

A makefile for L^AT_EX, Sweave, and Beamer

Charlie Gibbons*
University of California, Berkeley

September 3, 2010

A makefile is a simple file that directs the compilation of a target file. Here, we create a makefile that compiles a simple `tex` file to PDF, Sweave a `Rnw` file to PDF, compile a Sweave Beamer document that produces multiple different handout options, and compile a `tex` file like a problem set or exam, producing both a handout version and a solutions set.

1 A basic makefile

To start the makefile, we provide a set of macros that allow this file to be highly flexible. The inputs here are the only aspects of the makefile that the user needs to change.

```
TEXFILE=Sample
PLOTDIR= ./Plots
PLOTS=           # List plots
FIGURES= $(PLOTDIR)/$(PLOTS)
INPUTS=         # List TeX includes
```

These macros require the user to specify the file names that should be used in compilation. The first specifies the `tex` or `Rnw` file to be used in compilation and the others specify additional files. If any of these files are updated, a call to the make utility updates the target document (*i.e.*, the PDF output).

- `TEXFILE` specifies the name of the `tex` or `Rnw` file, excluding its suffix.
- `PLOTDIR` specifies the folder where plots are located. This is not necessary if no figures are included or if those figures are created by Sweave.¹
- `PLOTS` accepts a list of plot names to be included in your document including suffixes (*e.g.*, `.pdf`). Again, this is not necessary if there are no figures or the plots are produced by Sweave.
- `FIGURES` uses the previous two lines and does not need to be changed by the user.
- `INPUTS` accepts a list of other `tex` files that are used as inputs to the main `tex` file.²

We add two dummy lines that indicate what the make utility needs to do.

*Web: <http://cgibbons.berkeley.edu>. E-mail: cgibbons@econ.berkeley.edu

¹The only reason to list the plots is to ensure that, if they are updated, the target document is updated. If the plots are created via Sweave, the target PDF is updated to reflect both the updated `tex` file and the updated plots.

²Input files are assumed to be in the same folder as the makefile. If the user has a different folder structure, then the same approach as used for figures can be applied here. In such a case, `INPUTS` would be analogous to `FIGURES` and the additional necessary macros can be defined.

```
all: $(TEXFILE).pdf; make clean
```

```
.PHONY: all clean
```

The first specifies the final outcome of the compilation. The value `clean` indicates that the file removes incidental files created by the `pdflatex` process.

Next, we compile the document:

```
$(TEXFILE).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
# Initial run
pdflatex $(TEXFILE)

# Run BibTeX if missing citations
@if(grep "Citation" $(TEXFILE).log > /dev/null);\
then \
    bibtex $(TEXFILE);\
    pdflatex $(TEXFILE); \
fi

# Recompile if instructed
@if(grep "Rerun" $(TEXFILE).log > /dev/null);\
then \
    pdflatex $(TEXFILE); \
fi
```

The first line indicates that a PDF file is to be made from a `tex` file plus figures and other inputs. If any of these files has a timestamp more recent than the PDF file, then the subsequent compilation steps take place.

An initial run of `pdflatex` compiles the document. The next block of text examines the log file produced by this compilation and looks for the phrase “Citation”, which indicates that `bibtex` needs to run to identify missing citations. Then, an additional run of `pdflatex` incorporates these missing citations. The last block of text also searches the log file, here looking for an instruction to rerun `pdflatex` to ensure that cross-references are correct. *Note that all indents are generated using tabs, not spaces.*

If the `tex` file is created by using Sweave in R, then the following lines are necessary:

```
$(TEXFILE).tex: $(TEXFILE).Rnw
R CMD Sweave $(TEXFILE).Rnw
R CMD Stangle $(TEXFILE).Rnw
```

A new `tex` file is produced if the timestamp on the `Rnw` file is more recent than that on the `tex` file. These lines produce both the `tex` file from the `Rnw` file and a `.R` file that gives the R code that was run by Sweave.³

Lastly, we remove the intermediate files used to generate the PDF output. Note that, if you have other files with these extensions that you don’t wish to delete, (like Stata log files), you might remove the extension from the list or indicate the files that you do want removed more clearly (*e.g.*, by listing `$(TEXFILE).log`).

³The resulting `$(TEXFILE).R` file can be included in a `tex` file using the `includeverbatim` command (as an appendix, for example).

```
## Remove unnecessary files
clean:
    -rm -f *.log *.aux *.out *.blg *.bbl *.nav *.snm *.toc
```

2 Using the Beamer class to create slides and handouts

A makefile can be used to create the presentation slides, a handout version of these slides that removes the pauses in the presentation version, and a notes version that places three slides on a single page with space for notes.

2.1 The makefile

Much of the code in this section simply expands on that found in the simple case. First, we add additional macros:

```
TEXFILE=Sample
HANDOUT=SampleSlides
NOTES=SampleNotes
PLOTDIR= ./Plots
PLOTS=                # List plots
FIGURES= $(PLOTDIR)/$(PLOTS)
INPUTS=                # List TeX includes
```

The additional set of macros allows the user to specify the file names for handout files compiled under the Beamer class.

- `HANDOUT` specifies the name of a set of slides that result from calling the `beamer` class with the `handout` option. This file consists of full-sized slides, but without any of the pauses that might exist in the presentation version.
- `NOTES` specifies the name of a handout that places three slides per page with lines for notes.⁴

We change the dummy line to note that two additional files are to be compiled:

```
all: $(TEXFILE).pdf $(HANDOUT).pdf $(NOTES).pdf; make clean
```

We add an additional line preceding the compilation step:

```
$(TEXFILE).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
    echo "\ishandoutfalse \isnotesfalse">HandoutTest.tex
    # Initial run
    pdflatex $(TEXFILE)
```

The second line creates a new file containing a line of `TEX` code. This file is an input to the `tex` file (see Section 2.2) indicating that this compilation step does not create a set of slides using the handout class

⁴As we note later, the Beamer functionality requires placing a set of directives in the header of the Beamer file. Other orientations of this set of notes (*e.g.*, two slides per page, no lines for notes) are available. See documentation for the `handoutWithNotes` package, the required package for this functionality.

(\ishandoutfalse) and is not the three-slides-per-page note format (\isnotesfalse). The remainder of this code chunk is unchanged from that given above.

The code chunk that produces slides using the `handout` option in the `beamer` class by setting `\ishandouttrue` is:

```
## Handout slides compilation
$(HANDOUT).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
    echo "\ishandouttrue \isnotesfalse">HandoutTest.tex
    # Initial run
    pdflatex -jobname $(HANDOUT) $(TEXFILE)

    # Run BibTeX if missing citations
    @if(grep "Citation" $(HANDOUT).log > /dev/null);\
    then \
        bibtex -jobname $(HANDOUT) $(TEXFILE);\
        pdflatex -jobname $(HANDOUT) $(TEXFILE); \
    fi

    # Recompile if instructed
    @if(grep "Rerun" $(HANDOUT).log > /dev/null);\
    then \
        pdflatex -jobname $(HANDOUT) $(TEXFILE); \
    fi
```

Note that we specify the name of the `pdflatex` output using the `-jobname` option.

Similarly, the code to produce the notes file sets `\ishandoutfalse` and `\isnotesttrue`:

```
## Notes compilation
$(NOTES).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
    echo "\ishandoutfalse \isnotesttrue">HandoutTest.tex
    # Initial run
    pdflatex -jobname $(NOTES) $(TEXFILE)

    # Run BibTeX if missing citations
    @if(grep "Citation" $(NOTES).log > /dev/null);\
    then \
        bibtex -jobname $(NOTES) $(TEXFILE);\
        pdflatex -jobname $(NOTES) $(TEXFILE); \
    fi

    # Recompile if instructed
    @if(grep "Rerun" $(NOTES).log > /dev/null);\
    then \
        pdflatex -jobname $(NOTES) $(TEXFILE); \
    fi
```

2.2 Preamble code

As mentioned, this implementation requires that a `tex` file created by `make` be input to the base `tex` file. The first lines of the base `tex` file need to be:

```
\newif\ifishandout
\newif\ifisnotes
\input{HandoutTest}
\ifishandout
  \documentclass[xcolor=dvipsnames, t, handout]{beamer}
\else
  \documentclass[xcolor=dvipsnames, t]{beamer}
\fi
\usepackage{handoutWithNotes}
\ifisnotes
  \pgfpagesuselayout{3 on 1 with notes}[letterpaper, border shrink=5mm]
\else
\fi
```

This code establishes the `\ishandout` and `\isnotes` commands and the associated if statements. The file generated by `make`, `HandoutText.tex` is also included. The user can change the `beamer` class options (leaving the `handout` option as is) as well as the `handoutWithNotes` options, the latter following the `handoutWithNotes` package documentation at the `\pgfpagesuselayout` line.

3 Problem sets and exams

We can use a single `tex` file to hold problem set or exam questions and the solutions and use a `makefile` to produce a plain version and one that includes solutions.

3.1 The makefile

The framework is analogous to that of the Beamer implementation above.

```
TEXFILE=Sample
SOLUTIONS=SampleSolutions
PLOTDIR= ./Plots
PLOTS=
FIGURES= $(PLOTDIR)/$(PLOTS)
INPUTS=          # List TeX includes

all: $(TEXFILE).pdf $(SOLUTIONS).pdf; make clean

.PHONY: all clean

## Problem set or exam compilation
$(TEXFILE).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
```

```

echo "\excludeversion{Solution} \includeversion{Assign}">SolutionsTest.tex
# Initial run
pdflatex $(TEXFILE)

# Run BibTeX if missing citations
@if(grep "Citation" $(TEXFILE).log > /dev/null);\
then \
bibtex $(TEXFILE);\
pdflatex $(TEXFILE); \
fi

# Recompile if instructed
@if(grep "Rerun" $(TEXFILE).log > /dev/null);\
then \
pdflatex $(TEXFILE); \
fi

## With solutions compilation
$(SOLUTIONS).pdf: $(TEXFILE).tex $(FIGURES) $(INPUTS)
echo "\excludeversion{Assign} \includeversion{Solution}">SolutionsTest.tex
# Initial run
pdflatex -jobname $(SOLUTIONS) $(TEXFILE)

# Run BibTeX if missing citations
@if(grep "Citation" $(SOLUTIONS).log > /dev/null);\
then \
bibtex -jobname $(SOLUTIONS) $(TEXFILE);\
pdflatex -jobname $(SOLUTIONS) $(TEXFILE); \
fi

# Recompile if instructed
@if(grep "Rerun" $(SOLUTIONS).log > /dev/null);\
then \
pdflatex -jobname $(SOLUTIONS) $(TEXFILE); \
fi

#$(TEXFILE).tex: $(TEXFILE).Rnw
# R CMD Sweave $(TEXFILE).Rnw
# R CMD Stangle $(TEXFILE).Rnw

## Remove unnecessary files
clean:    # May want to keep *.bbl if lots of citations
-rm -f *.log *.aux *.out *.blg *.bbl *.nav *.snm *.toc SolutionsTest.tex

```

The lines needed for Sweave are shown commented out; remove the # for use with a Sweave document. Note that the intermediate file, `SolutionsTest.tex` is removed during the clean operation.

3.2 Preamble code

Here, too, code needs to be added to the standard preamble to produce the desired results. The following can appear anywhere in the preamble:

```
\usepackage{versions}  
\input{SolutionsTest}
```

The `versions` package allows two versions of a `tex` output to be held in one file. We create two versions—Assign (for the actual assignment) and Solution (for the solutions). The `\includeversion{}` and `\excludeversion{}` commands placed in `SolutionsTest.tex` hide and reveal the appropriate parts of the document.

When writing the document, the solution to a question needs to be enclosed in the `Solutions` environment that the `versions` package creates:

```
\begin{Solution}  
New Hampshire was the ninth state to ratify the U.S. Constitution, upon which the document  
achieved the two-thirds majority needed for adoption.  
\end{Solutions}
```

Code can also be included only in the plain version using the `Assign` environment. For example, vertical space can be added for students to answer the question:

```
\begin{Assign}  
\vspace*{1.25in}  
\end{Assign}
```

Any code in the `Solutions` environments is only included in the solutions file and any code in the `Assign` environments only appears in the plain version.

4 Suggested usage

The makefile should be called `makefile`. It is executed by issuing the terminal command `make` from the directory where the makefile is located. Macro values can be passed via the command line, `make TEXTFILE=Sample`, for example.

A user can add the following lines to his `.emacs` file in his home directory:

```
(global-set-key (kbd "C-c m")  
                (lambda ()  
                  (interactive)  
                  (save-buffer)  
                  (shell-command (concat "make")))))
```

If a user types `Ctrl-c m` from within the `tex` or `Rnw` file used for compilation, the file is saved, the `make` utility is run, and the target documents are produced.

In passing, it should be noted that a downside to using makefiles is that errors are more difficult to identify.