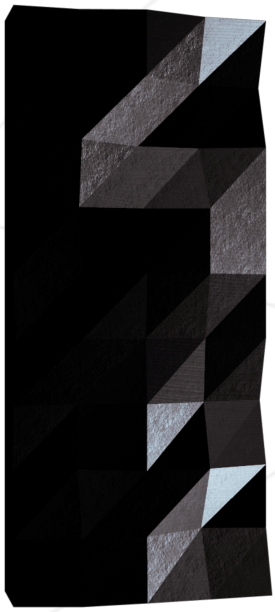

Minishell

Aussi mignon qu'un vrai shell

Summary: L'objectif de ce projet est de créer un simple shell. Ca sera votre propre petit bash, ou zsh. Vous en apprendrez beaucoup sur les process et les file descriptors.

Contents

Chapter I

Introduction

L'existence des shells est intrinsèquement liée à l'existence de l'informatique.
À l'époque, les developpeurs étaient tous d'accord pour dire que `communiquer avec un ordinateur en utilisant des interrupteurs 1/0 était fortement irritant`.
La suite logique a été d'inventer un moyen de communiquer via des lignes de commandes interactives dans un langage jusqu'à un certain point proche de l'anglais.

Avec `Minishell`, vous allez voyager dans le temps et rencontrer les problèmes que les gens croisaient alors que `Windows` n'existait pas encore.

Chapter II

Règles communes

- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans une fichier `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapter III

Partie Obligatoire

Program name	`minishell`
Turn in files	
Makefile	`Yes`
Arguments	
External functs.	`printf, malloc, free, write, open, read, close, fork, wait, waitpid, wait3, wait4, signal, kill, exit, getcwd, chdir, stat, lstat, fstat, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, errno, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs`
Libft authorized	`Yes`
Description	`Écrivez un shell`

Votre shell doit :

- Ne pas utiliser plus d'une variable globale et vous devrez justifier son utilisation.
- Afficher un prompt en l'attente d'une nouvelle commande
- Chercher et lancer le bon executable (basé sur une variable d'environnement PATH ou en utilisant un `path` absolu), comme dans bash
- Vous devez implémenter les builtins suivants :
  - `echo` et l'option '-n'
  - `cd` uniquement avec un chemin absolu ou relatif
  - `pwd` sans aucune option
  - `export` sans aucune option
  - `unset` sans aucune option
  - `env` sans aucune option ni argument
  - `exit` sans aucune option

- `;` dans la ligne de commande doit séparer les commandes
- `'` et `"` doivent marcher comme dans bash, à l'exception du multiligne.
- Les redirections `<`, `>` et "`>>`" doivent marcher comme dans bash, à l'exception des aggrégations de fd
- Pipes `|` doivent marcher comme dans bash à l'exception du multiligne.
- Les variables d'environnement (`$` suivi de caractères) doivent marcher comme dans bash.
- `$?` doit marcher comme dans bash
- `ctrl-C`, `ctrl-D` et `ctrl-\` doivent afficher le même résultat que dans bash.
- en utilisant termcap (`man tgetent` pour voir des exemples) Touches haut et bas pour naviguer dans l'historique de commandes, que nous pouvons ensuite éditer (inline, pas dans l'historique)

Chapter IV

Partie Bonus

- Si la partie obligatoire n'est pas complète, vous n'avez pas accès aux bonus
- Vous n'avez pas a faire tous les bonus
- Redirection "`<<`" comme dans bash
- Historique et édition de ligne avance en utilisant termcaps (`man tgetent` pour voir des exemples)
    - Éditer la ligne ou le curseur est placé
    - Déplacer le curseur vers la gauche ou la droite pour éditer la ligne à un endroit spécifique. Évidemment, les caractères ajoutés doivent être rajoutés au milieu de ceux existants, comme dans bash.
    - Copier, couper et coller tout ou une partie d'une ligne à l'aide d'une séquence de touches de votre choix
    - Se déplacer de mot en mot à l'aide des touches `CTRL+gauche et CTRL+droite`
    - Aller directement au début ou à la fin de la ligne en utilisant les touches `home` et `end`
    - Écrire ET éditer une commande sur plusieurs lignes. Dans cette situation, nous adorerions que `CTRL+haut` et `CTRL + bas` permettent de se déplacer entre les lignes tout en restant dans la même colonne, ou la plus appropriée.
- `&&` et `||` avec les parenthèses pour les priorités, comme dans bash
- wildcard `*` comme dans bash