Merge as you go

Chris Berarducci Handspring January 2003

Introduction

This paper explains how Handspring utilizes the standard Perforce branch mechanism and a branch specification to keep multiple project trees up to date with the appropriate changes and bug fixes. Handspring refers to this process as "Merge as you go", which consists of a change based merge and merge daemon script.

A change based merge is simply merging changes from a single specific change number in Perforce to another branch or source tree. This keeps a one-to-one relationship between the original submission and the submission on the destination source tree.

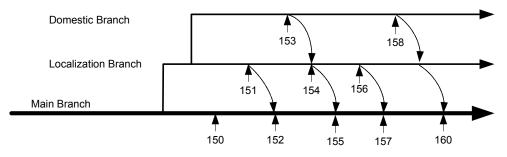
To make this process sustainable in a live development environment we utilize a daemon process to schedule the merges every N minutes.

A successful process requires more than just strong SCM tools — it also requires a balance between the tools and the entire development team. We hope to illustrate how clear ownership and role definitions, as well as cooperation from the whole team, reduce extra overhead and contribute to the success of "Merge as you go" in our development environment.

Overview

Handspring's development model is evolving depending on the project needs. When the "Merge as you go" process was being created the development team tended to work on similar projects at the same time and utilize software branches to isolate the changes. An example of this can be seen in Figure 1. Throughout the explanation we will be referring to this development model.

Figure 1



Project A was separated into two branches, a domestic branch and a localized branch. The domestic development branch is isolated from the localized changes but the localized branch takes every domestic change to the project. All changes migrate to the main branch.

Auto merging of source code can be very powerful provided the merges are done correctly. With this in mind the *merge* part of "Merge as you go" is handled with standard Perforce commands in the automation scripts. Any merge that the merge daemon completes could have been accomplished with native Perforce commands from a user's command line shell. We are simply using automation to separate the merges that Perforce flags as a conflict and completing the merges that Perforce does not flag; with this approach we feel that the quality of our merge process is very high.

A change based merge will merge the changes from a single Perforce submission into another branch maintaining the following:

- The new submission will contain only the files in the original Perforce change number. Possibly a subset, if not all the files in the original change are represented in the branch specification. This is standard Perforce merging.
- The new submission will carry the comments from the original Perforce change number into the new change. This is a script enhancement in our process.

• The new submission will add additional information about the branch specification used for the merge and the change number that the merge originated from. This is a script enhancement in our process.

All of this together helps to create a standard look and feel for a merge. Additionally, we have adopted a semi-standard syntax that engineers use when submitting changes to Perforce.

Anybody reviewing a changed based merge submission can easily see the comments from the original change. While this sounds trivial, when we couple this with our semistandard syntax it quickly shows a non-technical person what changes have been merged in. For example, bug fixes or features are called out in a certain way and are easy to see in the submission comments. Also, a merged change may actually be merged multiple times depending on the branch methodology in use. In these cases it is invaluable for us to carry the comments forward. Program Managers and Quality Engineers, who are not typically Perforce users, create release notes from the submission notes. Having the merged submission comments brought forward makes their job easier and the resulting release notes more accurate.

Changed based merging is a technique that can be used locally by an engineer to manually merge a change. In this case all of the benefits listed above are received but the merge is not done in a scheduled manner. While good for one or two changes it does not scale well to have engineers spending time doing single change merges.

The auto merge daemon allows for specific branches to be scheduled for change-based merges. It allows the development engineers to concentrate on coding and not merging; from a process standpoint they are not the gating factor for a merge taking place but are available to resolve conflicts when they occur. When conflicts do occur there is immediate feedback, based on the frequency of the merge daemon. Generally speaking, the conflict is easier, more lightweight to resolve being that one engineer has made the changes and is still familiar with what they were trying to accomplish in the code.

The "Merge as you go" process builds upon the standard Perforce merge commands which turns out to be a nice benefit because it does not limit any options. In other words because we use the built in merge features Perforce does not distinguish between a change based merge or any other merge, it is simply a merge. In that way Perforce is the authority on what has been merged and what integrations are still outstanding.

"Merge as you go" organization:

"Merge as you go" can be separated into two functional parts:

• "mergechanges.pl" – This script takes as input a branch specification name, the direction of the merge and a change number. It then attempts to merge that change into the destination branch based on the specified branch specification. This script enables automating the change based merge that we talked about in the summary.

When executed from the merge daemon status of the merge will be communicated in email. The "mail" configuration file controls who will receive the email and is checked into Perforce.

For each configured merge an engineer can specify 3 lines in the "mail" configuration file. They are:

Branch – The name of the branch that you are defining notification for.

"Merge OK" – When a merge is successfully completed for this branch notify these folks. It can be a comma-separated list or a group alias that is resolved at the mail server.

"Merge Conflict" – When a merge has conflicts notify these folks. It can be a comma-separated list or a group alias that is resolved at the mail server.

For example:

Branch: Localized_reverse_Automerge MERGE OK: Localized_OK_notification@handspring.com MERGE CONFLICT: joe_engineer@handsprint.com

• "mergereview.pl" – A daemon script to determine if there are any valid Perforce submissions that need to be merged. If there are it executes "mergechanges.pl" and performs any post merge steps that are needed based on feedback from "mergechanges.pl".

The "mergereview.pl" daemon only runs in one instance and is data driven. A "merge" configuration file is checked into Perforce controls it. Each time the daemon wakes up it syncs the "merge" configuration file from Perforce and reads the merge data. Based on this data a Perforce submission may or may not be considered a valid merge to attempt.

This approach of separating the configuration data from the script greatly enhances the engineer's ability to control the merges. The daemon script does not need to be started and stopped as configurations are added, modified or deleted.

In the "merge" configuration file a line that begins with '#' is a comment. Any other non-blank line is a merge definition line.

Each merge definition line has the following structure:

Branch Name, Direction of merge, Pattern Match, Change Number

For example, the lines

```
Project_A_Localized, reverse,,150
```

Would configure the following:

- A Perforce counter based on the branch specification name and the direction of the merge. Subsequent runs would use that counter to start looking for changes to merge. This means each configured branch has its own counter, potentially two, one for each direction.
- The daemon to start at change 150 the very first time it ran for this project.
- Execute mergechanges.pl when there are outstanding changes to merge.

Not shown in this example is the "Pattern Match" text field. It is an additional filter for determining whether to attempt the merge or not. To set up a filter you simply type the text that is required in the 3rd position of the configuration line. If a specific change number is eligible for the merge based on information from Perforce the daemon script then additionally looks for the text in the submission notes of the original check in. This is specialized usage and left for the reader to explore further.

It's easy to turn the auto merge daemon off by commenting out the configuration line and there is no harm in moving the change number back to previous changes. Since Perforce is managing all the merge information it will not merge something that is already merged, unless (obviously) you force it.

Merge Resolve Steps:

By default "mergechanges.pl" takes a pretty conservative approach to what is merged. Its rules work like this:

It first attempts:

P4 resolve -af

From the Perforce help files:

"The -af flag forces 'p4 resolve' in automatic mode to accept the merged file even if there are conflicts."

From the list of files resolved any files that are flagged for manual resolution or have conflicts are tracked.

For the files requiring manual resolution we attempt to resolve them with the following command:

```
p4 resolve -t -as
```

From the Perforce help files:

The -as flag performs a 'safe' automatic resolve, accepting only files that have either your changes or their changes, but not both. Files with changes to both yours and theirs are skipped.

Any files that had a conflict or could not be integrated with the given commands will abort the merge. At this point the files will be reverted and if ran from the merge daemon email notification will be sent.

Roles and responsibilities:

Since Perforce is primarily used as an engineering tool all of the responsibility falls into the engineering group, however Handspring does have some resources available to internal tool development. The internal tools group is responsible for supporting the merge and daemon scripts; any bugs found are reported and fixed.

The configuration of the auto merge daemon is the responsibility of the engineering lead for the project. The lead engineer configures the branch name, the direction of the merge, which change number to start with and any header information to key off of when doing the merge.

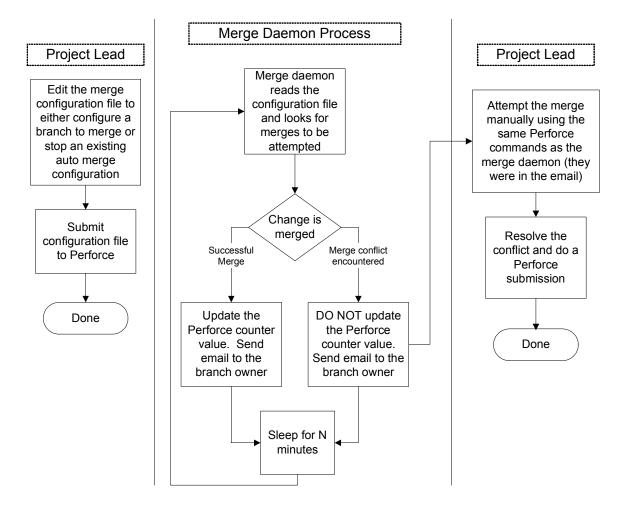
Additionally, the development engineers have agreed to manually resolve any merge conflicts that occur in a timely manner. When there is a conflict the project lead is notified via email from the merge review daemon and they work with the development engineer to correct the issue.

Product development cycle:

A common goal of branching is to isolate changes and promote stability in the source tree, especially when close to critical milestones in the project. However, if you are using the "Merge as you go" concept then you potentially are bringing in changes that may destabilize the quality of the branch. In a case like this it makes complete sense for the project lead to turn off auto merging during this critical time period. The merge daemon is using Perforce counters to track which change number was last merged. That information will be preserved and available when the branch specification is added back to the configuration file.

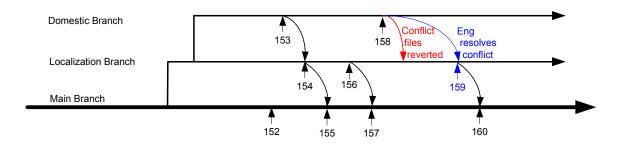
Process flow

Each column represents a task in parallel.



Examples

1. Example merge scenario:



Project A was separated into two branches, a domestic branch and a localized branch. The domestic development branch is isolated from the localized changes but the localized branch takes every domestic change to the project. All changes migrate to the main branch.

The following definitions have been entered in the "merge" configuration file.

Domestic_Branch, reverse,,150 Localized_Branch, reverse,,150

They would configure the following:

- A Perforce counter based on the branch specification name and the direction of the merge, in these cases the counter name would be:
 - O Domestic_reverse_Automerge
 - O Localized_reverse_Automerge
- The initial counter value for both counters would be '150'. Subsequent runs would use that counter to start looking for changes to merge.
- Execute mergechanges.pl when there are outstanding changes to merge.

Items in **bold** signify the auto merge process.

A more detailed example:

- 1. Change 152 Engineer submitted to the Main branch. Miscellaneous change not directly related to any active projects.
- 2. Change 153 User submitted to the Domestic branch. This is a feature being added and needs to be propagated to the localization branch.
- 3. Change 154 Auto merge submitted to the Localization branch, the daemon adds additional header information to the submission notes.
- 4. Change 155 Auto merge submitted to the Main branch, the daemon adds additional header information to the submission notes.

- 5. Change 156 User submitted to the Localization Branch. This is a text change required for localization of the product. The user changed foo.h line 73 in this submission.
- 6. Change 157 Auto merge submitted to the Main branch, the daemon adds additional header information to the submission notes.
- 7. Change 158 User submitted to the Domestic Branch. This is a text change required for a new feature in the product. The user change foo.h line 73 in this submission.
- 8. Merge attempted Conflicts occurred shown with the red line above. The files were reverted for the pending change and email was sent to the appropriate engineers. The merge of change 158 is still outstanding.
- 9. Change 159 The project lead manually merged change 158 to the localization branch. The resolved the conflicts in foo.h
- 10. Change 160 Auto merge submitted to the Main Branch, the daemon adds additional header information to the submission notes

2. Example Merge Submission Notes:

```
P4 describe -s 154
Change 150 by reviewdaemon@_reviewdaemon_mycomputer_software on
2002/09/17 15:39:36
"MergeChanges - Domestic_reverse_Automerge"
>> Change 153 by JoeEngineer@JoeEngineer_2kPC on 2002/09/17 15:37:37
>>
>> Fixed bug 25 Device crashes after repeatedly changing fonts
>>
>> Affected files ...
```

Affected files ...

>>

>>

... //branches/Localized/Libraries/TextEngine/TextEngine.c#26 integrate

>> ... //branches/Domestic/Libraries/TextEngine/TextEngine.c#21 edit

3. Example Merge Conflict Mail:

The title of the mail is:

MERGE CONFLICT, Domestic reverse 158

The content starts with the mergechanges command that was run and then proceeds to describe all the files that had issues.

```
mergechanges.pl -s -c=158 -b= Domestic -m -r
The following files had merge conflicts
//branches/BranchA Localized/foo.h
Diff chunks: 255 yours + 7 theirs + 0 both + 1 conflicting
The following commands can be run to do the merge on your system
p4 integrate -r -b Domestic @158,@158
p4 resolve -af '//branches/Localized/foo.h'
Here are the notes from the check in we are merging
>> Change 158 by JoeEngineer@JoeEngineer 2kPC on 2002/10/07 14:13
:37
>>
>> Fixed bug 26 : Fatal error when trying to open the Calculator
>>
>> Affected files ...
>>
>> ... //branches/Domestic/foo.c#150 edit
>>
```

Closing Summary

"Merge as you go" makes the most sense when you have projects with different trees that share some of the same features or bugs. For those projects there are benefits for the development engineers, QA and program management groups. The "Merge as you go" process is set up so there is no additional overhead for the tools teams , whether they are supporting 1 or 10 projects.

"Merge as you go" works well at Handspring when it makes sense, not all of our projects benefit from this approach. Those other projects have no restrictions to pursue their best merge options. The idea is to use the tools that make your team productive and "Merge as you go" is another tool that is available.

Our experience has led us to the following conclusions:

- Critical to the "Merge as you go" success is:
 - The Perforce merge facilities
 - Established roles and responsibilities
 - The daemon runs in one instances
 - Conflicts can be handled on the engineers CPU
- Development engineers like it:
 - They own the configuration and merging; they do not need to consult with another group or person to turn on or off the auto merge daemon.
 - They are relieved from thinking about merges unless there is a conflict or configuration change needed. When it is on and configured, changes checked into one tree will be migrated to the destination tree(s) in a timely and reliable manner.
 - It's easy to set up and the configuration files are tracked in Perforce.
 - It's easy to turn off and on
- More information is provided in the submission comments:
 - Easier for QA and Program Management to generate release notes.

The overhead and benefit is directly related to the development model, the number of developers, the number of projects and the complexity of the source tree.

Depending on project activity and how merge configurations are set up, Handspring experiences anywhere from 1 to 25 auto merges daily. Usually any resulting conflicts are resolved within 24 to 48 hours of the conflict being reported; it is not usually a high priority task but more a background task.

The scripts are written in Perl and execute the Perforce commands from the command shell. At the time of development we didn't know of a Perforce API or Perl module that could be used to access Perforce. The original developments of these scripts occurred about 2 years ago and have had only minor updates since then.

Our Perforce server runs Solaris on a Sun Ultra 60 CPU and most of our development occurs on the Windows platform. The "mergereview.pl" script runs on a Linux configured system and "mergechanges.pl" runs on both Windows and Linux.