make

Statistical Programming

Fall 2021

Dr. Colin Rundel

make

- build tool for the creation of software / libraries / documents by specifying dependencies
  - Almost any process that has files as input and outputs can be automated via make
- Originally created by Stuart Feldman in 1976 at Bell Labs
- Almost universally available (all flavors of unix / linux / MacOS / Windows w/ RTools)
- Dependencies are specified using a `Makefile` with a simple syntax

Makefile

A `Makefile` provides a list of target files along, their dependencies, and the steps necessary to generate each of the targets from the dependencies.

```
target1: depend1 depend2 depend3 ...
    step1
    step2
    step3
    ...

depend1: depend1.1
    step1.1
    step1.2
```

In the above example `target*` and `depened*` are all just files (given by a relative or absolute path).

Makefile (basic example)

```
paper.html: paper.Rmd fig1/fig.png fig2/fig.png
    Rscript -e "rmarkdown::render('paper.Rmd')"

fig1/fig.png: fig1/fig.R
    cd fig1;Rscript fig.R

fig2/fig.png: fig2/fig.R
    cd fig2;Rscript fig.R
```

Smart Building

Because the `Makefile` specifies the dependency structure `make` knows when a file has changed (by examining the file's modification timestamp) and only runs the steps that depend on the file(s) that have changed.

- After running `make` the first time, I edit `paper.Rmd`, what steps run if I run `make` again?
- What about editing `fig1/fig.R`?

Variables

Like R or other language we can define variables

```
R_OPTS=--no-save --no-restore --no-site-file --no-init-file --no-environ

fig1/fig.png: fig1/fig.R
    cd fig1;Rscript $(R_OPTS) fig.R
```

Special Targets

By default if you run `make` without arguments it will attempt to build the first target in the `Makefile` (whose name does not start with a `.`). By convention we often include an `all` target which explicitly specifies how to build everything within the project.

`all` is an example of what is called a phony target - because there is no file named `all` in the directory. Other common phony targets:

- `clean` - remove any files created by the Makefile, restores to the original state
- `install` - for software packages, installs the compiled programs / libraries / header files

We specify phony targets by including a line with `.PHONY` as the target and the phony targets as dependencies, i.e.:

```
.PHONY: all clean install
```

Builtin / Automatic Variables

- `$@` - the file name of the target
- `$<` - the name of the first dependency
- `$^` - the names of all dependencies
- `$(@D)` - the directory part of the target
- `$(@F)` - the file part of the target
- `$(<D)` - the directory part of the first dependency
- `$(<F)` - the file part of the first dependency

See GNU make documentation for a complete list

Pattern Rules

Often we want to build several files in the same way, in these cases we can use `%` as a special wildcard character to match both targets and dependencies.

So we can go from

```
fig1/fig.png: fig1/fig.R
    cd fig1;Rscript fig.R

fig2/fig.png: fig2/fig.R
    cd fig2;Rscript fig.R
```

to

```
fig%/fig.png: fig%/fig.R
    cd $(<D);Rscript $(<F)
```

Makefile (fancier example)

```
all: paper.html

paper.html: paper.Rmd fig1/fig.png fig2/fig.png
    Rscript -e "library(rmarkdown);render('paper.Rmd')"

Fig%/fig.png: Fig%/fig.R
    cd $(<D);Rscript $(<F)

clean:
    rm -f paper.html
    rm -f Fig*/*.png

.PHONY: all clean
```

HW4's Makefile

Live Demo