

Working With Tags, instead of Branches

The "traditional" /BriefCase approach to release management did not involve branching. To maintain multiple releases of a set of files comprising a product, consider the following:

1. When the tip revisions of all project files are finalized, a release administrator can use the `admtag` command, in the top-level project directory (e.g. `~/<project>%`), to tag them all with a "release tag". e.g.: `Rel-1_0`:

```
cd ~/<project>%  
admtag Rel-1_0
```

Then, regardless of any new development that has been checked in, all the files for `Rel-1_0` can be checked out into a new directory and built. For example, assuming a top down Makefile build process:

```
cd  
mkdir <project>%  
cd <project>%  
ntree  
ncoAll  
make
```

If new development and a maintenance release are to be developed in parallel, the administrator can assign "aliases" to the `Rel-1_0` tag using the `admtag_alias` command:

Usage: `admtag_alias [-f] [-R] [-u] TagName TagAlias`

'TagAlias' is added as alias tag to those revisions of files in the current working directory currently tagged with 'TagName'. The '-R' (recursion) flag causes TagAlias to be added to all files both in and under the current directory.

The -f flag causes any branch tag references in TagName to be frozen at the tip revision number of the branch in TagAlias.

Files with TagAlias already bound to a different rev/branch will cause an error message to be emitted and are left unchanged unless the -u (update) flag is specified.

(BCadmin only)

For example, to create a new development tag:

```
cd mkdir <project>%newDev%  
cd <project>%newDev%  
admtagalias Rel-1_0 newDev-001
```

and, to create a maintenance release tag:

```
cd mkdir <project>%maint1%
cd <project>%maint1%
admtagalias Rel-1_0 Rel-1_0-Maint_001
```

Developers working on both the new release and the maintenance release can make a work replica directory for each, and keep their maintenance work separate from work on new development for the next release.

All changes are checked in to the main trunk, so it is essential for developers to use the appropriate tags for all check-out and check-in operations! For example, in a new development replica directory (e.g: <project>%myNewCode%), a developer would start by checking out all the project files using the newDev-001 tag:

```
mkdir <project>%newDev-001%
cd <project>%newDev-001%
ntree
ncoAll -n newDev-001
```

Any changes (e.g. to fix bug # 043) should be checked in and tagged with a private tag of the developer's choice:

```
nci -n feature-01_<userid> <filenames>
```

If multiple developers are working on the same bug, in different files, they can use the same tag for check-in (which would have their userid in the tag).

After testing, the release administrator can incorporate the "feature-01" tagged files from each developer into newDev-001, again using admtag_alias:

```
TAGLIST='show_tags | grep feature-01'
for nn in $TAGLIST
do
    admtag_alias -R $nn newDev-001
done
```

The same strategy can be used to manage bug fixes in the maintenance release.

#-----

If bug fixes are made in the same files being worked on for new feature development, or vice-versa, their respective changes can be merged from one set of tags to another set of tags whenever desirable, using the release tag (Rel_1_0) as the base (common ancestor). For example, to merge fixBug-043 into a YOUR new feature-01 revision of file foobar.cpp (and only your version), you could, in any project working directory or replica:

cd to directory containing the file to be merged

```
nco -n fixBug-043_<otherUserId> foobar.cpp
vmerge foobar.cpp Rel-1_0 feature-01_<yourUserId> > mergedFile
ls -l foobar.cpp.mergeErrs
[ if there are no merge errors ]
ncol -b feature-01_<yourUserId> foobar.cpp
mv mergedFile foobar.cpp
vdiff foobar.cpp # to review the changes before check-in
nci -n feature-01_<yourUserId> foobar.cpp
```

#-----

To build and test a project in which some subdirectories have tagged changes (e.g.: bug fixes, new features) you want to include in the build, use the ncoAll -o (overlay) option to: 1) check out the main trunk, then 2) overlay the tagged revisions in the working directory. For example:

```
cd ~/<project>%
# check out the tip revisions from the main trunk
ncoAll
# overly fixes checked in by <userid1> & <userid2>
#    and enhancements checked in by <userid3>
ncoAll -o -n <fixBug-043_<userid1>
ncoAll -o -n <fixBug-043_<userid2>
ncoAll -o -n <feature-01_<userid3>
# top-down build
make
```

WARNING: that once a fix or enhancement is merged back into the main trunk, DO NOT use the overlay option for that fix/enhancement.

WARNING: if changes in any fixes and/or features conflict with each other (i.e.: make conflicting changes to the same files) the overlay will contain only the revision last overlayed.

In some cases this may be an easier alternative to branching, in other case, maybe not.

#-----