bt: A build tool for LSST code

Goals:

- Enable the developer to easily bootstrap the development environment by downloading all the code necessary to build a specific package
- Enable the developer to easily download and build the code for a ticket they're reviewing
- Enable a tester to easily download and build the code they've been asked to test
- Enable the developer to rebuild a set of packages from source, given a list (a manifest) of how they've been built before
- Provide interface to buildbot to do periodic and on-demand builds
- Enable automated builds and tagging of release branches

Example workflows for the build tool

```
# Initializing the build directory (the directory into which the repositories will be cloned to)
bt init

# Configuring the build directory
bt config upstream.pattern <pattern list>                        # where to find the git repositories (now
known as REPOSITORY_PATTERN)
bt config exclusions <filename>                                                # list of exclusion regexes,
when following the dependency chain
bt config versiondb.url <versiondb repository url>        # URL of the version database (mapping between
(product,version,deps_versions) -> +N suffix)
bt config versiondb.writable <true|false>                        # Whether versiondb is read only or writable
as well (ro is the default, only the CI system can write)
bt config build.prefix <string>                                                # EUPS build tag prefix
(defaults to username if unspecified)

# ... or do the config using a remote config file
bt init --config http://....

# Cloning all repositories required to build lsst_devel and lsst_sims products
# and their master. bt remembers that lsst_devel and lsst_sims are top-level products.
# bt build builds the remembered top-level products (and their dependencies)
bt pull lsst_devel lsst_sims
bt build

# Cloning/pulling like above, but checking out tickets/DM-1234 (if it exists, falling back to master otherwise)
# [note: if no repositories are specified on the command line, bt pull will pull the remembered ones]
bt pull --ref tickets/DM-1234
bt build

# Continuing a build after a code change (say, after a build failure)
# The -a flag tells build to recompute versions (otherwise it will refuse to build a "dirty"
# tree -- "dirty" compared to the state just after pull)
bt build
... build fails ...
... chdir, edit, commit, push, etc ...
bt build -a

# Clone/pull given a build manifest. Build manifest is a text file of products,SHA1,version,dependencies tuples.
# This allows one to reproduce a build someone else made (with bt).
bt pull --manifest manifest.txt
bt build


# Push changes to all repositories (top-level + dependences)
bt push

# Create a branch in all repositories
bt branch release/8.1.0

# Tag all repositories (matching tag.pattern config parameter)
bt tag 8.1.0.0

# Increment version tag on all repositories matching tag.pattern. The goal is to allow simple increments of .W
releases
```

```
# on release branches.
#
# For each repository, this will:
#   * check if there's an annotated tag on HEAD. If yes, continue to next repository.
#   * Find the closest tag in history. If the tag prefix matches the version part of the
#     branch name (e.g., in release/8.1.0, that would be 8.1.0), split off any remaining
#     integer <N> from the end (e.g., if the tag is 8.1.0.5, N=5). Otherwise, see if the
#     tag matches <prefix>-<version>.N (e.g., a tag 1.3.4-8.1.0.7 would match this
#     pattern, with N=7). Otherwise, assume <prefix>==tag, and N=-1
#   * Increment N by one (check for tag collisions, increment further if necessary).
#     Construct new tag using prefix, version, and N.
#
# The algorithm above is designed to tag LSST repositories with dotted-quad tags, while
# external packages with LSST patches will be tagged as <their_version>-<lsst_version>.
#
bt tag --increment

#
# Usage by the CI system
#

# Automated daily & triggered builds
bt pull && bt build                                                                    # build
master
bt pull --ref tickets/DM-1234 && bt build                              # build tickets/DM-1234, reverting to
master if unavailable
bt pull --ref tickets/DM-1234 --ref next && bt build        # build tickets/DM-1234, reverting to next,
reverting to master

# ... or, with some syntactic sugar:
bt build --ref tickets/DM-1234 --ref next

# Automated builds of release branches
# These are different in that we wish them to be tagged before they're built
bt pull --ref release/8.1.0
bt tag --increment                                                      # the default increment prefixes will
be taken from the current branch name
bt push
bt build -a


# ... or, with some syntactic sugar:
bt build --ref release/8.1.0 --tag-increment
```