2018 git based workflow.

Peter Teuben, UMD

16th Dec 2018

# Introduction

These notes follow up on Keith’s excellent Aug 2017 description of the ADASS proceedings editing workflow. What we describe here are some proposed important modifications that were made to make things even easier.

The current workflow ADASS Proceedings Preparation Kit (APPK?) is in github, and a git branch is used to isolate files the editing team is modifying to tailor to their particular situation. However, with the intention that they be merged into the master before the next year team takes over. Of course the 2019 team could also branch off the 2018 branch in case the 2018 team is late. Merging might now be a bit harder.

Keith’s workflow is MacOS oriented. The 2018 team is Linux (2), Mac (1) and Windows (1) based.

E.g. ImageMagick should be advertised for easy conversion into “eps” files if authors have “jpg” or “png”.

To summarize, the main deviations from what I know as the 2015/16 workflow is:

1. We have an ADASSProceedings preparation kit in git (currently on github).
2. We have an ADASSProceedings2018 implementation in git (currently in github). Editors will need both git trees to work efficiently.
3. We use a pre-processor (tex2inc.py) that turns a standalone paper (the “tex” file the author submits) into an “inc” file that the final aspvolume will then include. The “inc” file will never be edited by us (but potentially by the ASP team), we edit the author’s “tex” file. I believe much of the work Finish.py does, is now done by tex2inc.py
4. We have integrated the ASCL index, which much like the subject index, %\ooindex{} lines
5. We keep all the bib files by themselves, there was no need to merge them into one (as far as we currently see)
6. We highly recommend a “make” based system. For this a public Makefile and author based makedefs file is used. This add one minor pain on the author to create this makedefs file, but at a huge benefit for the editors.
7. On the top level, everything is directed by self-describing hierarchical Makefiles.
8. We use the TEXINPUTS environment variable to control where figures and photos are located, so no need to edit the path in “tex” file for this, as some years had to do
9. The author contributions go into the **papers/PID** directory (where PID is something like P1-12, note we didn’t use the AuthorLastName but there is no reason this cannot be used).
10. If authors really submit the correctly named tar or zip file, they can be dumped straight into the papers/PID directory, and the ingestion process really should be 10 seconds.
11. The preprocessed “inc” file plus “bib” and “eps” figures go into one flat **authors/** directory for final book inclusion. This directory is a build directory,and no files in this directory are edited by humans.
12. To maintain the “make” environment the author uses, the editors need to use the “../pmake” script that emulated this environment from within each papers/PID directory.

# Some new scripts.

Some new scripts were developed for the 2018 ADASS:

ascl.py

tex2inc.py

The 2018 team hasn’t made use of the ADASS\_Configuration file. I propose it be a .dotfile though. Currently we use some environment variables (pmake and make use different ones), so this could be confusing and improved.

On Keith’s laptop, the configuration file looks like this:

#

#  A D A S S \_ C o n f i g u r a t i o n

#

#  Defines the location of various files used in the course of editing an

#  ADASS Proceedings volume.

#

#  Entries in this file are name/value pairs of strings, the name then the

#  value. Strings with spaces can be quoted, either using 'single' or "double"

#  quotes.

MainSubjectIndexFile "~/Trieste ADASS/Proceedings/Files/subjectKeywords.txt"

NewSubjectIndexFile "~/Trieste ADASS/Proceedings/Files/newKeywords.txt"

# Starting with a new paper

The convention for the final volume seems to be that all the files for one paper, let’s call it P14-1, by Smith, should be in a separate directory named with the paper ID (here P14-1) and the name of the first author (here Smith). So the files for the paper should be extracted into a directory called, in this case, P14-1\_Smith. (It simplifies things not to use accents in names here, and quotes and spaces are also awkward, so O’Toole could be just OToole.)

In some cases, authors have an additional layer in their directory structure. Smith might have put all their files in a directory called P14-1, for example. It makes things easier if they’re moved so all files are in the top level.

In principle, at least, at this point the files should look as they did to the author when they submitted them, and it should be possible to run LaTeX and BibTex on them without any problems. You might need to make sure the asp2014.sty and asp2014.bst files can be picked up. In some cases authors include these – and in some cases they include slightly different versions to the ones we expect! I make soft links to copies I have of these files, but there are other options that work.

You could run the PaperCheck.py script that we supply to the authors. This will pick up a number of potential problems with the paper. You might think that most authors would have run this and fixed any such problems, but that doesn’t seem to be the case. However, there seem to be many fewer serious problems this year than last, so I think it has helped. In any case, many of the problems PaperCheck.py finds are with author lists, and most of these are trivial to fix. The most awkward problems are cases where an author has supplied references as \bibitem entries, where an author has used LaTeX packages that may cause problems for the final volume – any package other than asp2014 is potentially a problem – or where graphics files are missing.

To run PaperCheck, set your default to the directory with the files for the paper, and type:

PaperCheck.py <paperID> <author>

In this case, this would be

PaperCheck.py P14-1 Smith

This should give you an idea of what problems you may be facing…

# Some problems that can be easy to fix

A lot of reference problems picked up by PaperCheck.py turn out to be because just before putting all the files in the .tar file, the author has renamed their .bib file properly as something like P14-1.bib, but has left the .tex file set up for their original name, such as \bibliography{example}

Sometimes PaperChase.py will report missing graphics files, but LaTeX will find them. This happens on Macs that are using case-insensitive file systems, where the file is called p14-1.eps but the .tex file refers to P14-1.eps. This should be fixed, because although this will typeset on a Mac, it won’t work on a Linux machine.

Sometimes graphics files or .bib files really haven’t been supplied in the .tar or .zip file, but have been supplied in an earlier version. It is possible to find the earlier versions for a paper and see if they have the files in question, but it may be quicker to contact the author.

# Unprintable characters

We have a lot of authors with names that have accents. In some cases, authors simply leave these off, which is a pity. Others will use the standard LaTeX sequences that set accented characters, such as \”{u}” for ü. Others will use extended character sets that include these accents. It is these extended character sets that can cause problems. Some LaTeX installations will handle these properly, some won’t. The version on my Mac laptop gets these wrong, and I suspect the version used by ASP will to (we’re still trying to confirm this for the Sydney Proceedings). On my laptop, these characters show as ‘unprintable’.

To play safe, there is a script, FixUnprintable.py, that runs through a .tex file looking for such unprintable characters. It knows about most of the ones used, and can modify the file to use the standard LaTeX sequences instead. Sometimes it will print a message about unexpected unprintable characters, and these need to be looked at individually.

To run it:

FixUnprintable.py filename

Where filename is the name of the .tex file in question. The script will modify the .tex file, and will save the original version, appending a numerical suffix to the file name.

I have seen one paper that used the package “accents”. I suspect this will cause problems, and I commented out the \usepackage line. I then got a number of very oddly-typeset names, which I had to fix by hand.

# Editing steps for 2018

This is my sequence, assuming the ingestion has been done, and you are located in one of the papers/PID directories. There will be the PID.tex and makedefs file, and optionally a PID.bib and some PID\_\*.eps files. Hopefully also a copyright file, but nothing more, and nothing less.

Open the copyright file and see if it really is a copyright file, and if it’s been signed properly (not all have).

../pmake check (runs PaperCheck.py)

../pmake fix (runs FixUnprintable.py)

../pmake pdf

xdg-open PID.pdf

Check that it has the right number of pages.

Check that the author list is correct and the presenting author is first.

See if there were any LaTeX errors (overfull \hbox etc, undefined references) or warnings (underful \hbox etc). (TexWorks will show you a separate sub-window with the LaTeX warnings and errors, which makes this easy.)

Make sure the running heads are correct. A lot of authors leave the title unchanged from the “Authors Final Checklist” used by the template, and many change it to the author list instead of the paper title. (The template will be improved for next year!). And make sure the title is not so long the page numbers run into the margin.

Read the paper looking for poor grammar and unclear text or illustrations.

Run a spelling checker (I just read the .tex file into Textedit and use that. I find I can ignore the errors from LaTeX directives easily enough, but a good LaTeX-aware checker would be nice. I’ve been experimenting with Excalibur, a LaTeX-aware spelling checker distributed with TexShop. but this needs to be configured for our set of LaTeX commands, and I found it slower to use than just looking at the file in Textedit.)

Fix any trivial problems, send non-trivial ones back to the author. (I define non-trivial as ones I can’t fix in less time than it takes to write a detailed e-mail explaining the problem, with an allowance for the time spent waiting for a reply etc.)

Check the names of the graphics files used – these are printed out by PaperCheck.py, – and make sure they follow the required convention. If necessary, change them.

Add %\aindex entries, one for each author, using Aindex.py

../pmake aindex

Add %\ssindex entries, as required using Index.py

../pmake index TERM=vla

Add %\ooindex entries, as suggested using ascl.py

../pmake ascl

# Fixing references

The PaperCheck.py script warns of references that aren’t defined in either a .bib file or using \bibitem. It also warns of usages such as \cite rather than \citep and \citet. And it warns of .bib and \bibitem entries that aren’t actually used, and prints out the reference IDs that the author has used. This gives a pretty good idea of the scope of any problems with references.

If the author has supplied just a few \bibitem references, they should be contacted and asked to supply a proper .bib file. For the Sydney papers, \bibitem entries were a major problem, but the new instructions to authors and the new template seem to have worked well this year, and there are very few papers with \bibitem entries this year.

\cite references need to be reworked. If the author hasn’t used ADS-style reference IDs (things like 2011ASPC..442...53A) what they have used needs to be reviewed and possibly changed, both in the .bib and the .ref.

You can always re-run PaperCheck.py after changing any references to make sure any problem has been fixed.

# ASCL index entries

This section is new. This used to be done manually by Alice Allen and Peter Teuben, and typically was ½ morning of work at the end of the whole editing process. It is better done during the editing, since the procedure is close to the other two indices.

We use the object index, \ooindex{} for the ASCL index.

ascl.py P1-12.tex

would scan the paper for potential occurences of ASCL codes, but be forwarned, there are a LOT of false positives. The line that matches the occurrence will be printed as well, making it easy to decipher if the entry is warrented. So have an editor open in another window.

The output might look as follows:

#[260] TOPCAT has many visualisation modes enabling   
  
%\ooindex{TOPCAT, ascl:1101.010}    
  
#[268] for {\it Gaia\/} data:   
  
%\ooindex{GAIA, ascl:1403.024}

In this case line 260 had a valid match, but line 268 did not. This particular case might be more appropriate for an %\ssindex{} entry.