1 Usage de `tar`

`tar` (comme « tape archive ») est une commande qui permet d'archiver dans un seul fichier toute une arborescence. Les explications suivantes vous faciliteront la lecture du `man` de `tar`.

`tar` possède trois options principales :
— `-c` pour créer un fichier d'archive à partir des répertoires à archiver
— `-x` pour extraire les répertoires archivés à partir d'un fichier d'archive
— `-t` (table of contents) pour voir le contenu d'un fichier d'archive

Pour **créer** un fichier d'archives, la ligne de commande est de la forme

`tar -cαf` *`fichierArchive adresse_1 adresse_2 ...`*

où α est en fait un ensemble d'options annexes « modificatrices » du comportement de `tar` et les *`adresse_i`* sont les répertoires ou fichiers à archiver dans le fichier *`fichierArchive`*. Les options modificatrices les plus courantes sont :
— `v` (verbose) pour que `tar` indique chaque fichier qu'il archive
— `j` pour que le *`fichierArchive`* soit automatiquement compressé par `bzip2` (sans compression on a coutume de donner le suffixe `.tar` au fichier, avec compression on lui done le suffixe `.tar.bz2`)
— `z` pour une compression plus faible mais plus rapide avec `gzip` (suffixe habituel `.tar.gz` ou `.tgz`)
— `p` pour préserver tous les droits d'accès et les propriétaires et groupes des fichiers archivés. Cette option annexe est utile pour `root` car cela permet de sauvegarder puis restaurer une arborescence sans changer le propriétaire et le groupe de chaque fichier. Elle est moins utile pour un utilisateur « standard » puisqu'il ne peut intrinsèquement créer que des fichiers qui lui appartiennent ! (même si l'archive d'origine archive les fichiers d'un autre utilisateur).
— de plus l'option `--atime-preserve` évite de mettre à jour la date de dernière utilisation des fichiers archivés (bien que `tar` les lise, naturellement)

et le `f` indique que `tar` devra placer le résultat dans le nom de fichier qui suit (ici *`fichierArchive`*), au lieu de sa sortie standard.

Ainsi par exemple pour sauvegarder `/home`, `root` peut taper les commandes :

```
cd /home
tar --atime-preserve -cvjf /sauvegardes/home.tar.bz2 .
```

Le point en dernier argument du `tar` après avoir fait `cd /home` permet d'avoir une archive où les adresses de fichier ne commencent pas par `home` : cela permet le cas échéant de restaurer les homedirs dans n'importe quel autre répertoire.

Ensuite il faut évidemment transférer ou copier le répertoire `/sauvegardes` en lieu sûr... et ne pas laisser lisibles les fichier d'archive par n'importe qui (mode 600 ou 400 donc) car sinon un utilisateur qui ferait une extraction de l'archive pourrait ainsi aller lire tous les fichiers d'autres utilisateurs ! Plus : en fait il faut chiffrer l'archive (par convention si *fichierArchive* vaut « - », c'est la sortie standard de `tar` ; il suffit alors de « piper » cette sortie standard dans un programme de chiffrement).

Pour **restaurer** un fichier d'archives, la ligne de commande est de la forme

`tar -xαf fichierArchive`

avec les mêmes options modificatrices les plus courantes que pour `-c`.

Par exemple pour restaurer `/home` à la suite d'un crash, `root` peut taper les commandes suivantes :

```
cd /home
for name in *
do
  mv $name $name.damaged
done
tar -xvjf /sauvegardes/home.tar.bz2
```

Et s'il s'agit simplement de vérifier le contenu de l'archive :

```
tar -tvjf /sauvegardes/home.tar.bz2
```

donnera la liste des fichiers de l'archive sur la sortie standard.

2 Scripts shell et interfaces graphiques

Les environnements graphiques permettent d'associer des commandes à des combinaisons de clefs. Ces associations sont souvent appelées « raccourcis d'applications » ou similaire et un menu permet de déclarer la combinaison de clefs et la commande qu'on y associe. Souvent, les clefs « méta » se nomment `Shift`, `Ctrl`, `Super` ou `Meta`, `Alt` et on les associent avec une touche « standard » (lettres de l'alphabet, ponctuations ou autres). Par exemple il est standard d'associer la combinaison de clefs `Ctrl Alt Suppr` à la commande `reboot`. L'énorme avantage de ces raccourcis est d'éviter de parcourir de longues hiérarchies de menus pour trouver une commande qu'on utilise souvent, ou encore de créer de nouvelles fonctionnalités du window manager (*e.g.* transporter la fenêtre courante entre les bureaux, ou retailler une fenêtre selon des critères personnels, *etc.*).
On peut bien sûr associer un shell script à une combinaison de clefs, de sorte que la seule limite à l'ergonomie de votre environnement graphique est votre imagination ! Il est alors utile de mettre en place au sein de ces shell scripts des dialogues qui ne se limitent pas aux entrée-sortie standard du processus.

Les packages `Xdialog` et `zenity` sont précisément faits pour avoir des interactions en mode graphique avec l'utilisateur. Le choix d'utiliser l'un ou l'autre est essentiellement affaire de goût.

On trouve un manuel complet et illustré de `Xdialog` ici : `http://xdialog.free.fr/doc/syntax.html` et pour `zenity` c'est ici : `https://help.gnome.org/users/zenity/stable/index.html.fr`
Voici quelques exemples de dialogues graphiques qu'on peut lancer à partir d'un script shell :

Nota : il est aussi possible de transformer un script shell non graphique en une application « trivialement » graphique. On peut en effet le lancer dans une fenêtre graphique indépendante dédiée à ce script. C'est particulièrement utile lorsqu'on veut pouvoir utiliser le script aussi bien depuis un shell interactif que depuis un menu ou une association de clefs. Il suffit de commencer le script lui-même par quelque chose du genre :

```
if ! tty -s
  then exec xfce4-terminal -e "$0 $*"
fi
...(suite du script en mode non graphique)...
```

Dans ces lignes, « `tty -s` » échoue si le processus qui l'appelle n'est pas issu d'un terminal. Dans ce cas, on lance donc une fenêtre de terminal (ici `xfce4-terminal` mais c'est une question de goût) et l'option `-e` lance `$0` (c'est-à-dire le shell script lui-même) au lieu du shell standard de l'utilisateur. Le shell script se retrouve donc « encapsulé » dans une fenêtre de terminal et bénéficie ainsi du clavier en entrée standard et de la fenêtre en sortie standard et d'erreur. En revanche si on lance la commande depuis une fenêtre de terminal, `tty -s` n'échoue pas et le script conserve les entrée et sorties du processus appelant.

Pour compléter le pilotage de l'interface graphique par le shell, il est bon de ne pas oublier la commande `wmctrl` (Window Manager Control). Elle permet de programmer des actions sur les fenêtres, qu'on fait habituellement à la main avec la souris. On peut épingler, maximiser, maximiser horizontalement ou verticalement une fenêtre avec l'option `-b`, fermer une fenêtre (option `-c`), changer son emplacement et sa taille ((option `-e`), *etc.* On peut même changer les barres de tâches et modifier de fond en comble l'apparence de l'environnement graphique.
Faire évidemment `man wmctrl` pour en savoir plus.

En complément, la commande `xdotool` permet de piloter le clavier et la souris en lignes de commandes. On peut par exemple assigner à une touche une chaîne de caractère entière que l'on tape souvent (e.g. son adresse mail [1]). On peut

1. mais surtout pas un mot de passe évidemment !

aussi déplacer le pointeur de la souris où l'on veut, et bien d'autres manipulations automatiques... Cependant il faut d'abord installer le package `xdotool` qui est rarement installé par défaut (puis faire `man xdotool`).

3 Un exemple de script avec Xdialog et usage de tableaux

Rappel du cours n°4 et petit exemple montré en temps réel en cours...

4 Les commandes les plus classiques

Voir Annexe 1 pour une palette plus large (et utiliser `man` bien sûr !).

- `alias` : permet de donner des diminutifs à des commandes souvent utilisées. `alias` *nom*="*vraie commande*" Essentiellement utilisé dans le fichier `$HOME/.bashrc`
- `basename` et `dirname` : respectivement de nom du dernier lien dans une adresse et le nom de son répertoire. Par exemple : `basename` appliqué à l'adresse `/toto/tutu/titi.txt` retourne `titi.txt` alors que `dirname` retourne `/toto/tutu`.
- `bzip2` : algorithme de compression de données remarquablement efficace (déjà mentionné à l'occasion l'option `-j` de `tar`).
- `evince`, `xpdf`, `epdfview`, `okular`... : outils de visualisation et impression de fichier PDF (versions open-source fiables de « readers » bien connus qui présentent souvent des trous de sécurité tant on ne sait pas ce qu'ils font). Ces commandes peuvent être utilisées dans un shell script graphique pour faire apparaître un fichier PDF.
- `file`, `mimetype` : détermine le type d'un fichier en explorant le début de son contenu réel (`file`) ou seulement le suffixe de son nom (`mimetype`).
- `gimp` : programme très puissant de manipulation d'images. Vaut le détour !
- `latex` ou `pdflatex` : un logiciel professionnel de formatage de texte (qualité d'un livre). L'idée est qu'au lieu de formater à la main visuellement la mise en page, on indique dans un fichier texte ce que l'on veut d'un point de vue logique (ici un nouveau paragraphe, ceci est une figure, ici une section ou une sous-section, un chapitre, *etc.*) et le programme `latex` fait la mise en page en appliquant les règles de mise en page des éditeurs professionnels.
- `make` : outil de compilation de gros logiciels.
- `mkdir` (pour rappel) : crée un répertoire
- `mount` : (pour rappel) permet de rattacher une partition à l'arborescence du système de fichiers.
- `rmdir` : supprime un répertoire s'il est vide, indique une erreur sinon.
- `shutdown` et `halt` : éteignent l'ordinateur. Voir également `reboot` et `who` (car mieux vaut ne pas éteindre un ordinateur sans prévenir ceux qui sont en train de l'utiliser !).
- `/bin/su` : pour « passer `root` » et plus généralement pour lancer un processus au nom d'un autre utilisateur. Sur une machine partagée, supprimez `sudo` qui donne par défaut les droits de `root` à tous les utilisateurs ! ou *a minima* le paramétrer pour le restreindre aux seuls véritables responsables de la machine. *Note* : c'est une bonne habitude d'appeler la commande `su` par son adresse absolue...
  L'option `-c` de `su` est particulièrement utile dans un shell script, surtout s'il est prévu pour être utilisé par `root`. Elle permet de lancer une commande au nom d'un utilisateur (et si elle est appelée par `root`, elle ne demande pas le mot de passe de l'utilisateur en question). Cela permet entre autres de passer par `ssh`, de faire des suites de commandes au sein du shell sans prendre le risque de leur donner les droits de `root`, *etc.*
- `top` est plus sophistiqué que `ps` : il fournit en temps réel la liste des processus les plus gourmands en puissance de calcul (ordre décroissant) de l'ordinateur. On en sort avec la touche « `q` ». Très pratique pour comprendre quels processus sont éventuellement passés hors de contrôle.
- `touch` : positionne les dates de dernière modification et de dernière utilisation d'un fichier à l'instant présent.
- `unrar` : extrait les archives de format « `rar` ». Il s'agit d'un format propriétaire, donc une commande `rar` serait illégale. Accessoirement ce format est celui des comic books CBR ; voir `zip`.
- `which` : prend en argument un nom de commande et explore la variable `$PATH` pour fournir l'adresse de son fichier exécutable. Échoue s'il ne trouve pas la commande.
- `who` : fournit la liste des utilisateurs connectés sur la machine ; utile par exemple pour vérifier que personne ne soit connecté avant de rebooter la machine.
- `whoami` : fournit le nom de login de l'utilisateur qui a lancé ce processus. Ça paraît idiot, mais quand on est `root` sur un parc de machines, on peut jongler avec plusieurs identités et il est bon de vérifier « qui on est » avant de lancer certaines commandes.
- `xterm`, `kterm`, `xfce4-terminal` ou autre « fenêtre de terminal » : fait apparaître dans une fenêtre graphique un espace textuel qui permet entre autres d'utiliser le shell.

— `zip` et `unzip` : moins riche en options que `tar` mais utile pour échanger des archives avec les windosiens qui n'ont ni `tar` ni `bzip2`. Accessoirement, ziper un ensemble d'images produit directement un comic book au format CBZ, il est donc très facile d'éditer un comic book.