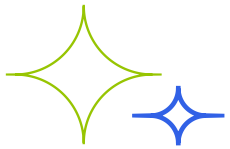


MONTANA
STATE UNIVERSITY

A Short Introduction to Makefile

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.03.27

Make Utility and Makefile

- The make utility is a software tool for managing and maintaining computer programs consisting many component files. The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.
- ***Make*** reads its instruction from Makefile (called the descriptor file) by default.
- Makefile sets a set of rules to determine which parts of a program need to be recompile, and issues command to recompile them.
- Makefile is a way of automating software building procedure and other complex tasks with dependencies.
- Makefile contains: **dependency rules**, **macros and suffix rules**

```c
/* main.c */
#include <stdio.h>
#include "functions.h"

int main() {
    print_hello();
    printf("\n");
    printf("The factorial of 5 is %d\n",
factorial(5));
    return 0;
}
```

```c
/* factorial.c */
#include "functions.h"

int factorial(int n)
{
    int i, fac = 1;
    if(n!=1){
        for(i=1; i<= n; i++)
            fac *= i;
        return fac;
 }
    else return 1;
}
```

```c
/* hello.c */
#include <stdio.h>
#include "functions.h"

void print_hello() {
    printf("Hello World!");
}
```

```c
/* functions.h */
void print_hello();
int factorial(int n);
```

Command Line Approach to Compile

- gcc -c hello.c main.c factorial.c
- ls *.o

  factorial.o hello.o main.o
- gcc -o prog factorial.o hello.o main.o
- ./prog

  Hello World!

  The factorial of 5 is 120

Suppose we later modified hello.c, we need to:

- gcc -c hello.cpp
- gcc -o prog factorial.o hello.o main.o

Example Makefile

```
# This is a comment line
CC=gcc
# CFLAGS will be the options passed to the compiler.
CFLAGS= -c -Wall

all: prog

prog: main.o factorial.o hello.o
    $(CC) main.o factorial.o hello.o -o prog

main.o: main.c
    $(CC) $(CFLAGS) main.c

factorial.o: factorial.c
    $(CC) $(CFLAGS) factorial.c

hello.o: hello.c
    $(CC) $(CFLAGS) hello.c

clean:
    rm -rf *.o prog
```

Basic Makefile Structure

Dependency rules

- A rule consists of three parts, one or more targets, zero or more dependencies, and zero or more commands in the form:

  target: dependencies

  <tab> commands to make **target**

  - <tab> character MUST NOT be replaced be spaces.
  - A "**target**" is usually the name of a file(e.g. executable or object files). It can also be the name of an action (e.g. clean)
  - "dependencies" are files that are used as input to create the **target**.
  - Each "command" in a rule is interpreted by a shell to be executed.
  - By default, *make* uses /bin/sh shell.
  - Typing "make" will:
    - Make sure all the dependencies are up to date
    - If target is older than any dependency, recreate it using the specified commands.

Basic Makefile Structure

Dependency rules

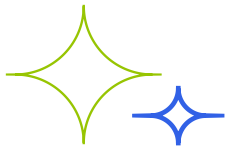
- By default, typing “make” creates first target in Makefile.
- Since prog depends on main.o factorial.o hello.o, all of object files must exist and be up-to-date. *make* will check for them and recreating them if necessary.

Phony targets

- A phony target is one that isn't really the name of a file. It will only have a list of commands and no dependencies.

E.g. clean:

```
rm -rf *.o
```



MONTANA
STATE UNIVERSITY

THE END

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.03.27