Make -- a command generator program

Input: a description file, normally called "Makefile"
(Also called "makefile" or by flags "-f filename".)

Output: a series of commands to a shell

A Simple Makefile:

```
# This is an example makefile
FILES = f1 f2 f3
result: f1 f2 f3
	cat f1 f3 f3 > result
clean:
	rm -f result
```

Parts of make files

comment lines: start with #

make variables: NAME=value
(aka macros)

targets: "result", "clean"

dependencies: f1 f2 f3

description line:
<tab>command

Make and Makefiles

To see tabs and "End of Lines":

- % cat -e -t -v Makefile

To run make

- % make
- % make target-name
- % make clean

Example ex1 here

Targets and Descriptions

Target: something to be made

- May be a file name to be made (myprog)
- May not make a file by that name (all)

Dependency or prerequisite

- if any dependency is newer than target -> make
- may use variables here ($(name))
  - Example ex2 here
- something to be made before the target
  - Example ex3 here
  - $@ -- special builtin variable -- target name

Compiling C files

```
# This is yet another makefile example!
#

hello: hello.o
	gcc -g -o hello hello.o

hello.o: hello.c
	gcc -g -c hello.c

clean:
	rm -r hello hello.o
```

Example ex4

Example ex5 -- Multiple C files

- Extra dependency list

Predefined Rules

```
# This is yet another makefile example!

CC=gcc
CFLAGS= -g -Wall

hello: hello.o bye.o
	gcc -g -o hello hello.o bye.o

clean:
	rm -r hello hello.o bye.o

# dependency list
hello.o bye.o: bye.h
```

Macros: CC, CFLAGS
No rules for "hello.o" and "bye.o"
Example ex6

More Macros

Builtin Macros

- $@ -- current target
- $? -- newer prerequisite list

Macro substituion

- SRCS = a.c b.c
- OBJS = ${SRCS:.c=.o}

Multiple targets

- a b c : d e f
- a.o b.o c.o: defs.h

Example ex7

Suffix Rules

Suffix: file.c => .c
Others: .o .cxx .b .p ...

How to turn a .c file into a .o file?

```
.SUFFIXES: .c .o .cxx

.c.o:; $(CC) $(CFLAGS) -c $<

.cxx.o:; $(CXX) $(CXXFLAGS) -c $<
```

Most make programs have default suffix rules for C compiles.
$< is name of current prerequisite (suffix rules only)

Example ex8

Other Notes

- @ at start of line causes no echo of command
- - at start of line causes errors to be ignored
- $$ is the $ character in the command
- $(SHELL) is often set to the name of the shell to call
- Continuation lines: Backslash (\) followed by end of line
- Some makes use #include or .include

Recursive makes

```
SUBDIRS = A B C

all:
        for d in $(SUBDIRS) ; do \
         (cd $$d; $(MAKE) all ) ; done
```