

Projet C : HACHE*

Anthony GRIFFON, Eliot CHAUVINEAU, Meriem EL QSIMI, Axel JUGUET

17 avril 2016

Le but de ce document, est de fournir un compte rendu de notre projet, le compte rendu prendra la forme d'un manuel d'utilisation détaillant les différentes fonctionnalités de notre terminal, HACHE.

A la fin du document, il y a un chapitre où chaque membre du projet fera un compte rendu personnel du projet.



Le projet HACHE, est découpé en deux parties, chaque partie comporte deux membres. La première équipe, chargé de s'occuper de la partie du terminal et de la mise en place de la compilation à l'aide de librairies statiques et dynamiques en lui même est composé de Anthony GRIFFON, et de Eliot CHAUVINEAU

La seconde équipe, qui se charge des différentes commandes à exécuter, se compose de Meriem EL QSIMI et Axel JUGUET.



Notre projet se trouve sur le Gitlab de l'Université de Nantes, à l'adresse : <https://gitlab.univ-nantes.fr/E158196C/Hache>.

1	Première utilisation du terminal	3
1.1	Compilation du programme	3
1.2	Mise en place des variables	4
1.3	Démarrage du programme	4
2	Commandes liées au terminal	7
2.1	Se déplacer dans les dossiers (cd)	7
2.2	Effacer le résultats des précédentes commandes (clear)	7
2.3	Quitter le terminal (exit)	7
2.4	Se connecter à un serveur distant (connect)	8
3	Le langage du terminal	11
3.1	Priorités	11
3.2	Les redirection	11
3.3	La pipe	12
3.4	Les opérateurs logiques	12
3.5	Les taches de fonds	13
4	Les différentes commandes	15
4.1	Afficher le contenu d'un dossier (myls)	15
4.2	Copier les fichiers (mycp)	15
4.3	Bouger les fichiers (mymv)	15
4.4	Créer des dossiers (mymkdir)	15
4.5	Afficher le répertoire courant (mypwd)	16
4.6	Afficher (mydu)	16
5	Analyse du code	17
5.1	Cppchecker	17
5.2	Valgrind	18
6	Compte-rendus personnels	21
6.1	Anthony GRIFFON	21
6.2	Eliot CHAUVINEAU	21
6.3	Meriem EL QSIMI	21
6.4	Axel JUGUET	22
	Bibliographie	23

Première utilisation du terminal

1.1 Compilation du programme

Avant d'être utilisé, le programme se doit d'être compilé. Une fois les sources obtenues, se placer dans le terminal usuel, et exécuter :

```
bash4.2$> make
```

La compilation peut se faire de différentes façons, nous allons voir toutes les options possibles au `make`.

Avertissement

Le terminal peut utiliser des bibliothèques et fonctions non utilisés sous les systèmes WINDOWS®, le logiciel a été testé sous UBUNTU 14.04 et OS X 10.11.2.

La compilation du programme nécessite la présence du compilateur GCC ≥ 4.8 ou CLANG $\geq 7.0.2$.

Le programme peut compiler sous d'autres versions de GCC ou CLANG, cependant, nous ne le garantissons pas.

Make all

En exécutant la commande "`make all`", le terminal ainsi que toutes les commandes attachées ([Voir chapitre lié](#)) seront générés en mode exécutable. L'exécutable du terminal sera alors enregistré dans le dossier `./bin/`.

Make all-lib

En exécutant la commande "`make all-lib`", le terminal ainsi que toutes les commandes attachées ([Voir chapitre lié](#)) seront générés en mode bibliothèque statique. L'exécutable pourra être déplacé n'importe où, on aura pas besoin d'indiquer au terminal où se trouvent les exécutables pour les exécuter. On pourra cependant quand même l'indiquer pour ajouter des exécutables.

Make all-lib-dyn

En exécutant la commande "`make all-lib-dyn`", la compilation sera effectuée en mode bibliothèques dynamiques, l'exécutable devra lire les commandes dans les fichiers `.so` compilés dans le dossier `"lib"`. De cette façon, on pourra mettre à jour les bibliothèques sans pour autant avoir à recompiler le terminal.

Make cmd

L'utilisation de cette commande va permettre la génération des fichiers commandes disponible dans ce chapitre ([Voir chapitre lié](#)).

Make proper

L'utilisation de cette commande entrainera la destruction de tous les fichiers générés à la compilation du programme, que ce soit les fichiers .o dans le dossier "build" ou les exécutables générés qui n'ont pas été déplacé.

Make clean

L'utilisation de cette commande entrainera la destruction de tous les fichiers générés à la compilation du programme hormis les fichiers exécutables.

1.2 Mise en place des variables

Une fois le programme compilé, si vous avez utilisé la compilation normale ou dynamiques, des exécutables ou des bibliothèques sont générés pour certaines commandes ([Voir chapitre lié](#)) et des chemins doivent être spécifiés.

Il est important pour notre terminal de spécifier l'emplacement de ces programmes. C'est pourquoi dans le répertoire principal de HACHE, là vous effectuez la commande "make", se trouve un script à exécuter.

```
define_variable.sh
#!/bin/bash
#
echo "Usage: . ./define_variable.sh"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$(pwd)/lib
export PTERMINAL=$(pwd)/commands
```

Ce script est à exécuter à l'aide de cette commande :

```
bash
bash4.2$> . ./define_variable.sh
```

Ou bien avec

```
bash
bash4.2$> source ./define_variable.sh
```

1.3 Démarrage du programme

Démarrons maintenant le programme, une fois la compilation terminée, l'exécutable du programme se trouve dans le dossier "bin".

bash

```
bash4.2$> cd bin
bash4.2$> ./hache
```

Une fois le programme lancé vous devriez avoir cet affichage ¹ :

Hache

```
Serveur lancé sur le port: 10719.
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

Si jamais vous avez le message : "Les librairies personnels ne sont pas chargés, merci d'indiquer le répertoire des executables dans la variable d'environnement PTERMINAL." reportez vous à [cette section](#).

On peut observer qu'un serveur se lance sur un port, pour plus d'information, il faut regarder la commande "connect".

Nous avons aussi le fil d'Ariane ² qui s'affiche avant d'entrer les commandes.

1. HACHE va prendre le premier port libre à partir de 10000.

2. Cela indique le dossier courant.

Commandes liées au terminal

Nous allons lister et expliquer les commandes liées au terminal, ce sont les commandes qui ne sont pas exécutés par des programmes externes mais par le programme HACHE.

2.1 Se déplacer dans les dossiers (cd)

La première de ces commandes concerne la capacité à naviguer entre différents dossiers, la commande "cd".

Cette commande ne prend qu'une option : le dossier où l'on veut aller. Voici quelques exemples de la commande.

```
                                Hache
Serveur lancé sur le port: 10719.
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> cd ..
prompt1.0: /Users/anthonygriffon/Desktop/Hache/> cd
prompt1.0: /Users/anthonygriffon/Desktop/Hache/> cd lizejdelzmdjkezmek
Erreur:: No such file or directory
prompt1.0: /Users/anthonygriffon/Desktop/Hache/>
```

2.2 Effacer le résultats des précédentes commandes (clear)

Il s'agit d'une commande n'ayant pour objectif qu'une meilleure visibilité. Cette commande permet d'effacer toutes les commandes et tous les résultats entrés précédemment pour qu'il ne reste plus qu'une ligne sur le terminal.

```
                                Hache
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

2.3 Quitter le terminal (exit)

Pour quitter le programme HACHE, il suffit d'entrer cette commande, le programme va alors quitter, le serveur de connection à distance va se stopper et vous allez retourner à votre précédent programme.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> exit  
Au revoir !
```

2.4 Se connecter à un serveur distant (connect)

Cette commande permet de se connecter à d'autres terminaux HACHE à distance, la connexion n'est pas sécurisée. On évitera de transmettre des données sensibles via cette commande.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> connect  
Utilisation: connect host port.
```

Deux arguments doivent être renseignés pour cette commande.

L'hôte

Le premier argument à être donné est l'hôte, il peut être sous la forme d'une adresse IP (192.168.1.23) ou alors d'un nom de domaine (http://perdu.com).

Le programme est capable d'atteindre l'hôte même en renseignant un domaine ou une adresse internet, cependant, il est préférable de privilégier l'adresse IP.

Le port

Le port à être renseigné est affiché sur le serveur auquel vous souhaitez vous connecter. Le port d'écoute du serveur s'affiche lors du démarrage d'un terminal HACHE.

La connection

Le principe de connection au serveur est différent des protocoles normaux, lorsque vous rentrez la commande connect avec les paramètres et que vous lancez la commande, cela ne vous connecte pas. Le terminal se met en mode passif, il va attendre les requêtes que vous voulez envoyer au serveur distant.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> connect localhost 12345  
prompt-dp$>
```

Pour envoyer une requête au serveur, il ne vous reste qu'à taper la commande et appuyer sur entrée.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> connect localhost 12345  
prompt-dp$> echo hey  
hey  
prompt-dp$>
```

Multi-utilisateurs

On peut se demander pourquoi le système est construit de cette façon, lors de la conception nous avons voulu privilégier un système qui permet à un serveur d'accepter plusieurs clients en même temps sans avoir à multiplier les systèmes d'écoutes sur le serveur.

C'est dans cette optique que nous est venu cette conception particulière, avec laquelle il est possible de connecter plusieurs utilisateurs sur le même serveur distant.

Quitter le mode connecté

Pour quitter le mode connecté, il suffit de taper la commande "exit".

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> connect localhost 12345
prompt-dp$> exit
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

Le langage du terminal

Dans ce chapitre nous allons aborder le langage du terminal, ces différentes possibilités pour taper des lignes de commandes plus ou moins complexes.

3.1 Priorités

Notre terminal est tel que on ne peut pas grouper les commandes dans des parenthèses, il va lire la ligne de gauche à droite dans l'ordre sans se soucier de groupes de commandes.

3.2 Les redirection

Nous allons voir les redirections, les redirections permettent d'orienter les programmes. Par exemple, si nous donnons le un poème écrit par Nabila l'entrée de notre programme de traduction et que nous orientons la sortie sur un fichier texte, nous allons avoir dans ce fichier texte, le poème, lisible.

Avec HACHE il est possible de :

- Rediriger la sortie d'une commande vers un fichier.
- Ajouter la sortie d'une commande à la fin d'un fichier.
- Rediriger le contenu d'un fichier vers l'entrée d'une commande.

Rediriger la sortie d'une commande vers un fichier

Il est possible de rediriger la sortie d'une commande vers un fichier avec l'opérateur ">".

```
Hache  
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> echo "Youhou" > file.txt  
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

Le résultat se trouve alors dans le fichier file.txt

```
file.txt  
"Youhou"
```

Ajouter la sortie d'une commande à la fin d'un fichier

Il est aussi possible d'ajouter à la suite d'un fichier la sortie d'une commande avec l'opérateur ">>".

Hache

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> echo "Youhou" > file.txt
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> echo suite >> file.txt
```

Le résultat se trouve alors dans le fichier file.txt

file.txt

```
"Youhou"
suite
```

Rediriger le contenu d'un fichier vers l'entrée d'une commande

Dans certaines commandes il peut être utile de remplacer l'entrée standard par un fichier, il est possible de le faire avec "<".

Hache

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> cat < file.txt
"youhou"
suite
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

3.3 La pipe

Pour connecter la sortie d'une commande sur l'entrée d'une autre, avec les redirections, il est nécessaire de diriger la sortie d'une commande vers un fichier et rediriger le fichier vers l'entrée de la commande.

Cette manipulation peut s'écrire en une ligne avec le pipe "|".

Pour l'instant, notre shell ne supporte qu'une seule redirection par commande saisie. Ainsi, ls test | grep test2 > test3 ne fonctionnera pas.

Hache

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> ls -l | grep README.md
-rwxr-xr-x  1 anthonygriffon  staff  3390  4 avr 11:53 README.md
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

3.4 Les opérateurs logiques

Il est possible de complexifier les commandes avec les opérateurs logiques "et" et "ou". Chaque programme exécuté se termine en renvoyant un code de sortie, il s'agit soit d'un code d'erreur soit d'un code de "réussite" (le code d'erreur 0).

L'opérateur ET

L'opérateur "ET" est symbolisé par "&&" dans HACHE.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> ls && pwd
```

Ici la commande ls s'exécute en première, si elle s'exécute sans erreur, alors la commande pwd s'exécute à son tour, si jamais ce n'est pas le cas, la commande pwd ne s'exécute pas.

Voici différents exemples :

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> echo 1 && pwd
1
/Users/anthonygriffon/Desktop/Hache/bin
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> ls dzkejdzhzkl && pwd
ls: dzkejdzhzkl: No such file or directory
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> pwd && pwd && echo 1 && ls ldezkdj
/Users/anthonygriffon/Desktop/Hache/bin
/Users/anthonygriffon/Desktop/Hache/bin
1
ls: ldezkdj: No such file or directory
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

L'opérateur OU

L'opérateur "OU" est symbolisé par "||" dans HACHE.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> ls || pwd
```

Ici la commande ls s'exécute avec succès, de ce fait la commande pwd ne s'exécutera pas. Quelques exemples :

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> echo 1 || pwd
1
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> ls dzkejdzhzkl || pwd
ls: dzkejdzhzkl: No such file or directory
/Users/anthonygriffon/Desktop/Hache/bin
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> pwd || pwd || echo 1 || ls ldezkdj
/Users/anthonygriffon/Desktop/Hache/bin
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

3.5 Les taches de fonds

Le terminal HACHE permet d'exécuter les commandes en tache de fond. Pour ce faire, il suffit d'utiliser l'opérateur & en fin de ligne.

Hache

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> sleep 5 && myls &  
[1] 22233  
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

Ici, les commandes vont s'exécuter dans un terminal HACHE avec pour PID 22233. Cependant tout ce qui s'affiche dans la commande en tâche de fond s'affiche dans le terminal actuel.

Hache

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> sleep 5 && myls &  
[1] 22233  
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>  
drwxr-xr-x 6 anthonygriffon staff      204 2016-04-16 21:49 .  
drwxr-xr-x 14 anthonygriffon staff     476 2016-04-16 22:10 ..  
-rw-r--r-- 1 anthonygriffon staff    6148 2016-04-16 19:26 .DS_Store  
-rw-r--r-- 1 anthonygriffon staff      15 2016-04-16 21:24 file.txt  
-rwxr-xr-x 1 anthonygriffon staff   29120 2016-04-16 21:49 Hache  
drwxr-xr-x 3 anthonygriffon staff     102 2016-04-16 21:04 Hache.dSYM  
  
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

Les différentes commandes

4.1 Afficher le contenu d'un dossier (myls)

La commande myls affiche le contenu du répertoire courant. Aucune options.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> myls
drwxr-xr-x 6 anthonygriffon staff      204 2016-04-16 21:49 .
drwxr-xr-x 14 anthonygriffon staff     476 2016-04-16 22:10 ..
-rw-r--r-- 1 anthonygriffon staff    6148 2016-04-16 19:26 .DS_Store
-rw-r--r-- 1 anthonygriffon staff      15 2016-04-16 21:24 file.txt
-rwxr-xr-x 1 anthonygriffon staff   29120 2016-04-16 21:49 Hache
drwxr-xr-x 3 anthonygriffon staff     102 2016-04-16 21:04 Hache.dSYM
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin>
```

4.2 Copier les fichiers (mycp)

La commande mycp permet de copier un fichier.

```
prompt1.0: /Users/anthonygriffon/Desktop/Hache> mycp
Usage : mycp source destination
prompt1.0: /Users/anthonygriffon/Desktop/Hache>
```

4.3 Bouger les fichiers (mymv)

La commande mymv permet de changer le chemin d'un fichier.

4.4 Créer des dossiers (mymkdir)

La commande mymkdir permet de créer un dossier.

4.5 Afficher le répertoire courant (mypwd)

4.6 Afficher (mydu)

Affiche le nombre d'octets occupés dans le répertoire courant.

```
prompt1.0: /Users/anthonygriffon/Desktop> mydu
Taille : 877091682
prompt1.0: /Users/anthonygriffon/Desktop>
```

Analyse du code

5.1 Cppchecker

```
─ cppcheck src/*.c src/*.h inc/*.c inc/*.h commands/*.h commands/*.c > log2.txt 2>&1 ─
```

```
Checking commands/mycp.c...
Checking commands/mycp.c: DYN...
Checking commands/mycp.c: EXEC...
1/17 files checked 14% done
Checking commands/mydu.c...
Checking commands/mydu.c: DYN...
Checking commands/mydu.c: EXEC...
2/17 files checked 19% done
Checking commands/mydu.h...
3/17 files checked 19% done
Checking commands/myls.c...
Checking commands/myls.c: DYN...
Checking commands/myls.c: EXEC...
4/17 files checked 28% done
Checking commands/myls.h...
5/17 files checked 28% done
Checking commands/mymkdir.c...
Checking commands/mymkdir.c: DYN...
Checking commands/mymkdir.c: EXEC...
6/17 files checked 33% done
Checking commands/mymkdir.h...
7/17 files checked 34% done
Checking commands/mypwd.c...
Checking commands/mypwd.c: DYN...
Checking commands/mypwd.c: EXEC...
8/17 files checked 36% done
Checking commands/mypwd.h...
9/17 files checked 37% done
Checking inc/functions.h...
10/17 files checked 38% done
Checking inc/getInput.h...
11/17 files checked 38% done
Checking inc/socket.h...
12/17 files checked 38% done
Checking src/functions.c...
Checking src/functions.c: LIB...
13/17 files checked 58% done
Checking src/getInput.c...
14/17 files checked 58% done
```

```
Checking src/main.c...
Checking src/main.c: DYN...
Checking src/main.c: LIB...
Checking src/main.c: __linux__...
Checking src/main.c: linux...
15/17 files checked 97% done
Checking src/opttest.c...
16/17 files checked 99% done
Checking src/socket.h...
17/17 files checked 100% done
```

5.2 Valgrind

Voici un rapport valgrind de HACHE après une utilisation de quelques commandes personnelles (mysls, mykdir, mydu, mypwd, exit).

```
valgrind ./Hache > log.txt 2>&1
```

```
==22326== Memcheck, a memory error detector
==22326== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==22326== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==22326== Command: ./Hache
==22326==
Les bibliothèques personnels ne sont pas chargés, merci d'indiquer le répertoire des
executables dans la variable d'environnement PTERMINAL.
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> Les bibliothèques personnels ne sont pas
chargés, merci d'indiquer le répertoire des executables dans la variable d'environnement
PTERMINAL
Serveur lancé sur le port: 19015.
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> mysls
--22328-- UNKNOWN mach_msg unhandled MACH_SEND_TRAILER option
--22328-- UNKNOWN mach_msg unhandled MACH_SEND_TRAILER option (repeated 2 times)
--22328-- UNKNOWN mach_msg unhandled MACH_SEND_TRAILER option (repeated 4 times)
--22328-- UNKNOWN mach_msg unhandled MACH_SEND_TRAILER option (repeated 8 times)
drwxr-xr-x 9 anthonygriffon staff      306 2016-04-16 22:44 .
drwxr-xr-x 14 anthonygriffon staff      476 2016-04-16 22:10 ..
-rw-r--r-- 1 anthonygriffon staff     6148 2016-04-16 19:26 .DS_Store
-rw-r--r-- 1 anthonygriffon staff        15 2016-04-16 21:24 file.txt
-rwxr-xr-x 1 anthonygriffon staff    29120 2016-04-16 21:49 Hache
-rw-r--r-- 1 anthonygriffon staff     4532 2016-04-16 22:43 hache-valgrind
drwxr-xr-x 3 anthonygriffon staff      102 2016-04-16 22:39 Hache.dSYM
drwxr-xr-x 2 anthonygriffon staff        68 2016-04-16 22:43 Hey
-rw-r--r-- 1 anthonygriffon staff      959 2016-04-16 22:45 log.txt
==22328==
==22328== HEAP SUMMARY:
==22328==      in use at exit: 3,270,931 bytes in 4,359 blocks
==22328==    total heap usage: 4,579 allocs, 220 frees, 3,290,809 bytes allocated
==22328==
==22328== LEAK SUMMARY:
==22328==      definitely lost: 1,344 bytes in 8 blocks
==22328==      indirectly lost: 4,200 bytes in 8 blocks
==22328==      possibly lost: 0 bytes in 0 blocks
==22328==      still reachable: 3,233,829 bytes in 4,149 blocks
==22328==      suppressed: 31,558 bytes in 194 blocks
==22328== Rerun with --leak-check=full to see details of leaked memory
==22328==
```

```

==22328== For counts of detected and suppressed errors, rerun with: -v
==22328== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> mykdir blbl
Erreur : No such file or directory
==22329==
==22329== HEAP SUMMARY:
==22329==   in use at exit: 3,253,094 bytes in 4,300 blocks
==22329== total heap usage: 7,458 allocs, 3,158 frees, 6,429,622 bytes allocated
==22329==
==22329== LEAK SUMMARY:
==22329==   definitely lost: 2,080 bytes in 4 blocks
==22329==   indirectly lost: 4,096 bytes in 2 blocks
==22329==   possibly lost: 0 bytes in 0 blocks
==22329==   still reachable: 3,215,360 bytes in 4,100 blocks
==22329==   suppressed: 31,558 bytes in 194 blocks
==22329== Rerun with --leak-check=full to see details of leaked memory
==22329==
==22329== For counts of detected and suppressed errors, rerun with: -v
==22329== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> mydu
Taille : 52756
==22330==
==22330== HEAP SUMMARY:
==22330==   in use at exit: 3,254,118 bytes in 4,301 blocks
==22330== total heap usage: 10,552 allocs, 6,251 frees, 9,626,726 bytes allocated
==22330==
==22330== LEAK SUMMARY:
==22330==   definitely lost: 3,104 bytes in 5 blocks
==22330==   indirectly lost: 4,096 bytes in 2 blocks
==22330==   possibly lost: 0 bytes in 0 blocks
==22330==   still reachable: 3,215,360 bytes in 4,100 blocks
==22330==   suppressed: 31,558 bytes in 194 blocks
==22330== Rerun with --leak-check=full to see details of leaked memory
==22330==
==22330== For counts of detected and suppressed errors, rerun with: -v
==22330== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> mypwd
/Users/anthonygriffon/Desktop/Hache/bin
==22331==
==22331== HEAP SUMMARY:
==22331==   in use at exit: 3,255,142 bytes in 4,302 blocks
==22331== total heap usage: 13,610 allocs, 9,308 frees, 12,772,278 bytes allocated
==22331==
==22331== LEAK SUMMARY:
==22331==   definitely lost: 4,128 bytes in 6 blocks
==22331==   indirectly lost: 4,096 bytes in 2 blocks
==22331==   possibly lost: 0 bytes in 0 blocks
==22331==   still reachable: 3,215,360 bytes in 4,100 blocks
==22331==   suppressed: 31,558 bytes in 194 blocks
==22331== Rerun with --leak-check=full to see details of leaked memory
==22331==
==22331== For counts of detected and suppressed errors, rerun with: -v
==22331== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
prompt1.0: /Users/anthonygriffon/Desktop/Hache/bin> exit
Au revoir !
==22326==
==22326== HEAP SUMMARY:
==22326==   in use at exit: 3,246,795 bytes in 4,292 blocks

```

```
==22326==    total heap usage: 16,675 allocs, 12,383 frees, 15,934,235 bytes allocated
==22326==
==22326== LEAK SUMMARY:
==22326==    definitely lost: 5,120 bytes in 5 blocks
==22326==    indirectly lost: 0 bytes in 0 blocks
==22326==    possibly lost: 0 bytes in 0 blocks
==22326==    still reachable: 3,215,360 bytes in 4,100 blocks
==22326==           suppressed: 26,315 bytes in 187 blocks
==22326== Rerun with --leak-check=full to see details of leaked memory
==22326==
==22326== For counts of detected and suppressed errors, rerun with: -v
==22326== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

On peut observer à la sortie 3,246,795 bytes utilisé, et seulement 5,120 bytes de fuite dans l'application, d'après le programme des optimisations sont encore possibles, on peut aussi remarquer quelques problèmes dans le myls.

Compte-rendus personnels

6.1 Anthony GRIFFON

A travers le projet, on a pu découvrir de nouvelles utilisations du C. Un projet tel que le terminal est pratique car on peut observer le programme tel qu'il devrait être et essayer de construire notre propre version. Il s'agit surtout de reproduction, mais cela permet de se poser pas mal de questions.

On remarque aussi toute l'importance de la conception initiale d'un projet. Il ne faut pas trop rapidement passer au code ou on peut se retrouver confronté à des problèmes difficiles à résoudre.

Dans notre cas, nous avons pensé tardivement à la possibilité de grouper des commandes avec les parenthèses et la gestion de priorités. Il était trop tard pour tout changer mais cependant, cela nous a permis de nous rendre compte que notre conception initiale avait des lacunes.



6.2 Eliot CHAUVINEAU

Ce projet mélange habilement toutes les notions de programmation/système que nous avons apprises durant nos cours de C/système. C'est un bon exercice de programmation mais aussi et surtout de conception, et d'organisation en équipe. On ne peut pas se lancer dans le code tête baissée sans avoir réfléchi en groupe à la conception du programme.

Malheureusement nous n'avons pas pu implémenter toutes les fonctions que nous voulions apporter à notre shell. Nous n'avons pas pris assez de temps à réfléchir ensemble à sa conceptions avant de commencer à programmer. Les problèmes sont arrivés à la fin, et il était trop tard pour changer la conception de notre shell.



6.3 Meriem EL QSIMI

Le projet s'est révélé très enrichissant, il nous a permis de découvrir comment fonctionnent les commandes linux.

De plus, il nous a permis d'appliquer nos connaissances en langage C.

Les principaux problèmes, que nous avons rencontrés, concernaient la conception initiale du projet,

nous avons directement commencé le code. Ainsi, j'ai eu au début quelques problèmes en utilisant le git sous windows.



6.4 Axel JUGUET

Tout d'abord ce projet était intéressant dans sa conception pour plusieurs points :

- Utilisation des appels systemes déjà appris et découverte de nouveaux
- Creation d'un programme autonome et améliorable par la suite
- Très différent de ce qu'on a pu faire jusque là

Pour ma part, ce projet m'a permis de me rendre compte de certaines lacunes en programmation C ainsi que dans mon organisation. J'ai passé beaucoup de temps à chercher et à fouiller dans la documentation UNIX pour comprendre le fonctionnement de certains appels systeme (tels que rename qui m'a posé beaucoup de problèmes). J'ai en revanche beaucoup mieux compris le fonctionnement des pointeurs, chose qui me paraissait encore un peu floue avant le projet.

Pour ce qui est des problèmes rencontrés, mis à part mes problèmes de codage, j'ai rencontré de nombreuses difficultés à cause de git notamment vis à vis du merge. Je n'ai pas pu donc commit la commande mymv que j'avais codée.



Bibliographie

- [1] Lars Madsen, *Various chapter styles for the memoir class*, 2011.
- [2] Peter Wilson, *The Memoir Class for Configurable Typesetting – User Guide*, 2010.
- [3] Basé sur un photocopié de Roberto Di Cosmo, Xavier Leroy et Damien Doligez. *Entrée, sortie, redirection*. <http://www.tuteurs.ens.fr/unix/shell/entreesortie.html>.
- [4] *Various statements in a makefile*. http://makepp.sourceforge.net/1.19/makepp_statements.html.
- [5] *Options Controlling the Preprocessor*. <http://gcc.gnu.org/onlinedocs/gcc-4.4.1/gcc/Preprocessor-Options.html>.
- [6] Stackoverflow. *Various topics*. <http://stackoverflow.com>.