5 SVN

Introduction

Subversion (SVN) is a tool that allows the user to efficiently save or restore different versions of a file. It is used by software developers to solve a number of problems that arise when multiple users collaborate on code. We use it for the same reasons.

Make.py

Every directory in our repository that produces output has a Python script called make.py that (i) deletes any old output files in the directory, (ii) imports clean copies of any external input files the directory needs to run, and (iii) executes all the code in the directory in the correct order. Thus, the output of the directory after running make.py is guaranteed to be the same regardless of when or where it is run.

You may only commit directories to the repository immediately after successfully running make.py. A successful run means that make.py runs to completion and that neither make.py nor any of the individual scripts it calls throws an error.

This is the single most important rule in using SVN. If you follow it, every directory in the repository is guaranteed to be replicable. If you violate it — run an additional script or copy a file manually before you commit, say — the entire system breaks down. The next person who tries to run the directory will not know about those manual steps and so will not be able to reproduce the output.

There are no exceptions to this rule. Even if you make a trivial modification to a directory, you must run make.py before committing. We realize this will sometimes cost a significant amount of time, but we believe this is a cost worth paying to have a robust system. In the very rare cases where you need to run and commit only a subset of the code in a directory, you should modify make.py to call only those scripts.

See the readmes here for details on writing and executing make.py scripts.

Note: There are directories in the repository that use an old format in which the make file is written in a DOS (.BAT) or BASH (.BH) format. When modifying these directories you must update them to use the make.py format. Note that directories using make.bat will expect that the user has a Perl script called get_externals.pl set up according to the instructions here .

Organization

Structure of the Repository

- /trunk/ - main trunk of files.
  - /raw/ - Stores raw data. Folders in /raw/ should only rarely be changed. Code performs minimal cleaning only.
  - /derived/ - Takes data as input (often from /raw/) and produces log files and data as output.
  - /analysis/ - Takes data as input (often from /derived/) and produces log files and results (not new data) as output.
  - /drafts/ - Stores drafts of papers
  - /slides/ - Stores slides for talks
  - /lib/ - Stores code libraries.
  - /replication/ - Produces replication files for journals and for sharing data online.
  - /admin/ - Administrative resources, including directory templates and this RA Manual.
  - /other/ - Miscellaneous directories, such as MG / JMS websites.
- /tags/ - contains "cheap copies" of particular revisions of directories that we want to be able to refer to later.
  - We primarily use this to point to revisions at which we "exposed" some of the repository's contents to other economists, such as

    - Tagging a revision of a "slides" directory when we give a talk
    - Tagging a revision of a "drafts" and/or "analysis" directory when we submit a draft of a paper to a journal or post it online.
  - We also sometimes use it to point to revisions prior to a major reorganization of a complex directory, so that we can easily refer to the pre-reorganization version.
    - Because by definition every change is versioned and logged, the need for this should be rare.
  - No other directories can call files from a tag.

- /branches/ - used for feature development separate from main trunk.
  - We primarily use this to experiment with significant modifications to a directory (often a code library in /lib) that we are not sure we want to make a part of the main trunk.
  - Only other branches can call files from a branch.

Structure of a Folder

Below we list the common elements of repository folders. Note that requirements differ by type of folder (/raw, /derived, etc.). Please review the template for each folder type for more details on the requirements for each.

- readme.txt - An overview of the directory's contents and any notes on using the directory.
- schema.mwb - A visual representation of the data tables in the directory.
- /code/ - Contains the code that runs the directory
  - make.py: See discussion above.
  - externals.txt: Contains a list of necessary files that live in other folders of SVN, or occasionally on the share drive.
    - get_externals.py copies necessary files from the repository and shared drive locations, which are listed in externals.txt
    - **Note**: There are directories in the repository that still use get_externals.bat. When modifying these directories you must update them to use externals.txt. To do this, we make use of translator.pl found in /lib/perl/translator in the repository. This file will read the old get_externals.bat file and create an externals.txt file from it. To do this, add a line to make.bat to run translator.pl on get_externals.bat. For full instructions, see the readme included in this directory and the readme.
- /external/ - Used for storing a local copy of data that has been stored on the Repository elsewhere.
  - Only make.py should put files in the external subfolder, calling files in externals.txt.
  - /external/ subfolder is not committed to SVN, but may appear when you run make.py.
- /temp/ - Used for storing temporary output/passing output between code.
  - This subfolder is not committed to SVN, but may appear when you run make.py.
- /output/ - Contains output from the directory.
  - make.log - A complete LOG of all code output. This can be diffed to determine whether a directory has changed.
  - data_file_manifest.log - A "checksum" for Stata datasets which are saved using **save_data**.
  - In /derived directories, /output/ will usually include both LOG files from code and data.

- In raw directories only:
  - /data/ - In raw directories only, contains data files as other folders would call them
  - /orig/ - In raw directories only, contains original data files.
    - Exists only if files in /data/ are not the original ones (i.e. orig files required cleaning)
  - /docs/ - Documentation files received from original source. Can also include correspondence (e-mails, notes on conversations, etc.) relevant to understanding data contents.

Key principles

Portability: Others can run the code, anywhere, anytime, and reproduce output.

- Directories in the SVN operate and interact as self-contained modules. Each one runs independently of the other folders.
- When directory B must use files from directory A, directory B calls those files with a specific revision number. That way, the output from directory B is stable even as directory A is modified.
- All file calls within a directory are to relative paths ("..\Ever\agree.txt" instead of "C:\Pat\Best\Path\Ever\agree.txt"). This ensures that directories are portable across PCs and locations.
- Code is deterministic. If randomization is required, set a specific seed ahead of time so that results do not vary across runs.

Readability: Others can read the directory and code and understand how it works.

- Code and output files have short, descriptive names that are consistent with the rest of the repository.
  - For example: Do not create a subfolder "programming scripts". That's what "code" subfolder is.
  - Code directory has code (not output), output directory has output (not code), etc.
- Everything has a purpose
  - Never commit a subdirectory with 0 files
  - Remove code that is not being used.
- The directory conforms to our standards as of when it is committed.
  - When we change our standards (for example switching from get_externals.bat–>externals.txt), we do not go through and update every directory.
  - Rather, we adopt the principle that users modifying directories are responsible for bringing them up to date when they do so.
  - As a result, all directories should be consistent with standards as of the revision at which they were last modified.

Work Cycle

- Typically there are three things you'll do with the repository: Create a new folder, modify an existing folder, and commit this creation/modification.
- Creating a new folder
  - SVN copy the appropriate folder template from /trunk/admin/Templates/ to the desired location in the trunk.
  - Check out the newly created directory.