An Arduino Sketch Development Example
$Revision: 1.2 $
by Dave Knight

1. Starting an Arduino sketch repository

Assuming you have numerous Arduino sketches organized under a "sketchbook" directory, it is really easy to migrate them into a /BriefCase project.

Suppose all your sketches are organized under the directory $HOME/Arduino, e.g.:

```
Arduino
  myBlinker
  LCDtest
  GPStest
```

you (or your /BriefCase Admin) could start your Arduino "project" with the command:

```
newBCproj Arduino users
```

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)
```


Once that's done, cd to your Arduino directory and use the nciR command to check in all your sketches:

```
cd Arduino
nciR *
```

Usage: nciR [-n tag] [-p] [-l] file_or_directory_name ...

  Similar to nci, except that it recursively descends subdirectories
  and nci's all files that it finds. May be used to check in new
  project files/trees after starting a new project with the 'newBCproj'
  command or to add new subdirectory (trees) to an existing project.
  When the -l flag is used, the checked in revision acquires the lock
  of its predecessor, as if it was ncol'd after checkin. preserves RCS
  Keyword information (e.g. $Revision:... $) from the file being
  checked in. If the -n tag flag is specified, the private tag tag
  is associated with each new leaf revision. For example, to check-in
  the all files in and under my_project:

        $ cd $HOME/my_project
        $ nciR .
        Enter comment for check-in: be-al and end-all project
        ...

  or, at some later date, to check-in all the test directory trees under
  my_project (named test.alpha test.beta ...):

        $ cd $HOME/the_project
        $ nciR test.*
        Enter comment for check-in: system test
        ...

  Project directories in the /BriefCase repository are created as needed.

  NOTE: The (one line) comment should generally describe the 'project' or
        the general nature of the directory (trees) being added, as it will
        appear as the RCS 'description' for all checked in files.

After checking in the files, your copies will be read-only. To modify a file, you need to obtain a writable
copy using the ncol (checkout and lock) command:

Usage: ncol [-b BranchTag] [file ...]

   Similar to nco except that retrieved files are writable and
   'locked' against changes by other developers. Files locked by
   ncol must be nci'd (checked in) or unlocked with the 'num' command
   to release the lock and allow others to change it. Ncol'd file
   revisions are NOT locked against nco (read-only) retrievals.

   ncol differs from nco in that it operates only on the latest
   (tip) revisions of non-Zapped files, or the tip of the branch
   named in the -b flag

   Unless the -b <branchTag> flag is specified, when ncol
   is run in a work replica directory whose suffix is a Branch Name,
   ncol will check out and lock the tip revisions of file sin that
   branch, as if '-b <branchName>' was specified, instead of tip
   revisions on the main trunk.

   See also: bcol, nci, bci

2. Accessing the Project Files

Assuming you will want to access these projects from more than one host, you would be well advised to change the name of your "Arduino" directory to "Arduino%", to enable cross-host lock integrity.

You can go to any /BriefCase client host and check out any or all of these Arduino sketches:

```
ssh <client host>
mkdir Arduino%
cd Arduino%
ntree
```

The ntree command mirrors the Arduino project's directory tree on the current client host:

Usage: ntree

     Builds or updates the current project directory tree.

After running ntree, the directory trees for all the sketches will be present. To check out the files for a particular sketch (read only), cd to its directory and run the nco command:

```
cd myBlinker
nco
```

If there are subdirectories for a sketch, use the ncoAll command to recursively check out the files in all its subdirectories:

Usage: ncoAll [-o] [-l|-n TagName|-d date|-r revNo]

   Check out files in and under the current directory.

   The -l flag will lock all tip revisions as they are checked out.

   Checkouts by Tagname (-n) are typically used to retrieve the set
   of files which comprise a particular (tagged) release.

   Checkouts by date (-d) are typically used to retrieve the set
   of files as they existed (as checked into the repository) on
   a specific date.

   NOTE: for -n and -d type checkouts, local read only files not
   checked in as of the specified date (for -d) or not part of the
   specified tagged release (for -n) will be deleted in an attempt
   to restore directory integrity to the date or tagged release
   baseline. Be aware that writable files are preserved and, if
   you have manually made a read only archived file writable, it
   will not be replaced by the proper tagged/dated revision!

   Checkouts by revision number (-r) will check out the specified
   revision number, if it exists, or the tip revision and no read-only
   files will be removed. This option should not be used unless all
   files are updated in parallel (i.e. all have the same revision
   numbers).

   The -r, -n -d and -l flags are mutually exclusive.

   Unless the -o (overlay) is specified for type -n and -d checkouts,
   tag name and date-specific file set integrity is preserved by removing
   any READ-ONLY files which are not in the set of revisions specified
   by the -n or -d flag, or not in the set of tip revisions when no
   flags are specified.

   The -o (overlay) flag is useful for joining sets of tagged revisions.
   However, if a file belongs to more than one tagged set, the resulting
   file will be the revision associated with the last tag containing
   the file.

   Zapped files are considered during -r, -d and -n type checkouts.

   Writable files in the current work replica are preserved and Warning
   messages are emitted for writable files which would otherwise have
   been removed or overwritten.

   Note that the current directory tree controls which directories
   are checked out. This means that an ncoAll in a pruned work tree
   may be used for a partial checkout. To ensure a full checkout, use
   the ntree command at the top of the tree to (re)create all project
   directories. Directories which exist in the repository but not in
   the current work replica are skipped.

   NOTE ALSO that with this release, when ncoAll is run in a project

replica directory whose replica suffix is a Branch Name, it will
operate on that branch instead of the main trunk.

Stdout and stderr are logged to the file ./ncoAll.log.

Running the ncoAll command in the $HOME/Arduino% directory will check out all checked-in sketches.

For normal edit/compile/test iterations I like to use the following set up to working on Arduino sketches:

1. Navigate to the sketch directory: cd ˜/Arduino%/myBlinker

2. In a cmd-line/shell widow, use the ncol command to obtain/lock a writable copy of the files youj
    want to change (.ino, .cpp, documents, etc.) and start your favorite text editor to work on your
    changes

3. In another command-line window, start the Arduino IDE in background:

    arduino myBlinker.ino &

    Compiler errors & other tool-chain messages will be displayed in this window (which easier to read
    than in the IDE's tiny output window). Start this in background, so that the "clear" command can be
    run to get rid of screen clutter and make it easer to see tool-chain error mesages.

4. When your changes are ready to test, save the file(s) and use the IDE to compile and upload to a
    lurking Arduino board.

5. If there are tool-chain errors, address them in the editor window, save the changes and compile
    again. When there are no errors, use the IDE to compile & upload.

6. When you are satisfied with the test results, exit the editor and use the nci command to check-in
    your modified file(s):

    nci myBlinker.ino

Usage: nci [-l] [-i] [-n tag] [-s] [-p] [-x] file ...

Check in modified version(s) of file(s). When prompted for a 'comment',
please enter a one-line description of the change(s) - include a Bug/Fix
reference number, if possible. When the -l flag is used, the new version
acquires the lock of its predecessor, as if it was ncol'd after check-in.
The -i (interactive) flag causes a prompt before each file is checked-in:

checkin filename [comment]? (y/n/c) >

Checkin is skipped if response is 'n'. The 'c' response
prompts for a new comment before checkin:

Change [current comment]:

Enter new comment text to replace 'current comment' or hit
ENTER to cause the 'current comment' to be (re)used as if user
had entered 'y' instead of 'c'.

When -n tag is specified, the private tag '<tag>_<userid>' (where
<tag> is the specified tag and <userid> is the current users id), is
assigned or updated to reflect the checked in revision(s), including
those which 'revert' to the original revision because there were
no changes. Use the -n flag when you wish to tag a set of files
containing one or more updated revisions, such as a set of files
to be built or released as a whole or 'published' (with BCpublish)
for import with the vximport command. If you wish to have the tag
applied ONLY to those files which are actually updated (such as the
files which fix a bug that is to be incorporated into a release),
use the -s flag to suppress tagging of unchanged/reverted revisions.

When the -p flag is specified, RCS Keyword expansions in the file
being checked in will be preserved. For example, a file containing
the $Revision: 1.2 $ RCS keyword expansion will be checked in
as rev 2.9. See the RCS ci command's '-k' flag description for
more info.

On initial check-in, identString and keyword expansion are handled per
the settings in the project (or default) ident_config. For subsequent
check-ins, if the keyword expansion setting for a file is inconsistent
with the current ident_config settings INFO messages to that effect will
be emitted. Specifying the -x flag will cause automatic update of the
keyword expansion to match ident_config and appropriate '#ident string'
action, per the (default or) project-specific ident_config configuration
(managed with the vi_project cmd).

Names which are directories or symbolic links are not checked it by nci.

See also: nciR, bci, vi_project, nkw_exp, BCpublish, vximport

Suppose, after several iterations of changes, you decide that this is not where you want to go with this
sketch. Assuming you've been diligent in your use of the ncol/nci commands at various stages of
development, you will be able to look at the list of revisions using the vlist command:

Usage: vlist [-r range] filename ...

   Prints the revision history for one or more files to stdout. If a
   range of revisions is specified (i.e. -r 1.4:1.6) the report excludes
   information about revisions outside the range.

   A range rev1:rev2 means revisions rev1 to rev2 on the same branch, :rev
   means revisions from the beginning of the branch up to and including rev,
   and rev: means revisions starting with rev to the end of the branch
   containing rev. An argument that is a branch means all revisions on
   that branch. A range of branches means all revisions on the branches in
   that range.

and "restore" the revision before your changes started down the wrong path. There are two ways to do this:

  1.   use "vget" to retrieve the last "good" version:

      vget myBlinker.ino > good.ino

     Usage: vget filename [revision]

      Retrieves a 'revision' (see nco for the ways to specify a specific
      revision) of TEXT file 'filename' and writes it to stdout.

     and check it in as the tip revision:

      ncol myBlinker.ino
      mv good.ino myBlinker.ino
      nci myBlinker.ino

  2.   or use (you or a /BreifCase Admin) "nobs" command to "obsolete" the errant revisions:

     nobs -r <badRev1>:<badRev2> myBlinker.ino

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. If rev is the
tip of a branch, and there are no locks or sub-branches in
that branch, the branch itself will be removed. The form rev:
obsoletes from revision rev to the end of the branch. If any
revisions remain in the branch, the branch will not be removed.
If any revision AFTER the revisions to be delete have branches or
locks, no revisions will actually be deleted, despite the "deleting
revision ..." messages.

Note that rev can be a tagname or a revision number, but not a
branchName.

WARNING: nobs ignores tagnames associated with revisions to be
deleted. Be careful not to obsolete revisions required for a tagged
release, as that will preclude rebuilding that tagged release.

Use the "show_tags -v" command to list the tags used by files in
the current directory. Use the vtree command to examine the full
revision tree for this file, and other files that may be part of
a tagged release, before obsoleting any revisions.

(BCadmin only)

Note that the 2nd option permanently removes all the "errant" code from the file's history,
eliminating possibility of restoring it, should you change your opinion of its usability.

There is much more that /BriefCase can do, especially for release management, but for basic development,
this should get you started.