A Programmer's Guide to Working with /BriefCase Branches
$Revision: 1.13 $
by Dave Knight

1. Update to RCS. the underlying /BriefCase revision control engine

NOTE: RCS Release 5.9.2 distributed with Linux Mint 17 Rosa has a bug that causes branch tip/leaf check-ins to fail. Use the command:

```
rcs --version
```

to display the "ambient" rcs version on your server. If it is rcs release 5.9.2, you should build RCS 5.9.4 (included in this /BriefCase release) on the /BriefCase server and install it, e.g. in /usr/local/bin.

The server's /BriefCase/BCconfig should point RCSBIN to /usr/local/bin or wherever you installed it.

Once these steps are done, run the following command to determine if /BriefCase will use the new RCS version:

```
$ `BCwhence rcs` --version
rcs (GNU RCS) 5.9.4
Copyright (C) 2010-2015 Thien-Thi Nguyen
Copyright (C) 1990-1995 Paul Eggert
Copyright (C) 1982,1988,1989 Walter F. Tichy, Purdue CS
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

REMINDER: Many of the advanced revision and release management capabilities of /BriefCase use symbolic names (AKA tagnames, branchNames and branchTags) to simplify operations on arbitrarily large sets of file revisions. Because /BriefCase uses RCS as its revision control engine, /BriefCase users must follow certain conventions of the RCS toolkit. To wit:

1. Filenames may not contain whitespace or punctuation.
2. Symbolic links are ignored.
3. Symbolic names used as tag names and branch names may only contain letters, numbers, '-', '_' , parentheses and square brackets. Symbolic names may NOT contain a period ("dot"), most (any) other punctuation, the $ and @ signs or have embedded whitespace.
4. REMINDER: /BriefCase "tags" come in two flavors: private, which can be set by any user, and administrative tags, which can only be set by a user in the BCadmin group. The intent is for one or more developers to use private tags to mark different phases of their work product, e.g. those revisions that fix a specific bug or new feature in the product. When the developer(s) feel that a fix or feature is ready to be released, the private tags for those changes are provide to the release administrator who will "alias" then to an administrative "release" 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)

Administrative tags are intended to be used by /BriefCase administrators and/or release managers to assemble a "releasable" set of revisions containing those tags that have been release by their respective developers.

2. Why Use /BriefCase? A short [re]introduction.

2.1 Revision Control, General Benefits

Like most "revision control" tools, using /BriefCase to maintain revisions of a file allows retaining the evolutionary history of a file's content. Old revisions can, for example, be reviewed or recalled as necessary if deleted content becomes relevant again.

For software developers, extending this concept to multiple files, in multiple directories, allows for disciplined enhancement of software products whether they be programs that are built from many files and released in a "package" or a manuscript assembled from multiple chapters into a PDF or other distributable form.

2.2 The /BriefCase approach

To help users develop "discipline", related files are organized in a directory tree under one's home directory and that directory tree is mirrored in the /BriefCase server's repository.

To make a project for a directory, use the command:

Usage: newBCproj ProjectName Group [private|secure]

Command to start a new project. The project directory "ProjectName" is created in the repository, its groupid is set to "Group" and its permissions are set to rwxrwxr-x. All users have read access the the project but write access is limited to the project owner and members of "Group".

When the "private" flag is specified, perms are set to rwxrwx---, prohibiting even read-access by users not in "Group".

When the "secure" flag is specified, perms are set to rwx------, allowing access only by the owner.

If ProjectName exists, "Group" [and permissions] are updated.

(BCadmin only)

For example:

newBCproj rpi users

instantiates the /projRCS/rpi directory on the /BriefCase server. On any /BriefCase client host, you can access the files in and under the ~/rpi working directory using /BriefCase commands.

"Discipline" is maintained by making checked-in files read-only wherever they are checked out (nco command). To change a file, it needs to be checked-out and locked (ncol command). When the changes are finalized, check it in again, with the nci command.

2.3 Lock Integrity, Tags and Replica Working Directories

Basic lock integrity among users is provided automatically, preventing another user from checking in a change to the revision you have locked. Lock integrity across multiple hosts is enabled by adding a special suffix character to the working directory name, e.g. something like "~/rpi%". The suffix can be one of the /BriefCase DIRSEPS='[.%+=]' special characters, your choice. This prevents a file checked out and locked on one host from being accidentally checked in by another user or from another host!

Most project release management can be handled with tags (symbolic names, in RCS parlance). File revisions related to a specific bug fix or enhancement can be tagged so they can be easily identified by other users who may need to review or work on them, or add their own contributions to the tag.

/BriefCase provides "replica" working directories (with names like <projectName%replicaName%) to assist developers who need to work on one more sets of changes simultaneously, e.g.: multiple bug fixes and/or enhancements.

2.4 Branches and Branch Replica Working Directories

To maintain multiple variants of a project (mutually-exclusive sets of changes, for example), named branches can be very handy. /BriefCase provides tools for merging changes among branches & the main trunk.

In addition, /BriefCase Release 3.6 provides an easy way for developers to work on (or in) one or more branch revisions simultaneously: "branch replica working" directories. Inter-host, inter-branch lock integrity is enabled for branch replica working directories whose names look like:

<directory>%<branchName>%

The <branchName> part guides /BriefCase commands to operate on the named branch revisions instead of the main trunk revisions. The trailing percent character, '%', enables inter-host lock integrity.

I usually use a % sign as the separator in my working directory names, but you could use any of the /BriefCase "DIRSEPS" characters (. % + or =), your choice.

Unless you are a /BriefCase administrator or release manager, the only /BriefCase commands you are likely to use frequently are: nco, ncol, num nci, nciUndo, ntag and nretag. Use the "--" and "-+" flags on these commands (or any other /BriefCase command) to display their help info.

3. Why does /BriefCase use Branches?

IMPORTANT NOTE: /BriefCase "branches" are NOT "vendor branches" as imagined by CVS!

Many problems have been reported against CVS vendor branches and I chose not to invite them into /BriefCase.

/BriefCase branches are used to isolate maintainable branches of /BriefCase project files that differ significantly from the main trunk and/or other branches of the same project files.

To "import" files from another directory tree, copy it into the project's working directory and use the nciR command to add it to the main trunk. Branching of imported files, if necessary, can be done using the

features described here.

/BriefCase is designed to support development, by multiple developers, of software products built from numerous files, from numerous levels of subdirectories, during a life cycle of multiple releases, for which maintenance may be required while new releases are being developed.

Branches derived from named releases of software products, combined with /BriefCase's unique working "replica" capabilities, make it easy for individual (or multiple) developers to prepare maintenance releases for one or more prior releases simultaneously with ongoing development in the main trunk and other branches.

/BriefCase tag management features & tools provide the means to name sets of specific file revisions in a directory tree for a formal, maintainable release, and named branches can be used to mark development milestones or other groupings of file revisions useful to developers and management.

For projects in which significant changes may be necessary to produce software variants specific to a client or development "experiments" that might become a stand-alone release, get merged back into a branch or the main trunk", or abandoned, branching of the source files may be easier to manage than embedding target-specific notes/caveats or conditional constructs in the files.

While the underlying RCS tools provide commands and options to manage branches within individual files, /BriefCase extends those capabilities to operate on sets of file revisions that comprise a "project" and thus simplifies development and maintenance of multiple releases.

This guide is intended to be an introduction to and beginner's *cookbook* for working with /BriefCase branches.

4. Starting new Branches

Don't get discouraged by this multi-step process, it's a one-time thing (for each branch)!

Even if your project consists of only a single file, it is easiest to start a branch from a tagged release! Typically a set of file revisions is tagged by developers or a "release manager" when they are deemed ready to be released. The simplest way to get started is to make sure that the tip revisions on the main truck of the revision tree are checked in and ready to go. That set of revisions is then "tagged" using the "ntag command":

Usage: ntag [-R] [-u] [-b <branchName>] PTag

Unless -b <branchName> is specified, ntag will tag the (main trunk) tip revisions of all files in the current working directory with a PRIVATE symbolic name 'PTag'. The -R (recursive) flag tags all files in AND UNDER the current working directory.

The '-u' flag updates a previously assigned PTag to include the latest revisions. Without the -u flag, files already tagged with PTag are identified with an error message and left unchanged.

When the -b <branchName> flag is used, PTag is associated with the tip revisions of the branch named <branchName> instead of the tip revisions of the main trunk.

If PTag does not end with '_<YourUserid>' it will be appended and you will be prompted for permission to continue.

NOTE: PTag, in the description above marks the tip revisions of files in the main trunk or in a branch. The term <branchName> here and elsewhere refers to a "dynamic" tag that does not name a specific revision of a file, rather it identifies the branch and is used ONLY when operating on tip revisions of files in the main trunk or in a branch. When working in a "branch replica" directory (discussed below), the <branchName> part of the directory name causes check-out and check-in operations to operate on file in that branch instead of the main trunk.

1. To start a new branch, create a list of the files to be in the branch:
    1. For an existing set of tagged revisions in the main trunk, use:

        nrevs -ln <tagname> > flist

        Note that nrevs lists ALL files tagged with <tagname>, i.e. all files in the current directory and its subdirectories, so the list can be fairly long.

    2. For an existing set of tagged revisions in a branch use:

        nrevs -ln <branchTag> > flist

        Note that <branchTag> is really just a tagname that has been associated with revisions in a branch, for example by the command:

        ntag -b <branchName> <tagname>

        to tag the tip revisions in the branch <branchName> with the tag <tagname>.
    3. For the tip revisions of files in the main trunk, temporarily tag the tip revisions with "tempTag" for example:

        ntag tempTag

        then create the file list with:

        nrevs -ln tempTag > flist
    4. For the tip revisions of the files in a Branch, temporarily tag the tip revisions of the branch with:

        ntag -b <branchName> tempTag

        then create the file list with:

        nrevs -ln tempTag > flist

    If all the files in flist are to be part of the new branch initiate the new branch with the <tagname> from :

    ```
    for nn in ‘cat flist‘
    do
      bcol -r <tagname> $nn
    done
    ```

    If only some files in a named release are to be in the new branch, edit the file list (named flist, here) to contain only the files you want to branch, then run the for loop above.

The bcol command creates a branch node and locks the associated branch "bud" (revision numbers x.y.z.0, for example) for each file.

After this step, a temporary tag ("tempTag" in this example) created in step 3 or 4 can be deleted with:

```
nuntag tempTag
```

2. Check-in and tag the current files as the initial revisions in the branch:

```
for nn in ‘cat flist‘
do
  nci -n <branchTag> $nn
done
```

WARNING: DO NOT use the same name you intend to use as the branchName as a branchTag! That will cause all manner of confusion later on! To keep it simple, you could start branch NAMES with an upper-case ’B’, for example.

3. The last step is to name the branch for those files with:

```
for nn in ‘cat flist‘
do
  bName <branchTag> <branchName> $nn
done
```

For "efficiency" when operating from a list of tagged files, the bcol, nci and bName commands used to start a branch can be run in a single for loop. Note that **bName** always displays what it is about to do and prompts for confirmation.

Once a branchName is established with bName, using the BRANCH NAME in a check-out operation will automatically check out the tip revisions of the branch whereas check-out with a branchTag will check out the specific set of revisions bearing that name (which may or may not be tip revisions).

Using a branchName or branchTag for check-out operations eliminates the need to keep track of the actual branch/tip revision numbers for each file for future edits, checkouts and other operations on branch revisions.

5. Deleting Branches

To delete one or more branch revisions, or an entire branch, from a file, use nobs:

```
nobs -r :<revision> filename
```

where <revision> is either a revision number or tagname. If <revision> is the tip revision of a branch, the entire branch will be deleted, including the branchName.

If <revision> is NOT the tip of a branch (i.e. one or more revisions remain on the branch), the branchName will be preserved along with all revisions newer than <revision>:

```
Usage: nobs -r rev1[:rev2] filename

  Obsoletes/deletes/outdates  a revision [range] for file ’filename’

  A range consisting of a single revision number means that revision.
```

A range consisting of a branch number means the latest revision on that branch. A range of the form rev1:rev2 means revisions rev1 to rev2 on the same branch, :rev means from the beginning of the branch containing rev up to and including rev, and rev: means from revision rev to the end of the branch containing rev. None of the outdated revisions may have branches or locks.

(BCadmin only)

Note that, for /BriefCase installations with a /BriefCase administrator and/or release managers, the nobs command may be restricted to users in the BCadmin group.

6. CheckIn and CheckOut Operations for named branches.

Ongoing Branch CheckOut and CheckIn and other operations.

1. CheckOut & Lock for edit:

    1. to check-out & lock the tip revisions of a branch:

        ncol -b <branchName> [file ...]

    2. to check-out and lock a set of revisions tagged with a specific tagname:

        ncol -b <tagname> [file ...]

        Generally, it's only necessary to edit a few file revisions in a set tagged with <tagname>, but if the file name(s) are omitted here, all the tagged files will be checked-out and locked.

        WARNING: Because operations using <tagname> CAN bridge the branch infrastructure within a project, it is very important that tagnames used in one branch (i.e. "branchTags") differ from tagnames used in other branches, and in the main trunk! Tagname bridging can be useful in some ways, but usually leads to confusion and/or release management chaos.

2. CheckIns After editing:

    for tip revision without branchTag update, use: nci file [file ...]

    to update the branchTag, use nci -n <branchTag> file [file ...]

    If nci complains about multiple locks, use bci (for each file):

        bci -r <branchName> [-n branchTag] filename

    DO NOT use the branchName in the "-n branchTag" option on a check-in as that will cause future checkouts using the branchName to check out the specific revision as opposed to the tip revision of the branch.

    Generally, the nci and bci commands will not allow a branchName to be misused as a branchTag, emitting an error if attempted.

3. Tag Manipulations:

    Use the ntag commands as needed. Note that ntag has been enhanced to support tagging of the tip revisions of a named branch.

Summary examples:

1. To start a new named branch off a named/tagged release in the current directory:

```
nrevs -ln <tagname>  >  flist
for fn in ‘cat flist‘
do
      bcol -b <tagname>  $fn
      nci -n <branchTag>  $fn
      bName <branchTag>  <branchName> $fn
done
```

2. To checkout & lock the tip revisions of a branch for editing, use:

```
ncol -b <branchName>  [file [file ...]]
```

3. After editing one or more files, use nci to check in the changes:

```
nci ‘nout -l‘  # checks in all locked files as new tip/leaf revisions
```

or, to check-in and associate/update a <tagname> for those files:

```
nci -n <tagname> ‘nout -l‘
```

To associate/update <branchTag> with tip revisions of all changed or new files, locked or not:

```
nci -n <branchTag> *  # associates <branchTag> with all tip revisions
```

To limit <branchTag> association updates to only the changed files, use the -s flag:

```
nci -sn <branchTag> *
```

to suppress the tag update/association of files that are not locked.

WARNING: new files checked in with a <branchTag> will not be considered part of the branch <branchName>, hence will not be checked out & locked by:

```
ncol -b <branchName>  [file [file ...]]
```

UNLESS you also create and name the branch in the NEW file(s) by creating a list of the new files (only) named flist and running a for loop like:

```
for fn in ‘cat flist‘
do
      bcol -b <tagname>  $fn
      nci -n <branchTag>  $fn
      bName <branchTag>  <branchName> $fn
done
```

WARNING: do not do this for files already in branchName, as that will start ANOTHER

branch off <branchTag>! Use:

show_branches <filename>

and

vtree <filename>.

to review <filename>'s branchName, tagname and lock status.:

7. Recovering From Mistakes

To remove a branch started in error, e.g.:

```
$ bcol -r noRGB_dmk foo
bcol: Info - foo[noRGB_dmk] resolves to rev 1.1.1.1
Starting new branch 1.1.1.1.1 off 1.1.1.1
/projRCS/Arduino/newGeiger/foo,v  -->  standard output
revision 1.1.1.1
/projRCS/Arduino/newGeiger/foo,v  <--  foo
new revision: 1.1.1.1.1.0; previous revision: 1.1.1.1
done
RCS file: /projRCS/Arduino/newGeiger/foo,v
1.1.1.1.1.0 locked
done

$ vtree foo

vtree: /projRCS/Arduino/newGeiger/foo,v

       1.1.1.0
          1.1.1.1.1.0 (lock:dmk%leno)
        1.1.1.1.1 (branch)
       1.1.1.1   noRGB_dmk
     1.1.1 (branch) Dynamic Tags:  BnoRGB_dmk
   1.1
$
```

use the "nobs" command:

nobs -r <branchNumber> <file>

where <branchNumber> is the NUMBER from the "locked" line from bcol or the (lock:<userid:host>) line from vtree, e.g.:

nobs -r 1.1.1.1.1.0 foo

in the example above.

To remove an unwanted <branchTag> from all files in (and under, with the -R flag) the current directory, use:

nuntag <branchTag>

Use the -R flag to recurse thru subdirectories if desired:

nuntag -R <branchTag>

To remove an unwanted tag from an individual file:

nretag <filename> - <branchTag>

8. Branch Management Tools

To review the current branches and revisions for one or more files, use the vtree command:

vtree [-v] file ... # show revision tree for file(s)

To see a concise list of branch names and branch tags in a file use:

show_branches [file ...]

if no filename is specified, branches are listed for all files in the current working directory.

For example:

```
dmk@nas2 ~/Arduino%/newGeiger $ show_branches
BranchName: B_SDlogger_dmk -> 1.3.2.1.1
BranchName: BnoRGB_dmk -> 1.3.2
  file: OpCheckSources contains branch: 1.3.2
  file: OpCheckSources contains branch: 1.3.2.1.1
BranchName: Btestit_dmk -> 1.1.1
BranchName: BnoRGB_dmk -> 1.1.1
  file: bar contains branch: 1.1.1
BranchName: BnoRGB_dmk -> 1.1.1
  file: foo contains branch: 1.1.1
BranchName: B_SDlogger_dmk -> 1.1.2.1.1
BranchName: BnoRGB_dmk -> 1.1.2
  file: gpl-3.0.rtf contains branch: 1.1.2
  file: gpl-3.0.rtf contains branch: 1.1.2.1.1
BranchName: B_SDlogger_dmk -> 1.31.2.5.1
BranchName: BusePin3_dmk -> 1.24.1
BranchName: BnoRGB_dmk -> 1.31.2
  file: newGeiger.ino contains branch: 1.24.1
  file: newGeiger.ino contains branch: 1.31.2
  file: newGeiger.ino contains branch: 1.31.2.5.1
BranchName: B_SDlogger_dmk -> 1.7.2.6.1
BranchName: BnoRGB_dmk -> 1.7.2
  file: newGeigerUI.txt contains branch: 1.7.2
  file: newGeigerUI.txt contains branch: 1.7.2.6.1
dmk@nas2 ~/Arduino%/newGeiger $
```

To see a sorted list of the tags existing in and under the current directory, use:

Usage: show_tags

```
Displays a list of all tags currently in use under the
current directory to stdout.
```

```
dmk@nas2 ~/Arduino%/newGeiger $ show_tags
b4_lcdUI_dmk
BnoRGB_dmk
B_SDlogger_dmk
Btestit_dmk
BusePin3_dmk
noRGB_dmk
Rel-0_1_dmk
Rel-0_2_dmk
Rel-0_3_dmk
Rel-0_4_dmk
Rel-0_5a_dmk
Rel-0_5_dmk
Rel-0_6_dmk
Rel-0_7_dmk
RHelec-01_dmk
SDlogger_dmk
test_dmk
usePin3_dmk
dmk@nas2 ~/Arduino%/newGeiger $
```

9. Branch Replica Directories Make Branch Maintenance Easy

Working with Branch Replica directories.

"Branch Replica" directories are work replica directories whose replica suffix is a Branch Name. Once a named branch is established (as above), if you create a project work replica directory named <project>%<branchName>%, the nci, nciUndo, nco, ncoAll, ncol and ntag commands will operate (by default) only on file revisions belonging to branchName! For example, running nco (and ncol) with no filename(s) will checkout (and lock) the tip revisions of all files in that branch, in that directory. Running ncoAll with no arguments will checkout all files from that branch in and under the current directory. Running 'ncoAll -l' will check out and lock the tip revisions of all files from that branch in and under the current directory.

The tip revisions of a branch will be tagged when ntag is run in a branch replica directory.

By using work replicas with branchName suffixes a developer can more easily work concurrently on several aspects of maintaining or enhancing one or more projects simply by moving from one branch replica directory to another.

Of course, this can lead to a user having multiple locks on different revisions (from different work replicas) of a file and nci will generally do the right thing based on the lock info. If nci complains about multiple locks, use bci to check in. To minimize multi-lock difficulties, be sure to name your branch replica directories like this:

<projectName>%<branchName>%

As mentioned above, the trailing % sign enables inter-host lock integrity.

To limit the proliferation of project replica directories (local and in the /Stage tree), use the ndelrep command to clean things up:

ndelrep <repdir> [<repdir> ...] # delete a work replica and /Stage area

Illustrative Examples:

1. To build using the tip revisions from the main trunk, cd to a project working directory, e.g. ~/<projectName>% and run:

   ncoAll

   to check-out the tip revisions of all components of the main trunk. Then run make or whatever toolchain is used for that project.

   Edited files can be checked in using the nci command.

2. To build a specific tagged release, cd to a project's working directory and run:

   ncoAll -n tagname

   to check-out all the components of that release. Then run make or whatever toolchain is used for that project.

   Edited files can be checked in using the nci command with specifying the tagname to be updated:

   nci -n tagname file [file ...]

   If edited files are checked in without the -n tagname flag they will simply be added as a new tip revisions on the main trunk and the tag will not be updated to include the new revisions.

3. To build using the tip revisions of a named branch, cd to the branch replica working directory, e.g.: ~/<project%branchName%> and run:

   ncoAll

   to check-out the tip revisions of all components of that BRANCH. Then run make or whatever toolchain is used for that project.

   Edited files checked in using the nci command:

   nci file [file ...]

   will become new tip revisions on that branch. The -n flag can be used to maintain tagged releases in a branch:

   nci -n <branchTag>

   WARNING: Take GREAT care to use branchTag names that are different from tagnames used for tagged releases from other branches and from the main trunk, otherwise, release management chaos will ensue! Use the show_tags command to list the existing tags before establishing a branchTag!

   The nci & bci commands will try to prevent this, but care is warranted anyway.

10. Where to find Help for /BriefCase Commands

The primary (though somewhat outdated) resource is the Developer's Handbook found in the /BriefCase/docs directory. For quick (up-to-date) help with any /BriefCase command, enter the command with just the "-+" flag to display a one-line description:

```
$ nretag -+
nretag [-f] FileName {+|-|Rev} PTag[,PTag...]  # chg/remove PTag(s) in FileName
```

or the use the "--" flag to display a "manpage" style description:

```
$ nretag --

Usage: nretag [-f] FileName {+|-|Rev} PTag1[,Ptag2...]

  Associate one or more private tags with revision Rev of file FileName.

  If a tag does not end with '_<YourUserid>' it will be appended
  and you will be prompted for permission to continue.

  If Rev is a plus ('+'), the tag(s) will be added/moved to the latest
  Revision of FileName.  If Rev is a dash ('-'), the tag(s) will be
  removed from file FileName.

  Frozen tags (i.e. tags beginning with the upper or lower case letters
  'frozen'  will not be moved unless the -f flag is specified.
```

11. A Small Project Example

This example illustrates a small project with multiple branches and branch levels. The main trunk of the newGeiger Arduino project supports an RGB LED on pins 11, 12 & 1.

One client did not want the RGB LED support, so the branch BnoRGB was made, and the RGB LED support stripped out of newGeiger.ino and newGeiger.txt.

Another client wanted use pins 11, 12 & 13 for an SDcard, so a new branch was started off the BnoRGB branch tip revisions.

Here is part of the vtree output for newGeiger.ino:

vtree: /projRCS/Arduino/newGeiger/newGeiger.ino,v

```
  ...
  1.16
  1.17
  1.18   Rel-0_3_dmk
  1.19   Rel-0_4_dmk
  1.20   Rel-0_5_dmk
  1.21
  1.22
  1.23
     1.24.1.0
     1.24.1.1
     1.24.1.2
     1.24.1.3   usePin3_dmk
     1.24.1.4
   1.24.1 (branch) Dynamic Tags:  BusePin3_dmk
  1.24   Rel-0_5a_dmk
  1.25
  1.26
  1.27
  1.28   Rel-0_6_dmk
  1.29
  1.30
     1.31.2.0
     1.31.2.1
     1.31.2.2
     1.31.2.3
     1.31.2.4   RHelec-01_dmk
        1.31.2.5.1.0
        1.31.2.5.1.1   SDlogger_dmk
        1.31.2.5.1.2 (lock:dmk%B_SDlogger_dmk%leno)
      1.31.2.5.1 (branch) Dynamic Tags:  B_SDlogger_dmk
     1.31.2.5
   1.31.2 (branch) Dynamic Tags:  BnoRGB_dmk
  1.31 (lock:dmk%leno)   Rel-0_7_dmk
```

I'm thinking of adding a serial GPS feed on the Arduino pin 12 to provide the so that the SD log entries can contain the actual date, time and GPS coordinates for the log entries. That could be done in the B_SDlogger branch or in a new branch (off the B_SDlogger branch).

Note that, although /BriefCase can handle multiple branch levels, branches off branches become increasingly difficult to manage (intellectually), despite the availability for branch replica working directories.

Unfortunately, adding the the GPS & SD logging code (imported from another project) exceeds the memory available on the target platform (Arduino Uno - 32K flash, 2K RAM). So, the B_SDlogger branch will be abandoned in favor of starting another project in which files from the GPS_SDlogger project will be enhanced with the basic counting & calculating code from newGeiger.ino.

Usually it is not necessary to delete an abandoned branch from all the files involved but, if you want to do that, it's another multi-step process. For example, cd to a top-level project working directory (not necessarily a branch replica directory):

1. Create a list (flist) of the files in the branch:

   nrevs -ln <branchName> | tee flist

   For example:

   ```
   dmk@leno:~/Arduino%$ nrevs -ln B_SDlogger_dmk | tee flist
   ./newGeiger/OpCheckSources
   ./newGeiger/gpl-3.0.rtf
   ./newGeiger/newGeiger.ino
   ./newGeiger/newGeigerUI.txt
   dmk@leno:~/Arduino%$
   ```

2. See if any of the branch revisions are locked:

   nout -A | grep <branchName>

   ```
   dmk@leno:~/Arduino%$ nout -A | grep B_SDlogger_dmk
   ./newGeiger/newGeiger.ino revision 1.31.2.5.1.3 locked by: dmk%B_SDlogger_dmk%leno;
   ./newGeiger/newGeigerUI.txt revision 1.7.2.6.1.1        locked by: dmk%B_SDlogger_dmk%leno;
   dmk@leno:~/Arduino%$
   ```

   and unlock them with the num command:

   num

   For example:

   ```
   dmk@leno:~/Arduino%$ num -r 1.31.2.5.1.3 ./newGeiger/newGeiger.ino
   RCS file: /projRCS/Arduino/./newGeiger/newGeiger.ino,v
   Revision 1.31.2.5.1.3 is already locked by dmk%B_SDlogger_dmk%leno.
   1.31.2.5.1.3 unlocked
   done
   Checking out specific revision: -r 1.31.2.5.1.3
   /projRCS/Arduino/newGeiger/newGeiger.ino,v  -->  standard output
   revision 1.31.2.5.1.3
   --- UPDATED File ./newGeiger/newGeiger.ino
   dmk@leno:~/Arduino%$
   dmk@leno:~/Arduino%$ num -r 1.7.2.6.1.1 ./newGeiger/newGeigerUI.txt
   RCS file: /projRCS/Arduino/./newGeiger/newGeigerUI.txt,v
   Revision 1.7.2.6.1.1 is already locked by dmk%B_SDlogger_dmk%leno.
   1.7.2.6.1.1 unlocked
   done
   Checking out specific revision: -r 1.7.2.6.1.1
   /projRCS/Arduino/newGeiger/newGeigerUI.txt,v  -->  standard output
   revision 1.7.2.6.1.1
   --- UPDATED File ./newGeiger/newGeigerUI.txt
   dmk@leno:~/Arduino%$
   ```

   If any of the locks are owned by other users, use the **num -U <revNo> <file>** option to forcibly remove the lock.c

3. Use the ntag command to tag the tip revisions of all file revisions in that branch with a special tag like "deleteMe":

   ntag -b <branchName> <specialTag>

   For example:

   ntag -b B_SDlogger_dmk deleteMe

   note that for projects with many files and subdirectories, this may take a while.

4. Finally, use the "flist" created in step 1, the nobs command to obsolete the range of revisions ending with that special tag. Because that tag is associated with the tip revision of that branch, all the revisions on that branch, and the branch node itself will be removed from that file, e.g.:

   ```
   for nn in ‘cat flist‘
   do
     nobs -r :deleteMe_dmk $nn
   done
   ```

Note that if any of the file revisions in the branch have sub-branches, they will not be deleted.