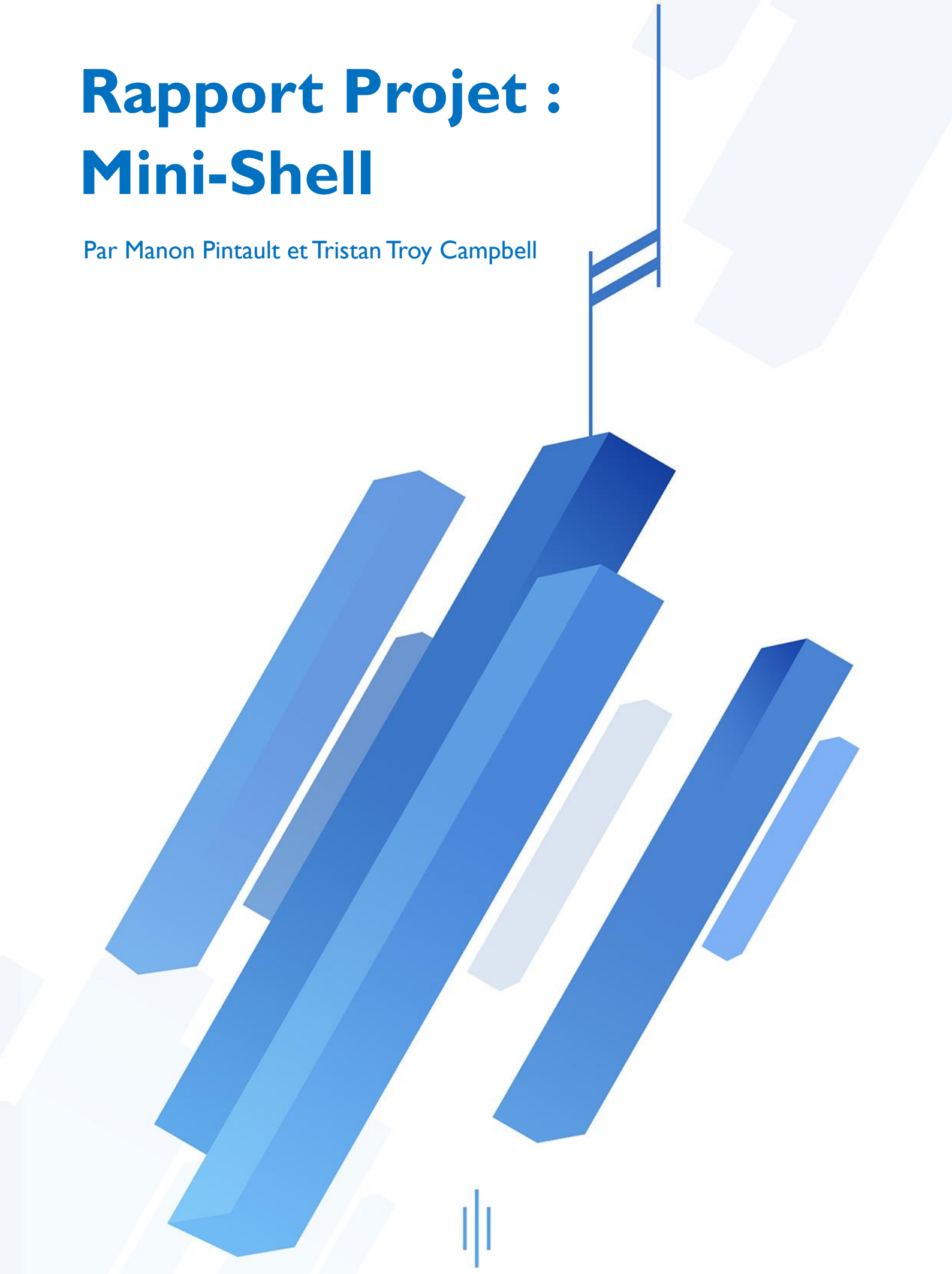
Rapport Projet : Mini-Shell

Par Manon Pintault et Tristan Troy Campbell



Introduction

Dans ce rapport, nous allons vous expliquer comment nous avons fait chacune des fonctions demandées en les classant par catégories :

- Commandes Externes ;
- Expressions ;
- Commandes Internes ;
- Remote Shell.

Ce projet a été géré avec Github dont voici le lien du dépôt :
https://github.com/manpintault/Projet_Prog_System

Les Fonctions

Commandes Externes

- Commande simple en arrière-plan (&)
  - Une commande exécutée en arrière-plan permet l'exécution d'autres commandes sans forcer ou attendre sa terminaison.
  - Tout simplement, pour exécuter une commande en arrière-plan, on lance un processus fils et on exécute la commande souhaitée sans mettre de fonction wait(…).
  - **Exemple** : emacs&

- Élimination des Zombies
  - Il n'existe pas de commande explicite dans le shell pour exécuter l'élimination des zombies. Il faut donc éliminer chaque zombie avant la terminaison du processus qui l'a créé.
  - Une fois qu'une expression donnée est réduite à une expression simple, on détermine si elle nécessite un processus fils afin d'exécuter la commande souhaitée. Si un processus fils est lancé, il exécute la commande souhaitée. Dans le processus père on fait un appel à un variant de la méthode wait(…) (selon les besoins de la commande lancée) pour vérifier que le processus fils termine avant que processus père se termine. Ainsi on vérifie l'élimination des zombies et la libération d'espace dans la table des processus.

Expressions

- Expression ; Expression
  - Une expression de la forme « *expression ; expression* » est composée de deux sous-expressions : une à gauche et une à droite. Pour exécuter les expressions comme celle-ci on fait un appel récursif à la fonction *evaluer_expr*(…) deux fois en passant chaque sous-expression comme paramètre.

- Expression || Expression
  - On essaie d'abord d'exécuter la première expression donnée. Si elle s'exécute sans problème, on ne traite pas la deuxième expression. Si on n'arrive pas à traiter la première expression, on essaie d'exécuter la deuxième expression.

- Expression && Expression
  - On essaie d'abord d'exécuter la première expression donnée. Si elle s'exécute sans problème, on procède au traitement de la deuxième expression. S'il y a une erreur dans l'exécution de la première, on ne traite pas la deuxième.

- (Expression)
  - Pour traiter une expression comme celle-ci, on passe tout simplement la sous-expression de gauche comme paramètre en faisant un appel récursif à la fonction *evaluer_expr*(…). Ainsi on vérifie que toutes les expressions regroupées dans les parenthèses sont évaluées ensemble.

- Expression | Expression
  - Pour diriger la sortie standard de la première expression vers la entrée standard de la deuxième expression, on fait d'abord un *pipe(…)* suivi d'un *fork()*.

    Dans le processus fils, on enregistre la sortie standard dans une variable en faisant un appel à *dup(…)*. Après, on dirige la sortie standard vers l'entrée du tube « *tube[1]* » avec *dup2(…)*. Après avoir exécuté la première expression, on redirige la sortie standard vers lui-même en faisant un deuxième appel à *dup2(…)*.

    Dans le processus père, après la terminaison du processus fils, on enregistre l'entrée standard dans une variable avec *dup(…)* et puis dirige l'entrée standard vers la sortie du tube « *tube[0]* » (le point de lecture) avec *dup2(…)*.

    Finalement, on exécute la deuxième expression, qui prend son entrée du tube, et on redirige l'entrée standard vers lui-même avec dup2(…).

- Expression > Fichier
  - Pour diriger la sortie standard vers un fichier souhaité, on fait d'abord un pipe(…) suivi d'un fork().

    Dans le processus fils, on enregistre la sortie standard dans un variable en faisant un appel à dup(…). Après, on dirige la sortie standard ver l'entrée du tube « tube[1] » avec dup2(…). Après avoir exécuté la première expression, on redirige la sortie standard vers lui-même ne faisant un deuxième appel à dup2(…).

    Dans le processus père, après la terminaison du processus fils, on crée un fichier avec le nom spécifié, ou s'il existe déjà on l'écrase. On lit directement du tube et écrit dans le fichier qu'on vient de créer.

- Expression < Fichier
  - Pour diriger le contenu d'un fichier vers l'entrée standard d'une expression, on fait un *pipe(…)* suivi d'un *fork()*.

    Dans le processus fils on lit du fichier spécifié et écrit directement dans l'entrée du tube. Dans le processus père, après la terminaison du processus fils, on fait une copie du *STDIN* avec *dup(…)* et puis on dirige *STDIN* vers la sortie du tube *tube*. Finalement, on exécute l'expression et redirige *STDIN* vers lui-même.

- Expression >> Fichier
  - On fait exactement ce qu'on fait avec l'expression « Expression > Fichier », sauf au lieu des modes O_TRUNC, on met O_APPEND afin de rajouter la sortie standard à la fin du fichier spécifié.

- Expression 2> Fichier
  - Pour diriger uniquement la sortie erreur vers un fichier souhaité, on fait exactement ce qu'on a fait dans le cas « *Expression > Fichier* » sauf qu'on remplace toutes références à STDOUT par STDERR.

- Expression &> Fichier

- Pour diriger la sortie standard et l'erreur standard vers le fichier souhaité, on fait exactement ce qu'on a fait dans le cas « *Expression > Fichier* » en rajoutant un *dup(…)* et deux *dup2(…)* en passant une référence à STDERR.

- Expressions Récursives
  - Ce cas est traité automatiquement.

Commandes Internes

- Echo, Date, Pwd, Hostname, Kill, Etc
  - Après avoir déterminé qu'une de ces commandes est appelée par l'utilisateur, on concatène l'entrée et on la passe comme paramètre dans la fonction *system(…)*.

- Cd
  - Après avoir déterminé que cette commande est spécifiée par l'utilisateur, on fait un appel à la fonction *chdir(…)* en passant la chaine de caractères donnée par l'utilisateur.

- History
  - Après avoir déterminé que cette commande est spécifiée par l'utilisateur, on imprime à la sortie standard le contenu stocké dans les structs *HIST_ENTRY* récupérées par la méthode *history_list()*.

- Exit
  - Après avoir déterminé que cette commande est spécifiée par l'utilisateur, on fait un appel à la fonction *exit(EXIT_SUCESSS)*.

Remote Shell

- Remote Add
  - On crée un fichier « connexion.txt » qui va enregistrer le nom des machines avec lesquelles nous allons établir une connexion ssh (« ssh infini1 bash & »).
  - On crée ensuite deux tableaux de caractères : le premier pour stocker le nom des machines et le deuxième pour la création des commandes ssh.
  - On teste s'il y a une ou plusieurs machines à lancer, puis on fait une boucle (s'il y en a plusieurs) dans laquelle on remplit nos tableaux et où on lance nos commandes ssh avec System().

- Remote Remove
  - On a deux tableaux de caractères: un dans lequel on stock les commandes « ssh machine exit » et l'autre qui va stocker les noms des machines connectées.
  - Pour avoir le nom de ces machines, on lit simplement le fichier « connexion.txt » qui regroupe le nom de toutes les machines connectées.
  - On fait une boucle dans laquelle on crée nos lignes de commandes ssh.
  - A la fin de la fonction, on lance nos commandes avec System() en lisant simplement le tableau qui contient les commandes.
  - On supprime à la fin de la fonction le fichier « connexion.txt » car il n'y a plus aucune machines de connectées.

- Remote List
  - On a une simple boucle dans laquelle on lit le fichier « connexion.txt » et on affiche les noms à l'écran.

- Remote nom_machine commande_simple
  - On lance une commande ssh, avec System(), qui prend le nom de la machine et la commande souhaitée (« ssh nom_machine commande_simple & » -> « ssh infini1 date & »).

- Remote all commande_simple
  - Même fonction que la commande « remote remove » sauf que l'on rajoute aux commandes ssh crées la commande souhaitée et que l'on ne supprime pas le fichier « connexion.txt » à la fin.

Actuellement nous ne faisons pas de PIPE dans ces fonctions.