From VMS to Linux HOWTO

Table of Contents

Table of Contents

From VMS to Linux HOWTO

By Guido Gonzato, guido ``at'' ibogeo.df.unibo.it, and Mike Miller, miller5@uiuc.edu

v1.1.3, 17 September 1999

This HOWTO is aimed at all those who have been using VMS and now need or want to switch to Linux, the free UNIX clone. The transition is made (hopefully) painless with a step−−to−−step comparison between commands and available tools.

6.Graphics

7.Mail and Internet Tools

8.Advanced Topics

9.Configuring

10.Useful Programs

11.Real Life Examples

12.Tips You Can't Do Without

13.Reading VMS tapes from Linux

14.The End

1.Introduction

1.1 Why Linux?

You've heard that UNIX is difficult and balk at the prospect of leaving VMS, don't you? Don't worry. Linux, one of the finest UNIX clones, is not more difficult to use than VMS; actually, I find it easier. Although VMS aficionados may not agree, in many people's opinion Linux is much more powerful and versatile.

Linux and VMS are both good operating systems and accomplish essentially the same tasks, but Linux has a few features that make it a good alternative to VMS. Moreover, Linux is available for PCs while VMS is not, and modern Pentium−based Linux machines can outperform a VAX. The icing on the cake is the excellent performance of modern video cards, which turn an X11−based Linux box into a fast graphic workstation; nearly always, quicker than dedicated machines.

I imagine you're a university researcher or a student, and that you use VMS for the following everyday tasks:

- writing papers with TeX/LaTeX;
- programming in Fortran;
- doing some graphics;
- using Internet services;
- et cetera.

In the following sections I'm going to explain to you how to do these tasks under Linux, exploiting your experience with VMS. Prerequisites:

- Linux and X Window System are properly installed;
- there's a system administrator to take care of the technical details (please get help from them, not from me ;–) ;
- your shell−−−the equivalent of `DCL`−−−is `bash` (ask your sysadm).

Please note that this HOWTO is not enough to acquaint you fully with Linux: it only contains the bare essential to get you started. You should learn more about Linux to make the most of it (advanced `bash` features, programming, regular expressions...). From now on, RMP means `please read the man pages for further details'. The man pages are the equivalent of the command `HELP`.

The Linux Documentation Project documents, available on ftp://sunsite.unc.edu:/pub/Linux/docs/LDP, are an important source of information. I suggest that you read Larry Greenfield's ``Linux User Guide''−−−it's invaluable for the novice user.

And now, go ahead.

1.2 Comparing Commands and Files

This table attempts to compare VMS' and Linux' most used commands. Please keep in mind that the syntax is often very different; for more details, refer to the following sections.

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


@COMMAND                        command                 must be executable
COPY file1 file2                cp file1 file2
CREATE/DIR [.dirname]           mkdir dirname           only one at a time
CREATE/DIR [.dir1.dir2]         mkdirhier dir/name
DELETE filename                 rm filename
DIFF file1 file2                diff -c file1 file2
DIRECTORY                       ls
DIRECTORY [...]file             find . -name file
DIRECTORY/FULL                  ls -al
EDIT filename                   vi filename,            you won't like it
                                emacs filename,         EDT compatible
                                jed filename            ditto---my favourite
FORTRAN prog.for                g77 prog.f,             no need to do LINK
                                f77 prog.f,
                                fort77 prog.f
HELP command                    man command             must specify `command'
                                info command            ditto
LATEX file.tex                  latex file.tex
LOGIN.COM                       .bash_profile,          `hidden' file
                                .bashrc                 ditto
LOGOUT.COM                      .bash_logout            ditto
MAIL                            mail,                   crude
                                elm,                    much better
                                pine                    better still
PRINT file.ps                   lpr file.ps
PRINT/QUEUE=laser file.ps       lpr -Plaser file.ps
PHONE user                      talk user
RENAME file1 file2              mv file1 file2          not for multiple files
RUN progname                    progname
SEARCH file "pattern"           grep pattern file
SET DEFAULT [-]                 cd ..
SET DEFAULT [.dir.name]         cd dir/name
SET HOST hostname               telnet hostname,        not exactly the same
                                rlogin hostname
SET FILE/OWNER_UIC=joe          chown joe file          completely different
SET NOBROADCAST                 mesg
SET PASSWORD                    passwd
SET PROT=(perm) file            chmod perm file         completely different
SET TERMINAL                    export TERM=            different syntax
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,                   for processes
                                lprm                    for print queues
SUBMIT command                  command &
SUBMIT/AFTER=time command       at time command
TEX file.tex                    tex file.tex
TYPE/PAGE file                  more file
                                less file               much better
```

But of course it's not only a matter of different command names. Read on.

2.Short Intro

This is what you absolutely need to know before logging in the first time. Relax, it's not much.

2.1 Files

- Under VMS filenames are in the form `filename.extension;version`. Under Linux, the version number doesn't exist (big limitation, but see Section Numbered Backups Under Linux); the filename has normally a limit of 255 characters and can have as many dots as you like. Example of filename: `This.is_a_FILEname.txt`.
- Linux distinguishes between upper case and lower case characters: `FILENAME.txt` and `filename.txt` are two different files; `ls` is a command, `LS` is not.
- A filename starting with a dot is a `hidden' file (that is, it won't normally show up in dir listings), while filenames ending with a tilde '~' represent backup files.

Now, a table to sum up how to translate commands from VMS to Linux:

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

$ COPY file1.txt file2.txt              $ cp file1.txt file2.txt
$ COPY [.dir]file.txt []                $ cp dir/file.txt .
$ COPY [.dir]file.txt [-]               $ cp dir/file.txt ..
$ DELETE *.dat;*                        $ rm *dat
$ DIFF file1 file2                      $ diff -c file1 file2
$ PRINT file                            $ lpr file
```

```
$ PRINT/queue=queuename file            $ lpr -Pprintername file
$ SEARCH *.tex;* "geology"              $ grep geology *tex
```

For other examples involving directories, see below; for details about protections, ownership, and advanced topics, see Section Advanced Topics.

2.2 Directories

* Within the same node and device, directories names under VMS are in the form `[top.dir.subdir]`; under Linux, `/top/dir/subdir/`. On the top of the directory tree lies the so−−called `root directory' called `/`; underneath there are other directories like `/bin`, `/usr`, `/tmp`, `/etc`, and others.
* The directory `/home` contains the so−−called users' `home directories': e.g. `/home/guido`, `/home/warner`, and so on. When a user logs in, they start working in their home dir; it's the equivalent of `SYS$LOGIN`. There's a shortcut for the home directory: the tilde '~'. So, `cd ~/tmp` is the same as, say, `cd /home/guido/tmp`.
* Directory names follow the same rules as file names. Furthermore, each directory has two special entries: one is `.` and refers to the directory itself (like `[]`), and `..` that refers to the parent directory (like `[-]`).

And now for some other examples:

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

$ CREATE/DIR [.dirname]                 $ mkdir dirname
$ CREATE/DIR [.dir1.dir2.dir3]          $ mkdirhier dir1/dir2/dir3
   n/a                                  $ rmdir dirname
                                        (if dirname is empty)
                                        $ rm -R dirname
$ DIRECTORY                             $ ls
$ DIRECTORY [...]file.*;*               $ find . -name "file*"
$ SET DEF SYS$LOGIN                     $ cd
$ SET DEF [-]                           $ cd ..
$ SET DEF [top.dir.subdir]              $ cd /top/dir/subdir
$ SET DEF [.dir.subdir]                 $ cd dir/subdir
$ SHOW DEF                              $ pwd
```

For protections, ownership, and advanced topics, see Section Advanced Topics.

2.3 Programs

- Commands, compiled programs, and shell scripts (VMS' `command files') don't have sort of mandatory extensions like `.EXE` or `.COM` and can be called whatever you like. Executable files are marked by an asterisk '`*`' when you issue `ls -F`.
- To run an executable file, just type its name (no `RUN PROGRAM.EXE` or `@COMMAND`). Caveat: it's essential that the file be located in a directory included in the *path of executables*, which is a list of directories. Typically, the path includes dirs like `/bin`, `/usr/bin`, `/usr/X11R6/bin`, and others. If you write your own programs, put them in a directory you have included in the path (see how in Section Configuring). As an alternative, you may run a program specifying its complete path: e.g., `/home/guido/data/myprog`; or `./myprog`, if the current directory isn't in the path.
- Command switches are obtained with `/OPTION=` under VMS, and with `-switch` or `--switch` under Linux, where `switch` is a letter, more letters combined, or a word. In particular, the switch `-R` (recursive) of many Linux commands performs the same action as `[...]` under VMS;
- You can issue several commands on the command line:

```
$ command1 ; command2 ; ... ; commandn
```

- Most of the flexibility of Linux comes from two features awkwardly implemented or missing in VMS: I/O redirection and piping. (I have been told that recent versions of DCL support redirection and piping, but I don't have that version.) Redirection is a side feature under VMS (remember the switch `/OUTPUT=` of many commands), or a fastidious process, like:

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

  which has the simple Linux (UNIX) equivalent:

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

  Piping is not readily available under VMS, but has a key role under UNIX. A typical example:

```
$ myprog < datafile | filter_1 | filter_2 >> result.dat 2> errors.log &
```

  which means: the program `myprog` gets its input from the file `datafile` (via <), its output is piped (via |) to the program `filter_1` that takes it as input and processes it, the resulting output is piped again to `filter_2` for further processing, the final output is appended (via >>) to the file `result.dat`, and error messages are redirected (via 2>) onto the file `errors.log`. All this in background (`&` at the end of the command line). More about this in Section Examples.

For multitasking, `queues', and the like, see Section Advanced Topics.

2.4 Quick Tour

Now you are ready to try Linux out. Enter your login name and password *exactly* as they are. For example, if your login name and password are `john` and `My_PassWd`, *don't* type `John` or `my_passwd`. Remember, UNIX distinguishes between capital and small letters.

Once you've logged in, you'll see a prompt; chances are it'll be something like `machinename:$`. If you want to change the prompt or make some programs start automatically, you'll have to edit a `hidden' file called `.profile` or `.bash_profile` (see example in Section Configuring). This is the equivalent of `LOGIN.COM`.

Pressing ALT−−F1, ALT−−F2, ... ALT−−F6 switches between `virtual consoles'. When one VC is busy with a full−−screen application, you can flip over to another and continue to work. Try and log in to another VC.

Now you may want to start X Window System (from now on, X). X is a graphic environment very similar to DECWindows−−−actually, the latter derives from the former. Type the command `startx` and wait a few seconds; most likely you'll see an open `xterm` or equivalent terminal emulator, and possibly a button bar. (It depends on how your sysadm configured your Linux box.) Click on the desktop (try both mouse buttons) to see a menu.

While in X, to access the text mode (`console') sessions press CTRL−−ALT−−F1 ... CTRL−−ALT−−F6. Try it. When in console, go back to X pressing ALT−−F7. To quit X, follow the menu instructions or press CTRL−−ALT−−BS.

Type the following command to list your home dir contents, including the hidden files:

```
$ ls −al
```

Press SHIFT−−PAG UP to back−scroll. Now get help about the `ls` command typing:

```
$ man ls
```

pressing 'q' to exit. To end the tour, type `exit` to quit your session. If now you want to turn off your PC, press CTRL−−ALT−−DEL and wait a few seconds (*never* switch off the PC while in Linux! You could damage the filesystem.)

If you think you're ready to work, go ahead, but if I were you I'd jump to Section Advanced Topics.

3.Editing Files

Linux doesn't have `EDT`, but there are scores of editors available. The only one that's guaranteed to be included in every UNIX version is `vi`−−−forget it, your sysadm must have installed something better. Probably the most popular editor is `emacs`, which can emulate `EDT` to a certain degree; `jed` is another editor that provides `EDT` emulation.

These two editors are particularly useful for editing program sources, since they have two features unknown to `EDT`: syntax hilighting and automatic indentation. Moreover, you can compile your programs from within the editor (command `ESC−X compile`); in case of a syntax error, the cursor will be positioned on the offending line. I bet that you'll never want to use the true blue `EDT` again.

If you have `emacs`: start it, then type `ESC−X edt−emulation−on`. Pressing ALT−−X or ESC−X is `emacs`' way of issuing commands, like `EDT`'s CTRL−−Z. From now on, `emacs` acts like `EDT` apart from a few commands. Differences:

- *don't* press CTRL−−Z to issue commands (if you did, you stopped `emacs`. Type `fg` to resume it);
- there's an extensive on−line help. Press CTRL−H ?, or CTRL−H T to start a tutorial;
- to save a file, press CTRL−X CTRL−S;
- to exit, press CTRL−X CTRL−C;
- to insert a new file in a buffer, press CTRL−X CTRL−F, then CTRL−X B to switch among buffers.

If you have `jed`: ask your sysadm to configure `jed` properly. Emulation is already on when you start it; use the normal keypad keys, and press CTRL−−H CTRL−−H or CTRL−? to get help. Commands are issued in the same way as `emacs`'. In addition, there are some handy key bindings missing in the original `EDT`; key bindings can also be tailored to your own taste. Ask your sysadm.

In alternative, you may use another editor with a completely different interface. `emacs` in native mode is an obvious choice; another popular editor is `joe`, which can emulate other editors like `emacs` itself (being even easier to use) or the DOS editor. Invoke the editor as `jmacs` or `jstar` and press, respectively, CTRL−X H or CTRL−J to get online help. `emacs` and `jed` are *much* more powerful than good ol' `EDT`.

4.TeXing

TeX and LaTeX are identical to their VMS counterparts−−−only quicker :−), but the tools to handle the `.dvi` and `.ps` files are superior:

- to run a TeX file through TeX, do as usual: `tex file.tex`;
- to turn a `.dvi` file into a `.ps` file, type `dvips −o filename.ps filename.dvi`;
- to visualize a `.dvi` file, type within an X session: `xdvi filename.dvi &`. Click on the page to magnify. This program is smart: if you edit and run TeX producing newer versions of the `.dvi` file, `xdvi` will update it automatically;

- to visualize a `.ps` file, type within an X session: `ghostview filename.ps &`. Click on the page to magnify. The whole document or selected pages can be printed. A newer and better program is `gv`.
- to print the `.ps`: usually the command `lpr mypaper.ps` will do, but if the PostScript printer is called, say, `ps' (ask your sysadm) you'll do: `lpr -Pps mypaper.ps`. For more information about print queues, go to Section Print Queues.

5.Programming

Programming under Linux is *much* better: there are lots of tools that make programming easier and quicker. For instance, the drudgery of editing−−saving−−exiting−−compiling−−re−editing can be cut short by using editors like `emacs` or `jed`, as seen above.

5.1 Fortran

Not substantial differences here, but note that at the time of writing the available (free) compilers are not 100% compatible with VMS'; expect some minor quirks. (It's actually the VMS compiler which has non−standard extensions.) See `/usr/doc/g77/DOC` or `/usr/doc/f2c/f2c.ps` for details.

Your sysadm has installed a native compiler called `g77` (good but, as of version 0.5.21, still not perfectly compatible with DEC Fortran) or possibly the Fortran to C translator, `f2c`, and one of the front−ends that make it mimic a native compiler. In my experience, the package `yaf77` is the one that provides best results.

To compile a Fortran program with `g77`, edit the source, save it with extension `.f`, then do:

```
$ g77 myprog.f
```

which creates by default an executable called `a.out` (you don't have to link anything). To give the executable a different name and do some optimisation:

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

Beware of optimisations! Ask your sysadm to read the documentation that comes with the compiler and tell you if there are any problems.

To compile a subroutine:

```
$ g77 -c mysub.f
```

This creates a file `mysub.o`. To link this subroutine to a program, you'll do

```
$ g77 -o myprog myprog.f mysub.o
```

If you have many external subroutines and you want to make a library, do the following:

```
$ cd subroutines/
$ cat *f >mylib.f ; g77 -c mylib.f
```

This will create `mylib.o` that you can link to your programs.

Finally, to link an external library called, say, `libdummy.so`:

```
$ g77 -o myprog myprog.f -ldummy
```

If you have `f2c`, you only have to use `f77` or `fort77` instead of `g77`.

Another useful programming tool is `make`, described below.

5.2 Using `make`

The utility `make` is a tool to handle the compilation of programs that are split into several source files. The VMS counterparts are `MMS` and `MMK`, which have a different syntax.

Let's suppose you have source files containing your routines, `file_1.f, file_2.f, file_3.f`, and a source file of the main program that uses the routines, `myprog.f`. If you compile your program manually, whenever you modify one of the source files you have to figure out which file depends on which, which file to recompile first, and so on.

Instead of getting mad, you can write a `makefile'. This is a text file containing the dependencies between your sources: when one is modified, only the ones that depend on the modified file will be recompiled.

In our example, you'd write a makefile like this:

```
# This is makefile
# Press the <TAB> key where you see <TAB>!
# It's important: don't use spaces instead.

myprog: myprog.o file_1.o file_2.o file_3.o
<TAB>g77 -o myprog myprog.o file_1.o file_2.o file_3.o
# myprog depends on four object files

myprog.o: myprog.f
<TAB>g77 -c myprog.f
# myprog.o depends on its source file

file_1.o: file_1.f
<TAB>g77 -c file_1.f
# file_1.o depends on its source file

file_2.o: file_2.f file_1.o
<TAB>g77 -c file_2.f file_1.o
# file_2.o depends on its source file and an object file

file_3.o: file_3.f file_2.o
<TAB>g77 -c file_3.f file_2.o
# file_3.o depends on its source file and an object file

# end of makefile.
```

Save this file as `Makefile` and type `make` to compile your program; alternatively, save it as `myprog.mak` and type `make -f myprog.mak`. And of course, RMP.

5.3 Shell Scripts

Shell scripts are the equivalent of VMS' command files, and allow for very powerful constructs.

To write a script, all you have to do is write a standard ASCII file containing the commands, save it, then make it executable with the command `chmod +x <scriptfile>`. To execute it, type its name.

Writing scripts under `bash` is such a vast subject it would require a book by itself, and I will not delve into the topic any further. I'll just give you a more−or−less comprehensive and (hopefully) useful example you can extract some basic rules from.

EXAMPLE: sample.sh

```
#!/bin/sh
# sample.sh
# I am a comment
# don't change the first line, it must be there
echo "This system is: `uname -a`" # use the output of the command
echo "My name is $0" # built-in variables
echo "You gave me the following $# parameters: "$*
echo "First parameter is: "$1
echo -n "What's your name? " ; read your_name
echo notice the difference: "hi $your_name" # quoting with "
```

```
echo notice the difference: 'hi $your_name' # quoting with '
DIRS=0 ; FILES=0
for file in `ls .` ; do
  if [ -d ${file} ] ; then # if file is a directory
    DIRS=`expr $DIRS + 1`  # this means DIRS = DIRS + 1
  elif [ -f ${file} ] ; then
    FILES=`expr $FILES + 1`
  fi
  case ${file} in
    *.gif|*jpg) echo "${file}: graphic file" ;;
    *.txt|*.tex) echo "${file}: text file" ;;
    *.c|*.f|*.for) echo "${file}: source file" ;;
    *) echo "${file}: generic file" ;;
  esac
done
echo "there are ${DIRS} directories and ${FILES} files"
ls | grep "ZxY--!!!WKW"
if [ $? != 0 ] ; then # exit code of last command
  echo "ZxY--!!!WKW not found"
fi
echo "enough... type 'man bash' if you want more info."
```

5.4 C

Linux is an excellent environment to program in C. Taken for granted that you know C, here are a couple of guidelines. To compile your standard `hello.c` you'll use the `gcc` compiler, which comes as part of Linux and has the same syntax as `g77`:

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

To link a library to a program, add the switch `-l<libname>`. For example, to link the math library and optimize do

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

(The `-l<libname>` switch forces `gcc` to link the library `/usr/lib/lib<libname>.a`; so `-lm` links `/usr/lib/libm.a`).

When your program is made of several source files, you'll need to use the utility `make` described above. Just use `gcc` and C source files in the makefile.

You can invoke some help about the C functions, that are covered by man pages, section 3; for example,

```
$ man 3 printf
```

There are lots of libraries available out there; among the first you'll want to use are `ncurses`, to handle text mode effects, and `svgalib`, to do graphics.

6.Graphics

Among the scores of graphic packages available, `gnuplot` stands out for its power and ease of use. Go to X and type `gnuplot`, and have two sample data files ready: `2D-data.dat` (two data per line), and `3D-data.dat` (three data per line).

Examples of 2-D graphs:

```
gnuplot> set title "my first graph"
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'
```

Example of 3-D graphs (each `row' of X values is followed by a blank line):

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

A single-column datafile (e.g., a time series) can also be plotted as a 2-D graph:

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

or as a 3-D graph (blank lines in the datafile, as above):

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

To print a graph: if the command to print on your Postscript printer is `lpr -Pps file.ps`, issue:

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

```
gnuplot> replot
```

then type `set term x11` to restore. Don't get confused−−−the last print will come out only when you quit `gnuplot`.

For more info, type `help` or see the examples in directory `/usr/lib/gnuplot/demos/`, if you have it.

7.Mail and Internet Tools

Since Internet was born on UNIX machines, you find plenty of nice and easy−to−use applications under Linux. Here are just some:

- **Mail**: use `elm` or `pine` to handle your email; both programs have on−line help. For short messages, you could use `mail`, as in `mail -s "hello mate" user@somewhere < msg.txt`. You may like programs like `xmail` or some such.
- **Newsgroups**: use `tin` or `slrn`, both very intuitive and self−explanatory.
- **ftp**: apart from the usual character−based `ftp`, ask your sysadm to install the full−screen `ncftp` or a graphical ftp client like `xftp`.
- **WWW**: the ubiquitous `netscape`, or `xmosaic`, `chimera`, and `arena` are graphical web browsers; a character−based one is `lynx`, quick and effective.

8.Advanced Topics

Here the game gets tough. Learn these features, then you'll be ready to say that you `know something about Linux' ;−)

8.1 Permissions and Ownership

Files and directories have permissions (`protections') and ownership, just like under VMS. If you can't run a program, or can't modify a file, or can't access a directory, it's because you don't have the permission to do so, and/or because the file doesn't belong to you. Let's have a look at the following example:

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

The first field shows the permissions of the file `ls` (owner root, group bin). There are three types of ownership: owner, group, and others (similar to VMS owner, group, world), and three types of permissions: read, write (and delete), and execute.

From left to right, `-` is the file type (`-` = ordinary file, `d` = directory, `l` = link, etc); `rwx` are the permissions for the file owner (read, write, execute); `r-x` are the permissions for the group of the file owner (read, execute); `r-x` are the permissions for all other users (read, execute).

To change a file's permissions:

```
$ chmod <whoXperm> <file>
```

where who is `u` (user, that is owner), `g` (group), `o` (other), X is either + or `-`, perm is `r` (read), `w` (write), or `x` (execute). Examples:

```
$ chmod u+x file
```

this sets the execute permission for the file owner. Shortcut: `chmod +x file`.

```
$ chmod go-wx file
```

this removes write and execute permission for everyone except the owner.

```
$ chmod ugo+rwx file
```

this gives everyone read, write, and execute permission.

A shorter way to refer to permissions is with numbers: `rwxr-xr-x` can be expressed as 755 (every letter corresponds to a bit: `---` is 0, `--x` is 1, `-w-` is 2...).

For a directory, `rx` means that you can `cd` to that directory, and `w` means that you can delete a file in the directory (according to the file's permissions, of course), or the directory itself. All this is only part of the matter---RMP.

To change a file's owner:

```
$ chown username file
```

To sum up, a table:

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

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

8.2 Multitasking: Processes and Jobs

More about running programs. There are no `batch queues' under Linux as you're used to; multitasking is handled very differently. Again, this is what the typical command line looks like:

```
$ command -s1 -s2 ... -sn par1 par2 ... parn < input > output &
```

where `-s1`, ..., `-sn` are the program switches, `par1`, ..., `parn` are the program parameters.

Now let's see how multitasking works. Programs, running in foreground or background, are called `processes'.

- To launch a process in background:

  ```
  $ progname [-switches] [parameters] [< input] [> output] &
  [1] 234
  ```

  the shell tells you what the `job number' (the first digit; see below) and PID (Process IDentifier) of the process are. Each process is identified by its PID.
- To see how many processes there are:

  ```
  $ ps -ax
  ```

  This will output a list of currently running processes.
- To kill a process:

```
$ kill <PID>
```

You may need to kill a process when you don't know how to quit it the right way... ;–). Sometimes, a process will only be killed by one of the following:

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

In addition to this, the shell allows you to stop or temporarily suspend a process, send a process to background, and bring a process from background to foreground. In this context, processes are called `jobs'.

- To see how many jobs there are:

  ```
  $ jobs
  ```

  jobs are identified by the numbers the shell gives them, not by their PID.
- To stop a process running in foreground:

  ```
  $ CTRL-C
  ```

  (it doesn't always work)
- To suspend a process running in foreground:

  ```
  $ CTRL-Z
  ```

  (ditto)
- To send a suspended process into background (it becomes a job):

  ```
  $ bg <job>
  ```

- To bring a job to foreground:

  ```
  $ fg <job>
  ```

- To kill a job:

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

8.3 Files, Revisited

More information about files.

- **stdin, stdout, stderr**: under UNIX, every system component is treated as if it were a file. Commands and programs get their input from a `file' called `stdin` (standard input; usually, the keyboard), put their output on a `file' called `stdout` (usually, the screen), and error messages go to a `file' called `stderr` (usually, the screen). Using < and > you redirect input and output to a different file. Moreover, >> appends the output to a file instead of overwriting it; `2>` redirects error messages (stderr); 2>&1 redirects stderr to stdout, while 1>&2 redirects stdout to stderr. There's a `black hole' called `/dev/null`: everything redirected to it disappears;
- **wildcards**: '`*`' is almost the same. Usage: * matches all files except the hidden ones; .* matches all hidden files; *.* matches only those that have a '.' in the middle, followed by other characters; p*r matches both `peter' and `piper'; *c* matches both `picked' and `peck'. '`%`' becomes '`?`'. There is another wildcard: the `[]`. Usage: `[abc]*` matches files starting with a, b, c; `*[I-N,1,2,3]` matches files ending with I, J, K, L, M, N, 1, 2, 3;
- `mv` (`RENAME`) doesn't work for multiple files; that is, `mv *.xxx *.yyy` won't work;
- use `cp -i` and `mv -i` to be warned when a file is going to be overwritten.

8.4 Print Queues

Your prints are queued, like under VMS. When you issue a print command, you may specify a printer name. Example:

```
$ lpr file.txt          # this goes to the standard printer
$ lpr -Plaser file.ps   # this goes to the printer named 'laser'
```

To handle the print queues, you use the following commands:

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

$ PRINT file.ps                         $ lpr file.ps
$ PRINT/QUEUE=laser file.ps             $ lpr -Plaser file.ps
$ SHOW QUEUE                            $ lpq
$ SHOW QUEUE/QUEUE=laser                $ lpq -Plaser
$ STOP/QUEUE                            $ lprm <item>
```

9.Configuring

Your sysadm has already provided you with a number of configuration files like `.xinitrc`, `.bash_profile`, `.inputrc`, and many others. The ones you may want to edit are:

- `.bash_profile` or `.profile`: read by the shell at login time. It's like `LOGIN.COM`;
- `.bash_logout`: read by the shell at logout. It's like `LOGOUT.COM`;
- `.bashrc`: read by non−−login shells.
- `.inputrc`: this file customises the key bindings and the behaviour of the shell.

To give you an example, I'll include my `.bash_profile` (abridged):

```
# $HOME/.bash_profile

# don't redefine the path if not necessary
echo $PATH | grep $LOGNAME > /dev/null
if [ $? != 0 ]
then
  export PATH="$PATH:/home/$LOGNAME/bin"  # add my dir to the PATH
fi

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

# aliases

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"

# A few useful functions

inst() # Install a .tar.gz archive in current directory.
{
  gzip -dc $1 | tar xvf -
}
cz() # List the contents of a .zip archive.
{
  unzip -l $*
}
ctgz() # List the contents of a .tar.gz archive.
{
  for file in $* ; do
    gzip -dc ${file} | tar tf -
  done
}
tgz() # Create a .tgz archive a la zip.
{
  name=$1 ; tar -cvf $1 ; shift
  tar -rf ${name} $* ; gzip -S .tgz ${name}
}
```

And this is my `.inputrc`:

```
# $HOME/.inputrc
# Last modified: 16 January 1997.
#
# This file is read by bash and defines key bindings to be used by the shell;
# what follows fixes the keys END, HOME, and DELETE, plus accented letters.
# For more 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) are "\e[[A" ... "\e[[E"

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

10.Useful Programs

10.1 Browsing Files: `less`

You'll use this file browser every day, so I'll give you a couple of tips to use it at best. First of all, ask your sysadm to configure `less` so as it can display not only plain text files, but also compressed files, archives, and so on.

Like recent versions of `TYPE`, `less` lets you browse files in both directions. It also accepts several commands that are issued pressing a key. The most useful are:

- first of all, press `q` to leave the browser;
- `h` gives you extensive help;
- `g` to go to beginning of file, `G` to the end, number+`g` to go to line `number' (e.g. `125g`), number+`%` to move to that percentage of the file;
- `/pattern` searches forwards for `pattern'; `n` searches forwards for the next match; `?pattern` and `N` search backwards;
- `m`+letter marks current position (e.g. `ma`); `'`+letter go to the marked position.
- `:e` examines a new file;

* `!command` executes the shell command.

10.2 Numbered Backups Under Linux

Alas, Linux doesn't still support file version numbers, but you overcome this limitation in two ways. The first is to use RCS, the Revision Control System, which allows you to keep previous versions of a file. RCS is covered in ``The RCS MINI–HOWTO'' (http://sunsite.unc.edu/mdw/HOWTO/mini/RCS.html).

The second way is to use an editor that knows how to deal with numbered backups; `emacs` and `jed` are OK. In `emacs`, add these lines in your `.emacs`:

```
(setq version-control t)
(setq kept-new-versions 15) ;;; or any other value
(setq kept-old-versions 15)
(setq backup-by-copying-when-linked t)
(setq backup-by-copying-when-mismatch t)
```

In `jed`, make sure you have version 0.98.7 or newer; the patch for numbered backups is available on `http://ibogeo.df.unibo.it/guido/slang/backups.sl`.

10.3 Archiving: tar & gzip

Under UNIX there are some widely used applications to archive and compress files. `tar` is used to make archives, that is collections of files. To make a new archive:

```
$ tar -cvf <archive_name.tar> <file> [file...]
```

To extract files from an archive:

```
$ tar -xpvf <archive_name.tar> [file...]
```

To list the contents of an archive:

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

Files can be compressed to save disk space using `compress`, which is obsolete and shouldn't be used any more, or `gzip`:

```
$ compress <file>
$ gzip <file>
```

that creates a compressed file with extension .Z (`compress`) or .gz (`gzip`). These programs don't make archives, but compress files individually. To decompress, use:

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

RMP.

The `unarj`, `zip` and `unzip` utilities are also available. Files with extension `.tar.gz` or `.tgz` (archived with `tar`, then compressed with `gzip`) are very common in the UNIX world. Here's how to list the contents of a `.tar.gz` archive:

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

To extract the files from a `.tar.gz` archive:

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

11.Real Life Examples

UNIX' core idea is that there are many simple commands that can linked together via piping and redirection to accomplish even really complex tasks. Have a look at the following examples. I'll only explain the most complex ones; for the others, please study the above sections and the man pages.

Problem: `ls` is too quick and the file names fly away.

Solution:

```
$ ls | less
```

Problem: I have a file containing a list of words. I want to sort it in reverse order and print it.

Solution:

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

Problem: my data file has some repeated lines! How do I get rid of them?

Solution:

```
$ sort datafile.dat | uniq > newfile.dat
```

Problem: I have a file called 'mypaper.txt' or 'mypaper.tex' or some such somewhere, but I don't remember where I put it. How do I find it?

Solution:

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

Explanation: `find` is a very useful command that lists all the files in a directory tree (starting from ~ in this case). Its output can be filtered to meet several criteria, such as `-name`.

Problem: I have a text file containing the word 'entropy' in this directory, is there anything like `SEARCH`?

Solution: yes, try

```
$ grep -l 'entropy' *
```

Problem: somewhere I have text files containing the word 'entropy', I'd like to know which and where they are. Under VMS I'd use `search entropy [...]*.*;*`, but `grep` can't recurse subdirectories. Now what?

Solution:

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

Explanation: `find .` outputs all the file names starting from the current directory, `-exec grep -l "entropy"` is an action to be performed on each file (represented by `{}`), \ terminates the command. If you think this syntax is awful, you're right.

In alternative, write the following script:

```
#!/bin/sh
# rgrep: recursive grep
if [ $# != 3 ]
then
  echo "Usage: rgrep --switches 'pattern' 'directory'"
  exit 1
fi
find $3 -name "*" -exec grep $1 $2 {} \; 2> /dev/null
```

Explanation: `grep` works like `search`, and combining it with `find` we get the best of both worlds.

Problem: I have a data file that has two header lines, then every line has 'n' data, not necessarily equally spaced. I want the 2nd and 5th data value of each line. Shall I write a Fortran program...?

Solution: nope. This is quicker:

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

Explanation: the command `awk` is actually a programming language: for each line starting from the third in `datafile.dat`, print out the second and fifth field, separated by a tab. Learn some `awk`---it saves a lot of time.

Problem: I've downloaded an FTP site's `ls-lR.gz` to check its contents. For each subdirectory, it contains a line that reads "total xxxx", where xxxx is size in kbytes of the dir contents. I'd like to get the grand total of all these xxxx values.

Solution:

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

Explanation: `zcat` outputs the contents of the `.gz` file and pipes to `awk`, whose man page you're kindly requested to read ;-)

Problem: I've written a Fortran program, `myprog`, to calculate a parameter from a data file. I'd like to run it on hundreds of data files and have a list of the results, but it's a nuisance to ask each time for the file name. Under VMS I'd write a lengthy command file, and under Linux?

Solution: a very short script. Make your program look for the data file '`mydata.dat`' and print the result on the screen (stdout), then write the following script:

```
#!/bin/sh
# myprog.sh: run the same command on many different files
# usage: myprog.sh *.dat
for file in $*  # for all parameters (e.g. *.dat)
do
  # append the file name to result.dat
  echo -n "${file}:    " >> results.dat
  # copy current argument to mydata.dat, run myprog
  # and append the output to results.dat
  cp ${file} mydata.dat ; myprog >> results.dat
done
```

Problem: I want to replace `geology' with `geophysics' in all my text files. Shall I edit them all manually?

Solution: nope. Write this shell script:

```
#!/bin/sh
# replace $1 with $2 in $*
# usage: replace "old-pattern" "new-pattern" file [file...]
OLD=$1          # first parameter of the script
NEW=$2          # second parameter
shift ; shift   # discard the first 2 parameters: the next are the file names
for file in $*  # for all files given as parameters
do
# replace every occurrence of OLD with NEW, save on a temporary file
  sed "s/$OLD/$NEW/g" ${file} > ${file}.new
# rename the temporary file as the original file
  /bin/mv ${file}.new ${file}
done
```

Problem: I have some data files, I don't know their length and have to remove their last but one and last but two lines. Er... manually?

Solution: no, of course. Write this script:

```
#!/bin/sh
# prune.sh: removes n-1th and n-2th lines from files
# usage: prune.sh file [file...]
for file in $*   # for every parameter
do
  LINES=`wc -l $file | awk '{print $1}'`  # number of lines in file
  LINES=`expr $LINES - 3`                 # LINES = LINES - 3
  head -n $LINES $file > $file.new        # output first LINES lines
  tail -n 1 $file >> $file.new            # append last line
done
```

I hope these examples whetted your appetite...

12.Tips You Can't Do Without

- **Command completion**: pressing <TAB> when issuing a command will complete the command line for you. Example: you have to type `less this_is_a_long_name`; typing in `less thi<TAB>` will suffice. (If you have other files that start with the same characters, supply enough characters to resolve any ambiguity.)
- **Back−scrolling**: pressing SHIFT−−PAG UP (the grey key) allows you to backscroll a few pages, depending on your PC's video memory.
- **Resetting the screen**: if you happen to `more` or `cat` a binary file, your screen may end up full of garbage. To fix things, blind type `reset` or this sequence of characters: `echo CTRL−V ESC c RETURN`.
- **Pasting text**: in console, see below; in X, click and drag to select the text in an `xterm` window, then click the middle button (or the two buttons together if you have a two−button mouse) to paste.
- **Using the mouse**: ask your sysadm to install `gpm`, a mouse driver for the console. Click and drag to select text, then right click to paste the selected text. It works across different VCs.

13.Reading VMS tapes from Linux

(This section was written by Mike Miller)

13.1 Introduction

From time to time you may want to read tapes made on a VMS machine (or tapes that are made to be readable by VMS and *nix systems). In general, this is quite easy for DECFILES11A tapes.

Although you may be reading this as part of a Linux mini−HOWTO, I believe that the information here is applicable to any *nix system − I've done this on Linux, HP, Sun and DEC *nix systems. The main platform dependences that I know are device names, which can differ on different systems, and the options to mt for specifying the device name (for example, mt −f on Linux and mt −t on HPUX 9).

Caveat − I've only tried this with Exabyte 8mm SCSI tape drives. If you've read other formats (still got those 9−tracks lying around?) let me know and I'll add a note here.

13.2 The Basics

When reading a tape that has been made with the VMS ``copy'' command (or has at least been made to look like it was made with copy) all you need to know is there will be three files on the tape for each actual data file − a header, the data, and a trailer. The header and trailer are interesting in that they contain info on the file as it existed under VMS. The data is, well, the data. Each of these files can be extracted from the tape with the dd command. The tape can be positioned by skipping around with the mt command.

Example: I've got VMS tape with a series of files on it. The first two were originally named ce66−2.evt and ce66−3.evt on a VMS system. The tape label is c66a2.

If I execute these commands:

```
> dd if=$TAPE bs=16k of=header1
> dd if=$TAPE bs=16k of=data1
> dd if=$TAPE bs=16k of=trailer1
> dd if=$TAPE bs=16k of=header2
> dd if=$TAPE bs=16k of=data2
> dd if=$TAPE bs=16k of=trailer2
```

I'm left with six files: header1, data1, trailer1, header2, data2 and trailer2. The syntax here is if="input file", bs="block size" and of="output file". TAPE is expected to be a variable containing the device name of your tape drive − for example, /dev/nts0 if you are using the first SCSI tape on Linux.

If you wanted to read the second file, but not the first, you didn't care about the header, and you wanted to use the original file name, do this:

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

Note the 4 − skip three files for the first file on the tape and one for the next header. The second mt skips the second file's trailer and positions the tape at the beginning of the next file − the third VMS header. You can also use mt to skip backwards (bsf), rewind (rewind) and rewind and unload the tape (offline, rewoffl).

13.3 Some details

The header and trailer files contain uppercase ASCII data used by VMS to store file information such as block size. They also contain the file name, which can be handy if you want to build scripts that automate read files or search for a particular file. The first header on a tape volume is slightly different than subsequent headers.

For a file that is the first file on the tape, as in header1 of the above example, the first four characters will be "VOL1" followed by the volume name. In the example, header1 starts with "VOL1C66A2". This is followed by a series of spaces terminated with a numeral. After that is the string "HDR1" which indicates that this is a file header. The characters immediately following the HDR1 string are the VMS file name. In in the example, this is "HDR1CE66−2.EVT". The next field is the volume name again.

For files that are not the first file on the tape, the initial VOL1 field is not present. Other than that the header has the same structure as for the initial file. Another useful field is the 7th field, which will end with "DECFILES11A". This must be present on tapes that conform to the DEC Files11A standard.

```
field     initial header         subsequent headers
=====     =============          ==================
  1       VOL1 + volume name     HDR1 + file name
  2       3HDR1 + file name      volume name
  3       volume name

  6                              ...DECFILES11A
  7      ...DECFILES11A
```

For full details on the header and trailer format, see the DEC FILES11A documentation (on the orange/grey wall − ask your local VMS folks :−).

13.4 Comment on Block Size

In the example, I used a block size of 16k. On a *nix system, there is no block size associated with a file on disk while, under VMS, each file has a specific block size. That means that block size doesn't matter too much on the Linux end... unless it makes it hard to read the tape. If you have difficulty figuring out the block size and reading a tape, you can try setting the hardware block size on your tape drive using `mt −f $TAPE setblk 0'. The exact form of the setblk option (and its availability) may depend on the version of mt, the tape drive hardware interface and on your particular flavor of *nix.

(Thanks to Wojtek Skulski (skulski@nsrlc6.nsrl.rochester.edu) for pointing out setblk.)

14.The End

14.1 Copyright

14.2 Disclaimer

This work was written following the experience we had at the Settore di Geofisica of the Universita' di Bologna (Italy), where a VAX 4000 has been superseded and replaced by Linux−based Pentium PCs. Most of my colleagues are VMS users, and some of them have switched to Linux.

``From VMS to Linux HOWTO'' was written by Guido Gonzato, guido ``at'' ibogeo.df.unibo.it, and Mike Miller, miller5@uiuc.edu who contributed the section on reading VMS tapes. Many thanks to my colleagues and friends who helped me define the needs and habits of the average VMS user, especially to Dr. Warner Marzocchi.

Please help me improve this HOWTO. I'm not a VMS expert and never will be, so your suggestions and bug reports are more than welcome.

Enjoy,

Guido `=8−)`
