Introduction to Makefile

Ray Song

February 25, 2012

Motivation

- Small programs -> single file -> manual compilation
- "not so small" programs
  - many files
  - multiple components
  - more than one programmers

Motivation

- Small programs -> single file -> manual compilation
- “not so small” programs
  - many files
  - multiple components
  - more than one programmers

Motivation

- Small programs -> single file -> manual compilation
- “not so small” programs
  - many files
  - multiple components
  - more than one programmers

Motivation

- Small programs -> single file -> manual compilation
- “not so small” programs
  - many files
  - multiple components
  - more than one programmers

Motivation

- Small programs -> single file -> manual compilation
- “not so small” programs
  - many files
  - multiple components
  - more than one programmers

Motivation - cont.

- Problems
  - harder to manage
  - every change require long compilation
  - division to components is desired

Motivation - cont.

- Problems
  - harder to manage
  - every change require long compilation
  - division to components is desired

Motivation - cont.

- Problems
  - harder to manage
  - every change require long compilation
  - division to components is desired

Motivation - cont.

- Problems
  - harder to manage
  - every change require long compilation
  - division to components is desired

Motivation - cont.

- Solution - Makefile

Makefile

- *Makefile* describes
  - project structure
  - instructions for files creation
- A **makefile** consists of many **rules**.

Makefile

- *Makefile* describes
  - project structure
  - instructions for files creation
- A **makefile** consists of many **rules**.

Makefile

- *Makefile* describes
  - project structure
  - instructions for files creation
- A **makefile** consists of many **rules**.

Makefile

- *Makefile* describes
  - project structure
  - instructions for files creation
- A **makefile** consists of many **rules**.

Rule syntax

TARGETS: PREREQUISITES

RECIPE

- In short, each rule describe instructions (**RECIPE**) to create files (**TARGETS**) with **PREREQUISITES**.
- **PREREQUISITES** are targets must be created prior to **TARGETS**.
- A **target** is considered **old** if its modification timestamp is smaller than one of its dependencies's.

Rule syntax

TARGETS: PREREQUISITES

RECIPE

- In short, each rule describe instructions (**RECIPE**) to create files (**TARGETS**) with **PREREQUISITES**.
- **PREREQUISITES** are targets must be created prior to **TARGETS**.
- A **target** is considered **old** if its modification timestamp is smaller than one of its dependencies's.

Rule syntax

TARGETS: PREREQUISITES

RECIPE

- In short, each rule describe instructions (**RECIPE**) to create files (**TARGETS**) with **PREREQUISITES**.
- **PREREQUISITES** are targets must be created prior to **TARGETS**.
- A **target** is considered **old** if its modification timestamp is smaller than one of its dependencies's.

Makefile – cont.

- **TARGETS** and **PREREQUISITES** are file names separated by spaces.
- Usually there is **only one** target per rule.
- **TARGETS** and **PREREQUISITES** may contain **wildcards**, e.g. %.c.
- Each line of **RECIPE** starts with a **TAB**.
- The first **rule** indicates the default target (not counting targets that contain wildcards).

Makefile – cont.

- **TARGETS** and **PREREQUISITES** are file names separated by spaces.
- Usually there is **only one** target per rule.
- **TARGETS** and **PREREQUISITES** may contain **wildcards**, e.g. %.c.
- Each line of **RECIPE** starts with a **TAB**.
- The first **rule** indicates the default target (not counting targets that contain wildcards).

Makefile – cont.

- **TARGETS** and **PREREQUISITES** are file names separated by spaces.
- Usually there is **only one** target per rule.
- **TARGETS** and **PREREQUISITES** may contain **wildcards**, e.g. %.c.
- Each line of **RECIPE** starts with a **TAB**.
- The first **rule** indicates the default target (not counting targets that contain wildcards).

Makefile – cont.

- **TARGETS** and **PREREQUISITES** are file names separated by spaces.
- Usually there is **only one** target per rule.
- **TARGETS** and **PREREQUISITES** may contain **wildcards**, e.g. %.c.
- Each line of **RECIPE** starts with a **TAB**.
- The first **rule** indicates the default target (not counting targets that contain wildcards).

Makefile – cont.

- **TARGETS** and **PREREQUISITES** are file names separated by spaces.
- Usually there is **only one** target per rule.
- **TARGETS** and **PREREQUISITES** may contain **wildcards**, e.g. %.c.
- Each line of **RECIPE** starts with a **TAB**.
- The first **rule** indicates the default target (not counting targets that contain wildcards).

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism

- **make** command reads a **makefile** and records these rules into its data base.
- **GNU Make** defaults to search **GNUmakefile**, **makefile**, **Makefile** in order, use the first of these which exists.
- The first goal (terminology used to refer to the list of **targets** you specified on the command line) should be created.
- Prerequisites which appeared in the target must be processed first.
- This is a recursive process (depth first search).
- After updating the dependencies , **make** decides whether it is necessary to **recreated** the target. This is the case when it is older than one of its dependencies. In the case we **recreate** the target, execute the associated recipe.

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

```
prog: main.c sum1.c sum2.c

gcc -o prog main.c sum1.c sum2.c
```

which requires compilation of all project when something is changed

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

```
prog: main.c sum1.c sum2.c

gcc -o prog main.c sum1.c sum2.c
```

which requires compilation of all project when something is changed

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

```
prog: main.c sum1.c sum2.c

	gcc -o prog main.c sum1.c sum2.c
```

  which requires compilation of all project when something is changed

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

  prog: main.c sum1.c sum2.c

  gcc -o prog main.c sum1.c sum2.c

  which requires compilation of all project when something is changed

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

  prog: main.c sum1.c sum2.c

  gcc –o prog main.c sum1.c sum2.c

  which requires compilation of all project when something is changed

make's mechanism - cont.

- **make** virtually construct a dependency DAG (directed acyclic graph).
- **make** ensures **minimum** compilation as long as the project structure is written properly.
- **Do not write** something like:

  prog: main.c sum1.c sum2.c

  gcc –o prog main.c sum1.c sum2.c

  which requires compilation of all project when something is changed

Automatic variables

- $@
- $^
- $<
- others including $, *$?*, *$+*, *$|*, *$%*, *$(%D)*, *%(*F), ...

Automatic variables

- $@
- $^
- $<
- others including $, *$?*, *$+*, *$|*, *$%*, *$(%D)*, *%(*F), ...

Automatic variables

- $@
- $^
- $<
- others including $, *$?*, *$+*, *$|*, *$%*, *$(%D)*, *%(*F), ...

Automatic variables

- $@
- $^
- $<
- others including $, *$?*, *$+*, *$|*, *$%*, *$(%D)*, *%(*F), ...

Example

```
convert: cmdline.o convert.o
	g++ $^ -o $@

convert.o: convert.cpp convert.h
	g++ -c $<

cmdline.o: cmdline.cpp cmdline.h convert.h
	g++ -c $<
```

Equivalent (implicit rules)

- **make** can dedude appropriate recipes according to suffixes

  convert: cmdline.o convert.o

  convert.o: convert.cpp convert.h

  cmdline.o: cmdline.cpp cmdline.h convert.h

Equivalent (implicit rules) - cont.

convert: cmdline.o

convert.o: convert.h

cmdline.o: cmdline.h convert.h

Another example

```
%.o: %.c
	gcc -c $<
```

Passing parameters

```
test: FORCE
	echo $(VAR)

FORCE:
```

- make VAR=hello
- make VAR=world

Passing parameters

test: FORCE

echo $(VAR)

FORCE:

- make VAR=hello
- make VAR=world

Force targets

- Targets without recipes or prerequisites

Phony targets

- They do not correspond to **real** file.
- Provide some utility, e.g. cleaning intermediate files, archiving the whole project, creating TAGS file for some editors
- Forced to run its recipe upon executing.

Phony targets

- They do not correspond to **real** file.
- Provide some utility, e.g. cleaning intermediate files, archiving the whole project, creating TAGS file for some editors
- Forced to run its recipe upon executing.

Phony targets

- They do not correspond to **real** file.
- Provide some utility, e.g. cleaning intermediate files, archiving the whole project, creating TAGS file for some editors
- Forced to run its recipe upon executing.

Example

```
all : prog1 prog2 prog3

.PHONY : all

prog1 : prog1.o utils.o
	cc -o prog1 prog1.o utils.o

prog2 : prog2.o
	cc -o prog2 prog2.o

prog3 : prog3.o sort.o utils.o
	cc -o prog3 prog3.o sort.o utils.o
```

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Practical options of **make**

- -n, print the commands to be run but do not execute them
- -t, touch files instead of running the recipes
- -B, unconditionally make all targets (overide timestamps)
- -W **file**, pretend **file** has been just modified
- -p, print the data base that results from reading the makefiles
- -d, debug mode
- others, RTFM

Other usage

- **Makefile**'s mechanism is not limited to programs
  - LaTeX sources
  - website deployment
  - describe any task where some files need updating as a result of changes of other files

Other usage

- **Makefile**'s mechanism is not limited to programs
  - LaTeX sources
  - website deployment
  - describe any task where some files need updating as a result of changes of other files

Other usage

- **Makefile**'s mechanism is not limited to programs
  - LaTeX sources
  - website deployment
  - describe any task where some files need updating as a result of changes of other files

Other usage

- **Makefile**'s mechanism is not limited to programs
  - LaTeX sources
  - website deployment
  - describe any task where some files need updating as a result of changes of other files

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Additional features

- Variables (two flavors: simple and recursive, the latter is also called **macros**)
- Functions including string substitution, file name manipulation, **foreach**, even the the root of evil – **eval**
- VPATH
- Ability to manipulate archives(.a)
- Including other makefiles
- Conditionals
- Secondary expansion
- Order-only prerequisites
- Static patterns
- Double-colon rules
- Target/pattern-specific variable values

Acknowledgements

- This slide incorporate some stuff from Roded Sharan's `http://www.cs.tau.ac.il/~roded/courses/softp-b06/Makefile.ppt`
- Markdown `http://daringfireball.net/projects/markdown/` in which this slide source is written.
- Pandoc `http://johnmacfarlane.net/pandoc/` by which this slide is generated.
- Haskell `http://www.haskell.org/` in which Pandoc is implemented.

Acknowledgements

- This slide incorporate some stuff from Roded Sharan's `http://www.cs.tau.ac.il/~roded/courses/softp-b06/Makefile.ppt`
- Markdown `http://daringfireball.net/projects/markdown/` in which this slide source is written.
- Pandoc `http://johnmacfarlane.net/pandoc/` by which this slide is generated.
- Haskell `http://www.haskell.org/` in which Pandoc is implemented.

Acknowledgements

- This slide incorporate some stuff from Roded Sharan's `http://www.cs.tau.ac.il/~roded/courses/softp-b06/Makefile.ppt`
- Markdown `http://daringfireball.net/projects/markdown/` in which this slide source is written.
- Pandoc `http://johnmacfarlane.net/pandoc/` by which this slide is generated.
- Haskell `http://www.haskell.org/` in which Pandoc is implemented.

Acknowledgements

- This slide incorporate some stuff from Roded Sharan's `http://www.cs.tau.ac.il/~roded/courses/softp-b06/Makefile.ppt`
- Markdown `http://daringfireball.net/projects/markdown/` in which this slide source is written.
- Pandoc `http://johnmacfarlane.net/pandoc/` by which this slide is generated.
- Haskell `http://www.haskell.org/` in which Pandoc is implemented.

References

- GNU Make Manual
  `http://www.gnu.org/software/make/manual/`