

Advanced Unix,
Basic C,
Program Compilation

Simple C Program Editing

- Create a directory called CS300 in your Documents folder
- Change into the CS300 directory
- Open up a simple text editor called Geany in the Integrated Environment

```
chadd@ralph:~/Documents/CS300> geany &
```

- The & causes the program to be launched in the background so you can still use the command line.

Create the C program.

Differences from C++ ?

```
/* this is a comment */  
  
#include <stdio.h>  
  
int main ()  
{  
    printf ("hello world\n");  
    return 0;  
}
```

Save, Build, Execute

- Save your program in Documents/CS300 with the name helloworld.c
- Change into the CS300 directory to see that the file helloworld.c program exists
- Hit the Build button
 - only works with no configuration for simple projects
 - what shows up in the bottom window?
- Hit the Execute button
- List the contents of CS300 now

More UNIX Commands

| Command / Symbol | Meaning |
|---|---|
| <code>tar czf file.tar.gz files...</code> | use the tar utility to compress file(s) |
| <code>tar xzf file.tar.gz</code> | use the tar utility to decompress file(s) |
| <code>./a.out > outputfile</code> | save the executable results in outputfile |
| <code>./a.out >> outputfile</code> | append the execution results to the end of outputfile |
| <code>./a.out less</code> | pipe the output of a.out to the input of less (useful if the output results are more than a screen in length) |
| | |

Problems

- tar up the file helloworld.c
- Copy (not move) the tarred file to the parent directory
- Change to the parent directory and untar the file
- Compile the untarred file
- Run the executable
- Capture the execution results in a file called **rslts**
- Type the command less **rslts**

C Topics

- include

```
<stdio.h>
```

```
<stdlib.h>
```

```
/* this is a comment */
```

```
#include <stdio.h>
```

- printf/scanf

```
int main ()
```

```
{
```

- comments

```
    int value;
```

```
    scanf ("%d", &value);
```

```
    printf ("hello world %d\n",  
           value);
```

```
    return 0;
```

```
}
```

Build on the command line

```
gcc -Wall -o runMe helloworld.c -g
```

```
./runMe
```

- The ./ is necessary, why?

```
echo $PATH
```

```
gcc -Wall -c -o helloworld.o helloworld.c -g
```

```
gcc -Wall -o runMe helloworld.o -g
```

*Separate
Compilation

```
ls -altr
```

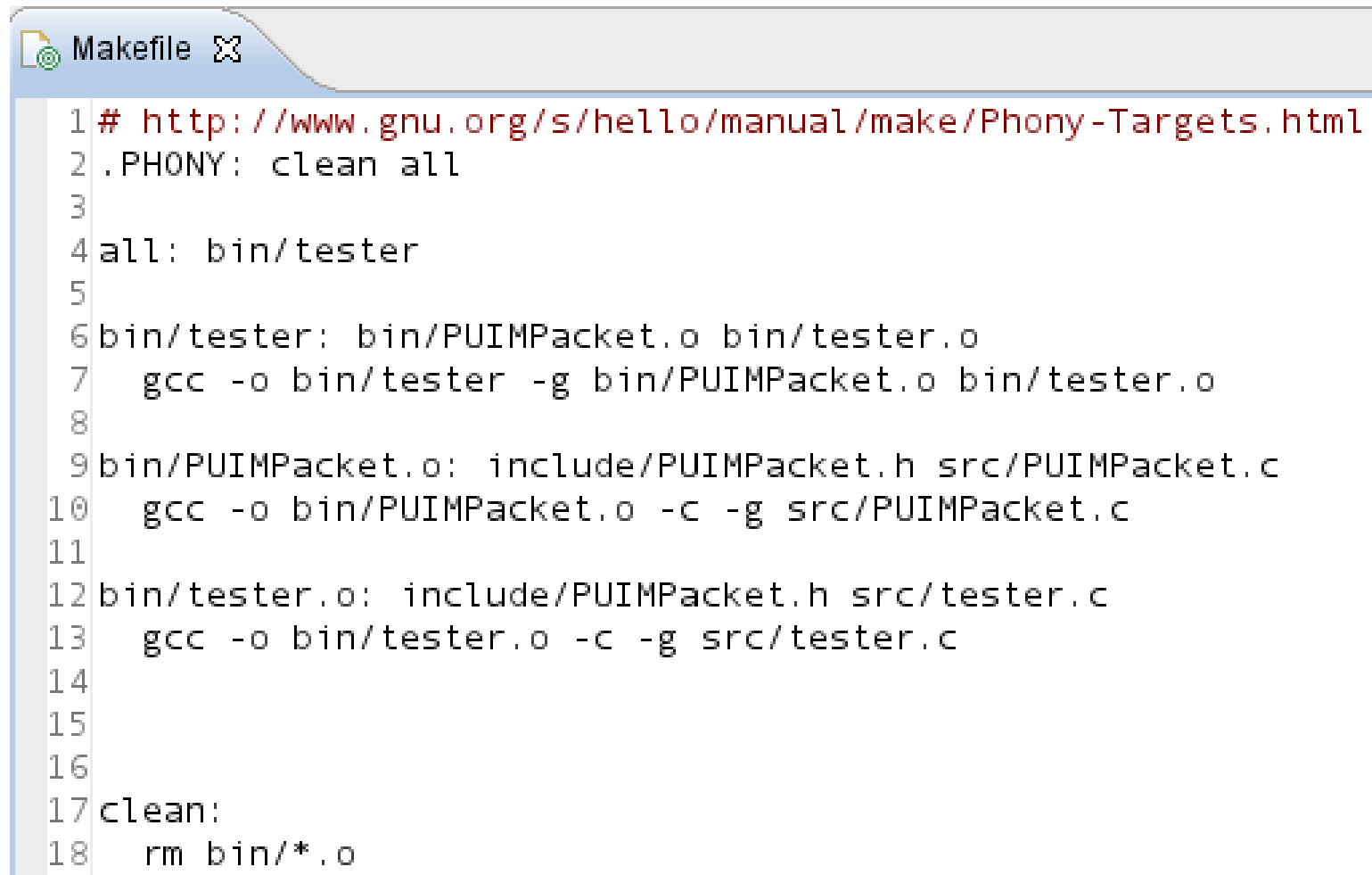
*Remember “Additional Dependencies” from CS250 Visual Studio
(Project Management -> Random.obj)

Makefiles

<http://www.eng.hawaii.edu/Tutor/Make/index.html>

- Description of how to build your executable
- useful if you have multiple source files
- GNU Make

make -h



```
Makefile 23
1 # http://www.gnu.org/s/hello/manual/make/Phony-Targets.html
2 .PHONY: clean all
3
4 all: bin/tester
5
6 bin/tester: bin/PUIMPacket.o bin/tester.o
7     gcc -o bin/tester -g bin/PUIMPacket.o bin/tester.o
8
9 bin/PUIMPacket.o: include/PUIMPacket.h src/PUIMPacket.c
10    gcc -o bin/PUIMPacket.o -c -g src/PUIMPacket.c
11
12 bin/tester.o: include/PUIMPacket.h src/tester.c
13    gcc -o bin/tester.o -c -g src/tester.c
14
15
16
17 clean:
18    rm bin/*.o
```

Makefile

```
target: dependency1 dependency2
    command1
    command2
```



tab!

Given a set of dependencies, make will only run the necessary commands to build the project. Build a **dependency graph**.

If a target is older than any of its dependencies the commands are run to build the target

target and dependencies are files

Command line

make tree

- looks for target named tree in Makefile and checks to see if it needs to be built

make

- looks for the first target in Makefile and checks to see if it needs to be built

Makefile

- Download Makefile Example from web

```
cd Downloads
```

```
tar xzf MakefileExampleCS300.tar.gz
```

```
cd MakefileExampleCS300
```

```
ls
```

```
geany Makefile include/* src/* &
```

- Let's look at the source code
 - rational.h
 - rational.c
 - driver.c

From the command line

- make
- make clean
- make driver
- make clean
- make tarball

C Topics

- `#ifdef` / `#ifndef`

```
#ifndef _EXAMPLE_  
#define _EXAMPLE_
```

- `#define`

- `static`

```
#include "localHdr.h"
```

- `array`

```
#define ARRAYSIZE 1024
```

- `include ""`

```
static int value;  
int bigArray[ARRAYSIZE];
```

```
#endif
```

POSIX

- Portable Operating System Interface for Unix
- standards for Unix
 - API
 - shells
 - utilities
- Provides portability of applications, scripts, etc.
 - cygwin provides POSIX support to Windows

man pages

- manual pages
 - man bash
 - man man
 - man ls

```
MAN(1)                                Manual pager utils                                MAN(1)

NAME
    man - an interface to the on-line reference manuals

SYNOPSIS
    man [-c|-w|-tZ] [-H[browser]] [-T[device]] [-X[dpi]] [-adhu7V] [-i|-I]
    [-m system[...]] [-L locale] [-p string] [-C file] [-M path] [-P
    pager] [-r prompt] [-S list] [-e extension] [--warnings [warnings]]
    [[section] page ...] ...
    man -l [-7] [-tZ] [-H[browser]] [-T[device]] [-X[dpi]] [-p string] [-P
    pager] [-r prompt] [--warnings[warnings]] file ...
    man -k [apropos options] regexp ...
    man -f [whatis options] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is
    normally the name of a program, utility or function. The manual page
    associated with each of these arguments is then found and displayed. A
    section, if provided, will direct man to look only in that section of
    the manual. The default action is to search in all of the available
    sections, following a pre-defined order and to show only the first page
    found, even if page exists in several sections.
```

also available online:

google → man bash

(may be different than what is on your machine)

man pages - Library Function

FOPEN(3)

Name

fopen, fdopen, freopen - stream open functions

Synopsis

```
#include <stdio.h>
```

```
FILE *fopen(const char *path, const char *mode);
```

Description

(arguments or command line options are listed here)

Return Value

Errors

See Also

Referenced By

manual sections

- 0 Header files (usually found in /usr/include)
- 1 Executable programs or shell commands
- 2 System calls (functions provided by the kernel)
- 3 Library calls (functions within program libraries)
- 4 Special files (usually found in /dev)
- 5 File formats and conventions eg /etc/passwd
- 6 Games
- 7 Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
- 8 System administration commands (usually only for root)
- 9 Kernel routines [Non standard]
p means POSIX!

Geany & Makefiles

Build | Set Includes and Arguments

Build: make

- When you press Build the Makefile will be invoked
 - make sure the Makefile is currently being displayed!
make all