Rapport - Kick off 42sh

Antoine Vallee - Chef
Xavier Gerondeau
Amaury Pruvot
Guillaume Moizan
Emmanuel Guet

Table des matières

Chapitre 1

Présentation du groupe

1.1 Antoine 'Nervous' VALLEE - vallee_a - Chef

Passionné d'informatique et de management, être chef de groupe sur un projet tel que 42sh est un immense plaisir. Composé d'un groupe d'amis de longue date, notre équipe était déjà prête depuis longtemps. J'avais personnellement hâte de débuter le 42sh, et je ferai mon maximum pour que ce projet soit une réussite pour le groupe, tant sur un plan humain que technique bien sûr. Ce projet permet la concrêtisation de toutes les notions de C - Unix apprises jusqu'à présent, ainsi que l'apprentissage à la gestion d'un projet plus conséquent qu'habituellement.

1.2 Xavier 'Courgette' GERONDEAU - gerond_x

Epitéen depuis la sup, 42sh est un projet dont j'ai longtemps entendu parler de la part des acdc ou des aoc. C'est donc avec plaisir que nous nous appretons à realiser ce projet qui est le premier vrai projet en equipe et sur une plus longue durée. les deadlines qui nous sont imposées face à la complexité du projet est impressionnant et est un veritable defi. Il s'agit aussi de montrer que de bonnes bases et des connaissances ont été acquises durant la piscine et les nombreux projets ayant suivis.

1.3 Emmanuel 'Green' GUET - guet_e

L'ING1, un départ sur les chapeaux de roues avec les deux semaines de piscine qui m'ont principalement servies à acquérir une rigueur que je n'avais

pas avant.

Les divers mini projets qui ont suivi n'ont fait qu'agrandir cette rigeur en plus de me donner un début de bagage technique de connaissances vraiment intéressant.

Enfin, le 42SH. Ce projet est une tradition à EPITA et marque la fin de la partie C de l'année. Il s'agit d'un projet difficile et très important, mais surtout, vraiment intéressant, et le premier projet d'ING1

qui s'effectue sur une période de plus d'une semaine avec une équipe de 5 personnes. Le 42SH s'annonce réellement enrichissant, et nous espérons bien évidemment aussi bien étendre nos connaissances qu'obtenir une très bonne note !

1.4 Guillaume 'Baguette' MOIZAN - moizan_g

Je suis étudiant à Epita depuis plus de deux ans, j'y i notamment fait parti des associations Lateb et Cast. Le projet 42sh est quelque chose que j'appréhende mais qui suscite également chez moi un intérêt certain. Ce sera pour moi l'occasion de développer de nouvelles connaissances sur de nombreux sujets. Je n'ai encore jamais eu l'occasion de travailler avec un groupe comportant autant de membres sur un projet de ce type. La quantité de travail requise dans un temps aussi réduit avec un équipe de 5 personnes est un défi pour lequel je porte beaucoup d'intérêt.

1.5 Amaury 'Chaf' PRUVOT - pruvot_a

J'ai effectué un an dans une classe prépa classique avant d'intégrer EPITA. J'ai pu découvrir plusieurs langages de programmation différents au cours de mon apprentissage. 42sh est un projet très complet qui va me permettre de rassembler toutes mes connaissances autour d'un travail de grande envergure. Mon objectif dans pour le déroulement de ce projet est de confirmer mes compétences dans le langage C avant de pouvoir passer à un autre. En effet, 42sh va faire appel à l'ensemble des notions vues et apprises lors de la piscine C-Unix et des projets qui l'ont suivie. Je pense être doté de qualités essentielles au bon déroulement du projet ainsi qu'à la bonne cohésion de l'équipe, je suis posé, réfléchi et j'ai une bonne capacité à prendre du recul sur une situation ce qui me permettra d'avancer efficacement dans la confection de notre Shell Unix.

Chapitre 2

Le Projet

Le projet 42sh est un projet réalisé en groupe de 5 étudiants de première année d'ingénieur à l'EPITA. Ce projet consiste à réaliser un Shell Unix répondant aux spécifications de la SUS (Single UNIX Specification) et plus précisément à la partie concernant le langage de command shell (Shell Command Language).

Le but de ce projet est de confectionner un shell fonctionnel et ainsi de découvrir les mécanismes cachés derrière cet outil que nous utilisons tous les jours mais qui restent cependant très opaques.

42sh, va nous demander dans un premier temps d'étudier la documentation du SCL (Shell Command Language) afin de comprendre et d'assimiler les mécanismes inhérants au bon fonctionnement d'un shell. Nous devrons ensuite implémenter un analyseur syntaxique et son analyseur lexical pour découper et traiter les commandes envoyées par l'utilisateurs puis enfin les exécuter. Certaines notions dans ces différentes étapes ont déjà été traitées dans des projets précédents (minimake, myHTTPd, ...) et seront donc plus aisées à établir bien qu'elles restent tout de même des étapes décisives.

L'interêt de ce projet est multiple. En effet, 42sh va nous permettre de mieux comprendre le comportement des shells que nous utilisons chaque jour (zsh, sh, bash, ...) et d'autre part il va nous permettre de réutiliser des méthodes déjà connues et d'en apprendre de nouvelles dans la mise en oeuvre et la réalisation de notre shell. De plus, ce projet sera le premier projet d'envergure qui va nous être donné de réaliser à EPITA. Ainsi, nous allons devoir nous confronter à de nouveaux problèmes d'ordre managérial et nous allons devoir gérer notre groupe de façon à éviter les conflits et travailler le plus efficacement possible.

Chapitre 3

Découpage du projet

3.1 L'analyseur lexical (lexer)

L'analyseur lexical de 42sh est l'entité chargée de découper les différentes lignes de commandes envoyées au shell par l'utilisateur. Son rôle consiste à éliminer les parasites de texte source (commentaires, espaces, tabulations, ...) et à reconnaitre les mots clés et opérateurs définis dans la grammaire reconnue ainsi que le chaînes de charactère, identifiants et constantes numériques données par l'utilisateur.

3.2 L'analyseur syntaxique

Base du 42sh, ce dernier sera basé sur une structure arborescente. L'analyseur syntaxique devra être développé rigoureusement afin d'obtenir une base solide pour notre projet. Nous effectuerons bien sûr un parser LL (top-down), reposant sur l'analyse descendante d'une grammaire. Utilisé pour analyser la grammaire du shell, il sera en lien direct avec l'analyseur lexical, qui lui enverra les tokens à analyser.

Ainsi, l'objectif de l'analyseur est de produire un arbre syntaxique abstrait (AST) à partir de la grammaire donnée. Ce dernier sera d'ailleurs affiché par un module développé lors de la version 0.5.

Ci dessous un exemple d'AST pour une expression arithmétique :

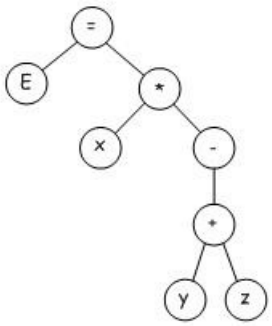

FIGURE 3.1 – Exemple d'un AST

3.3 Execution

Une fois les analyseurs lexical et syntaxique terminés, nous pourrons commencer à développer toutes les commandes basiques d'un shell, aussi appelées "built-ins".

Après avoir reconnu une ligne comme étant une commande, différents cas peuvent se présenter. Tout d'abord si le nom de la commande ne contient pas de slash, alors parmi les cas suivants, le premier valide est choisi :

- Si le nom correspond à une builtin, alors celle ci est executée - Sinon, si le nom correspond à une fonction connue de 42SH, alors cette fonction est executée - Sinon, la fonction sera executée si le nom correspond à une fonction dans la liste suivante : alias bg cd command false fc fg getopts jobs kill newgrp pwd read true umask unalias wait

Si le nom contient au moins un slash, alors le processus du dessus sera effectué dans un autre processus que celui de 42SH actuel.

3.4 Readline

Le readline est une partie importante du projet, faisant office d'interface entre notre projet et l'utilisateur. Il est donc important que celui ci soit facile et agréable d'utilisation. Le travail du readline sera de lire les entrées utilisateur, de les afficher et finalement de les envoyer au lexer quand nécessaire. A terme il serait appréciable que le readline communique avec d'autres modules, notamment un module d'historique et un module de completion afin de faciliter son utilisation.

3.5 Suite de test

Pour un projet de ce type et de cette ampleur, la nécessité d'une suite de test complète et automatisée nous est apparue comme une évidence. Cette suite de test permettra de tester facilement notre projet au fur et à mesure de son avancement. Cela nous permettra de tester nos nouvelles fonctionnalités et de nous assurer que les anciennes fonctionnent toujours et ainsi éviter les régressions. Il sera important de ne pas négliger certains aspects majeurs : notre suite de tests se devra d'être complète pour détecter toute erreur potentielle ; ses résultats doivent être lisibles et interprétables facilement afin qu'elle soit efficace. Nous avons décidé d'utiliser le langage python pour réaliser notre suite de test. C'est ce langage qui a retenu notre attention car il est particulièrement adapté à cette utilisation de par sa nature. Ce langage est facile à appréhender et il est utilisable facilement pour produire des scripts. De plus, c'est un langage "à la mode" qui rassemble une communauté très conséquente et permet de disposer d'un support actif en cas de problème.

3.6 Génération de fichier de compilation

Un projet aussi complet requiert une routine de compilation particulièrement soignée pour permettre une utilisation facile du code. Cela permet d'améliorer la portabilité du projet et de s'assurer que l'étape de la compilation sera correctement réalisée, sans oubli de dépendance ou autre erreur humaine. L'utilisation d'un programme de création de fichier de compilation est donc une aide grandement appréciée qui permet d'être plus efficace en évitant les erreurs inutiles. Nous avons choisi d'utiliser l'outil Cmake pour remplir ce rôle. C'est la portabilité et la simplicité d'utilisation de ce programme qui ont été des facteurs déterminants dans notre choix.

3.7 Variables, Alias et Extensions

Les alias seront substitués directement dans le parseur selon les spécifications de la SCL.

Les variables de la forme $name devront aussi être gérées, notamment les variables d'environnement.

La gestion des extensions sera elle effectuée pour : Les chemins des fichiers Les expressions arithmétiques (Exemple : $((1 + 2))

Chapitre 4

Répartition du travail

4.1 Timeline