Guide to Unix

Contents

Articles

References

Article Licenses

Guide to Unix

This is the **Wikibooks Guide to Unix Computing**, or for short, **Guide to Unix**. It describes Unix and Unix-like systems for *users* and *system administrators*. We include a **guide to commands** which lists several of the common shell commands.

Brief contents

Authors

1. Kernigh, added much content, started Explanations
2. CFeyecare. I have been working on OpenBSD and I will continue to other BSDs later.
3. The authors of Guide to UNIX/Files and Guide to UNIX/Commands
4. Other and anonymous contributors
5. Other Sources

Contribute

Guide to Unix is incomplete. At Wikibooks, ***you can edit this book now****. Here are some suggestions for improvements:*

- *improve content, add detail*
- *add content, follow naming policy*
- *enforce neutral point of view, reduce bias*
- *provide graphics, follow media policy*
- *fix bad info, spelling, grammar*
- *put questions and comments on talk pages*

Other Wikibooks

There are some other Wikibooks that one might find useful.

- A Neutral Look at Operating Systems *comparison between Unix, Unix, and non-Unix*
- UNIX Computing Security *keep that computer safe*
- Wikibooks for Unix utilities:
  - Bourne Shell Scripting *a common Unix shell*
  - Learning the vi Editor *a frequently-encountered text editor*
- Many Unix-like systems feature free software, and some are entirely made of it:
  - FLOSS Concept Booklet and Open Source explain the concept.
  - Use the Source is a more general description of the purpose and origin of free and open source software.
- Wikibooks for the graphical user interface and the GNOME and KDE desktops:
  - Guide to X11 *X Window System*
  - Using GNOME *GNU Network Object Model Environment*
  - Using KDE *K Desktop Environment*
- Programming Wikibooks
  - C - C is *the* programming language for Unix
  - C++ - C++ is a popular programming language for Unix, it evolved from C.
  - Wikibook module Serial communications in Linux and Unix from the Serial Data Communications book.

Introduction

This book is a *Guide to Unix* and Unix-like operating systems, such as GNU/Linux and *BSD. Other systems like Mac OS X, Solaris, and OSF/Tru64 also belong in the list.

Because of this book's incomplete state, it might be hard to find the chapter that you want.

Structure of this book

After this introduction, there are three main parts of this book.

- The ../Why Unix-like/ page introduces the Unix system-like platform. The various Platforms such as ../BSD/ then highlight particular Unix-like distributions. This book needs to add more platform pages other than ../BSD/ and ../Linux/.
- The ../Explanations/ section contains a list of various topics which *introduce* and *explain* the use and administration of Unix-like systems. For example, ../Explanations/Shell Prompt/ introduces the shell prompt, and ../Explanations/Filesystems and Swap/ explains the use of storage devices. Many of the sections are incomplete or missing, but this section is being expanded. The current division into six parts might need to be changed later.
- The Quick References chapters list the ../Environment Variables/, ../Files/, ../Commands/. There actually is not much here except for the **../Commands/** section.

Conventions

This book uses (or will use) the following conventions.

Shell prompt

The shell prompt looks like:

```
$
```

A root shell prompt (see ../Explanations/Becoming Root/) looks like:

```
#
```

When the user types commands or other text, it appears in bold. The following is an example. The user typed the "cat" command and then several lines which the computer echoed.

```
$ cat
This is an example.
This is an example.
^D $
```

Control characters are written like **^D**. This means to hold the Control key and press D. Note that some control characters sometimes do not appear on the screen. For example, the user typed **^D** but no "^D" actually appears on the screen.

Commands and files

This convention might need improvement. Currently, a good example is ../Explanations/Shell Prompt/.

A command name is introduced in bold, like **uname**. Later, it is mentioned as "uname". In the future, the bold version might be a link to the command in ../Commands/.

Filenames usually appear like `/dev/null` and `/etc/ssh/sshd_config`. Entire commands look like **`ls /dev/null /etc/sshd_config`** or **`echo a1 a2 a3`**. Mentioning again the parts or arguments of these commands looks like "a1" and "a2". When introducing a new part (like a new option) not mentioned before, that looks like **-r** or **-o loop**. Text from files is also quoted, for example "# comment".

Audience

*This is a **proposed** convention, because it is mostly unimplemented in this book.*

The book targets multiple audiences.

- Unix or non-Unix users seeking background
- Unix system users (background and user instructions)
- Unix system administrators (background and administrator instructions)

To handle this, there might be some templates.

- {{Guide to Unix:u}} begin user instructions
- {{Guide to Unix:eu}} end user instructions
- {{Guide to Unix:i}} begin administrator instructions
- {{Guide to Unix:ei}} end administrator instructions

Command links

*This is a **proposed** convention, because it is mostly unimplemented in this book.*

1. In Guide to Unix/Commands, there is a {{Guide to Unix/Clink}} with links to outside Internet resources like manual pages and wikis.
2. In addition, the clink template links between Guide to Unix/Commands and Guide to Unix/Explanations. There are two template calls; link both ways. For example, connect Guide to Unix/Explanations/bc and bc's entry in Guide to Unix/Commands.

Why Unix-like

The operating system installed on many servers and some workstations is **Unix-like**. But what does it mean to be like Unix? In this book, a Unix-like system is one that is similar to *BSD, GNU/Linux, Solaris, and the original Unix. Today, Mac OS X also qualifies as a Unix-like system.

General Concepts

As opposed to the (point and click) (graphical user interfaces) familiar to the general computer user, work usually gets done in unix in a text-based way, through what's known as the (command line shell).

As opposed to a (single-user) operating system which permits computer usage only by one person at one particular time, unix is a multi-user system, that allows access of multiple users to the computer simultaneously. Normally this is achieved by having the users access the system (remotely), through digital networks.

Whether accessing (remotely) or not, users need a (user account) before being granted access to the system; for the purposes of accounting, security, logging and resource management.

Since each particular user account has varying degrees of control over the system and its resources, having the ability to verify the true identity of a given user is crucial, so a method exists that verifies each user account (username) against a corresponding (password); in a process known as (logging in).

All Unix-like systems are similar. As with many operating systems for servers, the Unix-like systems can host multiple users and programs simultaneously. Some features are specific to Unix-like systems. The Unix-like systems provide a common command line interface called the *shell*. They also provide a common programming interface for the C language. The latter fact allows most Unix-like systems to run the same application software and desktop environments.

Unix is popular with programmers for a variety of reasons. A primary reason for its popularity is the building-block approach, where a suite of simple tools can be streamed together to produce very sophisticated results. Another reason is the philosophy that 'everything is a file', which means that a standardized set of operations and functionality can be performed on different file types (directories vs. regular files), hardware devices, and even system processes.

The shell

The *shell* is a program unique to Unix-like systems. It lets you type commands to launch other programs.

When you login to the system through a text-only terminal, Unix gives you a login shell. If your system has a graphical user interface such as GNOME, KDE, or anything that uses X Window System, you can access the shell through a program called a console emulator or terminal emulator. This emulates the text-only terminal that the shell requires for running.

Unfortunately, the shell and the commands are hard to learn. Further, many commands require "arguments" to work. For example, the **rm** command, which removes files, needs one or more "arguments" naming the files to delete. This book has a ../Explanations/Shell Prompt/ chapter which introduces the shell and its many features.

One can automate tasks by saving shell commands in a text file called a shell script. For example, shell scripts are used to boot the system.

The Unix shell is unique to Unix-like systems. Actually, there are multiple shells available for Unix. These shells extend their features in different ways. For the "Bourne-compatible" shells, there is a book in Wikibooks,Bourne Shell Scripting, to describe them. Most shell scripts are for Bourne-compatible shells.

The C language

Unix is the origin of the popular C language. Every program on the computer links to the *C library* which provides basic system features including access to the kernel. Even if an application is written in another language, like C++, it still links to the C library. An interpreted language like perl needs a perl interpreter linked to the C library.

This dependence on C can be a disadvantage. Most Unix-like kernels are written entirely in C; most common programs use C, C++ or Objective-C. It is difficult to add code to these programs in another language such as Fortran. In contrast, some non-Unix systems allow different programming languages to interact more easily.

This book does not describe how to program with C; that is the job of the book Programming:C. However, planned additions to this book will describe how to build and run Unix programs when you obtain their C source code.

The kernel and userland

Each Unix-like system has a *kernel*. This program controls the computer hardware.

All other programs on the system are part of *userland*, which means outside the kernel. The kernel shares the system between all running userland programs. To use the keyboard, the network connection, or another part of the hardware, a userland program must contact the kernel using system calls. The kernel allows multiple programs to share the hardware safely. It also switches programs in and out of the processors; thus it is a "multitasking" kernel.

For example, a web server like Apache, through the kernel, can simultaneously make multiple network connections to multiple web browsers. The same computer might also be running other server programs. As another example, a user on desktop system (with GNOME, KDE, or Xfce for example) can simultaneously open multiple windows containing file managers, word processors, and games. While the file manager copies files and the word processor prints a document, the user can play a Tetris clone at the same time.

Today, a very popular kernel is ../Linux/. Linux qualifies as free and open source; even if Linux is running a server for millions of users, there is no licensing fee. Linux is usually combined with GNU to form the UNIX-like operating system, GNU/Linux (although there are some instances where it isn't combined with GNU). Thus, an easy way for a home user to obtain a Unix-like system is to install a GNU/Linux distribution that includes a friendly desktop.

When we want to configure the network connection, storage systems, or other parts of the hardware, we must often use special utility programs that configure the running kernel. Some of those programs are described in this book.

Neutral point of view

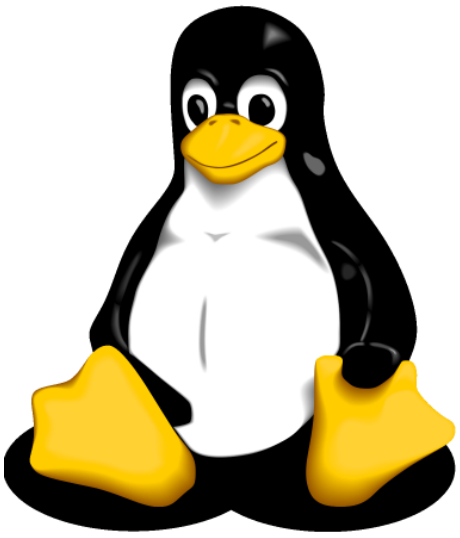
The intention is that this book has neutral point of view. Thus, this book claims not that Unix is always the best choice for any task. It will mention both advantages and disadvantages of Unix-like systems.

Linux

Linux® is an open source, Unix®-like kernel and operating system. The author of the kernel itself is Linus Torvalds, plus a loosely-knit team of programmers who enhance it in a collaborative effort over the Internet. This page provides a brief overview of the main features of the Linux kernel and system, especially in comparison to other Unix-like systems. There are several other Wikibooks about Linux with more information.

General Introduction

- *Distributions:* The Linux system that you actually obtain, install, and run, is one of the many available Linux distributions. A Linux distribution (or 'distro' for short) is a bundle of the Linux kernel and applications and a suite of programs for installing and maintaining a Linux system.
  - There is much variation between distributions concerning the available packages, the installation procedure, the preferred desktop environment, and even the Linux kernel configuration. Some distributions have different target audiences: these might be home users, enterprise users, free software supporters, hobbyists, or users of certain applications.
  - Typically, distributions are in CD or DVD form. One can download them as a raw disc image in ISO format and burn it. One can also inexpensively purchase a Linux distro as a retail package. Live CDs or DVDs allow users to boot from a disc and run Linux using a RAM disk rather than a hard disk.
  - A Neutral Look at Operating Systems has a comparison of Linux systems.
  - Wikipedia has a list of Linux distributions.



After Linus Torvalds suggested a penguin, Larry Ewing (lewing@isc.tamu.edu) used The GIMP to draw this popular image of Tux, the Linux mascot. Tux is now also the hero of some Linux games. In Guide to UNIX, "tux" has an account on the example machines.

- *Packages:* A complete GNU/Linux operating system uses many free software (open source) packages from the BSD, X11, GNU, and other projects. There are also non-free packages without source code, or requiring one to buy licenses. Each distro decides which packages to provide.
- *Mascot:* The official mascot of Linux is Tux, the Linux penguin.

Unix Core

Kernel

- *Copyright license:* The Linux kernel and its source code are released under the GNU General Public License. Many of the userland packages also use this license. If you modify Linux or another package using this license, you can redistribute your modified version, but you must use the same license, which means you must provide source code. Every Linux vendor provides Linux source code.
- *Separate development:* The Linux kernel is developed and maintained separately from userland.
  - With most other Unix-like systems, the kernel is maintained at least with the basic set of utilities, which would include the shell, shell utilities, C compiler, C library, system configuration utilities, and system boot scripts.

    - Linus Torvalds and his team can worry about the kernel (and userland programs which query and configure the kernel) and let other teams develop the userland system separately.
    - Both kernel and userland maintain extra compatibility so that one can update only the kernel, or only some userland package like the C library. In general, the Linux distro vendors take care of making updates install correctly.
- *Customizable kernel:* Linux is probably the most customizable Unix-like kernel. There are even several ways to edit the kernel configuration before building the kernel.
    - Most builds of Linux use modules aggressively (though use of modules is optional). Modules allow pieces of the monolithic kernel, such as device drivers and network protocol implementations, to reside in separate files, load on demand, and unload when not used.
    - Many systems have multiple versions of Linux headers installed, in `/usr/include/linux-`*`version`*. Both the Linux kernel and some userland software, such as the C library, need these. On non-Linux systems, `/usr/include/sys` would sometimes have files used shared by the kernel and userland.
- *Special filesystems:* Linux has some special filesystems which are not used for normal files.
    - `/proc` holds information about running processes. Other Unix-like systems sometimes provide /proc, but it is not as important as with Linux. Programs that use /proc often have trouble with ports to non-Linux systems.
    - `/sys` holds other information about the running kernel.
    - `/dev` is often a dynamic filesystem where device nodes appear and disappear dynamically, when modules load or unload or when devices are connected or disconnected. However, you can also just have `/dev` contain static device nodes like on other Unix-like systems.

Userland

- *GNU project:* The GNU (GNU's Not Unix) project created the shell, core commands, C compiler and build tools for a Unix-like system. However, their kernel, Hurd, never worked well, and thus GNU encourages users to use Linux kernel.
    - Nearly every Linux distribution uses these programs, which makes them "GNU slash Linux" distributions. But with the exception of the GNU C library and core commands, many GNU tools and applications are also found on other Unix-like systems. In fact, *BSD and Mac OS X, like Linux, depend on the GNU C compiler, assembler, and linker.
    - The GNU project always insists on providing their packages as free software, using licenses like the GNU General Public License.
    - The GNU project also has a connection to the desktop environment called GNOME (GNU Network Object Model Environment).
- */lib:* The directory `/lib` exists on Linux and holds libraries needed by `/bin` and `/sbin` because `/usr` might not be mounted.
- */usr:* Stuff provided by your Linux distro often becomes installed in `/usr` instead of `/usr/local`. This frees `/usr/local` for packages which your Linux distro does not provide.
- *Text editors:* Most distros provide the traditional Unix editors "vi" and "emacs", but some install *neither* by default. Even "ed" could be missing. Traditional Unix skills for text editing are not necessary when GNU "nano" is the installed non-X11 editor.
- *Package management:* Most distros provide a package management system such as "apt" (Advanced Package Tool), "rpm" (Red Hat Package Manager), or Portage. These automate the installation and upgrade of packages. Both application packages and base system packages use this system. The package manager is often the biggest difference between distros.
- *Graphical user interface:* Many of the popular Linux distros now boot the user straight into a desktop environment (GNOME or KDE, both based on X11 window system) immediately after installation. This is in contrast to some Linux distros and most *BSD distros which drop you at a root shell prompt, and expect you to

use the root shell prompt to install X11, GNOME or KDE if you want it.
- *Source distros:* Some Linux distros, such as Gentoo, emphasise building from source. Though every Linux distro with a compiler gives one the opportunity to rebuild the Linux kernel and other packages from source, these source distros provide scripts that not only automate the process, but help the system administrator make customisations and optimisations.

Unix Administration

Objective: To equip linux novice with essential administration skills to be proficient with the Linux environment.

System Administration

cron

- Definition: Cron is a scheduler that automatically executes a group of commands at certain time interval.
- Configuration files:
  - 1. /etc/crontab - Main configuration
  - 2. /etc/cron.d/ - User based configuarton

Links to more information

Wikibooks for Linux

Wikibooks has several other books on its Linux section of the computing bookshelf. Here is a sample:

- Linux Guide

In addition, some books have Linux chapters:

- Computer Programming has information on Linux Programming
- LPIC101 Certification: LPI Linux Certification/LPIC1 Exam 101/Linux Installation & Package Management
- Movie Making Manual has information on Linux in film production

External Links

- **The Linux Kernel Archives** [1]
- **The Linux Documentation Project** [2]
  - *Introduction to Linux: A Hands On Guide* [3]
- Linux Wikicity *a GFDL wiki*
- Linux Online [4] - "About the Linux Operating System" [5]
- GNU Project [6] - GNU Manuals Online [7]
- Linux.com [8] - "Introduction to Linux and Linux.com" [9]
- Introduction to Unix and GNU/Linux [10]
- LinuxForums.org - Linux forum community for support and more [11]
- LinuxQuestions.org - Where Linux users come for help [12]
  - LinuxQuestions.org Wiki *a CC-BY-SA and GFDL wiki, "share your knowledge"*
- Free Linux book downloads [13] (many Google ads)
- A guide to set up the server using Linux [14] (book for sale)

Distribution Related

- Distro Watch [15]
- Linux ISOs [16]

References

[1] http://kernel.org
[2] http://www.tldp.org
[3] http://www.tldp.org/LDP/intro-linux/html/index.html
[4] http://www.linux.org
[5] http://www.linux.org/info/index.html
[6] http://www.gnu.org
[7] http://www.gnu.org/manual/manual.html
[8] http://www.linux.com
[9] http://www.linux.com/article.pl?sid=02/03/09/1727250
[10] http://free-electrons.com/doc/unix_linux_introduction/img0.html
[11] http://www.linuxforums.org/
[12] http://www.linuxquestions.org/
[13] http://www.techbooksforfree.com/linux.shtml
[14] http://www.packtpub.com/linuxemail/book
[15] http://www.distrowatch.com
[16] http://linuxiso.org

BSD

Berkeley Software Distribution or **BSD** is a flavor of UNIX that was developed at the University of California, Berkeley. Today, the **BSD** or ***BSD** systems are free Unix-like operating systems based on the university's distribution. To honor this, these systems put BSD at the end of their names. The *BSD systems are DragonFly BSD, FreeBSD, NetBSD, and OpenBSD.

Contents

Distributions

- FreeBSD: *focuses on speed and performance.*
- Mac OS X: *Darwin and BSD*
- NetBSD: *focuses on portability*
- OpenBSD: *focuses on code correctness (and thus security) and easy to security*
- Historical: *BSD operating systems that have been unmaintained or abandoned.*

Other Wikibooks

- A Neutral Look at Operating Systems has a comparison of the *BSD systems.
- Mac OS X Tiger

Explanations

This part of the guide contains a list of various topics which *introduce* and *explain* the use and administration of Unix-like systems.

There is an attempt to organise the subpages into parts.

Part 1: Command Line

- Shell Prompt *introducing the shell, commands, and arguments*
- Quoting and Filename Expansion
- Pipes and Job Control
- Signals
- Choice of Shell
- Environment *environment variables and process limits*
- Connecting to Remote Unix *accessing remote shells and files*

Part 2: Common Programs

- Scheduling Jobs *at, cron*
- Introduction to Editors *available text editors*
- Text Utilities *cut, sort, sed, ...*
- awk
- bc *utility for arbitrary-precision arithmetic*
- ed and sed *text editors in command mode*
- roff *text formatting, often for manual pages*

Part 3: Administration

- Becoming Root *su and sudo for running commands as the root user*
- Installing Packages *binary packages from your OS-vendor*

Part 4: Hardware

- Determining Hardware *what hardware is my system using?*
- Filesystems and Swap *mounting disks, formatting and mounting disk images*
- Partitioning Disks *creating and editting partition maps*
- Compact Discs *playing audio, ripping with cdparanoia, burning with cdrecord*

Part 5: Networking

- Interprocess Communication *pipes, sockets, packets, local versus network*
- Real Networking Interfaces *Ethernet, PPP, wireless*
- Fake Networking Interfaces *loopback, bridges, encapsulation*
- This list needs more topics.

Part 6: Electronic Mail

- Mail Server Overview *Incoming Mail, Outgoing Mail, Local Mail, Client Connectivity*
- Mail Server Related Terminology

Part 7: Security

- Firewalls *How firewalls work and how to configure them*

Part 8: Beyond This Book

- Finding Additional Help *search engines and user groups*

Commands

The Unix command line is often considered difficult to learn. This book aims to help beginners by introducing various commands in lucid and simple language. Unlike most command references, this book is designed to be a self-study guide.

Contents

Section	Contents
Summary	
Getting Help	man info apropos whatis makewhatis
File System Utilities	ls mkdir cd pwd chroot cp mv rm touch df link ln unlink chown chmod mount
Finding Files	find whereis which locate
File Viewing	cat more less od head tail
File Editing	pico nano zile vi joe emacs
File Compression	gzip gunzip zcat gzcat tar pax bzip2 zip compress
File Analysing	file wc cksum stat
Multiuser Commands	who finger su
Self Information	whoami groups id tty
System Information	uptime uname dmesg free vmstat top df hostname
Networking	ifconf ifdown ifup
Process Management	nohup ps kill pgrep pidof killall
Devices	fuser lsof fstat
Kernel Commands	lsmod modprobe sysctl
Compress Commands	tar bunzip2
Miscellaneous	sync echo cal date time from mail clear PS1

Environment Variables

An **environment variable** is a setting normally inherited or declared when a shell is started. You can use shells to set variables; the syntax varies but Bourne shells use:

```
$ VARNAME="new value"
$ export VARNAME
or
$ export VARNAME="new value"
```

Each program started from that shell will have *VARNAME* set to *new value*. The names of environment variables are case-sensitive; by convention they are uppercase.

A **shell variable** is like an environment variable, except that it is not exported to new programs started from that shell. (You could export it, but normally you just write a shell initialisation script to set it in each shell.)

EDITOR

The editor program called by sudoedit, vipw, and other such programs when you tell them to edit a file.

Examples:

```
EDITOR=vi
```

```
EDITOR=emacs
```

Also see **VISUAL**.

HOME

The home directory of the user. Most programs use this shell variable to find your home, thus you can set this variable to override the setting in `/etc/passwd` for your home directory. This way, you can start programs that put dotfiles or other files in a different directory than your usual home directory.

In most shells, `~` refers to your home directory. In C shell, and some more recent versions of Bourne shell, `~tux` always refers to the home directory of user tux as specified in `/etc/passwd`, while `~` (without a username after it) always refers to the value of HOME, even if it differs from your home directory in `/etc/passwd`.

LOGNAME

The name of the user. This is an easy way for a user to get own username. However, programs must not trust this variable because it can be set to an arbitrary value.

Both LOGNAME and USER should be set to the username.

Examples:

```
LOGNAME=tux
```

```
LOGNAME=sudhir
```

MAIL

The location of incoming local email. When **mail** or another local email reader inherits this environment variable, it uses this variable to find the inbox.

Some users do not have email at their local Unix box, but instead use the Internet to access their mail server, in which case the MAIL environment variable is irrelevant.

Many users do not have MAIL set, in which case the email reader uses the default setting. The default value for user "tux" would be `/var/mail/tux`, which is where many systems deliver mail.

MAILCHECK

This is a shell variable, not normally exported as an environment variable.

The frequency for which "bash" checks and alerts you for new local email.

MANPATH

The path used by the man(1) command to search for manual pages. The MANPATH environment variable is formatted with ':' separators just like the PATH environment variable.

PAGER

The pager called by man and other such programs when you tell them to view a file.

Examples:

```
PAGER=less
```

```
PAGER=more
```

PATH

A space or colon separated list of directories in which the shell searches for executables when a command is run without an absolute path. For example **ls** doesn't have an absolute path, but **/bin/ls** does).

Some systems set PATH using the system shell initialization files, such as `/etc/profile` for Bourne shells. Some systems set PATH before this as part of the login procedure, for example in `/etc/login.conf` for OpenBSD systems. For example, a Linux box could set the PATH at login, then add `/usr/X11R6/bin` to the path using `/etc/profile`, then add `/home/ambler/bin` to the path using `~/.bash_profile`.

The system boot scripts also set PATH. On some Linux boxes, the first command to set the path would seem to be in /etc/rc.d/rc.sysinit, which is one of the shell scripts invoked by the init process (inittab).

Examples:

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/home/puffy/bin
```

If this PATH is set and you type the shell command

```
$ uname -r
```

then the shell searches for the "uname" executable program. First it searches in /bin, then /sbin, then /usr/bin. If /bin/uname is an executable (which it should be), then the shell stops searching and runs it. If /home/puffy/bin/uname also is executable, it is not run, because the search never reached that directory.

In most cases, you only append to the PATH shell variable before exporting it to the PATH environment variable. To delete a directory from your search path, you must reassign the entire PATH variable to a new, shorter string, and

often this takes a great deal of typing or some cut and paste operations.

PS1

This is a shell variable, not normally exported as an environment variable.

The **bash** and public domain **ksh** shells use this as the prompt string.

Things that can be put in the prompt string include `\h` (hostname), `\u` (username), `\w` (absolute pathname of working directory), `\W` (name of working directory w/o path), `\d` (date), `\t` (time).

On some Red Hat boxes, the primary prompt string is set in the `/etc/bashrc` file. The prompt is also set in `/etc/profile`, but the setting in bashrc seems to take precedence. A `~/.bashrc` file runs /etc/bashrc, which sets the prompt. Because every instance of "bash" runs `~/.bashrc`, the prompt also appears in X sessions started from a display manager such as "xdm".

On some Slackware boxes, the command line prompt is set in /etc/profile. The xterm and rxvt prompts are different. The prompt is not set for X sessions, but it would be if you write a `~/.bashrc` to do that. Prompts are shell variables set from shell initialisation scripts. They are not xterm settings set by X resources such as `/usr/X11R6/lib/X11/app-defaults/XTerm`.

- *A Practical Guide to Linux*, by Mark G. Sobell and published by Addison-Wesley (1998), has more information on prompt strings at page 331.

PS2

This is a Primary shell variable, not normally exported as an environment variable.

The **bash** and public domain **ksh** shells use this as a secondary prompt string.

USER

This variable should have the same setting and purpose as LOGNAME.

VISUAL

This variable is used to specify the "visual" - *screen-oriented* - editor. Generally you will want to set it to the same value as the EDITOR variable. Originally EDITOR would have be set to **ed** (a line-based editor) and VISUAL would've been set to **vi** (a screen-based editor). These days, you're unlikely to ever find yourself using a teletype as your terminal, so there is no need to choose different editors for the two. Nevertheless, it is useful to have both set: Many programs, including less and crontab, will invoke VISUAL to edit a file, falling back to EDITOR if VISUAL is not set - but others invoke EDITOR directly.

Examples:

```
VISUAL=mg
```

```
VISUAL=vi
```

See also **EDITOR**.

References

- environ(7) manual page (FreeBSD [1], NetBSD [2], OpenBSD [3])
- Sobell, Mark G. (1998), *A Practical Guide to Linux*. Addison-Wesley
- SSC (2000), "Bash Reference Card", http://www.digilife.be/quickreferences/QRC/Bash%20Quick%20Reference.pdf (updated link)

References

[1] http://www.freebsd.org/cgi/man.cgi?query=environ&sektion=7
[2] http://netbsd.gw.com/cgi-bin/man-cgi?environ+7
[3] http://www.openbsd.org/cgi-bin/man.cgi?query=environ&sektion=7

Files

/etc/

/etc/fstab

The **fstab** (for *file systems table*) file is commonly found on Unix and Unix-like systems and is part of the system configuration. The fstab file typically lists all used disks and disk partitions, and indicates how they are to be used or otherwise integrated into the overall system's file system.

Traditionally, the fstab was only read by programs, and not written to. However, more modern system administration tools can automatically build and edit fstab, or act as graphical editors for it. It is the duty of the system administrator to properly create and maintain this file.

The file may have other names on a given Unix variant; for example, it is `/etc/vfstab` on Solaris.

Example

The following is an example of a fstab file on a Red Hat Linux system:

```
# device name             mount point             fs-type      options            dump-freq pass-num
LABEL=/                   /                       ext3    defaults        1 1
none                      /dev/pts                devpts  gid=5,mode=620  0 0
none                      /proc                   proc    defaults        0 0
none                      /dev/shm                tmpfs   defaults        0 0

# my removable media
/dev/cdrom                /mnt/cdrom              udf,iso9660 noauto,owner,kudzu,ro 0 0
/dev/fd0                  /mnt/floppy             auto    noauto,owner,kudzu 0 0

# my NTFS Windows XP partition
/dev/hda1                 /mnt/WinXP              ntfs    ro,defaults     0 0

/dev/hda6                 swap                    swap    defaults        0 0

# my files partition shared by windows and linux
/dev/hda7                 /mnt/shared             vfat    umask=000       0 0
```

(kudzu is an option specific to Red Hat and Fedora Core)

The first column indicates the device name or other means of locating the partition or data source. The second column indicates where the data is to be attached to the filesystem. The third column indicates the filesystem type, or algorithm to use to interpret the filesystem. The fourth column gives options, including if the filesystem should be mounted at boot. The fifth column adjusts the archiving schedule for the partition (used by dump). The sixth column indicates the order in which the fsck utility will scan the partitions for errors when the computer powers on. A value of zero in either of the last 2 columns disables the corresponding feature (http://www.humbug.org.au/talks/fstab/fstab_structure.html).

To get more information about the fstab file you can read the man page about it.

The Kfstab graphical configuration utility is available for KDE for editing fstab.

See also

- mtab

/etc/group

/etc/group stores the definitive list of the users groups and their members.

A typical entry is:

```
root::0:root,alice
```

It has four sections which going from left to right are,

- *(root)* The group name.
- *()* The group password in a hashed form. Normally not. When it is, it allows *any* user knowing the password access to the group, as such it *lessens* security. Group-passwords may be shadowed and stored in a separate file.
- *(0)* The unique id assigned to the group. Group ids below 10 are reserved for system use. Some unixs such as HP-UX reserve other groups numbers as well.
- *(root,alice)* The list of users who are members of that group.

/etc/passwd

/etc/passwd is the user authentication database, it contains a list of users and their associated internal user id numbers. Historically it also included passwords, however as this file needs to world readable (so all programs can use it to convert between username and user id) it is no longer considered secure to keep passwords in this file.

An entry in this file is of the form:

```
alice:*:134:20:Alice Monkey:/home/alice/:/bin/bash
```

It has seven sections which going from left to right are,

- *(alice)* The username.
- *(*)* The password in a hashed form. In modern systems a star indicates shadowing is in use and hence the password can be found in /etc/shadow/.
- *(134)* The unique id assigned to the user. Some unique ids have special purposes. For example the user id 0 is used for the root user.
- *(20)* The group that the user is assigned to upon login.
- *(Alice Monkey)* The GCOS field, can be used for anything or left blank. Normally used for personal information abou the user such as full name.
- *(/home/alice/)* The home directory of the user.
- *(/bin/bash)* The users default shell.

/etc/profile

/etc/profile contains the system default settings for users who login using the Bourne shell, "/bin/sh". When these users login, the Bourne shell runs the commands in this file before giving the shell prompt to the user. Most of these commands are variable assignments which configure the behavior of the shell.

Some Bourne-compatible shells also use this file, but other shells, such as the C shell, do not.

/etc/shadow

/etc/shadow contains the passwords for users in systems which use shadowing.

```
alice:43SrweDe3F:621:5:30:10:100:900:
```

The sections are:

- *(alice)* The username.
- *(43SrweDe3F)* The password in hashed form.
- *(621)* date of last password change.
- *(5)* the mimimum number of days before the password may be changed.
- *(30)* the maximum number of days before the user is forced to change their password.
- *(10)* the number of days after which a user is advised to change their password.
- *(100)* the maximum number of days an account can be inactive for before it is suspeneded.
- *(900)* the date the account will expire, if left blank the account will remain indefinitely. Most often used for the purpose of temporary accounts.

/etc/sysctl.conf

/etc/sysctl.conf configures the behavior of the running Unix kernel. During system boot, the scripts read this file and use "sysctl" to set the parameters shown in the file. Changing the file has no effect before the next reboot.

Files to be merged in to the list

- /etc/aliases - file containing aliases used by sendmail and other MTAs (mail transport agents). After updating this file, it is necessary to run the newaliases utility for the changes to be passed to sendmail.
- /etc/bashrc - system-wide default functions and aliases for the bash shell
- /etc/conf.modules - aliases and options for configurable modules
- /etc/crontab - shell script to run different commands periodically (hourly, daily, weekly, monthly, etc.)
- /etc/DIR_COLORS - used to store colors for different file types when using ls command. The dircolors command uses this file when there is not a .dir_colors file in the user's home directory. Used in conjunction with the eval command (see below).
- /etc/exports - specifies hosts to which file systems can be exported using NFS. Man exports contains information on how to set up this file for remote users.
- /etc/fstab - contains information on partitions and filesystems used by system to mount different partitions and devices on the directory tree
- /etc/HOSTNAME - stores the name of the host computer
- /etc/hosts - contains a list of host names and absolute IP addresses.
- /etc/hosts.allow - hosts allowed (by the tcpd daemon) to access Internet services
- /etc/hosts.deny - hosts forbidden (by the tcpd daemon) to access Internet services
- /etc/group - similar to /etc/passwd but for groups
- /etc/inetd.conf - configures the inetd daemon to tell it what TCP/IP services to provide (which daemons to load at boot time). A good start to securing a Linux box is to turn off these services unless they are necessary.

- /etc/inittab - runs different programs and processes on startup. This is typically the program which is responsible for, among other things, setting the default runlevel, running the rc.sysinit script contained in /etc/rc.d, setting up virtual login terminals, bringing down the system in an orderly fashion in response to `[Ctrl][Alt][Del]`, running the rc script in /etc/rc.d, and running xdm for a graphical login prompt (only if the default runlevel is set for a graphical login).
- /etc/issue - pre-login message. This is often overwritten by the /etc/rc.d/rc.S script (in Slackware) or by the /etc/rc.d/rc.local script (in Mandrake and Red Hat, and perhaps other rpm-based distributions). The relevant lines should be commented out (or changed) in these scripts if a custom pre-login message is desired.
- /etc/lilo.conf - configuration file for lilo boot loader
- /etc/motd - message of the day file, printed immediately after login. This is often overwritten by /etc/rc.d/rc.S (Slackware) or /etc/rc.d/rc.local (Mandrake/Red Hat) on startup. See the remarks in connection with /etc/issue.
- /etc/mtab - shows currently mounted devices and partitions and their status
- /etc/passwd - contains passwords and other information concerning users who are registered to use the system. For obvious security reasons, this is readable only by root. It can be modified by root directly, but it is preferable to use a configuration utility such as passwd to make the changes. A corrupt /etc/passwd file can easily render a Linux box unusable.
- /etc/printcap - shows the setup of printers
- /etc/profile - sets system-wide defaults for bash shell. It is this file in Slackware that sets up the DIR_COLORS environment variable for the color ls command. Also sets up other system-wide environment variables.
- /etc/resolv.conf - contains a list of domain name servers used by the local machine
- /etc/securetty - contains a list of terminals on which root can login. For security reasons, this should not include dialup terminals.
- /etc/termcap - ASCII database defining the capabilities and characteristics of different consoles, terminals, and printers
- /etc/X11/XF86Config - X configuration file. The location in Slackware is /etc/XF86Config.

/proc/

- /proc/cpuinfo - cpu information
- /proc/filesystems - prints filesystems currently in use
- /proc/interrupts - prints interrupts currently in use
- /proc/ioports - contains a list of the i/o addresses used by various devices connected to the computer
- /proc/kcore - The command ls -l /proc/kcore will give the amount of RAM on the computer. It's also possible to use the free command to get the same information (and more).
- /proc/version - prints Linux version and other info

/var/

- /var/log/messages - used by syslog daemon to store kernel boot-time messages
- /var/log/lastlog - used by system to store information about last boot (can be read with **lastlog**)
- /var/log/wtmp - contains binary data indicating login times and duration for each user on system (can be read with **last**)
- /var/log/btmp - contains binary data indicating *failed* attempts to login (can be read with **lastb**)
- /var/run/utmp - contains binary data indicating *currently logged-in* users (can be read with **who**)

/boot/

- /boot/vmlinuz - the typical location and name of the Linux kernel. In the Slackware distribution, the kernel is located at /vmlinuz.
- /boot/grub/menu.lst - has configuration settings for users of the GRUB bootloader for the available kernels and OS's which can continue the boot process.

/dev/

/dev/cdrom

/dev/cdrom is not an actual device, but on many systems it is a symbolic link to the actual CD device. For example, a Linux system with `/dev/hdb` for its floppy drive is likely to have a link `/dev/cdrom` which redirects to `/dev/hdb`.

/dev/fd*

At Linux, `/dev/fd0` is the first floppy disk drive at the system. Use `/dev/fd0H1440` to operate the first floppy drive in high density mode. Generally, this is invoked when formatting a floppy drive for a particular density. Slackware comes with drivers that allow for formatting a 3.5" diskette with up to 1.7MB of space. Red Hat and Mandrake do not contain these device driver files by default.

Likewise, `/dev/fd1` is the second floppy disk drive.

/dev/hd*

At Linux, `/dev/hda` is the first IDE hard drive. The second drive is either `/dev/hdb` or `/dev/hdc`, depending on the hardware configuration. Some IDE hardware allows up to four drives, including `/dev/hdd`.

Many machines have one hard drive (hda) and one cdrom drive (hdc on many machines, but hdb on some). Often, `/dev/cdrom` is a symbolic link to the cdrom drive.

Partitions are numbered from 1, like `/dev/hda1`, `/dev/hda2`, ...

/dev/null

/dev/null is a do-nothing device to use when one wants to ignore or delete program output. This file is useful when a program expects to save to a file, but you want not to save anything. This file can also be used as input to a program to represent an empty file.

There is no actual hardware associated with the `/dev/null` device.

Examples:

Deleting file called "x" (command **rm x**) sometimes causes an error, for example if the file does not exist:

```
$ rm x
rm: x: No such file or directory
```

One can hide the error by redirecting it to a file. By using `/dev/null` as the file, the error never saves to an actual file.

```
Bourne shell:
$ rm x > /dev/null 2>&1
```

In the Bourne shell, the "2>&1" redirects the standard error of "rm" (where the error appears) to standard output, then the ">" redirects the standard output to `/dev/null`.

One way to make an empty file called "y" is:

```
$ cat /dev/null > y
```

The "cat" command copies the file "/dev/null" to standard output, and the shell operator ">" redirects this output to "y". The "/dev/null" file seems empty when read, so the file "y" appears, but is also empty. (Note that in this case, simply "> y" will do the same thing.)

Dot files

There is some redundancy across these programs. For example, the look and behavior of emacs can be customized by usinng the .emacs file, but also by adding the appropriate modifications to the .Xdefaults file. Default versions of these files are often installed in users' home directories when the software packages that use them are installed. If a program doesn't find its configuration file in the user's home directory, it will often fall back on a system-wide default configuration file installed in one of the subdirectories that the package lives in.

- .bash_logout - file executed by bash shell on logout
- .bash_profile - initialization of bash shell run only on login. Bash looks first for a .bash_profile file when started as a login shell or with the -login option. If it does not find .bash_profile, it looks for .bash_login. If it doesn't find that, it looks for .profile. System-wide functions and aliases go in /etc/bashrc and default environment variables go in /etc/profile.
- .bashrc - initialization command run when bash shell starts up as a non-login shell
- .cshrc - initialization commands that are run automatically (like autoexec.bat) when C shell is initiated
- .emacs - configuration file for emacs editor
- .fvwmrc - configuration file for fvwm window manager
- .fvwm2rc - configuration file for fvwm2 window manager
- .jedrc - configuration file for the jed text editor
- .lessrc - typically contains key bindings for cursor movement with the less command
- .login - initialization file when user logs in
- .logout - commands run when user logs out
- .wm_style - gives choice of default window manager if one is not specified in startx
- .Xdefaults - sets up X resources for individual user. The behavior of many different application programs can be changed by modifying this file.
- .xinitrc - initialization file when running startx. Can be used to activate applications, run a given window manager, and modify the appearance of the root window.
- .xsession - configuration file for xdm

Directories

Different distributions have different directory structures, despite attempts at standardization such as the Linux Filesystem Hierarchy Standard (FHS) organization.

- /bin - essential UNIX commands such as ls, etc. Should contain all binaries needed to boot the system or run it in single-user mode
- /boot - files used during booting and possibly the kernel itself are stored here
- /dev - contains device files for various devices on system
- /etc - files used by subsystems such as networking, NFS, and mail. Includes tables of disks to mount, processes to run on startup, etc.
- /etc/profile.d - contains scripts that are run by /etc/profile upon login.
- /etc/rc.d - contains a number of shell scripts that are run on bootup at different run levels. There is also typically an rc.inet1 script to set up networking (in Slackwar), an rc.modules script to load modular device drivers, and an

rc.local script that can be edited to run commands desired by the administrator, along the lines of autoexec.bat in DOS.

- /etc/rc.d/init.d - contains most of the initialization scripts themselves on an rpm-based system.
- /etc/rc.d/rc*.d - where ``*'' is a number corresponding to the default run level. Contains files for services to be started and stopped at that run level. On rpm-based systems, these files are symbolic links to the initialization scripts themselves, which are in /etc/rc.d/init.d.
- /etc/skel - directory containing several example or skeleton initialization shells. Often contains subdirectories and files used to populate a new user's home directory.
- /etc/X11 - configuration files for the X Window system
- /home - home directories of individual users
- /lib - standard shared library files
- /lib/modules - modular device driver files, most with .o extensions
- /mnt - typical mount point for many user-mountable devices such as floppy drives, cd-rom readers, etc. Each device is mounted on a subdirectory of /mnt.
- /proc - virtual file system that provides a number of system statistics
- /root - home directory for root
- /sbin - location of binaries used for system administration, configuration, and monitoring
- /tmp - directory specifically designed for programs and users to store temporary files.
- /usr - directory containing a number of subdirectory with programs, libraries, documentation, etc.
- /usr/bin - contains most user commands. Should not contain binaries necessary for booting the system, which go in /bin. The /bin directory is generally located on the same disk partition as /, which is mounted in read-only mode during the boot process. Other filesystems are only mounted at a later stage during startup, so putting binaries essential for boot here is not a good idea.
- /usr/bin/X11 - most often a symbolic link to /usr/X11R6/bin, which contains executable binaries related to the X Window system
- /usr/doc - location of miscellaneous documentation, and the main location of program documentation files under Slackware
- /usr/include - standard location of include files used in C programs such as stdio.h
- /usr/info - primary location of the GNU info system files
- /usr/lib - standard library files such as libc.a. Searched by the linker when programs are compiled.
- /usr/lib/X11 - X Window system distribution
- /usr/local/bin - yet another place to look for comon executables
- /usr/man - location of manual page files
- /usr/sbin - other commands used by superuser for system administration
- /usr/share - contains subdirectories where many installed programs have configuration, setup and auxiliary files
- /usr/share/doc - location of program documentation files under Mandrake and Red Hat
- /usr/src - location of source programs used to build system. Source code for programs of all types are often unpacked in this directory.
- /usr/src/linux - often a symbolic link to a subdirectory whose name corresponds to the exact version of the Linux kernel that is running. Contains the kernel sources.
- /var - administrative files such as log files, used by various utilities
- /var/log/packages - contains files, each of which has detailed information on an installed package in Slackware. The same file can also be found at /var/adm/packages, since the adm subdirectory is a symbolic link to log. Each package file contains a short description plus a list of all installed files.
- /var/log/scripts - package installation scripts in Slackware are stored here. You can inspect these scripts to see what special features are included in individual packages.
- /var/spool - temporary storage for files being printed, mail that has not yet been picked up, etc.

External link: Modified Directory Structure [1]

References

[1] http://markhobley.yi.org/mdirs/index.html

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on.

These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
>
> Permission is granted to copy, distribute and/or modify this document
>
> under the terms of the GNU Free Documentation License, Version 1.3
>
> or any later version published by the Free Software Foundation;
>
> with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
>
> A copy of the license is included in the section entitled "GNU
>
> Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the

Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

OpenBSD

This module contains information that is specific to OpenBSD.

Background

OpenBSD calls itself the "multi-platform, ultra-secure operating system". The OpenBSD team believes in strong security and code correctness. The OpenBSD team has a between six and twelve developers working on finding bugs and security holes in the system. OpenBSD also strives to be secure by default, meaning that the user does not have to be a security expert to secure the system. The OpenBSD project normally makes a new version every six months.

OpenBSD is known for having strict rules regarding bugs. Such as an application, not having a manual is considered a bug and therefore the application will not be included in the port (or package) tree. These rules are also included in the operating system. Should an application pose a security risk OpenBSD will kill the process on the spot. If there is a bug, and it is announced on the mailing list, then the OpenBSD team will come out with a patch in a matter of days. OpenBSD has only had two remotly exploitable bugs (both were in ssh and soon there afterwards patched), after a default install (with no additional services turned on) in over 10 years.

Notable Security Features

- strlcpy() and strlcat()
- Memory protection purify
  - W^X
  - .rodata segment
  - Guard pages
  - Randomized malloc()
  - Randomized mmap()
  - atexit() and stdio protection
- Privilege separation
- Privilege revocation
- Chroot jailing
- New uids
- ProPolice

Installation

OpenBSD has a easy although non-graphical installer.

Customization

You can change the default configuration files and install additional packages, by creating your own custom iso file.

As a firewall

OpenBSD uses **pf** ("packet filter") as a firewall. Though the authors originally contributed pf to OpenBSD, because it is free, other operating systems are including pf.

As a desktop

Despite is reputation for being only for servers OpenBSD can also serve as a great workstation/desktop. OpenBSD uses pkg_add as their binary package management system. The pkg_add automatically resolves dependencies. If you get the packages from ftp, pkg_add is able to resolve all of the dependencies for you. Some of the desktop related packages in the binary package system include: KDE (3.5), Xfce, Gnome, fluxbox, blackbox, and e16. Many other packages are available through the port system which also resolves and compiles the dependencies for you.

References

- "OpenBSD Security." 31 Oct. 2008. OpenBSD. 31 Oct. 2008 <http://www.openbsd.org/security.html>.

A Neutral Look at Operating Systems

The purpose of this book is to provide a neutral view of as many Operating Systems as possible. This book strives to provide solid information on Operating Systems without the ever-prevalent "distribution/Operating System bias".

Wikipedia has a comparison of operating systems.

Brief contents

1. Unix and derivatives
    1. Unix
    2. Solaris - Wikipedia:Solaris Operating Environment
    3. Berkeley Software Distribution (BSD)
    4. Linux
    5. Hurd - Wikipedia:GNU Hurd
    6. Plan 9 - Wikipedia:Plan 9 (operating system)
    7. Inferno - Wikipedia:Inferno (operating system)
    8. QNX - Wikipedia:QNX
2. CP/M and derivatives
    1. Control Program for Microcomputers
    2. DOS
    3. Microsoft Windows
    4. OS/2
3. RISC OS - Wikipedia:RISC OS
4. EROS - Wikipedia:EROS

5. AmigaOS - Wikipedia:AmigaOS
    1. Aros - Wikipedia:Aros
    2. AmigaOS 4.x - Wikipedia:AmigaOS 4
    3. MorphOS - Wikipedia:MorphOS
6. BeOS - Wikipedia:BeOS
7. Mac OS

Extended contents

This lists the operating system variants contained in this book.

1. Unix derivatives
    1. Unix
    2. Solaris - Wikipedia:Solaris Operating Environment
    3. Berkeley Software Distribution
        1. DragonFly BSD
        2. FreeBSD
            1. PC-BSD
            2. DesktopBSD
        3. NetBSD
        4. OpenBSD
            1. MirOS BSD
        5. Quasijarus
    4. Linux
        1. Live CDs
            1. Knoppix
            2. Fedora
            3. Ubuntu
        2. Distributions for New Users
            1. Fedora Core
            2. Mandriva Linux
            3. Puppy Linux
            4. Ubuntu Linux
            5. SUSE
        3. Distributions for Power Users
            1. Debian GNU/Linux
            2. Slackware
        4. Distributions for Masochists (just kidding)
            1. Gentoo
            2. Linux From Scratch
            3. SourceMage
    5. Hurd - Wikipedia:GNU Hurd
    6. Plan 9 - Wikipedia:Plan 9 (operating system)
    7. Inferno - Wikipedia:Inferno (operating system)
    8. QNX - Wikipedia:QNX
2. CP/M derivatives
    1. Control Program for Microcomputers

    2. DOS
        1. FreeDOS
    3. Microsoft Windows
        1. /Microsoft Windows#Windows 1, Windows 2, Windows 3, Windows 3.1, Windows 3.11, Windows 3.12
        2. Windows 95, Windows 98, Windows 98 SE, Windows Me
        3. Windows NT, Windows 2000, Windows XP, Windows 2003
        4. ReactOS
    4. OS/2
3. Mac OS
    1. Mac OS to version 9
    2. Mac OS X
4. RISC OS - Wikipedia:RISC OS
5. EROS - Wikipedia:EROS
6. AmigaOS - Wikipedia:AmigaOS
    1. AmigaOS 1.0, AmigaOS 1.1, AmigaOS 1.2, AmigaOS 1.3
    2. AmigaOS 1.4
    3. AmigaOS 2.0, AmigaOS 2.05, AmigaOS 2.1
    4. AmigaOS 3.0, AmigaOS 3.1
    5. AmigaOS 3.5, AmigaOS 3.9
        1. Aros
        2. AmigaOS 4.x - Wikipedia:AmigaOS 4
        3. MorphOS - Wikipedia:MorphOS
7. BeOS - Wikipedia:BeOS

Authors

- Kernigh, expanded Berkeley Software Distribution chapter
- aGGreSSor, expanded AmigaOS chapter
- Other and anonymous contributors

Further reading

- "A Neutral Look at Operating Systems" generally looks at OSes designed for desktop and laptop personal computers, which are less than 2% of all the computers in the world.[1] For OSes designed to run on the other 98% of all the computers in the world, see Embedded Systems/Common RTOS.

[1] "The Two Percent Solution" (http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900861) by Jim Turley 2002

UNIX Computing Security

UNIX Computing Security describes the practical security aspects of UNIX and UNIX-like systems, and how the available defenses can be applied to restrict unauthorized access or use of a system. This book is intended for readers familiar with the basics of UNIX.

Contents

Bourne Shell Scripting

Hi there! Welcome to this Wikibook on the wonderful world of Bourne Shell Scripting!

This book will cover the practical aspects of using and interacting with the Bourne Shell, the root of all shells in use in the world. That includes interacting with the shell on a day-to-day basis for the purposes of operating the computer in normal tasks, as well as grouping together commands in files (scripts) which can be run over and over again. Since it's not practical to talk about the Bourne Shell in complete isolation, this will also mean some short jaunts into the wondrous world of Unix; not far, just enough to understand what is going on and be able to make full use of the shell's very wide capabilities.

There are also some things this book **won't** do for you. This book is not an in-depth tutorial on any kind of programming theory -- you won't learn the finer points of program construction and derivation or the mathematical backings of program development here. This book also won't teach you about or any other type of Unix or Unix itself or any other operating system any more than is necessary to teach you how to use the shell. Nothing to be found here about , joe, vi, or any other specific program. Nor will we cover firewalls and networking.

We *will* cover the Bourne Shell, beginning with the basic functionality and capabilities as they existed in the initial release, through to the added functionality specified by the international POSIX standard POSIX 1003.1 for this shell. We will have to give you *some* programming knowledge, but we hope that everyone will readily understand the few simple concepts we explain.

Having said that, the authors hope you will find this book a valuable resource for learning to use the shell and for using the shell on a regular basis. And that you might even have some fun along the way.

Chapters

Authors

1. BenTels, started the book
2. Kernigh, added Substitution and Loops chapters
3. Quick reference was originally by Greg Goebel and was from http://www.vectorsite.net/tsshell.html (was public domain licensed) and was partly wikified by unforgettableid.
4. Other and anonymous contributors

External References

- IEEE Std 1003.1, 2004 Edition [1] - The 2004 Edition of IEEE/POSIX standard 1003.1 (one-time registration required).
- An Introduction to the Unix Shell [2] - HTML format republication of Steve Bourne's original tutorial for the Bourne Shell.
- UNIX Shell Script Tutorials & Reference [3]
- **Beginner**
  - BASH Programming - Introduction HOWTO [4]
  - Linux Shell Scripting Tutorial - A Beginners handbook [5]
- **Advanced scripting guide**
  - Advanced Bash-Scripting Guide [6]
- **Print**
  - UNIX IN A NUTSHELL: A Desktop Quick Reference for System V & Solaris 2.0 (2^{nd} edition)
    Daniel Gilly et al.
    August 1994
    ISBN 1-56592-001-5

References

[1] http://www.unix.org/version3/online.html
[2] http://steve-parker.org/sh/bourne.shtml
[3] http://www.injunea.demon.co.uk/pages/page201.htm
[4] http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
[5] http://bash.cyberciti.biz/guide/Main_Page
[6] http://tldp.org/LDP/abs/html/index.html

Learning the vi Editor

```
This book aims to teach you how to use the vi
editor, common to many Unix and Unix-like operating systems.
~
~
~
~
~
"Learning_the_vi_editor" [New file].
```

The above text is a little example of how the vi editor's screen looks.

Contents

Other sources of information:

- http://www.texteditors.org
- http://thomer.com/vi/vi.html
- An introduction to the vi editor [1]
- Functions of VI editor [2]

Acknowledgements

References

[1] http://planzero.org/tutorials/introduction_to_the_vi_editor.php
[2] http://blog.eukhost.com/2006/10/14/fuctions-of-vi-editorlinux

FLOSS Concept Booklet

Reading not your thing? Much of the content below is explained in audio and video here [1].

Intro / Concept

What does FLOSS stand for?

FLOSS stands for "Free/Libre Open-Source Software".

Legally, Free Software and Open-Source take quite different attitudes to sharing source code and what obligations those who share legally require. The different attitudes are a product of political ideology and cannot be easily reconciled - though they can be neutrally explained (only) by referring to the legal concepts of 'share-alike' and a 'consortium'. The term FLOSS emerged to simply avoid having to discuss or define these ideas to ordinary users who want to know about the implications for end users. Which do not normally include the legal negotiations between contributors.

What is free software?

Free software is software that anyone is free to use, copy, improve, examine or distribute, either free of cost or for a price. More precisely, it refers to four fundamental freedoms, which users of the software should have:

- Users should be able to run the software for any purpose. (freedom 0 — many things in computers start at 0)
- Users should be able to closely examine and study the software and should be able to freely modify and improve it to suit their needs better. (freedom 1)
- Users should be able to give copies of the software to other people for whom the software will be useful, either gratis or for a fee. (freedom 2)
- Users should be able to improve the software and freely distribute their improvements to the broader public so that they, as a whole, benefit. (freedom 3)

There is nothing new or special about this. This is how software used to be developed in the early 'Big Iron' days of companies which made money by selling hardware. But then, software companies came in, they started changing the rules of the game. They saw software only as a means of making money rather than as a means of making life easier, and with the advent of the 'Shrink-Wrap-License' even the need to provide a properly tested program was obviated.

Legally, free software is defined by a license to use and distribute the software that guarantees all four of these freedoms. Any user who grants all of these freedoms to other users retains their access to the software. In strict legal terms, the software is share-alike but only to those users who uphold these obligations. All others are disqualified and may lose their right to use, extend, distribute or derive new works from it.

It isn't open content, which has its own rules

Some related share-alike movements do not accept the so-called freedom 0 - it remains highly controversial. Accordingly those movements are neither free software nor open-source. They are part of the broader open content movement however. A typical license that restricts use by certain people or groups but permits it under share-alike terms to others is CC-by-nc-sa (the Creative Commons attribution-non-commercial-share-alike license) that is extremely popular among content and code developers that do not wish to grant commercial entities the power to build on their work without paying.

As a user, why would I want to examine and modify my software?

What is important is not that you modify or view the sources, but that you cannot be prevented from doing so or having it done (for you) and are not dependent on a particular person or entity to do it. Not to mention the fact that companies do go out of business - taking their proprietary products down with them - sometimes leaving the users who relied on them without any means of supporting some critical product.

As technology evolves, hardware, software and users' requirements change. And software, being a tool to make life easier, too has to be subject to easy and quick modification. So, even if you personally cannot change or modify the software, you want to be sure that you are not subject to monopoly power, or even simply the whims and fancies, of whoever created the software.

That apart, it is necessary to be able to examine the software, to see if it has malicious features. For example, to check whether the program is spying on you. One version of Windows was designed to report to Microsoft all the software on your hard disk. But Microsoft is not alone: the KaZaa music sharing software is designed so that KaZaa's business partner can rent out the use of your computer to their clients. You (or anyone else) need to be able to examine and modify your software to be able to protect yourself against such mistreatment. Even if you don't know what bad things to look for, someone who does will soon find this "bad thing" in the program and spread the word about it.

There are other reasons such as being able to fix bugs, and modify programs to your needs. These will be explained a little later.

Doesn't "free" mean that I do not have to pay for the software?

No. The word "free" has two meanings in the English language.

1. The "free" in "free beer", which refers to zero cost.
2. The "free" in "free speech" and "free market", which refers to freedom.

The free in free software refers to the freedoms that we've talked about above that people have. There's nothing in the definition of free software that says that you cannot sell it to someone for a price. Indeed, there are companies whose entire business model is centred around collecting, compiling and selling free software. However, since someone to whom free software is licensed is free to sell or give it away in turn, you can almost always easily find it openly (and legally) downloadable on the Internet or other places [2].

When you hear of "free software", think of liberty, freedom, and "free enterprise".

Well, what's not "free" about other kinds of software?

Most non-free software in the world today is not sold, it is licensed. From complex operating systems to tiny games or screen savers, the end users of the software have a license to use it under conditions laid out in an End User License Agreement. This agreement lists out the conditions under which the user can use the software – often restrictions are imposed on the use to which the software can be put. In almost all cases, users are explicitly prohibited from "taking the software apart" to study how it works, cannot modify or improve it, are only allowed to make a single copy of the software (for backup purposes) and are strictly prohibited from giving copies to other people.

What do you mean by "copyleft"? What's wrong with copyright? How is this different?

Legally, a copyleft is a share-alike clause in a license that requires certain conditions (or "freedoms") be upheld by the user, distributor or anyone basing a derivative work on the original. One of those conditions is to treat all other users, distributors and derived work authors equally under the conditions of the license. One of those equal conditions is the withdrawing of all rights under the license from anyone who does not do so.

Free software historically implemented copyleft by requiring all modified and extended versions of the program to be free software as well, using the copyright in the program (and associated rights to restrict use of it or derivations of it) as leverage. This did not deal with every problem. In particular, it did not prevent authors of derived works from filing a software patent on their improvements, nor anyone from building a web service on the original software and extending it by 'mashup' methods. *These deficiencies are supposed to be addressed in the GPL version 3.0.*

The original GPL, despite deficiencies, was an attempt to build a very minimal consortium among all users who would agree to its four freedoms. To understand this we must review the alternatives.

The simplest way to make a program free is to put it in the public domain, uncopyrighted. This allows people to share the program and their improvements, if they are so minded. But it also allows uncooperative people to convert the program into proprietary software. They can make changes, many or few, and distribute the result as a proprietary product. People who receive the program in that modified form do not have the freedom that the original author gave them; the middleman has stripped it away. Also the developers of free software will be forced to compete with improved versions of their own software (which open-source does permit).

Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. If they do not, they lose their rights. As Stallman puts it, "Copyleft guarantees that every user has freedom, and ensures that somebody does not remove the freedom from free software."

To copyleft a program, first state that it is copyrighted; then add distribution terms in the form of a license document – they comprise a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code or any program derived from it but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable.

According to the Free Software Foundation, "Proprietary software developers use copyright to take away the users' freedom; we use copyright to guarantee their freedom. That's why we reverse the name, changing 'copyright' into 'copyleft'." Historically however the term was simply a joke or pun added to a letter to Richard Stallman by Don Hopkins who had put "copyleft - all rights reversed" as one of many annotations.

The most popular copyleft license is the GNU General Public License [3].

What licenses exist to protect free software?

There are many licenses that make a piece of software free. But only some of them preserve user freedom aka copyleft. The non-copyleft license include X11, BSD, Artistic... The strongest copyleft and most widely used free software license is the GNU General Public License [3], or GNU GPL for short.

For a more complete list of licenses check out the GNU website [4].

Okay, so I can see that free software is legal but surely if I duplicate something that means that someone is losing out somewhere along the way?

We in the free software community feel that the harm caused by obstructing software use cannot be justified by the profit obtained through selling software. We use other means to earn money.

Also, chances are if someone didn't get the software for no price, they wouldn't have gotten it at all. Take all the people that get proprietary software illegally from peer-to-peer programs for example.

Contrary to your assumption that allowing distribution and modification causes loss, Richard Stallman lists three levels of material harm caused by restrictions on distribution and modification:

1. Fewer people use the program.
2. None of the users can adapt or fix the program.
3. Other developers cannot learn from the program, or base new work on it.

For a detailed analysis, check out the essays "Why Software Should Not Have Owners" [5] and "Why Software Should Be Free" [6] by Stallman.

This freedom with software programs is interesting. Can this be extended to other forms of information like books?

Yes. Most user manuals for free software programs, for example are released under either free or copyleft licenses. Already, much literature is available under permissive terms, such as the GNU Free Documentation License (GFDL) [7] and Creative Commons [8] Commercial-Attribution(-ShareAlike).

But unlike software, books and articles contains speech and personal opinion. And, personal credit for specific work is important. So, the benefit of unlimited modification is not always desirable in literature, science or other content. The open content [9] movement accordingly focuses more on share-alike clauses.

These clauses are not restricted to those free software or open-source require. They include for instance the Creative Commons NON-Commercial licenses which prevent use by commercial parties unless they negotiate a parallel commercial license (which may also be a share-alike license or may be more like a proprietary license). Other share-alike clauses have been proposed to require specific processes of dispute resolution, scientific method or journalism be followed by creators of derived works.

Like the original free software foundation, each such clause would create a global consortium relying mostly on the license to enforce its rules and using the leverage of rights to use or improve the content as the leverage.

FSF itself has recognized the difference between open content and free software. The GNU Free Documentation License [10] used by both Wikipedia and Wikibooks itself contains clauses that deny the right to use the content to anyone who does not credit authors or other contributors, but these are similar to the free software conditions. GFDL does clearly and explicitly permit commercial use and it supports annotations and "Invariant Sections" that can reliably qualify the origin or reliability of the material or summarize objections to it - though these are not used by Wikipedia or Wikibooks at present, they serve necessary purposes for those services that do not allow every user to edit.

History of Free Software

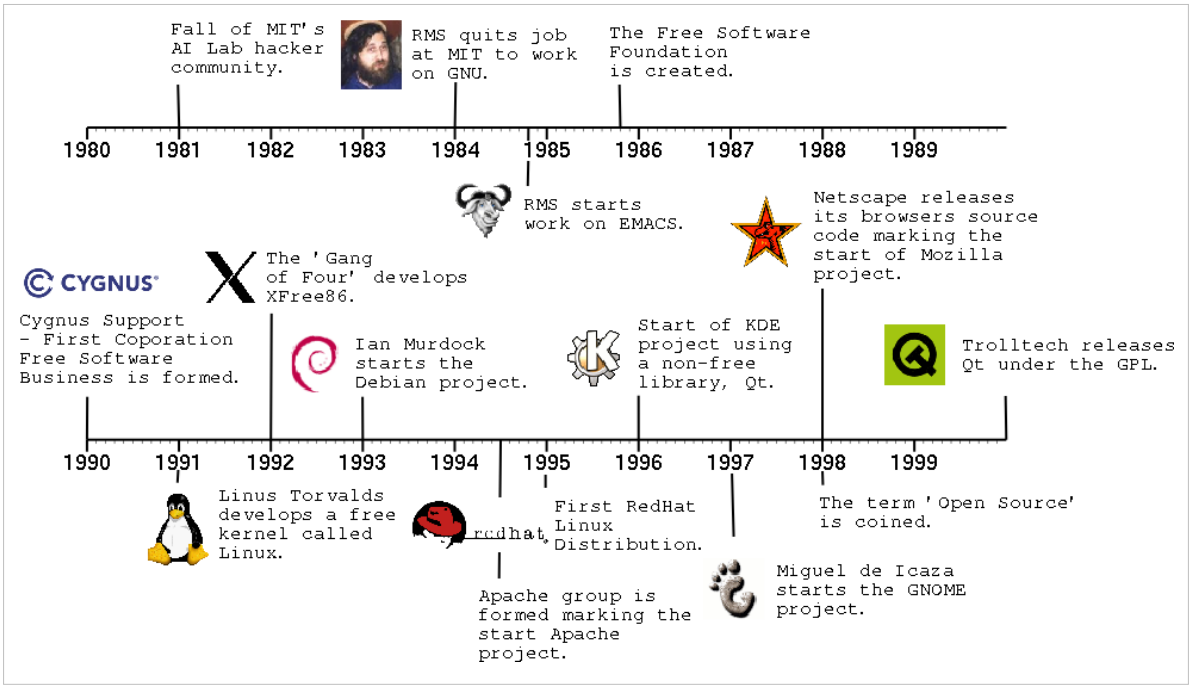
When did this whole free software thing start?

Richard Stallman started the GNU project and created the General Public Licence.

Software sharing is as old as computers. But the Free Software Movement traces its history to a software sharing community at the MIT Artificial Intelligence Lab. Richard Stallman became a member of this software sharing community when he joined the lab in 1971.

In the early 1980s due to a series of events, the hacker community collapsed. Richard Stallman was one of the few hackers left out in the lab and he was faced with a stark moral choice. He could accept the world has changed and start using proprietary software. Or he could create a free operating system which could recreate a community of cooperating hackers. In short, he could either change himself or change the world.

Stallman decided to change the world. And thus the GNU (pronounced Guh-new, it rhymes with canoe) project, a free replacement for the Unix operating system was born. In January 1984, Stallman quit his job at MIT and began writing GNU software. One of the first pieces of GNU software that Stallman wrote was the Emacs text editor. Slowly more hackers joined Stallman and started putting together a complete body of free software. This ranged from tiny programs like ls and cp to the huge packages like the GNU compiler collection and the Bash shell.



Okay, but now what is Linux? When did that come about?

Until 1991, most of the programs required to create a complete operating system had been created by the GNU project, except for one important piece - the kernel. In a nutshell, the kernel is software that provides secure access to a machine's hardware and decides what software get to use what hardware when. By then, a Finnish student named Linus Torvalds had written a free Unix compatible kernel called Linux (pronounced Lee-nux or Lin-ux). The kernel combined with the rest of GNU packages formed a complete usable operating system. Today this combination is called GNU/Linux, although it is often inaccurately called just Linux [11].

Open Source

What is this open source thing?

Open source is similar to free software. The biggest difference is that open source sees FLOSS as just a way of making better software, and doesn't value the freedom of the people.

How did open source start?

In 1998, some free software developers figured that by keeping quiet about ethics and freedom, and talking only about the practical benefits of free software, they might be able to spread free software better, especially to businesses. The term "open source" is offered as a way of doing this.

Why are open-source and free/libre software often grouped together?

Most (*not* all) open-source software can also qualify as free software, and vice versa.

What's better? Free Software or open-source software?

The two terms virtually always apply to the same software programs, but free software as a movement is better, because it values your freedom and offers a superior model for software development, whereas open-source only advocates a development model.

Production Methodology

How is Free Software actually made?

The same way other software is made. People sit on the computer keyboard and type in commands and code and compile them.

To properly understand how free software is made, we need to have a good idea how all software – free and proprietary - is made.

Follow this link to find more about *Who is doing it?* http://widi.berlios.de/paper/study.html

Okay, how is software made then?

A software program is simply a set of instructions to a computer to do something. Since the computer is a machine without any capacity to think for itself, it can only understand instructions written in a particular language – this format is called "object code". Unfortunately, to human beings, object code looks like gibberish. While they are creating software, humans use a particular format that they can easily understand called "source code". Source code uses letters, numbers and punctuation and, like any human language, can be understood by humans who learn to do so. So now we have source code, a format that programs are written in or created by humans but which looks like gibberish to a computer and object code – a set of instructions that a computer can understand but which look like gibberish to a human. A special program called a compiler transforms the source code into the object code. To reiterate, a human being writes down what it wants the computer to do in a format called source code. It's then translated into the only language that computers understand – object code. A special program called a compiler does this translation.

So now that I understand how software is made, how is free software made?

Remember, most humans cannot easily understand object code. Therefore, if a human wants to closely study, modify or improve a piece of software, they has to have access to the source code of the software. Since being free to examine, modify or improve software is central to the concept of Free Software, it follows that humans have to have access to the source code of a piece of software for it to be considered free.

Unlike proprietary software where only the original software creators (or those they explicitly provide access) have access to the source code, anyone who is interested can get access to the source code of free software. Therefore, if a user of free software wants to modify or improve it, she is free to do so. In many cases, the people who make the improvements make the improved software available to the broader public via the Internet. By definition and in practice, people who, in most cases, are connected to each other just through the Internet create free software collaboratively. One critical aspect of the creation of free software that is often overlooked is the feedback in the form of complaints and suggestions from normal users. This feedback is actively sought and many tools exist that make it easy for lay users to integrate these complaints, bug reports and suggestions into the production methodology.

So how is this different from the production of other kinds of software?

Typically, entities that produce non-free software usually have very tight restrictions on who has access to the source code of their programs and distribute their software only in object code format. The reason for this is that while it is very easy to compile source code into object code, it is very difficult to get the original source code from the object code. An analogy would be curds. While it's easy to make curds from milk, it's pretty much impossible to get milk from curds.

What kinds of people make free software?

Many people who write free software are volunteers, they probably have an unrelated day time job. These people spend their free time developing free software.

Commercial organizations that benefit from free software distribution or that provide free software support also develop free software by investing portions of their profit. An example of such an organization is RedHat.

There are many non-profit organizations that raise funds to develop free software, through donations from free software users. The Free Software Foundation is one such organization. Other examples are SPI, Gnome Foundation, Mozilla Foundation, and the like.

Some free software packages are developed by universities. The Festival text to speech engine, Octave - the Matlab clone are examples of software developed by universities.

Many commercial organizations also contribute to the development of free software, because these organizations benefit from the existing free software code base. For example IBM maintains the port of Linux to PowerPC CPUs, because it needs an OS for its CPU.

But I still don't understand why anyone would want to give away their work for free? What's in it for them!?

For love or for money.

Yes, people do make money by releasing software created by them as free software. Corporate bodies like MySQL and RedHat to name a few, make money because they release software created by them as free software (and offer support contracts for free software). And they do find that they are able to make more money than they ever would have made if they kept the software as non-free.

It's misleading to use the term "give away" to mean "distribute a program as free software". It implies the issue is price, not freedom. One way to avoid the confusion is to say "release as free software".

There are many reasons why people write free software. Some people might just want their work to spread out to the world. Many people would like to live in freedom. They contribute to free software so that they can continue to live in freedom. Some people write free software just for the fun of it. They love programming and hence use their programming skills to do something useful.

You might want to read Eric Raymond's paper titled Homesteading the Noosphere [12]. Eben Moglen in his paper titled Anarchism Triumphant: Free Software and the Death of Copyright [13] explains "Moglen's Metaphorical Corollary to Faraday's Law". This law also explains why people develop free software.

Arguments for using Free Software

I'm still not convinced. Surely a big computer company knows best when it comes to designing software? Why would I want to use software designed by an amateur?

Free software is NOT created by amateurs. The free software development process is open and transparent. If you want to include your code into a free software project, it will be scrutinised by several people. Amateur and or badly written code will be rejected outright.

Though your assumption that "big companies" are better at designing software is questionable, "big companies" are free to develop free software too. And they do develop free software. For example RedHat, Sun, Novell, and IBM, all develop free software.

If there are a group of people who would like to have a particular software written, they can bring together funds, hire a programming company, get the software written by professionals and then release it as free software. There is no reason why free software has to be written by amateurs.

Even if the free software designed by an amateur is inferior to the non-free software designed by a professional, you might want to use the free software because it gives you freedom, which is more important. Also, a better programmer who likes the idea could contribute to the project.

It is certainly not true that larger companies know more about designing software. They depend on a small group of programmers and are not generally in close contact with the users as in the case of free software. Thus, free software developers are often more aware of the needs of the users and their complaints about the existing versions. Some large companies allow technically incompetent marketing experts to make software design decisions that prioritize marketing requirements over user's needs and software soundness.

In any case, examinations of many free software applications show that today they are as good as or better than equivalent non-free applications, or are reaching there fast. And a large fraction of free software developers are not amateurs. And how do you know if the big company which sells non-free software did not hire an amateur programmer to write that code?

What we use depends on what we want. For computer users, software that can do the things they want done is a necessity. If such software does not exist, then they cannot do that particular kind of work. How well the work can be done, and how quickly, depends on the quality of the software and hardware available. It is, therefore, desirable to have software that enables the users to do the work with the least effort and to get the best possible output. These qualities of the software generally improve with time, the term 'standing on the shoulders of giants' applies to free software, so improvement begets improvement. Under non-free software it's always a matter of 'reinventing the wheel', or worse, to circumvent proprietary restrictions for another company or person to improve the code.

But more important than all these is the quality of freedom that the software has. If the software is restricted, and the company that makes it withholds all information about how the software is created and in what format the files are created, then the users become dependent on the company, and subject to exploitation. Because these are much more important in the larger context, it is important to use free software rather than non-free software.

But what about bugs? Surely free software is more likely to be virus prone?

The question involves two different types of computer related problems - bugs and viruses.

Bugs are unintentional errors in programs. With free software when you find a bug in the program, you have the freedom to exercise freedom two, the freedom to help yourself and correct the program. If you are not a programmer you are free to report the bug to the maintainer of the software or hire any programmer and correct the program. You are not under the mercy of any single organization. By submitting the bug fixes to the maintainer of the package the software package becomes better and better.

Virus is a malicious program that infects other programs by embedding a copy of itself in them. For more on these security risks, see the section below.

Since the source code is available, won't it be easy for someone to find out a security loop hole and exploit it?

Yes, but before the source gets to the hands of people who exploit loopholes, it passes through hands of people who develop free software. And they usually fix such loopholes.

And with free software, exploitation of a loophole is very, very, very quickly reported and fixed, often within hours.

Look at it this way, since the source is available, it would be easy for someone to find out a security loop hole, and make a patch for it, even before its exploited!

Also, the GNU operating system was designed with security in mind from the start.

So you're saying that free software actually evolves at a faster pace than non-free software?

Free software tends to evolve extremely rapidly. This is primarily because of the way it involves its users, who contribute bug reports and even code patches to the tool's development. The path and the pace of development is extremely open and only features and issues that are needed ever get done. Release fast and release often is the main 'mantra' of this kind of development.

Note however that this benefits only tools which have reached the stage of being in popular use. Once this happens it is in the interest of the users too to promote and actually contribute to the development. In a way this ensures only the deserving and useful tools gets supported in this manner.

Contrary to popular assumptions this model is laissez faire in the truest and most efficient manner as possible. Demand and supply works best with more suppliers and consumers. And this is what happens here.

The pace of evolution is actually controlled by the popularity and the usefulness of the software. The more popular a piece of software is, the faster it evolves. In fact if something hinders this pace in any manner, like a dis-interested maintainer or an abandoned company, often other people step up as developers of the project. Or sometimes, forks of the project spring up (not that they don't always at other times) to continue with the development, under another banner since the original source is free to be used or maintained by anyone.

Compared to the isolated model of development employed by proprietary software, this is a tremendous plus. There exists no question about the continuity of the tool either, which is the ultimate barrier in the path of software evolution.

This concept is explained really well in Eric Raymond's paper The Cathedral and the Bazaar [14].

Free software is only something used by computer enthusiasts, right?

Wrong.

Free software is used by people who value their freedom more than anything else.

Many millions of people the world over use free software without actually realizing it the moment they access popular sites on the Internet - see the section below.

Have any established organizations actually used free software to their advantage?

- This page is hosted on a server running free software (GNU/Linux, Apache/1.3.29 (Unix), PHP/4.3.4).
- More than 98% of all the Domain Name Servers, which identify the machine on which a page (like http://wikibooks.org/) is situated, use free software called BIND.
- More than 80% of all web servers use free software, called Apache, to serve their sites. But sadly, since Apache isn't copylefted, many of these servers run proprietary versions of Apache.

- More than 60% of all network servers run the Linux kernel, another free software.
- The TCP/IP implementation on most computers, (including those running non-free operating systems) is free software.
- "Digital Domain used 105 DEC Alphas running RedHat [GNU/]Linux to simulate and render water for James Cameron's *Titanic*" -- http://www.computer.org/computer/homepage/0202/ec/

Personal Relationship to Free Software

What kinds of problems might I expect to encounter using free software?

Exactly the same kind of problems you face with non free software.

Some problems with free software today are,

- Incoherence. Take manuals for example. Some free software comes with info manuals, some with man pages, some with html documents, some others with plain text files, and yet others with only source code comments!
- Inability to use certain patented algorithms. But that is not a problem with free software as such.
- Non-availability of drivers for certain devices, especially "WinModems."
- Free software cannot be legally used (at least in the US) for certain activities that involve copy-protection techniques. This includes playing encrypted DVDs.

Okay, but why would I want to modify my software anyway?

There are plenty of reasons. Say for example a software does not support your local language. You would like this software to be available in your local language so that you can use it. If the software is proprietary you will have to go and beg the "owner" of the software. If he finds making the change wouldn't be profitable, he will not make the change. With free software, you can make the change yourself or you can go to a programming company and ask them to make the changes for you (Which is how Richard Stallman supported himself for some time). With free software you are not helpless.

Also consider this ever plausible situation. Some time during your education, you would have written a project, or thesis or dissertation. In all probability, you may have used a computer belonging to a friend or the university to create this work. There is no assurance that you would get the same software or package which you used to create this file a few years from now, when in a nostalgic mood, you decide to go through your past work. This is a time when you will feel like modifying whatever is available to read a new file format.

If you are any kind of organisation which needs (or as often happens, is obliged by law) to retain certain information over a long time, the right to modify your software through a third party vendor is 'the' most important right.

Look, I'm no computer whiz! Isn't it easier for me to just use packaged software? Who do I turn to when something goes wrong?

Using free software is all the more important if you are not a computer expert. That way, you do not have to depend on the company from whom you purchased non-free software.

First, read the documentation that came with the software. If that doesn't help, try searching the web with a search engine like mozDex [15]. *Last* try one of the things below.

If you are looking for "free of cost" help, there will always be a free software user group or Linux User Group in your locality. Find one. Or ask any of the innumerable mailing lists or forums which provide support. You will be surprised at the response and support you receive. One forum is Nuxified.org [16]. There are more popular forums, however some don't set a good example - by having nonfree software (e.g. vBulletin) power their sites.

If you are willing to spend money, you can always hire a company or a consulting programmer to help you.

You can also buy support from expert companies. Since the software itself is open, the support vendor cannot lock you down like happens with proprietary software. You are always free to go to another support vendor.

Also if your requirement is pretty huge, it will be highly cost effective to have an in house software development and support team who will take the free software and customize it for your needs and constantly maintain it.

But how do I know I can trust someone not linked to a big company that has a reputation to uphold?

Several big companies do provide support for free software. Several big companies charge several times more than several individual or small companies for their services. Remember, in the free software scenario, you are paying only for the services, not the product or package. So, there will be immense savings, and companies or consultants who work on free software will always have less turnover, even if they are experts and market leaders in their field.

Plus a little-known company/person can gain a bad reputation.

Which GNU/Linux distribution (or distro for short) should I use?

There are many publicly available GNU/Linux distros. Distrowatch [17] maintains a list of distros with summaries of their features. Fedora Core [18] (formerly RedHat), OpenSuse [19] and Ubuntu [20] are popular ones for new users (listed in alphabetical order). There are also many more distros such as Damn Small Linux, Gentoo, Morphix, Slackware, etc.

Ubuntu [20] is a distribution based on Debian GNU/Linux but aimed at being easy to use. They have non-free software available, but it is not enabled by default. It has a Live CD available, which allows you to test the distro without installing it on your computer.

OpenSuse [19] is a powerful yet easy-to-use distro. Their Evaluation edition has non-free software, but the Open Source Software edition does not.

Fedora Core [18] is also new-user friendly.

Try Debian if you are good with computers and have a little working knowledge of the GNU/Linux system. The primary strength of Debian is the sheer number of packages. It also has support for many different architectures. Debian also has distributions which don't use Linux as the kernel such as Debian GNU/Hurd and Debian GNU/kFreeBSD. Use these only if you are really interested in developing free software. Be warned though, Debian has nonfree software in its package repositories that are suffixed with "-nonfree.

To avoid non-free software you should avoid Mandriva (formerly Mandrake), MEPIS, KNOPPIX, Linspire... These distributions come with non-free software packages by default.

If you are looking for a 100% free distribution, you should try gNewSense [21] or Trisquel [22].

Depending on your needs there are even specialised GNU/Linux distributions. There is one which plays only Video CDs. There are several which are used only as firewalls. Some distros run from one or two floppy disks. Some distros are intended to be used as rescue disks. There are also distros even for musicians and Geographical Information Systems (GIS). Most major distros come with several Compact Disks, but you can have a running and usable system with usually the first one or two CDs.

Where can I find free software?

Some good sources are:

- Free Software Directory [23]
- GNU Savannah [24]

NOTE: The next two also list non-free software, although most of it is free.

- SourceForge [25]
- FreshMeat [26]

I have some software for Windows that I can't find a replacement for on GNU/Linux! Where can I find one?

Look here [27] and here [28].

Okay, so how would I begin to install free software on my machine?

Once again, the same way you begin to install non-free software on your machine. You can buy computers with free software pre-installed. Or you can get a friend to do it for you. Or find a professional to do it.

If you decide to do it yourself, try lots of different software until you find what you like.

Most distributions have a built-in way of easily installing software and finding all its dependencies, called a package manager. Some package managers are:

- Smart [29] supports many types of repositories, including yum, apt, apt4rpm and urpmi.
- Yum [30] for Fedora Core
- Apt [31] for Debian
- Apt4RPM [32] for RPM distributions
- Portage [33] for Gentoo
- Ports [34] for FreeBSD

How can I make software I've written free software?

See the last section of the GNU GPL [35]. Be sure to write free documentation [36] (or get someone else to do it)!

How can I make money off of free software?

The easiest is selling free software. Be sure to explain to them that it's a matter of freedom, not price. Distributing this book with the software might be a good idea. Remember that if the software's under the GNU GPL and you distribute binaries, you must include the source with the binaries and/or offer the source separately.

You could also write free documentation [36] and sell that. The same things in the above paragraph applies here.

Another way is, if you can write software, put yourself for hire to add/change something in free software. If you are the maintainer of the software, you could have people pay you to make a(n) addition/change higher on the priority list for the project. This is how Richard Stallman used to support himself, and he's still here.

References

[1] http://audio-video.gnu.org/
[2] http://www.freedomtoaster.co.za/
[3] http://www.gnu.org/copyleft/gpl.html
[4] http://www.gnu.org/licenses/license-list.html
[5] http://www.gnu.org/philosophy/why-free.html
[6] http://www.gnu.org/philosophy/shouldbefree.html
[7] http://www.gnu.org/copyleft/fdl.html
[8] http://creativecommons.org
[9] http://en.wikipedia.org/wiki/open_content
[10] http://en.wikipedia.org/wiki/GFDL
[11] http://www.gnu.org/gnu/why-gnu-linux.html
[12] http://www.catb.org/~esr/writings/homesteading/homesteading/
[13] http://emoglen.law.columbia.edu/my_pubs/anarchism.html
[14] http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/
[15] http://www.mozdex.com/
[16] http://www.nuxified.org/forums/
[17] http://distrowatch.com/
[18] http://fedora.redhat.com/
[19] http://en.opensuse.org
[20] http://ubuntulinux.org
[21] http://www.gnewsense.org
[22] http://www.trisquel.info/en
[23] http://directory.fsf.org/
[24] http://savannah.gnu.org/
[25] http://sourceforge.net/
[26] http://freshmeat.net/
[27] https://help.ubuntu.com/community/FreeSoftwareAlternatives
[28] http://www.gnu.org/software/for-windows.html
[29] http://labix.org/smart
[30] http://fedora.redhat.com/docs/yum/
[31] http://www.debian.org/doc/manuals/apt-howto/index.en.html
[32] http://apt4rpm.sourceforge.net/
[33] http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=1
[34] http://www.freebsd.org/ports/
[35] http://www.gnu.org/copyleft/gpl.html#SEC4
[36] http://www.gnu.org/philosophy/free-doc.html

Open Source

This wikibook is intended to be an introduction to the Open source software movement. It is still getting organized.

Introduction

In the last few years the Open Source movement has changed the face of computing more than almost anyone would have thought possible. Many people in the computer industry believe that these changes are the most important changes in computing since IBM contracted Microsoft to write an operating system for their personal computers. This book is intended to discuss the history, philosophy, and legal issues relating to Open Source, as well as ways for software developers to get involved by selecting and joining projects they want to contribute to. More detail will be provided here than the Wikipedia articles on Open Source and Free Software, and information found in the Wikibook on Freeware will be enhanced and extended. Please feel free to add/correct information throughout the book. This book is **not** meant to be a listing of projects nor a guide to picking the best software to use (although some pointers are provided later in the book).

What is Open Source?

Open Source or *open-source software* is different from *proprietary* software. In Open Source, the source code used in the software is available to *anyone* to examine, evaluate, and adapt. Open source has had an important impact on the way many developers view and create software. End users often use the term open-source to cover a variety of *free and open source software*.

The Open Source Initiative has this for a definition of open-source software:

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale - with the exception of the Creative Commons Attribution Non-commercial versions of the standard Creative Commons licence.

Rationale: By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure

for cooperators to defect.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

Rationale: We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Rationale: The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

Rationale: Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.

Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

Rationale: In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process.

Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licensees of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Rationale: The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

Rationale: This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

Rationale: This clause forecloses yet another class of license traps.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

Rationale: Distributors of open-source software have the right to make their own choices about their own software.

Yes, the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Rationale: This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap" may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.

Free software is open-source software, but not all open-source software is free.

What is Free Software?

Free software is software that adheres to the concept of the four freedoms of software, as articulated by the GNU Project. These four freedoms are as follows:

- The user is free to use the software for any purpose.
- The user is free to study the mechanisms by which the program operates and to adjust these mechanisms to a specific purpose.
- The user is free to redistribute the software to another user.
- The user is free to adapt and improve the program and release these adaptations and improvements to the public.

Free Software shares much of its philosophy with Open-Source software, but many people within the open source community feel that there are important distinctions between the terms, as described in the section Free Software vs. Open-source software.

Often Free Software is referred to as "free as in speech, not as in beer", stressing the idea that the Free-software movement is concerned with *freedom*, not with *price*.

Free Software vs. Open Source software

The primary distinction of open source software is that it's not about freedom, it's about what software does things better.

This requirement is not true of all open source licenses. In this book we hope to provide enough information so that the reader can understand what the two are and make choices accordingly.

As a whole the movement is often called *"Free and Open-Source software"* (FOSS). While many people point to the differences between the two, this book will focus on what unites the two. For the purposes of clarity the abbreviation *FOSS* will be used when describing issues that apply to all open source software projects, whereas *OSS* will be used when describing issues that apply only to software with open-source licenses that allow future developers to close the source code, and *FSS* will be used when talking about issues that apply only to software with free licenses requiring future developers to maintain the previous style of licensing (if not the license itself).

Public Domain Software

Another common form of open-source software is software that has entered into the public domain. This is software that is unrestricted by copyright or licenses, and therefore free to use for any purpose. Before the rise of the open-source software movement a great deal of software written in academic circles was released into the public domain for peer-review. This practice has changed with the rise of a more intentional open-source movement, resulting in much software that was once public domain, now being released as FOSS. The public domain is where many resources used in open-source projects come from, but there are few major projects that operate totally within the public domain (Examples include SQLite).

FOSS does not mean no cost

This is a common misunderstanding about FOSS, in no small part because nearly all FOSS programs are available free of charge. For example when the text editor Emacs was first released Richard Stallman charged from time to time to get copies. Developers have the choice to charge under most FOSS licenses, although they rarely choose to. The only requirement to be a truly FOSS project is that the publisher provides the source code with the program, and to allow the user to edit that code.

On top of the initial cost of purchasing software, there are other ongoing costs associated with all software. This can come in the form of support agreements, the cost of customization, training costs support personnel and other sources. This is true of both traditional commercial software and FOSS programs. There is a large and active debate about which type of software is more expensive over the long run for large corporations, for individual users there is little to no question that FOSS is cheaper by far.

History

The history given here is not intended to be a complete, and perfect rendering of the history of the open source movement. Indeed it would be quite impossible to do so, as we are currently in the middle of the movement, and proper histories cannot be written about current events. Instead this chapter is intended to give the reader a general understanding of how the open source movement got started, so they can better understand the current debates.

In the beginning (1960's and 1970's) nearly all code was provided as a form of open-source. Since you had to get programs to run on many different machines, companies that wanted customers to be able to run their software they had to provide the source code to compile themselves. Frequently this code was not encumbered by licensing agreements, or those licenses were never enforced. Often the users would notice flaws in the programs, fix them, and provide the improvements back to the publisher at no charge. Unix user groups often shared code with each other as well, creating an even larger body of code to work from. At the time most companies believed that hardware would always generate far more profit than software.

At the same time when academics wrote programs as part of their research, they would often release the code into the public domain to others to learn from their work. The first email server, ftp server, and web server were all public domain projects that were created by academics and shared as public domain software.

Over time the hardware became more standardized. This allowed software publishers to tighten up on what segments of code they would allow their users to see and edit. This tightening started to frustrate users, especially academics, since they could no longer fix the problem they found, and from time to time found that they couldn't use the hardware or software they wanted to. This trend mostly continued in main stream software, strengthed by the rise of Microsoft.

During the late 1970's, many small businesses attempted to survive writing software for the new microcomputers. None were able to rely on hardware sales or after sale support contracts to subsidize software development. Few of these people, whether from an academic tradition of freely distributed software or not, were willing to make their products available without fee. Perhaps most famously, Bill Gates (a co-founder of Microsoft) realized that software, not hardware, held the prospect of being the greatest profit source in the new world of computing. When IBM needed an operating system for their personal computer, instead of writing their own, Microsoft licenced an early version of DOS, but retained the licensing rights for other hardware manufacturers. This choice changed the face of computing by creating a standard operating system that most personal computers would run on, it lead directly to Microsoft's dominance of personal computing, and later allowed them major access to the server side market as well. Microsoft has grown into the largest software producer in the world, competing in almost every segment of the market. There are many people that feel that the only way to compete with Microsoft is through open source projects which leverage a far larger support base than most companies can provide.

In this sea of activity, the FOSS movement primarily grew out of two places: the Berkeley Software Distribution license from the University of California and Richard Stallman's Free Software Foundation Gnu Public License.

BSD

During the 1970s, after Ken Thompson of Bell Labs spent a year teaching at the University of California at Berkeley, several research groups there began to develop software for the Unix operating system. And, as Bell Labs had no plans to port the system to the then new DEC VAX computer system, the Computer Science Research Group at Berkeley undertook the port. That port became very popular at other universities, and the CSRG began distribution of it, with their modifications and additions, under the name Berkeley Software Distribution. The researchers at Berkeley frequently found ways to improve or add to the existing versions of Unix. Bell Labs accepted some of the BSD changes, and passed on others. Over time the CSRG found ways to improve the basic aspects of Unix (eg, the file system). Concurrently, AT&T made improvements of their own, causing the two versions to move in different (although similar) directions. As development progressed on the Berkeley version of Unix, it grew further and further away from the work that AT&T was doing on their version (the last version of which was System V). In the early 1990s there was a major legal skirmish over the Berkeley Software Distribution (BSD) between AT&T, the University of California at Berkeley, and the Unix System Laboratories (USL), to which AT&T had turned over the marketing and development of Unix. The suit resulted in most of BSD being found to be unencumbered by copyright restrictions by those who controlled the original Bell Labs code. By the mid-1990s, the CSRG released 4.4BSD-Lite, Release 2 as their final product as they were closed down by the Regents of the University of California. The BSD releases were the root of the various versions of BSD that are common today, most notably FreeBSD, NetBSD and OpenBSD. Throughout its development, BSD was distributed in an open manner, resulting in contributions from many different contributors.

The Free Software Foundation and the GNU Project

In 1983 Richard Stallman (then of MIT) became frustrated with the growing commercialization of the computer development work that had been done at MIT, and with the increasing limitations imposed on software users. After a time, he began to create software that gave control to users. His vision was to create an entire operating system totally free of the restrictions being imposed by proprietary licensing. His first major software development effort (even before he became disenchanted with trends in the software industry) was the text editor, GNU Emacs. His next was a LISP system, a C Programming language compiler, and finally the GNU project.

In 1985 Stallman founded the Free Software Foundation (FSF) to help generate support for the GNU Project. FSF has grown into one of the most important organizations in the FOSS movement. While the primary FSF mission continues to be the completion of the GNU operating system, FSF has also taken on the role of *"free-software evangelist"* by protecting and supporting free software. FSF also holds the copyrights to much of the source code written for the GNU project, ensuring that it always remains freely availabe to users.

Over the next 10 years Stallman gathered a group of people together who essentially developed all of the core utilities found in Unix and the Unix-like operating systems. In the Unix spirit, these consist of hundreds (perhaps thousands) of small utility programs and tools. This project is still operating and is known as the GNU Project. Many of these programs have become standard on the BSD variants as well (see above). In 1994, Linus Torvalds released the first version of the Linux kernel, and when combined with the GNU utilities already available from the GNU Project, the Linux operating system came to be. The Debian distribution of Linux is called GNU/Linux, in recognition of Stallman's position that it is a joint production, to recognize that the GNU Project provides most of the essential utilities. The GNU Project continues its slow progress on its own operating system kernel, which will be known as Hurd. Hurd is intended to be offered as the official GNU replacement for the Unix kernel, though it is currently at a beta development stage.

While the FSF has become somewhat controversial, it has certainly had a major impact on computing. Millions of copies of the GNU software are used every day throughout the world.

For More Information

Much of the information for this article came from the following sources. They also contain more information about their areas of open source history if you are interested.

Wikipedia Articles:

- Free software article's history section

Outside Sources:

- Official GNU project History [1]
- History of BSD [2]

Philosophy

There is no one philosophy that brings people to open-source software. There are however two main branches of thought in which many people share some similar views and have helped create a strong movement. To understand the open-source movement as a whole you need to understand both of these main philosophies.

Software Freedom

An earlier software movement similar to Open Source is the Free Software Movement (embodied in the Free Software Foundation). Although there is some debate over its precise definition, there are generally three primary components of software freedom.

- **Freedom to Use**

Many believe that software should not have limitations on their private use, aside from restrictions on the creator's liability. In contrast, some commercial packages (like Microsoft Windows) disallow certain usages while a more expensive product allows such a use. According to US copyright law, this right is implicit unless limited by private contracts.

- **Freedom to modify**

Possibly more important than the freedom to use is the freedom to modify existing programs. Although certainly related to the overall open-source ideal, this freedom also implies that your modifications remain yours. Some open-source licences require the modifier to assign the rights to their work to the originator of the license (Microsoft's Shared Source program is generally considered one example). Obviously, very few completely commercial software packages give users this freedom. In US copyright law, this is an implicit exclusive right of the creator of a work. To give others this right, the creator must agree to do so.

- **Freedom to distribute**

A final key freedom is the ability to distribute software, modified or unmodified. Software packages that limit their distribution limit others freedom to do with it what they will. It can also prevent them from providing their improvement to the community. Similar to the freedom to modify, some open-source licences also limit the freedom to distribute, often requiring changes to be submitted back to some central repository. As with the right to modify, this is an exclusive right of the creator.

Notably absent from these freedoms is the requirement that free software be economically free. Several licences do not prevent merchants from selling the software whether modified or unmodified. In fact, several groups have historically charged for various pieces of software which are generally considered free software. Depending on the licence, the merchant may still be under other restrictions.

An example of a licence that provides software freedom with some restrictions on the distributer of the software is the GNU General Public Licence, or GPL. The GPL was written by Richard Stallman et al., for the GNU Project. The GPL has all of the above freedoms listed within. The most important difference from the above baseline is what the Creative Commons calls a "Share and Share Alike" clause. If you distribute a modification to a particular software package under the GPL, then you must agree that your portion of the work is also licenced by the GPL. For more information about the GPL there is more information in the licenses section of this book

Open Source Development Model

The second main branch of open-source philosophy revolves around the opportunity for a new software development model. Open-source is also about sharing ideas, and spreading the effort of creating software over a large number of interested developers. This sharing can lead to better software in shorter development times. It also allows for better feedback directly to the developers than is often present in traditional development models. Major companies like Apple and IBM have recognized that there can be a significant advantage to writing software in this way. Apple by way of the Darwin project and IBM through several significant donations of software and patents to open source groups have paved the way for other companies to seriously consider using this development model.

Licences and Patents

What Is A License?

In legal theory, a **license** is the permission or right granted to engage in some act without which the act might be otherwise unlawful. A license must be granted by a person, institution, or government that has the legal authority to do so. The term "license" also refers to the document that specifically describes these permissions and rights.

Important FOSS Licenses

While there are many open source licenses in use today, a few have emerged as more popular (and therefore more important) than most. We will take some time here to examine these important licenses in some detail. They are listed here in alphabetical order.

For an extensive list of open source licenses see the GNU Project's list of licenses [3], or The OSI list of licenses [4].

Apache

The Apache Software Foundation (ASF) is a non-profit corporation to support Apache software projects. Apache's projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license. Among the ASF's objectives are to provide legal protection to volunteers working on Apache projects, and to protect the Apache brand name from being used by other organizations.

There are currently two forms of the Apache in wide use today: versions 1.1 and 2.0. While 2.0 is considered by ASF to be the current version and most users have happily switched, a few large groups objected to new clauses inserted into 2.0 regarding patents, and have refused to use any software with the new license (most notable the OpenBSD project).

Code under Apache license can be linked with the proprietary code that is developed in the companies. Because of this the license is accepted inside the most of the companies. Companies time to time contribute to the code under Apache license to boost the activity of these projects. For instance, IBM and Intel have contributed big parts of the alternative java implementation to the Apache Harmony project.

BSD

BSD is both a license, and a class of licenses (generally referred to as BSD-like). The modified BSD license (in wide use today) is very similar to the license originally used for the BSD version of Unix (see BSD history above). The BSD license is a simple license that merely requires that all code licensed under the BSD license be licensed under the BSD license IF redistributed in source code format. BSD (unlike some other licenses) does not require that source code be distributed at all.

BSD is actually very close, though still distinct, from a public domain licence. Like works in the public domain, BSD-like licences permit nearly free modification and distribution of a work. There are only two significant ways in which BSD-like licences differ from public domain. First, the BSD licence carries the requirement that the licence, along with it's statement of copyright, is carried along with the work, or modified copies. This does not however limit code under different licences from being combined with it. Second, the BSD-like licences have a standard disclaimer of guarantees, such as fitness for a particular use and merchantability. This latter portion should not come as any surprise, since practically all software packages carry a clause nearly identical to this one.

In summary, the major difference between BSD-like licences and the public domain is that BSD licenced works must remain attributed to the original creator(s). Some BSD licences take this further, and require similar copyright statements in advertising materials for a distributed software product. Most however are simply limited this basic requirement.

For those who are opposed to the GPL for its limitations on modification and distribution, the BSD licence is often quite acceptable. Due to the inherent lack of limitations, BSD code can be incorporated into any other project, commercial or otherwise, removing a lot of the worries of licencing issues from code. Similarly, for those who endorse the licences such as the GPL for their ability to protect software freedoms, the BSD is looked upon as a step in the right direction, but not enough to guarantee said freedoms. They often encourage the use of more GPL like licences for this reason.

BSD is usually accepted inside the companies. The Apple Mac OS is based on the originally BSD - licensed operating system but with big parts of proprietary code.

GPL

The GNU General Public License is a free software license, created by the Free Software Foundation (FSF); version 2 was released in 1991. It is usually abbreviated to GNU GPL, or, simply, GPL. FSF recently announced that they have started work on version 3 of this license, and should be releasing the new version in the spring of 2007. The motivations for these changes have not been articulated yet, but based on a lecture by Richard Stallman held at New York's Cooper Union on November 29, 2005, the changes are related to the World Intellectual Property Organization's (WIPO) work on world copyright standardization.

The GPL protects the 3 main articles of software freedom as defined in the section on Software Freedom. The GPL also contains a clause requiring publishers of GPL licensed programs, that make modifications to the software, to distribute that modification under the GPL. This is something that many company managers do not like because they want proprietary control on all code that was written in the company. Even if the software is for internal use only and is never sold, the managers still want proprietary control to be sure that the software will not be used by a concurrent. Trying to create the negative attitude, GPL oponents gave GPL the label "viral", since it apparently "infects" any code which it touches.

From the other side, it is possible to say that GPL is tuned to support and even enforce the code sharing as much as possible. Companies and individuals that write code and release it under GPL get access to the large amount of the existing code under GPL. That way they are rewarded for picking this license and probably can afford to experiment with GPL compatible business models that otherwise bring less direct profit than proprietary control on the code.

It is not true that GPL is never accepted inside the companies: many companies have used software libraries under this license and did released all code that interacts with these libraries, as the license requires. This is how GNU gcc compiler have got C++ support, for instance. It is also true that GPL seems one of most popular licenses in the world. However the most of the companies will likely first try to contact the authors of the GPL-covered software, asking to sell the "commercial" license.

LGPL

An important corner-case of the GPL requires that only GPL code can link with with GPL libraries, even if the library is contained in a separate file, and the code which uses it contains no GPL code unto itself. This is seen by some as too strict, since such an action could be considered a "use" of the library, not as a modification. In order to allow for such a distinction, the LGPL was created. The LGPL fixes precisely this problem: If someone uses an LGPL work as a library which is symbolically linked with a project, the rest of the project does not have to be put under the LGPL. On the other hand, if any modifications are made to the LGPL'ed work which are distributed along with the software package as a whole, the changes must be made public, similar to the rules of the GPL. The "L" in the LGPL stands for different things for different people. Initially it was considered as a "Library" licence, since the clause specifically dealt with issues of linking. However, the GNU Foundation describes this as the "Lesser" GPL, since it gives fewer protections to the work than the GPL does. While the GNU Foundation does not frown upon the use of the LGPL, it encourages the use of the GPL when possible.

LGPL makes a clear difference between the code that has been covered by this license (usually software library) and the external code that is just linked to the LGPL covered code. The external code can stay proprietary. However modifications inside the LGPL covered library must be shared with public.

Companies usually find LGPL terms acceptable and it is common to find public libraries with this license inside the commercial software.

GPL + linking exception

GPL + linking exception license sounds like GPL but in additional chapters explicitly allow to link the code it covers together with any other code, regardless of its licensing terms. Hence while it sounds similar, GPL + exception in practice looks more like LGPL. This license has been recently used for several large projects, including java implementation by Sun Microsystems.

A brief note about copyrights and copylefts

Software licensing should not be confused with copyright protections. Open-source licensing explicitly relies on copyright law protections and no code licensed under most of those licenses are a release to the public domain. The author of a piece software can limit the use of what he created, since the code is a written work and subject to copyright law provisions. A creator can limit use of the software as desired, since copyright is a privilege of exclusive use. Copyright protections allow a copyright holder (not necessarily an author) to control use and distribution of software via licensing. That license may allow changes (as does the GPL) or prohibit them (as do most commercial EULA agreements).

The concept of copyleft also grew out of the work by GNU and others in the open source movement. While copyright is intended to most protect written works from unauthorized use, and most commercial copyright licenses do so by prohibiting changes or distribution, copyleft licenses encourage free and widespread use.

Note: Wikibooks is made available under the copyleft GNU Free Documentation License [7].

Patents

In the past few years the issue of software patents has become one of the most pressing to those in the open-source movement. In the United States and some other countries software is subject to patent protections. While only some other countries have placed similar restrictions on the production of software the patent system in the US has become problematic worldwide.

Economics of FOSS

Zero Marginal Cost

At the heart of the economics of FOSS is the zero marginal cost of goods in a digital environment. In this respect, the emergence of FOSS represents a confirmation of classical microeconomic price theory - that an equilibrium price in a perfect market approximates the marginal cost. From this perspective, FOSS can be understood as a pioneer in reaching what can be understood as an evolutionarily stable dynamic Nash equilibrium in truly free market.

Income-generation opportunities

While contributing time and effort in developing, enhancing and documenting FOSS does not provide any direct income, the development of expertise in FOSS provides a broad range of income-generating opportunities - from generating in-house savings from enhancements to FOSS to consulting opportunties in installing, training, customizing and the provision of technical support for FOSS installations. In Part I: The Networked Information Economy of *The Wealth of Networks: How Social Production Transforms Markets and Freedom* (Yale University

Press, 2006) [5], Yochai Benkler provides an excellent analysis - with IBM's stategy as a key example - of ways that income and wealth are being generated through open source and open content strategies.

Opportunities for the companies

Other possible ways of getting money from FOSS include:

- Dual license model, when the same code is provided both under aggressive FOSS licence (usually GPL) and "commercial" license that does not demand sharing of the derived works. The GPL version ensures wider user base, more popularity and may serve as an "evaluation version". This model is used by many companies, most notably by Trolltech with QT.
- Paid documentation model, when the software and convinceable examples are delivered as FOSS but some really good documentation is described in a book that must be bought, or in a separate documentation package which is for sale.
- Use of trademark. If the derived works cannot use the original trademark (logo, name) then the original work may be preferred by the user who is willing to pay for that one understands as a sign of high quality. This model is only possible if the trademark owner is already widely known as an expert in software development. It is successfully used by Red Hat.
- Use of the paid auto update service. In this case the paying user receives opportunity to get and install security and other important updates automatically. Non-paying user needs much more effort to keep the system up to date. This model is used by many commercial Linux distributions.
- Using the certification system, when the owner of the leading FOSS technology provides the paid certification exams to prove the qualification of the user (usually software developer). Many owners of the popular FOSS products (Sun Microsystems, Red Hat and others) offer the paid certifications that are recognized and valued by employers.
- Providing paid support for porting the FOSS to the new processor, device or operating system.
- Developing FOSS that at the given moment runs primarily on the device that is sold by the same company. FOSS adds the value to the device that now can attract more attention, be sold for the higher price or just simply become usable. This model is frequently used when porting Linux into mobile devices and other unusual platforms, it also enables companies to develop open source drivers for the hardware they produce.
- Using FOSS components (including components under GPL) to develop highly specialized software for internal use inside the same company. As such software is never officially distributed, there is no need to get profit from selling it. GPL does not demand to publish the source of the software that has been never released to public.
- Reaching new groups of customers. There are many FOSS enthusiasts in the world. The company that makes an open source - friendly product will likely attract these customers who prefer such products over its alternatives. An example is the FIC OpenMoko project where tens of thousands of FOSS - friendly mobile phones have been sold even while still unsuitable for the normal end user. Also, some Linksys routers and most notably the NSLU2 device have a completely unexpected group of users who were just interested in hacking their internals.

Opportunities for individuals

- For the individual developer, joining the FOSS project and contributing the code may provide expertise that is difficult to obtain in alternative ways. Software engineer that knows capabilities of the available FOSS, how to build it, how to port it and how to get the most use from it may be able to save large sums for her company. In some cases developing the FOSS replacement gives the direct expertise on the highly similar proprietary product. The notable examples include GNU Classpath and Sun Microsystem's Java, DotGNU or Mono and Microsoft's .NET and also various FOSS and proprietary implementations of J2EE. Such expertise is the most easy to get by contributing to the existing, highly successful FOSS project.
- Participating in FOSS helps to build a professional social network: a group of people that know well enough and if needed can recommend you as a programmer.

- For the maintainers of the small but comparatively popular projects, offering the paid documentation may work, especially if this documentation is easy to buy online.

Problems with traditional commercial software

First it should be noted that the commercial software industry is one of the largest and most important industries in the world. The rise of the Open Source movement does not necessarily spell the end of the commercial software industry. On the contrary, many people argue that commercial software can be strengthed by the use of open-source techniques. Commercial software is designed to provide a product worth paying for and most of it is (that's why the industry is so big). Despite its price tag, commercial software is often far from perfect.

Commercial products will be updated frequently in order to reflect the fluctuating demands of the market and customer needs. These needs can lead to software practices that rearrange and rewrite the software too frequently, or release beta versions as commercial endeavours resulting in high rates of bugs in early versions. Some commercial programs are over designed and written in sloppy code leading to bloated, slow, running programs. Open Source by contrast is driven by the needs of the end users. The skills of the coders is in taking personal pride in what they do, not rushing to meet artificially imposed deadlines (except their own). Thereby their code is often of a superior calibre than that of programmers in the commercial environment. There is also usually a very extensive degree of feedback as the programmers test their products within a wide network of people.

When source code is available, it can be checked against various "back doors" and other security holes that may be intentionally or unintentionally left in the closed source software. In the past such holes have been found in multiple proprietary products, including software, used by the government. Advanced developers value source code of the used components as an important additional documentation.

Having source code also means that the software can be easily ported to run under different processor, device or operating system. Proprietary software can usually only be legally ported by the original developer that may or may not treat such request as important enough to work on it.

Another problem with traditional commercial software is its closed nature. There is usually no or limited freedom to tamper with the copyrighted product. Also, companies often force users to follow upgrade paths that they may not wish to follow. Open source software enables the individual to customize the software to his end needs in all freedom. Another heavily criticised aspect of commercial software is that customers are frequently locked-in to a product because to keep using data files you are often forced to continue to use the same program. If you wish to share files with users that have upgraded, you often have to upgrade yourself or be written off as irrelevant. Since open source software allows competing programs to share different data file types, there is no reason to be trapped into any one program. If a newer version has a new file format, then there often are converters that allow users of older versions to keep up their data files up to date.

Free doesn't mean no Cost

While FOSS is free to the end user, there is a cost associated with developing the software. These costs may be smaller than developing proprietary software, because developing the project under FOSS license means that:

- Numerous web portals like SourceForge would offer web hosting, content repository, mailing lists and other essential features for free.
- The cost of advertising a FOSS project (like presenting it in the related conferences) is usually lower.
- Developing something under GPL may give free access to high quality components (like QT) that are otherwise expensive to buy or not available at all.

Still, development of any software first requires the developer time. Only very popular projects may expect to get a high quality code contributions for free.

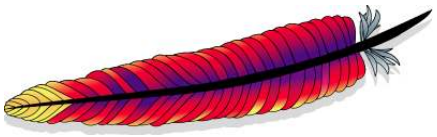
Internationalization

The open nature of Open Source Software allows for extensive internationalization of software. OS's can be modified for any language, for example recently a version of Linux was translated into Welsh (spoken in Wales). While many major commercial projects are provided in multiple languages, it is often too expensive to translate them into more than 3 or 4 languages. Many FOSS projects are available in 10 or more languages, as the translation can be done by a programmer who wants the package to be available in his or her native tongue.

Case Studies

Probably the four best known Open Source projects in the world today are: the Apache web server, the BSD operating system, the Linux kernel, and the Mozilla web browser (or its spin off, Firefox).

Apache



Logo of the Apache Foundation.

The Apache web server is currently the most popular web server in the world. Nearly 70% of all web sites are served by Apache.

The Apache project grew out of an attempt to improve upon the original web server httpd. httpd, was a public domain software project that was largely abandoned in the early 1990's. At that time it was the most popular web server in use, and several developers recognized that development needed to continue. Since httpd was in the public domain they were able to continue work on the program and include a great number of improvements. They quickly organized themselves into the Apache Foundation, to have a stable organization to oversee the on-going work on their new web server. Apache's version of httpd quickly overtook the original httpd as the most popular web server in use.

Over time the Apache Foundation grew to house many related open source projects. Including Tomcat, a popular Java servlet server, and many others.

After many years of developing Apache 1.x, the Apache Foundation found that some of the original architecture of Apache was starting to show it's age, and further extensions were becoming impossible. Therefore a team of mostly volunteer developers started to develop Apache 2. Apache 2 is a complete rewrite of Apache, and has allowed the Apache Foundation to add new flexibility of their flagship product.

The stature of the Apache Foundation has also allowed them to become a repository for significant donations of software to the open source community. Most notably 2 large donations from IBM to the Apache Foundation in 2004.

BSD

BSD (short for *Berkley Software Distribution*) is a family of three free Unixes, namely OpenBSD, FreeBSD and NetBSD. They are all successors of BSD 4.4.

Linux

The Linux kernel is currently one of the most active, and most important, free and open source software development projects. In 1991, Linus Torvalds used the Internet to invite collaboration on his infant project to develop an operating system kernel suporting a system very much like Unix. He wanted a Unix-like operating system that would work on his computer, an early and expensive 386 machine, and was unsatisfied with anything available at an affordable cost, including academic teaching examples like Minix (from Professor Andrew Tannenbaum). Since then the Linux kernel has been used in multiple Linux system distributions that are currently supported by organizations like IBM, Novell, HP, many universities (eg, ETH-Zurich), Bell Labs, and Borland. The Linux kernel is the kernel used in the GNU/Linux operating system, and is currently in its 2nd major version release, and 6th minor version. The official home page of the Linux kernel project is kernel.org [6]

Tux the Penguin, the mascot of Linux

GNU/Linux operating system is a textbook example why freedom to share the modified works is important. In his discussions Tannenbaum writes that he have frequently received suggestions to extend Minix in one or another way but was very strict when accepting these extensions. He tried to keep the system simple, easy to study and understand. People who wanted to convert it into the true next generation OS were not able to do this because direct sharing of the modified code was not allowed by the license, requiring to publish patches against the Tannebaum's version instead.

The second textbook example covers the FSF itself who have decided to refuse the available monolithic kernel and turned into difficult way of developing the microkernel based Hurd that even in 2009 have not yet been production ready. This error likely was serious enough to kill all GNU project, but FSF was protected by its own philosophy: GPL have allowed another, Linux, kernel to come, so the work spent building numerous other parts of the alternative operating system was not lost.

Mozilla (Firefox)

Mozilla and the more recent Firefox are open source derivatives of the Netscape Navigator web browser. As a result of the release of Firefox 1.0 the web saw the first major shift in web browser usage in 5 years.

GNU Classpath

GNU Classpath is a project aiming to create a free software implementation of the standard class library for the Java programming language. GNU Classpath development started in 1998 with five developers. The initial progress was rather slow because Sun was already offering its own java implementation free of charge (including commercial use) and with the source code available. Still, porting to new platforms, connecting to new virtual machines and various unusual projects required to distribute the modified versions - this was not allowed by Sun's license at that time. During the history, the project merged several times with other projects having similar goals (Kaffe, libgcj). In the past, GNU Classpath supplied its own virtual machine (Japhar). As Classpath was becoming a base library, shared with a lot of different projects, this virtual machine received less and less attention until it was dropped as no longer

supported. The project started to accelerate after approaching completeness of java 1.2 API, when it was clearly seen that some companies are joining, allowing they developers to contribute to the project at working time. GNU Classpath has become a part of near any Linux distribution and was ported to numerous processors and operating systems; some of them do not have any other java support till these days. Wave of talks passed through the conferences, demonstrating for the surprised public that there are *fully functional* alternative Free implementations of Swing, CORBA and other complex libraries that - it was believed - can never be completed by the community of volunteers.

The project status have changed after Sun released it own java implementation under near identical license (GPL+Exception). The primary purpose - to have the efficient java implementation under Free Software license - has been reached in another way. As both libraries implemented the same API and now had nearly identical licenses, it seemed doubtful that they can find they own groups of users rather than just taking one under another.

While the rise of Sun's OpenJDK meant the end of the increased status and prospects to join the history as authors of the long lived alternative java system, GNU Classpath community have shown self control keeping to serve the Free Software as good as they could: there were no opposition in the mailing lists, no attempts to create the negative image of Sun's project - shortly, no any trying to prevent replacement of GNU Classpath by OpenJDK that - now being open source - still was superior in performance. Sun has released its own libraries quite at time when GNU Classpath was about to become a fully working alternative and real competitor and it quite may be that this have contributed to the Sun's decision. Hence GNU Classpath may stay in a history as a successfull project. This project seems still active and attracts contributions, maybe because its virtual machine interface is historically much more adapted to connect various virtual machines. It is a common choice as extension of the Google Android to provide parts that were removed from the Google implementation as "unnecessary".

Joining the existing Open Source Project

Creating your own Open Source Project may be the first thing that comes in mind, but the general advice for the beginner would be **do not do this**. The project must be based on a really good idea and needs cooperation of more than one developer to be really successful. Otherwise you may just observe the download counter slowly ticking up, but never get any other feedback. It is usually much better to start from joining the existing project and starting your own later, when you already have enough experience. For the same reason it is better not to start from the attempt to revive the abandoned project which has already lost its previous team (see why [7]).

Writing and using Free software is not just a programming, is a kind of philosophy. While programming language is all you need to program, it seems also important to join the community, get friends, do a great work together and become a respected specialist with profile you cannot get anywhere else.

Steps

Initial preparation

Be sure you are competent enough to make valuable contribution as a programmer. Without this, you are useless to the developer community so nobody will talk to you. The next most important tool to master is the version control (CVS [8], SVN [9]). Understand how to create and apply patches (text difference files). The most of the FOSS development in the community is done creating, discussing and applying various patches.

Searching for the first project to join

While a highly popular and very successful project may give you lots more expertise, such projects are also quite demanding to their contributors. You are expected to follow coding style, write CHANGELOG entries, create and apply patches, cover your code with tests and so on. Without experience that is expected and why, all this may end up by conflict between you and the project community. If you are totally new with FOSS, it may be more efficient to

start from something less demanding, at least for a short time. Most of the small projects that are very easy to join now can be found on SourceForge.net. The suitable project must:

1. Use the programming language you already know.
2. Be active, with recent releases. Usually a project that has no release for more than a year is considered dormant if not dead.
3. Already have three to five developers or about.
4. Use version control.
5. Have some part you think you can immediately start implementing without modifying the existing code too much.
6. Apart from the code, a good project also has the active discussion lists, bug reports, receives and implements requests for enhancement and shows other similar activities.

Some FOSS portals offer "job openings" where administrators of the active projects are actively posting invitations for new developers. These may be interesting to read but it may be better just to search for a project you like, regardless if it posts invitations or not.

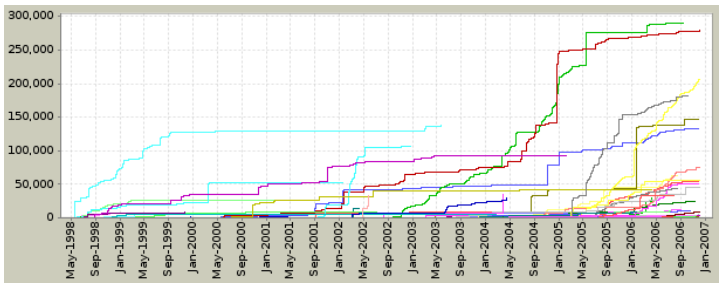
Joining the first project

1. Contact the administrator of the selected project. In a small project with few developers your help will usually be immediately accepted.
2. Carefully read the rules of the project and more or less follow them. The rules of the coding style or necessity to document your changes in a separate text file may first appear ridiculous to you. However the purpose of these rules is to make the shared work possible - and the most of projects do have them.
3. Work in this project for several months. Listen carefully that the administrator and other project members say. Apart programming, you have a lot of things to learn. But if you really do not like something, just go away to another project.
4. Do not stick with the underground project for too long. As soon as you find yourself successfully working in that team, it is time to look for the serious one.

Joining the second project

1. Find a serious, high level Free software or Open source project.
2. As we are doing a serious jump now, be ready for the far cooler acceptance. You will likely be asked to work for some time without direct write access to the code repository. The previous underground project should, however, teach you a lot - so after several months of the productive contribution you may try do demand rights you think you could have.
3. Take and do a serious task. Go on even after you discover that the task is lots more difficult than you initially thought - in this step it is important not to give up.
4. If you can, apply with your serious task to the Google's "Summer of code" to get some money from this adventure. But just do not care if the application would not be accepted as they have far less funded positions than really good hackers working around.
5. Look for a suitable conference happening nearby ("Linux days" or something similar) and try to present your project there (*all project*, not just the part you are programming). After you tell you are representing a serious Free / Open source project, the organizers frequently release you from the conference fee (if they do not, the conference is likely unsuitable anyway). Bring your Linux laptop (if you have one) and run demos. Ask the project administrator for the material you may use when preparing your talk or poster.
6. Complete the task, cover with automatic tests and contribute to the project. At this point the initial goal of becoming the active member of the serious project is achieved. You may continue in the same project or later migrate into another one to learn something new.

7. For better understanding, look into real example of the development history for a Free Software project (above). Each raising curve represents a contribution (lines of code) from single developer. Developers tend to become less active over years but the project frequently even accelerates as new people join. Hence if you already come with some useful skills, there are no reasons why the team would not invite you.



Development history

Tips

- If you still do not trust yourself enough, start from some part of code that you think is missing and can be written from scratch. Changes in existing code are much more likely to attract criticism.
- For the beginning, select a class, module or some other unit under which nobody is very actively working at the moment. Working together on the same class or even function needs more skills and a lot of care from all sides.
- Before asking any questions about the working rules inside the project, try to search for the answer in the project documentation and mailing list archives.
- Avoid asking any questions related to fundamentals of programming or programming tools. The time of the Free software programmer is as valuable as gold and nobody can afford to teach you the basic.
- For the same reason, *never* expect an older hacker writing a detailed description of your task or even providing any kind of supervision for you. While open source projects may have a lot of strict rules, they usually work along the lines of that is known as extreme programming in the programming methodology.
- Linux is a rather good system now, but hackers widely used it even when it was far less complete in the past. For a hacker it is a lot more important that all corners of this system come with source code. They are open to explore, to understand, to change and to share your changes with others. Understanding starts from the trivial daily use (writings, mail, web and surely programming). Hence you must install Linux at least on a dual boot and start from using it for routine daily tasks like web surfing or programming.
- The employers of many hackers seem motivated enough to allow contributions during their working time (usually because the institution uses the Free/Open source program that the hacker is developing). Think, maybe you can get at least part of the needed time this way.
- Always continue the hacking you started. Does not build, does not run, crashes? There *are* reasons for everything and if you have source code this usually means that you *can* force the system to do whatever you want, especially with the help of the web search. This rule has its limits, but, indeed, never yield easily.
- Only say you are a hacker after some true hacker community recognizes you as such.

Warnings

While FOSS communities are in general very friendly and cooperative, it is important not to make some trivial mistakes that may result in an unpleasant experience.

- Avoid Windows, especially if you plan to meet Free software developers. Mac OS is tolerated somewhat better, but also not welcome. If you do bring your laptop, it must run Linux or another operating system that is considered as "Free software". This may be less important in the Open Source communities but you likely will never miss by bringing Linux or a dual-boot laptop.
- If your mail client supports html messages, turn this feature off. Never attach documents that only proprietary software (like MS Word) can open properly. Hackers understand this as insulting.

- While the word "hacker" sounds respectable in most academic environments, for some uninformed people it may be associated with breaking into security systems and other computer-related crimes that a different social group, crackers do. Unless you are ready to explain, look to whom are you telling this word. Real hackers as they are meant in this article *never* join programming activities that seem for them illegal. First, they are proud of following the hacker ethic [10]. Second, the law violations are not necessarily better paid.
- Do not volunteer to the company-owned projects that are not releasing some parts of they code under approved Open Source license. In such cases the really important parts of the project are likely to stay behind the closed doors of the owner, preventing you from learning anything useful.
- Do not start from small code optimizations, extra comments, coding style improvements and other similar "small-scale" stuff. It may attract far more criticism than any serious contribution.

Sources and Citations

- http://sourceforge.net - place to find the first project to join.
- http://www.fsf.org/campaigns/priority.html - list of the top level, high priority projects at GNU.
- http://www.apache.org/- Apache community (they call they software "open source" and have slightly different license)
- http://catb.org/~esr/faqs/smart-questions.html - A true and useful guide how to ask questions to other hackers.
- http://freesoftware.mit.edu/papers/lakhaniwolf.pdf - A serious scientific study about Free software hackers.
- http://www.gnu.org/software/classpath/docs/hacking.html#SEC8 - Rules of the typical Free Software project

Starting and Maintaining an Open Source Project

Getting Started

Vapourware is software that is 'floated' to see if there is interest. Many projects, if donated to the open source methodology would find usage, development and markets much quicker. Lost code might be resurrected and enhanced. More and more computer users are learning to use the excellent development tools created and enhanced by programmers. If you have a good idea, open source is a way of making it happen.

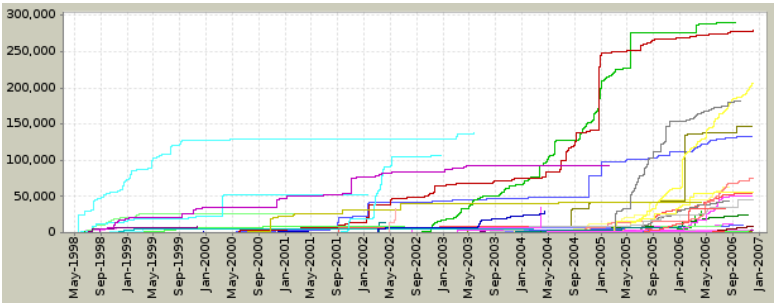
Project Management

Necessary processes of the FOSS project

There is a typical set of tools that are usually involved in managing near any FOSS project. Small projects may not use some of them; large projects near always use them all. These tools, roughly by the the frequency of usage, are:

- Version control repository that contains all past revisions of the project and can be accessed remotely. The past projects mostly used CVS. The recent projects usually use SVN or other tools.
- Bug/issue tracker that contains bug reports and requests for enhancement, together with subsequent comments and implementation status. One of the most frequently used tools there is Bugzilla.
- Mailing lists to discuss various aspects of the project development.
- Suite of automatic tests.
- CHANGELOG where all code changes are manually documented. This is especially important when there are many developers.
- Website that describes the project and provides both "getting started" and advanced documentation.

Contributors of the typical FOSS project



The real example of the development history for a Free Software project (above). Each raising curve represents a contribution (lines of code) from single developer.

If the project depends on external contribution, it is important to attract new contributors over time. Volunteers tend to become less and less active over time, and donations from the companies are also frequently not continuous. This does not mean that the project stagnates: it may become even more active over time as new developers join. However it is obvious that novices must be welcome and maintaining the small group of "elite developers" that does not allow anybody else near to the code is problematic. The software quality is ensured using other methods (automatic tests and at least brief checking of all patches by older developers).

Version Numbers

Many new users find the FOSS projects often use strange version numbering systems. For instance it is very common to see software in wide distribution that has a version number less than 1.0. For example the web browser Firefox was widely used by version 0.6, more than a year before version 1.0 was finished. In the world of FOSS software it is generally considered important to work out all the known bugs before releasing version 1.0. While this is not a totally attainable goal, many users find early versions of software much more stable than early versions of their commercial counterparts.

Generally FOSS projects break their version numbers into 3 pieces: major version, minor update, point release. Major version is the first number in the version number; it is rare to see a free software project have a major version greater then 2 (although GNU Emacs is currently on version 21, so it is not unheard of). The minor update is the second number (the x in 2.x). A change in the number often signifies a significant update of the program. The third number (the y in 2.x.y) generally indicates a small update to fix a bug, or security problem. These numbers get very high, often over 30 or 40 over time. When a software project is well established most of their updates will come in the form of point releases.

Version numbers may also carry extra meanings. For instance ever since the Linux kernel went to version 2, it has been the practice of the design team to use odd numbers for development kernels, and even numbers for stable kernels. Therefore while it is common to see 2.2, 2.4, and 2.6 all in use today, 2.1, 2.3, 2.5 are almost no where to be seen. There are still teams that work on supporting bug/security fixes to the 2.0, 2.2, 2.4 and 2.6 kernels. As of March 2005, this policy was under review by the primary authors of the Linux kernel, however there are many projects that use similar numbering schemes, and so it holds as a useful example.

When a software project changes the major version number, this generally indicates a complete rewrite of the software, or at least an extensive over haul. The Apache web server started a version 2.0 in 2003, after many years of 1.3.x versions. Apache2 is a near total rewrite of Apache, which provides many new features, and significantly improved the underlying code base. The apache foundation still maintains the 1.3.x code base, although most efforts for new code are done on the 2.0.x code base.

Writers of software may completely rewrite a program, improve software or unnecessarily inflate perfectly good software. If software works then there is little need to upgrade.

Also upgrading may solve some problems and add new ones. To give you an example 'Abi Word' is a fine word processor but it crashed on me. The complete package in Open Office has never done this nor has notepad or WordPad. New improved 'notepads' have occasionally crashed. So I stopped being seduced by what has many

features. Any software that has the hidden feature of crashing is a liability. I have used Miranda IM without problem, others find it does not work.

Open Source Hardware

Various computer hardware frequently has its own specific features that only specialized piece of the software can read properly. As a rule, the operating system contains device drivers that deal with all specifics, providing more general access for other programs. A manufacturer may choose to disclose the documentation that is required to write a driver or to keep it secret, offering the finished drivers instead.

The proprietary driver is usually a problem for the open source project. First, it usually has licensing limitations that limit its usage (being closed source is automatically incompatible with GPL). Second, there is no way to check such driver against bugs and security flaws. Finally, the driver is frequently simply not available for the platform of interest; it is quite usual to provide such drivers for only one operating system.

Because of these reasons Linux, Solaris and other open source operating systems may not provide the needed drivers for the hardware which does not have sufficient documentation. In cases when such hardware is very widespread and popular, experienced hackers may succeed to reverse-engineer the device and write their own driver without documentation. However this is a difficult and long process and the finished driver may still lack some functionality.

Hence before writing the open source software that uses some specific hardware like camera, advanced video or wireless it is always important to choose the equipment from the manufacturer that shares the necessary technical information or at least provides its own driver for your system, your platform and under conditions that are for you acceptable. This must be carefully checked in advance, without expecting that "it will work anyway" and especially without assuming that the manufacturer may change its position later.

For instance, when designing the fully open mobile phone (OpenMoko project), the FIC company have selected Hammerhead PMB 2520 chip for its first prototype, Neo1973. The documentation on accessing this chip directly has been secret. After long negotiations the manufacturer provided some driver but due licensing problems every user needed to download it from the special web site and install individually. As the phone was primarily targeted to the open source enthusiasts that do not like such "binary blobs", in many cases the driver at the end was refused, leaving the GPS on the phone non functional. Further negotiations were not successful and FIC was forced to switch into completely different chip in the next iteration. Switching, however, required to redesign some circuitry. When the new phone has been delivered, it was discovered that due lack of time the new GPS device has not been properly tested and only worked when the SD card has been removed from its slot. While at the end all problems have been fixed and resolved, long delays and non-working devices likely reduced popularity of the project that had financial problems afterwards.

Open Source Hardware [11] is developing free BIOS and CPU specs and electronic designs.

Future Developments

FOSS is creating a new outlook. The attitude is of particular benefit to the commercial sector who obtain code and tools to commercialise. Developing nations and people can use no cost materials and resources. The developer community is trying out new systems and most importantly leaving a legacy that will be available for generations to come.

Wikimedia projects use the same principles as the open source and free software movement but apply it to knowledge.

Ideas in use outside of technology

Grass roots organizers have started to take on some of the ideas generated by the Free software movement. By encouraging more transparency and sharing of ideas grass roots groups have found they can motivate more people with less effort.

Projects like Wikipedia and Wikibooks were founded with many of these same principals in mind.

Tips for picking software

There is no one best way to find FOSS projects that will best serve your needs. There are several rules of thumb you can use to help you make informed decisions.

Is this Open Source?

> Some companies try to attract more users, pretending that they offer FOSS software when they actually do not. The simple way to check this is to look if they license is OSI approved [4]. Unusual license frequently means that the project just pretends being FOSS.

Do you already have something that will do the job well?

> Before you hunt up new software, make sure you do not already have a program that will do the job nicely. From the other side, having possibility to do the job does not mean that there is no software to do this more efficiently so it may be good to evaluate alternatives time to time.

Does the program do what you want?

> While this might seem obvious, every IT support person can tell you stories of users that came to them asking for help using a particular tool because a friend recommended it, but it is the wrong tool for the job.

Do you care about the ideologies behind free software and open source software, or need to work with people who care?

> There are many people that feel that using free software is an important part of how they live and work. Such people will not replace FOSS program by proprietary regardless of the quality of the two. However recently many FOSS is not much worse than proprietary alternatives and some is better.

Do you know other people that already use it?

> Online reviews are great, but there is still nothing like talking to someone you know about a program they use every day. Real users can tell you how the program has performed over the long run. Your friend may have discovered that while it worked great at first it was missing an important feature for a project you have in mind. Also if you are doing a common task with a program that no one else is using, there's probably a reason (and it's not likely that you're smarter than everyone else...sorry).

Is the current version over a year old?

If yes, then the project has most likely died or stagnated. Active projects tend to produce new updates every few weeks or months. While old releases may be a sign of a well written program, that needs little to no updating, often it is a sign that the programmers have moved on to other endeavours and abandoned the project.

Are there user support mailing lists you can join?

Particularly when you are new to a program it is nice to be able to get support learning your way around. Email lists or web discussion boards are often excellent sources of support for FOSS programs. Also look at the list's archives, if they are active lists you can get a sense of what the common problems, and how many people are using the program.

Other Frequently Asked Questions

Is Open Source software always of better quality than other types of software?

This depends strongly on that do you understand as the main requirements. But, in general, far from it. The same is true of any kind of software (semi-free, proprietary, etc.). Everywhere it is possible to find bad software and good software.

Programmers may still be learning, and provide their projects in an open-source venue as a means of quicker development by opening the project to others. Lack of interest may result in abandoned projects. If the need and idea is good, open source is rapidly and effectively developed. The original developer may benefit from their experience as well as faster bug discovery by others in the community. Expect many minor releases from Free and Open Source software.

Is Open Source always of worse quality, then ?

Some managers think that due lack of funding FOSS just cannot be as good as the proprietary alternative and is only used because it is cheaper to get. In recent days, it is also not necessarily so, especially taking into consideration that the criteria may be different. It is possible to propose acceptance criteria where Linux beats Windows (there *are* tests that run faster and at least less viruses), Thunderbird beats Lotus Notes (less capable but way faster and also cross platform) and KPdf beats Acrobat Reader (at least starts faster). Software like Linux kernel, OpenSolaris operating system, Sun java implementation, Firefox, Thunderbird, Google Android and many others have been created investing huge amount of money in the past. It is a myth that open source is developed only by scarce isolated individuals so there is no any particular reason why it should always be worse.

Where can I find Open Source Software?

The most reliable way to get such software is to download it from the popular, recognized web portal. The situation changes over time, but the most popular portal currently seems being sourceforge.net [12]. Also see www.gnu.org [6] for a list of projects sponsored by the Free Software Foundation. Linux users usually get lots of the Open Source software with their distribution and can search for more with user friendly software management tools that are part of the operating system.

There are also Freeware sites that offer trial, restricted or even fully functional versions of some software for free. However it is important to understand that offering software for free does not make it Open Source by itself.

Other Resources

- Article on why FOSS works [13]

Contributors

- Lobster: started this book
- ahc: Took over/reshaped the book. Started work fall 2004
- User:audriusa: Extended a little bit in 2008.

References

[1] http://www.gnu.org/gnu/gnu-history.html
[2] http://daemonz.org/bugs/history.html
[3] http://www.gnu.org/philosophy/license-list.html
[4] http://www.opensource.org/licenses
[5] http://habitat.igc.org/wealth-of-networks/
[6] http://www.kernel.org
[7] http://www106.pair.com/rhp/hacking.html
[8] http://www.linux.ie/articles/tutorials/cvs.php
[9] http://www.linux.ie/articles/subversion/index.php
[10] http://en.wikipedia.org/wiki/Hacker_ethic
[11] http://opencollector.org/Whyfree/
[12] http://www.sourceforge.net
[13] http://www.dwheeler.com/oss_fs_why.html

Use the Source

Welcome! You have come upon the beginning of a lighter-side look at Open Source, open source, Free Software, free software, Unix, Unix-likes and the whole mess known today as the Internet.

This book touches on the concepts of hacking, cracking, scripting and programming; illustrating the differences between them all and why there is a gap between the people doing these things and the masses unaware of the differences between them. Other topics will include the reasons for using different licenses for programs and what some of the more common ones mean to users and programmers.

Contents

1. What do you mean Open Source isn't Free Software? - Explaining the differences in the terms used for the various types of open source software.
2. I have to give you what? - Talking about the rights given in licenses and those that are not.
3. Theft or Copying? - Talking about "stealing" source, how it is actually copying with no loss of property involved.
4. In the Beginning... - Explaining where the various open source movements began and the start of the Internet, do not go into detail the groups, just their basics - further detail will be in their specific chapters.
5. Unix - The beginnings of Unix, it's relation to Multics and it's involvement with BSD. Mention Bell's K&R and the beginning of C.
6. BSD - Talk about the BSDs, their founders and their strengths and weaknesses, go into more detail than In the Beginning....
7. Let's get ready to RUMBLE! - The lawsuit involving BSDi, USL and the University of California; discuss the result of the suit as well as how it effected the BSD and Linux movements.
8. BSD's Not Unix - The Unix trademark, Unix-likes, System V.

9. Gnu's Not Unix - Talk about Richard Stallman and the Free Software Foundation and about the various licenses they have made and how they view other licenses.
10. Linux - Talk about Linus Torvalds and the kernel he made, describe the relation to Minix and Unix.
11. Gnu's Not Linux! - Talk about the mincing of words between the Linux and GNU camps.
12. Gnu/Linux - The partial result of the GNU and Linux camps spat.
13. OpenSolaris - Discuss OpenSolaris, what it means to other open source projects.
14. Divide and Conquer - Talk about community friction, the concept of allies and enemies within the open source world.
15. Cancer - Talk about the stance given by Microsoft regarding the General Public License, Linux and GNU.
16. Counter-Culture - Talk about the rising of an anti-Microsoft movement, its relation to the open source groups and its effects on their usage and development.
17. l33t l-l4X0rz!?! - Hacking, cracking, script kiddies and the like.
18. Glossary - Definitions of words, technical and cultural terms, acronyms and notes.
19. Licenses - A much more detailed look at licenses, will not be for the faint of heart
20. Additional Information - A list of informational sources, the bibliography, thanks and contributors.

Related wikibooks

- Open Source

Guide to X11

Wikibooks Guide to X11

Authors · History · Print

This book is about the *X Window System*, the environment popular with Unix systems.

Contents

Supplements

- Guide to Unix/Commands/X11 *Unix commands for X11*

Using GNOME

Using GNOME is an unofficial user's guide for GNOME3.X. Its core goal is to provide helpful information on how to use GNOME, and its application stack with ease. We will also go over any and all core concepts, and the terminology that goes along with it.

- Note it is currently in the process of being converted from GNOME2.X, so many pages may be incomplete, or old ...

Table of Contents

Using KDE

This is a book that is aimed to help everyone from beginners to power users with KDE. It contains a lot of step-by-step tutorials and other general information.

The KDE 4 logo.

Table of Contents

Article Sources and Contributors

Guide to Unix *Source*: http://en.wikibooks.org/w/index.php?oldid=2537169 *Contributors*: *nix, A thing, Adrignola, AlbertCahalan, Aya, CFeyecare, Dan Polansky, Darklama, DavidCary, Dysprosia, Ellisun, Imran, Kernigh, Markhobley, Mkn, Panic2k4, Query, QuiteUnusual, Reisio, Robert Horning, Techman224, Thenub314, Uncle G, 4 anonymous edits

Introduction *Source*: http://en.wikibooks.org/w/index.php?oldid=1908175 *Contributors*: Adrignola, CFeyecare, Kernigh, Reisio

Why Unix-like *Source*: http://en.wikibooks.org/w/index.php?oldid=1619570 *Contributors*: A333, Adrignola, Bensin, Darklama, Fitzhugh, Hagindaz, Jaosoa, Kernigh, Murban, Reisio, Webaware, 8 anonymous edits

Linux *Source*: http://en.wikibooks.org/w/index.php?oldid=1928157 *Contributors*: Abd, Adrignola, Ahc, AlbertCahalan, Anoopalias01, Bensin, Conan, Eibwen, Goodgerster, IO, Jguk, Kernigh, Lobster, ManuelGR, Mike.lifeguard, Perl, PhilippWeissenbacher, Query, Reisio, Reub2000, Webaware, 32 anonymous edits

BSD *Source*: http://en.wikibooks.org/w/index.php?oldid=2063163 *Contributors*: A thing, Adrignola, Avicennasis, CFeyecare, Darklama, Kernigh, Reisio, 4 anonymous edits

Explanations *Source*: http://en.wikibooks.org/w/index.php?oldid=1939757 *Contributors*: A thing, Adrignola, Aholt, CFeyecare, Kernigh, Markhobley, Query, Reisio

Commands *Source*: http://en.wikibooks.org/w/index.php?oldid=2240862 *Contributors*: A thing, Adrignola, Arky, CFeyecare, Codeman, DavidCary, Davidbspalding, Djamund, Eibwen, Fractal3, Fudz, Herbythyme, Hyad, Imran, Jfmantis, Jomegat, Kernigh, Krischik, Perl, Reisio, RrsunN, Siddharthc, Uncle G, 22 anonymous edits

Environment Variables *Source*: http://en.wikibooks.org/w/index.php?oldid=2551200 *Contributors*: A thing, CFeyecare, Kernigh, Reisio, Xela, 11 anonymous edits

Files *Source*: http://en.wikibooks.org/w/index.php?oldid=2053611 *Contributors*: Avicennasis, DuLithgow, Imran, Jguk, Kernigh, Reisio, Spoon!, Waratah, 9 anonymous edits

GNU Free Documentation License *Source*: http://en.wikibooks.org/w/index.php?oldid=510353 *Contributors*: Jguk, Kernigh, Reisio

OpenBSD *Source*: http://en.wikibooks.org/w/index.php?oldid=2578373 *Contributors*: Adrignola, CFeyecare, Darklama, Gronau, Kernigh, Nmontague, PaulRg, Reisio, 11 anonymous edits

A Neutral Look at Operating Systems *Source*: http://en.wikibooks.org/w/index.php?oldid=2556366 *Contributors*: Adrignola, AlbertCahalan, CarbonUnit, Darklama, DavidCary, Hashar, Hyperlink, Jats, Jcarroll, Jguk, Jonmmorgan, Kernigh, KirbyMeister, MARQUIS111, Noogz, Panic2k4, Polluks, Prosfilaes, Robert Horning, Thenub314, Trevj, Wereon, Whiteknight, 28 anonymous edits

UNIX Computing Security *Source*: http://en.wikibooks.org/w/index.php?oldid=1963309 *Contributors*: Adrignola, Jguk, Kernigh, Ksetty, Panic2k4, RJHall, Robert Horning, Thenub314, Whiteknight, 3 anonymous edits

Bourne Shell Scripting *Source*: http://en.wikibooks.org/w/index.php?oldid=2578003 *Contributors*: Adrignola, Alsocal, BenTels, Dallas1278, Hagindaz, JesseW, Jguk, JimD, Kernigh, Mahanga, Mike.lifeguard, Mkn, Nixcraft, Reach Out to the Truth, Serge, Smitty0157, Tchoř, Unforgettableid, 21 anonymous edits

Learning the vi Editor *Source*: http://en.wikibooks.org/w/index.php?oldid=1915012 *Contributors*: Adrignola, Andreas Ipp, Darklama, Dreftymac, Dysprosia, Fück wikiböoks, Hannes Röst, Heptite, Jguk, Jonathan Webley, Josh-A, Kernigh, Krischik, Liblamb, Mercy, Mkn, Panic2k4, Remi, Robert Horning, Thenub314, Waxmop, Whiteknight, 16 anonymous edits

FLOSS Concept Booklet *Source*: http://en.wikibooks.org/w/index.php?oldid=2554864 *Contributors*: A thing, Ab, Adrignola, Avicennasis, CoyneT, Darrelljon, DavidCary, Derbeth, Geocachernemesis, Guanaco, Jeebesh, Kernigh, Kream, Mkn, OsamaK, Paul tomlinson, Robert Horning, Satyakam, Soeb, Spiritia, Tom Morris, Vault, Vijaykumar, Whiteknight, 121 anonymous edits

Open Source *Source*: http://en.wikibooks.org/w/index.php?oldid=2509459 *Contributors*: A thing, Adrignola, Ahc, Anuvrat, AriadneA, Audriusa, Az1568, Beta m, Beuc, Califman831, Chuckhoffmann, Churchill17, Cspurrier, DavidCary, Derbeth, Dlrohrer2003, Dysprosia, Everlong, Evershade, Fdfdffdfdf, Gavintlgold, Gecko, Guanaco, HahnRholo, Information Ecologist, Irunongames, Jguk, K.s.sampathkumar, Lobster, MeMoria, Neoptolemus, Paddu, Panic2k4, Psoup, QuiteUnusual, Robert Horning, Techman224, TheDonk, Tom Morris, Webaware, Westhaking, Wknight8111, Zoohouse, 149 anonymous edits

Use the Source *Source*: http://en.wikibooks.org/w/index.php?oldid=1963659 *Contributors*: Adrignola, Darklama, GreatWhiteNortherner, Jguk, Kernigh, Large and in charge, Mike.lifeguard, Nmontague, Panic2k4, Robert Horning, Swift, Whiteknight, 4 anonymous edits

Guide to X11 *Source*: http://en.wikibooks.org/w/index.php?oldid=2508275 *Contributors*: Adrignola, D'Arby, Darklama, Ellisun, Jguk, Kernigh, Panic2k4, QuiteUnusual, Robert Horning, Whiteknight, Zondor, 1 anonymous edits

Using GNOME *Source*: http://en.wikibooks.org/w/index.php?oldid=2304790 *Contributors*: Adrignola, CommonsDelinker, Darklama, Jameshales, Jguk, Karl Wick, Kernigh, Krik, Mkn, Panic2k4, Robert Horning, SBJohnny, Techman224, Webaware, Whiteknight, Wikimi-dhiann, Youlysses, 5 anonymous edits

Using KDE *Source*: http://en.wikibooks.org/w/index.php?oldid=2320728 *Contributors*: Adrignola, Alsocal, Darklama, Herbythyme, Herraotic, Jguk, Kernigh, Mattwj2002, Panic2k4, QuiteUnusual, Robert Horning, Soeb, Swift, Techman224, ThePCKid, Whiteknight, 10 anonymous edits

Image Sources, Licenses and Contributors

Image:Tux.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:Tux.svg *License*: Attribution *Contributors*: Larry Ewing, Simon Budig, Anja Gerwinski

File:Richard Matthew Stallman.jpeg *Source*: http://en.wikibooks.org/w/index.php?title=File:Richard_Matthew_Stallman.jpeg *License*: GNU Free Documentation License *Contributors*: BrokenSegue, DCEvoCE, Dbenbenn, Durin, Emijrp, Guety, John Vandenberg, MichaelSchoenitzer, Niqueco, Porao, Sj, Solon, Tacsipacsi, Ævar Arnfjörð Bjarmason, 2 anonymous edits

Image:FLOSS Timeline.png *Source*: http://en.wikibooks.org/w/index.php?title=File:FLOSS_Timeline.png *License*: BSD *Contributors*: Original uploader was Vijaykumar at en.wikibooks

Image:Os1.jpg *Source*: http://en.wikibooks.org/w/index.php?title=File:Os1.jpg *License*: GNU Free Documentation License *Contributors*: Az1568, Iamunknown, Lobster

File:BSD wordmark.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:BSD_wordmark.svg *License*: Public Domain *Contributors*: The FreeBSD Project

File:ASF-logo.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:ASF-logo.svg *License*: Apache *Contributors*: Apache Software Foundation (ASF)

File:NewTux.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:NewTux.svg *License*: Attribution *Contributors*: gg3po (kde-look.org source)

Image:GcAct_85.png *Source*: http://en.wikibooks.org/w/index.php?title=File:GcAct_85.png *License*: GNU Free Documentation License *Contributors*: Audrius Meskauskas, uploader. Samd image is uploaded to Wikihow by me.

File:Wikimedia logo text RGB.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:Wikimedia_logo_text_RGB.svg *License*: Public Domain *Contributors*: Logo designed by .

Image:Gnome-2.20-screenshot.png *Source*: http://en.wikibooks.org/w/index.php?title=File:Gnome-2.20-screenshot.png *License*: GNU General Public License *Contributors*: Original uploader was Ermey at ca.wikipedia

File:KDE_logo.svg *Source*: http://en.wikibooks.org/w/index.php?title=File:KDE_logo.svg *License*: GNU Lesser General Public License *Contributors*: KDE

License