Using waf to produce a tarball for submission to the arXiv.

BV

[2014-03-08 Sat 14:44]

This topic discusses using `waf` to produce a tarball from a waf-built LaTeX source area which is suitable for submission to the arXiv.

1 Overview

Waf has good built-in support for building LaTeX documents. When submitting to the arXiv one has to prepare a tarball that contains just the files needed to produce the document and little more.

Waf also has `waf dist` to produce a tarball of the "source" files. When I tried this in a LaTeX source area it tarred up everything. Unfortunately the document in question has a lot of extraneous files in the working directory (and repository) that should not be included. This is particularly important as the arXiv has a size limit on the submitted files.

I asked about it on waf's google groups and got some tips to try out. Waf has to figure out all the files the document is built from in order to know when they change and a rebuild is needed. So, a big part of the required files should be able to be automatically determined.

Testing this inside the desired document directory is dog slow so I mocked up a little test document here. This test document is a little unusual in order to mimic some "features" of the actual document. In particular, two "main" files are required: `./document.tex` and `./document-bib.tex`. The first includes the Bibitex `.bbl` file produced as a side effect of building the second. The two are otherwise identical. These shenanigans are needed because arXiv does not run Bibtex for us and so we need a way to generate it and produce a main which is independent from Bibtex.

The method I ended up adopting eschews `waf dist` so that I can do some massaging of the paths in the produced tar ball. In particular I want

to strip off the `build/` directory for files that are created during the build. This flattening is only needed because arXiv runs `pdflatex` in the source directory and obviously doesn't know about waf nor its conventions.

The only reason I wanted to use `waf dist` was in hopes I could leverage `waf distcheck` to check that the tarball is complete. However, `distcheck` seems to ignore the `dist()` function and packages up the entire source directory (default `dist` behavior). Something to follow up on.

2 The `wscript` file

This section annotates the example `wscript` file. It starts by defining some conventional variables that waf normally uses for `dist` and hooking in the `tex` tool to do the heavy lifting building the document.

```
#!/usr/bin/env python

APPNAME = "document"
VERSION = '0.0.0'

def options(opt):
    opt.load('tex')
    return

def configure(cfg):
    cfg.load('tex')
    cfg.env.append_value('PDFLATEXFLAGS','-halt-on-error')
```

Next comes the `build()` function. It first defines two task generators, one for each main document. The first is done just for building the `.bbl` file. It ends in an `add_group()` in order to assure this file is built. Next comes the task for the "real" document. Some nodes are defined for use there and in the final task generator which is for making the distribution tarball.

```
def build(bld):

    bbl_tex = 'document-bib.tex'
    bld(features = 'tex',
type = 'pdflatex',
source = bbl_tex,
outs = 'pdf',
```

```
prompt = 0)

    bld.add_group()

    extensions = ['.tex','.pdf','.manifest','-%s.tar.gz'%VERSION]
    nodes = [bld.path.find_or_declare(APPNAME+ext) for ext in extensions]
    tex, pdf, man, tar = nodes

    bld(features = 'tex',
type = 'pdflatex',
source = tex,
outs = 'pdf',
prompt = 0)

    bld(rule = tarball,
source = [man, tex],
target = tar,
prefix = APPNAME +'-'+ VERSION + '/', # needs trailing "/"
extra = '**/document-bib.bbl ' + str(tex)
    )
```

The `prefix` argument defines to this last `bld()` call is a string tacked on to each file as it's stored into the tarfile. The `extra` parameter is a list of Ant globs to match files that are not otherwise picked up by waf's tex scanner. These have to be determined by trial an error. They must remain as unresolved globs as the files they match may not exist until after the document has been built. Files that match under the `build/` directory are okay as described below.

The `tarball` rule invoked above is defined as:

```
import tarfile
def tarball(task):
    bld = task.generator.bld
    prefix, extra = task.generator.prefix, task.generator.extra

    globs = task.inputs[0].read() + ' ' + extra
    nodes = bld.path.ant_glob(globs)

    tfname = task.outputs[0].abspath()
    ext = os.path.splitext(tfname)[1][1:]
```

```
    with tarfile.open(tfname, 'w:'+ext, ) as tf:
for node in nodes:
    tar_path = node.nice_path()
    if node.is_bld():
tar_path = node.bldpath()
    tf.add(node.nice_path(), prefix + tar_path)
```

Not much to say here. The globs are resolved and added to the tar file one by one. Any node which is under the build directory is stored in to the tar file at a location relative to the build directory. Otherwise a path relative to the source directory is used.

Next comes the heart of the glue into waf's "tex" feature. Two methods are defined. The first creates a task implemented by the second in such a way that all scanned files get passed.

```
import os
from waflib.TaskGen import feature, after_method
@feature('tex')
@after_method('apply_tex')
def create_another_task(self):
    tex_task = self.tasks[-1]
    at = self.create_task('manifest', tex_task.outputs)
    doc = tex_task.outputs[0]
    man = os.path.splitext(str(doc))[0] + '.manifest'
    man_node = self.bld.path.find_or_declare(man)
    at.outputs.append(man_node)
    at.tex_task = tex_task
    # rebuild whenever the tex task is rebuilt
    at.dep_nodes.extend(tex_task.outputs)
```

The second turns those scanned files into a "manifest" file.

```
from waflib.Task import Task
class manifest(Task):
    def run(self):
man_node = self.outputs[0]
self.outputs.append(man_node)
idx = self.tex_task.uid()
nodes = self.generator.bld.node_deps[idx]
with open(man_node.abspath(), 'w') as fp:
    for node in nodes:
fp.write(node.nice_path() + '\n')
```

3 Building

The document is built in the usual manner

```
waf configure build
```

4 Testing

As mentioned the `waf distcheck` doesn't seem to honor the tar file produced by `waf dist` and anyways this approach does not use `dist`. So, testing is done by "hand".

```
tar -xvf build/document-0.0.0.tar.gz
pushd document-0.0.0/
pdflatex document.tex > /dev/null
pdflatex document.tex > /dev/null
ls -l
popd
```

```
document-0.0.0/document-bib.bbl
document-0.0.0/document-0.0.0/document-bib.bbl
document-0.0.0/document.tex
document-0.0.0/figs/waf-logo.png
document-0.0.0/main.tex
document-0.0.0/preamble.tex
~/org-pub/topics/waf-latex-arxiv/document-0.0.0 ~/org-pub/topics/waf-latex-arxiv
total 104
drwxr-xr-x 2 bv bv  4096 Jan 23 18:16 document-0.0.0
-rw-r--r-- 1 bv bv   281 Jan 23 18:16 document.aux
-rw-r--r-- 1 bv bv   410 Mar  8  2014 document-bib.bbl
-rw-r--r-- 1 bv bv  7147 Jan 23 18:16 document.log
-rw-r--r-- 1 bv bv 67438 Jan 23 18:16 document.pdf
-rw-r--r-- 1 bv bv   150 Mar  8  2014 document.tex
drwxr-xr-x 2 bv bv  4096 Jan 23 18:16 figs
-rw-r--r-- 1 bv bv   184 Mar  8  2014 main.tex
-rw-r--r-- 1 bv bv    42 Mar  8  2014 preamble.tex
~/org-pub/topics/waf-latex-arxiv
```