

Course 7: Programming Practises

Kristaps Dzonsons

12 December, 2011

Course site: *http://kristaps.bsd.lv/minicourse_12_2011*

Problem: How do I write programmes?

In this mini-course, I cover some helpful topics in programming practises.

- ① editing code
- ② accessing resources
- ③ compiling, managing dependencies
- ④ debugging
- ⑤ versioning
- ⑥ documenting

Where and how does one “write code” efficiently on UNIX systems? By using an editor. My choice: `vim`, a popular descendant of 2BSD's `vi`.

```
% vimtutor  
% vim example.c
```

UNIX is well-known for its built-in manual pages (“manpages”). These are accessed by `man`, and searched with `whatis` and `apropos`.

```
% apropos mpi
% whatis pthreads
% man pthread_create
```

Always read a function’s manpage before you use that function!

Compiling, Managing Dependencies

Many times we've invoked `cc` to compile our code. What if we have multiple files, or complex arguments for `cc`? The `make` utility can manage compilation for us.

```
% cat Makefile
CFLAGS += -I/usr/local/include
LDFLAGS += -L/usr/local/lib
LIBS = -lmpi -lpthread

example8: example8.o
    $(CC) $(CFLAGS) $(LDFLAGS) -o example8 example8.o $(LIBS)
% make
% make
% touch example8.c
% make
```

Your executable will crash. What do you do?

- ① make sure coredumps are enabled
- ② make sure you compile with debugging symbols
- ③ run the debugger over faulting executables

```
% ulimit -c unlimited
% cc -g -W -Wall willfail.c
% ./a.out
Abort trap (core dumped)
% gdb a.out a.out.core
(gdb) backtrace
#0  0x0000000201c6130a in kill () from /usr/lib/libc.so.60.1
#1  0x0000000201cc3121 in abort () at ...libc/stdlib/abort.c:68
#2  0x0000000000400969 in main () at foo.c:1
```

Versioning: Single-User

Versioning is extremely powerful: it lets us keep track of the development of our utilities. UNIX has many built-in versioning tools. For a single-developer system, `rcs` may be used.

```
% mkdir RCS  
% vim code.c  
% ci -l code.c
```

Versioning: Multi-User

For a multi-developer system, `cvs` (Concurrent Versioning System) suits most needs. It extends `rcs` to a client-server model.

```
% cvs up
% vim code.c
% cvs commit
```


Documentation is so critical it almost goes without saying (unfortunately, it often does go without saying – with unfortunate results). If you write an executable utility, you can write your own manpage for it in the *mdoc* language. Then, other users of your system will know what it does and how to run it.

It's sometimes customary to create a *README* file in your package directory for compilation of your utility (which doesn't really belong in the manpage).

On `gamelab`, those with accounts can also publish sources. See me privately for how to orchestrate this.