R Markdown Makefile Examples

Overview

In order to streamline the development of course content, a lecture instructor may wish to have one source R Markdown file to create both a verbose lecture version of content for distribution as well as a more compact (sparse) version for presentation or notes. From this single source file, both version of the document can be generated using conditional statements.

This document gives a brief introduction to the technologies used to implement conditional evaluation of content in R Markdown files using the Generic Preprocessor. GNU Make can also be used as a wrapper for the preprocessor to make the process more convenient.

After these technologies are outlined, a brief example is included to demonstrate the process.

All code shown is being run in a terminal.

Generic Preprocessor (GPP)

The Generic Preprocessor (`GPP`) is analogous to the preprocessors used to compile programming languages. `GPP` is designed to extend the functionality of preprocessors to act on any text file, and copies a text file to another making systematic changes or replacements defined by a variety of macros.

While `GPP` is extremely powerful, we are currently restricted our use to conditionally including content to be used in lecture content. We can use the `#ifdef` conditional macro to include or exclude content from R Markdown source files (to be compiled into HTML/pdf/etc).

It should be noted that the `GPP` program has not been updated since March 2007. While we may eventually want to consider moving to a more modern system, the current version seems to be working well and does everything we need. More information can be found at the developer's site.

Conditional Statements

The `#ifdef` macro checks to see if some variable has been defined (either in another macro or in the command used to run `GPP`. If the variable has been defined, content included within the start and end tags of the macro will be copied into the output text file. If it has not been defined, the content is left out.

```
<#ifdef verbose>
Content to be conditionally included
<#endif>
```

The included makefile and example code uses the variable `verbose` as the flag. You may wish to use whatever variable name you want, and the makefile may be extended eventually to accept an argument for this variable name.

Using GPP

`GPP` is a command line utility for Linux systems. While there are many flags that control its functionality, we will only be using a few:

- -o is used to specify the output file
- -H is used to specify that we are using the HTML-like syntax for macros
- -D is used to define the variable checked for by `#ifdef`

Using these flags, `GPP` is run as follows:

```
gpp -H -D verbose=1 -o outfileLecture.Rmd infile.Rmd
```

The preceding command takes some the input file `infile.Rmd`, which includes the previously described `#ifdef` statements, and creates the output file `outfileLecture.Rmd` keeping the verbose content. It should be noted that *what* you define `verbose` as is arbitrary. `#ifdef` only checks to see *that is has been defined.*

To create the sparse version:

```
gpp -H -o outfileNotes.Rmd infile.Rmd
```

From here, either of these output files can be processed as you would typically process a .Rmd file (using RStudio's utilities or compiling manually with `knit` and `pandoc`).

For more information about `GPP`, you can use the flag `--help` to get a brief overview of the other flags or `man` to access the full manual.

```
gpp --help
man gpp
```

The makefile

To make the process more convenient, a simple makefile was written that takes an input file with conditional statements and uses various rules to process the file with `GPP` and compile it with calls to `R` and `pandoc`.

In the simplest case, the makefile is in your working directory, and you want to all possible output types. In this case, you only need to specify the input file using the source flag:

```
make source=infile.Rmd
```

Target Rules

However, you can specify what types of output file you want using different target rules.

```
make source=infile.Rmd target_rule
```

The target rules are:

Verbose Rules	Notes Rules	Description
`verbose_all`	`sparse_all`	All output types for long or short version files.
`verbose_source`	`sparse_source`	Only the .Rmd file
`verbose_knit`	`sparse_knit`	The .Rmd and knitted .md file
`verbose_doc_html`	`sparse_doc_html`	An HTML document (single page)
`verbose_doc_pdf`	`sparse_doc_pdf`	A pdf document
`verbose_slides_pdf`	`sparse_slides_pdf`	A PDF slideshow
`verbose_slides_html`	`sparse_slides_html`	An HTML slideshow (Slidy)

The pdf and HTML output depend on the .Rmd and .md files, so `_source` and `_knit` are run no matter what the target is and these files are produced. They are included as named target rules in case you are only interested in these output files (for distribution, etc.).

All of the output files are named according to whether they are long (verbose) or short (sparse) output files. For example, if you had the input file `foo.Rmd` and compiled with the target rule `verbose_doc_pdf`, three files are produced: `fooLong.Rmd`, `fooLong.md`, and `fooLong.pdf`.

Of course, just because the makefile *can* make that output doesn't mean that it should. For example, if you don't have reasonable page breaks defined in the source file (with `## Heading` for titled slides or `---` for untitled slides) and tried to compile an HTML or pdf slideshow, `pandoc` will not know how to split content into slides.

Compiling from Nested Directories

So far, we have only talked about cases where the makefile is in your working directory. However, if you wanted to have a hierarchy of directories for your content, you wouldn't want to copy the makefile into every section/chapter/course subdirectory. A more ideal situation would be to have the makefile in a parent directory and reference it from your subdirectories when compiling.

GNU Make allows you to do this with the `-f` flag. For example, if you were writing content for Section 3 of Chapter 2 from the course Foo-101, and you organized your files by semester, you might be in the working directory `~/Teaching/F2015/Foo-101/C02/S03/`, you can keep a single makefile in `~/Documents/Teaching/`.

```
make source=chap02S03.Rmd notes_doc_html -f ~/Teaching/makefile
```

The resulting output files will be contained in your working directory (the directory from which you called Make).

Example

Included with this document is the subdirectory `example`, which contains a source file `example.Rmd`. This file contains some small examples of content that can be conditionally evaluated. From here forward, it is assumed that `example` is your working directory and the make file is in the immediate parent directory.

GPP Example

First, we will call `GPP` directly and create the .Rmd containing the lecture (verbose) content.

```
gpp -H -D verbose=1 -o exampleLecture.Rmd example.Rmd
```

Using the Makefile

Compile the PDF slideshow (Beamer) using the makefile:

```
make source=example.Rmd verbose_slides_pdf -f ../makefile
```

Running this results in the following files being created: `exampleLong.Rmd`, `exampleLong.md`, `exampleLongSlides.pdf`.

Compile the HTML Document:

```
make source=example.Rmd lecture_doc_html -f ../makefile
```

Note that the only new file created is `exampleLong.html`, since `exampleLong.Rmd` and `exampleLong.md` already exist.

Compile all note file types:

```
make source=example.Rmd notes_all -f ../makefile
```

Note that seven new files have been created: `exampleShort.Rmd`, `exampleShort.md`, `exampleShort.html`, `exampleShort.pdf`, `exampleShortSlides.html`, and `exampleShortSlides.pdf`.

Modifying the makefile

As it exists, the output file names are defined by the following pattern substitutions defined at the top of the makefile:

```
sparse = $(patsubst %.Rmd, %Short, $(source))
verbose = $(patsubst %.Rmd, %Long, $(source))
```

If, for example, you wanted the sparse file names to have `Notes` appended instead of `Short`, and `Lecture` instead of `Long`, you can modify these lines to be:

```
sparse = $(patsubst %.Rmd, %Notes, $(source))
verbose = $(patsubst %.Rmd, %Long, $(source))
```

It is also possible to define new targer rules that combine other target rules. For example, if your typical output is HTML documents for both the verbose and sparse versions, you can create a new rule like:

```
html_docs: verbose_doc_html sparse_doc_html
```

and compile with:

```
make source=infile.Rmd html_docs -f /pathtomakefile/.../makefile
```