et *Mike Miller*
Traducteur : *Dimitri Ara*

HOWTO Comment migrer de VMS à Linux

v1.1.3, 17 Septembre 1999

Ce HOWTO est destiné à toutes les personnes qui utilisent VMS et qui ont maintenant besoin ou envie de passer à Linux, le clone libre d'Unix. Nous effectuerons la transition — je l'espère, sans douleur — en comparant les commandes et les outils disponibles sur ces deux systèmes.

Contents

1 Introduction

1.1 Pourquoi Linux ?

Vous avez entendu dire qu'Unix est difficile et vous êtes hésitant à la perspective d'abandonner VMS ? Pas de panique. Linux, l'un des meilleurs clones d'Unix, n'est pas plus difficile à utiliser que VMS. En fait, je le trouve même plus facile. D'ailleurs, la plupart des gens trouve Linux beaucoup plus puissant et versatile que VMS (évidemment les aficionados de VMS ne sont pas de cet avis).

Linux et VMS sont tous les deux de bons systèmes d'exploitation et accomplissent essentiellement les mêmes taches. Cependant, les outils de Linux sont à mon humble avis supérieurs. Leur syntaxe est souvent plus concise et ils ont souvent les quelques fonctionnalités de plus qui font la différence et permettent de gagner du temps (vous entendrez souvent que VMS et Unix ont des philosophies différentes). De plus, Linux est disponible sur les PC alors que ce n'est pas le cas de VMS (les derniers PC étant d'ailleurs plus puissants que les VAX). Et, cerise sur le gâteau, les excellentes performances des nouvelles cartes graphiques transforment votre tite boîte Linux, grâce à X, en une puissante station de travail graphique bien souvent plus rapide qu'une machine spécialement prévue pour cette tâche.

J'ai plusieurs raisons de croire que la combinaison Pentium/Linux est préférable à celle de VAX/VMS, mais ces préférences sont strictement personnelles et vous ne serez peut-être pas d'accord. Vous en déciderez de vous-même dans quelques mois.

Je prends en hypothèse que vous êtes une étudiant ou un chercheur à l'université et que vous utilisez régulièrement VMS pour les tâches suivantes :

- écrire des papiers avec TeX/LaTeX ;
- programmer en Fortran ;
- faire des graphiques ;
- utiliser Internet ;
- etc.

Dans la section suivante je vais vous expliquer comment faire ces tâches sous Linux en utilisant votre expérience de VMS. Mais avant tout vérifiez que :

- Linux et le *Système X Window* sont correctement installés ;
- vous avez un administrateur système pour s'occuper des détails techniques (s'il-vous-plaît, demandez de l'aide à eux, pas à moi :-) ;
- votre *shell* — l'équivalent de `DCL` — est `bash` (demandez à votre administrateur).

Notez que ce HOWTO n'est pas suffisant pour faire de vous un linuxien pur et dur : il contient seulement le strict nécessaire pour vous permettre de commencer. Vous devriez en apprendre plus sur Linux pour pouvoir en tirer le maximum (fonctionnalités avancées de `bash`, programmation, expressions régulières, etc.).

Les documents du *Linux Documentation Project* (projet de documentation de Linux), disponibles sur *metalab.unc.edu:/pub/Linux/docs/LDP* , sont une importante source d'informations.

NDT : je vous suggère également de lire le *Guide du rootard* d'Éric Dumas et plus généralement tout ce que l'on peut trouver sur *http://www.freenix.fr/linux* et *http://www.traduc.org* .

Et maintenant, c'est parti !

1.2 Commandes et fichiers comparables

Ce tableau compare les commandes les plus utilisées sous VMS et Linux. Gardez à l'esprit que leur syntaxe est souvent très différente ; pour plus de détails allez faire un tour dans les sections suivantes.

```
VMS                             Linux                   Notes
------------------------------------------------------------------------------
@COMMAND                        command                    (doit être exécutable)
COPY fichier1 fichier2          cp fichier1 fichier2
CREATE/DIR [.répertoire]        mkdir répertoire           (seulement un par un)
CREATE/DIR [.rép1.rép2]         mkdirhier rép/rép
DELETE fichier                  rm fichier
DIFF fichier1 fichier2          diff -c fichier1 fichier2
DIRECTORY                       ls
DIRECTORY [...]fichier          find . -name fichier
DIRECTORY/FULL                  ls -al
EDIT fichier                    vi fichier,            (vous n'allez pas l'aimer)
                                emacs fichier,                    (compatible EDT)
                                jed fichier                (idem --- mon préféré)
FORTRAN prog.for                g77 prog.f,        (pas besoin de lier avec LINK)
                                f77 prog.f,
                                fort77 prog.f
HELP commande                   man commande     (la commande doit être précisée)
                                info commande
LATEX fichier.tex               latex fichier.tex
LOGIN.COM                       .bash_profile,                    (fichier caché)
                                .bashrc                                    (idem)
LOGOUT.COM                      .bash_logout                               (idem)
MAIL                            mail,                                 (un peu cru)
                                elm,                              (beaucoup mieux)
                                pine                             (encore meilleur)
PRINT fichier.ps                lpr fichier.ps
PRINT/QUEUE=laser fichier.ps    lpr -Plaser fichier.ps
PHONE utilisateur               talk utilisateur
RENAME fichier1 fichier2        mv fichier1 fichier2      (ne marche pas avec des
                                                              fichiers multiples)
RUN progname                    programme
SEARCH fichier "motif"          grep motif fichier
SET DEFAULT [-]                 cd ..
SET DEFAULT [.rép.rép]          cd rép/rép
SET HOST machine                telnet machine,            (pas exactement pareil)
                                rlogin machine
SET FILE/OWNER_UIC=paul         chown paul fichier        (complètement différent)
SET NOBROADCAST                 mesg
SET PASSWORD                    passwd
SET PROT=(perm) fichier         chmod perm fichier       (complètement différent)
SET TERMINAL                    export TERM=         (la syntaxe est différente)
```

```
SHOW DEFAULT                  pwd
SHOW DEVICE                   du, df
SHOW ENTRY                    lpq
SHOW PROCESS                  ps -ax
SHOW QUEUE                    lpq
SHOW SYSTEM                   top
SHOW TIME                     date
SHOW USERS                    w
STOP                          kill
STOP/QUEUE                    kill,                  (pour les processus)
                              lprm             (pour supprimer un travail
                                                de la file d'impression)
SUBMIT commande               commande &
SUBMIT/AFTER=durée commande   at durée commande
TEX fichier.tex               tex fichier.tex
TYPE/PAGE fichier             more fichier
                              less fichier                (beaucoup mieux)
```

Bien sûr, les différences des deux systèmes ne se limitent pas aux noms des commandes. Continuez donc à lire.

2 Petite introduction

Voila ce que vous devez absolument savoir avant de vous loguer pour la première fois. Détendez-vous, il y a relativement peu de chose.

2.1 Fichiers

- Les noms des fichiers sous VMS on la forme `fichier.extension.version`. Sous Linux, le numéro de version n'existe pas (c'est une grosse limitation mais on peut la compenser par d'astucieux moyens : jetez un oeil à la section 10.2 (Numéros de version sous Linux)) ; les noms des fichiers sont normalement limités à 255 caractères et peuvent contenir autant de points que vous le désirez. Par exemple, `C_est.un.FICHIER.txt` est un nom de fichier valide.
- Linux fait la distinction entre les majuscules et les minuscules. Ainsi, `FICHIER.txt` et `fichier.txt` sont deux fichiers différents et `ls` est une commande alors que `LS` n'en est pas une.
- Un fichier dont le nom commence par un point est un fichier caché (ce qui veut dire qu'il ne sera normalement pas affiché quand on listera les fichiers du répertoire) alors qu'un fichier dont le nom finit par un tilde (« ˜ ») représente une sauvegarde de fichier (ou *backup*).

Maintenant, voici un tableau présentant les correspondances entre les commandes de VMS et celle de Linux en ce qui concerne la gestion des fichiers.

```
VMS                                     Linux
----------------------------------------------------------------------

$ COPY fichier1.txt; fichier2.txt;      $ cp fichier1.txt fichier2.txt
$ COPY [.rép]fichier.txt;1 []           $ cp rép/fichier.txt .
$ COPY [.rép]fichier.txt;1 [-]          $ cp rép/fichier.txt ..
$ DELETE *.dat.*                        $ rm *dat
```

```
$ DIFF fichier1 fichier2              $ diff -c fichier1 fichier2
$ PRINT fichier                       $ lpr fichier
$ PRINT/queue=imprimante fichier      $ lpr -Pimprimante fichier
$ SEARCH *.tex.* "géologie"           $ grep géologie *tex
```

Regardez en plus loin dans le document pour avoir d'autres exemples. Si vous voulez vous attaquer aux notions de droits, de propriétaires et aux sujets avancés, reportez vous à la section 8 (Sujets avancés).

2.2 Répertoires

- Les noms des répertoires sous VMS sont de la forme `[père.rép.sousrép]`. L'équivalent sous Linux est : `/père/rép/sousrép/`. Le père de tous les répertoires est le répertoire racine appelé `/` ; il contient d'autres répertoires comme `/bin`, `/usr`, `/tmp`, `/etc`, et bien d'autres.
- Le répertoire `/home` contient les répertoires *home* (NDT : « home » signifie « maison ») des utilisateurs : par exemple, `/home/pierre`, `/home/paul` et ainsi de suite. Quand un utilisateur se logue, il commence dans son répertoire *home* ; c'est l'équivalent de `SYS$LOGIN`. Il y a un raccourci pour le répertoire *home* : le tilde (« `~` »). Ainsi, `cd ~/tmp` est équivalent à, disons, `cd /home/paul/tmp`.
- Les noms des répertoires sont soumis aux mêmes règles que ceux des fichiers. En plus de cela, chaque répertoire a deux entrées spéciales : l'une est `.` : elle représente le répertoire lui-même (comme `[]`) ; l'autre, `..`, représente le répertoire parent (comme `[-]`).

Et maintenant quelques autres exemples :

```
VMS                                   Linux
---------------------------------------------------------------------

$ CREATE/DIR [.répertoire]            $ mkdir répertoire
$ CREATE/DIR [.dir1.dir2.dir3]        $ mkdirhier rép1/rép2/rép3
   non/disponible                     $ rmdir répertoire
                                      (si le répertoire est vide)
                                      $ rm -R répertoire
$ DIRECTORY                           $ ls
$ DIRECTORY [...]fichier.*.*          $ find . -name "fichier*"
$ SET DEF SYS$LOGIN                   $ cd
$ SET DEF [-]                         $ cd ..
$ SET DEF [père.rép.sousrép]          $ cd /père/rép/sousrép
$ SET DEF [.rép.sousrép]              $ cd rép/sousrép
$ SHOW DEF                            $ pwd
```

Si vous voulez en savoir plus sur les droits, les propriétaires, ou tout simplement en savoir plus tout court, sautez à la section 8 (Sujets avancés).

2.3 Programmes

- Les commandes, les programmes compilés et les scripts *shell* (équivalent des fichiers de commandes de VMS) n'ont pas forcément une extension comme `.EXE` or `.COM` et peuvent s'appeler comme bon vous semble. Les fichiers exécutables sont marqués d'un astérisque (« `*` ») lorsque vous exécutez `ls -F`.

- Pour lancer un fichier exécutable, il suffit de taper son nom (pas de `RUN`, ni de `PROGRAM.EXE`, ni encore de `@COMMAND`). Il est donc nécessaire que le fichier soit situé dans un répertoire du *path*, qui est une liste de répertoires. En général, le *path* contient des répertoires comme `/bin`, `/usr/bin`, `/usr/X11R6/bin`, etc. Si vous écrivez vos propres programmes, placez-les dans un répertoire de votre *path* (pour savoir comment, reportez-vous à la section 9 (Configurer)). Vous pouvez aussi lancer un programme en indiquant son chemin complet, par exemple `/home/paul/données/monprog` ou `./monprog` si le répertoire courant n'est pas dans le *path*.
- Les options des commandes sont passées sur la ligne de commande grâce à `OPTION=` sous VMS et grâce à -*une_option* ou -*une_option* sous Linux, *une_option* étant une lettre, différentes lettres combinées ou un mot. En particulier, l'option `-R` (récursif) de plusieurs commandes de Linux permet de faire la même chose que le `[...]` de VMS.
- Vous pouvez lancer plusieurs commandes sur la ligne de commande :

```
$ commande1 ; commande2 ; ... ; commande
```

- Toute la flexibilité de Linux repose sur deux fonctionnalités (l'une n'existe pas sous VMS, l'autre est mal implémentée) : la redirection des entrés/sorties et les *pipes*. (Pour être sincère, j'ai entendu que les versions récentes de IDL supportent la redirection et les *pipes* mais je n'ai pas ces versions.) La redirection sous VMS n'est qu'un effet de bord (souvenez vous de l'option `/OUTPUT=`) ou une tâche fastidieuse comme

```
$ DEFINE /USER SYS$OUTPUT OUT
$ DEFINE /USER SYS$INPUT IN
$ RUN PROG
```

  dont l'équivalent sous Linux (Unix) est simplement :

```
$ prog < in > out
```

  Utiliser des *pipes* est tout simplement impossible sous VMS mais ils jouent un rôle clé sous Unix. En voici un exemple typique :

```
$ monprog < données | filtre1 | filtre2 >> résultat.dat 2> erreurs.log &
```

  Traduisons. Le programme `monprog` utilise le fichier `données` comme entrée, sa sortie est canalisée (grâce à |) vers le programme `filtre1` qui l'utilise en tant qu'entrée et la traite. La sortie résultante est canalisée (*pipée*) vers `filtre2` pour être encore une fois traitée. La sortie finale est ajoutée (grâce à >>) au fichier `résultat.dat`, et les messages d'erreurs sont redirigés (grâce à 2>) vers le fichier `errors.log`. Tout ceci est exécuté en arrière-plan (grâce au `&` à la fin de la ligne de commande). Pour en savoir plus à ce sujet, reportez-vous à la section 11 (Exemples).

Pour le multitâche, les files, et tout ce qui s'y rapporte, reportez-vous à la section 8 (Sujets avancés).

2.4 Visite guidée

Maintenant vous êtes prêt pour essayer Linux. Entrez votre identifiant et votre mot de passe. Attention, Unix distingue les minuscules des majuscules. Ainsi, si votre login et votre mot de passe sont `pierre` et `Mon_Code`, ne tapez *pas* `Pierre` ou `mon_code`.

Une fois que vous êtes logué, le prompt s'affiche. Il y a des chances pour que se soit quelque chose du genre `nom_de_la_machine:$`. Si vous voulez le changer ou lancer des programmes automatiquement, vous devrez éditer le fichier caché `.profile` ou `.bash_profile` (jetez un oeil aux exemples dans la section 9 (Configurer)). C'est l'équivalent de `LOGIN.COM`.

Appuyer sur alt + F1, alt + F2, ..., alt + F6 permet de changer de console virtuelle. Quand une console virtuelle est occupée avec une application plein écran, vous pouvez changer de console et continuer à travailler. Essayez et loguez-vous sur une autre console virtuelle.

Maintenant, vous voulez peut-être lancer le *Système X Window* (que nous appellerons maintenant X). X est un environnement graphique similaire au DECWindows — en fait, ce dernier dérive de X. Tapez la commande `startx` et attendez quelques secondes ; vous verrez probablement s'ouvrir un `xterm` ou un autre émulateur de terminal, et peut-être une barre de boutons (cela dépend de la manière dont votre administrateur système a configuré votre machine Linux). Cliquez sur le menu (essayez les deux boutons de la souris) pour voir les menus.

Quand vous utilisez X, vous devez appuyez sur ctrl + alt + F1, ..., ctrl + alt + F6 pour accéder au mode texte (console). Essayez. Quand vous êtes dans une console, vous pouvez revenir à X en appuyant sur alt + F7. Pour quitter X, suivez les instructions de votre menu ou appuyez sur ctrl + alt + backspace.

Tapez la commande suivante pour obtenir une liste du contenu du répertoire courant (incluant les fichiers cachés) :

```
$ ls -al
```

Appuyez sur shift + page up pour faire défiler l'écran vers le haut. Maintenant, pour obtenir de l'aide sur la commande `ls` tapez

```
$ man ls
```

Appuyez sur q pour quitter. Pour finir notre tour d'horizon, tapez `exit` pour quitter votre session. Si maintenant vous voulez éteindre votre PC, appuyez sur ctrl + alt + suppr et attendez quelques secondes (n'éteignez *jamais* votre PC tant que Linux tourne ; cela peut causer des dommages dans le système de fichier).

Si vous pensez que vous êtes prêt pour travailler, allez-y. Mais si j'étais vous, je passerais d'abord par la section 8 (Sujets avancés).

3 Éditer des fichiers

`EDT` ne tourne pas sous Linux, mais il y a beaucoup d'autres éditeurs disponibles. Le seul qui soit sûr d'être présent sur tout système Unix est `vi` — oubliez-le, votre administrateur en a sûrement installé un meilleur. L'éditeur le plus populaire est probablement `emacs`, qui peut émuler `EDT` jusqu'à un certain degré ; `jed` est un autre éditeur qui permet l'émulation de `EDT`.

Ces deux éditeurs sont particulièrement utiles pour éditer des sources de programmes puisque qu'ils ont deux fonctionnalités inconnue de `EDT` : la coloration syntaxique et l'indentation automatique. De plus, vous pouvez compilez vos programmes à partir de l'éditeur (grâce à M-x compile sous emacs — pour comprendre la notation « M-x compile » lisez la suite ; en cas d'erreur de syntaxe, le curseur se positionnera tout seul sur la ligne en question. Je parie que vous ne voudrez plus jamais utiliser `EDT` après.

Si vous avez `emacs`, lancez-le. Tout d'abord vous devez comprendre le système de notation de combinaison de touche d'`emacs`. C désigne contrôle et M la touche méta (en général alt ou échap). Maintenant, tapez M-x edt-emultation-on. M-x permet de lancer des commandes avec `emacs` comme ctrl + z avec `EDT`. À partir de maintenant, `emacs` fait comme s'il était `EDT` à part pour quelques commandes :

- n'appuyez *pas* sur ctrl + z pour lancer une commande. Si vous l'avez fait, vous avez stoppé emacs. Tapez `fg` pour reprendre votre session emacs ;

- appuyez sur C-h ? pour obtenir de l'aide ou sur C-h t pour lancer un tutoriel ;
- pour sauver un fichier, appuyez sur C-x C-s ;
- pour quittez, appuyez sur C-x C-c ;
- pour insérez un nouveau fichier dans un buffer (pour ouvrir un fichier, en gros), appuyez sur C-x C-f, et ensuite C-x b pour changer de buffer.

Si vous avez `jed`, demandez à votre administrateur de le configurer comme il faut. L'émulation est active dès que vous le lancez. Utilisez les même touches que sur EDT et appuyez sur échap ? h pour obtenir l'aide. Les commandes sont lancées de la même manière que dans `emacs`. De plus, il y a quelques raccourcis pratiques faisant défaut à EDT ; vous pouvez en plus configurer ces raccourcis clavier. Demandez à votre administrateur.

Vous pouvez aussi utilisez un autre éditeur avec une interface complètement différente. `emacs` en mode natif est un choix courant. Un autre éditeur populaire est `joe` qui peut émuler d'autres éditeurs comme emacs lui-même (en étant même plus facile à utiliser) ou l'éditeur DOS. Lancez l'éditeur sous le nom `jmacs` ou `jstar` et appuyez respectivement sur ctrl + x h ou ctrl + j pour obtenir l'aide en ligne. `emacs` et `jed` sont *beaucoup* plus puissants que ce bon vieux EDT.

4 TeXer

TeX et LaTeX sont identiques à leurs homologues de VMS — seulement plus rapides :-), mais les outils pour manipuler les fichiers `.dvi` et `.ps` sont bien supérieurs :

- Pour compiler un fichier TeX faites comme d'habitude `tex file.tex`.
- Pour convertir un fichier `.dvi` en `.ps`, tapez `dvips -o fichier.ps fichier.dvi`.
- Pour visualiser un fichier `.dvi`, tapez lors d'une session X `xdvi fichier.dvi &`. Cliquez sur la page pour zoomer. Ce programme est intelligent : si vous éditez et lancez TeX pour produire une nouvelle version de votre fichier `.dvi`, `xdvi` actualisera l'affichage.
- Pour visualiser un fichier `.ps`, taper lors d'une session X `ghostview fichier.ps &`. Cliquez sur la page pour zoomer. Le document entier ou des pages sélectionnés peuvent être imprimés. Un programme plus récent et meilleur permet également de faire ça : `gv` ;
- Pour imprimer un fichier `.ps` on utilise généralement la commande `lpr fichier.ps`. Cependant si l'imprimante postscript s'appelle, par exemple, « ps » (demandez à votre administrateur système), il faudra faire `lrp -Pps fichier.ps`. Pour plus d'informations au sujet des files d'impressions, allez à la section 8.4 (Files d'impressions).

5 Programmer

Programmer sous Linux est *beaucoup* plus agréable : il existe un grand nombre d'outils qui rendent la programmation plus facile et plus rapide. Par exemple, la torture qu'est le cycle édition, sauvegarde, sortie de l'éditeur, compilation, réédition, etc. peut être raccourci en utilisant des éditeurs comme `emacs` ou `jed`, comme nous l'avons vu au dessus.

5.1 Fortran

Il n'y a pas de réelle différence pour le fortran, mais sachez qu'au moment où j'écris ces lignes, les compilateurs (libres) ne sont pas totalement compatibles avec ceux de VMS ; attendez-vous à quelques problèmes mineurs (en fait, le compilateur de VMS utilise des extensions qui ne sont pas standard). Jetez un oeil à `/usr/doc/g77/DOC` ou `/usr/doc/f2c/d2c.ps` pour plus de détails.

Votre administrateur a sans doute installé soit le compilateur natif `g77` (bien, mais, au jour de la version 0.5.21, toujours pas parfaitement compatible avec le Fortran de DEC), soit le traducteur de Fortran en C, `f2c`, et un des ses front-ends qui font de lui une imitation de compilateur natif. D'après mon expérience, le paquetage `yaf77` est celui qui donne les meilleurs résultats.

Pour compiler un source en Fortran avec `g77`, éditez le et sauvez le avec l'extension `.f`, et faites

```
$ g77 monprog.f
```

Cela va créer par défaut un exécutable appelé `a.out` (vous n'avez pas à effectuer les liens). Pour donner à l'exécutable un nom différent et faire quelques optimisations :

```
$ g77 -O2 -o monprog monprog.f
```

Méfiez-vous des optimisations ! Demandez à votre administrateur système de lire la documentation fournie avec le compilateur et de vous dire s'il y a des problèmes.

Pour compiler une sous-routine :

```
$ g77 -c masub.f
```

Un fichier `masub.o` sera créé. Pour lier cette sous-routine à un programme, vous devrez faire

```
$ g77 -o monprog monprog.f masub.o
```

Si vous avez plusieurs sous-routines externes et que vous voulez créer une bibliothèque, faites

```
$ cd sousroutines/
$ cat *f > mabib.f ; g77 -c mabib.f
```

Le fichier `mabib.o` créé pourra alors être lié à vos programmes.

Pour finir, pour lier une bibliothèque externe appelée, disons, `liblambda.so` , utilisez

```
$ g77 -o monprog monprog.f -llambda
```

Si vous avez `f2c`, vous n'aurez qu'à utiliser `f77` ou `fort77` à la place de `g77`.

Un autre outil de programmation utile est `make`. Il est décrit ci-dessous.

5.2 Utiliser `make`

`make` est un outil pour gérer la compilation de programmes divisés en plusieurs fichiers sources.

Supposons que vous ayez des fichiers sources contenant vos routines appelés `fichier_1.f, fichier_2.f et fichier_3.f`, et un fichier source principal qui utilise ces routines appelé `monprog.f`. Si vous compilez votre programme à la main, quand vous modifierez un fichier source vous allez devoir chercher quel fichier dépend de quel fichier, et les recompiler en tenant compte des dépendances.

Plutôt que de devenir fou, je vous propose d'écrire un *makefile*. C'est un fichier texte qui contient les dépendances entre les sources : quand un source est modifié, seuls les sources qui dépendent du fichier modifié seront recompilées.

Dans notre cas, le *makefile* ressemblerait à ceci :

```
# Voici le Makefile
# Attention : appuyez sur la touche tabulation quand « <TAB> »
# est écrit ! C'est très important : n'utilisez pas d'espace à la place.

monprog: monprog.o fichier_1.o fichier_2.o fichier_3.o
<TAB>g77 -o monprog monprog.o fichier_1.o fichier_2.o fichier_3.o
# monprog dépend de quatre fichiers objets

monprog.o: monprog.f
<TAB>g77 -c monprog.f
# monprog.o dépend de son fichier source

fichier_1.o: fichier_1.f
<TAB>g77 -c fichier_1.f
# fichier_1.o dépend de son fichier source

fichier_2.o: fichier_2.f fichier_1.o
<TAB>g77 -c fichier_2.f fichier_1.o
# fichier_2.o dépend de son fichier source et d'un fichier objet

fichier_3.o: fichier_3.f fichier_2.o
<TAB>g77 -c fichier_3.f fichier_2.o
# fichier_3.o dépend de son fichier source et d'un fichier objet

# fin du Makefile
```

Enregistrez ce fichier sous le nom `Makefile` et tapez `make` pour compiler votre programme. Vous pouvez aussi l'appeler `monprog.mak` et taper `make -f monprog.mak`. Et bien sûr, si vous voulez en savoir plus : `info make`.

5.3 Scripts *shell*

Les scripts *shell* sont les équivalents des fichiers de commandes de VMS et, pour changer, sont beaucoup plus puissants.

Pour écrire un script, tout ce que vous avez à faire est d'écrire un fichier au format ASCII contenant les commandes, l'enregistrer et le rendre exécutable (`chmod +x fichier`). Pour le lancer, tapez son nom (précédé de `./` s'il n'est pas dans le *path*).

Écrire des scripts avec `bash` est un sujet tellement vaste qu'il nécessiterait un livre entier, et je ne ne vais pas m'attarder sur le sujet. Je vais juste vous donner un exemple plus ou moins compréhensible et, je l'espère, utile, à partir duquel vous pourrez comprendre quelques règles de base.

```
#!/bin/sh
# exemple.sh
# Je suis un commentaire.
# Ne modifiez pas la première ligne, elle doit être présente.
echo "Ce système est : `uname -a`" # utilise la sortie de la commande
echo "Mon nom est $0" # variable interne
echo "Vous m'avez donné les $# paramètres suivants : "$*
echo "Le premier paramètre est : "$1
echo -n "Quel est votre nom ? " ; read votre_name
echo remarquez la différence : "Salut $votre_name" # cité avec "
echo remarquez la différence : 'Salut $votre_name' # cité avec '
REPS=0 ; FICHIERS=0
for fichier in `ls .` ; do
  if [ -d ${fichier} ] ; then # si le fichier est un répertoire
    REPS=`expr $REPS + 1`  # REPS = REPS + 1
  elif [ -f ${fichier} ] ; then
    FICHIER=`expr $FICHIER + 1`
  fi
  case ${fichier} in
    *.gif|*jpg) echo "${fichier}: fichier graphique" ;;
    *.txt|*.tex) echo "${fichier}: fichier texte" ;;
    *.c|*.f|*.for) echo "${fichier}: fichier source" ;;
    *) echo "${fichier}: fichier quelconque" ;;
  esac
done
echo "Il y a ${REPS} répertoires et ${FICHIERS} fichiers"
ls | grep "ZxY--!!!WKW"
if [ $? != 0 ] ; then # valeur de sortie de la dernière commande
  echo "ZxY--!!!WKW n'a pas été trouvé"
fi
echo "Ça suffit... tapez 'man bash' si vous voulez plus d'informations."
echo "Note du traducteur : 'info bash' est plus complet."
```

5.4 C

Linux est un excellent environnement pour la programmation en C. Si vous connaissez le C, voici quelques conseils pour vous débrouiller sous Linux. Pour compiler le célèbre `hello.c` vous utiliserez le compilateur `gcc`, qui est standard dans le monde de Linux et qui a la même syntaxe que `g77` :

```
$ gcc -O2 -o hello hello.c
```

Pour lier une bibliothèque à un programme, ajoutez l'option `-lbibliothèque`. Par exemple pour lier la bibliothèque math et optimiser faites

```
$ gcc -O2 -o mathprog mathprog.c -lm
```

(L'option `-lbibliothèque` force `gcc` à lier la bibliothèque `/usr/lib/libbibliothèque.a` ; ainsi `-lm` lie `/usr/lib/libm.a`.)

Quand votre programme est divisé en plusieurs fichiers sources, vous aurez besoin du programme `make` décrit juste au dessus.

Vous pouvez obtenir de l'aide sur quelques fonctions de la libc dans la section 3 des pages man. Par exemple :

```
$ man 3 printf
```

Il existe beaucoup de bibliothèques disponible. Parmi les premières que vous voudrez probablement utiliser, il y a `ncurses`, qui permet de gérer quelques effets du mode texte et `svgalib` pour faire du graphisme.

6 Graphiques

Parmi la masse de paquetages de graphiques disponibles, `gnuplot` sort du lot pour sa puissance et sa facilité d'utilisation. Créez tout d'abord deux fichiers de données : `2D-data.dat` (deux données par ligne) et `3D-data.dat` (trois par ligne). Puis, sous X, lancez `gnuplot`.

Exemple d'un graphe en 2D :

```
gnuplot> set title "mon premier graphe"
gnuplot> plot '2D-data.dat'
gnuplot> plot '2D-data.dat' with linespoints
gnuplot> plot '2D-data.dat', sin(x)
gnuplot> plot [-5:10] '2D-data.dat'
```

Exemple d'un graphe en 3D (chaque « ligne » de x valeurs est suivie d'une ligne vide) :

```
gnuplot> set parametric ; set hidden3d ; set contour
gnuplot> splot '3D-data.dat' using 1:2:3 with linespoints
```

Un fichier de données d'une seule colonne (une série de temps par exemple) peut aussi être dessiné comme un graphe en 2D :

```
gnuplot> plot [-5:15] '2D-data-1col.dat' with linespoints
```

ou en 3D (avec des lignes vides dans le fichier comme au dessus) :

```
gnuplot> set noparametric ; set hidden3d
gnuplot> splot '3D-data-1col.dat' using 1 with linespoints
```

Pour imprimez un graphe, si la commande pour imprimer sur votre imprimante postscript est `lpr -Pps fichier.ps`, faites

```
gnuplot> set term post
gnuplot> set out '| lpr -Pps'
gnuplot> replot
```

Tapez ensuite `set term x11` pour réafficher sur votre serveur X. Ne soyez pas déconcerté : la dernière impression se lancera seulement quand vous quitterez `gnuplot`.

Pour plus d'informations, tapez `help` ou regardez le répertoire des exemples (`/usr/lib/gnuplot/demos/`) s'il existe.

7 Mail et outils pour Internet

Du fait qu'Internet est né sur des machines Unix, on trouve plein d'applications de qualité et facile d'utilisation sous Linux. En voici quelques-unes :

- Mail : utilisez `elm` ou `pine` (NDT : mutt est très bien aussi) pour gérer votre courrier. Ces deux programmes ont une aide en ligne. Pour des messages courts, vous pouvez utilisez `mail` (`mail -s "Salut" utilisateur@quelquepart < msg.txt`). Vous préférez peut-être d'autres programmes comme `xmail` ou équivalent.
- Newsgroups : utilisez `tin` ou `slrn`. Ils sont tous les deux très intuitifs.
- FTP : en plus de l'inévitable `ftp`, demandez à votre administrateur d'installer l'excellent `ncftp` ou un même un client graphique comme `xftp`.
- WWW : `netscape`, `xmosaic`, `chimera` et `arena` sont des browsers graphiques ; `lynx` quant à lui utilise la console et est rapide et pratique.

8 Sujets avancés

Là, le jeu devient coriace. Apprenez ça, et ensuite vous pourrez dire que vous « connaissez quelque chose à Linux » ;-)

8.1 Droits et propriété

Les fichiers et les répertoires ont des droits et des propriétaires, comme sous VMS. Si nous ne pouvez pas lancer un programme, ne pouvez pas modifier un fichier ou ne pouvez pas accéder à un répertoire, c'est parce vous n'avez pas les droits adéquats pour le faire et/ou parce que le fichier ne vous appartient pas. Regardez l'exemple suivant :

```
$ ls -l /bin/ls
-rwxr-xr-x   1 root     bin         27281 Aug 15  1995 /bin/ls*
```

Le premier champ indique les droits du fichier `ls`. Il y a trois types de propriété : le propriétaire, le groupe et les autres (comme le propriétaire, le groupe et le reste du monde sous VMS) et trois droits : lecture, écriture et exécution.

De gauche à droite, `-` est le type du fichier (`-` désigne un fichier ordinaire, `d` un répertoire, `l` un lien, etc.) ; `rwx` sont les droits du propriétaire (lecture, écriture, exécution) ; `r-x` sont les droits du groupe du propriétaire (lecture, exécution) ; `r-x` sont les droits pour tous les autres utilisateurs (lecture, écriture).

Pour changer les droits d'un fichier :

```
$ chmod <quiXdroit> <fichier>
```

Avec *qui* représentant `u` (ce sont alors les droits du propriétaire qui sont affectés), `g` (ceux du groupe) ou `o` (ceux des « autres »). X est soit `+` (dans ce cas il donne les droits), soit `-` (il les enlève). Et *droit* est `r`, `w` ou `x`. Voici un exemple :

```
$ chmod u+x fichier
```

Cela permet de rendre le fichier exécutable pour le propriétaire. Il existe un petit raccourci `chmod +x fichier`.

```
$ chmod go-wx fichier
```

Là, on enlève les droits d'écriture et d'exécution au groupe et aux autres (donc à tout le monde sauf au propriétaire).

```
$ chmod ugo+rwx fichier
```

Tous les droits sont donnés à tout le monde.

Une manière plus courte de préciser les droits se base sur les nombres : `rwxr-xr-x` peut être exprimé par 755 (chaque lettre correspond à un bit : `--` vaut 0, `-x` 1, `-w-` 2, etc).

Pour un répertoire, `rx` signifie que vous pouvez vous placer dans ce répertoire et `w` que vous pouvez effacer un fichier dans ce répertoire (en tenant compte des droits du fichier évidemment) ou le répertoire lui-même. Tout ça n'est qu'une petit partie du sujet : man est votre ami.

Pour changer le propriétaire d'un fichier :

```
$ chown <utilisateur> <fichier>
```

Pour résumer, voici un tableau :

```
VMS                             Linux
------------------------------------------------------------------------------

SET PROT=(O:RW) fichier.txt     $ chmod u+rw fichier.txt
                                $ chmod 600 fichier.txt
SET PROT=(O:RWED,W) fichier     $ chmod u+rwx fichier
                                $ chmod 700 fichier
SET PROT=(O:RWED,W:RE) fichier  $ chmod 755 fichier
SET PROT=(O:RW,G:RW,W) fichier  $ chmod 660 fichier
SET FILE/OWNER_UIC=JOE fichier  $ chown joe fichier
SET DIR/OWNER_UIC=JOE [.dir]    $ chown joe rep/
```

8.2 Multitâche : processus et tâches (*jobs*)

En voici plus à propos de la manière de lancer les programmes. Il n'y a pas de file d'attente sous Linux ; le multitâche est géré très différemment. Voici à quoi ressemble une ligne de commande typique :

```
$ commande -s1 -s2 ... -sn par1 par2 ... parn < entrée > sortie &
```

Maintenant, voyons comment marche le multitâche. Les programmes qui tournent en avant-plan (*foreground*) ou arrière-plan (*background*) sont appelés des processus.

- Pour lancer un processus en arrière plan :

  ```
  $ programme [option] [< entrée] [> sortie] &
  [1] 234
  ```

Le shell vous donne le numéro de *job* (le premier nombre ; regardez ci-dessus) et le PID (le numéro du processus). Chaque processus est identifié par son PID.

Pour voir combien de processus sont lancés :

```
$ ps -ax
```

La liste des processus actifs va être affichée.

- Pour tuer un processus :

```
$ kill <PID>
```

Vous aurez peut-être besoin de tuer un processus quand vous ne savez pas le quitter de la bonne manière... ;-) Parfois, une processus sera seulement tué par une des commandes suivantes :

```
$ kill -15 <PID>
$ kill -9 <PID>
```

En plus de ça, le shell vous permet de stopper ou de suspendre temporairement un processus, envoyer un processus en arrière-plan ou en ramener un en avant-plan. Dans ce contexte, les processus sont appelées `jobs`.

- Pour voir combien de *jobs* sont actifs :

```
$ jobs
```

Les *jobs* sont identifés par le nombre que le shell leur donne et non pas par leur PID.

- Pour stopper un processus qui tourne en avant-plan appuyez sur ctrl + c. (Ça ne marche pas toujours.)
- Pour suspendre un processus tournant en avant-plan appuyez sur ctrl + z (Idem.)
- Pour envoyer un processus suspendu en arrière-plan (qui devient alors un *job*) :

```
$ bg <job>
```

- Pour envoyer un *job* en avant-plan :

```
$ fg <job>
```

- Pour tuer un *job* :

```
$ kill %<job>
```

8.3 Fichiers, deuxième

Voici plus d'information sur les fichiers.

- *stdin*, *stdout* et *stderr* : sous Unix, tous les composants du système sont assimilés à des fichiers. Les commandes et les programmes puisent leur entrée dans un « fichier » appelé *stdin* (l'entrée standard : généralement le clavier), redirigent leur sortie vers un « fichier » appelé *stdout* (généralement l'écran) et leurs erreurs vers un « fichier » appelé *stderr* (généralement l'écran).

  En utilisant $<$ et $>$ vous redirigez l'entrée et la sortie vers un fichier différent. De plus, $>>$ ajoute la sortie à un fichier à la place de l'écraser ; 2$>$ redirige les messages d'erreur (*stderr*) ; 2$>$&1 redirige *stderr* vers *stdout*, alors que 1$>$&2 redirige *stdout* vers *stderr*. Il y a un « trou noir » appelé `/dev/null` : tout ce qui est redirigez vers lui disparaît.

- Jokers/ : sur la ligne de commande `*` correspond à (et désigne) tous les fichiers sauf ceux qui sont cachés ; `.*` correspond à tous les fichiers cachés ; `*.*` correspond seulement ceux qui ont un « . » au milieu de leur nom suivi par d'autres caractères ; `l*c` correspondra à « loïc » et « luc » ; `*c*` correspondra à « piocher » et « picorer ». `%` devient `?`. Il existe également un autre joker : `[]`. Par exemple : `[abc]*` désigne les fichiers commençant par a, b ou c ; `*[I-N123]` désigne les fichier finissant par I, J, K, L, M, N, 1, 2 ou 3.
- `mv` (`RENAME`) ne permet pas de renommer plusieurs fichiers d'un coup. Ainsi, `mv *.xxx *.yyy` ne marchera pas.
- Utilisez `cp -i` et `mv -i` pour être prévenu quand un fichier va être écrasé.

8.4 Files d'impression

Vos fichiers à imprimer sont placés dans une file comme sous VMS. Quand vous lancez une commande d'impression, vous aurez peut-être à préciser le nom de l'imprimante. Par exemple

```
$ lpr fichier.txt # ce fichier est placé dans la file de l'imprimante standard
$ lpr -Plaser fichier.ps # celui dans celle de l'imprimante « laser »
```

Pour gérer la file d'impression utilisez les commandes suivantes :

```
VMS                                     Linux
-------------------------------------------------------------------------------

$ PRINT fichier.ps                      $ lpr fichier.ps
$ PRINT/QUEUE=laser fichier.ps          $ lpr -Plaser fichier.ps
$ SHOW QUEUE                            $ lpq
$ SHOW QUEUE/QUEUE=laser                $ lpq -Plaser
$ STOP/QUEUE                            $ lprm <numéro de job>
```

9 Configurer

Votre administrateur système a dû vous fournir certains fichier de configuration comme `.xinitrc`, `.bash_profile` et `.inputrc`. Ceux que vous voudrez peut-être modifier sont :

- `.bash_profile` ou `.profile` : ce fichier est lu par le shell au moment du login. C'est l'équivalent de `LOGIN.COM`.
- `.bash_logout` : celui-ci est lu en fin de session. C'est l'équivalent de `LOGOUT.COM`.
- `.bashrc` : il est lu par les shells non interactifs
- `.inputrc` : ce fichier configure les rôles des touches du clavier et le comportement du shell.

Pour vous donner un exemple, j'ai inclus une partie mon `.bash_profile` :

```
# $HOME/.bash_profile

# on ne redéfinit pas le path si ce n'est pas nécessaire
echo $PATH | grep $LOGNAME > /dev/null
if [ $? != 0 ]
```

```
then
  export PATH="$PATH:/home/$LOGNAME/bin"  # ajoute mon répertoire au path
fi

export PS1='LOGNAME:\w\$ '
export PS2='Continued...>'

# alias

alias bin="cd ~/bin" ; alias cp="cp -i" ; alias d="dir"
alias del="delete" ; alias dir="/bin/ls $LS_OPTIONS --format=vertical"
alias ed="jed" ; alias mv='mv -i'
alias u="cd .." ; alias undel="undelete"

# Quelques fonctions utiles

inst() # installe un tarball gzipé dans le répertoire courant
{
  tar xvfz $1
}
cz() # liste le contenu d'une archive .zip
{
  unzip -l $*
}
ctgz() # liste le contenu d'un tarball gzipé
{
  for fichier in $* ; do
    tar tfz ${fichier}
  done
}
tgz() # crée une archive .tgz à la zip.
{
  nom=$1 ; tar cvf $1 ; shift
  tar -rf ${nom} $* ; gzip -S .tgz ${nom}
}
```

Et voici mon `.inputrc` :

```
# $HOME/.inputrc
#
# Ce fichier est lu par bash et définit les fonctions attachés aux touches
# par le shell ; ce qui suit permet d'avoir un comportement courant pour
# les touches <it/fin/, <it/home/ (la touche au dessus de fin), <it/suppr/
# et les caractères accentués.
# Pour plus d'information, man readline.

"\e[1~": beginning-of-line
"\e[3~": delete-char
"\e[4~": end-of-line
```

```
set bell-style visible
set meta-flag On
set convert-meta Off
set output-meta On
set horizontal-scroll-mode On
set show-all-if-ambiguous On

# (F1 .. F5) sont "\e[[A" ... "\e[[E"

"\e[[A": "info "
```

10 Programmes utiles

10.1 Visionneur de fichiers : `less`

Vous utiliserez un tel programme tous les jours, c'est pourquoi je vais vous donner quelques astuces pour l'utiliser au mieux. Avant tout, demandez à votre administrateur de configurer `less` pour qu'il puisse afficher non seulement des fichiers textes mais aussi les fichiers compressés, les archives, etc.

Comme les dernières versions de `TYPE`, `less` vous permet de vous déplacer dans le fichier dans les deux directions. Il accepte aussi plusieurs commandes qui sont lancées en appuyant sur une touche. Les plus utilisés sont :

- tout d'abord, appuyez sur `q` pour quitter ;
- `h` pour obtenir l'aide ;
- `g` pour aller au début du fichier, `G` à la fin, un nombre suivi de `g` pour aller à cette ligne (par exemple `125g`), un nombre suivi de `%` pour aller à ce pourcentage du fichier ;
- `/motif` recherche vers l'avant le motif ; `n` recherche vers l'avant la l'occurrence suivante ; `?pattern` et `N` font la même chose vers l'arrière ;
- `m` suivi d'une lettre marque la position courante (par exemple `ma`) ; `'` suivi de la même lettre rappelle cette position ;
- `:e` ouvre un autre fichier ;
- `!commande` exécute un shell.

10.2 Numéros de version sous Linux

Hélas, Linux ne supporte toujours pas les numéros de version nativement. Cependant, on peut régler ce problème de deux manières. La première est d'utiliser RCS (Revision Control System) qui vous permet de garder la trace des précédentes versions d'un fichier. RCS est traité dans *Le mini-Howto RCS*.

La seconde est d'utiliser un éditeur qui sait traiter les numéros de version. `emacs` ou `joe` feront l'affaire. Pour emacs, ajoutez ces lignes dans votre `.emacs`.

```
(setq version-control t)
(setq kept-new-versions 15) ;;; ou toute autre valeur
(setq kept-old-versions 15)
(setq backup-by-copying-when-linked t)
(setq backup-by-copying-when-mismatch t)
```

Pour jed, vérifiez que vous avez une version supérieure à la 0.98.7. Le patch pour les numéros de version est disponible sur *http://ibogeo.df.unibo.it/guido/slang/backups.sl*

10.3 Archiver : tar et gzip

Sous Unix il existe quelques applications très répandues qui sont utilisés pour archiver et compresser des fichiers. `tar` est utilisé pour faire des archives (c'est à dire un regroupement de fichiers). Pour faire une archive :

```
$ tar -cvf <archive.tar> <fichier> [fichier...]
```

Pour extraire des fichiers d'une archive :

```
$ tar -xpvf <archive.tar> [fichier...]
```

Pour lister le contenu d'une archive :

```
$ tar -tf <archive.tar> | less
```

Les fichiers peuvent être compressés en utilisant `compress` ou `gzip`. `compress` est aujourd'hui obsolète et on n'utilise plus que `gzip`.

```
$ compress <fichier>
$ gzip <fichier>
```

Ceci créera un fichier compressé avec l'extension .Z (pour `compress`) ou .gz (pour `gzip`). Ces programmes ne font pas d'archives mais compressent des fichiers individuellement. Pour décompresser utilisez

```
$ compress -d <fichier.Z>
$ gzip -d <fichier.gz>
```

Faites un tour du côté de leurs pages man.

Les utilitaires `unarj`, `zip` et `unzip` sont aussi disponibles.

Les fichier avec l'extension `.tar.gz` ou `.tgz` (archivés par `tar`, puis compressés par `gzip`) sont très communs dans le monde Unix. Voici comment lister le contenu d'une telle archive :

```
$ tar -ztf <fichier.tar.gz> | less
```

Pour extraire les fichiers à partir d'un `.tar.gz` :

```
$ tar -zxf <fichier.tar.gz>
```

11 Exemples du monde réel

Le principe central d'Unix est qu'il existe plusieurs commandes simples qui peuvent être liées ensemble grâce aux *pipes* et aux redirections pour accomplir des tâches plus compliquées. Regardez les exemples suivants (je n'expliquerai que les plus compliqués ; pour les autres, reportez vous aux sections précédentes ou aux pages man).

- `ls` est trop rapide et je ne peux pas lire le nom de tous les fichiers.

  ```
  $ ls | less
  ```

- J'ai un fichier qui contient une liste de mots. Je veux les trier dans l'ordre inverse et les imprimer.

  ```
  $ cat fichier.txt | sort -r | lpr
  ```

- Mon fichier de données contient des doublons. Comment les effacer ?

  ```
  $ sort fichier.dat | uniq > nouveaufichier.dat
  ```

- J'ai un fichier appelé papier.txt ou papier.tex ou quelques choses dans le genre mais je ne m'en rappelle plus. Comment le retrouver ?

  ```
  $ find ~ -name "papier*"
  ```

  Expliquons. `find` est une commande très utile qui liste tous les fichier dans une arborescence (qui commence à partir du répertoire *home* dans ce cas). Sa sortie peut-être filtrée selon plusieurs critères comme par exemple `-name`.

- J'ai un fichier texte qui contient le mot « entropie » dans ce répertoire. Existe-t-il quelque chose de comparable à `SEARCH` ?

  Bien sûr, essayez cela :

  ```
  $ grep -l 'entropie' *
  ```

- Quelque part, j'ai un fichier texte qui contient le mot « entropy » et j'aimerai le retrouver. Sous VMS j'aurais utilisé `search entropy [...]*.*.*`, mais `grep` n'est pas récursif.

  ```
  $ find . -exec grep -l "entropy" {} \; 2> /dev/null
  ```

  `find .` sort tous les noms des fichiers à partir du répertoire courant, `-exec grep -l "entropy"` lance une commande sur chacun des fichiers (représentés par {}), \ termine la commande). Si vous pensez que la syntaxe est horrible... vous avez raison :-)

  Vous auriez aussi pu écrire le script suivant :

```
#!/bin/sh
# rgrep: grep récursif
if [ $# != 3 ]
then
  echo "Utilisation : rgrep <paramètres> <motif> <répertoire>"
  exit 1
fi
find $3 -name "*" -exec grep $1 $2 {} \; 2> /dev/null
```

  Voici l'explication. `grep` marche comme `SEARCH` et combiné avec `find` nous obtenons le meilleur des deux mondes.

- J'ai un fichier qui a deux lignes d'en-tête et qui a ensuite n données par ligne, pas nécessairement espacées de la même manière. Je veux extraire la deuxième et le cinquième champs de chaque ligne. Dois-je écrire un programme en Fortran ?

  Nan. Ceci est plus rapide :

  ```
  $ awk 'NL > 2 {print $2, "\t", $5}' fichier.dat > nouveaufichier.dat
  ```

`awk` est en fait un langage de programmation. Pour chaque ligne à partir de la troisième, on affiche le second et le cinquième champ en les séparant par une tabulation. Apprenez à vous servir de awk — il vous fera gagner beaucoup de temps.

- J'ai téléchargé le `ls-lR.gz` d'un FTP. Pour chaque sous répertoire, il y a une ligne « total x », avec x la taille en Kilo-octets du répertoire. J'aimerais avoir le total de toutes ces valeurs.

  ```
  zcat ls-lR.gz | awk ' $1 == "total" { i += $2 } END {print i}'
  ```

  `zcat` sort le contenu du fichier `.gz`. La sortie est envoyée vers awk dont je vous recommande chaudement de lire la page man.

- J'ai écrit un programme en Fortran, `monprog`, pour calculer un paramètre à partir d'un fichier de données. Je voudrais le lancer sur des centaines de fichiers et avoir la liste des résultats, mais c'est une calamité de demander chaque fois le nom du fichier. Sous VMS, j'aurais écrit un long fichier de commande. Et sous Linux ?

  Un script très court. Faites votre programme pour qu'il cherche le programme mesdonnées.dat et pour qu'il afficher le résultat sur l'écran (stdout) et écrivez ensuite ce petit script :

```
#!/bin/sh
# monprog.sh: lance la même commande sur plusieurs fichiers
# usage: monprog.sh *.dat
for fichier in $*  # pour tous les paramètres (e.g. *.dat)
do
  # ajouter le nom du fichier dans résultat.dat
  echo -n "${fichier}:    " >> résultats.dat
  # copie le paramètre courant dans mesdonnées.dat et lance monprog
  # et ajoute le sortie à résultats.dat
  cp ${fichier} mesdonnées.dat ; monprog >> résultats.dat
done
```

- Je veux remplacer « géologie » par « géophysique » dans tous mes fichiers textes. Dois-je les éditer manuellement ?

  Nan. Écrivez ce script :

```
#!/bin/sh
# remplace $1 par $2 dans $*
# utilisation : remplace "vieux-motif" "nouveau-motif" fichier [fichier...]
VIEUX=$1           # premier paramètre
NOUVEAU=$2         # second
shift ; shift      # enlever les deux premier paramètres ; les suivants sont
                   # les noms des fichiers
for fichier in $*  # pour tous les fichier donnés en paramètres
do
# remplace toutes les occurrences de VIEUX par NOUVEAU et sauve cela
# dans un fichier temporaire
  sed "s/$VIEUX/$NOUVEAU/g" ${fichier} > ${fichier}.nouveau
# renommer le fichier temporaire
  /bin/mv ${fichier}.new ${fichier}
done
```

- J'ai des fichier contenant des données. Je ne connais pas leur taille et je dois enlever leur avant-dernière et leur avant-avant-dernière lignes. Heu... à la mimine ?

  Bien sûr que non :

```
#!/bin/sh
# prune.sh: efface les n-1ème et n-2ème ligne de fichiers
# utilisation : prune.sh fichier [fichier...]
for fichier in $*   # pour chaque paramètre
do
  LIGNES=`wc -l $fichier | awk '{print $1}'`  # nombre de ligne dans fichier
  LIGNES=`expr $LIGNES - 3`                   # LIGNES = LIGNES - 3
  head -n $LIGNES $fichier > $fichier.new     # sort les premières lignes
                                              # de LIGNES

  tail -n 1 $fichier >> $fichier.new          # ajoute la dernière ligne
done
```

  NDT : il est tout de même beaucoup plus élégant d'utiliser `ed` :

```
#!/bin/sh
# prune.sh: efface les n-1ème et n-2ème ligne de fichiers
# utilisation : prune.sh fichier [fichier...]
for fichier in $*   # pour chaque paramètre
do
  printf '$-2,$-1d\nw\nq\n' | ed -s $fichier
done
```

J'espère que ces exemples vous auront ouvert l'appétit.

12 Astuces dont on ne peut se passer

- La complétion de commande : l'appui sur la touche tabulation quand vous tapez une commande va compléter la ligne de commande. Par exemple, disons que vous devez taper `less un_nom_de_fichier_très_long`. Il vous suffira de taper `less un` puis d'appuyer sur tabulation (si vous avez plusieurs fichier qui commencent par les mêmes caractères tapez-en assez pour résoudre l'ambiguïté).
- Faire défiler l'écran vers le haut : en appuyant sur Shift-Page_up vous pouvez faire défiler l'écran vers le haut de quelques pages (le nombre de page dépend de la mémoire vidéo de votre PC).
- Rétablir l'écran : s'il vous arrive de faire un `more` ou un `cat` sur un fichier binaire votre écran pourra se retrouver plein de symboles bizarres. Pour arranger les choses, à l'aveuglette tapez `reset` ou cette séquence de caractères : `echo ctrl-v Echap c Entrée`.
- Copier du texte sur la console : demandez à votre administrateur d'installer `gpm`, un driver pour la souris en mode texte. Cliquez et glissez pour sélectionner du texte et ensuite appuyez sur le bouton droit pour coller le texte sélectionné. Ça marche entre plusieurs consoles virtuelles.
- Copier du texte sous X : cliquez et glissez pour sélectionnez le texte dans un `xterm` et cliquez ensuite sur le bouton du milieu (ou les deux boutons) pour coller.

13 Lire des bandes VMS depuis Linux

(Cette section a été écrite par Mike Miller)

13.1 Introduction

De temps en temps vous pourrez être amené à lire des bandes enregistrées à partir d'une machine sous VMS (ou des bandes enregistrées pour être lisibles sous VMS et les systèmes Unix). En général, c'est assez facile pour les bandes DECFILES11A.

Bien que l'on puisse lire ceci comme une partie d'un mini-HOWTO Linux, je crois que les informations ici présentées sont appliquables à tout système Unix – j'ai fait cela avec Linux et les systèmes Unix d'HP, de SUN et de DEC. La principale dépendance à la plate-forme est le nom des fichiers périphériques qui peuvent varier d'un système à l'autre, et les options de `mt` pour spécifier le nom du périphérique (par exemple, `mt -f` avec Linux et `mt -t` sur HPUX 9).

Avertissement : j'ai seulement essayé cela avec des lecteurs de bande Exabyte 8 mm SCSI. Si vous avez lu d'autres formats (vous avez encore des lecteurs 9 pistes qui traînent quelque part ?), faites-le moi savoir et je les ajouterai.

13.2 Les bases

Pour lire une bande qui a été écrite avec la commande `copy` de VMS (ou une commande compatible) tout ce que vous devez savoir est que, pour chaque fichier effectif, il y aura trois fichiers de données sur la bande : une entête, les données et une terminaison. L'entête et la terminaison sont intéressantes car elles contiennent des informations sur le fichier tel qu'il était sous VMS. Le fichier de données est... le fichier de données. Chacun de ces fichiers peut être extrait de la bande avec la commande dd. La bande peut être positionnée en utilisant la commande `mt`.

Prenons un exemple. J'ai une bande VMS. Les deux premiers de ses fichiers étaient appelés `ce66-2.evt` et `ce66-3.evt` sur le système VMS. Le label de la bande est c66a2.

Si j'exécute les commandes suivantes

```
$ dd if=$TAPE bs=16k of=entête1
$ dd if=$TAPE bs=16k of=donnée1
$ dd if=$TAPE bs=16k of=terminaison1
$ dd if=$TAPE bs=16k of=entête2
$ dd if=$TAPE bs=16k of=donnée2
$ dd if=$TAPE bs=16k of=terminaison2
```

je me retrouve avec six fichiers : `entête1`, `donnée1`, `terminaison1`, `entête2`, `donnée2` et `terminaison2`. La syntaxe ici est `if="fichier d'entrée"`, `bs="taille des blocs"` et `of="fichier de sortie"`. TAPE doit être une variable d'environnement contenant le nom du périphérique de votre lecteur de bande — par exemple /dev/nts0 si vous utilisez le premier disque SCSI sous Linux.

Si vous aviez voulu récupérer le second fichier, mais pas le premier, sous son non d'origine, vous n'auriez pas eu à vous préoccuper de l'entête :

```
$ mt -f $TAPE fsf 4
$ dd if=$TAPE bs=16k of=ce66-2.evt
$ mt -f $TAPE fsf 1
```

Notez le 4. Il permet de sauter trois fichiers pour le premier fichier et un pour l'entête du deuxième. Le second `mt` saute la seconde terminaison et positionne la bande au début du prochain fichier : la troisième entête. Vous pouvez aussi utilisez `mt` pour sauter des fichiers en arrière (`bsf`), rembobiner (`rewind`) ou décharger la bande (`offline`, `rewoffl`).

13.3 Quelques détails

Les fichiers d'entêtes et de terminaisons contiennent des données ASCII en majuscules utilisées par VMS pour stocker des informations sur le fichier comme par exemple la taille des blocs. Ils contiennent aussi le nom du fichier, ce qui peut être utile si vous voulez faire des scripts qui lisent automatiquement des fichiers ou recherchent un fichier particulier. La première entête de la bande est un peu différente des suivantes.

L'en-tête du premier fichier de la bande (comme par exemple dans l'exemple ci-dessus) débute par la chaîne « VOL1 ». Suit le nom de volume. Dans notre exemple, le fichier `entête1` commence par « VOL1C66A2 ». Vient ensuite une série d'espace terminée par un chiffre. Après, on trouve la chaîne « HDR1 » qui indique que l'on a à faire à un fichier d'entête. Les caractères qui suivent immédiatement la chaîne « HDR1 » forment le nom du fichier VMS. Dans l'exemple, cette chaîne est « HDR1CE66-2.EVT ». Le prochain champ est le nom du volume.

Le champs initial des autres fichiers (c'est-à-dire tous sauf le premier de la bande) est quelque peu différent. « VOL1 » n'est pas présent. Le reste est identique. Un autre champ utile est le septième. Il se termine par « DECFILES11A ». C'est un signe de conformité au standard DEC Files11A.

```
Champ              Entête initiale              Entêtes suivantes
1                  VOL1 + nom du volume         HDR1 + nom du fichier
2                  HDR1 + nom du fichier        Nom du volume
3                  Nom du volume                ...
4                  ...                          ...
5                  ...                          ...
6                  ...                          DECFILES11A
7                  DECFILES11A
```

Pour une documentation complète sur le format de l'entête et de la terminaison, voyez la documentation DEC FILES11 (sur le mur gris/orange — demandez à votre communauté VMS locale :-).

13.4 Commentaires sur la taille des blocks

Dans l'exemple, j'ai utilisé une taille de bloc de 16 ko. Sur un système Unix, il n'y a pas de taille de bloc associée à un fichier sur un disque alors que, sous VMS, chaque fichier à une taille de bloc spécifique. Cela veut dire que la taille du bloc importe peu sous linux... sauf que cela rend la lecture des bandes plus compliquée. Si vous avez des difficultés pour trouver la taille du bloc et pour lire une bande, vous pouvez essayer de régler la taille de bloc matérielle de votre lecteur de bande en utilisant `mt -f $TAPE setblk 0`. La syntaxe exacte de l'option setblk (et sa disponibilité) dépend de la version de `mt`, de l'interface matérielle de votre lecteur de bande et de votre environnement Unix.

(Merci à *Wojtek Skulski* pour avoir signalé l'option setblk.)

14 La fin

14.1 Copyright

14.2 Avertissement

Ce travail a été écrit d'après l'expérience que nous avons eue au *Settore di Geofisica of Bologna University* (Italie), ou un VAX 4000 a été remplacé par un Pentium tournant sous Linux. La plupart de mes collègues sont des utilisateurs de VMS, et certains d'entre eux l'ont abandonné pour Linux.

Le *HOWTO Comment migrer de VMS à Linux* a été écrit par *Guido Gonzato* et *Mike Miller* qui a écrit la partie sur la lecture des bandes VMS. Merci beaucoup à mes collègues et amis qui m'ont aidé à définir les besoins et les habitudes de l'utilisateur lambda de VMS, et particulièrement au Dr. Warner Marsocchi.

S'il vous plaît, aidez moi à améliorer ce HOWTO. Je ne suis pas un expert de VMS et je n'en serai jamais un. C'est pourquoi vos suggestions et reports de bogues seront plus que bienvenus.

Amusez-vous bien,

Guido `=8-)`