Document organization and testing

M. Schwab[1]

keywords: *not available*

ABSTRACT

Authors should organize electronic documents around the processes they describe. In other words, documents should be object oriented. This structure allows authors to easily copy or move chapters. Additionally, authors should avoid any code duplication.

DIRECTORY STRUCTURE

The structure for the canonical SEP document is based on two principles.

- The structure minimizes code duplication. Code duplication aggravates software maintenance and usually leads to inconsistent code copies.
- The structure isolates each document, chapter, and each figure as a standalone object. Such an object oriented style facilitates easy reconfiguration and re-usability of parts of an available document.

Unfortunately, both principles collide, as we expect from most good principles.

In the following paragraphs I suggest a directory structure which compromises between our two lofty goals. This directory structure has evolved from Claerbout's book structure and the SEP report structure. It may need to be extended to comfortably accommodate more interdependent thesis documents.

Book or Thesis

Like all SEP documents, a book or thesis is a UNIX directory tree. Its top directory includes usually following subdirectories:

- **Adm** includes all kind of administrative files. The directory has to contain `all.tex`, DOCNAME.csh, and DOCNAME.dvi. (DOCNAME is the capitalized name of

[1] **email:** matt@sep.stanford.edu

the document, such as TDF, or SEP86.) The central rule in the Adm Makefile is `install-to-read`. If you need to create such an Adm directory than copy and modify one of an older document and talk to the person in charge for TEX. Every document needs an `Adm` directory.

- **Dvi** stores the documents dvi file. The file name is `DOCNAME.dvi`. The Makefile rule `install-to-read` in the `Adm` directory installs the dvi file in this directory. Every document needs an `Dvi` directory.
- **Lib** stores the source code for a subroutine library and a Makefile which manages the library updates. Such a library becomes necessary when several chapters (see below) share subroutines.
- **Src** stores the source code for a set of main routines and a Makefile that manages the corresponding executables. Such a source directory becomes necessary when several chapters share common main routines.
- **Bin** stores executables shared by several chapters. The corresponding source code resides in the Src directory.
- **Data** contains data sets which are shared between different chapters. The data sets comprise the header and data file and should probably combine them into a single file to avoid possible absolute path name definitions (see below).
- **chap1** corresponds to the chapter of a conventional book. A book or thesis contains usually a set of chapters. Each chapter is independent of all the other ones except the shared files in the Lib, Src, Bin, and Data directory. Depending on the complexity of its contents such a chapter directory can vary in its structure. In the next paragraph, I give a standard example.

Chapter

The chapters contain the building blocks of the document. Here the author expresses his thoughts and shows his results pertaining a specific topic. Depending on the complexity of the chapter's contents, the structure of such a chapter varies considerably.

As a minimum a chapter contains following files and directories:

- **Makefile** contains the rules on how to generate the chapter's result files from its source files and how to remove all intermediate junk files which are created in the process.
- **Fig** is a directory which stores the local result (usually figure) files.
- **paper.tex** is the author's text file. The LATEXlanguage allows the author to integrate push buttons and figures in his text (cite SEPHELP textour, referencing).
- **foo.c** is a possible source code file which is needed for generating the local result files but which is not shared with other chapters.

If the chapter contains a wealth of independent figures each with its own local source and executable files, it is beneficial to separate all rules and sources pertaining a single figure into small, manageable, separate subdirectories. The subdirectory's Makefile nevertheless deposits the final figure in the chapter's top figure directory `Fig`. The top Makefile lists all figures created in all of its subdirectories in its `RESULT`. Its clean rule has to descend into each subdirectory. The script `makemake` automatically generates such a top directory Makefile from its subdirectory Makefiles.

Some issues about how to organize a complex and interdependent document remain unresolved. For further illustration I suggest you have a close look at Claerbout's organization in his most recent books (cite TDF BEI).

SEP report

A standard SEP report is a simplification of the slightly more complex book and thesis structure described above. In a report an author's paper acts as a chapter. Since these report papers are usually independent from each other, the top level document directory does not contain Src, Lib, Bin, or Data subdirectories (except in some rare cases where it may makes sense to share a data set which several author's use in their individual papers).

INTERACTIVITY

Interactivity in a document comes in very different forms. The author can define his own form of interaction by defining a rule called `figname.action`. However, the standard SEP idoc rules offer a set of default interaction rules based on the naming conventions of the figure name suffix (cite idoc SEPHELP; needs up-dating).

REPRODUCIBILITY

The goal of reproducibility is to help communicate a computational process to a reader by allowing him to apply and modify the process easily. A finished document includes all result files and all the source files to re-create the result files. (The source files I am addressing here are files which are specific to the computational task. The document requires additional "environment" files which it assumes to be present such as a compiler, a Xwindow system, a UNIX operating system, or a program displaying dvi files.)

The three basic actions a reader wants to take when reproducing a result are:

- **burn** removes all reproducible result files.
- **build** updates all result files (even the irreproducible ones)
- **clean** removes all files except the source and result files (those files I call intermediate files).

The burn and build rules an author does not have to define himself if he lists all his figure names in a definition for `RESULTS` and includes SEP.idoc.rules. These idoc rules supply a default for the burn and build rule as long as the author adhers to the SEP naming convention for result files.

These standard naming conventions distinguish three classes of files in terms of their reproducibility:

- **ER** are files which a reader can burn and rebuild as described above.
- **CR** are files which a reader can burn and rebuild under certain conditions as outlined in the accompanying warning file. If the result file takes more than 15 minutes to compute, it is conditional reproducible.
- **NR** are files which a reader is not able to rebuilt. Such result files end with the suffix `save`.

TESTING OF DOCUMENTS

Here I suggest a suite of tests which an author should run on each chapter after finishing his research (or maybe even while he is building his new document). The same tests are run when the document is tested for a cdrom.

- **make clean** removes all intermediate files. Make sure clean visits all subdirectories. Intermediate files not removed by clean will prevent a successful reproduction of your results on a read-only cdrom. Take a careful look if all intermediate files have been removed. Be careful when using someone else's: what he considers junk files maybe precious to you. Make sure you remove all executable files.
- **Wastestat** looks for files which are junk. If you find a file that is not needed, then remove it. If it is an intermediate file generated by your clean rule, then include it into your clean rule. I suggest you run the different options of Wastestat separately, so that you know why the script considers the file a junk file.
- **Execstat** lists all executables in your chapter. Your chapter should probably not include any executables but source code. The reader creates the needed executable when needed.
- **Resstat** computes three lists: figures in your RESULTS statement, figures in your paper.tex plot statements, and files in your figure directory `Fig`. It then compares the three lists for consistency. At this point of our testing suite, Resstat should not show any inconsistencies and the summary statement at then end should count all figures.
- **make burn** burns all reproducible figures at once.
- **Resstat** lists now all reproducible figures as missing. The summary statistics should list no reproducible figures.

- **make build** up-dates all figures and rebuilds the missing ones. This command should finish without interference by you. A common mistake is to display a figure after having reproduced it. This prevents the command from running efficiently in a batch-mode as it is necessary when testing an entire document: reproducing and displaying a figure should be done by two separate make rules.
- **Resstat** should yield a result identical to the first Resstat: no inconsistencies.
- **make clean** removes all intermediate files.
- **make build** checks all files if they are up-to-date. No action should result from this command. A common problem arises when your hierarchy of make rules contains a rule without dependencies. All the files depending on such a rule are always out-of-date.

Once you have tested each chapter, you can test the entire document using the script `Tour` (see selfdoc) and a similar cycle of commands as above. Try to write your make rules so that they produce little or no output when they execute successfully. A small output log during testing is much easier to check for problems.

The structures and tests described in the previous paragraphs help develop any self-contained reproducible and interactive document. The observations are independent from the documents medium such as on-line, cdrom, or on-the-web. To store such a document on a read-only cdrom adds a few constraints which I will discuss next.

CDROM

Storing a document on cdrom leads to three significant consequences, which an author needs to know about when writing a document:

- Space limitation: Currently SEP cdroms can contain 350 MBytes of document. The authors of a document have to ensure that their document does not exceed 350 MBytes (clean your document with `Tour clean` and then run `du -s .` in your top directory). Usually, data files and figure files consume most of the space. You may want to consider compressing some of your files using `gzip`. Unfortunately, removing data and figure files affect directly the reproducibility and beauty of a document. The space limitation also prevents us from offering on a cdrom the entire computational environment we enjoy on our home system. For example, we only supply a subset of SEPlib executables on each cdrom (See SEPHELP cdrom).
- Read-only file system: In a read only file system, existing files are not simply overwritten. Attempting to write in such a file leads to errors which do not occur on SEP's standard file system. Please make sure that your clean rule removes all intermediate files. It would be great if someone could come up with a test for this (e.g. changing a the file protection on a documents files after cleaning it up).

- Unknown mount point: The cdrom can be mounted at very different points on very different systems. It is safe to assume that, when the cdrom is mounted at Elf in the French back country, there will be no directory called /scratch/me. A rule using such a directory is bound to fail. Instead the author should limit himself to refer only to files and directories on the cdrom and by using relative path names. Especially, since SEPlib header files point to their corresponding data files by absolute pathname, the author needs to edit the header file or to combine the header and data file into a single file (cite SEPlib). I wish I could think of a way to test this.

SUMMARY

I hope this document helps the novice when writing his first document by giving him a broad overview on the issues involved. Such a novice will still need to start by copying a working document and modifying it. This document is not intended to help him write such a document from scratch.

A seasoned SEP author should find this document helpful when testing his document. The outlined testing procedure should help the author to detect errors and problems in his document early on. I suggest every author should successfully complete these tests before attempting to include his document in a cdrom.

ACKNOWLEDGEMENT

The knowledge in this paper has accumulated over many years here at SEP. Especially, Jon Clearbout drives the development of the living document by consequently applying the outlined procedures in his books. Martin Karrenbach, Steve Cole, and Dave Nichols implemented most of the software and structure discussed here.