1. Resolving Parallel Development, A Practical Example

Suppose that two developers have worked in parallel to change different revisions of a single source file; one worked on bug fix to rev 1.10 and the other worked on an enhancement to the tip revision, rev 1.11.

The final bug-fix was checked in as rev 1.10.1.1 and the final enhancement code was checked in as rev 1.12.

Now, a decision is made that the bug fix needs to be merged back into the "main trunk" of the revision tree so that all future revisions will have that particular bug fixed.

By looking at the revision tree for foo.cc, as shown by the command:

$ **vtree foo.cc**
vtree: /projRCS/foo.cc/foo.cc,v

```
...
1.8
1.9
1.10  Baseline:
branches:  1.10.1;
   1.10.1.1
1.11
1.12
```

we can determine that both revision 1.10.1.1 and 1.12 have actually been checked in and that rev 1.10 is the most recent common ancestor of both revisions 1.10.1.1 and 1.12.

To merge the changes in rev 1.10.11 with those in rev 1.12, use the **vmerge** command:

```
-------------------------------------------------------------------------
Usage: vmerge FileName BaseRev [TargetRev]

    When run in FileName's normal project directory, vmerge will merge
    your current copy of FileName with its tip [TargetRev] revision from
    the /BriefCase repository (based on their relative differences from
    the "BaseRev" revision) and write the merged result to stdout.
    Any changes in your revision of FileName which conflict (overlap)
    with changes in the tip [TargetRev] revision will be presented in
    the merged output as:

        <<<<< Your:FileName
           ... lines from your copy of FileName ...
        =============
           ... conflicting lines from the tip [TargetRev] revision ...
        >>>>> Rev-<tipRev|TargetRev>:FileName

    and cause vmerge to print "merge: overlaps during merge".

    Overlaps may be resolved by choosing one set of lines and removing
    the other set of lines and the overlap presentation lines or by
    implementing a compromise.

-------------------------------------------------------------------------
```

as follows:

1. Having determined that rev 1.12 is the the specific revision (i.e. "TargetRev") into which the bug-fix changes (in rev 1.10.1.1) are to be merged, use the **nlock** command to lock the target revision:

   $ **nlock -r 1.12 foo.cc**

   If rev 1.12 is not the tip (latest) revision, use the **bcol** command to start a branch off rev. 1.12 (i.e.: 1.12.1):

   $ **bcol -r 1.12 foo.cc**

   The merged result checked in with "nci foo.cc" will be checked in as 1.13, the tip of the main trunk, or 1.12.1.1, the tip revision of the new branch.

2. Having determined that the most recent "common ancestor" (i.e. "BaseRev") of both the target revision and the bug-fix revision is rev 1.10, Use the commands:

   $ **nco -r 1.10.1.1 foo.cc**
   $ **vmerge foo.cc 1.10 1.12 > new.foo.cc**

   to check out a (read-only) copy of the bug-fix revision (rev 1.10.1.1), merge it with the "enhancement" (rev 1.12), relative to their common ancestor (base) revision (rev 1.10) and save the resulting merged code as the file "new.foo.cc".

   If **vmerge** emits a warning message like:

   merge: warning: overlaps or other problems during merge

   You and/or the developers must resolve the overlaps (represented as indicated in the **vmerge** command description above) before using the merged code.

3. Once the overlaps have been resolved, replace your foo.cc (rev 1.10.1.1) file with the new.foo.cc file:

   $ **cp -f new.foo.cc foo.cc**

4. Use the **vmatch** command, again, to check that the merged code differs least from the target revision. For example:

```
$ vmatch foo.cc
13 foo.cc -r1.12
21 foo.cc -r1.11
53 foo.cc -r1.10
65 foo.cc -r1.9
144 foo.cc -r1.8
138 foo.cc -r1.7
185 foo.cc -r1.6
187 foo.cc -r1.5
245 foo.cc -r1.4
230 foo.cc -r1.3
246 foo.cc -r1.2
394 foo.cc -r1.1
39 foo.cc -r1.10.1.1
```

   shows that the local foo.cc file differs from the checked-in revision 1.12 by only 13 lines - fewer

than for the other revisions. Note that this may not always be the case, as when some portion of the merged code "reverts" to an older revision, but it may help identify gross errors like overwriting the "foo.cc" file with the wrong file.

5. As another, more detailed, check on the merge, use the command:

   $ **vdif foo.cc 1.12**

   to print the exact lines in the new "foo.cc" file which differ from the target revision and verify that they accurately represent the "bug-fix" changes from rev 1.10.1.1

6. Once the merged code has been tested or otherwise verified, it can be checked in with the command:

   $ **nci foo.cc**

   The nci command will complain if you own multiple locks on this file (from the current work replica and, optionally, from the current client host), in which case, use the command:

   $ **bci -r 1.12 foo.cc**

   or, if you started a branch at 1.12.1:

   $ **bci -r 1.12.1 foo.cc**

   to specify the specific lock against which to check in this revision.

Note that vmerge, like the rcsmerge engine it uses, is a FILE merge tool!

When merging several files containing related modifications, care must be taken to ensure that interdependent overlaps are resolved in a consistent manner.

For example, if an overlap in a header file (the number of parameters in a function prototype) is resolved in favor of developer A, all other files which define or refer to that function must be resolved in the same direction or they will be incompatible with the merged header file.

In addition, any of Developer B’s files which defined or referenced his original function (the one resolved in A’s favor) must be modified to call A’s function, even though they may not have been part of the merge - perhaps they were new files!

Best Practices for software engineering and project management suggest that a "data dictionary" approach would prevent overlaps of this type.