Conference Paper: tcworld 2015

Branch and Merge:

A content manager's dream or a tech writer's nightmare?

An Unrealistic Dream

The dream for any writer is to produce the perfect content. A single author writes the perfect document from start to finish in a single session. It never needs updating because it's already perfect. What could such a document be about? If the document described a business process, that process would also have to be perfect. If it wasn't, our perfect document would need to change as the process itself changes. If the document was a maintenance manual for a machine, the machine itself would have to be perfect. Changes to the machine design to improve it would need updates to the manual.

This is, of course, an unrealistic scenario. We all know that these documents don't exist. The changing nature of the subjects we write about prevents us from attaining perfection. Perfection may be a writer's dream, but it is an unrealistic one.

The Reality

In reality, writing a document is an iterative process. Content goes through several draft versions before we're happy to submit it as releasable. Once the author is happy, the document passes to a reviewer, who will make further changes. Finally, the document is ready for approval.

The life of a document doesn't end there. If the business process we're documenting changes, our document needs updating. If the machine has an improved component installed, we need to ensure that we change the manual. The document needs reworking, reviewing and editing all over again. This is a common scenario that all tech writers are familiar with.

The right tools can help us to manage changes in this update cycle. Most editing software includes change tracking functionality. This allows you to record your own changes or view a reviewer's suggestions.

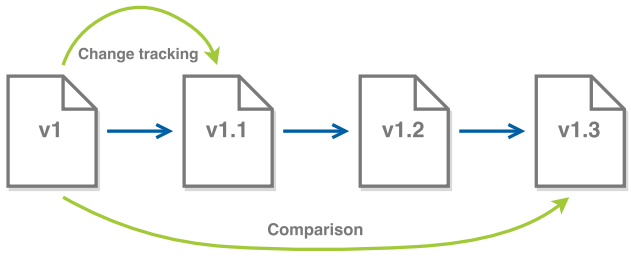

Figure 1: A simple document version workflow

This works well when all parties use the same editor but what about when that isn't the case? Even with change tracking, it's difficult to manage changes across several document versions. For this use case, a comparison tool is the best solution. They are able to highlight change across a range of document versions without having to track changes. These tools can even generate change-tracking entries, allowing you to view changes in a text editor.

The Nightmare

The workflow so far has been linear, but how can we handle more complicated workflows? For this we need to make use of branching. There are various reasons to use branching. We may be managing user documentation for many versions or releases of a product. This use is known as release management. Branching is also used to separate the documentation of a new product feature. If it's uncertain which product release will use the new feature, a feature branch allows progress without interrupting the main branch. Once the feature is complete, its documentation is included in the appropriate release.

Branching is often used to manage product variants, such as platform-specific documentation. The main branch holds common documentation, and further branches hold documentation for specific platforms.

These are just a few of the possible use cases for branching. You will need to decide which strategy best suits your specific use case and implement a branching policy for your writing team.

A common aspect of all the branching strategies is a need to be able to pass changes between the branches. How to achieve this will depend on how the branches are managed and maintained.

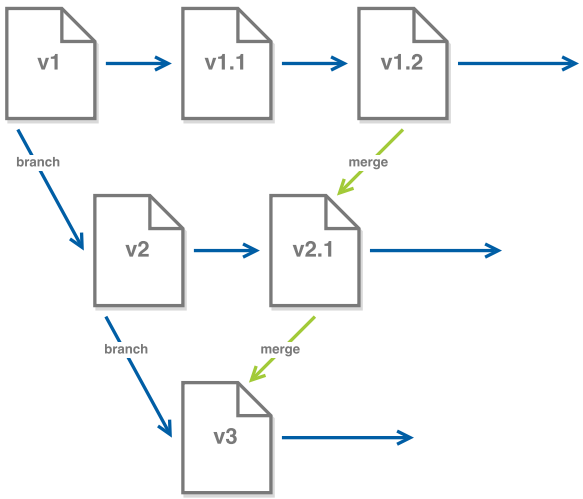

Figure 2: A complex workflow using a simple branching model

DeltaXML Ltd Malvern Hills Science Park, Malvern, Worcestershire, WR14 3SZ, UK
info@deltaxml.com +44 1684 532 130

Imagine a product documentation scenario that starts with a simple workflow as shown in Figure 1. The documentation for version 2 of the product starts while authors also work on v1.2 updates. Version 2 documentation is held on its own branch. After version 2 is complete, changes are made for v1.2. These are relevant to include in the version 2 branch to produce 2.1 but how do we include those changes? First, we need to know what the changes were. For this we can use comparison tools. But then we need to apply them to version 2 of the document. Without suitable tooling, this has to be a manual process. Authors may use cut and paste, or view the documents side by side and manually edit version 2 to include the changes. Even with a simple branching model, such as that in Figure 2, this becomes difficult to manage. When branching becomes more complex, we find ourselves in a nightmare scenario.

Dreaming a new dream

How do we wake up from this nightmare? We need to dream a new dream that includes appropriate tooling to help us manage the situation. We need a tool that removes the manual process of applying changes and the potential for human error. We need to use a merge tool.

A merge tool will identify the differences between branches and merge changes from one branch to another. It is applicable in all the branching scenarios discussed earlier. For a 'feature branch', it can merge the branch back into the main trunk once the feature is complete. In release management, it can merge edits made to previous versions into later version branches to keep them updated.

Banishing the nightmare

Source control systems such as Git and Mercurial feature merge tools for performing merge operations between branches. While these are used successfully in many situations, merging structured content such as XML can be problematic and can lead to fundamental errors in the merge result. The structure of XML allows for a more intelligent merge because our understanding of the structure informs the merge operation.

A structured XML-aware merge tool provides the following features, many of which gives significant benefits over a line-based merge tool.

Well Formed Result

This is a key requirement when content is stored as structured XML. Line-based merge tools may break the integrity of the XML content even to the extent that it can no longer be parsed. An XML merge tool must be able to ensure that all output can be parsed.

Valid Result

Extending the previous feature, validity against a specific XML grammar, e.g. DITA, is vital if we are to use the result in our document workflow. A merge tool must be capable of understanding the constraints of such a grammar and must present the output in a way that doesn't break the validity of the content.

Fewer Conflicts

When documents are merged, there will invariably be conflicts at some stage. These occur when more than one writer has made changes to the same area of the content. Merge tools that do not understand XML syntax may identify changes that are not relevant in an XML context e.g. changes to attribute order. This will lead to unnecessary and unhelpful conflicts. An XML-aware merge tool can minimise the number of conflicts that occur by ignoring such false changes.

Cherry Picking

This term is used to describe the selection of a subset of identified changes. It may not be appropriate to apply all changes from one branch into another branch. Enabling the content manager to select a subset of changes to apply is an important feature in any merge tool.

Auto-Resolution

In some branching models, it may be possible to programmatically resolve any conflicts that occur. If logical rules can be applied in order to resolve conflicts, the level of human interaction can be minimised. Merge tools that provide this feature can be extremely beneficial in streamlining a merge workflow

Customisable Granularity

Changes can always be presented at different levels of detail. Simple word changes inside a paragraph could be presented at the word level or they could be presented as two versions of a paragraph. They could even be presented as two versions of the whole section. The ability to customise the granularity of change can help to improve the review process.

Conclusion

The appropriateness of particular content management strategies is dependent both on the context in which content is created as well as on the subject of the content itself. These strategies range from simple, linear workflows to complex branched structures, often appropriate when documenting software releases that employ a similar branching model. Even simple strategies benefit from tooling such as change-tracking and comparison. As the model becomes more complex, more sophisticated tools are needed to support it. Standard line-based merge tools are not sufficient for handling structured XML content. Content curators should use XML-aware merge tools to ensure the correct handling of the structure and syntax of XML. With the correct feature-set, these tools can turn the nightmare of manually merging complex branching models into a content manager's dream come true.

Written By: Tristan Mitchell, Product Manager At DeltaXML, UK