Data Repository Format and Management

Jerod Weinman
jerod@acm.org

October 21, 2014

Keywords Reproducible research, version control, experimental computer science

1 Introduction

A properly managed data repository is important for several reasons. First, we want any data processing experiments to be **repeatable**. That means, we should be able to re-run them with the same results. For that to happen, the code and any input to any experiment must be stored. Alternatively, the version controlled source code and the method for creating that input must be stored, even if the data has been removed and the source code long since changed. Therefore, our data repository must *always* store the method and software versions used for creating data; we prefer also to store the data itself for as long as possible. The second major reason for the data repository is that we want to record **how** a particular data set was generated. This understanding may be important for describing and interpreting experiments that rely on the data.

Data collections are never modified; they are immutable. They may only be recreated as new collections with bugs fixed or parameters altered. Thus, traditional version control methodology is inapplicable, but orthogonal and complementary. With these motivations in mind, we outline a format and procedures for maintaining a data repository for reproducible research.

2 Format

Every directory in the collection is one of only five types. We shall describe these in more detail later, but the following is an overview of these five types:

Collection	All the elements for one data set, with subdirectories:	
	Data	The actual data (potentially with its own subdirectories)
	Source	Scripts or program source for generating the data
	Binary	Executables or object files for generating the data
Hierarchy	Partitioning directory for collections	

2.1 Hierarchy Directories

The hierarchy directories organize and partition processed data into meaningful areas. These may be recursive, with certain restrictions: promote a uniform partitioning, the subdirectories of a hierarchy directory must be all hierarchy directories or all collection directories.

Names should be short, preferably one or two words, contain no capital letters (unless part of an acronym), and use underscores for separation. Full words should be used. Hierarchy directories should have the most general names possible, to promote further division of contents as necessary, with minimal renaming.

2.2 Collection Directories

Collection directories contain all the information about a particular data set, including dependencies, meta-information (author and description), as well as the actual data. Once the collection is established, no part of it should be altered, with the exception of a single file (`BUGS` below) listing errata or *nota bene*.

The name of a collection directory should have the format `collection_name-YYYYMMDDhhmmss`, where the latter portion indicates the creation time. Besides creating an informative name, the timestamp will help place the work in a temporal context of development and also make it easier to informally refer to a collection without a complete path, with little chance of ambiguity. (See Management, below, for suggestions on creating a new collection.)

The collection name should balance descriptiveness and specificity with brevity. This balance may prove difficult in practice, especially as new elements are discovered that may need to be varied and, thus, indicated in the name for distinguishing collections. The most general portions of the name should be first, and most specific portions last, to promote the ease of future collection partitioning with hierarchies. Abbreviations should be avoided, and words should be separated by underscores. No capital letters should be used (unless part of an acronym).

The remainder of this section outlines, in more detail, the necessary contents of a collection directory.

2.2.1 Dependency file `DEPS`

The dependency file must list *any* data dependencies the generation process for the data in the collection relies upon. It is a text file in the collection directory called `DEPS`, with each line having the format

```
alias relative/path/to/collection
```

The alias should be a one or two word simple shorthand for the required collection. No capital letters should be used in the alias, though an underscore may be used to separate words. A single space should separate the alias name from the collection. No two

lines should have the same alias in a given collection (though different collections may have identical DEPS files). The path to the collection should be relative to the data repository root, which is specified externally by the environment variable DATA_REPOSITORY.

Tools are provided as a part of the repository to process DEPS files in various languages (e.g., bash, C, and Matlab), so that other data collections can be easily referred to and accessed via the aliases.

The DEPS file should **never** be altered after the collection has been made, except as warranted by a data collection move into a hierarchy directory (see Management below).

2.2.2 Information file INFO

The information file is a human readable description of the data collection. It should contain the following headings:

COLLECTION A short, single clause description of the data collection (not even a complete sentence); mostly this should be an expansion of the collection name

AUTHOR The author's name (if a student, graduation year should be included)

SUMMARY A reasonably detailed objective description of the data and generation process; also any information about reading/interpreting data (e.g, if a special format is used); include the revision number of any external code repositories the data creation relies upon

PURPOSE Brief, original motivation for creating the data (though future use may change)

SEE ALSO The originating collection name (not the full or relative path); used only if this collection folder (sans data) was copied from another (via copy-collection, see below) and modified

The INFO file should not be modified after creation. If amendments are warranted, the likely place is the BUGS file (see below).

2.2.3 Makefile

The Makefile is a standard file format for creating the data using the command make. The first (default) target should create the data; the standard target name is collection. If an executable is required, it should also be built by the default target. To build only the binary executable, without the data, the target bin should be used (for the command make bin). To remove the data and any executables or object files, the target clean should be used. This should harmlessly remove the data and bin directories described below.

2.2.4 Source subdirectory src

The source directory contains the script(s) or human-readable source code used to create the data. These files should contain **no** hard-coded absolute directory paths. Only the tools for accessing the contents of the DEPS should be relied on for accessing external data. All other parameters should be visible and not externally dependent. The contents of this directory should **never** be modified once initial debugging is complete. If later bugs are found, a BUGS file may be created in the collection directory. Because the outcomes of other experiments or collections may rely on this data, to maintain repeatability, it should not be altered, even if it is "incorrect" with respect to the summary and/or purpose listed in the INFO file.

2.2.5 Binary subdirectory bin

The (optional) bin directory may contain binary executables, object files, or other byproducts of compilation that may be created by the Makefile. The directory should not contain anything integral to the collection, only transitory/temporary files, as it may be deleted by a make clean command.

2.2.6 Data subdirectory data

Finally (and most importantly!), the directory data will contain the results of running make. The contents of this directory are considered the data of the collection. Any other collection that depends on this one must only rely on the contents of data and no other part of this collection directory.

2.2.7 BUGS file

Because the collection should not be modified once it has been created, a BUGS file may be created if and when any limitations, issues, bugs, or general *nota bene* are discovered in the collection. Each note should be accompanied by a date stamp and attributed to an author (see **Author** guidelines for INFO file, above). For example,

```
The Matlab function normstack (from the VIDI
package), which this data relies upon, was
discovered to have a bug.  Inputs of type
other than single gave erroneous output.  This
collection passed inputs as doubles, rendering
the data incorrect.
[Jerod Weinman 16 June 2009]
```

2.3 External source code

One of two types of external source code dependencies may exist. The first dependency may be a standard software packages or libraries. These system-level dependencies (i.e., version of Matlab, standard C library, etc.) are taken as given and are undocumented (though arguments against this practice are valid). The other type of dependency is the research code base. All research code that is not part of a collection resides in some Subversion or Git repository and is accessed via the $DATA_REPOSITORY/svn or $DATA_REPOSITORY/git environment variable path. Collection INFO files must be manually documented with the revision number of any repository used to generate results. For example write

```
svn://weinman/vision/tools r89
```

for local (svn.cs.grinnell.edu) subversion paths, or otherwise the full server name. For git, write

```
git://hub-user/repo/branch commit-hash
```

if the hub is github.com; for another hub, document as

```
git://hub-url.tld/path-to/repo/branch commit-hash
```

3 Management

Creating Creating a new collection from scratch should be done by copying a skeleton template. The creation utility may be invoked by

```
utils/make-collection collection_name
```

which will append the current time in the appropriate format to the name and populate the directory.

Copying To copy a collection (e.g., `DEPS`, `INFO` and `src` files, but no data) for creating a variant experiment, use the utility

```
utils/copy-collection src dest
```

Note that the destination collection must first be created using `make-collection` (i.e., to generate datestamps correctly).

Warning! Do not reverse the source and destination collection arguments, lest your source collection be overwritten. (A daily backup *may* exist.)

The SEE ALSO field of the new `INFO` file should automatically be updated by `copy-collection` to reflect the source collection.

Locking After a collection has been run, it should be preserved by removing all write permissions using the utility

```
utils/lock-collection collection_name
```

Moving [Not yet implemented.] To move a collection (or a set of collections) into a new hierarchy directory, the utility `move-data-collection` should be used. This will allow you to move multiple collections and update the `DEPS` files of the *entire* repository with a single invocation. First, the new hierarchy directory should be created. Then, the move tool usage is

```
move-data-collection \
  dst-hier-dir \
  src-coll-dir [src-coll-dir2 ...]
```

If the source and destination directories are relative, the move command will translate them into their absolute paths (using the current working directory) for the purposes of modifying `DEPS` files. At least one source collection directory must be given, but shell wildcards will allow you to move multiple collections into the destination directory.

Data Repository Quick Reference

Hierarchy Directories

- Recursively partition processed data
- All subdirectories of same type (hierarchy or collection)
- Naming
  - General names
  - Full words (no abbreviations)
  - Short (one or two words)
  - Lower case letters and underscore only
  - Capital letters only in acronyms

Collection Directories

- Contains one data set
- Only `BUGS` file may be modified
- Naming
  - Format `collection_name-YYYYMMDDHHMMSS`
  - Generated by `date +%Y%m%d%H%M%S`
  - Descriptive but brief
  - General before specific
  - Avoid abbreviations
  - Lowercase letters, numbers, and underscores only
- `DEPS`
  - Text file, each line in format:
    `alias /path/to/collection`
  - Lowercase letters and underscore only in `alias`
  - alias and path space-separated
  - Each alias is unique in a collection
  - Never modified (except by repository utilities)
- `INFO`
  - Human readable description of collection
  - Contains headings
    - Collection
    - Author
    - Summary
    - Purpose
    - See also (blank unless `copy-collection` used)
  - Includes versions of any code repositories used
  - Never modified
- `Makefile`
  - Default target creates data
  - Target `bin` (optional) creates binary executable
  - Target `clean` removes `data` and `bin` directories
  - Never modified
- `src/`
  - Scripts/source code for creating data
  - No hard coded directories, only aliases from `DEPS`
  - Visible parameters with no external dependencies
  - Never modified
- `bin/`
  - Temporary binary executables, object files, etc.
  - May be deleted without harm to reproducibility
- `data/`
  - Output resulting from running make
  - Other collections may only rely on this directory
- `BUGS`
  - Only modifiable portion of collection
  - Adds limitations, issues, bugs, or *nota bene*
  - Sign each entry with name and date:
    `[First Last dd Month YYYY]`

Review Checklist

Students will submit collections from their personal copies of the data repository for addition to the master copy. Before doing so, each collection should be reviewed by a peer who must attest affirmative answers to all the following questions.

Location

- Is the collection in the proper location for the data it contains and the nature of the experiment?
- Does the collection in a directory with no sibling hierarchy directories?
- Does the location promote a uniform partitioning; is the leaf hierarchy directory unlikely to become cluttered with many collections?
- Does the path of hierarchy directories contain - only one or two (maybe three) word names? - underscores, rather than capitals (except for acronyms) - full words?

Name

- Is the collection name descriptive?
- Is the collection name as brief as it can be? If not, consider reorganizing the hierarchy to put more qualifiers in directory names, rather than collection names.
- Does the collection name put general nomenclature before specific distinguishing nomenclature?
- Does the collection name avoid abbreviations?
- Are words separated by underscores?
- Is the collection name free of capital letters (except for acronyms or variables as appropriate)?

Collection

- Is the collection directory free of backup files?
- Does the collection directory contain only `DEPS`, `INFO`, `Makefile`, `src/`, `data/` and `bin/` (optional)?

`DEPS`

- Are all aliases one or (rarely) two word shorthands
- Are multi-word aliases separated by underscores?
- Are all aliases all lowercase?
- Are all aliases used by the collection?

`INFO`

- Are all lines 80 characters or less?
- Are all sections separated by two blank lines?
- Is the collection description a single line?
- Is the collection description adequate?
- Does the author's name include a graduation year (e.g., Herbie Hancock '60)
- Do authors' names appear on separate lines?
- Does the summary use single blank lines between paragraphs?
- Does the summary accurately describe the processing or experiment being performed?
- Does the summary describe how the results are stored completely and clearly? (I.e., so that one could create a dependent collection without reading the source)
- Does the summary include revision numbers of any SVN repositories it relies on?
- Does the purpose statement fit on one line?
- If in experiments, does the purpose statement intimate the question being addressed by the collection?
- Does the see also field include only the name of a collection (without a leading path or trailing `/`)?

`Makefile`

- Does the lead target (`collection`) create the data?
- If non-data (i.e., compiled code) must be created, is there a `Makefile` target for it?

`src/`

- Is the the `src` directory free of backup files (`*.~`)?
- Is the `src` directory free of data files and program output?
- Does the source link or add paths *only* to libraries/repositories that are used by the collection?
- Are all source code lines 80 characters or less?
- Is the source code all effectively "glue" code?
- Is the source code adequately documented?
- Is the source code clear, well-organized, and efficient?
- Is the source code free of hard-coded filepaths, using only relative file paths (i.e., from `DEPS` or `$DATA_REPOSITORY/svn`)?
- Is the source code sufficient to generate everything in `data/`?