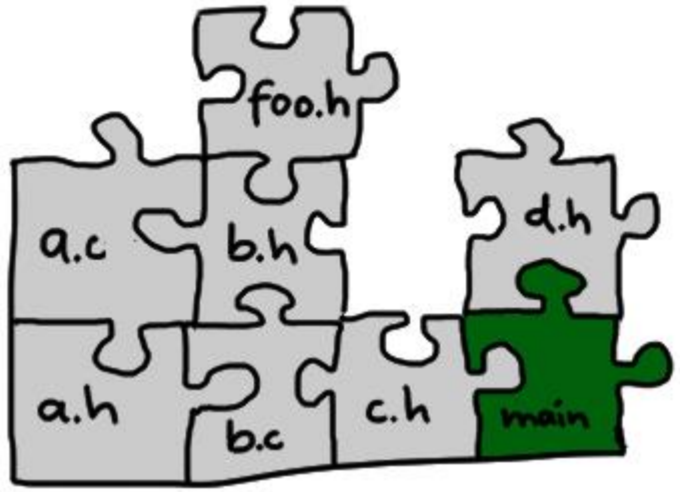
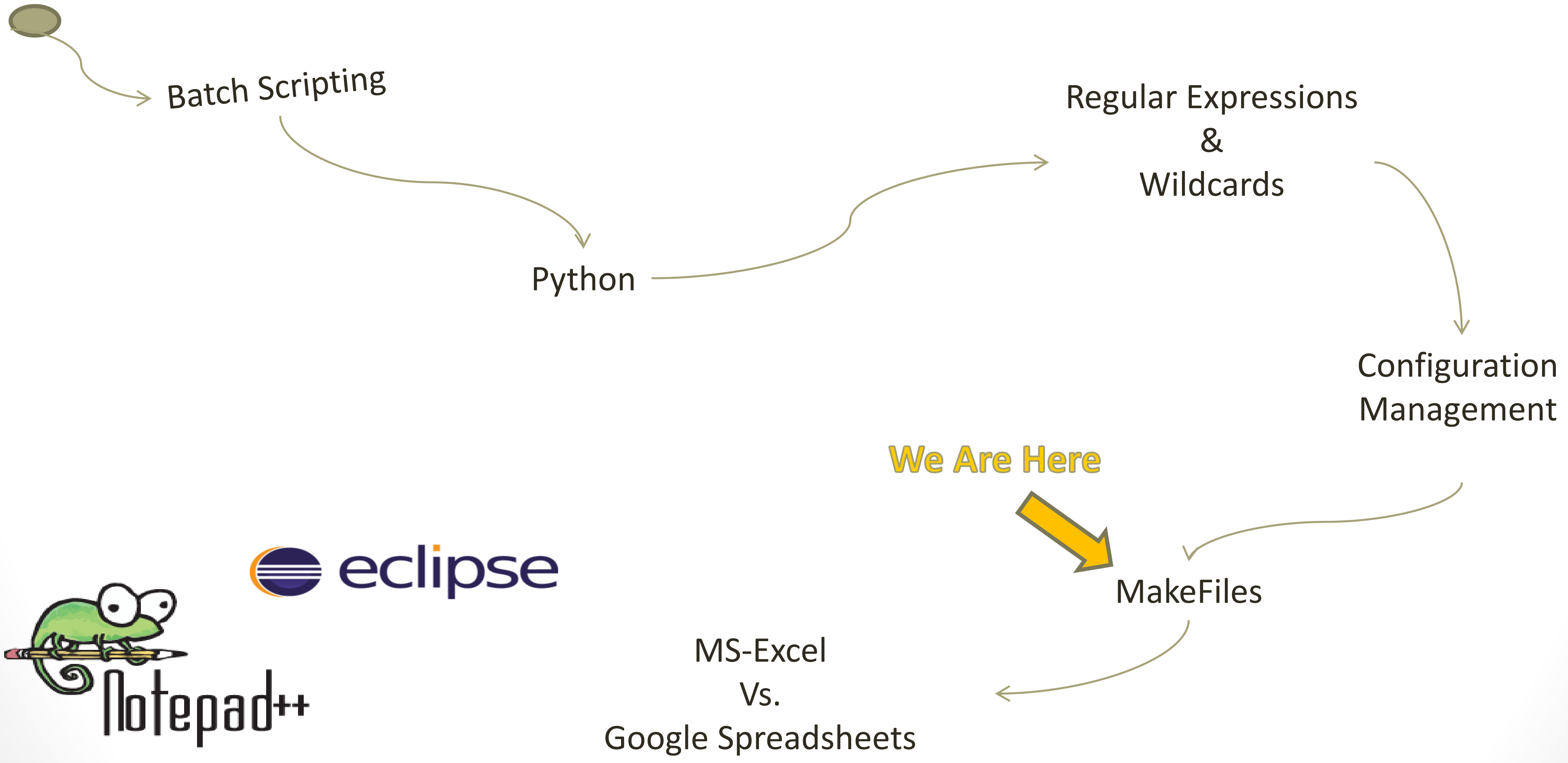
Make Tool

Starter's Guide



Course Map


Batch Scripting
Python
Regular Expressions & Wildcards
Configuration Management
We Are Here
MakeFiles
MS-Excel Vs. Google Spreadsheets

Agenda

Introduction

- MinGw will by used in this course as following:
  - mingw32-gcc.exe will be used as the C compiler in this course (please rename it gcc to call it by typing gcc instead of mingw32-gcc).
  - msys-make will be used as the make tool in this course.
  - MSYS base installation is required to use Linux shells and commands.
  - Please don’t forget to place MinGW bin folders in your PATH env variable.

- Download the demo project that will be used through this course from the link below:
  - https://drive.google.com/file/d/0B6Hf8UvSSqTXci1rcFpvcG9fd0E/view?usp=sharing

Introduction
What is make?

- Make is a tool that is used for generation of files based on other files.
  - In a program, typically, the executable file is updated from object files, which are in turn made by compiling source files. Once a suitable makefile exists, each time you change some source files, a simple shell command (make) performs all necessary recompilations

- Make tool parses an input file “makefile” to define the rules by which files are going to be generated
  - makefile describes the relationships among files in your program and provides commands for updating each file.

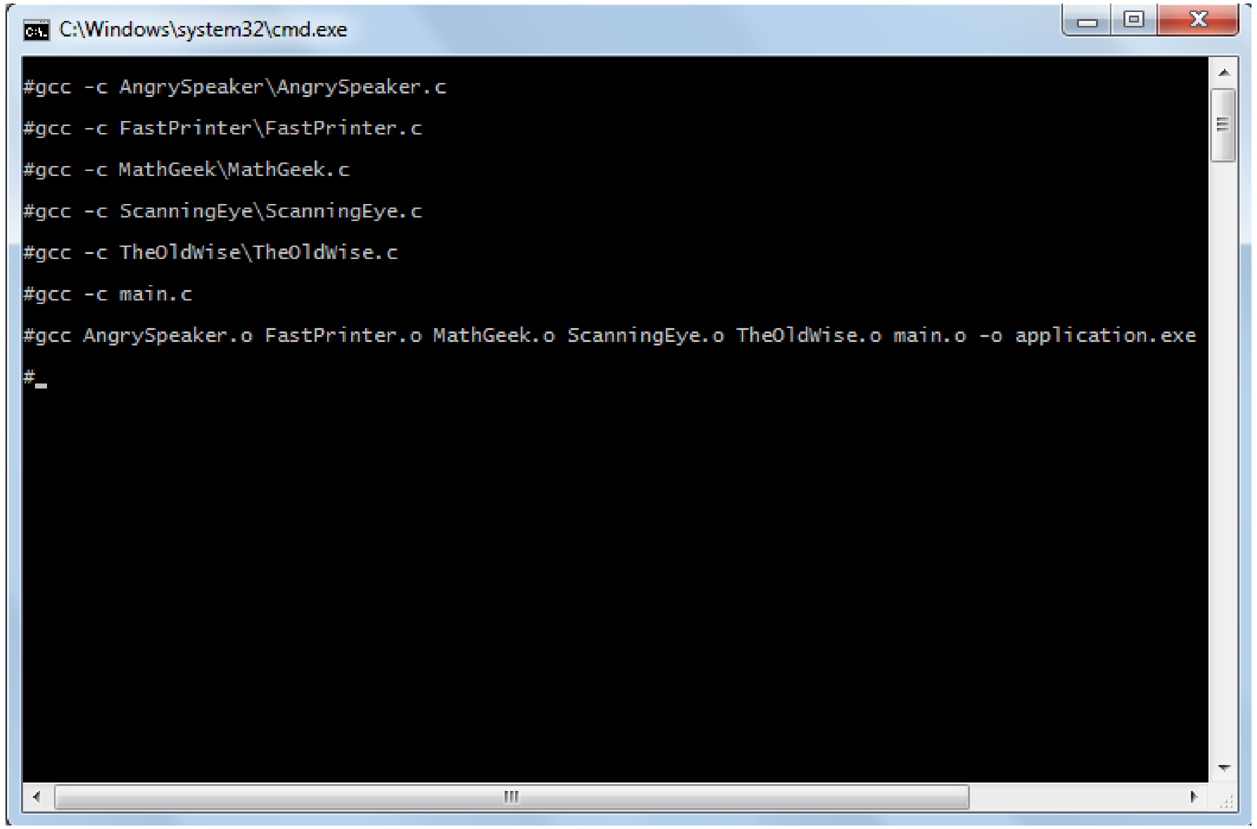
- There are many derivatives for the Make tool. (Two famous examples would be gnu make and Borland make).

Introduction
Why do we need make ?

- There are different ways to organize the compilation process of a project.
- These ways may or may not satisfy your needs based on the project size and other factors.
- In the following slides we will demonstrate different ways of organizing the compilation process:
  - Compile file by file in a command window (e.x windows cmd)
  - Write a batch script to organize the compilation process of a project.
  - Use make tool to organize the compilation process of a project.

Introduction
Why do we need make ?

- Different ways to compile
  - Compile file by file in the command window



```
#gcc -c AngrySpeaker\AngrySpeaker.c
#gcc -c FastPrinter\FastPrinter.c
#gcc -c MathGeek\MathGeek.c
#gcc -c ScanningEye\ScanningEye.c
#gcc -c TheOldWise\TheOldWise.c
#gcc -c main.c
#gcc AngrySpeaker.o FastPrinter.o MathGeek.o ScanningEye.o TheOldWise.o main.o -o application.exe
#_
```

Introduction
Why do we need make ?

- Different ways to compile
  - Write a Batch Script to compile

```
@echo off
Set ProgramName=MasterApplication
Set File1=main
Set File2=MathGeek/MathGeek
Set File3=FastPrinter/FastPrinter
Set File4=ScanningEye/ScanningEye
Set File5=AngrySpeaker/AngrySpeaker
Set File6=TheOldWise/TheOldWise
Set CC=gcc
if [%1]=[clean] goto Clean
if [%1]=[compile] goto Compile
if [%1]=[link] goto Link
if [%1]=[execute] goto Execute

:Clean
IF EXIST %File1%.o del %File1%.o
IF EXIST %File2%.o del %File2%.o
IF EXIST %File3%.o del %File3%.o
IF EXIST %File4%.o del %File4%.o
IF EXIST %File5%.o del %File5%.o
IF EXIST %File6%.o del %File6%.o
GOTO END

:Compile
%CC% -c %File1%.c -o %File1%.o
%CC% -c %File2%.c -o %File2%.o
%CC% -c %File3%.c -o %File3%.o
%CC% -c %File4%.c -o %File4%.o
%CC% -c %File5%.c -o %File5%.o
%CC% -c %File6%.c -o %File6%.o
GOTO END

:Link
%CC% %File1%.o %File2%.o %File3%.o %File4%.o %File5%.o %File6%.o -o %programName%
GOTO END

:Execute
CALL %programName%.exe

:END
exit
```

Introduction
Why do we need make ?

- Different ways to compile
  - Use make tool to compile

```
MasterApplication.exe : main.o MathGeek/MathGeek.o \
                        FastPrinter/FastPrinter.o ScanningEye/ScanningEye.o \
                        AngrySpeaker/AngrySpeaker.o TheOldWise/TheOldWise.o
        gcc main.o MathGeek/MathGeek.o FastPrinter/FastPrinter.o \
                ScanningEye/ScanningEye.o AngrySpeaker/AngrySpeaker.o \
                TheOldWise/TheOldWise.o -o MasterApplication

main.o : main.c
        gcc -c main.c -o main.o

MathGeek/MathGeek.o : MathGeek/MathGeek.c
        gcc -c MathGeek/MathGeek.c -o MathGeek/MathGeek.o

FastPrinter/FastPrinter.o : FastPrinter/FastPrinter.c
        gcc -c FastPrinter/FastPrinter.c -o FastPrinter/FastPrinter.o

ScanningEye/ScanningEye.o : ScanningEye/ScanningEye.c
        gcc -c ScanningEye/ScanningEye.c -o ScanningEye/ScanningEye.o

AngrySpeaker/AngrySpeaker.o : AngrySpeaker/AngrySpeaker.c
        gcc -c AngrySpeaker/AngrySpeaker.c -o AngrySpeaker/AngrySpeaker.o

TheOldWise/TheOldWise.o : TheOldWise/TheOldWise.c
        gcc -c TheOldWise/TheOldWise.c -o TheOldWise/TheOldWise.o
```

Introduction
Why do we need make ?

- Here is a comparison between the different method we tried in the previous slides:

Method	Effort saving	Re-compile according to changes only	Scalability (can be used with larger projects)
Using cmd	No	No	No
Using batch script	Yes	No	No
Using make tool	Yes	Yes	Yes

Introduction
Why the make method is better

- Make does not recompile already compiled files
  - The make program uses the makefile rules and the last-modification times of the files to decide which of the files need to be updated.
  - This makes it Time saving

- Make is scalable
  - The make file can be edited efficiently to adapt with your project increasing size

- Make offers a lot of features that can help you without adding too much complexity
  - Parallel execution, Generating Dependencies automatically and etc.

Introduction
How to use make tool

- Step 1: write the make file
- Step 2: save the make file with any name in the required directory (for example myFirstMake.mak)
- Step 3: open CMD in the directory containing the make file
- Step 4: write the following command in the command window
  - make -f myFirstMake.mak

If you name your make file "makefile" you can simply call "**make**" without any arguments to call your make file

Introduction
Help

- To get help for the make command use

```
make --help
```

Introduction
Make User Manual

- The Make tool course is based on gnu make user manual
  - http://www.gnu.org/software/make/manual/make.html

GNU Make Syntax and Features

- Make file consists of a set of rules that tells the make tool how to generate the needed targets

```
Target : prerequisite
        recipe
```

$\underbrace{\qquad\qquad}_{\boxed{\text{Tab}}}$

GNU Make Syntax and Features

- A *target* is usually the name of a file that is generated by a program; examples of targets are executable or object files. A target can also be the name of an action to carry out, such as 'clean'.
    - By default if no targets are passed to the make file when calling it, make starts with the first target written in the make file. This is called the *default goal*. Usually the first target in the make file is the final output of your program (i.e the final executable file).
    - Note that you can pass a specific target to the make file when calling it, this will be the target that make starts with :
        - E.x make clean
- A *prerequisite* is a file that is used as input to create the target. A target often depends on several files.
- A *recipe* is an action that make carries out. A recipe may have more than one command, either on the same line or each on its own line. **Please note:** you need to put a tab character at the beginning of every recipe line!

```
MasterApplication.exe : main.o MathGeek/MathGeek.o \
                        FastPrinter/FastPrinter.o ScanningEye/ScanningEye.o \
                        AngrySpeaker/AngrySpeaker.o TheOldWise/TheOldWise.o
        gcc main.o MathGeek/MathGeek.o FastPrinter/FastPrinter.o \
                ScanningEye/ScanningEye.o AngrySpeaker/AngrySpeaker.o \
                TheOldWise/TheOldWise.o -o MasterApplication
```

Target

Prerequisite

Recipe

```
main.o : main.c
        gcc -c main.c -o main.o
```

Rule

```
MathGeek/MathGeek.o : MathGeek/MathGeek.c
        gcc -c MathGeek/MathGeek.c -o MathGeek/MathGeek.o

FastPrinter/FastPrinter.o : FastPrinter/FastPrinter.c
        gcc -c FastPrinter/FastPrinter.c -o FastPrinter/FastPrinter.o

ScanningEye/ScanningEye.o : ScanningEye/ScanningEye.c
        gcc -c ScanningEye/ScanningEye.c -o ScanningEye/ScanningEye.o

AngrySpeaker/AngrySpeaker.o : AngrySpeaker/AngrySpeaker.c
        gcc -c AngrySpeaker/AngrySpeaker.c -o AngrySpeaker/AngrySpeaker.o

TheOldWise/TheOldWise.o : TheOldWise/TheOldWise.c
        gcc -c TheOldWise/TheOldWise.c -o TheOldWise/TheOldWise.o
```

GNU Make Syntax and Features Comments and echoing

- Comment Lines are done using "#"
  - Example
    - `# This is a comment inside the make file`

- When a line starts with '@', the echoing of that line is suppressed.
  - Example
    -
      ```
      main.o : main.c
              @gcc -c main.c -o main.o
      ```

GNU Make Syntax and Features Variables

- Variables makes the make file simpler and configurable

- Example 1:

```
fileSrc1 = main.c
Main.o : $(fileSrc1)
```

- Example2:

```
CC = gcc
MasterApplication.exe : $(allObjs)
$(CC) $(allObjs) -o MasterApplication
```

makefile

```
fileSrc1 = main.c
fileObj1 = main.o
fileHeader1 =

fileSrc2 = MathGeek/MathGeek.c
fileObj2 = MathGeek/MathGeek.o
fileHeader2 = MathGeek/MathGeek.h

fileSrc3 = FastPrinter/FastPrinter.c
fileObj3 = FastPrinter/FastPrinter.o
fileHeader3 = FastPrinter/FastPrinter.h

fileSrc4 = ScanningEye/ScanningEye.c
fileObj4 = ScanningEye/ScanningEye.o
fileHeader4 = ScanningEye/ScanningEye.h

fileSrc5 = AngrySpeaker/AngrySpeaker.c
fileObj5 = AngrySpeaker/AngrySpeaker.o
fileHeader5 = AngrySpeaker/AngrySpeaker.h

fileSrc6 = TheOldWise/TheOldWise.c
fileObj6 = TheOldWise/TheOldWise.o
fileHeader6 = TheOldWise/TheOldWise.h


FinalTargetName=MasterApplication.exe


allObjs = $(fileObj1) $(fileObj2) $(fileObj3) $(fileObj4) $(fileObj5) $(fileObj6)
CC = gcc
```

makefile

```
$(FinalTargetName) : $(allObjs)
	$(CC) $(allObjs) -o $(FinalTargetName)

$(fileObj1) : $(fileSrc1)
	$(CC) -c $(fileSrc1) -o $(fileObj1)
$(fileObj2) : $(fileSrc2)
	$(CC) -c $(fileSrc2) -o $(fileObj2)
$(fileObj3) : $(fileSrc3)
	$(CC) -c $(fileSrc3) -o $(fileObj3)
$(fileObj4) : $(fileSrc4)
	$(CC) -c $(fileSrc4) -o $(fileObj4)
$(fileObj5) : $(fileSrc5)
	$(CC) -c $(fileSrc5) -o $(fileObj5)
$(fileObj6) : $(fileSrc6)
	$(CC) -c $(fileSrc6) -o $(fileObj6)
```

GNU Make Syntax and Features Variables

Environment Variables:

- Environment variables can be accessed from within make
- This means that the used shell can set variables to be used from within the make file
- Every environment variable that make sees when it starts up is transformed into a make variable with the same name and value.

GNU Make Syntax and Features Variables

Targeted variables:

- Variables that have different value per rule

- Syntax
  - `Target : variable=value`

- Example
  - `$(FinalTargetName) : CFLAG =`
  - `$(allObjs): CFLAG = -c`

makefile

```
$(FinalTargetName) : CFLAG =
$(allObjs): CFLAG = -c

$(FinalTargetName) : $(allObjs)
	$(CC) $(CFLAG) $(allObjs) -o $(FinalTargetName)
$(fileObj1) : $(fileSrc1)
	$(CC) $(CFLAG) $(fileSrc1) -o $(fileObj1)
$(fileObj2) : $(fileSrc2)
	$(CC) $(CFLAG) $(fileSrc2) -o $(fileObj2)
$(fileObj3) : $(fileSrc3)
	$(CC) $(CFLAG) $(fileSrc3) -o $(fileObj3)
$(fileObj4) : $(fileSrc4)
	$(CC) $(CFLAG) $(fileSrc4) -o $(fileObj4)
$(fileObj5) : $(fileSrc5)
	$(CC) $(CFLAG) $(fileSrc5) -o $(fileObj5)
$(fileObj6) : $(fileSrc6)
	$(CC) $(CFLAG) $(fileSrc6) -o $(fileObj6)
```

GNU Make Syntax and Features
Shell and CMD Commands

- It is possible to call shell commands directly from within a recipe (this is dependent on the default shell used i.e. if the defined shell using MAKESHELL or SHELL variable is cmd.exe then cmd built in commands will work, in our case the msys make can understand cmd and sh and all installed msys linux commands) such as:
  - rm OR del
    - Used to delete files or folders (-R is used to delete folders)
  - mv OR move
    - Used to move or rename files or folders
  - cp OR copy
    - Used to copy files or folders
  - mkdir
    - Create a directory

- Example:

```
$(FinalTargetName) : $(allObjs)
        $(CC) $(CFLAG) $(allObjs) -o $(FinalTargetName)
        pwd OR echo %CD% #command to print current working directory
```

GNU Make Syntax and Features External commands

- It is possible to call external commands from the makefile directly from within the recipe
- External commands can be executable files or batch scripts or python scripts or etc.
  - Example
    -
      ```
      $(FinalTargetName) : $(allObjs)
              $(CC) $(CFLAG) $(allObjs) -o $(FinalTargetName)
              create_log_file.bat
              python parse_log_files.py
      ```

Lab1

- Download **make_lab1** from the following link. And fill in the **makefile** empty recipes in **20 mins** to do the required functionality.
  - https://drive.google.com/open?id=0B6Hf8UvSSqTXeGNWRDUxc1NUbGc

GNU Make Syntax and Features Phony Targets

- What is wrong when we implement the clean recipe as shown below??

```
clean:
	DEL $(FinalTargetName) $(allObjs)
```

- What happens if a file named clean exists in the same directory of make?
  - This command will never run because there are no prerequisites to this target and make will always consider “clean” to be up to date

GNU Make Syntax and Features
Phony Targets

- Using a special built in target named **PHONY**, it is possible to define for the make tool that these targets are not real files to solve any conflicts .

- Example
  - `.PHONY : clean`
  - `clean:`

```
    DEL $(FinalTargetName) $(allObjs)
```

GNU Make Syntax and Features Include

```
fileSrc1 = main.c
fileObj1 = main.o
fileHeader1 =

fileSrc2 = MathGeek/MathGeek.c
fileObj2 = MathGeek/MathGeek.o
fileHeader2 = MathGeek/MathGeek.h

fileSrc3 = FastPrinter/FastPrinter.c
fileObj3 = FastPrinter/FastPrinter.o
fileHeader3 = FastPrinter/FastPrinter.h

fileSrc4 = ScanningEye/ScanningEye.c
fileObj4 = ScanningEye/ScanningEye.o
fileHeader4 = ScanningEye/ScanningEye.h

fileSrc5 = AngrySpeaker/AngrySpeaker.c
fileObj5 = AngrySpeaker/AngrySpeaker.o
fileHeader5 = AngrySpeaker/AngrySpeaker.h

fileSrc6 = TheOldWise/TheOldWise.c
fileObj6 = TheOldWise/TheOldWise.o
fileHeader6 = TheOldWise/TheOldWise.h


FinalTargetName=MasterApplication.exe

CC = gcc
```

makeConfiguration.mk

```
include makeConfiguration.mk


allObjs = $(fileObj1) $(fileObj2) $(fileObj3) $(fileObj4) \
$(fileObj5) $(fileObj6)
$(FinalTargetName) : CFLAG =
$(allObjs): CFLAG = -c

$(FinalTargetName) : $(allObjs)
		$(CC) $(CFLAG) $(allObjs) -o $(FinalTargetName)

$(fileObj1) : $(fileSrc1)
		$(CC) $(CFLAG) $(fileSrc1) -o $(fileObj1)
$(fileObj2) : $(fileSrc2)
		$(CC) $(CFLAG) $(fileSrc2) -o $(fileObj2)
$(fileObj3) : $(fileSrc3)
		$(CC) $(CFLAG) $(fileSrc3) -o $(fileObj3)
$(fileObj4) : $(fileSrc4)
		$(CC) $(CFLAG) $(fileSrc4) -o $(fileObj4)
$(fileObj5) : $(fileSrc5)
		$(CC) $(CFLAG) $(fileSrc5) -o $(fileObj5)
$(fileObj6) : $(fileSrc6)
		$(CC) $(CFLAG) $(fileSrc6) -o $(fileObj6)

.PHONY : clean link onlyCompile

clean:
		DEL $(FinalTargetName) $(allObjs)
link:
		$(CC) $(allObjs) -o $(FinalTargetName)
onlyCompile: $(allObjs)
```

makefile

GNU Make Syntax and Features
Include

- If the included file does not exist. Make will throw a warning and an error may occur.
- To force make to ignore errors when the included file is not found, “-include” is used.

GNU Make Syntax and Features Automatic Variables

- Automatic variables are special variables that have special meanings inside the recipe:
  - $@
    - Target File Name
  - $<
    - The name of the first prerequisite
  - $?
    - The names of all the prerequisites that are newer than the target
  - $^
    - The names of all the prerequisites.
  - $(@D)
    - The directory part in the target file name
  - $(@F)
    - The file name part in the target file name

Lab2

- Download the **makefile** that we developed up to this point and replace variables used in recipes with **automatic variables**. Complete this lab in **15** mins.
  - https://drive.google.com/open?id=0B6Hf8UvSSqTXOXFTWm40ODFQTDA

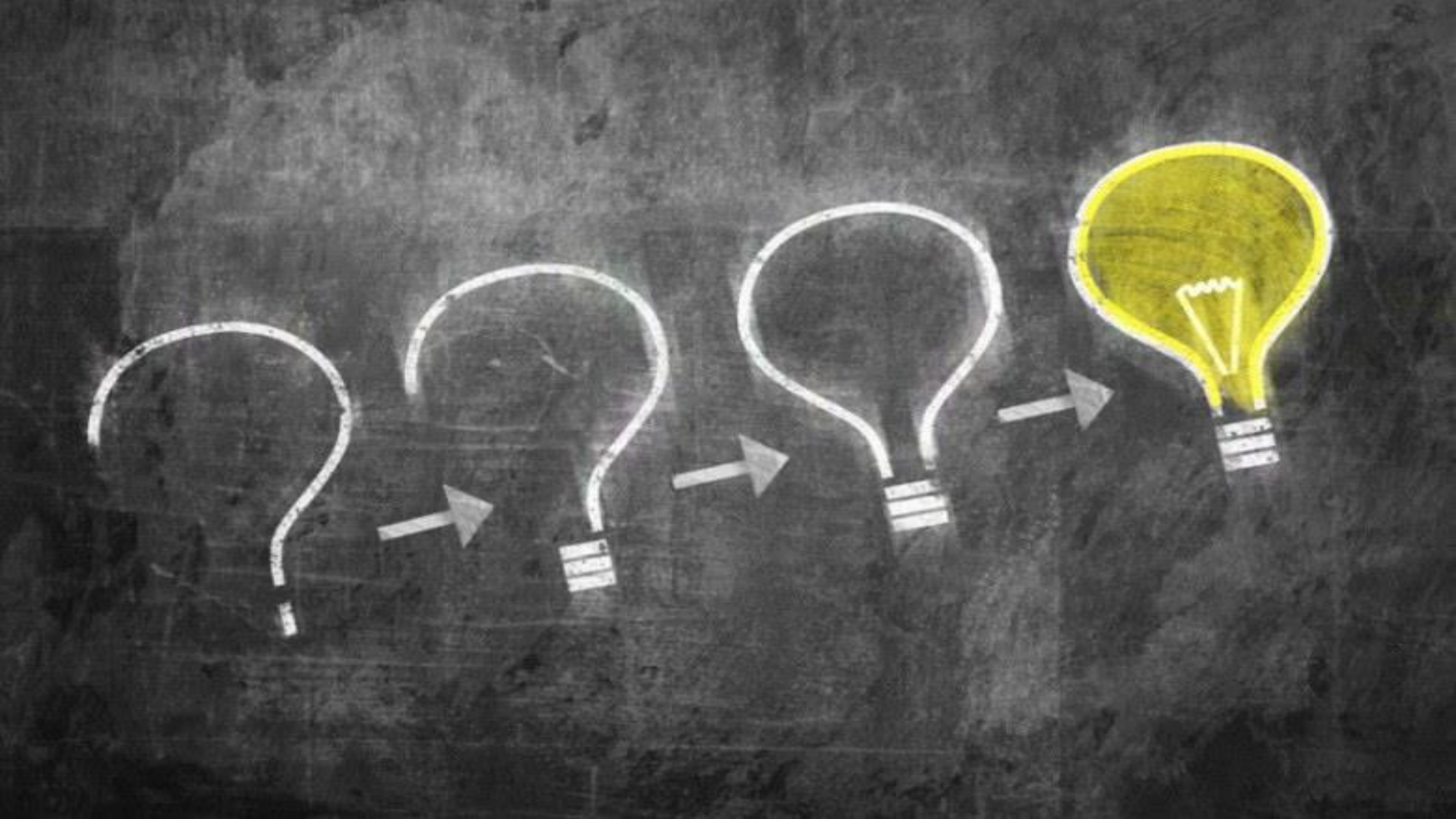
GNU Make Books

- For more on GNU make :
  - Check Managing Projects with GNU Make, Third Edition
    - http://www.wanderinghorse.net/computing/make/

Credits

- The original version of this material was created by:
  - Eng. Ahmed ElAbyad: ahmd.ghassan@gmail.com
  - Eng. Mohammad Alaa Hekal: embeddedgeek.34@gmail.com

Copy Rights



Eng. Mohammad Alaa Hekal: embeddedgeek.34@gmail.com
Omar Soliman: Omar.Soliman@imtSchool.com