

# Analyzing File Content History

**James Creasy**  
Perforce Software

**Abstract:** *Have you ever needed to figure out what broke the build? Or needed to track down when, where, and why a particular line of code was added to a file? Because Perforce stores all changes made to a file by individual line, and can cross-reference this information with changelists, integration and other metadata, the answers to these questions are already stored in your Perforce system. This paper discusses file content history and Perforce's powerful tools for extracting and visualizing the content history of files.*

## File Content History

File content history is the record of every line that is or was in a file. This record can include other information related to each line and the changes that introduced or deleted the line. Perforce links changelist, user, time, date and other information with the history of changes. This data, which already exists in Perforce depots, represents a rich store of information that can be analyzed to solve common tasks.

## Using File Content History

File content history proves useful in a variety of common situations encountered by SCM (Source Content Management) systems. Analyzing the history has two main parts:

- 1 **Extracting data:** Retrieve the pertinent information from the depot. The record of all the changes for a file and the associated data is extracted from the repository. Perforce supplies the command line **p4 annotate** command for this purpose.
- 2 **Visualizing and interpreting data:** Use tools such as Perforce's Time-lapse View to

help filter, search and interpret the retrieved data. *Filtering* strategies help narrow down the data to search.

By analyzing the content history, you can answer common questions, such as:

- Who wrote a line of code?
- Why was this line of code added? (View the changelist description for an added line)
- When was this line added?
- Was this variable *ever* used in this file? If so, where and when?
- What changes were made to this file in the last week?
- How did this work before it was changed? Often a bug isn't found until weeks after the change that introduced it. The file might have had many revisions since then, and clues to what changed might be scattered among multiple revisions.

## Extracting Data

The **p4 annotate** command returns a block of text representing the history based on the changes to the specified file. To obtain the user and other data, you must then run a **p4 filelog** on the revision.

This method creates a static text file that can be loaded into a filtering and visualization tool. Performance is good because the extraction communication and processing is done first, so the visualization tool does not need to communicate with the server.

However, large blocks of text that combine this amount of information are clumsy to work with. You must use *filtering* to narrow the amount of data, and *visualization* to provide an interactive graphical view on the filtered data.

## Filtering

A file in Perforce often has a rich history of integrations and revisions that have built up over years through dozens of changes. Because Perforce stores integration records, the history of the file can be traced back through the integrations to the very first version that was added to Perforce. The amount of data in the complete history of the file is often very large, possibly hundreds of times as large as the file itself. Filtering excludes parts of the entire content history of a file to make searching and visualization easier.

Common filtering criteria include:

- Content for revisions within a specific branch

- Content from all files that contributed content through integrations to the file you are viewing
- Content from all files that contributed content through integrations and contain the line of code for the file you are viewing

### **Multi-branch revision range selection**

Often, most lines in a file are the result of integration, specifically, the integration that created the branch containing the revision you are interested in. If you explore the entire set of integrations associated with this initial revision, the resulting diagram looks like a tree, except for the case where a file is integrated back into the line it came from. An effective filtering tool will allow the user to select revision ranges that span multiple branches of this tree. This allows you to track content through the integration history and to the exact revision it was introduced in.

### **Searching**

Searching is a special form of filtering that highlights specific matches to an inquiry. Searching takes two main forms:

- **Initial state:** When a visualization tool is invoked, information from the context can be used to set the initial state of visualization. For example, if you start the tool from an editor, the selected text can be searched for and the visualization information (such as a changelist description) for that text can be shown when the application starts.
- **User-invoked searches:** Using a typical find dialog, you can search the history for specified text. The text searched can be the entire history of that file, or a filtered selection.

### **Visualization**

After the desired filtering has been applied, there still can be an overwhelming amount of data. Tools like Perforce's Time-lapse View provide an interactive window on this wealth of data. A graphical interface enables you to further filter the data, search for specific text within the data and provides specialized graphing and display tools for working with the data.

Visualization features can include:

- Controls to filter to a range of revisions, changelists, or dates

- View of text, filtered by the selected range
- Visual display of integration history for selecting revision ranges outside a single branch (tree-style graph)
- Graphical representation of the history of changes to a file (showing text deletion and addition)
- Use of colors and gradients to show age of additions, deletions, and replacements of lines

The goal is to provide an interactive interface that helps you find the specific changes or text needed to answer your questions.

## **Time-lapse View**

Time-lapse View is a powerful tool in Perforce for visualizing file content history. It includes features to filter, visualize, and analyze the file content through the history of a file in the Perforce depot. You can view single revisions of the file, or diffs between two adjacent revisions. A new and important feature enables you to view all revisions and the included changes concurrently in one view. This feature is called “Multiple Revisions” mode.

### **Multiple Revisions mode**

Multiple Revisions mode is similar to a single-pane diff tool, except that instead of being limited to just two revisions, hundreds of revisions can be viewed concurrently. The revision range slider allows you to instantly constrain the revisions shown to a subset of all the revisions for the file. For example, if the file has 76 revisions, you can set the revision range control to show just the range of content history stretching from the 4th to the 16th revision.

### **Lifetimes**

A powerful visualization tool called “Lifetimes” displays the span between addition and deletion of each line in the file. If, for example, a line of code was added in revision 2, and deleted in revision 16, a bar stretches from the 2 to the 16 position on the scale that displays the total number of revisions.

Code that was added or deleted in the filtered range is colored red or blue. Red indicates a line that was deleted in the range and blue indicates a line that was added. All other lines are colored gray, as they have not changed in the filtered range. The Lifetimes graphical display uses the same colors as the text. This approach makes differences in

the filtered range stand out while scanning the Lifetimes view.

Lifetimes also helps you visualize replacements of lines in a file. If a line has been removed and another line substituted in its place in the same revision, the Lifetimes view indicates this with a curved shape.

In summary, colors indicate changes and curves indicate replacements.

### **Other features**

Time-lapse View includes simple and interactive ways to view diffs between adjacent revisions and the text of single revisions. The date or changelist associated with a revision can be displayed in the place of the revision numbers, making it easier to filter by these criteria. The user responsible for adding or deleting a line of text is easily displayed, and line numbering can be displayed using the viewing mode for single revisions. An additional visualization feature called "Show Aging" colors lines indicating how long ago the line was added to the file.

### **Use Cases**

The following sections describe how you can use Time-lapse View to answer common questions.

#### **What broke the build?**

The build had a fatal error and you have the name of the file that failed to compile. You know that the build worked the night before, and the file content history between then and now contains valuable clues to the source of the problem.

Using Time-lapse View in Multiple Revisions mode you filter by showing dates for the revisions. Use the revisions range control to show the text for all the changes in the last day. Time-lapse View now shows all changes in this range in a color- red or blue. Gray chunks of text can not have contributed to the broken build.

Further filtering options include showing just the changed (colored) chunks, buttons to jump from change to change, and filtered searches that search only the changed (colored) chunks.

#### **Why was a particular line of code added and when?**

Using Time-lapse View, you open the file in Multiple Revisions mode and search for the

text of the line you are interested in. You want to do as little content filtering as possible because you are searching the entire history of the file. Clicking on the line displays the revision information for the changelist in which the code was introduced and the changelist that removed it (if it was deleted.)

Further searching capabilities might include searching the history of integrations of the file, and filtering to search only revisions that contain the selected line. Note that this would be especially useful if the origin of the line was in the head revision through integration from another branch- a common situation.

### **Content history is more than single files**

There is a relationship between changes in a file and the other files in the changelist for that change. Because a changelist describes one atomic change to the larger body of code in the depot, every change to every file in the changelist has an essential contribution to the work described by the changelist.

A common example occurs when you change the return type of a function in C++, which requires an update to the implementation and header file. Searching the content history of only one of the files in this changelist might miss an important clue.

However, there are difficulties in how to visualize this larger set of changes. A simple approach is to open one view for each file, but there is a practical limit if the changelist includes many files.

### **Conclusion**

The wealth of file history maintained by Perforce is a storehouse of information. Tools like Time-lapse View help filter, visualize, and analyze the data with a dynamic and visual interface, allowing you to more easily solve common problems, even problems you have not have tried to solve before.