Minishell

- Liens utiles

Sujet: https://cdn.intra.42.fr/pdf/pdf/34930/en.subject.pdf

Manuel POSIX:
https://pubs.opengroup.org/onlinepubs/9699919799/utilities/contents.html

- Cahier des charges

Nom du programme:

minishell

Fonctions externes:

- readline
- rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay
- add_history
- printf
- malloc, free
- write, access, open, read, close
- fork
- wait, waitpid, wait3, wait4,
- signal, sigaction, kill, exit
- getcwd
- chdir
- stat, lstat, fstat
- unlink
- execve
- dup, dup2
- pipe
- opendir, readdir, closedir
- strerror, perror
- isatty, ttyname, ttyslot
- ioctl
- getenv
- tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs

Libft autorisé:

oui

Votre shell doit:

- Pas interpréter des quotes pas fermé ou des caractères spéciaux non spécifiés comme \ ou ;
- Pas utiliser plus d'une variable globale et être prêt à expliquer pourquoi avoir utilisé cette variable globale
- Afficher un prompt lorsqu'une nouvelle commande est attendue
- Avoir un historique fonctionnel
- Rechercher et exécuter le bon exécutable (baser sur la variable PATH ou en utilisant le chemin relatif ou absolu)
- Implémenter les builtins:
    - **echo** avec l'option -n
    - **cd** avec uniquement un chemin relatif ou absolu
    - **pwd** sans options
    - **export** sans options
    - **unset** sans options
    - **env** sans options ni arguments
    - **exit** sans options
- ' empêche toute interprétation d'une séquence de caractère
- " empêche toute interprétation d'une séquence de caractère sauf pour $
- Redirections
    - < doit rediriger l'input
    - > doit rediriger l'output
    - << lit l'input depuis la source en cours jusqu'à ce qu'une ligne contienne uniquement le délimiteur. (Pas besoin de mettre à jour l'historique!)
    - >> doit rediriger l'output en mode "append" (ajoute à la fin du fichier)
- Les Pipes | Les output de chaque commande dans le pipeline sont connectés via un pipe à l'input de la commande d'après
- Les variables d'environnement ($ suivi par des caractères) doivent correspondre (s'étendre) à leur valeurs
- $? doit correspondre (s'étendre) au statut de sortie du plus récent pipeline exécuté au 1er plan
- ctrl-C ctrl-D ctrl-\ doivent fonctionner comme dans bash
- When interactive:
    - ctrl-C écrit un nouvel invite de commande sur une nouvelle ligne
    - ctrl-D quitte le shell
    - ctrl-\ ne fait rien
- La fonction **readline** peut faire des leaks, vous ne devez pas fix ça
- Mais attention votre propre code ne doit pas avoir de leak
- Vous devriez vous limiter à la description du sujet
- Tout ce qui n'est pas demandé n'est pas requis

- Points clés du manuel POSIX

2 Shell Command languages

2.1 Shell Introduction

1. Récupérer la ligne
2. Séparer les tokens et stocker le type de chacun(word, operator)
3. Parse to regroup commands with their arguments
4. Expand les variables d'environnement + $? (last exit status)
5. Appliquer les redirections + supprimer les opérateurs de redirections et les arguments de redirections
6. Exécuter la/les commande(s)
7. Wait optionally for the command to complete its job, then collect exit status

2.3 Token Recognition

/ ! \ La récupération des tokens est différente de celle des here-docs

La récupération du here-doc effectue un readline par ligne

1. Si l'input est valide, délimiter le token courant (?)
2. Pendant la récupération des tokens, si le caractère précédent est un opérateur et que le caractère actuel n'est lié à aucun opérateur, il s'agit d'un nouveau token
3. Relire les étapes de ladite section pour s'orienter au niveau du parsing

/ ! \ Durant la récupération des tokens, il faut garder les quotes et les traiter dans un second temps (expansions y compris)

2.6 Word Expansions

Un contenu entre single quote ne s'expanse pas

En premier lieu, on expand les variables

Ensuite, on retire les quotes

2.7 Redirections

En cas de multi-redirections, on les traite de la première rencontrée à la dernière

Si un file échoue à l'open() ou au create(), la redirection doit échouer

2.7.1 Redirections Input

Implique d'open en O_RDONLY le fichier a droite de l'opérateur, ou sur le stdin si aucun fichier n'est spécifié

2.7.2 Redirections Output

Implique d'open en O_CREATE et O_TRUNC le fichier a droite de l'opérateur. En cas d'absence de fichier destination, ERREUR de parsing

2.7.3 Redirections Output Append

Implique d'open en O_CREATE et O_APPEND le fichier a droite de l'opérateur. En cas d'absence de fichier destination, ERREUR de parsing

2.7.4 Here-Document

/ ! \ Les Here-Docs sont lancés AVANT l'exécution des commandes !!!

Le Here-Doc doit être considéré comme un et un seul mot uni qui commence après le NEWLINE, et continue jusqu'à rencontrer une ligne composée uniquement du délimiteur suivi d'un NEWLINE

Ensuite, le Here-Doc suivant commence si il y en a un

Dans le cas spécifique ou le délimiteur est donné entre quotes, il faut récupérer le délimiteur sans les quotes et SANS l'expand, même entre doubles quotes !!!

2.8 Exit Status and Errors

2.8.2 Exit Status for Commands

Si une commande n'est pas trouvée, exit status(127)

Si la commande est trouvée mais inexécutable, exit status(126)

Si la commande foire pendant son exécution, pour une raison tierce, exit status(1-125)

Creuser la macro WEXITSTATUS

Si la commande est interrompue à cause d'un signal, exit status(128 + N) N étant le numéro du signal

2.9 Shell Commands

Si une command line résulte en un nom de commande et, optionnellement, des arguments, les actions suivantes sont effectuées :

1. Si le nom de la commande ne contient aucun Slash :
    a. Si c'est un builtin, celui-ci est appelé
    b. Sinon on cherche dans le PATH
        i. Si la recherche est un succès, on exécute la commande
        ii. Sinon, exit status(127) + message d'ERREUR sur stderr
2. Si le nom de la commande contient AU MOINS 1 Slash, alors on essaye d'exécuter la commande tel quel.
    a. Si la commande n'existe pas, ERREUR + exit status(127).
    b. Si la commande n'est pas exécutable, ERREUR + exit status(126)

    Si la commande est lancée avec un ou plusieurs des fd 0, 1, ou 2 fermes, what do we do (?)

2.9.2 Pipelines

On fait D'ABORD les pipelines, et ENSUITE les redirections si il y en a

Le stdin, le stdout ou les deux d'une commande doivent être considéré comme être assigné par le pipeline avant toute redirection spécifiée par les opérateurs de redirections qui font partie de la commande.

2.12 Shell Execution Environment

- Ouvertures de tous les fichiers concernés par la command line
- Mise en place du working directory avec `cd` (?)
- Mise en place des trap (signaux) (Ctrl-C, Ctrl-D, Ctrl-\)
- Envoi des variables exportées à l'env

2.14 Special Built-in Utilities

L'output des builtins doit être écrit sur le stdout (s'il y a un output) et il doit être possible de le rediriger ou de le pipe avec n'importe quelle commande.
Le terme builtins signifie que le shell n'a pas besoin de chercher le chemin de la commande.
Il y a une différence entre les builtins et les special builtins qui sont présenté dans cette section:

- Un builtin dit Special est un builtin qui, s'il fail peut faire planter le shell exécutant. On en déduit qu'un tel builtin est exécuté dans le même process que le shell.
- A contrario, un builtin dit Simple ne peut pas faire planter le shell exécutant s'il fail.

Les special builtins dans la section ne doivent pas pouvoir être exécuté par une fonction de la famille exec() (POSIX.1-2017)

Ce que je comprends c'est qu'un builtin (en tout cas pour nous) va limite appeler une fonction plutôt qu'un programme. Et donc on va devoir les spoter en tant que builtin dans le parsing.

- exit :
  - Usage : exit [n]
  - Quitte le shell actuel avec l'exit status(n), n étant un unsigned int.
    Si le shell actuel est un sous-shell, on retourne dans le shell parent avec l'exit status(n), puis on continue l'exécution du parent
    Si l'exit status n'est pas entre 0 et 255 inclus, le exit status est undefined
  - Exit Status :
    - Si n est spécifié, l'exit status doit être n, sauf si n n'est pas un uint8_t.
    - Si n n'est pas spécifié, la valeur de exit doit être l'exit status de la dernière commande exécutée, ou zéro si aucune commande n'a été exécutée
    - Si exit est utilisé dans un trap, n prend la valeur de l'exit status de la dernière commande exécutée avant le trap
  - Rationale :
    - Comme explique precedemment, certains exit status sont réservés à des cas de figure bien définis :
      - 126 - Un fichier qui a été trouvé n'est pas exécutable
      - 127 - L'utilitaire a exécuté est introuvable
      - 128+ - Une commande a été interrompue par un signal
    - En cas de mauvais Usage, le comportement de exit() est undefined. Une valeur plus grande que 255 doit être tronquée. La valeur tronquée ne doit pas valoir zéro (ou, si on est en mode interactif, et que l'erreur ne quitte pas le shell, on doit stocker la valeur tronquée dans $?)
      - Tip troncage : Tant Que N > 255 Faire un bitshift de 7 sur la droite
- export :
  - Usage : export name=word ...
  - Déclare une variable `name` dans l'env du shell actuel, et y assigne la valeur `word`.
- unset :
  - Usage : unset name ...
  - Si name est une variable de l'env, elle est retirée entièrement de l'env
  - Si name n'est pas une variable de l'env, rien ne se passe
  - Exit Status :
    - Si toutes les variables ont été unset correctement, exit status(0)
    - Si au moins un unset a fail, exit status(>0)

4 Utilities

- cd
    - Usage : cd [directory]
    - Change le WorkingDirectory de l'environnement d'exécution du shell actuel de la façon suivante :
        1. Si aucun argument n'est donné et que HOME n'existe pas ou est vide, alors rien ne se passe
        2. Si aucun argument n'est donné et que HOME est non-vide, alors directory prend la valeur HOME
        3. Si directory commence par un '/', curpath prend la valeur de l'operand, et on passe à la step 7
        4. Si directory commence par '.' ou par '..', on passe à la step 6
        5. On parcourt les chemins contenus dans la variable d'environnement CDPATH si il est non-nul, on teste si le chemin absolu résultant de la concaténation du chemin de CDPATH actuellement testé et de directory existe.
           Note : Si le chemin actuellement testé ne contient pas slash a la fin, on doit le rajouter manuellement avant de concatener
           Si un chemin de CDPATH est nul, on teste './directory'
           Au premier chemin absolu obtenu existant, curpath prend la valeur du dit chemin absolu, et on passe à la step 7
        6. Set curpath to the directory operand
        7. Si le curpath ne commence pas par un '/', curpath prend la valeur de PWD + [/] + curpath.
        8. The **curpath** value shall then be converted to canonical form as follows, considering each component from beginning to end, in sequence:
            a. Dot components and any <slash> characters that separate them from the next component shall be deleted.
            b. For each dot-dot component, if there is a preceding component and it is neither root nor dot-dot, then:
                i. If the preceding component does not refer (in the context of pathname resolution with symbolic links followed) to a directory, then the *cd* utility shall display an appropriate error message and no further steps shall be taken.
                ii. The preceding component, all <slash> characters separating the preceding component from dot-dot, dot-dot, and all <slash> characters separating dot-dot from the following component (if any) shall be deleted.
            c. An implementation may further simplify **curpath** by removing any trailing <slash> characters that are not also leading <slash> characters, replacing multiple non-leading consecutive <slash> characters with a single <slash>, and replacing three

or more leading <slash> characters with a single <slash>. If, as a result of this canonicalization, the **curpath** variable is null, no further steps shall be taken.

9. Si jamais directory ne dépasse pas PATH_MAX, mais que curpath le dépasse, lui, alors on doit essayer de trouver un équivalent de ce chemin absolu en chemin relatif. Une telle conversion n'est considérée possible que si curpath contient PWD
10. On appelle la fonction chdir() en envoyant curpath en argument. Si pour x ou y raison cet appel échoue, cd doit afficher un message d'erreur approprié. Si cd a réussi, PWD prend la valeur de curpath avant conversion si il y en a eu une (pas de chemin relatif, uniquement des chemins absolus). Si il y a eu un problème de permission pendant l'exécution, le comportement n'est pas spécifié.

Si l'exécution a mené à une mise à jour de la variable d'environnement PWD, la variable d'environnement OLDPWD doit également être mise à jour

Note : Si directory est une chaîne vide, le comportement n'est pas spécifié.

    - Operand
        - Soit c'est un chemin, absolu ou relatif, qui deviendra le prochain WorkingDirectory. L'interprétation du chemin relatif dépend du CDPATH et de PWD.
        - Si c'est le caractère '-', cela revient à faire `cd "$OLDPWD" && pwd`
        - Si le changement de dossier s'effectue à partir d'un chemin contenu dans CDPATH, il faut aussi effectuer un `pwd` après déplacement
- echo
    - Usage : echo [string...]
    - Description : echo écrit ses arguments sur stdout, suivi par un <newline>. S'il n'y a pas d'arguments, uniquement le <newline> est écrit.
    - Options : -n : n'écrit pas le <newline> final.
- env
    - Écrit les variables d'environnement avec une *name=value* paire par ligne. (Chaque ligne finit par un <newline>).
- pwd
    - Écrit sur stdout un pathname absolu de l'actuel working directory qui ne contient pas '.' ou '..' et qui se finit par un <newline>.

- Notes

La variable $? doit être expand elle aussi au parsing.

Elle ne fait pas partie de l’env.

Il faudra utiliser la variable globale pour l’exit status.

- Mémo des cas tordus

CDPATH

Command

```
$> export CDPATH=/
$> cd /home
$> ls bin
ls: cannot access 'bin': No such file or directory
$> cd bin
```

Expected

```
/bin
$> pwd
/bin
```

Empty output redirection to file

Command

```
$> rm -f pouic
$> >pouic
$> ls -l pouic
```

Expected

```
-rw-r--r-- 1 jodufour user42 0  3 déc.  16:19 pouic
```

Broken pipe

Command

```
$> rm -f pouic
$> echo koala >pouic |
```

Expected

```
Any handle but no crash
```

Redirection priority

Command

```
$> rm -f pouic
$> rm pouic > pouic
$> ls -l pouic
```

Expected

```
ls: cannot access 'pouic': No such file or directory
```

Empty output redirection to pipe to file

Command

```
$> rm -f pouic
$> >|pouic
```

Expected

```
bash: syntax error near unexpected token `|'
```

Quand j'exécute `> | file` j'ai un message d'erreur et le fichier n'est pas créé

Token merging

Command-0

```
$> echo koala > pouic
$> cat"-e" pouic
bash: cat-e: command not found
```

Expected-0

```
bash: cat-e: command not found
```

Command-1

```
$> e'c'"h"o koala>pouic
$> cat 'po'"u"ic
```

Expected-1

```
koala
```

Options are considered as arguments

Command

```
$> export OPT=-n
$> "echo $OPT" tutu
```

Expected

```
bash: echo -n: command not found
```

Consecutive pipes

Command

```
$> echo koala|||cat
```

Expected

```
bash: syntax error near unexpected token `|'
```

Forcing consider arguments AND stdin

Command

```
$> echo lolilol > pouic
$> echo tutu > poneyvif
$> cat < pouic poneyvif -
```

Expected

```
tutu
lolilol
```

Expanding unseparated env variables

Command

```
$> export TUTU=koala FOO=google
$> echo $TUTU$FOO
```

Expected

```
koalagoogle
```

Expanding the right variables

Command

```
$> export A=1 B=2 C=3 D=4 E=5 F=6 G=7 H=8
$> echo "$A'$B"'$C"$D'$E'"$F"'"'$G'$H"
```

Expected

```
1'2$C"$D5"$F"'7'8
```

No expanding when $ has no valid name behind

Command-0

```
$> echo " $ " | cat -e
```

Expected-0

```
 $ $
```

Command-1

```
$> echo $:$= | cat -e
```

Expected-1

```
$:$=$
```

Remove $ when last character of an unquoted portion followed by a quoted portion

Command

```
$> echo foo$"HOME"
```

Expected

```
fooHOME
```

Detect commands that are placed AFTER the redirections

Command

```
$> echo koala>pouic
$> <pouic cat
koala
```

Expected

```
koala
```

Detect quotes that result of an expanding

Command

```
$> export FOO='"'
$> echo " $FOO " | cat -e
```

Expected

```
 " $
```

Multiple commands execution

Command

```
$> echo koala > pouic
echo titi > foo
echo tutu > bar
cat pouic foo bar
```

Expected

```
koala
titi
tutu
```

Split one token into N + 1 tokens when an unquoted expand contains N spaces

Command

```

$> mkdir dir_test
$> cd dir_test
$> touch tutu
$> echo koala > titi
$> export FOO='s -ls'
$> l$FOO
```

Expected

```
total 12
drwxr-xr-x  2 jodufour user42 4096  9 déc.  23:43 .
drwxr-xr-x 10 jodufour user42 4096  9 déc.  23:42 ..
-rw-r--r--  1 jodufour user42 6  9 déc.  23:43 titi
-rw-r--r--  1 jodufour user42 0  9 déc.  23:42 tutu
```

Do not expand here-doc’s delimiter

Consider only the last output redirection as effective

Command

```
$> rm -f out0 out1
$> echo tutu>out0>out1
$> ls -l out0 out1
```

Expected

```
-rw-r--r-- 1 jodufour user42 0 15 déc.  20:52 out0
-rw-r--r-- 1 jodufour user42 5 15 déc.  20:52 out1
```

Do not exit the shell if the exit command is piped or piping

Command-0

```
$> echo koala > pouic
$> exit | cat -e pouic
```

Expected-0

```
koala$
```

Command-1

```
$> echo koala > pouic
$> cat -e pouic | exit
```

Expected-1

```
(nothing)
```

Input redirections override pipe redirections

Command

```
$> echo koala>foo
$> echo pouic>bar
$> cat foo|cat<bar
```

Expected

```
pouic
```

Expand here-doc content ONLY when delimiter is unquoted

Command-0

```
$> export TUTU='hello world'
$> cat << stop -e
> koala
> '$TUTU'
> $TUTU
> "$TUTU"
> stop
```

Expected-0

```
koala$
'hello world'$
hello world$
"hello world"$
```

Command-1

```
$> export TUTU='this is me'
$> cat << "eof" -e
> milou
> '$TUTU'
> $TUTU
> "$TUTU"
> eof
```

Expected-1

```
milou$
'$TUTU'$
$TUTU$
"$TUTU"$
```