https://vader-fr.fr/spip.php?article97

Script bash Linux de copie de fichiers playlistés (m3u)

- Linux & Logiciels Libres -

Publication date: lundi 8 octobre 2012

Un fichier .m3u est un fichier de type playlist (comme les fichiers .pls), ne contenant en fait que du texte, listant ainsi les fichiers à lire.
Ce type de fichier est créé par un lecteur multimédia : Winamp, VLC.... sous Windows, Audacious, Totem, VLC... sous Linux.

Pour faire simple, ce script bash lit le fichier playlist m3u indiqué en premier paramètre et copie les musiques sur la destination indiquée en second paramètre, le tout en respectant la hiérarchie des répertoires musicaux à partir de la plus petite racine de toutes les musiques.

Ainsi, si les musiques sont dans `/media/KastorPollux/Musiques et sons` et la destination `/media/Kle usb`, il copiera dans `/media/Kle usb` le répertoire "`Musiques et sons`" puis les fichiers musiques listés dans le fichier m3u, dans leurs sous-répertoires respectifs.

Les fichiers playlist au format .m3u ne sont pas codés de la même façon que les .pls
Ce script est fait pour le format m3u uniquement.

voici donc à quoi ressemble le contenu d'un fichier m3u :

```
#EXTM3U
#EXTINF:181,Star Wars - The imperial march (Dark Vador
/media/KastorPollux/Musiques et sons/T.V et films/Star Wars/Star-Wars épisode V L'Empire
Contre-Attaque/01_The imperial march (Dark Vador's theme).mp3
#EXTINF:489,John Williams - Star Wars Episode V - 02 - Main title / The ice planet Hoth
/media/KastorPollux/Musiques et sons/T.V et films/Star Wars/Star-Wars épisode V L'Empire
Contre-Attaque/02_Main title  The ice planet Hoth.mp3
#EXTINF:184,John Williams - Star-Wars Episode V - 04-03 attacking a star destroyer
/media/KastorPollux/Musiques et sons/T.V et films/Star Wars/Star-Wars épisode V L'Empire
Contre-Attaque/4-03-attacking a star destroyer.mp3
#EXTINF:363,John Williams - Star-Wars Episode V - 04-05 imperial starfleet deployed-city in the clouds
/media/KastorPollux/Musiques et sons/T.V et films/Star Wars/Star-Wars épisode V L'Empire
Contre-Attaque/4-05-imperial starfleet deployed-city in the clouds.mp3
...
```

la première ligne est toujours `#EXTM3U` et décrit le "format" du fichier.
les lignes suivantes alternent entre ligne de "commentaire" `#EXTINF:********` et le chemin du fichier à charger dans le lecteur multimédia.

CopierPlaylist

Pour utiliser le script, il faut :

- le décompresser
- puis ouvrir un terminal (menu applications, outils système, terminal, sous Gnome 2)
- aller dans le répertoire où est le script (avec la commande `cd`
- le rendre exécutable (`chmod u+x CopierPlaylist.sh`)
- et enfin le lancer : `./CopierPlaylist playlist.m3u destination`

Bien sûr, certains diront "mais c'est de la ligne de commande".
 Certes, oui, mais il est quand même bien plus facile d'utiliser l'auto-complétion en bash pour fournir les chemins du fichier playlist et celui de la destination, plutôt que d'avoir à écrire entièrement à la main les mêmes chemins, au risque de se tromper.

Voici le code commenté du script, qui du reste est très simple :

Le script utilise cut et awk, il faut donc avoir installé les paquets correspondants.

Comme tout script sh, on indique l'interpréteur de commandes à utiliser. Ici, on utilise bash, car sh peut ne pas gérer les tableaux.
 Le texte après un # est un commentaire

```
#!/bin/bash
# on attend 2 arguments. Plus simple à mettre en ligne de commande, avec l'auto-complétion, si il y a par exemple des espaces dans l'arborescence de la destination
if [ $# -eq 2 ]
then
 fichier=$1 # premier argument : le fichier playlist
 destination=$2 # deuxième argument : la destination où copier les musiques
```

Dans le doute, on demandera pour la destination un chemin absolu (dons sous Linux, commençant par /, la racine).
 A priori, chemin absolu ou relatif ça ne devrait rien changer au final, mais dans le doute....

On utilise donc la commande cut, avec option `-c` pour avoir un "séparateur par caractères seuls", et nom sur des champs séparés par " ;", tabulation, espace, "/" ou autre séparateur classique.
 En script shell, pour appeler une commande "externe" au script, on peut utiliser le format $(commande), qui renverra son résultat.

```
 # on récupère le premier caractère du chemin destination. la destination doit être en chemin absolu, donc commencer par /, la racine
 # l'option -c de cut indique qu'on veut lire un caractère précis, le numéro 1 ensuite spécifiant de lire le premier
 premiercard=$(echo "$destination" | cut -c1)
 if [ "$premiercard" != "/" ]
```

```
 then
 echo "Il faut mettre la destination en chemin absolu"
 else
 total=0 # taille totale des fichiers à copier
 stop=0  # phase 1/2 :marqueur d'arrêt si répertoire lu != celui empilé dans le tableau, pour trouver la racine ...
 # phase 3 :...mais aussi marqueur de départ, puisque correspondant à la case du tableau ayant le nom du répertoire racine
```

Dans la première phase, on va d'une part calculer la somme des tailles des fichiers à copier, résoudre le chemin du fichier racine des musiques (le chemin commun à toutes les musiques, ici donc /media/KastorPollux/Musiques et sons) et comparer la taille requise (somme calculée) à la taille disponible sur la destination.

On va donc lire le fichier playlist ligne par ligne, en ne s'intéressant qu'aux lignes ne commençant pas par #, qui sont des lignes commentaires, comme vu plus haut.
Mais avant de commencer la lecture, il peut être utile de vérifier que le premier paramètre fourni (mis dans la variable "fichier") correspond bien à un fichier **et** que celui-ci peut être lu, sans quoi le système enverra une erreur à l'exécution du script.
Cela se fait avec les tests "`-f`" ("est fichier") "`-a`" ("et") "`-r`" ("est lisible").
Une condition shell est habituellement entre crochets ("`[ ]`").

- Si c'est une condition classique, on aura `if [` *variable1 comparateur variable2* `]` (exemple `if [ "$premiercar" == "#" ]`)
- Si c'est un test, comme ici, on aura `if [` test variable `]` (exemple `if [ -f $fichier ]`)

- Les "comparateurs" sont `=`, `==`, `!=` (différent) et..... `gt` (greater than, >), `ge` (greater or equal, >=), `lt` (lesser than, <) et `le` (lesser or equal, <=).
- On peut combiner plusieurs conditions avec -a (ET logique) ou -o (OU logique)
- Une négation sera indiquée par " !". Exemple `if [ "$premiercard" != "/" ]` (si premiercard différent de /) peut s'écrire `if [ ! "$premiercard" == "/" ]` (si N'EST PAS premiercard égal à /)

Attention
La variable somme des tailles des fichiers ("total") étant incrémentée dans la boucle et devant être lue en dehors, pour comparer à la taille disponible, il faut éviter la boucle "while" basée sur la commande de lecture de fichier "cat", car cela ouvrirait un sous-shell, et la variable incrémentée dans la boucle reprendrait sa valeur d'avant boucle en sortie (ici la valeur 0, lors de son initialisation juste avant)

Il faudra donc utiliser le format `while read ligne / do .... done < fichier`, qui indique la "source" de la boucle avec le caractère spécial "<".

```
 # ====== phase 1 et 2 en même temps : voir la taille requise/dispo + trouver racine
 # si la variable "fichier" correspond à un fichier (test "-f") et (opérateur "-a") si le fichier peut être lu (test "-r")
 # ici, le test (avant la variable) remplace la condition de comparaison (<, >, ==, !=...) entre deux variables/valeurs
 if [ -f "$fichier" -a -r "$fichier" ]
```

```
 then
 echo "Lecture du fichier playlist"
 # note : si on utilise cat (cat fichier | wile read ligne { } ), un sous-shell est créé et la variable
"total" sera a zéro en sortie de boucle
 # c'est pourquoi on fait while read ligne {  } < fichier, le < indiquant qu'il faut lire "fichier"
 while read ligne
 do
 # si ligne commence par # on ignore, car c'est un commentaire sur la musique.
 # On ne s'intéresse qu'aux lignes contenant les chemins des fichiers musique.
 premiercar=$(echo "$ligne" | cut -c1)
 if [ "$premiercar" != "#" ]
 then
```

Pour savoir si l'on peut effectivement copier sur la destination l'ensemble des fichiers listés dans le fichier playlist m3u fourni en premier paramètre, on va additionner les tailles de ces fichiers.
On obtient la taille d'un fichier avec la commande "`du`" ("disk usage"), ici en kilo-octet avec l'option "-k"

Pour récupérer la taille du fichier, qui est le premier champ de la commande "`du`, on utilise la commande `awk`, en demandant ensuite d'afficher le n-ième champ (`'{ print $i }'`). On ne précise par le séparateur de champs, qui est ici la tabulation, donc le séparateur par défaut de awk.

```
 # === récupération de la taille du fichier en Ko, par la commande "du" (disk usage, ne pas confondre avec
disk free)
 # $(   ) permet d'invoquer directement une commande Linux/Unix, dont le résultat sur la sortie standard
sera renvoyé au script.
 # awk permet de scinder la sortie du "du -k" en plusieurs champs, et récupérer ici le premier par la
commande awk "print $1"
 # awk utilise le séparateur par défaut, à savoir la tabulation
 taille=$(du -k "$ligne" | awk '{ print $1 }')
```

Il y a deux manières d'effectuer un calcul numérique en shell :

- la commande expr : `expr $1 + $2`
- les parenthèses : `($1 + $2)`

```
 # on incrémente la taille totale requise
 total=$(($total + $taille))
```

Pour résoudre le chemin commun à toutes les musiques, on empile dans les cases d'un tableau chaque répertoire du chemin de la première musique dans un tableau, puis pour les lignes suivantes on compare le chemin à parcourir avec celui enregistré dans le tableau.

Dès qu'une divergence est trouvée, un index est mis sur le dernier répertoire commun, qui contient donc le numéro de la case du tableau correspondant.
Lors de la lecture des lignes, on n'ira jamais plus loin que l'index "dernier répertoire commun trouvé pour l'instant"

Afin de découper le chemin (par exemple /media/KastorPollux/Musiques et sons/.....) en répertoires, on utilise la commande awk, en précisant qu'ici les champs sont séparés par "/" grâce à l'option "-F" (Field separator, séparateur de champs), et en demandant ensuite d'afficher le n-ième champ (`'{ print $i }'`).

Ici, vu que l'on doit récupérer un nombre de champs qui est variable, on va d'abord calculer le nombre de champs, puis demander chaque champ un par un dans une boucle.

Par conséquent, il faudra fournir à la commande awk (

```
'{ print $X }') une variable shell, identifiée par un $.
Il faudra donc "casser" la chaîne du print, laquelle est habituellement entourée de ' (guillemet simple),
pour y "glisser" notre variable


 # === découpage et empilage des répertoires dans un tableau, jusqu'à ce que répertoire lu dans ligne !=
case correspondante,
 # auquel cas le répertoire précédent pourra être considéré comme une racine des fichiers musiques,
 # et on indiquera le numéro de case pour ne pas avoir à chercher la racine plus loin à l'avenir
 # NF est le nombre de champs, c'est une variable spéciale d'awk
 nchamp=$(echo "$ligne" | awk -F / '{ print NF }')
 # la première case étant 1, la variable stop ne peut être à zéro que si elle n'est pas initialisée.
 if [ $stop -eq 0 ]
 then
 # on l'initialise donc, et elle décrémentera par la suite au fur et à mesure qu'on "reculera"
 # jusqu'à trouver le premier répertoire commun à toutes les musiques.
 stop=$nchamp
 fi
 if [ "$premiercar" == "/" ]
 then
 # si le premier caractère de la ligne lue dans le fichier playlist est "/",
 #  alors le fichier musique est indiqué en chemin absolu.
 # le séparateur utilisé plus loin pour différencier les répertoires étant "/",
 # le premier "/" sépare donc le premier champ (rien) du deuxième (premier répertoire)
 # et il faudra donc commencer par lire le deuxième champ, vu qu'il n'y en a pas de premier
 # au cas où, le "curseur" de la première boucle (i) et aussi celui de la deuxième boucle (j) sont donc
initialisés à 2
 i=2 # premier champ nul, car on débute par la racine, / (normalement)
 j=2
 else
 # si le premier caractère de la ligne lue dans le fichier playlist n'est pas "/",
 # alors le fichier musique est indiqué en chemin relatif
 #  ce qui peut poser problème, si l'on lance ce script depuis un répertoire différent de celui du fichier
playlist
 #  on ne pourra alors pas trouver les fichiers musique
```

```
 # le séparateur utilisé plus loin pour différencier les répertoires étant "/",
 #  le premier "/" sépare donc le premier champ (rien) du deuxième (premier répertoire)
 # et ici, il y aura donc un premier champ
 # au cas où, le "curseur" de la première boucle (i) et aussi celui de la deuxième boucle (j) sont donc
initialisés à 1
 i=1
 j=1
 fi
 # tant que i est inférieur au nombre de champs (répertoires) à lire dans la ligne
 # ET i est inférieur au nombre de champs intéressants (communs à toute les lignes déjà lues)
 while [ $i -le $nchamp  -a $i -le $stop ]
 do
 # on fournit au print de l'awk une variable pour afficher le ième champ, qui ira dans une variable
 # or, la commande awk traitant ses variables numéro de champs avec le caractère "classique" $
 # il faut donc "casser" la chaîne du print, laquelle est habituellement
 # entourée de ' (guillemet simple, voir plus haut), pour y "glisser" notre variable
 # lors de l'exécution, cela donnera : awk -F / '{ print $2 }'    .... si $i vaut 2, bien sûr
 # l'option -F indique le "field-separator", le caractère de séparation de champs
 chaine=$(echo "$ligne" | awk -F / '{ print $'$i' }')
```

Pour accéder à un tableau en shell, il faut indiquer que la variable est un tableau en l'entourant d'accolades : `${arborescence[$i]}` correspond donc à la i-ème case du tableau "arborescence".
On pourrait aussi avoir un tableau a plus de dimensions. (ex `${arborescence[$i][$j][$k]}`)

```
 # si répertoire lu différent de celui empilé dans le tableau
 # c'est à dire celui situé à la même "profondeur" lu pour le premier fichier musique
 if [ "$chaine" != "${arborescence[$i]}" ]
 then
 # soit le répertoire empile n'est pas nul = le tableau a été initialisé
 #  et auquel cas le répertoire précédent (parent) peut être la racine des fichiers musiques....
 if [ "${arborescence[$i]}" != "" ]
 then
 # .... et donc on n'ira pas plus loin que le répertoire précédent lors de l'analyse des prochaines lignes
 stop=$(expr $i - 1)
 else
 # soit le répertoire empilé et nul et en fait le tableau n'est pas encore initialisé,
 # on lit la première ligne et on empile le répertoire dans le tableau
 arborescence[$i]="$chaine"
 fi
 fi
 # comme c'est un while, il faut surtout ne pas oublier d'incrémenter i, sinon boucle infinie
 i=$(expr $i + 1)
 done
 fi # finsi premier caractère de la ligne n'est pas "#"
 done < $fichier
```

La boucle de la première phase étant finie, on dispose du tableau des répertoires à parcourir et de son index sur la racine de tous les fichiers musique, ainsi que de la taille requise en kilo-octets pour la copie de tous les fichiers musiques listés.

Il ne reste donc plus qu'à "reconstruire" le chemin de la racine des musiques, en séparant les répertoires par des "/", et déterminer s'il y a assez de place disponible sur la destination pour y copier les musiques.

On commence par reconstruire la racine, en lisant le tableau jusqu'à l'index et en empilant les répertoires (listés dans le tableau) dans une variable unique, en les séparant par des "/".

```
# pour l'instant, les répertoires sont dans des cases séparées du tableau.
# on va donc lire ce tableau jusqu'au curseur "stop", qui indique la racine,
# afin de composer le chemin absolu de la racine des fichiers musiques listées dans la playlist
# j initialisé en même temps que i, dans la boucle while précédente
racine=""
# tant qu'on n'est pas sur la case contenant la racine, on compose le chemin
while [ $j -le $stop ]
do
# en empilant les répertoires les uns derrière les autres, séparés par des "/".
# lors de la première itération, "racine" étant vide, on aura "/premier_dossier",
# et à la deuxième itération, "/premier_dossier/deuxieme_dossier"
racine="$racine""/""${arborescence[$j]}"
j=$(expr $j + 1) # et on oublie pas d'incrémenter j, car sinon boucle infinie
done
echo "Racine trouvée : $racine"
```

La taille disponible sur le périphérique/partition destination peut être récupéré par la commande "`df`" ("disk free") en kilo-octet par l'option "-k".
La taille disponible elle-même étant le 4ème champ du résultat de la commande, toujours récupéré avec `awk`.

Avant de demander la taille disponible, un test permettra de s'assurer que l'on vise bien un répertoire. (test "-d" pour "directory", répertoire)

```
echo "Taille totale requise : $total Ko"
# même chose que pour le deuxième if
# le test "si est dossier" ("-d")
if [ -d "$destination" ]
then
# ici on utilise la commande df (disk free) pour avoir des infos sur le périphérique destination,
# y compris la taille libre, en 4ème champ, que l'on récupère avec awk
dispo=$(df -k $destination | tail -n 1 | awk '{ print $4 }')
echo "Taille libre sur $destination : $dispo Ko"
# si la taille dispo est "greater than" (>, supérieur strict) la taille totale requise
if [ $dispo -gt $total ]
```

```
then
echo "Il y a assez de place sur $destination"
```

S'il y a assez de place sur la destination (taille dispo > taille totale requise), on pourra copier.....si l'on a bien sûr le droit d'écriture sur le répertoire.
Le test "-w" permet de s'en assurer.

Pour copier les fichiers, on re-lit une deuxième fois le fichier ligne par ligne, à chaque fois en copiant les répertoires et le fichier musique à partir du répertoire racine déterminé plus haut.

```
# test "si on peut écrire sur destination"
if [ -w "$destination" ]
then
# ====== phase 3 : copie
echo "Copie des fichiers..."
#  on re-lit une deuxième fois le fichier playlist
# même remarque pour cat | while read ligne
while read ligne
do
# si ligne commence par # on ignore
premiercar=$(echo "$ligne" | cut -c1)
if [ "$premiercar" != "#" ]
then
# là on fait l'inverse de la phase d'avant.
# on prend tous les champs à partir de la racine jusqu'au fichier .mp3 (ou autre) en fin de ligne
# afin de copier le fichier musique et ses répertoires
# y compris racine des fichiers musiques si besoin est sur la destination
nchamp=$(echo "$ligne" | awk -F / '{ print NF }')
# on créé le répertoire avant d'y copier le fichier,
# on va donc s'arrêter à l'avant dernier champ, le dernier étant le fichier musique lui-même
fin=$(expr $nchamp - 1)
k=$(expr $stop + 1) # pour copier racine/dossier1/ssdossier1.....
# dossier est initialisé à la racine des fichiers musique, donc k initialisé à racine +1
dossier="${arborescence[$stop]}"
# tant que k "lesser or equal" (<=, inférieur ou égal) à "fin"
while [ $k -le $fin ]
do
# on empile le répertoire dans un chemin à reconstituer sur la destination
# racine/dossier1/dossier2......
chaine=$(echo "$ligne" | awk -F / '{ print $'$k' }')
dossier="$dossier""/""$chaine"
k=$(expr $k + 1)
done
# la musique elle-même est le dernier champ
musique=$(echo "$ligne" | awk -F / '{ print $'$nchamp' }')
# on récupère avec cut le dernier caractère de "destination".
# pour connaitre sa position on utilise la commande unix "word count"
```

```
 # avec option "-m" pour avoir le nombre de caractères
 ncar=$(echo "$destination" | wc -m)
 # la position du dernier caractère est donc :
 lcar=$(expr $ncar - 1)
 derniercar=$(echo "$destination" | cut -c$lcar)
 # si derniercar de destination = /,
 # alors pas besoin d'ajouter un "/" entre la destination et
 # ce qu'il y a à y copier (dossier1/dossier2/musique.mp3)
 # dans le cas contraire, il faudra préciser ce séparateur
 if [ "$derniercar" != "/" ]
 then
 # l'option -p permet d'indiquer au mkdir de créer les dossiers parents,
 # s'ils n'existent pas sur la destination.
 $(mkdir -p "$destination""/""$dossier")
 $(cp "$ligne" "$destination""/""$dossier""/""$musique")
 else
 # l'option -p permet d'indiquer au mkdir de créer les dossiers parents,
 # s'ils n'existent pas sur la destination.
 $(mkdir -p "$destination""$dossier")
 $(cp "$ligne" "$destination""$dossier""/""$musique")
 fi
 fi
 done < $fichier
 echo "Copie terminée !"
 else
 echo "Pas d'accès en écriture sur $destination"
 fi
 else
 echo "Il n'y a pas assez de place sur $destination"
 fi
 else
 echo "$destination n'est pas un répertoire"
 fi
 else
 echo "$fichier n'est pas un fichier"
 fi
 fi
else
 # $# représente le nombre d'arguments lors de l'appel du script
 echo "il faut 2 arguments, et non $#"
fi
```