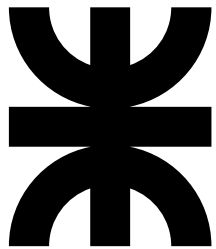
Informática II

Construcción de proyectos con `make`

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2017 –

.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- Automatiza: qué partes construir, cómo construirlas, y cuando.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- Automatiza: qué partes construir, cómo construirlas, y cuando.
- Le permite al programador poder concentrarse en el código.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- `make` simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- Automatiza: qué partes construir, cómo construirlas, y cuando.
- Le permite al programador poder concentrarse en el código.
- `make` minimiza el tiempo de construcción (determina qué archivos cambiaron), además trabaja con dependencias.

Contrucción de proyectos con `make`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- **`make`** simplifica el proceso de construcción de proyectos de múltiples archivos fuentes, que generalmente requieren varias llamadas al compilador.
- Automatiza: qué partes construir, cómo construirlas, y cuando.
- Le permite al programador poder concentrarse en el código.
- **`make`** minimiza el tiempo de construcción (determina qué archivos cambiaron), además trabaja con dependencias.

(Optimiza el tiempo del ciclo editar-compilar-verificar)

Contrucción de proyectos con `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

Contrucción de proyectos con `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados

Contrucción de proyectos con `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`

Contrucción de proyectos con `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`
- Qué pasa si se modifica `b.h` y no se recompila `2.c`

Contrucción de proyectos con `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`
- Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.o
main.o: main.c a.h
2.o: 2.c a.h b.h
3.o: 3.c b.h c.h
```

Contrucción de proyectos con `make`

Archivo `Makefile`

Un archivo **`Makefile`** es un archivo de texto que contiene *reglas* que le indican a **`make`** qué construir y cómo. Una *regla* consiste en:

Contrucción de proyectos con `make`

Archivo `Makefile`

Un archivo `Makefile` es un archivo de texto que contiene *reglas* que le indican a `make` qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

Contrucción de proyectos con `make`

Archivo `Makefile`

Un archivo **`Makefile`** es un archivo de texto que contiene *reglas* que le indican a **`make`** qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

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

Contrucción de proyectos con `make`

Archivo `Makefile`

Un archivo **`Makefile`** es un archivo de texto que contiene *reglas* que le indican a **`make`** qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

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

Cuando se ejecuta, **`make`** busca los archivos **`GNUmakefile`**, **`makefile`**, y **`Makefile`**, en ese orden.

Contrucción de proyectos con `make`

Ejemplo de archivo `Makefile`

```
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 screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

Contrucción de proyectos con `make`

Ejemplo de archivo `Makefile`

```
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 screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Contrucción de proyectos con `make`

Ejemplo de archivo `Makefile`

```
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 screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Construir/compilar el proyecto **`editor`**

```
$ make
```

Contrucción de proyectos con `make`

Ejemplo de archivo `Makefile`

```
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 screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12
13 clean :
14   rm editor *.o
```

(Tiene 5 targets. Una por defecto.)

Construir/compilar el proyecto **`editor`**

```
$ make
```

(**`$ make clean`**)

Contrucción de proyectos con make

Probando make

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

Contrucción de proyectos con make

Probando make

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error

Contrucción de proyectos con `make`

Probando `make`

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`

Contrucción de proyectos con `make`

Probando `make`

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`

Contrucción de proyectos con `make`

Probando `make`

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`

Contrucción de proyectos con `make`

Probando `make`

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile

Contrucción de proyectos con make

Probando make

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile
- `$ touch b.h`

Contrucción de proyectos con make

Probando make

```
1  myapp: main.o 2.o 3.o
2    gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5    gcc -c main.c
6
7  2.o: 2.c a.h b.h
8    gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11   gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile
- `$ touch b.h`
- Eliminar el archivo `2.o`, y probar nuevamente

Contrucción de proyectos con make

Probando make. Comentarios y macros.

```
1  all: myapp
2
3  # Which compiler
4  CC = gcc
5
6  # Where are include file kept
7  INCLUDE = .
8
9  # Options for development
10 CFLAGS = -g -Wall -ansi
11
12 # Options for release
13 # CFLAGS = -O -Wall-ansi
14
15 myapp: main.o 2.o 3.o
16   $(CC) -o myapp main.o 2.o 3.o
17
18 main.o: main.c a.h
19   $(CC) -I$(INCLUDE) $(CFLAGS) -c main.c
20
21 2.o: 2.c a.h b.h
22   $(CC) -I$(INCLUDE) $(CFLAGS) -c 2.c
23
24 3.o: 3.c b.h c.h
25   $(CC) -I$(INCLUDE) $(CFLAGS) -c 3.c
```

($ rm myapp *.o, y $ make -f Makefile2)