Chapter No. 7
"Dealing with Conflicts"
In this package, you will find:

A Biography of the authors of the book
A preview chapter from the book, Chapter NO.7 "Dealing with Conflicts"
A synopsis of the book’s content
Information on where to buy this book

About the Authors

**Robert Cowham** is from a software development background with experience in roles from programming to testing and project management. He came across Perforce as a user in the early days of the company when there were only four employees. He subsequently became one of the two pre-qualified Perforce Consulting Partners, and became a Perforce Certified Trainer as soon as that program was implemented. Since then he has consulted for many companies and organizations, and trained thousands of users around the globe, from California to Japan, including giving training courses in German, French, and Italian.

He has also written a number of integrations with Perforce. This includes P4OFC, which integrates with Microsoft Office, and is still provided as an officially-supported public download. APIs developed by him include P4COM, a COM based API for use on Windows, and P4Python which he subsequently handed over to Perforce to support. He has also written various full history migration tools for customers to get them into Perforce.

For More Information:
He has long had an interest in all aspects of configuration management and was for several years a co-author with Brad Appleton and Steve Berczuk of the Agile SCM column in the online CM Journal. He was Chair of the specialist group for Change, Configuration and Release Management of the British Computer Society for seven years, organizing many events and conferences, and is still active on the committee. He regularly speaks at industry events.

As Services Director for Square Mile Systems, Robert now also works with configuration management of infrastructure and data centers, applying the same principles to the physical world.

Robert has practiced the Japanese martial art of Aikido for over 20 years, and runs a dojo near his home in London. He has even managed to combine his interests with a well-received presentation on "Black belt SCM techniques," including physical demonstrations of the principles involved!

I would like to thank my family for their love and support during the writing of this book. My wife Phoebe covered for the many hours spent closeted away, and my sons Benedict and Dominic also coped admirably with the competition for my attention.

I have worked together with Neal many times over the years. We have different but complementary strengths and writing styles, and it has been a pleasure to work with him on this book.

**Neal Ralph Firth** assumed both individual contributor and management roles during the design, test, and development of both hardware and software used in stand-alone, embedded system, and real-time environments in his early career. Since the late 90's he has focused on source control and the automation of test and build systems. He first encountered Perforce in its early days while investigating source control solutions for one of his first consulting customers. He subsequently became a Perforce Consulting Partner and a Perforce Certified Trainer as soon as those certifications were available to him. He has consulted for hundreds of companies and organizations, and trained thousands of Perforce users and administrators across the US and Canada.

For More Information:  
His career has had a focus on automation and the tools that make people more productive. His early work with hardware microcode simulation was chronicled as part of Tracy Kidder's Pulitzer prize winning book "The Soul of a New Machine". He has presented papers and has spoken at conferences in the US, the United Kingdom, and Germany. He has published articles on hardware, software, and business topics. Hardware topics include peer reviewed IEEE articles on microcode. Software articles have dealt with the application of automated processes with a focus on legacy tool integrations and tool migrations.

As the principal provider of Perforce related services for VIZIM Worldwide, Neal's focus is on the migration of information between SCM products and integrations with legacy systems. He developed the framework for VIZIM's full history migration tool sets and authored the ClearCase and VSS-specific versions of those tools. He has created many Perforce-specific integrations for features such as the Perforce Defect Tracking gateway and IDEs such as JDeveloper.

I would like to thank my family for their support during the writing of this book. My wife Lynn for being a loving supportive partner, my son Eric for reminding me there is life beyond the keyboard, and my daughter Carolyn for reminding me to watch Dr. Who.

Robert and I have worked together many times over the years. Our different yet complementary skills have made for a pleasurable and dynamic interaction while working on this book.

For More Information:
Learning Perforce SCM

This book is about using Perforce. There are many tools within the Perforce ecosystem ranging from clients, to plug-ins for IDEs (Integrated Development Environments), such as Eclipse or Microsoft's Visual Studio.

The focus of this book is the P4V client (Perforce Visual Client). P4V has many graphical and presentation features that make it an ideal client for new users, and can be used on a variety of different operating systems including Windows, Unix/Linux, and Macs.

What This Book Covers

Chapter 1, Getting Started with Perforce, is an introduction to the technology, terminology, and concepts at the core of Perforce operation. A basic knowledge of these core factors will make it easier to relate your actions to the results you're trying to achieve.

Chapter 2, The P4V GUI, introduces the P4V interface. The chapter focuses on the mechanisms for viewing status, performing actions, and viewing the results of those actions.

Chapter 3, Basic Functions, covers the techniques for adding, modifying, and deleting files within a workspace. It discusses how to do this as an individual and as a member of a team.

Chapter 4, Changelists, describes changelists which are the core record of change within Perforce. The chapter covers how to use changelists for organizing work, communicating with other users, and avoiding problems.

Chapter 5, File Information, is the key to understanding the history of a repository, how a code base has evolved, and what is happening to it now. This chapter covers how to make the most efficient use of the Perforce reporting commands to examine the information associated with a file.

Chapter 6, Managing Workspaces, describes the Perforce model where client programs work with copies of repository files in local storage areas called workspaces. This chapter covers how to establish and maintain the relationship between server copies of files and workspace copies of those same files.

For More Information:
Chapter 7, *Dealing with Conflicts*, covers how conflict in Perforce refers to the need for a human to resolve issues that may arise from independent modifications of the same repository file. Conflict scenarios are a natural consequence of the flow of development and this chapter addresses the origins of conflicts, and ways of predicting future conflicts, identifying current conflicts, resolving conflicts, and avoiding them.

Chapter 8, *Classic Branching and Merging*, introduces branching as the key version control technology for managing parallel development. This chapter covers the classic Perforce branching interface that provides you with complete control over the entire range of branching features supporting almost any branching pattern that you can envision.

Chapter 9, *Perforce Streams*, describes Perforce streams which use the mainline model to support parallel development. This chapter looks at the concept of branch stability and how streams use the merge-down, copy-up paradigm to support stability.

Chapter 10, *The P4V User Experience*, covers various ways that you can adapt P4V to make your work easier and more productive.

Appendix A, *A Demo Server*, provides a straightforward, step-by-step setup for running examples that align with the contents of this book. Only a limited level of experience is required to follow these steps.

Appendix B, *Command Line*, documents the underlying Perforce interfaces and the common set of commands used. This appendix relates P4V features to the underlying commands that implement them as an aid to understanding the automation associated with builds and other activities.

For More Information:  
To Perforce, conflict does not indicate a problem. Rather, it refers to the need for a human to resolve issues that may arise from independent modifications of the same repository file. If there is more than one user, branch, or workspace there are conflict scenarios.

As we will see, conflict scenarios are a natural consequence of the flow of development. In this chapter, we'll review the origins of Perforce conflicts, predicting future conflicts, identifying current conflicts, resolving conflicts, and ways to avoid them.

In this chapter we will cover:

- File conflicts
- Content conflicts
- Interactive resolution
- Automatic resolution
- Avoiding conflicts

The origin of a conflict

Conflicts arise from concurrent changes to the same file.

Let's consider what happens when two people concurrently modify the same file without tool support. The second person to save their changes risks overwriting the other person's changes. Most of us have experienced this at one time or another! This is the essence of a conflict.

P4V detects and manages conflicts so users can coordinate their changes effectively and avoid losing their work.

For More Information:

Dealing with Conflicts

Other systems sometimes refer to conflicts and the actions to resolve them as merge. Since Perforce allows you to manage more than a simple merge of content they use the more accurate conflict and resolve terminology.

Working with conflicts

When using Perforce, users Bruno and Gale both make changes to the same repository file `//depot/dir/README.txt`, but independently, each using their own workspace copy as shown here:

Both Bruno and Gale checked out and modified version #1 of the file. When Gale submitted her changes she created revision #2, which in turn created a conflict for Bruno. Because of the conflict, any attempt by Bruno just to submit his changes will not be allowed by the server. Instead, Bruno needs to resolve the conflict by combining his changes with Gale's changes using a standard version control technique known as a 3-way merge. Only once the conflict is resolved does the server allow Bruno to submit his file which creates version #3.

How the server detects conflicts, and the details of a 3-way merge, are explained in detail in the following section.

For More Information:
Conflicts and submit

In the preceding example, both users started with version #1 of the file in their workspace which is the #head revision. Both users checked the file out and modified it. The checkout and modification did not create the conflict. The conflict was created when one user submitted their changes; in this example Gale. When the server creates version #2 of the file it knows that Bruno is working on an older version of the file (which is no longer the #head), and thus there is a conflict.

So, how does Perforce communicate the conflict situation, and what will happen if Bruno tries to submit his changes without resolving the conflict? When P4V performs a refresh of the display it shows that there is a newer version of the file Bruno is working on. Since Bruno's version is out of date, a (yellow) triangle decorates the icon to indicate the version status. The tooltip provides Bruno with additional information about the conflict, as seen here:

The tooltip is explaining that he now has an "old" version of the file.

If Bruno attempts to submit his changes he will get this dialog:
Dealing with Conflicts

He sees the warning that the file is Out of date and the Submit option is disabled. This will happen even if he has hundreds of files in his pending change list and only one of them has a conflict.

He has several options at this point. The first is to revert his changes, get the latest version from the repository, check it out, and redo his changes. While possible, this is potentially time consuming and seldom the best option. The second is to cancel the attempted submit and perform resolve actions on the files in the pending change list. However, when a small number of files are involved, it is also possible and indeed easy, to do everything without quitting the submit dialog:

Directly from the submit dialog, he can right-click and perform the Get Latest Revision action which P4V is suggesting.

As noted in the original submit dialog warning message, this "Get Latest" will not update the local copy (since that would mean his changes getting lost). Instead, the file is marked as being in the conflict state. Conflicts need to be resolved, hence, the needs resolve tooltip is then displayed:

Note that the icon decoration has changed from a (yellow) triangle to a (red) question mark. This indicates that a potential conflict has now become an acknowledged conflict that needs resolution. The server will not allow a submit if the pending change list is based on out of date files or contains a file with an acknowledged conflict.

An acknowledged conflict does not imply anything about the actual content of the file. It just means there were independent updates which need a human to resolve, or at least be consulted!

For More Information:
By right-clicking on the file and selecting **Resolve...** you will get the following dialog:

This dialog appears complex on the surface, but the basics are fairly straightforward:

- Perforce has detected a conflict state and a decision is required.
- The status information shown is guiding the user.
- The user needs to perform one of the **Accept** actions. This will tell Perforce that a decision has been made, and to change the state of the file so that it is no longer marked as "in conflict".
- The recommended **Accept** action is highlighted. Which one it is, depends on the specific file changes that have been made.
- Closing the dialog box will abort the resolve and leave all of the files unchanged.

In our preceding example, if Bruno clicks on **Accept Merged** and submits the file, he will get a clean merge result, and most likely all will be well. That is because his changes are able to be merged cleanly with Gale's changes (we cover other possibilities later in the **File content during merge** section of this chapter).
Dealing with Conflicts

Base, Source, and Target: a 3-way merge

When merging the contents of a file, Perforce and many other source control tools use a technique known as 3-way merge. A 3-way merge provides users with difference information relative to a version of the file both users had in common. As we will see, this difference information makes it easier to identify and combine changes.

3-way merge uses three significant terms that relate to Perforce objects:

- **Target**: the workspace file that needs to be resolved.
- **Source**: the repository file version causing the conflict.
- **Base**: the repository file version that Target and Source have as a common ancestor.

3-way merge and 3-way merge tools pre-date P4V and include the command-line version of P4. These tools use the terminology **Theirs** instead of **Source**, and **Yours** instead of **Target**. For compatibility reasons you still see references to **Yours** and **Theirs** in error, log, and other messages within P4V.

An introduction to the P4Merge tool

It is also an option for Bruno to click on **Run Merge tool** as shown in the following example:

![P4Merge tool example](image)

For More Information:

This shows the base (common ancestor) in the middle pane, and source and target file version changes to the left and right respectively. The bottom pane shows the default merge results. The default results are also the automatic results.

The status bar at the top contains a summary of the differences. The tool bar icons allow you to step through the file, one difference at a time (shown below). These options also appear on the Search menu.

We will revisit this tool with an expanded example shortly. For now Bruno can just close the tool, and click on OK to the prompt **Do you want to save your changes to the merge file?** Alternatively he could use File | Save and File | Exit or click the appropriate toolbar icons. P4V saves the new merged file on top of his file in the workspace, and he can submit it without any further problems.

![Saving the merge results overwrites the workspace copy of the file. There is no automatic backup of the original contents.](image)

### Differences from base

Differences from base is what allows 3-way merges to determine a default set of merge actions. **Differences** are a sequence of one or more consecutive lines within a file. The following diagram shows a basic example. For simplicity in this example, all differences are a single line rather than the typical multi-line sequences normally encountered.

![Diagram showing differences from base](image)

For More Information:

Dealing with Conflicts

As we can see in the Differences summary column and the combined screenshot excerpt from P4Merge, there are two places where the source file differs from the base, and just one place where the target file differs from the base. Where a difference is common to both (Line 5A as shown in the preceding screenshot), it means that the same character-for-character change was made in both the source and target. Note that a deletion (Line 3 as shown in the preceding screenshot) is just treated as a difference, it is not a special type of difference.

In the preceding example, P4V will merge the changes to give us the result shown. Because none of the changes overlap with each other the merge is easy to understand and in most (but not all) cases is going to be correct!

Dealing with content conflicts

So far we've dealt with merges where every difference is either common to both files or unique to one file. But what happens when different changes are made to the same section in both files? This is known as content conflict and requires human resolution.

An example of content conflict is shown as follows:

As we can see, there are conflicting changes to line 5 and the summary shows Conflicts: 1 (in red). Perforce will merge the non-conflicting changes (lines 1 and 3) but the recommendation to Run Merge Tool means the user needs to decide what the appropriate action to take should be.

When there are conflicts present, the (pink) toolbar icon to go forward and backward to the next conflict is enabled, as shown in the following screenshot:
In this instance the merge pane shows the merged lines and has the conflict lines highlighted (in pink):

By clicking on the appropriate icon: (yellow) square for base, (blue) triangle for source, or (green) circle for target, the user can select the content for the merge result:

In this example he has chosen to include only the target version. Notice that for lines 1 and 3, the respective icons are highlighted to show which version was selected.

**Editing in the merge pane**

Sometimes the best thing to do is to click in the merge pane and directly edit the file to get the results that you want:

In this instance the icons for Line 5 are all greyed because none of them is directly being used. Instead, the resulting text is marked (also in grey).

For More Information:

Dealing with Conflicts

If you're doing follow-along, set up this conflict example and explore it. You can create the conflict using two users and two workspaces. Or, you can have one user use two workspaces to create the correct sequence of checkout and submit.

File content during merge
The interpretation of file content is a human activity. Perforce just looks for blocks of text, without considering any possible meaning or semantics, and merges them.

How often does the automatic merge do the right thing? It will all depend on the types of files being merged and their content. In our experience, the majority of merges give the correct semantic result. Of course, especially with source code, what looks like a correct result may not actually be correct.

Any time you do a merge, it is always a good idea to test the result. If a merge breaks the build because the resulting source file won't compile, then that is easy to spot. It is more dangerous when a merge looks OK, compiles successfully, but has introduced a subtle bug. Good automated unit and functional tests can really pay dividends in such situations.

Some files are inherently difficult to merge in such a textual fashion (for example, XML files). With experience, you will understand what types of code or changes will merge easily and safely. When in doubt, step through the merge manually, difference by difference.

Overwriting or discarding changes on purpose
We discussed the resolve actions earlier in the chapter, but didn't cover what Accept Source or Accept Target did:

In the earlier scenario, Gale's changes are in the repository as revision #2. These changes are also the source of a 3-way merge. Bruno's changes are still in his workspace as the target of the merge.

For More Information:
If Bruno selects **Accept Target**, what happens? P4V will leave his file unchanged, consider the conflict resolved and allow him to submit his file. The contents from Bruno's workspace will overwrite Gale's version in the repository. It's up to Bruno to determine if this is appropriate, and Gale may have an alternate opinion!

On the other hand, if he selects **Accept Source**, what happens? P4V will overwrite the file in Bruno's workspace, consider the conflict resolved and allow him to submit his file. It's up to Bruno to determine if that is appropriate (as we discussed in *Chapter 6, Managing Workspaces*, workspace defaults may result in no file being submitted if his file is not different to the latest version in the repository). Of course in this case he could also simply revert the file and there would be no conflict and nothing to submit.

In some circumstances, a merge can appear too complex to merge using P4Merge. In such situations it may help to save a copy of your changes locally, do an **Accept Source**, and then reapply your changes on top. With experience you will know which option works well in which situation.

### Other P4Merge options

At the bottom of the resolve dialog is an **Additional Actions** button which offers a variety of potentially useful options:

If you are trying to work out what to do about a conflict, you can perform comparisons between various combinations of files. When it refers to **Merged Result**, this is a temporary file that P4V creates, which contains the results of an automatic 3-way merge.

This can be very helpful in understanding the history of the file and also what might be the most appropriate way to resolve the conflict.

For More Information:  
Dealing with Conflicts

You can go to Open File | Merged Result to see the contents of the temporary merge in your favorite editor:

![Image of file editor with merge markers]

However, note that P4V is showing us special conflict markers around the blocks of text. These markers are used by legacy merge tools so don't expect them to change from ORIGINAL (base), THEIRS (source), and YOURS (target). Although this option can be useful, most users stick with the graphical merge tool.

The way to resolve conflicts when shown like this is to manually edit the file, removing the markers and leaving behind the desired file contents. When you save and exit, the file will be assumed to be the results of the merge that you want to keep.

What if you miss a conflict?

Despite due diligence, you are bound to sometimes miss conflicts that exist in a pending change list prior to submission. After all, P4V can only create icon decorations and tooltips based on what it knows. If the version information known to the server changes after the most recent P4V request for status information there will be conflicts you don't know about.

Not to worry. The worst case scenario is that the server rejects your submit. You will need to resolve the conflicts before the submit will be allowed, but there are no changes to the repository to deal with.

If your submit fails, then the files in the pending change list may be locked. We recommend that to avoid issues with your colleagues, you either unlock the files or you resolve the conflicts as soon as you can. See the discussion on locking next.

For More Information:
Delaying resolution

There is nothing that requires that a conflict be resolved as soon as it is detected. You can delay resolution until just before you submit. However, keep in mind that conflict resolution can change file content. Also remember content changes impact your workspace environment, which in turn has a potential impact on testing that happens with content from before the conflict resolution.

Delays are fine, but conflicts must be resolved before you can submit a file. Best practice is to resolve conflicts as soon as possible.

Automatic resolution

To this point in the chapter, we have been discussing interactive resolves which work well when you are dealing with a small number of files.

There is an Auto resolve option which is selected at the top of the resolve dialog, shown below:

This option allows you to perform the same resolve action on all of the files in the pending change list. This can be a major time saver when lots of files are involved.

However, we tend to find that this option is most useful, indeed essential, when dealing with merging between branches, so we will cover the details in Chapter 8, Classic Branching and Merging.

Avoiding conflicts

While we encourage you to allow Perforce to manage conflicts, there are times when it is best to avoid them. In particular, this can be true when your changes have priority over other peoples’ or when a file cannot easily be merged. We’ll cover both of these scenarios in this section.

For More Information:

Dealing with Conflicts

Priority modifications – locking files

Sometimes you need to take priority when modifying a file. In Perforce, this equates to guaranteeing that you are the next person to submit a file. The mechanism is called locking a file. Note that needing priority is common after a failed submit, so P4V automatically locks all of the files in a failed submit.

Although it is not a best practice, you can always explicitly lock any file you have checked out. The most common way is to select Lock from the context menu in the tree panel or the pending change lists view panel.

Next, we see the result of Bruno locking a file for both himself and Gale:

Notice the padlock icon decoration indicating a locked file (the decoration is on the left or right of the file icon depending on who is viewing it). The tooltip provides valuable information about the lock.

If a file is locked, then only the person who locked it is able to submit the file. Other people can check out the file, but will not be able to submit their changes until the lock is released.

Locks are not intended for general use. If everyone immediately locks files when they are checked out overall team productivity will suffer. The best practice is to lock a file for the shortest time possible. If you can't resolve your issues quickly, it is best to unlock the file and let Perforce manage conflicts in the usual way.

Locked files are automatically unlocked if the file is reverted or the file is part of a successful submit. Files can also be explicitly unlocked using the Unlock menu option in the context menus of the tree panel or the pending change lists view panel.

For More Information:
Unlike some other SCM tools with lock capabilities, Perforce allows users to checkout a file locked by another user. This behavior allows the Perforce server to track intended changes. Better to track a pending change than potentially lose track of it.

Files that can't be merged
Some files don't have effective merge tools. Spreadsheets, documents, graphics, and other binary formats are common examples of these files. Because they can't be merged, you can only sensibly resolve conflicts by accepting either the source or the target. This makes it important to avoid conflicts with these files. Otherwise, you might have a potentially significant amount of work manually merging content changes.

Closely monitoring the checking out of such files might avoid conflicts, but it's not a viable strategy. You could attempt to lock such files one-by-one as discussed earlier, but that only tends to delay the conflict discovery. A better solution is the exclusive checkout Perforce file type.

You change the file type by right-clicking on the file and then selecting Change Filetype... option (also on the Actions menu this option is only available if the file is open for some action) shown as follows:

For More Information:  
Dealing with Conflicts

Check the +l **Exclusive checkout** attribute, leaving the base file type unchanged. The base type is likely to be binary since 3-way binary merge tools are relatively rare. However, any file type can be exclusively checked out. Submit the file once you have changed its file type. Another user (such as Gale) might see this result:

![Image of file checkout](image)

In the preceding example, Bruno has checked out a file which has the Perforce exclusive lock file type. The tooltip tells Gale that an **exclusive checkout** is in force, and she will get an error message if she attempts to check the file out.

**Summary**

Conflicts are a natural consequence of concurrent development. In this chapter, we've seen that dealing with conflicts only looks complex. In most cases, the identification and resolution of conflicts is actually rather straightforward. Most users only need to apply a small set of the available Perforce features for most of their conflict resolution needs. However, when more advanced techniques are required, those are available too. Perhaps most importantly, Perforce does not allow accidents. The server ensures that users acknowledge and resolve conflicts before it allows updates to the repository.

In the next chapter, we'll look at classic branch and merge operations, and we will see how merging between branches builds on the conflict resolution we went through in this chapter.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.