1 Usage de `find`

`find` est une commande très puissante pour appliquer un traitement quelconque dans toute une sous-arborescence du système de fichiers.

Mon premier conseil est bien sûr de faire « `man find` ». Ces explications préliminaires peuvent cependant vous faciliter la lecture du man.

L'usage le plus fréquent de find est de ce type :

`find` ***adresses critères... commandes...***

où :

adresses est simplement l'ensemble des adresses des racines des sous-arborescences à traiter (le plus souvent, en pratique, une seule adresse).

critères est une suite de conditions à remplir pour faire l'objet d'un traitement. Par exemple :

- « `-name '*.txt'` » ne retiendra que les adresses dont le nom a pour suffixe `.txt` ; remarquer que `'*.txt'` est entre quotes pour éviter que le « `*` » ne soit interprété par le shell car on veut que ce soit le critère `-name` de `find` qui interprète l'étoile à chaque niveau de l'arborescence, et non pas le shell (qui d'ailleurs ne l'interpréterait que dans le répertoire courant depuis lequel on lance la commande).
- « `-newer toto/truc` » ne retiendra que les fichiers plus récents (en date de dernière modification) que le fichier `toto/truc`.
- « `-mtime +300` » ne retiendra que les fichiers dont la date de dernière modification (ou de création) remonte à plus de 300 jours. Par convention, « `-mtime -300` » ne retiendra que les fichiers modifiés depuis moins de 300 jours et « `-mtime 300` » ceux qui ont été modifiés (ou créés) il y a exactement 300 jours.
- « `-atime +150` » ne retiendra que les fichiers dont la date de dernière utilisation (i.e. de dernière lecture) remonte à plus de 150 jours. Les conventions « `-150` » et « 150 » fonctionnent de même.
- « -type d » ne retiendra que les fichiers qui sont des répertoires (autres types bien utiles : `f` pour les « vrais » fichiers et `l` pour les liens symboliques).
- « `-size +2048` » ne retiendra que les fichiers de taille supérieure à 2048 octets ou caractères. Les conventions `-2048` pour une taille inférieure à 2048, ou `2048` pour une taille exactement de 2048 octets fonctionnent de même.
- Si on met plusieurs critères les uns à la suite des autres, c'est par défaut la conjonction (le *et*) des critères. Voir le manuel pour les disjonctions.

commandes sont les commandes qui sont exécutées sur les adresses retenues par les critères précédents. Par exemple :

- « `-print` » se contente d'écrire sur la sortie standard les adresses retenues. C'est la commande par défaut si on ne donne aucune commande dans les arguments du `find`.
- « `-exec basename '{}' ';'` » écrira successivement seulement le `basename` de chaque adresse retenue.
- Plus généralement « `-exec` » est très puissant car on peut lui donner n'importe quelle commande à la place de `basename`, même un shell script écrit par ailleurs, et même avec des arguments. Par convention « '{}' » est successivement remplacé par chacune des adresses retenues et on exécute donc autant de fois la commande que le nombre d'adresses retenues par les critères. Enfin, il faut toujours terminer par un point-virgule, que l'on quote lui aussi pour qu'il ne soit pas interprété par le shell.
- « `-ok rm -f '{}' ';'` » proposera de supprimer chacun des fichiers retenus par les critères du `find` ; pour chaque fichier retenu, si l'utilisateur confirme alors la commande est effectuée, sinon elle ne l'est pas. « `-ok` » fonctionne donc comme « `-exec` » sauf que l'utilisateur confirme ou non pour chacune des adresses retenues.

2 Usage de `rsync`

`rsync` (comme « remote synchronisation ») est une commande qui permet de maintenir une arborescence « miroir » identique à une arborescence source d'origine, aussi bien sur une machine locale qu'au travers du réseau. Il faut bien sûr faire « `man find` » pour en dominer l'usage. Ces explications préliminaires peuvent néanmoins faciliter la lecture du manuel.

On utilise `rsync` sous la forme :

`rsync` ***options...*** ***source*** ***miroir***

Si la source et la destination sont des répertoires sur la machine courante, `rsync` se comporte comme `cp -a`, c'est-à-dire qu'il crée une copie conforme de la source dans le miroir en respectant toutes les dates de dernière modification et dernière lecture. En revanche le gain de temps est *très* conséquent si le miroir diffère peu de la source car `rsync` prend en compte les fichiers identiques et ne les recopie pas. C'est donc un outil idéal pour les sauvegardes régulières.

Bien sûr, pour faire des sauvegardes, la source et le miroir doivent être sur des machines aussi distantes physiquement que possible (e.g. en cas d'incendie). Deux approches sont alors possibles :
— utiliser NFS (Network File System) qui permet de monter la partition du miroir sur la machine de la source au travers du réseau, ou l'inverse ; dès lors ***source*** et ***miroir*** apparaissent comme des répertoires locaux sur toute machine bénéficiant de ces montages NFS
— ou bien utiliser directement `rsync` en lui donnant soit une adresse source distante, soit une adresse miroir distante (mais pas les deux à la fois).
La première solution a l'avantage de permettre à `root` d'utiliser `rsync` directement (selon la paramétrisation choisie du montage NFS), alors que la seconde passe *a priori* par `ssh` et dès lors `rsync` ne peut pas être utilisé par `root` [1].

Pour mettre en place la première solution, il faut que le miroir soit serveur NFS et mette la partition à la disposition de NFS avec les droits de `root` : autant dire que ce n'est généralement possible que si l'on est gestionnaire du miroir et que la source et le miroir sont sur le même sous-réseau.
Sinon on est contraint de passer par la seconde approche, et pour mirorer des répertoires appartenant à plusieurs utilisateurs, il faut que `root` lance `rsync` pour chacun des utilisateurs. La syntaxe des adresses distantes est alors de la forme *`user@machine:répertoire`*. La partie *`user@`* est optionnelle ; si elle est absente, l'utilisateur distant est supposé être le même nom de login que l'utilisateur local.

Les options les plus utiles sont les suivantes :
— l'option `-a` (comme « archive ») indique à `rsync` de reporter intégralement tous les droits d'accès et les dates de dernière modification ou usage de tous les fichiers, option indispensable, donc, pour faire des sauvegardes
— l'option `-v` (voir) rend `rsync` un peu plus verbeux et ça peut être utile si l'on souhaite suivre la progression de la synchronisation
— l'option `-progress` estime un pourcentage de transfert de chaque fichier, donc encore plus verbeux/précis sur la progression de la synchronisation
— l'option `-delete` ou `-delete-after` est nécessaire pour obtenir un réel miroir, car sans elle, les fichiers présents sur ***miroir*** mais absents sur ***source*** ne sont pas détruits !
— l'option `-exclude=`*`motif`* permet d'exclure de la synchronisation tous les fichiers qui filtrent ce *`motif`* au sens des expressions régulières simplifiées ; on peut ajouter dans les arguments autant de `-exclude` que de motifs qu'on souhaite exclure
— enfin l'option `-exclude-from=`*`fichier`* lit les motifs à exclure dans le *`fichier`*, à raison d'un motif par ligne.
Ces deux dernières options sont utiles pour ne pas mirorer des fichiers temporaires, ou qui sont construits automatiquement, ou encore qui sont spécifiques à la machine sur laquelle se trouve les répertoires ***source*** ou ***miroir*** et ne doivent donc pas être écrasés.
Nota il y a des subtilités sur les *`motif`*s de `rsync` ; en voici trois les principales :
— s'il commence par un « / » alors cela indique une adresse à partir des répertoires ***source*** ou ***miroir***
— s'il se termine par un « / » alors le motif ne filtrera que les répertoires
— le « * » ne filtre que les chaînes de caractères qui ne contiennent pas de « / »
Il existe bien d'autres options de `rsync` : faire `man rsync`.

3 Usage de `grep` et de `sed`

`grep` et `sed` sont des commandes qu'on utilise souvent dans des shell scripts.

`grep` extrait de son entrée standard les lignes qui correspondent (ou non) à un motif donné en argument, et les ressort sur sa sortie standard. On peut aussi lui donner un nom de fichier en argument à la place de son entrée standard. `grep` s'utilise principalement sous la forme « `grep` *`motif`* » avec une adresse en argument supplémentaire si l'on veut un fichier à la place de l'entrée standard.
Le *`motif`* est une expression régulière standard mais, avec l'option `-F`, il est possible de recherche une chaîne de caractère fixée, sans l'interpréter comme une expression régulière : « `grep -F` *`chaine`* ». D'autres options très utiles sont `-i` pour ignorer la casse, `-v` pour faire le contraire du comportement par défaut (ne garder que les lignes qui ne filtrent pas le motif)... et il y a bien d'autres options (`man grep`).

1. Certains serveurs peuvent lancer un démon pour `rsync` au lieu de passer par `ssh`, voir `man rsync`.

`grep` est aussi souvent utilisé dans une expression conditionnelle (`if` ou `while` typiquement) car il échoue s'il ne trouve aucune ligne qui réponde au motif.

`sed` (stream editor) est également un grand classique des systèmes Unix. On l'utilise pour éditer automatiquement (transformer) chaque ligne de son entrée standard et il fournit le résultat sur sa sortie standard. Son usage le plus courant est de la forme « `sed -e 'script-1' -e 'script-2'...` » et, pour chaque ligne, les scripts de transformation sont appliqués successivement.
Les *script*s les plus utiles sont de la forme `'adresse commande'` :
— l'*adresse* peut être un numéro de ligne (avec la convention qu'un `$` représente la dernière ligne), ou une expression régulière qu'on place entre deux « / », ou encore quelques autres formats d'adresse moins souvent utilisés...
— la *commande* est alors exécutée sur toutes les lignes qui répondent à l'*adresse*. Les commandes les plus utiles sont `q` pour quitter sans lire la suite, `r fichier` pour ajouter le texte contenu dans le fichier, `d` pour supprimer la ligne, et surtout `s/expr/remplace/` pour remplacer l'expression régulière *expre* par *remplace* au sein de la ligne ;
— enfin on peut placer une négation « `!` » entre l'adresse et la commande (`'adresse ! commande'`), et dans ce cas la commande est exécutée sur les lignes qui ne répondent pas à l'adresse.
Il est possible de ne pas donner d'*adresse* du tout, auquel cas la commande s'applique à toutes les lignes. C'est surtout utile pour `s/expr/remplace/` bien sûr. Si la ligne ne contient aucune sous-chaîne qui filtre *expr*, elle reste inchangée. De plus *expr* peut contenir des sous-expression délimitées par des `\(...\)` et la portion de texte qui les filtre peut être réutilisée dans *remplace* en écrivant `\1` pour la première sous-expression puis `\2`, *etc.*
Là encore, il y a d'autres scripts possibles, faire `man sed`...

4 Un exemple de shell script

Une commande de gestion de la luminosité d'un écran :

```
#! /bin/bash -
#
minimum=15 #pourcent de brightness
pas=5      #incrément ou décrément
maj=4      #dixièmes de secondes entre deux mises à jour
###
cmd="‘basename $0‘"
tmp=/tmp/$cmd$$
#
aide () {
  echo "Usage:  $cmd [pourcentage|+|-]"
  echo
  echo "Le pourcentage doit etre compris entre $minimum et 100."
  echo "Il est demandé par interface graphique (et mis à jour en temps réel) si aucun"
  echo "argument n’est donné."
  echo "Les arguments + ou - augmentent ou diminuent le pourcentage de $pas %."
  exit 1
}
#
ajuste () {
  if [ -n "$p" ]
    then # Calcul de la brightness:
      p="‘expr "$1" + 0‘"
      if [ "$p" -gt 100 -o "$p" -lt $minimum ]
        then aide
      elif [ "$p" -eq 100 ]
        then b="1.0"
      elif [ "$p" -ge 10 ]
        then b="0.$p"
        else b="0.0$p"
      fi
      # On y va:
      for e in $ecrans
      do
        xrandr --output $e --brightness $b
```

```
      done
  fi
}
# paramètres actuels:
ecrans="`xrandr -q | sed -e '/ connected/ ! d' | sed -e 's/\([^ ]*\) .*/\1/1'`"
actuel="`xrandr --verbose -q | sed -e '/Brightness: / ! d' | \
 sed -e '1 ! d' -e 's/ *Brightness: *\([^ ]*\).*/\1/1' -e s/1\..*/100/1 -e 's/0\.\(..\).*/\1/1'`"
actuel=`expr $actuel + 0`
# Analyse des arguments:
case "$1" in
 +|++) p=`expr "$actuel" '+' "$pas"`
   if [ "$p" -gt 100 ]
     then p=100
   fi
   ajuste $p
   ;;
 -|--) p=`expr "$actuel" '-' "$pas"`
   if [ "$minimum" -gt "$p" ]
     then p=$minimum
   fi
   ajuste $p
   ;;
 [0-9]|[0-9][0-9]|[0-9][0-9][0-9]) p="$1"
   ajuste $p
   ;;
 '') msg="Choisir un pourcentage de luminosite :"
   echo $actuel > "$tmp"
   # "stdbuf -oL" réduit buffering de sortie à une ligne max pour alimenter $tmp en temps réel
   (
    stdbuf -oL Xdialog --interval ${maj}00 --stdout --title "$cmd" --no-cancel --rangebox "$msg"\
                                                0 0 $minimum 100 $actuel 2> /dev/null >> "$tmp"
    rm -f "$tmp"
   )&
   while [ -r "$tmp" ]
   do
     p="`tail -1 "$tmp"`"
     ajuste $p
     sleep 0.$maj
   done
   ;;
 *) aide
esac
```

et au passage on note l'intérêt de `/dev/null`, des commandes `read`, `Xdialog` (et de son manuel en pages HTML), *etc.*