



MINISHELL

Loïc BINE & Jean STILL

ITII - CNAM 16/04/2018

<https://github.com/LePingWin/MiniSHeLL>



Table des matières

.....	0
Objectifs	1
Fonctionnalités	1
Choix Technique	2
Deux modes de fonctionnement.....	2
Parsing.....	2
Construction de l'arbre	3
Exécution de l'arbre	3
GCOV	4

Objectifs

L'objectif de ce projet est de reproduire le fonctionnement d'un Shell Unix à moindre échelle

Fonctionnalités

- **FM01** – Le binaire est capable d'exécuter une commande simple (ie : ls -l ; ps ; who) ✓
- **FM02** – Le binaire est capable d'exécuter un sous-ensemble de plusieurs commandes de sorte à prendre en compte : ✓
 - Les opérateurs de contrôle : && et ||
 - Les redirections de flux simples : |, >, <, >>, <<
 - L'exécution en arrière-plan : &
- **FM03** – L'exécution des commandes internes (fonctionnalités built-in) suivantes : ✓
 - cd - Permettant de se déplacer au sein d'une arborescence de fichier.
 - pwd – Affichant la valeur de la variable contenant le chemin du répertoire courant.
 - exit – Permettant de quitter l'interpréteur.
 - echo – Permettant d'afficher du texte sur la sortie standard.
- **FM04** - La persistance des commandes saisie dans un fichier (historique) ✓

D'autres fonctionnalités optionnelles peuvent être implémentés :

- **FO01** – La réalisation d'un mode batch (ie : ./my_shell -c « ls -al | grep toto ») ✓
- **FO02** – La création de variables d'environnement ✗
- **FO03** – La prise en charge d'alias ✗

Concernant les exigences techniques attendues, vous devez respecter les contraintes suivantes :

- CT01 – La compilation du projet doit se faire via un Makefile. ✓
- CT02 – La définitions des structures doit se faire dans un fichier typedef.h. ✓
- CT03 – La définition des méthodes prototype (.h) & implémentation (.c) doit se faire de manière séparée autant que faire se peut. ✓
- CT04 – Le code produit doit être documenté. ✓
- CT05 – La gestion des erreurs doit se faire via « les mécanismes proposés par errno ». ✓

D'autres contraintes techniques peuvent être prises en compte :

- CTO01 – La documentation du code générée via l'utilitaire doxygen. ✓
- CTO02 – Le code est soumis à un contrôle de couverture via l'utilitaire gcov. ✓
- CTO03 – Une page de manuel Linux est rédigée pour détailler l'exécution du shell. ✓

Choix Technique

Deux modes de fonctionnement

Le programme est prévu pour exécuter le programme de deux façons différentes.

Deux modes :

- Mode standard :

```
./bin/miniSHell  
/home/jean/Documents/shellProject/MiniSHell$ ls -al - grep b && echo ok  
/home/jean/Documents/shellProject/MiniSHell$ exit
```

- Mode Batch : argument -c [Commande(s)]

```
./bin/miniSHell -c "ls -al | grep b && echo ok || echo nok >> cat.txt"
```

Cela permet d'exécuter les commandes depuis l'appel du programme

Parsing

Le Parsing est réalisé en plusieurs phases :

- Séparation des sous-chaines de commandes délimitées par « ; »
- Boucle sur chaque sous-commande
 - Recherche du « & » et suppression de celui-ci dans la chaîne
 - Affectation de la valeur « true » au booléen de traitement en arrière-plan
 - Séparation de la chaîne par espaces « »
 - Parcours du tableau résultant de l'étape précédente et création d'un tableau séparant commandes et opérateurs
 - Si pas d'opérateur, on concatène la chaîne dans la case du tableau à retourner

- Si un/des opérateur(s) reconnu(s) (via REGEX) : &&,| |,<,<,>,>>
 - Incrément de l'index courant
 - Stockage dans la case actuelle du tableau à retourner
 - Incrément de l'index courant
- Le simple Pipe : |, n'est pas considéré comme un opérateur dans notre programme. Il reste donc dans la commande et n'est pas traité par le Parsing.

Construction de l'arbre

On se base sur l'algorithme Shunting-yard pour construire directement un arbre binaire : https://www.wikiwand.com/fr/Algorithme_Shunting-yard

Durant cette phase on se sert des structures suivantes :

- Pile<BinaryTree> : Stocke temporairement tout ce qui n'est pas opérateur
- Pile<String> : Stocke temporairement les opérateurs : &&,| |,<,<,>,>>
- BinaryTree<String> : Arbre binaire permettant le parcours et l'exécution des différentes commandes

A la fin de la séparation en pile, on finit par dépiler les deux stacks jusqu'à qu'elles soient vides et que l'arbre soit construit.

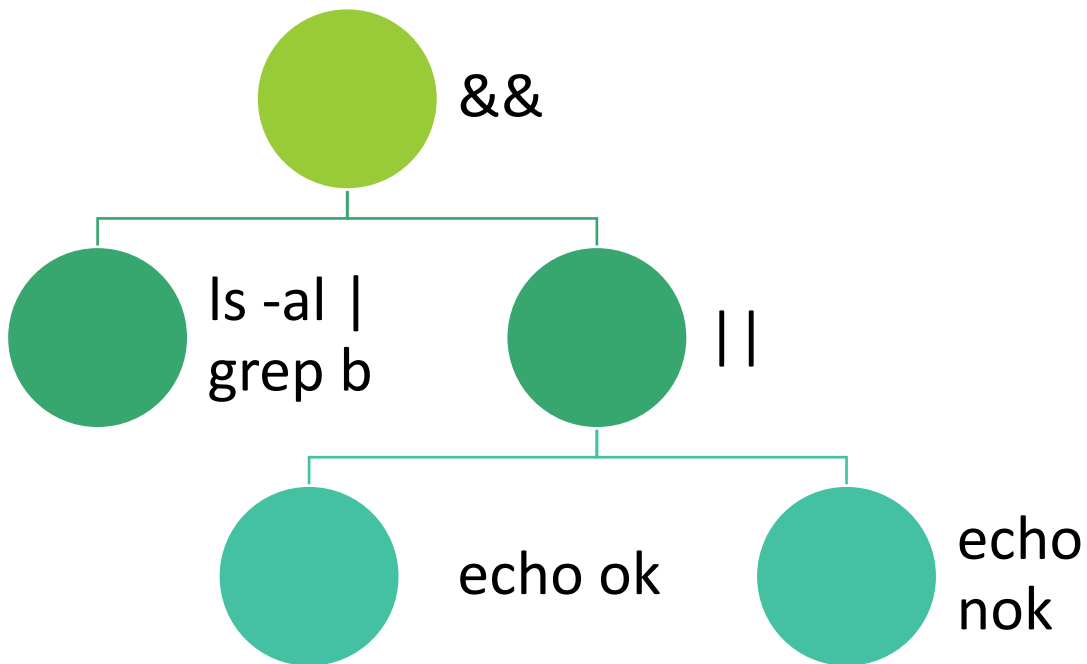
Exécution de l'arbre

On effectue un parcours préfixé (gauche-droite) de l'arbre et l'on effectue un traitement différent en fonction de la chaîne récupérée :

- Si opérateur, on effectue un traitement récursif
- Si pas opérateur, on traite la commande directement et on renvoie un booléen témoignant de la bonne exécution de la commande

La structure de l'arbre pour cette commande ressemble à ceci pour la commande suivante :

```
ls -al | grep b && echo ok || echo nok
```



Nous avons fait le choix de mettre les opérateurs (hors simple Pipe) au niveau des racines des arbres et constituer les feuilles avec les commandes.

GCOV

Nous exécutons gcov grâce à la commande suivante :

```
make gcov
```

Cela va lancer une première fois le miniShell en mode standard.

Nous lui renseignons les commandes suivantes :

```
/home/jean/Documents/shellProject/MiniShell$ ls ; ls  
/home/jean/Documents/shellProject/MiniShell$ exit
```

Cela va lancer une seconde fois le miniShell en mode Batch avec la commande suivante afin de tester un maximum nos fonctionnalités :

```
ls | grep sdfsd && echo ok1 && echo ok2 || echo nok | pwd > cat.txt && ls -al  
>> cat.txt ; cat < cat.txt ; cd .. ; cd MiniShell ; wc << ; cd sdfsd ; ls &
```

Nous obtenons la couverture de code suivante :

LCOV - code coverage report

Current view: [top level](#) - src








Test: [report.info](#)

Date: 2018-04-16 14:49:18

Lines:

Hit	Total	Coverage
375	402	93.3 %
38	38	100.0 %

Functions:

Filename	Line Coverage ↕			Functions ↕	
builtin.c		100.0 %	12 / 12	100.0 %	3 / 3
main.c		100.0 %	26 / 26	100.0 %	2 / 2
parser.c		94.0 %	47 / 50	100.0 %	5 / 5
shell.c		93.2 %	221 / 237	100.0 %	12 / 12
stack.c		95.0 %	19 / 20	100.0 %	4 / 4
stackTree.c		91.7 %	22 / 24	100.0 %	5 / 5
tree.c		84.8 %	28 / 33	100.0 %	7 / 7

Generated by: [LCOV version 1.12](#)