From DOS to Linux HOWTO

By Guido Gonzato <guido@ibogfs.df.unibo.it> v1.0, 11 December 1996

This HOWTO is dedicated to all the (soon to be former?) DOS users who have just taken the plunge and decided to switch to Linux, the free Unix clone for 386+ computers. Given the similarities between DOS and Unix, the purpose of this document is to help the reader translate his or her knowledge of DOS into the Linux environment, so as to be productive asap.

Contents

1 Introduction

1.1 Is Linux Right for You?

You want to switch from DOS to Linux? Good idea, but beware: it might not be useful for you. IMHO, there is no such thing as "the best computer" or "the best operating system": it depends on what one has to do. That's why I don't believe that Linux is the best solution for everyone, even if it is technically superior to many commercial oses. You're going to benefit immensely from Linux if what you need is sw for programming, the Internet, TeX... technical sw in general, but if you mostly need commercial sw, or if you don't feel like learning and typing commands, look elsewhere.

Linux is not (for now) as easy to use and configure as Windows or the Mac, so be prepared to hack quite a bit. In spite of these warnings, let me tell you that I'm 100% confident that if you belong to the right user type you'll find in Linux your computer Nirvana. It's up to you. And remember that Linux + DOS/Windows can coexist on the same machine, anyway.

Prerequisites for this howto: I'll assume that

- you know the basic DOS commands and concepts;
- Linux, possibly with X Window System, is properly installed on your PC;
- your shell—the equivalent of `COMMAND.COM`—is `bash`;

- you understand that this guide is only an incomplete primer. For more information, please refer to Matt Welsh's "Linux Installation and Getting Started" and/or Larry Greenfield's "Linux User Guide" (`sunsite.unc.edu:/pub/Linux/docs/LDP`).

This howto replaces the old "From DOS to Linux - Quick!" mini-howto.

1.2 It Is. Tell Me More

You installed Linux and the programs you needed on the PC. You gave yourself an account (if not, type `adduser` *now!*) and Linux is running. You've just entered your name and password, and now you are looking at the screen thinking: "Well, now what?"

Now, don't despair. You're almost ready to do the same things you used to do with DOS, and many more. If you were running DOS instead of Linux, you would be doing some of the following tasks:

- running programs and creating, copying, viewing, deleting, printing, renaming files;
- CD'ing, MD'ing, RD'ing, and DIR'ring your directories;
- formatting floppies and copying files from/to them;
- mending your `AUTOEXEC.BAT` and `CONFIG.SYS`;
- writing your own .BAT files and/or `QBasic` programs;
- the remaining 1%.

You'll be glad to know that these tasks can be accomplished under Linux in a fashion similar to DOS. Under DOS, the average user uses very few of the 100+ commands available: the same, up to a point, holds for Linux.

A few things to point out before going on:

- first, how to get out. To quit Linux: if you see a textmode screen, press CTRL-ALT-DEL, wait for the system to fix its innards and tell you everything is OK, then switch off the PC. If you are working under X Window System, press CTRL-ALT-BACKSPACE first, then CTRL-ALT-DEL. *Never* switch off or reset the PC directly: it may damage the filesystem;
- unlike DOS, Linux has built-in security mechanisms, due to its multiuser nature. Files and directories have permissions associated to them, and therefore some cannot be accessed by the normal user; (see Section 2.3). only the user whose login name is "root" has the power. (This guy's the system administrator. If you work on your own PC, you'll be root as well.) DOS, on the contrary, will let you wipe out the entire contents of your hard disk;
- you are strongly encouraged to experiment, play, try by yourself: it surely won't hurt. You can get some help typing at the prompt ($ is the standard prompt, # is the prompt for root):

```
$ help
```

(this gives help about `bash`), or get info about a command typing

```
$ man command
```

which, if you have installed the man pages, will invoke the manual ("man") page associated to `command`. You may also try:

```
$ apropos command
$ whatis command
```

and press 'q' to exit;

- most of the power and flexibility of Unix comes from the simple concepts of redirection and piping, more powerful than under DOS. Simple commands can be strung together to accomplish complex tasks. Do use these features!
- conventions: `<...>` means something that must be specified, while `[...]` something optional. Example:

```
$ tar -tf <file.tar> [> redir_file]
```

`file.tar` must be indicated, but redirection to `redir_file` is optional.

- from now on "RMP" means "please read the man pages for further information".

1.3 For the Impatient

Want to strike out? Have a look at this:

```
DOS                     Linux                   Notes
------------------------------------------------------------------------------


BACKUP                  tar -Mcvf device dir/   completely different
CD dirname\             cd dirname/             almost the same syntax
COPY file1 file2        cp file1 file2          ditto
DEL file                rm file                 beware - no undelete
DELTREE dirname         rm -R dirname/          ditto
DIR                     ls                      not exactly the same syntax
EDIT file               vi file                 I think you won't like it
                        emacs file              this is better
                        jstar file              feels like dos' edit
FORMAT                  fdformat,
                        mount, umount           quite different syntax
HELP command            man command             same philosophy
MD dirname              mkdir dirname/          almost the same syntax
MOVE file1 file2        mv file1 file2          ditto
NUL                     /dev/null               ditto
PRINT file              lpr file                ditto
```

PRN	/dev/lp0,	
	/dev/lp1	ditto
RD dirname	rmdir dirname/	almost the same syntax
REN file1 file2	mv file1 file2	not for multiple files
RESTORE	tar -Mxpvf device	different syntax
TYPE file	less file	much better
WIN	startx	poles apart!

If you need more than a table of commands, please refer to the following sections.

2 Files and Programs

2.1 Files: Preliminary Notions

Linux has a file system—meaning by that "the structure of directories and files therein"—very similar to that of DOS. Files have filenames that obey special rules, are stored in directories, some are executable, and among these most have command switches. Moreover, you can use wildcard characters, redirection, and piping. There are only a few minor differences:

- under DOS, file names are in the so-called 8.3 form; e.g. `NOTENOUG.TXT`. Under Linux we can do better. If you installed Linux using a filesystem like ext2 or umsdos, you can use longer filenames (up to 255 characters), and with more than one dot in them: for example, `This_is.a.VERY_long.filename`. Please note that I used both upper and lower case characters: in fact...
- upper and lower case characters in file names or commands are different. Therefore, `FILENAME.tar.gz` and `filename.tar.gz` are two different files. `ls` is a command, `LS` is a mistake;
- there are no compulsory exensions like .COM and .EXE for programs, or .BAT for batch files. Executable files are marked by an asterisk '`*`' at the end of their name when you issue the `ls -F` command. For example:

  ```
  $ ls -F
  letter_to_Joe  cindy.jpg  cjpg*  I_am_a_dir/  my_1st_script*  old~
  ```

  The files `cjpg*` and `my_1st_script*` are executable—"programs". Under DOS, backup files end in .BAK, while under Linux they end with a tilde '~'. Further, a file whose name starts with a dot is considered as hidden. Example: the file `.I.am.a.hidden.file` won't show up after the `ls` command;
- DOS program switches are obtained with `/switch`, Linux switches with `-switch` or `--switch`. Example: `dir /s` becomes `ls -R`. Note that many DOS programs, like `PKZIP` or `ARJ`, use Unix-style switches.

You can now jump to Section 2.4, but if I were you I'd read on.

2.2 Symbolic Links

Unix has a type of file that doesn't exist under DOS: the symbolic link. This can be thought of as a pointer to a file or to a directory, and can be used instead of the file or directory it points to; it's similar to Win 95 shortcuts. Examples of symbolic links are `/usr/X11`, which points to `/usr/X11R6`; `/dev/modem`, which points to either `/dev/cua0` or `/dev/cua1`.

To make a symbolic link:

```
$ ln -s <file_or_dir> <linkname>
```

Example:

```
$ ln -s /usr/doc/g77/DOC g77manual.txt
```

Now you can refer to `g77manual.txt` instead of `/usr/doc/g77/DOC`.

2.3 Permissions and Ownership

DOS files and directories have the following attributes: A (archive), H (hidden), R (read-only), and S (system). Only H and R make sense under Linux: hidden files start with a dot, and for the R attribute, read on.

Under Unix a file has "permissions" and an owner, who belongs to a "group". Look at this example:

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

The first field contains the permissions of the file `/bin/ls`, which belongs to root, group bin. Leaving the remaining information aside (Matt's book is there for that purpose), remember that `-rwxr-xr-x` means (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); (I won't cover the concept of group, you can survive without it as long as you're a beginner ;-) `r-x` are the permissions for all other users (read, execute).

This is why you can't delete the file `/bin/ls` unless you are root: you don't have the write permission to do so. To change a file's permissions, the command is:

```
$ 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 but the owner.

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

this gives everyone read, write, and execute permission.

```
# chmod +s file
```

this makes a so-called "setuid" or "suid" file—a file that everyone can execute with root privileges.

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, `-wx` is 3...). It looks difficult, but with a bit of practice you'll understand the concept.

root, being the so-called superuser, can change everyone's file permissions. There's more to it—RMP.

2.4 Translating Commands from DOS to Linux

On the left, the DOS commands; on the right, their Linux counterpart.

```
COPY:          cp
DEL:           rm
MOVE:          mv
REN:           mv
TYPE:          more, less, cat
```

Redirection and plumbing operators: `< > >> |`

Wildcards: `* ?`

`nul:  /dev/null`

`prn, lpt1:  /dev/lp0 or /dev/lp1; lpr`

- EXAMPLES -

```
DOS                                     Linux
---------------------------------------------------------------------

C:\GUIDO>copy joe.txt joe.doc           $ cp joe.txt joe.doc
C:\GUIDO>copy *.* total                 $ cat * > total
C:\GUIDO>copy fractals.doc prn          $ lpr fractals.doc
C:\GUIDO>del temp                       $ rm temp
C:\GUIDO>del *.bak                      $ rm *~
```

```
C:\GUIDO>move paper.txt tmp\         $ mv paper.txt tmp/
C:\GUIDO>ren paper.txt paper.asc     $ mv paper.txt paper.asc
C:\GUIDO>print letter.txt            $ lpr letter.txt
C:\GUIDO>type letter.txt             $ more letter.txt
C:\GUIDO>type letter.txt             $ less letter.txt
C:\GUIDO>type letter.txt > nul       $ cat letter.txt > /dev/null
        n/a                          $ more *.txt *.asc
        n/a                          $ cat section*.txt | less
```

Notes:

- * is smarter under Linux: * 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';
- when using `more`, press SPACE to read through the file, 'q' or CTRL-C to exit. `less` is more inuitive and lets you use the arrow keys;
- there is no `UNDELETE`, so *think twice* before deleting anything;
- in addition to DOS's < > >>, Linux has `2>` to redirect error messages (stderr); moreover, `2>&1` redirects stderr to stdout, while `1>&2` redirects stdout to stderr;
- Linux has another wildcard: the `[]`. Use: `[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;
- there is no DOS-like `RENAME`; 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.

2.5 Running Programs: Multitasking and Sessions

To run a program, type its name as you would do under DOS. If the directory (Section 3) where the program is stored is included in the PATH (Section 5.1), the program will start. Exception: unlike DOS, under Linux a program located in the current directory won't run unless the directory is included in the PATH. Escamotage: being `prog` your program, type `./prog`.

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. You can issue several commands on the command line:

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

That's all about running programs, but it's easy to go a step beyond. One of the main reasons for using Linux is that it is a multitasking os—it can run several programs (from now on, processes) at the same time. You can launch processes in background and continue working straight away. Moreover, Linux lets you have several sessions: it's like having many computers to work on at once!

- To switch to session 1..6:

  ```
  $ ALT-F1 ... ALT-F6
  ```

- To start a new session without leaving the current one:

  ```
  $ su - <loginname>
  ```

  Example:

  ```
  $ su - root
  ```

  This is useful, for one, when you need to mount a disk (Section 4): normally, only root can do that.

- To end a session:

  ```
  $ exit
  ```

  If there are stopped jobs (see later), you'll be warned.

- To launch a process in foreground:

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

- To launch a process in background, add an ampersand '`&`' at the end of the command line:

  ```
  $ progname [-switches] [parameters] [< input] [> output] &
  [1] 123
  ```

  the shell identifies the process with a job number (e.g. `[1]`; see below), and with a PID (123 in our example).

- To see how many processes there are:

  ```
  $ ps -a
  ```

  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 either 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
```

  here jobs are identified by their job number, not by their PID.

- To stop a process running in foreground (it won't always work):

```
$ CTRL-C
```

- To suspend a process running in foreground (ditto):

```
$ CTRL-Z
```

- 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>
```

  where <job> may be 1, 2, 3, ... Using these commands you can format a disk, zip a bunch of files, compile a program, and unzip an archive all at the same time, and still have the prompt at your disposal. Try this with DOS! And try with Windows, just to see the difference in performance.

2.6 Running Programs on Remote Computers

To run a program on a remote machine whose IP address is `remote.bigone.edu`, you do:

```
$ telnet remote.bigone.edu
```

After logging in, start your favourite program. Needless to say, you must have an account on the remote machine.

If you have X11, you can even run an X application on a remote computer, displaying it on your X screen. Let `remote.bigone.edu` be the remote X computer and `local.linux.box` be your Linux machine. To run from `local.linux.box` an X program that resides on `remote.bigone.edu`, do the following:

- fire up X11, start an `xterm` or equivalent terminal emulator, then type:

```
$ xhost +remote.bigone.edu
$ telnet remote.bigone.edu
```

- after logging in, type:

```
remote:$ DISPLAY=local.linux.box:0.0
remote:$ progname &
```

(instead of `DISPLAY...`, you may have to write `setenv DISPLAY local.linux.box:0.0`. It depends on the remote shell.)

Et voila! Now `progname` will start on `remote.bigone.edu` and will be displayed on your machine. Don't try this over a ppp line, though.

3 Using Directories

3.1 Directories: Preliminary Notions

We have seen the differences between files under DOS and Linux. As for directories, under DOS the root directory is under Linux / is. Similarly, nested directories are separated by under DOS, by / under Linux. Example of file paths:

```
DOS:    C:\PAPERS\GEOLOGY\MID_EOC.TEX
Linux:  /home/guido/papers/geology/mid_eocene.tex
```

As usual, `..` is the parent directory, `.` is the current directory. Remember that the system won't let you `cd`, `rd`, or `md` everywhere you want. Each user starts from his or her own directory called 'home', given by the system administrator; for instance, on my PC my home dir is `/home/guido`.

3.2 Directories Permissions

Directories, too, have permissions. What we have seen in Section 2.3 holds for directories as well (user, group, and other). For a directory, `rx` means 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.

For example, to prevent other users from snooping in `/home/guido/text`:

```
$ chmod o-rwx /home/guido/text
```

3.3 Translating Commands from DOS to Linux

```
DIR:            ls, find, du
CD:             cd, pwd
MD:             mkdir
RD:             rmdir
DELTREE:        rm -R
MOVE:           mv
```

- EXAMPLES -

```
DOS                                     Linux
---------------------------------------------------------------------

C:\GUIDO>dir                            $ ls
C:\GUIDO>dir file.txt                   $ ls file.txt
C:\GUIDO>dir *.h *.c                    $ ls *.h *.c
C:\GUIDO>dir/p                          $ ls | more
C:\GUIDO>dir/a                          $ ls -l
C:\GUIDO>dir *.tmp /s                   $ find / -name "*.tmp"
C:\GUIDO>cd                             $ pwd
        n/a - see note                  $ cd
        ditto                           $ cd ~
        ditto                           $ cd ~/temp
C:\GUIDO>cd \other                      $ cd /other
C:\GUIDO>cd ..\temp\trash               $ cd ../temp/trash
C:\GUIDO>md newprogs                    $ mkdir newprogs
C:\GUIDO>move prog ..                   $ mv prog ..
C:\GUIDO>md \progs\turbo                $ mkdir /progs/turbo
C:\GUIDO>deltree temp\trash             $ rm -R temp/trash
C:\GUIDO>rd newprogs                    $ rmdir newprogs
C:\GUIDO>rd \progs\turbo                $ rmdir /progs/turbo
```

Notes:

1. when using `rmdir`, the directory to remove must be empty. To delete a directory and all of its contents, use `rm -R` (at your own risk).
2. the character '~' is a shortcut for the name of your home directory. The commands `cd` or `cd ~` will take you to your home directory from wherever you are; the command `cd ~/tmp` will take you to `/home/your_home/tmp`.
3. `cd -` "undoes" the last `cd`.

4 Floppies, Hard Disks, and the Like

4.1 Managing Devices

You have never thought about it, but the DOS command `FORMAT A:` does a lot more work than it seems. In fact, when you issue the command `FORMAT` it will: 1) physically format the disk; 2) create the A: directory (= create a filesystem); 3) make the disk available to the user (= mount the disk).

These three steps are addressed separately under Linux. You can use floppies in MS-DOS format, though other formats are available and are better—the MS-DOS format won't let you use long filenames. Here is how to prepare a disk (you'll need to start a session as root):

- To format a standard 1.44 meg floppy disk (A:):

  ```
  # fdformat /dev/fd0H1440
  ```

- To create a filesystem:

  ```
  # mkfs -t ext2 -c /dev/fd0H1440
  ```

  or

  ```
  # mformat a:
  ```

  to create an MS-DOS filesystem. Before using the disk, you must mount it.

- To mount the disk:

  ```
  # mount -t ext2 /dev/fd0 /mnt
  ```

  or

  ```
  # mount -t msdos /dev/fd0 /mnt
  ```

  Now you can address the files in the floppy. When you've finished, before extracting the disk you *must* unmount it.

- To unmount the disk:

  ```
  # umount /mnt
  ```

Now you can extract the disk. Obviously, you have to `fdformat` and `mkfs` only unformatted disks, not previously used ones. If you want to use drive B:, refer to `fd1H1440` and `fd1` instead of `fd0H1440` and `fd0` in the examples above.

All you used to do with A: or B: is now done using `/mnt` instead. Examples:

```
DOS                                     Linux
---------------------------------------------------------------------

C:\GUIDO>dir a:                         $ ls /mnt
C:\GUIDO>copy a:*.*                     $ cp /mnt/* /docs/temp
C:\GUIDO>copy *.zip a:                  $ cp *.zip /mnt/zip
C:\GUIDO>a:                             $ cd /mnt
A:>_                                    /mnt/$ _
```

Needless to say, what holds for floppies also holds for other devices; for instance, you may want to mount another hard disk or a CD-ROM drive. Here's how to mount the CD-ROM:

```
# mount -t iso9660 /dev/cdrom /mnt
```

This was the "official" way to mount your disks, but there's a trick in store. Since it's a bit of a nuisance having to be root to mount a floppy or a CD-ROM, every user can be allowed to mount them this way:

- as root, create the directories `/mnt/a`, `/mnt/a:`, and `/mnt/cdrom`
- add in `/etc/fstab` the following lines:

```
/dev/cdrom      /mnt/cdrom  iso9660 ro,user,noauto         0       0
/dev/fd0        /mnt/a:     msdos   user,noauto            0       0
/dev/fd0        /mnt/a      ext2    user,noauto            0       0
```

Now, to mount a DOS floppy, an ext2 floppy, and a CD-ROM:

```
$ mount /mnt/a:
$ mount /mnt/a
$ mount /mnt/cdrom
```

`/mnt/a`, `/mnt/a:`, and `/mnt/cdrom` can now be accessed by every user. I've found that to write on `/mnt/a` without being root, right after preparing the floppy it's necessary to do:

```
# mount /mnt/a
# chmod 777 /mnt/a
# umount /mnt/a
```

Remember that allowing anyone to mount disks this way is a gaping security hole, if you care.

4.2 Backing Up

Now that you know how to handle floppies etc., a couple of lines to see how to do your backup. There are several packages to help you, but the very least you can do for a multi-volume backup is (as root):

```
# tar -M -cvf /dev/fd0H1440 /dir_to_backup
```

Make sure to have a formatted floppy in the drive, and several ready. To restore your stuff, insert the first floppy in the drive and do:

```
# tar -M -xpvf /dev/fd0H1440
```

5 Tayloring the System

5.1 System Initialization Files

Two important files under DOS are `AUTOEXEC.BAT` and `CONFIG.SYS`, which are used at boot time to initialise the system, set some environment variables like PATH and FILES, and possibly launch a program or batch file. Under Linux there are several initialisation files, some of which you had better not tamper with until you know exactly what you are doing. I'll tell you what the most important are, anyway:

```
FILES                           NOTES

/etc/inittab                    don't touch for now!
/etc/rc.d/*                     ditto
```

If all you need is setting the PATH and other environment variables, or you want to change the login messages or automatically launch a program after the login, have a look at the following files:

```
FILES                           NOTES

/etc/issue                      sets pre-login message
/etc/motd                       sets post-login message
/etc/profile                    sets PATH and other variables, etc.
/etc/bashrc                     sets aliases and functions, etc. (see below)
/home/your_home/.bashrc         sets your aliases + functions
/home/your_home/.bash_profile   sets environment + starts your progs
/home/your_home/.profile        ditto
```

If the latter file exists (note that it is a hidden file), it will be read after the login and the commands in it will be executed.

Example—look at this `.profile`:

```
# I am a comment
echo Environment:
printenv | less   # equivalent of command SET under DOS
alias d='ls -l'   # easy to understand what an alias is
alias up='cd ..'
echo "I remind you that the path is "$PATH
echo "Today is `date`"  # use the output of command 'date'
echo "Have a good day, "$LOGNAME
# The following is a "shell function"
ctgz() # List the contents of a .tar.gz archive.
{
  for file in $*
  do
    gzip -dc ${file} | tar tf -
  done
}
# end of .profile
```

PATH and LOGNAME, you guessed right, are environment variables. There are many others to play with; for instance, RMP for apps like `less`.

5.2 Program Initialization Files

Under Linux, virtually everything can be tailored to your needs. Most programs have one or more initialization files you can fiddle with, often as a `.prognamerc` in your home dir. The first ones you'll want to modify are:

`.inputrc`: used by `bash` to define keybindings.

`.xinitrc`: used by `startx` to initialize X Window System.

`.fvwmrc`: used by the window manager `fvwm`. A sample is in: `/usr/lib/X11/fvwm/system.fvwmrc`

`.Xdefault`: used by `rxvt`, a terminal emulator for X, and other programs.

For all of these and the others you'll come across sooner or later, RMP.

6 A Bit of Programming

6.1 Shell Scripts: .BAT Files on Steroids

If you used .BAT files to create shortcuts of long command lines (I did a lot), this goal can be attained by inserting appropriate alias lines (see example above) in `profile` or `.profile`. But if your .BATs were more complicated, then you'll love the scripting language made available by the shell: it's as powerful as QBasic, if not more. It has variables, structures like while, for, case, if... then... else, and lots of other features: it can be a good alternative to a "real" programming language.

To write a script—the equivalent of a .BAT file under DOS—all you have to do is write a standard ASCII file containing the instructions, save it, then make it executable with the command `chgmod +x <scriptfile>`. To execute it, type its name.

A word of warning. The system editor is called `vi`, and in my experience most new users find it very difficult to use. I'm not going to explain how to use it, because I don't like it and don't use it, so there. See Matt Welsh's "Linux installation...", pag. 109. (You had better get hold of another editor like `joe` or `emacs` for X.) Suffice it here to say that:

- to insert some text, type 'i' then your text;
- to quit `vi` whithout saving, type <ESC> then :q!
- to save and quit, type <ESC> then :wq

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 an example of shell script, from which you can extract some basic rules:

```
#!/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 look the difference: "hi $your_name" # quoting with "
echo look 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`  # 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."
```

6.2 C for Yourself

Under Unix, the system language is C, love it or hate it. Scores of other languages (FORTRAN, Pascal, Lisp, Basic, Perl, awk...) are also available.

Taken for granted that you know C, here are a couple of guidelines for those of you who have been spoilt by Turbo C++ or one of its DOS brothers. Linux's C compiler is called `gcc` and lacks all the bells and whistles that usually accompany its DOS counterparts: no IDE, on-line help, integrated debugger, etc. It's just a rough command-line compiler, very powerful and efficient. To compile your standard `hello.c` you'll do:

```
$ gcc hello.c
```

which will create an executable file called `a.out`. To give the executable a different name, do

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

To link a library against a program, add the switch -l<libname>. For example, to link the math library:

```
$ gcc -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`).

So far, so good. But when your prog is made of several source files, you'll need to use the utility `make`. Let's suppose you have written an expression parser: its source file is called `parser.c` and #includes two header files, `parser.h` and `xy.h`. Then you want to use the routines in `parser.c` in a program, say, `calc.c`, which in turn #includes `parser.h`. What a mess! What do you have to do to compile `calc.c`?

You'll have to write a so-called `makefile`, which teaches the compiler the dependencies between sources and objects files. In our example:

```
# This is makefile, used to compile calc.c
# Press the <TAB> key at appropriate positions!

calc: calc.o parser.o
<TAB>gcc -o calc calc.o parser.o -lm
# calc depends on two object files: calc.o and parser.o

calc.o: calc.c parser.h
<TAB>gcc -c calc.c
# calc.o depends on two source files

parser.o:  parser.c parser.h xy.h
<TAB>gcc -c parser.c
# parser.o depends on three source files

# end of makefile.
```

Save this file as `makefile` and type

```
$ make
```

to compile your program; alternatively, save it as `calc.mak` and type

```
$ make -f calc.mak
```

And of course, RMP.

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 textmode effects, and **svgalib**, to do graphics. If you feel brave enough to tackle X programming, get XForms (bloch.phys.uwm.edu/pub/xforms) and/or MGUI (www.volftp.vol.it/IT/IT/ITALIANI/MORELLO/index.htm), two terrific libraries that make X programming easy. Moreover, if you can't live without an IDE a la Borland, get the package **xwpe** from sunsite.unc.edu:/pub/Linux/apps/editors/. Chances are you'll like it.

7 The Remaining 1%

7.1 Making Virtual Memory

Although Linux can in theory run with only 2 megs of RAM, the more you have, the more you can do. X Window System won't run unless you have at least 8 megs. To create an additional 8 megs of virtual memory, type as root:

```
# dd if=/dev/zero of=/swapfile bs=1024 count=8192
# mkswap /swapfile 8192
# sync
# swapon /swapfile
```

Add the last line in /etc/rc.d/rc.local to make the swapfile available the next time you boot, or add this line in /etc/fstab:

```
/swapfile   swap   swap   defaults
```

7.2 Using tar & gzip

Under Unix there are some widely used applications to archive and compress files. **tar** is used to make archives—it's like PKZIP but it doesn't compress, it only archives. 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
```

You can compress files 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 can compress only one file at a time. To decompress, use:

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

RMP.

The `unarj`, `zip` and `unzip` (PK??ZIP compatible) utilities are also available. Files with extension `.tar.gz` or `.tgz` (archived with `tar`, then compressed with `gzip`) are as common in the Unix world as .ZIP files are under DOS. Here's how to list the contents of a `.tar.gz` archive:

```
$ gzip -dc <file.tar.gz> | tar tf - | less
```

7.3 Installing Applications

First of all: installing packages is root's work. Some Linux applications are distributed as `.tar.gz` or `.tgz` archives, specifically prepared so that they can be decompressed from / typing the following command:

```
# gzip -dc <file.tar.gz> | tar xvf -
```

The files will be decompressed in the right directory, which will be created on the fly. Users of the Slackware distribution have a user-friendly `pkgtool` program; another is `rpm`, which is available on all distributions thanks to Red Hat.

Other packages shouldn't be installed from /; typically, the archive will contain a directory called `pkgname/` and a lot of files and/or subdirectories under `pkgname/`. A good rule is to install those packages from `/usr/local`. Besides, some packages are distributed as C or C++ source files, which you'll have to compile to create the binaries. In most cases, all you have to do is issue `make`. Obviously, you'll need the `gcc` compiler.

7.4 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 `gcc this_is_a_long_name.c`; typing in `gcc thi<TAB>` will suffice. (If you have other files that start with the same characters, supply enough characters to resolve any ambiguity.)

Backscrolling: pressing SHIFT + PAG UP (the grey key) allows you to backscroll a few pages, depending on how much video memory you have.

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. There is also `xclipboard` (alas, only for text); don't get confused by its very slow response.

Using the mouse: 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.

Messages from the kernel: have a look at `/var/adm/messages` or `/var/log/messages` as root to see what the kernel has to tell you, including bootup messages.

7.5 Useful Programs and Commands

This list reflects my personal preferences and needs, of course. First of all, where to find them. Since you all know how to surf the Net and how to use `archie` and `ftp`, I'll just give you three of the most important addresses for Linux: `sunsite.unc.edu`, `tsx-11.mit.edu`, and `nic.funet.fi`. Please use your nearest mirror.

- `at` allows you to run programs at a specified date and time;
- `awk` is a simple yet powerful language to manipulate data files (and not only). For example, being `data.dat` your multifield data file,

  ```
  $ awk '$2 ~ "abc" {print $1, "\t", $4}' data.dat
  ```

  prints out fields 1 and 4 of every line in `data.dat` whose second field contains "abc".
- `delete-undelete` do what their name means;
- `df` gives you info about the mounted disk(s);
- `dosemu` allows you to run several (not all) DOS programs—including Windows 3.x, with a bit of hacking;
- `file` <`filename`> tells you what `filename` is (ASCII text, executable, archive, etc.);
- `find` (see also Section 3.3) is one of the most powerful and useful commands. It's used to find files that match several characteristics and perform actions on them. General use of `find` is:

  ```
  $ find <directory> <expression>
  ```

  where <expression> includes search criteria and actions. Examples:

  ```
  $ find . -type l -exec ls -l {} \;
  ```

  finds all the files that are symbolic links and shows what they point to.

  ```
  $ find / -name "*.old" -ok rm {} \;
  ```

  finds all the files matching the pattern and deletes them, asking for your permission first.

  ```
  $ find . -perm +111
  ```

  finds all the files whose permissions match 111 (executable).

  ```
  $ find . -user root
  ```

finds all the files that belong to root. Lots of possibilities here—RMP.

- `gnuplot` is a brilliant program for scientific plotting;
- `grep` finds text patterns in files. For example,

```
$ grep -l "geology" *.tex
```

  lists the files *.tex that contain the word "geology". The variant `zgrep` works on gzipped files. RMP;
- `gzexe` compresses executable binaries keeping them executable;
- `joe` is an excellent editor. Invoking it by typing `jstar` you'll get the same key bindings as WordStar and its offspring, including DOS and Borland's Turbo languages editors;
- `less` is probably the best text browser, and if properly configured lets you browse gzipped, tarred, and zipped files as well;
- `lpr` <file> prints a file in background. To check the status of the printing queue, use `lpq`; to remove a file from the printing queue, use `lprm`;
- `mc` is a great file manager;
- `pine` is a nice e-mailing program;
- `script <script_file>` copies to `script_file` what appears on screen until you issue the command `exit`. Useful for debugging;
- `sudo` allows users to perform some of root's tasks (e.g. formatting and mounting disks; RMP);
- `uname -a` gives you info about your system;
- `zcat` and `zless` are useful for viewing gzipped text files without ungzipping them. Possible use:

```
$ zless textfile.gz
$ zcat textfile.gz | lpr
```

- The following commands often come in handy: `bc, cal, chsh, cmp, cut, fmt, head, hexdump, nl, passwd, printf, sort, split, strings, tac, tail, tee, touch, uniq, w, wall, wc, whereis, write, xargs, znew.` RMP.

7.6 Common Extensions and Related Programs

You may come across scores of file extensions. Excluding the more exotic ones (i.e. fonts, etc.), here's a list of who's what:

- `.1 ... .8`: man pages. Get `man`.
- `.arj`: archive made with `arj`. `unarj` to unpack.
- `.dvi`: output file produced by TeX (see below). `xdvi` to visualize it; `dvips` to turn it into a postscript .ps file.

- `.gif`: graphic file. Get `seejpeg` or `xpaint`.
- `.gz`: archive made with `gzip`.
- `.info`: info file (sort of alternative to man pages). Get `info`.
- `.jpg, .jpeg`: graphic file. Get `seejpeg`.
- `.lsm`: Linux Software Map file. It's a plain ASCII file containing the description of a package.
- `.ps`: postscript file. To visualize or print it get `gs` and, optionally, `ghostview`.
- `.tgz, .tar.gz`: archive made with `tar` and compressed with `gzip`.
- `.tex`: text file to submit to TeX, a powerful typesetting program. Get the package `tex`, available in many distributions; but beware of NTeX, which has corrupted fonts and is part of Slackware until version 96.
- `.texi`: texinfo file (cp. `.info`). Get `texinfo`.
- `.xbm, .xpm, .xwd`: graphic file. Get `xpaint`.
- `.Z`: archive made with `compress`.
- `.zip`: archive made with `zip`. Get `zip` and `unzip`.

8 The End, for Now

Congratulations! You have now grasped a little bit of Unix and are ready to start working. Remember that your knowledge of the system is still limited, and that you are expected to do more practice with Linux to use it comfortably. But if all you had to do was get a bunch of applications and start working with them, I bet that what I included here is enough.

I'm sure you'll enjoy using Linux and will keep learning more about it—everybody does. I bet, too, that you'll never go back to DOS! I hope I made myself understood and did a good service to my 3 or 4 readers.

8.1 Copyright

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the HOWTO documents, and would like to be notified of any plans to redistribute the HOWTOs.

If you have questions, please contact Greg Hankins, the Linux HOWTO coordinator, at gregh@sunsite.unc.edu via email.

8.2 Disclaimer

"From DOS to Linux HOWTO" was written by Guido Gonzato, `guido@ibogfs.df.unibo.it`. Many thanks to Matt Welsh, the author of "Linux Installation and Getting Started", to Ian Jackson, the author of "Linux frequently asked questions with answers", to Giuseppe Zanetti, the author of "Linux", to all the folks who emailed me suggestions, and especially to Linus Torvalds and GNU who gave us Linux.

This document is provided "as is". I put great effort into writing it as accurately as I could, but you use the information contained in it at your own risk. In no event shall I be liable for any damages resulting from the use of this work.

Feedback is welcome. For any requests, suggestions, flames, etc., feel free to contact me.

Enjoy Linux and life,

Guido `=8-)`