapant « r cc » on relance la dernière commande commençant par « cc » et en tapant « r » « r » on ré-exécute la commande précédente.

Lorsque la première forme est utilisée, la valeur renvoyée est 0, sauf si une option illégale est trouvée, ou si *première* ou *dernière* indiquent des lignes d'historique inexistantes. Avec l'option **-e**, la valeur de retour est celle de la dernière commande exécutée, ou un échec si une erreur s'est produite avec le fichier temporaire des commandes. Avec la seconde forme, le code de retour est celui de la commande ré-exécutée, à moins que *commande* ne soit pas une ligne valide de l'historique, auquel cas **fc** renvoie un échec.

fg [*job*]

Placer le *job* en avant-plan, et en faire le job en cours. Si aucun *job* n'est mentionné, le shell considèrera qu'il faut ramener le *job en cours* en avant-plan. La valeur de retour est celle de la commande placée en avant-plan, ou un échec si le contrôle des jobs n'est pas actif, ou encore si le *job* n'existe pas ou a été lancé sans contrôle des jobs.

getopts *chaîne_d_options nom* [*arguments*]

getopts est utilise par les scripts shell pour analyser les paramètres positionnels. La *chaîne_d_options* contient l'ensemble des lettres d'options à reconnaître. Si une lettre est suivie par un deux-points (:), l'option est censée comprendre un argument, qui peut en être séparé par une espace. À chaque invocation, **getopts** place l'option suivante dans la variable shell dont le *nom* est fourni, en la créant si elle n'existe pas. Il place également l'index de l'argument suivant à analyser dans la variable **OPTIND**. **OPTIND** est initialisé à 1 à chaque lancement d'un shell ou d'un script. Quand une option nécessite un argument, **getopts** place celui-ci dans la variable **OPTARG**. Le shell ne réinitialise pas **OPTIND** automatiquement, il faut le faire manuellement si des appels multiples à **getopts** au sein du même shell utilisent des ensembles différents de paramètres.

Lorsque la fin des options est atteinte, **getopts** se termine avec un code de retour supérieur à zéro. **OPTIND** est rempli avec le rang du premier argument non-option, et **nom** contient '?'.

getopts analyse en principe les paramètres positionnels, mais si d'autres *arguments* lui sont transmis, **getopts** analysera ces derniers.

getopts peut renvoyer des erreurs de deux manières. Si le premier caractère de la *chaîne_d_options* est un deux-points, les erreurs sont renvoyées de manière *silencieuse*. Lors d'un fonctionnement normal, des messages de diagnostique sont affichés si une option est illégale, ou s'il manque un argument d'option. Si la variable **OPTERR** vaut 0, aucun message d'erreur ne sera affiché, même si le premier caractère de la *chaîne_d_options* n'est pas un deux-points.

Si une option illégale est rencontrée, **getopts** met un point d'interrogation ? dans la variable *nom*, affiche un message d'erreur (s'il n'est pas en mode silencieux), et détruit la variable **OPTARG**. Si **getopts** fonctionne en mode silencieux, le caractère d'option est placé dans **OPTARG** et aucun message de diagnostique n'est affiché.

Lorsqu'un argument d'option est manquant, et si **getopts** n'est pas silencieux, un point d'interrogation (**?**) est placé dans la variable *nom*, **OPTARG** est détruit, et un message de diagnostique est affiché. Si **getopts** est silencieux, un deux-points (**:**) est placé dans la variable *nom* et **OPTARG** est rempli avec le caractère d'option trouvé.

getopts renvoie la valeur Vrai si une option, prévue ou imprévue, est trouvée. Il renvoie Faux si la fin des options est atteinte, ou si une erreur se produit.

hash [**-lr**] [**-p** *fichier*] [**-dt**] [*nom*]
Pour chaque *nom* fourni, le chemin d'accès complet de la commande est déterminé, en examinant les répertoires de **$PATH** et mémorisé. Si l'option **-r** est fournie, aucune recherche de chemin d'accès n'a lieu, et le *fichier* est utilisé comme chemin complet pour la commande. L'option **-r** force le shell à oublier tous les emplacements mémorisés.

L'option **-d** force le shell à oublier tous les emplacements mémorisés de chaque *nom*. Si l'option **-t** est fournie, le chemin complet de chaque *nom* correspondant est affiché. Si plusieurs arguments *nom* sont fournis avec**-t**, le fInom est affiché avant chaque chemin complet haché. L'option **-l** force la sortie à être affichée dans un format réutilisable en entrée. option causes output to be displayed in a format that may be reused as input. If no arguments are given, or if only **-l** is supplied, information about remembered commands is printed. Sans arguments ou si seulement **-l** est fournie, **hash** affiche des informations sur les commandes mémorisées. La valeur de retour est Vrai, sauf si *nom* n'est pas trouvé, ou si une option est illégale.

help [*motif*]
Affiche des informations d'aide au sujet des commandes internes. Si un *motif* est indiqué, **help** fournit de l'aide sur toutes les commandes correspondant à ce *motif*, sinon, une liste des commandes internes est affichée. Le code de retour est 0, sauf si aucune commande ne peut correspondre au *motif*.

history [*n*]
history -c
history -d *offset*
history -anrw [*fichier*]
history -p *arg* [*argument ...*]
history -s *arg* [*argument ...*]
Sans options, affiche la liste des commandes de l'historique, avec des numéros de ligne. Les lignes marquées avec un astérisque ***** ont été modifiées. Un argument *n* permet de ne lister que les *n* dernières lignes. Si la variable shell **HISTTIMEFORMAT** existe et n'est pas vide, elle est utilisée comme une chaîne de format pour **strftime**(3) afin d'afficher l'horodatage associé à chaque entrée de l'historique affichée. Il n'y aura pas d'espace entre l'horadatage ainsi formaté et la ligne de l'historique. Si un argument est fourni, il est utilisé comme nom de fichier historique, sinon on utilise la valeur de la variable **HISTFILE**. Les options ont les significations suivantes :

-a
Ajouter les « nouvelles » lignes d'historiques (celles créées depuis le début de la session actuelle de **bash**) dans le fichier.

-n
Lire dans le fichier, et ajouter à la liste d'historique, les lignes non encore lues. Il s'agit des lignes qui ont été ajoutées dans le fichier depuis le début de la session actuelle de **bash**.

-r
Lire le contenu du fichier, et l'utiliser comme historique courant.

-w
Écrire dans le fichier l'historique actuel, en écrasant le contenu precedent.

-c
Effacer le fichier d'historique en détruisant toutes ses entrées.

-p
Effectuer la substitution sur les arguments suivants, et afficher le résultat sur la sortie standard. Ne pas mémoriser les résultats dans la liste d'historique. Chaque *argument* doit être protégé pour éviter le développement normal.

-s
Enregistrer les *arguments* dans la liste d'historique comme entrée unique. La dernière commande de la liste d'historique est supprimée avant que les *arguments* soient ajoutés.

Si la variable **HISTTIMEFORMAT** existe, l'information d'horodatage associée à chaque entrée de l'historique est écrite dans le fichier d'historique. La valeur de retour est 0, sauf si une option illégale est fournie, ou si une erreur se produit durant la lecture ou l'écriture du fichier, si une valeur d'*offset* non valide est fournie comme argument à **-d**, ou si le développement de l'historique fourni à **-p** échoue.

jobs [**-lnprs**] [*job* ...]
jobs -x *commande* [*arguments* ...]

La première forme affiche la liste de jobs actifs. Les options ont les significations suivantes :

-l
Affiche les PIDs, en plus des informations normales.

-p
N'affiche que le PID du leader du groupe de processus du job.

-n
N'affiche que les jobs dont le statut a varié depuis le dernier affichage.

-r
N'affiche que les jobs en cours d'exécution (running). **-s** N'affiche que les jobs arrêtés (stopped).

Si une spécification de *job* est fournie, l'affichage ne comporte que les informations à propos de ce job. Le code de retour est 0, à moins qu'une option illégale soit transmise, ou que le *job* n'existe pas.

Si l'option **-x** est indiquée, **jobs** va remplacer toutes les spécifications de jobs rencontrées dans la *commande* ou dans ses *arguments* avec les Process Group ID correspondant, puis il exécutera la *commande* en lui transmettant les *arguments*, et en renvoyant son code de retour.

kill [**-s** *sigspec* | **-n** *num-signal* | **-***sigspec*] [*pid* | *job*] ...
kill -l [*sigspec* | *exit_status*]

Envoyer le signal indiqué par *sigspec* ou *num-signal* aux processus spécifiés par *pid* ou par *job*. On peut fournir *sigspec* soit sous forme de nom de signal (la casse étant sans importance), comme **SIGKILL** (le préfixe **SIG** est facultatif), soit sous forme numérique ; *num-signal* est un numéro de signal S'il n'y a pas d'indication de *sigspec*, le signal envoyé est **SIGTERM**. Un argument **-l** fournit la liste des noms de signaux. S'il y a des arguments avec l'option **-l**, les noms des signaux indiqués sont affichés, et le code de retour est nul. L'argument *exit_status* de l'option **-l** est un nombre correspondant soit à un numéro de signal soit au code de retour renvoyé par un processus tué par un signal. **kill** renvoie Vrai si au moins un signal a été envoyé, ou Faux si une erreur s'est produite, ou si une option illégale a été trouvée.

let *argument* [*argument* ...]

Chaque *argument* est une expression arithmétique à évaluer (voir plus haut le paragraphe **ÉVALUATION**ARITHMÉTIQUE**).** Si l'évaluation du dernier *argument* donne 0, **let** renvoie 1, sinon elle renvoie 0.

local [*nom*[=*valeur*] ...]

Pour chaque argument, on crée une variable locale ayant le *nom* indiqué, et à laquelle on affecte la *valeur* fournie. Quand **local** est utilisée dans une fonction, le *nom* de la variable n'est visible que dans la fonction, et ses descendantes. Sans opérandes, **local** affiche la liste des variables locales sur la sortie standard. Il ne faut pas utiliser **local** en dehors des fonctions. La valeur de retour est 0, sauf si **local** est invoqué en dehors d'une fonction, ou si un *nom* illégal est fourni.

logout

Quitter un shell de login.

popd [**-n**] [**+***n*] [**-***n*]

Enlève des éléments de la pile des répertoires. Sans arguments, seul le répertoire du sommet de la pile est éliminé, et un **cd** est effectué en direction du nouveau répertoire au sommet. Les arguments éventuels ont la signification suivante

+*n*
supprime le *n*-ième élément de la liste affichée par **dirs**, en comptant depuis la gauche à partir de 0. Par exemple : « popd +0 » enlève le premier répertoire, « popd +1 » le second.

-*n*
supprime le *n*-ième élément de la liste affichée par **dirs**, en comptant depuis la droite à partir de 0. Par exemple: « popd -0 » supprimer le dernier répertoire, et « popd -1 » l'avant-dernier.

-n
Empêche le changement normal de répertoire lors de la suppression d'un répertoire de la pile. On ne fait que manipuler la pile.

Si la commande **popd** réussit, un **dirs** est effectué, et le code de retour est nul. **popd** renvoie Faux si une option illégale est trouvée, si la pile de répertoires est vide, si on a indiqué un répertoire absent de la pile, ou si le changement de répertoire a échoué.

printf [**-v** *var*] *format* [*arguments*]

Écrit sur la sortie standard les *arguments* fournis, sous contrôle du *format*. Ce *format* est une chaîne contenant trois types de caractères : normaux, qui sont simplement copiés sur la sortie standard, les

caractères d'échappement qui sont convertis puis copiés sur la sortie standard, et les spécifications de format qui permettent d'afficher les *argument*. En plus des formats standards de **printf**(1), **%b** demande à **printf** de développer les séquences d'échappement se trouvant dans l'*argument* correspondant (excepté que **\c** termine la sortie, que les barres obliques dans **\(aq**, **\"** et **\?** ne sont pas supprimées, que les séquences d'échappement pour les nombres octaux commençant par **\0** peuvent contenir jusqu'à quatre chiffres), et **%q** demande l'affichage dans un format réutilisable en entrée d'un script shell.

L'option **-v** force la sortie à être affectée à la variable *var* plutôt que d'être affichée sur la sortie standard.

Le *format* est réutilisé jusqu'à consommer tous les *arguments*. Si le *format* demande plus d'*arguments* que fournis, les spécifications supplémentaires seront alimentées avec des valeurs nulles ou des chaînes vides.

pushd [**-n**] [*rép*]
pushd [**-n**] [+*n*] [-*n*]
Ajoute un répertoire au sommet de la pile des répertoires, ou effectue une rotation dans la pile, en ramenant le répertoire en cours au sommet. Sans argument, cette commande échange les deux répertoires au sommet, et renvoie 0, sauf si la pile est vide. Les arguments éventuels ont les significations suivantes :

+*n*
Effectue une rotation dans la pile, de manière à amener au sommet le *n*-ième répertoire de la liste affichée par **dirs** (en comptant à
partir de la gauche).

-*n*
Effectue une rotation dans la pile, de manière à amener au sommet le *n*-ième répertoire de la liste affichée par **dirs** (en comptant à partir de la droite).

-n
Empêche le changement normal de répertoire lors d'un ajout dans la pile. Seule la pile est manipulée.

rép
Ajoute le répertoire *rép* au sommet de la pile, et en fait le nouveau répertoire de travail.

Si la commande **pushd** réussit, elle effectue un **dirs**. Si on utilise la première forme, **pushd** renvoie 0 sauf si le déplacement vers *rep* échoue. Avec la seconde forme, **pushd** renvoie 0 sauf si la pile est vide, si on a réclamé un élément inexistant de la pile, ou si le changement de répertoire a échoué.

pwd [**-LP**]
Affiche le chemin d'accès complet au répertoire de travail actuel. Le chemin ne contiendra pas de liens symboliques si l'option **-P** est fournie, ou si l'option **-o physical** de la commande interne **set** est active. Si l'option **-L** est utilisée, les liens symboliques sont suivis. Le code de retour est nul, sauf si une erreur s'est produite durant la lecture du chemin d'accès au répertoire courant ou si une option invalide est transmise.

read [**-ers**] [**-u** *fd*] [**-t** *timeout*] [**-a** *nom_a*] [**-p** *invite*] [**-n** *nb_car*] [**-d** *delim*] [*nom* ...]
Une ligne est lue depuis l'entrée standard, ou à partir du descripteur de fichier *fd* fourni en argument à l'option **-u**, puis le premier mot de cette ligne est affecté au premier *nom*, le second mot au second *nom*, et ainsi de suite. Les mots restants sont affectés au dernier *nom*. S'il y a moins de mots lus dans le flux d'entrée que de variables, celles restantes sont remplies avec des valeurs vides. Les caractères contenus dans la variables **IFS** sont utilisés comme délimiteurs de mots. Le caractère barre oblique inverse (****) permet de supprimer toute signification spécial pour le caractère suivant, et autorise la continuation de lignes. Les options éventuelles ont les significations suivantes :

-a *nom_a*
Les mots sont affectés aux rangs successifs d'une variable tableau de nom *nom_a*, en commençant à 0. *nom_a* est détruit avant d'affecter de nouvelles valeurs. Les autres arguments *nom* sont ignorés.
.TP **-d** *delim* Plutôt que le saut de ligne, le premier caractère de *delim* est utilisé pour marquer la fin de la ligne de l'entrée.

-e
Si l'entrée standard provient d'un terminal, la bibliothèque **readline** (voir **READLINE** plus bas) est utilisée pour obtenir la ligne.

-n *nb_car*
read revient après avoir lu *nb_car* caractères plutôt que d'attendre la ligne complète de l'entrée.

-p *invite*
Affiche *invite* sur la sortie d'erreur, sans caractère nouvelle-ligne à la fin, afin d'essayer de lire une nouvelle entrée. L'invite est affichée seulement si l'entrée vient d'un terminal.

-r
La barre oblique n'agit plus comme un caractère d'échappement mais est considérée comme faisant parti de la ligne. En particulier, la paire barre oblique-nouvelle ligne n'est pas considérée comme une continuation de ligne.

-s
Mode silencieux. Si une entrée arrive à partir d'un terminal, les caractères ne sont pas affichés.

-t *timeout*
Force **read** à arriver au bout de son délai et revenir en échec si une ligne complète d'entrée n'a pas été lue au bout de *timeout* secondes. Cette option n'a aucun effet si *read* ne lit pas l'entrée à partir d'un terminal ou d'un tube.
-u *fd*
Lire l'entrée à partir du desripteur de fichier *fd*.

Si aucun *nom* n'est fourni, la ligne lue est affectée entièrement à la variable **REPLY**. Le code de retour est nul, sauf si une fin-de-fichier est rencontrée, si **read** arrive au bout de son délai ou si un descripteur de fichier non valide a été fourni à l'argument **-u**.

readonly [**-apf**] [*nom*[=*mot*] ...] Les *noms* indiqués reçoivent un attribut lecture-seule. Les valeurs affectées à ces *noms* ne pourront plus être modifiées. Si l'option **-f** est fournie, on considère les fonctions correspondant à ces *noms*. L'option **-a** restreint l'action aux variables tableaux. Si aucun *nom* n'est fourni, ou si l'option **-p** est utilisée, on affiche la liste des noms en lecture-seule. L'option **-p** demande l'affichage dans un format susceptible d'être réutilisé en entrée. Si le nom d'une variable est suivi par=*mot*, la variable est remplie avec *mot*. Le code de retour est nul, sauf si une option illégale a été trouvée, si l'un des *noms* n'est pas un nom légal de variables shell, ou si l'option **-f** est fournie avec un *nom* qui ne soit pas une fonction.

return [*n*]
La fonction shell en cours d'exécution se termine en renvoyant la valeur *n*. Si *n* est omis, la valeur renvoyée est celle de la dernière commande exécutée dans le corps de la fonction. Si cette commande est exécutée en dehors d'une fonction, mais durant l'exécution d'un script avec la commande **.** (**source**) ce script s'arrête, et la valeur renvoyée est *n* ou celle de la dernière commande exécutée. Si elle est utilisée en dehors d'une fonction ou d'un script exécuté par **.**, le code retour est Faux. Toute commande associée à la capture (Ndt : trap) de **RETURN** est exécutée avant que l'exécution reprenne après la fonction ou le script.

set [**--abefhkmnptuvxBCHP**] [**-o** *option*] [*arg* ...]
Sans options, le nom et la valeur de chacune des variables shell sont affichés dans un format susceptible d'être réutilisé en entrée pour positionner ou réinitialiser les variables actuelles. les variables en lecture seule ne peuvent pas être réinitialisées. Dans le *mode posix*, seules les variables shell sont affichées. La sortie est triée en fonction de la localisation en cours. Lorsque des options sont fournies, elles activent ou inhibent des attributs du shell. Tous les arguments restant après traitement des options sont considérés comme des valeurs affectées dans l'ordre aux paramètres : **$1**, **$2**, **...** **$***n*. Les options éventuelles ont les significations suivantes :

-a
Marquage automatique des variables et fonctions modifiées ou créées pour qu'elles soient exportées dans l'environnement des commandes exécutées ultérieurement.
-b
Affichage immédiat des statuts des jobs en arrière-plan qui se terminent, plutôt que d'attendre l'affichage du prochain symbole d'accueil. N'a d'utilité que si le contrôle des jobs est activés.
-e
Sortie immédiate lorsqu'une *commande simple* (voir **GRAMMAIRE DU SHELL** plus haut) se termine avec un code non nul. Le shell ne se termine pas si la commande qui échoue fait parti d'une liste de commande immédiatement suivie par un mot-clef **while** ou **until**, dans un test *if*, dans une liste **&&** ou ||, ou si le code de retour est inversé par **!**. Une capture sur **ERR**, si elle est configurée, est exécutée avant que le shell quitte.
-f
Désactivation du développement des noms de fichiers.
-h
Mémorisation de l'emplacement des commandes lors de leur exécution. Activé par défaut.
-k
Tous les arguments du type affectation sont placés dans l'environnement des commandes, pas uniquement ceux saisis avant le nom de la commande.
-m
Mode supervision. Activation du contrôle des jobs. Ce mode est actif par défaut pour les shells interactifs qui le supportent (voir le paragraphe **CONTRÔLE DES TÂCHES** plus haut). Les processus en arrière-plan sont exécutés dans des groupes de processus séparés, et une ligne indiquant leur statuts est affichée lorsqu'ils se terminent.
-n
Lecture de commandes sans exécution. Ceci peut être utilisé pour rechercher les erreurs de syntaxe dans un script. Cette option est ignorée par les shells interactifs.
-o *nom_d_option*
Le *nom_d_option* est l'un des suivants :

allexport
Identique à **-a**.
braceexpand
Comme **-B**.
emacs

Utiliser une interface « à la emacs » pour les lignes de commande. C'est le comportement par défaut pour un shell interactif, sauf s'il est lance avec l'option **--noediting**.

errtrace
Identique à **-E**.

functrace
Identique à **-T**.

errexit
Identique à **-e**.

hashall
Comme **-h**.

histexpand
Identique à **-H**.

history
activer l'historique des commandes, comme décrit plus haut au paragraphe **HISTORY**. Cette option est validée par défaut pour un shell interactif.

ignoreeof
Identique à l'exécution d'une commande shell « IGNOREEOF=10 »'. (voir le paragraphe **Variables du Shell** plus haut)

keyword
Comme **-k**.

monitor
Identique à **-m**.

noclobber
Identique à **-C**.

noexec
Identique à **-n**.

noglob
Identique à **-f**. **nolog** Actuellement ignoré.

notify
Identique à **-b**.

nounset
Identique à **-u**.

onecmd
Identique à **-t**.

physical
Identique à **-P**.

pipefail
Si elle est active, la valeur de retour d'un tube est la valeur de la dernière commande (la plus à droite) à quitter avec un code de retour non nul, ou zéro si toutes les commandes s'achèvent avec succès. cette option n'est pas active par défaut.

posix
Modifier le comportement de **bash** lorsque les opérations par défaut s'éloignent de POSIX afin d'être conforme au standard (*mode posix*).

privileged
Identique à **-p**.

verbose
Identique à **-v**.

vi
Utiliser une interface d'édition des lignes de commande « à la vi ».

xtrace
Identique à **-x**.

Si **-o** est invoqué sans *nom_d_option*, les valeurs actuelles des options sont affichées. Si **+o** est fourni sans *nom_d_option*, la série de commande **set** permettant de recréer la configuration courante est affichée sur la sortie standard.

-p
Basculer en mode *privilégié*. Dans ce mode, le fichier **$ENV** et **$BASH_ENV** ne sont pas traités, les fonctions shells ne sont pas héritées de l'environnement, et la variable **SHELLOPTS**, si elle est dans l'environnement, est ignorée. Si le shell démarre avec un UID (resp. GID) effectif différent de son UID (GID) réel, et si l'option **-p** n'est pas fournie, ce comportement est appliqué, et l'UID (GID) effectif reprend la valeur de l'UID (GID) réel. Si l'option **-p** est fournie au démarrage l'UID (GID) effectif n'est pas modifié. Désactiver cette option force la réinitialisation des UID et GID effectifs à la valeur des UID et GID réels.

-t
Sortie automatique après lecture et exécution d'une commande.

-u

Considérer les variables inexistantes comme des erreurs durant l'expansion des paramètres. Si on tente de développer une variable inexistante, le shell affiche un message d'erreur, et, s'il n'est pas interactif, se termine avec un code de retour non-nul.

-v

Affichage des lignes de commandes au fur et à mesure de leur lecture.

-x

Après l'expansion de chaque *commande simple*, commande **for**, commande **case**, commande **select** ou commande arithmétique **for**, **bash** affiche la valeur de **PS4**, suivie par la commande et ses arguments développés ou la liste de mots associés.

-B

Le shell effectue l'expansion des accolades (voir plus haut) Cette opriont est active par défaut.

-C

Avec cette option, **bash** n'écrasera pas un fichier existant avec les opérateurs **>**, **>&**, et **<>**. Ce comportement peut être surchargé lors de la création d'un fichier de sortie en utilisant l'opérateur de redirection **>|** à la place de **>**.

-E

Si elle est présente, tout déroutement (Ndt : trap) de **ERR** est hérité par les fonctions shell, les substitutions de commandes et les commandes exécutées dans un sous-shell. Le déroutement de **ERR** n'est normalement pas hérité dans de tels cas.

-H

Autoriser le style **!** de substitution historique. C'est le comportement par défaut lorsque le shell est interactif.

-P

Ne pas suivre les liens symboliques lors d'une commande comme **cd** qui modifie le répertoire de travail. Utiliser à la place le répertoire physique. Par défaut **bash** suit la chaîne logique des répertoires lors des commandes qui modifient le répertoire en cours.

-T

Si elle est présente, tout déroutement de **DEBUG** et **RETURN** est hérité par les fonctions shell, les substitutions de commandes et les commandes exécutées dans un sous-shell. Les déroutements de **DEBUG** et **RETURN** ne sont normalement pas hérité dans de tels cas.

--

Si aucun argument ne suit ce drapeau, alors les paramètres positionnels sont détruits. Sinon, les paramètres positionnels sont remplis avec les *arguments*, même si certains d'entre eux commencent par un **-**.

-

Fin des options. Tous les *arguments* restants sont affectés aux paramètres positionnels. Les attributs **-x** et **-v** sont désactivés. S'il n'y a pas d'*arguments*, les paramètres positionnels ne sont pas modifiés.

Par défaut les attributs sont désactivés, sauf indication contraire. En utilisant + à la place de - on désactive explicitement un drapeau. Les attributs peuvent également être fournis en argument lors de l'invocation du shell. Le jeu de drapeaux actuellement actifs est disponible dans **$-**. La valeur de retour est vraie sauf si une option invalide est rencontrée.

shift [*n*]

Les paramètres positionnels à partir de *n*+1 ... sont renommés en **$1 ....** Les paramètres représentés par les nombres **$#** jusqu'à **$#**-*n*+1 sont supprimés. *n* doit être un entier non-négatif inférieur ou égal à **$#**. Si *n* vaut 0, aucun paramètres n'est modifié. Si *n* est omis, on suppose qu'il vaut 1. Si *n* est supérieur à **$#**, les paramètres positionnels ne sont pas modifiés. Le code de retour est supérieur à 0 si *n* est supérieur à **$#** ou inférieur à 0. Sinon le code de retour est nul.

shopt [**-pqsu**] [**-o**] [*nom_opt* ...]

Bascule la valeur des variables contrôlant le comportement optionnel du shell. Sans option, ou avec l'option **-p**, une liste de toutes les options configurables est affichée, avec l'indication de l'état de chacune d'entre elles. L'option **-p** réclame un affichage susceptible d'être réutilisé en entrée. Les autres options ont les significations suivantes :

-s

Activer chaque *nom_opt* indiqué.

-u

Désactiver chaque *nom_opt* indiqué.

-q

Supprimer la sortie normale (mode silencieux). Le code de retour indique si l'option *nom_opt* est active ou non. Si plusieurs *nom_opt* sont fournis en argument de l'option **-q**, le code de retour est nul si tous les *nom_opt* sont actifs, et non-nul sinon.

-o

Restreindre les valeurs des *nom_opts* à celles définies pour l'option **-o** de la commande **set** interne.

Si l'option **-s** ou **-u** est utilisé sans argument *nom_opt*, l'affichage est limité aux options qui sont actives ou inactives, respectivement. sauf indication contraire, les options **shopt** sont désactivés par

défaut.

Le code de retour lors d'un affichage est zéro si tous les *nom_opt* sont actifs, non-nul sinon. Lors d'une activation ou inhibition, le code de retour est nul sauf si *nom_opt* n'est pas une option valide du shell.

La liste des options **shopt** est :

cdable_vars
Si cette option est active, un argument de la commande interne **cd** qui n'est pas un répertoire est supposé être un nom de variable dont la valeur est le répertoire visé.

cdspell
Les erreurs minimes de frappe dans un composant du répertoire en argument de la commande **cd** seront corrigées. Les erreurs corrigées sont les inversions d'un caractère, un caractère manquant et un caractère en trop. Si une correction est possible, le nom de répertoire corrigé est affiché, et la commande est exécuté. Cette option n'est utilisé que par les shells interactifs.

checkhash
bash vérifiera si une commande trouvée dans la table de hachage existe avant d'essayer de l'exécuter. Si une commande hachée n'existe plus, la recherche normale de chemin est assurée.

checkwinsize
bash vérifiera la taille de la fenêtre après chaque commande et, au besoin, mettra à jour les valeurs des variables **LINES** et **COLUMNS**.

cmdhist
Lorsque cette option est active, **bash** essaiera de sauvegarder les commandes s'étendant sur plusieurs lignes en une seule entrée d'historique. Ceci facilite l'édition ultérieure de commandes multilignes.

dotglob
Si cette option est active, **bash** inclut les noms de fichiers commençant par un « . » dans les résultats des développements de noms de fichiers.

execfail
Un shell non-interactif ne se terminera pas s'il ne peut exécuter un fichier fourni en argument de la commande interne **exec**. Un shell interactif ne se termine pas si **exec** échoue.

expand_aliases
Les alias sont développés comme décrit plus haut, au paragraphe **ALIASES**. Cette option est activée par défaut pour les shells interactifs.

extdebug
Si elle est présente, le comportement voulu par les débogueurs est actif :

1. L'option **-F** de la commande interne **declare** affiche le nom du fichier source et le numéro de ligne correspondant à chaque nom de fonction fourni comme argument.
2. Si la commande lancée par le détournement de **DEBUG** renvoie une valeur non nulle, la commande suivante est sautée et n'est pas exécutée.
3. Si la commande lancée par le détournement de **DEBUG** renvoie la valeur 2 et si le shell s'exécute dans une sous-routine (une fonction shell ou un script shell lancé par les commandes internes **.** ou **source**), un appel à **return** est simulé.
4. **BASH_ARGC** et **BASH_ARGV** sont mises à jour comme expliqué dans leur description plus haut.
5. Le traçage de fonction est actif : subtitution de commandes, fonctions shell est sous-shell invoqués avec **(** *command* **)** héritent des déroutement de **DEBUG** et **RETURN**.
6. Le traçage d'ereur est actif : subtitution de commandes, fonctions shell est sous-shell invoqués avec **(** *command* **)** héritent du déroutement de **ERROR**.

extglob
Les fonctionnalités étendues de mise en correspondance décrite au paragraphe **Développement des noms de fichiers** sont activées.

extquote
Si elle est présente, les citations **$**aq*chaîne*aq et **$**"*chaîne*" sont effectuées dans les développements de **${***paramètres***}** entre guillemets. Cette option est active par défaut.

failglob

Si elle est présente, les motifs qui échouent à faire correspondre les noms de fichiers pendant le développement des chemins font qu'ils provoqueront une erreur de développement.

force_fignore
Si elle est présente, les suffixes spécifiés par la variable shell **FIGNORE** forceront les mots à être ignorés lors de la complétion de mots, même si les mots ignorés sont des complétions possibles. Voir le paragraphe **VARIABLES SHELL** plus haut pour une description de **FIGNORE**. Cette option est active par défaut.

gnu_errfmt
Si elle est présente, les messages d'erreur du shell seront écrits dans le format standard GNU des messages d'erreurs.

histappend
La liste d'historique est ajoutée au fichier dont le nom est contenu dans la variable **HISTFILE** lorsque le shell se termine, plutôt que d'écraser ce fichier.

histreedit
Si la bibliothèque **readline** est en service, un utilisateur peut ré-éditer une substitution d'historique qui a échoué.

histverify
Si la bibliothèque **readline** est en service, le résultat de la substitution d'historique n'est pas transmise immédiatement au shell. À la place, la ligne résultant est chargée dans le tampon d'édition de **readline** permettant des modifications ultérieures.

hostcomplete
Si la bibliothèque **readline** est en service, **bash** essayera d'assurer la complétion des noms d'hôtes lorsqu'un mot contient un @ (voir **Completion** au paragraphe **READLINE** plus haut). Cette option est active par défaut.

huponexit
bash enverra un signal **SIGHUP** à tous les jobs lorsqu'un shell de login interactif se termine.

interactive_comments
Un mot commençant par un # autorisera tous les autres caractères de la ligne à être ignorés dans un shell interactif (voir le paragraphe **COMMENTAIRES** plus haut). Cette option est active par défaut.

lithist
Si cette option, et l'option **cmdhist** sont actives, les commandes multilignes sont sauvées dans l'historique avec des sauts-de-ligne imbriqués plutôt que des points-virgules.

login_shell
Le shell positionne cette option s'il est démarré en tant que shell de connexion (voir le paragraphe **INVOCATION** plus haut). Cette valeur ne peut être modifiée.

mailwarn
Si cette option est active et si un fichier que **bash** surveille pour les mails a été accédé depuis la dernière vérification, le message « The mail in *fichier* has been read » est affiché.

nocaseglob
Si cette option est active, **bash** traite les noms de fichiers sans différences minuscules/majuscules lors du développement des noms de fichiers (voir plus haut).

nocasematch
Si elle est présente, **bash** fait correspondre les motifs de manière insensible à la case lors de la mise en correspondance pendant l'exécution des commandes conditionnelles **case** ou **[[**.

nullglob
Si cette option est active, **bash** autorise les motifs ne correspondant à aucun fichier (voir **Développement des noms de fichiers** plus haut) à se développer en une chaîne nulle plutôt qu'en une valeur littérale.

promptvars
Si cette option est active, la chaîne de l'invite de commande est soumise au développement de paramètres, substitution de commandes, développement arithmétique et suppression des apostrophes après avoir été développée comme décrit dans le paragraphe **PROMPTING** plus haut. Cette option est active par défaut.

restricted_shell
Le shell active cette option s'il est démarré en mode restreint (voir **SHELL RESTREINT** plus bas). Cette valeur ne peut pas être changée. Elle n'est pas réinitialisée lorsque les fichiers de démarrage sont exécutés, ce qui permet à ces fichiers de découvrir si un shell est resteint ou non.

shift_verbose
La commande interne **shift** affiche un message d'erreur lorsque le nombre de décalage dépasse le nombre de paramètres positionnels.

sourcepath
La commande interne **source** (**.**) utilise la valeur de la variable **PATH** pour trouver le répertoire contenant le fichier fourni en argument. Cette option est active par défaut.

suspend [**-f**]
Suspend l'exécution du shell jusqu'à la réception d'un signal **SIGCONT**. L'option **-f** élimine le message d'erreur s'il s'agit d'un shell de login, et force sa suspension. Le code de retour est nul sauf si l'on utilise un

shell de login et si l'option **-f** n'est pas fournie, ou si le contrôle des jobs n'est pas disponible.

test *expr*
[*expr* **]**

Renvoie la valeur 0 (vrai) ou 1 (faux) en fonction de l'évaluation de l'expression conditionnelle *expr*. Chaque opérateur et opérande doit être représenté par un argument distinct. Les expressions sont composées des unités élémentaires décrites plus haut dans **EXPRESSIONS**CONDITIONNELLES **test** n'accepte aucune option, n'accepte pas non plus et ignore l'argument **--** signifiant la fin des options.

Les expressions peuvent être combinées avec les opérateurs suivant, par ordre de précédence décroissante :

! *expr*
Vrai si *expr* est fausse

(*expr*)
Renvoie la valeur de *expr*. Peut servir à surcharger la précédence normale des opérateurs.

expr1 -**a** *expr2*
Vrai si *expr1* et *expr2* sont toutes deux vraies.

expr1 -**o** *expr2*
Vrai si *expr1* ou *expr2* est vraie.

test et **[** évaluent les expresssions conditionnelles en fonction d'un jeu de règles dépendant du nombre d'arguments.

0 argument
L'expression est fausse

1 argument
L'expression est vraie si et seulement si l'argument est non nul.

2 arguments
Si le premier argument est **!**, l'expression est vraie si et seulement si le second argument est nul. Si le premier argument est l'un des opérateurs conditionnel unaires décrits plus haut au paragraphe **EXPRESSIONS**CONDITIONNELLES l'expression est vraie si le test unaire est vrai. Si le premier argument n'est pas un opérateur conditionnel unaire; l'expression est fausse.

3 arguments
Si le second argument est l'un des opérateur conditionnels binaires décrits plus haut au paragraphe **EXPRESSIONS**CONDITIONNELLES**,** le résultat de l'expression est le résultat du test binaire utilisant le premier et le troisième argument en tant qu'opérandes. Si le premier argument est **!**, la valeur est la négation du test binaire utilisant les deuxième et troisième arguments. Si le premier argument est exactement **(** et le troisième argument est exactement **)**, le résultat est le test unaire du second argument. Sinon l'expression est fausse. Les opérateurs **-a** et **-o** sont considérés comme des opérateurs binaires dans ce cas.

4 arguments
Si le premier argument est **!**, le résultat est la négation de l'expression à trois arguments composée des arguments restants. Sinon l'expression est analysée et évaluée selon les règles de précédence décrites ci-dessus.

5 argument ou plus.
L'expression est analysée et évaluée selon les règles de précédence décrites ci-dessus.

times
Affiche les durées cumulées utilisateur et système pour le shell et les processus lancés par ce shell. Le code de retour est nul.

trap [**-lp**] [[*cmd*] *sigspec* ...]
trap [**-lp**] [[*arg*] *sigspec* ...] La commande *cmd* sera lue et exécutée lorsque le shell recevra l'un des signaux *sigspec*. Si *cmd* est absent (et il n'y a qu'un seul*sigspec*) ou si elle vaut **-**, chaque signal indiqué reprend son comportement d'origine (qu'il avait au démarrage du shell). Si *cmd* est une chaîne nulle, les signaux *sigspec* seront ignorés par le shell et par les commandes qu'il invoque. Si *cmd* est **-p** alors le comportement pour chacun des *sigspec* est affiché. Si aucun argument n'est fourni, ou si l'option **-p** est seule, **trap** affiche les commandes associées avec chaque signal. L'option **-l** force le shell à afficher la liste des noms de signaux et leurs numéros correspondants. Les signaux *sigspec* peuvent être spécifiés en indiquant des noms de signaux définis dans <*signal.h*>, ou les numéros des signaux. Les noms de signaux sont insensible à la casse et le prefix SIG est optionnel. Si *sigspec* vaut **EXIT** (0) la commande *cmd* sera exécutée au moment de quitter le shell. Si *sigspec* est **DEBUG**, la commande *cmd* sera exécutée avant chaque *commande simple*, commande *for*, commande *case*, commande *select*, chaque commande arithmetique *for* et avant la première commande qui s'exécute dans une fonction shell (voir **GRAMMAIRE DU SHELL** plus haut). Veuillez vous reporter à la description de l'option **extdebug** de la commande interne **shopt** pour les détails de l'effet du déroutement **DEBUG**. Si un *sigspec* est **ERR**, la commande *arg* est exécutée chaque fois qu'une commande simple a un code de sortie non nul, sujet aux conditions suivantes. Le déroutement de **ERR** n'est pas effectué si la commande qui a échoué fait partie d'une liste de commandes suivant immédiatement un mot-clef **while** ou **until**, d'un test dans une instruction *if* d'un **&&** ou d'une liste || ou si la valeur de retour de la commande est inversée par **!**. Ce sont les mêmes conditions qui obéissent à l'option **errexit**. Si un *sigspec* est **RETURN**, la commande *arg* est exécutée chaque fois qu'une fonction shell ou un

script lancé avec les commandes internes **.** ou **source** finit son exécution. Les signaux ignorés lors de l'invocation du shell ne peuvent pas être capturés ni reprendre leurs comportements par défaut. Lors de la création d'un processus fils, les signaux capturés qui ne sont par ignorés reprennent leur comportement par défaut. Le code de retour est faux si un nom ou un numéro de signal est invalide, sinon **trap** renvoie une valeur vraie.

type [**-aftpP**] *nom* [*nom* ...]
Sans option, indique comment chaque *nom* doit être interprété si on l'utilise en nom de commande. Si l'attribut **-t** est utilisé, **type** affiche une des phrases suivantes *alias*, *keyword*, *function*, *builtin*, ou *file* selon que le *nom* est un alias, un mot-clé réservé au shell, une fonction, une commande interne, ou un fichier sur le disque. Si le *nom* n'est pas trouvé, rien n'est affiché et le code de retour est faux. Si l'argument **-p** est utilisé, **type** renvoie le nom du fichier qui sera exécuté si l'on tape le *nom* en guise de commande, ou rien si « type -t name » ne renvoyait pas *file*. L'option **-P** force une recherche dans **PATH** pour chaque *nom*, même si « type -t name » ne renvoyait pas *file*. Si une commande est disponible dans la table de hachage, **-p** et **-P** affichent la valeur de cette table, qui n'est pas nécessairement le fichier apparaissant en premier dans le **PATH**. Si l'option **-a** est invoquée, **type** affiche tous les emplacements contenant un exécutable du *nom* indiqué. Ceci inclut les alias et les fonctions, sauf si l'option **-p** est également présente. La table de hachage des commandes n'est pas consultée avec l'option **-a**. L'option **-f** supprime la fonction shell de consultation de table, comme avec la commande interne **command**. **type** renvoie vrai si l'un des arguments est trouvé, et faux si aucun n'a été trouvé.

ulimit [**-SHacdfilmnpqstuvx** [*limite*]]
fournit, sur les systèmes qui le permettent, un mécanisme de contrôle des ressources disponibles pour le shell et pour les processus qu'il lance. Les options **H** et **S** précisent si la limite doit être dure (Hard), ou souple (Soft). Une limite dure ne peut pas être augmentée une fois qu'elle a été définie. Une limite souple peut être augmentée jusqu'à la hauteur de la limite dure correspondante. Par défaut, les limites fixées sont souples. La valeur de la *limite* peut être un nombre (utilisant les unités particulières de la ressources), ou l'une des valeurs spéciales **hard**, **soft** ou **unlimited**, qui signifient, respectivement, la limite dure courante, la limite douce courante et pas de limite. Si la *limite* est omise, on affiche la valeur de la limite souple pour la ressource indiquée, sauf si l'option **H** est fournie. Quand plusieurs ressources sont indiquées, leurs noms, et leurs unités respectives sont affichés avant les valeurs. Les options sont interprétées ainsi :

-a
affichage de toutes les limites actuelles
-c
la taille maximale pour la création d'un fichier core
-d
la taille maximale du segment de données d'un processus
-e
la priorité d'ordonnancement maximum (« nice »)
-f
la taille maximale d'un fichier écrit par le shell et ses fils
-i
le nombre maximal de signaux en attente
-l
la taille maximale que l'on peut verrouiller en mémoire
-m
la taille maximale de la partie résidente d'un processus
-n
le nombre maximal de descripteurs de fichiers ouverts (la plupart des systèmes ne permettent pas de modification)
-p
La taille d'un tuyau (pipe) en blocs de 512 octets (parfois non modifiable)
-q
le nombre maximal d'octets dans les files de message POSIX
-r
la priorité d'ordonnancement temps réel maximum
-s
la taille maximale de la pile
-t
la durée maximale, en seconde, de temps CPU accordé à un processus.
-u
le nombre maximal de processus autorisés pour un seul utilisateur
-v
la quantité de mémoire virtuelle disponible pour le shell
-x
le nombre maximum de verrouillages fichiers

Si une *limite* est fournie, la ressource correspondante recevra cette limite (l'option **-a** ne permet que l'affichage). Si aucune option n'est indiquée, la ressource **-f** est prise par défaut. Les valeurs s'expriment en kilo-octets, sauf pour **-t**, ou elles s'expriment en secondes, **-p**, qui utilise des unités de

blocs de 512 octets, ainsi que **-n** et **-u**, qui n'ont pas d'unités. La valeur renvoyée est 0 sauf si une option ou un argument non valide sont fournis, ou si une erreur se produit durant la modification de la limite.

umask [**-p**] [**-S**] [*mode*]
Le masque de création de fichier de l'utilisateur est fixé au *mode* indiqué. Si *mode* commence par un chiffre, il est interprété comme un nombre octal, sinon il est considéré comme un masque symbolique, semblable à ceux acceptes par *chmod*(1). Si *mode* est omis, ou si l'option **-S** est fournie, la valeur courante du masque est affichée. L'option **-S** affiche le masque sous forme symbolique, l'affichage par défaut étant en octal. Si l'option **-p** est fournie, et si le *mode* est omis, la sortie a lieu dans un format réutilisable en entrée. Le code de retour est nul si le mode a pu être changé correctement, ou si on a omis l'argument *mode*, et faux sinon.

unalias [**-a**] [*nom* ...]
Supprime le *nom* de la liste des alias définis. Si l'option **-a** est demandée, toutes les définitions d'alias sont supprimées. La valeur renvoyée est vraie sauf si un *nom* mentionné n'est pas un alias défini.

unset [**-fv**] [*nom* ...]
Pour chaque *nom* indiqué, supprimer la variable ou la fonction correspondante. Si aucune option n'est fournie, ou si l'option **-v** est demandée, chaque *nom* se réfère à une variable shell. Les variables en lecture-seule ne peuvent pas être détruites. Si l'option **-f** est indiquée, chaque *nom* se réfère à une fonction shell, et la définition de la fonction est supprimée. Chaque variable ou fonction est supprimé de l'environnement transmis aux commandes ultérieures. Si l'une des variables **RANDOM**, **SECONDS**, **LINENO**, **HISTCMD**, ou **DIRSTACK** est supprimée, elle perd ses propriétés spécifiques, même s'il elle est recréée ultérieurement. Le code de retour est vrai, à moins qu'un des *noms* ne soit en lecture seule.

wait [*n* ...]
Attend que chaque processus indiqué se termine, et renvoie son code de retour. *n* peut indiquer un PID, ou un job. S'il s'agit d'un indicateur de job, on attendra la fin de tous les processus du tube de ce job. Si *n* est omis, on attend la fin de tous les processus fils actuellement actifs, et le code de retour sera zéro. Si *n* se rapporte à un processus ou un job inexistant, le code de retour sera 127. Dans tous les autres cas, le code de retour sera celui du dernier processus ou job attendu.

SHELL RESTREINT

Si **bash** démarre sous le nom **rbash**, ou si l'option **-r** est fournie lors de son invocation, le shell devient restreint. Un shell restreint permet de configurer un environnement plus contrôlé qu'un shell standard. Il se comporte de même manière que **bash** à la différence des actions suivantes qui sont désactivées ou non réalisées :

* changer de répertoire avec **cd** ;
* modifier ou détruire les valeurs de **SHELL**, **PATH**, **ENV**, ou **BASH_ENV** ;
* indiquer des noms de commandes contenant un **/** ;
* indiquer un nom de fichier contenant un **/** comme argument de la commande **.** interne ;
* importer une définition de fonction dans l'environnement au démarrage ;
* analyser les valeurs de **SHELLOPTS** au démarrage ;
* rediriger la sortie en utilisant un opérateur >, >|, <>, >&, &>, et >> " ;"
* utiliser la commande interne **exec** pour remplacer le shell par une autre commande " ;"
* ajouter ou supprimer des commandes internes avec les options **-f** et **-d** de la commande interne **enable** ;
* utiliser la commande interne **enable** pour activer les commandes internes du shell désactivées ;
* indiquer l'option **-p** de la commande interne **command** interne " ;"
* supprimer le mode restreint avec **set +r** ou **set +o restricted**.

Ces restrictions sont mises en place après la lecture des fichiers de démarrage.

Lorsqu'une commande trouvée dans un script shell est exécutée Lorsqu'une commande trouvée dans un script shell est exécutée (voir le paragraphe **EXÉCUTION DES COMMANDES** plus haut), **rbash** supprime les restrictions dans le shell servant à exécuter le script.

sh(1),
ksh(1), **csh**P(1)

emacs(1),
vi(1)
readline(3)

FICHIERS

/bin/bash2
L'exécutable **bash**.
/etc/profile
Le fichier d'initialisation globale, exécutée par les shells de login.
~/.bash_profile
Le fichier d'initialisation personnalisée, exécutée par les shells de login.
~/.bashrc
Le fichier de démarrage personnel, pour les shells interactifs.
~/.bash_logout
Le fichier de sortie des shells de login lors de la déconnexion.
~/.inputrc
Le fichier d'initialisation personnalisée de *readline*.

AUTEURS

Brian Fox, Free Software Foundation (auteur principal)
bfox@ai.MIT.Edu

Chet Ramey, Case Western Reserve University
chet@po.cwru.edu

BOGUES

Si vous trouvez un bogue dans **bash,** vous êtes encouragés à nous en faire part. Tout d'abord vous devez vous assurer qu'il s'agit réellement d'un dysfonctionnement, et qu'il apparaît bien dans la dernière version de **bash** disponible. La dernière version est toujours disponible à partir de *ftp://ftp.gnu.org/pub/bash/*.

Une fois que vous êtes sûrs qu'il s'agit bien d'un bogue, utilisez la commande *bashbug* pour nous envoyer un rapport. Si vous avez trouvé une correction, n'hésitez pas à l'envoyer en même temps. Les simples suggestions, ou les remarques d'ordre « philosophique » peuvent être transmises à bug-bash@*prep.ai.MIT.Edu* ou postées dans le groupe Usenet **gnu.bash.bug**.

TOUS les rapports de bogue doivent contenir :

Le numéro de version de **bash**
Le matériel et le système d'exploitation utilisés
La version du compilateur utilisé pour compiler **bash**.
Une description du comportement du bogue. [Ndt : en anglais évidemment]
Un petit script, ou une manipulation qui met en évidence le bogue.

bashbug insère automatiquement les trois premiers éléments de cette liste dans le formulaire qu'il fournit pour remplir le rapport de bogue.

Les commentaires et rapports d'erreurs concernant la version originale de cette page de manuel doivent être transmis à *chet@po.cwru.edu*. Ceux concernant la version française peuvent être envoyés à *manpagesfr*AT*free*DOT*fr*.

BOGUES

Cet interpréteur est volumineux et lent.

Il y a quelques différences subtiles de comportement entre **bash** et les versions traditionnelles de **sh**, principalement à cause des spécifications **POSIX**.

L'utilisation des alias peut avoir des conséquences inattendues.

Les commandes et les fonctions internes ne peuvent pas être arrêtées/relancées.

Les séquences de commandes de la forme « a ; b ; c » ne sont pas gérées proprement lors d'une interruption de processus. En cas de suspension d'une telle séquence, l'interpréteur exécute immédiatement la commande suivante de la séquence. Il suffit de placer cette séquence entre parenthèses pour la forcer à créer un sous-shell, qui, lui, sera interruptible en bloc.

Les commandes à l'intérieur de **$(...)** ne sont pas analysées avant la réalisation de la complétion. Cela peut retarder l'affichage de messages d'erreurs pendant un certain temps après la frappe de la commande. Par exemple, des parenthèses qui ne concordent pas, même à l'intérieur de commentaires, provoqueront des messages d'erreur lorsque la construction sera lue.

Les variables de type tableau ne peuvent pas (encore) être exportées.

VOIR AUSSI

Bash Features, Brian Fox et Chet Ramey
The GNU Readline Library, Brian Fox et Chet Ramey
The GNU History Library, Brian Fox et Chet Ramey
Portable Operating System Interface (POSIX) Part 2: Shell and Utilities, IEEE

TRADUCTION

Ce document est une traduction réalisée par Christophe Blaess <http://www.blaess.fr/christophe/> et Thierry Vignaud le 2 octobre 1997, mise à jour par Alain Portal <aportal AT univ-montp2 DOT fr> le 15 mai 2006 et révisée le 25 avril 2008.

L'équipe de traduction a fait le maximum pour réaliser une adaptation française de qualité. La version anglaise la plus à jour de ce document est toujours consultable via la commande : « **LANG=C man 1 bash** ». N'hésitez pas à signaler à l'auteur ou au traducteur, selon le cas, toute erreur dans cette page de manuel.

Index

Dernière mise à jour : 26 mai 2008

LES FONCTIONS

Syntaxe :

```
1        # !/bin/bash
2        function saluer
3    {
4        echo ''Hello !''
5     }
6
7     saluer
```

```
recule@reculeVirtualBox:-$ ./formation.sh
Hello !
```

Ou autre syntaxe :

```
1    # !/bin/bash
2    saluer ()
3    {
4        echo ''Hello !''
5     }
6
7     saluer
```

```
recule@reculeVirtualBox:-$ ./formation.sh
Hello !
```

Exemple inventé :

```
1    # !/bin/bash
2    saluer ()
3
4    {
4        echo ''Hello !''
5    }
6
7    nommer ()
8
9   {
10       echo ''damien''
11   }
12
13
14   famille ()
```

```
15
16    {
17        echo ‘’recule’’
18    }
19
20    aurevoir ()
21
22    {
23        echo ‘’salut !’’
24    }
25
26    saluer ; nommer ; famille ; recule
```

```
recule@reculeVirtualBox:~$ ./formation.sh
salut
damien
recule
salut !
```

=> enchainement de 4 fonctions, qu’on désigne dans un premier temps et qu’il suffit de nommer à la fin du scripte.

LES TABLEAUX

Nous pouvons créer deux sortes de tableau avec **bash** :

Tableau par indices :

```
1       # !/bin/bash
2
3       [0] [1] etc.. ←= utilisation d'indices [0]=valeur 1 , [1] valeur 2 etc …
4
```

Tableau associatif :

```
1       # !/bin/bash
2
3       [clé] + [valeurs]
4
```

Tableau par indices : en stipulant **index_array**

```
1       # !/bin/bash
2
3       declare -a index_array=(142 65 2 98)
4       declare -A assoc_array=(["prenom"]="damien" ["nom"]="recule")
5
6       for number in ${index_array[@]}
7       do
8               echo $number
9       done
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
142
65
2
98
```

Tableau par associatif : en stipulant **assoc_array**

```
1       # !/bin/bash
2
3       declare -a index_array=(142 65 2 98)
4       declare -A assoc_array=(["prenom"]="damien" ["nom"]="RECULE")
5
6       for number in ${assoc_array[@]}
7       do
8               echo $number
9       done
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
RECULE
damien
```

Tableau par indices : en stipulant qu'une valeur apres **index_array**

```
1       # !/bin/bash
2
3       declare -a index_array=(142 65 2 98)
4
5       echo ${index_array[0]} ←===avec ici [0] / valeur 1, il exécutera que la première valeur (142)
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
142
```

Tableau par associatif : en stipulant qu'une valeur apres **assoc_array**

```
1       # !/bin/bash
2
3       declare -A assoc_array=(["prenom"]="damien" ["nom"]="RECULE")
4
5       for number in ${assoc_array[prenom]}
6
7       echo $number
8
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
damien
```

Tableau par associatif : Si on ajoute une seconde valeur pour remplacer la premiere

```
1       # !/bin/bash
2
3       declare -a index_array=(142 65 2 98)
4
5       echo ${index_array[0]}
6       index_array [0]=366
7       echo ${index_array[0]} ←= le fait d'ajouter une seconde valeur (366) remplacera la première (142)
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
366
```

=> et même si en valeur final on utilise [@] (@pour l'intégralité des valeurs), on voit bien que 366 à définitivement remplacé 142 :

```
(…)
echo ${index_array[@]}
recule@reculeVirtualBox:-$ ./tableau.sh
366 65 2 89
```

Récupérer les indices :

=> il suffit d'utiliser un ! juste avant **index_array** ou **assoc_array** etc …
=> ici exemple avec **assoc_array** (valeurs = prenom et nom / damien recule)
(…)
echo ${!index_array[@]}
recule@reculeVirtualBox:-$ ./tableau.sh
prenom nom

Ajouter un nouvel indice :

Tableau par associatif :

```
1       # !/bin/bash
2
3       declare -A assoc_array=(["prenom"]="damien" ["nom"]="RECULE")
4
5       echo ${assoc_array[@]}
6       assoc_array[age]=45          <== ajout d'une nouvel indice (la valeur : age=45)
7       echo ${assoc_array[@]}       <==
8
```

recule@reculeVirtualBox:-$ ./tableau.sh
45 damien RECULE

Autre ajout :

Ajouter 55 en le placant à la fin de la nouvelle commande déclare :

```
1       # !/bin/bash
2
3       declare -a index_array=(142 65 2 98)
4       declare -A assoc_array=(["prenom"]="damien" ["nom"]="recule")
5
6       declare -a new_index_array=(${index_array[@]} 55) <======
7
8       echo ${new_index_array[@]}
```

recule@reculeVirtualBox:-$ ./tableau.sh
142 65 2 98 55 <====

Pour protéger le fichier en lecture seule, on peut aussi remplacer la première commande déclare par readonly :

```
1       # !/bin/bash
2
3       readonly -a index_array=(142 65 2 98)    <=========
4       declare -A assoc_array=(["prenom"]="damien" ["nom"]="recule")
5
6       declare -a new_index_array=(${index_array[@]} 55
7
8       echo ${new_index_array[@]}
```

Le tableau à deux dimensions : utilisation des deoubles indices :

1 2 3
4 5 6

la première ligne (1, 2, 3) = [0] et ensuite 00 pour 1, 01 pour 2 et 02 pour 3 suivant l'emplacement des nombres.
la deuxième ligne (4, 5, 6) = [1] et ensuite 10 pour 4, 11 pour 5 et 12 pour 6 suivant l'emplacement des nombres.

[0] [0] = 1
[0] [1] = 2
[0] [2] = 3
[1] [0] = 4
[1] [1] = 5
[1] [2] = 6

Exemple

```
1       # !/bin/bash
2
3       declare -A assoc_array=([0:0]=1 [0:1]=2 [1:0]=4 [1:1}=5
4
5       echo ${!assoc_array[@]}
6       echo ${assoc_array[@]}
```

```
recule@reculeVirtualBox:-$ ./tableau.sh
0:1 0:0 1:0 1:1
2 1 5 4
```

Ensemble de commandes de réseau Linux

Je vous présente un ensemble de commandes de réseau Linux que j'ai l'habitude d'utiliser afin de me permettre de configurer, maintenir, dépanner et gérer le réseau de mes serveurs Linux.

Rechercher un article ?

1) Introduction

Dans cet article, je vous présente un **ensemble de commandes de réseau Linux** que j'ai l'habitude d'utiliser afin de me permettre de configurer, maintenir, dépanner et gérer le réseau de mes serveurs Linux.

2) Les fichiers

Je vous liste d'abord des fichiers qui peuvent vous être utile lors de vos manipulations réseau.

- **/etc/hosts** : associer des adresses IP à des noms d'hôtes
- **/etc/networks** : associer des noms d'hôtes à des adresses IP
- **/etc/services** : noms des services tcp/udp ainsi que leur numéro de port

3) Les commandes

3.1) ping

Sans surprise, j'utilise la fameuse commande `ping` , elle est souvent exploitée pour tester la connectivité entre deux systèmes sur un réseau local (LAN) ou un réseau étendu (WAN). Pour information cette commande utilise le protocole ICMP (Internet Control Message Protocol) pour communiquer avec les nœuds d'un réseau.

Vous indiquerez dans la commande simplement une adresse IP ou un nom d'hôte :

```
$ ping 192.168.1.3

PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.636 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=6.72 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.327 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=64 time=0.352 ms
64 bytes from 192.168.1.3: icmp_seq=5 ttl=64 time=0.265 ms
^C
--- 192.168.1.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.265/1.661/6.726/2.535 ms
```

Voici les options disponibles que je suis susceptible d'utiliser :

- **-c** : nombre de paquet
- **-i** : intervalle de relance

3.2) arp

J'utilise l'outil `arp` pour traduire les adresses IP en adresses physique. Ci-dessous quelques exemples d'utilisation :

Afficher l'adresse MAC de l'ip 192.168.1.3 :

```
$ arp 192.168.1.3

Address                  HWtype  HWaddress           Flags Mask            Iface
node01                   ether   02:42:ac:11:00:37   C                     ens3
```

Afficher la table ARP afin de connaître les adresses MAC des machines de votre réseau :

```
$ arp

Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.0.120             ether   02:42:ac:11:00:78   C                     ens3
172.17.0.67              ether   02:42:ac:11:00:43   C                     ens3
172.17.0.1               ether   02:42:f9:85:98:f5   C                     ens3
172.17.0.130             ether   02:42:ac:11:00:82   C                     ens3
node01                   ether   02:42:ac:11:00:37   C                     ens3
172.16.0.47              ether   06:fe:47:9d:e6:b8   C                     ens3
```

Ajouter une entrée dans la table :

```
$ arp -s 192.168.1.3 02:42:ac:11:00:37
```

3.3) traceroute

Commande qui m'est très utile pour découvrir la source de blocage d'un paquet, puisqu'elle permet de suivre le chemin complet de votre système local à un autre système réseau. Elle affiche le nombre de sauts (adresses IP du routeur) dans le chemin emprunté pour atteindre le serveur final.

```
$ traceroute google.com

traceroute to google.com (216.58.215.46), 30 hops max, 60 byte packets
1  _gateway (192.168.0.1)  1.060 ms  2.942 ms  2.894 ms
2  10.44.64.1 (10.44.64.1)  22.544 ms  22.936 ms  22.843 ms
3  213-245-252-89.rev.numericable.fr (213.245.252.89)  22.817 ms  22.786 ms  23.535 ms
4  172.19.132.146 (172.19.132.146)  23.969 ms  23.942 ms  23.985 ms
5  72.14.196.224 (72.14.196.224)  23.905 ms  23.853 ms  23.764 ms
6  108.170.244.193 (108.170.244.193)  23.684 ms 108.170.245.1 (108.170.245.1)  31.061 ms  23.020
```

3.4) ip

Avant j'avais tendance à utiliser la commande `ifconfig` mais comme cette commande est de moins en moins maintenue , il est conseillé d'utiliser maintenant la commande `ip` permettant d'afficher et manipuler le routage, les périphériques réseaux et les interfaces.

3.4-1) Affichage d'informations

```
### Afficher des informations sur toutes les interfaces réseau
ip a

### N'afficher que de l'IPv4
ip  -4 a
### N'afficher que de l'IPv6
ip  -6 a

### Ne montre que l'interface ens3
ip a show ens3

## Affiche uniquement les interfaces à l'état UP
ip link ls up
```

3.4-2) Assigner/Supprimer une adresse IP à une interface réseau

```
### Assigner l'adresse 192.168.1.3 avec le masque sous réseau 255.255.255.0 à l'interface
ip a add 192.168.1.3/255.255.255.0 dev ens3
## ou
ip a add 192.168.1.3/24 dev ens3

### Supprimer l'adresse IP 192.168.1. de l'interface ens3
ip a del 192.168.1.3/24 dev ens3
```

3.4-3) Changer l'état d'une interface en UP ou DOWN

```
### Désactiver l’état du périphérique ens3
ip link set dev ens3 down

### Rétablir l’état du périphérique ens3
ip link set dev ens3 up
```

3.5) route

J'utilise la commande `route` afin d'afficher ou de manipuler la table de routage IP d'un système Linux. Elle est principalement utilisée pour définir un chemin de route statique dans les tables de route. Voici mes différents cas d'utilisation.

3.5-1) Afficher la table de routage IP

```
$ route

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.17.0.1      0.0.0.0         UG    0      0        0 ens3
10.32.0.0       *               255.240.0.0     U     0      0        0 weave
172.17.0.0      *               255.255.0.0     U     0      0        0 ens3
172.18.0.0      *               255.255.255.0   U     0      0        0 docker0
```

3.5-2) Ajouter un itinéraire réseau à la table de routage

3.5-3) Supprimer une entrée de route spécifique de la table de routage.

```
$ sudo route del -net 192.168.1.3/24 gw 192.168.1.1 dev eth0
```

Copier

3.6) nslookup

Je me sers de cette commande afin d'interroger le serveur DNS dans le but de traduire une adresse IP en un nom de domaine, ou inversement.

```
$ nslookup google.com

Server:         62.210.16.6
Address:        62.210.16.6#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.213.142
```

Copier

3.7) dig

dig (Domain Information Groper) est un outil flexible pour interroger des informations liées au DNS telles que l'enregistrement A, CNAME, l'enregistrement MX, etc.

```
$ dig facebook.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> facebook.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57188
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;facebook.com.                  IN      A

;; ANSWER SECTION:
facebook.com.           84      IN      A       157.240.21.35

;; Query time: 1 msec
;; SERVER: 62.210.16.6#53(62.210.16.6)
;; WHEN: Mon Sep 23 11:39:32 UTC 2019
;; MSG SIZE  rcvd: 57
```

Copier

3.8) netstat

La commande `netstat` m'offre un moyen simple d'examiner chacune de mes connexions réseau et de mes sockets ouverts.

3.8-1) Écouter les ports et les programmes

Par exemple, la commande suivante affiche tous les ports TCP en mode d'écoute et les programmes en cours d'écoute.

```
$ sudo netstat -lntp

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      995/sshd
tcp6       0      0 :::22                   :::*                    LISTEN      995/sshd
```

Copier

> **Information**
>
> Remplacez l'option `-t` par un `-u` pour examiner les ports UDP.

3.8-2) Afficher les statistiques par protocole

Par défaut, les statistiques sont affichées pour les protocoles TCP, UDP, ICMP et IP. Le paramètre `-s` peut être utilisé pour spécifier cet ensemble de protocoles.

```
### Statistique de tous les protocoles
netstat -s

### Statistique que pour le protocole TCP
netstat -st

### Statistique que pour le protocole TCP
netstat -su
```

3.9) nmap

L'utilitaire `nmap` me permet de vérifier le port ouvert sur un serveur.

```
$ nmap google.com

Starting Nmap 6.40 ( http://nmap.org ) at 2018-07-12 09:23 BST
Nmap scan report for google.com (172.217.166.78)
Host is up (0.0036s latency).
rDNS record for 172.217.166.78: bom05s15-in-f14.1e100.net
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 4.92 seconds
```

On peut aussi scanner une plage d'IP :

```
### Scanner par masque sous réseau (va scanner de l'ip 192.168.1.0 à 192.168.1.255)
nmap 192.168.1.0/24

### Scanner la plage IP de 192.168.1.1 à 192.168.1.200
nmap 192.168.1.1-200
```

3.10) tcpdump

Dans le cas d'une utilisation d'une interface graphique j'aurai plus tendance à utiliser l'analyseur de paquets libre et gratuit wireshark mais en ligne de commande j'utilise l'outil `Tcpdump` , qui est largement utilisé pour capturer et analyser les paquets TCP/IP transmis ou reçus sur un réseau ou sur une interface spécifique.

Voici les options que j'utilise le plus souvent :

- **-i** : utiliser une interface réseau
- **-A** : Voir le contenu d'un paquet IP
- **port** : Filtrage par port
- **host** : adresse de destination et/ou source
- **dst** : adresse de destination
- **src** : adresse source
- **-w** : capturer les paquets dans un fichier qui pourra être analysé plus tard

```
### écouter le port http (80) sur l'interface ens3 et voir le contenu du
tcpdump -A -i ens3 port http

### Affiche tous les paquets en provenance de 192.168.1.2 vers 192.168.1.3 sur le port 22 en tcp.
tcpdump src host 192.168.1.2 and dst host 192.168.1.3 and port 22 and tcp

### stocker la capture dans le fichier capture.tdp
tcpdump -w capture.tdp

### lire la capture dans le fichier capture.tdp
tcpdump -v -r capture.tdp
```

3.11) UFW

UFW est un nouvel outil de configuration simplifié en ligne de commande qui est une alternative à l'outil iptables. Il est par défaut sur les distributions Debian et Ubuntu Linux et est utilisé pour ajouter/supprimer/modifier/réinitialiser les règles de filtrage de paquets du pare-feu de votre système.

Avant de rajouter des règles, il faut d'abord vérifier le statut de l'outil UFW à l'aide de la commande suivante:

```
$ sudo ufw status

Status: active
```

S'il n'est pas activé alors lancez la commande suivante :

```
$ sudo ufw enable
```

Par défaut, UFW bloquera toutes les connexions entrantes et autorisera toutes les connexions sortantes. Cela signifie que toute personne essayant d'accéder à votre serveur ne pourra pas se connecter à moins que vous n'ouvriez spécifiquement un port, tandis que toutes les applications et tous les services exécutés sur votre serveur pourront accéder au monde extérieur.

Autoriser et refuser les connexions :

```
### Autoriser un protocole ou une ip
sudo ufw allow ssh
sudo ufw allow 80/tcp

### Autoriser l’accès en sortie à un serveur ssh
ufw allow out 22/tcp


### Autoriser l’accès en entré (de l’extérieur) à notre serveur en ssh
ufw allow in 22/tcp

## Seul l'ip 192.168.1.3 est autorisée à accéder à notre serveur en ssh
ufw allow from 192.168.1.3 to any port 5789

### Autoriser une plage de port
sudo ufw allow 1000:2000/tcp
```

Pour refuser vous remplacez `"allow"` par `"deny"`. N'oubliez pas de charger vos nouvelles règles avec la commande ci-dessous :

```
$ ufw reload
```

Voici la commande pour vérifier l'état actuel de votre firewall :

```
$ ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22                         ALLOW IN    Anywhere
22 (v6)                    ALLOW IN    Anywhere (v6)
```

Copier

Chaque règle possède un numéro que vous pouvez lister avec la commande suivante :

```
$ ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 22 (v6)                    ALLOW IN    Anywhere (v6)
```

Copier

Supprimer simplement une règle d'après son numéro :

```
$ sudo ufw delete [numéro]
```

Copier

4) Conclusion

Voici les commandes réseau Linux que j'utilise habituellement, nous aurons peut-être l'occasion de voir plus en détails certaines d'entre elles. N'hésitez pas aussi à me dire dans l'espace commentaire quel outil utilisez-vous le plus !

linux réseau

23/09/2019

article rédigé par **ajdaini-hatim**

-- BASH SUITE –

(Manipulations sur les chaines de caractères)

Rappel :

Les guillemets uniques (‘) ne sont pas suffisants pour que le texte soit interprété comme une commande, il faut utiliser les double-guillemets (‘’) :

Exemple en ligne de code (fichier => **papa.sh**)

```
1  # !/bin/bash
2
3  name=’world’
4  str=’hello $name’
5
6  echo $str
```

```
recule@reculeVirtualBox-:$ ./papa.sh
hello $name
```

=> on peut voir qu’au lieu d’exécuter $name comme valeur world, le fait d’avoir mis des guillemets uniques (‘world’), la ligne de commande n’a pas été interprétée mais juste lu comme un une ligne de texte.

=> pour que la ligne de commande fonctionne, il faut utiliser les doubles-guillemets, cela permet à ta ligne de code d’être correctement interprétée :

```
1  # !/bin/bash
2
3  name=”world”
4  str=”hello $name”
5
6  echo $str
```

```
recule@reculeVirtualBox-:$ ./papa.sh
hello world
```

=> Et voilà !

Attention : si entre guillemets ce n’est pas un texte mais une commande (ex: ls -l), il ne faut pas utiliser les doubles-guillemets mais les guillemets simple (‘ ‘) [=> ceux de la touche 7]

Commande pour noter le nombre de caractère d'une chaine : avec {#str}

```
1  # !/bin/bash
2
3  str="hello world daminus !"
4
5  echo ${#str}
```

⬅======== cette commande notera le nombre de caractère écrit (hello (…) => 21 caractères

```
recule@reculeVirtualBox-:$ ./papa.sh
21
```

=> Et voilà !

Pour extraire une partie d'une chaine : ${str:6}

```
1  # !/bin/bash
2
3  str="Hello world !"
5
6  echo ${str:6}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
world
```

=> Et voilà ! la commande extrait le texte de la chaine « Hello world ! » à partir du 6 caractère, à savoir l'espace juste avant world !

Pour ne récupérer que Hello :

```
1  # !/bin/bash
2
3  str="Hello world !"
5
6  echo ${str:0:5}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Hello
```

=> Ici, on écrit ${str:0:5} avec zéro (0) comme premier caractère pour indiquer que je souhaite prélever le texte depuis le premier caractère et 5 pour délimiter la fin de l'extraction, ce qui donne les 5 premiers caractères (Hello).

Pour passer un texte en MASJUSCULES : ${str^^}

```
1  # !/bin/bash
2
3  str=”Hello world !”
5
6  echo ${str^^}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
HELLO WORLD !
```

=> Ici, le texte Hello world ! devient HELLO WORLD !

Pour passer un texte en minuscules : ${str^}

```
1  # !/bin/bash
2
3  str=”Hello world !”
5
6  echo ${str^}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
hello world !
```

=> Ici, le texte Hello world ! devient hello world !

Pour remplacer un caractère par un autre : / et //

Pour passer un texte en MASJUSCULES : ${str^^}

```
1  # !/bin/bash
2
3  str=”Hello world !”
5
6  echo ${str/l/k}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Heklo world !
```

=> Ici, la commande a changé le premier caractère rencontré correspondant à un l par un k, sans toucher aux autres, ce qui nous donne Heklo au lieu de Hello.
=> pour que tous les l soient modifiés, il faut doubler le / => // => ${str//l/k}

```
1  # !/bin/bash
2
3  str=”Hello world !”
5
6  echo ${str//l/k}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Hekko workd
```

=> Ici, le texte Hello world ! devient hekko workd

Pour demander à modifier juste la première lettre de la chaine, il faut ajouter un # devant le premier de la chaine, à savoir H : ${str/#H/k}

```
1  # !/bin/bash
2
3  str="Hello world !"
5
6  echo ${str/#H/s}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
sello world !
```

=> si vous essayez d'écrire ${str/#l/k} avec un l pour changer le premier l rencontré, cela ne fonctionnera pas, le caractère l n'étant pas le premier caractère de la chaine. (# ne prendra en compte que le premier caractère).

=> à l'inverse de #, le signe % désigne le dernier caractère de la chaine, ici le !
=> pour le subtiliser à un espace par exemple :

```
1  # !/bin/bash
2
3  str="Hello world !"
5
6  echo ${str/%!/T}  <========  donc % + le dernier caractère (!) / puis T
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Hello world T
```

Pour faire une suppression :

```
1  # !/bin/bash
2
3  str="Hello world !"
4
5  echo ${str//l/}  <= //l  (ensemble des l) + / + rien (les l seront donc remplacés par rien, donc supprimés.
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Heo word !
```

Autre exemple pour faire une suppression :

```
1  # !/bin/bash
2
3  str="Hello world !"
4
5  echo ${str#H*o}  <= # (depuis le premier caractère) + H + * (jusqu'à) + o (le 1er o rencontré).
```

```
recule@reculeVirtualBox-:$ ./papa.sh
world !
```

=> il a donc supprimé depuis le premier caractère de la chaine (H) jusqu'au premier o rencontré => Hello

Notes : si vous doublez le ##, la commande comprendra qu'il faut s'arrêter au dernier o rencontrer et effacera donc Hello wo

```
1  # !/bin/bash
2
3  str="Hello world !"
4
5  echo ${str##H*o}  ←= ##* (depuis le premier caractère H + jusqu'au dernier o rencontré).
```

```
recule@reculeVirtualBox-:$ ./papa.sh
rld !
```

=> il a donc supprimé depuis le premier caractère de la chaine (H) jusqu'au dernier o rencontré => Hello wo

Enchainement de commande pour écrire **''Helli world !''** au lieu d' **''Hello world'' ! :**

```
1  # !/bin/bash
2
3  str="Hello world !"
4
5  echo ''${str :0 :4}i ${str :5}''  ←= (extraire depuis le premier caractère jusqu'au 4eme Hell puis ajouter i) +
(extraire                                    (extraire depuis le 5eme caractère jusqu'à la fin world)
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Helli world
```

Replacement avec des variables :

*=> printf (pour l'anglais **print formatted**, soit « imprimer formater ») est une commande Unix permettant de faire afficher une chaîne de caractères à l'écran.*

C'est aussi un nom de fonction du langage C, et de nombreux autres langages informatiques permettant d'afficher une ou plusieurs variables de façon formatée dans le flux de sortie.

Commande **printf** :

%c : caractère

%s : chaine de caractères

%d : nombre entier

%f : nombre flottant

```
1  # !/bin/bash
2
3  animal=chat
4  age=5
5
6  printf ''Bonjour le %s de %d ans\n'' ${animal} ${age}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Bonjour le chat de 5 ans
```

=> Commande printf + ‘’Bonjour le + %s (prendra en compte la première variable rencontrée qui contiendra une chaine de caractère, à savoir ${animal} qui rapporte à animal=’chat’) + de + %d (nombre entier, se reportera donc à la première variable rencontrée qui contiendra un nombre entier [à savoir ${age] qui rapporte à age=5) => ce qui donne Bonjour le chat de 5 ans.

On peut même doubler l’opération de la commande **printf** :

%c : caractère

%s : chaine de caractères

%d : nombre entier

%f : nombre flottant

```
1  # !/bin/bash
2
3  animal=chat
4  age=5
5  ami=chien
6  annee=10
7
8  printf ‘’Bonjour le %s de %d ans\n’’ ${animal} ${age} ${ami} ${annee}
```

```
recule@reculeVirtualBox-:$ ./papa.sh
Bonjour le chat de 5 ans
Bonjour le chien de 10 ans
```

Samba

(partage de dossiers et imprimantes dans un réseau local)

Le **partage de dossiers et d'imprimantes dans un réseau local** est une fonctionnalité des systèmes d'exploitation modernes permettant d'accéder à des ressources d'un ordinateur (dossiers de données et imprimantes) à partir d'un autre ordinateur situé dans un même réseau local (réseau domestique ou d'entreprise).

Le logiciel **Samba** est un outil permettant de partager des dossiers et des imprimantes à travers un réseau local. Il permet de partager et d'accéder aux ressources d'autres ordinateurs fonctionnant avec des systèmes d'exploitation Microsoft® Windows® et Apple® Mac OS® X, ainsi que des systèmes GNU/Linux, *BSD et Solaris dans lesquels une implémentation de Samba est installée.

Pour partager de manière simple des ressources entre plusieurs ordinateurs, l'utilisation de Samba est conseillée.

SMB : (Server Message Block) : Le protocole SMB (Server Message Block) est un protocole de partage de fichiers réseau qui permet à des applications installées sur un ordinateur d'accéder en lecture et en écriture à des fichiers et de solliciter des services auprès de programmes serveur sur un réseau informatique.

Installation :

recule@recule-VirtualBox:-$ sudo apt install samba
&
recule@recule-VirtualBox:-$ sudo apt-get install samba
[INSTALLATION]

Supprimer le fichier par défaut (**smb.conf**) en utilisant rm :

recule@recule-VirtualBox:-$ sudo rm /etc/samba/smb.conf
[sudo] Mot de passe de recule :

Ouvrir un nouveau fichier **smb.conf** avec nano :

recule@recule-VirtualBox:-$ nano /etc/samba/smb.conf

Sur nano

```
[Mon-Partage]
        Path=/home/recule/Bureau
        browseable=yes
        read only=no


# notes : [NOM DU REPERTOIRE]
          Path=/ (le chemin ou sera placé le répertoire communiquant)
          Browseable (visibilité sur le réseau)
          Read only (pour lecture seul ou pas)
```

Ajouter un utilisateur à notre partage :

recule@recule-VirtualBox:-$ sudo smbpasswd -a root (nom à choisir, ici root)
New SMB password :
Retype new SMB password :

Demarrer le service SAMBA :

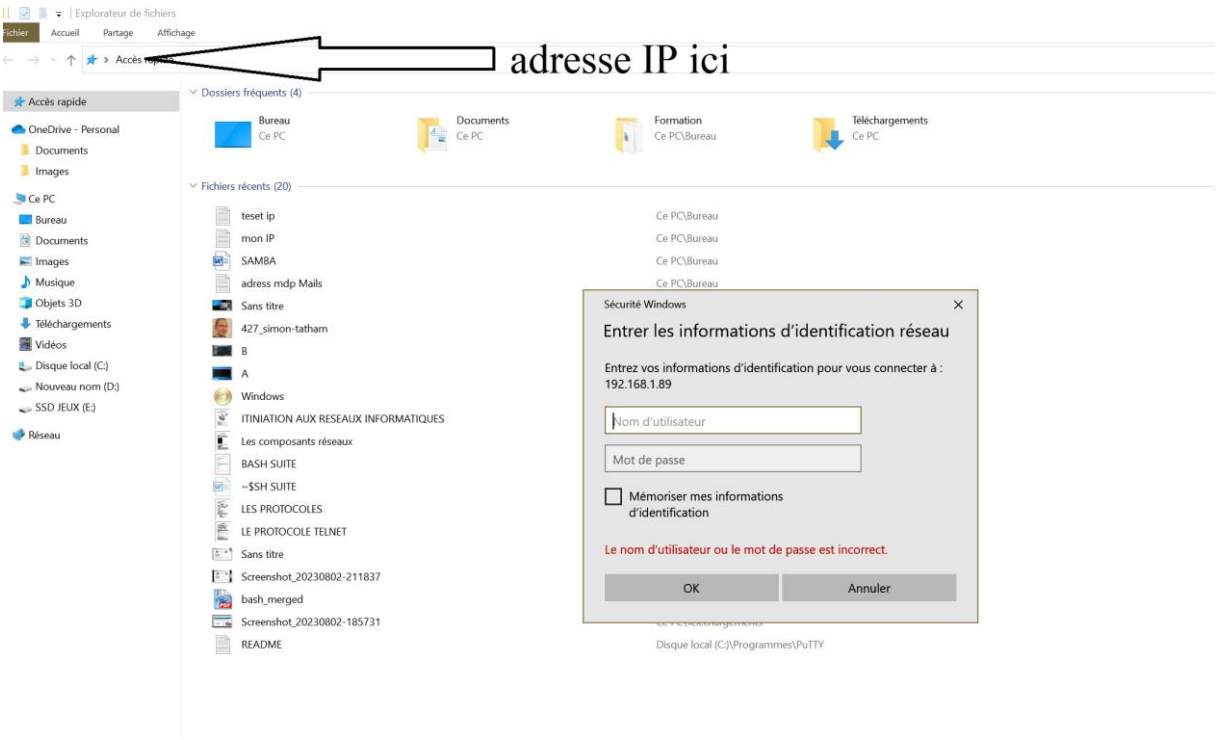
recule@recule-VirtualBox:-$ sudo /etc/init.d/smbd restart
Restarting smbd (via systemctl) : smbd.service.

=> Voilà, samba a démarré avec succès !

MAINTENANT ALLER SUR WINDOWS :

Taper cette ***adresse IP*** sur la barre d'adresse de l'explorateur fichiers.

\\192.168.1.89 + entrée



Un dossier de partage apparaitra dans **/réseau/192.168.1.89/Mon-Partage**, nous mettant en contacte direct avec le bureau de ma distribution Linux.

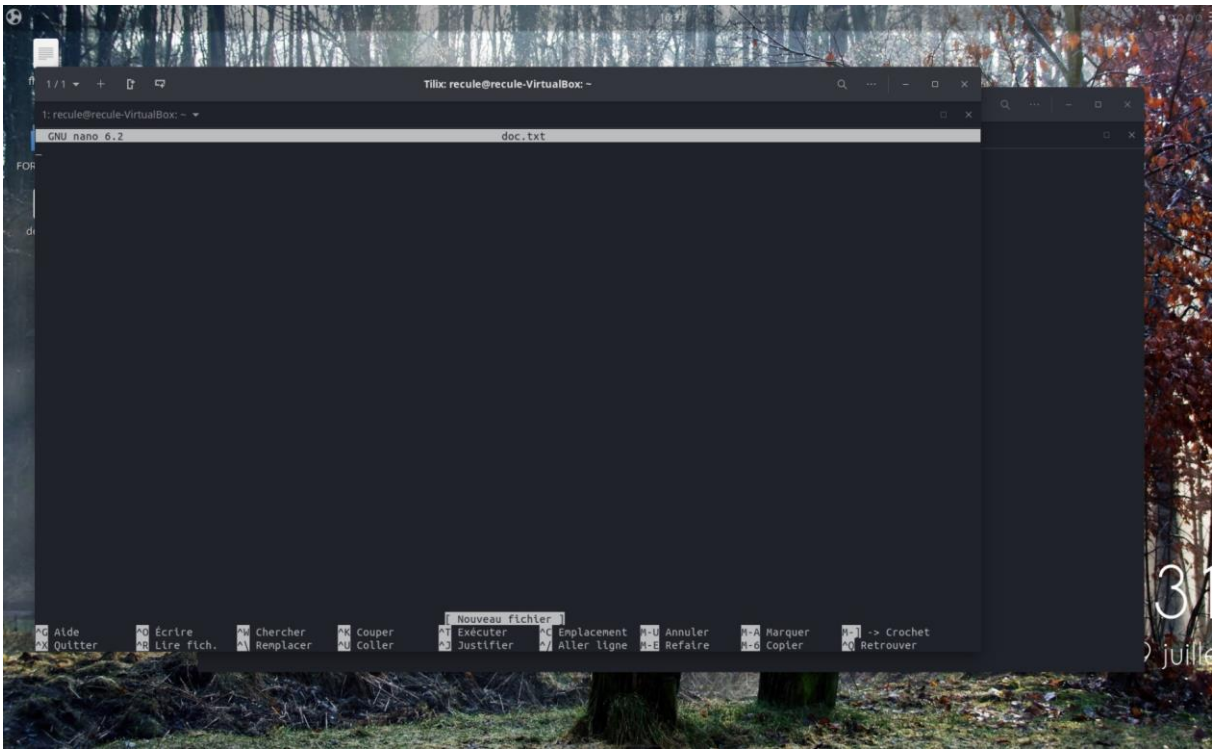
EDITEUR MODE CONSOLE

(Outil Nano)

Créer fichier.txt avec nano et l'ouvrir avec l'outil nano :

recule@recule:-$ nano doc.txt

Outil nano :



=> les différentes commandes sont disponibles en bas de fenêtre :

- sauvegarder **Ctrl+o**

- fermer **Ctrl+x**

- aide **Ctrl+g**

- couper **Ctrl+k**

- coller **Ctrl+u**

etc…

Commandes shell à l'ouverture du fichier :

=> les options de nano :

nano -B doc.txt => permet de toujours garder une copie de fichier de sauvegarde au cas où, cela permet de pouvoir revenir sur une version antérieure si besoin.

nano -l doc.txt => ajoute les numéros de lignes au texte.

=> notes : Vous remarquerez que lorsque vous sauvegardez le fichier après avoir écrit votre script, nano ajoute une ligne vide supplémentaire. Cette ligne ajoutée peut, lors de certaines utilisations, poser problèmes et générer des erreurs, voici la commande pour annuler cette ligne finale :

nano -L doc.txt

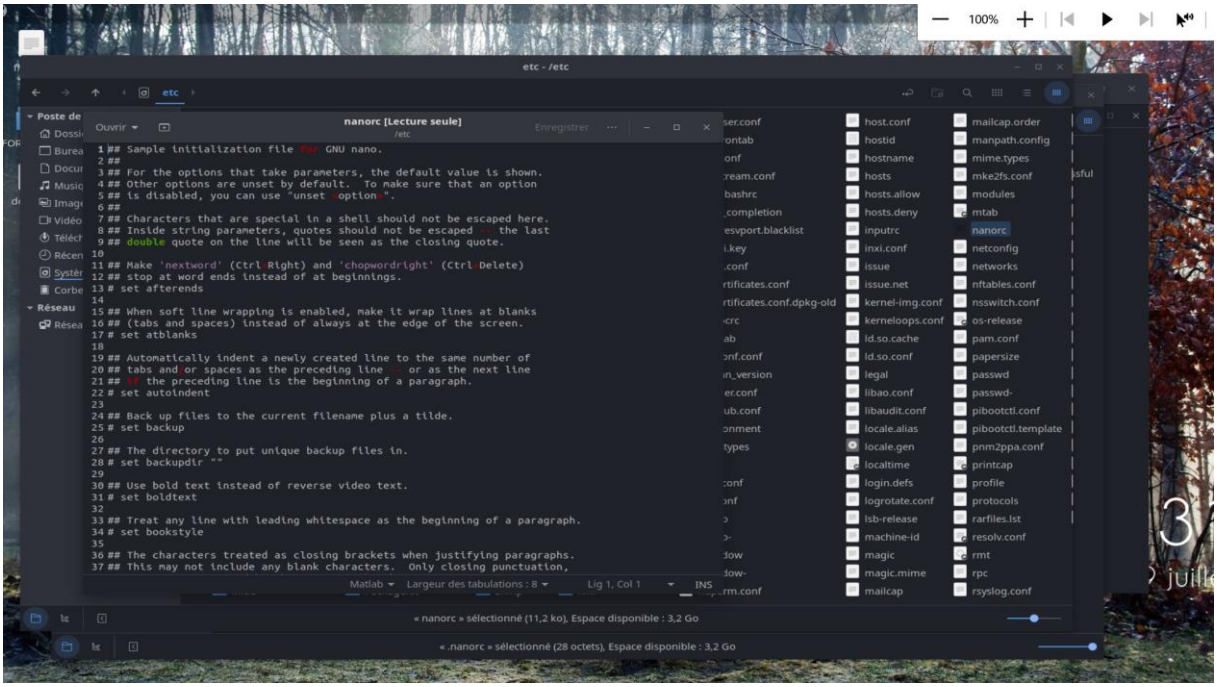
recule@recule:-$ nano -L doc.txt

Lorsque l'on se trouve dans l'outil nano on doit normalement utiliser le clavier pour exécuter les taches et déplacer le curseur, néanmoins il est possible d'utiliser une commande via au SHELL à l'ouverture du fichier :

nano -m doc.txt (il faut peut-être installer un paquet => **sudo apt-get install gmp**

recule@recule:-$ nano -m doc.txt

Le fichier contenant les données de configuration de nano se trouve ici, toujours dans le répertoire /etc/nanorc :



=> Il comporte les différentes configurations utilisables.

Si vous voulez créer votre propre fichier de configuration afin de faire votre propre réglage ou amélioration :

- se rendre dans votre dossier personnel et créer un fichier .nanorc (avec un point devant le nom) pour qu'il soit caché :

recule@recule:-$ sudo touch .nanorc

(ou bien le faire depuis le mode graphique avec clique droit de la souris))

L'ouvrir avec nano :

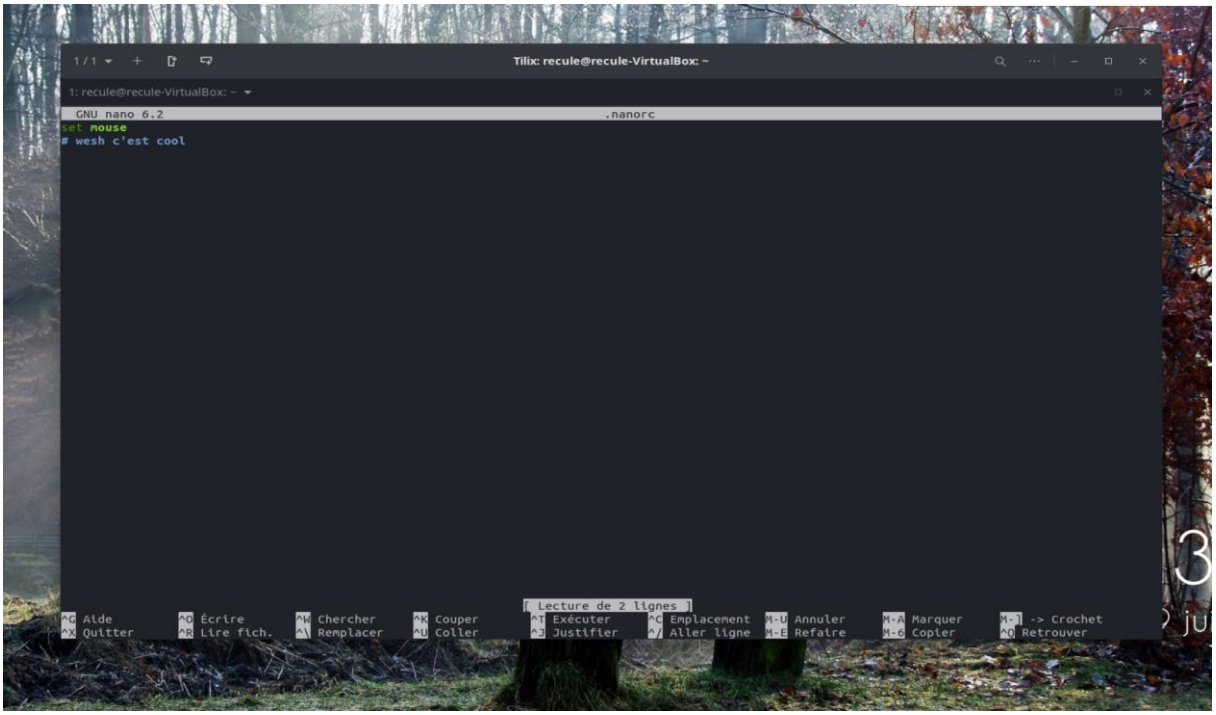
recule@recule:-$ sudo nano .nanorc

et écrire un exemple de commande à écrire à l'intérieur :

set mouse

=> cette option permet, par exemple, de pouvoir utiliser la souris en utilisant nano et pouvoir sélectionner les caractères textuels etcc ….

(+ n'importe quelle commande d'annotation optionnelle, ce n'est pas obligatoire bien entendu)



PROCESSUS

Nos ordinateurs ne fonctionnent que par le biais de programmes. Lorsqu'ils sont en cours d'exécution (ou d'utilisation) ces programmes deviennent des " process " ou des processus en français. Que sont-ils et quels sont les différents états d'un process ?

Qu'est-ce qu'un processus en informatique

En informatique, un processus est l'instance d'un programme informatique en cours d'exécution par un ou plusieurs threads d'un ordinateur. Concrètement, cela signifie qu'un processus permet l'exécution de diverses instructions par le microprocesseur, en fonction du programme en cours de fonctionnement. Un process contient à la fois le code du programme mais également son activité. En fonction du système d'exploitation utilisé (Linux, macOS, Windows…), un processus peut être constitué de plusieurs threads d'exécution qui exécutent des instructions simultanément. À l'inverse, le multitâche permet à un ordinateur de gérer de multiples processus à l'aide de plusieurs processeurs (CPU).

Les commandes :

La commande **ps** va mon donner des premières informations :

recule@recule-:$ ps

PID	TTY	TIME	CMD
2776	pts/0	00:00:00	bash
2959	pts/0	00:00:00	ps

PID => Processus ID, lorsque le système a interprété une commande, un processus indépendant associé à un identificateur de processus (PID) unique, est créé pour l'exécution de cette commande. Le PID permet au système de suivre l'état en cours de chaque processus.
TTY => Très peu mentionné, pourtant le sous-système TTY est au cœur de la conception de Linux et UNIX en général. Sur Unix, tty provient du mot anglais TeleTYpewriter que l'on peut traduire par téléimprimeur, télétype ou téléscripteurs. Dans Linux TOUT est fichier et
Donc le terminal est également représenté comme un fichier, il vous permet d'interagir avec le système en passant sur les données que vous entrez via le SHELL.
Dans sa signification la plus courante, le terminal est synonyme de TTY. Certaines TTYS sont fournies par le noyau pour le compte d'un périphérique matériel, par exemple avec l'entrée provenant du clavier et de la sortie à un écran de mode texte, ou avec l'entrée et la sortie transmises sur une ligne de série.
Les autres TTYS, parfois appelées pseudo-états, sont fournis (à travers une mince couche de noyau) par des programmes appelés émulateurs de terminaux, tels que Xterm (exécuté dans le système X Window), l'écran (qui fournit une couche d'isolement entre un programme et

un autre terminal), SSH (qui relie un terminal sur une machine avec des programmes sur une autre machine). Ainsi, votre terminal est représenté par le fichier /dev/tty, une copie de fichiers vers ce dernier montre que l'on obtient en sortie ce fichier.

TIME => Le moment où a été exécuté le processus.

CMD => Stipule par où nous sommes passé pour lancer le processus, c'est-à-dire avec quelle shell, ici bash.

Passons à un listing un peu plus complet :

Syntaxe standard :
Ps -elf
recule@recule-:$ ps -elf

Syntaxe plus poussée : (listing plus grand)
Ps -auwx
recule@recule-:$ ps -auwx

(une longue liste de processus défile)

Exemple d'information du listing :

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
Root	1	0.0	0.1	225532	9208	?	Ss	15:10	0.01	/sbin/init splash
Recule	1882	0.0	0.1	354628	7636	?	Sst	15:11	0:10	/usr/lib/gvfs-fuse

Concernant les informations données :
En orange => un exemple de processus démarré par le système (root) (au démarrage ou autre)
En violet => un exemple de processus démarré par l'utilisateur.

USER => Information désignant celui qui a lancé le processus.
PID => Désigne l'**I**dentificateur **D**e **P**rocessus unique.
%CPU => Le pourcentage d'utilisation du processeur (CPU).
%MEM => Le pourcentage d'utilisation de la mémoire (RAM).
VSZ => VSZ est la taille de la mémoire virtuelle. Elle inclut toute la mémoire à laquelle le processus peut accéder, y compris la mémoire qui est échangée, la mémoire allouée (mais non utilisée) et la mémoire provenant des bibliothèques partagées.
RSS=> RSS est la partie de la mémoire occupée par un processus qui est conservée dans la mémoire principale (RAM). Elle ne comprend pas la mémoire qui est échangée. Elle inclut la mémoire des bibliothèques partagées, pour autant que les pages de ces bibliothèques soient effectivement en mémoire. Elle inclut toute la mémoire de la pile et du tas.

TTY => TTY est au cœur de la conception de Linux et UNIX en général. (Voir plus haut)
STAT => Le statuts du processus en question.
START => L'heure à laquelle le processus a été lancé.
TIME => Le moment où a été exécuté le processus.
COMMANDE => D'où a été lancé le processus, c'est-à-dire le nom du programme ou de la tâche en cours (avec le chemin /).

=> Peux importe le nombre de processus identique lancé (par exemple si vous lancez plusieurs fois à la suite le lecteur VLC), chacun de ces processus va utiliser le même code d'exécution pour pouvoir exécuter les mêmes instructions, ça ne changera pas. Par contre ce qui va différencier un processus d'une autre, c'est son contexte d'exécution qui va permettre d'avoir un suivi sur sone exécution. Un processus a un code exécutable qui va rester le même peux importe le nombre d'instance que vous lancez, par contre chaque processus est propre à son contexte d'exécution.

Reprenons :

Processus INIT

Lorsque l'on va démarrer son ordinateur, on va avoir le processus INIT qui est en fait le processus 'père' qui va être le premier à se lancer et le dernier à s'arrêter. C'est lui qui va être à l'origine de tout les autres, un processus d'héritable dont découlera les autres processus.

INIT (abréviation d'**init**ialisation) est le premier programme informatique exécuté sur les systèmes d'exploitation basés Unix. Il est exécuté comme un daemon informatique et possède l'identifiant de processus (PID) 1. init est lancé par le noyau (kernel en anglais) et reste actif jusqu'à ce que le système soit éteint.

Commandes :

Lister seulement les processus lancés par l'utilisateur :

Ps -u [nom de l'utilisateur]

recule@recule-:$ ps -u recule

Etat du processus :

=> Pour avoir un état des processus, utiliser la commande **top**, elle se met à jour toutes les 3 secondes.
recule@recule-:$ top

Il y a différent état d'un processus :

=> un processus qui est en exécution aura un état **R** (colonne STAT du listing de la commande **ps**). Une fois top activé, vous pouvez avoir accès à un manuel expliquant les différentes commandes interactives que vous pouvez utiliser, pour cela il faut appuyer sur la touche **H** du clavier. (Touche Q pour quitter le programme top).

=> le **S** identique le processus est en sommeil, il est en cours mais attend que l'utilisateur fasse une action, utilise le dit 'processus'.

=> il existe le **processus zombie**, normalement un processus qui cesse son activité envoie un code de retour (de fin d'exécution) au 'père'. S'il ne fait pas après son arrêt, il en résulte un processus dit « orphelin », même s'il ne coute rien en mémoire, mais pour autant il va être dérangeant car au niveau de la table de processus du système (nombre de processus limité), ce qui pourrait saturer le système (il y a un grand nombre de processus prévu, mais cela est tout de même possible). Il faut éviter ça, pour cela on peut par exemple faire adopter un processus dit « zombie » par un nouveau « père » (par exemple INIT).

=> Chaque processus (qui a contexte d'utilisation propre à chacun) à sa propre entrée standard, sa sortie standard et sa sortie d'erreur (voir le cours sur LES FLUX de redirections), et chaque processus a un propriétaire (systèmes, utilisateur etc …) et a une histoire de priorité.

=> Les priorité : en utilisant le terminal et l'espace graphique en même temps, on a l'impression, à notre niveau, qu'ils s'exécutent en même temps. En réalité, le système donne la main à tour de rôle au terminal puis à l'espace graphique. L'ordonnancement des taches est planifier grâce à un ordonnanceur. Pour que cela soit géré au mieux, il faut un système de priorité, il y a certains programmes qui seront dit « partager » (on leur donne la main lorsque c'est possible).
Il y a une sorte de system d'élection, il y a un processus qui est choisi pour être 'élu', une fois que ce droit lui est retiré, il va passer soit en état « bloqué (ne sera pas dans l'attente d'avoir la main par l'ordonnanceur). Puis, lorsqu'il se retrouvera dans la file d'attente pour être à nouveau prêt à avoir la main pour exécuter des tâches, il passera donc dans l'état que nomme **état prêt**.
Donc 3 états :
- **état élu** : celui auquel le processeur donne la main à un instant T.
- **état prêt** : dans la filez d'attente, prêt à prendre la main (selon des priorités).
- **état bloqué** : ceux qui ne sont pas encore prêt à recevoir la main par l'ordonnanceur.

Notes :
Lorsqu'un processus est en train de s'exécuter (qu'il utilise le microprocesseur), on dit que le processus est dans l'état "élu". Un processus qui se trouve dans l'état élu peut demander à accéder à une ressource pas forcément disponible instantanément (par exemple lire une donnée sur le disque dur).

Pour un OS, c'est exactement la même chose, il va ordonnancer les taches, donner des priorités, c'est une succession d'autorisation données. L'ordonnanceur est un peu comme un chef d'orchestres, il peut lancer un instrument, en bloquer un autre etc …

----- TEMPS D'EXECUTIONS ----

=> Lorsque, par exemple, on lance la commande **ls** :

recule@recule-:$ ls
Bureau doc.txt Documents Images Musiques etc …
recule@recule-:$

Le SHELL exécute la commande et nous redonne la main immédiatement.

Puis, lorsque qu'on lance, par exemple, une commande : **ls -l** (lister plus spécifiquement)

recule@recule-:$ ls -l
(grande liste qui defil)
recule@recule-:$

=> le temps d'attente du listing est beaucoup plus long, pour abréger le processus, nous pourrions faire CTRL+C. En faisant ça, nous venons d' envoyer un signal au processus pour lui dire d'interrompre le processus (défilement du listing).

Processus père et fils

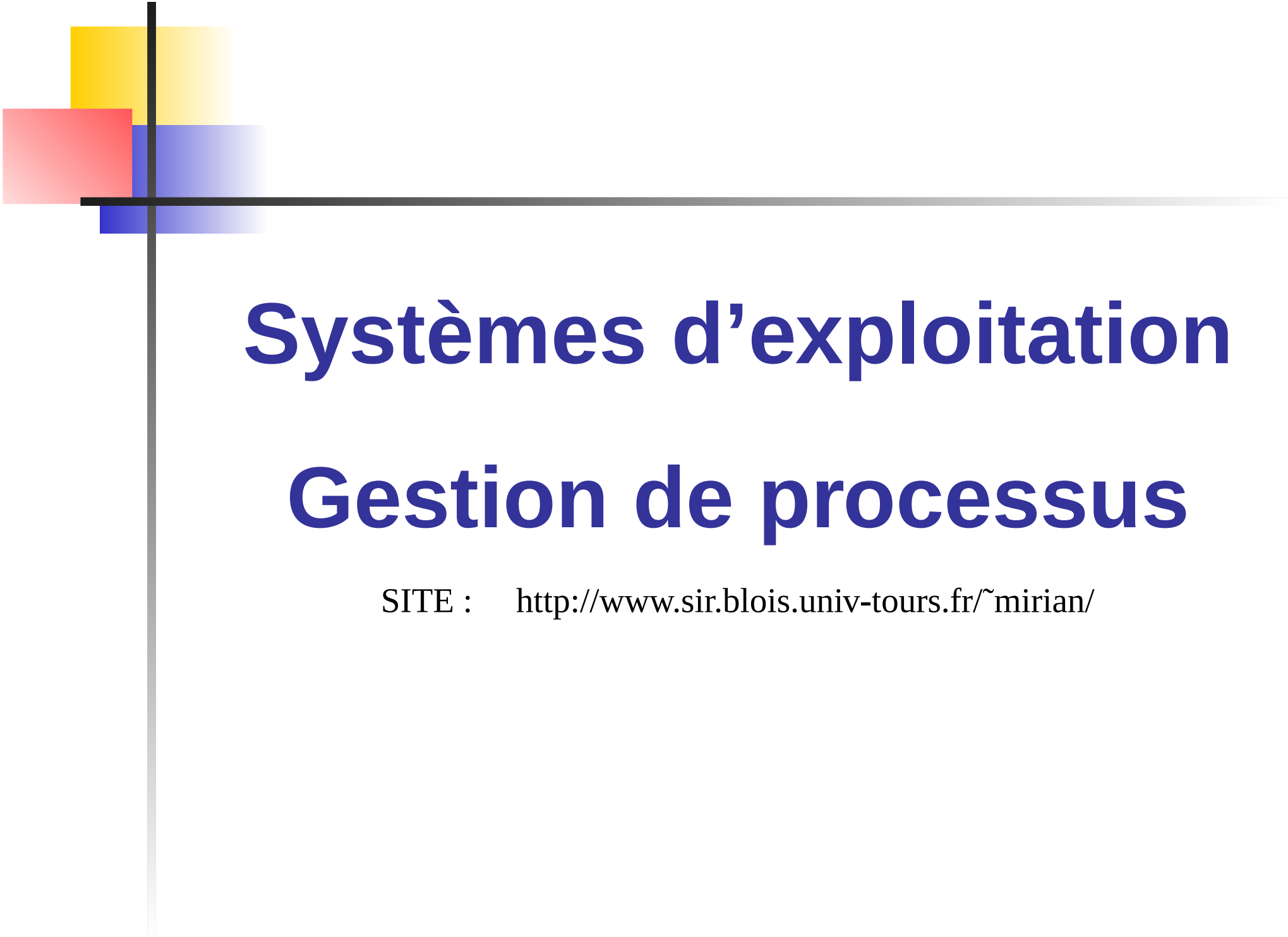
Sur le terminal :

recule@recule-:$ firefox

Si je lance firefox depuis le terminal (Plutôt que le raccourcis du bureau graphique), il se lance dans une fenêtre à coté du terminal, le terminal devient donc le 'père' du processus 'firefox' (qui lui devient son 'fils'. Ainsi, si on ferme le terminal en fermant la fenêtre, Firefox disparait également, si on 'tue' le père, on tue également les processus qui lui sont liés.

=> Parc contre, si on utilise la commande nohup on peut lancer firefox sans qu'il soit en lien avec le terminal :

recule@recule-:$ nohup firefox



Systèmes d’exploitation

Gestion de processus

SITE : http://www.sir.blois.univ-tours.fr/˜mirian/

Les processus, à quoi ça sert?

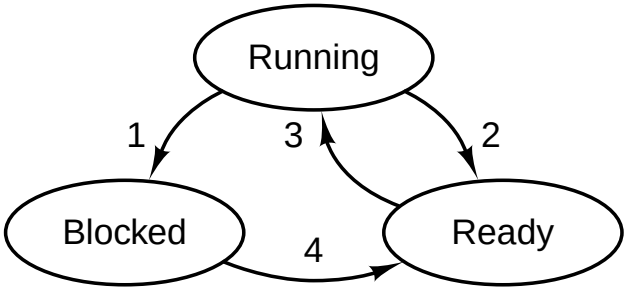
- À faire plusieurs activités ”en même temps”.
- Exemples
  - Faire travailler plusieurs utilisateurs sur la même machine. **Chaque utilisateur a l’impression d’avoir la machine à lui tout seul.**
  - Compiler tout en lisant son mail
- Problème: Un processeur ne peut exécuter **qu’une seule instruction à la fois**.
- BUT: **Partager** un (ou plusieurs) processeur entre différents processus.
- Attention!!! Ne pas confondre processus et processeur

Concept de processus

- Un processus est un programme en exécution
- L'exécution d'un processus doit progresser séquentiellement, cad, à n'importe quel moment *une seule* instruction au plus est exécutée au nom du processus
- Processus $\neq$ Programme

États des processus

- Quand un processus s'exécute, il change d'état.
- Chaque processus peut se trouver dans chacun des états suivants :
  - **En exécution:** Les instructions sont en cours d'exécution (en train d'utiliser la CPU).
  - **En attente:** Le processus attend qu'un événement se produise.
  - **Prêt:** Le processus attend d'être affecté à un processeur.
- Un **seul** processus peut être en exécution sur n'importe quel processeur à tout moment.
- Toutefois, plusieurs processus peuvent être prêts et en attente

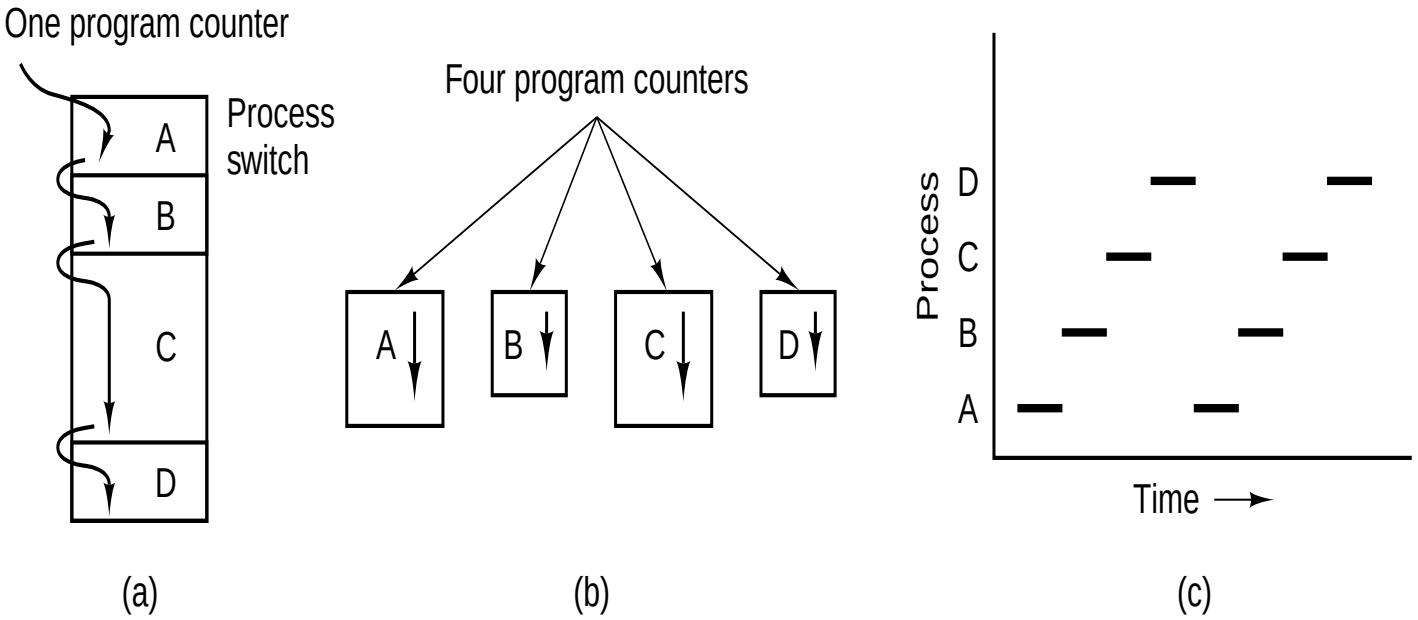


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Processus

- Point de vue conceptuel: chaque processus possède son processeur virtuel.
- Réalité: **le processeur bascule constamment d'un processus à l'autre**.
- Ce basculement rapide est appelé **multiprogrammation**.
- Lorsque le processeur passe d'un processus à un autre, **la vitesse de traitement d'un processus** donné n'est pas uniforme et probablement **non reproductible** si le même processus s'exécute une nouvelle fois.

Processus


One program counter
A
B
C
D
Process switch
(a)
Four program counters
A
B
C
D
(b)
Process
D
C
B
A
Time
(c)

Bloc de contrôle de processus(1)

Chaque processus est représenté dans le SE par un PCB (*process control block*)

PCB	
Pointeur	État du processus
Numéro du processus	
Compteur d'instructions	
Registres	
Limite de la mémoire	
Liste des fichiers ouverts	
. . .	

Bloc de contrôle de processus(2)

- PCB: contient plusieurs informations concernant un processus spécifique, comme par exemple:
  - L'état du processus.
  - Compteur d'instructions: indique l'adresse de l'instruction suivante devant être exécutée par ce processus.
  - Informations sur le scheduling de la CPU: information concernant la priorité du processus.
  - Informations sur la gestion de la mémoire: valeurs des registres base et limite, des tables de pages ou des tables de segments.
  - Informations sur l'état des E/S: liste des périphériques E/S allouées à ce processus, une liste des fichiers ouverts, etc.

Linux/UNIX: Lister des informations sur les processus

Commande ps

```
[mirian@home2 Transp]$ ps -af
UID        PID  PPID  C STIME TTY          TIME CMD
mirian    4576     1  0 21:22 pts/1    00:00:00 wxd
mirian    4705  3949  0 21:28 pts/1    00:00:04 xpdf SE2007-Proces
mirian    5068  3949  0 21:40 pts/1    00:00:00 ps -af
```

- The process ID or PID: a unique identification number used to refer to the process.
- The parent process ID or PPID: the number of the process (PID) that started this process.
- Terminal or TTY terminal to which the process is connected.

Linux/UNIX: Lister des informations sur les processus

En supprimant qqs champs de l'exemple, voilà ps avec des autres options:

```
[mirian@home2 Transp]$ ps au
USER       PID ... TTY      STAT START   TIME COMMAND
root      3093 ...  tty1     Ss+  20:57   0:00 /sbin/mingetty tty1
...
mirian    3881 ...  pts/0    Ss+  21:08   0:00 bash
mirian    3949 ...  pts/1    Ss   21:09   0:00 bash
mirian    4576 ...  pts/1    S    21:22   0:00 wxd
mirian    4705 ...  pts/1    S    21:28   0:00 xpdf SE2007-Process
mirian    4841 ...  pts/1    R+   21:33   0:00 ps au
```

STAT: status du processus

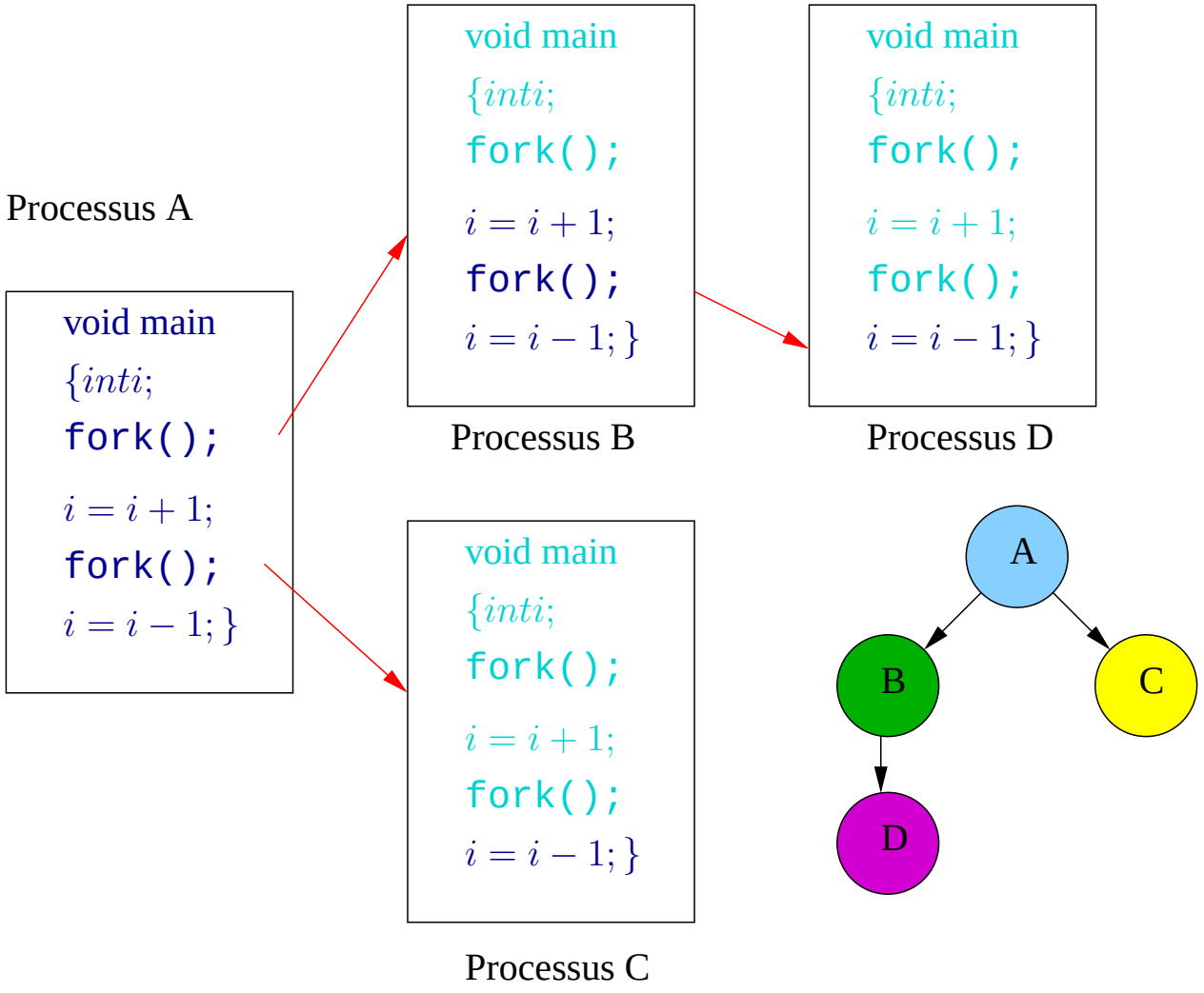
Certains états des processus

- R Running or runnable (on run queue)
- S Interruptible sleep (waiting for an event to complete)
- Z Defunct ("zombie") process, terminated but not reaped by its parent.

Hiérarchie de processus

- Dans certains SE, lorsqu'un processus crée un autre processus, les processus parent et enfant continuent d'être associés d'une certaine manière. Le processus enfant peut lui même créer plusieurs processus, formant un **hiérarchie de processus**.
- **Un processus a un seul parent et peut avoir** 0 **ou plusieurs fils**.
- Linux/UNIX:
  - Si le processus A crée le processus B, A est le parent de B, B est le fils de A (A par défaut, exécute le même code que B) B peut à son tour créer des processus. Un processus avec tous ses descendants forment un groupe de processus représenté par un arbre de processus.
  - `fork` est le seul appel système de création de processus.

Hiérarchie de processus



Linux/UNIX

- Démarrage Linux/UNIX: un processus spécial appelé `init` est présent dans l'image d'amorçage.
- Lorsqu'il s'exécute, il lit un fichier indiquant combien de terminaux sont présents; il génère un nouveau processus par terminal.
- Ce processus attendent une ouverture de session (`login`)
- Si l'une d'elles réussit, le processus de login exécute un **SHELL pour accepter des commandes**.
- Ces commandes peuvent lancer d'autres processus, et ainsi de suite.
- Tous les processus de l'ensemble du SE appartiennent à un arborescence unique, dont `init` est la racine.

Création d'un processus

- Événements provoquant la création d'un processus : initialisation système, exécution d'un appel système par un processus en cours, requête utilisateur sollicitant la création d'un processus.
- Linux/UNIX: L'appel système `fork` crée un clone du processus appelant.
- Après le `fork` les deux processus, père et fils, ont la même image mémoire et les mêmes fichiers ouverts.
- Généralement, le processus enfants exécute l'appel système `execve` pour modifier son image mémoire et exécuter un nouveau programme.
- Exemple:
  - Utilisateur adresse la commande `sort` au shell.
  - Le shell crée un processus fils qui exécute le programme `sort`
- Une fois qu'un processus est créé, le père et le fils disposent de leur propre espace d'adressage: s'il arrive que l'un des processus modifie un mot dans son espace d'adressage, le changement n'est pas visible par l'autre.
- Linux/UNIX: l'espace d'adressage initial du fils est une copie de celui du père : pas de mémoire partagée en écriture.

La fin d'un processus

Un processus peut se terminer suite à l'un des 4 événements :

- Sortie normale, lorsque le processus a terminé sa tâche (sous Unix par l'appel système `exit`)
- Sortie suite à une erreur (e.g. division par 0, inexistence d'un fichier passé en paramètre)
- Tué par un autre processus (sous Unix par l'appel système `kill`)

Toutes les ressources du processus sont libérées par le SE.

Exemple programme C avec `fork`

- Un nouveau processus est crée par l'appel système $fork$
- Le nouveau processus consiste en une copie de l'espace adresse du processus d'origine. Ce mécanisme permet au processus père de communiquer facilement avec son fils.
- Les 2 processus continuent l'exécution à partir de l'instruction située après le $fork$ mais avec une différence: le code de retour pour le $fork$ est zéro pour le nouveau processus (fils), tandis qu'on retourne au père l'identificateur du fils.
- L'appel système $execve$ est utilisé après un $fork$ par l'un des deux processus, afin de remplacer l'espace mémoire du processus par un nouveau programme.
- Si le père n'a aucune tâche à effectuer pendant que le fils s'exécute, il peut émettre un appel système $wait$ afin de sortir lui-même de la file d'attente des processus prêts jusqu'à la terminaison du fils.

Scheduling de processus

- Dans un système multitâche plusieurs processus sont en cours simultanément, mais le processeur ne peut, à un moment donné, exécuter qu'une instruction (d'un programme) à la fois. Le processeur travaille donc en **temps partagé**.
- L'ordonnanceur (scheduler) est le module du SE qui s'occupe de sélectionner le processus suivant à exécuter parmi ceux qui sont prêts.

Scheduling de processus

- Un processus passe entre les diverses files d'attente pendant sa durée de vie (file d'attente des processus prêts (attendant pour s'exécuter), fille d'attentes des périphériques, ...).
- Le SE doit sélectionner les processus à partir de ces files d'attente d'une manière quelconque.
- Le processus de sélection est mené à bien par le **scheduleur** approprié.
- Classification des processus
  - *Tributaire des E/S:* dépense la plupart du temps à effectuer des E/S plutôt que des calculs
  - *Tributaire de la CPU:* génère peut fréquemment des requêtes d'E/S, cad, passe plus de temps à effectuer des calculs

Scheduling des processus

- Le **scheduleur à long terme** (ou scheduleur de travaux)
  - Sélectionne le processus qui doit aller dans la fille de processus prêts.
  - S'exécute moins fréquemment : peut être lent (seconds, minutes).
  - Contrôle le degré de multiprogrammation (le nombre de processus dans la mémoire).
  - Il est important que le scheduleur à long terme réalise un bon mélange de processus tributaires de la CPU et tributaires des E/S
- Le **scheduleur à court terme** (ou scheduleur de la CPU):
  - Choisit parmi les processus prêts celui à être exécuté (alloue la CPU à lui).
  - Appelé assez fréquemment : doit être TRÈS rapide
- Possibilité d'introduire un **scheduleur à moyen terme**
  - *Idée clé:* Il peut être avantageux de supprimer des processus de la mémoire et réduire ainsi le degré de multiprogrammation.
  - Plus tard, un processus peut être réintroduit dans la mémoire et son exécution peut reprendre là ou elle s'était arrêtée
  - Ce schéma est appelé **swapping** (transfert des informations de la mémoire principale à la mémoire auxiliaire et vice-versa)

Scheduling des processus

- Les algorithmes d'ordonnancement (scheduleur) peuvent être classés en deux catégories:

1. **Non préemptif**
   - Sélectionne un processus, puis le laisse s'exécuter jusqu'à ce qu'il bloque (soit sur une E/S, soit en attente d'un autre processus) où qu'il libère volontairement le processeur.
   - **Même s'il s'exécute pendant des heures, il ne sera pas suspendu de force**.
   - Aucune décision d'ordonnancement n'intervient pendant les interruptions de l'horloge.
2. **Préemptif**:
   - Sélectionne un processus et le laisse s'exécuter pendant un délai déterminé.
   - Si le processus est toujours en cours à l'issue de ce délai, il est suspendu et le scheduleur sélectionne un autre processus à exécuter.

Scheduling des processus

- Les divers algorithmes de scheduling de la CPU possèdent des propriétés différentes et peuvent favoriser une classe de processus plutôt qu'une autre
- Critères utilisés:
  - *Utilisation de la CPU*: maintenir la CPU aussi occupée que possible.
  - *Capacité de traitement (Throughput)*: nombre de processus terminés par unité de temps.
  - *Temps de restitution (Turnaround time)*: temps nécessaire pour l'exécution d'un processus.
    Temps de restitution = temps passé à attendre d'entrer dans la mémoire + temps passé à la file d'attente des processus prêts + temps passé à exécuter sur la CPU + temps passé à effectuer des E/S
  - *Temps d'attente:* quantité de temps qu'un processus passe à attendre dans la file d'attente des processus prêts.
  - *Temps de réponse:* temps écoulé à partir du moment où on soumet une requête jusqu'à l'arrivée de la première réponse .

Algorithmes de Scheduling - FCFS

Scheduling du premier arrivé, premier servi
(FCFS: *First-come, first-served*)

- L'implantation de la politique FCFS est facilement gérée avec une file d'attente FIFO.
- Une fois que la CPU a été allouée à un processus, celui-ci garde la CPU jusqu'à ce qu'il la libère (fin ou E/S).
- FCFS est particulièrement incommode pour le temps partagé où il est important que chaque utilisateur obtienne la CPU à des intervalles réguliers.
- Temps moyen d'attente: généralement n'est pas minimal et peut varier substantiellement si les temps de cycles de la CPU du processus varient beaucoup
- Effet d'accumulation (*Convoy effect*): provoque une utilisation de la CPU et des périphériques plus lente que si on permettait au processus le plus court de passer le premier.

Algorithmes de Scheduling - SJF

Scheduling du travail plus court d'abord
(*SJF: Shortest job first*)

- Associe à chaque processus la longueur de son *prochain* cycle de CPU. Quand la CPU est disponible, elle est assignée au processus qui possède le prochain cycle le plus petit. Si deux processus possèdent la même longueur, le FCFS est utilisé.
- SJF est optimal: il obtient le temps moyen d'attente minimal pour un ensemble de processus donné.
- Difficulté: pouvoir connaître la longueur de la prochaine requête de la CPU
- Deux schémas: Non preemptif et preemptif.

Algorithmes de Scheduling - avec des priorités

- Une priorité est associée à chaque processus.
- CPU est allouée au processus de plus haute priorité (priorité $= 1 >$ priorité $= 2$
- Les processus ayant la même priorité sont schedulés dans un ordre FCFS.
- SJF: priorité déterminée par la durée du prochain cycle de CPU.
- Exemple

Processus	Temps de cycle	Priorité
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

- 2 schémas: Preemptif et non preemptif
- Problème: **Famine (*starvation*)** ou blocage indéfinie: processus avec des basses priorités peuvent attendre indéfiniment (ne jamais être exécutés).
- Solution: **Vieillissement (*aging*)**: technique qui consiste à augmenter

Algorithmes de Scheduling - RR

Scheduling du Tourniquet
(*RR: Round Rodin*)

- Chaque processus a une petite unité de temps appelée tranche de temps, ou quantum (en général de 10 à 100 ms).
- Le scheduleur de la CPU parcourt la file d'attente des processus prêts en allouant la CPU à chaque processus pendant un intervalle de temps allant jusqu'à un quantum.
- La file d'attente des processus prêts est implantée comme une file FIFO.
  - Cycle de CPU du processus < 1 quantum
    $\Rightarrow$ le processus lui-même libérera la CPU volontairement
  - Cycle de CPU du processus > 1 quantum
    $\Rightarrow$ l'horloge s'arrêtera et provoquera un interruption au SE
    $\Rightarrow$ Commutation de contexte
    $\Rightarrow$ Processus inséré en queue de la file d'attente

Algorithmes de Scheduling - RR

- Le temps moyen d'attente est souvent très long
- S'il existe n processus dans la file d'attente et quantum q alors:
  - Chaque processus obtient $1/n$ du temps processeur en morceaux d'au maximum q unités de temps.
  - Chaque processus ne doit pas attendre plus de $(n-1)q$ unités de temps jusqu'à sa tranche de temps suivante.
- Performance: dépend de la taille du quantum q
  q très grand $\Rightarrow$ FCFS
  q très petit $\Rightarrow$ partage du processeur
  MAIS... q doit être suffisamment grand par rapport au temps de commutation

Algorithmes de Scheduling - files d'attente à multiniveaux

Scheduling avec les files d'attente feedback à multiniveaux

- Séparer les processus ayant des caractéristiques différentes de cycle de CPU en différentes files. Permet aux processus de se déplacer entre les files.
  - Processus employant beaucoup de temps processeur: file de priorité inférieure
  - Processus tributaire E/S: file de priorité supérieure
  - Processus qui attend trop longtemps dans une file de priorité basse peut être déplacé à un file de priorité supérieure
- Exemple: 3 files d'attente
  - Q_0: quantum $8ns$; Q_1: quantum $16ns$; Q_2: FCFS
  - Un nouveau processus est inséré dans Q_0
  - Processus de Q_0: S'il ne finit pas en $8ns$, alors il est interrompu et placé en Q_1
  - Si Q_0 est vide, le processus en tête de Q_1 prend la CPU. S'il ne termine pas en $16ns$, il est interrompu et mis dans Q_2
  - Processus de Q_2: exécutés avec FCFS (seulement quand Q_0 et Q_1 sont vides)

Algorithmes de Scheduling - Évaluation

Comment choisir un algorithme de scheduling de CPU pour un système particulier?

Il existe différentes méthodes d'évaluation:

- Modélisation déterministe: Prend en compte une charge de travail prédéterminée et définit la performance de chaque algorithme pour cette charge de travail.
- Modèles de file d'attente
- Simulations
- Implantation

Commutation de contexte

- Le contexte d'un processus est l'ensemble des informations dynamiques qui représente l'état d'exécution d'un processus.
- La **commutation** de contexte est le mécanisme qui permet au SE de remplacer le processus élu par un autre processus éligible.
- Pour changer d'un processus à un autre, il faut
  - Sauvegarder l'état de l'ancien processus.
  - Charger l'état sauvegardé pour le nouveau processus.
- Le temps de la commutation de contexte est une surcharge car le système ne réalise pas de travail utile pendant qu'il commute.
- Le temps de la commutation de contexte dépend du support matériel.

Redirections - Linux/UNIX

Lorsqu'un processus démarre de façon standard, 3 fichiers spéciaux sont ouverts automatiquement :

- **Entrée standard** (stdin): descripteur 0, par défaut équivalent au clavier.
- **Sortie standard** (stdout): descripteur 1, par défaut équivalent à l'écran.
- **Sortie erreur standard** (stderr): descripteur 2, par défaut équivalent à l'écran

Ces 3 entrées/sorties peuvent être redirigées afin que la lecture/écriture se fasse à partir/vers d'autres fichiers.

- Pour rediriger la sortie d'un processus vers un fichier: $>$
- Pour rediriger la sortie d'un processus vers un fichier sans écraser son contenu: $>>$
- Pour lire l'entrée d'un processus à partir d'un fichier: $<$

Redirections - Linux/UNIX

`sort`	Trie les lignes tapées au clavier; écrit le résultat trié à l'écran
`sort < f1`	Trie les lignes du fichier $f1$; écrit le résultat trié à l'écran
`sort > f2`	Trie les lignes tapées au clavier; écrit le résultat trié dans le fichier $f2$ (si $f2$ existait avant il est efface et recréé)
`sort » f2`	Trie les lignes tapées au clavier; attache le résultat trié à la fin du fichier $f2$ (si $f2$ n'existait pas avant il créé)
`sort 2> f3`	Trie les lignes tapées au clavier; écrit le résultat trié à l'écran. Les messages d'erreur éventuels sont écrits dans le fi
`sort <f1 >f2 2>f3`	Trie le lignes du fichier $f1$; écrits le résultat trié dans le fichier $f2$ et les messages d'erreur éventuels sont écrits dans $f3$.
`sort < f1 2» f3`	Trie le lignes du fichier $f1$; attache les messages d'erreur éventuels à la fin de $f3$.
`<f1 >f2 2>f3 sort`	Trie le lignes du fichier $f1$; écrits le résultat trié dans le fichier $f2$ et les messages d'erreur éventuels sont écrits dans $f3$.

Les tubes

Nous pouvons aussi rediriger la sortie d'un processus vers la l'entrée d'un autre processus. Pour cela nous utilisons le pipe (**le tube**).

Exemple: : compter le nombre de mots uniques dans une liste .

Solution 1:

```
sort -u <f1 >f2 #Trie l'entrée standard en laissant
                #une occurrence unique de chaque ligne

wc -l <f2       #word count : affiche sur la sortie
                #standard le nombre de lignes
                #dans l'entrée standard
```

Solution 2: sans création de fichier intermédiaire.

```
sort -u <f1 | wc -l
```

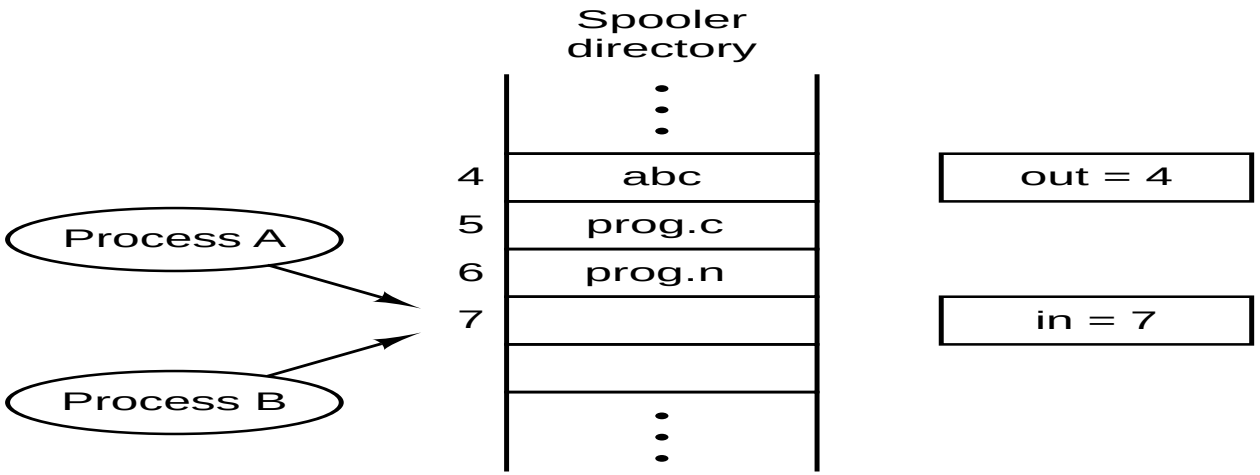
La communication inter-processus

- Les processus ont souvent besoin de communiquer entre eux (Exemple: pipeline - shell).
- Nécessité d'une communication entre processus de préférence de façon structurée, sans utiliser les interruptions.
- Le SE doit fournir aux processus coopératifs les moyens de communiquer entre eux par l'intermédiaire d'une fonction de communication inter processus.

IPC: *Interprocess communication*

Les problèmes de base de l'IPC

1. Comment un processus peut passer l'information à un autre?
2. Comment être sûr que les processus n'interviennent pas les uns avec les autres au moment d'exécuter des activités critiques?
   Exemple: 2 processus essaient de saisir les derniers $100K$ de mémoire
3. Comment assurer une séquence correcte quand il existe des dépendances ?
   Exemple: Si le processus A produit des données et le processus B les imprime, B doit attendre que A produise des données



Les conditions de concurrence

Race Conditions

Situation où plusieurs processus accèdent et manipulent en concurrence les mêmes données et où le résultat de l'exécution dépend de l'ordre dans lequel on accède aux instructions

- Comment éviter les **race conditions**?

  Interdire les processus de lire/écrire des données partagées au même moment.

- Nous avons besoin de **l'exclusion mutuelle (*mutual exclusion*)**

Problème de la section critique

- n processus $\{P_0, P_1, \ldots, P_{n-1}\}$ en concurrence pour utiliser des données partagées.
- **Section critique**: Segment de code dans lequel le processus peut accéder les données partagées, modifier des variables communes, mettre à jour une table, etc.
- Problème: Assurer que quand un processus exécute sa section critique, **aucun** autre processus n'est autorisé à exécuter la sienne.
- Structure du processus P_i:

repeat

Entry section

CRITICAL SECTION

Exit section

Reminder section

until false;

Conditions pour avoir une bonne solution

- **Exclusion mutuelle (*Mutual exclusion*):**
  Si le processus P_i exécute sa section critique, *aucun* autre processus peut exécuter la sienne.
- **Déroulement (*Progress*):**
  Un processus qui n'est pas dans la section critique ne peut pas bloquer un autre.
- **Attente limitée (*Bounded waiting*):**
  Aucun processus doit attendre indéfiniment.
- Aucune supposition concernant la vitesse relative de n processus.

Solutions

1. Via logiciel
2. Via matériel

Exclusion mutuelle avec attente active

Busy wait

Désactiver les interruptions

- Permettre à un processus de désactiver les interruptions pendant qu'une variable partagée est modifiée (cad, jusqu'après avoir entrer dans sa section critique).
- Avec la désactivation des interruptions, les interruptions de l'horloge ne sont pas possible et donc un processus peut garder le contrôle de la CPU.
- Mauvaise solution:
  - Pas très intelligent de donner au processus utilisateur le pouvoir de désactiver les interruptions.
  - Si le système est multiprocesseur, la désactivation des interruptions concerne seulement une CPU - les autres peuvent accéder la mémoire partagée.

Exclusion mutuelle avec attente active

Solution logicielle: Variables verrou (lock)

- Solutions pour 2 processus:
  Seulement 2 processus à la fois: P_0 et P_1
  Représentation: P_i
  Autre processus: P_j ($j = 1 - i$)
  Nous étudions 3 algorithmes
- Solution pour plusieurs processus:
  Algorithme du boulanger (*Bakery Algorithm*)

Algorithme 1

- Variables partagées (Shared variables):
  - `var turn: (0..1);` Initially `turn = 0`
  - `turn = i` P_i can enter its critical section .

Process P_i

repeat

while $turn \neq i$ do no-op;

critical section

turn := j;

remainder section

until false;

Satisfies mutual exclusion, but not progress.

Algorithme 2

- Variables partagées (Shared variables):
  - `var flag:  array [0..1] of boolean;`
    Initially $flag[0] = flag[1] = false$.
  - `flag[i] = true`
    P_i ready to enter its critical section

Process P_i

repeat

flag[i] := true;

while flag[j] do no-op;

critical section

flag[i] := false;

remainder section

until false;

Satisfies mutual exclusion, but not progress.

Algorithme 3

- Shared variables: Combined shared variables of algorithms 1 and 2.

Process P_i

repeat

$flag[i] := true$;

turn := j;

while ($flag[j]$ and $turn = j$) do no-op;

critical section

$flag[i] := false$;

remainder section

until false;

Meets all three requirements; solves the critical-section problem for two processes.

Algorithme du boulanger

Bakery Algorithm

- n processus
- Avant d'entrer dans la SC, un processus reçoit un numéro
- Celui qui a le plus petit numéro est celui qui va entrer dans sa SC

  Si P_i et P_j reçoivent le même numéro et si $i \leq j$
  alors P_i est servi d'abord
  sinon P_j est servi d'abord

  Noms des processus sont uniques et totalment ordonnés; l'algorithme est complètement déterministe.
- Le système de numérotation: ordre croissant

Bakery Algorithm

- Notation $<$: lexicographical order (ticket $\#$, process id $\#$).
- $(a, b) < (c, d)$ if $a < c$ or if $a = c$ and $b < d$.
- $max(a_0, ..., a_{n-1})$ is a number, k, such that $k \geq a_i$ for $i = 0, \ldots, n-1$
- Shared data

```
var choosing: array [0..n-1] of boolean;
    number: array [0..n-1] of integer;
```

- Data structures are initialized to false and 0, respectively

Bakery Algorithm

repeat

$choosing[i] := true;$

$number[i] := max(number[0], number[1], ..., number[n-1]) + 1;$

$choosing[i] := false;$

$for\ j := 0\ to\ n-1$

do begin

while choosing[j] do no-op;

while number[j] = 0

and (number[j],j) < (number[i], i) do no-op;

end;

critical section

number[i] := 0;

remainder section

until false;

Problème de l'attente active

- Gaspille le temps de CPU
- Peut avoir des résultats inattendus: inversion de la priorité
  Exemple:
  - 2 processus:
    H : priorité plus grande
    L: priorité basse
  - Scheduling: Si H est dans la file d'attente des processus prêts, alors H est choisi pour être exécuté
  - Situation:

Temps t_i: L est en exécution, L est dans la SC

Temps t_{i+j}: H arrive dans la fille des processus prêts

Donc... L est interrompu et H prend la CPU (H reste dans la boucle d'attente)

H ne sera jamais interrompu, L ne peut pas sortir de sa SC

Primitives IPC qui bloquent un processus

1. **Sleep**: appel système qui bloque un processus jusqu'à l'instant où il sera réveillé par un autre.
2. **Wakeup**: appel système pour réveiller un processus.

Exemple: Problème du producteur-consommateur

- Buffer plein et producteur veut ajouter un élément
  - Producteur va dormir
  - Reveillé quand le consommateur consomme
- Buffer vide et consommateur veut retirer un élément
  - Consommateur va dormir
  - Réveillé quand le producteur ajoute un élément
- Problème: *Race conditions*
  Producteur et consommateur peuvent dormir pour toujours

Producteur-consommateur

Producteur

```
repeat
        . . .
        if counter = n then Sleep();
        buffer[in] := nextp;
        in := (in + 1) mod n;
        counter := counter + 1
        if (counter = 1) then Wakeup(consumer);
until false
```

Consommateur

```
repeat
        if counter = 0 then Sleep();
        nextc := buffer[out];
        out := (out + 1) mod n;
        counter := counter - 1
        if (counter = n - 1) then Wakeup(producer);
        . . .
until false
```

Sémaphores

Un sémaphore S est une variable entière à laquelle (sauf pour l'initialisation) on accède seulement à travers deux opérations standards atomiques:

1. *Wait* (ou $DOWN$ ou P de *proberen*): correspond à $sleep$:
   - L'opération *DOWN* sur un sémaphore détermine si sa valeur est supérieure à 0.
   - Si c'est le cas, elle la décrémente et poursuit son activité.
   - Si la valeur est 0, le processus est placé en sommeil sans que le *DOWN* se termine.
2. *Signal* (ou UP ou V de *verhogen*): correspond à $wakeup$:
   - L'opération *UP* incrémente la valeur du sémaphore concerné.
   - Si un ou plusieurs processus dorment sur ce sémaphore, incapable de terminer une opération *DOWN* antérieure, l'un deux est choisi (aléatoirement) par le système et est autorisé à terminer son *DOWN*.
   - Une fois un *UP* accompli sur un sémaphore contenant des processus qui dorment, le sémaphore sera toujours à 0 mais il contiendra un processus en sommeil de moins.

Producteur-consommateur

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item( );                /* generate something to put in buffer */
          down(&empty);                         /* decrement empty count */
          down(&mutex);                         /* enter critical region */
          insert_item(item);                    /* put new item in buffer */
          up(&mutex);                           /* leave critical region */
          up(&full);                            /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                          /* decrement full count */
          down(&mutex);                         /* enter critical region */
          item = remove_item( );                /* take item from buffer */
          up(&mutex);                           /* leave critical region */
          up(&empty);                           /* increment count of empty slots */
          consume_item(item);                   /* do something with the item */
     }
}
```

L’échange de messages

- Méthode de IPC qui permet aux processus de communiquer sans avoir des variables partagées.
- Utilise 2 primitives: $send$ et $receive$
- Format:
  $send(destination, message)$
  $receive(source, message)$
- Un système de message doit traiter des problèmes de conception étrangers aux sémaphores et moniteurs - surtout si les processus sont dans des machine différentes. Exemple: message perdu

Producteur consommateur

producer

```
repeat
        produceItem (item);
        receive (consumer, message);
        buildMessage (message, item);
        send(consumer, message);
until false;
```

consummer

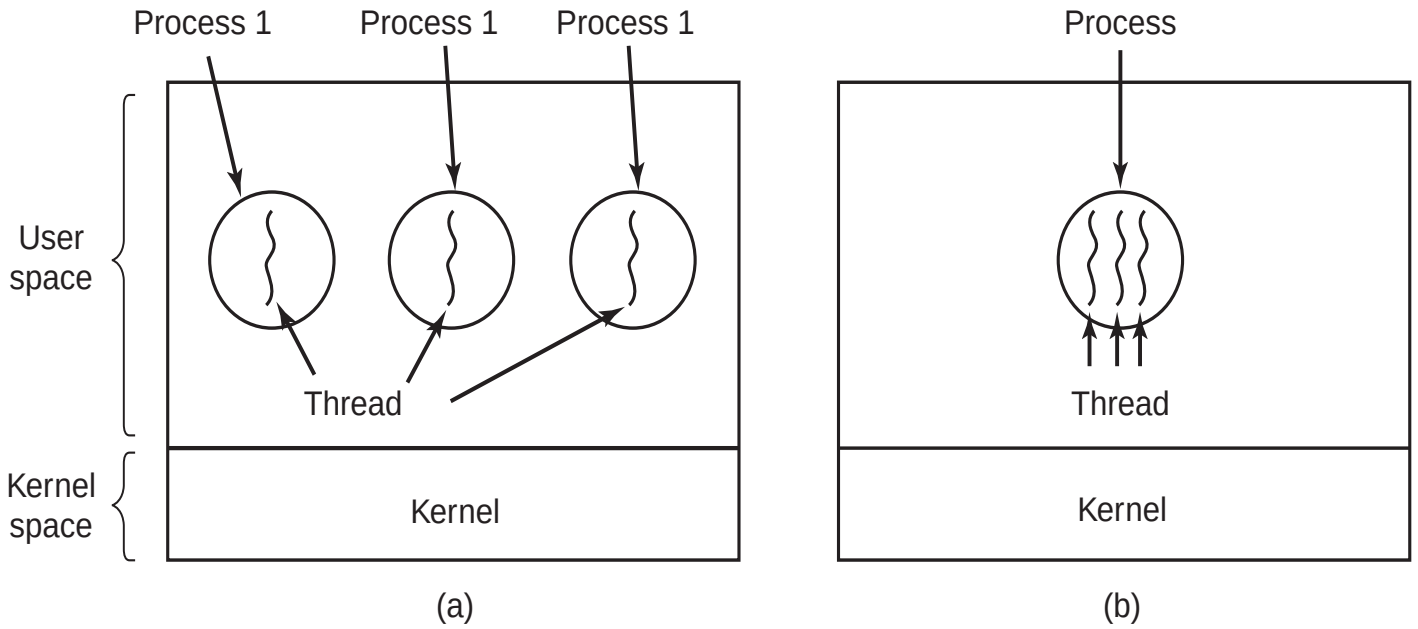
for $i = 1$ to N do send(producer, message)

```
repeat
        receive(producer, message);
        extractItem (message, item);
        send(producer, message);
        consumeItem(item);
until false;
```

Threads - Les processus légers

- Un thread contient:
  1. 1 compteur d'instructions
  2. 1 ensemble de registres
  3. 1 pile
- Un thread partage (avec des threads ressemblants)
  1. La section de code
  2. La section de données
  3. Les ressources du SE
- Processus traditionnels (lourds): *task* + *thread*
- *Threads:*
  1. Niveau système
  2. Niveau utilisateur

Threads - Les processus légers



---- LES SIGNAUX ----

Un signal est un message envoyé à un processus. Le corps du message est en fait très simple puisqu'il est constitué en tout et pour tout d'un entier indiquant le type du signal. Typiquement, l'arrivée d'un signal interrompt, plus ou moins brutalement l'exécution du processus qui le reçoit.

=> Pour la plupart des signaux, vous pourrez redéfinir la procédure de réponse par défaut en installant un gestionnaire. Toutefois certains signaux ne peuvent être déviés de leur signification première. Tout processus peut envoyer un signal à un autre processus, à partir du moment où il connaît son PID. Toutefois, il faut bien se rendre compte que la plupart des signaux sont émis par le système pour signaler tel ou tel évènement, par exemple, une segmentation violation.

Numéro	Nom	Description
1	SIGHUP	Instruction (HANG UP) - Fin de session
2	SIGINT	Interruption
3	SIGQUIT	Instruction (QUIT)
4	SIGILL	Instruction illégale
5	SIGTRAP	Trace trap
6	SIGABRT (ANSI)	Instruction (ABORT)
6	SIGIOT (BSD)	IOT Trap
7	SIGBUS	Bus error
8	SIGFPE	Floating-point exception - Exception arithmétique
9	SIGKILL	Instruction (KILL) - termine le processus immédiatement
10	SIGUSR1	Signal utilisateur 1
11	SIGSEGV	Violation de mémoire
12	SIGUSR2	Signal utilisateur 2
13	SIGPIPE	Broken PIPE - Erreur PIPE sans lecteur
14	SIGALRM	Alarme horloge
15	SIGTERM	Signal de terminaison
16	SIGSTKFLT	Stack Fault
17	SIGCHLD ou SIGCLD	modification du statut d'un processus fils
18	SIGCONT	Demande de reprise du processus
19	SIGSTOP	Demande de suspension imbloquable
20	SIGTSTP	Demande de suspension depuis le clavier
21	SIGTTIN	lecture terminal en arrière-plan
22	SIGTTOU	écriture terminal en arrière-plan
23	SIGURG	évènement urgent sur socket
24	SIGXCPU	temps maximum CPU écoulé
25	SIGXFSZ	taille maximale de fichier atteinte
26	SIGVTALRM	alarme horloge virtuelle
27	SIGPROF	Profiling alarm clock
28	SIGWINCH	changement de taille de fenêtre
29	SIGPOLL (System V)	occurence d'un évènement attendu
29	SIGIO (BSD)	I/O possible actuellement
30	SIGPWR	Power failure restart
31	SIGSYS	Erreur d'appel système
31	SIGUNUSED	

Pour trouver le PID d'un processus :

[nom] & (& sert à lancer le petit programme tout en gardant la main sur le terminal)

recule@recule:-$ xeyes &

[1] 14824

=> le PID est 14824 (**xeyes** est un petit programme sympa avec des yeux qui suivent les mouvements de la souris)

où

ps |grep « xeyes"

recule@recule:-$ ps | grep "xeyes"

14840 pts/0 00:00:00 xeyes

La commande kill

kill l (pour lister tous les signaux)

kill -19 (signal STOP)

Arrêtons xeyes avec son PID

recule@recule:-$ kill -19 14824
[1] Arrêté xeyes
recule@recule:-$

=> là le programme n'est pas fermé mais juste en sommeil (le yeux ne bougent plus).

Maintenant réveillons le :
(-18 SINGCONT) => cont = continue

recule@recule:-$ kill -18 14824
recule@recule:-$
=> le programme est réactivé (les yeux bougent à nouveau).

Maintenant pour l'arrêter (le tuer / kill) :
(-9 SIGKILL)

recule@recule:-$ kill -9 14824
recule@recule:-$
Processus arrêté xeyes

=> le programme est fermé.

Killall [nom du processus]
=> pour fermer tous les processus en lien avec firefox

recule@recule:-$ Killall firefox
recule@recule:-$

Commande ps -elf | grep “firefox »
=> pour connaitre tous les processus en lien avec firefox

recule@recule:-$ ps -elf | grep “firefox”
0 S recule@recule 15473 14580 0 80 0 - 3594 pipe_w 15:28 pts/0 00:00:00 grep –color=auto firefox
recule@recule:-$

=> ici, un seul processus en cours …

Note :
La commande **sleep**

recule@recule:-$ sleep 5
.
.
.
.
.
recule@recule:-$
=> Le système se met en attente durant 5 secondes, et ensuite nous reprenons la main.

TRAITEMENTS DES TACHES POUR GERER LES PRIORITES

=> relançons xeyes :
recule@recule:-$ xeyes &

[1] 14824

=> le PID est 14824 (**xeyes** est lancé)

Avec CTRL+Z on peut stopper le programme xeyes [sigstop] , une fois stopper on va pouvoir exécuter des actions.

Deux commandes :

fg (for frown) => passer un processus qui est stopper au premier plan)

bg (back grown)=> passer un processus en arrière-plan)

recule@recule:-$ fg xeyes

recule@recule:-$ bg xeyes

Commandes jobs :

=> pour voir les processus en train de tourner :

Lançons xeyes (avec & pour garder la main sur le terminal)

recule@recule:-$ xeyes &

[1] 5644

=> le PID est 5644

recule@recule:-$ jobs

[1]+ En cours d'exécution xeyes &

=> Ici xeyes est le seul processus à tourner.

ENVIRONNEMENT MUTLTI-UTILISATEURS

Ajouter un utilisateur hors interface graphique :

=> pour créer un nouvel utilisateur nommé mano :

recule@recule:-$ sudo adduser mano
[sudo] Mot de passe de recule :
Ajout de l'utilisateur « mano » …
Ajout du nouveau groupe « mano » (1002) …
Ajout du nouvel utilisateur « mano » (1002) avec le groupe « mano » …
Création du répertoire « /home/mano » …
Copie des fichiers depuis « /etc/sket » …
Entrez le nouveau mot de passe UNIX :
Retapez le nouveau mot de passe UNIX :
Passwd : le mot de passe a été mis à jour avec succès
Modification des informations relatives à l'utilisateur mano
Entrez la nouvelle valeur ou « Entrée » pour conserver la valeur proposée
    Nom complet [] : Mano Recule
    N° de bureau [] :
    Téléphone professionnel [] :
    Téléphone personnel [] :
    Autres [] :
Ces informations sont-elles correctes ? [O/n]
recule@recule:-$

=> pour se connecter au nouvel utilisateur :
recule@recule:-$ su mano
Mot de passe :
recule@mano:-$

pour changer le mot de passe de mano :
recule@mano:-$ sudo passwd mano

pour modifier les informations demandées lors de la création du nouvel utilisateur depuis recule :
recule@recule:-$ usermod mano
usermod : aucune option
Utilisation : usermod [options] LOGIN
Options :
(liste de commandes)

pour ajouter mano au groupe sudo (super utilisateur) depuis recule :
recule@recule:-$ sudo usermod mano -G sudo

Pour pouvoir ajouter un nouveau groupe :

recule@recule:-$ sudo addgroup Atlantis

Pour ajouter un utilisateur a un groupe :
(pour ajouter mano au groupe atlantis)
recule@recule:-$ usermod -a -G atlantis mano
où
recule@recule:-$ usermod -aG atlantis mano

Pour changer de groupe ou ajouter un utilisateur à un groupe :
recule@recule:-$ sudo adduser mano kheops
(=> nom du nouveau groupe)

Pour voir les identifiants utilisateurs et les identifiants de groupes :

Identifiant groupes
recule@recule:-$ id
uid=1000(recule) gid=1000(recule) groups=1000(recule), 4(adm), 24(cdrom), 27(sudo), 30 (dip), 46(plugdev), 121(lpadmin), 134(lxd), 135(sambashare)

uid => identifiant utilisateur
gid => identifiant groupe personnel de l'utilisateur
groups => indentifiant des groupes de l'utilisateur
adm =>
cdrom =>
sudo => groupe sudo de l'utilisateur
dip =>
plugdev =>
padmin =>
lxd =>
sambashare =>

Indentation utilisateur
recule@recule:-$ id recule
uid=1000(recule) gid=1000(recule) groups=1000(recule), 27(sudo)

Plusieurs sessions d'utilisateurs

Commandes **users**
=> pour savoir combien d'utilisateurs sont connectés sur le réseau de l'hôte
recule@recule:-$ users
recule mano

Commande **who**

who est une commande UNIX permettant d'afficher des informations concernant des utilisateurs qui sont connectés. La commande who est proche de la commande w , qui fournit le même type d'informations, mais w fournit des informations plus détaillées et des statistiques.

=> pour connaitre les utilisateurs qui se sont connecté sur une machine en particulier, étant l'hôte du réseau, il est possible de savoir quelles ont été toutes les sessions démarrées sur l'ordinateurs.

```
recule@recule:-$ who
recule tty7        2023-11-11 17:04  (:0)
```

commande **w** :

```
recule@recule:-$ w
20:59:18  up  55  min,     1  user,    load  average: 0,00,   0,05,   0,07
UTIL.    TTY     DE        LOGIN@ IDLE  JCPU   PCPU   QUOI
Recule  tty7     :0        17/04  54/50  54,28s  0.27s /usr/lib/gnome.sessionj/gnome-session=binary
```

=> permet de savoir ce que chaque utilisateur connecté au réseau est en train de faire, utilisation du CPU anormal etc …

Commande **tload**

```
recule@recule:-$ tload
```

=> tload affiche un graphe de la charge moyenne du système sur le terminal (tty) spécifié en argument (ou celui du processus tload si pas d'argument).

Options

L'option **-s** *échelle* permet de spécifier une échelle verticale ainsi, une petite valeur représente une grande échelle et vice versa.
L'option **-d** *délai* permet de configurer le délai, en secondes, entre les mises à jour du graphe.

```
recule@recule:-$ tload -s -d 2
```

=> avec une actualisation du graphe toute les 2 secondes :

PLANIFICATIONS DE TACHES

Planifier des tâches, c'est déterminer leur ordre d'exécution les unes par rapport aux autres. Il s'agit donc de repérer les antériorités de chacune d'elles. On distingue les tâches simples et les tâches complexes. Parfois, une tâche n'est précédée que d'une tâche antérieure (c'est-à-dire exécutée avant). Les outils de planification des tâches gèrent des files d'attente de tâches dans un ordre logique. Ils classent les tâches par ordre de priorité afin d'assurer rapidité et efficacité. La plupart des systèmes d'exploitation ont une sorte de fonctionnalité intégrée de planification des tâches. Les entreprises peuvent également acheter des logiciels supplémentaires pour gérer les tâches et les lots d'instructions correspondants.

Commandes :

date => pour connaitre l'heure et la date.

recule@recule:-$ date

dimanche 30 juillet 2023, 14 :35 :00 (UTC+0200)

Codes à connaitre :

%A ou %a : jour (lettres – complet ou abrégé)
%B ou %b : mois (lettres – complet ou abrégé)

%d : numéro du jour (01 à 31)
%m : numéro du mois (01 à 12)
%Y ou %y : numéro de l'année (ex : 2019 à 19)

%H : heure (00 à 23)
%M : minute (00 à 59)
%S : seconde (00 à 59)

Raccourcis :
%F <=> %Y - %m – %d
%T <=> %H :%M :%S

EX :
date ''+ [texte + code %]''

recule@recule:-$ date ''+Nous sommes %A''
nous sommes dimanche

=> Ainsi %A nous donnera le jour.

date ''+%D''

recule@recule:-$ date ''+%D''
07/30/23
=> l'ordre étant à l'américaine (mois+jour+année).

Date ''+%F %T''

recule@recule:-$ date ''+ %F %T''
2023-07-30 14 :55 :00

=> Ici nous avons modifié l'ordre (mois+jour+année) avec %F et donné l'heure avec %T

Date %Z (pour avoir le time zone) Fuseau horaire

recule@recule:-$ date ''+%Z''
CEST

Modification de la date / heure

Sudo date JJMMhhmmYYYY =>> (jour/mois/heure/minute/année)
recule@recule:-$ sudo date 073014552023
2023-07-30 14 :55 :00

Outil de planification de tâche :

(Installer at => sudo apt-get install at)

Commande at :

=> pour lancer une demande de tâche avec un déclenchement au moment choisi :

recule@recule:-$ at 18:48
Warning: commands will be executed using /bin/sh
at> mv /home/recule/daminus.txt /home/recule/dada.txt
at>
CTRL+D => pour sortir de l'invite at> sans que cela annule la tâche que j'ai planifiée

=> Ici nous avons demandé à déplacer le fichier daminus.txt vers la même destination (/home/recule) mais en le renommant dada.txt, l'opération est demandée à 18 :48.

=> nous pouvons suivre l'avancement et vérifier la bonne prise en charge de la commande avec atq :

recule@recule:-$ atq
5 Fri May 24 18 :48 :00 2023 a recule

=> le numéro de tâche est **5** suivis des informations de la tâche à accomplir.
=> après le temps imparti, la modification est effectuée.

=> pour lancer une demande de tâche avec un déclenchement au moment choisi et en y ajoutant une date ultérieure :
recule@recule:-$ at 18:48 07/31/2023
recule@recule:-$ at 18:48 today (pour une tâche ce jour)
recule@recule:-$ at 18:48 tomorrow (pour une tâche demain etc…).
recule@recule:-$ at 18:48 +6 months (pour une tâche dans 5 mois …).

=> pour annuler une tâche planifiée :

atrm (pour at remove) + [numéro de la tâche]

recule@recule:~$ atrm 5

=> pour planifier l'extinction de la machine :

shutdown + [option]

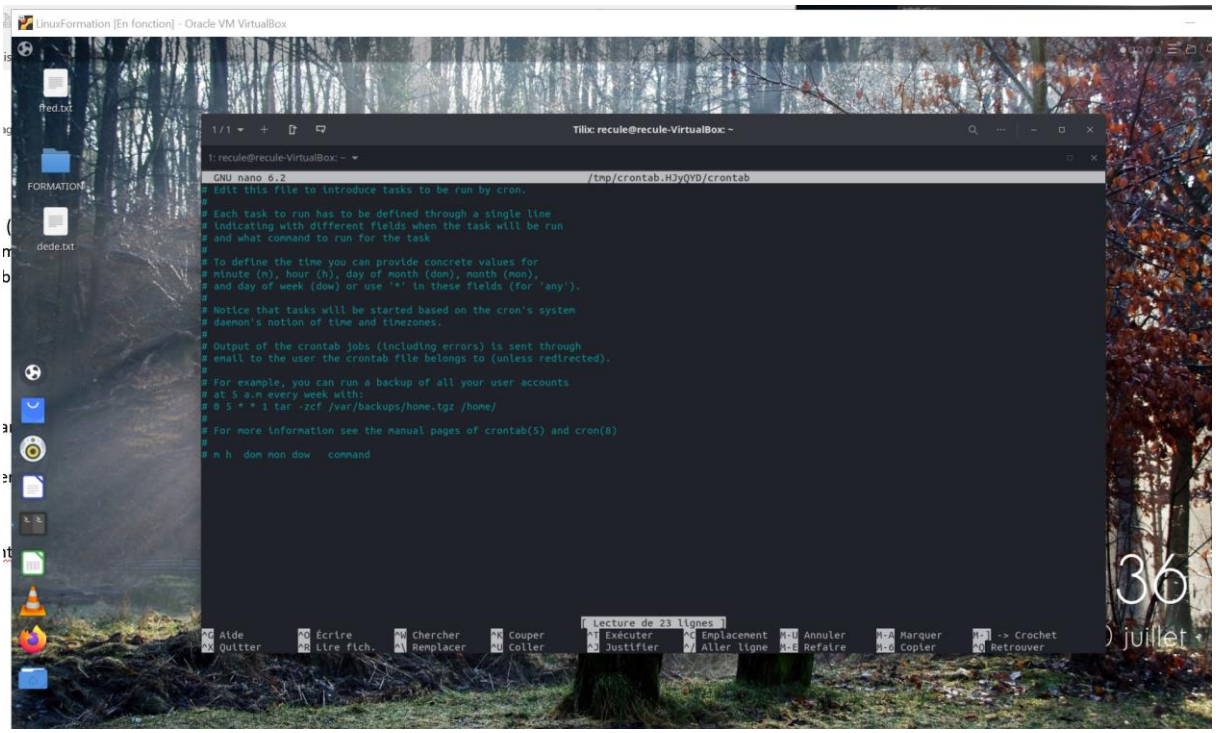
Options :
-H => mettre en veille (ou juste la commande **halt**)
-P => extinction de la machine (PowerOff).
-r => redémarrage (reboot)
où
halt -p => Power Off

commande **cron :**

pour utiliser la commande cron, on utilise le fichier cron :

Pour cela, on va utiliser la commande **crontab -e** pour ouvrir un sélecteur d'éditeur et choisir nano (réponse 1). Concrètement crontab va utiliser le programme **cron** pour créer le fichier **crontab** qui est destiné à la planification de la tâche.

recule@recule:~$ crontab -e



Rendons-nous à la dernière ligne du fichier :

m h dom mon dow command

=> la ligne commence par # (qui signifie que c'est une ligne qui est placée en tant que commentaire et qui n'est pas exécutée. Pour l'exécuter il faut, dans un premier temps enlever le #.

m h dom mon dow command

minute, heure, jour-du-mois (**day-of-month**), mois (**month**), jour-de-la-semaine (**day-of-week**)

ex : (une tâche que l'on voudrait exécuter tous les jours à 18 :00, par exemple une mise à jour journalière de la mise des dépôts (upgrade) et des logiciels (upgrade))

[en utilisant les commande apt-get update et apt-get upgrade] dont l'exécutable (fichier binaire) est placé dans **/usr/bin**.

=> 00 18 * * * /usr/bin/apt-get update && /usr/bin/apt-get upgrade

(notez que comme nous n'avons pas besoin du jour du mois etc … il suffit de mettre un * pour l'occulter).

Autre ex : (une tâche que l'on voudrait exécuter tous les dimanches à 18 :00, par exemple une mise à jour journalière de la mise des dépôts (upgrade) et des logiciels (upgrade))

=> 00 18 * * 0 /usr/bin/apt-get update && /usr/bin/apt-get upgrade

=> il faudra donc mettre un 0 à la place de dow (jour de la semaine) : notez que dans le langage informatique le premier de la semaine est le dimanche et que les jours vont de 0 à 6 (dimanche=0, lundi=1, mardi=2 etc..).

Autre ex : (une tâche que l'on voudrait exécuter tous les dimanches et mardi à 18 :00, par exemple une mise à jour journalière de la mise des dépôts (upgrade) et des logiciels (upgrade))

=> 00 18 * * 0,2 /usr/bin/apt-get update && /usr/bin/apt-get upgrade

=> il faudra donc mettre un 0 à la place de dow (jour de la semaine) : notez que dans le langage informatique le premier de la semaine est le dimanche et que les jours vont de 0 à 6 (dimanche=0, lundi=1, mardi=2 etc..).

où

=> 00 18 * * 0-2 /usr/bin/apt-get update && /usr/bin/apt-get upgrade
Pour que la tâche soit effectuée du dimanche au mardi (0-2)

Pour annuler une tâche planifiée dans un fichier crontab, il suffit d'utiliser la commande crontad -r (pour remove) et le fichier sera effacé.

```
recule@recule:~$ crontab -r
```

et crontab -l pour vérifier que la commande crontab est bien annulée :

```
recule@recule:~$ crontab -l
no crontab for recule
```

INFORMATIONS RESEAU

COMMANDE wget

wget est un programme en ligne de commande non interactif de téléchargement de fichiers depuis le Web. Il supporte les protocoles HTTP, HTTPS et FTP ainsi que le téléchargement au travers des proxies HTTP. Wget peut travailler en arrière-plan et ainsi vous permettre de lancer un téléchargement et de vous déconnecter du système ! Utile car il ne requiert pas d'action de l'utilisateur et vous permet d'effectuer ses tâches en arrière-plan, ce qui peut être très utile pour les téléchargements de données nombreuses et lourdes. Vous pouvez ainsi changer de session et laisser Wget finir le travail !

Ce logiciel libre permet le simple téléchargement d'un fichier mais aussi la recopie en local de tout ou partie d'un site qui sera par la suite consultable hors-ligne. Point fort appréciable, Wget vous permet de reprendre un téléchargement échoué suite à divers problèmes (connexions instables ou très lentes etc…). Les nombreuses options de Wget en font un outil de téléchargement très puissant !

recule@recule:-$ wget https://releases.ubuntu.com/22.04.2/ubuntu-22.04.2-desktop-amd64.iso

(le téléchargement commence ici même)

Au niveau du réseau :

=>le nom qui apparait dans le terminal est composé du nom d'utilisateur (recule) puis d'un autre nom précédé d'un @ (@recule-VirtualBox).

recule@recule-VirtualBox:-$

=> c'est ce nom (en rouge) qui va servir d'indentifiant sur un réseau, c'est derrière ce nom que ce trouve l'adresse IP de l'ordinateur. Ce 'hostname' (nom d'hôte) se retrouve dans un fichier situé ici => **/etc/hostname** , avec la commande **cat** on peut afficher son contenu et on peut également éditer ce fichier, ce qui nous permet d'éventuellement changer le nom d'hôte si besoin.
Notes : le simple fait de taper hostname sur le terminal nous donnera ce nom.

La commande **host**
=> cette commande nous permet de récupérer des informations sur un nom de domaine (appelé vulgairement une adresse de site).

```
recule@recule:-$ host google.fr
google.fr has address 172.217.19.227
google.fr has IPv6 address 2a00:1450:4007:80b::2003
google.fr mail is handled by 40 alt3.aspmx.l.google.com.
google.fr mail is handled by 30 alt2.aspmx.l.google.com.
google.fr mail is handled by 50 alt4.aspmx.l.google.com.
google.fr mail is handled by 20 alt1.aspmx.l.google.com.
google.fr mail is handled by 10 aspmx.l.google.com.
```

Elle nous donne l'IP associée au nom de domaine (ipv4 : 172.217.19.227).
Elle nous donne l'IP associée au nom de domaine (ipv6 : 2a00:1450:4007:80b::2003).

Différence entre ipv 4 et ipv6 :

La différence la plus évidente est que l'IPv4 utilise une adresse de 32 bits alors que l'IPv6 utilise une adresse de 128 bits. Cela signifie que l'IPv6 offre 1 028 fois plus d'adresses que l'IPv4, ce qui résout essentiellement le problème de la « pénurie d'adresses » (du moins dans un futur proche).

IPv6 est également une adresse alphanumérique séparée par des points, tandis qu'IPv4 est uniquement numérique et séparée par des points. Voici à nouveau un exemple de chacune :

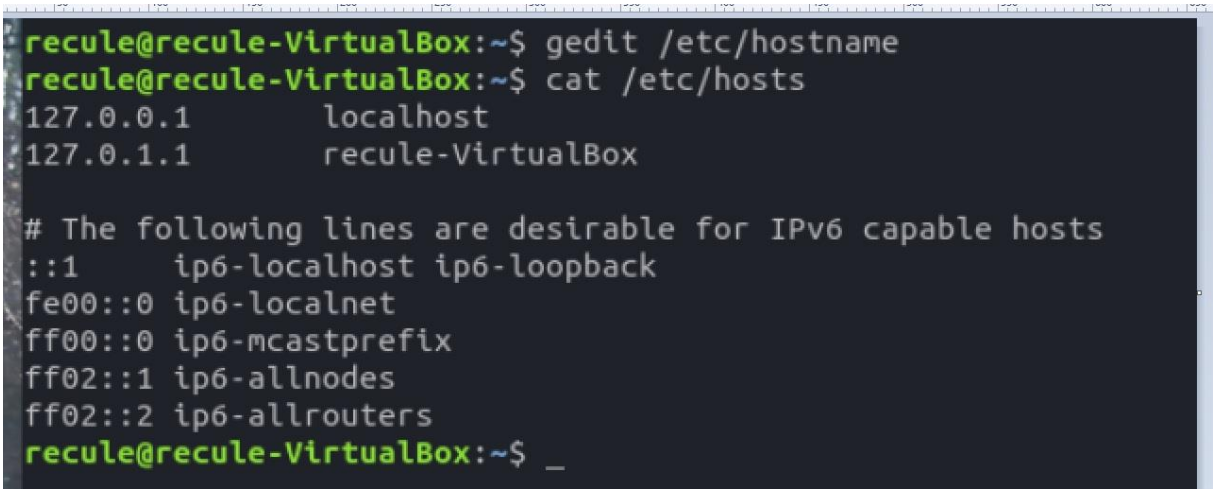
IPv4 – 192.168.10.150
IPv6 – 3002:0bd6:0000:0000:0000:ee00:0033:6778

=> Il existe également quelques différences techniques entre IPv4 et IPv6, bien que les non-développeurs n'aient pas vraiment besoin de les connaître.

Voici quelques-unes des différences techniques les plus notables :

- IPv6 inclut une qualité de service (QoS) intégrée.
- IPv6 possède une couche de sécurité réseau intégrée (IPsec).
- IPv6 élimine la traduction d'adresses réseau (NAT) et permet une connectivité de bout en bout au niveau de la couche IP.
- La multidiffusion fait partie des spécifications de base d'IPv6, alors qu'elle est facultative en IPv4. La multidiffusion permet la transmission d'un paquet vers plusieurs destinations en une seule opération.
- IPv6 a des en-têtes de paquets plus grands (environ deux fois plus grands qu'IPv4).

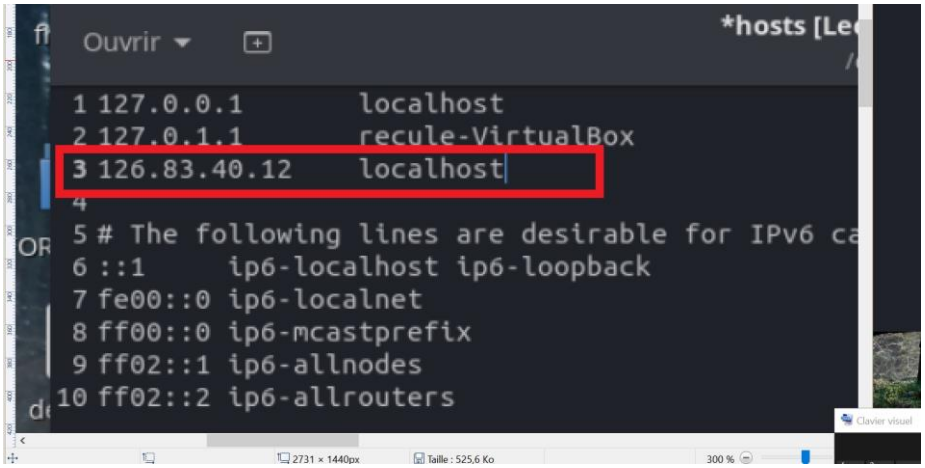
Entrons cette commande :
recule@recule:-$ cat /etc/hosts



127.0.0.1 est l'adresse IP du mon ordinateur en réseau local (localhost).
127.0.1.1 est l'adresse IP de l'ordinateur d'utilisateur.
=> Même si on désigne la même machine, on utilise le nom localhost quand on veut en toutes circonstances utiliser la boucle locale; et on utilise le nom "nomMachine" quand on veut désigner cette machine, quel que soit l'emplacement de celui qui utilise l'adresse.
=> Les adresse ipv6 qui suivent sont l'équivalent ipv6 de l'adresse ipv4 du localhost.

A savoir :

SI par exemple, on ne veut pas qu'une personne (ou un enfant) accède à un site en particulier, on va récupérer son nom de domaine (ex : google.fr) et avec récupérer son adresse IP avec host (comme nous l'avons fait plus haut). Ensuite nous pouvons ouvrir le fichier /etc/hosts et ajoute l'adresse IP à la suite des autres en la désignant comme local host, ainsi lorsqu'une personne tentera de se connecter sur ce site, elle ne pourra pas y accéder et sera directement redirigée en local.

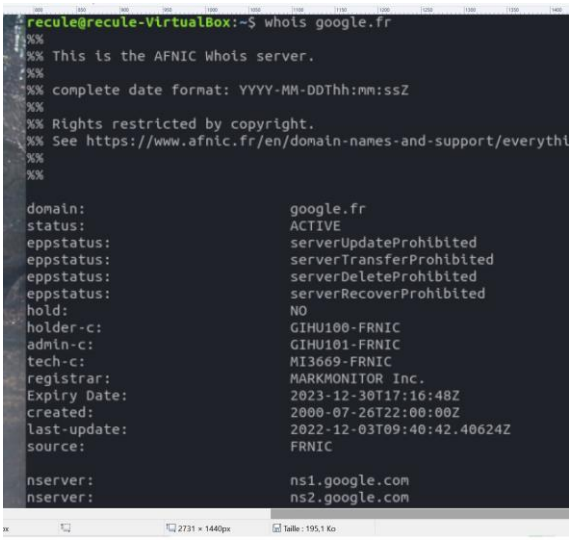


Commande **whois**

whois est utilisé pour obtenir les informations détaillées d'un domaine, par exemple les dates de contrat, les données du registrant, le DNS attribué. Tout cela est tiré d'une base de données mondiale, où ces informations sont stockées. En même temps Il permet à une IP de nous dire de quel pays il provient, des informations de l'opérateur Internet qui le gère et de nombreuses informations techniques supplémentaires.

=> Ainsi on peut récupérer de nombreuses informations concernant un nom de domaine.

recule@recule:-$ whois google.fr



C'est quoi un Resolveur DNS ?

Un résolveur DNS est un type de serveur qui gère la traduction "nom vers adresse", dans laquelle une adresse IP est mise en correspondance avec un nom de domaine et renvoyée à l'ordinateur qui l'a demandée. Les résolveurs DNS sont également connus sous le nom de résolveurs récursifs.

CONFIGURATION RESEAU

COMMANDE **ping**

Ping est l'une des commandes réseau les plus utilisées. C'est un utilitaire de commande Terminal / Shell utilisé comme un moyen simple de vérifier qu'une machine a accès à Internet et peut communiquer avec d'autres ordinateurs ou périphériques réseau.

Certaines des fonctions de la commande ping sont :

- Tester la connectivité réseau : réseau local, internet

- Dépannage de la carte d'interface réseau

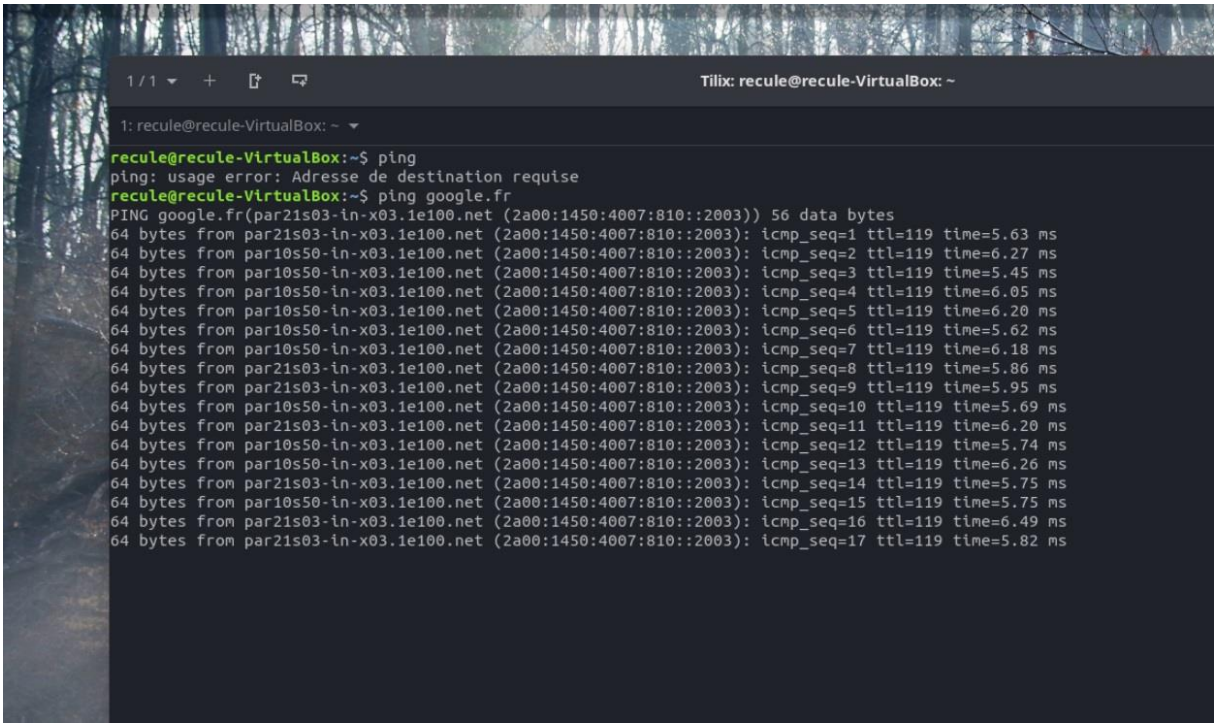
- Tester les problèmes de résolution de noms DNS

- La plupart des systèmes d'exploitation comme Windows, macOS et Linux utilise ping.

=> Peut-être avez-vous vu ping dans les conversations normales, par exemple, « Je vous ferai un ping une fois que j'aurai terminé le travail ». Dans le langage courant, ping signifie simplement un message envoyé d'un ordinateur à un autre.

reccule@reculeVirtualBox:-$ ping google.fr

=> si on utilise la commande ping avec le nom de domaine de google, on va tester la connectivité entre google et mon ordinateur, c'est-à-dire mon périphérique réseau.



Au démarrage de l’ordinateur :

Lorsque l’on démarre un ordinateur, INIT se lance.

C'est quoi init Linux ?

init (abréviation d’initialisation) est le premier programme informatique exécuté sur les systèmes d'exploitation basés Unix. Il est exécuté comme un daemon informatique et possède l'identifiant de processus (PID) 1. init est lancé par le noyau (kernel en anglais) et reste actif jusqu'à ce que le système soit éteint.

Ubuntu a mis en place, en 2017, un outil destiné à configurer le réseau avec une autre surcouche, Netplane :

=> Netplan permet de configurer à la fois NetworkManager (Ubuntu Desktop) et systemd-networkd (Ubuntu Server).

=> Netplan est un nouvel outil de configuration réseau qui utilise des fichiers de description yaml

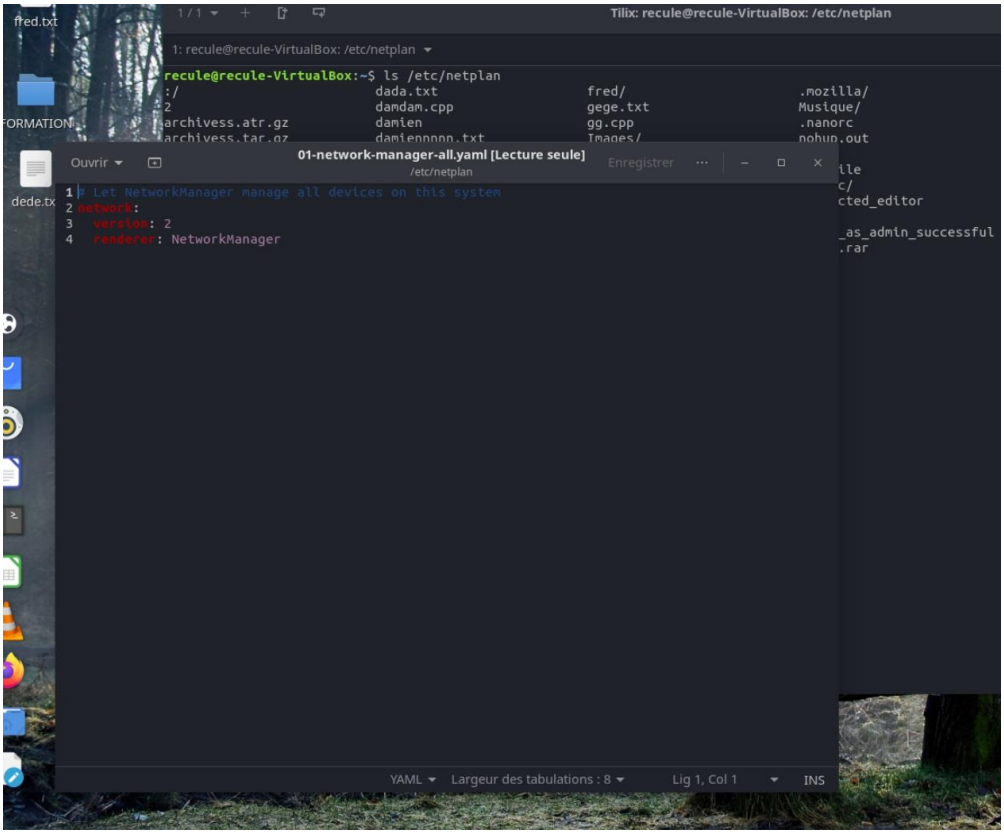
=> Rendons-nous dans le répertoire netplan pour ouvrir le fichier **.yaml**

reccule@reculeVirtualBox:-$ cd /etc/netplan

reccule@reculeVirtualBox:/etc/netplan/$ ls

01-network-manager-all.yaml

reccule@reculeVirtualBox:-$ cat 01-network-manager-all.yaml



=> Dans un fichier **.yaml**, ne pas utiliser de tabulations, il faut utiliser des espaces, sinon cela pose des problèmes (généralement en 2, 4, 8 espaces).

Le fichier :

```
# Let NetworkManager manage all devices on this system
network:
  version:  2
  renderer: NetworkManadger
```

---

* network => ce qui représente notre réseau

* renderer => ce qui va nous permettre de gérer, de manadger la configuration réseau., il nous informe que notre renderer est NetworkManadger. Il est l'outil que nous allons devoir utiliser.

Le DHCP :

La fonction Dynamic Host Configuration Protocol (DHCP) est un protocole client/serveur qui fournit automatiquement une adresse Internet Protocol (IP) et d'autres informations de configuration pertinentes à un hôte IP (par exemple, masque de sous-réseau et passerelle par défaut).

Ainsi, nous n'avons pas besoin de faire les réglages (adresse IP, masque de sous réseau etc …), DHCP s'occupe de tout.

Pourquoi utiliser DHCP ?

Chaque appareil sur un réseau TCP/IP doit avoir une adresse IP unique pour accéder au réseau et à ses ressources. Sans DHCP, les adresses IP des nouveaux ordinateurs ou ordinateurs qui sont déplacés d'un sous-réseau à un autre doit être configurées manuellement ; les adresses IP des ordinateurs supprimés du réseau doivent être récupérées manuellement.

Avec DHCP, l'ensemble de ce processus est automatisé et géré de manière centralisée. Le serveur DHCP gère un pool d'adresses IP et loue une adresse à n'importe quel client avec DHCP lorsqu'il démarre sur le réseau. Étant donné que les adresses IP sont dynamiques (louées) plutôt que statiques (affectées de manière permanente), les adresses qui ne sont plus utilisées sont automatiquement retournées au pool pour la réaffectation.

Exemple de modification du fichier :

```
# Let NetworkManager manage all devices on this system
network:
  version:  2
  renderer: NetworkManadger
```

> modification du fichier de base :

```
network:
   version: 2
   renderer: networkd
   ethernets:
     enp3s0:
       dhcp4: true
```

- changement de NetworkManadger pour networkd (nouvel outil très récent)

- ajout d'un système filaire (emp3s0)

- utilisation du DHCP4, c'est-à-dire une adresse ipv4 automatique (=> de type 192.128.0.1)

=> Or donc, ce fichier nous stipule que le système de rendu (pour la configuration réseau), c'est networkd, ensuite on configure une interface filaire qui utilisera DHCP en IPV4.

=> Une fois le fichier modifié avec notre nouvelle configuration, on va la générer et la tester avant de l'appliquer :

Générer => reccule@reculeVirtualBox:-$ netplan generate

Test => reccule@reculeVirtualBox:-$ netplan try

Appliquer => reccule@reculeVirtualBox:-$ netplan -d apply (avec l'option -d pour que le système nous dise s'il y a des problèmes éventuels sur le terminal.

Redémarrer le service qui s'occupe de tout ce qui est gestion réseau :

reccule@reculeVirtualBox:-$ systemctl restart system-networkd

=> le service redémarre et prend en compte votre nouvelle configuration. (puis vérifier avec un **ping)**

Admettons que l'on ait besoin de mettre les informations manuellement :

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp3s0:
      addresses:
       - 192.168.54.12/24
      gateway4: 192.168.10.4
      namaservers:
      search: [mydomain.com, mydomain.fr]
      addresses: [1,2,3,4,  1,2,2,4]
```

Explications:

network:

Version: 2

Renderer: NetworkManager/ networkd

ethernets:

[INTERFACE]:

Dhcp4: yes/no (en fonction de l'adresse IP dynamique ou statique)

Addresses: [ADRESSE_IP/NETMASK] (Adresse IP du périphérique dans la notation de préfixe. N'utilisez pas de masque de réseau)

Gateway: [PASSERELLE] (Adresse IP de la passerelle pour se connecter à un réseau extérieur

Nameservers)

Addresses: [SERVEUR_DNS_1, SERVEUR_DNS_2] (Adresse IP des serveurs de noms DNS)

ATTETION : un espace mal placé et le new config ne fonctionnera pas

Configuration WIFI type

```
network:
   version: 2
   renderer: networkd
   ethernets:
     enp3s0:
        addresses:
          - 192.168.54.12/24
        gateway4: 192.168.10.4
        namaservers:
        search: [mydomain.com, mydomain.fr]
        addresses: [1,2,3,4,  1,2,2,4]
  wifis:
    wlp2s0b2 :
      addresses : [192.168.54.12/24]
      gateway4 : [x.x.x.x]
      access-points:
        Livebox-D745:
           password: monjolimotdepass1234
```

=> pour récupérer les noms pour les interfaces wifis :

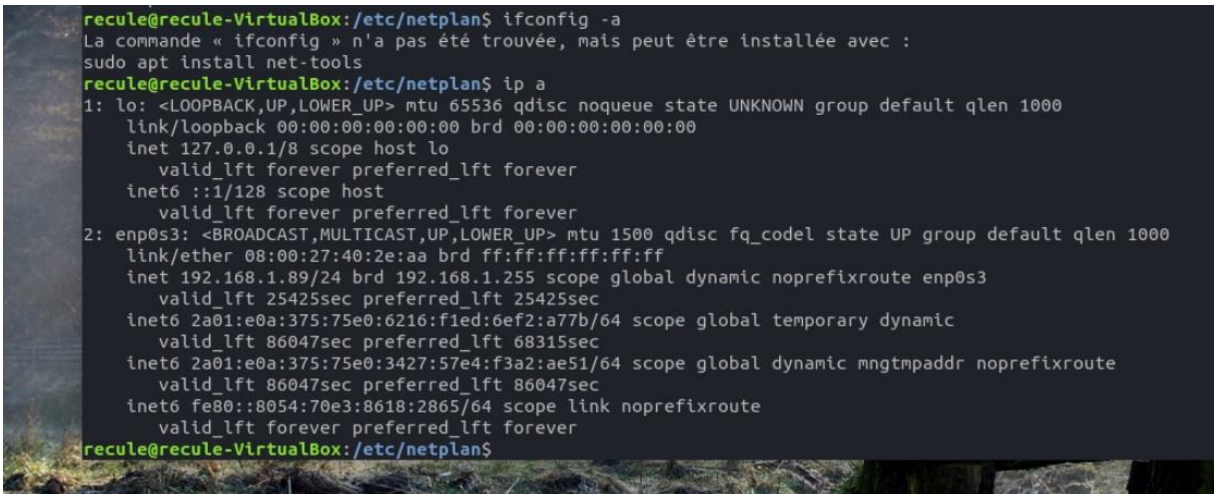
A l'ancienne :

reccule@reculeVirtualBox:-$ ifconfig -a

=> mais généralement ce paquet n'est plus installé sur les nouvelles distributions.

Commande à utiliser :

reccule@reculeVirtualBox:-$ ip a (ou **ip addr** => plus explicite d'information)



=> Pour couper ou remettre interface wifi :

reccule@reculeVirtualBox:-$ ip link wlp2s0b2 down (nom de wifi à trouver)

reccule@reculeVirtualBox:-$ ip link wlp2s0b2 up

=> Pour couper ou remettre interface filaire :

reccule@reculeVirtualBox:-$ ip link emp3s0 down (nom filaire à trouver)

reccule@reculeVirtualBox:-$ ip link emp3s0 up

Résolution DNS

La résolution DNS des noms de domaine, la résolution DNS des noms de domaine

L'homme a une très mauvaise mémoire des suites de chiffres. Or les ordinateurs et serveurs communiquent entre eux en s'identifiant via une adresse IP, suite de nombres ou mix de chiffres et nombres sensiblement complexe à mémoriser et différencier.

Pour aider les hommes dans leurs communications sur les réseaux, le Système de Noms de Domaine (DNS) a été inventé. Ce service est un annuaire géant de l'Internet, hiérarchisé et distribué au plan mondial, qui fait correspondre des noms de domaine avec des adresses IP.

Lorsqu'un internaute saisit un nom de domaine dans son navigateur, celui-ci interroge un serveur DNS qui va rechercher la réponse à cette adresse humainement compréhensible, le plus souvent une adresse IP, menant au bon site web, ordinateur ou réseau. On appelle ce processus d'interrogation la « résolution DNS ».

Découvrez dans cette fiche « 5 minutes pour comprendre », le fonctionnement de la résolution DNS.

En validant ce formulaire vous aurez accès au téléchargement de la fiche 5 minutes pour comprendre : La résolution DNS des noms de domaine.