Paper 365-2012

Simple Version Control of SAS® Programs and SAS Data Sets

Magnus Mengelbier, Limelogic Limited, London, United Kingdom

ABSTRACT

SAS data sets and programs that reside on a local network are most often stored using a simple file system with no version control, no audit trail of changes, and none of the benefits. In this presentation, we show you how to capitalize on the capabilities of Subversion and other simple, straightforward conventions to provide version control and an audit trail for SAS data sets, standard macro libraries, and programs without changing the SAS environment. Extending the interaction with Subversion using a standard SAS macro is also explored.

INTRODUCTION

Most organisations will use the benefits of a local network drive, a mounted share or a dedicated SAS server file system to store and archive study data in multiple formats, analytical programs and their respective logs, outputs and deliverables.

A manual process is most often implemented to retain versions and snapshots of data, programs and deliverables with varying degrees of success. Although not perfect, the process is sufficient to a degree. The step from a local file system to enterprise environments can be a fair investment and a high degree of change management if you already have an evolving analytics environment.

Off-the-shelf software, both Open Source and commercial, exist that provide simple source code control with versioning, audit trail and other features such as electronic signatures that can complement or even be combined with the current file system storage with little or no change to the current IT infrastructure.

Subversion is one of the popular Open Source version control systems that would allow version control and audit trail to easily be implemented. Additional features such as electronic signatures and business controls can also be added, dependent on and specific to an organisation's requirements.

A complete discussion is beyond the scope of this paper, but we shall discuss general considerations for version control of SAS programs, data sets and outputs, deploying Subversion and examples how SAS processes and tools can be integrated to draw upon Subversion functionality and facilities. The result is a very functional environment that allows both simple and quick implementations for a SAS analytics environment as well as advanced integration with formal compliance controls.

VERSION CONTROL

We perform some form of version control when creating documentation, a SAS program, publishing a SAS data set or creating an output. The simplest form is an implied draft versus final. In some cases, it is a manual step of creating a back-up copy of a single program, multiple files or an entire directory structure to allow for reverting to a previous version. Others would rely on an IT back-up in order to restore a deleted item, a previous version or reset a directory structure to some arbitrary point in the past.

VERSION LABELS

Versions and version numbering are frequent in daily activities, in both software and standard project documentation. A popular convention is to identify a version with a major and minor reference, such as 1.2 or 5.1. The first number could indicate a major revision or release while the second number would be a minor revision with corrections, fixes, tweaks, additions, etc.

Some software would have more intricate version numbers, such as 17.0.963.78, that would only apply to a certain style of process. In almost all cases, version numbers are a label applied according to a set of evolving rules and not defined by some universal standard.

Version labels do have one thing in common regardless of standard and convention; they denote some level of revision.

SAS PROGRAMS

SAS analytics programs tend to go through a simple revision cycle with an initial program and major or minor updates replacing the previous version until a final version is created (Figure 1). Each revision is either driven by a step-wise development or based on new data or changes to the specification. Seldom are multiple versions of a SAS program retained in parallel.

Figure 1. Versions of a SAS program

DATA SETS

Data sets tend to follow the same revision principle driven by incremental data updates, snapshots or extracts with the exception that it may be of interest to retain multiple snapshots or revisions in parallel such that they can be compared or used independently (Figure 2).

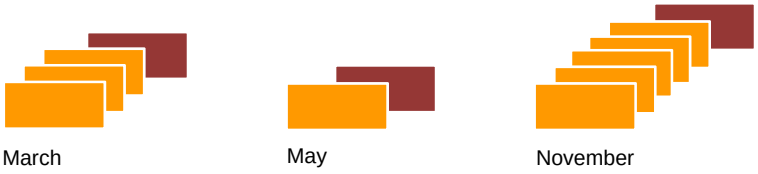


Figure 2. Versions of SAS data sets

In our example above, the three snapshots March, May and November would be managed independently with each containing only the relevant number of revisions. There are also no requirements that the different snapshots have to be sequential as the context for each snapshot may be dictated by process and activities. For example, the March snapshot could be an interim analysis and May a safety review while November is associated with the final report being populated continuously throughout the project.

OUTPUTS

Output created by a SAS program using an input data source (Figure 3) are similarly impacted by some form of revision with the addition of formal dependencies, e.g. an output is dependent on the SAS program that created it and the data that was used (colour shades in each of the three stacks). If either the program or data is updated, a new version of output could possibly be required. The vagueness is a product of process controls rather than formal relationships.

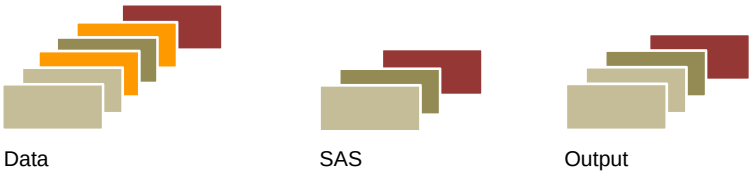


Figure 3. Versions of outputs

This iterative process would continue until the final output is produced. Throughout this process, the output is dependent on both the version of the program and data, which may be independent of each other. In our example below, our data has six revisions before final, our program has three and we have four revisions of our output.

There is also a potential workflow dependency. The workflow or a formal quality assurance process may rely on the chronological sequence of events whereby if a program recreated output, its time stamp would change and therefore any prior quality control or downstream events would have to be performed once again, perhaps initiating a parallel or sequential set of revisions and tasks.

Version control of SAS programs, data sets and outputs can become very complex as there are undoubtedly complexities, relationships and dependencies that extend beyond the three entities mentioned, but those highlight the basic issues faced when considering version control of SAS analytics environments.

We will revisit these factors throughout the paper below.

SUBVERSION

Apache Subversion, sometimes referred to as SVN, is a centralized version and revision control system that was initially designed as a replacement for the popular CVS and is today actively developed and widely used throughout Open Source projects, software communities and commercial applications. The basic nature and simple features also make Subversion a very elegant and efficient repository for SAS data sets, programs, logs, outputs and other associated files.

THE BASICS

Subversion is extremely simple. It is a system that manages repositories of files and folders, not much unlike your regular file server, while at the same time keeping track of changes. Analytics environments that employ Subversion usually consist of two environments, the repository and the working copy.

The *repository* is the central database with all the version-controlled files including the complete history of changes. Each repository is treated independently with its own directory structure, history and permissions.

The *working copy* is the directory structure where all the actual changes are performed. The working copy is a copy, or a download, that can be updated without impacting the central repository until changes are explicitly committed to the repository.

THE REPOSITORY

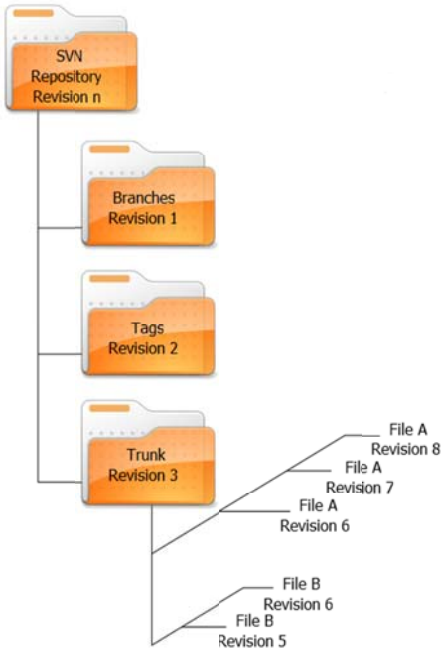
The Subversion repository "file system" (Figure 4) is most often described as a two dimensional directory structure. The first dimension of the directory structure is a *path* reference that follows the same principles as folders and directories in Windows, Linux and Unix.

The second dimension is the *revision*, which is the Subversion version identifier and is simply a counter unique to the entire repository.

One important aspect of the revision is that it is not restricted to one item, e.g. a single file or folder, in the repository. The example in Figure 4 includes two files, *File A* and *File B* in the folder *Trunk*, that are part of revision 6.

The revision is used in two contexts, the repository revision which denotes the revision of the last commit. The second context is the revision for each folder and file, which is the revision when the folder and file was added to the repository or the file's last update. Note that a parent folder revision does not increment when it has content added or updated.

Another perspective on revision is that it denotes a slice in time for the repository. For example, the repository at revision 5 in our example would include the folders *Branches*, *Tags* and *Trunk* as well as the initial version of file *File B*. Note that *File A* and the update to *File B* would not be included since they were not added until revision 6.



Source: Apache Subversion – Wikipedia.org

Figure 4. Subversion "file system"

One very powerful feature of Subversion is the use of transactions for repository commits, e.g. saving content to the repository. If you include many files in your commit, say 10 files, all those files are associated with the revision. If the commit would be interrupted or the commit of any one of those files would fail, the entire transaction fails and no items are updated in the repository. In CVS, the interrupted commit operations would sometimes cause serious issues with inconsistent and sometimes corrupt repositories.

PROJECTS AND DIRECTORY TREES

A Subversion repository is empty by default and does not require any specific directory or folder structure, and certainly not a directory or folder structure convention to function, although some workflow based tools do expect certain folders and directories to exist.

The majority of the recommendations and documentation do however highlight two simple conventions. Projects are defined as either the entire repository or represented by folders in the root of the repository. Subversion does

not impose to select one approach over another, but it may facilitate for the end users if a consistent approach is established.

The documentation and examples also note that each project contains the three project root directories *Trunk*, *Branches* and *Tags*. The discussion on how to use these directories is comprehensive and beyond the scope of this paper, but we briefly consider a common approach to highlight some very useful concepts and attributes of Subversion.

Translate this into a directory structure and each directory has a defined role.

- *Trunk* is essentially the main development area
- *Branches* could contain the development areas for the different versions, such as "1.0", "1.1", "2.0", etc.
- Tags would contain the different releases, e.g. a Branch frozen in time, similar to taking a snapshot or locking a directory tree to disallow changes and updates.

The concept is to strive for as much of the development to be performed through the trunk and any change to an active branch is merged into the trunk as often as possible as shown in Figure 5.

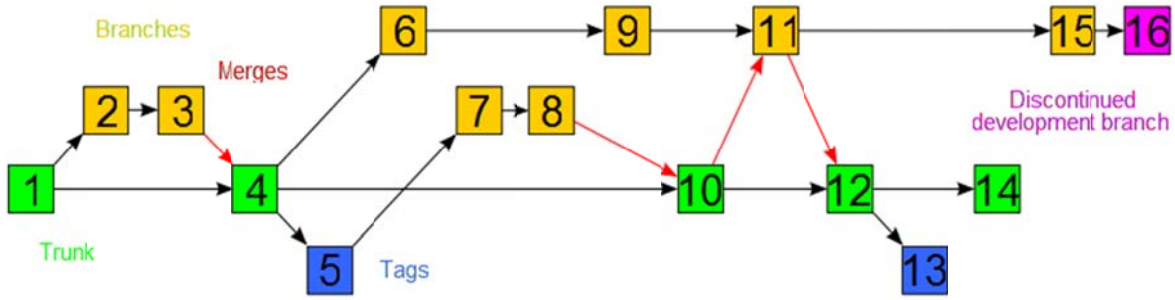


Figure 5. Trunk, Branches, Tags and merging

Source: Apache Subversion – Wikipedia.org

The concept behind the common Trunk – Branches – Tags directory structure can be adapted to specific cases as well. As an abbreviated example, Data Management and Biostatistics within Life Sciences could use a similar concept to Trunk – Branches – Tags (Table 1) when executing a clinical trial and subsequently generating Tables, Listings and Figures for clinical trial reports.

Software Development	Data Management & Biostatistics
Trunk	Standards
Branches	Reports
Tags	Snapshots

Table 1. Data Management and Biostatistics project root

The approach in Table 1 is based on a process where common effort across all reports is performed under the *Standards* directory tree. This can encompass both Data Management and Biostatistics reporting activities, even though they are most often separate but dependent processes.

The *Reports* directory tree would include all individual reports, such as internal reviews, Investigator Brochures, Investigational New Drug applications, interim analysis, clinical study reports, etc. Any reporting specific to a particular report would only reside in the appropriate branch and only merged into the Standards tree when applicable. The Reports directory is not restricted to Biostatistics as it is equally relevant to Data Management deliverables.

The *Snapshots* directory tree would include the deliverables for both groups, which could be validated data extracts, draft outputs sent for review to the Clinical team members, the final reports, etc. Workflow may also dictate how SAS data sets and programs are added and removed from branches and tags, although removing items from Tags, or the Snapshot in our example, is either rigorously discouraged and enforced or frowned upon unless severely restricted, closely governed and audited.

For example, an organisation implements a classic workflow with development, quality control (QC) and production.

A data set or program under development resides in the working copy until final. When the data set or program is final, it is committed to the Standards tree or an appropriate QC branch in the repository. If a data set or program

passes quality control steps and activities, it is moved to the Production ready branch. Otherwise, the data set or program is removed from the branch.

At the time of delivery, data sets, programs and any associated inputs and outputs are tagged as part of the release process. Only tagged data sets and programs are used to create deliverables to be shared.

ONE OR MANY REPOSITORIES

Subversion can manage a single very large repository or many smaller repositories effectively. There are benefits to both, but a convention of one repository per project or integrated projects can have benefits.

- Simplified access control
- Less revisions to track, e.g. revision 1,236,425 or 1,431
- Revision is specific to effort on a project
- Greater control over process compliance
- Easy to migrate to a new process standard

The coordination of larger distributed groups with regional Subversion repositories is also greatly simplified with project specific repositories, which we will discuss further on.

There are several commercial and Open Source Subversion administration tools that make management of one or more Subversion repositories a simple exercise. Administration of Subversion can also be integrated into existing process tools using the standard Subversion utilities with little effort (Figure 6).

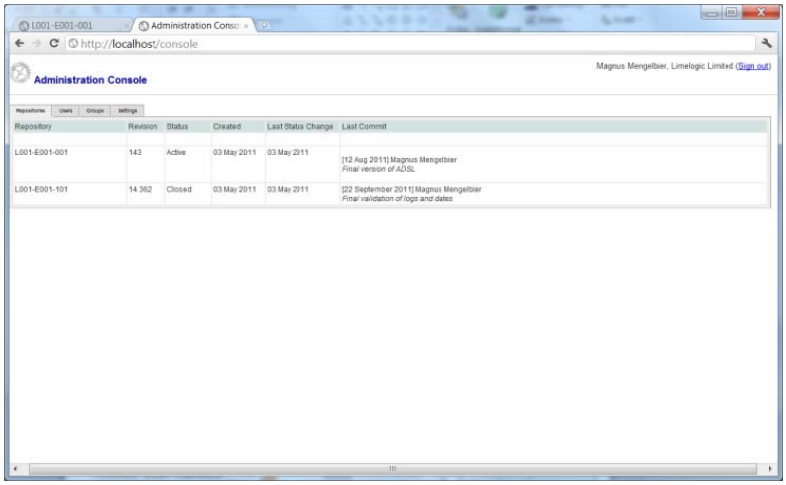


Figure 6. Custom administration console

DEPLOYING SUBVERSION

Subversion is a server application that can easily and quickly be deployed on Windows, Linux and Unix systems. Beyond the standard Open Source Apache Subversion packages, there exist several commercial and open source alternatives such as VisualSVN and SubversionEdge.

One very important aspect of Subversion is that it is primarily installed on regular file server and uses the regular file system for all configuration and content storage, which greatly simplifies deployments and configuration as no relational database is required. Additional features can be added using a Web server, such as Apache HTTP Server, but for the most basic configuration this is not necessary. Subversion has been referred to as IT support friendly as most organisations already have deployed at least 2 of the 3 standard components.

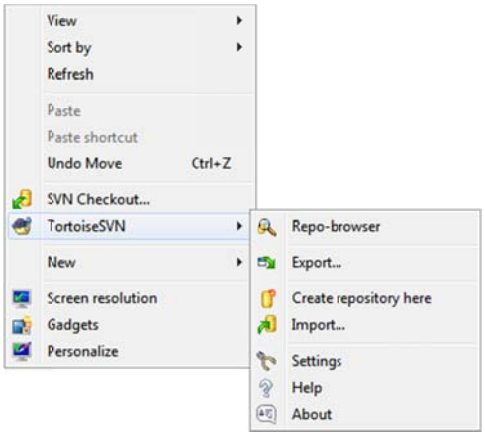


Figure 8. TortoiseSVN in Microsoft Windows File Explorer

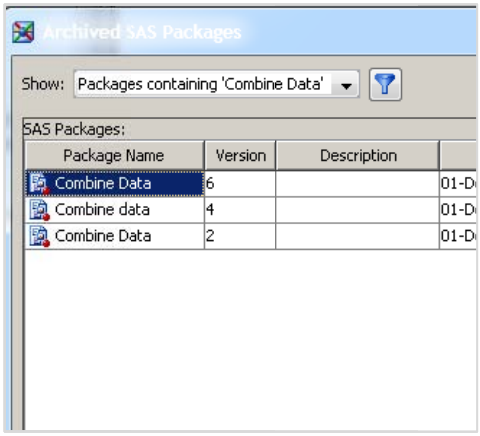


Figure 7. Clinical Data Integration Studio

The Subversion server does not include a default graphical interface, but the commercial applications mentioned above or including small utilities can provide simple and efficient interfaces depending on your requirements. Subversion access is also already or can be integrated into tools and utilities, such as Microsoft Windows File Explorer (Figure 8) or SAS Clinical Data Integration Studio (Figure 7).

One of the most common configurations for working with a Subversion repository in application development is to have one central repository and each workstation with a local private working copy managed through an application development application such as Eclipse. As SAS analytics environments tend to use a shared project directory and business process developed for that model, migrating to private working copies could be difficult to justify. However, it is not uncommon that the business process and tools allow and benefit from both a shared working copy and local private working copies managed through a development application.

EXISTING ENVIRONMENTS

A very common and quick approach for adding version control features to a SAS analytics environment is simply to add Subversion to the existing infrastructure, whether it is based on PC SAS and a file server or larger SAS server environments.

The approach will store the shared working copy of each project on the existing file server or share (Figure 9) with no real requirement to change the existing directory structure beyond adapting to a Subversion style work flow. This kindly eliminates comprehensive updates to standard configuration utilities and macros that are common for migrating to a new analysis environment.

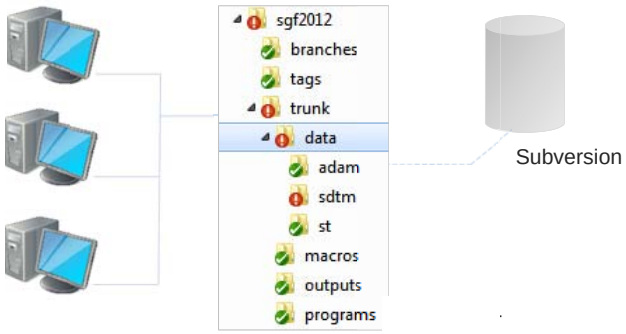


Figure 9. Subversion added to the existing analysis environment

The Subversion server can be deployed to the existing file server, but this could have implications on validation status and other compliance processes. Often a second file server is deployed dedicated to host the Subversion server, which with today's virtualization technology may reside on the same hardware as the file server.

An additional benefit with this implementation is that the existing business process tools can be retained virtually unchanged as the Subversion features are all managed through the workstations and the new Subversion server, and no major change or new software on the file server is required. This would also simplify upgrading Subversion when new versions are released as the impact on the working environment, e.g. workstations and the file server serving the working copy, is minimal. Experience has also shown that only small adaptations to existing standard operating procedures, work instructions and workflows are required in most cases.

SUBVERSION BEHIND THE SCENES

The Subversion programming interfaces and command line tools also provide a mechanism to integrate Subversion into new or existing tools, either by providing the primary user interface or simply integrating Subversion into a workflow and the workflow tools (Figure 10).

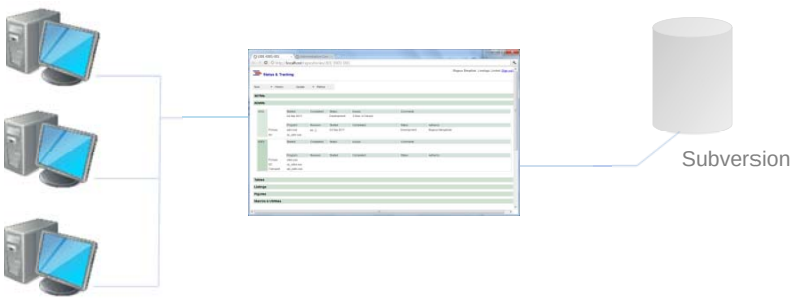


Figure 10. Workflow tool with Subversion as a background service

There are several examples, even a few with Microsoft Excel status sheets, integrating with Subversion via the command line client tools as the client tools readily produce XML structure output with a simple command line flag that can be consumed by the parent system or application.

REGIONAL SUBVERSION SERVERS

Subversion also includes built-in support for large global analysis environments with local SAS environments. Most implementations introduce one master repository with local regional environments (Figure 11) rather than attempt to do round-robin or other replication schemes. Although both are equivalent in most cases, global organisations may find the logistics of a central master repository easier to manage both on a central administration and project team level.

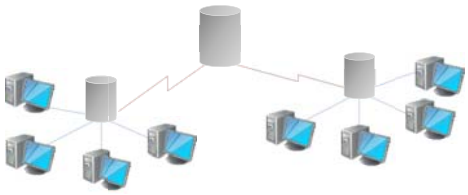


Figure 11. A master repository with regional slaves

The Subversion standard *svnsync* utility provides a simple mechanism to synchronise a read/only repository to different regions as a cost effective back-up and fail-over mechanism.

The approach used with *svnsync* includes a single master repository and one or more synchronised slave repositories, sometimes referred to as mirrors or sinks. In a general sense, all commits are redirected to the master repository by design and then on successful completion, will replicate the commit to each slave, which also interestingly includes the local regional slave repository if one is present. If the master is not reachable, the commit can be performed to the local repository and only replicated to the master and other slaves whenever the master is updated with the latest changes.

The master-slave relationships implemented with *svnsync* do require some supervision and active administration as the link may disconnect, but the approach is preferable to external replication scripts or other custom adaptations of Subversion actions since the complete history is retained within the master and each slave. If you would implement an external replication script, each replication would be treated as a commit to each slave repository.

In practice, the svnsync utility is useful to manage a set of regional read-only Subversion mirror repositories where you wish to eliminate continuous transfers of a largely static content volume. Good candidates for svnsync are standard code libraries or data repositories where the content is static with periodic updates.

An organisation may require that not all content within a repository or project is replicated to another specific region; either by business requirement or possibly that a business agreement dictates that data should not physically reside in a named region. A simple approach is for all projects to implement a project repository, rather than a single large repository for all projects, and each project repository to be synchronised with the master. Each region subscribes to the project repositories that it needs and thereby eliminates unnecessary replication.

Smaller organisations may opt to implement mirror repositories as redundancy or even when working with third parties, such as Contract Research Organisations (CROs). A CRO may have a dedicated Subversion mirror repository for their projects which would allow the Sponsor to retain an active copy of all work performed by the CRO.

SUBVERSION AND SAS

SAS and Subversion work extremely well in the same environment, primarily because of the simple nature of both systems and environments. SAS is a file-based analysis environment. Subversion manages folder and files as atomic artefacts.

There are essentially three primary functions in Subversion; reading content from the repository, the commit of posting content to the repository and getting information for a specific file or folder . The latter will be highlighted through a SAS programming example, but let us first consider the read and commit.

ACCESS CONTROL

Subversion has only two permissions; read and write.

The read will allow a user to browse the repository, view content and retrieve a working copy. Just because a user is allowed to create a local working copy does not necessarily mean that the same user can post an update back into the repository. The write permission grants a user the right to commit to the repository.

Most Open Source projects have anonymous read access and a tightly controlled and small group of committers. Removing anonymous read access is a simple administrative exercise.

Access is granted or revoked for a user through individual access rights or through group membership with group access with permission defined on both the entire repository and on individual directories and both support inheritance.

CONTROLLING COMMITS

The commit operation in Subversion is one of the more critical operations.

If you recall the brief encounter with Subversion revisions and commit transactions, the commit is valid for the entire repository and can essentially include all or any selection of files in the repository. This also implies that the comment associated with the commit is valid for the entire repository and simply stating “fixed missing” in the commit comment may not be the most practical summary.

Subversion provides a very simple, flexible and inherently powerful mechanism to take control of the commit to add simple rules surrounding the commit and any associated comment. Subversion provides hooks, small program scripts, that are executed in association with commits, one prior to the commit (pre-commit hook), during the actual commit transaction (start-commit hook) and one after the commit has completed successfully (post-commit hook). Hooks are also available for a subset of other Subversion events and actions beyond the commit, but we will focus on the commit hooks within the scope of this paper.

A commit does not by default require a comment and there are no rules that govern that the commit, and not just the comment, has to be sensible. A set of business rules can be added to the commit hook in order to assert some control. Prior to considering some examples, one important note is that the commit hook is executed on the commit transaction and can control if the commit is allowed to proceed or fails with an informative message. This implies that the commit hook can “inspect” the comment, the repository and even the files being committed.

The list of possible business rules are practically endless, but there are a few that can be very useful for a SAS environment and the requirements imposed on SAS environments from an industry and organisations point of view.

Scope of the commit

One simple business rule may be to control the scope of a commit. It is allowed to commit the entire working copy with all changes, which can be an issue if the working copy is shared. Subversion only really understands that a file or folder is new or has changed, but not the business context and rules that govern. Restricting the commit to only be allowed for a specific directory level or context, e.g. a single or group of related SAS program and associated logs and output, may provide a simple control to ensure sensible commits.

Commit comment

A rule may be imposed that the comment associated with the commit consists of a minimum of 5 words from a business process dictionary, which would eliminate the single period or some garbled text such as “The quick brown fox jumps over the lazy dog”. At the same time, a business rule may require that the word “new” is included for the first commit of a file and one of the words “update”, “updated” and “fixed” be required for any updates.

A comment for a SAS environment may also require a reference to include any SAS program names, if they are included in the commit. For example, a shortcut keyword #programs could be expanded to the list of SAS programs as a means to make comments more user friendly.

Commit files

Subversion does not understand SAS and SAS files by default, but simple rules can be added to the commit hook to ensure that associated SAS files are included in a commit. For example, a SAS log file can be required if a data set or another output such as an RTF file from a specific subdirectory is included in the commit. Further compliance checks can be added if business rules exist on the naming and location of SAS files to ensure that not just any file is included to satisfy the requirement.

Another interesting possibility is to use the file suffix to determine if a file type is allowed to exist in a specific folder. For example, a folder dedicated to data sets may refuse a SAS program, log or output file to be added to the data directory in the repository. To accommodate that extraordinary special case that will always occur, the user information in the commit transaction can be used to allow an override.

It is possible to keep adding rules and additional compliance checks. One important note is that the commit hook is not restricted to the repository or repository server. The post commit hook could forward the comment and additional details about the transaction to other systems and applications, which makes integrating Subversion with current systems and tools fairly easy.

REVISION IN A FOOTNOTE

Adding the revision of a SAS program to the standard footnote of output that it generates is a good example of querying the working copy or repository and using information in a SAS program. The approach can be wrapped in a SAS macro as a standard utility, but for clarity we will use regular SAS code.

The approach used is the standard command line utility *svn* with first parameter *info*, e.g. *svn info <something>*, to query the local working copy and some simple string parsing to extract the required information, essentially all wrapped into two DATA steps. You can equally interrogate a remote repository by specifying the item URL and any required credentials. If you query the local working copy, no username and password credentials are required as it is assumed that if you can see the file, you are granted access. To add security restrictions within a working copy, this is accomplished by the standard local file server mechanisms.

The *svn info* command can return the requested information as an XML structure with the command line switch *--xml*, which is very useful. Output 1 contains an excerpt that we will process to obtain the revision, author and last committed date of our example SAS program.

```
…
<commit
   revision="2">
<author>mmr</author>
<date>2012-02-17T08:49:01.157176Z</date>
</commit>
…
```

Output 1. Example XML output from the svn info command

The first of our two data steps is to interrogate the working copy, read in the *svn info* output and isolate the commit XML node, which contains the three properties revision, author and date we wish to retain.

```
data work.svninfo ( keep = str_line ) ;
   length str_line $ 1024 ;
   retain keep_line 0 ;  * commit xml block may be more than one line;

   infile "svn info X:\mypath\myprogram.sas --xml " pipe;
   input str_line $ 1-1024;

   *  identify <commit ...>  ;
   if ( index( str_line, "<commit" ) > 0 ) then keep_line = 1;

   *  if in commit block ... keep lines for later processing;
   if ( keep_line = 1 ) then output;

   *  end of commit block, e.g. </commit>;
   if ( index( str_line, "</commit>" ) > 0 ) then keep_line = 0;
run;
```

The second data set will parse the XML node using regular string manipulation. You can equally rely on an XML Map to more efficiently extract the required information.

```
data work.properties (keep = revision author date);
   set work.svninfo   end = eof ;
   length revision author date $ 200;
   retain revision author date ;  * keep properties as we inspect and parse each line of xml ;

   *  obtain revision – note the compress() function to clean up our string ;
   if ( index( compress(str_line, " "), "revision=" ) > 0 ) then
      revision = compress( scan( substr( str_line, index( str_line, "revision" )), 2, "="),
                           '"<>');

   *  obtain committer ;
   if ( index( str_line, "<author>" ) > 0 ) then
      author = scan( substr( str_line, index(str_line, "<author>") + 8 ), 1, "<");

   *  obtain date ;
   if ( index( str_line, "<date>" ) > 0 ) then
      date = scan( substr( str_line, index(str_line, "<date>") + 6 ), 1, "<");

   if eof then output;
run;
```

The result is a data set PROPERTIES with the character variables REVISION, AUTHOR and DATE. How and where those details are further used will most probably be dictated by business process, but as an example, the

standard system footnote in an output that makes note of the SAS programs used to generate the table, listing, figure, etc. could be extended to include the revision rather than just the program name (Output 2).

```
_____________________________________________________________________________
Age is the subject age at informed consent.

Program: myprogram.sas (rev 10236)
```

Output 2. Footnote in output that includes SAS program revision

There is no restriction that this approach cannot be extended to more than one file and to other dependencies, if we consider the output's dependency on a SAS program and any input data sets discussed previously. Similarly, if your footnote includes reference to say analysis data sets or source listings, it is possible to query their revision and add that as well given that the data set and listing files follow some standard convention on naming and location.

CONTROLLING SAS LIBRARIES

A more rigorous example of how the Subversion utilities can be used in compliance is simply to verify that specified input data sets or other input files in the working copy or executing environment is of the latest or a pre-specified revision and fail with an error if this constraint is violated.

This highlights one of the minor nuances of Subversion and the working copy as users have to ensure that the working copy remains up-to-date with the latest or pre-specified revision.

The *svn status --show-updates --xml* will verify that there are no changes in the working copy that needs to be committed to the repository and conversely that the working copy has the latest updates. The example in Output 3 shows that the DM data set has been added to the SDTM folder. Not the revision number -1, which recognises that the DM data set has not been committed to the repository.

```
<?xml version="1.0" encoding="UTF-8"?>
<status>
<target
   path="sdtm">
<entry
   path="sdtm\dm.sas7bdat">
<wc-status
   props="none"
   item="added"
   revision="-1">
</wc-status>
</entry>
</target>
</status>
```

Output 3. Output from svn status in XML format

By wrapping the call to *svn status* in a SAS macro, say %svn_library(), the program can verify that specified data sets or the entire library folder is up-to-date.

```
%svn_library( library = sdtm, select = DM AE LB, revision = HEAD );
```

The macro as defined above will verify that the working copy revisions of the data sets DM, AE and LB are in the repository and that the working copy contains the latest, e.g. the revision HEAD. The HEAD revision is a Subversion keyword reference to the latest revision. If we would like to use revision 945 instead, the revision number would be specified for the revision parameter value.

This check could simply be circumvented by not calling the %svn_library() macro. One programming trick to avoid this is to have the macro create a new library VSDTM with one SAS data view for each data set specified in the select statement. If the input data set is not specified, a corresponding view is not created and any reference would result in standard SAS errors. As the program and log should refer to the library VSDTM and not SDTM, a simple log checker can assert that the convention is followed.

This compliance approach can of course be extended to other input files and environments that do not use Subversion, but that is beyond the scope of this paper.

THE DIFF

One of the most powerful features with Subversion is the ability to easily identify the differences between two files. For SAS programs and other text files, the default setup will produce a very informative difference. Both the standard command line utility and clients like TortoiseSVN (Figure 12) accept custom scripts for identifying differences, most often referred to as diffs.

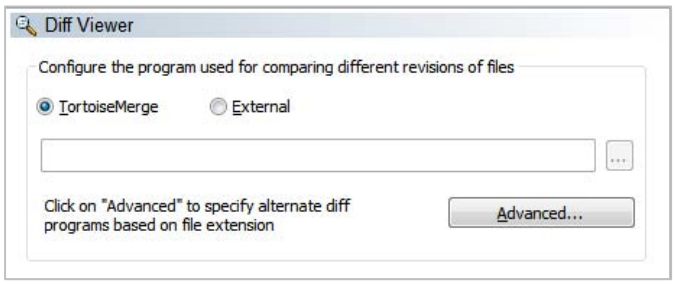


Figure 12. TortoiseSVN custom diff setting dialog

We can therefore create a custom script to create and display a diff on SAS data sets or any other file type where we want to generate and display a custom diff. The later versions of TortoiseSVN come with diff viewers for files such as Word, PowerPoint and Excel, but not for RTF and PDF files (TortoiseSVN version 1.7.6).

MOVING, RENAMING AND DELETING FILES

One of the last, but still important nuances faced by users is the procedure to move, rename or delete files. The best, and strongly recommended approach, is to use the Subversion utilities and clients such as TortoiseSVN rather than to just delete a file with file system menu options and commands and then have to try and fix it with a full repository commit. If a file is moved, renamed and deleted using the utility and tools, comments will be associated to the action as you would expect.

CONCLUSION

The basic nature, flexibility and simple features make Subversion a very elegant and efficient repository for SAS data sets, programs, logs, outputs and other associated files. The different options for deployment and the almost endless possibilities for process integration, will allow a quick, simple and an appropriately customized environment with minimal effort. Add the many available user interfaces along with simple command line utilities and the environment can continue to evolve and further be integrated with SAS, analytics environments and other supporting business systems, tools and utilities as requirements change.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Magnus Mengelbier
Limelogic Limited
Regent House, 316 Beulah Hill
London SE19 3HF, United Kingdom

Phone: +44 208 144 5701
E-mail: mmr@limelogic.com
Web: www.limelogic.com