Make and Makefiles:
A Quick Introduction

Ross Ihaka

What Make Does

The make *program* automates the procedure of making new files out of older ones. It was originally designed for building programs from source code, but it is useful for any process in which older files are used to produce newer ones.

A simple example of a process which can be automated is the process of building a PDF file from an Sweave document. If the Sweave document is called `mydoc.Rnw`, the the steps are as follows[1].

```
R CMD Sweave mydoc.Rnw
R CMD pdflatex mydoc.tex
R CMD pdflatex mydoc.tex
```

Once the process has been automated with `make` these commands can be replaced with a simpler command.

```
make
```

How Make Works

Make works by using a set of *rules* which describe how one group of files can be used to produce another *target* file. The rules are entered into a file called either `Makefile` or `makefile`. The process above could be described by a file containing the following rule.

```
mydoc.pdf: mydoc.Rnw
        R CMD Sweave mydoc.Rnw
        R CMD pdflatex mydoc.tex
        R CMD pdflatex mydoc.tex
```

The first line of the rule describes the dependency. In this case, the target `mydoc.pdf` depends on `mydoc.Rnw`. If the target is older than than the files it depends on, then it needs to be remade. The remaining lines of the rule give the commands which must be run to bring the target up to date[2].

---

[1] pdfLaTeX is run twice in case any cross-references have changed.

[2] Note that the 2nd through 4th lines in the makefile all start with a TAB character. Normally Emacs does not insert TABs into files, but in the case of makefiles it does.

Multiple Rules

Now let's make the example more complicated. We suppose that in addition to working on the Sweave file `mydoc.Rnw` we're also making changes to the `Sweave.sty` file to customise the way the output looks. We could just add an extra dependency to make this work as follows.

```
mydoc.pdf: mydoc.Rnw Sweave.sty
        R CMD Sweave mydoc.Rnw
        R CMD pdflatex mydoc.tex
        R CMD pdflatex mydoc.tex
```

When only the `Sweave.sty` file is changed, this rule does more work than is necessary; the command `R CMD Sweave mydoc.Rnw` does not need to be run in this case. The following makefile provides a better alternative.

```
mydoc.pdf: mydoc.tex Sweave.sty
        R CMD pdflatex mydoc.tex
        R CMD pdflatex mydoc.tex

mydoc.tex: mydoc.Rnw
        R CMD Sweave mydoc.Rnw
```

This makefile breaks the process into two steps. One deals with converting the `.Rnw` file into a LaTeX file, the other with converting the LaTeX file into a PDF file. The target of the first rule in the makefile is the the target which is to be constructed by make. The other rules are used to help to bring that target up to date[3].

Dummy Rules

Sometimes it is useful to have rules which only describe dependencies and don't have any associated commands. The following Makefile describes how to make two documents.

```
all: mydoc1.pdf mydoc2.pdf

mydoc1.pdf: mydoc1.tex Sweave.sty
        R CMD pdflatex mydoc1.tex
        R CMD pdflatex mydoc1.tex

mydoc1.tex: mydoc1.Rnw
        R CMD Sweave mydoc1.Rnw
```

---

[3]It is possible to specify other targets for the make process on the command line. For example, the command "`make mydoc.tex`" and be used to create the LaTeX file without going all the way to PDF.

```
mydoc2.pdf: mydoc2.tex Sweave.sty
        R CMD pdflatex mydoc2.tex
        R CMD pdflatex mydoc2.tex

mydoc2.tex: mydoc2.Rnw
        R CMD Sweave mydoc2.Rnw
```

Meta-Rules

There is a great deal of duplication in the makefile above. This duplication can
be removed by the use of *meta-rules*. Rather than specifying how to carry out
conversion of specific files, meta-rules specify how to convert one kind of file to
another. The following meta-rule specifies how to convert `LaTeX` files into PDF
files.

```
.tex.pdf:
        R CMD pdflatex $*.tex
```

The rule says that to convert a file with suffix `.tex` into one with a `.pdf` suffix
requires running `pdflatex` on the `.tex` file. (Note that the `$*` is understood to
be the non-suffix part of the file name.)

In order for meta-rules to work, it is necessary to list all the suffixes that
make will work with. To make the rule above work, the makefile must contain
the line

```
.SUFFIXES: .tex .pdf
```

Using meta-rules we can write the previous makefile much more simply.

```
.SUFFIXES: .tex .pdf .Rnw

all: mydoc1.pdf mydoc2.pdf

mydoc1.pdf mydoc2.pdf: Sweave.sty

.Rnw.tex:
        R CMD Sweave $*.Rnw

.tex.pdf:
        R CMD pdflatex $*.tex
        R CMD pdflatex $*.tex
```

More `.pdf` targets are now very easy to add.

Abbreviation

One final feature of *make* which is useful is the ability to abbreviate. The
following abbreviation makes it possible refer to all the target PDF files with
one name.

```
.SUFFIXES: .tex .pdf .Rnw

PDFS= mydoc1.pdf mydoc2.pdf

all: $(PDFS)

$(PDFS): Sweave.sty

.Rnw.tex:
        R CMD Sweave $*.Rnw

.tex.pdf:
        R CMD pdflatex $*.tex
        R CMD pdflatex $*.tex
```

Abbreviation declarations can be spread over several lines by escaping the line ends with "\." The abbreviation above could be written as follows.

```
PDFS= mydoc1.pdf \
      mydoc2.pdf
```

This can be used to make the makefile easier to read and more maintainable.