Using the BASIS Git Archive

* * * * *

Getting Started

Register for Access to the AddonSoftware Git Archive

- To get a user account for the addon archive (ssh://git@git.basis.com/addon), generate an OpenSSH key pair.

- Name the public key <your username>.pub

- Verify the name at the end of the public key is your username.

  For example, if John Doe sent me a file named jdoe.pub, the contents of the public key would look like this:

```
ssh-rsa XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX jdoe
```

- Send the public key to shaney@basis.com.

Install Git

- If you don't already have Git installed, follow the instructions below.

  On Linux
  - At the shell prompt as root, type:
    `yum install git`

  On Windows
  - Browse to http://code.google.com/p/msysgit/downloads/list. At this time, it seems any current version of Git is a "preview."
  - Obtain the Git at the top of the list, download, and follow the installer instructions.
  - You can also install TortoiseGit (code.google.com/p/**tortoisegit)** along with msysgit to provide a graphical interface via Windows Explorer

  Note: msysgit is a commandline version of Git for Windows in the form of a convenient git bash shell. Under Windows, anytime in these instructions that you would type a command at the command prompt, use the Git Bash shell.

Download the Git Archive

If you are upgrading from a version prior to 12.20, there are two archives you'll want to download for Addon. The first, called "**addon,**" is a subset of the complete Addon archive, containing the code files that Git will analyze and merge during an upgrade (backend programs in the various "src" folders, and callpoint code from the "cdf" folder). The other archive is called "**side_addon**" and contains all Addon files, including images, data/bar/*.xml files, etc.

If upgrading version 12.20 or greater, the "**addon**" archive is all you need because it contains all files for the later versions.

The remainder of the document refers to the "**addon**" archive. Substitute "**side_addon**" if appropriate for your upgrade.

- cd ..At a bash (or other UNIX shell) prompt,
  - Switch to the directory you want to be the parent of the Git archive's directory.
  - Type `git clone ssh://git@git.basis.com/addon` and press [Enter] to download the desired Git archive to your machine.
- Using TortoiseGit
  - Launch Windows Explorer and navigate to the folder that you want to be the parent of the Git archive's directory.
  - Right-click and select `Git Clone...`
  - Enter the URL for the BASIS Git repository (ssh://git@git.basis.com/**addon**).
  - Browse to and select the private key (.ppk) from the key pair you created earlier, then click OK.

Set Global Git Properties

Git global configuration properties determine how Git will behave when performing its various operations. You will want to configure certain properties before using Git.

The first two fields identify you and will help you determine which changes you have made. Enter *your information* with the following syntax:

```
user.name=John.Doe
user.email=jdoe@anycompany.com
```

These fields tell Git how hard to work looking for renames and copies:

```
diff.renames=copies
```

your changes, the original file, and the changes in the version to which you are trying to merge. This property is especially helpful when resolving merge conflicts. For example,

```
merge.conflictstyle=diff3
diff.renamelimit=999999
merge.renamelimit=999999
```

This property, merge.conflictstyle, tells Git to show conflicts and how to format the list containing
To set these properties, use:
git config --global <property name> <value>

ie.,

```
git config --global user.name John.Doe
git config --global user.email jdoe@anycompany.com
git config --global diff.renames copies
git config --global diff.renamelimit 999999
git config --global merge.renamelimit 999999
git config --global merge.conflictstyle diff3
```

Using Git

Modifications to Addon forms and callpoints should be done by creating a separate Barista project (a "mods" project), then using application replication mode during development to save your work into the mods project structure. Modified back-end code (reports, updates, etc.) should be saved in the mods project as well, typically in the "prog" directory. This development strategy keeps your modifications separate from the core product so you don't risk losing them when you move to a different version of Addon (see Customizing Barista Applications for more information).

Prior to an upgrade, you should always begin by checking out the **addon** archive to a work directory so Git can show you any differences between standard Addon and your running copy.

Analyzing modifications to Addon core

Begin by resetting the Git **addon** archive to the version of Addon you're currently running.

Reset Master Using Git Bash

1. At the command line, `cd` to the directory to which you cloned the **addon** Git archive.

2. Locate the version tag in which you made your code modifications with the following:
   `git log --decorate`

   You will see lines such as the following:

```
commit cc0591f9cac3b68f54ef305cfc91a506938622ed (tag: BBJ1003GA)
Author: Shaun.Haney <shaney@basis.com>
Date:   Tue May 1 13:39:04 2012 -0600

    Revision 10.03
```

   The piece of information you're looking for is the tag, e.g. (`tag: BBJ1003GA`).

3. Reset the archive back to the desired Addon version by supplying the tag info:
   `git reset --hard BBJ1003GA`

Reset Master Using TortoiseGit

1. Launch Windows Explorer and navigate to the local copy of the archive
2. Right-click on the **addon** folder in the archive, and select `Git Show log...`
3. You will see all of the branches/tags in the show log window. Right click on the desired version and select `Reset "Master" to this...`, then select the Hard reset option.

Apply and Analyze Differences

Once the archive is set to the correct version, copy/paste your Addon files over the **addon** archive's files. Then use the Git "Add all files now" command to add the files to the archive so Git can detect differences. With TortoiseGit, any files that are different from the repository version of Addon will show with a red exclamation point icon. You can right-click and select `Git Diff...` on any of these to launch a graphical diff utility. Alternatively, you can right-click and use the Git History option to see a list of all changed files and the differences in each.

If you're using Git Bash, type `git diff --name-only` to see a list of file names, or just `git diff` to see a detailed textual diff of each file.

Depending on your current version of Barista Addon, differences that Git discovers in the data/bar/ directory may be a result of developing while in application replication mode (version 11.10 and prior). If that is the case, then the corresponding file should also be present in your mods project's data/bar/ directory, and should be virtually identical (date/time stamps may differ) from the one you've pasted into the Git archive. If you're upgrading from a newer version, then review those files to determine if they are legitimate differences that should be harvested off into the mods project. For example, you may have a mods project, but at some point inadvertently made a form and/or callpoint change in the Addon project itself. In this case, you'll see a difference in the form resource file (data/bar/ directory), callpoint definition file (data/cdf/ directory), or both. Prior to the upgrade, you may want to re-do this mod with application replication mode set, so the change is saved into your mods project.

You may have added other files, such as images, to the core project. You should be aware of any such changes so you can restore thesee files after the upgrade, or consider moving them into your mods project and changing globals or pathnames to locate the file in that location.

In general, back-end code (reports, publics, etc.) that has been modified in standard Addon can simply be copy/pasted into the prog folder of your mods project (you may need to change global pathname or prefix settings to locate the file once moved).

Once all of the core modifications are successfully saved in your mods project, you can safely upgrade to a new version of Addon without the risk of losing changes inadvertently made against the standard product. See the Upgrading Mods with Git document to learn how Git can help you upgrade your mods project.

Upgrading when core Addon backend code has been modified

If you've made backend code modifications directly in Addon rather than creating a separate Barista project, you may wish to upgrade this version of Addon *without* saving the modifications in a separate project file structure. Use the instructions below to upgrade using Git. *IMPORTANT NOTE: Do not use this method if you have made form and/or callpoint modifications directly in core, as Git should not be used to merge the data/bar/*.xml files. Form and/or callpoint mods must be done using application replication mode to a separate Barista project.*

Reset Using Git Bash

1. At the command line, `cd` to the directory to which you cloned the **addon** Git archive.
2. Locate the version tag in which you made your code modifications with the following:

   ```
   git log --decorate
   ```

   You will see lines such as the following:

   ```
   commit cc0591f9cac3b68f54ef305cfc91a506938622ed (tag: BBJ1003GA)
   Author: Shaun.Haney <shaney@basis.com>
   Date:   Tue May 1 13:39:04 2012 -0600

       Revision 10.03
   ```

   The piece of information you're looking for is the tag, e.g. `(tag: BBJ1003GA)`.

4. Reset the archive back to the Addon version against which you made your modifications using the desired tag:

   ```
   git reset --hard BBJ1003GA
   ```

   Note: Skip this step for subsequent upgrades to continue where you left off from your previous upgrade.

Reset Using TortoiseGit

1. Launch Windows Explorer and navigate to the local copy of the archive
2. Right-click on the **addon** folder in the archive, and select `Git Show log...`
3. You will see all of the branches/tags in the show log window. Right click on the desired version and select `Reset "Master" to this...`, then select the Hard reset option.

Apply and Commit Changes

1. Apply your changes by copying your files over the archive's files. If you've renamed any of the files, copy your file into the archive and delete the original file.
2. For any files not in the original archive or renamed, use `git add <filename>` to add each file to the archive (with TortoiseGit, right-click and select `TortoiseGit => Add`). Git does not actively track the renamed files, but can determine renamed files during merges or when listing a detailed log.
3. Commit your changes in Git Bash by typing `git commit.` In TortoiseGit, right-click on the addon folder and select `Git Commit...`
   Be sure to add a comment indicating that these are your changes.

Merge Changes into new Addon

Your modified Addon code is now committed to your local repository, and you are ready to have Git take you to the next level: merging your modifications into the latest version of Addon.

Using Git Bash

1. Use `git pull ssh://git@git.basis.com/`**`addon`** to merge your changes to the latest version.
2. Use `git status` to list which files have conflicts and then resolve them accordingly. See Resolving Conflicts.
3. As you modify each file with a conflict, use `git add <filename>` to stage the file for committing.
4. Once all conflicts are resolved, enter `git commit` to commit all files to the archive.
5. Check your current status with `git status`. No files should be remaining in the working directory that are untracked or waiting to be committed. You can also see the log for your changes by typing `git log --decorate --topo-order --graph.`
6. Keep your work for the next upgrade.

Using TortoiseGit

1. Right-click on the `addon` directory in your local repository and select `TortoiseGit => Pull...`, then click OK.
2. Files that have conflicts appear with a yellow triangle icon. You can resolve them using either a graphical tool such as KDiff3, or by using Git Bash and typing `git status` (see Resolving Conflicts)

3. As you resolve each conflicted file, right-click and select `TortoiseGit => Add...` to stage the file for committing.
4. When all conflicts are resolved, right-click on the `addon` folder and select `Git Commit Tool` to review and commit all files to the archive. You should not see any files in "unstaged" status.
5. Keep your work for the next upgrade.

Resolving Conflicts with git status (textual diff)

Each file with conflicts will embed the conflict at the location where it occurs in the format shown below:

```
<<<<<<< HEAD
Changes from the mod directory.
||||||| merged common ancestors
The original file before the merge took place.
=======
The latest Addon changes
>>>>>>> b72ea1e9123a697ace3b91a2c09e592076129d0c
```

The portions above highlighted in green are the standard lines that Git uses to show a conflict. The last line shows the commit id is actually involved in the conflict.

The best way to demonstrate how to solve a conflict is to show an example. Suppose the file `ivr_stkmovemnt.aon` contains the following conflict:

```
<<<<<<< HEAD
rem --- Copyright (c) 1981-2007 AddonSoftware
||||||| merged common ancestors
rem --- Copyrights 1981-2011, BASIS International Ltd. All Rights
Reserved.
=======
rem --- Copyright BASIS International Ltd. All Rights Reserved.
>>>>>>> b72ea1e9123a697ace3b91a2c09e592076129d0c
```

This conflict occurs on the copyright line, and shows that the latest copyright has been set to:

```
rem --- Copyright BASIS International Ltd. All Rights Reserved.
```

The way to resolve this conflict, although fairly trivial, is to just erase everything in the conflict except for the latest change. In this case, delete both the AddonSoftware copyright and the BASIS copyright that specifies years. Of course, also delete the auxiliary lines that Git has placed in the file to assist in locating and resolving the conflict.

Once a conflict is resolved in a file, add the file back and commit the changes with the following lines:

```
git add iv/src/ivr_stkmovemt.aon
```
