INFO-F201 – OS
Projet Bash

Alexis Reynouard Yannick Molinghen Joël Goossens

22 octobre 2021

ImageMagick [1] est une suite logicielle permettant de créer, de modifier et de composer des images matricielles (par opposition à vectorielles). Elle peut lire, convertir et écrire des images dans de nombreux formats.

Le but de ce projet est d'écrire un script bash `compress_jpg.sh` pour compresser des JPEG, du type de photos souvenirs, en utilisant la commande `convert` fournie par ImageMagick.

1 Utilisation d'ImageMagick

Pour le projet, vous aurez besoin d'appeler :
`convert -auto-orient -colorspace RGB` *`input_filename`* `-resize` *`resolution`*`x`*`resolution`*`^> -strip -quality 85% -colorspace sRGB -interlace Plane -define jpeg:dct-method=float -sampling-factor 4:2:0` *`output_filename`*.

En supposant que *`output_filename`* a l'extension d'un fichier JPEG, cette commande convertit de façon précise le fichier *`input_filename`* en un JPEG progressif [2] de qualité 85% [3], avec la chrominance réduite de moitié [4], et enregistre le résultat dans *`output_filename`*. De plus, la résolution de l'image obtenue est telle que le ratio n'est pas changé (*i.e.* le rapport entre la largeur et la hauteur de l'image n'est pas changé : l'image n'est pas déformée) et que le plus petit côté de l'image fait *`resolution`* pixels, s'il était plus grand dans l'image originale.

C'est le seul appel à ImageMagick dont vous avez besoin, si ce n'est que l'option `-strip` ne doit être donnée que si elle est explicitement demandée par l'utilisateur (voir plus bas). Cette option supprime les méta-données du fichier produit (les données comme la date de prise de vue...).

Vous devez prendre garde à ce que le caractère spécial "`>`" ainsi que les caractères spéciaux potentiellement présents dans les noms de fichiers (et particulièrement l'espace), ne soient pas interprétés par bash et arrivent tels quels à ImageMagick.

Si *`output_filename`* a l'extension d'un autre type d'image, le fichier produit par cette commande est une image d'un type correspondant à son extension. ImageMagick va prendre en compte toutes les options données que le format demandé supporte.

1. https://imagemagick.org/
2. Un JPEG progressif permet de charger l'image non pas de haut en bas en bonne qualité, mais dans son entièreté avec une qualité qui augmente. Vous avez un exemple ici https://www.hostinger.fr/tutoriels/jpeg-progressif.
3. La signification exacte de la qualité d'un JPEG est une notion complexe, mais on pourrait la résumer ainsi : la qualité de l'image est réduite à 85% de sa qualité originelle, relativement à la sensibilité de l'œil humain. Les détails auxquels l'œil est sensible sont mieux conservés que les autres.
4. L'œil humain est plus sensible à la luminosité qu'à la teinte ou chrominance.

2 Invocation

Votre script `compress_jpg.sh` doit donc appeler ImageMagick (*via* la commande `convert` donnée ci-dessus) sur des fichiers, JPEG ou non : si l'utilisateur spécifie un répertoire, votre script ne doit travailler que sur les JPEG s'y trouvant. Si l'utilisateur spécifie un fichier, votre script appelle `convert` dessus sans effectuer au préalable de vérification du type du fichier. Ces fichiers sur lesquels votre script appelle `convert` seront dits *sélectionnés (pour la compression)*.

Nous appellerons *entrées* les fichiers et dossiers indiqués par l'utilisateur. Il peut les spécifier de deux façons différentes :

— soit par un argument sur la ligne de commande, dans ce cas on ne peut indiquer qu'une seule entrée à chaque appel à `compress_jpg.sh`.

— soit en utilisant le `stdin` de `compress_jpg.sh`, dans ce cas `compress_jpg.sh` traite chaque ligne de son `stdin` comme une entrée.

Ces deux méthodes ne peuvent pas être utilisées simultanément. Si une entrée est donnée sur la ligne de commande, `compress_jpg.sh` ignore son `stdin`.

Si l'utilisateur le demande (avec l'option `-e`, voir plus bas), en plus d'appeler `convert`, votre script normalise l'extension des fichiers JPEG qui sont sélectionnés pour la compression. Cette normalisation n'est jamais effectuée sur d'autres types de fichier.

Votre script doit accepter les paramètres suivants :

`./compress_jpg.sh [-c] [-r] [-e` *`extension`*`] `*`resolution`*` [`*`filename_or_directory`*`]`

`-h`, `--help`, `help`
: Si une de ces options est donnée, les autres options sont ignorées, un message d'aide détaillé est affiché sur la `stdout` et le script quitte avec succès.

`-r`, `--recursive`
: Si un dossier est donné en entrée, `compress_jpg.sh` travaille aussi dans les sous-dossiers récursivement (vous ne devez pas gérer les cas où cela induirait des boucles infinies dues à des arborescences particulières.) Sinon, cette option n'a pas d'effet.

`-c`, `--strip`
: Cette option active le passage de l'argument `-strip` à `convert`.

`-e` *`extension`*, `--ext` *`extension`*
: Pour chaque fichier JPEG sélectionné pour la compression : `compress_jpg.sh` change l'extension du fichier en l'extension donnée, ou ajoute l'extension si le fichier n'en a pas. Ce renommage n'a pas lieu si le fichier sélectionné n'est pas un fichier JPEG ou si le nom de fichier produit existe déjà pour un autre fichier. Ce renommage a lieu si un JPEG est sélectionné pour compression, même si la compression n'a finalement pas lieu (voir plus bas). Le script vérifie que l'extension donnée est une de `jpg`, `jpeg`, `jpe`, `jif`, `jfif`, `jfi`, `JPG`, `JPEG`, `JPE`, `JIF`, `JFIF`, `JFI`. Si ce teste échoue, un message d'erreur est affiché sur `stderr` et le script quitte avec erreur.

`resolution`
: La taille (en pixels) demandée pour le plus petit côté de l'image, si celui-ci est plus grand dans le fichier d'origine. Ce paramètre est un entier positif (chaîne de caractère qui ne correspond pas au pattern "`''|*[!0-9]*`" utilisable dans un `case` bash).

`directory_or_filename`
: Spécifie une entrée.

  — Si un dossier est donné, travaille sur tous les JPEG (détectés comme tels par `file`) du dossier et, si `-r` est spécifié, des sous-dossiers récursivement.

— Si un fichier est donné, travaille sur ce fichier. Si ce n'est pas un fichier JPEG, `-e` est ignoré et on laisse ImageMagick décider que faire.
— Si cette option n'est pas donnée, le script lit son `stdin` en traitant chaque ligne comme une entrée.
— Si cette option est donnée mais n'est ni un dossier, ni un fichier standard ou n'existe pas, le script affiche un message d'erreur sur `stderr` et quitte avec une erreur (3)

Pour éviter des erreurs, le script doit refuser les noms commençants par un tiret ("`-`"). Si l'utilisateur veut donner une entrée commençant par un tiret, il peut utiliser le `stdin`, par exemple : `echo -mon-fichier.jpg | ./compress_jpg.sh 1600`.

Tous les paramètres sont facultatifs, sauf la résolution. Les paramètres peuvent apparaître dans n'importe quel ordre. Cependant, si *`directory_or_filename`* est un nombre entier, il doit apparaître après la résolution. L'espace est obligatoire entre chaque option et entre `-e` et l'*`extension`*.

3 Implémentation

Nous présentons ici une description détaillée du mode de fonctionnement du script.

1. Le script cherche dans les paramètres reçus, le paramètre `-h` ou un équivalent. S'il existe, il affiche un message d'aide détaillé sur la `stdout` et quitte avec succès.
2. Il parse les paramètres reçus. Si une erreur est détectée, il affiche sur la `stderr` un message décrivant l'erreur suivi d'une description courte de la syntaxe à utiliser, puis quitte en retournant un code d'erreur (1).
3. Ensuite, pour chaque image sur laquelle il doit travailler (déterminées comme précisé plus haut) il :
   (a) renomme le fichier le cas échéant (voir plus haut) et affiche le nom du fichier (qu'il soit renommé ou non).
   (b) appelle ImageMagick par la commande `convert` pour écrire une version compressée de l'image dans un fichier temporaire. Si l'appel échoue (`convert` retourne non-zéro), `compress_jpg.sh` affiche un message d'erreur sur sa `stderr`, le fichier temporaire est supprimé s'il avait été créé et le script passe aux images suivantes s'il y en a.
   (c) Si le temporaire obtenu est plus léger que l'image d'origine, ce fichier temporaire est utilisé pour remplacer le fichier d'origine. Sinon, il est supprimé, le fichier d'origine est laissé intact (mais possiblement renommé) et un message est affiché sur la `stdout`.

Votre script doit retourner
— 1 si la commande n'a pas été utilisée avec une syntaxe correcte.
— 2 s'il y a eu au moins un appel à `convert` qui a échoué.
— 3 si au moins une entrée reçue n'existe pas ou n'est ni un fichier standard ni un dossier.
Si les erreurs 2 et 3 surviennent, vous pouvez décider quel code d'erreur retourner.

4 Exemples

Rappel : "`$?`" vaut la valeur de retour de la dernière commande exécutée.
Les lignes 1 et 9 montrent des appels non valides.

La ligne 17 montre un appel où le script lit les noms de fichiers/dossiers depuis son `stdin`. (La ligne 18 est tapée à la main.) À la ligne 20, avant l'affichage du `0`, l'utilisateur a écrit le caractère de fin de fichier avec la combinaison : `[Ctrl] + [D]`.

La ligne 21 montre un appel sur un dossier avec l'option `-r`.

La ligne 27 montre un appel sur un fichier PNG déjà petit. Le résultat de l'appel à `convert` n'ayant pas produit un fichier plus petit, la version originale est conservée. Notez que ceci n'est pas une erreur, le message est affiché sur la `stdout` et la commande retourne 0.

La ligne 31 montre un appel qui réduit le fichier PNG. On voit que l'option `-e` n'a pas eu d'effet.

Les lignes 34 et 38 sont des appels normaux sur un dossier. On remarque que deux appels consécutifs à `convert` permettent d'obtenir une meilleur compression. Seuls les JPEG sont traités.

À la ligne 42, `convert` ne produit plus de fichiers plus petits, mais le renommage a bien lieu.

À la ligne 48, `compress_jpg.sh` reçoit les noms de fichiers depuis son `stdin`, redirigé pour recevoir la sortie du `echo`. Comme un fichier n'existe pas, la commande retourne 3.

À la ligne 54, on appelle `compress_jpg.sh` sur un fichier qui existe mais n'est pas un JPEG. `convert` est appelé mais l'appel échoue. La commande retourne 2.

Aux lignes 61 et 64, on voit qu'un fichier JPEG est traité même s'il ne possède pas la bonne extension.

(Notez qu'avec une très grande valeur pour la résolution, ce script peut servir à renommer les JPEG sans extensions ou avec une mauvaise extension.)

```
1  $ ./compress_jpg.sh ; echo $?
2  Missing resolution
3  USAGE
4      ./compress_jpg.sh [-c] [-r] [-e extension] resolution [filename_or_directory]
5  or
6      ./compress_jpg.sh --help
7  for detailed help.
8  1
9  $ ./compress_jpg.sh 200 -e png ; echo $?
10 -e argument must be one of jpg, jpeg, jpe, jif, jfif, jfi (or uppercase version
   ↪  of one)
11 USAGE
12     ./compress_jpg.sh [-c] [-r] [-e extension] resolution [filename_or_directory]
13 or
14     ./compress_jpg.sh --help
15 for detailed help.
16 1
17 $ ./compress_jpg.sh 200 -e jpeg ; echo $?
18 test-files/large.jpg
19 test-files/large.jpeg
20 0
21 $ ./compress_jpg.sh 200 -e JPG test-files -r ; echo $?
22 test-files/large.JPG
23 test-files/small.JPG
24 test-files/subdir/large.JPG
25 test-files/subdir/small.JPG
26 0
27 $ ./compress_jpg.sh 200 test-files/favicon-48.png ; echo $?
28 test-files/favicon-48.png
29 Not compressed. File left untouched. (normal)
30 0
```

```
$ ./compress_jpg.sh 16 test-files/favicon-48.png -e jpg ; echo $?
test-files/favicon-48.png
0
$ ./compress_jpg.sh 260 test-files/ ; echo $?
test-files/large.JPG
test-files/small.JPG
0
$ ./compress_jpg.sh 260 test-files/ ; echo $?
test-files/large.JPG
test-files/small.JPG
0
$ ./compress_jpg.sh 260 test-files/ -e jpg ; echo $?
test-files/large.jpg
Not compressed. File left untouched. (normal)
test-files/small.jpg
Not compressed. File left untouched. (normal)
0
$ echo 'test-files/nonexist.jpg
> test-files/subdir/large.JPG' | ./compress_jpg.sh 2000 -e jpg -c ; echo $?
test-files/nonexist.jpg does not exist
test-files/subdir/large.jpg
3
$ touch not-an-image
$ ./compress_jpg.sh 200 not-an-image ; echo $?
not-an-image
convert-im6.q16: no decode delegate for this image format `' @
 ↪  error/constitute.c/ReadImage/560.
convert-im6.q16: no images defined `compressed_jpg.8FkK3Tco3' @
 ↪  error/convert.c/ConvertImageCommand/3258.
Error while compressing not-an-image. File left untouched.
2
$ mv test-files/subdir/small.JPG test-files/subdir/small.png
$ ./solution.sh test-files/subdir/ 200
test-files/subdir/large.jpg
test-files/subdir/small.png
$ ./solution.sh test-files/subdir/ 160 -e jpg
test-files/subdir/large.jpg
test-files/subdir/small.jpg
```

5 Conseils, aides et consignes

Votre code doit être clair et suffisamment commenté. Entre autres, les choix relatifs aux ambiguïtés de l'énoncé doivent être mentionnés.

Vous avez sur l'UV un fichier de base pour vous faire gagner du temps.

Dans ce fichier, les variables `USAGE` et `SHORT_USAGE` possèdent des codes de formatage[5]. Ceci permet d'avoir un affichage avec de la "mise en forme" dans la console. Pour afficher la mise en

5. https://misc.flogisoft.com/bash/tip_colors_and_formatting

forme, il faut utiliser `echo -e "$USAGE"` ou `echo -e "$SHORT_USAGE"`.

Vous pouvez supprimer le formatage (par exemple, si la sortie ne se fait pas sur une console mais a été redirigée vers un fichier), avec la fonction `print_without_formatting` fournie. Ce genre de détails fera gagner des points bonus. On peut vérifier que la `stdout` est une console avec `[ -t 1 ]` et que la `stderr` est une console avec `[ -t 2 ]`. (Voir Conditional Expressions [6] et Redirections [7]). Si ces tests échouent les sorties ont sûrement été redirigées avec un opérateur de redirection tel que `>`.

Lorsqu'un dossier est donné, vous devez travailler sur tous les fichiers contenus dedans qui sont des JPEG. On considère qu'un fichier est un JPEG si la sortie de `file -i` *`nom_de_fichier`* contient `image/jpeg`. C'est cette même règle qui détermine si on normalise l'extension du fichier.

Vous pouvez parser les paramètres avec un `case` bash [8], comme illustré dans le fichier ajouté à l'UV dans le labo 4.

Vous pouvez rediriger les entrées et sorties standards de plusieurs commandes groupées. Par exemple `echo 1 >file ; echo 2 >file` est équivalent à `{ echo 1 ; echo 2 ; } >file`. Attention : le point-vigule ou le retour à la ligne avant "`}`" est obligatoire pour que celui-ci soit détecté comme le caractère de fin de groupe. De même, `function foo () { echo 3 ; } >/file` définit une fonction qui écrira `3` dans `file`.

Rappel : `"$@"` a une signification spéciale : il vaut `"$1" "$2" "$3" "$4"...`, ce qui le rend utile pour passer les paramètres reçus à une autre commande.

L'option `-q` de grep est très utile pour vérifier si une sous-chaîne se trouve dans une chaîne de caractère sans rien afficher à l'écran.

`exit` [9] sort du shell courant. Ceci signifie que `exit` ne finit pas votre script si vous l'appelez au sein d'un `$(...)`, puisque cette construction appelle une commande dans un sous-shell. Par exemple :

```
function foo () { if [ "$1" = "hello" ] ; then echo "Nice to meet you" ; else
↪  exit 1 ; fi ; }
reponse=$(foo "hi") # Ici le script n'est pas quitté.
fail=$? ; if [ $fail -ne 0 ] ; then exit $fail ; fi # Ici le script est quitté
# Dans ce cas, si foo avait utilisé un return, le résultat aurait été le même.
# Mais dans la ligne suivante provoque l'arrêt du script seulement si foo utilise
↪  un exit.
foo "hi!" # $
```

On peut vérifier qu'une chaîne de caractère est non-vide avec `[ -n chaîne ]`. Attention : Si `$var` n'est pas défini ou vaut la chaîne de caractères vide, alors `[ -n $var ]` retourne vrai. Pourquoi ? Premièrement, bash remplace `$var` par sa valeur. Ensuite, il appelle `[` avec comme seul paramètre `-n`. Quand `[` a un seul paramètre, il retourne vrai si ce paramètre est non-vide. D'où le résultat. Maintenant si on avait écrit `[ -n "$var" ]`, bash aurait remplacé `$var` par sa valeur, puis `""` par la chaîne vide, puis aurait appelé `[` avec deux arguments : `-n` et la chaîne vide. Lorsque `[` a deux paramètres, il considère le premier comme un opérateur unaire. Ici, il reconnaît bien l'opérateur `-n` que nous voulions utiliser, et retourne faux.

La commande `mktemp` donne un nom de fichier qui n'existe pas sur la machine. Elle est spécialement adaptée à la création de fichier temporaire dont le nom n'a pas d'importance. Si vous

6. https://www.gnu.org/software/bash/manual/bash.html#Bash-Conditional-Expressions
7. https://www.gnu.org/software/bash/manual/bash.html#Redirections
8. https://www.gnu.org/software/bash/manual/bash.html#index-case
9. https://www.gnu.org/software/bash/manual/bash.html#index-exit

souhaitez que votre code soit compatible avec MacOS, vous pouvez remplacer `mktemp` par `mktemp 2> /dev/null || mktemp -t compress_jpg`

Rappelez-vous que vous pouvez utiliser ces syntaxes :

```
if command ; then
  # command returned 0
else
  # command returned non-0
fi
if ! command1 && command2 ; then
  # command1 returned non-0 and command2 returned 0
elif [ -a filename ] && command ; then
  # ...
fi
while read line ; do
  echo $line
done # $
```

Si vous n'implémentez pas toutes les fonctionnalités demandées, indiquez-le clairement dans votre code source, dans un commentaire au début, ou dans un README.

Enfin, souvenez-vous que vous pouvez faire des appels récursifs, soit en appelant la fonction dans laquelle vous êtes, soit en appelant le script qui vous écrivez (`"$0" $arguments` par exemple appelle le script courant avec les arguments contenus dans la variable `$arguments`).

6 Rendu

Vous devez remettre sur l'UV, avant le 14 novembre 2021, donc au plus tard le 13 novembre à 23 h 59 :

— un script bash `compress_jpg.sh` fonctionnant sur les machines du NO4.

— un unique fichier txt, rst, md, html ou pdf, indiquant vos choix, les limitations de votre script ou les fonctionnalités non demandées que vous auriez développées (conseil : ne perdez pas de temps pour le projet).