Modernising ATLAS Software Build Infrastructure

E Ritsch[1], G Gaycken[2], C Gumpert[1], A Krasznahorkay[1], W Lampl[3], E Moyse[4], E Obreshkov[5], K Potamianos[6], G A Stewart[7], A Undrus[8], F Winklmeier[9] *on behalf of the ATLAS Collaboration*

[1] European Laboratory for Particle Physics, CERN, [2] University of Bonn, [3] University of Arizona, [4] University of Massachusetts, Amherst, [5] The University of Texas at Arlington, [6] DESY, Hamburg and Zeuthen, [7] University of Glasgow, SUPA - School of Physics and Astronomy, [8] Brookhaven National Laboratory (BNL), [9] University of Oregon

E-mail: elmar.ritsch@cern.ch

Abstract. In the last year ATLAS has radically updated its software development infrastructure hugely reducing the complexity of building releases and greatly improving build speed, flexibility and code testing. The first step in this transition was the adoption of CMake as the software build system over the older CMT. This required the development of an automated translation from the old system to the new, followed by extensive testing and improvements. This resulted in a far more standard build process that was married to the method of building ATLAS software as a series of 12 separate projects from Subversion.

We then proceeded with a migration of the code base from Subversion to Git. As the Subversion repository had been structured to manage each package more or less independently there was no simple mapping that could be used to manage the migration into Git. Instead a specialist set of scripts that captured the software changes across official software releases was developed. With some clean up of the repository and the policy of only migrating packages in production releases, we managed to reduce the repository size from 62 GiB to 220 MiB.

After moving to Git we took the opportunity to introduce continuous integration so that now each code change from developers is built and tested before being approved.

With both CMake and Git in place we also dramatically simplified the build management of ATLAS software. Many heavyweight homegrown tools were dropped and the build procedure was reduced to a single bootstrap of some external packages, followed by a full build of the rest of the stack. This has reduced the time for a build by a factor of 2. It is now easy to build ATLAS software, freeing developers to test compile intrusive changes or new platform ports with ease. We have also developed a system to build lightweight ATLAS releases, for simulation, analysis or physics derivations which can be built from the same branch.

1. Introduction

In the first quarter of 2017 the ATLAS [1] collaboration completed a year-long process of modernising the build infrastructure of the Athena software stack. Athena is the workhorse of ATLAS' extensive software infrastructure. Athena production releases are deployed to the LHC Computing Grid [2] for Monte Carlo event generation, detector simulation and digitization, event reconstruction as well as generating derived data formats for physics analysis.

For this modernisation, new modern development workflows were established and custom made tools were replaced by community developed third-party tools. Approximately 4 million lines [3] of source code were migrated from the Subversion [4] version control system to Git [5].

Continuous integration tests and code reviews were established, which are both an essential component of the new workflow.

2. Structure of the ATLAS Athena repository

The ATLAS Athena repository is divided into individual packages and packages are structured in directories (see Figure 1). Each package contains a set of unique source code files, which typically get built into one dynamically linked library per package. There are, however, some packages that are statically linked against other packages. Packages may depend on each other at build- and at run-time. This implies that the packages must be built in a certain order so that all build-time dependencies are available upon building each individual package. As of summer 2017, the ATLAS Athena repository contains more than 2000 packages.

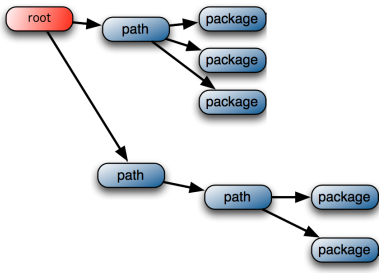

Figure 1. Structure of the ATLAS Athena repository. The repository is divided into packages. Packages are structured into (sub)directories.

3. Software development workflow

In any software project, the consistent application of a well structured development workflow plays a crucial role in the success of the project.

3.1. Past workflow

For many years, the ATLAS experiment applied a custom development workflow, which consisted of the following basic steps:

Code change: A developer changes an aspect of the software on their private working directory.

Tag package: The developer commits their private changes to the central `atlasoff` Subversion repository and creates a new version (called 'tag') of the package they modified.

Tag collection: The new package tag is added to all relevant nightly builds. This is done through a custom tool named the ATLAS Tag Collector [6].

Nightly builds and tests: On a daily basis all package tags that are collected for each respective nightly are built. After the completion of the build process (which takes several hours), a series of tests are executed. These tests include typical unit tests, integration tests, sanity checks as well as regression tests.

In the years 2015/2016 an internal report reviewing the ATLAS software infrastructure at the time identified the following problems, based on the experience of having applied the above workflow for a number of years:

- A number of internal tools were developed over time to facilitate the workflow applied by the ATLAS collaboration. Due to their internal nature, maintenance of these tools needed to be done by the collaboration itself. Moreover, developers joining the collaboration needed to first learn how to use these tools efficiently – and once a developer left the collaboration, this knowledge was no longer relevant to them even within the field of high energy physics. Some of these tools would become obsolete if a more modern and standard workflow were adopted.

- Dependencies of one package on other packages are documented in the source code – as it is required to successfully build each package. However, dependencies of packages on particular *versions* of other packages are *not* necessarily documented since the collaboration chose to not use this feature of the build configuration tool. This information may be documented elsewhere (for example on wiki pages) or, more commonly, is known only to the developer at the time the change was made. This often leads to manual trial and error approaches to identify compatible versions of dependencies of a given package when doing local development or updating nightly builds.
- Updates that require changes to multiple packages are tedious. It requires that all changed packages are tagged (i.e. versioned) separately, but the resulting package tags then need to be collected as one 'bundle' into the nightly builds. This 'bundle creation' is done manually by the developer when they request their package tags to be included in the next nightly build. The information about this bundle is not recorded in the source repository itself, rather in logs and emails that the ATLAS Tag Collector emits.
- Managing one access control list per package creates administrative overhead. Until the year 2012, individual developers were added manually (by the ATLAS Software librarian) to the access control list of a package if they requested write access to it. Later, ATLAS developed a custom tool (ATLAS SVN Access Manager) which allowed all developers with write access to a given package to grant write access to other developers. Effectively, developers still had to request write access to every package the first time they needed to modify it.

3.2. New workflow – ATLAS Flow

Among the most crucial changes in the recently updated ATLAS software infrastructure was the adoption of a new development workflow. The new workflow is heavily based on the GitLab Flow [7], which is sometimes referred to as *ATLAS Flow* (see Figures 2 and 3).

The new workflow is made possible due to the migration of the ATLAS Athena codebase from Subversion to Git (see section 4). Its main features are:

Release branches: A new Git branch is opened in the main code repository with every new major ATLAS Athena release that is made.

Adding bugfixes and features: New features are introduced in the `master` branch and in release branches, bugfixes are typically made directly in a release branch. The 'upstream first' policy (which is recommended in GitLab Flow) is not enforced in the ATLAS Flow. The reason is that bugs are mostly reported for released versions of the code, and those bugs are often difficult to reproduce in a branch different from the respective release branch (for example due to significant changes in the code between release branches and the `master` branch). Therefore, to reduce the 'time to production', the ATLAS Flow allows for bugfixes to be made in release branches directly.

Automated 'sweeps': Changes made to any release branch are automatically `cherry-pick`ed into the `master` branch on a daily basis. This is to ensure that all bugfixes and features are also included in future releases. These sweeps are done via our own cutom made script which run regularly every night. If there are conflicts they need to be handled manually by the developer or release coordinator.

4. Version control – From Subversion to Git

The most crucial change done by the ATLAS collaboration in its effort to modernise the software infrastructure was to move the source control system from Subversion to Git. Subversion had been used by the ATLAS collaboration for about 9 years (since the year 2008). Due to the custom structure of the ATLAS Athena repository stored in Subversion (named `atlasoff`, see Figure

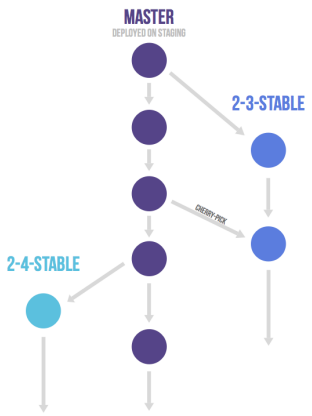

Figure 2. The original GitLab Flow [7] with release branches. Every time a new major release is made a corresponding Git branch is created. The 'upstream first' policy is recommended, which means that any updates (i.e. bugfixes) are done in the `master` branch first and then Git `cherry-pick`ed into relevant release branches if necessary.

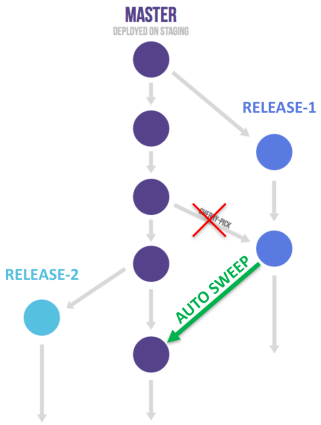

Figure 3. The new software development workflow employed by the ATLAS collaboration as of 2017. The workflow is derived from GitLab Flow. The difference is that ATLAS does not enforce the 'upstream first' policy, but allows for changes to be made directly in release branches. Automated daily 'sweeps' copy those changes into the `master` branch through Git cherry-picking.

4) a custom migration script was developed to copy relevant versions of individual packages from `atlasoff` into a new Git repository (`atlas/athena`). The `atlas/athena` Git repository is hosted on CERN's instance of GitLab Enterprise Edition [8].

Whereas packages were versioned independently in the `atlasoff` Subversion repository, in the `atlas/athena` Git repository the state of the entire set of packages is versioned at once. This means that a given version of the `atlas/athena` repository represents a given version of the entire ATLAS Athena stack. This eliminates the need for tools like the ATLAS Tag Collector that was used to define which package versions (tags) constitute a given ATLAS Athena build or release.

The following steps were carried out during the migration. Most of these steps were done in an automated fashion by the custom migration script, however, a small amount of manual intervention was required nevertheless:

- In order to facilitate debugging and 'code archaeology' tasks in the newly created `atlas/athena` Git repository, parts of the commit history were copied from the Subversion repository. The state of the code of any release from the last (approximately) 2 years was copied into respective *archive* branches in the Git repository.
- Files larger than 100 KiB were not copied from Subversion to Git. It was discovered that the majority of the files affected were plain text data files that had no good justification to reside inside the source code repository.
- A copyright statement was put at the head of each source file.

After applying the above steps, the initial `atlas/athena` Git repository had a size of approximately 220 MiB – the `atlasoff` Subversion repository had a size of approximately 62 GiB at the time of the migration. It is important to note that the size required can not be compared directly as only a partial commit history was migrated from Subversion to Git, and moreover, many obsolete packages and files were not migrated at all.

A triangular Git workflow [9] was established to keep the main Git repository (`atlas/athena`) clean and well structured (see Figure 5). Therefore, developers cannot directly modify this repository, but are required to create personal forks on CERN's GitLab instance which they can modify as required. Modifications that developers want to include in the official ATLAS Athena codebase need to be requested into `atlas/athena` via GitLab Merge Requests. The GitLab Merge Request interface is used to facilitate code reviews and to communicate automated build and test results. Every Merge Request that is created (whether automated or manually by a developer) triggers the following actions:

Multi-stage code review: ATLAS code reviewers are tasked to identify potential problems and weaknesses in the modifications before they are merged into the main Git repository. The code review is divided into three levels (thus 'multi-stage'): level 1, level 2 and expert level. Level 1 and 2 are handled by shifters. Level 1 reviewers inspect the code for logic errors, common programming errors, consistency with the rest of the codebase, conformity to style guides [15] and similar issues. Some code style guidelines are in process of being automated. Level 2 reviewers inspect the general structure of the code change and its integration into the ATLAS Athena software stack. If a modification is deemed critical and deep domain knowledge is required to assess its validity, a domain expert is included in the code review process. The time to review the code varies greatly depending on the number of changes done, their complexity, expereince of the developer and the experience of the reviewers.

Continuous Integration build and tests: To ensure that the code in the `atlas/athena` Git repository remains in a working and useable state at any time, any requested code change gets automatically compiled and a defined set of unit and integration tests are run on it. The continuous integration tool Jenkins [10] is used to schedule and distribute the build and test jobs on a set of dedicated build machines. As of summer 2017, 14 dedicated build machines form the backbone of the continuous integration system.

When a Merge Request passes the review stage as well as the corresponding build and tests, the release coordinator for the given target release branch is responsible for accepting the Merge Request at a suitable time.

5. Building binaries and releases – From CMT to CMake

The ATLAS collaboration moved from the Configuration Management Tool (CMT [11]) to CMake [12] to handle the setup of the software environment as well as the build process.

A converter was implemented to automatically translate CMT's `requirements` file format into CMake's `CMakeLists.txt` files for each individual package. In the majority of cases, the automatic conversion worked without issue. In a few cases, manual action was required to convert specific parts of the CMT syntax into the corresponding CMake syntax. The old CMT `requirements` files were removed from the code repository once the entire codebase could be built successfully with CMake.

Nightly builds of all release branches and the `master` branch are made on a daily basis. As of summer 2017, 7 different branches are included in the nightly builds. RPM [13] packages containing the build results are generated for every nightly build. They are installed on the CernVM File System [14] to make the build accessible globally to all developers. Builds are done in different 'flavors', where each flavor contains a different sub-set of all packages. Therefore they build faster and their release sizes are smaller compared to the full build. Currently, 8 different flavors of the Athena software are in use by the ATLAS collaboration. Among them are lightweight ATLAS releases for simulation, analysis or physics derivations.

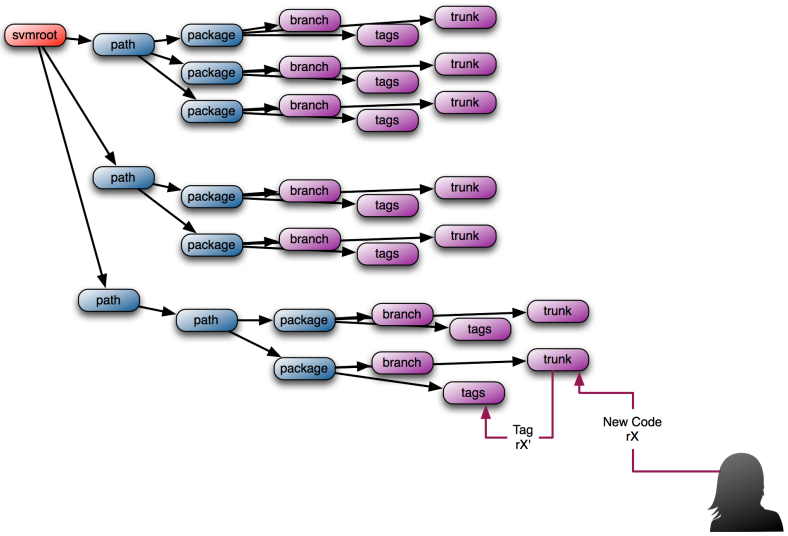

Figure 4. The structure of the ATLAS Athena repository as it is stored in Subversion (named `atlasoff`).

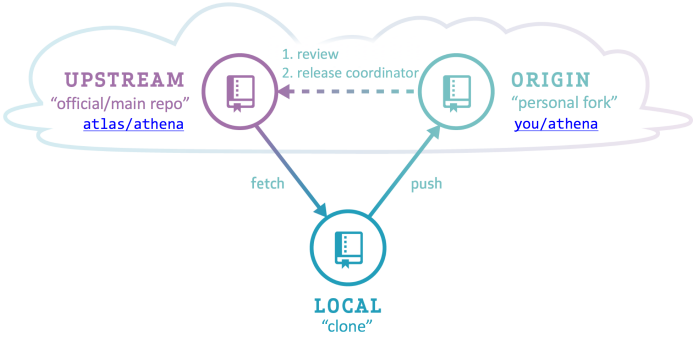

Figure 5. The ATLAS collaboration uses a triangular Git workflow to manage the interaction of developers with the Athena repository. (Image based on [9])

6. Tutorials and public documentation

In order to prepare the members of the ATLAS collaboration for the change of tools and workflow, a number of interactive tutorials were held before the changes were carried out.

A public website [16] was put in place to make the documentation on the new workflow and tools accessible and searchable for everyone. The website hosts step-by-step walkthroughs to help developers getting started with the new ATLAS software development environment.

7. Conclusions

After years of experience using custom software tools and workflows it became clear that these solutions were less than optimal when compared to well-established third-party tools and workflows. This not only needed the custom tools to be maintained by the ATLAS collaboration, but the custom workflows necessitated a learning phase for developers joining the collaboration – even for highly experienced HEP developers.

Moving the software infrastructure to well-established workflows and third-party tools alleviates much of the overhead that the custom infrastructure caused to the ATLAS

collaboration in the past. In particular the move to Git – as the de facto industry standard for source control management – was well received by the ATLAS developer community. Continuous integration builds and tests speed up development significantly as a faster feedback to the developer is provided. Code reviews improve the quality and consistency of the code, and they aid in spreading the knowledge of the codebase among the entire ATLAS developer community.

Though most of changes were disruptive in nature and the collaboration had to adopt the new workflows quickly, the current infrastructure is clearly an advantage for the ATLAS collaboration.

References

[1] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3** (2008) S08003.

[2] WLCG Collaboration, *WLCG - Worldwide LHC Computing Grid*, http://wlcg-public.web.cern.ch/.

[3] G. Stewart, W. Lampl, The ATLAS Collaboration, *How to review 4 million lines of ATLAS code*, ATL-SOFT-PROC-2017-047, 2017, https://cds.cern.ch/record/2248494.

[4] The APACHE Software Foundation, *Subversion*, https://subversion.apache.org/.

[5] Git Project, *Git*, https://git-scm.com/.

[6] S. Albrand, J. Fulachier, J. Collot, F. Lambert, *The TAG Collector: A Tool for ATLAS Code Release Management*, 2005, https://cds.cern.ch/record/865644.

[7] S. Sijbrandij, GitLab Inc., *GitLab Flow*, 2014, https://about.gitlab.com/2014/09/29/gitlab-flow/.

[8] GitLab Inc., *GitLab*, https://about.gitlab.com/.

[9] M. Haggerty, GitHub Inc., *Git 2.5 including multiple worktrees and triangular workflows*, 2015, https://github.com/blog/2042-git-2-5-including-multiple-worktrees-and-triangular-workflows.

[10] Jenkins project, *Jenkins*, https://jenkins.io/.

[11] C. Arnault, G. Rybkin, *CMT*, http://www.cmtsite.net/.

[12] Kitware Inc., *CMake*, https://cmake.org/.

[13] Red Hat Inc. and Community, *RPM Package Manager*, http://rpm.org/.

[14] CERN, *CernVM File System (CernVM-FS)*, https://cernvm.cern.ch/.

[15] The ATLAS Collaboration, *ATLAS C++ coding guidelines, version 0.6* , http://atlas-computing.web.cern.ch/atlas-computing/projects/qa/draft_guidelines.html.

[16] The ATLAS Collaboration, *ATLAS Software Documentation*, https://atlassoftwaredocs.web.cern.ch/.