2. Open source applications and licenses

2.3 Major Open Source Applications

The Linux kernel can run a wide variety of software across many hardware platforms. A computer can act as a *server*, which means it primarily handles data on other's behalf, or can act as a *desktop*, which means a user will be interacting with it directly. The machine can run software or it can be used as a *development machine* in the process of creating software. You can even run multiple roles as there is no distinction to Linux about the role of the machine; it's merely a matter of configuring which applications run.

One advantage of this is that you can simulate almost all aspects of a production environment, from development, to testing, to verification on scaled down hardware, which saves costs and time. As someone learning Linux, you can run the same server applications on your desktop or inexpensive virtual servers that are run on a large Internet Service Provider. Of course, you will not be able to handle the volume a large provider would, as they will have much more expensive hardware. But you can simulate almost any configuration without needing powerful hardware or server licensing.

Linux software generally falls into one of three categories:

- **Server software** – software that has no direct interaction with the monitor and the keyboard of the machine it runs on. Its purpose is to serve information to other computers, called **clients**. Sometimes server software may not talk to other computers but will just sit there and "crunch" data.
- **Desktop software** – a web browser, text editor, music player, or other software that you interact with. In many cases, such as a web browser, the software is talking to a server on the other end and interpreting the data for you. Here, the desktop software is the client.
- **Tools** – a loose category of software that exists to make it easier to manage your system. You might have a tool that helps you configure your display, or something that provides a Linux **shell**, or even more sophisticated tools that convert source code to something that the computer can execute

Additionally, we will consider *mobile applications*, mostly for the benefit of the LPI exam. A mobile application is a lot like a desktop application but it runs on a phone or tablet instead of a desktop computer.

Any task you want to do in Linux can likely be accommodated by any number of applications. There are many web browsers, many web servers, and many text editors (the benefits of each are the subject of many UNIX holy wars). This is no different than the closed source world. However, a benefit of open source is that if someone that doesn't like the way their web server works, they can start building their own. One thing you will learn as you progress with Linux is how to evaluate software. Sometimes you'll go with the leader of the pack, sometimes you'll want to look over the bleeding edge.

2.3.1 Server Applications

Linux excels at running server applications because of its reliability and efficiency. When considering server software, the most important question is "what service am I running?" If you want to serve web pages, you will need web server software, not a mail server!

One of the early uses of Linux was for *web servers*. A web server hosts content for web pages, which are viewed by a web browser using the **Hypertext Transfer Protocol (HTTP)** or its encrypted flavor, HTTPS. The web page itself can be static which means that when the web browser requests the page the web server just sends the file as it appears on disk. The server can also serve *dynamic content*, meaning that the request is sent by the web server to an application, which generates the content. WordPress is one popular example. Users can develop content through their browser in the **WordPress** application and the software turns it into a fully functional website. Each time you do online shopping, you are looking at a dynamic site.

Apache (http://projects.apache.org/projects/http_server.html) is the dominant web server in use today. Apache was originally a standalone project but the group has since formed the **Apache Software Foundation** and maintains over a hundred open source software projects.

Another web server is **nginx** (http://nginx.org/) which is based out of Russia. It focuses on performance by making use of more modern UNIX kernels and only does a subset of what Apache can do. Over 65% of websites are powered by either nginx or Apache.

Email has always been a popular use for Linux servers. When discussing email servers it is always helpful to look at the 3 different roles required to get email between people:

- **Mail Transfer Agent (MTA)** – figures out which server needs to receive the email and uses the **Simple Mail Transfer Protocol (SMTP)** to move the email to that server. It is not unusual for an email to take several "hops" to get to its final destination, since an organization might have several MTAs.
- **Mail Delivery Agent (MDA**, also called the **Local Delivery Agent)** – takes care of storing the email in the user's mailbox. Usually invoked from the final MTA in the chain.
- **POP/IMAP server** – The **Post Office Protocol** and **Internet Mail Access Protocol** are two communication protocols that let an email client running on your computer talk to a remote server to pick up the email.

Sometimes a piece of software will implement multiple components. In the closed source world, Microsoft Exchange implements all the components, so there is no option to make individual selections. In the open source world there are many options. Some POP/IMAP servers implement their own mail database format for performance, so will also include the MDA if the custom database is desired. People using standard file formats (such as all the emails in one text file) can choose any MDA.

The most well-known MTA is **sendmail** (http://www.sendmail.com/sm/open_source/). **Postfix**(http://www.postfix.org/) is another popular one and aims to be simpler and more secure than sendmail.

If you're using standard file formats for storing emails, your MTA can also deliver mail. Alternatively, you can use something like **procmail** (http://www.procmail.org/) which lets you define custom filters to process mail and filter it.

Dovecot (http://dovecot.org/) is a popular POP/IMAP server owing to its ease of use and low maintenance. **Cyrus IMAP** (http://cyrusimap.web.cmu.edu/) is another option.

For file sharing, **Samba** (http://www.samba.org/) is the clear winner. Samba allows a Linux machine to look like a Windows machine so that it can share files and participate in a Windows domain. Samba implements

the server components, such as making files available for sharing and certain Windows server roles, and also the client end so that a Linux machine may consume a Windows file share.

If you have Apple machines on your network, the **Netatalk** (http://netatalk.sourceforge.net/) project lets your Linux machine behave as an Apple file server.

The native file sharing protocol for UNIX is called the **Network File System (NFS)**. NFS is usually part of the kernel which means that a remote file system can be mounted just like a regular disk, making file access transparent to other applications.

As your computer network gets larger, you will need to implement some kind of directory. The oldest directory is called the *Domain Name System* and is used to convert a name like http://www.linux.com to an *IP address* like 192.168.100.100, which is a unique identifier of that computer on the Internet. DNS also holds such global information like the address of the MTA for a given domain name. An organization may want to run their own DNS server to host their public facing names, and also to serve as an internal directory of services. The Internet Software Consortium (http://www.isc.org/) maintains the most popular DNS server, simply called *bind* after the name of the process that runs the service.

The DNS is largely focused on computer names and IP addresses and is not easily searchable. Other directories have sprung up to store other information such as user accounts and security roles.
The **Lightweight Directory Access Protocol (LDAP)** is the most common directory which also powers Microsoft's Active Directory. In LDAP, an object is stored in a tree, and the position of that object on the tree can be used to derive information about the object in addition to what's stored with the object itself. For example, a Linux administrator may be stored in a branch of the tree called "IT department", which is under a branch called "Operations". Thus one can find all the technical staff by searching under the IT department branch. **OpenLDAP**(http://www.openldap.org/) is the dominant player here.

One final piece of network infrastructure is called the **Dynamic Host Configuration Protocol (DHCP)**. When a computer boots up, it needs an IP address for the local network so it can be uniquely identified. DHCP's job is to listen for requests and to assign a free address from the DHCP pool. The Internet Software Consortium also maintains the **ISC DHCP** server, which is the most common player here.

A *database* stores information and also allows for easy retrieval and querying. The most popular databases here are **MySQL** (http://dev.mysql.com/) and **PostgreSQL** (http://www.postgresql.org/). You might enter raw sales figures into the database and then use a language called **Structured Query Language (SQL)** to aggregate sales by product and date in order to produce a report.

2.3.2 Desktop Applications

The Linux ecosystem has a wide variety of desktop applications. You can find games, productivity applications, creative tools, and more. This section is a mere survey of what's out there, focusing on what the LPI deems most important.

Before looking at individual applications, it is helpful to look at the desktop environment. A Linux desktop runs a system called **X-Windows**, also known as **X11**. The Linux X11 server is **X.org**, which provides a way for software to operate in a graphical mode and accept input from a keyboard and a mouse. Windows and icons are handled by another piece of software called the *window manager* or *desktop environment*. A window manager is a simpler version of desktop environment as it only provides the code to draw menus and manage

the application windows on the screen. A desktop environment layers in features like login windows, sessions, a file manager, and other utilities. In summary, a text-only Linux workstation becomes a graphical desktop with the addition of X-Windows and either a desktop environment or a window manager.

Window managers include **Compiz**, **FVWM**, and **Enlightenment**, though there are many more. Desktop environments are primarily **KDE** and **GNOME**, both of which have their own window managers. Both KDE and GNOME are mature projects with an incredible amount of utilities built against them, and the choice is often a matter of personal preference.

The basic productivity applications, such as a word processor, spreadsheet, and presentation package are very important. Collectively they're known as an *office suite*, largely due to Microsoft Office who is the dominant player in the market.

OpenOffice (sometimes called OpenOffice.org, http://www.openoffice.org/) and **LibreOffice**(https://www.libreoffice.org/) offer a full office suite, including a drawing tool that strives for compatibility with Microsoft Office both in terms of features and file formats. These two projects are also a great example of how politics influence open source.

In 1999 Sun Microsystems acquired a relatively obscure German company that was making an office suite for Linux called **StarOffice**. Soon after that, Sun rebranded it as OpenOffice and released it under an open source license. To further complicate things, StarOffice remained a proprietary product that drew from OpenOffice. In 2010 Sun was acquired by Oracle, who later turned the project over to the Apache Foundation.

Oracle has had a poor history of supporting open source projects that it acquires, so shortly after the acquisition by Oracle the project was forked to become LibreOffice. At that point there became two groups of people developing the same piece of software. Most of the momentum went to the LibreOffice project which is why it is included by default in many Linux distributions.

For browsing the web, the two main contenders are **Firefox** (http://www.mozilla.org) and **Google Chrome** (http://www.google.com/chrome) . Both are open source web browsers that are fast, feature rich, and have excellent support for web developers. These two packages are a good example of how diversity is good for open source – improvements to one spur the other team to try and best the other. As a result, the Internet has two excellent browsers that push the limits of what can be done on the web and work across a variety of platforms.

The Mozilla project has also come out with **Thunderbird**, a full featured desktop email client. Thunderbird connects to a POP or IMAP server, displays email locally, and sends email through an external SMTP server.

Other notable email clients are **Evolution** (https://projects.gnome.org/evolution/) and **KMail**(http://userbase.kde.org/KMail) which are the GNOME and KDE project's email clients. Standardization through POP and IMAP and local email formats means that it's easy to switch between email clients without losing data. Web based email is also another option.

For the creative types, there is **Blender** (http://www.blender.org/), **GIMP** (http://www.gimp.org/), and **Audacity** (http://audacity.sourceforge.net/) which handle 3D movie creation, 2D image manipulation, and audio editing respectively. They have had various degrees of success in professional markets. Blender is used for everything from independent films to Hollywood movies, for example.

2.3.3 Console Tools

The history of the development of UNIX shows considerable overlap between the skills of software development and systems administration. The tools that let you manage the system have features of computer languages such as loops, and some computer languages are used extensively in automating systems administration tasks. Thus, one should consider these skills complementary.

At the basic level, you interact with a Linux system through a *shell* no matter if you are connecting to the system remotely or from an attached keyboard. The shell's job is to accept commands, such as file manipulations and starting applications, and to pass those to the Linux kernel for execution. Here, we show a typical interaction with the Linux shell:

```
sysadmin@localhost:~ $ ls -l /tmp/*.gz
-rw-r--r-- 1 sean root 246841 Mar  5  2013 /tmp/fdboot.img.gz
sysadmin@localhost:~ $ rm /tmp/fdboot.img.gz
```

The user is given a prompt, which typically ends in a dollar sign ($) to indicate an unprivileged account. Anything before the prompt, in this case *sysadmin@localhost:~*, is a configurable prompt that provides extra information to the user. In the figure above, *sysadmin* is the name of the current user, *localhost* is the name of the server, and ~ is the current directory (in UNIX, the tilde symbol is a short form for the user's home directory). We will look at Linux commands in more detail in further chapters, but to finish the explanation, the first command lists files with the *ls* command, receives some information about the file, and then removes that file with the *rm* command.

The Linux shell provides a rich language for iterating over files and customizing the environment, all without leaving the shell. For example, it is possible to write a single command line that finds files with contents matching a certain pattern, extracts useful information from the file, then copies the new information to a new file.

Linux offers a variety of shells to choose from, mostly differing in how and what can be customized, and the syntax of the built-in scripting language. The two main families are the **Bourne shell** and the **C shell**. The Bourne shell was named after the creator and the C shell was named because the syntax borrows heavily from the C language. As both these shells were invented in the 1970's there are more modern versions, the **Bourne Again Shell** (Bash) and the **tcsh** (tee-cee-shell). Bash is the default shell on most systems, though you can almost be certain that tcsh is available if that is your preference.

Other people took their favorite features from Bash and tcsh and have made other shells, such as the **Korn shell** (ksh) and **zsh**. The choice of shells is mostly a personal one. If you can become comfortable with Bash then you can operate effectively on most Linux systems. After that you can branch out and try new shells to see if they help your productivity.

Even more dividing than the selection of shells is the choice of text editors. A text editor is used at the console to edit configuration files. The two main camps are **vi** (or the more modern **vim**) and **emacs**. Both are remarkably powerful tools to edit text files, they differ in the format of the commands and how you write plugins for them. Plugins could be anything from syntax highlighting of software projects to integrated calendars.

Both vim and emacs are complex and have a steep learning curve. This is not helpful if all you need is simple editing of a small text file. Therefore **pico** and **nano** are available on most systems (the latter being a derivative of the former) and provide very basic text editing.

Even if you choose not to use vi you should strive to gain some basic familiarity because the basic vi is on every Linux system. If you are restoring a broken Linux system by running in the distribution's recovery mode you are certain to have vi available.

If you have a Linux system you will need to add, remove, and update software. At one point this meant downloading the source code, setting it up, building it, and copying files on each system. Thankfully, distributions created *packages* which are compressed copies of the application. A*package manager* takes care of keeping track of which files belong to which package and even downloading updates from a remote server called a *repository*. On Debian systems the tools include **dpkg**, **apt-get**, and **apt-cache**. On Red Hat derived systems, you use **rpm** and **yum**. We will look more at packages later.

2.3.4 Development Tools

It should come as no surprise that as software built on contributions from programmers, Linux has excellent support for software development. The shells are built to be programmable and there are powerful editors included on every system. There are also many development tools available, and many modern languages treat Linux as a first class citizen.

Computer languages provide a way for a programmer to enter instructions in a more human readable format, and for those instructions to eventually become translated into something the computer understands. Languages fall into one of two camps: *interpreted* or *compiled*. An interpreted language translates the written code into computer code as the program runs, and a compiled language is translated all at once.

Linux itself was written in a compiled language called C. C's main benefit is that the language itself maps closely to the generated machine code so that a skilled programmer can write code that is small and efficient. When computer memory was measured in the Kilobytes, this was very important. Even with large memory sizes today, C is still helpful for writing code that must run fast, such as an operating system.

C has been extended over the years. There is C++, which adds object support to C (a different style of programming), and Objective C that took another direction and is in heavy use in Apple products.

The Java language takes a different spin on the compiled approach. Instead of compiling to machine code, Java first imagines a hypothetical CPU called the Java Virtual Machine (JVM) and compiles all the code to that. Each host computer then runs JVM software to translate the JVM instructions (called bytecode) into native instructions.

The extra translation with Java might make you think it would be slow. However, the JVM is fairly simple so it can be implemented quickly and reliably on anything from a powerful computer to a low power device that connects to a television. A compiled Java file can also be run on any computer implementing the JVM!

Another benefit of compiling to an intermediate target is that the JVM can provide services to the application that normally wouldn't be available on a CPU. Allocating memory to a program is a complex problem, but that's built into the JVM. This also means that JVM makers can focus their improvements on the JVM as a whole, so any progress they make is instantly available to applications.

Interpreted languages, on the other hand, are translated to machine code as they execute. The extra computer power spent doing this can often be recouped by the increased productivity the programmer gains by not having to stop working to compile. Interpreted languages also tend to offer more features than compiled languages, meaning that often less code is needed. The language interpreter itself is usually written in another language such as C, and sometimes even Java! This means that an interpreted language is being run on the JVM, which is translated at runtime into actual machine code.

Perl is an interpreted language. Perl was originally developed to perform text manipulation. Over the years, it gained favor with systems administrators and still continues to be improved and used in everything from automation to building web applications.

PHP is a language that was originally built to create dynamic web pages. A PHP file is read by a web server such as Apache. Special tags in the file indicate that parts of the code should be interpreted as instructions. The web server pulls all the different parts of the file together and sends it to the web browser. PHP's main advantages are that it is easy to learn and available on almost any system. Because of this, many popular projects are built on PHP. Notable examples include WordPress (blogging), cacti (for monitoring), and even parts of Facebook.

Ruby is another language that was influenced by Perl and Shell, along with many other languages. It makes complex programming tasks relatively easy, and with the inclusion of the Ruby on Rails framework, is a popular choice for building complex web applications. Ruby is also the language that powers many of the leading automation tools like Chef and Puppet, which make managing a large number of Linux systems much easier.

Python is another scripting language that is in common use. Much like Ruby it makes complex tasks easier and has a framework called Jingo that makes building web applications very easy. Python has excellent statistical processing abilities and is a favorite in academia.

A language is just a tool that makes it easier to tell the computer what you want it to do. A library bundles common tasks into a distinct package that can be used by the developer. ImageMagick is one such library that lets programmers manipulate images in code. ImageMagick also ships with some command line tools that enable you to process images from a shell and take advantage of the scripting capabilities there.

OpenSSL is a cryptographic library that is used in everything from web servers to the command line. It provides a standard interface so that you can add cryptography into your Perl script, for example.

At a much lower level is the C library. This provides a basic set of functions for reading and writing to files and displays, which is used by applications and other languages alike.

2.4 Understanding Open Source Software and Licensing

When we talk about buying software there are three distinct components:

- **Ownership** – Who owns the intellectual property behind the software?
- **Money transfer** – How does money change hands, if at all?

- **Licensing** – What do you get? What can you do with the software? Can you use it on only one computer? Can you give it to someone else?

In most cases, the ownership of the software remains with the person or company that created it. Users are only being granted a license to use the software. This is a matter of copyright law. The money transfer depends on the business model of the creator. It's the licensing that really differentiates *open source software* from *closed source software*.

Two contrasting examples will get things started.

With Microsoft Windows, the Microsoft Corporation owns the intellectual property. The license itself, the **End User License Agreement (EULA)**, is a custom legal document that you must click through, indicating your acceptance, in order to install the software. Microsoft keeps the source code and distributes only binary copies through authorized channels. For most consumer products you are allowed to install the software on one computer and are not allowed to make copies of the disk other than for a backup. You are not allowed to reverse engineer the software. You pay for one copy of the software, which gets you minor updates but not major upgrades.

Linux is owned by Linus Torvalds. He has placed the code under a license called **GNU Public License version 2 (GPLv2)**. This license, among other things, says that the source code must be made available to anyone who asks and that you are allowed to make any changes you want. One caveat to this is that if you make changes and distribute them, you must put your changes under the same license so that others can benefit. GPLv2 also says that you are not allowed to charge for distributing the source code other than your actual costs of doing so (such as copying it to removable media).

In general, when you create something, you also get the right to decide how it is used and distributed. **Free and Open Source Software (FOSS)** refers to software where this right has been given up and you are allowed to view the source code and redistribute it. Linus Torvalds has done that with Linux – even though he created Linux he can't tell you that you can't use it on your computer because he has given up that right through the GPLv2 license.

Software licensing is a political issue and it should come as no surprise that there are many different opinions. Organizations have come up with their own license that embodies their particular views so it is easier to choose an existing license than come up with your own. For example, universities like the Massachusetts Institute of Technology (MIT) and University of California have come up with licenses, as have projects like the Apache Foundation. In addition groups like the Free Software Foundation have created their own licenses to further their agenda.

2.4.1 The Free Software Foundation and the Open Source Initiative

Two groups can be considered the most influential forces in the world of open source: The **Free Software Foundation (FSF)** and the **Open Source Initiative (OSI)**.

The Free Software Foundation was founded in 1985 by **Richard Stallman (RMS)**. The goal of the FSF is to promote **Free Software**. Free Software does not refer to the price, but to the freedom to share, study, and modify the underlying source code. It is the view of the FSF that *proprietary software* (software distributed

under a closed source license) is bad. FSF also advocates that software licenses should enforce the openness of modifications. It is their view that if you modify Free Software that you should be required to share your changes. This specific philosophy is called *copyleft*.

The FSF also advocates against software patents and acts as a watchdog for standards organizations, speaking out when a proposed standard might violate the Free Software principles by including items like **Digital Rights Management (DRM)** that could restrict what you could do with the service.

The FSF have developed their own set of licenses, such as the GPLv2 and GPLv3, and the Lesser GPL licenses versions 2 and 3 (LGPLv2 & LGPLv3). The lesser licenses are much like the regular licenses except they have provisions for *linking* against non-Free Software. For example, under GPLv2 you can't redistribute software that uses a closed source library (such as a hardware driver) but the lesser variant allows this.

The changes between version 2 and 3 are largely focused on using Free Software on a closed hardware device which has been coined **Tivoization**. TiVo is a company that builds a television digital video recorder on their own hardware and used Linux as the base for their software. While TiVo released the source code to their version of Linux as required under GPLv2, the hardware would not run any modified binaries. In the eyes of the FSF this went against the spirit of the GPLv2 so they added a specific clause to version 3 of the license. Linus Torvalds agrees with TiVo on this matter and has chosen to stay with GPLv2.

The Open Source Initiative was founded in 1998 by **Bruce Perens** and **Eric Raymond (ESR)**. They believe that Free Software was too politically charged and that less extreme licenses were necessary, particularly around the copyleft aspects of FSF licenses. OSI believes that not only should the source be freely available, but also that no restrictions should be placed on the use of the software no matter what the intended use. Unlike the FSF, the OSI does not have its own set of licenses. Instead, the OSI has a set of principles and adds other licenses to that list if they meet those principles, called *Open Source licenses*. Software that conforms to an Open Source license is therefore *Open Source Software*.

Some of the Open Source licenses are the BSD family of licenses, which are much simpler than GPL. They merely state that you may redistribute the source and binaries as long as you maintain copyright notices and don't imply that the original creator endorses your version. In other words "do what you want with this software, just don't say you wrote it." The MIT license has much the same spirit, just with different wording.

FSF licenses, such as GPLv2, are also Open Source licenses. However, many Open Source licenses such as BSD and MIT do not contain the copyleft provisions and are thus not acceptable to the FSF. These licenses are called *permissive free software licenses* because they are permissive in how you can redistribute the software. You can take BSD licensed software and include it in a closed software product as long as you give proper attribution.

2.4.2 More Terms for the Same Thing

Rather than dwell over the finer points of Open Source vs. Free Software, the community has started referring to it all as Free and Open Source software (FOSS).

The English word "free" can mean "free - as in lunch" (as in *no cost*) or "free - as in speech" (as in *no restrictions*).

This ambiguity has lead to the inclusion of the word **libre** to refer to the latter definition. Thus, we end up with **Free/Libre/Open Source Software** (FLOSS).

While these terms are convenient, they hide the differences between the two schools of thought. At the very least, when you're using FOSS software, you know you don't have to pay for it and you can redistribute it as you wish.

2.4.3 Other Licensing Schemes

FOSS licenses are mostly related to software. People have placed works such as drawings and plans under FOSS licenses but this was not the intent.

A **Public domain license** means that the author has relinquished all rights, including the copyright on the work. In some countries, this is the default when the work is done by a government agency. In some countries, copyrighted work becomes public domain after the author has died and a lengthy waiting period has elapsed.

The **Creative Commons (CC)** organization has created the **Creative Commons Licenses** which try to address the intentions behind FOSS licenses for non software entities. CC licenses can also be used to restrict commercial use if that is the desire of the copyright holder. The CC licenses are:

- **Attribution (CC BY)** – much like the BSD license, you can use CC BY content for any use but must credit the copyright holder
- **Attribution ShareAlike (CC BY-SA)** – a copyleft version of the Attribution license. Derived works must be shared under the same license, much like in the Free Software ideals
- **Attribution No-Derivs (CC BY-ND)** – you may redistribute the content under the same conditions as CC-BY but may not change it
- **Attribution-NonCommercial (CC BY-NC)** – just like CC BY, but you may not use it for commercial purposes
- **Attribution-NonCommercial-ShareAlike (CC-BY-NC-SA)** – Builds on the CC BY-NC license but requires that your changes be shared under the same license.
- **Attribution-NonCommercial-No-Derivs (CC-BY-NC-ND)** – You are sharing the content to be used for non commercial purposes, but people may not change the content.
- **No Rights Reserved (CC0)** – This is the Creative Commons version of public domain.

The licenses above can all be summarized as ShareAlike or no restrictions, and whether or not commercial use or derivations are allowed.

2.4.4 Open Source Business Models

If you are giving your software away for free, how can you make money off of it?

The simplest way to make money is to sell support or warranty around the software. You may make money by installing the software for people, helping people when they have problems, or fixing bugs for money. You are effectively a consultant.

You can also charge for a service or subscription that enhances the software. The Open Source MythTV digital video recorder project is an excellent example. The software is free, but you can pay to hook it up to a TV listing service to know what time particular television shows are on.

You can package hardware or add extra closed source software to sell alongside the free software. Appliances and embedded systems that use Linux can be developed and sold. Many consumer firewalls and entertainment devices follow this model.

You can also develop open source software as part of your job. If you create a tool to make your life easier at your regular job you may be able to convince your employer to let you open source it. It may be a situation where you were working on the software while getting paid but licensing as open source would allow other people with the same problem to be helped and even contribute.

In the 1990's, Gerald Combs was working at an Internet service provider and started writing his own network analysis tool because similar tools at the time were very expensive. Over 600 people have now contributed to the project, called Wireshark. It is now often considered better than commercial offerings and has led to a company being formed around Gerald to support Wireshark and to sell products and support that make it more useful. This company was later bought by a large network vendor who supports its development.

Other companies get such immense value out of open source software that they find it worth their while to hire people to work on the software full time. The search engine Google has hired the creator of the Python computer language, and even Linus Torvalds is hired by the Linux Foundation to work on Linux. The American telephone company AT&T gets such value out of the Ruby and Rails projects for their Yellow Pages property that they have an employee who does nothing but work for those projects.

One final way that people make money indirectly through open source is that it is an open way to judge one's skills. It is one thing to say you performed certain tasks at your job, but showing off your creation and sharing it with the world lets potential employers see the quality of your work. Similarly, companies have found that open sourcing non critical parts of their internal software attract the interest of higher caliber people.