POLYTECH
NICE-SOPHIA

SI3 / 2014-2015

ERICK GALLESIO

Projet C/Système

Polysh: un mini-interprète de commandes

Le but de ce projet est de construire un mini interprète de commandes (*shell*) pour Unix. Les principales caractéristiques de cet interprète, que nous appelerons `polysh`, sont détaillées ci-dessous.
Les fichiers qui vous sont distribués permettent de construire et donc d'exécuter `polysh`. Par contre la plupart des codes sources de ce programme ne vous sont pas livrés. Votre travail va consister à réécrire les différents composants du shell et à rendre les sources des composants réécrits. Ainsi, vous aurez toujours un programme qui fonctionne en substituant petit à petit les composants qui vous ont été distribués, par ceux que vous avez réécrits.
Parmi les fichiers sources qui vous sont fournis, il y a un certain nombre de *headers* (des ".h") qui définissent l'interface des composants qui vous sont donnés. Ces fichiers sont des **contrats** et ne **doivent pas** être modifiés. Les modules que vous remplacerez doivent être **compatibles** avec l'interface décrite dans le *header* correspondant.

Syntaxe d'appel

L'appel à l'interprète se fait par

```
polysh [fichier [arg1 arg2 ...] ]
```

Si le paramètre optionnel *fichier* est présent, le contenu du fichier spécifié est exécuté comme une suite de commandes `polysh`. Notez qu'un script `polysh` peut aussi accepter un nombre quelconque de paramètres.

Editeur de commandes / historique des commandes

`Polysh` dispose d'un mini éditeur de commande. Cet éditeur permet de se déplacer dans la ligne de commande en cours de saisie, mais aussi dans l'historique des commandes. Les commandes de l'éditeur que vous devez implémenter sont les suivantes:

Control-A : aller au début de la ligne

Control-B : caractère précédent

Control-D : effacer le caractère courant (sortie de *polysh* si la ligne est vide)

Control-E : aller à la fin de la ligne

Control-F : caractère suivant

Control-H (Backspace) ou Delete : effacer le caractère précédent

Control-K : effacer jusqu'à la fin de la ligne

Control-L : ré-afficher la ligne courante

Control-M (Return) : exécuter la ligne saisie

Control-N : commande suivante de l'historique

Control-P : commande précédente de l'historique

Control-Y : restituer le texte détruit par Control-K

Par ailleurs, votre éditeur devra gérer les flèches du clavier. Les touches fléchées du clavier n'envoient pas un caractère unique, mais une séquence de caractères commençant par le caractère ESCAPE. Ces séquences peuvent dépendre du type de terminal que l'on utilise, mais sur Linux, la plupart du temps, les consoles UNIX utilisent les séquence suivantes:

- ESCAPE [A : flèche vers le haut
- ESCAPE [B : flèche vers le bas
- ESCAPE [C : flèche vers la droite
- ESCAPE [D : flèche vers la gauche

L'historique des commandes est géré en mémoire et sa taille peut être fixée à l'initialisation. À la fin de l'exécution du shell, les commandes stockées dans l'historique, peuvent être sauvegardées dans un fichier. Les commandes ainsi sauvegardées serviront à initialiser l'historique pour la prochaine utilisation du shell. Le fichier de sauvegarde de l'historique est le fichier `/.history_polysh`.

Notes:

- Si la fonction d'initialisation de l'historique n'est pas appelée, on créera un historique de taille `HISTORY_DEFAULT_SIZE` qui ne sera pas sauvegardé.
- Il n'y a pas de fonction publique qui déclenche la sauvegarde de l'historique des commandes: la sauvegarde se fera automatiquement lorsque le shell se termine.

Variables *polysh*

Polysh permet à l'utilisateur de définir des variables internes. Toutes les variables *polysh* sont exportées. La consultation de la valeur d'une variable se fait de façon classique en précédant le nom de la variable du caractère `'$'`.
Les variables suivantes sont pré-définies:

$HOME qui contient le chemin où l'on va si on fait un cd sans paramètre

$PATH (comme *sh*)

$IFS (comme *sh*)

$PROMPT qui contient la chaîne affichée en guise de prompt

$PWD qui contient le répertoire courant

$SHELL qui contient le nom du shell

$SHELL_VERSION qui contient la version du shell

$1, $2 ... qui contiennent les paramètres de la commande passés à *polysh*

$# qui contient le nombre de paramètres

$* qui contient les paramètres séparés par un espace

Macro caractères

Polysh reconnaît les macro-caractères classiques utilisés en général par les shells UNIX. Votre shell reconnaîtra donc les caractères `'*'`, `'?'`, `'['`, ... Vous reconnaîtrez aussi la notation `tilde` qui permet de désigner le *"home directory"* d'un utilisateur. Exemple:

```
polysh$ ls
a  b  c  d  e  f
polysh$ echo [a-c] ~eg
a b c /home/eg
polysh$
```

Heureusement, vous pouvez vous passer d'écrire la fonction de reconnaissance de ces expressions grâce à la primitive `glob(3)` qui permet de calculer l'expansion d'un mot contenant un ou des macro-caractères.

Commandes internes

Les commandes suivantes sont interprétées directement par *polysh* :

cd ou chdir permet de changer de répertoire. L'utilisation de cette commande provoquera la mise à jour de la variable PWD.

pwd pour afficher le répertoire courant (i.e celui qui est contenu dans la variable PWD)

set peut être utilisée de deux façons:

- *set*: affichage des variables de l'environnement
- *set var val*: affecte la valeur *val* à la variable *var*

alias peut être utilisée de deux façons:

- *alias*: affichage de tous les alias définis.
- *alias ali val*: affecte la valeur *val* à l'alias *ali*

Voir plus bas, la section sur les alias.

echo permet d'afficher tous ses paramètres sur la sortie standard avant de passer à la ligne. Si le premier paramètre est `-n`, la commande n'affiche pas de saut de ligne.

exit pour sortir de *polysh* (code de retour facultatif, par défaut 0)

read permet de lire une ligne sur le fichier standard d'entrée et l'affecte à la variable qui lui est passée en paramètre.

history affiche l'historique des commandes.

Aliases

Polysh permet de définir des *alias* comme les shells modernes. Un alias permet de substituer un mot par une chaîne de caractères quand il est utilisé comme premier mot d'une commande. Exemple d'utilisation d'un alias:

```
polysh$ alias heure "date +%H:%M"
polysh$ heure
12:00
polysh$ echo heure    # ici heure n'est pas vue comme un alias
heure
```

Un alias peut bien sûr utiliser un autre alias dans son expansion[1].

[1] Attention;: cela peut entraîner des boucles. Lorqu'on expanse une commande, on vérifiera donc que l'on ne fait pas *trop* d'expansions.

```
polysh$ alias H heure
polysh$ H          # H → heure → date +%H:%M"
12:01
polysh$ alias X "H | "
polysh$ X wc -c    # équivalent à date +%H:%M | wc -c
6
polysh$ alias a b; alias b c; alias c a
polysh$ a          # on introduit une boucle infinie
polysh: alias loop or line too long
```

Le traitement complet et correct des alias est un peu délicat. Ne vous lancez pas tout de suite dessus.

Redirections

Polysh permet de rediriger les flux standard d'entrée et de sortie au moyen des notations classiques:

- `<` redirection du fichier standard d'entrée
- `>` redirection du fichier standard de sortie
- `>>` redirection du fichier standard de sortie (mode append)
- `|` pipe

Pour simplifier, on ne traitera pas la redirection de la sortie d'erreur standard

Boucle de l'interprète

L'interprète `polysh` est en fait une boucle infinie qui empile les différents "mots" rencontrés sur une ligne de commande et les range dans un tableau de type `argc/argv`. Quand on tombe sur une redirection, il faut se souvenir du fichier qui est redirigé afin de mettre effectivement en place ces redirections lorsqu'on arrive à la fin de la commande (i.e. sur un fin de ligne, un point virgule, ...). La boucle centrale de l'interprète ne travaille pas directement sur les caractères de la commande, mais sur des entités un peu plus grosses construites par un analyseur lexical (ou encore *tokenizer*). Celui-ci reconnaît des assemblages de caractères et les renvoie comme une unité de type `struct token`. Regarder le fichier `tokenize.h` et les fichiers de test `test-token.c` et `test-token2.c` pour plus de détail.
La construction de la boucle centrale de l'interprète demande d'écrire les choses soigneusement. Ce n'est pas très compliqué, mais il est facile de se retrouver avec du code difficile à gérer, des descripteurs de fichiers non fermés...
Quelques conseils pour réaliser la boucle centrale du shell:

- dans un premier temps ne pas s'occuper des redirections
- ajouter ensuite les commandes internes
- ajouter ensuite les redirections en entrée
- ajouter ensuite les redirections en sortie (mode normal et mode *append*)
- ajouter les pipes en dernier, c'est le plus compliqué

Déroulement du projet

Ce sujet et les fichiers qui l'accompagnent, doivent être considérés comme une **spécification**, c'est-à-dire que votre programme doit implémenter exactement ce qui demandé (en particulier l'ajout inconsidéré de fonctionnalités sera sanctionné). A titre d'exemple et pour fixer les idées, voici le nombre de lignes sources pour les parties qui ne vous sont pas livrées:

```
  81 alias.c
 236 editline.c
 125 environ.c
 121 history.c
 141 intern.c
 154 tokenize.c
 354 toplevel.c
```

soit environ 1200 lignes (commentées). Il n'y a a pas de raison pour que vous ayez des choses fondamentalement différentes. Si c'est beaucoup plus gros, c'est que vous vous y êtes probablement mal pris. N'essayez pas bien sûr de concentrer votre code, au risque de le rendre illisible, pour tenir dans ces chiffres.

Ce qui doit être fait

Il **ne faut pas** implémenter toplevel.c (vous pouvez utiliser celui qui est dans la librarie). Il vous est donné une version partielle de editline. Cette dernière ne gère pas les touches control-K, Control-Y (qui permettent de faire le couper/coller) et les flèches. Il faudra les rajouter au code qui vous est donné.
L'évaluation de votre travail prendra en compte principalement la qualité (et non pas la quantité) du code que vous nous rendrez. Par ailleurs, vous serez interrogé sur loe code que vous rendrez et qu'il faudra savoir expliquer/defendre en détail.