UNIX Command Line HOWTO

Howard Gibson hgibson@eol.ca

Version 0.9 – 2022/01/15

Abstract

This is the UNIX Command Line HOWTO. This document is intended for new GNU/Linux users, although most of it should be useful for anyone starting to use any form of UNIX. This document provides an explanation of the UNIX keyboard commands that are most likely to be useful to general users.

Contents

1 Introduction

1.1 Scope

UNIX was developed long before Graphical User Interfaces (GUIs), so it was designed to work from a keyboard. There are now UNIX GUI tools that let you click with a mouse, but these do not give you the complete functionality of the command line, and they are not available if the GUI is not working. You really ought to learn some UNIX.

1.2 Document Structure

John Barrymore, when asked why he needed cue cards in a movie studio when he possessed perfect recall elsewhere, replied...

> My memory is full of beauty: Hamlet's soliloquies, Queen Mab's speech, the Song of Solomon. Do you expect me to clutter up all of that with this horse shit?

There is no need to memorize the stuff in this article unless you plan to become a UNIX administrator. This document is structured as a task-by-task reference. You should read through the section on UNIX, which explains pathnames and the command structure. You probably should read the section on accessing the UNIX shell. After that, you can look up your task in the table of contents, and just read that section.

All of the commands described here are rigorously documented by help files, as described below under Getting Help. The descriptions in this article are simplified to assist beginners, so lots of information has been left out. If you are doing something weird or difficult, or you just want more information, you should read the documentation.

1.3 Copyright

This document is copyright © 1999 and 2022, by Howard Gibson. You may copy this document onto bulletin boards, web pages and other computer media, as long as the file is unaltered, and the distribution is not-for-profit. All other rights are reserved.

2 UNIX

2.1 UNIX Pathnames

UNIX files and directories are called up by pathnames. You can visualize just about any computer file system as a tree. The beginning directory is designated as “/”, and is generally known as root. Each file has its own unique path. Pathnames can be absolute or relative. An absolute path starts from the root directory, and it selects the same file, regardless of where you are when you use the pathname.

Absolute pathnames…

```
/usr/local/bin/gs
/home/howard/.xinitrc
```

The tilde character (˜) has a special meaning in most UNIX shells. It indicates that you are specifying a user’s home directory. In the following examples, the pathnames are absolute, starting at the current user’s home directory, and at the home directory of a user named “howard”, respectively.

/readme howard/readme

A relative path starts from the current directory. The file selected by the pathname depends on what the current directory is.

Relative pathnames…

```
projects/linux/UnixCommandLine-HOWTO.sgml
readme
./readme
../readme
```

In the last two lines above, we are using special codes to specify the current directory, and the directory the next level down from current, respectively.

If we were sitting in `/usr/local/src/sgml-tools` , the latter two pathnames would give us `/usr/local/src/sgml-tools/readme` , and `/usr/local/src/readme` , respectively.

2.2 File Extensions

File extensions are not formally a part of a UNIX pathname. Dots are just another character. For the sake of clarity, a number of applications, commands and users use extensions, or at least they use strings of characters that look like file extensions.

2.3 What is all this about foo?

It is traditional in UNIX to use the filename `foo` or `foobar` when giving examples of commands that affect files. This is covered in some detail by Wikipedia – `http://en.wikipedia.org/wiki/Foobar`.

2.4 Pathname Wildcards

In many cases, you want to specify a group of files with similar pathnames. UNIX has a sophisticated system of wildcards to let you do this.

foo*bar: The asterisk is a substitute for a string of characters of any length, including zero. The pathname specified above is anything starting with `foo` and ending in `bar` , including `foobar` . You can embed several asterisks in the pathname. The pathname `*foo*` is anything containing the sequence `foo` , including `foobar` , `barfoo` , and `foo` .

foo?bar: The question mark substitutes for one character, no more, no less. The pathnames `foo1bar` , `foolbar` , `foo-bar` and `foo.bar` all conform to this description.

foo[1-5]bar: The square brackets and their contents substitute for one character. Any filename, where that character ranges from “1” to “5” will be selected.

I rarely use anything other than the asterisk wildcard.

2.5 UNIX Commands

A UNIX command line contains up to three different kinds of element – the command itself, the command line switch which provide additional instructions to the command, and command information, such as pathnames.

Eg.

```
rm -ri foobar
```

The above UNIX command is `rm` , which is the command for deleting files. There are two switches, “r” and “i”. Switches are usually (but not always) indicated by a dash “-”. The “r” tells `rm` to recurse into subdirectories to search for files. The “i” tells `rm` to prompt you before deleting files. The name `foobar` is a pathname, which in this case would probably be a directory.

2.6 Getting Help (man)

The standard help resource on any UNIX machine is `man` , which stands for manual. The actual manual files are often called “man pages”. To read the help files on how to delete files you type...

```
man rm
```

On the Sun SPARCstations I learned UNIX on, `man` just dumped its text out to the screen. On any Linux machine I have ever used, `man` ’s output was run through `less` . This text reader is described below under Reading Text Files .

A more recent help resource, at least on Linux systems, is the `info` command. A number of Linux `man` pages state that the man pages are no longer being maintained, and that the help files are now in info format.

```
info info
```

This is how you check the help files on the info command. It's worth a look.

Many of not most commands provide help at the command line.

```
<yourcommand> -?
<yourcommand> --help
```

The `-help` switch is often more informative.

3 Accessing the UNIX Shell

3.1 Types of UNIX Shell

There are a variety of UNIX command shells. Most Linux systems run the Bourne Again SHell (`/bin/bash`). Sun Microsystems used to use C SHells (`/bin/csh`). When I took my Solaris Administration course back around 1995, we were taught how to use the Korn shell (`/bin/ksh`). The original UNIX shell is the Bourne Shell (`/bin/sh`). The Bourne Shell usually doesn't exist on Linux systems. The command is actually a symbolic link to bash. Clones of C and Korn shells are available on a lot of Linux machines as `/bin/tcsh` and `/bin/pdksh` respectively.

If you are curious about these shells, you can read the man pages. Most of the time, it doesn't matter what shell you are running.

The standard UNIX shell prompt is a dollar sign ($), except for C Shells which prompt with the hostame and a percent sign (%). These can be reprogrammed. On Linux, they usually are. The prompt can identify the machine, the user, the current directory, the date. They usually end with a dollar sign, though.

3.2 Console and Terminal Shells

If you logged into UNIX through a VT type terminal, through telnet, ssh, or through a modem, you are probably in a command shell. Sometimes you find yourself in a menu, which offers the option of launching a command shell. The author's ISP used to offer a list of shells, one (`/bin/tcsh`) of which, was "recommended". It is best to take administrators at their word on things like this. They probably spent some time and effort setting up the configuration on that shell, and they didn't bother with the others. All the system commands and utilities will work with the recommended shell. Perhaps they won't work with the others.

3.3 X11

The GUI for most UNIX systems is X11, or the X Window System. While X11 offers GUI tools that perform the functions of a command shell, you still need to know how to get at the keyboard. All sorts of tools are used to launch command shells. Typically, these are identified in your GUI as "Terminals". If all else fails, search the menu for these under "Administration", or "System".

3.4 X11 Keyboard Copy and Paste

In X11, this is incredibly simple, although rarely documented. On any text window on your X Window System screen, highlight some text. Move the mouse to where you want to copy the text. Click on the middle mouse key. Simple eh?

Don't you wish someone had told you that earlier?

This also works on a Linux console if you are running gpm.

If you have a roller mouse, click the roller. If you do not have a roller or a middle mouse button, try pushing the left and right buttons at the same time. This works on all the laptop keyboards I have tried. Cheap mice used to not always work. I have acquired a miniature wireless keyboard where this does not work.[1]

[1] `http://home.eol.ca/~hgibson/GearheadKeyboard.html`

4 Managing Directories (Folders)

Traditionally, a container of UNIX files is called a *directory*. GUI file managers tend to represent these as *folders*, and this terminology is now popular.

4.1 What Directory Am I In?

pwd: Indicate what directory we are in.

4.2 Changing Directories

cd foobar: Move to the directory `foobar`. You may specify the directory as an absolute or relative address.

cd: If you don't tell cd where to go, it will move you to your home directory.

cd -: Move to the previous directory. You were in some directory prior to moving to the current one. This puts you back to it.

4.3 Listing Directory Contents

The command for listing directories is `ls`. Running `ls` by itself results in nothing more than a list of filenames, of the files in the directory. The command line switches in the following examples can be mixed and matched to suit your requirements.

ls: List the files in the current directory

ls -l: List files in current directory, showing file permissions, owners, groups and file size. On SunOS 4.1, you won't get the group listing.

ls -lg: Use this in SunOS 4.1 to include the groups in your detailed file listing.

ls -l foo: List the directory entry of `foo`, showing file permissions, owners and size. If foo is a directory, List its contents, instead.

ls -ld foo: List the directory entry of `foo`, showing file permissions, owners and size, even if `foo` happens to be a directory.

ls -a: List the contents of the current directory, including the hidden files. When a filename starts with a period, UNIX will hide it. In other words, `ls` will not list it unless specifically told to do so. Try this command out from your home directory.

ls -ld foo*: Do a detailed listing of everything in the current directory whose names start with foo. Do not list the contents of subdirectories.

ls -l | more: Do a detailed listing of files in the current directory, pausing at the end of each screen. The character in front of the word "more" is called a pipe. On most keyboards, it is the uppershift backslash, sitting somewhere near the backspace key.

The behaviour of ls varies from UNIX to UNIX. You should read the man page for this.

Here is a sample directory listing...

```
-rw-r--r--   1 howard   users        6425 Apr  3  2000 bar
-rw-r-----   1 howard   users        2498 Apr  3  2000 foo
drwxr-x---   2 howard   users        1024 Oct 25 21:21 foobar
```

The files `bar` and `foo` contain 6425 and 2498 bytes, respectively. They are owned by the user `howard` and the group `users`. There were both last modified on April 3, 2000. Both files may be read from and written to by howard, and may be read by member of the group `users`. The file `bar` may be read by anyone with access to the computer. The file `foo` cannot be read by anyone outside the group users. The other pathname `foobar` is a directory, indicated by the leading character `d` with the file permissions. Files in the directory were last modified on the evening of October 25 of the current year. The directory can only be written to by howard, but members of the group users can read it.

A long time ago when I was a Solaris administrator, someone came to me complaining that the computer would not load one of his files. I eventually ran `ls` with a tag that showed me a control character he had somehow embedded in the filename. I don't bother to keep track of stuff like this. I read the man page back then. I haven't had the problem since.

4.4 Creating a New Directory

mkdir foo: Create the new directory `foo`. This could be a new directory in the current directory, or it could be an absolute pathname.

4.5 Deleting a Directory

rmdir foo: Delete the directory foo. This must be a directory. It can be a relative or absolute pathname. UNIX will refuse to delete the directory if there is anything in it.

rm -ri foo: Delete the directory `foo`, and all of its contents, prompting you before deleting each file.

rm -rf foo: Delete the directory `foo` and all of its contents. Too bad if you made a typo!

4.6 Setting Directory Permissions

File permissions determine who can access your directories. There are three entities you can provide access to, the owner (you), members of your group (see `/etc/group`), and everyone.

If you list a directory with the command `ls -ld foo`, the first column (on Linux anyway) will be the file permissions, shown as something like `drwxr-x--`. The letter `d` shows that the entry is a directory. This particular code shows that the owner can read from, write to, and execute the contents of the directory (`rwx`). Members of the owners's group can read or execute the contents of the directory, and everybody else is allowed no access to the directory.

If you want to change the permissions, you must use the command `chmod` , as follows...

```
Command          Permissions from command ls
-------          ---------------------------
chmod 700 foo    drwx------
chmod 750 foo    drwxr-x---
chmod 770 foo    drwxrwx---
chmod 755 foo    drwxr-xr-x
chmod 777 foo    drwxrwxrwx
```

UNIX directories do not work unless the execute permissions are turned on, as indicated by the `x` 's above.

Computer geeks – note how careful we are to not use the word “octal”.

5 Managing Files

Many, if not most of the UNIX files you will encounter will be generated by an application of some kind. To edit these files, you will need the application. You can still copy them, move them about and delete them. Many of the following commands work better on ASCII text files than they do on binary ones.

5.1 What Kind of File is This?

file foo: Indicate the file type of `foo` . The capability of this command varies from UNIX to UNIX. Under Linux, it is effective, identifying text, executables, application files and graphic formats. This command reads a data file that is generally called a *magic file*.

5.2 Reading Text Files

Aside from text editors, there are three UNIX utilities for reading text files – `cat` , `more` and `less` . The command `cat` dumps the file text straight to your terminal. If you are in an xterm with a scrollbar, you can scroll back and read all of it. The command `more` reads the text screen by screen, waiting for you to hit Enter at the end of each one. The command `less` is not always installed on UNIX, but as far as I know, all GNU/Linux distributions include it.

cat foo.txt: Print foo.txt to the terminal,

more foo.txt: Print foo.txt to the terminal, pausing at the end of each screen.

less foo.txt: Use less to read the file foo.txt.

The command `less` scrolls up and down, using the cursor keys as well as “u” and “d”. To do keyword searches, hit “/”, followed by the string of characters you want to search for, then hit Enter. To exit, hit “q”.

5.3 Editing Text Files

There are an awful lot of text editors available to UNIX. Some of these are applications in themselves, with full GUI support, and syntax highlighting. At least one text editor has a built-in web browser, a minesweeper game, and an online psychoanalyst that you can feed the built-in Zippy the Pinhead quotes to. You can even edit text with it!

All of the following editors run from the command line, although some of them have GUI versions. With one horrible exception, all of the following editors provide on-screen help, so you don’t have to know how it works, to run it. They are listed in descending order of user friendliness.

nano foobar: This is a version of `pico` that does not automatically wrap text. This has been installed on all the GNU/Linux distributions I have tested lately, including Slackware. If you are not a dedicated GNU/Linux geek, this probably is your best text editor.

pico -w foobar: This is the text editor used by the mail reader Alpine. Alpine is a new version of what used to be Pine. Alpine definitely can be installed on your GNU/Linux distribution. Pico was actually written to write email messages, so it wraps text for you. This is unacceptable for most command text, so you must turn the feature off using the `-w` switch, as shown. Pico displays a command menu at the bottom of the screen. You can also type `pico -?` to get a list of command line options.

jed foobar: Emulates a bunch of other editors, including Emacs and Wordstar. It provides a command menu at the top of the screen.

joe foobar: Joe's Own Editor. This used to be available with most Linux systems. It tells you how to get help at the top of the screen.

emacs foobar: Emacs is *the* standard UNIX text editor,[2] originally written by Richard Stallman. On newer systems, including Linux, EMACS comes up as a GUI editor, with menus accessible using the mouse. The command line version is a bit awkward to figure out. Most of the help for Emacs is in the info pages, and you should read these carefully before launching it. To get help, hit [CTRL]h. To exit Emacs, hit [CTRL]x, then [CTRL]c. There are several editors based on Emacs, including Jove (Jonathan's Own Version of Emacs), and Xemacs, the editor with the features listed above and which has an excellent GUI interface. New users will probably want to find something a little more friendly for the command line.

vi foobar: This is the horrible exception. Vi is an efficient editor once you have learned it, especially if you touch type. Unfortunately, It has no intuitive qualities. If you have somehow gotten into `vi` , hit the Escape key a couple of times, hit the colon ":" key, type q! ("que bang")[3] at the prompt at the bottom of the screen, then hit Enter. If you plan to use/manage a UNIX/GNU/Linux box, you really should learn vi. Search the internet for the vi HOWTO. There are all sorts of web pages that explain it. Most of this HOWTO was typed up with `vi` .

vim foobar: If you are running Linux, your `vi` is a symbolic link either to `vim` or to `elvis` . Probably, it's `vim` . Try typing `vi` or `vim`

[2]EMACS used to stand for Eight Megs And Continues Swapping. Back in the day, eight megabytes was a lot of memory.

[3]No, this is not a Maserati!

by itself, without the filename. You should get a useful information message. From vim, you can type `:help` to get into their help system.

5.4 Copying Files

cp foo foobar: Copy the file foo to foobar. If foobar is a directory, foo will copied into it, and named foo.

cp -i foo foobar: Same as above, except that if you already have a file called `foo`, or `foobar/foo`, you will be prompted before it is overwritten.

cp foo* foobar: Copy a whole bunch of files whose names start with `foo`, into the directory `foobar`. If `foobar` is not a directory, or it doesn't exist, the command will return an error.

5.5 Moving (Renaming) Files

mv foo foobar: Move the file `foo` to `foobar`. If `foobar` is a directory, `foo` will be moved into it.

mv -i foo foobar: Same as above, except that if you already have a file called `foo`, or `foobar/foo`, you will be prompted before it is overwritten.

5.6 Deleting Files

rm foo: Delete the file foo.

rm -i foo*: Delete all files whose names start with foo. The `-i` switch causes `rm` to prompt you before deleting each file.

5.7 Setting File Permissions

Permissions determine who can access your files. There are three entities you can provide access to, the owner (you), members of your group (see `/etc/group`), and everyone.

If you list a directory, the first column (on GNU/Linux anyway) will be the file permissions, shown as something like `-rw-r---` . This particular code shows that the owner can read from, write to, or delete the file (`rw-`). Members of the owners's group can read the file, but cannot write to or delete it, and everybody else is refused access.

If you want to change the permissions, you must use the command chmod, as follows...

```
Command          Permissions from command ls
-------          ---------------------------
chmod 600 foo           -rw-------
chmod 640 foo           -rw-r-----
chmod 660 foo           -rw-rw----
chmod 644 foo           -rw-r--r--
chmod 666 foo           -rw-rw-rw-
chmod 755 foo           -rxwr-xr-x
```

The execute permissions are only required if the file is an executable of some kind. You need not worry about this unless you are writing shell scripts.

5.8 Intelligent File Searches (find)

The tool for file searching in UNIX is `find` . This allows you to search filesystems for files or directories conforming to a list of specified characteristics. The structure of the find command is as follows...

```
find foobar -mtime -10 -size +1000 -ls
```

This command will searche the directory `foobar` for files modifed sometime in the last ten days, and occupying more than 1000 disk blocks.

When it finds such a file, it will list it to the screen. There are a lot of switches for find, and you can use any number of them...

-mtime: Modify time, in days. `-10` means less than ten days old. `+10` means more than ten days old. `10` means exactly ten days old.

-size: Size, in disk blocks. `+500` means more than 500 blocks. `-500` means less than 500 blocks. Disk blocks vary in size.

-name: Do wildcard searches for filenames. Unfortuntately, we can't just use the asterisk character. It must be "escaped" with a backslash, eg. `foobar` .

-type: Select files by the type of entity they are. The switch `-type f` will select only files. The switch `-type d` will select only directories.

-print: Print the pathname of any file conforming to the list to the terminal.

-ls: Print a detailed listing of any file conforming to the list to the terminal.

Read the man pages for this. This command does a lot more stuff.

5.9 Intelligent File Searches (grep)

The command `grep` and `zgrep` are pattern searchers. You feed them a pattern and some text, and they spit out the lines that match the pattern. Here are some examples. . .

grep Howard *.txt: Search text files in the current directory for the pattern "Howard". Print out any lines that contain the pattern. Note that I have supplied an extension, to limit the search a little. Otherwise, grep would try to read any subdirectories it found, and this would cause an error.

grep -i Howard *.txt: Do the same search as above, except that now the pattern matching is case insensitive. The text "howard", "HOWARD", "hOwArD" would all match, and the text lines containing them would be printed.

grep -li Howard *.txt: Do the case insensitive search as above, except now, only print the filenames. Don't print the actual text.

ls -l | grep Mar: You can feed data to grep using a pipe. In this case, we are doing a detailed directory listing, and using grep to filter the results. Only directory lines containing the characters "Mar" somewhere, will be printed to the terminal. Effectively, this lists all files saved or updated in March. Unfortunately, it will also print any files with "Mar" in the filename, username or group. Oh well.

5.10 Search Through Office Suite Files

Most office suite files are file packages compressed by zip, with the actual text stored as XML.. To search your word processor, spreadsheet and presentation files for text, you must first composess them.

```
zgrep -i Howard *.doc
```

If `zgrep` finds the text "Howard", it will indicate that the file matches.

5.11 Intelligent File Searches (find and grep)

Here is how to search through a file system for files containing a text string. . .

```
find . -type f -exec fgrep -li howard {} \;
```

The above line searches through the directory tree for files. Each file is processed by executing the command fgrep, with the switch `-li` . The string "howard" is searched for. Each file containing the string somewhere, will be listed to the terminal.

The `-exec` switch allows you to use standard UNIX commands with your find search. The `-exec` sequence is terminated with a semicolon. A number of characters must be “escaped” in the command line with a backslash. These include the semicolon, as shown above, the asterisk (*), useful for filename substitution, and the brackets.

5.12 Creating a CD-ROM/DVD Database

You can easily generate and use a listing of everything you have stored on CD-ROM and DVD. This will also work on floppies for those among you who still have floppy drives, and on USB sticks.

We need to do two things here. First, we must create a listing of the files on the disk. Then, we need a procedure for searching through this listing.

The first thing to do is create a directory where your CD-ROM listings will be stored. On my system this is `/home/howard/disks/cdroms` . Create this directory, then move into it.

```
mkdir /home/howard/disks/cdroms
cd /home/howard/disks/cdroms
```

To catalogue an individual CD-ROM, you must mount it. On current Linux distributions, this happens automatically when you insert the disk into your machine. Probably, you need a file manager running.

If your disk does not mount, you will have to read up on your mount command and your `/etc/fstab` file. These are outside the scope of this HOWTO.

Once your disk is mounted, there should be a file path leading to it. You should be able to copy this from your file manager, and paste it into your terminal. I want to list files, and I want to go down into all the subdirectories and list their contents too. Let’s assume my disk is called “Downloads201112".

```
ls -lR /run/media/howard/Downloads201112 > Downloads201112
```

The results of the listing are being redirected from the standard output (my terminal) to a file called `Downloads201112` .

When I have done this for each of my CD-ROMs, I have a directory with a complete file listing for each of my CD-ROMs. Each file on the CD-ROMs is listed as a complete pathname including the mount point of my CD-ROM.

Note how the directory `/home/howard/disks` can be broken into subdirectories. I would probably want to store my photos in a directory deparate from cdroms.

Now, weeks, months or years later, I want to search for something that reads the Kodak Photo CD-ROMs. The files on my photo CD have the extension `.pcd` , so I figure that a program that reads them will include this string. I am sitting in my home directory.

```
grep -ri pcd disks
```

The system will respond by searching the directories under disks, and printing out every line of each of my CD-ROM listings that contain the sequence `pcd` in any combination of upper and lower case (the `-i` switch). These will be printed with the filename, so I will see what CD-ROM these were found on.

Now all I have to do is mount the CD-ROM and take a look at the file. Since the complete pathname is being displayed, I can easily copy and paste directly from the output.

6 Archives, Distributions and File Compression

There are two issues here.

- Almost all software is distributed as some form of archive. If you want to install software, you are going to have to unpack an archive of some sort. Increasingly, commercial software is being delivered as some sort of an executable that unpacks itself, but we don't want to confine ourselves to this, do we?

- Sometimes, you need either to store or recover the entire contents of file systems. There are quite a few file formats and commands for this. The two most popular are tar and zip. RPM is used for Red Hat based Linux program distributions.

Exchanging file packages with people can be a challenge. You need to know what the other user's operating system can do, as well as what the other user can do. Linux, out of the box, can unpack just about anything generated on Microsoft Windows. Windows users require software in addition to the standard Windows distribution to unpack archives. You can exchange files with MacIntosh users, but you may have to install some software.

Usually, file packages are compressed, to save disk space and download time.

6.1 Gnuzip (.gz) and Bzip2 (.bz2)

The commands `gzip` , `gunzip` , `bzip2` and `bunzip2` are file compression utilities.

gzip foo: Compress the file `foo` . The compressed file will be saved as `foo.gz` .

bzip2 foo: Compress the file `foo` . The compressed file will be saved as `foo.bz2` .

gunzip foo: Uncompress the file `foo` . Gzip does not require you to type in the extension.

bunzip2 foo.bz2: Uncompress the file `foo` . Bunzip2 requires you to type in the whole filename.

Bzip2 compresses somewhat better than Gzip does. Bzip2 does a better job of extracting files from a faulty archive. Gzip runs way faster.

Gzip and Bzip2 are installed with most new Linux distributions. Both are available as Gnu free software with source code. On Microsoft Windows, WINZIP can uncompress gzipped files.

6.2 Zip (.zip)

The UNIX command Zip manages file archives compatible with the standard MS-DOS and Windows PKzip and WINZIP format. If you are exchanging files with Windows users, this is the best format.

zip: Get a brief description of the command zip.

zip -r foobar.zip foobar: Create a zip file `foobar.zip` from the directory `foobar` . The `.r` switch makes it preserve your directory structure. Almost certainly, this is good.

unzip -l foobar.zip: Print out a list of the contents of `foobar.zip` . Almost certainly, you want to unpack your files into a sub-directory. Unpacking into a current directory full of unrelated files is a disaster. If the zip file does have contain a sub-directory, you will have create one, and move into it before unpacking.

unzip foobar.zip: Unpack the archive `foobar.zip` .

Pkzip is available for Linux. This is the original program for generating `*.zip` archives. The program even supports self-unpacking archives. Unfortunately, they self-unpack on Linux, not on Windows, where you are more likely to have users without archiving programs.

6.3 Xz (.xz and .lzma)

I have encountered this installing FreeBSD. Apparently, WINZIP supports it. Wikipedia claims it compresses better than Gnuzip and Bzip2, that it runs much slower than Gnuzip, and that it runs slower than Bzip2.

xz -z foobar: Compress file `foobar` . The extension `.xz` will be attached to the file.

xz -d foobar.xz: Uncompress file `foobar` . You have to type the file extension, as shown.

xz –help: Show help instructions.

xz –long-help: Show detailed help instructions.

I have not used ths. If you want to play with it, read the `man` pages and help files.

The `man` page mentions .lzma compression.

6.4 Rar (.rar)

I believe this is being pushed as the latest archiving program for Microsoft Windows. A Free Software version of `unrar` is supplied with most GNU/Linux distributions now.

The publisher RARLAB, has Linux and FreeBSD versions of their software. These are shareware, and you are expected to pay for a license. I have not played with them much. They provide command line help.

6.5 Compress (.Z)

Historically, UNIX files were compressed using the command `compress` . This might be available on your system. The compressed files usually have the extension `*.Z` .

compress foobar: Compress the file `foobar` . The name will be changed to `foobar.Z` .

uncompress foobar: Uncompress the file `foobar.Z` . The name will be changed back to `foobar` . You don't have to tell compress about the file extension.

I don't think this is used much anymore. As of 2022/01/07, it is not available for Fedora 33.

6.6 Tape Archives (.tar .tar.gz .tgz)

Tape archives were originally intended as a format for writing to magnetic tape. They are now used systematically to distribute GNU free software, usually with some sort of data compression. They are not used much outside the UNIX/GNU world.

tar -cf archive.tar foobar: Create an archive called `archive.tar` from `foobar`, which almost certainly is a directory. There would be no point to doing this to an individual file.

tar -cv archive.tar foo bar barfoo foobar: Create an archive called `archive.tar` from the files or directories `foo`, `bar`, `barfoo` and `foobar`.

tar -cvf archive.tar foobar: Create an archive called `archive.tar` from `foobar`. The `v` switch means verbose output. You will see a file listing printed to the screen as your archive is created.

tar -zcf archive.tar.gz foobar: Create an archive called `archive.tar` from `foobar`, and then compress it using Gnuzip. This works only on systems using Gnu tar. GNU/Linux comes with Gnu tar. An alternate file extension for Gnu zipped tar files is `*.tgz`.

sudo tar -cf /dev/fd0 foobar: Write out your tar format file directly to a storage device. On GNU/Linux, `/dev/fd0` is a floppy drive. There is no need to format the floppy. It will be in tar format, and your file manager will *not* mount it. This works on USB sticks and other USB devices. It does not work on CD-ROM, DVD and Blu-ray. You must write the tar to a file, then transfer it to the disk. You need `sudo` to access the device.

tar -tvf archive.tar: Print out a list of the files in `archive.tar`.

sudo tar -tvf /dev/fd0: List the contents of an external device. `/dev/fd0` is a floppy disk on a Linux box. This will work on various USB devices too, and on CD-ROMs, DVDs and Blu-rays. You need `sudo` to access the device.

tar -ztvf archive.tar.gz: List the contents of a compressed tar archive.

tar -ztvf archive.tgz: Print out the directory of a Gnu zipped tar file. This only works on Gnu tar.

tar -xf foobar.tar: Unpack the archive `foobar.tar`. By unpack, I mean that the files contained in the archive will be written out to the current directory.

tar -xvf foobar.tar: As above, but this time, print out the list of files being unpacked.

tar -zxf foobar.tgz: Unpack the Gnuzipped archive `foobar.tgz`. This only works with Gnu tar. As noted above, a Gnuzipped tar file can have the extension `*.tar.gz`. By unpack, we mean that the files contained in the archive will we written out to the current directory.

tar -zxvf foobar.tgz: As above, but print the list of files being unpacked.

If you write raw `tar` files out to floppy disks, USB devices, CD-ROMs, DVDs, and Blu-rays, they will not be mounted by your file manager. Only *you* will know how to access the data. Possibly, this is a good thing.

The switches `c`, `t` and `x` are used to Create, lisT and eXtract files to and from the archive. The `f` switch indicates you are providing a pathname to something other than your default tape device. The `v` switch requests Verbose output, which means that tar lists each files as it adds it or extracts it to/from the archive. If you want to list an archive using the `t` switch, you must use the verbose switch as well. The `z` switch works only in Gnu tar, and it activates Gnu zip.

If you are not running Gnu tar, you do not have the `z` switch to activate Gnu zip. You will have to uncompress the archive with Gnu zip as a separate operation. Heck, you may have to install Gnu zip!

The command line switch dash ("-") is optional with Gnu tar, and possibly, some of the other versions.

If you are preparing to unpack an archive, you should always check the contents to verify that it will unpack into a new subdirectory. Unpacking an archive into an existing directory that already has a lot of stuff in it usually results in a mess.

Microsoft Windows users can unpack tar files, even if they have been Gnuzipped. WINZIP supports the format. WINZIP first removes the gnuzip, providing you with the tar file. When I did this, I had to change the tar file's name from `foobar_tar` to `foobar.tar` . Then, WINZIP unpacks it.

6.7 Red Hat Package Manager (.rpm)

This is used to distribute binaries for Red Hat, Fedora, Suse and others. It is quite an elaborate program. To understand it properly, you will have to read Red Hat's documentation. The following instructions will enable you to inspect, install and uninstall software from RPMs. Rpm provides quite a bit of information if you run it by itself, without operands. To install or remove software with `rpm` , you need `root` or `sudo` access.

rpm -qlp foobar.rpm: List the contents of the RPM file `foobar.rpm` .

sudo rpm -i foobar.rpm: Install the package `foobar.rpm` . Rpm will check your system for dependencies before installing the package. Rpm will refuse to install `foobar` if a current or later version of `foobar` is already installed, or if some package required by foobar has not already been installed.

sudo rpm -i –nodeps foobar.rpm: Install the package `foobar.rpm` . Do not check for dependencies. Dependency checking can be frustrating at times, but you had had better know exactly what you are doing if you want to skip them. You have been warned!

sudo rpm –erase foobar: Remove the package `foobar` from your system. Again, the dependencies are checked, and foobar will not be removed if some other package requires it.

sudo rpm –erase –nodeps foobar: Be very, very careful with this. See my remarks above.

rpm -q foobar: Print out brief information on the package `foobar` you have already installed on your system. Basically, you get the version number. Read the rpm documentation for more information on this.

rpm -qa: Print out a list of all the packages installed by rpm.

Red Hat's `dnf` command installs rpm packages, and/or it searches online for the required package(s). It checks for software packages, dependencies, required by your package, and it finds and installs these. The `dnf` command supercedes the `yum` command.

sudo dnf install foobar: Download and install the package `foobar` .

dnf search foobar: Search the package repository for anything containing the string "foobar".

sudo dnf install foobar.rpm: Install the package `foobar` from the rpm file in your current directory.

6.8 Debian Packages (.deb)

I have minimal experience with Debian packages. The command `apt` (formerly `apt-get`) is a package installer similar to Red Hat's `dnf` . When you call it, it searches the internet for the package. It searches for and installs any other software packages it depends on.

sudo apt install foobar: Download and install package `foobar` .

sudo dpkg -i foobar.deb: Install the package `foobar` , which is sitting in your current directory.

6.9 Slackware Packages (.tgz)

Slackware installs its binaries from `*.tgz` files which have a little script embedded in them. You should try to use Slackware's `pkgtool` to install these things, but `tar zxf` will do in a pinch.

If you are reading this document, you should not be installing or administering Slackware! :)

6.10 FreeBSD Packages

FreeBSD uses a command called `pkg`.

$ pkg search firefox: Search for packages containing the text “firefox”. Note how I do not need to log in as root. I am showing the command line prompts.

# pkg install firefox: Install the package `firefox` . Now, we do need to be root.

6.11 MacIntosh Archives

I haven’t had much practise with Macs. There was something out there called `macutils` , but I cannot find it for Fedora, It contains things for communicating with MacIntosh computers. This could *not* unpack the one MacIntosh file I tested.

I have used the package `uudeview` . You can get this from `http://www.fpx.de/fp/Software/UUDeview` You will have to read the instructions.

You can compile your kernel so that your GNU/Linux box can read MacIntosh floppies. This should not be much more difficult than actually finding MacIntosh floppies. :)

The Fedora distribution of GNU/Linux comes with `hfsutils` , which provides resources for managing Mac `hfs` file systems. I have no access to a Mac, so I cannot test it.

7 Command Shell Scripts

The time will come when you decide you want to automate some UNIX command procedures. You are in luck. UNIX shells are sophisticated programming tools.

A good plan when writing shell scripts is to write for the Bourne shell. The Bourne shell is the original UNIX shell. Anything that runs Bourne shell can be run on any other UNIX machine. The following instructions will show you how to create a Bourne shell script. The script won’t do anything useful, and it won’t show you nearly everything the Bourne shell language does, but it will provide a good starting point for hacking your own scripts.

To create a Bourne shell script, you must create the file, change the permissions to make it executable, then write in the UNIX commmands.

touch funfunfun: The command `touch` either creates a new file with zero bytes, or it sets the modify date to the present, depending on whether or not the file exists.

chmod 755 funfunfun: You must set the permissions to execute. You are allowing anybody to execute the file, but only you can write to it.

emacs funfunfun: Load the script into your favourite text editor. My favourite text editor is `vi` , but you don't want to run this unless you know what you are doing. Even Emacs is nasty if you are not running in X11. If you are on a modem or a VT terminal, you might want to try `nano` instead.

Now, we type in the actual script. You can copy and paste the following text from this window to your text editor.

```
#!/bin/sh
# Anything past this pound sign is a comment, except for
# the pound sign in the command following these comments.
# The string $# is the number of operands you typed in on
# the command line.  The string $0 is the command you typed.
# $1 is the first operand, $2 is the second operand, etc.

OPERANDS=$#

# The ‘echo’ command is used to print text to the screen.
# Note how I have left a space as the last character
# before the quote.  You don’t actually have to do this,
# or even use the quotes, most of the time.  Using them makes
# the command more reliable, since some of the characters
# you may want to use as text may be seen by the computer
# as commands.  Just for fun, change the period at the end
# of ‘operands’ to an exclamation mark, then delete the
# trailing space -> ...operands!"  ->  Then, run the script
# and see what happens.
```

```
echo "You provided ${OPERANDS} operands. "

if [ ${OPERANDS} = 0 ]; then
   echo "You did not provide operands, so I will list "
   echo "this directory... "
   ls -l
elif [ $1 = $LOGNAME ]; then
   # $LOGNAME is the environment variable that holds your
   # username under most UNIXes.  Some UNIXes recognize
   # $USER as well, or instead.
   echo "Ohh!  You typed your username! "
   echo "Hello there $1! "
elif [ -f $1 ]; then
   echo "Ooh!  You have a file called $1!  Let's list it.  "
   ls -l $1
elif [ -d $1 ]; then
   echo "Ooh!  You have a directory called $1!  Let's list its contents.  "
   ls -l $1
else
   echo "You provided operands.  I will list your directory anyway... "
   ls -l
fi
########################### Bottom of Script ########################
```

To run this, type

```
./funfunfun
```

To run it with an operand, type

```
./funfunfun <operand>
```

For operands, try anything, including valid file and pathames, and your username. Then, modify the program, and see what you can make it do.

The command `if` tests a logic command. When the command returns `TRUE` , it executes the following sequence of commands. When the command returns `FALSE` , it executes the following `elif` or `else` command, otherwise, it stops at `fi` , and goes to the next line in the script.

The line…

```
elif [ -f $1 ]; then
```

…tests whether or not the file specified, exists. This is done using the switch -f. This could also have been written as…

```
elif test -f $1; then
```

There are a bunch of these switches available. You can test for valid files, for valid directories, valid pathnames, if files can be written to, and lots of other stuff. These switches are listed by the `man` page for `test` .

8 TODO List

Since this is not a very sophisticated document, there should be little need to manage new information. I guess the thing to do is ponder how clear this document is, and how complete it ought to be. I will be grateful to anyone who writes in with suggestions on this.

I have not used the Debian package manager at all. I have installed stuff using `apt` .

I have no access to a Mac. Macs run on top of some form of BSD UNIX.

I have now installed FreeBSD on one of my hack machines. Otherwise, I have not touched a UNIX system other than GNU/Linux in over twenty years. That is an awful lot of computer years. SunOS and HPUX have had lots of time to change stuff.