```
# 1) separation nom-repertoire
#

fic=/usr/bin/ldd
nom=${fic##*/}
chemin=${fic%/*}

# 2) script formater
#

# Il realise une lecture ligne a ligne sur un fichier texte. La boucle
# s'arrete a la premiere ligne vide : on peut donc s'arreter avant la fin
# de l'entree standard, ce qui n'est pas toujours souhaitable.

# 3) Script formater (suite)
#
# Amelioration par rapport a la version precedente: on s'arrete lorsque
# read echoue (fin des donnees sur l'entree standard)

#
#!/bin/sh
#
while read fic; do
   echo "${fic##*/} / ${fic%/*}"
done


# 4) script filtrer
#
# On doit conserver la partie gauche (1er champ) de la derniere ligne lue
# -> variable dernier ci-dessous. S'il coincide avec la partie gauche de la ligne
# courante, alors c'est un doublon, on l'affiche, et on reitere jusqu'a
# ce que la partie gauche change. Sinon, on poursuit sans rien afficher
#
# Interet de "/" : aucun nom de fichier ne peut contenir de caractere /, donc
# utilise en premiere occurence, " / " separe nom de fichier et repertoire sans
# ambiguite. On peut aussi l'utilser pour initialiser dernier : c'est une
# valeur impossible.

#!/bin/sh
#
dernier=/

while read ligne; do
      nom="${ligne%% / *}"
      if [ "$nom" = "$dernier" ]; then
          echo "$ligne"
      fi
      dernier="$nom"
done



# 5) Script doublons
#

#!/bin/sh
#

if [ $# -lt 1 ]; then
        echo "Utilisation $0 repertoire..."
        exit 1
fi

find $* | formater | sort | filtrer


# 6) Options
# L'idee est de preparer les options qu'on devra passer a find dans une
# variable (findopt) en fonction des options -L et -d qu'on va trouver en
# analysant les arguments. En placant cette variable dans la ligne d'appel
# a find sans la proteger, elle sera remplacee par autant de mots qu'elle
```

```
# contient.
#
# Pour analyser les arguments, il suffit de se limiter a l'argument 1 ($1)
# et de decaler tous les parametres d'un rang avant l'iteration suivante
# grace a shift. Une fois les options consommees, on se retrouve exactement
# dans le meme cas qu'a la question precedente.


#!/bin/sh
#

findopt="-type f"

# Tant qu'on a une option dans le 1er argument
while [ ! "${1#-}" = "$1" ]; do
    case "$1" in
        -L) findopt="$findopt -or -type l" ;;
        -d) findopt="$findopt -or -type d" ;;
        -*) echo "$1 : option invalide" 1>&2; exit 1 ;;
    esac
    shift
done

# A ce niveau, la variable # a ete diminuee d'autant de fois qu'on a
# appele shift : tout se passe donc comme s'il n'y avait jamais eu
# d'options.
if [ $# -lt 1 ]; then
        echo "Utilisation $0 repertoire..."
        exit 1
fi

find $* $findopt | formater | sort | filtrer



# 7) Il suffit d'ajouter la taille du fichier devant son nom dans formater, la
# commande sort faisant le reste du travail (deux fichiers identiques auront
# necessairement meme taille et meme nom, donc leurs lignes seront consecutives
# apres tri).
# Nouvelle version de formater:

#!/bin/sh
#

while read fic; do
    taille=$(stat -c "%s" "$fic")
    echo "$taille + ${fic##*/} / ${fic%/*}"
done

# Remarque : on peut utiliser un delimiteur autre que '/' dans la sortie produite,
# comme '+' par exemple (mais tout caractere hors de 0-9 convient).
# Auquel cas, on peut recrire soit filtrer, soit doublons pour se debarasser de
# l'information de taille, qui n'est pas demandee. Une commande cut suffit
# a faire cela tres facilement.
# Nouvelle version de filtrer (noter qu'on a utilise {} pour eviter toute ambiguite
# de priorite entre le premier pipe et while :

#!/bin/sh
#
dernier=/

cut -d+ -f2- | {
    while read ligne; do
        nom="${ligne%% / *}"
        if [ "$nom" = "$dernier" ]; then
            echo "$ligne"
        fi
        dernier="$nom"
    done;
}
```

```
# Si l'on ne veut pas utiliser cut, on peut aussi se contenter du pattern
# matching du shell:

#!/bin/sh
#
dernier=/

while read preligne; do
    ligne="${preligne#*+ }"
    nom="${ligne%% / *}"
    if [ "$nom" = "$dernier" ]; then
        echo "$ligne"
    fi
    dernier="$nom"
done


# 8) On propose une solution recurrente sans priorisation sous forme de fonction:
# pour chacun des fichiers du repertoire recu en unique argument:
#   si c'est un repertoire, alors relancer la fonction sur lui
#   sinon si c'est un fichier accepte (fichier valide : toujours, lien et repertoire:
#     optionnellement) alors l'afficher
# fin pour
#
# On analyse les options exactement de la meme maniere que dans doubons, mais
# l'analyse marque cette fois-ci les options "on inclue les repertoires" et
# "on inclue les liens" dans deux variables globales separees.
#
# On a interet a utiliser une fonction a l'interieur d'un script pour eviter de
# tester inutilement le nombre d'arguments transmis a chaque recurrrence, ce
# test ayant lieu une fois pour toutes en debut de script.

#!/bin/sh
#


function trouver_rec () {

    for f in "$1"/*; do
        if test -f "$f" -o \( -d "$f" -a "$reps" = oui \) -o \( -h "$f" -a "$liens" = oui \)
            then echo "$f"
        fi
        if test -d "$f"; then
            trouver_rec "$f"
        fi
    done
}



# Debut du script

reps=non
liens=non
while [ ! "${1#-}" = "$1" ]; do
    case "$1" in
        -L) liens=oui ;;
        -d) reps=oui ;;
        -*) echo "$1 : option invalide" 1>&2; exit 1 ;;
    esac
    shift
done


if [ $# -lt 1 ]; then
        echo "Utilisation $0 repertoire..."
        exit 1
fi

trouver_rec "$1"
```