

Projet linux - Script Shell - Manipulation d'une bitmap

Achille Gaborit et Ophélie Le Mentec

28 avril 2014

Table des matières

1	Notre projet	2
1.1	Choix du projet	2
1.2	Analyses préliminaires	2
2	Réalisation	5
2.1	Difficultés rencontrées	5
2.2	Le projet final	8
2.3	Conclusion	9

Chapitre 1

Notre projet

1.1 Choix du projet

Ayant peu de connaissances sur linux, nous avons mis beaucoup de temps à choisir un sujet. Au départ nous voulions installer Apache pour être sûres d'aller jusqu'au bout de notre réalisation. En revanche nous étions tout de même tentés par la rédaction d'un Script Shell. Nous en avons déjà rédigé un en TP, donc nous savions que c'était réalisable selon le sujet choisi. Finalement, nous avons choisi de rédiger un Script Shell en partant du principe que même si nous n'arrivons pas au bout de notre projet, nous apprendrons tout de même des notions intéressantes. Nous avons opté pour un traitement d'image. En effet, cela nous paraissait simple au départ puisque d'après nos connaissances de bases une image bmp était un fichier qui contient les valeurs des pixels de l'image. Nous n'aurions donc qu'à modifier le contenu d'un fichier par l'intermédiaire du script shell. Nous avons décidé de faire un script shell capable à partir d'une image indiquée par l'utilisateur de créer une copie de cette image avec un effet noir et blanc.

1.2 Analyses préliminaires

Tout d'abord nous avons décidé de traiter des images de type BMP puisque nous avons l'habitude d'utiliser ce type d'image. De plus nous avons fait des recherches sur le contenu exact des images. Nous avons trouvé ce tableau sur wikipedia (Figure 1.1), celui-ci explique l'organisation du fichier pour une Bitmap.

Nous avons vérifié ces informations en ouvrant une bitmap avec un éditeur de texte. Nous pouvons voir sur la Figure 1.2 une image au format bmp

Offset#	Taille	Valeur
0x0000	2 octets	le nombre magique correspondant à l'utilisation du fichier BMP <ul style="list-style-type: none"> • BM - Windows 3.1x, 95, NT, etc. • BA - OS/2 <i>Bitmap Array</i> • CI - OS/2 Icône Couleur (<i>Color Icon</i>) • CP - OS/2 Pointeur Couleur (<i>Color Pointer</i>) • IC - OS/2 Icône (<i>Icon</i>) • PT - OS/2 Pointeur (<i>Pointer</i>)
0x0002	4 octets	la taille du fichier BMP en octets
0x0006	2 octets	réservé pour l'identifiant de l'application qui a créé le fichier
0x0008	2 octets	réservé pour l'identifiant de l'application qui a créé le fichier
0x000A	4 octets	l' <i>offset</i> (l'adresse de départ) du contenu du BMP

FIGURE 1.1 – Tableau représentant l'organisation d'une Bitmap
??

et l'organisation de son fichier. Comme indiqué sur le tableau, les 2 premiers octets du fichier sont bien "BM", ce qui indique que l'image est bien au format bmp. Nous constatons aussi que les informations de l'image sont stockées sous forme de caractères ascii. Or, nous avons vu dans la 5eme ligne du tableau que l'on va devoir lire dans le fichier l'adresse de départ du contenu du BMP. Nous allons donc devoir convertir ces caractères ascii pour obtenir la valeur de l'adresse. Les octets sont stockée en little-endian et l'adresse est stockée sur 4 octets. Nous aurons donc juste à lire chacun des octets et à les multiplier par $255^{\text{numero-octet}}$ puis à tout additionner pour obtenir l'adresse.

Nous n'aurons pas ce type de problème avec les pixels. En effet nous n'avons pas besoin d'obtenir leur valeur pour les traiter. Pour obtenir les pixels grisés nous pouvons multiplier chacun des composantes rouge verte bleue par les coefficients définis par la commission internationale de l'éclairage.

Nous concluons que notre script va devoir :

A cartoon illustration of a penguin, likely representing the Linux mascot, Tux. The penguin is black and white with a large yellow beak and feet. It is standing and facing forward.

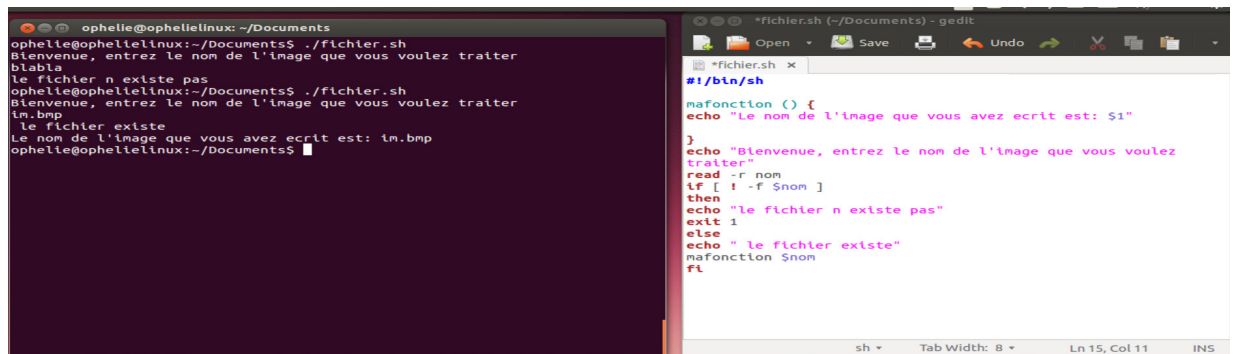
Chapitre 2

Réalisation

2.1 Difficultés rencontrées

Démarrage

La toute première étape consiste à demander à l'utilisateur un nom de fichier et à lire ce nom pour vérifier que le fichier existe. (figure 2.1)



The image shows two side-by-side windows. The left window is a terminal with the following text:

```
ophelle@ophellelinux: ~/Documents
ophelle@ophellelinux:~/Documents$ ./fichier.sh
Bienvenue, entrez le nom de l'image que vous voulez traiter
blabla
le fichier n existe pas
ophelle@ophellelinux:~/Documents$ ./fichier.sh
Bienvenue, entrez le nom de l'image que vous voulez traiter
in.bmp
le fichier existe
Le nom de l'image que vous avez ecrit est: in.bmp
ophelle@ophellelinux:~/Documents$
```

The right window is a text editor (gedit) editing the file `*fichier.sh`. The code is as follows:

```
#!/bin/sh

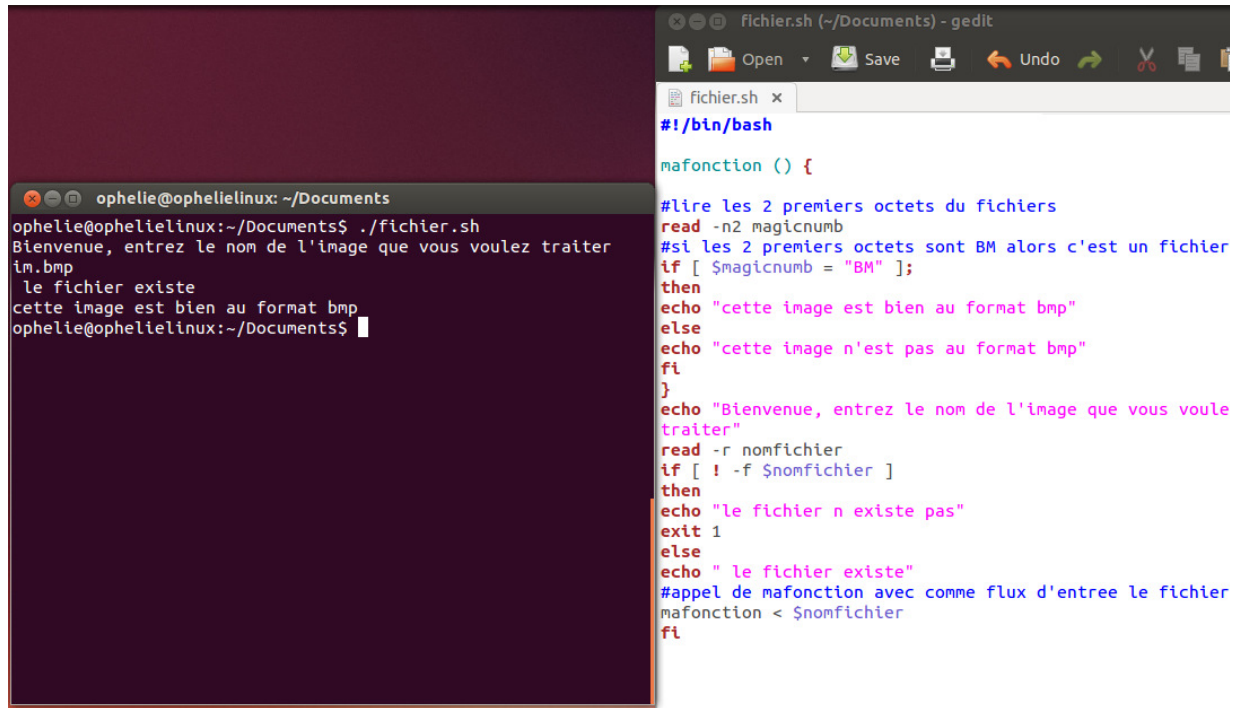
mafonction () {
    echo "Le nom de l'image que vous avez ecrit est: $1"
}

echo "Bienvenue, entrez le nom de l'image que vous voulez
traiter"
read -r nom
if [ ! -f $nom ]
then
    echo "le fichier n existe pas"
    exit 1
else
    echo " le fichier existe"
    mafonction $nom
fi
```

FIGURE 2.1 – Premier essai

Cela marche, nous avons même envoyé le nom du fichier dans la fonction. En revanche de cette manière nous n'avons pas réussi à lire dans le fichier indiqué par l'utilisateur. Nous voulions lire dans le fichier avec la commande `read`. Nous avons pour cela indiqué lors de l'appel de la fonction que le flux d'entrée est le fichier (et n'est plus l'entrée de l'utilisateur) à l'aide du signe `<`. Nous avons aussi appris que pour rediriger la sortie on peut utiliser le signe `>`.

Nous pouvons voir dans la figure 4 que nous pouvons lire le fichier. Nous



The image shows a terminal window on the left and a text editor window on the right. The terminal window, titled 'ophelie@ophelielinux: ~/Documents', shows the execution of a script named 'fichier.sh'. The user enters 'im.bmp' as input, and the script outputs: 'Bienvenue, entrez le nom de l'image que vous voulez traiter', 'im.bmp', 'le fichier existe', and 'cette image est bien au format bmp'. The text editor window, titled 'fichier.sh (~/.Documents) - gedit', shows the source code of the script. The script is a bash script that reads the first two octets of a file to determine if it is a BMP image. It uses 'read -n2 magicnumb' to read the first two octets, then checks if they are 'BM'. If they are, it prints 'cette image est bien au format bmp'. If not, it prints 'cette image n'est pas au format bmp'. It also checks if the file exists using 'if [! -f \$nomfichier]'. If the file does not exist, it prints 'le fichier n existe pas' and exits with status 1. If the file exists, it prints 'le fichier existe' and calls the 'mafonction' function with the filename as an argument.

```
ophelie@ophelielinux: ~/Documents
ophelie@ophelielinux:~/Documents$ ./fichier.sh
Bienvenue, entrez le nom de l'image que vous voulez traiter
im.bmp
le fichier existe
cette image est bien au format bmp
ophelie@ophelielinux:~/Documents$
```

```
#!/bin/bash

mafonction () {
#lire les 2 premiers octets du fichiers
read -n2 magicnumb
#si les 2 premiers octets sont BM alors c'est un fichier
if [ $magicnumb = "BM" ];
then
echo "cette image est bien au format bmp"
else
echo "cette image n'est pas au format bmp"
fi
}
echo "Bienvenue, entrez le nom de l'image que vous voulez
traiter"
read -r nomfichier
if [ ! -f $nomfichier ]
then
echo "le fichier n existe pas"
exit 1
else
echo " le fichier existe"
#appel de mafonction avec comme flux d'entree le fichier
mafonction < $nomfichier
fi
```

FIGURE 2.2 – Redirection d'entrée

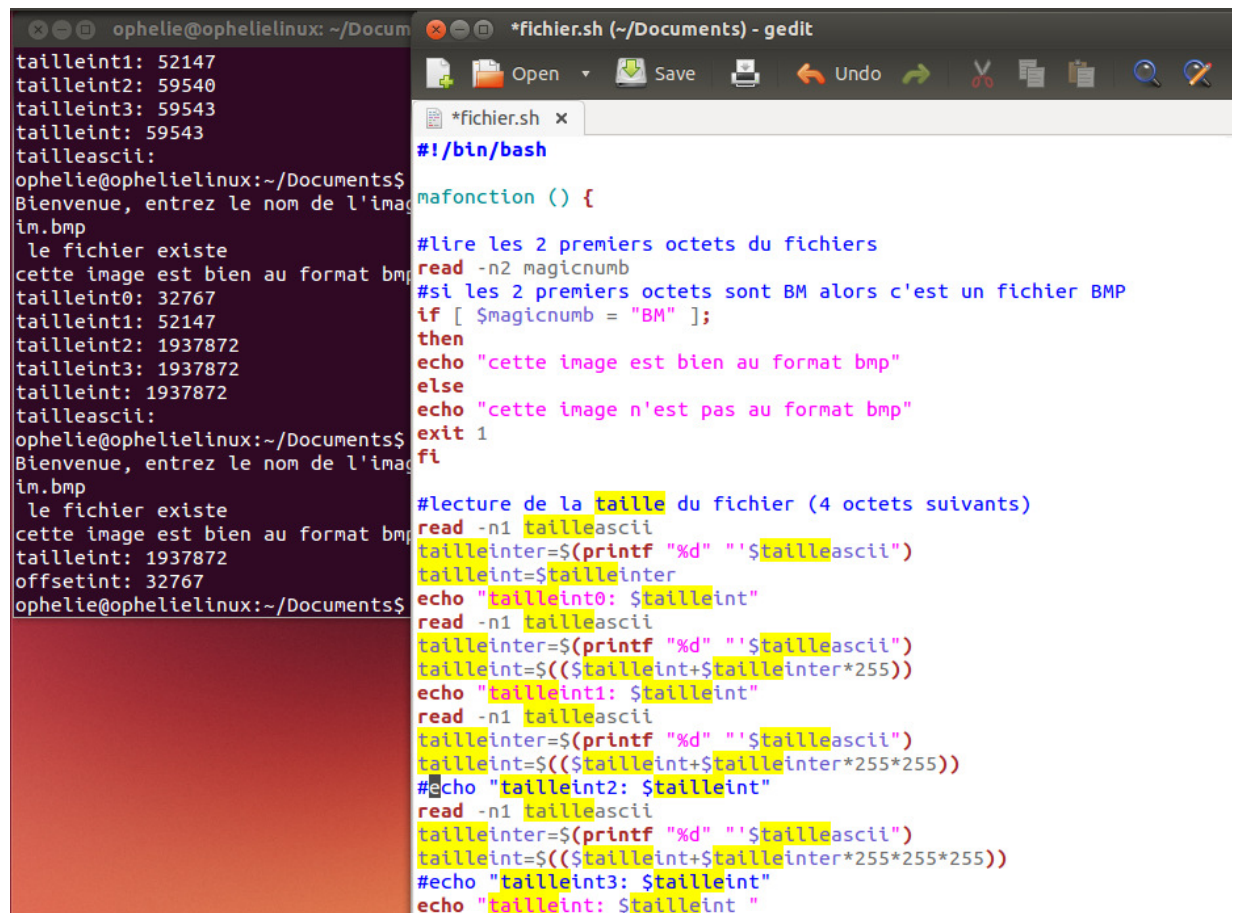
verifions que les 2 premiers octets correspondent bien aux caractères BM ce qui prouve que l'on a une image de type BMP. Pour lire le nombre X d'octets que l'on souhaite lire, nous utilisons `read -nX variable`. Nous sommes passé du shell `sh` au `bash` car `bash` avait cette fonctionnalité.

Nous avons eu une autre difficulté par la suite. Pour lire les données décimales nous avons utilisé la valeur retournée par le `printf` de notre octet en entier. En revanche au départ les valeurs indiquées étaient fausses.

Par exemple dans la figure 2.3 la taille de l'image n'était pas celle correspondante. Nous avons finalement résolu ce problème avec difficulté.

Pour cela, nous avons loggé dans un fichier texte toutes les valeurs des composantes des pixels. Nous avons vu qu'elles étaient fausses (par exemple la composante rouge valait 32767 alors qu'elle n'est pas sensé dépasser 255) nous en avons déduit que les valeurs des pixels étaient mal lu. Après de longues recherches nous avons supposé que la raison à cela était que les caractères étaient lu en `utf-8` et pas en `ascii`. Or `utf-8` comprend plus de caractères que la table `ascii`, de plus pour `utf-8` certains caractères sont codés sur 2 octets au lieu de 1 et certains caractères n'ont pas la même équivalence

binaire que dans la table ascii. Cela expliquerait les valeurs corrompues. Nous avons rajouté les lignes suivantes : `export LANG="C"` et `export LC ALL="C"` qui permettrait au shell de lire en "mode ascii".



```

ophelie@ophelielinux: ~/Documents
tailleint1: 52147
tailleint2: 59540
tailleint3: 59543
tailleint: 59543
tailleascii:
ophelie@ophelielinux:~/Documents$
Bienvenue, entrez le nom de l'image:
im.bmp
le fichier existe
cette image est bien au format bmp
tailleint0: 32767
tailleint1: 52147
tailleint2: 1937872
tailleint3: 1937872
tailleint: 1937872
tailleascii:
ophelie@ophelielinux:~/Documents$
Bienvenue, entrez le nom de l'image:
im.bmp
le fichier existe
cette image est bien au format bmp
tailleint: 1937872
offsetint: 32767
ophelie@ophelielinux:~/Documents$

*ichier.sh (~/.Documents) - gedit
*ichier.sh x
#!/bin/bash

mafonction () {
#lire les 2 premiers octets du fichiers
read -n2 magicnumb
#si les 2 premiers octets sont BM alors c'est un fichier BMP
if [ $magicnumb = "BM" ];
then
echo "cette image est bien au format bmp"
else
echo "cette image n'est pas au format bmp"
exit 1
fi

#lecture de la taille du fichier (4 octets suivants)
read -n1 tailleascii
tailleinter=$((printf "%d" "'$tailleascii"))
tailleint=$tailleinter
echo "tailleint0: $tailleint"
read -n1 tailleascii
tailleinter=$((printf "%d" "'$tailleascii"))
tailleint=$((tailleint+tailleinter*255))
echo "tailleint1: $tailleint"
read -n1 tailleascii
tailleinter=$((printf "%d" "'$tailleascii"))
tailleint=$((tailleint+tailleinter*255*255))
#echo "tailleint2: $tailleint"
read -n1 tailleascii
tailleinter=$((printf "%d" "'$tailleascii"))
tailleint=$((tailleint+tailleinter*255*255*255))
#echo "tailleint3: $tailleint"
echo "tailleint: $tailleint"

```

FIGURE 2.3 – Ascii - utf-8

Cela nous a pris beaucoup de temps de réussir à lire le fichier correctement. En revanche, à partir du moment où nous avons compris comment rediriger les sorties et les entrées, comment lire le nombre souhaité d'octet et comment lire et convertir correctement des valeurs numériques exprimées en ascii, tout est allé beaucoup plus vite.

2.2 Le projet final

Pour lancer le programme il faut "se trouver" dans le dossier contenant le .sh et taper la ligne de commande suivante : `./desaturer2.sh nomDeLImageQueLonSouhaiteTraiter.bmp` ou alors on peut écrire `./desaturer2.sh nomDeLImageQueLonSouhaiteTraiter.bmp NomDeLimageGriseSouhaitée.bmp` si l'utilisateur écrit juste `./desaturer2.sh` l'utilisation lui sera précisée comme-ci dessous. Si l'utilisateur ne précise pas de nom d'image souhaitée en sortie par défaut ce sera out.bmp

```
ophelie@ophelielinux:~/Documents/proj2$ ./desaturer2.sh
Utilisation: desaturer.sh image_entree.bmp [image_sortie.bmp]
ophelie@ophelielinux:~/Documents/proj2$ ./desaturer2.sh flag.bmp
Le fichier existe
Le fichier fait 46182 octets
Cette image est bien au format BMP
L'image est bien en 24 bits par pixels
Reecriture de l'en tete de l'image... ok
Desaturation de l'image... 368/15376 pixels
```

FIGURE 2.4 – Utilisation du script shell

Vous pouvez observer le résultat ci-dessous.



FIGURE 2.5 – Resultat

2.3 Conclusion

Nous avons réussi à obtenir le résultat voulu. Le traitement d'image met un certain temps à se réaliser, nous pensons que la raison à cela est que chaque traitement est une ligne de commande donc est un appel vers un programme. Un programme en C aurait certainement été plus rapide mais nous sommes contents d'avoir réalisé ce projet car nous avons pu voir certaines des fonctionnalités du `sh` et du `bash`. Notamment nous avons appris à rediriger les entrées et les sorties, à lire un nombre d'octet voulu, à écrire dans un fichier et à la suite d'un fichier sans écraser le reste. Nous avons pu tester différentes fonctionnalités de `read`. Nous avons aussi fait des recherches parallèles liées à notre projet, par exemple nous connaissons maintenant le codec d'une Bitmap. Lors de ce projet nous avons traversé beaucoup de difficultés inattendues. Nous avons la satisfaction de les avoir résolues par une grâce à de nombreuses recherches. En effet la plupart des problèmes que nous avons eu, d'autres l'ont eu avant nous. De plus le fait de se retrouver face à des problèmes inattendus tel que l'ascii et l'utf-8 nous a forcé à chercher loin et nous aidera peut-être plus tard à anticiper ce type de problèmes.

Nos principales sources ont été : doc.ubuntu-fr.org/ et ses forums, wikipedia et stackoverflow.