

Sommaire

Aperçu avant impression.....	1
Objectif.....	1
Le fichier /etc/printcap.....	1
La configuration du filtre.....	2
Le script qui prévisualise.....	10
Utilisation.....	13
Conclusion.....	13
Copyright.....	14

Aperçu avant impression

Aperçu avant impression

par Fred

Utiliser le système d'impression de Linux pour offrir une prévisualisation à toutes vos applications

Objectif

Vous avez enfin réussi à configurer votre imprimante, mais vous vous êtes aperçu que beaucoup de vos logiciels n'offrent pas la fonction si pratique de l'aperçu avant impression, et que ceux qui le proposent, n'offrent que des fonctions très approximatives. C'est idiot, car sous Linux, on passe presque toujours par ghostscript pour imprimer quelque chose, et ghostscript est tout à fait capable d'afficher ce qu'il est capable d'imprimer. Qu'à cela ne tienne, nous allons utiliser ghostscript pour gérer l'aperçu avant impression.

Note : ceci ne fonctionne pas avec CUPS (du moins pas en l'état) et avec les filtres d'impression : rhs-printfilters de la RedHat (présents dans toutes les Mandrake). Cet article nécessite sans doute pas mal de travail pour l'adapter à une autre distribution.

Le fichier /etc/printcap

La configuration d'un service d'impression sous Linux passe par la configuration de plusieurs fichiers. Le plus important d'entre eux est /etc/printcap. Ce fichier contient une entrée pour chaque imprimante reliée à votre système. Nous allons en ajouter une autre : virtuelle. Son rôle ne sera pas d'imprimer vraiment, mais d'afficher tout ce que nous lui enverrons. Nous allons donc ajouter à /etc/printcap :

Offrons la prévisualisation à Linux :

```
preview|Ghostview:\
    :sd=/var/spool/lpd/preview:\
    :mx#0:\
    :sh:\
    :lp=/dev/null:\
    :if=/var/spool/lpd/preview/filter:
```

La première ligne indique les différents noms sous lesquels notre nouvelle imprimante sera connue. Pour avoir la prévisualisation du fichier toto, nous ferons :

```
lpr -Ppreview toto
```

En lieu et place de preview, nous aurions pu mettre Ghostview.

La seconde ligne indique où sont stockés les fichiers de notre nouvelle file d'attente.

L'avant dernière ligne indique qu'il ne s'agit pas d'une véritable impression : on envoie tout sur /dev/null. Vu les modifications que nous allons apporter aux autres scripts, il est probable que nous aurions pu mettre n'importe quoi ici.

La dernière ligne indique quel est le script qui se chargera de filtrer ce que nous allons envoyer à "l'imprimante" (en l'occurrence : ghostview).

La configuration du filtre

Pour faire notre office, nous allons utiliser une modification du script d'impression de la Mandrake/RedHat (du paquetage rhs-printfilters qu'il vous faut avoir installé, si vous avez laissé La Mandrake/RedHat installer votre imprimante, c'est déjà fait).

Première étape, créons le script suivant : /var/spool/lpd/preview/filter (c'est une recopie presque brutale du script original master-filter du paquetage rhs-printfilters)

```
#!/bin/bash
#
#
# New smart print filter
#
#
# determines input file magic
#
# looks in the default filter plugin (FPI) directory
# finds all *.fpi files, then finds a combination that will
# yield the desired file type, or will indicate this is impossible.
#

function filtfrom {
    echo -ne ${1%-to-*}
}

function filtto {
    echo -ne ${1#*-to-}
}

#
# given filters as input vars, find next level available given the
# first arg is the starting point
#
function nextlvl {

    local try
    local start
    local all
    local depth

#
#
# $1 is starting point, find something that will work
#
    start="$1"
    shift
```

```

depth="$1"
shift

all="$@"

#
# get out of here if too deep!
#
if [ $depth -ge $MAX_DEPTH ]; then
    return 1
fi
if [ $DEBUG_TREE ]; then
    echo "Starting point = $start" >> /tmp/filter.debug
fi

if [ $(filtto $start) = $DESIRED_TO ]; then
    echo " DONE"
    return 0
fi

while [ $1 ]; do
    try=$1
    shift
    if [ $DEBUG_TREE ]; then
        echo "for $start trying $try" >> /tmp/filter.debug
    fi
    if [ $(filtfrom $try) = $(filtto $start) ]; then
        echo -n "$start.fpi:$depth:CONT "

        if [ $(filtto $try) = $DESIRED_TO ]; then
            echo -n "$try.fpi:$((depth+1)):DONE "
            return 0
        else
            # echo -n $try
            nextlvl $try $((depth+1)) $all
            # echo "|G is $G| "
            if [ $DEBUG_TREE ]; then
                echo "|rt is $?" >> /tmp/filter.debug
            fi
            if [ "$?" = "0" ]
            then
                if [ $DEBUG_TREE ]; then
                    echo "for $start we are done" >> /tmp/filter.debug
                fi
            # return 0
            else
                if [ $DEBUG_TREE ]; then
                    echo "for $start we have failed" >> /tmp/filter.debug
                fi
                return 1
            fi
        fi
    fi

```

```

    fi
  fi
#   echo ""
done
}

#
# MAIN
#
#
#
#
# setup some global variables used by this script
#
#
#
# FPIDIR points to where print filter plug-ins are stored
# Normally these will be installed with a package via RPM
#
FPIDIR=/usr/lib/rhs/rhs-printfilters/

PATH=${FPIDIR}:${PATH}

#
# MAX_DEPTH determines how long a string of filters will be
# tried as a possible printing solution. How many input
# formats will take 6 filters to output Postscript!
# Unlikely this will need to be changed.
#
MAX_DEPTH=6

#
# define these to gets lots of feedback
# output is appended on /tmp/filter.debug
#
DEBUG_TREE=""
DEBUG_FILTER=""

#
# Setup variables available to all filter plug-ins
#
#
#
# SPOOLDIR is directory which lpd is spooling from
#
export SPOOLDIR=$(pwd)

```

Admin-admin_imp-apercu

```
#
# Get queue specific information (which was written by printtool)
#
source ${SPOOLDIR}/general.cfg

if [ "$DEBUG_FILTER" != "" ]; then
    echo "Desired print format is $DESIRED_TO" >> /tmp/filter.debug
    echo "Paper size is $PAPERSIZE" >> /tmp/filter.debug
    echo -n "A form feed will " >> /tmp/filter.debug
    if [ "$SEND_EOF" = "" ]; then
        echo "not be sent." >> /tmp/filter.debug
    else
        echo "be sent." >> /tmp/filter.debug
    fi
fi

cd $FPIDIR
fpis=$(ls *.fpi 2> /dev/null | tr '\n' ' ' | sed 's/\.fpi//g')

#
# let's see if its a compressed file first
#
#
# Figure out the magic of the input file
#
magic=$(file -)
$FPIDIR/rewindstdin
magic=${magic#*: }
if [ "$DEBUG_FILTER" != "" ]; then
    echo "Magic is |$magic|" >> /tmp/filter.debug
fi
case `echo $magic | tr 'A-Z' 'a-z` in
    *bzip2*)
        DECOMPRESS="bunzip2 -d";;
    *bzip*)
        DECOMPRESS="bunzip -d";;
    *packed*|*gzip*|*compress* )
        DECOMPRESS="gzip -dc";;
    * )
        DECOMPRESS="";;
esac

#
# Figure out the magic of the input file
#
if [ "$DECOMPRESS" = "" ]; then
    magic=$(file -)
else
    magic=$(($DECOMPRESS | file -)
fi
$FPIDIR/rewindstdin
```

```

magic=${magic#*: }
if [ "$DEBUG_FILTER" != "" ]; then
  echo "Magic is |$magic|" >> /tmp/filter.debug
fi
case `echo $magic | tr 'A-Z' 'a-z` in
  *empty* )
    exit;;
  "pc bitmap data"* )
    startpnt="INPUT-to-bmp";;
  "gif image data"* )
    startpnt="INPUT-to-gif";;
  "png image data"* )
    startpnt="INPUT-to-png";;
  "jpeg image data"* )
    startpnt="INPUT-to-jpeg";;
  "tiff image data"* )
    startpnt="INPUT-to-tiff";;
  "sun raster image data"* )
    startpnt="INPUT-to-rast";;
  "pgm"*|"pbm"*|"ppm"* )
    startpnt="INPUT-to-pnm";;
  postscript* )
    startpnt="INPUT-to-ps";;
  "PDF document"* )
    startpnt="INPUT-to-pdf";;
  "tex dvi file"* )
    startpnt="INPUT-to-dvi";;
  "fig image text"* )
    startpnt="INPUT-to-fig";;
  "troff or preprocessor"* )
    startpnt="INPUT-to-troff";;
  "rpm"* )
    startpnt="INPUT-to-rpm";;
  *ascii*|*text*|*english*|*script* )
    startpnt="INPUT-to-asc";;
  *data*|*escape* )
    startpnt="INPUT-to-prdata";;
  *pcl* )
    startpnt="INPUT-to-prdata";;
  "sgi image"* )
    startpnt="INPUT-to-sgi";;
  "kodak photo cd"* )
    startpnt="INPUT-to-pcd";;
  "fits image"* )
    startpnt="INPUT-to-fits";;
  "fit image"* )
    startpnt="INPUT-to-fit";;
  "iff image"* )
    startpnt="INPUT-to-ilbm";;
  *iff*ilbm* )
    startpnt="INPUT-to-ilbm";;

```

Admin-admin_imp-apercu

```
"rle image"* )
    startpnt="INPUT-to-rle";;
"X pixmap image"* )
    startpnt="INPUT-to-xpm";;
"fbm image"* )
    startpnt="INPUT-to-fbm";;
* )
    startpnt="INPUT-to-unknown";;
esac

#
# here is where we could put in hook to call user routine(s) to
# handle extra magics they've defined filters for
#
# call_user_magic_hook()
#
if [ "$DEBUG_FILTER" != "" ]; then
    echo "Type of file is $startpnt" >> /tmp/filter.debug
fi

if [ "$startpnt" = "Dont know" ]; then
    echo "Error - input file type is unknown - cannot print"
    exit 1
fi

#
# catch some easy cases without having to figure out best path the hard way
#
bestpath=""
foundbest="NO"
if [ $(filtto $startpnt) = "asc" ]; then
    if [ "$ASCII_TO_PS" = "NO" ]; then
        bestpath="$startpnt | asc-to-printer.fpi"
        foundbest="YES"
    fi
elif [ $(filtto $startpnt) = "prdata" ]; then
    bestpath="$startpnt | cat -"
    foundbest="YES"
elif [ $(filtto $startpnt) = $DESIRED_TO ]; then

# envoyons les données à l'écran plutôt qu'à l'imprimante :
bestpath="$startpnt | $DESIRED_TO-to-x11.fpi"

    foundbest="YES"
fi

if [ "$foundbest" != "YES" ]; then
#
```

```

# we go through and find best path
#
G=`nextlvl "$startpnt" "0" $fpis`

if [ "$DEBUG_FILTER" != "" ]; then
    echo "$G" >> /tmp/filter.debug
fi

#
# now sort out the best path of all available
#
#
# if no processing required, depth will equal 'DONE'
#
if [ "${G#*}" != "DONE" ]; then
    root=""
    bestdepth=$((MAX_DEPTH*2))
    bestpath=""
    curdepth="0"
    depth="0"
    foundbest="NO"
    for i in $G; do
        entry=${i%%:*}
        depth=${i#*:}
        depth=${depth%:*}
        if [ $depth -le $curdepth ]; then
            while [ (($depth <= $curdepth && $curdepth >= 0)) -eq 1 ]; do
                root=${root%* | *}
                curdepth=$(( $curdepth - 1 ))
            done
        fi
        if [ (($curdepth < 0)) -eq 1 ]; then
            root=""
        fi
        curdepth=$depth
        if [ "$root" = "" ]; then
            root="$entry"
        else
            root="$root | $entry"
        fi
        if [ ${i##*:} = "DONE" ]; then
            if [ "$DEBUG_FILTER" != "" ]; then
                echo "$root -> depth = $depth" >> /tmp/filter.debug
            fi
            if [ $depth -lt $bestdepth ]; then
                foundbest="YES"
                bestdepth=$depth
                bestpath=$root
            fi
        fi
    fi
fi

```

```

done
fi

if [ "$foundbest" = "YES" ]; then

# envoyons les données à l'écran plutôt qu'à l'imprimante :
bestpath="$bestpath | $DESIRED_TO-to-x11.fpi"

fi
#
# end of doing it the hard way
#
fi
#
# we have to add filter to convert desired format to something the
# printer can handle. May be as simple as 'cat'
#
#
# ok we got here, and if input data magic is 'data' we'll let it
# through, hoping it really will work on the printer!
# Note we still reject lots of magics, like ELF, by doing this
# which is what we want
#
#
# getting bad, but trapping all "special" cases here
#
#
if [ "$foundbest" = "NO" ]; then
    printf "No way to print this type of input file: $magic \014"
    exit 0
else
#
# fix up the best path so we can run it
#
    if [ "$DECOMPRESS" = "" ]; then
        bestpath="cat - ${bestpath#* }"
    else
        bestpath="$DECOMPRESS ${bestpath#* }"
    fi
fi
#
# any post-filter to run (like smbclient?)
#
if [ "$PRINTER_TYPE" = "SMB" ]; then
    bestpath="$bestpath | ${FPIDIR}/smbprint ${SPOOLDIR}/acct"
elif [ "$PRINTER_TYPE" = "NCP" ]; then
    bestpath="$bestpath | ${FPIDIR}/ncpprint ${SPOOLDIR}/acct"
fi

```

```

if [ "$DEBUG_FILTER" != "" ]; then
    echo "Best path of depth $bestdepth is $bestpath" >> /tmp/filter.debug
fi

#
# run the command!
#

eval $bestpath 2>/dev/null

#
#
# see if we need to send a form feed to eject the page from printer
#
# if [ "$SEND_EOF" != "" ]; then
#     printf "\014"
# fi

exit 0

```

Ce script n'est qu'une modification minimale du script standard. Il n'est pas nécessaire de comprendre le script, en tout cas moi je ne le comprends qu'en partie. La seule chose importante, c'est de savoir que le script standard essaie de reconnaître le type des données qui lui sont transmises, puis les envoie à un autre script : ps-to-printer.fpi. La modification que j'ai apportée, c'est d'envoyer les données vers ps-to-x11.fpi (voir ci-dessous) qui est une modification de ps-to-printer.fpi pour envoyer les données vers ghostview plutôt que vers l'imprimante. Pour le reste, je n'ai rien changé à ce script.

Remarque : les modifications du script sont en **gras**.

Le script qui prévisualise

Voyons maintenant le script chargé de l'affichage de nos données /usr/lib/rhs/rhs-printfilters/ps-to-x11.fpi :

```

#!/bin/sh
#
# convert ps to the format required by the printer on this queue
#
# if the printer is a PostScript printer, just cat it through
# if the printer uses ghostscript, we'll run it now
# if the printer is neither, we die (with good message to someone)
#
#
# read in PostScript configuration settings
#
source ${SPOOLDIR}/postscript.cfg

```

Admin-admin_imp-apercu

```
#
# see if we should reverse order
#
# support will be added for this in the future
# psorder needed and is part of netatalk, which we dont currently ship
#

if [ "$PAPERSIZE" = "letter" ]; then
    mpage_paper="Letter"
elif [ "$PAPERSIZE" = "a4" ]; then
    mpage_paper="A4"
    ppaA4="-s a4"
elif [ "$PAPERSIZE" = "legal" ]; then
    mpage_paper="Legal"
else
    mpage_paper="Letter"
fi

#
# weird case – some PS doesnt get handled by mpage well
# so we allow nup=1 to just cat PS to printer w/o mpage interferring
#
if [ "$NUP" = "1" ]; then
    border="-o"
    mpage_cmd="cat -"
else
    border=""
    mpage_cmd="mpage -b$mpage_paper $border -$NUP -m${RTLFTMAR}lr -m${TOPBOTMAR}tb"
fi

TMPFILE=`mktemp /tmp/printtmp.XXXXXX`

#
# if the driver is
# "POSTSCRIPT"    it means the printer handles Postscript natively,
#                 no need to run gs.
# "TEXT"          it means the printer cannot handle PS input
# "uniprint"      use the driver set by COLOR in postscript.cfg
# "hp720/820/1000" use the pbm2ppa converter; these are for the GDI
#                 printers of the HP720/HP820/HP1000 families.
#

# envoyons les données à l'écran plutot qu'à l'imprimante :
eval "$mpage_cmd | sed 's/[[:%:.*%]].*flush/g\' > $TMPFILE"
#
# ici vous devez initialiser DISPLAY à la valeur correspondant à votre
# écran !
export DISPLAY=:0
gv -media $PAPERSIZE $TMPFILE
rm -f $TMPFILE
```

Le script qui prévisualise

```
#  
#  
# see if we need to send a form feed to eject the page from printer  
#  
if [ "$PS_SEND_EOF" = "YES" ]; then  
    printf "\004"  
fi  
exit 0
```

Pour que tout fonctionne, il nous faut encore quelques fichiers de configuration :
var/spool/lpd/preview/general.cfg

```
#  
# General config options for printing on this queue  
# Generated by PRINTTOOL, do not modify.  
#  
export DESIRED_TO=ps  
export PAPERSIZE=a4  
export PRINTER_TYPE=LOCAL  
export ASCII_TO_PS=YES  
var/spool/lpd/preview/postscript.cfg
```

```
#  
# configuration related to postscript printing  
# generated automatically by PRINTTOOL  
# manual changes to this file may be lost  
#  
GSDEVICE=x11  
RESOLUTION=72x72  
COLOR=  
PAPERSIZE=a4  
EXTRA_GS_OPTIONS=  
REVERSE_ORDER=NO  
PS_SEND_EOF=NO
```

```
#  
# following is related to printing multiple pages per output page  
#  
NUP=1  
RTLFTMAR=18  
TOPBOTMAR=18  
var/spool/lpd/preview/textonly.cfg
```

```
#  
# text-only printing options for printing on this queue  
# Generated by PRINTTOOL, do not modify.  
#  
TEXTONLYOPTIONS=  
CRLFTRANS=YES
```

TEXT_SEND_EOF=YES

Certains de ces fichiers ne sont certainement pas très utiles (en particulier, textonly.cfg mais ça ne coûte pas cher de les mettre, et ça m'évite de comprendre complètement comment tout cela fonctionne).

Veillez à ce que tous ces fichiers soient lisibles par tout le monde, faites les appartenir à root ou daemon et tout sera parfait.

(par un `chown root:root le_fichier ; chmod 644 le_fichier`)

À partir de maintenant vous disposez d'une nouvelle file d'impression : preview qui permet de prévisualiser ce que vous lui envoyez.

Utilisation

Dans vos programmes il suffit de remplacer

`lpr -Pimprimante`
ou

`lpr`
par

`lpr -Ppreview`

et c'est tout ! Maintenant, tous vos programmes qui savent imprimer, savent aussi afficher un aperçu avant impression ! Le seul problème, c'est que tout ceci n'est pas trop prévu pour fonctionner dans un réseau. Il faut faire toutes ces modifications pour chaque poste. Il y a probablement un problème quand on lance une prévisualisation depuis un poste qui n'est pas à la fois le client et le serveur X : la variable `$DISPLAY` n'est sans doute pas initialisée correctement. A vous d'expérimenter tout cela et de m'envoyer vos modifications.

Conclusion

Une nouvelle fois, avec un peu d'astuce, la lisibilité des fichiers de configuration de Linux et les multiples possibilités de configuration de Linux nous ont permis d'obtenir un résultat en peu de temps, pour que l'on soit prêt à mettre les mains dans le cambouis. On peut obtenir tout ce que l'on veut de Linux : il suffit de lui demander gentiment. Bonne configuration !

Cette page est issue de la documentation 'pré-wiki' de Léa a été convertie avec HTML::WikiConverter. Elle fut créée par Frédéric Bonnaud le 01/05/2000.

Copyright

Copyright © 01/05/2000, Frédéric Bonnaud



*Ce document est publié sous licence Creative Commons
Attribution, Partage à l'identique, Contexte non commercial 2.0 :
<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>*