第二章　UNIX的软件工具

- 2.1 Unix/Linux软件开发工具简介
- 2.2 BACKUS系统
- 2.3 vi
- 2.4 sed
- 2.5 awk
- 2.6 make
  - 2.6.1 make的工作原理
  - 2.6.2 make的功能
  - 2.6.3 进一步使用make
  - 2.6.4 make描述文件应用示例

➢2.6 Make

➢ 2.6.1 make的工作原理

make是UNIX/LINUX的软件开发维护程序。

➢ 软件编程过程

软件编程是一个不断循环的过程：

思考...修改...生成...调试运行

- 修改：编辑源文件，例如

  $ vi main.c

- 生成：编译生成目标文件和可执行文件，例如

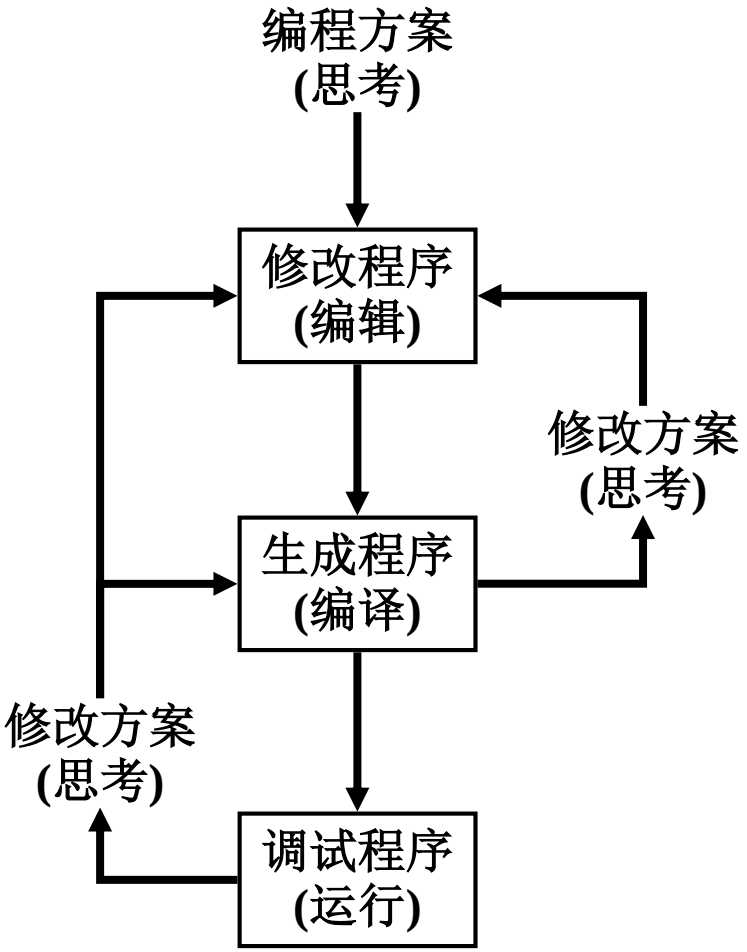
  $cat left_top left_bottom > left.c

  $cc -c main.c

  $cc -o try main.o left.o right.o

- 调试运行：例如

  $ try



➢2.6.1 make的工作原理

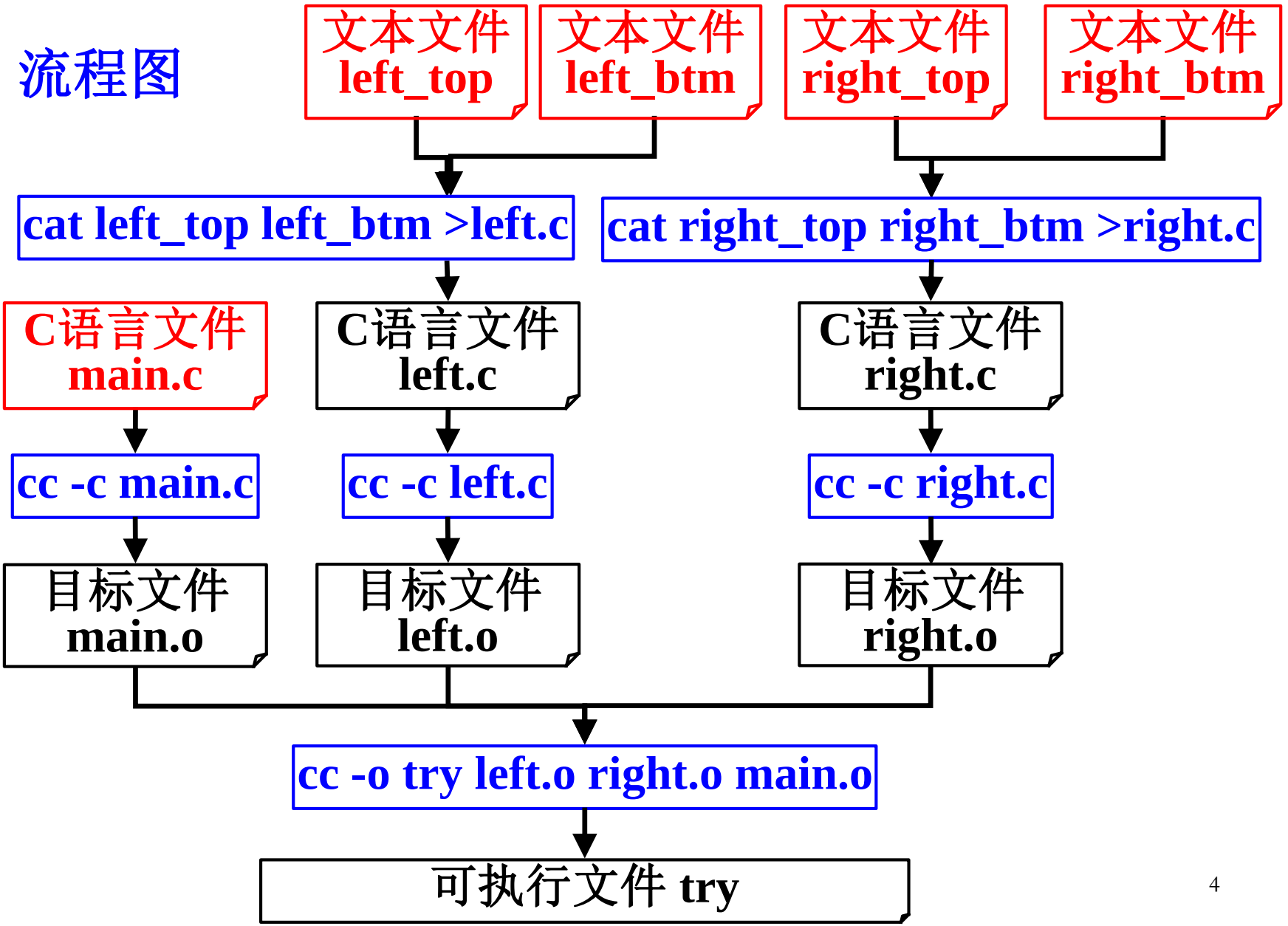
软件编程过程示例

•考虑设计5个源文件：main.c, left_top, left_btm, right_top和right_btm, 并由此生成一个可执行文件try，源文件内容分别为：

<table>
<tr>
<td>源文件main.c:
#include <stdio.h>
main() {
  right();
  left();
}</td>
<td>源文件left_top:
#include <stdio.h>
left() {
源文件left_btm:
  printf(“left\n”);
}</td>
<td>源文件right_top:
  #include <stdio.h>
  right() {
源文件right_btm:
  printf(“right\n”);
}</td>
</tr>
</table>

➢2.6.1 make的工作原理

流程图



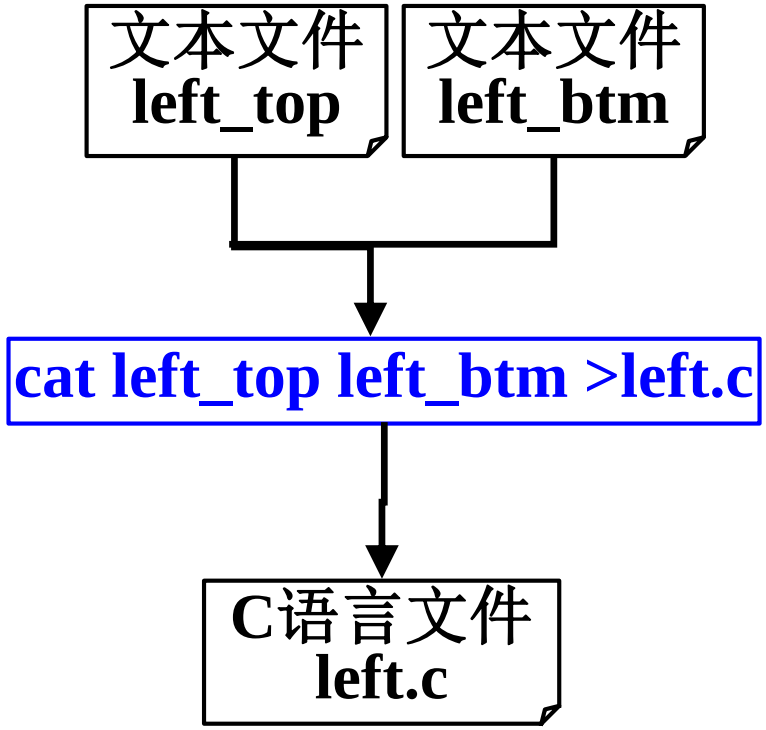
➢2.6.1 make的工作原理

软件编程过程示例

- 生成关系和依赖关系

例如，**left_top**和**left_btm**与**left.c**具有生成和依赖的相互关系，由**left_top**和**left_btm**的合并将生成**left.c**，称**left.c**依赖于**left_top**和**left_btm**。也就是说，有了**left_top**和**left_btm**，才能生成**left.c**，而如果**left_top**和**left_btm**发生变动，必须重新生成**left.c**。

同理，**left.c**与**left.o**具有生成和依赖的相互关系，简称依赖关系



➢2.6.1 make的工作原理

软件编程过程示例

•依赖关系的描述

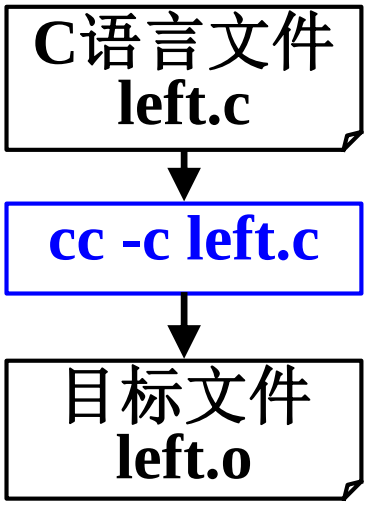
以描述left.c与left.o的依赖关系为例，可表达为：

if (left.c <NT> left.o)　　<NT>表示newer-than

cc -c left.c　　生成left.c所执行的命令

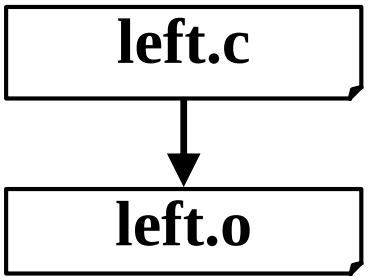
其中，left.c <NT> left.o的含义包括：

- left.c的生成时间比left.o的生成时间新，
- 或者left.c存在而left.o不存在



流程图

如果省略流程图中的操作命令，可以仅用箭头表示文件的生成关系，显然也同样表达了依赖关系。



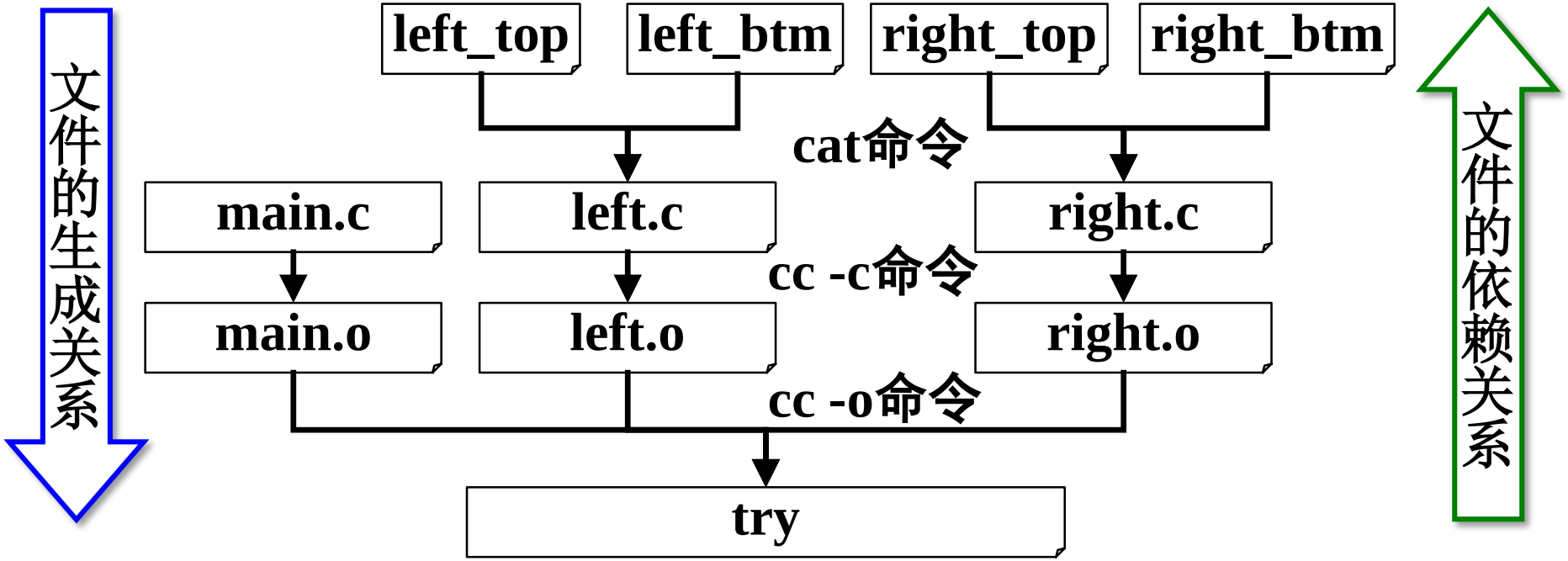
依赖关系

➢2.6.1 make的工作原理

软件编程过程示例

- 递推原则

依赖关系符合递推原则。例如，由于left.o依赖于left.c，而left.c依赖于left_top和left_btm，因此left.o也依赖于left_top和left_btm，依此类推。从而可以完整描述有关try的依赖关系。



➢2.6.2 make应用基础

➢启动make

在make中，需要建立一个描述文件，用于表述所有的依赖关系。描述文件通常取名为Makefile或者makefile，在Linux中还可以取名为GNUmakefile。

在shell状态下启动make的格式为：

make [-f *make_file*]

如果使用可选项-f，make将把*make_file*作为make描述文件。否则，make将按照Makefile，makefile以及GNUmakefile的顺序寻找make描述文件。

➢make在执行时，根据Makefile文件中对所有文件编译顺序的描述，判断需要更新哪些文件；同时对需要更新的文件进行重建。

➢2.6.2 make应用基础

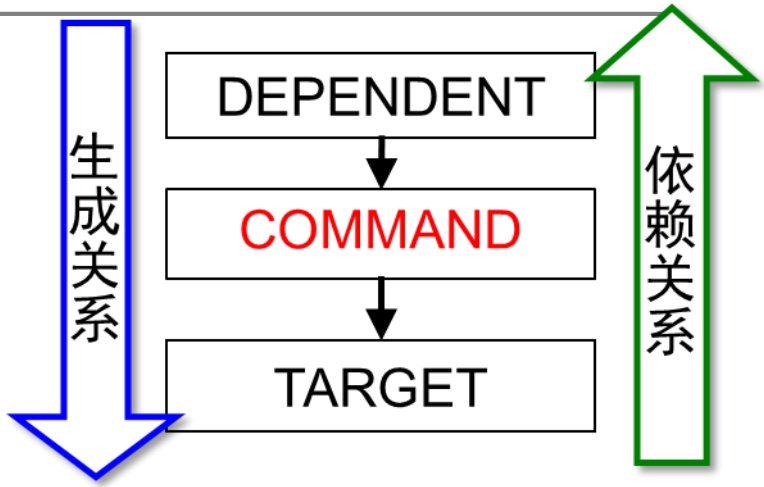
➢2.6.2.1 Makefile的书写规则

TARGET*: *DEPENDENT

<TAB> *COMMAND*

或者

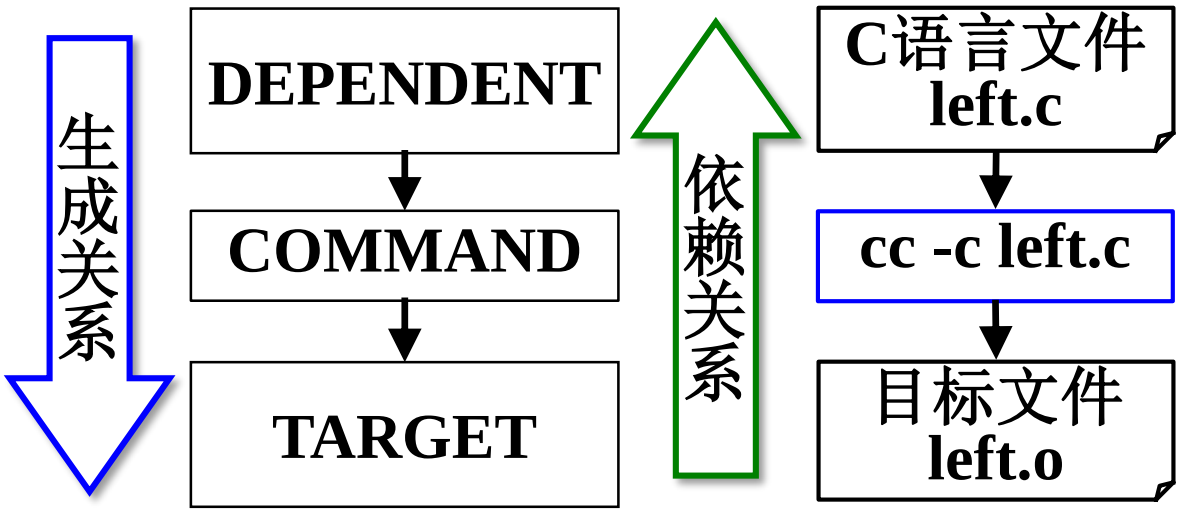
TARGET：DEPENDENT；COMMAND



- 规则描述了在何种情况下使用什么命令来重建一个特定的文件，此文件被称为规则“目标”（TARGET，通常规则中的目标只有一个）
- 规则中除“目标”外列出的文件DEPENDENT表示TARGET所依赖的文件，或称为依赖
- 规则的命令COMMAND表示生成TARGET所需要执行的命令
- 据make的语法规定，*COMMAND*一行的开头必须使用制表符<TAB>，而不能使用空格。

➢2.6.2 make应用基础

➢2.6.2.1 Makefile的书写规则



例如，用make的方式描述left.c与left.o的依赖关系为：

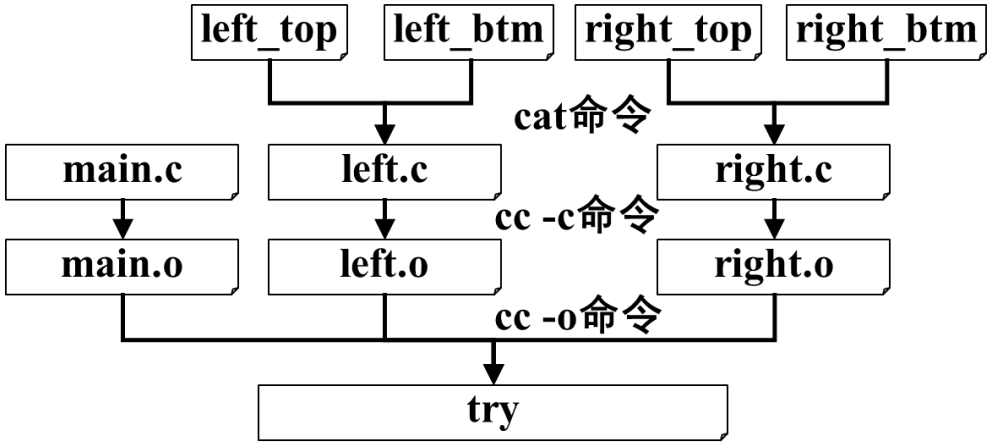
```
left.o:    left.c
<TAB>  cc -c left.c
```

➢2.6.2 make应用基础

➢Makefile示例

如果建立一个Makefile，并且描述try的所有依赖关系，内容为：

```
try:      left.o right.o main.o
<TAB> cc -o try left.o right.o main.o
main.o: main.c
<TAB> cc -c main.c
left.o:   left.c
<TAB> cc -c left.c
right.o: right.c
<TAB> cc -c right.c
left.c:   left_top left_btm
<TAB> cat left_top left_btm >left.c
right.c: right_top right_btm
<TAB> cat right_top right_btm >right.c
```



➢2.6.2 make应用基础

➢Makefile示例

- 首先解析“终极目标”（try）所在的规则（Makefile中的第一个规则），按照依赖文件列表从左到右的顺序寻找创建这些依赖文件的规则。
- 创建或者更新每一个规则依赖文件的过程都是按照依赖文件列表顺序，使用同样方式（按照同样的过程）去重建每一个依赖文件，在完成对所有依赖文件的重建之后，最后一步才是重建此规则的目标。

➢2.6.2 make应用基础

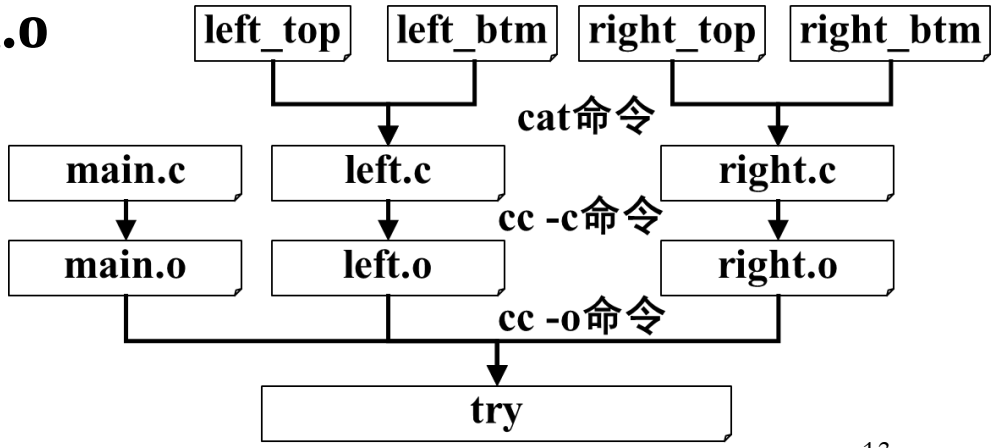
➢Makefile示例

如果建立描述文件Makefile后第一次执行，即：

```
$ make
```

将依次显示运行的命令如下：

```
cat left_top left_btm >left.c
cc -c left.c
cat right_top right_btm >right.c
cc -c right.c
cc -c main.c
cc -o try left.o right.o main.o
```



➢2.6.2 make应用基础

➢Makefile示例

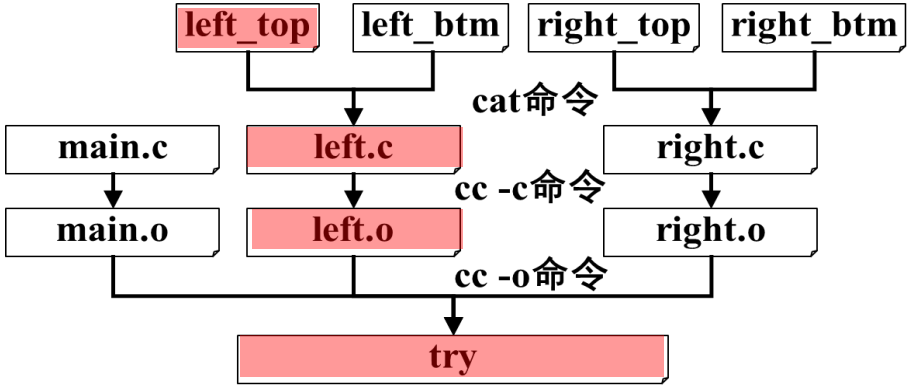
如果再次执行make（未对任何文件做修改）：

```
$ make
make: 'try' is up to date
```

如果修改了源文件left_top，然后再执行make命令：

（$ touch left_top ）

```
$ make
cat left_top left_btm >left.c
cc -c left.c
cc -o try left.o right.o main.o
```



make就会循着依赖关系，发现left_top是left.c的依赖文件，根据递推原则，执行所需的三条命令。

➢2.6.3 make应用进阶

➢依赖关系的描述

make提供以下几种依赖关系的描述方法：

***TARGET*: [*DEPENDENT* | ;]**

<TAB> [@]*COMMAND*

如果缺省***DEPENDENT***，表示***TARGET***没有依赖源，需要无条件执行命令***COMMAND***。在这种情况下，依赖关系的描述也可采用以下形式：

TARGET*:; [@]*COMMAND

如果在***COMMAND***的前面没有@，表示在执行命令时，首先将显示命令。如果加@，表示不显示命令。例如：

在命令前不加@：

```
greet:;  echo “hello!”
```

执行make的效果为：

```
$ make
echo “hello!”
hello!
```

在命令前加@：

```
greet:;  @ echo “hello!”
```

执行make的效果为：

```
$ make
hello!
```

➢2.6.3 make应用进阶

➢TARGET规则

·指定目标

通常在描述文件中存在多个TARGET，启动make时，将从第一个TARGET开始执行。例如修改了left_top，启动try的描述文件时，执行效果为：

```
$ make
cat left_top left_btm >left.c          生成left.c
cc -c left.c                           生成left.o
cc -o try left.o right.o main.o        生成try
```

也可以指定某个TARGET，例如：

```
$ make left.c                          指定TARGET为left.c
cat left_top left_btm >left.c          生成left.c
```

由于以left.c为TARGET的依赖关系为：

```
left.c:    left_top left_btm
<TAB> cat left_top left_btm >left.c
```

因此只执行一个cat命令。

➢2.6.3 make应用进阶

➢TARGET规则

·伪目标

以下内容是在描述文件中常见的：

```
.....
clean:;      rm *.o
backup:;  cp *.cpp backup/
```

其中，clean和backup没有倚赖关系，表示需要无条件执行的操作。由于clean和backup不是文件，称为伪目标。如果需要执行伪目标，应该使用make *TARGET*的形式，例如：

```
$ make clean  或者      $ make backup
```

➢2.6.3 make应用进阶

➢其他书写技巧

•注释

在描述文件中，可以加注释，并且用#作为前导。例如：

```
#greeting
greet:;  echo “hello!”
```

或者

```
greet:    #greeting
<TAB>  echo “hello!”
```

•续行

可以使用反斜杠“\”作为续行符。将一个较长行来分解为多行。注意：反斜线之后不能有空格。

例如：

```
clean:;  rm main.o left.o right.o left.c right.c
```

可以改写为

```
clean:;  rm main.o left.o \<CR>
<TAB>  right.o left.c right.c
```

在“\”和换行符<CR>之间不能有空格。

➢2.6.3 make应用进阶

➢宏定义

make的宏定义又称为make的变量，简称为宏。

宏的引用：$(宏名)　或者　${宏名}

•宏的应用示例

在try的描述文件中，开头的两行为：

```
try:      left.o right.o main.o
<TAB> cc -o try left.o right.o main.o
```

如果使用宏定义，可以改写为：

```
TASK = try
OBJ=left.o right.o main.o
$(TASK):  $(OBJ)
<TAB>     cc -o $(TASK) $(OBJ)
```

类似C语言中我们要求尽量使用宏定义来表示常数，在make中我们也要求尽量使用宏定义来表示文件，使得描述文件的表述简明、清晰和易于理解，同时也易于修改和扩充软件的框架结构。

➢2.6.3 make应用进阶

➢特殊变量

TARGET表示依赖关系中的目标文件名，可以用特殊变量$@来替代。例如，以下描述：

```
try:      left.o right.o main.o
<TAB> cc -o try left.o right.o main.o
```

可以改写为：

```
try:      left.o right.o main.o
<TAB> cc -o $@ left.o right.o main.o
```

特殊变量$^表示规则中使用空格分隔的所有依赖文件列表。例如：

可以进一步改写为：

```
try:    left.o right.o main.o
<TAB> cc -o $@ $^
```

➢2.6.3 make应用进阶

➢特殊变量

$<：表示规则的第一个依赖文件名。例如：

```
right.o: right.c abc.h
<TAB> cc -c right.c
```

可以改写为：

```
right.o: right.c abc.h
<TAB> cc -c $<
```

➢2.6.3 make应用进阶

➢传递规则

在大多数依赖关系中，根据依赖源生成目标文件是有规律的。例如，C语言目标文件(.o文件)是由源文件(.c文件)生成的。例如，在try的描述文件中，从.c文件生成.o文件的依赖关系描述为：

```
main.o: main.c
<TAB> cc -c main.c
left.o:   left.c
<TAB> cc -c left.c
right.o: right.c
<TAB> cc -c right.c
```

因此在make中将其取为缺省的传递规则。传递规则可以采用后缀传递规则(后缀规则)描述，在GNU make中，还可以采用模式传递规则(模式规则)。

➢2.6.3 make应用进阶

➢传递规则

•后缀规则

使用后缀规则，以上内容可以改写为：

```
.c.o:
<TAB> cc -c $<
```

表示.c和.o分别是依赖源文件和目标文件的后缀。$<表示传递规则中的依赖源文件，或者称为传递规则的前者。

在try的makefile中，“cc -c $<”成为left.o，right.o和main.o的编译命令“cc -c main.c”，“cc -c left.c”和“cc -c right.c”的统一表述。

如果make发现某个.o文件需要重新编译，例如main.c的修改时间(mtime)比main.o新，则将根据传递规则，用main.c替换$<，从而执行命令“cc -c main.c”生成main.o。

➢2.6.3 make应用进阶

➢传递规则

·后缀规则

$*也是make的特殊变量，表示依赖源文件和目标文件的共同前缀。因此更新main.o时，将用main替换$*，从而用main.c替换$*.c，然后执行命令“cc -c main.c”生成main.o。因此，以上内容也可以改写为：

```
.c.o:
<TAB> cc -c $*.c
```

➢2.6.3 make应用进阶

➢传递规则

•后缀规则

如果希望将编译产生的出错信息保留到后缀为.err的文件中，可以进一步改写以上的依赖关系：

```
.c.o:
<TAB> cc -c $<  >$*.err
```

或者

```
.c.o:
<TAB> cc -c $*.c >$*.err
```

•模式规则

在GNU make中，模式规则的描述为：

```
%.o:%.c
<TAB> cc -c $<
```

或者

```
%.o:%.c; cc –c $<
```

➢2.6.3 make应用进阶

➢控制结构

make提供了类似shell的控制结构，如if，for，while等。

以for结构为例，例如一个B shell程序文件的内容为：

```
for dir in d1 d2 d3          d1，d2和d3是三个目录
do
   cd $dir                   进入某个目录
   make -f ${dir}_make       执行某个描述文件，如d1_make
   cd ..                     返回父目录
done
```

➢2.6.3 make应用进阶

➢控制结构(续）

则可以在make的描述文件中，使用类似的for结构：

```
SUBDIR = d1 d2 d3
build :
<TAB> @for dir in $(SUBDIR); do \          用反斜杠表示续行
        cd $${dir}; make -f $${dir}_make; \  用$$引用变量
        cd ..; \                             不显示cd命令
        done
```

其中，在引用for结构中的索引变量时，也需要在$符号之前再加一个$。例如必须用$${dir}来引用dir，而不能简单地使用${dir}。

在make中，for结构为一个语句组单位，需要用分号代替shell的回车符。如果一个控制结构的描述超过一行，则需要在行尾用反斜杠表示续行。

➢2.6.4 make描述文件应用示例

➢try的描述文件

应用以上make的各种功能，可以改写try的描述文件Makefile，内容为：

```
# Function: description file to make “try”
# Usage: $ make
TASK   =    try
R_TXT = right_top  right_btm
L_TXT = left_top  left_btm
LS =        left.c
RS =        right.c
OBJ =       try.o  left.o  right.o
$(TASK): $(OBJ)
<TAB>    cc -o  $@  $(OBJ)
.c.o:
<TAB>    cc -c $<
$(LS):      $(L_TXT)
<TAB>    cat $(L_TXT) >$(LS)
$(RS):      $(R_TXT)
<TAB>    cat $(R_TXT) >$(RS)
```

在windows下完成C语言程序的编写：VC

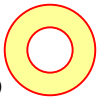
在linux下完成C语言程序的编写

- 文本编辑器：vi
- C语言编译工具：gcc

  $ gcc -c main.c　　“编译”生成main.o

  $ gcc -o try main.o　　“链接”
- 使用make维护程序的编译及链接，例如编写Makefile内容如下：

```
TARGET = linelist
SRC = ListMain.c ListSub.c
OBJ = ListMain.o ListSub.o
$(TARGET) : $(OBJ)
        gcc -o $@ $(OBJ)
.c.o:
        gcc -c $<
clean:
        rm $(OBJ)
```



➢作　业

- 习题　　2.11

- 5月18日校运会停课一次，5月18日及5月24日相应停上机课
- 期中考试暂定5月27日（周日）上午10:00-11:30，考试教室待定、下堂课通知。