Key Points of make by Capes in [Cap16]

- **Introduction**: Make allows us to specify what depends on what and how to update things that are out of date
- **Makefiles**:
  - Use `#` for comments in Makefiles.
  - Write rules as `target: dependencies`.
  - Specify update actions in a tab-indented block under the rule.
  - Use `.PHONY` to mark targets that don't correspond to files.
- **Automatic Variables**:
  - Use `$@` to refer to the target of the current rule.
  - Use `$^` to refer to the dependencies of the current rule.
  - Use `$<` to refer to the first dependency of the current rule.
- **Dependencies on Data and Code**:
  - Make results depend on processing scripts as well as data files.
  - Dependencies are transitive: if A depends on B and B depends on C, a change to C will indirectly trigger an update to A.
- **Pattern Rules**:
  - Use the wildcard `%` as a placeholder in targets and dependencies.
  - Use the special variable `$*` to refer to matching sets of files in actions.
- **Variables**:
  - Define variables by assigning values to names.
  - Reference variables using `$(...)`.
- **Functions**:
  - Make is actually a small programming language with many built-in functions.
  - Use `wildcard` function to get lists of files matching a pattern.
  - Use `patsubst` function to rewrite file names.
- **Self-Documenting Makefiles**: Document Makefiles by adding specially-formatted comments and a target to extract and format them.
- **Conclusion**: Makefiles save time by automating repetitive work, and save thinking by documenting how to reproduce results.

Basic structure and some examples

Listing 1: Makefile

```
target: dependency1 dependency2 ...
        action1 (start line with TAB!)
        action2
        ...
```

Listing 2: Makefile2

```
%.o: %.c
        # %: wildcard as placeholder
        # gcc is the used compiler
        # - Wall: print all warnings
        # -O2, -O3: optimization levels
        # -g: produce debug information for gdb
        # -c: compilation only (no linking)
        # $<: the first dependency of the current rule

        gcc -Wall -O3 -g -c $<
```

Listing 3: Makefile3

```
# Makefile to compile one executable per source

CC = gcc
CFLAGS = -std=c99 -g -Wall -Wextra

# Pattern substitution (from, to, source-list):
# for each *.c file from the source-list, a *.x executable
  ↪ is build
all: $(patsubst %.c, %.x, $(wildcard *.c))

%.x: %.o
        $(CC) ${CFLAGS} -o $@ $<

# $@ - name of the executable
# $< - first item in the depedency list (the .o file)

clean:
        rm -f *.x
        rm -f *.o
        rm -f *~
```

Listing 4: Makefile4 by [Mer16]

```
1  CC = gcc
2  CFLAGS = -g -Wall
3  OUTPUT = my_prog
4  OBJFILES = lib.o prog.o
5
6  $(OUTPUT): $(OBJFILES)
7      $(CC) $(CFLAGS) $(OBJFILES) -o my_prog
8
9  %.o: %.c
10     # $<: dependency (%.c)
11     # $@: target (%.o)
12     $(CC) $(CFLAGS) -c $< -o $@
13
14 clean:
15     rm *.o $(OUTPUT)
```

Implicit Rules [SMS14, p.116 f]

List of some of the more common **variables used as names of programs in built-in rules**:

- **CC**: Program for compiling C programs; default 'cc'.
- **CXX**: Program for compiling C ++ programs; default 'g++'.
- **CPP**: Program for running the C preprocessor, with results to standard output; default '$(CC) -E'.

List of **variables whose values are additional arguments** for the programs above. The default values for all of these is the empty string, unless otherwise noted.

- **ASFLAGS**: Extra flags to give to the assembler (when explicitly invoked on a '.s' or '.S'file).
- **CFLAGS**: Extra flags to give to the C compiler.
- **CXXFLAGS**: Extra flags to give to the C++ compiler.
- **LDFLAGS**: Extra flags to give to compilers when they are supposed to invoke the linker, 'ld', such as -L. Libraries (-lfoo) should be added to the LDLIBS variable instead.
- **LDLIBS**: Library flags or names given to compilers when they are supposed to invoke the linker, 'ld'.LOADLIBES is a deprecated (but still supported) alternative to LDLIBS. Non-library linker flags, such as -L, should go in the LDFLAGS variable.

Bibliography

[Cap16] Gerard Capes. Automation and make. `http://swcarpentry.github.io/make-novice/`, October 2016. (last accessed: 2016-10-21).

[Mer16] Karmi Merimovitch. Introduction to the programm make. `http://tzvimelamed.com/lab/pdf/OS-Lab-03-Make.pdf`, October 2016. (last accessed: 2016-10-21).

[SMS14] Richard M Stallman, Roland McGrath, and Paul D Smith. Gnu make manual. *Free Software Foundation*, 3, 2014.