GitHub and Source File Organization

Goals

The immediate goal to is put source for ScienceTools_User (somewhat reduced, non-ROOT version of the complete ScienceTools now built at SLAC) in GitHub as well as continuing to maintain it in CVS. Developers would work from GitHub. Changes there would be automatically reflected to CVS, allowing Release Manager to continue to be triggered by new tags in CVS. Subsequently we might want to

- Keep the larger distribution of ScienceTools in GitHub as well
- Keep GlastRelease in GitHub
- Modify Release Manager to work off of GitHub (or use something else altogether for builds, testing and distribution)

The first hurdle is our relatively complex package organization. Even a two-level hierarchy (container + subpackages) requires some sort of glue to get the right results for operations like tagging and cloning; the three levels we now have make the job harder.

Background

Subpackage structure and sharing

For purposes of this discussion ScienceTools_User (ST_U), ScienceTools (ST) and GlastRelease (GR) will be known as *containers*; the taggable components will be called *subpackages*. Apart from the special subpackage **containerSettings** ST_U is a proper subset of ST. Certain subpackages of ST and ST_U also belong to GR. ST_U doesn't officially exist yet, but all the issues discussed would apply to ST_U as well as ST.

The current structure of these containers is not suitable for GitHub because certain files (SConstruct, allExternals.scons, ..) are shared by all containers via sym links into their root directories. These files need to be moved into a subpackage which can be treated like other subpackages. Such a scheme is under development with the new subpackage SConsShared.

Syncing repositories

~~Because of the container reorganization needed to use SConsShared I don't believe there is any hope of moving development for all containers to GitHub initially; it will have to be done piecemeal. Neither will modifying ReleaseManager (or substitute) to work from GitHub happen instantaneously. That means we will have to live with the "same" source in both GitHub and CVS. We need reliable syncing mechanisms for code and tags.~~

(Aug. 29, 2017) **UPDATE:** an informal poll reveals that developers are willing to tolerate an outage of weeks in order to get this done "all at once". This means we don't have to put effort into figuring out how to reliably keep git and CVS in sync.

Physical structure

Much of the organization and all the sharing is handled in the CVS repository by means of sym links. There are top-level directories ScienceTools-scons and GlastRelease-scons. There are also top-level directories for subpackages such as astro, Likelihood, etc. In almost all cases the subpackage directories are sym-linked into the container directory. Shared subpackages like astro and celestialSources are sym-linked into both. The one exception is **containerSettings**. Each container has its own version of this subpackage, which is a regular subdirectory of the container directory. When you check a container out of CVS, sym links are followed, so you end up with all source files in regular subdirectories.

Subsubpackages

There are two subpackages (irfs, celestialSources) which have subsubpackages (that is, separately taggable physical subdirectories). celestialSources belongs to both ST and GR. SCons and Release Manager recognize subpackages and subsubpackages by the existence of a file called SConscript. In addition to specifying to SCons how to build the package, this file also contains version information.

Handling subpackages and subsubpackages

The single fact causing us the most grief may be this: CVS tags apply to a directory (optionally recursively); git supports tags only of entire repositories. Hence we surely need separate repositories for subpackages and a way to reassemble the hierarchy when they're checked out. It's less clear what to do about subsubpackages. There are only two subpackages with subsubpackages: celestialSources and irfs. Our custom subpackage-tagging software goes to extra trouble to avoid tagging subsubpackages when the parent subpackage is tagged.

Strategy summary

- Put subsubpackages in their own repositories in Git; use some existing tool, like subtrees or subrepos, to keep track of the "real" structure
- Put subsubpackages in their own repositories in Git; use homegrown scripting to bridge the gap between git structure and CVS structure
- Eliminate subsubpackages by changing organization in CVS and updating references as needed (but we still need an existing or custom homegrown tool which handles the two levels)

Git or third-party "Sub" tools

- Git submodule doc
- Git subtree doc (May, 2015) more doc by the same guy (May, 2013; updated Jan. 2017)
- Git subrepo doc, source

From the documentation for subtree and subrepo it seems likely that git submodule, the original and fully-integrated-into-git attempt to solve this kind of problem, is *not* suitable for our case (three levels of repositories, none of them external and all potentially undergoing active development). The other two, at least, deserve a closer look, but the discussions in the documentation cited above are sobering. Before trying to decide if any of these are suitable for us we should think hard about how we plan to do development.

1. Using forks?
2. Branches?
3. Or just commit to master, identifying known good commits with tags and avoiding conflicts among developers with informal communication?

In CVS we've reserved branches for large-scale differences, usually only suitable for GR and have mostly followed model 3), but branches are easier to use in git. 1. seems like overkill for the small number of developers involved.

Both subtree and subrepo appear to handle branches reasonably well. In the more recent subtree doc the description of how to push changes to the subtree back to the remote is a bit daunting, especially since this is likely to be a common operation for developers. subrepo doesn't have this issue, in fact fixing it was one - perhaps the chief - motivation for writing it.

I don't see any discussion of tags in the documentation of any of these tools but, since we've traditionally used home-grown tools rather than naked CVS commands for tagging already (C++ program **stag** for individual subpackages; **tagCollector.py** for release and release candidate tags), perhaps this isn't so important.

My tentative conclusion is that ***we can eliminate submodule and subtree from consideration; subrepo should be investigated further***. In particular we should see how it does with a three-level hierarchy.

See also this discussion of reorganization and beginning experiments with subrepro.

Scripting bridge

Each subpackage or subsubpackage (that is, anything with an SConscript in its top directory) goes in a separate repository, as with previous strategy, but we write our own custom tool to help with the scenarios we expect to encounter, and not much more.

Flatten hierarchy

Having a third level to our hierarchy introduces significant complication. Since there are only two packages where this is an issue it's tempting to change the structure to make the subsubpackages full fledged subpackages. This would affect other subpackages somewhat, e.g. in #include references; we need to make a survey to find out just how much work this would entail. In the case of celestialSources, this would impact GR as well as ST and ST_U.

Typical operations

The optimal choice of file organization and tools depends on what people and automated systems for building, testing and creating release need to do. It should not be hard to take care of users, who only need to run the code, no matter what choices we make, so here "people" really means "developers".

Developers

- Clone container and all contents from git into working directory
- Start with distribution of tagged release (including source) in working directory; clone individuals subpackages to replace distributed version in working directory.
- Push changes to GitHub for purpose of backup, sharing with other developers (e.g. on a branch?)
- New code ready for production (e.g. push or merge into master branch)
- Tag a subpackage
- Tag container and all subpackages (and subsubpackages) with a release candidate tag (e.g. ScienceTools-HEAD-1-2345)
- Tag container and all subpackages (and subsubpackages) with a release tag (e.g. ScienceTools-11-4-5)

Automated

As long as all code changes and tags are synced back to CVS the Release Manager can continue to handle all functions just as it has been, but it's worthwhile thinking about the operations it now does which would be impacted if it had to interface to a git repository instead.

- Mechanism to determine contents of a container.
  - For release and release candidate builds the file packageList.txt should do. Could perhaps make it serve for LATEST as well with some minor changes to the file or reinterpretation.
- Clone container and all contents from git repository into working directory
- Detect creation of new subpackage (or subsubpackage) tag, which acts as trigger for LATEST
- Detect creation of new release candidate or release tag