SPEAKER_NOTES.md

Speaker Notes for the 2015-11-16 CoDiMa Software Carpentry Make lesson.

Capturing shell output

Set this up before slides/speaking.

USE A SHORT PATH, e.g. `~/Desktop/make`.

```
$ export PATH=${PATH}:~/Development/GitHub/Teaching/recording-script/
$ recordsession.sh git@github.com:kkwakwa/2015-11-16-manchester-codima.git codima_les
```

Also, download the lesson materials

```
$ wget http://swcarpentry.github.io/make-novice/make-lesson.tar.gz
```

TYPE ALL EXAMPLES AS YOU GO. THIS KEEPS THE SPEED SANE, AND ALLOWS YOU TO EXPLAIN EVERY STEP.

USE ls TO SHOW FILE CREATION/DELETION

Go through introduction to `Make`

- `Make` is a build manager
- We want to build things: *targets*
    - executable programs
    - data analysis outputs
    - visualisations of data
    - documents/presentations
- The *targets* may depend on input or intermediate files (which may also be targets)
    - data files
    - analysis files
- **`Make` organises the construction of output `targets` from their inputs**
- `Makefiles` define the dependencies and rules for building
- `Make` will only build a target:

    - if it is missing
    - if a dependency has changed

- Other tools do similar things, but the fundamental concepts are the same
- `Make` is used in many contexts
    - compiling programs
    - bioinformatics pipelines
    - visualisation of analyses
    - combining text and figures for papers

Getting ready

Download materials before you start, and make sure you're in the right directory.

PAUSE to solve problems with the downloads

Join in the live coding by unpacking the archive:

```
$ tar -xvf make-lesson.tar.gz
$ cd make-lesson
```

Test if `Make` works:

```
 make
make: *** No targets specified and no makefile found.  Stop.
$ make -h | head
```

PAUSE to be sure everyone has `Make` working.

Automation and `Make`

- The script `wordcount.py` reads a text file, counts the words, and outputs a data file - **slides while live coding**

```
$ python wordcount.py books/isles.txt isles.dat
```

- Look at the output

```
$ head -n 5 isles.dat
the 3822 6.7371760973
of 2460 4.33632998414
and 1723 3.03719372466
to 1479 2.60708619778
a 1308 2.30565838181
```

- Three columns of text, one row per word

    1. the word
    2. the count of the word in the text
    3. the fraction of all words in the text that are this word

- We might run it on one file, **but we could need to run it on several**.

- Second example:

```
$ python wordcount.py books/abyss.txt abyss.dat
$ head -n 5 abyss.dat
the 4044 6.35449402891
and 2807 4.41074795726
of 1907 2.99654305468
a 1594 2.50471401634
to 1515 2.38057825267
```

- There's a script `plotcount.py` that produces a graph from the data - shows a bar chart of the 10 most frequent words

```
$ python plotcount.py isles.dat show
```

- This produces a live window.
- **Close the window to proceed**

PAUSE in case there are any problems. It doesn't matter if there's an issue, as we'll use the output files, anyway.

- The script also generates output files

```
$ python plotcount.py isles.dat isles.jpg
```

- These are the steps of a generic analysis and visualisation workflow
- **EMPHASISE GENERIC WORKFLOW NATURE**

    1. Read a file
    2. Do analysis
    3. Write intermediate results
    4. Visualise/process results
    5. Write final output

- Working at the command line is OK for a small number of files, but doesn't scale well when we need to process 100s of inputs

    - boring
    - errors can happen - **ESPECIALLY FOR COMPLEX SEQUENCE OF COMMANDS**

    - **going back to check for errors is also boring and error-prone**

- Automation is critical - **AUTOMATE WORKFLOWS**
- We could do this with a script, so why use `Make` ?
- With a script:
    - Unless you're very clever, you run on all files. **But what if only one input changes?** That's slow and wasteful
- With `Make` :
    - Rebuilds only happen if dependencies change, or targets go missing
    - Dependencies are explicit - **SELF-DOCUMENTING**
    - One system to cover many eventualities
    - **THE FUNDAMENTAL CONCEPTS COMMON TO BUILD TOOLS**

Makefiles

- Create a `Makefile`
- **USE NANO IN ANOTHER SHELL/TERMINAL**
- **STUDENTS: USE THE EDITOR YOU LIKE BEST**
- Makefile contents:

```
# Count words.
isles.dat : books/isles.txt
        python wordcount.py books/isles.txt isles.dat
```

PAUSE to be sure everyone can edit and save

- Go through the elements, prompted by slides
    - `#`
    - target/dependency line
    - action line(s)

THE SPACE AT THE START OF THE ACTION LINE IS A TAB!!

!!!Live coding exercises!!!

DELETE .dat AND .jpg FILES

```
$ rm *.dat *.jpg
```

Try the `Make` :

```
$ make
make: *** No targets specified and no makefile found.  Stop.
$ make
Makefile:3: *** missing separator (did you mean TAB instead of 8 spaces?).  Stop.
$ make
python wordcount.py books/isles.txt isles.dat
```

PAUSE There will be errors! Let them be dealt with

DEMONSTRATE ERRORS (NO MAKEFILE; SPACE FOR TAB)

- A successful `Make` prints out the actions it executes.
- **RERUN MAKEFILE**

```
$ make
make: `isles.dat' is up to date.
```

- `Make` tells us if nothing needs to be done

DEMONSTRATE TOUCH

- We can use the `touch` command to update the timestamp on one of the dependencies
    - This makes it look like it's been edited

```
$ touch books/isles.txt
```

COMPARE TIMESTAMPS

```
$ ls -l books/isles.txt isles.dat
```

isles.dat NOW LOOKS OLDER THAN isles.txt DEPENDENCY triggering a rebuild

```
$ make
python wordcount.py books/isles.txt isles.dat
```

- Add a second rule to the Makefile:

```
abyss.dat : books/abyss.txt
	python wordcount.py books/abyss.txt abyss.dat
```

RUN MAKE WITH NO ARGUMENTS

```
$ make
make: `isles.dat' is up to date.
```

- Make tries to build only the first (default target)
- We have to tell it to build `abyss.dat` explicitly

```
$ ls
$ make abyss.dat
python wordcount.py books/abyss.txt abyss.dat
$ ls
```

PAUSE - everyone OK?

- We may want to remove all our data files so we can recreate them.
- We use a **NEW TARGET** and a **NEW RULE**, `clean`

```
clean :
	rm -f *.dat
```

- **THIS RULE HAS NO DEPENDENCIES**
- Run the Makefile

```
$ ls
$ make clean
rm -f *.dat
$ ls
```

- We're **NOT BUILDING SOMETHING CALLED CLEAN**
- This can cause problems:
  - **Create a directory called `clean` and run `make clean`**

```
$ mkdir clean
$ ls
$ make clean
make: `clean' is up to date.
```

- **Make finds `clean` and, because it has no dependencies, assumes it's up to date**
- We need to tell Make to always execute this rule when asked
- We need to **make `clean` a PHONY TARGET**

```
.PHONY : clean
clean :
	rm -f *.dat
```

- Now `make clean` works:

```
$ make clean
rm -f *.dat
```

- Let's add a similar phony target to make all the data files

```
.PHONY : dats
dats : isles.dat abyss.dat
```

- This is a **RULE WHOSE DEPENDENCIES ARE TARGETS OF OTHER RULES**
- Make **checks to see if the dependencies exist**; if not, it **looks for rules that create them**
- We use these kinds of rules to trigger builds of the dependencies.

NOTE: THE ORDER OF BUILDING DEPENDENCIES IS ARBITRARY

DEPENDENCIES MUST BE A DIRECTED ACYCLIC GRAPH

- Running the build (**REPEAT TO TEST**)

```
$ ls
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
$ ls
$ make dats
make: Nothing to be done for `dats'.
```

!!!Back to slides!!!

PAUSE Everyone up to speed?

RECAP ELEMENTS to help catch up

- phony target - triggers build
- one rule per target data file
- phony target to clean data

- show dependency graph

Exercise (10min)

- Describe exercise on slide, and have a breather.

- Demonstrate solution

```
# Count words.
.PHONY : dats
dats : isles.dat abyss.dat last.dat

isles.dat : books/isles.txt
	python wordcount.py books/isles.txt isles.dat

abyss.dat : books/abyss.txt
	python wordcount.py books/abyss.txt abyss.dat

last.dat : books/last.txt
	python wordcount.py books/last.txt last.dat

# Generate archive file.
analysis.tar.gz : isles.dat abyss.dat last.dat
	tar -czf analysis.tar.gz isles.dat abyss.dat last.dat

.PHONY : clean
clean :
	rm -f *.dat
	rm -f analysis.tar.gz
```

```
$ make analysis.tar.gz
python wordcount.py books/last.txt last.dat
tar -czf analysis.tar.gz isles.dat abyss.dat last.dat
```

- Show dependency graph

Automatic variables

PAUSE to make sure everyone is caught up

- Our Makefile has duplication.
- There are lots of **REPEATED NAMES** - this can be a problem:
  - forgetting to rename after a change
  - typos from repetitive typing
- Goal is to **REDUCE REPETITION** for **CODE ROBUSTNESS**
- One type of repetition is the name of target of a rule
- There's a special **AUTOMATIC VARIABLE** that can be used to replace the target of the current rule in any actions: `$@`

REPLACE IN Makefile

```
analysis.tar.gz : isles.dat abyss.dat last.dat
	tar -czf $@ isles.dat abyss.dat last.dat
```

TEST

```
$ touch books/*.txt
$ make analysis.tar.gz
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
tar -czf analysis.tar.gz isles.dat abyss.dat last.dat
```

- Another type of repetition is that dependencies show up in the dependencies and the action
- There's a special **AUTOMATIC VARIABLE** that can be used to replace the dependencies of the current rule in any actions: `$^`

REPLACE IN Makefile

```
analysis.tar.gz : isles.dat abyss.dat last.dat
	tar -czf $@ $^
```

TEST

```
$ touch books/*.txt
$ make analysis.tar.gz
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
tar -czf analysis.tar.gz isles.dat abyss.dat last.dat
```

!!!Live coding exercises!!!

TRY THE BASH WILDCARD

- Let's try using the `bash` wild-card for our dependencies in the `analysis.tar.gz` rule

```
# Generate archive file.
analysis.tar.gz : *.dat
	tar -czf $@ $^
```

- `touch` inputs and re-run

```
$ touch books/*.txt
$ make analysis.tar.gz
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/last.txt last.dat
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat
```

- This all seems to work well, **but…**

DELETE DATA FILES AND RERUN RULE

```
$ make clean
rm -f *.dat
rm -f analysis.tar.gz
$ make analysis.tar.gz
make: *** No rule to make target `*.dat', needed by `analysis.tar.gz'.  Stop.
```

- This doesn't work.
- **No files match the pattern `*.dat`**
    - Make tries to use `*.dat` as a filename, but there isn't one - so there's an error

WE NEED TO REBUILD THE `.dat` FILES EXPLICITY

```
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
$ make analysis.tar.gz
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat
```

!!!Back to the slides!!!

MCQ: Updating Dependencies

Ask question - use stickies to indicate the answer: DON'T EXECUTE THE CODE

ASK PEOPLE TO FIND SOMEONE NEARBY WHO HAS A DIFFERENT ANSWER - DISCUSS THEIR ANSWERS: WHY DO YOU THINK YOU'RE RIGHT?

Ask question again - people can change their answer

Ask people to run the code

```
$ touch *.dat
$ make analysis.tar.gz
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat
```

The answer is (4) - only `analysis.tar.gz` is recreated

- The `$^` automatic variable works well if all the dependencies are treated the same
- Sometimes we want to treat the first dependency differently from the rest
    - e.g. as the only input file from all of the dependencies

- Make has an automatic variable meaning "the first dependency of the current rule": `$<`

- More automatic variable info at https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html

Exercise (5min)

- Describe exercise on slide, and have a breather.
- Demonstrate solution

```
isles.dat : books/isles.txt
    python wordcount.py $< $@

abyss.dat : books/abyss.txt
    python wordcount.py $< $@

last.dat : books/last.txt
    python wordcount.py $< $@
```

```
$ make clean
rm -f *.dat
rm -f analysis.tar.gz
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
$ make analysis.tar.gz
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat
```

Dependencies on data and code

BEFORE GOING ON TO FIRST SLIDE IN SECTION

QUESTION: Would changing `wordcount.py` potentially change the output?

QUESTION: What happens if we modify `wordcount.py` ?

- The output data files are not just a product of the data, but also the code that generates them
- If `wordcount.py` changes, the output might change, but our Makefile doesn't reflect this yet.
- **We need to add `wordcount.py` as a dependency**
- **OUTPUT DEPENDS ON BOTH CODE AND DATA**

!!Live coding exercise!!

Change the `.dat` rules:

```
isles.dat : books/isles.txt wordcount.py
        python wordcount.py $< $@

abyss.dat : books/abyss.txt wordcount.py
        python wordcount.py $< $@

last.dat : books/last.txt wordcount.py
        python wordcount.py $< $@
```

```
$ make clean
rm -f *.dat
rm -f analysis.tar.gz
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
```

PAUSE to let people catch up

- What happens if we pretend to edit `wordcount.py` ?

TOUCH `wordcount.py`

```
$ make dats
make: Nothing to be done for `dats'.
$ touch wordcount.py
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
```

!!Back to slides!!

- Dependency graph
- All the `.dat` files now also depend on `wordcount.py`
- **QUESTION: why don't `.txt` files depend on `wordcount.py` ?
    - `.txt` files are input files and have no dependencies. To make these depend on `wordcount.py` would introduce a false dependency.

!!Live coding exercise!!

- Now that the final output depends on `wordcount.py` , we should add the code to our complete archive

```
# Generate archive file.
analysis.tar.gz : *.dat wordcount.py
	tar -czf $@ $^
```

```
$ make analysis.tar.gz
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat wordcount.py
```

PAUSE Is everyone up to this point?

Pattern rules

- We still have repetition
  - The `.dat` rules only vary by text and data filenames
  - We can replace these with a single pattern rule

THE PATTERN RULE LETS US BUILD ANY `.dat` FILE FROM a `.txt` FILE IN `books/`

- `%` is a Make wildcard
  - Used only in targets and dependencies, **NOT ACTIONS**
  - Matches dependencies with targets
  - `$*` is a special variable that "catches" the contents of `%` - it is replaced by the stem which matches the `%` pattern rule
- Replace the three `.dat` rules

```
%.dat : books/%.txt wordcount.py
	python wordcount.py $< $*.dat
```

```
$ make clean
rm -f *.dat
rm -f analysis.tar.gz
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
```

PAUSE Is everyone keeping up?

- Our Makefile is now much shorter, and cleaner

`Make` variables

- Make allows us to define variables (or macros) that can hold values.
  - Putting a value in a variable is called *assignment*

- If we define a variable `VAR`, then we have to reference it in parentheses as `$(VAR)`
- We have tried to reduce repetition, but we have introduced some
  - `wordcount.py` occurs several times
  - if we renamed the script, we'd have to make several changes
- To reduce duplication further, let's use a variable to replace all occurrences of `wordcount.py`

AT THE TOP OF THE Makefile

```
1 COUNT_SRC=wordcount.py
```

- This script is always invoked by passing it to `python`.
  - **That may not be true for a replacement script**
  - we can use another variable to give us flexibility in our script language
  - comment so we know what we're doing

```
1 # Count words script.
2 COUNT_SRC=wordcount.py
3 COUNT_EXE=python $(COUNT_SRC)
```

Exercise (10min): Use variables

- Describe exercise on slide, and have a breather.
- Demonstrate solution

```
1 %.dat : books/%.txt $(COUNT_SRC)
2         $(COUNT_EXE) $< $*.dat
3
4 # Generate archive file.
5 analysis.tar.gz : *.dat $(COUNT_SRC)
6         tar -czf $@ $^
```

```
1 $ touch books/*.txt
2 $ make analysis.tar.gz
3 python wordcount.py books/abyss.txt abyss.dat
4 python wordcount.py books/isles.txt isles.dat
5 python wordcount.py books/last.txt last.dat
6 tar -czf analysis.tar.gz abyss.dat isles.dat last.dat wordcount.py
```

IT'S GOOD PRACTICE TO WRITE MODULAR CODE

- Decoupling source code from configuration (e.g. input filenames) is good practice
  - **modular**

    - **flexible**
    - **maintainable**
    - **reusable**

- Putting input names at the top of the Makefile is convenient

    - Putting them into a separate configuration Makefile is better
    - Changing a script name only requires editing a configuration, **not source code**

CREATE `config.mk`

```
$ nano config.mk
```

```
# Count words script.
COUNT_SRC=wordcount.py
COUNT_EXE=python $(COUNT_SRC)
```

REMOVE SAME LINES FROM Makefile

- Replace with `include`

```
include config.mk
```

```
$ make clean
rm -f *.dat
rm -f analysis.tar.gz
$ make dats
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/last.txt last.dat
$ make analysis.tar.gz
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/last.txt last.dat
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat wordcount.py
```

CONGRATULATIONS - YOU'VE WRITTEN MODULAR, MAINTAINABLE CODE!

`Make` functions

- **We can write more complex rules, using Make functions**

    - **We might want to analyse all `.txt` files in a directory, without knowing ahead of time what they are called**

- We can use the `wildcard` function to get a list of files that match some pattern, and save them in a variable.

ADD THE CODE, AND EXPLAIN THE FUNCTION STRUCTURE

```
TXT_FILES=$(wildcard books/*.txt)
```

- We can use a `.PHONY` target called `variables` to show the value of that variable

```
.PHONY : variables
variables:
    @echo TXT_FILES: $(TXT_FILES)
```

QUESTION: why do we use `@echo` rather than `echo`?

!!Live coding exercise!!

```
 make variables
TXT_FILES: books/abyss.txt books/isles.txt books/last.txt books/sierra.txt
```

- The file `sierra.txt` is now included in our set of input files

To illustrate the point about `@echo`

```
.PHONY : variables
variables :
        echo TXT_FILES: $(TXT_FILES)
```

```
$ make variables
echo TXT_FILES: books/abyss.txt books/isles.txt books/last.txt books/sierra.txt
TXT_FILES: books/abyss.txt books/isles.txt books/last.txt books/sierra.txt
```

Using `@echo` suppresses writing the command to STDOUT

!!Back to slides!!

- Dependency graph shows that all four `books/*.txt` files are included
    - **If you add another `.txt` file to that directory, it will also be included**
- The `patsubst` function replaces one sequence of characters with another
    - The function takes a pattern, a replacement string, and a list of names, in that order.
    - Each name in the list matching the pattern, is replaced by the replacement string - **only the stem represented by the `%` is kept**

Explain the function on the slide

Create the variable and extend the `variables` target

```
DAT_FILES=$(patsubst books/%.txt, %.dat, $(TXT_FILES))

.PHONY : variables
variables :
        @echo TXT_FILES: $(TXT_FILES)
        @echo DAT_FILES: $(DAT_FILES)
```

!!Live coding exercise!!

```
$ make variables
TXT_FILES: books/abyss.txt books/isles.txt books/last.txt books/sierra.txt
DAT_FILES: abyss.dat isles.dat last.dat sierra.dat
```

- Now that `sierra.txt` is processed, we can modify the `dats` rule dependencies

```
# Count words.
.PHONY : dats
dats : $(DAT_FILES)
```

- and our clean target

```
.PHONY : clean
clean :
        rm -f $(DAT_FILES)
        rm -f analysis.tar.gz
```

- Let's test:

```
$ make clean
rm -f  abyss.dat  isles.dat  last.dat  sierra.dat
rm -f analysis.tar.gz
```

```
$ make dats
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/last.txt last.dat
python wordcount.py books/sierra.txt sierra.dat
```

- We can also rewrite the `analysis.tar.gz` rule, and test the Makefile

```
$ make clean
rm -f abyss.dat  isles.dat  last.dat  sierra.dat
rm -f analysis.tar.gz
$ make analysis.tar.gz
python wordcount.py books/abyss.txt abyss.dat
python wordcount.py books/isles.txt isles.dat
python wordcount.py books/last.txt last.dat
python wordcount.py books/sierra.txt sierra.dat
tar -czf analysis.tar.gz abyss.dat isles.dat last.dat sierra.dat wordcount.py
```

PAUSE see if people have caught up

!!Back to slides!!

The problem with `*.dat` (required us to run `make dats`) has gone away

Using functions lets us generate `.dat` filenames automatically from `books/*.txt` files

THIS ALLOWS US TO PROCESS ALL `books/*.txt` FILES WITHOUT KNOWING THEIR NAMES AHEAD OF TIME

NO INPUT FILES ARE NAMED IN THIS CODE

Conclusions

- Make
  - Automates repetitive commands
  - Reduces risk of error
  - Only updates files/outputs when dependencies have changed
  - Only builds what hasn't already been built
  - Code acts as documentation, recording dependencies and specifying how to generate all outputs from their inputs

Exercise (15min)

- Explain the exercise
- Have a breather

Before showing the solution, show the dependency graph

SHOW THE SOLUTION

- For `config.mk`:

```
# Count words script.
COUNT_SRC=wordcount.py
COUNT_EXE=python $(COUNT_SRC)
PLOT_SRC=plotcount.py
PLOT_EXE=python $(PLOT_SRC)
```

- For `Makefile`:

```
include config.mk

# All text files
TXT_FILES=$(wildcard books/*.txt)
DAT_FILES=$(patsubst books/%.txt, %.dat, $(TXT_FILES))
JPG_FILES=$(patsubst books/%.txt, %.jpg, $(TXT_FILES))

.PHONY : variables
variables :
	@echo TXT_FILES: $(TXT_FILES)
	@echo DAT_FILES: $(DAT_FILES)
	@echo JPG_FILES: $(JPG_FILES)

# Count words.
.PHONY : dats
dats : $(DAT_FILES)

%.dat : books/%.txt $(COUNT_SRC)
	$(COUNT_EXE) $< $*.dat

# Plot counts
.PHONY : jpgs
jpgs : $(JPG_FILES)

%.jpg : %.dat $(PLOT_SRC)
	$(PLOT_EXE) $< $*.jpg

# Generate archive file.
analysis.tar.gz : $(DAT_FILES) $(JPG_FILES) $(COUNT_SRC) $(PLOT_SRC)
	tar -czf $@ $^

.PHONY : clean
clean :
	rm -f $(DAT_FILES) $(JPG_FILES)
	rm -f analysis.tar.gz
```