# Unix Makefile Tutorial

Makefile là một công cụ xây dựng chương trình chạy trên Unix, Linux và các hương vị của chúng. Nó hỗ trợ đơn giản hóa việc thực thi chương trình xây dựng có thể cần nhiều mô-đun khác nhau. Để xác định cách các mô-đun cần được biên dịch hoặc biên dịch lại cùng nhau, thực hiện **make** với sự trợ giúp của các tệp thực hiện do người dùng xác định. Hướng dẫn này sẽ nâng cao kiến ​​thức của bạn về cấu trúc và tiện ích của makefile.

## Khán giả

Hướng dẫn Makefile tiện ích **make** trong khi biên dịch và liên kết các mô-đun chương trình. Bất kỳ ai muốn biên dịch chương trình của mình bằng tiện ích **make** và muốn có kiến ​​thức về makefile nên đọc hướng dẫn này.

## Điều kiện tiên quyết

Hướng dẫn này mong đợi sự hiểu biết tốt về ngôn ngữ lập trình như C và C++. Người đọc cần có kiến ​​thức về liên kết, tải các khái niệm, cách biên dịch và thực thi chương trình trong môi trường Unix/Linux.

# Why Makefile

Biên dịch các tập tin mã nguồn có thể tốn rất nhiều thời gian đặc biệt là khi bạn phải bao gồm nhiều tập tin mã nguồn và gõ câu lệnh biên dịch mỗi khi bạn cần biên dịch. Tập tin Makefile là giải pháp để đơn giản hóa nhiệm vụ này.

Tập tin Makefile là các tập tin định dạng đặc biệt giúp xây dựng và quản lý dự án tự động.

Ví dụ: Giả sử chúng ta có các tập tin mã nguồn sau.

* main.cpp
* hello.cpp
* factorial.cpp
* functions.h

**File:** main.cpp

|  |
| --- |
| #include <iostream>  using namespace std;    #include "functions.h"  int main(){  print\_hello();  cout << endl;  cout << "The factorial of 5 is " << factorial(5) << endl;  return 0;  } |

**File:** hello.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  #include "functions.h"  void print\_hello(){  cout << "Hello World!";  } |

**File:** factorial.cpp

|  |
| --- |
| #include "functions.h"  int factorial(int n){    if(n!=1){  return(n \* factorial(n-1));  } else return 1;  } |

**File:** functions.h

|  |
| --- |
| void print\_hello();  int factorial(int n); |

Cách đơn giản để biên dịch các tệp và lấy tệp thực thi là chạy lệnh.

|  |
| --- |
| gcc main.cpp hello.cpp factorial.cpp -o hello |

Lệnh này tạo nhị phân *xin chào .* Trong ví dụ này, chúng tôi chỉ có bốn tệp và chúng tôi biết trình tự của các lệnh gọi hàm. Do đó, có thể gõ lệnh trên và chuẩn bị một nhị phân cuối cùng.

Tuy nhiên, đối với một dự án lớn nơi chúng tôi có hàng nghìn tệp mã nguồn, việc duy trì các bản dựng nhị phân trở nên khó khăn.

Lệnh **make** cho phép bạn quản lý các chương trình hoặc nhóm chương trình lớn. Khi bạn bắt đầu viết các chương trình lớn, bạn nhận thấy rằng việc biên dịch lại các chương trình lớn mất nhiều thời gian hơn so với việc biên dịch lại các chương trình ngắn. Hơn nữa, bạn nhận thấy rằng bạn thường chỉ làm việc trên một phần nhỏ của chương trình (chẳng hạn như một hàm duy nhất), và phần lớn chương trình còn lại không thay đổi.

Trong phần tiếp theo, chúng ta sẽ thấy cách chuẩn bị tệp thực hiện cho dự án của mình.

# Makefile - Macro

Chương trình **make** cho phép bạn sử dụng các macro, tương tự như các biến. Macro được xác định trong Makefile dưới dạng = cặp. Một ví dụ đã được hiển thị dưới đây -

|  |
| --- |
| MACROS = -me  PSROFF = groff -Tps  DITROFF = groff -Tdvi  CFLAGS = -O -systype bsd43  LIBS = "-lncurses -lm -lsdl"  MYFACE = ":\*)" |

## Macro đặc biệt

Trước khi đưa ra bất kỳ lệnh nào trong bộ quy tắc đích, có một số macro đặc biệt được xác định trước :

* $@ là tên của tệp sẽ được tạo.
* $? là tên của những người phụ thuộc đã thay đổi.

Ví dụ: chúng ta có thể sử dụng một quy tắc như sau:

|  |
| --- |
| hello: main.cpp hello.cpp factorial.cpp  $(CC) $(CFLAGS) $? $(LDFLAGS) -o $@  Alternatively:  hello: main.cpp hello.cpp factorial.cpp  $(CC) $(CFLAGS) $@.cpp $(LDFLAGS) -o $@ |

Trong ví dụ này, $@ đại diện cho hello và $? hoặc $@.cpp chọn tất cả các tệp nguồn đã thay đổi.

Có hai macro đặc biệt hơn được sử dụng trong các quy tắc ngầm định.

* $< tên của tệp liên quan đã gây ra hành động.
* $\* tiền tố được chia sẻ bởi các tệp đích và tệp phụ thuộc.

Quy tắc ngầm chung là để xây dựng các tệp .o (đối tượng) từ .cpp (tệp nguồn).

|  |
| --- |
| .cpp.o:  $(CC) $(CFLAGS) -c $<  Alternatively:  .cpp.o:  $(CC) $(CFLAGS) -c $\*.c |

## Macro thông thường

Có nhiều macro mặc định khác nhau. Bạn có thể nhìn thấy chúng bằng cách gõ "make -p" để in ra các giá trị mặc định. Hầu hết là khá rõ ràng từ các quy tắc mà chúng được sử dụng.

Các biến được xác định trước này, tức là các macro được sử dụng trong các quy tắc ngầm được chia thành hai lớp. Chúng như sau -

* Macro là tên của các chương trình (chẳng hạn như CC)
* Macro có chứa đối số của chương trình (chẳng hạn như CFLAGS).

Dưới đây là bảng một số biến phổ biến được sử dụng làm tên chương trình trong các quy tắc tích hợp của makefiles −

|  |  |
| --- | --- |
| **Sơ Không** | **Biến & Mô tả** |
| 1 | **thực tế tăng cường**  Lưu trữ-bảo trì chương trình; mặc định là `ar'. |
| 2 | **BẰNG**  Chương trình biên dịch tập tin hợp ngữ; mặc định là `như'. |
| 3 | **CC**  Chương trình biên dịch chương trình C; mặc định là `cc'. |
| 4 | **khí CO**  Chương trình kiểm tra các tệp từ RCS; mặc định là `co'. |
| 5 | **CXX**  Chương trình biên dịch chương trình C++; mặc định là `g++'. |
| 6 | **CPP**  Chương trình để chạy bộ tiền xử lý C, với kết quả là đầu ra tiêu chuẩn; mặc định là `$(CC) -E'. |
| 7 | **FC**  Chương trình biên dịch hoặc tiền xử lý các chương trình Fortran và Ratfor; mặc định là `f77'. |
| số 8 | **LẤY**  Chương trình trích xuất tệp từ SCCS; mặc định là `lấy'. |
| 9 | **LEX**  Chương trình sử dụng để biến ngữ pháp Lex thành mã nguồn; mặc định là `lex'. |
| 10 | **YACC**  Chương trình sử dụng để biến ngữ pháp Yacc thành mã nguồn; mặc định là `yacc'. |
| 11 | **LINT**  Chương trình sử dụng để chạy lint trên mã nguồn; mặc định là `lint'. |
| 12 | **M2C**  Chương trình sử dụng để biên dịch mã nguồn Modula-2; mặc định là `m2c'. |
| 13 | **máy tính**  Chương trình biên dịch chương trình Pascal; mặc định là `pc'. |
| 14 | **THÔNG TIN**  Chương trình chuyển đổi tệp nguồn Texinfo thành tệp Thông tin; mặc định là `makeinfo'. |
| 15 | **văn bản**  Chương trình tạo tệp TeX dvi từ nguồn TeX; mặc định là `tex'. |
| 16 | **TEXI2DVI**  Chương trình tạo tệp TeX dvi từ nguồn Texinfo; mặc định là `texi2dvi'. |
| 17 | **DỆT**  Chương trình dịch Web sang TeX; mặc định là `dệt'. |
| 18 | **DỆT**  Chương trình dịch C Web sang TeX; mặc định là `cweave'. |
| 19 | **TANG**  Chương trình dịch Web sang Pascal; mặc định là `tangle'. |
| 20 | **CTANGLE**  Chương trình dịch C Web sang C; mặc định là `ctangle'. |
| 21 | **RM**  Lệnh xóa tệp; mặc định là `rm -f'. |

Đây là bảng các biến có giá trị là đối số bổ sung cho các chương trình trên. Các giá trị mặc định cho tất cả các giá trị này là chuỗi rỗng, trừ khi có ghi chú khác.

|  |  |
| --- | --- |
| **Sơ Không.** | **Biến & Mô tả** |
| 1 | **ARFLAGS**  Cờ để cung cấp cho chương trình lưu trữ duy trì; mặc định là `rv'. |
| 2 | **ASFLAG**  Các cờ bổ sung để cung cấp cho trình hợp dịch mã khi được gọi một cách rõ ràng trên tệp `.s' hoặc `.S'. |
| 3 | **CỜ**  Cờ bổ sung để cung cấp cho trình biên dịch C. |
| 4 | **CXXFLAG**  Cờ bổ sung để cung cấp cho trình biên dịch C. |
| 5 | **COFLAGS**  Cờ bổ sung để cung cấp cho chương trình đồng RCS. |
| 6 | **CPPFLAGS**  Các cờ bổ sung để cung cấp cho bộ tiền xử lý C và các chương trình sử dụng nó (chẳng hạn như trình biên dịch C và Fortran). |
| 7 | **CỜ**  Cờ bổ sung để cung cấp cho trình biên dịch Fortran. |
| số 8 | **GFLAGS**  Các cờ bổ sung để cung cấp cho chương trình nhận SCCS. |
| 9 | **LDFLAGS**  Các cờ bổ sung để cung cấp cho trình biên dịch khi chúng được cho là gọi trình liên kết, `ld'. |
| 10 | **LFLAGS**  Thêm cờ để tặng cho Lex. |
| 11 | **YFLAGS**  Thêm cờ để tặng cho Yacc. |
| 12 | **CỜ**  Các cờ bổ sung để cung cấp cho trình biên dịch Pascal. |
| 13 | **RFLAGS**  Các cờ bổ sung để cung cấp cho trình biên dịch Fortran cho các chương trình Ratfor. |
| 14 | **lintflags**  Cờ bổ sung để cung cấp cho lint. |

**LƯU Ý** − Bạn có thể hủy tất cả các biến được sử dụng bởi các quy tắc ngầm với tùy chọn '-R' hoặc '--no-buildin-variables'.

Bạn cũng có thể xác định macro tại dòng lệnh như hình bên dưới:

|  |
| --- |
| make CPP = /home/courses/cop4530/spring02 |

# Defining Dependencies in Makefile

Điều rất phổ biến là một tệp nhị phân cuối cùng sẽ phụ thuộc vào các tệp tiêu đề nguồn và mã nguồn khác nhau. Các phụ thuộc rất quan trọng vì chúng cho phép **biết** về nguồn đối với bất kỳ mục tiêu nào. Xem xét ví dụ sau -

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

Ở đây, chúng tôi nói với **make** rằng hello phụ thuộc vào các tệp main.o, factorial.o và hello.o. Do đó, bất cứ khi nào có thay đổi trong bất kỳ tệp đối tượng nào, **make** sẽ thực hiện hành động.

Đồng thời, chúng ta cần cho người **tạo** biết cách chuẩn bị các tệp .o. Do đó, chúng ta cũng cần xác định các phụ thuộc đó như sau -

|  |
| --- |
| main.o: main.cpp functions.h  $(CC) -c main.cpp  factorial.o: factorial.cpp functions.h  $(CC) -c factorial.cpp  hello.o: hello.cpp functions.h  $(CC) -c hello.cpp |

# Xác định quy tắc trong Makefile

Bây giờ chúng ta sẽ tìm hiểu các quy tắc cho Makefile.

Cú pháp chung của quy tắc đích Makefile là -

|  |
| --- |
| target [target...] : [dependent ....]  [ command ...] |

Trong đoạn mã trên, các đối số trong ngoặc là tùy chọn và dấu chấm lửng có nghĩa là một hoặc nhiều. Ở đây, lưu ý rằng tab để mở đầu mỗi lệnh là bắt buộc.

Một ví dụ đơn giản được đưa ra dưới đây, nơi bạn xác định quy tắc để làm cho mục tiêu của bạn xin chào từ ba tệp khác.

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

**LƯU Ý** - Trong ví dụ này, bạn sẽ phải đưa ra các quy tắc để tạo tất cả các tệp đối tượng từ các tệp nguồn.

Ngữ nghĩa rất đơn giản. Khi bạn nói "tạo mục tiêu", lệnh **tạo sẽ** tìm quy tắc đích áp dụng; và, nếu bất kỳ người phụ thuộc nào mới hơn mục tiêu, **hãy** thực hiện từng lệnh một (sau khi thay thế macro). Nếu bất kỳ phụ thuộc nào phải được thực hiện, điều đó sẽ xảy ra trước (vì vậy bạn có một đệ quy).

**Thực hiện** chấm dứt nếu bất kỳ lệnh nào trả về trạng thái lỗi. Quy tắc sau đây sẽ được hiển thị trong trường hợp như vậy -

|  |
| --- |
| clean:  -rm \*.o \*~ core paper |

**Thực hiện** bỏ qua trạng thái được trả về trên các dòng lệnh bắt đầu bằng dấu gạch ngang. Ví dụ, ai quan tâm nếu không có tập tin cốt lõi?

**Thực hiện** lặp lại các lệnh, sau khi thay thế macro để cho bạn thấy điều gì đang xảy ra. Đôi khi bạn có thể muốn tắt nó đi. Ví dụ -

|  |
| --- |
| install:  @echo You must be root to install |

Mọi người đã mong đợi một số mục tiêu nhất định trong Makefiles. Bạn nên luôn luôn duyệt đầu tiên. Tuy nhiên, thật hợp lý khi hy vọng rằng tất cả các mục tiêu (hoặc chỉ tạo), cài đặt và dọn dẹp đều được tìm thấy.

* **make all** − Nó biên dịch mọi thứ để bạn có thể thực hiện kiểm tra cục bộ trước khi cài đặt ứng dụng.
* **thực hiện cài đặt** - Nó cài đặt các ứng dụng ở đúng nơi.
* **làm sạch** - Nó dọn dẹp các ứng dụng, loại bỏ các tệp thực thi, mọi tệp tạm thời, tệp đối tượng, v.v.

## Quy tắc ngầm Makefile

Lệnh này phải hoạt động trong mọi trường hợp khi chúng tôi xây dựng một tệp thực thi x từ mã nguồn x.cpp. Điều này có thể được phát biểu như một quy tắc ngầm -

|  |
| --- |
| .cpp:  $(CC) $(CFLAGS) $@.cpp $(LDFLAGS) -o $@ |

Quy tắc ngầm định này cho biết cách biến x ra khỏi xc -- chạy cc trên xc và gọi đầu ra là x. Quy tắc là ngầm định vì không có mục tiêu cụ thể nào được đề cập. Nó có thể được sử dụng trong mọi trường hợp.

Một quy tắc ngầm phổ biến khác là xây dựng các tệp .o (đối tượng) từ .cpp (tệp nguồn).

|  |
| --- |
| .cpp.o:  $(CC) $(CFLAGS) -c $<  alternatively  .cpp.o:  $(CC) $(CFLAGS) -c $\*.cpp |

# Xác định quy tắc hậu tố tùy chỉnh trong Makefile

**Make** có thể tự động tạo tệp ao, sử dụng cc -c trên tệp .c tương ứng. Các quy tắc này được tích hợp sẵn trong **make** và bạn có thể tận dụng lợi thế này để rút ngắn Makefile của mình. Nếu bạn chỉ cho biết các tệp .h trong dòng phụ thuộc của Makefile mà mục tiêu hiện tại phụ thuộc vào, thì **make** sẽ biết rằng .cfile tương ứng đã được yêu cầu. Bạn không cần phải bao gồm lệnh cho trình biên dịch.

Điều này làm giảm Makefile hơn nữa, như được hiển thị bên dưới -

|  |
| --- |
| OBJECTS = main.o hello.o factorial.o  hello: $(OBJECTS)  cc $(OBJECTS) -o hello  hellp.o: functions.h  main.o: functions.h  factorial.o: functions.h |

**Make** sử dụng một mục tiêu đặc biệt, có tên là *.SUFFIXES* để cho phép bạn xác định các hậu tố của riêng mình. Ví dụ: tham khảo dòng phụ thuộc được đưa ra bên dưới -

|  |
| --- |
| .SUFFIXES: .foo .bar |

Nó thông báo **rằng** bạn sẽ sử dụng các hậu tố đặc biệt này để tạo quy tắc của riêng mình.

Tương tự như cách **make** đã biết cách tạo tệp *.o* từ tệp *.c* , bạn có thể xác định các quy tắc theo cách sau –

|  |
| --- |
| .foo.bar:  tr '[A-Z][a-z]' '[N-Z][A-M][n-z][a-m]' < $< > $@  .c.o:  $(CC) $(CFLAGS) -c $< |

Quy tắc đầu tiên cho phép bạn tạo tệp *.bar từ tệp .foo* . Về cơ bản, nó xáo trộn tệp. Quy tắc thứ hai là quy tắc mặc định được sử dụng bởi **make** để tạo tệp *.o* từ tệp *.c* .

# Makefile - Chỉ thị

Có rất nhiều chỉ thị có sẵn trong các hình thức khác nhau. Chương trình **tạo** trên hệ thống của bạn có thể không hỗ trợ tất cả các lệnh. Vì vậy, vui lòng kiểm tra xem sản phẩm của bạn **có** hỗ trợ các chỉ thị mà chúng tôi đang giải thích ở đây không. **GNU** hỗ trợ các chỉ thị này.

## Chỉ thị có điều kiện

Các chỉ thị có điều kiện là -

* Lệnh **ifeq** bắt đầu điều kiện và chỉ định điều kiện. Nó chứa hai đối số, được phân tách bằng dấu phẩy và được bao quanh bởi dấu ngoặc đơn. Thay thế biến được thực hiện trên cả hai đối số và sau đó chúng được so sánh. Các dòng của tệp thực hiện sau ifeq được tuân theo nếu hai đối số khớp nhau; nếu không thì chúng bị bỏ qua.
* Lệnh **ifneq** bắt đầu điều kiện và chỉ định điều kiện. Nó chứa hai đối số, được phân tách bằng dấu phẩy và được bao quanh bởi dấu ngoặc đơn. Thay thế biến được thực hiện trên cả hai đối số và sau đó chúng được so sánh. Các dòng của tệp thực hiện sau ifneq được tuân theo nếu hai đối số không khớp; nếu không thì chúng bị bỏ qua.
* Lệnh **ifdef** bắt đầu điều kiện và chỉ định điều kiện. Nó chứa đối số duy nhất. Nếu đối số đã cho là đúng thì điều kiện trở thành đúng.
* Lệnh **ifndef** bắt đầu điều kiện và chỉ định điều kiện. Nó chứa đối số duy nhất. Nếu đối số đã cho là sai thì điều kiện trở thành đúng.
* Chỉ thị **khác** làm cho các dòng sau được tuân theo nếu điều kiện trước đó không thành công. Trong ví dụ trên, điều này có nghĩa là lệnh liên kết thay thế thứ hai được sử dụng bất cứ khi nào lệnh thay thế thứ nhất không được sử dụng. Không bắt buộc phải có thêm else trong câu điều kiện.
* Chỉ thị **endif** kết thúc điều kiện. Mỗi điều kiện phải kết thúc bằng một endif.

## Cú pháp của Chỉ thị Điều kiện

Cú pháp của một điều kiện đơn giản không có other như sau:

|  |
| --- |
| conditional-directive  text-if-true  endif |

Text-if-true có thể là bất kỳ dòng văn bản nào, được coi là một phần của tệp thực hiện nếu điều kiện là đúng. Nếu điều kiện là sai, không có văn bản nào được sử dụng để thay thế.

Cú pháp của một điều kiện phức tạp như sau -

|  |
| --- |
| conditional-directive  text-if-true  else  text-if-false  endif |

Nếu điều kiện là true, text-if-true được sử dụng; mặt khác, text-if-false được sử dụng. Text-if-false có thể là bất kỳ số dòng văn bản nào.

Cú pháp của lệnh điều kiện giống nhau cho dù điều kiện đơn giản hay phức tạp. Có bốn chỉ thị khác nhau kiểm tra các điều kiện khác nhau. Chúng như đã cho -

|  |
| --- |
| ifeq (arg1, arg2)  ifeq 'arg1' 'arg2'  ifeq "arg1" "arg2"  ifeq "arg1" 'arg2'  ifeq 'arg1' "arg2" |

Các chỉ thị ngược lại của các điều kiện trên như sau -

|  |
| --- |
| ifneq (arg1, arg2)  ifneq 'arg1' 'arg2'  ifneq "arg1" "arg2"  ifneq "arg1" 'arg2'  ifneq 'arg1' "arg2" |

Ví dụ về Chỉ thị Điều kiện

|  |
| --- |
| libs\_for\_gcc = -lgnu  normal\_libs =  foo: $(objects)  ifeq ($(CC),gcc)  $(CC) -o foo $(objects) $(libs\_for\_gcc)  else  $(CC) -o foo $(objects) $(normal\_libs)  endif |

## Chỉ thị bao gồm

Chỉ **thị bao gồm cho** phép tạm dừng việc đọc tệp tạo tệp hiện tại và đọc một hoặc nhiều tệp tạo tệp khác trước khi tiếp tục. Chỉ thị là một dòng trong tệp thực hiện trông như sau -

|  |
| --- |
| include filenames... |

Tên tệp có thể chứa các mẫu tên tệp shell. Khoảng trắng thừa được phép và bỏ qua ở đầu dòng nhưng không được phép sử dụng tab. Ví dụ: nếu bạn có ba tệp `.mk', cụ thể là `a.mk', `b.mk' và `c.mk', và $(bar) thì tệp này sẽ mở rộng thành bish bash và sau đó là tệp sau sự biểu lộ.

|  |
| --- |
| include foo \*.mk $(bar)  is equivalent to:  include foo a.mk b.mk c.mk bish bash |

Khi **make** xử lý một lệnh include, nó tạm dừng việc đọc makefile và lần lượt đọc từ mỗi file được liệt kê. Khi điều đó kết thúc, **hãy** tiếp tục đọc tệp thực hiện trong đó chỉ thị xuất hiện.

## Chỉ thị ghi đè

Nếu một biến đã được đặt với một đối số lệnh, thì các phép gán thông thường trong tệp thực hiện sẽ bị bỏ qua. Nếu bạn muốn đặt biến trong tệp tạo tệp ngay cả khi nó được đặt bằng đối số lệnh, bạn có thể sử dụng lệnh ghi đè, là một dòng trông như sau−

|  |
| --- |
| override variable = value  or  override variable := value |

# Makefile - Biên dịch lại

Chương trình **tạo** là một tiện ích thông minh và hoạt động dựa trên những thay đổi bạn thực hiện trong các tệp nguồn của mình. Nếu bạn có bốn tệp main.cpp, hello.cpp, factorial.cpp và functions.h, thì tất cả các tệp còn lại phụ thuộc vào hàm.h và main.cpp phụ thuộc vào cả hello.cpp và factorial.cpp. Do đó, nếu bạn thực hiện bất kỳ thay đổi nào trong hàm.h, thì quá trình **tạo sẽ** biên dịch lại tất cả các tệp nguồn để tạo các tệp đối tượng mới. Tuy nhiên, nếu bạn thực hiện bất kỳ thay đổi nào trong main.cpp, vì điều này không phụ thuộc vào bất kỳ tệp nào khác, thì chỉ tệp main.cpp được biên dịch lại, còn help.cpp và factorial.cpp thì không.

Trong khi biên dịch một tệp, lệnh **thực hiện** kiểm tra tệp đối tượng của nó và so sánh các dấu thời gian. Nếu tệp nguồn có dấu thời gian mới hơn tệp đối tượng, thì nó sẽ tạo tệp đối tượng mới với giả định rằng tệp nguồn đã được thay đổi.

## Tránh biên dịch lại

Có thể có một dự án bao gồm hàng nghìn tệp. Đôi khi bạn có thể đã thay đổi một tệp nguồn nhưng bạn có thể không muốn biên dịch lại tất cả các tệp phụ thuộc vào nó. Ví dụ: giả sử bạn thêm macro hoặc khai báo vào tệp tiêu đề mà các tệp khác phụ thuộc vào đó. Bảo thủ, **hãy** giả định rằng bất kỳ thay đổi nào trong tệp tiêu đề đều yêu cầu biên dịch lại tất cả các tệp phụ thuộc, nhưng bạn biết rằng chúng không cần biên dịch lại và bạn không muốn lãng phí thời gian chờ chúng biên dịch.

Nếu bạn biết trước vấn đề trước khi thay đổi tệp tiêu đề, bạn có thể sử dụng cờ `-t'. Cờ này yêu cầu **make** không chạy các lệnh trong quy tắc mà thay vào đó đánh dấu mục tiêu được cập nhật bằng cách thay đổi ngày sửa đổi cuối cùng của nó. Bạn cần làm theo quy trình này -

* Sử dụng lệnh `make' để biên dịch lại các tập tin nguồn thực sự cần biên dịch lại.
* Thực hiện các thay đổi trong tệp tiêu đề.
* Sử dụng lệnh `make -t' để đánh dấu tất cả các tệp đối tượng là đã cập nhật. Lần sau khi bạn chạy make, những thay đổi trong tệp tiêu đề không gây ra bất kỳ sự biên dịch lại nào.

Nếu bạn đã thay đổi tệp tiêu đề vào thời điểm khi một số tệp cần biên dịch lại thì đã quá muộn để thực hiện việc này. Thay vào đó, bạn có thể sử dụng cờ `-o file', nó đánh dấu một file cụ thể là "cũ". Điều này có nghĩa là bản thân tệp sẽ không được làm lại và không có gì khác sẽ được làm lại trên tài khoản của nó. bạn cần làm theo quy trình này -

* Biên dịch lại các tệp nguồn cần biên dịch vì những lý do độc lập với tệp tiêu đề cụ thể, với `make -o tệp tiêu đề'. Nếu có liên quan đến nhiều tệp tiêu đề, hãy sử dụng một tùy chọn `-o' riêng biệt cho mỗi tệp tiêu đề.
* Cập nhật tất cả các tệp đối tượng bằng `make -t'.

# Makefile - Các tính năng khác

Trong chương này, chúng ta sẽ xem xét nhiều tính năng khác của Makefile.

## Sử dụng đệ quy Make

Việc sử dụng đệ quy **make** có nghĩa là sử dụng **make** như một lệnh trong tệp thực hiện. Kỹ thuật này rất hữu ích khi bạn muốn các tệp thực hiện riêng biệt cho các hệ thống con khác nhau tạo thành một hệ thống lớn hơn. Ví dụ: giả sử bạn có một thư mục con tên là `thư mục con' có tệp tạo tệp riêng của nó và bạn muốn tệp tạo tệp tạo của thư mục chứa chạy tệp **tạo** trên thư mục con. Bạn có thể làm điều đó bằng cách viết đoạn mã dưới đây -

|  |
| --- |
| subsystem:  cd subdir && $(MAKE)  or, equivalently:    subsystem:  $(MAKE) -C subdir |

**Bạn có thể viết các lệnh tạo** đệ quy chỉ bằng cách sao chép ví dụ này. Tuy nhiên, bạn cần biết về cách chúng hoạt động và lý do cũng như cách sản phẩm phụ liên quan đến sản phẩm cấp cao nhất.

## Giao tiếp các biến với Sub-Make

Các giá trị biến của tạo cấp cao nhất **có** thể được chuyển đến tạo phụ thông qua môi trường theo yêu cầu rõ ràng. Các biến này được định nghĩa trong phần tạo phụ dưới dạng mặc định. Bạn không thể ghi đè những gì được chỉ định trong tệp tạo tệp được sử dụng bởi tệp tạo tệp tạo phụ trừ khi bạn sử dụng khóa chuyển `-e'.

Để truyền xuống hoặc xuất một biến, **hãy** thêm biến đó và giá trị của nó vào môi trường để chạy từng lệnh. Đổi lại, sản phẩm phụ sử dụng môi trường để khởi tạo bảng giá trị biến của nó.

Các biến đặc biệt SHELL và MAKEFLAGS luôn được xuất (trừ khi bạn hủy xuất chúng). MAKEFILES được xuất nếu bạn đặt thành bất kỳ thứ gì.

Nếu bạn muốn xuất các biến cụ thể sang một sản phẩm phụ, hãy sử dụng lệnh xuất, như minh họa bên dưới -

|  |
| --- |
| export variable ... |

Nếu bạn muốn ngăn việc xuất một biến, hãy sử dụng lệnh unexport, như minh họa bên dưới –

|  |
| --- |
| unexport variable ... |

## Biến MAKEFILES

Nếu biến môi trường MAKEFILES được xác định, **hãy** coi giá trị của nó là danh sách tên (được phân tách bằng khoảng trắng) của các tệp tạo tệp bổ sung sẽ được đọc trước các tệp tạo tệp khác. Điều này hoạt động giống như chỉ thị bao gồm: các thư mục khác nhau được tìm kiếm cho các tệp đó.

Công dụng chính của MAKEFILES là giao tiếp giữa các lời gọi đệ quy của **make** .

## Bao gồm tệp Tiêu đề từ các Thư mục khác nhau

Nếu bạn đã đặt các tệp tiêu đề trong các thư mục khác nhau và bạn đang chạy **make** trong một thư mục khác, thì bạn phải cung cấp đường dẫn của các tệp tiêu đề. Điều này có thể được thực hiện bằng cách sử dụng tùy chọn -I trong makefile. Giả sử rằng tệp tin functions.h có sẵn trong thư mục /home/tutorialspoint/header và các tệp còn lại có sẵn trong thư mục /home/tutorialspoint/src/, thì makefile sẽ được viết như sau:

|  |
| --- |
| INCLUDES = -I "/home/tutorialspoint/header"  CC = gcc  LIBS = -lm  CFLAGS = -g -Wall  OBJ = main.o factorial.o hello.o  hello: ${OBJ}  ${CC} ${CFLAGS} ${INCLUDES} -o $@ ${OBJS} ${LIBS}  .cpp.o:  ${CC} ${CFLAGS} ${INCLUDES} -c $< |

## Nối thêm văn bản vào biến

Thông thường, việc thêm nhiều văn bản hơn vào giá trị của một biến đã được xác định sẽ rất hữu ích. Bạn làm điều này với một dòng chứa `+=', như được hiển thị -

|  |
| --- |
| objects += another.o |

Nó lấy giá trị của các đối tượng biến, và thêm văn bản `another.o' vào đó, trước một khoảng trắng như minh họa bên dưới.

|  |
| --- |
| objects = main.o hello.o factorial.o |

|  |
| --- |
| objects += another.o |

Đoạn mã trên đặt các đối tượng thành `main.o hello.o giai thừa.o other.o'.

Sử dụng `+=' tương tự như:

|  |
| --- |
| objects = main.o hello.o factorial.o  objects := $(objects) another.o |

## Dòng tiếp tục trong Makefile

Nếu bạn không thích các dòng quá lớn trong Makefile của mình, thì bạn có thể ngắt dòng của mình bằng dấu gạch chéo ngược "\" như minh họa bên dưới -

|  |
| --- |
| OBJ = main.o factorial.o \  hello.o  is equivalent to  OBJ = main.o factorial.o hello.o |

## Chạy Makefile từ Command Prompt

Nếu bạn đã chuẩn bị Makefile với tên "Makefile", thì chỉ cần viết make tại dấu nhắc lệnh và nó sẽ chạy tệp Makefile. Nhưng nếu bạn đã đặt bất kỳ tên nào khác cho Makefile, thì hãy sử dụng lệnh sau -

|  |
| --- |
| make -f your-makefile-name |

# Makefile - Ví dụ

Đây là một ví dụ về Makefile để biên dịch chương trình hello. Chương trình này bao gồm ba tệp *main.cpp* , *factorial.cpp* và *hello.cpp* .

|  |
| --- |
| # Define required macros here  SHELL = /bin/sh  OBJS = main.o factorial.o hello.o  CFLAG = -Wall -g  CC = gcc  INCLUDE =  LIBS = -lm  hello:${OBJ}  ${CC} ${CFLAGS} ${INCLUDES} -o $@ ${OBJS} ${LIBS}  clean:  -rm -f \*.o core \*.core  .cpp.o:  ${CC} ${CFLAGS} ${INCLUDES} -c $< |

Bây giờ bạn có thể xây dựng chương trình **hello** bằng cách sử dụng **make** . Nếu bạn đưa ra một lệnh **làm sạch** thì nó sẽ xóa tất cả các tệp đối tượng và tệp lõi có sẵn trong thư mục hiện tại.