Compiling from source

networking lecture & practice for CSC420 Operating Systems Spring 2022 Lyon College

README

- This file accompanies lectures on the shell and `bash(1)`. To gain practice, you should type along in your own Org-mode file. You have to have Emacs and my `.emacs` file installed on your PC or the Pi you're working with.
- This section is based on chapter 16 of Shotts, The Linux Command Line (2e), NoStarch Press (2019) - "Networking".
- To make this easier, use the auto expansion (`<s`). This will only work if you have my `.emacs` file (from GDrive) installed and the `org-tempo` library loaded.

- Add the following two lines at the top of your file, and activate each line with `C-c C-c` (this is confirmed in the echo area as `Local setup has been refreshed`)):

```
#+PROPERTY: header-args:bash :results output :exports both
```

- Remember that `C-M-\` inside a code block indents syntactically (on Windows, this may only work if you have a marked region - set the mark with `C-SPC`).

- To **not** see the emphatic characters like ~ or * or / in the Org file text, run the following code chunk (or put the code in your `/.emacs` file): if successful, you should see `"t"` in the minibuffer.

```
(setq-default org-hide-emphasis-markers t)
```

  If you don't put it in your `/.emacs` file, the command will only work for the current Emacs session.

- If you have difficulty distinguishing the code blocks from the documentation, change your Emacs theme with `M-x custom-themes` - `Leuven` is great if you like a light theme, or `Manoj-dark` if you like it dark.

Overview

- The **availability of source code** is the essential freedom that makes Linux possible: the ecosystem of Linux development relies on free exchange of code among developers.
- For desktop users, **compiling** is a lost art: it used to be common. Now, distribution ("distro") providers maintain pre-compiled binaries ready for download and use.
- Example: the Debian repository contains more than 70,000 packages.
- [ ]

  Why compile software at all?

  1. Availability: some distros may not contain all the desired application for your computer architecture.
  2. Timeliness: many distros do not have the latest, most secure, fastest versions of a program available.

- In practice, compilation relies on four programs:

- obtaining (Linux: `wget`, `sftp`, `curl`)
- unpacking (Linux: `tar`)
- configuring (Linux: `configure`)
- building (Linux: `make`)

- Example in this script: compiling a C program (following Shotts, chapter 23).

What is compiling?

1. Conversion of high-level programs into machine language
2. Linking common task libraries to the program
3. Checking types, pre-debugging and other housekeeping tasks

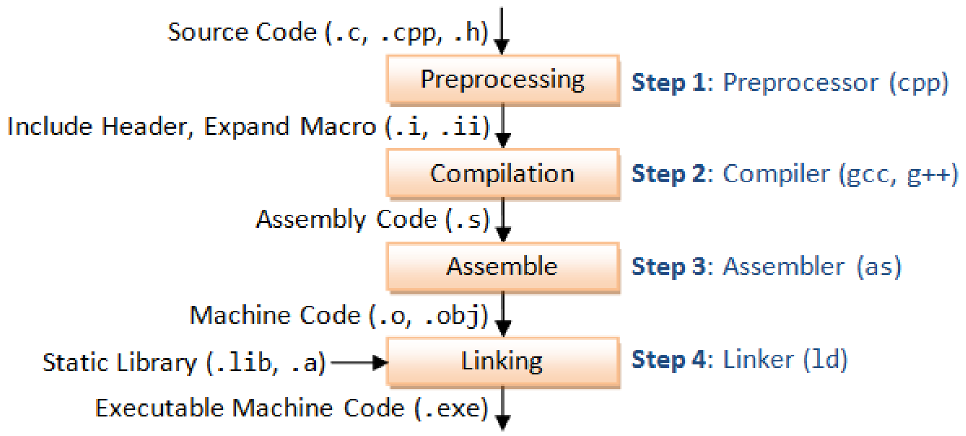
Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

Figure 1: GCC compilation process (Source: Hock-Chuan, 2018).

Why are interpreted languages so popular?

- Languages like PHP, Python, R, Ruby, perl etc.
- Faster development cycles because: no compilation
- Downside: interpreted languages are significantly slower
- Interface libraries can help somewhat (e.g. Rcpp for R)
- Shell script languages (like `bash(1)`) are not slow

Getting the compiler

- See the FAQ for detailed instructions on how to get GCC, the GNU compiler for (among many others) C and C++.
- [ ]

  Check that GCC is available on your computer.

```
which gcc
```

```
/usr/bin/gcc
```

- `[ ]`

  Check the version of GCC.

```
gcc --version
```

```
gcc (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

  On my Windows box:

```
: gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
: Copyright (C) 2019 Free Software Foundation, Inc.
: This is free software; see the source for copying conditions.  There is NO
: warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Obtaining the source code

- We will compiler a program from the GNU project called `diction`.
- `[ ]`

  Create a directory for the source code named `src` and then download the source code into it using `ftp`.

```
mkdir src
cd src
ftp ftp.gnu.org
```

- `[ ]`

  On the FTP server, change to `gnu/diction` with `cd` and `get` the latest version of the `tar` archive file.

  Copy of the screen dialog:

```
pi@raspberrypi:~/GitHub/org$ ftp ftp.gnu.org
Connected to ftp.gnu.org.
220 GNU FTP server ready.
Name (ftp.gnu.org:pi): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd gnu/diction
250 Directory successfully changed.

ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 3003     65534       68940 Aug 28  1998 diction-0.7.tar.gz
-rw-r--r--    1 3003     65534       90957 Mar 04  2002 diction-1.02.tar.gz
-rw-r--r--    1 3003     65534      141062 Sep 17  2007 diction-1.11.tar.gz
-rw-r--r--    1 3003     65534         189 Sep 17  2007 diction-1.11.tar.gz.sig
```

```
226 Directory send OK.
ftp> get diction-1.11.tar.gz
local: diction-1.11.tar.gz remote: diction-1.11.tar.gz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for diction-1.11.tar.gz (141062 bytes).
226 Transfer complete.
141062 bytes received in 0.51 secs (268.8837 kB/s)
ftp> bye
221 Goodbye.
pi@raspberrypi:~/GitHub/org$ ls *tar*
diction-1.11.tar.gz
pi@raspberrypi:~/GitHub/org$
```

- [ ]

  We could also download the source code using the `wget` program:

  ```
  wget https://ftp.gnu.org/gnu/diction/diction-1.11.tar.gz
  ```

  Copy of the screen dialog:

  ```
  pi@raspberrypi:~/Downloads$ wget https://ftp.gnu.org/gnu/diction/diction-1.11.tar.gz
  --2022-05-02 22:23:11--  https://ftp.gnu.org/gnu/diction/diction-1.11.tar.gz
  Resolving ftp.gnu.org (ftp.gnu.org)... 209.51.188.20, 2001:470:142:3::b
  Connecting to ftp.gnu.org (ftp.gnu.org)|209.51.188.20|:443... connected.
  HTTP request sent, awaiting response... 200 OK
  Length: 141062 (138K) [application/x-gzip]
  Saving to: ‘diction-1.11.tar.gz’

  diction-1.11.tar.gz   100%[=====================>] 137.76K   679KB/s    in 0.2s

  2022-05-02 22:23:12 (679 KB/s) - ‘diction-1.11.tar.gz’ saved [141062/141062]

  pi@raspberrypi:~/Downloads$
  ```

Unpacking the archive

- Source code is usually supplied in the form of a compressed so-called *tarball*. It contains the *source tree*, a hierarchy of directories and files.
- [ ]

  You can look at the tarball using Emacs `Dired`:

  Top of the tarball for `diction`:

  ```
  -rw-r--r-- michael/user      35068 diction-1.11/COPYING
  -rw-r--r-- michael/user       9416 diction-1.11/INSTALL
  -rw-r--r-- michael/user       3920 diction-1.11/Makefile.in
  -rw-r--r-- michael/user       1448 diction-1.11/README

  -rw-r--r-- michael/user        152 diction-1.11/NEWS
  -rwxr-xr-x michael/user     144080 diction-1.11/configure
  -rwxr-xr-x michael/user      13184 diction-1.11/install-sh
  ```

- [ ]

You can also look at the tarball without Emacs:

```
cd src
tar tzvf diction-1.11.tar.gz | head
```

```
-rw-r--r-- michael/user  35068 2007-07-30 15:47 diction-1.11/COPYING
-rw-r--r-- michael/user   9416 2007-08-03 02:03 diction-1.11/INSTALL
-rw-r--r-- michael/user   3920 2007-08-03 05:05 diction-1.11/Makefile.in
-rw-r--r-- michael/user   1448 2007-08-30 05:20 diction-1.11/README
-rw-r--r-- michael/user    152 2007-08-30 03:08 diction-1.11/NEWS
-rwxr-xr-x michael/user 144080 2007-08-30 03:06 diction-1.11/configure
-rwxr-xr-x michael/user  13184 2007-08-03 02:03 diction-1.11/install-sh
-rw-r--r-- michael/user   2621 2007-03-30 16:45 diction-1.11/de
-rw-r--r-- michael/user  24830 2007-03-30 16:45 diction-1.11/en
-rw-r--r-- michael/user  25043 2007-03-30 16:45 diction-1.11/en_GB
```

- `[ ]`

  Unpack the archive with the `tar` command:

```
cd src
tar xzf diction-1.11.tar.gz
ls -l
```

```
total 144
drwxr-xr-x 3 pi pi   4096 May  2 23:18 diction-1.11
-rw-r--r-- 1 pi pi 141062 Sep 17  2007 diction-1.11.tar.gz
```

Examining the source tree

- `[ ]`

  Examine the source tree.

```
cd src
cd diction-1.11
ls -F
```

```
config.guess*
config.h
config.h.in
config.log
config.status*
config.sub*
configure*
configure.in
COPYING
de
de.po
diction*
diction.1
diction.1.in
diction.c
diction.o
diction.pot
diction.spec
diction.spec.in
diction.texi
diction.texi.in
en
en_GB
en_GB.po
getopt1.c
getopt1.o
getopt.c
getopt.h
getopt_int.h
getopt.o
INSTALL
install-sh*
Makefile
Makefile.in
misc.c
misc.h
misc.o
NEWS
nl
nl.po
README
sentence.c
sentence.h
sentence.o
style*
style.1
style.1.in
style.c
style.o
test/
```

- Always carefully read the files `README` and `INSTALL`.
- `[ ]`

  The `.c` files contain the two C programs supplied by the package, *style* and *diction*.

  ```
  cd src/diction-1.11
  ls *.c
  ```

  ```
  diction.c
  getopt1.c
  getopt.c
  misc.c
  sentence.c
  style.c
  ```

- `[ ]`

  The `.h` files contain descriptions of libraries to be linked.

  E.g. at the top of the `diction.c` source file:

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "getopt.h"
#include "misc.h"
#include "sentence.h"
```

The first group lives outside the source tree - to find these files, they need to be in the `PATH`. They were installed with the compiler, GCC.

The second group consists of header files that live in the source tree.

Building the program

- Most programs build with a simple, two-command sequence:

```
./configure
make
```

- `configure` is a shell script that is supplied with the source tree. Its job is to analyze the *build environment*.
- Most code is meant to be *portable* - but small changes usually need to be made during the build to accommodate differences.
- `configure` checks that the necessary external tools and components are installed and ready to run.
- `[ ]`

Run configure - prefix the program name with the current directory locator (period): run in a shell - the command produces a lot of messages.

```
./configure
```

Sample screen output:

```
pi@raspberrypi:~/GitHub/admin/spring22/os420/src/diction-1.11$ ./configure
checking build system type... armv7l-unknown-linux-gnu
checking host system type... armv7l-unknown-linux-gnu
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...

checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for a BSD-compatible install... /usr/bin/install -c
checking for strerror... yes
checking for library containing regcomp... none required
checking for broken realloc... no
checking for msgfmt... no
configure: creating ./config.status
config.status: creating Makefile
config.status: creating diction.1
config.status: creating diction.texi
config.status: creating diction.spec
config.status: creating style.1
config.status: creating test/rundiction
config.status: creating config.h
pi@raspberrypi:~/GitHub/admin/spring22/os420/src/diction-1.11$
```

- At the end, `configure` created several new files in the source directory.

  ```
  ls -lt src/diction-1.11
  ```

  Output:

  ```
  total 684
  -rw-r--r-- 1 pi pi   9591 May  2 22:50 config.log
  -rw-r--r-- 1 pi pi    350 May  2 22:50 config.h
  drwxr-xr-x 2 pi pi   4096 May  2 22:50 test
  -rw-r--r-- 1 pi pi  11987 May  2 22:50 style.1
  -rw-r--r-- 1 pi pi   1059 May  2 22:50 diction.spec
  -rw-r--r-- 1 pi pi   8994 May  2 22:50 diction.texi
  -rw-r--r-- 1 pi pi   4737 May  2 22:50 diction.1
  -rw-r--r-- 1 pi pi   4320 May  2 22:50 Makefile
  -rwxr-xr-x 1 pi pi  23676 May  2 22:50 config.status
  ```

- The most important one is the `Makefile`. It is a configuration file that instructs the `make` program exactly how to build the program.
- `[ ]`

  `Makefile` is an ordinary text file, so you can view it.

  ```
  less Makefile
  ```

- The `Makefile` contains *flags* (like `CC=gcc`) and *targets* (like `diction.o:`):

  ```
  CC=         gcc
  ...
  diction.o:    diction.c config.h getopt.h misc.h sentence.h
  ```

  The command specified to build `diction.o` is handled by a general target that compiles *any* `.c` file into an `.o` file:

  ```
  .c.o:
              $(CC) -c $(CPPFLAGS) $(CFLAGS) $<
  ```

- `[ ]`

  Run `make` in the `src` directory. It produces this output:

  ```
  gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
  gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
  gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
  gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
  ```

```
gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
gcc -o diction -g diction.o sentence.o misc.o \
        getopt.o getopt1.o
gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
gcc -o style -g style.o sentence.o misc.o \
        getopt.o getopt1.o -lm
```

- `[ ]`

  All the targets from the `Makefile` are now present in our directory, including the main programs `diction` and `style`. Run `ls` to confirm this.

```
ls -lt src/diction-1.11
```

```
total 952
-rwxr-xr-x 1 pi pi  77684 May  2 23:13 style
-rwxr-xr-x 1 pi pi  44500 May  2 23:13 diction
-rw-r--r-- 1 pi pi   1844 May  2 23:13 getopt.o
-rw-r--r-- 1 pi pi  79064 May  2 23:01 style.o
-rw-r--r-- 1 pi pi   1844 May  2 23:01 getopt1.o
-rw-r--r-- 1 pi pi   1836 May  2 23:01 misc.o
-rw-r--r-- 1 pi pi  20540 May  2 23:01 sentence.o
-rw-r--r-- 1 pi pi  30648 May  2 23:01 diction.o
-rw-r--r-- 1 pi pi   9591 May  2 22:50 config.log
-rw-r--r-- 1 pi pi    350 May  2 22:50 config.h
-rw-r--r-- 1 pi pi  11987 May  2 22:50 style.1
-rw-r--r-- 1 pi pi   8994 May  2 22:50 diction.texi
-rw-r--r-- 1 pi pi   4737 May  2 22:50 diction.1
-rw-r--r-- 1 pi pi   4320 May  2 22:50 Makefile
-rwxr-xr-x 1 pi pi  23676 May  2 22:50 config.status
drwxr-xr-x 2 pi pi   4096 Sep 13  2007 test
-rw-r--r-- 1 pi pi   1059 Sep  7  2007 diction.spec
-rw-r--r-- 1 pi pi   1448 Aug 30  2007 README
-rw-r--r-- 1 pi pi    152 Aug 30  2007 NEWS
-rwxr-xr-x 1 pi pi 144080 Aug 30  2007 configure
-rw-r--r-- 1 pi pi   1709 Aug 30  2007 configure.in
-rw-r--r-- 1 pi pi  11489 Aug  9  2007 en_GB.po
-rw-r--r-- 1 pi pi  12157 Aug  9  2007 nl.po
-rw-r--r-- 1 pi pi  11946 Aug  9  2007 de.po
-rw-r--r-- 1 pi pi   8001 Aug  9  2007 diction.pot
-rw-r--r-- 1 pi pi  37859 Aug  9  2007 style.c
-rw-r--r-- 1 pi pi   7468 Aug  9  2007 sentence.c
-rw-r--r-- 1 pi pi  18210 Aug  9  2007 nl
-rw-r--r-- 1 pi pi   4723 Aug  3  2007 diction.1.in
-rw-r--r-- 1 pi pi  11979 Aug  3  2007 style.1.in
-rw-r--r-- 1 pi pi   3920 Aug  3  2007 Makefile.in
-rw-r--r-- 1 pi pi   9416 Aug  3  2007 INSTALL
-rwxr-xr-x 1 pi pi  13184 Aug  3  2007 install-sh
-rwxr-xr-x 1 pi pi  32603 Aug  3  2007 config.sub
-rwxr-xr-x 1 pi pi  44347 Aug  3  2007 config.guess
-rw-r--r-- 1 pi pi    640 Jul 31  2007 misc.h
-rw-r--r-- 1 pi pi   1264 Jul 31  2007 misc.c
-rw-r--r-- 1 pi pi  35068 Jul 30  2007 COPYING
-rw-r--r-- 1 pi pi   1092 Jul 30  2007 sentence.h
-rw-r--r-- 1 pi pi   8990 Jul 30  2007 diction.texi.in
-rw-r--r-- 1 pi pi  12235 Jul 30  2007 diction.c
-rw-r--r-- 1 pi pi    279 Mar 30  2007 config.h.in
-rw-r--r-- 1 pi pi   2621 Mar 30  2007 de
-rw-r--r-- 1 pi pi   1064 Mar 30  2007 diction.spec.in
-rw-r--r-- 1 pi pi  24830 Mar 30  2007 en
```

```
-rw-r--r-- 1 pi pi  25043 Mar 30  2007 en_GB
-rw-r--r-- 1 pi pi   4776 Mar 30  2007 getopt1.c
-rw-r--r-- 1 pi pi  33125 Mar 30  2007 getopt.c
-rw-r--r-- 1 pi pi   5729 Mar 30  2007 getopt.h
-rw-r--r-- 1 pi pi   4766 Mar 30  2007 getopt_int.h
```

- `[ ]`

  Now run `make` again! The message appears:

  ```
  make: Nothing to be done for 'all'.
  ```

- The `make` program only builds what needs building and checks all dependencies. To show this, get rid of some intermediate programs and run `make` again:

  ```
  cd src/diction-1.11
  rm getopt.o
  make
  ```

  ```
  gcc -c -I. -DSHAREDIR=\"/usr/local/share\" -DLOCALEDIR=\"/usr/local/share/locale\"
  gcc -o diction -g diction.o sentence.o misc.o \
          getopt.o getopt1.o
  gcc -o style -g style.o sentence.o misc.o \
          getopt.o getopt1.o -lm
  ```

- You see that `make` rebuilds and relinks the main programs `diction` and `style` because they depend on the missing module.
- `make` also keeps targets up to date and ensures that all code is built using the most recent source code.
- `[ ]`

  Use the `touch` program to "update" one of the source code files - as if a programmer had changed `getopt.c`.

  ```
  cd src/diction-1.11
  ls -l diction getopt.c
  ```

  ```
  -rwxr-xr-x 1 pi pi 44500 May  2 23:18 diction
  -rw-r--r-- 1 pi pi 33125 Mar 30  2007 getopt.c
  ```

  ```
  cd src/diction-1.11
  touch getopt.c
  ls -l diction getopt.c
  ```

  ```
  -rwxr-xr-x 1 pi pi 44500 May  2 23:18 diction
  -rw-r--r-- 1 pi pi 33125 May  2 23:18 getopt.c
  ```

  `getopt.c` is now more recent than the built file, and `make` will discover and restore the target to being newer than the dependency.

```
cd src/diction-1.11
make &> /dev/null
ls -l diction getopt.c
```

```
-rwxr-xr-x 1 pi pi 44500 May  2 23:18 diction
-rw-r--r-- 1 pi pi 33125 May  2 23:18 getopt.c
```

Installing the program

- The special `make` target `install` will install the final product in the operating system ready for use. Usually, this is `/usr/local/bin`, which is not writable for regular users, so we must use `sudo`.

```
cd src/diction-1.11
sudo make install
which diction
```

```
[ -d /usr/local/bin ] || /usr/bin/install -c -m 755 -d /usr/local/bin
/usr/bin/install -c diction /usr/local/bin/diction
/usr/bin/install -c style /usr/local/bin/style
/usr/bin/install -c -m 755 -d /usr/local/share/diction
/usr/bin/install -c -m 644 ./de /usr/local/share/diction/de
/usr/bin/install -c -m 644 ./en /usr/local/share/diction/en
(cd /usr/local/share/diction; rm -f C; ln en C)
/usr/bin/install -c -m 644 ./en_GB /usr/local/share/diction/en_GB
/usr/bin/install -c -m 644 ./nl /usr/local/share/diction/nl
[ -d /usr/local/share/man/man1 ] || /usr/bin/install -c -m 755 -d /usr/local/share
/usr/bin/install -c -m 644 diction.1 /usr/local/share/man/man1/diction.1
/usr/bin/install -c -m 644 style.1 /usr/local/share/man/man1/style.1
make install-po-no
make[1]: Entering directory '/home/pi/GitHub/admin/spring22/os420/src/diction-1.11
make[1]: Nothing to be done for 'install-po-no'.
make[1]: Leaving directory '/home/pi/GitHub/admin/spring22/os420/src/diction-1.11'
/usr/local/bin/diction
```

Reference

Hock-Chuan (2018). GCC and Make: Compiling, Linking and Building C/C++ Applications [website]. URL: ntu.edu.sg.

Shotts (2019). The Linux Command-Line: A Complete Introduction. NoStarch Press. URL: linuxcommand.org.

Author: Marcus Birkenkrahe
Created: 2022-05-02 Mon 23:18
Validate