Chapitre II

Introduction

Dans toute scolarité, il éxiste des moments qui marquent les étudiants pour toujours. Le `42sh` est l'un de ces moments. Réaliser ce projet, c'est marquer une étape importante à `42`.

Il s'agit ici d'écrire un shell `UNIX` le plus stable et le plus complet possible. Vous connaissez déja de nombreux shells et chacun possède ses propres caractéristiques, du humble `sh` présent sur toutes les distributions `UNIX` du monde, au très complet et très complexe `zsh` que vous êtes si nombreux à utiliser sans savoir pourquoi. Bien sur, il existe de nombreux autres shells, comme `bash`, `csh`, `tcsh`, `ksh`, `ash`, etc. Etant donné que `42sh` sera votre premier vrai shell, il est fréquent parmis les étudiants de vouloir choisir un shell de référence pour essayer d'en reproduire le comportement. C'est une bonne idée, mais à condition de choisir votre shell de référence en connaissance de cause. En effet, ceux d'entre vous qui choisiront `zsh` comme shell de référence s'engageront dans une quête longue et difficile, bien que très riche en enseignements. Par exemple, ils pourront y apprendre l'humilité et la notion de quantité de travail colossale.

La meilleure façon d'aborder la question du shell de référence consiste tout simplement à en essayer plusieurs et à se faire une idée de leurs différences, souvent subtiles, parfois tordues. Toutefois ne perdez pas de vue que si `sh` est souvent considéré comme le shell le plus "simple", le recoder complètement de manière stable est une réussite bien plus intéressante qu'un shell qui promet monts et merveilles mais qui se révèle incapable de faire plus que quelques pipes et redirections.

Le maître mot içi est "stabilité". Un `42sh` humble mais indestructible vaudra toujours plus qu'un `42sh` proposant toutes les options imaginables mais qui plante dans un cas imprévu, puisque ce dernier vaudra 0. Assurez-vous donc de rendre un `42sh` stable, je n'insisterai jamais assez sur ce point.

Le projet est composé de 2 parties décrites ci-dessous et est légèrement différent des autres sujets auxquels vous êtes habitués.

- Une partie **obligatoire**, à réaliser impérativement. Ce sont les pré-requis minimum d'un shell.
- Une partie **optionelle**, qui ne sera prise en considération que si la partie obligatoire fonctionne dans son intégralité.

La partie obligatoire, seule, ne permet pas de valider le projet - il vous faudra choisir et implémenter plusieurs points de la partie optionnelle pour valider. Ceci pour deux raisons : la première est que la partie obligatoire représente un shell extrêmment basique ; la seconde, c'est pour vous obliger à selectionner des fonctionnalités de la partie optionnelle et à vous fixer vos propres objectifs dans la réalisation de ce projet.

Un dernier point, l'utilisabilité de votre `42sh` sera largement prise en compte. Il serait judicieux de vous assurer qu'un utilisateur d'un shell ordinaire comme `sh` ou `bash` soit capable d'utiliser votre `42sh` de manière intuitive ...

Chapitre III

Partie obligatoire

- Une acquisition de ligne minimale.
  - Affichage d'un prompt.
  - lecture de la ligne de commande, sans édition de ligne.
  - Gestion correcte des espaces et des tabulations.

- Les builtins suivantes avec toutes leurs options si elles en ont (au strict minimum les options dictées par le standard POSIX) :
  - `cd`
  - `echo`
  - `exit`
  - `env`
  - `setenv`
  - `unsetenv`

- Exécution de commandes simples avec leurs paramètres et gestion du `PATH`.
- Gestion des erreurs et de la valeur de retour des commandes
- Les opérateurs de redirection suivants : `">"`, `">>"`, `"<"` et `"|"`.
- Les opérateurs logiques `"&&"` et `"||"`.
- Le séparateur `";"`.

Chapitre IV

Partie optionelle

Le shell demandé dans la partie obligatoire représente le strict minimum de ce qu'on attend d'un shell fonctionnel. Il va maintenant vous falloir choisir et implémenter des features un peu plus avancées. Un minimum de **5 features** de la liste de features optionnelles ci-dessous est requis pour valider le projet.

Gardez toutefois à l'esprit que la stabilité sera beaucoup plus importante que la quantité. N'incluez pas une option qui pose un problème au reste du programme par exemple. Et n'oubliez pas que cette partie ne sera évaluée que si la partie obligatoire est complète et indestructible.

Les features optionnelles :

- Les inhibiteurs `"""` (double quote), `"'"` (simple quote) et `"\"` (backslash).
- Les redirections avancées : l'aggrégation des sorties de fichier et le heredoc `"<<"`.
- Le globing : `"*"`, `"?"`, `"[]"`, `"{}"`, etc. (sans utiliser la fonction `glob(3)` !)
- Les back quotes `"`"`.
- Les sous shells avec les operateurs `"()"`.
- les variables locales et les builtin `unset` et `export`.
- L'historique des commandes et les builtins `history` et `"!"` avec toutes leurs options si elles en ont.
- Edition de ligne telle que demandée dans le `ft_sh3`.
- Les descripteurs de fichiers et la builtin `read` avec toutes ses options.
- Complétion dynamique.

Et en bonus très appréciés par le barème :

- Le Job control et les builtins `job`, `fg` et `bg`, et l'operateur `"&"`.
- Le shell script.

En cas de doute sur le comportement d'une fonctionnalité, référez-vous au standard POSIX. Il est tout à fait acceptable de choisir d'implémenter différemment une fonctionnalité, si ça vous semble cohérent pour votre shell, mais il n'est pas acceptable de faire moins que les specs POSIX par "flemme".

Chapitre V

Consignes

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- Votre exécutable doit s'appeler `42sh`
- Vous devez rendre un Makefile avec toutes les règles usuelles.
- Si vous souhaitez utiliser votre `libft`, vous devez en rendre les sources dans un dossier nommé `libft` à la racine de votre dépot, avec un Makefile pour la compiler. Aucune version déja compilée de votre biliothque ne sera accepté en soutenance. Le `Makefile` de votre `42sh` doit bien entendu compiler et linker avec votre `libft`.
- Votre projet doit être à la Norme.
- Vous devez gérer les erreurs de façon pertinente. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...).
- Votre terminal ne doit jamais afficher n'importe quoi, gérez intelligemment ses settings.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant vos logins, à raison d'un par ligne, de cette façon :

```
$>cat -e auteur
xlogin$
ylogin$
zlogin$
alogin$
$>
```

- Vous avez le droit d'utiliser les fonctions suivantes :
  - Toute la section 2 des mans
  - Les fonctions autorisées du minishell
  - Toutes les fonctions de la bibliothèque `termcaps`.
- Vous pouvez poser vos questions sur le forum, sur jabber, IRC, ...
- Bon courage à tous !