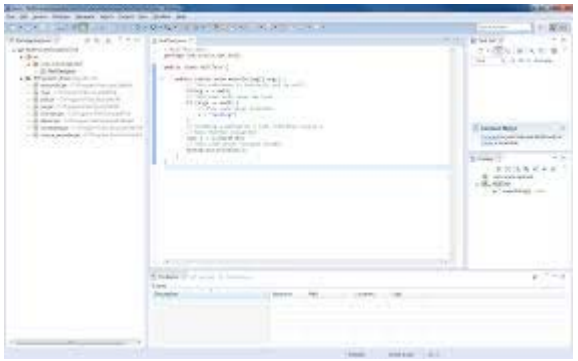
UNIX/Linux environment

C programming tools

Stefano Quer and Pietro Laface
Dipartimento di Automatica e Informatica
Politecnico di Torino

Introduction

- Integrated Development Environment (IDE)
  - Code::Blocks
    - C, C++, Fortran
    - http://www.codeblocks.org/
  - Eclipse
    - Java, C++, etc.
    - http://www.eclipse.org/
  - Geany
    - Very simple, few plug-ins
  - MonoDevelop
  - Anjuta



Introduction

- Editor
  - **VI (VIM)**
  - gEdit
  - geany
  - Nano
  - Pico
  - **Emacs**
  - Xemacs (next generation Emacs)
- Compiler
  - GCC
  - G++
  - Makefile
  - Configure
- Debugger
  - GDB

Editor: VI (VIM)

- Standard text editor of all Unix systems
- In its base version
  - Is not functional for extensive file editing
  - Very useful if other editors cannot be used, or give some problem
- Extended and improved
  - VIM = VI Improved
  - In the newer versions can be used for editing large projects

Editor: VI (VIM)

Action	Command
Insert mode	i
Command mode	esc
Cursor movements	←,↑,→,↓ (h, j, k, l)
Delete a character	x
Undo	U
Redo	Ctrl-r
Save file	ZZ or :w!
Quit	:q!
Show help	:help

Documentation
Local help : man vim
Online resources: http://www.vim.org/docs.php
Resources in PDF: ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf

Editor: emacs

- Preferred by many advanced programmers: powerful, extensible, flexible
- Various versions, but the most popular are GNU Emacs e XEmacs
- Available for
  - GNU, GNU/Linux
  - FreeBSDm, NetBSD, OpenBSD
  - Mac OS X
  - MS Windows

Editor: emacs

- Base commands available through
  - Menu
  - Character sequences
    - Control commands: control + character (c-key)
    - Meta commands: alt + character (m-key)

Documentation
Local help : man emacs
Online resources : http://www.gnu.org/software/emacs/manual/emacs.html
Resources in PDF: http://www.gnu.org/software/emacs/maanual/pdf/emacs.pdf

Compiler: gcc

- Open-Source GNU project
  - gcc compiler and linker
  - Supports C and C++

Documentation
Local help : man gcc
Online resources : http://www.gnu.org

- Command syntax
  - gcc <options> <arguments>
    - Options: list of flags that control the compiler and the linker
    - Arguments: list of files that gcc reads and process depending on the given options

Examples

- Compilation of a set of files that produces the corresponding object files
  - gcc –c file1.c
  - gcc –c file2.c
  - gcc –c main.c
- Link of the object files produces the executable file
  - gcc –o myexe file1.o file2.o main.o
- Compilation and linking with a single command
  - gcc –o myexe file1.c file2.c main.c

gcc options

- Most common options
  - -c file
    - Compilation only
  - -o file
    - Specifies the executable name
  - -g
    - gcc does not produce optimized code, but inserts additional information useful for debugging (see gdb)
  - -Wall
    - Output a warning for all possible code errors

gcc options

- -Idir
  - Additional header file search directories
    - More than one directory can be specified
  - There are no spaces between –I and the directory name
- -lm
  - Specifies to use the math library
- -Ldir
  - Specifies the search directories for pre-existing libraries to be linked

Example

```
gcc –Wall –g –I. –I/myDir/subDir –o myexe \
  myMain.c \
  fileLib1.c fileLib2.c file1.c \
  file2.c file3.c -lm
```

- Compilation of many source files, followed by linking and creation of the executable file
  - Multi-row command
  - Provides "All Warnings"
  - Debug option
  - Find the header files in two directories
  - Links the math library

Make

- Make
  - Is an automation tool that automatically builds executable programs and libraries from source code by reading a file called Makefile which specify how to derive the target program
- Help
  - man make

Makefile

- Make has two main aims
  - Automatically perform repetitive tasks
  - Avoid (re)doing unnecessary tasks
    - by verifying the file dependencies and modification times (e.g., **re-compile** only the **files** that have been **modified** since the previous make command
- Make interprets a file, called **`Makefile`**
    - A text file similar to a shell script

Make options

- Make can be executed using different options
  - Does not execute, just displays the commands
    - -n
  - Ignores possible errors and proceeds with the next commands
    - -i, --ignore-errors
  - Output debug information during the execution
    - -d
  - --debug=[options]
    - Options: a = print all info, b = basic info, v = verbose = basic + other, i = implicit = verbose + other

Makefile options

- **`make`** can take as argument a source file,
  rather than the standard ones (**`makefile`** or **`Makefile`**)
  - **`make`**
    - Executes the default **`Makefile`** in the current directory
  - **`make --file`** or **`-f`**
    - Executes the specified filename
      - **`make –f Makefile`**
      - **`make –f <myMakefile>`**

Makefile format

```
target: dependency
  <tab>command
```

- A **`Makefile`** specifies one or more targets
- Executing a **`Makefile`**
  - by default, the target is the first target
  - If more targets are specified, the desired target can be passed as an argument to make
    - `make < targetName >`
    - `make –f <myMakefile> <targetName>`

Makefile format

- A makefile consists of "rules" like this:

```
target: dependency
  <tab>command
```

- A target includes
  - Target Name
  - dependency list that must be verified to execute the target
  - Command
    - A command is preceded by a mandatory **TAB** character

Example 1

```
target:
  <tab>gcc –Wall –o myExe main.c
```

Notice: TAB

- Specifies
  - A single target with name **target**
  - The target does not have dependencies
- The target is obtained executing the **`makefile`**
- The target is in this case a compilation command

Example 2

```
target:
  <tab>gcc -Wall -o my main.c
  <tab>cp my /home/myuser/bin

clean:
  <tab>rm -rf *.o *.txt
```

- Multiple targets
  - It is necessary to indicate the selected the target
  - The first target is the default target
- The first target executes two commands (**`gcc`** and **`cp`**)

Example 2

```
target:
  <tab>gcc -Wall -o my main.c
  <tab>cp my /home/myuser/bin

clean:
  <tab>rm -rf *.o *.txt
```

- Explicit selection of the first target
  - **`make target`**
  - **`make -f Makefile target`**
- Selection of the second target
  - **`make clean`**
  - **`make -f Makefile clean`**

Example 3

```
target: file1.o file2.o
  <tab>gcc –Wall –o myExe file1.o file2.o

compile:
  <tab>gcc –Wall –g –I./dirI –c file1.c
  <tab>gcc –Wall –g –I./dirI –c file1.c
```

- **`target: file1.o file2.o`** indicates the presence of dependencies
- Make
  - Verifies if the target dependencies are more recent than the target
  - In such a case the target is recursively built

Example 3

```
target: file1.o file2.o
  <tab>gcc -Wall -o myExe file1.o file2.o

compile:
  <tab>gcc -Wall -g -I./dirI -c file1.c
  <tab>gcc -Wall -g -I./dirI -c file1.c
```

- If the target is **`filename.o`**, there is an automatic dependency with respect to the **`filename.c`**
- Inclusion file dependencies are automatically verified
  - It is not necessary to explicitly analyze all dependencies recursively

Debugger: gdb

- GNU debugger **gdb** is available for almost all Operating Systems
- It can be used
  - As a stand-alone tool
  - Integrated with the emacs editor
  - Embedded in some graphical IDE
- Abbreviate form of commands can be given

Debugger: gdb

Action	Command
Execution commands	run (r) <arguments> next (n) next <number of times> step (s) step <number of times> stepi (si) finish (f) continue (c)
Breakpoint commands	info break break (b) break LineNumber break FunctionName break fileName:LineNumber disable BreakpointNumber enable BreakpointNumber

Debugger: gdb

Action	Command
Print commands	print (p) print expression display expression
Stack operations	down (d) up (u)
Code listing commands	list (p) list LineNumber list FirstLine, LastLine
Miscellaneous commands	file fileName exec fileName

http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf