Scripts and my editing workflow.

Keith Shortridge, AAO

23rdJan 2017

# Introduction

These notes describe the way I’ve been editing the first few of the papers assigned to me for the Trieste ADASS conference, and how I’ve been using the various Python scripts I’ve developed. This has changed from the way I did things for the Sydney Proceedings, and I think the use of the scripts is now much simpler.

For the Sydney Proceedings, the various papers were submitted as .tar or .zip files and there was quite a complicated process carried out by a script called Setup.py, which tried to identify the latest .tar or .zip file for each paper, and then extracted its contents into a directory where they could be edited. At that point it also made, automatically, a number of changes to the .tex file that would be needed before the paper was included in the final volume. For example, because the final volume had to use one single .bib file for all the papers, Setup.py renamed the .bib file for each paper to ‘adassXXVreferences.bib’ and changed the .tex file accordingly.

For the Trieste proceedings, most of the submitted papers were tagged with a timestamp which made it much easier to identify the latest submission for each paper. We have already gone through a preliminary process of collecting these latest versions, and running an initial check on each paper (using the PaperCheckBatch.py script). Also, it seems to make more sense to work on the paper as submitted by the author when we go through the editing process, rather than one that has been modified automatically. All the various modifications like changing the name of the .bib file can be left to the final stage when the individual papers are combined to form the complete volume.

Additionally, the original Sydney scripts assumed a very specific file layout. To make it easier to use them, the newer versions of the scripts make almost no assumptions about where files are. Instead, they use a configuration file in the user’s home directory that specifies where various files like the list of subject keywords can be found.

# Using the scripts.

The scripts described in this document should all be located in the same directory. At the moment, the files in question are:

Aindex.py

Index.py

AdassChecks.py

AdassConfig.py

AdassIndex.py

FixUnprintable.py

PaperCheck.py

They should have the ‘execute’ flag set, most easily done using:

chmod +x \*.py

and ideally the directory should be in your execution path, so that all you have to do is type the script name from the command line in order to run it. (The name has to include the .py, which may seem a bit awkward – you could rename the scripts to fix this.)

In your home directory, there should be a file called ADASS\_Configuration. This is a text file that tells the scripts some system-dependent information, such as where to find certain files.

At present, the only other file that the scripts need to be able to find is the file that gives the set of subject index entries used by previous ADASS conferences. There is also provision for a second file containing new entries used by papers in this conference – this is for entries where no existing entry was suitable.

On my laptop, my 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

This is my sequence.

Set my default directory containing the files for the paper.

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

If PaperCheck.py complained about unprintable characters, run FixUnprintable.py on the .tex file.

Run LaTeX on the file. (I use TeXworks on the Mac, and actually run pdflatex.)

If a .bib file has been supplied run BibTeX.

Run LaTeX a couple more times to sort out any citations.

Open the resulting PDF file (since I was running pdflatex) and print a copy (I need something to write on when I edit).

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

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

The next sections go into a bit more detail about references and adding index entries.

# 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.

# Author index entries

There ought to be one %\aindex{} entry added to each paper, just before the abstract section, in the form %\aindex{Author} where Author is in the form ‘Surname, initials’, eg %\aindex{Smith,~J.~I.}. At this stage these need to be commented out, because \aindex is not defined. They will be uncommented later.

Putting these in is quite tedious, but the Aindex.py script does most of the work. From the directory containing the files for the paper, type:

Aindex.py <paper ID>

Eg

Aindex.py P14-1

This will read the \author entry in the .tex file and print out suitable \aindex entries for each author. These can be copied and pasted into the .tex file.

The reason the script does not automatically insert the entries is because names are very difficult and very varied, and the ASP rules for the \author format are complicated and very few authors get them exactly right. This means that the Aindex.py script often is uncertain that it has the names exactly correct. Making the editor paste the results in makes it more likely that they will spot any problems. In any case, most problems that the script finds indicate some trivial problem with the format of the author list, and these should be fixed. (The rules about commas are very hard for authors to get right.) And Spanish authors with two surnames are very hard for a script to pick properly.

Aindex.py often finds something to complain about, or to ask about. For example, if the author is given in the \author list as “G. Bruce Berriman” it will ask “might Bruce Berriman be a surname?”. In this case the answer is no. If the author is given as “M. Vargas Llosa” it will ask “might Vargas Llosa be a surname?” and in this case it will be right. Mostly, though, it complains about commas.

Author lists that include things like “for the XYZ team” really confuse Aindex.py. If the string looks OK in the author list in the paper, it can stay. But you will have to remove the wrong entries that Aindex.py will generate, such as “\aindex{Team,~f.~t.~X.}”.

# Subject index entries

There ought to be a reasonable number of %\ssindex{} entries added to each paper, in the form %\ssindex{topic!sub-topic!sub-topic}. At this stage these need to be commented out, because \ssindex is not defined. They will be uncommented later. The more consistent these entries are, the better.

There is a script, Index.py, to simplify the insertion of subject index entries.

The ADASS\_Configuration file lists the locations of two files usually called subjectKeywords.txt and newKeywords.txt. The first is the list of keywords based on the previous ADASS proceedings, so it starts like this:

> more subjectKeywords.txt

adaptive optics

algorithm

analysis

artificial skepticism

A'Trous wavelet reconstruction

automatic

basket-weaving

CLEAN

ClumpFind

clustering

spatial

control

and so on.

My newKeywords.txt file contains all the subject keywords I’ve used that weren’t in that subjectKeywords.txt file. I don’t change subjectKeywords.txt file at all.

I have a simple script called Index.py that searches both these files for entries that match strings supplied on the command line. So, if, for example, you think a paper needs a subject index entry for the VLA, you can try:

> Index.py vla

Looking for index entries matching 'vla'

%\ssindex{archives!individual!EVLA}

%\ssindex{observatories!ground-based!EVLA}

%\ssindex{observatories!ground-based!VLA}

3 entries in master index

No entries in new index

What I find most useful about this script is that it supplies the strings you need to put into the text as commented-out \ssindex entries. You select the one you want in the terminal window and paste it into the .tex file. Note that it needs to go in as a separate line, otherwise having it commented is going to cause problems. It also makes it much easier to find automatically later when it has to be uncommented.

If the entry you want isn’t there, which will happen if your search string is completely new, you can use a similar string to see what similar entries have been used. Say you want an entry for the TAIPAN instrument:

> Index.py TAIPAN

Looking for index entries matching 'TAIPAN'

No entries in master index

No entries in new index

OK, nothing. But it is an instrument, so what about:

> Index.py instrument

Looking for index entries matching 'instrument'

%\ssindex{instruments}

%\ssindex{instruments!bolometer}

%\ssindex{instruments!camera}

%\ssindex{instruments!camera!submillimeter array}

%\ssindex{instruments!individual}

%\ssindex{instruments!individual!2dF}

…..

%\ssindex{instruments!individual!WFC3}

%\ssindex{instruments!individual!WFPC2}

%\ssindex{instruments!individual!X-shooter}

%\ssindex{instruments!interferometer}

%\ssindex{instruments!multibeam}

%\ssindex{instruments!polarimeter}

50 entries in master index

%\ssindex{instruments|individual|HAWK-I}

%\ssindex{instruments|individual|VIMOS}

%\ssindex{instruments|individual|VISIR}

3 entries in new index

(The real output shows many more individual instruments. There are a lot of entries for instruments!) From that, you can cut and paste one and edit it, probably as

%\ssindex{instruments!individual!TAIPAN}

of course, if you mean TAIPAN the survey, that’s different.

# Finishing a paper

Once the editing for a paper is complete, a number of things need to be done. Those \aindex and \ssindex entries need to be uncommented, and a number of other items such as \begin{document} and \end{document} need to be commented.

What’s more, the entries in the .bib for this paper need to be added to the master version of this file to be used for the final volume, and (amongst other things, for the purposes of the Index.py script) the new subject entries need to be added to the newKeywords.txt file.

For Sydney, this was done using a script called Finish.py. This still needs to be reworked for the Trieste papers, but doesn’t really need to be run until we are much closer to producing the final volume.