2

Intellectual Property

Dominion Over Property

Software isn't *free*, as in the expression "the birds are free to fly." Software is someone's property, and you can't use another person's property—to fly or to do anything else—without that owner's permission.

And so this explanation of the law relating to software freedom actually starts with the other side of the coin, property rights.

Most people think of property as something tangible, discernible by touch. We exercise dominion over tangible land and call it our *real property*. We put personal things on our land and call that tangible stuff our *personal property*. We expect to have wide-ranging rights to use our property for our own benefit and enjoyment, with minimal interference from others. We assert that we *own* our property, and we often have the deeds or purchase receipts to prove it. We believe we have the right to prevent others from trespassing upon or taking our property.

In common usage, we also treat computer software as tangible personal property. We go to stores to *buy* software and pay for it with the same credit card we use to buy mouse pads in the next aisle. We take our new software home, put it in our computer, and it does our bidding.

But this concept of software as personal property is incomplete. There is much more to software than the disk it comes on. As one California court wrote in 1948, property is a very broad concept that includes not only the tangible but also "every intangible benefit and prerogative susceptible of possession or disposition." Computer software is this kind of intangible property because, under the law, it comes with specific but intangible benefits and prerogatives that can be separately owned and disposed of.

Software is a product of human intellect, and therefore it is a kind of *intellectual property*. Intellectual property is a valuable *property* interest, and the law allows its owner to possess and control it. The programmer who writes software—or the company that hires that person to write software—is deemed to be the first owner of intellectual property embodied in that software. That owner may exercise dominion over that intellectual property. He can give it away, sell it, or license others to use it. That owner has the prerogative to create copies of the intellectual property, and he or she may prevent others from making, using, or selling those copies.

Because of these partly tangible and partly intangible aspects of computer software, it is possible to have different owners own (1) a tangible copy of software purchased at a computer store or downloaded from a website, and (2) the intellectual property embodied in that software.

Never confuse these two aspects of intellectual property, for the laws apply differently to each.

In most respects, intellectual property law is very different from the law of both real and tangible personal property but, in at least one respect, the laws are similar. An owner of any form of real or personal property, including intellectual property, may sell or gift it, dispose of it upon his death by will or trust, or have it taken from him by a bankruptcy court. I will discuss the effects on open source software of the laws of disposition of property at the appropriate places in this book.

The first task, however, is to identify the varieties of intellectual property that can be embodied in software. That will help explain why the owners of intellectual property in software do not have unlimited rights to its exploitation and use, but they often have enough rights to protect their property from unauthorized exploitation by others.

Right Brain and Left Brain

Art is said to be the product of our right brain, the right hemisphere of our cerebral cortex that supposedly controls feelings and emotions. Scientific creations, it is said, are the product of our left brain, the left hemisphere that uses logic. Whether true or not, this bicameral description of the two products of human intellect—art and science—is useful to help us understand what we do when we create software.

Intellectual property law distinguishes these two kinds of intellectual creations. Our right brain creations are in the nature of *expression*, most often found in painting, music, fiction, and poetry. Our left brain creations are in the nature of *idea*, found in our scientific and technical innovations. Expressions are subject to *copyright* law; ideas are subject to *patent* law. (A third form of intellectual property, *trademark*, will be discussed later.)

The boundary line between expression and idea is very fuzzy in computer software. There may be two hemispheres, but there is one brain, and ultimately the software products of our creative intellect are simultaneously art and science, simultaneously expression and idea.

I remember, for example, while a graduate computer science student reading Donald Knuth's *The Art of Computer Programming*, coming to appreciate that his programs (and a few of mine) were truly works of art in ways sometimes unrelated to the functions they performed. The way Knuth expressed a particular algorithm, for example, became an object of beauty to that young computer programmer. Only someone who has written a tight computer program that does something well can appreciate how much expression goes into writing a piece of software and how emotionally rewarding that creative process can be simply because of the elegance and precision of the code.

Soon after that, I began to write software for Stanford University. As I became immersed in the practical world of grant proposals, teaching, and other university activities, I realized that the functions performed by my programs were far more important to my customers than the beauty of my code. Still later, when I moved into the high technology industry and began to worry about how commercial products are designed, manufactured, distributed, and supported, the *art* of computer programming became less and less relevant. What was essential were the functions that the software performed, the *ideas* that it implemented.

Truth be known, both perspectives are correct. When we create software, we create both *copyrightable expressions* and *patentable ideas*. The best functioning software is often the best-written software. Elegant source code usually leads to elegant software that does amazing things.

The law didn't originally allow software to be treated as intellectual property, and neither copyright nor patent laws applied to software. Finally, after much debate, in 1980 Congress decided that software should be copyrightable, and in 1981 the U.S. Supreme Court decided that software-enabled inventions should also be patentable. Federal courts and the U.S. Patent Office have since broadened patent coverage of software to include *computer readable media that store software*. This means that software is patentable. Some still complain about those decisions, but that's the law, at least in the United States.

Other countries have similar laws and policies. Readers in other countries are encouraged to ask their local attorneys for legal advice about specific differences, since some countries do not allow some kinds of software to be patented.

A low level of expressive creativity is sufficient to create copyrightable software, but the standards for obtaining a patent in software are substantially higher. Notwithstanding that difference, the laws of copyright and patent do not require that all art be at the standard of Picasso or that all ideas be at the level of Einstein. Quality of expression and profundity of idea are the province of art critics and the marketplace. To obtain a copyright you must simply be an author of an original work; to obtain a patent you must merely be the first inventor of something new, useful, and unobvious.

Acquiring Copyrights and Patents

Copyright is said to subsist in an original work of authorship. An author need not undertake any formal act—other than the act of original creation and fixation—to obtain a copyright. This applies to software as well. Any original software that is written down is automatically protected by copyright.

Formalities still may be useful. You should mark an original work with a copyright notice in the form:

© *Copyright <year> <author>*

Such a notice is no longer required to obtain a copyright, but it provides added protection. If you mark your software with a prominent copyright notice, a defendant can't argue that he was unaware who owned the copyright on the work. Registering a copyright isn't strictly necessary to have a copyright, but registration *is* required to initiate litigation to enforce the copyright. Furthermore, early registration provides added protection in the form of statutory damages and attorneys' fees if litigation becomes necessary to enforce the copyright. If it becomes important to do so, registration involves filling out a short form and paying a small fee (currently $30) to the Library of Congress (similar processes apply in other countries). But as a matter of law and international treaty, neither a copyright notice nor registration is required *to have* a copyright. Copyright merely subsists.

For the most part, because of international treaties, a copyright in one country is a copyright in all countries.

Obtaining patents is far more time-consuming and expensive. An application must be submitted to the patent office of each country (or group of countries) where patent protection is sought, describing with specificity the invention being claimed. Trained patent examiners review the patent application and the prior art to determine whether the claimed invention meets patentability standards. If it passes tests of novelty and unobviousness—and other legal tests relating to patentability—a patent will be issued. Even then, a patent certificate from the government provides only a presumption of validity, a presumption that can be challenged in court.

Just as with copyright notices, there are advantages to marking products with patent notices identifying specific patents. Even the phrase "Patent Pending" can be useful to alert others that patent protection is being sought. Patent notices are not a mandatory prerequisite to patent enforcement, but using them may allow a patent owner to obtain damages for infringement starting prior to the date of filing of an infringement lawsuit.

Anyone who owns a copyright or patent may license the intellectual property rights to others.

Original Works of Authorship

Open source software always starts with one or more original authors and their original works of authorship. Copyright law describes an original work in the following broad terms:

> *Copyright protection subsists ... in original works of authorship fixed in any tangible medium of expression, now known or later developed, from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device.... (17 U.S.C. § 102.)*

Understanding this statute may be easier if you initially broaden your perspective beyond computer software. An original work of authorship can be many things, including literary works; musical works; dramatic works; pantomimes and choreographic works; pictorial, graphic, and sculptural works; motion pictures and other audiovisual works; sound recordings; and architectural works.

Here's one example: The tune that Windows plays when it first loads is copyrightable. The author of that tune (or the author's employer) owns the copyright. The fact that it is computer software that communicates that tune through your computer speakers is irrelevant. The fact that the musical score

resides not on sheet music in your piano bench but as bits and bytes on your computer disk is irrelevant. Any original work of authorship, including that ubiquitous tune announcing the start of Windows, is copyrightable.

The source code that defines a computer program is copyrightable, as is the translated object code that actually executes on the computer. It makes no difference whether the program actually works. It makes no difference what programming language was used. Software is copyrightable if it is fixed on paper, on a disk drive or a CD-ROM, or even (for those who remember those technologies) on paper tape and punched cards. When future storage mechanisms are invented, software stored on those will also be copyrightable.

Original works of authorship include things that can be "perceived, reproduced, or otherwise communicated." Thus programs downloaded from the Internet are copyrightable, as are music, movies, photographs, and any form of literary work.

Because an author of an original work of authorship may transfer the copyright by sale, gift, will, or trust, it is sometimes more appropriate to refer to the owner of a copyright rather than simply to the author of the work. Whether the copyright is held by the original author or by a successor in interest, I will often refer to that person in what follows as the copyright owner.

Works Made for Hire

Not every author is the owner of his or her original works of authorship. Many works are prepared by employees within the scope of their employment; those are *works made for hire*. In most countries, such works are owned by the employer. It is the employer who can decide whether or how to dispose of the

work. The employee has no right, title, or interest in the work once the work is done. Here's what the U.S. Copyright Act says:

> *In the case of a work made for hire, the employer or other person for whom the work was prepared is considered the author..., and, unless the parties have expressly agreed otherwise in a written instrument signed by them, owns all of the rights comprised in the copyright. (17 U.S.C. § 201[b].)*

Sometimes employees create software on their own time using their own computers, software that has nothing to do with their real jobs as employees. In the United States, as long as writing that software is outside the scope of his or her employment, the employee owns the software and can dispose of it as he or she wishes.

Copyright law deals with works for hire differently in different jurisdictions, and even within the United States each state has different rules concerning ownership of employees' creations. Be careful to consult an attorney.

Not everyone who writes software for someone else is an employee. Many programmers are independent contractors who move from company to company, or from assignment to assignment, writing software on demand. In most jurisdictions, the copyrights to original works prepared by contractors are owned by the contractors themselves, unless there is a written agreement between the parties specifying otherwise.

What happens if there is no written contract? In this situation, even though a contractor owns the copyrights to software written for someone else, the person who hired the contractor to write the software will be entitled to a nonexclusive license to use the software for its intended purpose. That is because he or she paid for the work; otherwise, contractors could hold their software hostage from the very companies that paid to have it developed.

The default law regarding ownership of employee and contractor inventions in the absence of a contract varies from jurisdiction to jurisdiction. Most companies protect their own interests by executing written invention agreements with their employees and contractors in order to contractually ensure that the rules of ownership for patents are similar to those for copyrights. If you want to be sure to own your creations, consult an attorney.

Exclusive Rights of Copyright and Patent Owners

Intellectual property, like other forms of personal property, is characterized by the things that nobody else can do without the owner's permission. If you own an automobile, for example, only you can drive it—unless you give others permission to do so. It is your prerogative to do what you want with your automobile, including keeping it in your garage for private showings if you are so inclined.

So too, if you own a copyright, you have an exclusive right to do certain things with your copyrighted intellectual property that others cannot do without your permission:

- You have an exclusive right to make copies.
- You have an exclusive right to prepare derivative works.
- You have an exclusive right to distribute copies of the original work or derivative works.
- In the case of certain kinds of works, including literary, musical, and motion picture works, you have an exclusive right to perform the work publicly.

- In the case of certain kinds of works, including literary, musical, pictorial, and sculptural works, you have an exclusive right to display the work publicly.

This list of exclusive rights is found in the U.S. Copyright Act, 17 U.S.C. § 106. A similar list is found in the copyright laws of most countries.

If you own a patent, you have a right to exclude others from doing certain things with your patented intellectual property:

- You have a right to exclude others from making products embodying your patented invention.
- You have a right to exclude others from using products embodying your patented invention.
- You have a right to exclude others from selling or offering for sale products embodying your patented invention.
- You have a right to exclude others from importing products embodying your patented invention.

This list of rights is found in the U.S. Patent Act, 35 U.S.C. § 154. A similar list is found in the patent laws of most countries.

You may have noticed that I described the rights of copyright and patent in two different ways. In the case of copyright, the owner *has an exclusive right to do* certain things; in the case of a patent, the owner *has a right to exclude others from doing* certain things.

This is an important distinction. Because copyright involves the affirmative act of creating an original work of authorship,

it is a simple matter to determine if someone has copied, modified, or distributed that work. The copyright owner has an exclusive right to do those things, and he or she may license those rights to others.

However, the owner of a patent does not necessarily have the exclusive right to practice his own patented invention because someone else may have invented a necessary prerequisite or broader invention. The most that a patent owner can do is prevent someone else from practicing his or her invention. The patent owner usually can't guarantee that his or her own patents are sufficient to make, use, sell or offer for sale, or import the software. Additional patent rights from third parties may be necessary.

This difference manifests itself in open source licenses by the language of the copyright and patent grants. The copyright grant is an affirmative license to copy, modify, or distribute the software owned by the licensor. The patent grant is an affirmative license to practice patents necessary to make, use, sell or offer for sale, or import the software, but only to the extent of patent claims actually owned or controlled by the licensor. Additional third-party patent rights may interfere with the right to do things with the software, and the licensor does not have authority to grant that broader license.

Copies

The author of an original work of authorship (e.g., the owner of the copyright) has the exclusive right to make (or not make) *copies* of a copyrighted work. Others must seek the permission of the author, given in legal form by a license, before they may make copies.

All of the open source licenses in this book grant an unlimited right to create copies.

> *"Copies" are material objects ... in which a work is fixed by any method now known or later developed, and from which the work can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device. The term "Copies" includes the material object ... in which the work is first fixed. (17 U.S.C. § 101.)*

Technology is always at least one step ahead of the copyright law, so the word *copies* isn't limited to photocopies, or to CD-ROM duplicates, or even to binary images fixed for a time in a computer's memory. Any method of copying, now known or later developed, can be used to create a copy and still meet the definition in the law.

The original of a work is merely the first copy. Any duplicate made from it, by any means, is a copy. Every instance of computer software, as long as it is fixed in some tangible form, is a copy.

The copyright owner of software has the exclusive right to make, or to allow others to make, copies of that software.

Exceptions to the Exclusive Right to Make Copies

There are two important exceptions under the law to the rule that the copyright owner has an exclusive right to make (or not make) copies of that work. These exceptions apply only to computer software and it reflects the unique nature of that technology.

First, everyone understands that software can be used on a computer only if it is copied onto a computer disk or into memory. Therefore, the authorized owner of a copy of software is given the right to make a copy of that software "as an essential step to the utilization" of the software "in conjunction with a machine." (17 U.S.C. § 117.) Without this exception, anyone purchasing a software program at a computer store would

require an additional license to copy it to a hard disk and to the computer's memory, clearly a wasteful burden for someone who merely wants to run a program he or she *bought*.

Second, software is effective only if there is a way to back it up for archival purposes. Therefore, the authorized owner of computer software is given the right to make archival copies, with the added requirement that those archival copies must be destroyed in the event that continued possession of the computer software "should cease to be rightful." (17 U.S.C. §117.)

These limited exceptions are not intended as wedges into which to drive a high-speed copy machine. These exceptions only apply to an authorized owner of a copy, someone who has a license from the copyright owner. These exceptions in the law to the exclusive rights of a copyright owner to make copies do not excuse the making of other copies not intended for these limited purposes.

Collective and Derivative Works

The terms *collective works* and *derivative works* will be the subject of more rigorous explanation later in the book. For now, it is important only to understand these terms in the context of open source software, as a way of describing what the participants in open source development and licensing actually do.

Before I define these terms, note one thing: Collective works and derivative works are also original works of authorship, and copyright subsists in them. (17 U.S.C. § 103.)

> *The subject matter of copyright as specified by section 102 includes compilations and derivative works.... (17 U.S.C. § 103.)*
>
> *The term "compilation" includes collective works. (17 U.S.C. § 101.)*

A collective work is:

> *...A work ... in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole. (17 U.S.C. § 101.)*

In the nonsoftware context, think of a collective work as an encyclopedia or an anthology. In the software context, a collective work is usually an aggregation of separately written software that is distributed as a single package or on one disk. An office productivity suite, for example, may contain separately written components such as a word processor, a spreadsheet program, and an email client. Each of those components is an original work of authorship as is the collective office suite as a whole.

The copyright in a collective work is a reflection of the originality of the collection and its organizational structure rather than of the individual components. Most software is a copyrightable collection of modules. The arrangement and organization of the collection of individual modules are often the most original aspects of a software program.

A derivative work is:

> *...A work based upon one or more preexisting works, such as a translation...or any other form in which a work may be recast, transformed, or adapted. (17 U.S.C. § 101.)*

Since there are so many varieties of derivative works, the statute merely lists examples of derivative works, including translations, editorial revisions, elaborations, modifications, or any other form in which a work may be recast, transformed, or adapted. This leaves it for the courts to sort out whether a specific work is or is not a derivative work. How the courts do that is the topic for much later in this book. For now, in the software context, think of derivative works as programs that have been improved or enhanced from earlier versions of a

program. Distributors of open source software often create successive versions containing improvements contributed by many programmers. Those successive versions are derivative works of earlier versions, and each such version is itself an original work of authorship.

It may be helpful to view ownership of open source software as being represented by a *chain of title*. An original work of authorship is the first link in the chain. That chain is elongated during the collaborative open source development process. People take original works of software, aggregate them with other such works, and make modifications, in the process creating collective and derivative works—each a new original work of authorship.

Title to each successive aggregation or modification is subject to the ownership rights of the copyright owners of the previous contributions and modifications, as each new derivative or collective work forges the next link in the chain of title.

Software improves through such aggregation and modification. This dynamic, fluid evolution of expressions and ideas in the open source community, manifested by evolving collective and derivative works, results in the creation of ever more powerful software. That process is described eloquently in Eric Raymond's book, *The Cathedral and the Bazaar*. Its observations and predictions about software quality have been proven applicable in a wide variety of open source projects. All this has been made possible by the free creation of collective and derivative works authorized by open source licensors.

The Chain of Title for Copyright

Collective and derivative works are entitled to copyrights as original works of authorship, but that doesn't mean that those

copyrights replace the earlier copyrights on the component parts. Here's how the Copyright Act describes it:

> *The copyright in a compilation or derivative work extends only to the material contributed by the author of such work, as distinguished from the preexisting material employed in the work, and does not imply any exclusive right in the preexisting material. The copyright in such work is independent of, and does not affect or enlarge the scope, duration, ownership, or subsistence of, any copyright protection in the existing material. (17 U.S.C. § 103[b].)*

Mature open source projects often consist of software passed through many such stages of aggregation and modification, their original works of authorship proudly displaying a long chain of title including the names of many individuals and organizations that preceded them.

The term *chain of title* is most frequently used to describe ownership of real property in the United States. Starting with the original land grant from the King of England (and usually ignoring completely the previous rights of the Native Americans who long preceded the king), it is possible to trace ownership of each parcel of land through the generations. As land is divided, easements are granted, and children inherit from their parents, title to the land passes from one owner to another. The current owner of the land holds that land subject to the restrictions and covenants agreed to by his forebears.

The chain of title becomes important in open source licensing when someone wants to create a collective or derivative work of a previous work that itself consists of contributions by many people. The new authors are subject to the licenses of previous authors who preceded them, and each of those contributions may have different license restrictions on its use.

That ever lengthening chain of title would appear to be an increasing burden on future generations of software developers, but the problem is not nearly so complex. Depending upon the open source licenses being used, it may only be necessary for new authors to ensure that they have licenses from their immediate predecessors and not all the way back to the first programmer writing the first version of the original contribution that started the chain.

The Chain of Title for Patents

Once again, a patent is a right to exclude others from making, using, selling or offering to sell, or importing a specific claimed invention. An inventor writes his claim in precise terms in a patent application and then a patent examiner reviews the claim for patentability. Only claims that meet a legal standard will be approved. The legal standard for patentability involves arcane criteria of novelty and unobviousness for which a qualified attorney is often indispensable. Upon approval of the patent application, the inventor (or his assignee) receives a limited monopoly right to prevent unauthorized practice of his patent.

A patent differs from a copyright in a fundamental way: A copyright prevents a third party from copying or modifying the original work, but a patent restricts everyone who uses the patented invention whether the invention has been copied or not. Even someone who independently creates the same invention and doesn't copy the first inventor still cannot make, use, sell or offer for sale, or import the patented invention because he's not the first inventor. It makes no difference whether the second inventor even knew of the first invention.

As with any other form of intellectual personal property, patent rights can be sold or given away, inherited, lost in bank-

ruptcy, or licensed. In that sense, there can be a chain of title for a patent just like there is for a piece of real property. It is often possible to trace that chain of title using public documents on file with the government patent office.

Unlike copyright rights to *collective* and *derivative* works that are subject to the prior licenses for each of the contributions and modifications that preceded them, a patent has only one current owner we must worry about. (There may actually be multiple inventors or owners of a patent, or different owners of the exclusive rights in a patent. For our purposes, we can treat those multiple owners as one person.) There is no concept of a collective or derivative work in patent law. One either infringes a patent or one doesn't.

Before you can implement a patent claim in software, you need to determine who actually owns the relevant patent rights and whether you have a license to practice it. The patent owner may be the original author of the copyrighted work from which you're creating a collective or derivative work, but it may also be someone entirely different, perhaps someone neither you nor the copyright owner ever heard of before.

It is a complex and enormously expensive task to find all relevant patent claims and analyze them to determine whether you have the right to make, use, sell or offer for sale, or import your software. It is no wonder that most software authors—open source ones and proprietary ones—don't devote the time or money needed to undertake that search and analysis. They often merely wait to be surprised by bad news. If a patent claim by a third party is asserted against your software, you can simply stop using the patented invention, or challenge its validity in the patent office or in court. Another obvious choice is to seek a license to the patent.

A patent license can be narrow or broad, specific to a particular implementation, or broad enough to cover any possible

implementation of the patent. Depending upon the specific terms of the patent license, it may not include the right to implement the patent in a collective or derivative work. There is no *free software* or *open source* definition for a patent license, and so each license must be analyzed to determine whether its terms are compatible with such software.

At least in theory, you must obtain a license from any patent owner whose patents are practiced in any software you make, use, sell or offer for sale, or import. In practice, hardly anyone bothers until it is too late. As I discuss various open source licenses later in this book, I will explain how each license handles—or doesn't handle—this potential patent problem.

Joint Works

Open source prides itself on being a cooperative development process. Communities of engineers work together over the Internet to write software. In this way, they may create collective works. But they may also, without realizing the difference, create an entirely different kind of work: The result of collaborative development may become a *joint work* rather than a *collective work*.

> *A "joint work" is a work prepared by two or more authors with the intention that their contributions be merged into inseparable or interdependent parts of a unitary whole. (17 U.S.C. § 101.)*

Obviously, joint authors of a work don't have to collaborate on each word of the final product. They can divide their activities to create a unified work—perhaps chapter by chapter, perhaps plot line by plot line, perhaps one writes the music and the other the words, perhaps they cooperate in more subtle ways. They may intentionally decide not to reveal which

author did which portion of the work. Joint authors usually manifest their intent to create a joint work by documenting in a contract between them the specific relationship that they intend to forge while working together on the work. Proof that something is a joint work requires proof of the intention of the authors, but that proof isn't always easy to provide for the authors who contribute to informal open source projects.

There is a very important legal difference between a collective work and a joint work. Each contribution to a collective work is owned by its author, and that author has the exclusive right to decide how that contribution is to be licensed. A contribution to a joint work is owned by all of its authors jointly.

In the United States, unless they agree otherwise, each of the joint authors may separately license a joint work—and all of its parts—without the consent of any of the other joint authors, and every author must account to the other authors for their share of the profits derived from the license. Consult local law to determine whether one owner of a joint work may license without the consent of the others or must account to the others for his or her licensing revenue.

For most projects, whether the software is a collective work or a joint work will be unimportant as long as the contributors all continue to agree on a licensing strategy. Only when disagreements occur and the licensing strategy is to be changed—what in open source circles is called *relicensing*—does it matter how the parties formally agreed to collaborate.

Relicensing a joint work is, in some ways, easier than relicensing a collective work because any one of the authors can do it without consulting the others, but it may leave some contributors angry with the results.

Assigning Ownership

This book is about licensing, but there is an alternative to licensing that is occasionally employed by open source projects to ensure that the projects themselves have the right to license contributions. Authors are encouraged to *assign* their entire ownership interest in open source software (and occasionally the ownership interest in any patents embodied in that software) directly to the project. This *assignment* is an effective way to ensure that the project itself has the authority to license the software.

You will recall that the owner of intellectual property may dispose of it as if it were real or personal property, including by sale or gift. Once transferred to a new owner, it is the new owner who has the exclusive rights described in this chapter.

This technique of copyright assignment is generally neither useful nor necessary, because an open source license can convey all rights as effectively as an assignment. There are only a few limited occasions when an assignment is preferable.

First, as I shall explain more fully in Chapter 12 on open source litigation, only the owner of a copyright, or an exclusive right under copyright, or the owner or exclusive licensee of a patent right (e.g., in an explicit territory or field of use) has the right to sue to enforce those rights or licenses. (17 U.S.C. § 501[b]; 35 U.S.C. § 281.) Second, since intellectual property is inheritable upon death of the owner, the owner may prefer to assign a valuable copyright or patent rather than burden his heirs with something they may not understand, appreciate, or know how to manage.

Copyright law in the United States requires that copyright assignments be in writing. (17 U.S.C. § 204[a].) Similar provisions apply to patent assignments. (35 U.S.C. § 261.) As an

exercise in legal drafting, an assignment usually includes the formalities needed to satisfy the writing and filing requirements of copyright and patent law.

One risk to the original author of assigning a copyright is that the author loses the right to license it yet again under different terms to different licensees. (I discuss dual licensing strategies in Chapter 11.) Once copyright ownership is assigned, the new owner has the exclusive right to decide on licensing strategies, and the original owner has no rights left (unless he or she receives a license-back, about which I will say nothing more in this book).

Another risk of assignment is that many open source projects have informal structures, often without a legal corporate entity behind them. Assigning a copyright to an informal entity leaves in doubt just who has the authority to commit to licensing decisions. Indeed, if a project makes licensing decisions that the original copyright owner dislikes, that original owner will have no legal basis to object and will be obligated to honor the express provisions of the written assignment that he or she signed.

Other than the infrequent situations described above, there is little advantage to open source projects to receive assignment of copyrights and patents. Everything that an open source project needs, including the rights to make copies, create derivative works, and distribute the software, is provided by any of the open source licenses described in this book as readily as by an assignment. Contributors and the open source projects that receive those contributions can usually accomplish their objectives with an open source license instead of an assignment.

Since a license accomplishes much the same thing in open source as an assignment, I will not bother describing the special language that would be needed for an assignment to make

it legally effective. Nor will I describe how to draft an assignment that includes a license-back to the original owner. These are questions best directed to your own attorney.

Duration of Copyright and Patent

There is another fundamental difference between most forms of real and personal property and the intellectual property embodied in software. Real and personal property rights generally last forever, but copyrights and patents are temporary ownership rights that terminate with the passage of time.

In the United States, the Constitution mandates that such rights shall be granted "for limited times," a particularly vague provision that allows Congress to define and change the terms of the copyright and patent monopolies, which it frequently does. Current U.S. law provides that, for new works, copyrights last for the life of the author plus 70 years or, for a work of corporate authorship, the shorter of 95 years from publication or 120 years from creation. New patents last for 20 years from the date the patent application is filed.

Upon expiration of the term of a copyright or patent, the intellectual property is said to pass into the *public domain.* The once exclusive rights of the owners of that intellectual property become available for exercise by anyone who wants them, freely and without charge.

The word *freely* is used here in a different way than when I was describing *software freedom* in the open source definition. Freedom under an open source license may be limited and conditioned by the copyright and patent owners. But once intellectual property enters the public domain, its owner can no longer restrict its exploitation and use in any way.

Through the passage of time, the intellectual works of Shakespeare, Mozart, and Newton have long since passed into

the public domain. That intellectual property is completely free for anyone to use. But the *free* intellectual property of Linux and Apache is still subject to the terms and conditions set by the owners of those original works of authorship, because those copyrights have not yet expired.

It is incorrect to suggest that open source licensing destroys intellectual property or is inconsistent with intellectual property laws. Quite the opposite. Open source software is owned by individuals and companies under the authority of the copyright and patent laws. Those owners license their software to the public. It is not public domain software. Or at least, it won't be public domain software until the copyrights and patents embodied in the software expire by the slow passage of time, as specified in the intellectual property laws.

Trademarks

It was presumptuous of me to suggest earlier in this chapter that the only two brains involved in creating successful software products are the right brain of copyright and the left brain of patent. This leaves out what is sometimes the most important brain of all—the one that captures consumer attention through effective marketing.

Often the keys to marketing success for open source projects are their product or brand names, or trademarks. More specifically, a trademark is a word, phrase, symbol, or design, or a combination of words, phrases, symbols, or designs that identify and distinguish the source of the goods of one person or company from those of others.

Trademarks are a form of intellectual property. Trademarks are owned, and they can be licensed. Consider, for example, the brand name *Linux*, a registered trademark owned by Linus Torvalds for:

> *Computer operating system software to facilitate computer use and operation. (U.S. Trademark # 1916230.)*

It is inevitable that more people recognize the Linux operating system by its trademark than would recognize even a single line of its copyrightable code or any patent claims it embodies. The success of Linux in the marketplace, while made possible by the underlying copyrightable and patentable subject matter, is largely now due to good brand recognition and the aura of accomplishment that the brand engenders in the public. As long as the contributors and distributors responsible for Linux software continue to focus on quality and reliability, the Linux brand name will prosper.

Other open source projects and companies also rely on trademark protection. Brand names such as Apache, MySQL, Open Office, JBoss, Red Hat, and Debian identify quality products to open source customers. And now that major software companies are becoming open source contributors and distributors, brand names like IBM, HP, Apple, Sun, Oracle, Novell, and Nokia adorn open source products.

As a matter of trademark law, a trademark would be lost if it were licensed under typical open source license terms. This is because a trademark owner must maintain control over the quality of the goods bearing his or her trademark when the trademark is licensed to others. But an open source licensor cannot control the quality of the licensees' derivative works. (Open Source Principle # 3.)

Because of that incompatibility between trademark law and open source principles, no open source license includes a trademark license. Some open source licenses even contain an explicit exclusion of trademark license. I will discuss such provisions in due course.

Exceptions to Intellectual Property Protection

Not everything in software is subject to copyright or patent protection. It is possible to write software that is not protected by either. For our present purposes, these are the two most important exceptions:

1. You cannot use copyright law to prevent someone from practicing the underlying ideas in the software. Copyright protects only expression. If there is only one way to implement an idea in software, anyone can copy the software unless it is also protected by patent (or by a trade secrecy restriction, something that never applies to open source software whose source code is published).
2. Before you can use patent law to prevent someone from practicing the underlying ideas in the software, you must actually apply for and obtain a valid patent. That can be both expensive and difficult. The validity of a patent can be challenged in court. If the author of the software doesn't have a patent, anyone can build equivalent software from scratch without asking the original author's permission.

These exceptions to copyright and patent, and a few others, often become important in intellectual property litigation. Authors of software always claim that they own intellectual property in the form of copyrights and patents, but at the end of the day, they may still have to prove in court that their software isn't one of those unprotected exceptions. I'll describe how that plays out in court when I discuss open source litigation in Chapter 12.