A VISUAL REFERENCE · GNU Make

GNU Make, one change at a time

From a first rule to reliable builds, LaTeX documents and publishing workflows. Complete source. A small change. Its actual result.

The Org file is the canonical source. Read the HTML guide beside your work, or download its PDF and source for offline reading.

> **Last changes: 2026-10-11.** Unified tour–chapter–example IDs, numbered chapter anchors and updated direct links, downloads and comparisons. Manual references now use descriptive links. Reader pages now omit publishing instructions and highlight PDF downloads.

90
worked examples

14
topic families

1
canonical Org source

READING THE TOUR

Three ways to see a change

Makefile

Read the exact Makefile used by the example, including targets, prerequisites and recipes.

Build result

Compare captured output and exit status with the files and commands declared by the rule.

Complete workflow

Expand the runnable reproducer or download its script to recreate all local inputs.

The primary panel shows the exact Makefile used by the runnable example. Expand Complete runnable example for setup files, invocation and checks, or download its `.mk` and `.sh` files. The PDF keeps the Makefile and output together; its downloads and attached Org source provide the full reproducer. The LaTeX section covers documents, bibliography tools, generated figures, draft/final builds, cleanup and a source package that rebuilds successfully.

Read from the beginning or jump to a stable example ID. Each example includes its complete input and captured result. Copy link preserves the address of the guide you are reading, including its current domain and path.

How to use this tour

Learn GNU Make through 90 complete examples in 14 topics. Read the Makefile, then compare the captured output. Every example has a MAK identifier, a direct URL anchor and a Copy link button in the HTML guide. Tabs introduce ordinary recipes; the visible-prefix example shows an alternative.

Requirements: GNU Make 4.3 or newer, Bash, and common Unix tools. The final three workflow examples also use AWK and gnuplot. The LaTeX section requires latexmk, pdfLaTeX, BibTeX, Biber with biblatex, and pdftotext. On macOS, /usr/bin/make is often GNU Make 3.81; install a current GNU Make with brew install make and use gmake. The runnable examples prefer gmake when it exists and otherwise use make.

Each example writes to its own newly created temporary directory. The directory is removed on exit. To retain its artifacts when running a downloaded script, use KEEP_WORK=1 bash MAK-13-004.sh. These are original examples, not extracts from the GNU manual. Content is CC0; the tour structure is CC BY 4.0.

The primary panel shows the exact Makefile used by the runnable example. Expand Complete runnable example for setup files, invocation and follow-up checks, or download its .sh file. The PDF keeps the Makefile and output together; its download links and attached Org source provide the complete reproducer. Parallel recipe output ordering may vary; MAK-11-003 accepts both complete block orders rather than pretending the scheduler has a fixed order.

Concepts to keep in mind

Make first reads and expands its configuration, then walks the dependency graph. A real file target is stale when missing or older than a normal prerequisite. A phony goal names an action. Order-only prerequisites establish ordering without making timestamp changes propagate. A recipe usually uses one shell per logical line. Make variables and shell variables have different expansion rules. Declare real dependencies before enabling parallel execution.

Read the GNU Make manual for the complete reference.

Selected manual sections:

- Rule syntax
- Normal and order-only prerequisites
- Variable flavors
- Automatic variables
- Parallel execution
- Rules with multiple targets

01 - Read and run a rule

MAK-01-001

A target and its recipe

A rule names a target before the colon. The following tab-indented recipe runs when that target is requested. Run: make hello.

Makefile

```
hello:
	printf "%s\n" "Hello from Make"
```

Actual output

Captured run · exit 0

```
printf "%s\n" "Hello from Make"
Hello from Make
```

Download: MAK-01-001.mk · MAK-01-001.sh

MAK-01-002

Silence recipe echo with @

Add @ to hide the recipe command while retaining its output. Compare with MAK-01-001. Run: make hello.

Makefile

```
hello:
	@printf "%s\n" "Hello from Make"
```

Download: MAK-01-002.mk · MAK-01-002.sh

Actual output

Captured run · exit 0

```
Hello from Make
```

MAK-01-003

Choose the default goal

.DEFAULT_GOAL makes the default explicit even when a helper rule comes first. Run: make.

Makefile

```
.DEFAULT_GOAL := all
helper:
	@printf "helper\n"
all:
	@printf "default all\n"
```

Download: MAK-01-003.mk · MAK-01-003.sh

Actual output

Captured run · exit 0

```
default all
```

MAK-01-004

Request a named target

Only the requested target and its prerequisites are considered. The default goal is irrelevant to make check.

Makefile

```
all:
	@printf "build\n"
check:
	@printf "check\n"
```

Download: MAK-01-004.mk · MAK-01-004.sh

Actual output

Captured run · exit 0

```
check
```

MAK-01-005

Request two goals

Specify both goals on the command line. This serial invocation prints their results in the requested order.

Makefile

```
one:
	@printf "one\n"
two:
	@printf "two\n"
```

Download: MAK-01-005.mk · MAK-01-005.sh

Actual output

Captured run · exit 0

```
one
two
```

MAK-01-006

Use another makefile name

-f selects a makefile explicitly. The runnable script renames the displayed file to rules.mk. Run: make -f rules.mk hello.

Makefile

```
hello:
	@printf "rules.mk selected\n"
```

Actual output

Captured run · exit 0

```
rules.mk selected
```

Download: MAK-01-006.mk · MAK-01-006.sh

02 - Files and incremental builds

MAK-02-001

Create a missing output

The missing report.txt must be built from input.txt. The setup file contains a single record, 7. Run: make report.txt, then cat report.txt.

Makefile

```
report.txt: input.txt
	@printf "building report\n"
	@cp input.txt report.txt
```

Download: MAK-02-001.mk · MAK-02-001.sh

Actual output

Captured run · exit 0

```
building report
7
```

MAK-02-002

Skip an up-to-date output

The second invocation has nothing to rebuild. A label separates the two runs; make -s keeps the no-op quiet.

Makefile

```
report.txt: input.txt
	@printf "building report\n"
	@cp input.txt report.txt
```

Actual output

Captured run · exit 0

```
first run
building report
second run
```

Download: MAK-02-002.mk · MAK-02-002.sh

MAK-02-003

Rebuild after a newer prerequisite

Changing the input only triggers a rebuild when its timestamp is newer. The script assigns explicit past timestamps to avoid sleeps and filesystem-resolution races.

Makefile

```
report.txt: input.txt
	@printf "building report\n"
	@cp input.txt report.txt
```

Actual output

Captured run · exit 0

```
building report
building report
9
```

Download: MAK-02-003.mk · MAK-02-003.sh

MAK-02-004

Content changes are not enough

Make compares timestamps, not file contents. The script replaces the input but dates it earlier than the report; the report keeps its old value. Content hashing needs an explicit design.

Makefile

```
report.txt: input.txt
	@cp input.txt report.txt
```

Download: MAK-02-004.mk · MAK-02-004.sh

Actual output

Captured run · exit 0

```
7
```

MAK-02-005

Build a dependency chain

Make completes the intermediate table before its dependent report. The order comes from the graph, not the order of recipe text.

Makefile

```
report.txt: table.txt
	@printf "report\n"
	@cp table.txt report.txt
table.txt: input.txt
	@printf "table\n"
	@cp input.txt table.txt
```

Download: MAK-02-005.mk · MAK-02-005.sh

Actual output

Captured run · exit 0

```
table
report
```

MAK-02-006

Share one prerequisite

Two outputs depend on the same prepared file. It is made once per invocation. Run: make all.

Makefile

```
all: left.txt right.txt
left.txt right.txt: prepared.txt
	@cp prepared.txt $@
prepared.txt:
	@printf "prepare once\n"
	@printf "ready\n" > $@
```

Download: MAK-02-006.mk · MAK-02-006.sh

Actual output

Captured run · exit 0

```
prepare once
```

03 - Phony goals and directories

MAK-03-001

A filename can hide a command goal

A file named check makes an ordinary target appear current. The script creates that file and demonstrates the skipped recipe. Compare with MAK-03-002.

Makefile

```
check:
	@printf "checked\n"
```

Download: MAK-03-001.mk · MAK-03-001.sh

Actual output

Captured run · exit 0

```
recipe was skipped
```

MAK-03-002

Declare a command goal phony

.PHONY says that check names an action rather than a real output. It runs even when a file named check exists.

Makefile

```
.PHONY: check
check:
	@printf "checked\n"
```

Download: MAK-03-002.mk · MAK-03-002.sh

Actual output

Captured run · exit 0

```
checked
```

MAK-03-003

Provide an aggregate all goal

An aggregate phony goal joins several real outputs without creating a file named all. Repeating make all does not rebuild current files.

Makefile

```
.PHONY: all
all: a.txt b.txt
a.txt b.txt:
	@printf "%s\n" "$@"
	@printf "ready\n" > $@
```

Download: MAK-03-003.mk · MAK-03-003.sh

Actual output

Captured run · exit 0

```
a.txt
b.txt
again
```

MAK-03-004

Create an output directory first

The vertical bar introduces an order-only prerequisite. Build the directory before report.txt without using its timestamp to decide whether the report is stale.

Makefile

```
build/report.txt: input.txt | build
	@cp input.txt $@
	@printf "report created\n"
build:
	@mkdir -p $@
```

Download: MAK-03-004.mk · MAK-03-004.sh

Actual output

Captured run · exit 0

```
report created
```

MAK-03-005

Directory changes do not rebuild outputs

Adding an unrelated file changes the directory timestamp. Because build is order-only, the existing report remains current.

Makefile

```
build/report.txt: input.txt | build
	@cp input.txt $@
	@printf "report created\n"
build:
	@mkdir -p $@
```

Download: MAK-03-005.mk · MAK-03-005.sh

Actual output

Captured run · exit 0

```
report created
again
```

MAK-03-006

A local clean goal

A clean action removes only the named generated output in the fresh demonstration directory. Phony clean never collides with a file of the same name.

Makefile

```
.PHONY: all clean
all: result.txt
result.txt:
	@printf "generated\n" > $@
clean:
	@rm -f result.txt
```

Download: MAK-03-006.mk · MAK-03-006.sh

Actual output

Captured run · exit 0

```
generated output removed
```

04 - Automatic variables

MAK-04-001

Name the current target with $@

$@ expands to the target being built. One rule can report or create the selected output name.

Makefile

```
alpha.txt beta.txt:
	@printf "%s\n" "$@"
	@printf "ready\n" > $@
```

Download: MAK-04-001.mk · MAK-04-001.sh

Actual output

Captured run · exit 0

```
beta.txt
```

MAK-04-002

Read the first prerequisite with $<

$< is the first prerequisite. Copying $< to $@ keeps this recipe independent of the actual names.

Makefile

```
copy.txt: source.txt
	@cp $< $@
```

Download: MAK-04-002.mk · MAK-04-002.sh

Actual output

Captured run · exit 0

```
source record
```

MAK-04-003

List unique prerequisites with $^

$^ contains prerequisite names with duplicates removed. A duplicate dependency still appears only once.

Makefile

```
all: alpha.txt beta.txt alpha.txt
	@printf "%s\n" "$^"
```

Download: MAK-04-003.mk · MAK-04-003.sh

Actual output

Captured run · exit 0

```
alpha.txt beta.txt
```

MAK-04-004

Keep duplicates with $+

$+ retains duplicates and their order. This is useful when repeated arguments matter, such as some linker command lines.

Makefile

```
all: alpha.txt beta.txt alpha.txt
	@printf "%s\n" "$+"
```

Download: MAK-04-004.mk · MAK-04-004.sh

Actual output

Captured run · exit 0

```
alpha.txt beta.txt alpha.txt
```

MAK-04-005

List newer prerequisites with $?

$? reports the prerequisites newer than the target. The script assigns fixed past timestamps so only changed.txt qualifies.

Makefile

```
report.txt: old.txt changed.txt
	@printf "newer: %s\n" "$?"
	@touch $@
```

Download: MAK-04-005.mk · MAK-04-005.sh

Actual output

Captured run · exit 0

```
newer: changed.txt
```

MAK-04-006

Separate normal and order-only inputs

$^ excludes order-only prerequisites; $| contains them. The build directory supplies ordering rather than content.

Makefile

```
build/report.txt: input.txt | build
	@printf "normal=%s order-only=%s\n" "$^" "$|"
	@cp $< $@
build:
	@mkdir -p $@
```

Download: MAK-04-006.mk · MAK-04-006.sh

Actual output

Captured run · exit 0

```
normal=input.txt order-only=build
```

05 - Variables and expansion

MAK-05-001

Define a reusable variable

Use $(NAME) to insert a Make variable. Single quotes in the recipe protect the expanded value as one shell argument.

Makefile

```
GREETING = Hello
all:
	@printf "%s\n" "$(GREETING), reader"
```

Download: MAK-05-001.mk · MAK-05-001.sh

Actual output

Captured run · exit 0

```
Hello, reader
```

MAK-05-002

Recursive assignment sees later values

= stores an expression that is expanded when used. Changing NAME after defining GREETING changes the eventual greeting.

Makefile

```
NAME = Ada
GREETING = Hello $(NAME)
NAME = Grace
all:
	@printf "%s\n" "$(GREETING)"
```

Download: MAK-05-002.mk · MAK-05-002.sh

Actual output

Captured run · exit 0

```
Hello Grace
```

MAK-05-003

Simple assignment captures the current value

:= expands its right-hand side immediately. GREETING therefore remembers Ada. Compare with MAK-05-002.

Makefile

```
NAME = Ada
GREETING := Hello $(NAME)
NAME = Grace
all:
	@printf "%s\n" "$(GREETING)"
```

Download: MAK-05-003.mk · MAK-05-003.sh

Actual output

Captured run · exit 0

```
Hello Ada
```

MAK-05-004

Provide a default with ?=

?= assigns a value only when the variable is undefined. A command-line value takes precedence over this default.

Makefile

```
MODE ?= release
all:
	@printf "%s\n" "$(MODE)"
```

Download: MAK-05-004.mk · MAK-05-004.sh

Actual output

Captured run · exit 0

```
release
debug
```

MAK-05-005

Append with +=

+= adds words to an existing variable. This is convenient for assembling option lists or target sets.

Makefile

```
WORDS := small
WORDS += complete
WORDS += examples
all:
	@printf "%s\n" "$(WORDS)"
```

Download: MAK-05-005.mk · MAK-05-005.sh

Actual output

Captured run · exit 0

```
small complete examples
```

MAK-05-006

Override from the command line

A command-line assignment beats an ordinary assignment in the Makefile. Run: make all NAME=Grace.

Makefile

```
NAME = Ada
all:
	@printf "Hello %s\n" "$(NAME)"
```

Actual output

Captured run · exit 0

```
Hello Grace
```

Download: MAK-05-006.mk · MAK-05-006.sh

06 - Variable scope and conditionals

MAK-06-001

Inspect the origin of a value

$(origin NAME) explains where NAME came from. This helps distinguish a Makefile value from a command-line override.

Makefile

```
NAME = Ada
all:
	@printf "%s: %s\n" "$(origin NAME)" "$(NAME)"
```

Download: MAK-06-001.mk · MAK-06-001.sh

Actual output

Captured run · exit 0

```
file: Ada
command line: Grace
```

MAK-06-002

An empty value is still defined

A command-line empty string counts as defined, so ?= does not replace it. This differs from a shell fallback that tests for nonempty text.

Makefile

```
NAME ?= Ada
all:
	@printf "[%s]\n" "$(NAME)"
```

Download: MAK-06-002.mk · MAK-06-002.sh

Actual output

Captured run · exit 0

```
[]
```

MAK-06-003

Export a value to the recipe shell

Exported Make variables become shell environment variables. Double dollars pass the dollar sign through Make to the shell.

Makefile

```
export LABEL = published
all:
	@printf "%s\n" "$$LABEL"
```

Download: MAK-06-003.mk · MAK-06-003.sh

Actual output

Captured run · exit 0

```
published
```

MAK-06-004

Use a target-specific value

A target-specific assignment applies when that target is built and is inherited by its prerequisites unless overridden.

Makefile

```
debug: MODE = debug
release: MODE = release
debug release:
	@printf "%s: %s\n" "$@" "$(MODE)"
```

Download: MAK-06-004.mk · MAK-06-004.sh

Actual output

Captured run · exit 0

```
debug: debug
release: release
```

MAK-06-005

Choose rules at parse time with ifeq

ifeq is processed while Make reads the file. It chooses the recipe text before any target is executed.

Makefile

```
MODE ?= release
ifeq ($(MODE),debug)
MESSAGE := extra diagnostics
else
MESSAGE := compact output
endif
all:
	@printf "%s\n" "$(MESSAGE)"
```

Download: MAK-06-005.mk · MAK-06-005.sh

Actual output

Captured run · exit 0

```
compact output
extra diagnostics
```

MAK-06-006

Deliberately override a command-line value

The override directive gives this assignment precedence over a command-line assignment. Use it only for values the Makefile really must control.

Makefile

```
override MODE := required
all:
	@printf "%s\n" "$(MODE)"
```

Actual output

Captured run · exit 0

```
required
```

Download: MAK-06-006.mk · MAK-06-006.sh

07 - Pattern rules and file discovery

MAK-07-001

Capture the stem with $*

% matches a stem shared by the target and prerequisite. $* contains that stem while the recipe runs.

Makefile

```
%.out: %.in
	@printf "stem=%s\n" "$*"
	@cp $< $@
```

Download: MAK-07-001.mk · MAK-07-001.sh

Actual output

Captured run · exit 0

```
stem=sample
```

MAK-07-002

Reuse one transformation rule

Both files follow the same pattern rule. tr uppercases their complete input. $< and $@ identify the current pair.

Makefile

```
all: alpha.upper beta.upper
%.upper: %.txt
	@tr "[:lower:]" "[:upper:]" < $< > $@
```

Download: MAK-07-002.mk · MAK-07-002.sh

Actual output

Captured run · exit 0

```
ADA
GRACE
```

MAK-07-003

Limit a static pattern to selected targets

A static pattern rule applies to the explicit target list before the second colon. It is more selective than a general implicit rule.

Makefile

```
OUTPUTS := alpha.out beta.out
all: $(OUTPUTS)
$(OUTPUTS): %.out: %.in
	@cp $< $@
```

Download: MAK-07-003.mk · MAK-07-003.sh

Actual output

Captured run · exit 0

```
one
two
```

MAK-07-004

An explicit recipe wins

An explicit recipe for special.out takes precedence over the generic pattern recipe for that target.

Makefile

```
%.out: %.in
	@cp $< $@
special.out: special.in
	@printf "special handling\n" > $@
```

Download: MAK-07-004.mk · MAK-07-004.sh

Actual output

Captured run · exit 0

```
special handling
```

MAK-07-005

Search for inputs with vpath

vpath supplies a search directory for matching prerequisites. $< resolves to the discovered path, so the recipe reads the correct file.

Makefile

```
vpath %.txt data
copy.txt: input.txt
	@printf "input=%s\n" "$<"
	@cp $< $@
```

Download: MAK-07-005.mk · MAK-07-005.sh

Actual output

Captured run · exit 0

```
input=data/input.txt
found record
```

MAK-07-006

Set a pattern-specific value

A pattern-specific assignment supplies a value to all matching targets. Other target patterns can use a different value.

Makefile

```
%.debug: MODE = debug
%.release: MODE = release
app.debug app.release:
	@printf "%s: %s\n" "$@" "$(MODE)"
```

Download: MAK-07-006.mk · MAK-07-006.sh

Actual output

Captured run · exit 0

```
app.debug: debug
app.release: release
```

08 - Functions for lists and paths

MAK-08-001

Discover existing files with wildcard

$(wildcard ...) returns matching existing filenames. The script creates alpha.txt and beta.txt before Make reads the expression.

Makefile

```
FILES := $(wildcard *.txt)
all:
	@printf "%s\n" "$(FILES)"
```

Download: MAK-08-001.mk · MAK-08-001.sh

Actual output

Captured run · exit 0

```
alpha.txt beta.txt
```

MAK-08-002

Transform filenames with patsubst

patsubst substitutes matching word patterns. Here source names become output names without executing a shell.

Makefile

```
SOURCES := alpha.txt beta.txt
OUTPUTS := $(patsubst %.txt,build/%.out,$(SOURCES))
all:
	@printf "%s\n" "$(OUTPUTS)"
```

Download: MAK-08-002.mk · MAK-08-002.sh

Actual output

Captured run · exit 0

```
build/alpha.out build/beta.out
```

MAK-08-003

Filter a list by suffix

filter keeps matching words; filter-out keeps the remainder. Both operate on whitespace-separated lists.

Makefile

```
FILES := notes.org data.tsv plot.gp
all:
	@printf "data: %s\n" "$(filter %.tsv,$(FILES))"
	@printf "other: %s\n" "$(filter-out %.tsv,$(FILES))"
```

Download: MAK-08-003.mk · MAK-08-003.sh

Actual output

Captured run · exit 0

```
data: data.tsv
other: notes.org plot.gp
```

MAK-08-004

Sort and remove duplicates

sort orders words and removes duplicates. It is suitable for a unique dependency list, but not for arguments where repetitions matter.

Makefile

```
FILES := beta.txt alpha.txt beta.txt
all:
	@printf "%s\n" "$(sort $(FILES))"
```

Download: MAK-08-004.mk · MAK-08-004.sh

Actual output

Captured run · exit 0

```
alpha.txt beta.txt
```

MAK-08-005

Build a list with foreach

foreach evaluates the expression once for each list item. addprefix supplies the corresponding directory prefixes.

Makefile

```
NAMES := alpha beta
OUTPUTS := $(foreach name,$(NAMES),$(addprefix build/,$(name)).out)
all:
	@printf "%s\n" "$(OUTPUTS)"
```

Download: MAK-08-005.mk · MAK-08-005.sh

Actual output

Captured run · exit 0

```
build/alpha.out build/beta.out
```

MAK-08-006

Capture a shell result once

$(shell ...) captures command output as a Make value. := evaluates this local command once as the file is read; repeated variable references reuse the value.

Makefile

```
VALUE := $(shell printf "computed")
all:
	@printf "%s / %s\n" "$(VALUE)" "$(VALUE)"
```

Actual output

Captured run · exit 0

```
computed / computed
```

Download: MAK-08-006.mk · MAK-08-006.sh

09 - The recipe shell

MAK-09-001

Escape a shell variable with $$

Make consumes one dollar sign. $$name reaches the shell as $name, while $(name) would refer to a Make variable.

Makefile

```
all:
	@name=Ada; printf "%s\n" "$$name"
```

Download: MAK-09-001.mk · MAK-09-001.sh

Actual output

Captured run · exit 0

```
Ada
```

MAK-09-002

Separate recipe lines use separate shells

A shell variable set on the first recipe line is absent on the next. ${value-unset} gives the shell a visible default.

Makefile

```
all:
	@value=kept
	@printf "[%s]\n" "$${value-unset}"
```

Download: MAK-09-002.mk · MAK-09-002.sh

Actual output

Captured run · exit 0

```
[unset]
```

MAK-09-003

Join commands that share state

Join commands with && to keep a directory change and its dependent command in the same shell. Each example starts in its own temporary directory.

Makefile

```
all:
	@mkdir -p sub
	@cd sub && printf "inside sub\n" > result.txt
	@cat sub/result.txt
```

Download: MAK-09-003.mk · MAK-09-003.sh

Actual output

Captured run · exit 0

```
inside sub
```

MAK-09-004

Use one shell for a complete recipe

.ONESHELL passes all recipe lines to one shell. Explicit -eu flags make an unset variable or a failed command stop the recipe.

Makefile

```
.ONESHELL:
SHELL := /bin/sh
.SHELLFLAGS := -eu -c
all:
	@value=kept
	printf "%s\n" "$$value"
```

Download: MAK-09-004.mk · MAK-09-004.sh

Actual output

Captured run · exit 0

```
kept
```

MAK-09-005

Choose a visible recipe prefix

.RECIPEPREFIX changes the recipe introducer from a tab to the first character of its value. The > prefix makes indentation visible.

Makefile

```
.RECIPEPREFIX := >
all:
>@printf "visible prefix\n"
```

Download: MAK-09-005.mk · MAK-09-005.sh

Actual output

Captured run · exit 0

```
visible prefix
```

MAK-09-006

Write a shell loop

The backslash continues the logical recipe line. $$item belongs to the shell loop, while Make processes the surrounding recipe first.

Makefile

```
all:
	@for item in alpha beta; do \
		printf "%s\n" "$$item"; \
	done
```

Actual output

Captured run · exit 0

```
alpha
beta
```

Download: MAK-09-006.mk · MAK-09-006.sh

10 - Includes and recursive Make

MAK-10-001

Include a small configuration file

include reads another makefile at this point. options.mk contains MODE = debug; the main file uses that setting.

Makefile

```
include options.mk
all:
	@printf "%s\n" "$(MODE)"
```

Download: MAK-10-001.mk · MAK-10-001.sh

Actual output

Captured run · exit 0

```
debug
```

MAK-10-002

Allow an optional include

-include permits a missing optional file. The default remains available when local.mk has not been created.

Makefile

```
-include local.mk
MODE ?= release
all:
	@printf "%s\n" "$(MODE)"
```

Download: MAK-10-002.mk · MAK-10-002.sh

Actual output

Captured run · exit 0

```
release
```

MAK-10-003

Generate an included makefile

Make can build a missing included file, then restart reading its configuration. The generated file defines MODE. .DEFAULT_GOAL prevents the helper rule becoming the default.

Makefile

```
.DEFAULT_GOAL := all
include options.mk
options.mk:
	@printf "writing options\n"
	@printf "MODE = generated\n" > $@
all:
	@printf "%s\n" "$(MODE)"
```

Download: MAK-10-003.mk · MAK-10-003.sh

Actual output

Captured run · exit 0

```
writing options
generated
```

MAK-10-004

Read several files with -f

Repeated -f options read makefiles in order. extra.mk appends a word to the list defined in the displayed base.mk.

Makefile

```
WORDS := base
all:
	@printf "%s\n" "$(WORDS)"
```

Download: MAK-10-004.mk · MAK-10-004.sh

Actual output

Captured run · exit 0

```
base extra
```

MAK-10-005

Delegate with $(MAKE)

$(MAKE) selects the same executable and marks a recursive invocation for Make. -C enters the child directory; the helper suppresses directory-announcement lines.

Makefile

```
.PHONY: all
all:
	@$(MAKE) -s -C child all
```

Download: MAK-10-005.mk · MAK-10-005.sh

Actual output

Captured run · exit 0

```
child built
```

MAK-10-006

Forward a command-line variable

Command-line assignments propagate to a recursive Make through its flags. The child Makefile uses a default that the parent invocation overrides.

Makefile

```
.PHONY: all
all:
	@$(MAKE) -s -C child all
```

Actual output

Captured run · exit 0

```
child sees Grace
```

Download: MAK-10-006.mk · MAK-10-006.sh

11 - Parallel builds

MAK-11-001

Build independent outputs concurrently

-j2 permits two recipes at once. Each writes its own file; the final cat uses a fixed order, independent of job scheduling.

Makefile

```
all: alpha.txt beta.txt
alpha.txt beta.txt:
	@printf "%s\n" "$@" > $@
```

Download: MAK-11-001.mk · MAK-11-001.sh

Actual output

Captured run · exit 0

```
alpha.txt
beta.txt
```

MAK-11-002

Dependencies enforce order under -j

Both outputs require prepared.txt. Their recipes cannot start before its recipe finishes, even with parallel execution enabled.

Makefile

```
all: alpha.txt beta.txt
alpha.txt beta.txt: prepared.txt
	@test -s prepared.txt
	@cp prepared.txt $@
prepared.txt:
	@printf "prepared\n" > $@
```

Download: MAK-11-002.mk · MAK-11-002.sh

Actual output

Captured run · exit 0

```
prepared
prepared
```

MAK-11-003

Group output per target

--output-sync=target keeps each recipe output together. The order of the left and right blocks may vary; both complete orders are accepted.

Makefile

```
all: left right
.PHONY: all left right
left right:
	@printf "%s begin\n" "$@"
	@printf "%s end\n" "$@"
```

Download: MAK-11-003.mk · MAK-11-003.sh

Actual output

Captured run · exit 0

```
left begin
left end
right begin
right end
```

MAK-11-004

Use one job for serial output

-j1 requests one job at a time. With this prerequisite list, the two command goals finish in left-to-right order.

Makefile

```
.PHONY: all left right
all: left right
left right:
	@printf "%s\n" "$@"
```

Download: MAK-11-004.mk · MAK-11-004.sh

Actual output

Captured run · exit 0

```
left
right
```

MAK-11-005

Disable parallel execution for an invocation

An empty .NOTPARALLEL target serializes this Make invocation even when -j2 is supplied. Recursive invocations have their own parallel policy.

Makefile

```
.NOTPARALLEL:
.PHONY: all left right
all: left right
left right:
	@printf "%s\n" "$@"
```

Download: MAK-11-005.mk · MAK-11-005.sh

Actual output

Captured run · exit 0

```
left
right
```

MAK-11-006

Share jobs with recursive Make

Use $(MAKE) for recursion so the child participates in the jobserver. Avoid adding an independent -j value to every child. Each module writes its own output.

Makefile

```
.PHONY: all alpha beta
all: alpha beta
alpha beta:
	@$(MAKE) -s -C $@ all
```

Actual output

Captured run · exit 0

```
alpha
beta
```

Download: MAK-11-006.mk · MAK-11-006.sh

12 - Inspect and diagnose a build

MAK-12-001

Preview an ordinary recipe with -n

-n prints this ordinary recipe without running it. The follow-up test confirms that result.txt was not created. Recursive $(MAKE), + recipes and makefile remakes have special dry-run behavior.

Makefile

```
result.txt:
	@printf "generated\n" > result.txt
```

Actual output

Captured run · exit 0

```
printf "generated\n" > result.txt
file remains absent
```

Download: MAK-12-001.mk · MAK-12-001.sh

MAK-12-002

Ask whether an output is current

-q runs no ordinary recipe: exit 0 means current, 1 means a rebuild is needed, and 2 means an error. The script distinguishes these statuses rather than treating every failure as stale.

Makefile

```
result.txt: input.txt
	@cp $< $@
```

Actual output

Captured run · exit 0

```
needs rebuild
current
```

Download: MAK-12-002.mk · MAK-12-002.sh

MAK-12-003

Trace why a recipe runs

--trace reports the rule and the prerequisite that prompted rebuilding. The script replaces the numeric line location with LINE for a stable transcript.

Makefile

```
copy.txt: source.txt
	@cp source.txt copy.txt
```

Actual output

Captured run · exit 0

```
Makefile:LINE: update target 'copy.txt' due to: source.txt
cp source.txt copy.txt
```

Download: MAK-12-003.mk · MAK-12-003.sh

MAK-12-004

Print a parse-time diagnostic

$(info ...) writes a message as the file is read. This precedes the target recipe and helps inspect an expanded variable.

Makefile

```
MODE := debug
$(info reading MODE=$(MODE))
all:
	@printf "running target\n"
```

Download: MAK-12-004.mk · MAK-12-004.sh

Actual output

Captured run · exit 0

```
reading MODE=debug
running target
```

MAK-12-005

Reject an invalid configuration

This is an intentional error. $(error ...) stops reading the file before the recipe can run; Make returns status 2. The captured standard error explains the missing setting.

Makefile

```
ifeq ($(strip $(INPUT)),)
$(error INPUT is required)
endif
all:
	@printf "would run\n"
```

Actual output

Captured run · exit 2

```
(no stdout)
```

Standard error

```
Makefile:2: *** INPUT is required.  Stop.
```

Download: MAK-12-005.mk · MAK-12-005.sh

MAK-12-006

Diagnose spaces where a tab is required

This is an intentional syntax error: the recipe begins with four spaces. Make reports missing separator and exits 2. Compare with MAK-01-001 or the visible prefix in MAK-09-005.

Makefile

```
all:
    printf "bad indentation\n"
```

Actual output

Captured run · exit 2

```
(no stdout)
```

Standard error

```
Makefile:2: *** missing separator.  Stop.
```

Download: MAK-12-006.mk · MAK-12-006.sh

13 - Reliable outputs and a complete workflow

MAK-13-001

Delete a target when its recipe fails

.DELETE_ON_ERROR removes an output modified by a failed recipe. The script verifies that the incomplete file is gone; this expected failure is caught explicitly.

Makefile

```
.DELETE_ON_ERROR:
report.txt:
	@printf "incomplete\n" > $@
	@false
```

Download: MAK-13-001.mk · MAK-13-001.sh

Actual output

Captured run · exit 0

```
partial output removed
```

MAK-13-002

Publish an output with an atomic rename

Write to a temporary sibling first, then rename it only after success. A simulated failure before mv preserves the previously complete report.

Makefile

```
.PHONY: all broken
all: report.txt
report.txt: input.txt
	@cp $< $@.tmp && mv $@.tmp $@
broken:
	@printf "partial\n" > report.txt.tmp; false
```

Download: MAK-13-002.mk · MAK-13-002.sh

Actual output

Captured run · exit 0

```
complete record
```

MAK-13-003

Declare outputs produced together

&: is a grouped-target rule, available in GNU Make 4.3 and later. One invocation creates both files. Asking for both with -j2 must still run the recipe only once.

Makefile

```
all: summary.txt count.txt
summary.txt count.txt &: input.txt
	@printf "one grouped recipe\n"
	@cp input.txt summary.txt
	@wc -l < input.txt | tr -d " " > count.txt
```

Actual output

Captured run · exit 0

```
one grouped recipe
Ada
Grace
2
```

Download: MAK-13-003.mk · MAK-13-003.sh

MAK-13-004

Connect data, AWK, gnuplot and HTML

A complete dependency graph aggregates three input records, plots the totals and writes a linked HTML report. data.tsv contains books 4, pens 2 and books 3. The full runnable example includes plot.gp. Run: make report.html.

Makefile

```
.DEFAULT_GOAL := report.html
totals.tsv: data.tsv
	@printf "aggregate data\n"
	@awk '{sum[$$1]+=$$2} END {for (key in sum) print key, sum[key]}' $< | sort >
$@
chart.svg: totals.tsv plot.gp
	@printf "draw chart\n"
	@gnuplot plot.gp
report.html: chart.svg
	@printf "publish report\n"
	@printf '<!doctype html><title>Make report</title><h1>Totals</h1><img
src="chart.svg" alt="Totals by category">\n' > $@
```

Download: MAK-13-004.mk · MAK-13-004.sh

Actual output

Captured run · exit 0

```
aggregate data
draw chart
publish report
books 7
pens 2
HTML and SVG ready
```

MAK-13-005

Rebuild only a changed downstream branch

Changing plot.gp requires a new chart and HTML report, while totals.tsv remains current. Fixed past timestamps make the dependency test repeatable without sleeping.

Makefile

```
.DEFAULT_GOAL := report.html
totals.tsv: data.tsv
	@printf "aggregate data\n"
	@awk '{sum[$$1]+=$$2} END {for (key in sum) print key, sum[key]}' $< | sort >
$@
chart.svg: totals.tsv plot.gp
	@printf "draw chart\n"
	@gnuplot plot.gp
report.html: chart.svg
	@printf "publish report\n"
	@printf '<!doctype html><title>Make report</title><h1>Totals</h1><img
src="chart.svg" alt="Totals by category">\n' > $@
```

Download: MAK-13-005.mk · MAK-13-005.sh

Actual output

Captured run · exit 0

```
aggregate data
draw chart
publish report
change only plotting settings
draw chart
publish report
```

MAK-13-006

Stage and package a local website

Stage only public HTML and SVG files, then archive that directory. This example performs no network deployment. Keep actual SSH uploads as a separate, explicit action.

Makefile

```
.DEFAULT_GOAL := report.html
totals.tsv: data.tsv
	@printf "aggregate data\n"
	@awk '{sum[$$1]+=$$2} END {for (key in sum) print key, sum[key]}' $< | sort >
$@
chart.svg: totals.tsv plot.gp
	@printf "draw chart\n"
	@gnuplot plot.gp
report.html: chart.svg
	@printf "publish report\n"
	@printf '<!doctype html><title>Make report</title><h1>Totals</h1><img
src="chart.svg" alt="Totals by category">\n' > $@

.PHONY: stage package
stage: report.html
	@mkdir -p public
	@cp report.html public/index.html
	@cp chart.svg public/chart.svg
package: stage
	@tar -czf website.tar.gz -C public .
```

Actual output

Captured run · exit 0

```
aggregate data
draw chart
publish report
public/chart.svg
public/index.html
website.tar.gz ready
```

Download: MAK-13-006.mk · MAK-13-006.sh

14 - LaTeX documents, references and generated figures

MAK-14-001

Build a PDF with latexmk

Make coordinates the project; latexmk runs the TeX passes needed for a stable PDF. The real target is build/main.pdf, with build as an order-only prerequisite. The recipe keeps auxiliary files and compiler diagnostics in build/. .DELETE_ON_ERROR removes a file target modified by a failed recipe. If compilation fails, it prints the saved log and returns failure. The driver checks the actual PDF text. Run: make all. The -norc flag makes this demonstration independent of personal latexmk settings.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
build/main.pdf: main.tex | build
	@printf "compile main.tex\n"
	@$(LATEXMK) $(FLAGS) -outdir=build $< > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
```

Download: MAK-14-001.mk · MAK-14-001.sh

Actual output

Captured run · exit 0

```
compile main.tex
PDF text verified; second build is current
```

MAK-14-002

Rebuild after editing an included chapter

A file target needs every relevant input in its prerequisites, including included chapters and custom .cls/.sty files. This small project lists sections/*.tex explicitly with wildcard. The driver changes the chapter text, assigns controlled past timestamps and verifies that the new text reaches the PDF. Wildcard covers existing files in this directory; it does not discover arbitrary nested inputs. See MAK-14-003 for delegation to latexmk.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
CHAPTERS := $(wildcard sections/*.tex)
.PHONY: all
all: build/main.pdf
build/main.pdf: main.tex $(CHAPTERS) | build
	@printf "compile document and chapters\n"
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
```

Download: MAK-14-002.mk · MAK-14-002.sh

Actual output

Captured run · exit 0

```
compile document and chapters
compile document and chapters
edited chapter appears in the PDF
```

MAK-14-003

Let latexmk discover inputs; preview a watch command

The pdf goal is phony, so Make always asks latexmk to inspect the document. Latexmk tracks inputs and reruns TeX only when needed; the driver verifies an unchanged TeX log on the second invocation, then changes an included chapter. This avoids maintaining a second list of nested TeX inputs. Declare generated figures or tables as pdf prerequisites so Make creates them first. The watch goal uses -pvc for continuous preview. Only make -n watch is exercised here: run make watch interactively and stop it with Ctrl-C. A latexmk watcher does not itself invoke unrelated Make rules when their source data changes.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := pdf
.PHONY: pdf watch
pdf: | build
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
watch: | build
	@$(LATEXMK) $(FLAGS) -outdir=build -pvc main.tex
build:
	@mkdir -p $@
```

Download: MAK-14-003.mk · MAK-14-003.sh

Actual output

Captured run · exit 0

```
unchanged input: TeX engine skipped
changed input: PDF updated
watch command checked without starting a watcher
```

MAK-14-004

Resolve a classic BibTeX bibliography

List the .bib database as a prerequisite and let latexmk choose the BibTeX and TeX passes. The sample cites a fictional book defined in references.bib. The driver checks the .bbl, the final PDF text and the absence of unresolved citations in the final TeX log. When using a local .bst file, add it to the prerequisites too. Run: make all.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
build/main.pdf: main.tex references.bib | build
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
```

Download: MAK-14-004.mk · MAK-14-004.sh

Actual output

Captured run · exit 0

```
BibTeX bibliography resolved and PDF verified
```

MAK-14-005

Use biblatex and Biber, then update a citation

With biblatex configured for backend=biber, latexmk detects and runs Biber automatically. Do not also run BibTeX on the same auxiliary data. The driver builds a real citation, changes the database title and verifies the revised reference in the PDF. This example needs compatible biblatex and Biber installations, normally supplied together by a TeX distribution. The bibliography entry is fictional.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
build/main.pdf: main.tex references.bib | build
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
```

Download: MAK-14-005.mk · MAK-14-005.sh

Actual output

Captured run · exit 0

```
Biber citation resolved
changed bibliography appears in the PDF
```

MAK-14-006

Build a gnuplot figure before the document

Make owns the graph data.tsv -> figures/chart.pdf -> build/main.pdf. gnuplot uses the pdfcairo terminal; graphicx includes that generated PDF without TeX shell escape. Both the data and plotting script are prerequisites of the figure. The driver changes one value and verifies that the figure and document really rebuild. Keep generated figures as file targets so unchanged ones can be reused. Run: make all.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
figures/chart.pdf: data.tsv plot.gp | figures
	@printf "generate PDF figure\n"
	@gnuplot plot.gp
build/main.pdf: main.tex figures/chart.pdf | build
	@printf "compile document with figure\n"
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build figures:
	@mkdir -p $@
```

Download: MAK-14-006.mk · MAK-14-006.sh

Actual output

Captured run · exit 0

```
generate PDF figure
compile document with figure
generate PDF figure
compile document with figure
updated figure and document verified
```

MAK-14-007

Generate a LaTeX table from data with AWK

A generated table is a normal prerequisite of the PDF. The AWK recipe emits rows for a tabular environment, writes a temporary sibling and renames it on success. $$1 and $$2 survive Make expansion as AWK field references. This fixture accepts only plain alphabetic labels and integer values; arbitrary text needs LaTeX escaping. The driver changes books from 4 to 7 and checks both the table source and PDF text.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
tables/rows.tex: data.tsv | tables
	@printf "generate table rows\n"
	@awk '{printf "%s & %d \\\\\n", $$1, $$2}' $< > $@.tmp && mv $@.tmp $@
build/main.pdf: main.tex tables/rows.tex | build
	@printf "compile document with table\n"
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build tables:
	@mkdir -p $@
```

Download: MAK-14-007.mk · MAK-14-007.sh

Actual output

Captured run · exit 0

```
generate table rows
compile document with table
generate table rows
compile document with table
edited data appears in the table and PDF
```

MAK-14-008

Keep draft and final builds in separate directories

draft.tex defines DraftVersion and both wrappers input main.tex. The pattern rule uses the stem ($*) to choose a wrapper, a separate output directory ($(@D)) for each variant, and jobname=main for a consistent PDF filename. Auxiliary state is never shared between draft and final. The driver inspects both PDFs, then confirms that make -q all is current. Run: make draft, make final or make all.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all draft final
all: draft final
draft: build/draft/main.pdf
final: build/final/main.pdf
build/%/main.pdf: %.tex main.tex
	@mkdir -p $(@D)
	@$(LATEXMK) $(FLAGS) -outdir=$(@D) -jobname=main $*.tex > $(@D)/latexmk.log
2>&1 || \
	  { cat $(@D)/latexmk.log >&2; exit 1; }
```

Download: MAK-14-008.mk · MAK-14-008.sh

Actual output

Captured run · exit 0

```
draft and final PDFs have independent auxiliary state
```

MAK-14-009

Track project compiler settings and resolve references

Put project settings in a versioned latexmkrc and load it explicitly with -norc -r. List the settings file as a prerequisite so editing it schedules another build. The Perl %O and %S placeholders are latexmk command substitutions. This document contains a section reference; latexmk performs the passes needed to resolve it. The driver checks the final PDF and rejects unresolved-reference warnings.

Makefile

```
LATEXMK ?= latexmk
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all
all: build/main.pdf
build/main.pdf: main.tex latexmkrc | build
	@$(LATEXMK) -norc -r latexmkrc -pdf -outdir=build main.tex > build/
latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
```

Download: MAK-14-009.mk · MAK-14-009.sh

Actual output

Captured run · exit 0

```
project settings loaded; section reference resolved
```

MAK-14-010

Build independent documents in parallel

Each document gets its own directory and log, so make -j2 can build both without colliding auxiliary filenames. The list functions produce build/article/main.pdf and build/notes/main.pdf; the pattern rule chooses article.tex or notes.tex using $*. Shared generated assets must be declared as prerequisites before adding them. The driver verifies both PDFs and reports a fixed result after both jobs finish.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
DOCS := article notes
PDFS := $(addsuffix /main.pdf,$(addprefix build/,$(DOCS)))
.PHONY: all
all: $(PDFS)
build/%/main.pdf: %.tex
	@mkdir -p $(@D)
	@$(LATEXMK) $(FLAGS) -outdir=$(@D) -jobname=main $< > $(@D)/latexmk.log 2>&1
|| \
	  { cat $(@D)/latexmk.log >&2; exit 1; }
```

Download: MAK-14-010.mk · MAK-14-010.sh

Actual output

Captured run · exit 0

```
two independent PDFs built with -j2
```

MAK-14-011

Clean auxiliaries while keeping the PDF

latexmk -c removes regenerable auxiliary files while keeping the final PDF. This project puts all generated document files in build/, so distclean can remove that owned directory. Neither goal deletes main.tex or source figures. The driver checks that clean keeps the PDF, distclean removes build/ and a subsequent build works. Do not place authoritative source files in the disposable build directory.

Makefile

```
LATEXMK ?= latexmk
FLAGS := -norc -pdf -no-shell-escape
FLAGS += -interaction=nonstopmode -halt-on-error
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
.PHONY: all clean distclean
all: build/main.pdf
build/main.pdf: main.tex | build
	@$(LATEXMK) $(FLAGS) -outdir=build main.tex > build/latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build:
	@mkdir -p $@
clean:
	@$(LATEXMK) -norc -c -outdir=build main.tex > cleanup.log 2>&1 || \
	  { cat cleanup.log >&2; exit 1; }
distclean: clean
	@rm -rf -- build
```

Download: MAK-14-011.mk · MAK-14-011.sh

Actual output

Captured run · exit 0

```
clean: PDF and source retained
distclean: generated directory removed, source retained
document rebuilt after cleanup
```

MAK-14-012

Package sources and verify a fresh rebuild

Archive the Makefile, document, included chapter and compiler settings explicitly. The package goal first verifies the local PDF build; auxiliary caches and local output directories are excluded from the source archive. The driver unpacks into another fresh directory and builds there with make -C, checking the resulting PDF text. Add real bibliography, class/style and figure inputs to the archive list when needed. This is a source package for collaborators with the documented tools installed, not a claim of byte-identical PDFs across TeX distributions.

Makefile

```
LATEXMK ?= latexmk
.DELETE_ON_ERROR:
.DEFAULT_GOAL := all
SOURCES := Makefile main.tex sections/intro.tex latexmkrc
.PHONY: all package
all: build/main.pdf
build/main.pdf: main.tex sections/intro.tex latexmkrc | build
	@$(LATEXMK) -norc -r latexmkrc -pdf -outdir=build main.tex > build/
latexmk.log 2>&1 || \
	  { cat build/latexmk.log >&2; exit 1; }
build dist:
	@mkdir -p $@
package: dist/paper-source.tar.gz
dist/paper-source.tar.gz: $(SOURCES) build/main.pdf | dist
	@tar -czf $@ $(SOURCES)
```

Download: MAK-14-012.mk · MAK-14-012.sh

Actual output

Captured run · exit 0

```
Makefile
main.tex
sections/intro.tex
latexmkrc
unpacked sources build a verified PDF
```

Example and command index

- MAK-01-001 · A target and its recipe
- MAK-01-002 · Silence recipe echo with @
- MAK-01-003 · Choose the default goal
- MAK-01-004 · Request a named target
- MAK-01-005 · Request two goals
- MAK-01-006 · Use another makefile name
- MAK-02-001 · Create a missing output
- MAK-02-002 · Skip an up-to-date output
- MAK-02-003 · Rebuild after a newer prerequisite
- MAK-02-004 · Content changes are not enough
- MAK-02-005 · Build a dependency chain
- MAK-02-006 · Share one prerequisite
- MAK-03-001 · A filename can hide a command goal
- MAK-03-002 · Declare a command goal phony
- MAK-03-003 · Provide an aggregate all goal
- MAK-03-004 · Create an output directory first
- MAK-03-005 · Directory changes do not rebuild outputs
- MAK-03-006 · A local clean goal
- MAK-04-001 · Name the current target with $@
- MAK-04-002 · Read the first prerequisite with $<
- MAK-04-003 · List unique prerequisites with $^
- MAK-04-004 · Keep duplicates with $+
- MAK-04-005 · List newer prerequisites with $?
- MAK-04-006 · Separate normal and order-only inputs
- MAK-05-001 · Define a reusable variable
- MAK-05-002 · Recursive assignment sees later values
- MAK-05-003 · Simple assignment captures the current value
- MAK-05-004 · Provide a default with ?=
- MAK-05-005 · Append with +=
- MAK-05-006 · Override from the command line
- MAK-06-001 · Inspect the origin of a value
- MAK-06-002 · An empty value is still defined
- MAK-06-003 · Export a value to the recipe shell
- MAK-06-004 · Use a target-specific value
- MAK-06-005 · Choose rules at parse time with ifeq
- MAK-06-006 · Deliberately override a command-line value
- MAK-07-001 · Capture the stem with $*
- MAK-07-002 · Reuse one transformation rule
- MAK-07-003 · Limit a static pattern to selected targets
- MAK-07-004 · An explicit recipe wins
- MAK-07-005 · Search for inputs with vpath
- MAK-07-006 · Set a pattern-specific value
- MAK-08-001 · Discover existing files with wildcard
- MAK-08-002 · Transform filenames with patsubst
- MAK-08-003 · Filter a list by suffix
- MAK-08-004 · Sort and remove duplicates
- MAK-08-005 · Build a list with foreach
- MAK-08-006 · Capture a shell result once
- MAK-09-001 · Escape a shell variable with $$
- MAK-09-002 · Separate recipe lines use separate shells
- MAK-09-003 · Join commands that share state
- MAK-09-004 · Use one shell for a complete recipe
- MAK-09-005 · Choose a visible recipe prefix
- MAK-09-006 · Write a shell loop

- MAK-10-001 · Include a small configuration file
- MAK-10-002 · Allow an optional include
- MAK-10-003 · Generate an included makefile
- MAK-10-004 · Read several files with -f
- MAK-10-005 · Delegate with $(MAKE)
- MAK-10-006 · Forward a command-line variable
- MAK-11-001 · Build independent outputs concurrently
- MAK-11-002 · Dependencies enforce order under -j
- MAK-11-003 · Group output per target
- MAK-11-004 · Use one job for serial output
- MAK-11-005 · Disable parallel execution for an invocation
- MAK-11-006 · Share jobs with recursive Make
- MAK-12-001 · Preview an ordinary recipe with -n
- MAK-12-002 · Ask whether an output is current
- MAK-12-003 · Trace why a recipe runs
- MAK-12-004 · Print a parse-time diagnostic
- MAK-12-005 · Reject an invalid configuration
- MAK-12-006 · Diagnose spaces where a tab is required
- MAK-13-001 · Delete a target when its recipe fails
- MAK-13-002 · Publish an output with an atomic rename
- MAK-13-003 · Declare outputs produced together
- MAK-13-004 · Connect data, AWK, gnuplot and HTML
- MAK-13-005 · Rebuild only a changed downstream branch
- MAK-13-006 · Stage and package a local website
- MAK-14-001 · Build a PDF with latexmk
- MAK-14-002 · Rebuild after editing an included chapter
- MAK-14-003 · Let latexmk discover inputs; preview a watch command
- MAK-14-004 · Resolve a classic BibTeX bibliography
- MAK-14-005 · Use biblatex and Biber, then update a citation
- MAK-14-006 · Build a gnuplot figure before the document
- MAK-14-007 · Generate a LaTeX table from data with AWK
- MAK-14-008 · Keep draft and final builds in separate directories
- MAK-14-009 · Track project compiler settings and resolve references
- MAK-14-010 · Build independent documents in parallel
- MAK-14-011 · Clean auxiliaries while keeping the PDF
- MAK-14-012 · Package sources and verify a fresh rebuild

Content: CC0 1.0. Structure and publishing framework: CC BY 4.0 by Lukas C. Bossert. Repository source · Publishing framework source · License scope and third-party notices.