Learn how to write a `Makefile`

Berend de Boer – eurotex 2001

introduction

Makefiles codify the following knowledge:

1. How to build: the specification how things need to be processed, i.e. the command to turn a `.tex` document into a `.pdf` document.

2. When to build: they have the ability to only build the things that have been changed. That's what makes them different from ordinary shell scripts.

structure of a Makefile

Makefiles are usually stored in a file called `Makefile`. Or they have the extension .mk.

The basic structure of a makefile is:

```
1   target: prerequisites ...
2       command
3       ...
4       ...
```

basic ConTEXt makefile

Before delving into the details, first an example of a ConTEXt makefile:

```
1   PRODUCT=test.pdf

2   .PHONY: clean

3   default: $(PRODUCT)

4   $(PRODUCT): test.tex graphic_a.1 graphic_b.1

5   %.pdf: %.tex
6           texexec $<

7   %.1: %.mp
8           mpost $<

9   clean:
10          texutil --purge
```

what Make to use

What make tool should I use, or simply, where can I get it?

The answer is: GNU make. Comes with almost any system. It has a free manual. BSD make might do, but is somewhat less powerful. The advanced options are not compatible.

Windooze users should get cygwin from `http://sources.redhat.com/cygwin/index.html`. Don't be tempted to use brain dead makes from Microsoft, Borland, or anyone else.

targets

A target is a file. A target is a file. A target is a file.

```
1  myfile.pdf: myfile.tex
2    texexec myfile.tex
```

Did I already say that a target is a file?

Makefiles can have phony targets (none files).

```
1   default: all

2   all: myfile.pdf

3   ...
```

If a phony target happens to be a file, you're out of luck. Really save phony targets are written as:

```
1   .PHONY: default
2   .PHONY: all clean

3   ...

4   clean:
5     texutil --purge
```

Make accepts as parameters names of targets:

```
1  make clean
2  make all
```

If no target is given, the first target is the default target:

```
1  .PHONY: all default clean

2  default: all

3  all: myfile.pdf

4  clean:
5    texutil --purge
```

prerequisites

A prerequisite is anything a target depends on:

1. Source file (files that cannot be generated).
2. Other targets.

`make` will build targets in dependency order.

```
1   myfile.pdf: myfile.tex graphic.1
2     texexec myfile.tex

3   graphic.1: graphic.mp
4     mpost graphic.mp
```

Wildcards are supported:

```
1  myfile.pdf: *.tex
2    texexec myfile.tex
```

commands

A command is always passed to your shell. A command is therefore anything (or limited too something) your shell understands. You will detect that `command.exe` (or `cmd.exe`) is extremely limited. If you have download Cygwin, you have a complete Unix shell.

WARNING: a command is always preceded by a TAB character.

You can use variables in commands. `$<`is the name of the first prequisite.

```
1   myfile.pdf: myfile.tex
2     texexec $<
```

The target is also available in `$@`

```
1  myfile.pdf: myfile.tex
2    texexec $< --result=$@
```

your turn

Help me to create a basic `Makefile`.

variables

You can use variables to make sure things are defined only once:

```
1   SRC=a.tex b.tex

2   a.pdf: $(SRC)
3     texexec a.tex

4   b.pdf: $(SRC)
5     texexec b.tex
```

Variables can be used anywhere: in targets, prerequites or commands.

implicit rules

It's annyoing to specify you want to run `texexec` for TEX files if you always want to do that. With implicit rules you can specify such things:

```
1   %.pdf: %.tex
2           texexec $<

3   %.1: %.mp
4           mpost $<
```

A '`%`' is a template character, a kind of wildcard.

Tips and Tricks

1. Do not directly generate the target with texexec and such: if they fail, `make` thinks they're updated when you do the next run.

```
1  myfile.pdf: myfile.tex
2    texexec $<x --result=temp.pdf
3    mv temp.pdf $@
```

2. Spaces in filenames or directories: make sure you don't have them.

3. You can include other Makefiles, for example you can include your TEX implicit rules with:

```
1  myfile.pdf: myfile.tex

2  include tex.mk
```

big example

```
1       #
2       # Makefile used with sql2context.tex
3       #
4       # tests/converts/compiles various files

5       .PHONY: all clean validate archive
6       .SUFFIXES:
7       .SUFfIXES: .xml .csv .sql .ib .ddl .txt .pdf .tex .ibdt .db2

8       # main target

9       all: sql2context.pdf

10      # my document

11      DOCSRC = \
12             sql2context.tex \
13             example.xml example.dtd \
```

```
14          flintstones.ddl flintstones.csv flintstones.xml \
15          csv2xml.pl ib2xml.pl \
16          select1.sql select1.ibout \
17          select2.sql select2.ibout select2.xml \
18          select3.sql select3.ibout select3.tex \
19          program.xml program.xsl eurotex.xml \
20          flintstones.db2 \
21          xsltprocessor.png

22    sql2context.pdf: $(DOCSRC)

23    # cleanup
24    clean:
25          #texutil --purge
26          rm flintstones.interbase flintstones.ib
27          rm flintstones.ibmdb2 flintstones.db2
28          rm *.ibout

29    # validation
30    validate: all
31          SAXCount *.xml
```

```
32      # packing
33      archive: sql2context.zip

34      sql2context.zip: sql2context.pdf $(DOCSRC) Makefile
35              -rm sql2context.zip
36              zip sql2context $(DOCSRC) Makefile

37      # file generation

38      flintstones.xml: flintstones.csv  example.dtd

39      flintstones.sql: flintstones.ddl

40      flintstones.ib: flintstones.ddl

41      flintstones.db2: flintstones.ddl

42      select2.xml: select2.ibout ib2xml.pl

43      select3.tex: select3.ibout
44              @echo cannot make select3.tex yet
```

```
45          @echo It should be included, so just say:
46          @echo   touch select3.tex
47          @echo and make again
48          exit 1

49    select1.ibout: select1.sql

50    select2.ibout: select2.sql

51    select3.ibout: select3.sql

52    eurotex.xml: program.xml program.xsl
53          testXSLT -in program.xml -XSL program.xsl -out eurotex.x

54    # db generation

55    flintstones.interbase: flintstones.ib
56          rm -f flintstones.gdb flintstones.out
57          echo 'create database "flintstones.gdb";' > createdb.sq
58          /opt/interbase/bin/isql -i createdb.sql
59          /opt/interbase/bin/isql -i flintstones.ib -o flintstones
```

```
60          echo 'select "surname", "age" from "flintstone" order by
61          touch flintstones.interbase
62          rm flintstones.out

63      flintstones.ibmdb2: flintstones.db2
64          #db2 start database manager
65          -db2 "drop database flint"
66          db2 "create database flint"
67          db2 -td\; -vf flintstones.db2
68          touch flintstones.ibmdb2

69      # rules

70      %.xml: %.csv csv2xml.pl
71          perl -w csv2xml.pl $< > tmp.tmp
72          mv tmp.tmp $@

73      %.xml: %.ibout ib2xml.pl
74          perl -w ib2xml.pl $< > tmp.tmp
75          mv tmp.tmp $@
```

intro-duction
targets
prereq-uisites
commands
Tips and Tricks
big example
close

```
76      %.ib: %.ddl
77              xplain2sql -interbase $< > tmp.tmp
78              mv tmp.tmp $@

79      %.db2: %.ddl
80              xplain2sql -db2 $< > tmp.tmp
81              mv tmp.tmp $@

82      %.sql: %.ddl
83              xplain2sql -ansi $< > tmp.tmp
84              mv tmp.tmp $@

85      %.pdf: %.tex
86              texexec $<

87      %.ibout: %.sql flintstones.interbase
88              if [ -e $@ ]; then rm $@; fi
89              /opt/interbase/bin/isql flintstones.gdb -i $< -o tmp.out
90              mv tmp.out $@
```

your turn again