

# make

My description of the program make is that it

- ☞ takes a set of rules describing dependencies and
- ☞ describing creation of new files

in order to satisfy the requirements for the “creation” of some target.



# make

Another description from Chapter 1 of the Gnu Make manual:

The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.



# Invoking make

There are several options that are generally useful with make:

```
-f MAKEFILE      # specify an alternative makefile to the defaults of
                  # 'GNUmakefile', 'Makefile', or 'makefile'
-k              # continue for other targets even after an error

-i              # completely ignore errors

-d              # print debugging information

-j [N]          # fork off children to handle tasks. If N is
                  # specified, create no more than N children
```



```
-C DIR      # change directory to DIR before starting the make process  
  
-s          # silent mode, don't echo commands
```



# Makefiles

Makefiles use rules to determine their actions. The rules look like:

```
target: [ prerequisites ]  
    -TAB- action  
    -TAB- action  
    -TAB- ...
```



# Targets

Targets usually either specify a file that is to be made via this rule or just identify the rules for execution (often called a “phony” target.)

Targets may also be implicit.



# Prerequisites

These generally define the files that the target depends on, and the general idea is that if any of those have a modified (or creation) time later than the target, then actions for the rule will be executed to create a new version of the target (which you should try to make sure has a new modified or created time.)



# Actions

These generally define the actions that are needed to create the target from the prerequisites. These actions are largely executions of discrete programs such as `gcc`, `make` (yes, recursion is quite common), `ld`, `bison`, `flex`, and so on. Rules must consist of consecutive lines that start with a TAB character. Since these are usually interpreted as shell commands, you can do things such as multi-lines (but use the backslash to make sure that the “single-linedness” of your construction is clear):





```
for name in dir1 dir2 dir3 \  
do ; \  
    ${MAKE} $name ; \  
done
```



# Actions

There are also actual make conditionals which are interpreted by make and not by the shell; these look like



```
ifeq (ARG1,ARG2)
```

```
...
```

```
endif
```

```
ifdef (ARG1)
```

```
...
```

```
endif
```



# Setting ordinary variables

You can use “=” and “?=” to set ordinary variables:

```
CFLAGS ?= -g -O3           # conditionally set ${CFLAGS} to
                             # ‘-g -O3’ iff it is not
                             # already defined

CC = /usr/bin/gcc ${CFLAGS} # unconditionally set ${CC} to
                             # ‘/usr/bin/cc’
```



# Pattern rules

One of the nice things that you can do with make is create “pattern rules”.

These are rules that let you abstract a pattern from a set of similar rules, and use that pattern in lieu of explicitly naming all of those rules.

For instance,



```
%.o : %.c
    cc -c $< -o $@    # $@ refers to the
                        # target, $< refers to
                        # the *first* (and only)
                        # prerequisite
```



# Automatic variables

`$@`      # the target of the rule

`$<`      # the first prerequisite

`$^`      # all of the prerequisites

`$?`      # all of the prerequisites that are newer than the target file

`$*`      # the ‘‘stem’’ only; essentially, this is the complement of the static portion  
# of the target definition... see Makefile-auto



# Example Makefiles

```
targets: 01-introduction-out.pdf 02-processes-out.pdf \  
03-shells1-out.pdf 03-shells2-out.pdf 04-shells3-out.pdf \  
05-shells4-out.pdf 06-environment-out.pdf 07-perl01-out.pdf \  
08-perl02-out.pdf 09-perl03-out.pdf 10-perl04-out.pdf \  
11-perl05-out.pdf 12-perl06-out.pdf 13-perl07-out.pdf \  
14-programdevel-out.pdf 15-programdevel02-out.pdf \  
16-programdevel03-out.pdf 17-programdevel04-out.pdf \  
18-programdevel05-out.pdf 19-programdevel06-out.pdf \  
20-programdevel07-out.pdf structure-out.pdf
```

```
%-out.pdf: %.tex  
    pdflatex $<  
    gij -jar pp4p.jar $*.pdf $*-out.pdf
```





# Example Makefiles

```
%.c:  
    echo $*
```

