Une nouvelle édition de ce document existe ici

# 2. Langage de commande Shell

Ce chapitre contient la définition du langage de commande Shell.

## 2.1 Introduction de Shell

Le shell est un interprète de langage de commande. Ce chapitre décrit la syntaxe de ce langage de commande tel qu'il est utilisé par *l'utilitaire sh et les* https://pubs.opengroup.org/onlinepubs/009695399/functions/popen.html fonctions système () et popen () définies dans le volume des interfaces système de l'IEEE Std 1003.1-2001.

La coque fonctionne selon l'aperçu général des opérations qui suit. Les détails spécifiques figurent dans les sections citées du présent chapitre.

1. Le shell lit son entrée à partir d'un fichier *(voir sh*), de l'option **-c** ou des https://pubs.opengroup.org/onlinepubs/009695399/functions/system.htmlhttps://pubs.opengroup.org/onlinepubs/009695399/functions/popen.html fonctions system () et popen () définies dans le volume des interfaces système de l'IEEE Std 1003.1-2001. Si la première ligne d'un fichier de commandes shell commence par les caractères « `# !` », les résultats ne sont pas précisés.
2. Le shell casse l'entrée en jetons : mots et opérateurs ; voir Reconnaissance des jetons.
3. Le shell analyse l'entrée en commandes simples (voir Commandes simples) et en commandes composées (voir Commandes composées).
4. Le shell effectue différentes extensions (séparément) sur différentes parties de chaque commande, ce qui donne une liste de noms de parcours et de champs à traiter comme une commande et des arguments ; voir Expansion des mots.
5. Le shell effectue la redirection (voir Redirection) et supprime les opérateurs de redirection et leurs opérandes de la liste des paramètres.
6. Le shell exécute une fonction (voir la commande Définition de la fonction), intégrée (voir Utilitaires spéciaux intégrés), fichier exécutable, ou script, donnant les noms des arguments comme paramètres de position numérotés de 1 à *n*, et le nom de la commande (ou dans le cas d'une fonction à l'intérieur d'un script, le nom du script) comme paramètre positionnel numéroté 0 (voir Recherche et exécution de commandes).
7. Le shell attend en option que la commande complète et recueille l'état de sortie (voir Statut de sortie pour les commandes).

## 2.2 Citation

La citation est utilisée pour supprimer la signification spéciale de certains caractères ou mots dans le shell. La citation peut être utilisée pour préserver le sens littéral des caractères spéciaux dans le paragraphe suivant, empêcher les mots réservés d'être reconnus comme tels, et empêcher l'expansion des paramètres et la substitution des commandes dans le traitement des documents ici (voir Ici-Document).

La demande doit mentionner les caractères suivants s'ils doivent se représenter :

```
|  &  ;  <  >  (  )  $  `  \  "  '  < espace >      < tab >  < newline >
```

et il peut être nécessaire de citer ce qui suit dans certaines circonstances. Autrement dit, ces caractères peuvent être spéciaux selon les conditions décrites ailleurs dans ce volume de l'IEEE Std 1003,1-2001 :

```
*   ?   [   #   ~   =   %
```

Les différents mécanismes de citations sont le caractère d'échappement, les guillemets simples et les guillemets doubles. Le présent document représente une autre forme de citation ; voir ici-document.

### *2.2.1 Caractère d'échappement (jeu de dos)*

Un jeu de dos qui n'est pas cité conserve la valeur littérale du caractère suivant, à l'exception d'une < newline > >. Si une < newline > suit le contre-temps, le shell doit l'interpréter comme une suite de ligne. Les jeux de fond et les « newline » doivent être supprimés avant de diviser l'entrée en jetons. La « newline » échappée étant retirée entièrement de l'entrée et n'étant remplacée par aucun espace blanc, elle ne peut servir de séparateur de jeton.

### *2.2.2 Guillemets uniques*

Les caractères annexés en guillemets uniques `(") conservent` la valeur littérale de chaque caractère dans les guillemets uniques. Une citation unique ne peut pas se produire dans une citation unique.

### *2.2.3 Doubles citations*

Les caractères annexés en guillemets doubles `(« ») conservent` la valeur littérale de tous les caractères compris dans les guillemets doubles, à l'exception du signe dollar, du backquote et du backslash, comme suit :

`$`

Le signe dollar doit conserver sa signification spéciale introduisant l'expansion des paramètres (voir Expansion des paramètres), une forme de substitution des commandes (voir Substitution des commandes), et l'expansion arithmétique (voir page1Expansion arithmétique).

Les caractères d'entrée dans la chaîne citée qui sont également inclus entre « `$ (` » et l'appariement `')'` ne doivent pas être affectés par les doubles-guillemets, mais plutôt définir cette commande dont la sortie remplace `le « $ (...) »` lorsque le mot est étendu. Les règles de tokénising dans la reconnaissance `de jetons, à` l'exclusion des substitutions d'alias dans la substitution `d'alias,` doivent être appliquées récursivement pour trouver l'appariement « ) ».

Dans la chaîne de caractères d'un « `$ { » enfermé à l'appariement '}',` un nombre pair de guillemets doubles ou de guillemets simples non échappés, le cas échéant, doit se produire. Un caractère rétrospectif précédent doit être utilisé pour échapper à un caractère littéral « { » ou « } ». La règle dans l'expansion des paramètres doit être utilisée pour déterminer la correspondance « } ».

`` ` ``

Le backquote doit conserver sa signification particulière introduisant l'autre forme de substitution de commande (voir Substitution de commande). La partie de la chaîne citée du backquote initial et les caractères jusqu'au backquote suivant qui n'est pas précédé d'un backslash, ayant supprimé les caractères d'échappement, définit cette commande dont la sortie remplace « `'...'` » lorsque le mot est étendu. L'un ou l'autre des cas suivants produit des résultats indéfinis :

- Une chaîne simple ou double-citée qui commence, mais ne se termine pas, dans la séquence « '... »
- Une `séquence` « '...' » qui commence, mais ne se termine pas, dans la même chaîne double-citée

`\`

Le backslash doit conserver sa signification particulière de caractère d'échappement (voir Escape Character (Backslash)) uniquement lorsqu'il est suivi d'un des caractères suivants lorsqu'il est considéré comme spécial :

```
$   «  »\< newline >
```

La demande doit s'assurer qu'un double devis est précédé d'un contre-temps à inclure dans le double devis. Le paramètre `'@'` a une signification particulière à l'intérieur des doubles-citations et est décrit dans les paramètres spéciaux.

## 2.3 Reconnaissance des jetons

Le shell doit lire son entrée en termes de lignes à partir d'un fichier, d'un terminal dans le cas d'un shell interactif, ou d'une chaîne dans le cas de *sh* **-c** ou *du système* (). Les lignes d'entrée peuvent être de longueur illimitée.

Ces lignes seront analysées selon deux modes majeurs : la reconnaissance de jeton ordinaire et le traitement des documents ici**.**

**Lorsqu'un** jeton io_here a été reconnu par la grammaire (voir Shell Grammar), une ou plusieurs des lignes suivantes immédiatement après le jeton suivant NEWLINE forment le corps d'un ou plusieurs documents ici et doivent être analysés conformément aux règles de Here-Document.

Lorsqu'il ne traite pas un **io_here**, le shell doit casser son entrée en jetons en appliquant la première règle applicable ci-dessous au caractère suivant dans son entrée. Le jeton doit être de la position courante dans l'entrée jusqu'à ce qu'un jeton soit délimité conformément à l'une des règles ci-dessous ; les caractères formant le jeton sont exactement ceux de l'entrée, y compris les caractères de citation. S'il est indiqué qu'un jeton est délimité et qu'aucun caractère n'a été inclus dans un jeton, le traitement se poursuit jusqu'à ce qu'un jeton réel soit délimité.

1. Si la fin de l'entrée est reconnue, le jeton courant doit être délimité. S'il n'y a pas de jeton courant, l'indicateur de fin d'entrée doit être retourné en tant que jeton.
2. Si le caractère précédent a été utilisé comme élément d'un opérateur et que le caractère courant n'est pas cité et peut être utilisé avec les caractères courants pour former un opérateur, il doit être utilisé comme élément de ce jeton (opérateur).
3. Si le caractère précédent a été utilisé comme élément d'un opérateur et que le caractère courant ne peut pas être utilisé avec les caractères courants pour former un opérateur, l'opérateur contenant le caractère précédent doit être délimité.
4. Si le caractère courant est rétrospectif, simple-citation ou double-citation `('\', ',' ou ') et` qu'il n'est pas cité, il affecte la citation des caractères suivants jusqu'à la fin du texte cité. Les règles de citation sont celles décrites dans Citation. Lors de la reconnaissance des jetons, aucune substitution ne doit être effectivement effectuée, et le jeton de résultat doit contenir exactement les caractères qui apparaissent dans l'entrée (à l'exception de l'adhésion < newline > >), non modifiés, y compris les guillemets ou opérateurs de substitution intégrés ou joints, entre le guillemet et la fin du texte cité. Le jeton ne doit pas être délimité par la fin du champ cité.
5. Si le caractère actuel est un `'$'` non coté ou « `'` », la coquille doit identifier le début de tous candidats à l'expansion de paramètre (l'Expansion de Paramètre), la substitution de commande (la Substitution de Commande), ou l'expansion arithmétique (l'Expansion Arithmétique) de leurs ordres de caractère non cotés préliminaires : '$' `ou` « $ { », « $ `( » ou »'` et « `$ (( »,` respectivement. Le shell doit lire suffisamment d'entrées pour déterminer la fin de l'unité à étendre (comme expliqué dans les sections citées). Lors du traitement des caractères, si des cas d'expansions ou de citations sont trouvés imbriqués dans la substitution, la coque doit les traiter récursivement de la manière spécifiée pour la construction trouvée. Les caractères trouvés depuis le début de la substitution jusqu'à sa fin, compte tenu de toute récursion nécessaire pour reconnaître les constructions incorporées, doivent être inclus sans modification dans le jeton de résultat, y compris les opérateurs ou citations de substitution incorporés ou fermants. Le jeton ne doit pas être délimité par la fin de la substitution.
6. Si le caractère courant n'est pas cité et peut être utilisé comme premier caractère d'un nouvel opérateur, le jeton courant (le cas échéant) doit être délimité. Le caractère courant doit être utilisé comme le début du jeton suivant (opérateur).
7. Si le caractère courant est un < newline > non cité, le jeton courant doit être délimité.
8. Si le caractère courant est un « blanc » non cité, tout jeton contenant le caractère précédent est délimité et le caractère courant doit être écarté.
9. Si le caractère précédent faisait partie d'un mot, le caractère courant doit être annexé à ce mot.
10. Si le caractère courant est un `'#',` il et tous les caractères suivants jusqu'à, mais en excluant, le < newline > suivant doivent être éliminés en tant que commentaire. La < newline > qui termine la ligne n'est pas considérée comme faisant partie du commentaire.
11. Le caractère actuel est utilisé comme le début d'un nouveau mot.

Une fois qu'un jeton est délimité, il est classé comme requis par la grammaire dans Shell Grammar.

### *2.3.1 Alias Substitution*

[En haut XSI] ⌦ Le traitement des alias doit être pris en charge sur tous les systèmes conformes à l'XSI ou si le système supporte l'option Utilitaires de portabilité utilisateur (et le reste de cette section n'est pas marqué davantage pour ces options ).⌫

Après qu'un jeton a été délimité, mais avant d'appliquer les règles grammaticales dans Shell Grammar, un mot résultant qui est identifié comme étant le mot de nom de commande d'une commande simple doit être examiné pour déterminer s'il s'agit d'un nom d'alias non cité et valide. Toutefois, les mots réservés dans un contexte grammatical correct ne doivent pas être des candidats à la substitution d'alias. Un nom d'alias valide (voir le volume Définitions de base de l'IEEE Std 1003.1-2001, section 3.10, Nom d'alias) https://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap03.html - tag_03_10 doit être celui qui a été défini par l'alias *utility* et qui n'a pas été défini par la suite en utilisant unalias. *Des implémentations* peuvent également fournir des alias valides prédéfinis qui sont en vigueur lorsque le shell est invoqué. Pour éviter que des boucles infinies ne s'écoulent en récursif, si la coque ne traite pas actuellement un alias du même nom, le mot doit être remplacé par la valeur de l'alias ; dans le cas contraire, il n'est pas remplacé.

Si la valeur de l'alias remplaçant le mot se termine par un « blanc », le shell doit vérifier le mot de commande suivant pour la substitution d'alias ; ce processus se poursuit jusqu'à ce qu'un mot qui n'est pas un alias valide ou qu'une valeur d'alias ne se termine pas par un « blanc ».

Lorsqu'elles sont utilisées comme spécifié dans le présent volume de la norme IEEE Std 1003.1-2001, les définitions d'alias ne doivent pas être héritées par des invocations distinctes de la coque ou par les environnements d'exécution d'utilité invoqués par la coque ; voir page1Environnement d'exécution Shell.

## 2.4 Mots réservés

Les mots réservés sont des mots qui ont une signification particulière pour la coquille ; voir Commandes Shell. Les mots suivants sont considérés comme réservés :

```
!       faire   esac  in
{       fait fi         alors
} élif pour jusqu'à
autre cas     si       pendant que
```

Cette reconnaissance n'intervient que lorsqu'aucun des caractères n'est cité et lorsque le mot est utilisé comme suit :

- Le premier mot d'une commande
- Le premier mot suivant l'un des mots réservés autres que **cas**, **pour**, ou **en**
- Le troisième mot dans une commande de cas (seulement **en** est valide dans ce cas)
- Le troisième mot dans une commande (seulement **in et do** sont valides dans ce cas)

Voir la grammaire dans Shell Grammar.

Les mots suivants peuvent être reconnus comme des mots réservés sur certaines implémentations (lorsqu'aucun des caractères n'est cité), entraînant des résultats non spécifiés :

**[[]] sélection de la fonction**

Les mots qui sont la concaténation d'un nom et d'un côlon (':') `sont` réservés ; leur utilisation produit des résultats non précisés.

## 2.5 Paramètres et variables

Un paramètre peut être désigné par un nom, un numéro ou un des caractères spéciaux énumérés dans les paramètres spéciaux. Une variable est un paramètre désigné par un nom.

Un paramètre est défini s'il a une valeur attribuée (null est une valeur valide). Une fois qu'une variable est définie, elle ne peut être désactivée qu'en utilisant la https://pubs.opengroup.org/onlinepubs/009695399/utilities/unset.htmlcommande spéciale de désactivation intégrée.

### *2.5.1 Paramètres de position*

Un paramètre de position est un paramètre noté par la valeur décimale représentée par un ou plusieurs chiffres, autre que le chiffre unique 0. Les chiffres indiquant les paramètres de position doivent toujours être interprétés comme une valeur décimale, même s'il y a un zéro avant. Lorsqu'un paramètre de position à plus d'un chiffre est spécifié, l'application doit inclure les chiffres dans les supports (voir Extension des paramètres). Les paramètres positionnels sont initialement attribués lorsque le shell est invoqué *(voir sh),* temporairement remplacés lorsqu'une fonction shell est invoquée (voir la commande Définition de la fonction), et peuvent être réaffectés avec la https://pubs.opengroup.org/onlinepubs/009695399/utilities/set.htmlcommande spéciale intégrée définie.

### *2.5.2 Paramètres spéciaux*

Les paramètres spéciaux et les valeurs auxquels ils doivent s'étendre sont énumérés ci-après. Seules les valeurs des paramètres spéciaux sont énumérées ; voir Expansion des mots pour un résumé détaillé de toutes les étapes de l'expansion des mots. `@`

S'étend aux paramètres positionnels, à partir d'un. Lorsque l'expansion se produit dans les guillemets doubles, et où le champ se divise (voir Division de champ) est effectuée, chaque paramètre de position doit se développer comme un champ séparé, avec la disposition que l'expansion du premier paramètre doit toujours être joint avec la partie de début du mot original (en supposant que le paramètre élargi a été intégré dans un mot), et l'extension du dernier paramètre doit toujours être jointe à la dernière partie du mot original. S'il n'y a pas de paramètres positionnels, l'expansion de `« @ » génère` des champs nuls, même lorsque `« @ » est` double-coté.

*

S'étend aux paramètres positionnels, à partir d'un. Lorsque l'expansion se produit à l'intérieur d'une chaîne double-citée (voir Double-citations), elle doit s'étendre à un seul champ avec la valeur de chaque paramètre séparé par le premier caractère de la *variable* IFS, ou par un espace si *IFS* est désactivé. Si *IFS* est réglé sur une chaîne nulle, cela n'équivaut pas à la désactiver ; son premier caractère n'existe pas, les valeurs des paramètres sont donc concaténées**.**

#

Augmente jusqu'au nombre décimal de paramètres positionnels. Le nom de la commande (paramètre 0) ne doit pas être compté dans le nombre donné par `'#'` parce qu'il s'agit d'un paramètre spécial et non d'un paramètre de position.

?

S'étend à l'état de sortie décimale du pipeline le plus récent (voir pipelines).

-

(Hyphen.) S'étend aux drapeaux d'options actuels (les noms d'options à une seule lettre concaténés en une chaîne) comme spécifié sur l'invocation, *par la* commande spéciale intégrée, ou implicitement par le shell.

$

S'étend à l'ID du processus décimal du shell invoqué. Dans un sous-coque (voir Environnement d'exécution de Shell), « `$` » doit s'étendre à la même valeur que celle du shell actuel.

!

S'étend à l'ID du processus décimal de la dernière commande de fond (voir Listes) exécutée à partir du shell courant. (Par exemple, les commandes de fond exécutées à partir de sous-coques n'affectent pas la valeur de « `$ !` » dans l'environnement shell actuel.) Pour un pipeline, l'ID du processus est celui de la dernière commande du pipeline.

0

(Zéro.) S'étend au nom du shell ou du script shell. Voir *sh* pour une description détaillée de la façon dont ce nom est dérivé.

Voir la description de la *variable* IFS dans Variables Shell.

### *2.5.3 Variables Shell*

Les variables doivent être initialisées à partir de l'environnement (tel que défini par le volume des définitions de base de l'IEEE Std 1003.1-2001, chapitre 8, Variables d'environnement et *fonction* exec dans le volume des interfaces système de l'IEEE Std 1003.1-2001) et peuvent recevoir de nouvelles valeurs avec des commandes d'affectation variables. Si une variable est initialisée à partir de l'environnement, elle doit être immédiatement marquée aux fins d'exportation ; voir l'exportation *spéciale intégrée*. De nouvelles variables peuvent être définies et initialisées avec des assignations variables, avec *les utilitaires* de lecture ou de getopts, avec le *paramètre* nom dans une boucle, avec l'expansion $ {*name* = *word*}, ou avec d'autres mécanismes prévus comme extensions d'implémentation.

Les variables suivantes influent sur l'exécution du réservoir :

*ENV*

[En haut XSI] ⌦ Le traitement de la variable ENV shell doit être pris en charge sur tous les systèmes conformes à l'XSI ou si le système supporte l'option User Portability Utilities .⌫

Cette variable, lorsque et seulement lorsqu'un shell interactif est invoqué, doit être soumise à une expansion de paramètre (voir l'expansion de paramètre) par le shell et la valeur résultante doit être utilisée comme un nom de chemin d'un fichier contenant des commandes shell à exécuter dans l'environnement courant. Le fichier n'a pas besoin d'être exécutable. Si la valeur augmentée *de ENV* n'est pas un chemin absolu, les résultats ne sont pas précisés. *ENV doit* être ignoré si les identifiants d'utilisateur réels et efficaces de l'utilisateur ou les identifiants de groupe réels et efficaces sont différents.

*Accueil*

Le nom de la page d'accueil de l'utilisateur. Le contenu de *HOME* est utilisé dans l'expansion de tilde (voir Expansion de Tilde).

*IFS*

(Séparateurs de champs d'entrée.) Une chaîne traitée comme une liste de caractères qui est utilisée pour diviser les champs et pour diviser les lignes en champs avec la *commande* de lecture. *Si l'IFS* n'est pas réglé, le shell doit se comporter comme si la valeur de l'IFS est < espace >, < onglet > et < newline > ; voir Division des champs. Les implémentations peuvent ignorer la valeur *de* l'IFS dans l'environnement au moment où le shell est invoqué, en traitant l'IFS comme s'il n'était pas défini.

*LANG*

Fournir une valeur par défaut pour les variables d'internationalisation qui sont non définies ou nulles. (Voir le volume des définitions de base de l'IEEE Std 1003.1-2001, section 8.2, Variables d'internationalisation pour la préséance des variables d'internationalisation utilisées pour déterminer les valeurs des catégories locales.)

*LC_ALL*

La valeur de cette variable dépasse les *variables* LC_ * et *LANG, telles que décrites* dans le volume des définitions de base de l'IEEE Std 1003.1-2001, chapitre 8, Variables environnementales.

*LC_COLLATE*

Déterminer le comportement des expressions de gamme, des classes d'équivalence et des éléments de collectage multi-caractères dans la correspondance de motifs.

*LC_CTYPE*

Déterminer l'interprétation des séquences d'octets de données textuelles en tant que caractères (par exemple, un octet par opposition aux caractères multi-octets), les caractères qui sont définis comme des lettres (classe de caractères **alpha) et < blank** > s (classe de caractères **blanc**), et le comportement des classes de caractères dans la correspondance de motifs. La modification *de la valeur de LC_CTYPE* après que le shell a commencé n'affecte pas le traitement lexical des commandes shell dans l'environnement d'exécution shell actuel ou ses sous-coques. Invoquer un script shell ou *exécuter exec sh soumet* https://pubs.opengroup.org/onlinepubs/009695399/utilities/sh.htmlle nouveau shell aux changements *de LC_CTYPE***.**

*LC_MESSAGES*

Déterminer la langue dans laquelle les messages doivent être écrits.

*LINENO*

Défini par le shell à un nombre décimal représentant le numéro de ligne séquentielle courant (numéroté à partir de 1) dans un script ou une fonction avant qu'il n'exécute chaque commande. Si l'utilisateur désactive *ou réinitialise LINENO, la* variable peut perdre sa signification particulière pour la durée de vie du shell. Si le shell n'exécute pas actuellement un script ou une fonction, la valeur *de LINENO n'est* pas précisée. Ce volume de l'IEEE Std 1003.1-2001 précise les effets de la variable uniquement pour les systèmes supportant l'option Services publics de portabilité des utilisateurs.

*NLSPATH*

[XSI] ⌦ Déterminer l'emplacement des catalogues de messages pour le traitement *des LC_MESSAGES* .⌫

*SENTIER*

Une chaîne formatée comme décrit dans le volume Définitions de base de l'IEEE Std 1003.1-2001, chapitre 8, Variables d'environnement, utilisée pour l'interprétation des commandes ; voir Recherche et exécution de commandes.

*PPID*

Défini par le shell à l'ID du processus décimal du processus qui a invoqué ce shell. Dans un sous-coque (voir Environnement d'exécution de Shell), *PPID* doit être réglé à la même valeur que celle du parent du shell courant. Par exemple, *echo $ PPID et* (*echo $* PPID*) produiraient* la même valeur. Ce volume de l'IEEE Std 1003.1-2001 précise les effets de la variable uniquement pour les systèmes supportant l'option Services publics de portabilité des utilisateurs.

*PS1*

Chaque fois qu'un shell interactif est prêt à lire une commande, la valeur de cette variable doit être soumise à l'expansion des paramètres et écrite à l'erreur standard. La valeur par défaut est `« $ »`. Pour les utilisateurs qui ont des privilèges spécifiques définis par l'implémentation, la valeur par défaut peut être une autre valeur définie par l'implémentation. Le shell doit remplacer chaque instance du caractère `« ! »` *dans PS1 par* le numéro d'historique de la commande suivante à taper. Échapper au « ! » avec `un autre « ! » (`c'est-à-dire, « ! ! ») `placera` le caractère littéral `« ! » dans l'invite`. Ce volume de l'IEEE Std 1003.1-2001 précise les effets de la variable uniquement pour les systèmes supportant l'option Services publics de portabilité des utilisateurs.

*PS2*

Chaque fois que l'utilisateur entre dans une « newline » avant de compléter une ligne de commande dans un shell interactif, la valeur de cette variable est soumise à l'expansion des paramètres et écrite à l'erreur standard. La valeur par défaut est `« > »`.
Ce volume de l'IEEE Std 1003.1-2001 précise les effets de la variable uniquement pour les systèmes supportant l'option Services publics de portabilité des utilisateurs.

*PS4*

Lorsqu'une trace d'exécution (set **-x)** est effectuée dans un shell interactif, avant chaque ligne de la trace d'exécution, la valeur de cette variable doit être soumise à l'expansion des paramètres et écrite à l'erreur standard. La valeur par défaut est `« + »`. Ce volume de l'IEEE Std 1003.1-2001 précise les effets de la variable uniquement pour les systèmes supportant l'option Services publics de portabilité des utilisateurs.

*PWD*

Défini par le shell comme un chemin absolu du répertoire de travail courant, ne contenant aucun composant de type lien symbolique, aucun composant qui sont des points, et aucun composant qui sont des points lorsque le shell est initialisé. Si une application fixe ou désactive la valeur de *PWD, les comportements* des https://pubs.opengroup.org/onlinepubs/009695399/utilities/cd.html*utilitaires* cd et pwd ne sont pas précisés.

## 2.6 Expansions de mots

Cette section décrit les différentes expansions qui sont effectuées sur les mots. Toutes les expansions ne sont pas effectuées sur chaque mot, comme expliqué dans les sections suivantes.

Les expansions de tilde, les expansions de paramètres, les substitutions de commande, les expansions arithmétiques et les suppressions de citation qui se produisent dans un seul mot s'étendent à un seul champ. C'est seulement le partage de champ ou l'expansion de pathname qui peut créer plusieurs champs à partir d'un seul mot. La seule exception à cette règle est l'extension du paramètre spécial `« @ » dans les` doubles-guillemets, comme décrit dans les paramètres spéciaux.

L'ordre d'expansion des mots est le suivant :

1. L'expansion de Tilde (voir Expansion de Tilde), l'expansion de paramètre (voir Expansion de paramètre), la substitution de commande (voir Substitution de commande) et l'expansion arithmétique (voir Expansion arithmétique) doivent être effectuées du début à la fin. Voir le point 5 dans Reconnaissance des jetons.
2. Le partage des champs (voir Division des champs) doit être effectué sur les parties des champs générées par l'étape 1, sauf si *IFS* est nul.
3. L'expansion du nom Pathname (voir Expansion du nom Pathname) doit être effectuée, à moins que *set -f ne soit* en vigueur.
4. La suppression de la citation (voir Suppression de la citation) doit toujours être effectuée en dernier.

Les expansions décrites dans cette section doivent se produire dans le même environnement de shell que celui dans lequel la commande est exécutée.

Si l'extension complète appropriée pour un mot aboutit à un champ vide, ce champ vide doit être supprimé de la liste des champs qui forment la commande complètement élargie, sauf si le mot original contenait des caractères de citation unique ou de double citation.

Le `caractère` '$' est utilisé pour introduire l'expansion des paramètres, la substitution des commandes ou l'évaluation arithmétique. Si un `'$'` non cité est suivi d'un caractère qui n'est pas numérique, du nom de l'un des paramètres spéciaux (voir Paramètres spéciaux), d'un premier caractère valide d'un nom de variable, d'un bracelet curly gauche ('{') ou d'une parenthèse gauche, le résultat n'est pas précisé.

### *2.6.1 Expansion d'inclinaison*

Un « préfixe-tilde » consiste en un caractère de tilde non cité au début d'un mot, suivi de tous les caractères précédant la première barre oblique non citée dans le mot, ou de tous les caractères dans le mot s'il n'y a pas de barre oblique. Dans une assignation (voir le volume Définitions de base de l'IEEE Std 1003.1-2001, section 4.21, Assignation variable), on peut utiliser plusieurs préfixes-tilde : au début du mot (c'est-à-dire suivant le signe égal de l'assignation), suivant tout côlon non cité, ou les deux. Un préfixe-tilde dans une affectation est terminé par le premier côlon non cité ou slash. Si aucun des caractères du préfixe-tilde n'est cité, les caractères du préfixe-tilde suivant le tilde sont traités comme un nom de connexion possible de la base de données utilisateur. Un nom de connexion portable ne peut pas contenir de caractères en dehors de l'ensemble donné dans la description de la variable d'environnement LOGNAME dans le volume Définitions de base de l'IEEE Std 1003.1-2001, section 8.3, Autres variables environnementales. Si le nom de connexion est nul (c'est-à-dire que le préfixe tilde ne contient que le préfixe tilde), le préfixe tilde est remplacé par la valeur de la variable *HOME. Si HOME* n'est pas défini, les résultats ne sont pas précisés. Dans le cas contraire, le préfixe-tilde doit être remplacé par un nom de parcours du répertoire de travail initial associé au nom de connexion obtenu en utilisant la *fonction* getpwnam () telle que définie dans le volume des interfaces système de l'IEEE Std 1003.1-2001. Si le système ne reconnaît pas le nom de connexion, les résultats ne sont pas définis.

### *2.6.2 Expansion des paramètres*

Le format pour l'expansion des paramètres est le suivant :

```
$ {expression}
```

où *l'expression* se compose de tous les caractères jusqu'à la correspondance « } ». Tout « } » échappé par un jeu de fond ou à l'intérieur d'une chaîne de caractères citée, ainsi que les caractères dans les expansions arithmétiques intégrées, les substitutions de commande et les expansions variables, ne doivent pas être examinés pour déterminer la correspondance « } ».

La forme la plus simple pour l'expansion des paramètres est :

```
$ {paramètre}
```

La valeur, *le cas échéant, du paramètre doit être* remplacée.

Le nom ou le symbole du paramètre peut être inclus dans des entretoises, qui sont facultatives sauf pour les paramètres de position à plus d'un chiffre ou lorsque *le paramètre* est suivi d'un caractère qui pourrait être interprété comme faisant partie du nom.
L'entretoise de fermeture correspondante doit être déterminée en comptant les niveaux d'entretoise, en sautant sur les chaînes citées fermées et en remplaçant les commandes.

Si le nom de paramètre ou le symbole n'est pas inclus dans les entretoises, l'extension doit utiliser le nom valide le plus long (voir le volume des définitions de base de l'IEEE Std 1003.1-2001, section 3.230, Nom), que le symbole représenté par ce nom existe ou non.

Si une expansion de paramètre se produit à l'intérieur de doubles-citations :

- L'expansion du nom Pathname ne doit pas être effectuée sur les résultats de l'expansion.
- Le fractionnement des champs ne doit pas être effectué sur les résultats de l'expansion, à l'exception de « @ » ; voir Paramètres spéciaux.

De plus, une expansion de paramètre peut être modifiée en utilisant l'un des formats suivants. Dans chaque cas où une valeur de *mot* est nécessaire (basée sur l'état du *paramètre*, comme décrit ci-dessous*), mot doit être* soumis à l'expansion de tilde, l'expansion de paramètre, la substitution de commande, et l'expansion arithmétique. Si *le mot n'est pas nécessaire*, il ne doit pas être élargi. Le caractère « } » qui délimite les modifications suivantes de l'expansion des paramètres doit être déterminé comme décrit précédemment dans la présente section et dans les doubles-citations. (Par exemple, $ {**foo-bar} xyz}** entraînerait l'expansion **de foo suivie** de la chaîne **xyz} si foo** est défini, sinon la chaîne « barxyz} »).

$ *{paramètre* : *-word}*
: **Utilisez les valeurs par défaut.** Si *le paramètre* est unset ou null, l'expansion du *mot doit* être substituée ; sinon, la valeur du *paramètre* doit être substituée.

$ *{paramètre* : = *word*}
: **Assigner des valeurs par défaut**. Si *le paramètre* est unset ou null, l'expansion du mot doit être affectée au *paramètre*. Dans tous les cas, la valeur finale du *paramètre* est remplacée. Seules les variables, et non les paramètres de position ou les paramètres **spéciaux,** peuvent être attribuées de cette façon.

$ *{paramètre* :**?** *[word]}*
: **Indiquer Erreur si Null ou Unset.** Si *le paramètre est* unset ou null, l'expansion du *mot (*ou un message indiquant qu'il est unset si le mot est omis) doit être écrite à l'erreur standard et le shell sort avec un statut de sortie non nul.
Dans le cas contraire, la valeur du *paramètre* doit être remplacée. Un shell interactif n'a pas besoin de sortir.

$ *{paramètre* : + *word*}
: **Utilisez la valeur alternative**. Si *le paramètre* **est non réglé ou** nul, nul doit être remplacé ; sinon, l'expansion du mot sera remplacée.

Dans les expansions de paramètres indiquées précédemment, l'utilisation du côlon dans le format donne lieu à un test pour un paramètre non réglé ou nul ; l'omission du côlon doit entraîner un test pour un paramètre qui n'est que non réglé. Le tableau suivant résume l'effet du côlon :

| | ***paramètre*** | ***paramètre*** | ***paramètre*** |
|---|---|---|---|
| | **Set and Not Null** | **Set But Null** | **Non jeu** |
| **$** *{paramètre* **:** *-word}* | *paramètre de remplacement* | remplacer *le mot* | remplacer *le mot* |
| **$** *{paramètre-word}* | *paramètre de remplacement* | remplacer null | remplacer *le mot* |
| **$** *{paramètre* **:** = *word***}** | *paramètre de remplacement* | *attribuer un mot* | *attribuer un mot* |
| **$** *{paramètre* = *word}* | *paramètre de remplacement* | remplacer null | *attribuer un mot* |
| **$** *{paramètre* **:***? word}* | *paramètre de remplacement* | erreur, sortie | erreur, sortie |
| **$** *{paramètre ? word}* | *paramètre de remplacement* | remplacer null | erreur, sortie |
| **$** *{paramètre* **:** + *word***}** | remplacer *le mot* | remplacer null | remplacer null |
| **$** *{paramètre* + *mot}* | remplacer *le mot* | remplacer *le mot* | remplacer null |

Dans tous les cas indiqués par « remplacer », l'expression est remplacée par la valeur indiquée. Dans tous les cas affichés avec « assigner *», le paramètre* est affecté de cette valeur, qui remplace également l'expression.

$ *{# paramètre}*
: **Longueur de chaîne**. La longueur en caractères de la valeur du paramètre *doit* être remplacée. Si *le paramètre est* **« * »** ou **« @ »,** le résultat de l'expansion n'est pas précisé.

Les quatre variétés suivantes d'expansion des paramètres prévoient un traitement de sous-traçage. Dans chaque cas, la notation correspondant au motif (voir Notation correspondant au motif), plutôt que la notation d'expression régulière, doit être utilisée pour évaluer les motifs. Si *le paramètre est* « * » ou « @ », le résultat de l'expansion n'est pas précisé. L'inclusion de la chaîne complète d'expansion des paramètres dans des doubles-citations ne doit pas entraîner la citation des quatre variétés suivantes de caractères de motif, alors que la citation de caractères dans les entretoises doit avoir cet effet**.**

$ *{paramètre % word}*
: **Supprimer le plus petit motif suffixe**. Le *mot* doit être élargi pour produire un motif. L'expansion du paramètre donne alors le *paramètre,* la plus petite partie du suffixe étant appariée au motif *supprimé*. **$ {paramètre** *%% mot}*
: **Supprimer le plus grand motif de suffixe**. Le *mot* doit être élargi pour produire un motif. L'expansion du paramètre se traduit alors par un *paramètre, la* plus grande partie du suffixe étant appariée au motif supprimé. *$ {paramètre # word}*
: **Supprimer le plus petit motif de préfixe**. Le *mot* doit être élargi pour produire un motif. L'expansion du paramètre se traduit alors par un *paramètre, la* plus petite partie du préfixe étant appariée au motif supprimé.

$ {paramètre # # *word*}
: **Supprimer le plus grand motif de préfixe**. Le *mot* doit être élargi pour produire un motif. L'expansion du paramètre se traduit alors par un *paramètre, la* plus grande partie du préfixe étant appariée au motif supprimé.

---

*Les sections suivantes sont instructives.*

**Exemples**

$ *{paramètre* : *-word}*

Dans cet exemple, *ls* n'est exécuté que si *x* est nul ou non défini. (La https://pubs.opengroup.org/onlinepubs/009695399/utilities/ls.htmlnotation $ (ls) de substitution de commande est expliquée dans Substitution de commande.)

```
{x $ : - $ (ls)}
```

$ *{paramètre* : = *word*}

```
unset X
echo $ {X : = abc}
ABC
```

$ *{paramètre* :*? word}*

```
non réglé posix
echo $ {posix :?}
sh : posix : paramètre nul ou non défini
```

$ *{paramètre* : + *word*}

```
ensemble a b c
echo $ {3 : + posix}
posix
```

$ *{# paramètre}*

```
HOME =/usr/posix
echo $ {# HOME}
10
```

$ *{paramètre % word}*

```
x = file.c
echo $ {x % .c} .O
file.o
```

$ *{paramètre %% mot}*

```
x = posix/src/std
echo $ {x % %/*}
posix
```

$ *{paramètre # word}*

```
x = $ HOME/src/cmd
echo $ {x # $ HOME}
/ src/cmd
```

$ *{paramètre* # # word}

```
x =/un/deux/trois
echo $ {x # # */}
```

La double citation des motifs est différente selon l'endroit où les doubles citations sont placées :

« $ {x # *} »
: L'astérisque est un caractère de motif.

{X # « * » $}
: L'astérisque littéral est cité et non spécial.

*Fin du texte informatif.*

---

### *2.6.3 Substitution de commandement*

La substitution de commande permet de substituer la sortie d'une commande à la place du nom de commande lui-même. La substitution de commande doit avoir lieu lorsque la commande est jointe comme suit :

```
$ (commande)
```

ou (version rétrocédée) :

Le shell doit étendre la substitution de commande en exécutant la *commande dans* un environnement de sous-coque (voir Environnement d'exécution de Shell) et en remplaçant la substitution de commande (le *texte de la*

*commande* plus le « `$ () » ou backquotes` d'encombrement) par la sortie standard de la commande, en supprimant les séquences d'une ou plusieurs « newline » s à la fin de la substitution. Les « newline » embarquées avant la fin de la sortie ne doivent pas être supprimées ; toutefois, ils peuvent être traités comme des délimiteurs de champs et éliminés lors de la séparation de champs, selon la *valeur de l'IFS et* la citation qui est en vigueur**.**

La recherche du backquote correspondant doit être satisfaite par le premier backquote trouvé sans un backslash précédent ; au cours de cette recherche, si un backquote non échappé est rencontré dans un commentaire shell, un ici-document, une substitution de commande embarquée du formulaire $ (commande), ou une chaîne citée, résultats indéfinis se produisent. Une chaîne de guillemets simple ou double qui commence, mais ne se termine pas, dans la séquence « '...' » produit des résultats indéfinis.

Avec le formulaire $ (commande), tous les caractères suivant la parenthèse ouverte à la parenthèse de fermeture correspondante constituent la *commande. Tout* script shell valide peut être utilisé pour la *commande,* sauf un script composé uniquement de redirections qui produit des résultats non spécifiés.

Les résultats de la substitution de commande ne doivent pas être traités pour une nouvelle expansion de tilde, une expansion de paramètre, une substitution de commande ou une expansion arithmétique. Si une substitution de commande se produit à l'intérieur de doubles guillemets, le découpage de champ et l'expansion de pathname ne doivent pas être effectués sur les résultats de la substitution.

La substitution de commande peut être imbriquée. Pour spécifier l'emboîtement dans la version rétrocédée, l'application doit précéder les backquotes internes avec des backslashs, par exemple :

Si la substitution de commande consiste en une seule sous-coque, telle que :

```
$ ((commande))
```

une demande conforme doit séparer les « `$ ( » et « ( »)` en deux jetons (c'est-à-dire les séparer avec de l'espace blanc). Ceci est nécessaire pour éviter toute ambiguïté avec l'expansion arithmétique.

### *2.6.4 Expansion arithmétique*

L'expansion arithmétique fournit un mécanisme pour évaluer une expression arithmétique et substituer sa valeur. Le format de l'expansion arithmétique est le suivant :

```
$ ((expression))
```

L'expression doit être traitée comme si elle était en double-citations, sauf qu'une double-citation à l'intérieur de l'expression n'est pas traitée spécialement. Le shell doit étendre tous les jetons dans l'expression pour l'expansion des paramètres, la substitution des commandes et la suppression des citations.

Ensuite, la coquille doit traiter cela comme une expression arithmétique et substituer la valeur de l'expression. L'expression arithmétique est traitée selon les règles indiquées *dans Précision arithmétique et Opérations, avec* les exceptions suivantes :

- Seule l'arithmétique des longs entiers signés est requise.
- Seules les constantes décimales, octales et hexadécimales spécifiées dans la norme ISO C, section 6.4.4.1, doivent être reconnues comme constantes.
- *L'opérateur de taille* () et les opérateurs préfixe et postfix « + + » et « -- » ne sont pas requis**.**
- Les énoncés de sélection, d'itération et de saut ne sont pas pris en charge.

Tous les changements de variables dans une expression arithmétique doivent être en vigueur après l'expansion arithmétique, comme dans l'expansion du paramètre « `$ {x = valeur} ».`

Si la variable shell *x* contient une valeur qui forme une constante entière valide, alors les expansions arithmétiques « `$ (x)` » et « `$ ($ x`) » retourneront la même valeur.

En guise d'extension, la coquille peut reconnaître des expressions arithmétiques au-delà de celles énumérées. Le shell peut utiliser un type entier signé avec un rang plus grand que le rang de **long signé. L'obus** peut utiliser un type flottant réel au lieu **de signé tant** qu'il n'affecte pas les résultats dans les cas où il n'y a pas de débordement. Si l'expression est invalide, l'expansion échoue et le shell doit écrire un message à erreur standard indiquant la défaillance.

---

*Les sections suivantes sont instructives.*

Un `exemple simple utilisant l'expansion arithmétique :`

```
# répéter une commande 100
fois x = 100
alors que [$ x -
gt 0] do
    commandx = $ (($ x-1))
```

*Fin du texte informatif.*

---

### *2.6.5 Découpage des champs*

Après l'expansion des paramètres (Expansion des paramètres), la substitution des commandes (Substitution des commandes) et l'expansion arithmétique (Expansion arithmétique), page1le shell doit analyser les résultats des expansions et des substitutions qui n'ont pas eu lieu dans les doubles-citations pour le partage des champs et plusieurs champs peuvent en résulter.

Le shell doit traiter chaque caractère de l'IFS comme un délimiteur et utiliser les délimiteurs pour diviser les résultats de l'expansion des paramètres et de la substitution des commandes en champs.

1. Si la valeur de *l'IFS* est un « espace », un « onglet » et un « newline », ou si elle n'est pas réglée, toute séquence de « espace », de « onglet » ou de « newline » au début ou à la fin de l'entrée doit être ignorée et toute séquence de ces caractères à l'intérieur de l'entrée doit délimiter un champ. Par exemple, l'entrée :

   `< newline > < space > < tab > foo < tab >` < tab > bar < space >

   donne deux champs, **foo** et **bar**.
2. Si la valeur de *IFS* est nulle, aucun partage de champ ne doit être effectué.
3. Dans le cas contraire, les règles suivantes doivent être appliquées successivement. Par « espace blanc IFS », on entend toute séquence (zéro ou plus) de caractères d'espace blanc qui sont dans la *valeur* IFS (par exemple, si *IFS contient* < space >/< comma >/< tab >, toute séquence de < space > s et < tab > s est considérée comme de l'espace blanc IFS).
   a. doit être ignoré au début et à la fin de l'entrée.
   b. Chaque occurrence dans l'entrée d'un *caractère* IFS qui *n'est pas* un espace blanc IFS, ainsi que tout espace blanc IFS adjacent, doit délimiter un champ, comme décrit précédemment.
   c. Un espace blanc IFS de longueur non nulle doit délimiter un champ.

### *2.6.6 Expansion du patronyme*

Après la division du champ, si *set* **-f** n'est pas en vigueur, chaque champ de la ligne de commande résultante doit être étendu à l'aide de l'algorithme décrit dans Pattern Matching Notation, qualifié par les règles dans Patterns Used for Filename Expansion.

### *2.6.7 Suppression de la citation*

Les caractères de la citation : `'\', ','` et '(backslash, simple-citation, double-citation) qui étaient présents dans le mot original doivent être supprimés à moins qu'ils n'aient été eux-mêmes cités.

## 2.7 Rédirection

Redirection est utilisé pour ouvrir et fermer des fichiers pour l'environnement d'exécution de shell courant (voir Shell Execution Environment) ou pour toute commande. Les opérateurs de redirection peuvent être utilisés avec des numéros représentant les descripteurs de fichiers (voir le volume des définitions de base de l'IEEE Std 1003.1-2001, section 3.165, Descripteur de fichiers) comme décrit ci-dessous.

Le format global utilisé pour la redirection est le suivant :

```
[n] redir-op word
```

Le numéro *n* est un numéro décimal facultatif désignant le numéro du descripteur de fichier ; la demande doit s'assurer qu'elle est délimitée de tout texte précédent et précéder immédiatement l'opérateur de redirection *redir-op. Si n est* cité, le nombre ne doit pas être reconnu comme faisant partie de l'expression de redirection. Par exemple:

`echo\2 >` a

Par exemple:

`echo 2\>` a

écrit les caractères 2 > a à la sortie standard. Le numéro optionnel, l'opérateur de redirection *et le mot* ne doivent pas apparaître dans les arguments fournis à la commande à exécuter (le cas échéant).

Les fichiers ouverts sont représentés par des nombres décimaux commençant par zéro. La valeur la plus grande possible est définie par la mise en œuvre ; toutefois, toutes les implémentations doivent supporter au moins 0 à 9, inclusivement, pour être utilisées par l'application. Ces numéros sont appelés « descripteurs de fichiers ». Les valeurs 0, 1 et 2 ont une signification particulière et des utilisations classiques et sont impliquées par certaines opérations de redirection ; ils sont appelés respectivement entrée standard, sortie standard et erreur standard. Les programmes prennent généralement leur entrée à partir de l'entrée standard, et écrivent la sortie sur la sortie standard. Les messages d'erreur sont généralement écrits sur erreur standard. Les opérateurs de redirection peuvent être précédés d'un ou plusieurs chiffres (sans que des « vierges » n'interviennent) pour désigner le numéro de descripteur de fichier.

Si l'opérateur de redirection est "`< <` ou" `< <-` ", le mot qui suit l'opérateur de redirection doit être supprimé ; il n'est pas précisé si les autres expansions se produisent. Pour les autres opérateurs de redirection, le mot qui suit l'opérateur de redirection doit être soumis à l'expansion de tilde, à l'expansion de paramètre, à la substitution de commande, à l'expansion arithmétique et à la suppression de citation. L'expansion du nom Pathname ne doit pas être effectuée sur le mot par un shell non interactif ; un shell interactif peut l'exécuter, mais ne doit le faire que lorsque l'expansion aboutirait à un mot**.**

Si plus d'un opérateur de redirection est spécifié avec une commande, l'ordre d'évaluation est du début à la fin.

Un défaut d'ouverture ou de création d'un fichier provoque l'échec d'une redirection.

### *2.7.1 Redirection des entrées*

La redirection d'entrée doit entraîner l'ouverture du fichier dont le nom résulte de l'expansion *du mot* pour lecture sur le descripteur de fichier désigné, ou l'entrée standard si le descripteur de fichier n'est pas spécifié.

Le format général pour rediriger les entrées est :

```
[n] < mot
```

où l'option *n représente* le numéro du descripteur de fichier. Si le numéro est omis, la redirection doit se référer à l'entrée standard (descripteur de fichier 0).

### *2.7.2 Sortie de redirection*

Les deux formats généraux pour rediriger la sortie sont :

```
[n] > mot
[n]>|word
```

où l'option *n représente* le numéro du descripteur de fichier. Si le numéro est omis, la redirection doit se référer à la sortie standard (descripteur de fichier 1).

La redirection de sortie au `format` '>' échoue si l'option *noclobber* est définie (voir la description de *l'ensemble -C)* et si le fichier nommé par l'extension *de mot* existe et est un fichier régulier. Dans le cas contraire, la redirection au format '>' ou '>|' provoque la création et l'ouverture du fichier dont le nom résulte de l'expansion du mot pour la sortie sur le descripteur de fichier désigné, ou la sortie standard si aucune n'est spécifiée. Si le fichier n'existe pas, il doit être créé ; sinon, il doit être tronqué pour être un fichier vide après avoir été ouvert.

### *2.7.3 Ajout de la sortie redirigée*

La redirection de sortie annexée provoque l'ouverture du fichier dont le nom résulte de l'expansion du mot pour la sortie sur le descripteur de fichier désigné. Le fichier est ouvert comme si la https://pubs.opengroup.org/onlinepubs/009695399/functions/open.htmlfonction open () telle que définie dans le volume des interfaces système de l'IEEE Std 1003.1-2001 était appelée avec le drapeau O_APPEND. Si le fichier n'existe pas, il doit être créé.

Le format général pour ajouter la sortie redirigée est le suivant :

```
[n] > > mot
```

où l'option *n représente* le numéro du descripteur de fichier. Si le numéro est omis, la redirection se réfère à la sortie standard (descripteur de fichier 1).

### *2.7.4 Ici-Document*

Les opérateurs de redirection « < < » et « < < - » permettent tous deux de rediriger des lignes contenues dans un fichier d'entrée shell, dit « ici-document », vers l'entrée d'une commande.

Le présent document sera traité comme un seul mot qui commence après la « nouvelle ligne » suivante et se poursuit jusqu'à ce qu'il y ait une ligne contenant seulement le délimiteur et une « nouvelle ligne », sans « blanc » entre les deux. Ensuite, le document suivant commence, s'il y en a un. Le format est le suivant :

```
[n] < < mot
    ici-document
```

où l'option *n représente* le numéro du descripteur de fichier. Si le numéro est omis, le document se réfère ici à l'entrée standard (descripteur de fichier 0).

Si un caractère dans *le mot est* cité, le délimiteur doit être formé en effectuant la suppression de la citation sur le *mot*, et les lignes du document ne doivent pas être élargies. Sinon, le délimiteur est le *mot lui-même*.

Si aucun caractère dans *le mot* n'est cité, toutes les lignes du document ici seront étendues pour l'expansion des paramètres, la substitution des commandes et l'expansion arithmétique. Dans ce cas, le backslash dans l'entrée se comporte comme le backslash à l'intérieur des doubles-citations (voir Double-citations). Toutefois, le caractère de double citation `('« ») ne doit` pas être traité spécialement dans un présent document, sauf lorsque la double citation apparaît dans « $ `() », « », ou « $ {} ».`

---

*Les sections suivantes sont instructives.*

Voici un exemple de document :

```
chat < < eof1 ; chat < < eof2
Salut,
Hélène.
```

*Fin du texte informatif.*

---

### *2.7.5 Dupliquer un descripteur de fichier d'entrée*

L'opérateur de redirection :

```
[n] < & word
```

doit dupliquer un descripteur de fichier d'entrée d'un autre, ou doit fermer un. Si *word* évalue à un ou plusieurs chiffres, le descripteur de fichier désigné par *n*, ou l'entrée standard *si n* n'est pas spécifié, doit être une copie du descripteur de fichier désigné par *mot* ; si les chiffres en *mot* ne représentent pas un descripteur de fichier déjà ouvert à l'entrée, il en résulte une erreur de redirection ; voir Conséquences des erreurs de Shell. Si *word* évalue à '-', `le` descripteur de fichier *n,* ou l'entrée standard si *n* n'est pas spécifié, doit être fermé. Les tentatives de fermeture d'un descripteur de fichier qui n'est pas ouvert ne constituent pas une erreur. Si *le mot s'évalue* à autre chose, le comportement n'est pas précisé.

### *2.7.6 Dupliquer un descripteur de fichier de sortie*

L'opérateur de redirection :

```
[n] > & word
```

doit dupliquer un descripteur de fichier de sortie d'un autre, ou doit fermer un. Si *le mot* est évalué à un ou plusieurs chiffres, le descripteur de fichier désigné par n, ou la sortie standard si *n* n'est pas spécifié, doit être une copie du descripteur de fichier désigné par *mot* ; si les chiffres en *mot* ne représentent pas un descripteur de fichier déjà ouvert en sortie, il en résulte une erreur de redirection ; voir Conséquences des erreurs de Shell. Si *word évalue* à `'-',` le descripteur de fichier *n*, ou la sortie standard si *n* n'est pas spécifié, est fermé. Les tentatives de fermeture d'un descripteur de fichier qui n'est pas ouvert ne constituent pas une erreur. Si *le mot s'évalue* à autre chose, le comportement n'est pas précisé.

### *2.7.7 Ouvrir les descripteurs de fichiers pour la lecture et l'écriture*

L'opérateur de redirection :

```
[n] < > mot
```

doit faire ouvrir le fichier dont le nom est l'expansion du mot à la fois pour la lecture et l'écriture sur le descripteur de fichier désigné par *n*, ou l'entrée standard *si n* n'est pas spécifié. Si le fichier n'existe pas, il doit être créé.

## 2.8 État de sortie et erreurs

### *2.8.1 Conséquences des erreurs de Shell*

Dans le cas d'un shell non interactif, une condition d'erreur rencontrée par un module spécial intégré (voir Utilitaires spéciaux intégrés) ou un autre type d'utilitaire doit amener le module à écrire un message de diagnostic à l'erreur standard et à sortir comme indiqué dans le tableau suivant :

| | **Spécial intégré** | **Autres services publics** |
|---|---|---|
| Erreur de syntaxe du langage Shell | Doit sortir | Doit sortir |
| Erreur de syntaxe utilitaire (erreur d'option ou d'opérande) | Doit sortir | Ne doit pas sortir |
| Erreur de redirection | Doit sortir | Ne doit pas sortir |
| Erreur d'affectation variable | Doit sortir | Ne doit pas sortir |
| Erreur d'expansion | Doit sortir | Doit sortir |
| Commande non trouvée | S.O. | Peut sortir |
| Script de point non trouvé | Doit sortir | S.O. |

Une erreur d'expansion est une erreur qui se produit lorsque les expansions de shell définies dans Word Expansions sont effectuées (par exemple, « `$ {x ! y}` », `parce que '!'` n'est pas un opérateur valide) ; une implémentation peut les traiter comme des erreurs de syntaxe si elle est capable de les détecter lors de la tokénisation, plutôt que lors de l'expansion.

Si l'une des erreurs indiquées comme « doit sortir » ou « (peut sortir) » se produit dans une sous-coque, la sous-coque doit (peut respectivement) sortir avec un statut non nul, mais le script contenant la sous-coque ne doit pas sortir à cause de l'erreur.

Dans tous les cas indiqués dans le tableau, un shell interactif doit écrire un message de diagnostic à erreur standard sans sortir.

### *2.8.2 Statut de sortie des commandes*

Chaque commande a un statut de sortie qui peut influencer le comportement d'autres commandes shell. L'état de sortie des commandes qui ne sont pas des utilitaires est documenté dans cette section. L'état de sortie des services publics standard est documenté dans leurs sections respectives.

Si une commande n'est pas trouvée, l'état de sortie doit être 127. Si le nom de la commande est trouvé, mais qu'il ne s'agit pas d'un utilitaire exécutable, le statut de sortie est 126. Les applications qui invoquent des utilitaires sans utiliser le shell devraient utiliser ces valeurs d'état de sortie pour signaler des erreurs similaires.

Si une commande échoue pendant l'expansion ou la redirection du mot, son état de sortie doit être supérieur à zéro.

En interne, afin de décider si une commande sort avec un état de sortie non nul, le shell doit reconnaître la totalité de la valeur d'état récupérée pour la commande par l'équivalent de la https://pubs.opengroup.org/onlinepubs/009695399/functions/wait.htmlfonction d'attente () WEXITSTATUS macro (telle que définie dans le volume des interfaces système de l'IEEE Std 1003.1-2001). Lors de la déclaration de l'état de sortie avec le paramètre spécial « `?` », le shell doit déclarer les huit bits complets de l'état de sortie disponibles. L'état de sortie d'une commande qui a pris fin parce qu'elle a reçu un signal doit être signalé comme étant supérieur à 128.

## 2.9 Commandes Shell

Cette section décrit la structure de base des commandes shell. Les descriptions de commande suivantes décrivent chacune un format de la commande qui n'est utilisé que pour aider le lecteur à reconnaître le type de commande, et ne représente pas formellement la syntaxe. Chaque description traite de la sémantique de la commande ; pour une définition formelle du langage de commande, consultez Shell Grammar.

Une *commande est* l'une des suivantes :

- Commande simple (voir Commandes simples)
- Pipeline (voir Pipelines)
- Liste composée (voir Listes)
- Commande composée (voir Commandes composées)
- Définition de la fonction (voir commande Définition de la fonction)

Sauf indication contraire, le statut de sortie d'une commande est celui de la dernière commande simple exécutée par la commande. Il ne doit y avoir aucune limite à la taille d'une commande shell autre que celle imposée par le système sous-jacent (contraintes de mémoire, {ARG_MAX}, etc.).

### *2.9.1 Commandes simples*

Une « commande simple » est une séquence d'affectations et de redirections variables optionnelles, dans toute séquence, éventuellement suivie de mots et de redirections, terminées par un opérateur de contrôle.

Lorsqu'une commande simple donnée doit être exécutée (c'est-à-dire lorsqu'une construction conditionnelle telle qu'une liste ET-OR ou une instruction de cas n'a pas contourné la commande simple), les extensions, affectations et redirections suivantes doivent être effectuées du début du texte de la commande à la fin :

1. Les mots qui sont reconnus comme des assignations variables ou des redirections selon les règles de grammaire Shell sont enregistrés pour traitement aux étapes 3 et 4.
2. Les mots qui ne sont pas des affectations ou des redirections variables doivent être étendus. Si des champs subsistent après leur expansion, le premier champ est considéré comme le nom de la commande et les champs restants sont les arguments de la commande.
3. Les redirections doivent être effectuées comme décrit dans Redirection.
4. Chaque affectation variable doit être étendue pour l'expansion de tilde, l'expansion de paramètre, la substitution de commande, l'expansion arithmétique et la suppression de citation avant d'attribuer la valeur.

Dans la liste précédente, l'ordre des étapes 3 et 4 peut être inversé pour le traitement des utilitaires spéciaux intégrés ; voir Utilitaires spéciaux intégrés.

Si aucun nom de commande ne résulte, les affectations variables doivent affecter l'environnement d'exécution actuel. Dans le cas contraire, les affectations variables doivent être exportées pour l'environnement d'exécution de la commande et ne doivent pas affecter l'environnement d'exécution actuel (à l'exception des éléments spéciaux intégrés). Si l'une quelconque des affectations de variables tente d'attribuer une valeur à une variable en lecture seule, une erreur d'assignation de variables doit se produire. Voir Conséquences des erreurs Shell pour les conséquences de ces erreurs.

S'il n'y a pas de nom de commande, toute redirection doit être effectuée dans un environnement de sous-coque ; il n'est pas précisé si cet environnement de sous-coque est le même que celui utilisé pour une substitution de commande au sein de la commande. (Pour affecter l'environnement d'exécution actuel, voir *l'exec () spécial* intégré.) Si l'une des redirections effectuées dans l'environnement d'exécution du shell en cours échoue, la commande doit immédiatement échouer avec un état de sortie supérieur à zéro, et le shell doit écrire un message d'erreur indiquant la défaillance. Voir Conséquences des erreurs de Shell pour les conséquences de ces défaillances sur les coques interactives et non interactives.

S'il y a un nom de commande, l'exécution se poursuit comme décrit dans Recherche et exécution de commande. S'il n'y a pas de nom de commande, mais que la commande contenait une substitution de commande, la commande doit être complétée par l'état de sortie de la dernière substitution de commande effectuée. Dans le cas contraire, la commande doit être complétée par un état de sortie nul.

## Recherche et exécution de commandes

Si une commande simple aboutit à un nom de commande et à une liste optionnelle d'arguments, les actions suivantes doivent être effectuées :

1. Si le nom de la commande ne contient pas de slashes, la première étape réussie de la séquence suivante doit avoir lieu :
   a. Si le nom de la commande correspond au nom d'un utilitaire intégré spécial, ce dernier doit être invoqué.
   b. Si le nom de la commande correspond au nom d'une fonction connue de ce shell, la fonction doit être invoquée comme décrit dans la commande de définition de la fonction. Si la mise en œuvre a fourni une utilité standard sous la forme d'une fonction, elle n'est pas reconnue à ce stade. Il doit être invoqué en conjonction avec la recherche du chemin à l'étape 1d.
   c. Si le nom de la commande correspond au nom d'un utilitaire énuméré dans le tableau suivant, cet utilitaire est invoqué.

| | | | |
|---|---|---|---|
| https://pubs.opengroup.org/onlinepubs/009696799/utilities/bg.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/fc.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/kill.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/true.html |
| https://pubs.opengroup.org/onlinepubs/009696799/utilities/alias.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/false.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/jobs.html | *lire attendre* https://pubs.opengroup.org/onlinepubs/009696799/utilities/wait.html |
| https://pubs.opengroup.org/onlinepubs/009696799/utilities/cd.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/getopts.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/pwd.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/umask.html |
| https://pubs.opengroup.org/onlinepubs/009696799/utilities/command.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/getconf.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/renice.html | https://pubs.opengroup.org/onlinepubs/009696799/utilities/unalias.html |

   d. Sinon, la commande doit être recherchée en utilisant la variable d'environnement PATH décrite dans le volume Définitions de base de l'IEEE Std 1003.1-2001, chapitre 8, Variables d'environnement :
      i. Si la recherche est réussie :
         a. Si le système a implémenté l'utilitaire en tant que fonction régulière intégrée ou en tant que fonction shell, il doit être invoqué à ce point de la recherche de chemin.
         b. Sinon, le shell exécute l'utilitaire dans un environnement utilitaire distinct (voir Shell Execution Environment) avec des actions équivalant à appeler *l'execve() fonctionnent* comme défini dans le volume des interfaces système de l'IEEE Std 1003.1-2001 avec l'argument *de chemin* défini au nom de chemin résultant de la recherche, *arg0 défini* au nom de la commande, et les arguments restants définis

aux opérandes, le cas échéant.

Si la https://pubs.opengroup.org/onlinepubs/009695399/functions/execve.htmlfonction execve () échoue en raison d'une erreur équivalente à l'erreur [ENOEXEC] définie dans le volume des interfaces système de l'IEEE Std 1003.1-2001, le shell exécute une commande équivalant à faire invoquer un shell avec le nom de parcours résultant de la recherche comme premier opérand, avec les arguments restants passés à la nouvelle coque, sauf que la valeur de « `$0` » dans le nouveau shell peut être défini au nom de la commande. Si le fichier exécutable n'est pas un fichier texte, le shell peut contourner cette exécution de commande. Dans ce cas, il doit écrire un message d'erreur et renvoyer un état de sortie de 126.

Une fois qu'un utilitaire a été recherché et trouvé (soit à la suite de cette recherche spécifique, soit dans le cadre d'une activité de démarrage de shell non spécifiée), une implémentation peut se souvenir de son emplacement et n'avoir plus besoin de rechercher l'utilitaire à moins que la variable PATH ait fait l'objet d'une affectation. Si l'emplacement mémorisé échoue pour une invocation ultérieure, le shell doit répéter la recherche pour trouver le nouvel emplacement de l'utilitaire, le cas échéant.

ii. Si la recherche est infructueuse, la commande échoue avec un état de sortie de 127 et le shell doit écrire un message d'erreur.

2. Si le nom de la commande contient au moins une barre oblique, le shell exécute l'utilitaire dans un environnement utilitaire distinct avec des actions équivalant à appeler *la fonction* execve () définie dans le volume des interfaces système de l'IEEE Std 1003.1-2001 avec les arguments path et arg0 réglés sur le nom de la commande, et les arguments restants réglés sur les opérandes, le cas échéant.

   Si la https://pubs.opengroup.org/onlinepubs/009695399/functions/execve.htmlfonction execve () échoue en raison d'une erreur équivalente à l'erreur [ENOEXEC], le shell exécute une commande équivalente à avoir un shell invoqué avec le nom de la commande comme premier opérand, avec les éventuels arguments restants passés au nouveau shell. Si le fichier exécutable n'est pas un fichier texte, le shell peut contourner cette exécution de commande. Dans ce cas, il doit écrire un message d'erreur et renvoyer un état de sortie de 126.

### *2.9.2 Pipelines*

Un *pipeline* est une séquence d'une ou plusieurs commandes séparées par l'opérateur de contrôle « | ». La sortie standard de toutes les commandes sauf la dernière doit être reliée à l'entrée standard de la commande suivante.

Le format d'un pipeline est le suivant :

La sortie standard du *commande1* doit être reliée à l'entrée standard *du commande2. L'entrée standard*, la sortie standard ou les deux d'une commande sont considérées comme attribuées par le pipeline avant toute redirection spécifiée par les opérateurs de redirection qui font partie de la commande (voir Redirection).

Si le pipeline n'est pas en arrière-plan (voir Listes asynchrones), le shell doit attendre que la dernière commande spécifiée dans le pipeline soit terminée, et peut aussi attendre que toutes les commandes soient terminées.

#### Statut de sortie

Si le mot réservé **!** ne précède pas le pipeline, le statut de sortie doit être le statut de sortie de la dernière commande spécifiée dans le pipeline. Dans le cas contraire, l'état de sortie doit être la logique NON de l'état de sortie de la dernière commande.
Autrement dit, si la dernière commande retourne à zéro, l'état de sortie est 1 ; si la dernière commande revient supérieure à zéro, l'état de sortie est nul.

### *2.9.3 Listes*

Une *liste* ET-OR est une séquence d'un ou plusieurs pipelines séparés par les opérateurs « `& &` » et « `||` ».

Une *liste est* une séquence d'une ou plusieurs listes ET-OR séparées par les opérateurs '`;` ' et '`&`' et éventuellement terminées par ',' `& 'ou`' newline '.

Les opérateurs « `& &` » et « `||` » ont une priorité égale et sont évalués avec l'associativité de gauche. Par exemple, les deux commandes suivantes écrivent **uniquement la barre** sur la sortie standard :

```
faux & & echo foo || echo bar
true || echo foo & & echo bar
```

Un `terminateur « ; » ou` « newline » doit faire exécuter séquentiellement la liste ET-OR précédente ; un « `&` » provoque l'exécution asynchrone de la liste ET-OR précédente.

Le terme « liste de composés » est dérivé de la grammaire de Shell Grammar ; elle équivaut à une suite de *listes,* séparées par des « newline » s, qui peuvent être précédées ou suivies d'un nombre arbitraire de « newline » s.

---

*Les sections suivantes sont instructives.*

Voici un exemple qui illustre < newline > s dans des listes composées :

```
# couple de < newline > s


#
 fichier de chat
# couple de < newline > s


# une autre liste
wc file > output & true



# 2 listes

fichier de chat
```

*Fin du texte informatif.*

---

#### Listes asynchrones

Si une commande est terminée par l'opérateur de contrôle ampères `('&'), le` shell exécute la commande asynchrone dans une sous-coque. Cela signifie que le shell ne doit pas attendre la fin de la commande avant d'exécuter la commande suivante.

Le format pour exécuter une commande en arrière-plan est :

L'entrée standard pour une liste asynchrone, avant toute redirection explicite, est considérée comme affectée à un fichier ayant les mêmes propriétés que/dev/null**.** S'il s'agit d'un shell interactif, cela n'est pas nécessaire. Dans tous les cas, la réorientation explicite de l'entrée standard doit prévaloir sur cette activité.

Lorsqu'un élément d'une liste asynchrone (la partie de la liste terminée par un amplificateur, tel que *commande1, ci-dessus)* est lancé par le shell, le processus ID de la dernière commande de l'élément de liste asynchrone devient connu dans l'environnement d'exécution du shell courant ; voir Environnement d'exécution Shell. Cette ID de procédé reste connue jusqu'à ce que :

1. La commande se termine et l'application attend l'ID du processus.
2. Une autre liste asynchrone invoquée avant « `$` ! » (correspondant à la liste asynchrone précédente) est élargi dans l'environnement d'exécution actuel.

L'implémentation ne doit pas conserver plus que les entrées {CHILD_MAX} les plus récentes dans sa liste d'identifiants de processus connus dans

l'environnement d'exécution de shell actuel.

#### Statut de sortie

L'état de sortie d'une liste asynchrone est nul.

#### Listes séquentielles

Les commandes séparées par un point-virgule (';') doivent être exécutées séquentiellement.

Le format d'exécution séquentielle des commandes est le suivant :

Chaque commande doit être étendue et exécutée dans l'ordre spécifié.

#### Statut de sortie

L'état de sortie d'une liste séquentielle est l'état de sortie de la dernière commande de la liste.

#### ET Listes

L'opérateur de contrôle « `& &` » désigne une liste ET. Le format est le suivant :

*Le premier commandement1 est* exécuté. Si son statut de sortie est nul, *le commande2* doit être exécuté, et ainsi de suite, jusqu'à ce qu'une commande ait un statut de sortie non nul ou qu'il ne reste plus de commandes à exécuter. Les commandes ne sont étendues que si elles sont exécutées.

#### Statut de sortie

L'état de sortie d'une liste ET doit être l'état de sortie de la dernière commande qui est exécutée dans la liste.

#### OU Listes

L'opérateur de contrôle « `||` » désigne une liste OU. Le format est le suivant :

*Premièrement, le commande1 est exécuté*. Si son statut de sortie est non nul, *le commande2* doit être exécuté, et ainsi de suite, jusqu'à ce qu'une commande ait un statut de sortie zéro ou qu'il ne reste plus de commandes à exécuter.

#### Statut de sortie

L'état de sortie d'une liste OU doit être l'état de sortie de la dernière commande qui est exécutée dans la liste.

### *2.9.4 Commandes composées*

Le shell a plusieurs constructions de programmation qui sont des « commandes composées », qui fournissent un flux de contrôle pour les commandes. Chacune de ces commandes composées comporte un mot réservé ou

opérateur de commande au début, et un mot réservé ou opérateur de terminaison correspondant à la fin. De plus, chacune peut être suivie de redirections sur la même ligne que le terminateur. Chaque redirection s'applique à toutes les commandes de la commande composée qui n'annulent pas explicitement cette redirection.

### Groupement des commandes

Le format de regroupement des commandes est le suivant :

(*liste composée)*
: Exécuter *une liste de composés dans un environnement de sous-coque* ; voir Environnement d'exécution Shell. Les affectations variables et les commandes intégrées qui affectent l'environnement ne doivent pas rester en vigueur après la fin de la liste.

{*liste composée ;}*
: Exécutez *la liste des composés dans l'environnement* de processus actuel. Le point-virgule représenté ici est un exemple d'opérateur de contrôle délimitant le mot réservé}. D'autres délimitateurs sont possibles, comme le montre Shell Grammar ; une < newline > est fréquemment utilisée**.**

### Statut de sortie

L'état de sortie d'une commande de regroupement doit être l'état de sortie de la *liste composée.*

### The for Loop

La **boucle** for exécute une séquence de commandes pour chaque membre dans une liste *d'éléments. La* boucle for exige que les mots réservés **soient** utilisés et effectués pour délimiter la séquence de commandes.

Le format de la boucle est le suivant :

```
pour nom [in [word...] ] ne
    liste composée
```

Premièrement, la liste des mots qui suivent **doit** être élargie pour générer une liste d'éléments. Ensuite, le *nom* de la variable doit être défini à chaque élément, à tour de rôle, et la *liste composée* exécutée à chaque fois. Si aucun élément ne résulte de l'expansion, la *liste composée ne* doit pas être exécutée. Omettre :

```
en mots...
```

équivaut à :

```
en « $ @ »
```

### Statut de sortie

Le statut de sortie de la commande a for est le statut de sortie de la dernière commande qui s'exécute. S'il n'y a pas d'éléments, l'état de sortie est nul.

### Affaire Construction conditionnelle

**Le cas** de construction conditionnelle doit exécuter *la liste de composés correspondant* à la première de plusieurs *motifs* (voir Notation d'appariement de motifs) qui est appariée par la chaîne résultant de l'expansion de tilde, de l'expansion de paramètres, de la substitution de commande, de l'expansion arithmétique et de la suppression de citation du mot donné. Le mot réservé **doit** indiquer le début des motifs à apparier. Plusieurs motifs ayant la même *liste de composés doivent* être délimités `par le symbole` « | ». L'opérateur de contrôle `')'` termine une liste de motifs correspondant à une action donnée. La *liste composée pour chaque* liste de motifs, à l'exception éventuelle de la dernière, doit se terminer `par « ; »`. **La construction** de cas se termine par le **mot esac réservé (**cas inversé).

Le format de la **construction** de cas est le suivant :

```
case word in
    [(] annexe 1) liste composée ;
    [[(] pattern [| pattern]...] liste composée ;]...
    [[(] pattern [| pattern]...] liste composée]
```

Le « `;` » est facultatif pour la dernière *liste de composés.*

Dans l'ordre du début à la fin de l'énoncé de **cas**, chaque *motif* qui *marque une liste composée* doit être soumis à l'expansion de tilde, à l'expansion de paramètre, à la substitution de commande et à l'expansion arithmétique, et le résultat de ces expansions doit être comparé à l'expansion de *mot,* selon les règles décrites dans Pattern Matching Notation (qui décrit également l'effet de citer des parties du motif). Après la première correspondance, plus aucun motif ne sera étendu, et la *liste composée* sera exécutée. L'ordre d'expansion et de comparaison des *motifs* multiples qui marquent un énoncé de liste composée n'est pas précisé.

### Statut de sortie

L'état de sortie du **cas** est nul si aucun schéma n'est adapté. Dans le cas contraire, l'état de sortie doit être l'état de sortie de la dernière commande exécutée dans la *liste composée*.

### La construction si conditionnelle

La **commande** if doit exécuter une *liste composée et* utiliser son statut de sortie pour déterminer s'il faut exécuter une autre *liste composée.*

Le format de si **construction** est le suivant :

```
si composé-listthène
    liste de composés [elif compound-listthen
    liste composée]...
[autre
    liste composée]
```

La **liste si composée** doit être exécutée ; si son état de sortie est nul, la *liste composée* doit alors être exécutée et la commande doit être terminée. Dans le cas contraire, **chaque liste composée** elif *doit* être exécutée à son tour, et si son état de sortie est nul, la **liste composée** doit alors être exécutée et la commande doit être terminée. Dans le cas contraire, la *liste composée* est exécutée.

### Statut de sortie

L'état de sortie de la **commande** if doit être l'état de sortie de la **liste composée** alors ou bien qui a été exécutée, ou zéro, si aucune n'a été exécutée.

### Le temps Boucle

La **boucle** d'exécution doit exécuter en continu une *liste de composés tant* qu'une autre *liste de composés* a un état de sortie nul.

Le format de la boucle est le suivant :

```
alors que composé-liste-1do
      composé-liste-2
```

La *liste composée-1* doit être exécutée, et si elle a un statut de sortie non nul, la **commande** en cours doit être terminée. Sinon, le *composé-liste-2 doit* être exécuté, et le processus doit se répéter.

### Statut de sortie

L'état de sortie de la boucle en cours doit être l'état de sortie du dernier *composé-liste-2 exécuté*, ou zéro si aucun n'a été exécuté.

### Le jusqu'à Boucle

La boucle d'attente doit exécuter en continu une *liste de composés tant* qu'une autre *liste de composés* a un statut de sortie non nul.

Le format de la boucle d'attente est le suivant :

```
jusqu'à ce que composé-liste-
            1do
       composé-liste-2
```

La *liste composée-1* doit être exécutée, et si elle a un statut de sortie zéro, jusqu'à **ce que** la commande se termine. Sinon, le *composé-liste-2 doit être exécuté*, et le processus se répète.

### Statut de sortie

L'état de sortie de la boucle d'attente doit être l'état de sortie du dernier *composé-liste-2 exécuté*, ou zéro si aucun n'a été exécuté.

### *2.9.5 Commande de définition de la fonction*

Une fonction est un nom défini par l'utilisateur qui est utilisé comme une simple commande pour appeler une commande composée avec de nouveaux paramètres de position. Une fonction est définie avec une « commande de définition de fonction ».

Le format d'une commande de définition de fonction est le suivant :

```
fname () compound-command [io-redirect...]
```

La fonction est nommée *fname ;* la demande doit s'assurer qu'il s'agit d'un nom (voir le volume Définitions de base de l'IEEE Std 1003.1-2001, section 3.230, Nom). Une implémentation peut autoriser d'autres caractères dans un nom de fonction comme extension. La mise en œuvre doit maintenir des espaces de noms distincts pour les fonctions et les variables.

La *commande de composé d'argument représente* une commande de composé, comme décrit dans les commandes de composé.

Lorsque la fonction est déclarée, aucune des expansions de Word Expansions ne doit être exécutée sur le texte *en compound-command ou io-redirect ; toutes les* expansions doivent être effectuées de manière normale chaque fois que la fonction est appelée. De même, les redirections optionnelles d'io-redirection et les affectations variables au sein *de la commande composée doivent être* effectuées pendant l'exécution de la fonction elle-même, et non la définition de la fonction. Voir Conséquences des erreurs de Shell pour les conséquences des défaillances de ces opérations sur les obus interactifs et non interactifs.

Lorsqu'une fonction est exécutée, elle doit avoir les propriétés d'erreur de syntaxe et d'affectation variable décrites pour les utilitaires spéciaux intégrés dans la liste énumérée au début des utilitaires spéciaux intégrés.

La *commande composée* doit être exécutée chaque fois que le nom de la fonction est spécifié comme le nom d'une simple commande (voir Recherche et exécution de la commande). Les opérandes de la commande deviennent temporairement les paramètres de position pendant l'exécution de la *commande composée ; le* `paramètre spécial « # »` doit également être modifié pour refléter le nombre d'opérandes. Le paramètre spécial 0 doit être inchangé. Lorsque la fonction est terminée, les valeurs des paramètres positionnels et du paramètre spécial `« # »` doivent être restaurées dans les valeurs qu'ils avaient avant l'exécution de la fonction. Si le *retour spécial intégré est exécuté dans la commande composée,* la fonction s'achève et l'exécution reprend avec la commande suivante après l'appel de la fonction.

### Statut de sortie

L'état de sortie d'une définition de fonction doit être nul si la fonction a été déclarée avec succès ; sinon, elle doit être supérieure à zéro. L'état de sortie d'une invocation de fonction est l'état de sortie de la dernière commande exécutée par la fonction.

## 2.10 Grammaire Shell

La grammaire suivante définit le langage de commande Shell. Cette syntaxe formelle a priorité sur la description de syntaxe de texte précédente.

### *2.10.1 Conventions lexicales de grammaire shell*

Le langage d'entrée du shell doit d'abord être reconnu au niveau des caractères. Les jetons obtenus sont classés selon leur contexte immédiat selon les règles suivantes (appliquées dans l'ordre). Ces règles doivent être utilisées pour déterminer ce qu'est un « jeton » qui est sujet à analyse au niveau du jeton. Les règles relatives à la reconnaissance des jetons dans la reconnaissance des jetons s'appliquent.

1. A < newline > doit être retourné comme identifiant du jeton **NEWLINE.**
2. Si le jeton est un opérateur, l'identifiant du jeton pour cet opérateur doit en résulter.
3. Si la chaîne est composée uniquement de chiffres et que le caractère délimiteur est un de "<" ou "" ", l'identifiant du jeton **IO_NUMBER doit** être retourné.
4. Sinon, l'identifiant de jeton **TOKEN résulte**.

Une autre distinction sur **TOKEN dépend** du contexte. Il se peut que le même **JETON** donne **WORD**, un **NOM**, une **AFFECTATION,** ou l'un des mots réservés ci-dessous, dépendant du contexte. Certaines des productions de la grammaire ci-dessous sont annotées avec un numéro de règle de la liste suivante. Lorsqu'un **TOKEN est vu** où l'une de ces productions annotées pourrait être utilisée pour réduire le symbole, la règle applicable doit être appliquée pour convertir le type d'identifiant de jeton du **TOKEN** en un identifiant de jeton acceptable à ce point de la grammaire. La réduction se fait ensuite sur la base du type d'identificateur de jeton obtenu par la règle appliquée. Lorsque plusieurs règles s'appliquent, la règle numérotée la plus élevée s'applique (qui à son tour peut se référer à une autre règle). (Notez qu'à l'exception de la règle 7, la présence `d'un '=' dans` le jeton n'a aucun effet.)

Les **jetons** WORD doivent avoir les règles d'expansion de mot qui leur sont appliquées immédiatement avant l'exécution de la commande associée, et non au moment où la commande est analysée.

### *2.10.2 Règles de grammaire de shell*

1. [Nom du commandement]

   Lorsque le **TOKEN est** exactement un mot réservé, il en résulte l'identifiant du jeton pour ce mot réservé. Sinon, le jeton **WORD** sera retourné. En outre, si l'analyseur se trouve dans un état où seul un mot réservé pourrait être le prochain jeton correct, procédez comme ci-dessus.

   **Note :**

   Parce qu'à ce stade les guillemets sont conservés dans le jeton, les chaînes citées ne peuvent être reconnues comme des mots réservés. Cette règle implique également que les mots réservés ne sont pas reconnus sauf dans certaines positions dans l'entrée, comme après une < newline > ou un point-virgule ; la grammaire suppose que si le mot réservé est destiné, il est correctement délimité par l'utilisateur, et ne tente pas de refléter directement cette exigence. Notez également que la jonction de ligne est faite avant la tokénisation, comme décrit dans Escape Character (Backslash), ainsi échappé < newline > s sont déjà supprimés à ce point.

   La règle 1 n'est pas directement référencée dans la grammaire, mais est visée par d'autres règles, ou s'applique globalement.

2. [Redirection vers ou à partir du nom du fichier]

   Les expansions spécifiées dans Redirection doivent avoir lieu. Tel que spécifié là, exactement un champ peut résulter (ou le résultat n'est pas spécifié), et il ya des exigences supplémentaires sur l'expansion de pathname.

3. [Redirection de ici-document]

   La suppression de la citation doit être appliquée au mot pour déterminer le délimiteur qui est utilisé pour trouver la fin du présent document qui commence après la « nouvelle ligne » suivante.

4. déclaration de cas]

   Lorsque le **TOKEN est** exactement le mot esac réservé, l'identifiant du jeton pour **esac** doit en résulter. Sinon, le jeton **WORD** sera retourné.

5. **[NOM** pour**]**

   Lorsque le **JETON satisfait aux** exigences relatives à un nom (voir le volume Définitions de base de l'IEEE Std 1003.1-2001, section 3.230, Nom), l'identifiant de jeton **NAME doit** en résulter. Sinon, le jeton **WORD** sera retourné.

6. [Troisième mot pour et **cas**]

   a. **seulement]**

      Lorsque le **TOKEN est** exactement le mot réservé **dedans**, l'identifiant de jeton pour **doit** aboutir. Sinon, le jeton **WORD sera** retourné.

   b. **[pour** seulement]

      Lorsque le **TOKEN est** exactement le mot réservé **dedans ou do,** l'identifiant du token dedans **ou do doit** en résulter, respectivement. Sinon, le jeton **WORD** sera retourné.

   (Pour a. et b. : Comme indiqué dans la grammaire*, un brise-lignée précède* les jetons **dans et faire. Si** des < newline > s sont présents à l'endroit indiqué, c'est le jeton après eux qui est traité de cette façon.)

7. [Affectation avant le nom de la commande]

   a. le premier mot]

      Si le **TOKEN** ne contient pas le caractère `'=', la règle` 1 est appliquée. Dans le cas contraire, le point 7b est appliqué.

   b. [Pas le premier mot]

      Si le **TOKEN** contient le même caractère de signe :

      - S'il commence par `'='`, le jeton **WORD** doit être retourné.
      - Si tous les caractères précédant « = » forment un nom valide (voir le volume Définitions de base de l'IEEE Std 1003.1-2001, section 3.230, Nom), le jeton **ASSIGNMENT_WORD doit être** retourné. (Les personnages cités ne peuvent pas participer à la formation d'un nom valide.)
      - Sinon, il n'est pas précisé si **c'est ASSIGNMENT_WORD ou WORD qui** est retourné.

   L'affectation au **NOM doit** se faire comme spécifié dans les commandes simples.

8. **[NOM** en fonction]

   Lorsque le **TOKEN est** exactement un mot réservé, il en résulte l'identifiant du jeton pour ce mot réservé. Dans le cas contraire**, lorsque le JETON** satisfait aux exigences d'un nom, l'identifiant du jeton **NAME doit** en résulter. Sinon, la règle 7 s'applique.

9. [Corps de fonction]

   L'expansion et l'affectation des mots ne doivent jamais avoir lieu, même lorsque les règles ci-dessus l'exigent, lorsque cette règle est analysée. Chaque **JETON qui** pourrait soit être élargi, soit faire l'objet d'une affectation, sera renvoyé sous la forme d'un seul **MOT composé** uniquement de caractères qui sont exactement le jeton décrit dans la Reconnaissance de jeton.

```
/* ------------------------------------------------------
   Les symboles de grammaire
   ------------------------                                */


% jeton
% jeton ASSIGNMENT_WORD
% jeton Nom
% jeton
% jeton IO NUMBER

/ * Les opérateurs mentionnés ci-dessus sont les
suivants. */


% jeton AND_IF    OR_IF
/*      '&&'      '||'      ';;'    */


% jeton                                LESSGREAT DLESSDASH
/*      '<<'   '>>'    '<&'      '>&'     '<>'       '<<-'   */


% jeton
/*      '>|'   */


/ * Les mots suivants sont réservés. */


% jeton
/*                                                      */


% jeton
/*                                           */


/ * Ce sont des mots réservés, pas des jetons d'opérateur, et sont
   reconnu lorsque les mots réservés sont reconnus */.


%
jeton
/*      '{'       '}'       '!'   */

%
jeton
/*              */


/* ------------------------------------------------------

   ------------------------------------------------------ */


% de démarragecomplete_command
%%
complete_command : séparateur de liste
                 |
                 ;
                 : Liste separator_op and_or
                 |                   and or
                 ;
and or           :
                                pipeline de rupture
                 | and_or AND_IF            de ligne
                                pipeline de rupture
                 | and_or OR_IF             de ligne
                 ;
                 :      pipe sequence
                 | Bang pipe_sequence
                 ;
pipe sequence    :
```

```
                | pipe_sequence « | » commande de
                ligne
                ;
                : simple_command
                | compound command
                | compound_command redirect_list
                | function_definition
                ;
compound_command : brace_group
                | sous-coque
                | for clause
                | case clause
                | if_clause
                | while clause
                | until_clause
                ;
                « ( » compound_list « ) »
                ;
compound list   :
                | newline_list terme
                |              séparateur de terme
                séparateur à | newline_list terme
                ;
                : séparateur de terme
                and_or
                |                and or
                ;
for clause      : Pour le nom linebreak                        do group
                | Pour le nom linebreak
                dans                              sequential sep do group
                | Pour nom linebreak dans la liste de mots
                sequential_sep do_group
                ;
                                          / * Appliquer la
nom             :Nom                      règle 5 */
                ;
                                          / * Appliquer la
                : En                      règle 6 */
                ;
                : wordlist WORD
                |
                ;
case clause     : Case WORD linebreak in linebreak case list
                | Case WORD linebreak in linebreak case list ns Esac
                | Case WORD linebreak in linebreak
                ;
case_list_ns    : case_list case_item_ns
                |          case item ns
                ;
case list       : case list case item
                |          case item
                ;
case item ns    :     pattern ') "
                |     pattern « ) » compound list linebreak
                | « ( » motif « ) »
                | « ( » motif « ) » compound list linebreak
                ;
                      pattern « ) »
case item       :     linebreak                DSEMI linebreak
                |     motif « ) » compound_list DSEMI linebreak
                | « ( » pattern « ) »
                linebreak                      DSEMI linebreak
                | « ( » motif « ) » compound_list DSEMI
                linebreak
                ;
                                        / * Appliquer la
                :                       règle 4 */
                                        / * Ne pas appliquer la
                | pattern « | » WORD    règle 4 */
                ;
if clause       Si compound list Alors compound list else part Fi
                | Si compound_list Alors compound_list
                ;
else_part       : Elif compound_list Alors else_part
                | Else compound list
                ;
while_clause    : Tandis que compound_list do_group
                ;
until clause    : Jusqu'à compound list do group
                ;
function_definition : fname « ( » « ) » linebreak function_body
                ;
                                          / * Appliquer
function body   : compound command        la règle 9     */
                                          / * Appliquer
                | compound command redirect list la règle 9     */
                ;
                                          / * Appliquer
                :Nom                      la règle 8     */
                ;
brace_group     : Lbrace compound_list Rbrace
                ;
do_group        : Do compound_list Done          / * Appliquer la règle 6 */
                ;
simple_command  :
                | cmd_prefix

                ;
                                  / *
                                  Applique
cmd_name        : MOT             r               */
                ;
cmd_word                          / *
                                  Applique
cmd_prefix      : MOT             r               */
                ;
                :            io_redirect
                 | cmd_prefix io_redirect
                |            ASSIGNMENT_WORD
cmd_suffix      | cmd_prefix ASSIGNMENT_WORD
                ;
                :            io_redirect
                 | cmd_suffix io_redirect
                |
redirect_list   | cmd_suffix MOT
                ;
                :              io_redirect
io_redirect     | redirect_list io_redirect
                ;
                :           io_file
                 | IO_NUMBER io_file
                |          io_here
io_file         | IO_NUMBER io_here
                ;
                : '<'
                | LESSAND
                | '>'
                | GREAT AND
                | DGREAT
                | LESSGREAT nom de fichier
                | Nom de fichier CLOBBER
                ;
io_here         : MOT                     / * Appliquer la règle 2 */
                ;
                : DLESS     here_end
here_end        | DLESSDASH here_end
                ;
newline_list    : MOT                     / * Appliquer la règle 3 */
                ;
                :              NEWLINE
                 | newline_list NEWLINE
                ;
                : newline_list
separator_op    | / * vide */
                ;
                : '&'
                |  ';'
                ;
                : separator_op linebreak
sequential_sep  | newline_list
                ;
                : ";" linebreak ";
                | newline_list
                ;
```

## 2.11 Traitement des signaux et des erreurs

Lorsqu'une commande est dans une liste asynchrone, le shell doit empêcher les signaux SIGQUIT et SIGINT du clavier d'interrompre la commande. Dans le cas contraire, les signaux doivent avoir les valeurs héritées par le réservoir de sa mère (voir aussi le *piège* spécial incorporé).

Lorsqu'un signal pour lequel un piège a été fixé est reçu pendant que la coquille attend l'achèvement d'un utilitaire exécutant une commande de premier plan, le piège associé à ce signal ne doit être exécuté qu'après l'achèvement de la commande de premier plan. Lorsque la coquille attend, au moyen de l'utilitaire *d'attente*, la fin des commandes asynchrones, la réception d'un signal pour lequel un piège a été réglé fait https://pubs.opengroup.org/onlinepubs/009695399/utilities/wait.htmlrevenir immédiatement l'utilitaire d'attente avec un état de sortie > 128, immédiatement après quoi le piège associé à ce signal doit être pris.

Si plusieurs signaux sont en attente pour le shell pour lequel il y a des actions pièges associées, l'ordre d'exécution des actions pièges n'est pas précisé.

## 2.12 Environnement d'exécution de Shell

Un environnement d'exécution de shell se compose des éléments suivants :

- Ouvrir les fichiers hérités de l'invocation du shell, plus les fichiers ouverts contrôlés par *exec*
- Répertoire de travail tel que défini par *cd*
- Masque de création de fichiers défini par *umask*
- Pièges à courant réglés *par piège*
- Paramètres Shell qui sont définis par assignation variable (voir l'ensemble spécial intégré) ou à partir du volume des interfaces système de l'environnement IEEE Std 1003.1-2001 hérité par le shell quand il commence (voir l'export *spécial* intégré)
- Fonctions Shell ; voir la commande Définition de la fonction
- Options activées à l'invocation ou par *set*
- Traiter les IDs des dernières commandes dans des listes asynchrones connues de cet environnement shell ; voir Listes asynchrones
- Aliases pour coque ; voir Alias Substitution

Les services publics autres que les services intégrés spéciaux (voir Services publics intégrés spéciaux) doivent être invoqués dans un environnement distinct qui consiste en ce qui suit. La valeur initiale de ces objets doit être la même que celle de la coque mère, sauf comme indiqué ci-dessous.

- Ouvrir les fichiers hérités de l'invocation du shell, ouvrir les fichiers contrôlés par *l'exec* spécial intégré plus les modifications, et les ajouts spécifiés par les redirections à l'utilitaire
- Répertoire de travail actuel
- Masque de création de fichiers
- Si l'utilitaire est un script shell, les pièges capturés par le shell doivent être réglés sur les valeurs par défaut et les pièges ignorés par le shell doivent être réglés pour être ignorés par l'utilitaire ; si l'utilitaire n'est pas un script shell, les actions du piège (par défaut ou ignorer) doivent être cartographiées dans les actions appropriées de traitement du signal pour l'utilitaire

  Les variables avec l'attribut *export*, ainsi que celles explicitement exportées pendant la durée de la commande, doivent être transmises aux variables de

l'environnement utilitaire L'environnement du processus shell ne doit pas être modifié par l'utilitaire sauf si la description de l'utilitaire le spécifie explicitement (par

exemple, *cd* et *umask*).

Un environnement de sous-coque doit être créé en double de l'environnement de la coque, sauf que les pièges à signaux définis par cet environnement de la coque doivent être réglés sur les valeurs par défaut. Les modifications apportées à l'environnement de la sous-coque ne doivent pas affecter l'environnement de la coque. La substitution de commande, les commandes regroupées entre parenthèses et les listes asynchrones doivent être exécutées dans un environnement de sous-coque. De plus, chaque commande d'un pipeline multi-commandes est dans un environnement de sous-coque ; en tant qu'extension, cependant, tout ou partie des commandes d'un pipeline peuvent être exécutées dans l'environnement actuel. Toutes les autres commandes doivent être exécutées dans l'environnement shell courant.

## 2.13 Notation correspondant au modèle

La notation d'appariement de motifs décrite dans cette section est utilisée pour spécifier des motifs d'appariement de chaînes dans le shell. Historiquement, la notation d'appariement des patrons est liée, mais légèrement différente, à la notation d'expression régulière décrite dans le volume Définitions de base de l'IEEE Std 1003.1-2001, chapitre 9, Expressions régulières. Pour cette raison, la description des règles pour cette notation d'appariement de motifs est basée sur la description de notation d'expression régulière, modifiée pour inclure le traitement d'échappement en contre-réaction.

### *2.13.1 Motifs correspondant à un seul caractère*

Les motifs suivants correspondant à un seul caractère doivent correspondre à un seul caractère : caractères ordinaires, caractères de motifs spéciaux et expressions de parenthèses de motifs. L'expression de la console doit également correspondre à un seul élément de collage. Un caractère de contre-réaction doit échapper au caractère suivant. Le jeu de dos qui s'échappe doit être écarté.

Un caractère ordinaire est un motif qui doit correspondre à lui-même. Il peut s'agir de n'importe quel caractère dans le jeu de caractères pris en charge, à l'exception de NUL, des caractères shell spéciaux dans Citations qui nécessitent une citation, et des trois caractères de motifs spéciaux suivants. L'appariement doit être basé sur le motif de bit utilisé pour encoder le caractère, et non sur la représentation graphique du caractère. Si un caractère (ordinaire, spécial shell, ou spécial pattern) est cité, ce motif doit correspondre au caractère lui-même. Les caractères spéciaux du shell nécessitent toujours une citation.

Lorsqu'ils ne sont pas cités et en dehors d'une expression entre parenthèses, les trois caractères suivants doivent avoir une signification particulière dans la spécification des motifs :

`?`
: Un point d'interrogation est un motif qui doit correspondre à n'importe quel caractère.

`*`
: Un astérisque est un motif qui doit correspondre à plusieurs caractères, comme décrit dans Patterns Matching Multiple Characters.

`[`
: La console ouverte doit introduire une expression de la console.

La description des expressions ordinaires de base entre parenthèses dans le volume des définitions de base de l'IEEE Std 1003.1-2001, section 9.3.5, Expression des parenthèses RE s'applique également à l'expression des parenthèses de motifs, sauf que le caractère de marque d'exclamation (« ! ») `remplace` le caractère circonflexe `(« ^ ») dans` son rôle dans une « liste non correspondante » dans la notation d'expression régulière. Une expression de parenthèse commençant par un caractère circonflexe non cité produit des résultats non spécifiés.

Lorsque la correspondance de motifs est utilisée lorsque la suppression de citation shell n'est pas effectuée (tel que dans l'argument à *la recherche - nom primaire lorsque trouver* est appelé en utilisant l'une des fonctions exec telles que définies dans le volume des interfaces système de IEEE Std 1003.1-2001, ou dans l'argument *de modèle* à la *fnmatch()*, les caractères spéciaux peuvent être échappés pour enlever leur signification spéciale en les précédant d'un caractère de jeu de dos. Ce recul échappatoire est écarté. La séquence `« \\ »` représente un jeu de dos littéral. Toutes les exigences et tous les effets de la mention sur les caractères ordinaires, spéciaux et de motifs spéciaux s'appliquent à l'évasion dans ce contexte.

### *2.13.2 Motifs correspondant à plusieurs caractères*

Les règles suivantes sont utilisées pour construire des motifs correspondant à plusieurs caractères à partir de motifs correspondant à un seul caractère :

1. L'astérisque `('*')` est un motif qui doit correspondre à n'importe quelle chaîne, y compris la chaîne nulle.
2. La concaténation de motifs correspondant à un seul caractère est un motif valide qui doit correspondre à la concaténation des caractères uniques ou des éléments de rassemblement correspondant à chacun des motifs concaténés.
3. La concaténation d'un ou plusieurs motifs correspondant à un seul caractère avec un ou plusieurs astérisques est un motif valide. Dans de tels motifs, chaque astérisque doit correspondre à une chaîne de caractères zéro ou plus, correspondant au plus grand nombre possible de caractères qui permet encore au reste du motif de correspondre à la chaîne.

### *2.13.3 Motifs utilisés pour l'expansion du nom de fichier*

Les règles décrites jusqu'à présent dans Patterns Matching a Single Character et Patterns Matching Multiple Characters sont qualifiées par les règles suivantes qui s'appliquent lorsque pattern matching notation est utilisé pour l'expansion du nom de fichier :

1. Le caractère « slash » d'un pathname doit être explicitement apparié en utilisant un ou plusieurs slash dans le modèle ; il ne doit pas être assorti d'un astérisque ou d'une marque d'interrogation, ni d'une expression entre crochets. Les lames du motif doivent être identifiées avant les expressions entre parenthèses ; ainsi, une barre oblique ne peut pas être incluse dans une expression de parenthèse utilisée pour l'expansion du nom de fichier. Si un caractère oblique est trouvé à la suite d'un caractère de crochet ouvert non échappé avant qu'un crochet de fermeture correspondant ne soit trouvé, le support ouvert est traité comme un caractère ordinaire. Par exemple, le motif `« a [b/c]` d » ne correspond pas à des patrons **tels que abd** ou **a/d**. Il correspond seulement à un pathname de littéralement **a [b/c]** d.
2. Si un nom de fichier commence par une période ('.') `, la` période doit être explicitement appariée en utilisant une période comme premier caractère du motif ou immédiatement après un caractère de barre oblique. La période de pointe n'est pas assortie :

   - Les caractères spéciaux de l'astérisque ou du point d'interrogation
   - Une expression entre crochets contenant une liste non assortie, telle que `« [! a] »`, une expression de gamme, `telle que « [% -0]` », ou une expression de classe de caractères, `telle que « [[: punct :]] »`

   Il n'est pas précisé si une période explicite dans une liste de concordance d'expressions entre parenthèses, telle que `« [.abc]` », peut correspondre à une période principale dans un nom de fichier.
3. Les modèles spécifiés doivent être appariés aux noms de famille et aux noms de famille existants, selon le cas. Chaque composant qui contient un caractère de motif doit avoir une autorisation de lecture dans le répertoire contenant ce composant. Tout composant, sauf le dernier, qui ne contient pas de caractère de motif doit être autorisé à effectuer une recherche. Par exemple, compte tenu du modèle :

   ```
   / foo/bar/x */bam
   ```

   une autorisation de recherche est nécessaire pour les répertoires/et **foo,** des autorisations de recherche et de lecture sont nécessaires pour la barre d'annuaire, et une autorisation de recherche est nécessaire pour chaque répertoire x *. Si le motif correspond à un nom de fichier ou à un nom de fichier existant, le motif doit être remplacé par ces noms de fichier et nom de fichier, triés en fonction de la séquence de collectage en vigueur dans la région courante. Si le motif contient une expression de parenthèse invalide ou ne correspond à aucun nom de fichier ou nom de pathologie existant, la chaîne de caractères doit rester inchangée.

## 2.14 Services spéciaux intégrés

Les https://pubs.opengroup.org/onlinepubs/009695399/idx/sbi.htmlutilitaires « spéciaux intégrés » suivants doivent être pris en charge dans le langage de commande shell. La sortie de chaque commande, le cas échéant, doit être écrite sur la sortie standard, sous réserve de la redirection et de la tuyauterie normales possibles avec toutes les commandes.

Le terme « intégré » implique que le shell peut exécuter l'utilitaire directement et n'a pas besoin de le rechercher. Une mise en œuvre peut choisir de faire de tout utilitaire un intégré ; toutefois, les services spéciaux intégrés décrits ici diffèrent des services réguliers intégrés à deux égards :

1. erreur de syntaxe dans un utilitaire intégré spécial peut provoquer l'interruption d'un shell exécutant cet utilitaire, tandis qu'une erreur de syntaxe dans un utilitaire intégré régulier ne doit pas provoquer l'interruption d'un shell exécutant cet utilitaire. (Voir Conséquences des erreurs de Shell pour les conséquences des erreurs sur les coques interactives et non interactives.) Si un utilitaire intégré spécial rencontrant une erreur de syntaxe n'interrompt pas le shell, sa valeur de sortie doit être non nulle.
2. Les affectations variables spécifiées avec les services publics spéciaux intégrés restent en vigueur après la fin de l'intégration ; cela ne doit pas être le cas d'une installation régulière ou d'un autre service public.

Les services spéciaux intégrés dans cette section ne doivent pas être fournis d'une manière accessible par l'intermédiaire de *la famille* exec de fonctions définies dans le volume des interfaces système de l'IEEE Std 1003.1-2001.

Certains des éléments spéciaux sont décrits comme conformes au volume des définitions de base de l'IEEE Std 1003.1-2001, section 12.2, Lignes directrices sur la syntaxe des services publics. Pour ceux qui ne le sont pas, l'exigence contenue dans *Description d'utilité par défaut selon laquelle* « -- » doit être reconnu comme un premier argument à rejeter ne s'applique pas et une demande conforme ne doit pas utiliser cet argument.