A Programmers's Guide to Linux

Timothy A. Gonsalves
TeNeT Group
Dept of Computer Science & Engineering
Indian Institute of Technology, Madras - 600 036
E-mail: tag@ooty.TeNeT.res.in

July 1999

Contents

1 Introduction

Linux is today the most widely-used variant of the Unix operating system. This guide introduces the most common features of Linux for a programmer. Much of the material also applies to most other Unix systems.

Most of the commands described have many more options and features than are covered in this guide. See section 2.2 for further details.

1.1 Typographical Conventions

We use the following conventions in this guide:

`emacs` the name of a specific command or file

file you should replace *file* with a specific name

Thu Jul 22 23:19:11 IST 1999 output that you see on the screen

`% date` at the Linux prompt, which we assume to be `%`, type the command `date` and press <ENTER>.

`Ctrl-x` hold down the Control key and press “x”

2 Getting Started

2.1 Logging in and out

To login, type your *username* at the `Login:` prompt, and your password at the `Password:` prompt. Note that your password is not displayed.

finger lists all the users who are currently logged in.

finger *user* gives information about *user* such as whether she is currently logged in and when she has last read her mail.

logout terminate your session.

passwd change your password.

Your password is the only key to the safety and privacy of your files. Choose a password that is easy for you to remember but that is non-obvious. *Keep it secret and change it periodically.* If you forget your password, get help from a system administrator.

2.2 Getting Help

man -k *word* displays an index of all manual pages that contain *word* in the title. If the output fills the screen, press <SPACE> for the next screen.

`man` ***name*** displays the manual page for *name*

`man` ***n name*** displays the manual page for *name* in section *n* of the Linux manual

command `--help` most commands when followed by `--help` give brief information on their usage. Try `less --help`. If the help scrolls off the top of the screen, type:

```
command --help | less
```

The Linux manual is organised into sections 1–8 as follows: section 1: commands, section 2: system calls, section 3: functions, section 4: special files, section 5: file formats and protocols, section 6: games, section 7: miscellaneous information, section 8: administration commands.

The man command looks for the given *word* in the sections in order. The man page for the first section in which *word* appears is displayed. If you want the man page from a later section, you must specify the section number also. For example, man chmod displays the man page for the chmod command from section 1, while man 2 chmod displays the man page for the chmod system call from section 2.

3 Files

A Linux file is simply a sequence of bytes. Files are central to Linux. You create files for your programs, data and documents. Every command is in a file. Even devices such as the keyboard, display, mouse and Ethernet LAN appear as special files. It is even possible to access the internals of the Linux kernel as a file!

Linux allows you to create directories for related files. A directory can contain sub-directories, resulting in a tree-structured hierarchy. All directories are part of a single tree. The *root* of the tree is "/". Your files are in your *home directory*, usually with a name such as /home/*servername*/*yourname*.

Each file has a name that is unique in its directory. The name is case-sensitive (Makefile and makefile are two distinct files), can contain a variety of special characters and can be quite long. a.out and A_Long_#%@!Name.txt.bak.gz are both valid names.

3.1 Handling Files

mv *src dest* move and/or rename files

cp *src dest* copies file *src* to *dest*.

less *file(s)* displays a text file one screen at a time. Press <SPACE> to see the next page.

cat *file(s)* displays the contents of *file(s)*

To see a list of the files contained in the current directory, type ls. For details such as the file size and date of last access, type ls -l. ls *.c lists all C source files. ls programs lists the contents of the sub-directory programs. Normally, ls does not list *hidden* files whose names start with ".". The -a option will also list such files.

mv *oldname newname* renames the file *oldname* to *newname*. To move files f1 and f2 to the sub-directory savedir, type:

```
% mv f1 f2 savedir
```

Note that mv can be used to move files between directories other than the current working directory. For example, mv ~/experimental/sort.c ~/programs/fastsort.c renames sort.c to fastsort.c and moves it from your exerimental directory to your *programs* directory.

The command less is a powerful version of more. For example, less allows you to scroll backwards and forwards, while more allows only forward scrolling. The latter is available on all Unix systems.

cat is the equivalent of the MS-DOS type command. *Caution:* if you cat a binary file, the display may get garbled or even hung. To recover, try the commands:

reset or stty sane

`cat` can be used to *concatenate* several files into one:

```
% cat part1.txt part2.txt part3.txt >whole.txt
```

Wildcards: Many Files at Once The use of *wildcards* allows a single command to process many files at once. Linux has a set of simple wildcards that allow one to select various subsets of file.[1]

***** matches any string of 0 or more character in the file name

? matches any single character

[abc...] matches any one of the characters in the '[]'

{n1,n2,n3,...} matches any one of the comma-separated names in the '{}'

Examples The `echo` command displays its arguments on the screen. For each of the commands below, typical output is shown to the right. Except for the last example, the files are only from in the current working directory.

`echo *.c` all C source files:
... myio.c myutils.c prog.c ttydriver.c ...

`echo *.[ch]` all C source and header files in the current directory:
myio.c myio.h myutils.c myutils.h prog.c ttydriver.c ...

`echo [A-Z]*` all files in the current directory starting with a capital letter:
Makefile Readme

`echo my{io,utils}.c` all C source files starting with the prefix `my`:
myio.c myutils.c

echo myutils.? all files named `myutils` with a single-character extension:
myutils.c myutils.h myutils.o ...

`echo {src,doc}/myutils.{[ch],txt}` all files named `myutils` with the extensions `.c, .h, .txt` in the subdirectories `src` and `doc`:
doc/myutils.txt src/myutils.c src/myutils.h

3.2 Program Output and Files

Many programs by default read from the keyboard and write to the display. You can easily change this without modifying the programs, using what is referred to as *I/O redirection.* This facility allows you to easily do complex tasks using several simple commands.

Output to a file '>': the command `ls >files.list` runs ls to obtain a listing of the files in the current directory. The listing is written into the file `files.list` rather than to the display. To append to an existing file, use '`>>`' instead. The following two commands will write the current date and time and the list of logged on users into the file `user.list`.

```
% date >user.list
% finger >>user.list
```

[1] Wildcards are actually provided by the shell (section 4). Despite some similarity, these wildcards are quite different from the *regular expressions* used by programs such as *grep*.

Output to another program “|”: if two commands are separated by a “|”, the output of the first is fed as input to the second. For example,

```
% ls -lR ~ | less
```

will display a detailed listing of all your files, one screen at a time. Such *piping* can be done between several commands on one line.

Input from a file “<”: `prog <infile` will run `prog` reading input from `infile` instead of from the keyboard.

Error output: when you use “>” or “>>” you may still find some error messages appearing on the display. To redirect these also to a file, type: `prog >& outfile`, `prog >>& outfile` or `prog |& prog2`.

3.3 Text Files

The most often used tool for text files is a text editor. Linux also has other commands for processing text files.

emacs a powerful text editor

joe an easy-to-use editor, similar to PC editors such as WordStar and Borland’s Turbo languages editor

vi one of the earliest full-screen editors for Unix

wc counts the characters, words and lines in one or more files

sort sorts lines of text in ascending or descending order

grep find all lines that contain a given pattern

Linux has many text editors. The editor of choice for programmers is *GNU Emacs*, a powerful customisable, extensible editor. To get started, run `emacs` and then type `Ctrl-h t` (i.e., press Control-h followed by ’t’). This will put you in an online tutorial where you can learn the basic Emacs commands. For more details, see the Emacs Info `Ctrl-h i`, the *Emacs Reference Manual* and *The Emacs Tutorial.*

Joe is similar to user-friendly PC editors, and provides basic editing features.

Vi is older editor that used to be popular on early versions of Unix. Some application programs, such as mail readers, still use vi as the default editor. If you find yourself unexpectedly put into vi, type `:q!` to quit.

To set your preferred editor to, say Emacs, set the environment variables EDITOR and VISUAL to `/usr/bin/emacs` (see section 4):

```
% setenv EDITOR /usr/bin/emacs
% setenv VISUAL /usr/bin/emacs
```

or

```
% EDITOR=/usr/bin/emacs; export EDITOR
% VISUAL=/usr/bin/emacs; export VISUAL
```

The command `grep` is used to find lines of text that contain *patterns.* The simplest pattern is a string. For example, to see all the lines in which the function `SomeFunc()` is used, type:

```
% grep SomeFunc *.c
```

If a function declaration starts at the left margin, and fits on one line, and does not have a comment after the “`)`”, then the following command lists the definition of `SomeFunc()`:

```
% grep '^[a-zA-Z].*SomeFunc(.*)$' *.c
```

To see all function definitions, type:

```
grep '^[a-zA-Z].*(.*)$' *.c
```

Note: the “”’ around the search pattern are required so that the shell does not take them to be wildcards.

3.4 Organizing Files – Directories

As with file names, directory names are case-sensitive and can be very long. The special name “~” refers to your home directory. `~username` refers to username’s home directory. A file name that starts with a “/” is an *absolute* name starting at the root of the tree. A name that does not start with “/” is relative to the current working directory. The name “`.`” refers to the current directory.

mkdir *dirname* make a directory *dirname*, which can be either absolute or relative.

rmdir *dirname* remove the directory *dirname*. You must own the directory and it must be empty.

cd *dirname* change the current working directory to *dirname*.

pwd print your current working directory.

mv *srcdir destdir* move a sub-directory from one place to another.

du *dirname* summarizes the disk space usage of all files in *dirname* and all its sub-directories.

ls lists files and sub-directories

ln -s make a symbolic link to another directory or file

find finds files matching certain criteria in a sub-tree of directories

`cd ..` changes to the parent of the current directory. `cd` by itself changes to your home directory. When `mv` is used to move a directory, it also moves all sub-directories contained in that directory.

If the current directory contains the file prog.c and the sub-directory `programs`, then `ls p*` will list `prog.c programs` and also all the contents of `programs`. If you do not want to see the contents of sub-directories, use the tt -d option.

Organizing your files in many sub-directories has one disadvantage – it may sometimes be difficult to remember in which sub-directory a particular file resides. Linux provides the `ln` and `find` commands to help in such cases.

While writing a network program, you may need to refer to the file `/usr/include/netinet/ip.h`. Rather than remember and type its long path name, you can make a symbolic link to it in your working directory:

```
% cd ~/programs/src
% ln -s /usr/include/netinet/ip.h .
% ls -l ip.h
lrwxrwxrwx 1 tag users 25 Jul 23 18:50 ip.h ->/sl /usr/include/netinet/ip.h
```

The command *find* is useful, for example, to list the names of all C source files in the sub-tree of the current directory:

```
% find .  -name .c
./admin/scripts/fork.c
./prog/c/test.c
./prog/c/CygToken.h
./util/xlstat/src/header.h
./util/xlstat/src/eth.c
./util/xlstat/src/userif.c
./tenet/ushacom/urlib.h
./tenet/ushacom/urlib.c
```

You can have `find` execute a command on each file that it finds:

```
%~find . -name \*.c -exec grep -l ’main(.*)’ \{\} \;
./prog/c/test.c
./util/xlstat/src/userif.c
```

This lists only those files that contain a *main()* function (with or without arguments).

Since the syntax of *find* is rather painful, it is convenient to define *aliases* (see section 4.2 for an example).

3.5 Permissions

In a multi-user OS, users can share files. Linux allows you to control who can access your files and how. Each file and directory has an *owner* and a *group*. The owner can set the file permissions separately for the owner, group and others.

In the output of `ls -l` below, `tag` is the owner of all three files and `users` is the group.

```
-rw-r--r--   1 tag      users          21 Apr 17 14:23 data
-rwxrwxr-x   1 tag      users       34525 Apr 17 15:34 ptest
drwx------   3 tag      users        1024 Jul 24 10:29 secret
drwxr-xr-x   3 tag      users        1024 Jul 24 10:29 src
lrwxrwxrwx   1 tag      users          26 Jul 23 18:14 sys -> /usr/include/sys
```

The first character on each line indicates the type of file: `-` for an ordinary file, `d` for a directory and `l` for a symbolic link. The next 9 characters consist of 3 groups of 3 characters. These groups give the access permissions for the owner, group and world, respectively. In each group, the characters are interpreted as follows:

1. `r` for read access, `-` for no read access
2. `w` for write access, `-` for no write access
3. `x` for execute access, `-` for no execute access

In the case of a directory, “execute” means the permission to list the contents of the directory.

Thus, in the example above, the file `data` can be read by anyone, but only written by the owner `tag`. The program `ptest` can be read, written and executed by `tag` and the group `users`, but everyone else can only read and execute it. The directory `secret` can be accessed only by its owner.

To change the permissions of a file or directory, use the **chmod** (“change mode”) command.

chmod u+x *file*   adds execute permission for the user (i.e. owner)

chmod a+x *file*   adds execute permission for all

chmod a-w *file* makes the file read-only

chmod go-rwx *file* removes all access for group and others

chmod g+w *file* gives write permission to group members

4 Shells

When you login, a *shell* is started up for user-interaction. It reads your command line and executes the commands on your behalf. Linux offers a number of shells, the most commonly-used being *bash* (Bourne-again shell) and *csh* (a shell with C-like syntax).

You can select your shell using the `chsh` command. Type `ls /bin/*sh` to see a list of available shells. Note that `csh` may be known by the name `tcsh`.

Some of the useful features of `bash` and `csh` are described below. The discussion applies to both shells, unless specifically noted.

4.1 Typing Aids

To reduce your typing, the shells provide some aids in addition to the wildcards described earlier in section 3.1.

<TAB> attempts to complete the command or filename that you've typed. Example, if the current directory contains the file `Makefile`:

```
% echo Mak<TAB>
```

will result in the completion

```
% echo Makefile
```

`<Ctrl-d>` displays all possible completions of the prefix that you've typed. Use this if the shell beeps when you type <TAB> after an ambiguous prefix. Example:

```
% echo myutils<Ctrl-d>
myutils.c myutils.h
% echo myutils
```

⇑ or `<Ctrl-p>` re-types the previous command, which you can then edit using the arrow keys or the Emacs cursor-movement keys before pressing <ENTER>. Repeatedly pressing ⇑ will get you through the history of your previous commands.

`!gcc` reexecutes the most recent command starting with the prefix `gcc`

`!!` reexecutes the last command

`history` displays your command history

¡Ctrl-r¿ in Bash, searches in reverse through your command history for a string. For example, `<Ctrl-r>incl` will retrieve the most recent command with the string "incl". You can then edit the command and re-execute it by pressing <ENTER>. Press `<Ctrl-r>` repeatedly to find earlier commands with the same string.

4.2 Command Aliases

You can define aliases for frequently-used commands, or to make Linux appear like some other operating system such as MS-DOS or VMS. To see the expansion of an alias, say `dir`, type the command: `alias dir`. To see all defined aliases type: `alias`.

Bash

`alias dir='ls -lF'` behaves somewhat like the MS-DOS `dir`

`alias gcc='gcc -g'` always use the `-g` option with `gcc`

`function findfile() { find $1( -name $2) -print` find files, for example: `findfile .  .c`

`function findexec() { find $1( -name $2) -exec $3{};` find files and execute a command on each, for example:
`findexec . \*.c 'grep -l main(.*)'`

C Shell

`alias dir 'ls -lF'` behaves somewhat like the MS-DOS `dir`

`alias gcc 'gcc -g'` always use the `-g` option with `gcc`

alias findfile 'find!:1(-name!:2*) -print' find files, for example: `findfile .  .c`

alias findexec 'find!:1(-name!:2) -exec!:3*{};' find files and execute a command on each, for example:
`findexec . \*.c grep -l 'main(.*)'`

4.3 Shell Variables

The shell has a number of *environment variables* that control the way the shell and other programs behave, and that describe some aspects of the user, the machine, and the operating system. By convention, these variables have upper-case names. To see the value of a variable such as OS, type: `echo $OS` which should result in *linux*. To see the values of all variables, type: `env | less`.

An important environment variable is `PATH`. This is a colon-separated list of directories in which the shell searches for every command that you issue. This variable should start with '.:', otherwise you will not be able to conveniently run programs that you compile. For example:

```
% echo $PATH
.:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/hosts
```

Another important variable is `TERM` which describes the type of terminal that Linux thinks you are using. If it is not correctly set, screen-oriented programs such as Emacs and Pine will not work properly. It is usually set to `linux` for the console. For many other terminals, `vt100` will work.

There may be several different versions of the same command in your PATH and you may also have defined an alias with the same name. Type:

```
% which command
```

to find out which of these versions will be executed when you run `command`.

Bash To set an environment variable and then display its value:

```
% XYZ='a new variable'; export XYZ
% echo $XYZ a new variable
```

C Shell To set an environment variable and then display its value:

```
% setenv XYZ 'a new variable'
% echo $XYZ
a new variable
```

4.4 Customization

When you customize the shell by defining aliases or setting environment variables, you would usually like this to take effect every time you login. This can be done by putting you customizations into the files that the shell reads when it starts up.

Bash On login, bash first reads `/etc/profile`, then reads the *first one* of the following three files that exists: `~/.bash_profile, ~/.bash_login, ~/.profile`.

If you start Bash at the command line by typing `bash`, it only reads `~/.bashrc`.

On logout, bash executes commands from `~/.bash_logout`.

C Shell On login, the C Shell first reads `/etc/csh.cshrc` and `/etc/csh.login`. It then executes commands from `~/.cshrc`, `~/.history` and `~/.login` in order.

If you start the C shell at the command line by typing `csh`, it reads `/etc/csh.cshrc` and `~/.cshrc` only.

However, if the C Shell is really `tcsh`, then if `~/.tcshrc` exists, it is read instead of `~/.cshrc`. Only if `~/.tcshrc` does not exist is `~/.cshrc` read.[2]

On logout, the C shell executes commands from `/etc/csh.logout` and `~/.logout`. It saves your command history in `~/.history`.

4.5 Graphical User Interface

Linux supports the X-Windows graphical user interface, with icons, menus, mouse and multiple over-lapping windows. Programs such as Emacs run under X-Windows also. The Netscape browser requires X-Windows to run on Linux.

Typically, X-Windows is run on the console. Type `startx` at the shell prompt to start X-Windows.

5 Several Tasks at Once

Linux is a *multi-tasking* operating system. It can run several programs concurrently. Each such program or task is referred to as a *process*.

Piping whenever you pipe the the output of one command into another command, say `man -k file | less`, Linux runs the commands concurrently.

[2] Type `ls -l /bin/csh`. You are really using `tcsh` if the output is:
lrwxrwxrwx 1 root root 4 Jun 4 21:50 /bin/csh ->tcsh.

Background tasks if you end a command line with "&", the shell runs the command in the background and immediately prompts you for the next command. It is useful to redirect the output into a file. For example, to run a compilation of a large file in the background:

```
% gcc -g -o large large.c > gcc.log &
```

Suspending a task while a program is running, you can suspend it by typing `<Ctrl-z>`. To resume the suspended task, type `fg`. To continue the task running in the background, type `bg`. Note that a suspended task does not execute until put in the background or brought to the foreground.

jobs lists your suspended and background jobs. For example:

```
[1]  - Running                      ftp lantana
[2]  + Suspended                    emacs
[3]    Running                      gcc -g -c myprog.c
[4]    Suspended                    pine
```

The file transfer program `ftp` and a C compilation are running while Emacs and Pine are suspended. `fg` by itself will resume the job marked with a "+", i.e., Emacs. To resume Pine, type `fg %4` or `fg %p` (any unambiguous prefix).

ps lists your processes. Useful options are `-u` for a more user-oriented output, `-l` for a detailed listing, and `-x` to see all processes.

kill -9 *%gcc* terminates a job, `gcc`, that is suspended or running in the background. Do not kill interactive jobs such as Emacs as you may lose files that are being edited. Rather, bring it to the foreground and exit.

kill -9 3856 terminates the process with pid 3856 (obtained from the `ps` command).

nice if you start up many processes concurrently, you could make the system respond slowly to other users. The `nice` command runs a program at lower priority.

```
% nice gcc ...
```

runs the compilation at priority 4, which is lower than the default priority of 0.

```
% nice +19 sim ...
```

runs a long simulation at the lowest priority, 19. Note that priority in Linux is relative — while other users are thinking, your lower priority jobs will run. Priority is also dynamic — as a process uses the CPU, its priority gets automatically lowered to give others a chance to run.

6 Program Development

Linux supports a number of programming languages and other software development tools.

gcc the GNU C compiler

g++ the GNU C++ compiler

java part of the Java Development Kit (jdk)

perl a language that is useful for system administration, processing of text files and networking

gdb the GNU symbolic debugger, supports C and C++. Use the `-g` compiler option before running `gdb`

make a utility for building and managing projects consisting of many source code files

rcs a revision control system for saving multiple versions of a file. Good for personal use.

cvs a revision control system that is suitable for large projects involving many programmers

Revision Control During development, a program typically goes through many revisions. Sometimes, it is necessary to revert to an earlier revision. A revision control system such as RCS or CVS helps a programmer to systematically save revisions of one or more files.

RCS is quite easy for a single programmer to use.

1. Assume that all your files, `*.c *.h`, are in one directory. In that directory, make a sub-directory RCS.
2. Create your source files
3. When you have made some changes and want to save a revision of a file, say `myprog.c`, check it into RCS:

   ```
   % ci myprog.c
   ```

   You will be asked to type in a brief description of the file, and it will be checked in with the initial revision number 1.1. The file has disappeared from the current directory, and been saved in RCS with "`,v`" appended.
4. Type `rlog myprog.c` to see the status of your file
5. To continue editing the file, you need to check it out:

   ```
   % co -l myprog.c
   ```

   The file `myprog.c` will re-appear in your current directory.
6. After further editing, check it in again with `ci`. This time you will be asked for a *log message*, and the file will be assigned a new revision number of 1.2.
7. Repeat the check-out – edit – check-in cycle as many times as you wish. At any time `rlog` shows you the details of all revisions of the file.
8. To retrieve an earlier revision, say 1.1:

   ```
   % co -l1.1 myprog.c
   ```
9. See the RCS man pages or info pages for further details

7 Networking

Networking is an integral part of Linux. Indeed, Linux is one of the first choices as the operating systems for network servers in the Internet.

telnet, rlogin login to a remote machine

ftp transfer files to/from a remote machine

rcp copy files to/from a remote machine (similar to `cp`)

rsh execute commands on a remote machine without logging into it

pine read and send e-mail

ping check if the network connection to another machine is okay

lynx a fast, text-based Web broswer

netscape a graphical Web Broswer

NFS (Network File System) allows directories on one machine to be mounted in the directory hierarchy of another machine and treated as though they were on a local disk

8 References

1. B.W. Kernighan & R. Pike, *The* UNIX *Programming Environment*, Prentice-Hall, 84.
2. R.J. Stallman, *GNU Emacs Manual*, Free Software Foundation.
3. T.A. Gonsalves, *A Guide to Emacs*, DONlab, Dept. of Computer Science & Engg., I.I.T., Madras, July 96.
4. T.A. Gonsalves, *A Guide to the gdb Debugger*, TeNeT Group, Dept. of Computer Science & Engg., I.I.T., Madras, July 96.