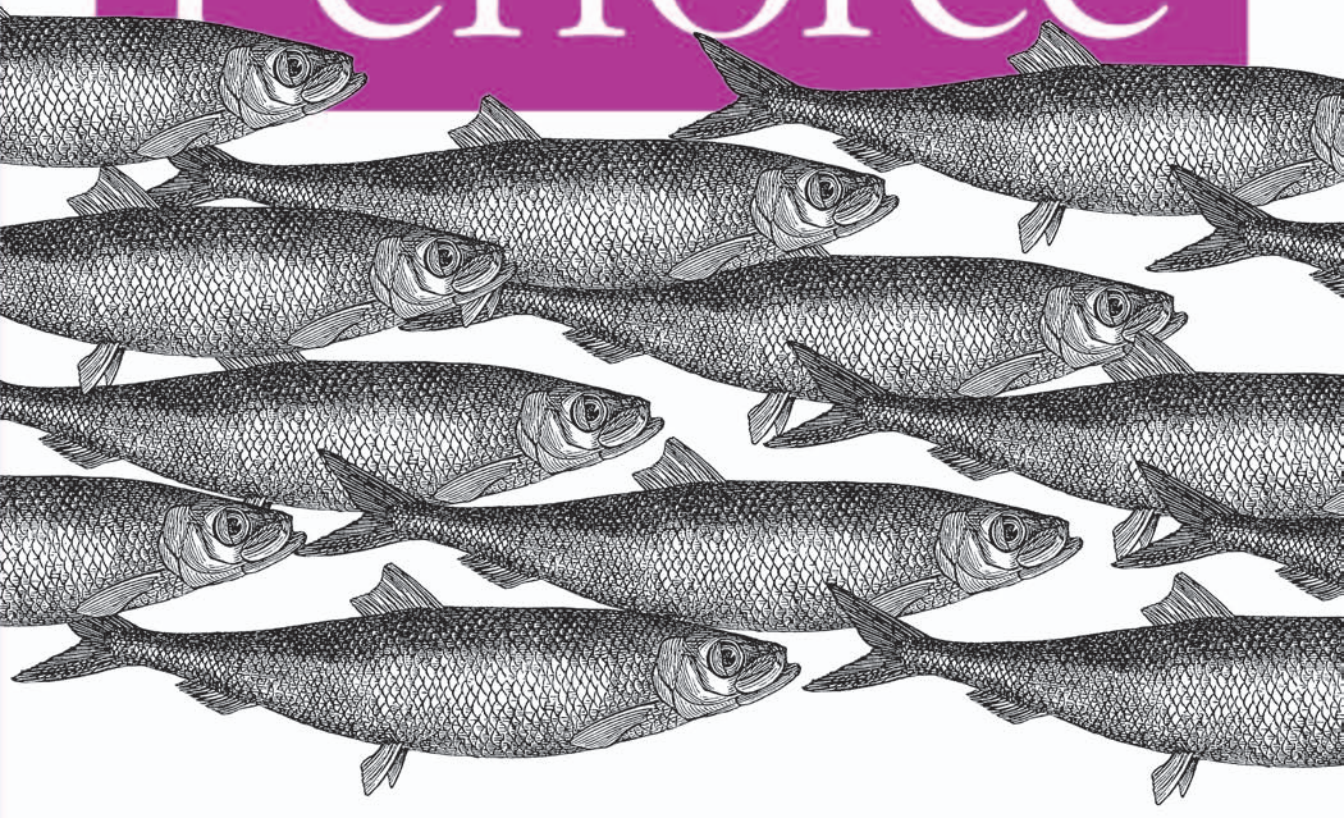
*Channeling the Flow of Change*
*in Software Development Collaboration*

# Practical Perforce



O'REILLY®

*Laura Wingerd*

O'REILLY®

# Practical Perforce

Good configuration management is critical to the success of any software project—indeed, to the success of any creative endeavor involving computer files. Where do you store your files? What do you do when several people edit the same file? How do you keep straight the code from multiple versions of a product? How do you explore a new idea without affecting current development? These are just some of the questions that any good software configuration management system must answer.

Perforce stands as one of the industry's preeminent software configuration management tools. It is hugely popular among its users, and once you've read this book you'll understand why. Yet tools are only as good as those who know how to use them. Author Laura Wingerd's goal with *Practical Perforce* is to take you to what she calls the "aha!" moment. To that end, this book:

- Goes beyond the official Perforce documentation to provide coverage of undocumented commands, explanations of esoteric internals, useful recipes, and clever ways to exploit Perforce
- Covers the big picture by telling you how to use Perforce commands—and who needs to use them and when—to shepherd software in development from inception to retirement
- Examines branching and merging in depth, arguably the hardest part of version control, by taking into account the consequences of refactoring, agile development, extreme programming, parallel development, emergency patches, and ongoing product maintenance

The "aha!" moment is when you really and truly understand what Perforce is, how it works, and how you can best take advantage of it. Don't settle for the ability to "just use" Perforce. Really learn the tool. Take time to understand its architecture. Learn the best practices. Take your skills at configuration management to the next level.

**Laura Wingerd** is the Vice President of Product Technology at Perforce Software. Her experience spans software configuration work at Relational Technology, Inc. (Ingres) and Sybase, and she has been with Perforce since the "garage days." Laura has written or cowritten many papers on configuration management, including the widely cited "High-Level Best Practices in Software Configuration Management."

**www.oreilly.com**

**US $39.95** **CAN $55.95**

ISBN-10: 0-596-10185-6

ISBN-13: 978-0-596-10185-5

9 780596 101855 53995

# Practical Perforce

*Laura Wingerd*

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

**Practical Perforce**
by Laura Wingerd



Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.



**Editor:** Jonathan Gennick

**Production Editor:** Adam Witwer

**Cover Designer:** Karen Montgomery

**Interior Designer:** David Futato

**Printing History:**

November 2005: First Edition.




This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 0-596-10185-6
[M]

# Table of Contents

# Preface

## What Is Perforce?

If you've picked up this book simply because of its riveting title, you may be wondering what Perforce is. Perforce is a *software configuration management* (SCM) system. SCM systems are used by software developers to keep track of all the software they build and all the components that go into it.

A good SCM system can explain the mysteries of software development and head off its disasters—mysteries like lost bug fixes, and disasters like botched file merges. In large-scale and commercial environments, good SCM is absolutely essential to producing good software.

### It's All Software and We're All Software Developers

SCM was once concerned with files that computer programmers produced. Now it is concerned with files of all types that a business produces. Software, when viewed from the perspective of SCM, is any endeavor that calls a computer home. Documention, web content, spreadsheets, schematics, graphics, sound—it's *all* software. If it's stored in computer files and gets built, embedded, or packaged into a deliverable result, it's software. The term *software developer* may not sound like it applies to web content authors, graphic artists, test engineers, and technical writers, but for the purpose of this book, anyone whose work involves creating computer files from intellectual thought is a software developer.

Perforce, like all SCM systems, keeps track of changes as people do concurrent, parallel work on files. It logs activity; reports who did what; compares, merges, and

branches files; and stores files and file configurations. Some of Perforce's most salient features are:

*The depot*
: Perforce stores files in a protected repository known as the *depot*. The depot is a centrally located, permanent archive of all file content submitted by users.*

*Workspaces*
: Perforce users work on files in *workspaces*, private disk areas of their own that contain copies of depot files. In this book we'll describe effective ways that developers can use workspaces, and we'll also discuss how workspaces can be used to automate nightly builds, release packaging, web staging, and other software production tasks.

*Changelists*
: Perforce *changelists* tie files changed together into single units of work. Every change to the depot can be traced to a changelist, and every changelist marks a known, reproducible state of the depot; the depot evolves as changelists are submitted. In the Perforce view of SCM, it is the changelist—not the file revision, nor the delta—that is the atomic transaction of software development. This book will discuss a variety of ways changelists can be used, including treating them as snapshots and using them to identify file dependencies.

*Filespecs and views*
: The Perforce *filespec* syntax, and the *views* that use it, allow selection of files for Perforce operations. Filespecs can define not only the common file collections, like directories, but arbitrary collections of files that constitute codelines, modules, delivery streams, and other containers. They are the key to treating collections of files as versioned objects that can be inspected, rolled back, branched, labeled, compared, and merged at any version.

*Jobs*
: In Perforce you can record externally defined tasks and states—bug reports, feature requests, and project milestones, for example—in objects it calls *jobs*. Jobs can be linked to changelists to provide a record of software changes related to tasks. Jobs are also the linchpin of any integration between Perforce and external systems, as we'll see in later chapters.

*Branching*
: Perforce uses *Inter-File Branching* to model file variants. In the traditional version tree branching model used by most SCM systems, a file can be branched and merged only into revisions of itself. In Perforce, any two files can have a branching relationship; branched files are peers, not offshoots, of their originals. A number of chapters in this book are dedicated to describing Perforce branching and its unexpectedly useful applications.

* The data format of the Perforce depot is not proprietary; it is, in fact, consistent with the RCS archive format. Because of this, there is a common misperception that Perforce is an RCS wrapper. It's not.

*Integration history*
: In Perforce, branching and merging are referred to as *integration*. Perforce records a history of integration events and uses it to direct merges and prevent unnecessary remerging. In this book you'll see how Perforce does that and learn how to anticipate the effect of merges you perform.

*Change tracking*
: Perforce combines filespecs, changelists, jobs, and integration history to track changes as they are merged from branch to branch. In this book you'll learn how these objects can be used to determine whether a change—a bug fix, for example—made in one branch has been merged to another, no matter how distantly related.

In addition to these features, which could be considered the interesting capabilities of Perforce, there are also the standard housekeeping and productivity features you're likely to find in any SCM system, including labels, triggers, change notification, graphical merge tools, file histories, and so forth.

## The Perforce System in a Nutshell

Perforce is a client/server system (see Figure P-1). The domain of a Perforce system encompasses a master file repository (the depot), a database, and a constellation of users running client programs. One Perforce Server typically serves an entire Perforce domain. Its job is to communicate with Perforce client programs, analyze and execute user commands, archive and serve up file content, run event triggers, and record system activity in the Perforce database. It also performs a variety of database housekeeping tasks, some on demand and some automatically.

The client component of Perforce, shown in Figure P-2, manages workspace files and communicates with the server. It's implemented in a variety of tools designed for users at almost every technical level.

Perforce client tools can be divided roughly into three categories:

*Graphical user interfaces*
: The Perforce GUIs are the point-and-click client applications. This category includes P4V, P4Win, and P4Web. (The latter is actually more of a plug-in, but because it turns your browser into a Perforce GUI it is marketed as a GUI itself.) Although they don't support every possible Perforce command, the GUIs do support the day-to-day operations of the typical software developer, and they are easy on the eyes. They also provide a variety of data-mining features, including some very nice visualizations of branching and file evolution. For these reasons, even die-hard command-line adherents find them useful. P4V, P4Win, and P4Web can be used interchangeably, although there are some variations in the range of operations they support. All three come with embedded help files that provide rudimentary coaching in how to use Perforce.

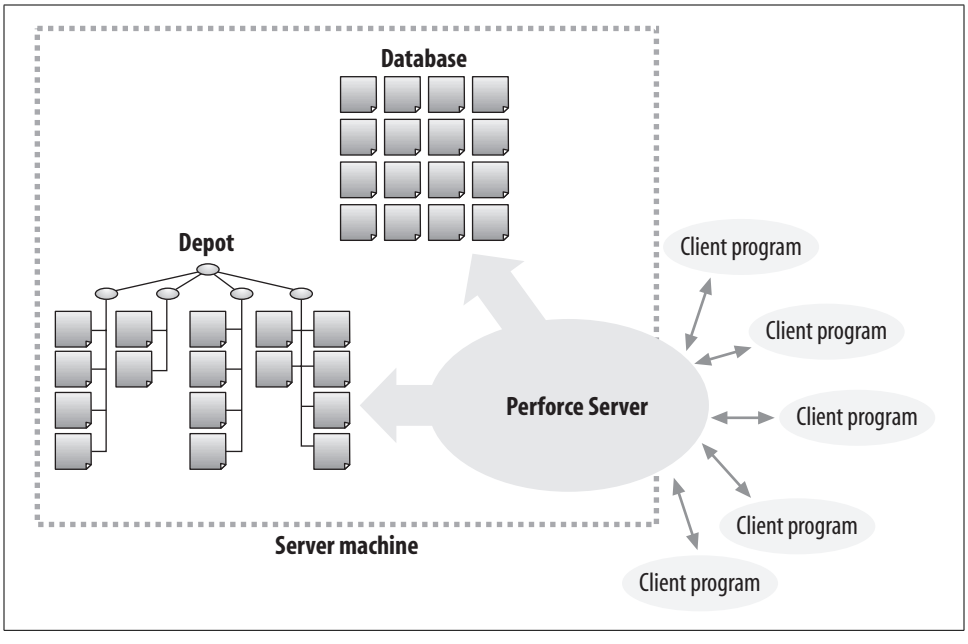



*Figure P-1. The Perforce client/server system*

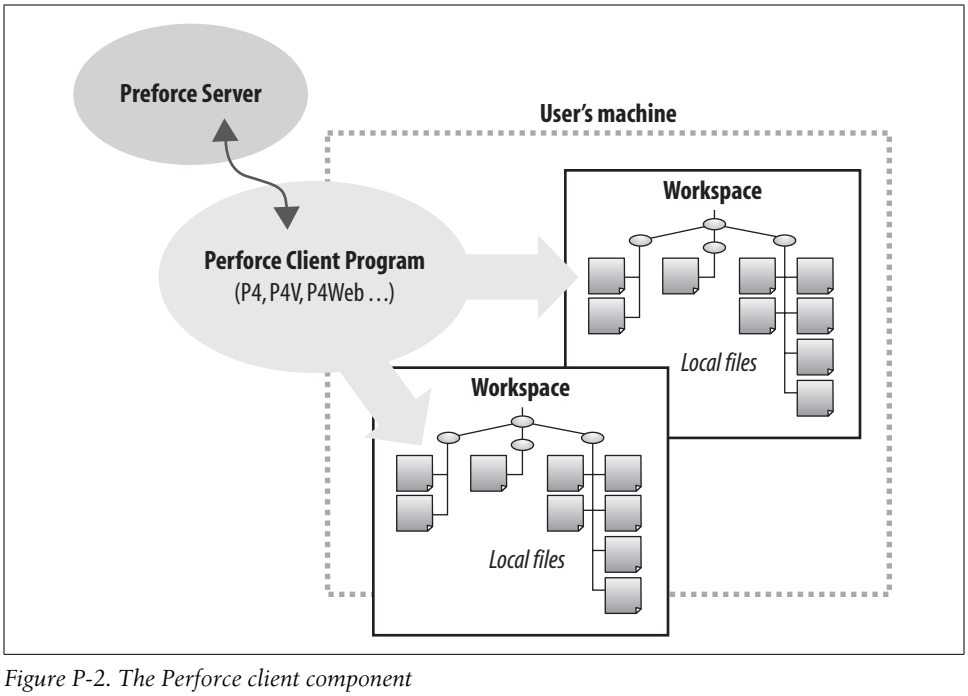



*Figure P-2. The Perforce client component*

*Plug-ins*

The Perforce plug-ins category consists of client programs that run behind the scenes, usually on the user's machine, to enable other applications to work with Perforce. The most widely used is the Perforce SCC Plug-in, which integrates Perforce with Visual Studio .NET. (That's SCC as in Microsoft Source Code Control API; any Windows application that supports the Microsoft SCC API is likely to work just fine with the Perforce SCC Plug-in.)

As Perforce's popularity grows, plug-ins are emerging that wed less technical applications to Perforce. P4FTP, for example, makes Perforce transparent to people using applications that rely on FTP, and P4Report turns the Perforce database into an ODBC data source for Windows spreadsheet and database tools.

*Programmable clients*

The programmable interfaces to Perforce are P4 and P4API. P4, the Perforce Command-Line Client, can be used in interactive shells and in scripts. It's the canonical client program—if you can't do it with P4, it can't be done. P4API, the Perforce C/C++ API, is available to embed the client component in applications, scripting languages, and other software. P4 and P4API run on all the operating system platforms Perforce supports—and there are a lot of them—and they support all Perforce operations, including administrative and privileged operations.

# Why Perforce?

The features and capabilities of an SCM system are important, but equally so is its ability to meet expectations and thrive in its habitat. Perforce runs as a self-reliant, self-contained system, and you don't need other software or hardware components installed to use it. Unlike many other SCM systems, Perforce fits into almost any computing environment, thanks to the following features:

*Speed*

Perforce is fast. It doesn't make developers wait to check out, check in, compare, or update files, and it doesn't add a processing burden to developers' machines.

*Centralized repository*

In the Perforce system, there is one centralized repository per domain for files and SCM data. Very large companies may have several Perforce domains, but that's typically an organizational choice, not a limitation of domain size. (Perforce domains at some large companies are known to encompass over 1,000 users each.) While it may be argued that a centralized repository puts your SCM system at risk of a single point of failure, that risk is vastly outweighed by several advantages. First, you have only one machine per domain to take care of to protect your assets. Second, you don't have to worry about where your assets

are. Office moves and machine upgrades don't perturb your central SCM repository. And third, as long as your central SCM server is running, all your users have access to it. The failure of one machine doesn't impede SCM access for users elsewhere in your system.

*No external database required*

Some SCM systems require you to configure and administer an external database system like SQL Server or MySQL. Perforce provides its own database. When you install Perforce, you're installing a reliable, self-contained database, customized for Perforce SCM.

Because the Perforce database can't be accessed by any other means than the Perforce Server, there's not much that can go wrong. It does require that the system administrator schedule regular checkpoints and backups, but other than that, very little hands-on administration is required. Database recovery performed after a disk failure or other misfortune can be done manually by the system administrator or through automated tools. Perforce provides tools for checkpointing, for recovery, and for automatically upgrading the database when a new release is installed.

*No reliance on networked file sharing*

Some SCM tools rely on networked file sharing (NFS) of one kind or another. NFS is not an ideal solution for SCM; network file sharing can be slow, and make it difficult for the SCM tool to handle file format differences. (Have you ever opened up a file in Notebook only to see all its lines running together? Or opened up a file in `vi` and seen `^M` characters at the end of every line?) NFS is also very machine-dependent; clock synchronization and other interoperability issues make version control difficult.

Perforce does not use NFS. Instead, it does its own file transport using TCP/IP. This approach gives it control over the files it cares about and, because TCP/IP is so universally supported, makes it capable of running on more operating system platforms.

*No HTTP server required*

Some SCM systems require you to configure an HTTP server, like Apache or IIS, to perform the duties of an SCM server. Perforce provides its own server and runs independently of your web servers.

Traditionally, software development organizations were formed of developers working together at the same company, at the same location. Most SCM systems, including Perforce, are suited for that kind of organization. But Perforce has built-in features that make it suitable for nontraditional teams, including teams formed of developers who work outside of the office, developers who work in separate divisions, and even developers who work for completely different companies:

*Process impartiality*
: Perforce imposes almost no built-in workflow or process rules. It's designed with certain software development activities in mind (all of which will be discussed in later chapters), but it can accommodate almost any procedure or methodology. Any workflow or process you have established (or that you would like to establish) can be automated with Perforce.

*File types*
: While some SCM systems have restrictions on handling certain file types, Perforce can store and manage text files, binary files, Unicode files, native Apple files on the Macintosh, Mac resource forks, and Unix symlinks in its repository.

*Product distribution and vendor drops*
: The Perforce Server can access file repositories in other Perforce domains. This makes a seamless, Perforce-to-Perforce distribution of software products possible. In other words, you can distribute your product directly from your Perforce repository to other organizations, as long as they have Perforce, too. And you can receive vendor drops from other organizations directly from their Perforce repositories. In fact, you can even branch or merge files from their repositories directly into yours. All the while, a history of what you've released and received is being collected and recorded in your SCM database.

*Firewalls and tunnels*
: As mentioned, Perforce uses TCP/IP to communicate between its components. The firewall that prevents external access to machines inside your network also prevents access to your Perforce repository. However, that doesn't mean that all your developers have to be inside your firewall. Perforce can be used in a Virtual Private Network (VPN), when one has been created, and authorized users can use Secure Shell (SSH) to tunnel through a firewall with Perforce commands. The advantage here is that you can extend your SCM—and hence your software development projects—to participants all over the world without having to give them direct access to your machines or intranet.

For a commercial product, Perforce is unusually accessible. Many new users are lured to Perforce simply because it's so much easier to get started with it than it is with any other SCM system:

*Easy to install*
: Unlike open source software, which generally has to be configured and built, Perforce tools are executable out of the box. By comparison, CVS and Subversion may be free, but they aren't free of the problems of building open source software. If you've ever been down the rabbit hole of trying to find, configure, compile, and install all the components in the dependency chain of an open source tool, you'll appreciate the simplicity of getting Perforce up and running. It's literally a 10-minute job: you download a couple of binary files, run one to start up a server, and run the other as your client-side interface to it.

*Runs everywhere*

Perforce runs on a huge variety of operating system platforms. Your laptop, the fully loaded machine at your office, the discount PC in your child's room, the old VAX you found on the sidewalk on trash collection day, even the groundbreaking new operating system you're developing—chances are very good that there's a version of Perforce that runs on it.* Since its inception, Perforce Software has made a point of porting its tools to as many platforms as possible. That's been relatively easy to do, because the core components of Perforce are small, standalone programs. And to this day, every version of Perforce ever released can be downloaded for free—in prebuilt, executable form, no less—from the Perforce FTP site.

*Costs nothing to try*

You can download all Perforce software and documentation for free, without having to talk to a sales rep or even fill out a form on the Web. The software you download is fully functional; it's the vendor's intent that you try Perforce and really see whether it meets your needs before you commit to buying it. If you want to test-drive Perforce in an environment with more than two users, Perforce Software will give you a limited-term license for as many users as you need. So instead of spending time in meetings arguing with everyone else about whether Perforce will meet your needs, you and your colleagues can spend time actually trying it out.

*Easy budget planning*

It's easy to plan a budget for Perforce. How many developers will you have? That's what you'll be paying for. Perforce is priced per user, regardless of what they're doing and the environments in which they're working.

*All-inclusive pricing*

Once you've paid for Perforce, you can download and run as many server programs as you need, on as many operating systems as you have, as long as you don't exceed the number of users you've paid for. You can run your servers anywhere in the world, and any of your users can use any of your servers (if you allow them to). The price you pay includes all of the Perforce client programs, plug-ins, and tools.

*Free to students, hermits, and saints*

In fact, if you're going to use Perforce for educational purposes—you're teaching a programming class, or developing software for a school project, for example—the vendor will provide you with a free license to cover as many users as are involved. Just contact Perforce Software and let them know about your project.

* The Perforce web site used to boast that "if the client program doesn't run on your platform, we'll port it there." Paradoxically, while compatibility with exotic, leading-edge platforms gave Perforce a foot up in the SCM market, the market itself has become more homogeneous. Today, established Windows and Linux operating systems seem to be the preferred platforms for even the newest software technology projects.

You need a license, by the way, only if you have more than two users accessing your Perforce repository. That means that if you're working on a software development project all on your own, or with just one other person, you can use Perforce for free, forever. And if you are one of the saints developing open source software for no remuneration, you can get a free Perforce license to cover you and everyone else working on your project.

## About This Book

This book is written with a particular reader in mind. The reader is familiar with SCM in general, and is most likely a programmer, a project manager, or a build engineer involved with software development. This book is written especially for the reader who wears more than one of those hats on the job and is responsible for some or all of the interconnection between the roles they represent. If you're pursuing better ways to keep it all connected, and are interested in seeing how Perforce fits in, this book is for you.

One purpose of this book is to present Perforce's potential as a software configuration management tool. This is a strictly academic purpose—you need not be a Perforce user to gain insight from it. Anyone interested in comparative SCM will find worthwhile material in this book.

The second purpose of this book is to help Perforce users understand *why* Perforce works the way it does. Most users come to this level of understanding on their own eventually; it is the level of understanding that prompts them to post "Aha!" messages to online Perforce discussions. This level of understanding also makes the difference between simply using Perforce to do what any SCM system can do and exploiting Perforce to accomplish what other systems can't. This book will get you to that level sooner.

There are two parts to this book:

- Part I (Chapters 1–5) describes Perforce commands and concepts. It's not a tutorial, nor is it a reference—it's more of a whirlwind technical tour. It will provide you with a baseline knowledge about fundamental Perforce operations.
- Part II (Chapters 6–11) describes the big picture, using Perforce in a collaborative software development environment. It outlines recommended best practices and shows how to implement them with the Perforce operations you were introduced to in Part I.

The examples in this book are based on Perforce 2005.1, although some features new to Release 2005.2 are covered as well.

## What's Not in This Book

This book contains no tutorials, no hands-on exercises, and no getting-started guides. Although it does contain numerous examples of basic and advanced commands, this book is not meant to be a primary source of instruction for new Perforce users. The role of this book is to complement the existing product manuals with tips and ideas for using Perforce to its full advantage.

This book doesn't document actual case studies. It's almost impossible to describe actual case studies without detail-laden examples that put a reader right to sleep. So we've foresaken realism on the principle that simple, readable examples can be extrapolated to complicated, real-world solutions more easily than simple solutions can be inferred from painstakingly realistic examples.

This book won't address industry standards, benchmarks, or certification models, although it will surely be of use to practioners of such standards. Perforce's strength is in its versatility and accessibility. It makes a robust foundation for a compliance process, but it does not itself enforce compliance.

This book is rather light on system administration issues. Perforce is a tool you can use to great effect without knowing anything about installation, security, backups, upgrades, migration, or performance. When you do need to know about these things, you'll find the Perforce manuals and other materials that are readily available at the Perforce Software web site to be a rich resource.

This book doesn't start with a chapter explaining SCM. There was a time when SCM was arcane and indistinct, but those days are gone, and the world now abounds with books and web sites designed to bring novices up to speed.

## Additional Reading

Software configuration management, as a topic, is finally conquering measurable shelf space in the computer section of bookstores. A number of SCM issues and challenges have been fully explored by other writers, and this book won't retread that ground. If you're a complete SCM novice, you might want to take a look at some of the introductory or complementary titles available, including:

*Real World Software Configuration Management*, by Sean Kenefick (APress)
: If software configuration management is in your job description, this book is for you. It's a no-nonsense explanation of SCM best practices with down-to-earth advice about getting going and sticking with them.

*Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, by Steve Berczuk with Brad Appleton (Addison Wesley)
: This is a concept book that manages to be quite practical nevertheless. Its detailed analyses of SCM problems and solutions are for the most part

independent of any particular SCM system. It also offers a comprehensive comparison of the terminology used by contemporary SCM systems.

*Configuration Management: The Missing Link in Web Engineering*, by Susan Dart (Artech)

This wide-ranging survey of risk management and return on investment includes brief case studies of a variety of SCM systems in use.

*Open Source Development with CVS*, by Karl Fogel and Moshe Bar (Paraglyph)

This very readable book combines a detailed guide to using CVS with an interesting discussion of its history and its application in open source projects. It's a good source of insight into how today's SCM terminology and usage conventions have evolved from their earliest progenitors.

*Software Configuration Management Strategies and Rational ClearCase*, by Brian A. White (Addison Wesley)

With its in-depth coverage of the ClearCase view of problems and solutions, this book presents an interesting contrast to SCM with Perforce.

*The Pragmatic Programmer*, by Andrew Hunt and David Thomas (Addison Wesley)

Not about SCM per se, this book touches on many software development practices that harmonize with good SCM.

Finally, while this book will teach you about Perforce, it won't teach you about all the Perforce commands, command forms, and command options available to you. For that level of detail, go to the Perforce web site and check out the following product manuals:

*The Perforce Command Reference*

An A-to-Z reference to P4 commands. You may wish to bookmark this manual and refer to it to find out more about—or alternatives to—the command forms and options shown in *Practical Perforce*.

*The P4 User's Guide*

A detailed guide to using Perforce for working with files. This manual is geared toward end users and uses P4 commands in its examples. Consult this manual for in-depth information about the Perforce user environment and a variety of typical developer tasks.

*The Perforce System Administrator's Guide*

A detailed guide to setting up a Perforce Server and managing a Perforce system. Consult this manual for in-depth information on backups, security, triggers, scripting, job customization, review daemons, performance, and OS-specific issues.

The online versions of these and other Perforce product manuals are available free at *http://www.perforce.com/perforce/technical.html*. You can also buy bound, hard copy versions of the same manuals; check the web site for details.

## Conventions Used in This Book

This book uses the following typographic conventions:

- `Constant width` is used for names of commands, command fragments, and command options.
- *Italic* is used for filenames and for characters used in the context of file identifiers.
- Button labels in graphical application windows are shown in regular text, and are often intercapped.
- Menu → Item → Item represents sequential selections in graphical application menus.
- Examples that show commands as they are typed, but that do not show command output, look like this:

```
type this command
```

- Examples that show commands as they are typed, and that show command output as well, look like this:

```
type this command
and you will see
output that looks like this
```

- Examples that show the contents of files and scripts look like this:

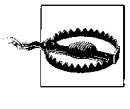
```
these are lines
that appear in
a file
```

- Examples of Perforce *spec forms* look like this:

```
Field1          value1
Field2          value2
```

In addition, the following formats are used to grab your attention and relieve the tedium of what could otherwise be monotonous reading:

Indicates a tip, suggestion, or general advice.



Indicates a warning or caution.

## Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Practical Perforce,* by Laura Wingerd. Copyright © 2006 O'Reilly Media, Inc., 0-596-10185-6."

If you feel that your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

## Safari® Enabled


When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

## How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed or that we have made mistakes. If so, please notify us by writing to:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (FAX)

You can also send messages electronically. To be put on the mailing list or request a catalog, send email to:

*info@oreilly.com*

To ask technical questions or comment on the book, send email to:

*bookquestions@oreilly.com*

We have a web site for this book, where you can find examples and errata (previously reported errors and corrections are available for public view there). You can access this page at:

*http://www.oreilly.com/catalog/practicalperforce*

## Acknowledgments

*Practical Perforce* couldn't have been written without the participation and encouragement of many people. I thank Christopher Seiwald, creator of Perforce the product and Perforce the company, for seeing the value in this project from its outset. I thank Kathy Baldanza for coaxing early drafts out of me and asking painful questions like "How's that book coming?" I thank everyone who reviewed the drafts (especially Jason Kao!) for catching so many of my dumb mistakes. I thank Jonathan Gennick at O'Reilly for making the endgame painless. And I thank Chris Comparini for making many hours spent sitting with a laptop a pleasant and companionable experience.

Above all, I thank the many people—users, customers, consultants, colleagues, and friends—who have indulged my compulsion to talk—at trade shows, at conferences, at the office, and at parties—about how SCM works in general and how Perforce pays off in particular. Their stories, their diagrams, and their sharp insights have shaped my ideas. They are the "we" in this book, the voice that narrates the knowledge I've tried to impart.

CHAPTER 1

# Files in the Depot

This chapter describes how Perforce stores files and directories in its repository, the depot. It starts by introducing the syntax that allows you to work with depot files and follows with examples of how to browse the depot and get information. Finally, it touches on file properties and their effect on how Perforce handles file content internally.

You may be happiest using a GUI (graphical user interface) for your day-to-day work. This book, however, bases most of its examples on P4, the Perforce Command-Line Client. One reason we stick with P4 is simply that it's easier to create and write about text examples than it is to create and write about screenshots. So don't take our bias toward P4 as a snub of the Perforce GUI programs. In fact, we'll point out some P4V features that show you at a glance what P4 would take thousands of lines of output to tell you. On the other hand, the GUIs are somewhat limited—only P4 offers the complete lexicon of Perforce commands. So, while you are encouraged to use a GUI, expect to use the command line from time to time to do the things the GUIs don't do.

## The Perforce Filespec Syntax

Perforce is widely used partly because it is so portable, and part of that portability comes from the platform-independent file syntax it provides. While native platform syntax can be used to refer to workspace files, Perforce provides its own uniform syntax for referring to workspace and depot contents. This syntax is known as a *file specifier*, or "filespec." A filespec can refer to a single file or a collection of files, to a specific revision or a range of revisions, and to depot files or workspace files. More importantly, the filespec syntax applies to all operating systems; Perforce converts filespecs to native file references for local operations.

## The depot hierarchy

Depots, where Perforce keeps master file content, and workspaces, where users work on files, are hierarchical structures of directories and files. A filespec uses *"//"* to indicate the root of the hierarchy, and *"/"* as a directory path and filename separator. For example:

```
//depot/projectA/doc/index.html
```

Although we often refer to an entire repository as "the depot," there can be multiple depots in a Perforce repository. The filespec root identifies the name of the depot. The filespec *//depot/projectA/doc/index.html* refers to a depot named "depot" (see Figure 1-1).

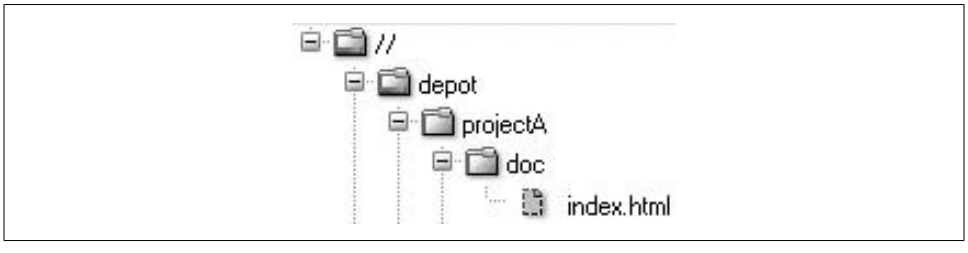



*Figure 1-1. Filespecs and the depot hierarchy*

A filespec can express a relative path as well as an absolute path. An unrooted filespec is a relative reference to the current directory (if you're using a command shell) or the current folder (if you're using a GUI). Depending on the context, *doc/index.html* or even just *index.html* could indicate the same file. In the Chapter 2 section "Local Syntax, Wildcard Expansion, and Special Characters," you'll find out how to use relative references to files and directories.

## Wildcards and file collections

When filespecs contain wildcards, they define entire collections of files instead of single files. For example, the "*" wildcard matches characters in filenames at a directory level. Depending on what files are actually present, a filespec like *projectA/d*/*.*html*, for example, can define a collection of files like:

```
projectA/dev/index.html
projectA/doc/diagnostics.html
projectA/doc/index.html
```

The "..." wildcard (pronounced "dot-dot-dot") matches filename characters at or below a directory level. A filespec that ends in */...*, in other words, is a succinct reference to the complete collection of files in a directory hierarchy. For example, *projectA/...* refers to the files in the *projectA* directory. Depending on what's in the directory, the filespec *projectA/...* might represent the following files:

```
projectA/bin/win32/app.exe
projectA/bin/win32/app.dll
projectA/dev/index.html
projectA/dev/main.cpp
projectA/doc/app/index.html
projectA/doc/app/reference.html
projectA/doc/diagnostics.html
projectA/doc/index.html
```

## Views and mappings

A filespec is a special case of the Perforce construct called a *view*. The Perforce database stores views for a variety of uses, including access permissions, labels, branching, triggers, and change reviews. The scope of every Perforce operation is constrained by the views that affect it.

Some of the views involved—filespec views, or workspace views, for example—are evident to users. Some views, however, like those that define access permissions, are not. For example, consider the P4 command that shows the history of changes to HTML files in the *//depot* path:

```
p4 changes //depot/.../*.html
Change 1386 on 2005/06/10 ... 'New page for promo...'
Change 1375 on 2005/06/05 ... 'Fix links on sign-up...'
Change 1369 on 2005/05/29 ... 'Add press releases...'
```

This command is affected by two views. The first is the filespec you see on the command line. The second is a view you don't see: the set of depot files you have permission to access. If, for instance, the access permission view is

```
//depot/projectA/...
//depot/projectB/...
```

the net effect is that you will see the history of the files in the *intersection* of the two views. In other words, you will see the history of the set of files defined by this view:

```
//depot/projectA/.../*.html
//depot/projectB/.../*.html
```

Views are also used to map files to each other. Client workspace views, for example, map depot files to workspace files, as you'll see in Chapter 2. In Chapter 4 you'll see how view mapping comes into play to relate branches to one another.

## File and directory revisions

Perforce stores file versions in a sequence of numbered revisions. Figure 1-2 illustrates the revisions of *//depot/projectA/doc/index.html*. A filespec can refer to an absolute, numbered file revision, prefixed with "#". For example, *index.html#10* is the tenth revision of *index.html*.

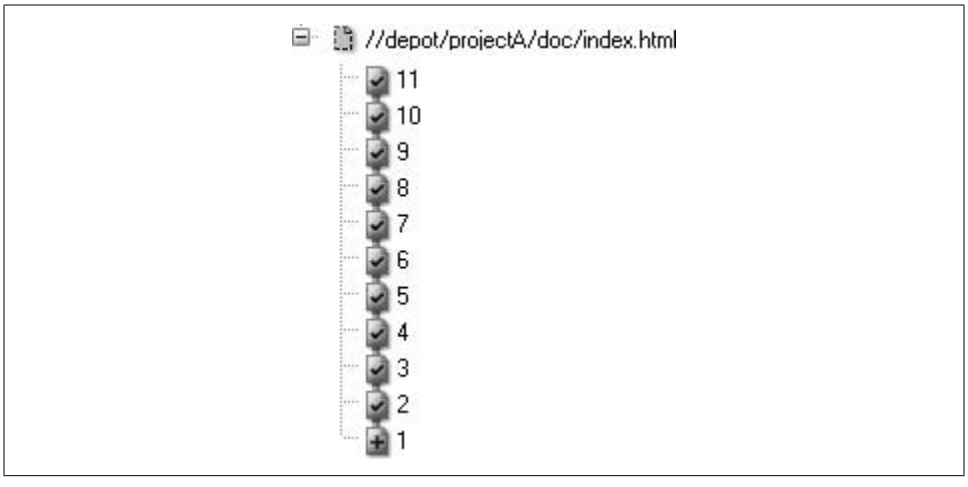



*Figure 1-2. Revisions of a single file*

Filespecs can also refer to dates and labels, prefixed with @. For example, *index.html@2004/11/21* is the revision of *index.html* as of November 21, 2004.

You can refer to directories by date as well. The filespec *//depot/projectA/...@2004/11/21* refers to the collection of files that made up the *//depot/projectA* directory as of November 21, 2004.

Two kinds of revision specifiers can be used in Perforce. One kind is the absolute revision. For instance, in this filespec

```
doc/index.html#14
```

the *#14* is a an absolute revision. It refers to the fourteenth revision of the file named *doc/index.html*.

Absolute revisions can't be used with directories. (A filespec like *doc/...#14* refers to the fourteenth revision of each and every file in the *doc* directory, not to the fourteenth revision of the directory.) However, you can use any of the symbolic revisions with both files and directories. For example, *#head* is a symbolic revision that refers to the newest, most up-to-date revision of a file or directory. For example:

```
doc/...#head
```

Perforce's reserved-word symbolic revisions are delimited by the character "#". Other symbolic revisions are delimited by "@." Dates, as you saw previously, are an example of the latter:

```
doc/...@2004/01/04
```

Labels can also be used as symbolic revisions. (You'll see how to create labels in Chapter 5.) A label can be used to refer to file revisions to which it has been applied:

```
doc/...@Good2Go
```

There are also symbolic revisions you can use to refer to files in a workspace, as you'll see in Chapter 2.

### Dates, Times, and Perforce

In a filespec, the date 2004/11/21 is actually shorthand for 2004/11/21:00:00:00. Saying *index.html@2004/11/21* refers to the revision of *index.html as of* November 21 is slightly misleading. It refers to the latest revision of the file as of *the commencement* of November 21, 2004.

Dates and times in Perforce are always relative to the Perforce Server. The revision *2004/11/21:12:00:00*, for example, specifies 12 noon on 21 November 2004 *in the server's time zone*. (See Appendix A.)

## Changes and changelists

Perforce uses *changelists* to track changes submitted to the depot. Changelists are numbered; when a changelist number is used as a symbolic revision, it refers to revisions that were newest at the moment the change occured. For example,

```
doc/...@3405
```

refers to the head revisions of the *doc* directory files at the moment changelist 3405 was submitted.

You'll notice in the preceding examples that the rightmost element of the filespec—exclusive of the revision specifier—is a filename, or a wildcard that matches a set of filenames. Perforce's filespecs always refer to files, not directories. In fact, there are no Perforce commands that operate on directories. This is not to say you can't organize your files into directories, or restore older versions of directories, or get the history of a directory. After all, when a Perforce command operates on the collection of files in a directory, it is in fact updating a directory. But in Perforce you don't explicitly create or version directories; it just happens automatically.*

In Perforce, a directory's revision (and its very existence, in fact) is construed from the file revisions it contains. You saw how file revisions can be identified by dates and changelists as well as by absolute revision numbers. Actually, you can refer to *any* file in the depot with *any* changelist number. Changelists represent points in time at which users submitted files. If you plot file changes over time, left to right, you'll see that changelist numbers slice file collections vertically—every changelist number is associated with a unique state of the collection.

* Yes, this *is* a bit of a challenge to the Perforce plug-ins. They bend over backward to support applications that think repository directories have to be created before new files can be added.

Consider the collection of files shown in Figure 1-3, for example. Here we see that in changelist *@100*, *foo.c* was added, creating *foo.c#1*. In changelist *@114*, *foo.c* was updated, creating *@foo.c#2*, and *bar.c* was deleted, creating *bar.c#2* (a deleted revision). *ola.c*, which was created in changelist *@105*, was unaffected by changelist *@114*. Therefore, revision *@114* refers to this collection of files:

```
foo.c#2
bar.c#2
ola.c#1
```

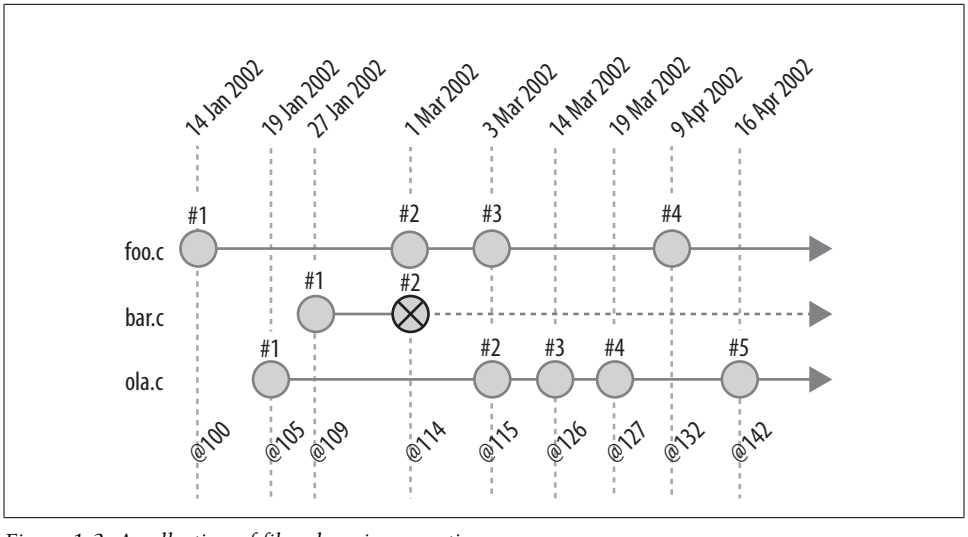



*Figure 1-3. A collection of files changing over time*

Note that labeling the time axis in a diagram like this with both dates and changelist numbers is redundant. Because changelists can't overlap—each marks a unique point in time—the sequence of Perforce changelists *is* a representation of time. It often makes just as much sense (and less clutter) to chart file evolution along the changelist axis, as we see in Figure 1-4.

The sequence of changelists associated with file revisions in a collection is, in fact, a history of the collection. And when a collection is a directory, the sequence of changelists associated with it is the history of the directory. If the *projectA* directory contains only the files shown in Figure 1-3, for example, collapsing the diagram into a single timeline would show the history of *projectA*. We see this in Figure 1-5.

In the next section you'll see how to list and compare directory revisions. Later chapters will show how directory revisions can be used for populating workspaces and in branching and merging operations.

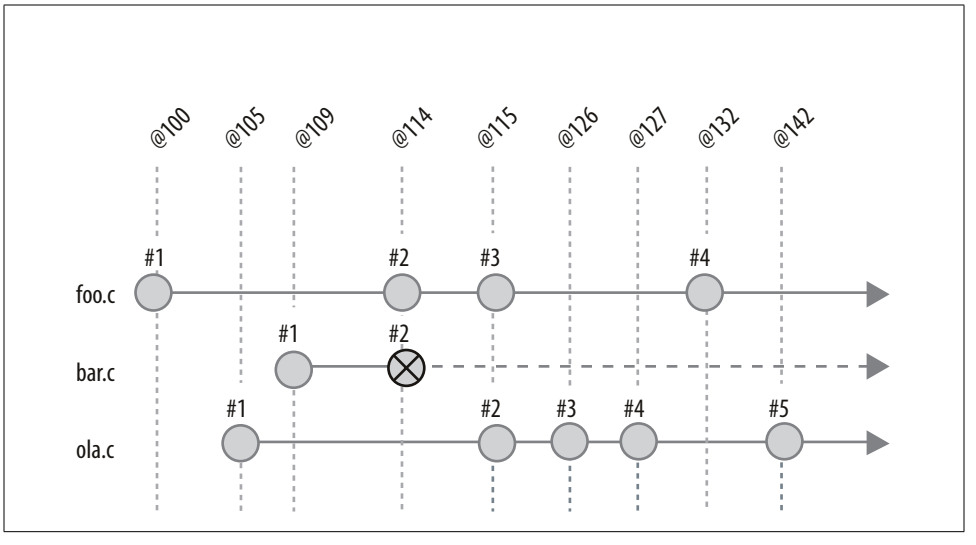



*Figure 1-4. The changelist axis*

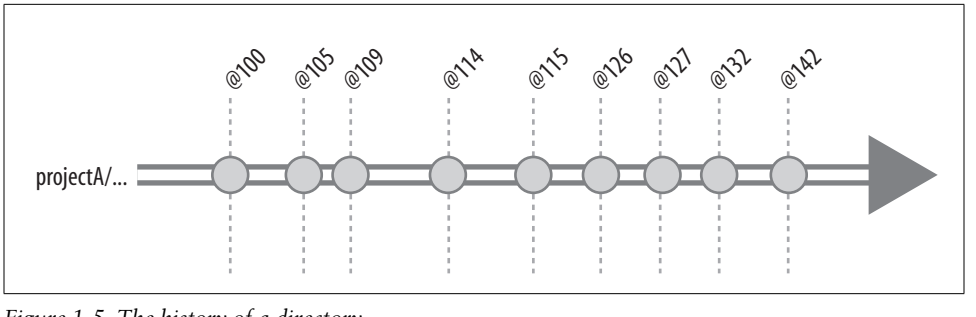



*Figure 1-5. The history of a directory*

# Browsing Depot Files

You can do extensive browsing in a Perforce depot without having to set up a workspace of your own. In fact, there is very little reason to reproduce depot files locally just to see their contents. You can explore the depot hierarchy, peruse file history, read change descriptions, examine file content, and compare depot files, without going to the trouble of setting up a workspace.

Many of the examples that follow are from the Perforce Public Depot. You, too, can browse the Public Depot by connecting to *public.perforce.com:1666* (see Appendix A). However, some of the outputs shown here have been somewhat abridged to shorten line lengths and reduce clutter. If you connect to the Public Depot and try these commands for yourself, you'll get more verbose results.

## Navigating the file tree

The depot is a file tree, and the easiest way to navigate it is with a GUI. With P4V, for example, all you have to do is point and click to step down the tree and expand its subdirectories (or folders, as they're called in P4V). A P4V depot tree is shown in Figure 1-6.

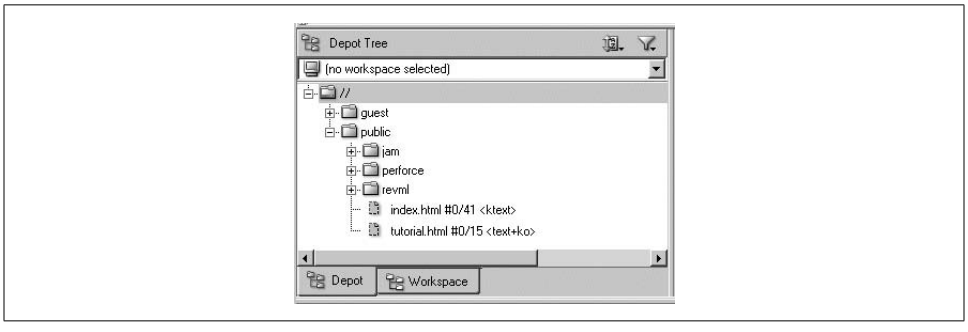



*Figure 1-6. Navigating the depot tree in P4V*

However, you can also navigate from the command line, using P4. To list the top-most levels of the tree, for example, use this `dirs` command:

```
p4 dirs "//*"
//guest
//public
```

Notice that the `dirs` argument is quoted—that's so the command shell won't expand the asterisk before passing it to the `p4` command.

Another way to show the top level of the depot hierarchy is with the `depots` command:

```
p4 depots
Depot guest  'Depot for guest users. '
Depot public 'Perforce's open source depot. '
```

## Listing directories

The `dirs` command can be used at any level of the depot tree to list the subdirectories at that level. For example:

```
p4 dirs "//public/*"
//public/jam
//public/perforce
//public/revml
```

## Listing directory history

The `changes` command shows the history of a directory, listing the most recent changes first:

```
p4 changes -m5 //public/revml/...
Change 4971 on 2005/05/21 ... '- Added test to make sure big_r'
Change 4970 on 2005/05/21 ... '- Allow sdbm files to handle la'
Change 4969 on 2005/05/21 ... '- Added a special command line '
Change 4968 on 2005/05/21 ... '- Use module name instead of lo'
Change 4967 on 2005/05/21 ... '- Removed "-d", leaving only "-'
```

(The `-m5` flag restricts the output to the five most recent changes. Each change is identified with a changelist number and the first 30-odd characters of a description. If you want to see entire descriptions, use `changes -l`.)

In P4V you can use Folder History to see the history of a directory, as Figure 1-7 shows.

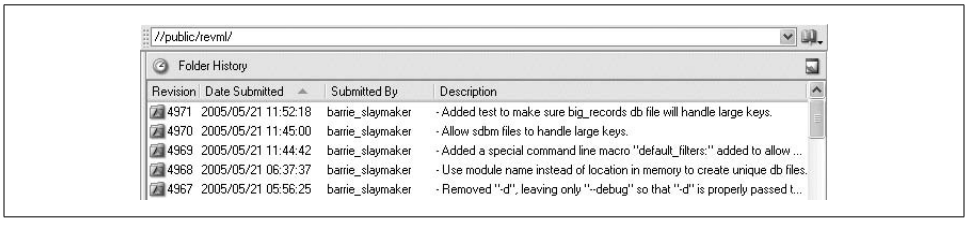



*Figure 1-7. Using P4V to browse the history of a directory*

## What's in a changelist?

In addition to marking points in time, changelists also record the files that were changed and the user who changed them. You can show the details of a changelist with the `describe` command:

```
p4 describe -s 4417
Change 4417 by barrie on 2004/08/19 20:11:50
  - Adapt to "estimated values" messages
  - Adapt to more accurate test suite
Affected files ...
... //public/revml/bin/gentrevml#56 edit
... //public/revml/lib/VCP/TestUtils.pm#65 edit
... //public/revml/t/91cvs2revml.t#16 edit
... //public/revml/t/91vss2revml.t#7 edit
... //public/revml/t/95cvs2p4.t#30 edit
```

(The `-s` flag suppresses diff output. If you use `describe` without it, you'll get a diff of every file in the changelist!)

## Listing files and file information

You can list the files in a directory with the `files` command:

```
p4 files "//public/revml/*"
//public/revml/CHANGES#81 - edit change 3640 (text)
//public/revml/MANIFEST#45 - edit change 4234 (text)
//public/revml/ui.png#1 - add change 3671 (binary)
//public/revml/ui.ps#1 - add change 3671 (text)
```

Each line of output gives a bit of information about the file revision shown. For example, *//public/revml/CHANGES#81* is a text file, last edited in change 3640.

You can list files in subdirectories recursively, using "..." with the `files` command:

```
p4 files //public/revml/...
//public/revml/CHANGES#81 - edit change 3640 (text)
//public/revml/MANIFEST#45 - edit change 4234 (text)
//public/revml/bin/analyze_profile#2 - edit change 2679 (xtext)
//public/revml/bin/compile_dtd#1 - add change 2454 (xtext)
//public/revml/dist/vcp.exe#10 - edit change 4233 (xbinary)
//public/revml/dist/vcp.pl#4 - add change 4235 (xtext)
```

(Note that the `dirs` command, by contrast, has no recursive form.)

## Finding files

As you can see, the `files` command has the potential to yield thousands of lines of output. If you're looking for a particular file, you can use wildcards to pare down the results. For example, here we're looking for files named *index.html*:

```
p4 files "//public/revml/.../index.html"
//public/revml/docs/html/index.html#2 - edit change 2307 (text)
//public/revml/product/release/0.90/html/index.html#1 - add change 4344 (text)
//public/revml/product/release/1.0.0/html/index.html#1 - add change 4311 (text)
```

## Perusing file history and file origins

You can use either `changes` or `filelog` to see a file's history. The output of `changes` is the same for a file as for a directory:

```
p4 changes //public/revml/dist/vcp.pl
Change 4235 on 2004/03/18 by barrie '- experimental dist/vcp.pl'
Change 4023 on 2003/12/11 by barrie '- Remove outdated "fat"'
Change 1859 on 2002/05/24 by barrie 'fat script version '
Change 1738 on 2002/04/30 by barrie 'Add "fat" script '
```

The `filelog` output, by comparison, shows file revision numbers and the action (add, delete, and so on) that took place at each revision:

```
p4 filelog //public/revml/dist/vcp.pl
//public/revml/dist/vcp.pl
... #4 change 4235 add    'experimental dist/vcp.pl'
... #3 change 4023 delete 'Remove outdated "fat" '
```

```
... #2 change 1859 edit   'fat script version '
... #1 change 1738 add    'Add "fat" script '
```

(You'll also see date, user, and file type information in `filelog` output. They've been removed here to make lines fit on the page.)

Normally `changes` and `filelog` limit their scope to the file you specify. However, files that have been renamed, cloned, or branched from other files inherit the history of their ancestors. You can use the `-i` flag with `changes` and `filelog` to show inherited* history:

```
p4 filelog -i //public/revml/lib/VCP/Dest/texttable.pm
//public/revml/lib/VCP/Dest/texttable.pm
... #5 change 4506 edit   '- testtable handled undef field'
... #4 change 4496 edit   '- minor POD cleanups to prevent'
... #3 change 4488 edit   '- BFD and Text::Table no longer'
... #2 change 4037 edit   '- VCP::Dest::texttable function'
... ... branch into //guest/timothee_besset/lib/VCP/Dest/texttable.pm#1
... #1 change 4036 branch '- VCP::Dest::texttable created.'
... ... branch from //public/revml/lib/VCP/Dest/csv.pm#1,#4
//public/revml/lib/VCP/Dest/csv.pm
... #4 change 4021 edit   '- Remove all phashes and all ba'
... ... branch into //guest/timothee_besset/lib/VCP/Dest/csv.pm#1
... ... branch into //public/revml/lib/VCP/Dest/texttable.pm#1
... #3 change 4012 edit   '- Remove dependance on pseudoha'
... #2 change 3946 edit   '- VCP::Source::vss now parses h'
... #1 change 3828 add    '- VCP::Dest::csv dumps rev meta'
```

P4V's `Revision Graph` gives you a bird's-eye view of a file's inherited history, as you can see in Figure 1-8.

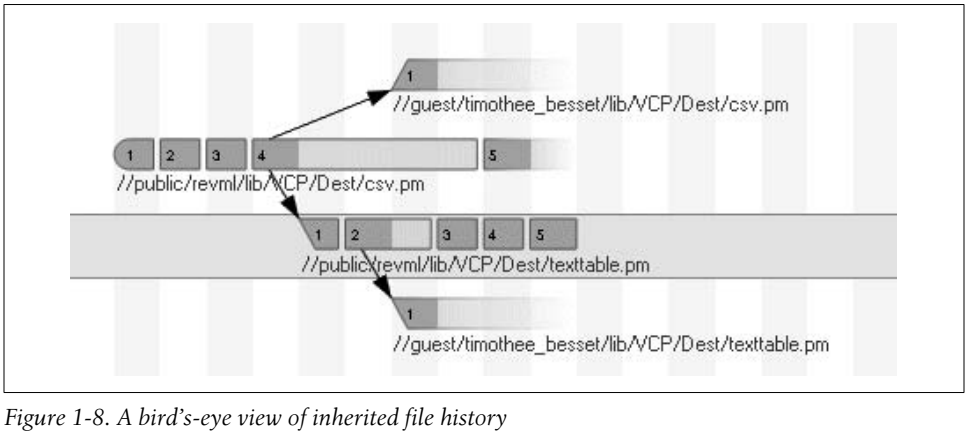



*Figure 1-8. A bird's-eye view of inherited file history*

* It's only coincidence that `-i` is the flag that makes `changes` and `filelog` show inherited history. The "i" really stands for "integration"; you'll see why later in the book.

## Perusing file content

P4V offers a nice content browsing tool for files. If you select a text file in P4V and click `Time-lapse View` you'll see the file's current content, along with a sliding control that changes the display to its content at any previous point in time. Other controls can be used to highlight the age of lines in the file, users who changed the lines, and the diffs for each revision. The black-and-white screenshot you see in Figure 1-9 doesn't begin to do justice to the usefulness of this tool.

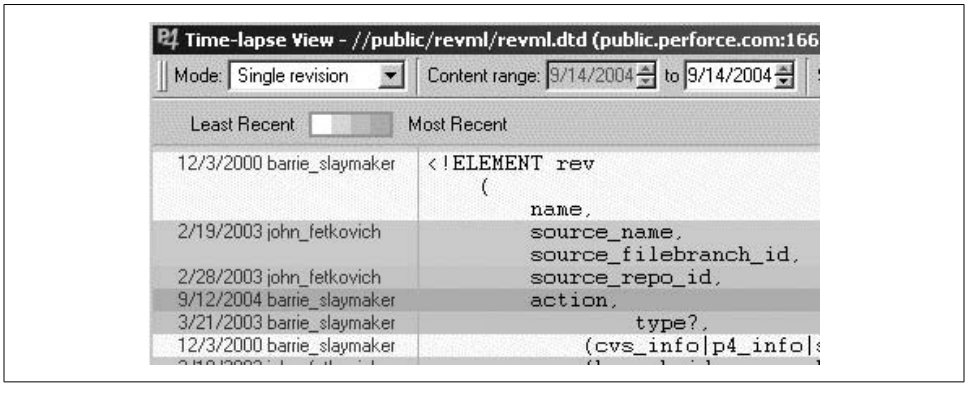



*Figure 1-9. P4V's Time-lapse View*

P4V's Time-lapse View is generated from the output of the `annotate` command, among others. You can get annotated file content in text form as well. For example, to see each line of a file annotated with a changelist number, you would use:

```
p4 annotate -c //public/revml/revml.dtd | more
//public/revml/revml.dtd#19 - edit change 4514 (text)
...
467: <!ELEMENT rev
467:     (
467:         name,
2743:         source_name,
2743:         source_filebranch_id,
2802:         source_repo_id,
...
```

To see plain, unadulterated file content, use the `print` command:

```
p4 print //public/revml/revml.dtd | more
//public/revml/revml.dtd#19 - edit change 4514 (text)
...
<!ELEMENT rev
    (
        name,
         source_name,
         source_filebranch_id,
         source_repo_id,
...
```

## Saving informal copies of files

The `print` command is also useful for saving informal copies of files. Simply redirect its output to a local file:

```
p4 print -q //public/revml/revml.dtd > revml.dtd
```

(The `-q` option suppresses the one-line header that `print` normally outputs.)

## Comparing depot files

To compare any two depot files, use the `diff2` command. For example:

```
p4 diff2 //public/jam/README //guest/dick_dunbar/jam/README
==== //public/jam/README#2 (text) -
     //guest/dick_dunbar/jam/README#1 (text) ==== identical
```

(This output has been edited to fit on the page.)

The same command can be used to compare any two revisions of a depot file:

```
p4 diff2 //public/revml/README#2 //public/revml/README#3
==== //public/revml/README#2 (text) -
     //public/revml/README#3 (text) ==== content
...
45,47c45,46
<    make
<    make test
<    make install
---
>    $ perl -MCPAN -eshell
>    cpan> install VCP
...
```

In P4V the Tools → Diff files command can be used to diff any two files or revisions. Figure 1-10 shows an example of a graphical diff in P4V.

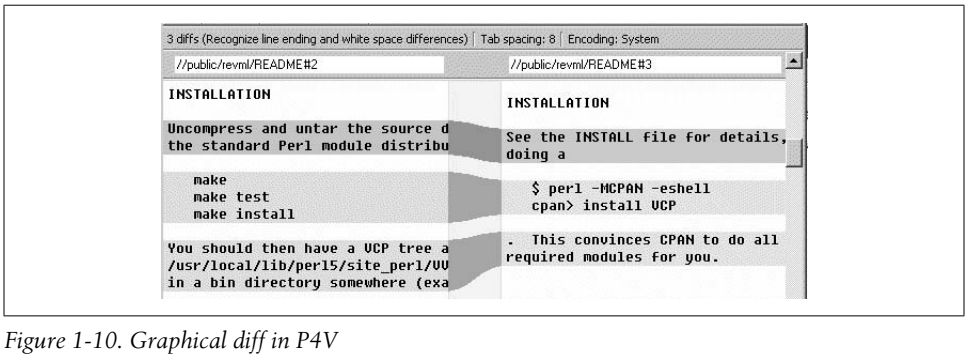



*Figure 1-10. Graphical diff in P4V*

## Comparing depot directories

You can also compare any two directories in the depot. For example, to compare *//public/revml* to *//guest/timothee_besset*:

```
p4 diff2 -q //public/revml/... //guest/timothee_besset/...
==== ... bin/gentrevml#56 - ... bin/gentrevml#1 ==== content
==== ... lib/VCP.pm#19 - ... lib/VCP.pm#1 ==== content
==== <none> - lib/VCP/Dest/ab.pm#1 ====
```

This shows us that there's a revision of *bin/gentrevml* in both directories, but their contents do not match. Same with *lib/VCP.pm*. And the *lib/VCP/Dest/ab.pm* file appears in the *//guest/timothee_besset* directory but not the *//public/revml* directory. (The `-q` flag is used on the `diff2` command to suppress line-by-line text diffs. Note that the output shown here has been drastically edited to fit the page.)

The same command can be used to compare any two revisions of a directory. For example:

```
p4 diff2 -q //public/revml/...@3660 //public/revml/...@4498
==== .../dist/packages.mball#1 - <none> ====
==== <none> - .../dist/vcp-rh8#4 ====
==== <none> - .../dist/vcp.exe#10 ====
==== .../dist/vcp.pl#2 - .../dist/vcp.pl#4 ==== content
```

This shows us that between revisions *@3660* and *@4498* of the *//public/revml* directory, the *dist/packages.mball* file has been deleted, *dist/vcp-rh8* and *dist/vcp.exe* have been added, and *dist/vcp.pl* has been modified.

P4V gives you the same directory comparisons in a much nicer display, as you can see in Figure 1-11. You can use Tools → Diff files to launch it, or just select `Folder History` on a folder and drag one folder revision to another.

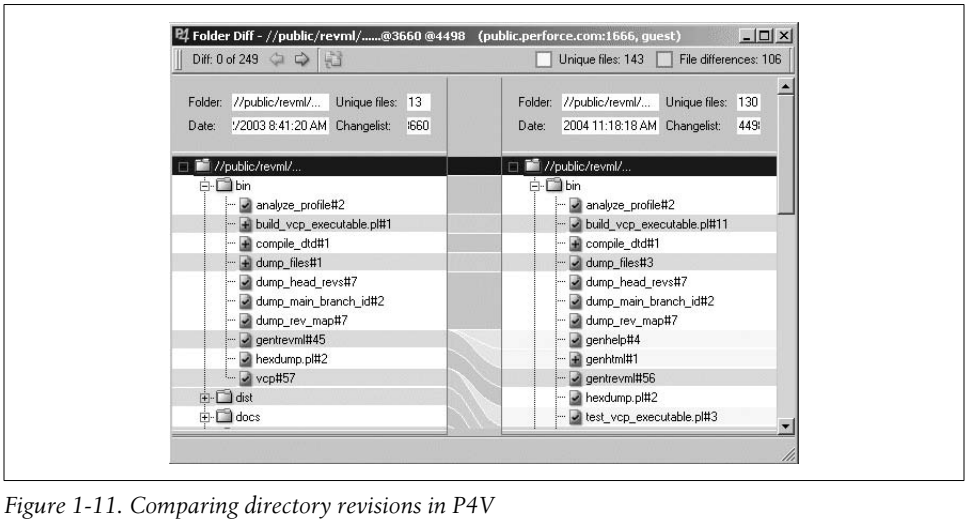



*Figure 1-11. Comparing directory revisions in P4V*