

1. Intro to Basic /BriefCase Operations

1.1 Essential Terms

- /BriefCase Server - the host on which the repository lives
- /BriefCase Client - the host from which the user accesses the repository (the same host may be both the server and a client)
- Project - a collection of source files organized into a directory "tree".
- Project Name - the name by which the repository knows the project, given when the project was initialized by a /BriefCase Administrator with the **newBCproj** command. A list of know projects is displayed by the **show_projects** command.
- Project Work Directory or Replica - a directory immediately under the user's home directory (\$HOME or ~) consisting of the project name plus an optional replica identifier suffix.
- Replica Identifier - a suffix appended to a project name with a "separator character" (.%+=) used to identify different replicas (project work areas). For example, "\$HOME/Foo.bug3" could be a replica of the "Foo" project in which the owner is working on a fix for "bug3". The Project Work Area Replica Naming Convention is:

<project> - basic RCS-style lock integrity based only on userid, e.g. lock value: <userid>

<project><SEP> - enables cross-client-host lock integrity, lock value: <userid><SEP><hostname>

<project><SEP><description> - enables cross-replica lock integrity, lock value: <userid><SEP><description>

<project><SEP><description><SEP> - enables cross-replica lock integrity, lock value: <userid><SEP><description><SEP><hostname>

where "<SEP>" is one of the separator characters (.%+=).

/BriefCase's expanded lock integrity precludes accidental checkin of changes from the wrong userid, replica or host.

- Repository - a central storage area for project source files.
- Separator Character - one the characters : ., %, + or = used to separate the project name from its replica identifier or as a trailing character to enable hostname lock integrity (or both).

1.2 How to get started with a Project

Assuming that your host and userid have been properly set up for /BriefCase access:

1. list known projects with the **show_projects** command

```
-----
Usage: show_projects
```

Displays list of existing /BriefCase projects on stdout.

2. choose the project you want to access and create a directory by that name (or a replica name which enables the level of lock integrity desired)
3. cd into the new directory
4. enter the **ntree** command to build the project directories (in the image of the project directories known to the repository)

Usage: ntree

Builds or updates the current project directory tree.

5. check out read-only the files you need using the **nco** or **ncoAll** commands.

Note that **ncoAll** checks out files in the current directroy and into directories *which exist* under the current directory. Thus, ncoAll can be used to checkout a subset of the project tree by "pruning" unwanted directories out of the tree created by **ntree**.

Usage: nco [-k] [-r RevisionNumber|-n TagName|-d date] [file ...]

Retrieves read-only copies of checked-in file revisions by revision number, tagname or date. If none of -r, -n or -d flags are specified, the latest (tip) revision(s) of the named file(s) are checked out. If no file name is specified, all checked in files in the current directory are retrieved.

To be as consistent as possible with the the notion of a release, the -n or -d options will cause your local, read-only copies of files not tagged with TagName or which did not exist on the specified date to be removed. Writable files in your working directory will not be overwritten or removed under any circumstances; an error msg is emitted.

The -k flag suppresses RCS keyword expansion for this checkout only.

Usage: ncoAll [-r RevisionNumber | -n TagName | -d date]

Checkout all project files in and under the current directory tree. Note that when checking out a specific release with "-n TagName", directories having no files that are part of the Tagged release (and all their subdirectories) are skipped.

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

6. check-out writable, locked copies of the file(s) you want to modify using the **ncol** or **bcol** command

Usage: ncol [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 allow others to change it. Ncol's files are NOT locked against nco (read-only) retrievals.

ncol differs from nco in that it operates only on the latest (tip) revisions.

See also: bcol, nci, bci

Usage: bcol -r RevNo file

Where RevNo refers to a specific revision number or a branch number. Revision numbers have an even number or components (e.g. 1.2) and branch numbers have an odd number (e.g. 1.2.3). When:

RevNo	Type	Exists	Result
1.2	rev	yes	usr prompt: start new branch or co/lock?
1.9	rev	no	error
2.3.1	branch	yes	co/lock tip/leaf of branch
2.3.1	branch	no	error

Checkout and lock (co/lock) requests will fail if the target revision is locked by another user. To check-in a locked branch revision, the nci command will work Unless the current user[replica[host]] owns more than one lock on this file. For example:

```
bcol -r 1.2 foo
... editing ...
nci foo      # checks in branch rev 1.2.5.1

bcol -r 1.2.1 foo # co/lock tip rev of branch 1.2.1
... editing ...
nci foo      # checks in foo as new tip on branch 1.2.1
```

If nci complains that the user owns more than one lock, the bci command can be used to check-in the revision against the appropriate lock:

```
bci -r 1.2.1 foo # checks in foo as new leaf on branch 1.2.1
```

Proper use of project working directory replicas should eliminate most occasions where a user[replica[host]] might need more than one lock on a single file.

see also: bci num nci

7. After your change is complete (or when you want to establish a check-point or roll-back point for complex changes), use the **nci** or **bci** command:

Usage: nci [-l] [-i] [-n Ptag] 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 Ptag is specified, the private tag "<Ptag>_<userid>", where <Ptag> is the specified tag and <userid> is the current users id) is associated with the checked in revision(s). Private tags can be managed using the ntag, ntag_alias, ntag_diffs, ntag_report and nuntag commands.

At checkin, an 'ident string' may be to files whose names match a pattern in either the default list (/BriefCase/ident_config) or a project-specific .ident_config file.

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

See also: nciR, bci, vi_project

Usage: bci -r REV [-n Ptag] filename

When REV specifies a BRANCH (i.e. has an odd number of parts: 1.2.3) bci checks in a new leaf/tip on branch 'REV' of file 'filename'.

New Branches must be started using the 'bci' command.

Bci can also be used to check in a specific, non-sequential revision. e.g to start a new major revision like 2.1 or 4.1., but will fail if the tip revision is not locked.

If the named revision exists, the revised file will be checked in as the next rev no (as with nci) thus 'bci' should be used to checkin a specific locked revision when 'nci' complains about multiple locks.

The -n Ptag flag, if specified, associates the tag Ptag with the new revision.

Branches have REVs like 1.4.2 (and odd number of parts) and Branch leaf Revisions have REVs like 1.4.2.7 (an even number of parts).

Filenames which are directories or symbolic links are not checked in.

See also: nci bciR

See the /BriefCase Reference and User's Guide, Chapters 2, 13, 15, 16 and 17 for more info on basic usage of /BriefCase and software development methodologies that take advantage of the unique features of /BriefCase. Chapters 4 thru 10 deal with advanced topics like Release Management and packaging, Branch Revisions and Merging, Named (Tagged) releases, Project Administration and /BriefCase installation and administration.

Chapters 11 and 12 are a /BriefCase command summary, by feature group and command reference, respectively.

In addition to the table of contents, this manual is fully indexed.