Makefiles, Unix, and Project 1 Q&A

15-441: Recitation 2

September 12, 2007

Overview

Today, I will try to teach you three things. You may also learn something about:

- gcc
- make and Makefiles
- How to use Unix effectively

Project 1 Q&A

D.R.Y.

Rule 1: Don’t Repeat Yourself

D.R.Y.

Rule 1: Don’t Repeat Yourself

(except for emphasis)

Compiling a Program: gcc

Compile a source file into a binary

```
% gcc source.c lib.c –o binary
```

Compile a source file into a .o file

```
% gcc –c source.c
% gcc –c lib.c
```

Compile .o files into a binary

```
% gcc lib.o source.o –o binary
```

More advanced gcc

Compile a source file with debugging hooks

```
% gcc –g source.c –o binary
% gdb binary
```

Activate all warnings, treat warnings as errors

```
% gcc –Wall –Werror …
```

Compile with optimizations; speeds up binary

```
% gcc –O2 …
```

Set a predefined macro (same as #define DEBUG)

```
% gcc –DDEBUG …
```

Use cases

```
% gcc -g -Wall -Werror -c source.c -o source.o
% gcc -g -Wall -Werror -c lib.c -o lib.o
% gcc -g -Wall -Werror lib.o source.o -o binary
```

But…

Don’t Repeat Yourself

Makefiles: Simplify

```
% gcc –g –Wall –Werror –c source.c –o source.o
% gcc –g –Wall –Werror –c lib.c –o lib.o
% gcc –g –Wall –Werror lib.o source.o –o binary
```

Simplify and modularize…

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
```

Makefiles: Simplify

```
target: dependency1 dependency2
  Unix command (start line with a TAB)
  Unix command
  …
% gcc lib.o source.o –o binary

binary: lib.o source.o
  gcc lib.o source.o –o binary
```

```
binary: lib.o source.o
  gcc -g -Wall lib.o source.o –o binary

lib.o: lib.c
  gcc -g -Wall –c lib.c -o lib.o

source.o: source.c
  gcc -g -Wall –c source.c –o source.o

clean:
  rm *.o *~ binary
```

```
binary: lib.o source.o
  gcc -g -Wall lib.o source.o –o binary

lib.o: lib.c
  gcc -g -Wall –c lib.c -o lib.o

source.o: source.c
  gcc -g -Wall –c source.c –o source.o

clean:
  rm *.o *~ binary
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary

$(OUTPUT): lib.o source.o
  $(CC) $(CFLAGS) lib.o source.o –o $(OUTPUT)

lib.o: lib.c
  $(CC) $(CFLAGS) –c lib.c -o lib.o

source.o: source.c
  $(CC) $(CFLAGS) –c source.c –o source.o

clean:
  rm *.o *~ $(OUTPUT)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary

$(OUTPUT): lib.o source.o
  $(CC) $(CFLAGS) lib.o source.o –o $(OUTPUT)

lib.o: lib.c
  $(CC) $(CFLAGS) –c lib.c -o lib.o

source.o: source.c
  $(CC) $(CFLAGS) –c source.c –o source.o

clean:
  rm *.o *~ $(OUTPUT)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
OBJFILES = lib.o source.o

$(OUTPUT): $(OBJFILES)
  $(CC) $(CFLAGS) $(OBJFILES) –o $(OUTPUT)

lib.o: lib.c
  $(CC) $(CFLAGS) –c lib.c -o lib.o

source.o: source.c
  $(CC) $(CFLAGS) –c source.c –o source.o

clean:
  rm *.o *~ $(OUTPUT)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
OBJFILES = lib.o source.o

$(OUTPUT): $(OBJFILES)
  $(CC) $(CFLAGS) $(OBJFILES) –o $(OUTPUT)

lib.o: lib.c
  $(CC) $(CFLAGS) –c lib.c -o lib.o

source.o: source.c
  $(CC) $(CFLAGS) –c source.c –o source.o

clean:
  rm *.o *~ $(OUTPUT)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
OBJFILES = lib.o source.o

$(OUTPUT): $(OBJFILES)
  $(CC) $(CFLAGS) $(OBJFILES) –o $(OUTPUT)

%.o: %.c
  # $< = the source: %.c
  # $@ = the target: %.o
  $(CC) $(CFLAGS) –c $< -o $@

clean:
  rm *.o *~ $(OUTPUT)
```

Using a Makefile

```
% make
% make clean
M-x compile
```

That’s it!

More about Makefiles

Rule 2: Use the Google.

Google: “make tutorial”
Google: “makefile example”
Google: “makefile template”

Simple Scripting

```
% ./server 6667 &
% cat testfile.01 | ./testscript.py
% cat testfile.02 | ./testscript.py
% killall -9 server
```

Simple Scripting

```
#/bin/sh

echo “Starting server on port 6667.”
./server 6667 &
SERVERPID = $!

echo “Running test files.”
cat testfile.01 | ./testscript.py
cat testfile.02 | ./testscript.py

echo “Killing server process.”
kill $(SERVERPID)
```

```
#/bin/sh

TESTFILES=testfile.01 testfile.02
  testfile.03

echo “Starting server on port 6667.”
./server 6667 &
SERVERPID = $!

for testfile in $(TESTFILES); do
  echo “Running test file ‘$testfile’”
  cat $testfile | ./testscript.py
fi

echo “Killing server process.”
kill $(SERVERPID)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
OBJFILES = lib.o source.o

all: $(OUTPUT)

$(OUTPUT): $(OBJFILES)
  $(CC) $(CFLAGS) $(OBJFILES) –o $(OUTPUT)

%.o: %.c
  $(CC) $(CFLAGS) –c %< -o %@



clean:
  rm *.o *~ $(OUTPUT)
```

```
CC = gcc
CFLAGS = -g –Wall –Werror
OUTPUT = binary
OBJFILES = lib.o source.o

all: $(OUTPUT) test

$(OUTPUT): $(OBJFILES)
  $(CC) $(CFLAGS) $(OBJFILES) –o $(OUTPUT)

%.o: %.c
  $(CC) $(CFLAGS) –c %@ -o %<

test: $(OUTPUT)
  sh ./testscript.sh

clean:
  rm *.o *~ $(OUTPUT)
```

Unix

Debugging: `gdb, valgrind`
Scripting: Perl, Python, Ruby
We will cover these in later recitations…

Editors: `vi, emacs, ed, whatever`
Tools: `sed, awk, grep, find, wget, tar`
Installing Unix packages

Unix

Unix Philosophy:

Do one thing, and do it well.

Rule 3:

Learn one thing at a time, and learn it well.

How do I…

…find all instances of “func_name”:

```
% grep –r “func_name”
```

…replace all instances of `bad_func_name` to `good_func_name`:

```
% sed –e “s/bad_func_name/good_func_name/g”\
   source.c > source.c.new
```

How do I…

…find a file named “source.c”:

```
% find . –name “source.c”
```

…download a .tar.gz file from the Internet:

```
% wget http://address/to/file.tar.gz
```

…untar and unzip the tarball:

```
% tar xzvf file.tar.gz
```

How do I...

…install a package from source:

```
% wget http://address/to/file.tar.gz
% tar xzvf file.tar.gz
% cd file
% less README (always important!)
% ./configure
% make && make install
```

In General…

Use the keyboard.

(Touching the mouse is a cache miss.)

Project 1

Checkpoint 1: Any problems?

Checkpoint 2: Be ready for it soon!

Due date: It will come faster than you think.

Questions?