MAKEFILE : TARGETS, DEPENDENCIES, COMMANDS

```
1 editor : editor.o screen.o keyboard.o          ---- 1st Rule
2 gcc -o editor editor.o screen.o keyboard.o     ---- 2nd
3
4 editor.o : editor.c editor.h keyboard.h screen.h ----
5 gcc -c editor.c
6
7 screen.o : screen.c screen.h
8 gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11 gcc -c keyboard.c
12
13 clean :
14 rm editor *.o
```

* To compile "editor", simply type "make" in the directory where the makefile exists.

* makefile has five rules - (i) The first target, editor, is called the default target.

↳ editor has three dependencies, editor.o, screen.o and keyboard.o; these three file must exists in order to build editor.

(ii) ↳ Line 2, is the command that "make" will exe. to create editor.

↳ "GNU cc" this command builds an executable named editor from the three object files. (individual object files.)

* The next three rules tell "make" how to build the indi-

MAKE IN LINUX

* A tool to control the process of building (rebuilding) software.
* Make automates what software gets built, how it gets built, and when it gets built.

Why make ?

* projects composed of multiple source files typically require long, complex compiler invocations; "Make" simplifies this by storing these difficult command lines in the "make file".
* "Make" also minimizes rebuild times because it is smart enough to determine which file have changed, thus only rebuilds files whose components have changed.
* finally, "make" maintains a data base of dependency information for your projects and so can verify that all of the files necessary for building a program are available each time you start a build.

Writing Makefile

* A makefile is a text file data base containing rules that tell "make" what to build and how to build it.
  * A rule consists of the following
    i) A target, the "thing" make ultimately tries to create.

```
1  editor : editor.o screen.o keyboard.o
2   gcc -o editor editor.o screen.o keyboard.o
3
4  editor.o : editor.c editor.h keyboard.h screen.h
5   gcc -c editor.c
6
7  screen.o : screen.c screen.h
8   gcc -c (keyboard.c) screen.c
9
10  keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13  .PHONY: clean
14
15  clean
16   rm editor *.o
```

clean removes the editor executable and its constituent object files.

* The final target in above ex., clean, is a phony target.
* Because clean does not have dependencies, its commands are not automatically executed.
* upon encountering the clean target, make sees if the dependencies exist and, because clean has no dependencies, make assumes the target is up to date.
* In order to build this target, you have to type "make clean".
* It has no dependencies, make would assume that it is up to date and not execute the commands on line 14. To deal with this situation use the special make target .PHONY.

* If you tried to build editor using the command from line 2, gcc would complain loudly and ceremoniously quit if the dependencies did not exists.

* "make", on the other hand, after seeing that editor requires these other files; verifies that they exist and, if they don't, executes the commands on lines 5, 8 and 11 first; then returns to line 2 to create the editor executable.

* of course, if the dependencies for the component such as keyboard.c or screen.h don't exist, "make" will also give up, because it lacks targets named, in this case, keyboard.c and screen.h.

* If a specified target does not exist in a place where make can find it, make (re) builds it.

* If the target does exist, make compares the timestamp on the target, make rebuilds the target, assuming that the newer dependency implies some code change that must be incorporated into the target.

phony targets → * In addition to the normal file targets, "make" allows you to specify phony targets

* phony targets are so named because they do not correspond to actual files.

* phony targets exist to specify commands that "make" should execute.

(ii) A list of one or more dependencies, usually files, required to build the target.

(iii) A list of commands to execute in order to create the target from the specified Dependencies when invoked.

* "GNU make" looks for the file named "GNU make-file", makefile or Makefile.

* most LINUX programmers use the last form, Makefile

```
target : dependency dependency [...]
    command
    command
    [...]
```

* target is generally the file, such as a binary or object file, that you want created.
* Dependency is a list of one or more files required as input in order to create target.