'Git'dy Up, Developers!

If you're not familiar with free, open source versioning tools, you might think we grossly overlooked a misspelling in the title and used poor grammar to boot! Now that we have your attention, read on.

Recently, BASIS incorporated Git distributed revision control and source code management (git-scm.com) into the AddonSoftware® by Barista® (Addon) upgrade procedure to make upgrading customized Addon packages easier than ever before. In this article, we examine the Addon upgrade process prior to the introduction of Git, what Git is, how it adds new intelligence to the Addon upgrade process, and Git's overall role in an Addon upgrade.

By Shaun Haney
Quality Assurance Engineer

By Chris Hawkins
Software Developer

Upgrading Customized AddonSoftware Installations

One of the advantages of developing Addon within the Barista Application Framework is Barista's application project structure. Developers wanting to make customizations to Addon can create a new Barista project and save customized forms, callpoints, reports, publics, custom classes, etc., inside that project file structure. This separation ensures that it preserves customizations when upgrading Addon. After installing a new version of Addon, the Barista Install Application Wizard (IAW) allows developers to re-install these custom projects by importing customized forms and/or data tables back into the standard product.

While this is the recommended process for making customizations, it is possible that some developers have modified the standard product directly or have blended some modifications in a separate project file structure with others made directly to the core product. In addition, the emphasis of the Barista IAW is on forms and/or data tables. The wizard doesn't reconcile changes in callpoint code or other code not related to the user interface.

Regardless of how customizations were made, the result is that Addon's modifications suit customers' needs. In many environments, such customizations leave the product "stuck in time," with no possibility of an application upgrade. With Barista and Git, developers can now modify both the user interface as well as other code, and still realize the benefits of an upgrade!

Life Before Git

Prior to Git, developers could use the Barista IAW to incorporate form or table modifications back into standard Addon. However, there was no automated way to preserve those customizations if they were made directly against the Addon installation (i.e., in the Addon file structure itself). In addition, the wizard is not concerned with changes in other source code files such as callpoints, reports, publics, etc., so there was no easy way to see changes in the standard code and determine if they should be included in the customized counterparts. In terms of callpoints, additional code that developers may have added to run before or after the standard

callpoint may be fine, but if the developer took a copy from a standard callpoint and augmented it to run instead of the standard, there was no easy way to see if subsequent changes to the standard should be incorporated into their "instead-of" callpoints. You can imagine that analyzing source code could be a tedious process; VARs either had to perform three-way comparisons by hand – comparing the original Addon to the customized version and then to the new Addon – or write their own in-house scripts to upgrade their code.

In the pre-Git life, there was no single, comprehensive process for upgrading Addon. Each process was specific to the company requiring the upgrade. Furthermore, manual upgrades delivered incomplete or "hybrid" versions of Addon where not all of the customer's Addon source is upgraded, resulting in files from different versions of Addon coexisting in the same installation. Last but not least, when using in-house procedures for upgrades, there often was no mechanism for remembering or recording decisions made in past upgrades. In these cases, the same questions, like whether to replace a piece of code with a newer version, typically resurfaced repeatedly with each upgrade.

What is Git?

To address the dilemma of having highly customized code in an old version of Addon, BASIS incorporated Git into Addon's upgrade process. But how exactly would Git solve this?

Git is a distributed source management control system. Linus Torvalds and other open source developers originally designed Git as a replacement for BitKeeper, previously the versioning system of choice for maintaining the Linux kernel. Much like CVS and Subversion, Git tracks file changes so users can access past versions of their files. Unlike Subversion and CVS, Git is a distributed revision system in which the user actually obtains a copy of the entire archive and works with their copy locally, rather than only checking out a single revision of the files from a central server. Having the entire archive instead of a single revision actually allows the user to keep their own custom version of the archive, while still allowing that user to get updates from the original archive or even other customized copies of the original archive.

So, how exactly does this work? Let's first look at a generic non-Addon scenario for using Git. Let's say your favorite free text editor is available from a public Git archive and you decide that your company needs some specific enhancements for highlighting part numbers with a color keyed to the department that produces that item. Since this feature is unique to your company only and the editor does not provide plug-in support, you decide to "clone" the archive and add the needed functionality into your own version of the editor. So you use the **`git clone`** command to copy the Git archive to your system. Then you checkout the latest version of the source code that actually appears in your Git directory so you can make changes in place. You add the features you want and then check them in. A new checkin appears in your archive that is not present in the remote archive.

A few months pass, and you read that they've enhanced the way that the text editor finds and highlights text. You want this enhancement but are concerned that some of their changes will overlap yours. This time, you pull the new version from the remote archive and Git attempts to merge its changes on top of yours automatically. Wherever you have changed code but the remote archive hasn't, your change is preserved. Wherever the remote archive has changed code, but yours has not, the remote archive change is automatically applied. However, if there are changes on the same line in both archives, Git denotes the conflict and will not complete the merge as a brand new revision until you resolve the conflict and commit your change. The total number of conflicts is typically only a small fraction of the total number of merges Git performs.

Git's automatic merge process allows you to focus only where it actually needs your intervention – on pieces of code that you changed and have also changed in the original source. Once you resolve the conflicts and commit the files, Git creates a new revision of the code that incorporates both your customizations and the original source. In this way, Git allows you to maintain custom code while still being able to incorporate upgraded code from the original source.

Git Brings New Intelligence to the Upgrade Process

With its ability to perform automatic merges, detect conflicts, and remember custom changes, Git speeds up the Addon upgrade process and ensures a complete upgrade. The basic steps in the Git cycle are listed below and illustrated in **Figure 1**, in which Addon revisions in Git appear as layers of an onion. Specific processing steps depend on whether customizations were made directly into the Addon source code or saved in a separate Barista project file system.

1. Clone the Git archive directly from BASIS. The Git archive contains every version of Addon.
2. Roll the archive back to the same version as the customer's current version of Addon.
3. Copy the customized Addon code into the archive.
   - If developers made customer customizations directly to the Addon file structure, then they could copy that core code into the archive as a new revision.
   - If developers maintain the code in a separate Barista project, BASIS has created a set of utilities that copy the code modifications into the archive. Developers can then check these modifications in to recreate the new revision.
4. After checking in the new revision, the developer rolls the archive forward to incorporate any changes made since the customer's current version all the way to the new version. Most of the changes will merge in smoothly, but a handful of changes will likely result in conflicts.
5. Review and correct the conflicts, and commit each resolved file. Once the files are committed, Git has a brand new revision containing the customized code and upgraded code that comes with the latest version of Addon.
6. Move the upgraded code back from the Addon archive to the customer's copy of Addon.
   - If developers made customer customizations directly to the Addon file structure, they can copy the upgraded code directly to the Addon core.
   - If developers maintain customer code in a separate Barista project, BASIS created a set of utilities to move the upgraded code back to the Barista project.

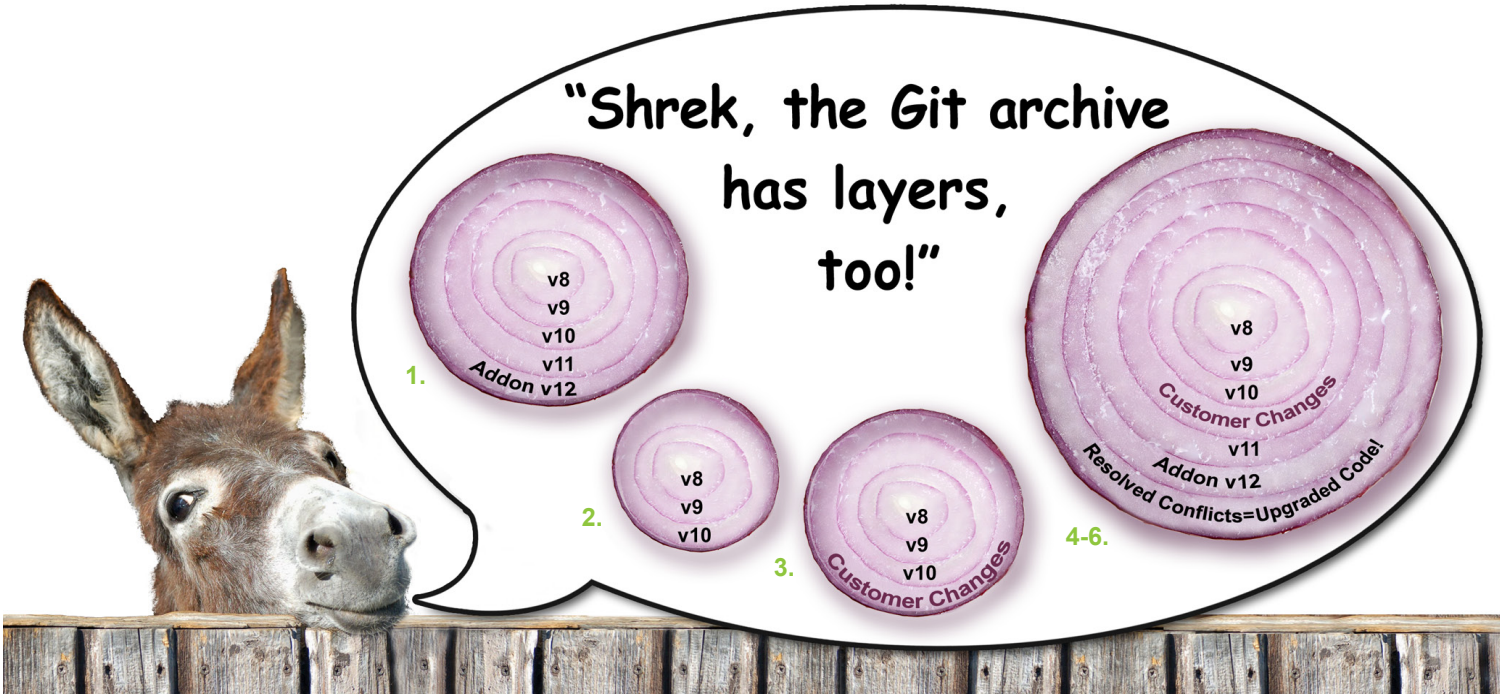

Figure 1. The basic steps in the Git cycle

At this point, keep and use the Git archive for the next Addon upgrade so that conflicts that occurred this time around will not need revisiting in the future.

'Git' the Big Picture

While Git plays a critical role in Addon's upgrade process, it is just one step in the whole process. Now that you're familiar with Git, let's review the entire process.

Download and Install AddonSoftware by Barista

The initial installation places Addon in the same directory structure as BBj®. BASIS intended this copy for demo or evaluation purposes and does not recommend it for a production installation. Once installed, use the AddonSoftware Install/Upgrade Wizard (AIUW) to make a new instance of Addon outside of the BBj home directory. This is the live/production version of Addon.

As mentioned, the recommended process for customizations is to use the Barista Create Application task to set up a separate Barista project structure for your modifications. You will save or create some files directly in the new project, and others will be saved there automatically by Barista when using application replication mode from the Barista Form Manager (see *Customizing Barista Applications* for more).

Upgrade AddonSoftware

When you're ready to upgrade to a new version of Addon, download and install the new Addon into BBj home (overlaying the current demo or eval copy). Then use the AIUW to perform the upgrade process. Note that the AIUW facilitates parallel operations so users can continue to run the live production copy of Addon during the upgrade process. Read more about the AIUW in *AddonSoftware Installation and Upgrade Processes*, but in short, this wizard chains together the following tasks:

- Make a new copy of Barista and Addon in a user-specified location
- Copy backed-up administrative and syn file data from the live production installation
- Run the Auto-synchronize process
- Copy and/or install other Barista projects (customization projects or verticals)

Copy, Upgrade, Install the Project

In order to upgrade the customization project via Git, opt to copy, not install, that project. Once the project is copied, you'll use the Git process to upgrade the project, and then use the Barista IAW to install it into the new Addon.

If you are not using a separate Barista project for customizations, then after using the AIUW to create your new instance of Addon, perform the Git processes and then copy the desired files directly into the new copy of Addon.

Conclusion

While customer installations vary and Git may not be the silver bullet that slays all villains, the incorporation of Git into the Addon upgrade process is a huge step towards standardizing Addon's upgrade process and greatly simplifying the arduous task of finding and incorporating Addon's upgraded code into a custom installation. Git also makes future upgrades even easier than the initial upgrade by ensuring a complete upgrade and remembering the decisions made in previous upgrades. For those developers who have put off an Addon upgrade due to the difficulties involved, Git is race horse that will "get" you across the finish line.

Hop on today and git-dy up! ■

For more information, read
- *Create Vertical and Customize Applications, Parts 1, 2, 3*
  links.basis.com/baristaref
- *AddonSoftware Installation and Upgrade Processes*
  links.basis.com/addoninstallupgr