HITACHI
Inspire the Next

PDI Techniques – Working with Git and PDI Enterprise Repository

HITACHI
Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes

Contents

This page intentionally left blank.

Overview

This document covers some best practices on working with version control systems that are different than the default provided by the Pentaho Data Integration (PDI) repository. We also cover how to work with and interact between local Development revisions and the repository environments consumed by PDI.

This is not intended to dictate what the best options are, but rather to present some best practices for customers who are integrating the PDI repository with another version control system. Some of the topics covered here include repository structures, working with Git patterns, methodological guidelines, migration, and a sample.

Our intended audience is Pentaho server administrators, or anyone with a background in repositories who is interested in working with Git.

Software	Version(s)
Pentaho	6.1, 7.x

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

This document assumes that you have knowledge about Pentaho Servers and Github and that you have already installed and configured Pentaho.

Integrating the DI-Only Repository with Git

The DI-only repository offers a basic version control system which is not intended to replace most common market tools, such as Git. The repository is characterized by a variety of options and features that are very mature and normally present in most IT departments but offers a simpler solution. A version control system in your Development environment is recommended to:

- Provide a central location for policies and experience regarding Development systems, and
- Keep track of changes if you want to use some advanced features, such as fork, tag, etc.

Repository Structures

The QA/Production environment repository must meet the customer's requirements but must also meet certain security and permissions criteria.

Developer Permissions

Developers should have `WRITE` access on the Dev server, but no access to the QA/Production environments, even if the same folder structure exists on both. Figure 1 shows an example of a Developer folder structure and permissions notes.

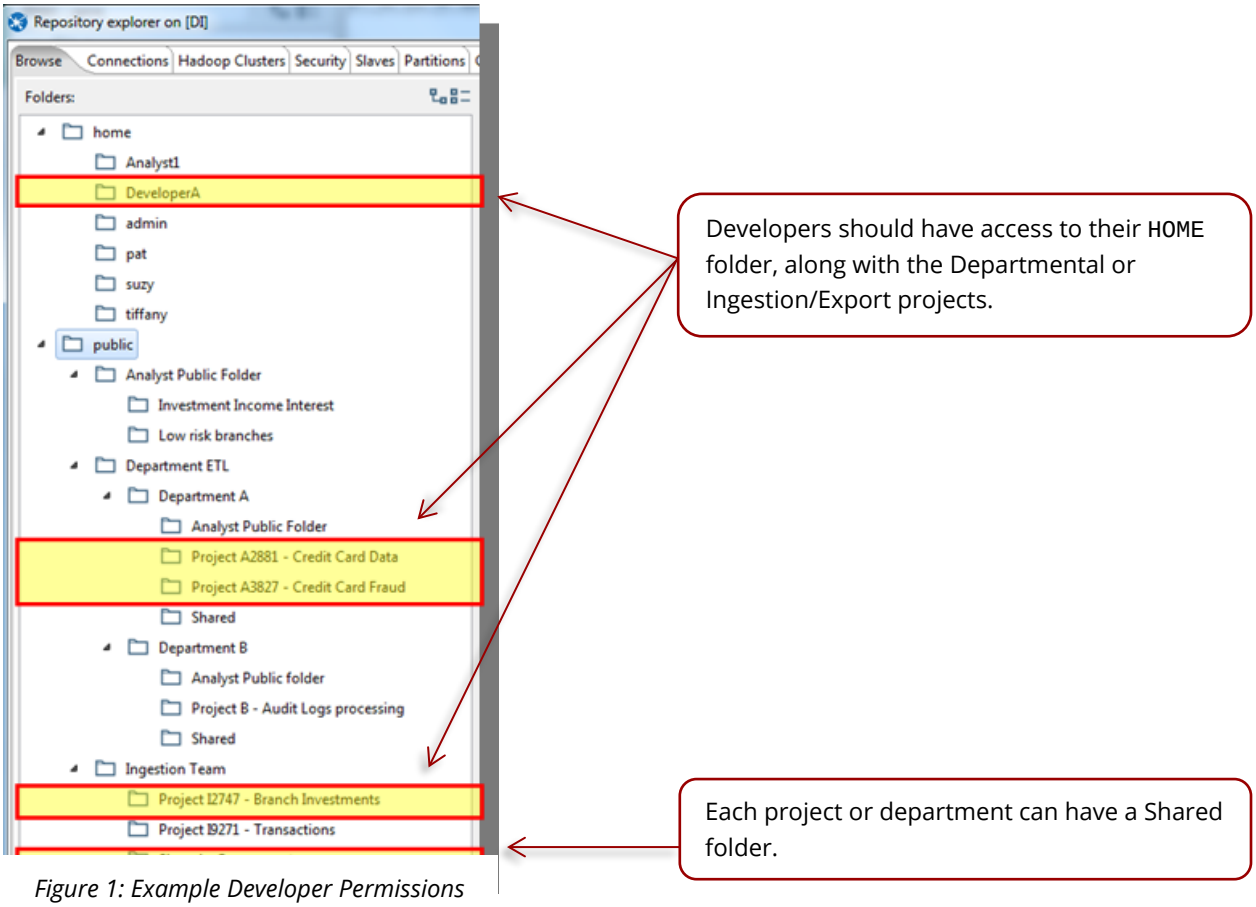

Figure 1: Example Developer Permissions

Analyst Permissions

Analysts can move extract, transform, and load (ETL) tasks from one folder to another if they have `READ/WRITE` permissions. Figure 2 shows an example of an Analyst folder structure and permissions notes.

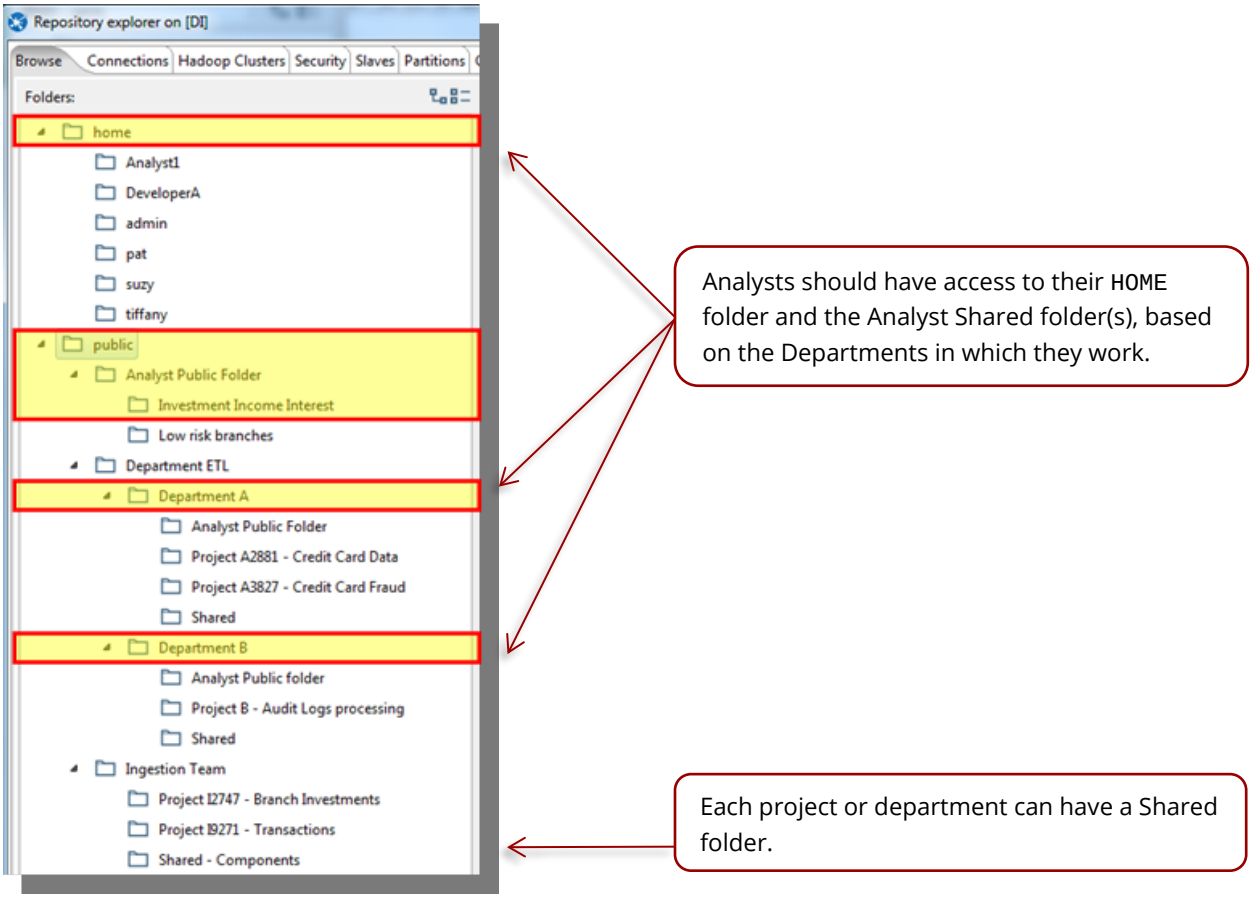

Figure 2: Example Analyst Permissions

Git Patterns, Methodologies, and Migration

These sections go over Git patterns, methodologies to be followed for each implementation, and some steps on migrating from a PDI enterprise repository to a Git repository.

Working with Git: Patterns

Enterprise customers have version control system policies and practices internally, and should consider the additional patterns:

We recommend that if you already have a Git process and policies set up, that should be used instead.

- Create Git accounts for all teams with folder restrictions (Security), following the same scheme used for the PDI repository tree structure.
- Teams work on local PCs, and **Commit to Git**, then try out the new solution in their testing environment.
- Once tested, the folder with the set of jobs/transformations/reports/misc. is uploaded using a controlled export action to the official PDI repository, for access to users and/or for DA scheduling requirements.
- Devs work on MASTER, and Branches/Tags are created for DEV, QA, and Live Deployments.

Don't move code directly from QA to Production.

- Production passes must be made with specific tags.

Setting up the PDI Environment: Methodological Guidelines

These methodological guidelines need to be followed for every implementation.

Your Development PDI repository must use an individual files solution without repositories for `.ktr` and `.kjb` ETL files. Use file-based solution ONLY in Development environments.

1. The repository must point to a local Git project checkout with the same structure as the PDI repository in the QA/Production environments.
2. Every ETL process must use the environment variable `${Internal.Entry.Current.Directory}` for any internal reference to another repository element to ensure proper functioning in different environments.
3. Communication between QA/Production and Development environments can only be made using the **Import** and **Export** processes.

4. QA/Production PDI repositories must have security configured to only allow authorized roles at import actions.
5. Disconnect version control from your enterprise PDI repository before beginning to import to development. Set PDI Version Control and Tracking Options has more information on how to do this.
6. Every **Import** must be based with a correct `tag` created for it.
7. Make sure all users are logged out of the PDI server before action (import, export, and purge) to prevent a corrupt bundle.

Migrating from PDI Enterprise Repository to Git Repository

Before you begin your export and migration, make a backup of your PDI repository. Here are the steps to follow when migrating from your PDI repository to your new Git Development repository:

1. Use export commands to extract all files and definitions from your PDI repository.
2. Upload the extracted structure to your Git project.[1]
3. Purge your actual Development PDI repository. This is a required step to follow for the migration process, in order to delete previous revisions.

Purging is permanent and data (shared objects such as servers, clusters, and databases, as well as content like transformations and jobs) cannot be restored. Purge Transformations, Jobs, and Shared Objects from the Pentaho Repository has more information about this process.

4. Disconnect versioning and components from every enterprise PDI repository. Set PDI Version Control and Tracking Options has more information on how to do this.

[1] Start a new git repository on Karl Broman's blog is also a good resource for this step.

Use Case and Step-by-Step Example

Here is an example use case followed by a step-by-step procedure. You will need to have administration permissions to execute this.

---

*The Department A Team completed its ETL Development for their project called **Project A2881 - Credit Card Data**, and wants to upload to the QA environment repository.*

They currently have all code uploaded to the Department A folder in Git, using the distributed version control system.

Here are the recommended steps to follow for the above use case:

1. As a best practice, create a new branch/tag in the Git repository for the project folder to be exported.
2. Create a .zip package with every job and transformation that needs to be exported to the PDI enterprise repository.
3. Open a `cmd` or `shell` window and point to the directory for the location of your running Data Integration server into PDI Production/QA server.
4. Use the `import_export` script (`.bat` or `.sh` depending on your OS) with the corresponding arguments to import the latest .zip solution package.

   Indicate the following important parameters for your environment:

   `--file-path` - the `.zip` file path to import into repository.

   `--path` - the path to the root directory in the repository structure to put your `.zip` file content.

   Here is an example script:

```
./import-export.sh --import --url=http://localhost:8080/pentaho --
username=admin --password=password –path=/public/ --
filepath=/home/Downloads/backup.zip --overwrite=true –
logfile=/temp/logfile.log
```

More information about import-export script arguments and options can be found in the Pentaho Help documentation.

Related Information

Here are some links to information that you may find useful while using this best practices document:

- Backup and Restore Pentaho Repositories
- Pentaho Documentation
- Purge Transformations, Jobs, and Shared Objects from the Pentaho Repository
- Set PDI Version Control and Tracking Options
- Upload and Download from the Pentaho Repository

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project:_____

Date of the Review:_____

Name of the Reviewer:_____

Item	Response	Comments
Is your Production environment structured for each use case?	YES_____ NO_____	
Did you set up developer permissions?	YES_____ NO_____	
Did you set up analyst permissions?	YES_____ NO_____	
Did you use the best practices explained in this document?	YES_____ NO_____	