INTRODUCTION TO MAKE

SUPPLEMENTARY MATERIAL

Bin Zhang ◆ NPU (binzhang@mail.npu.edu)

Introduction to make

- What is make
  - make is a programming utility that maintains, updates, and regenerates related programs and files
    - make automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them
  - make is originated from UNIX, but it is now also available on Windows (with better GUI)
  - make can be used with any programming language

Makefile

- Makefile: contains rules that describe how to bring a target up to date with respect to those on which it depends
  - makefile is an input file to make
  - makefile development is an integral part of software development
    - In a small development environment, makefile is developed by a programmer
    - In a large development environment, makefile is developed by a configuration management engineer

Makefile Rules

- What is a rule
  - A rule appears in the makefile and says when and how to remake certain files, called the rule's targets. It lists the other files that are the dependencies of the target, and commands to use to create or update the target
- Rule format

```
target ... : dependencies ...
        command
        ....
```

  - target: file names (wildcards are ok)
  - dependencies: file names separated by spaces
  - command: shell commands (must start with a tab!)

makefile Example 1

- makefile example: version 1
  - `mf_example_1`
  - Try these two commands
    - `make -f mf_example_1`
      - execute the first target (the default)
    - `make -f mf_example_1 clean`
      - execute the target clean
- The concept of a dependency tree

The sample code is at /home/NPU_NETLAB/bzhang/class/cs510/code/make_examples

makefile Example 2

- makefile example: version 2
  - `mf_example_2`
- What is new in version 2
  - Variables

makefile Example 3

- makefile example: version 3
  - **`mf_example_3`**
- What is new in version 3
  - Implicit rules
    - Verify that the dependency tree is still working
  - Macros
    - **`CFLAGS`** is used by the implicit rules
    - Notice the `-g` option is used on every compile command

makefile Example 4

- makefile example: version 4
  - **`mf_example_4`**
- What is new in version 4
  - More macros
    - `$@`: the name of the current target
    - `$<`: the name of a dependency file, derived as if selected for use with an implicit rule
  - More compact (and cryptic) style
  - Don't forget the makefile itself is also a dependency

Invoke make (I)

- Useful make Options
  - `-d` : debug mode (display the reasons why make chooses to rebuild a target)
  - `-n` : no execution mode (print out the commands)
  - `-p` : print out the complete set of macro definitions and target descriptions
  - `-f` : use non-default makefile

Invoke make (II)

- Default makefile
    - make looks for files **`makefile`** and **`Makefile`** in the current directory. If make finds it, it will use it as the default makefile
    - Save typing, no need for **`-f`**
    - Example default makefile: **`makefile`**