Code Commit Procedure

Process

1. Personal code review
2. Peer code review
3. Commit log entry
4. Commit! (single or multiple)

One commit or many?

If your changes are unrelated and don't need to go out together, consider doing your commits as several separate ones. This is especially important if you plan to merge the commit to another branch. You should never have a commit that only some of the files are then merged to the other files--that's a clear indication that it should be multiple commits.

A common scenario where using a single commit for unrelated changes is often necessary is one where the commit introduces breaking changes and new binaries are being pushed along with the check-in to fix downstream dependents. Since any breaking change should be committed along with a full build, it's usually better to have those together as a single commit.

Commit log

Each log entry should be detailed and informative. For larger commits, break down the components by file or by entities. Here are some examples:

```
<project> (<svn branch>): reviewed by <reviewer>
* BUG 1234: Bugs resolved in a commit need to be mentioned like this
* detail each functionally separate change
  * use indented sub-bullets for more complex details
* if the details don't obviously tie the change to a file, mention the file/class
```

Bug fix commit log

Bug fixes ALWAYS require the bug information. After commiting a bugfix, update the bug report with the revision the fix is at. Backports also require bugs to commit.

```
Bugfix #XXX: bug title goes here
 * Backport r.REVBACKPORTING to <branch>
```