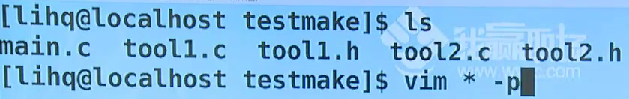
makefile

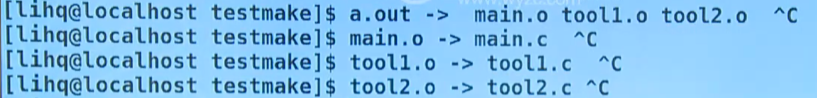
make指的是工程管理器，用来帮你管理工程，最主要是帮你分析工程当中的依赖与被依赖关系，它所执行的一个脚本可以这样来理解，就是makefile，程序员发布出去的源码包都是大写的M开头的makefile，程序员写的makefile只是告诉你，我当前工程编译出来大概是怎么样的编译过程，会把编译过程和依赖关系在里面声明好，如果你认为当前这个组件的编译过程需要做变动，你可以自己写一个小写的m开头的makefile，那它执行的时候优先调的是小写的makefile





用Ctrl+PgUp/PgDn在各文件间进行切换，写完一些简单的操作后（好像可以用:wqa来全体退出），用gcc \*.c进行编译生成a.out文件

若成功编译，则以下就是它们的依赖关系



其实这些关系就是要写进makefile的内容



首先把总的目标放进来，在这里，我总的目标是mytool（就是那个a.out，改个名），依赖于main.o, tool1.o, tool2.o

然后下面一句，tab之后就是具体的命令，gcc main.o tool1.o tool2.o -o mytool，这是我最终的那句话，而现在其实我们是没有main.o的

继续写呗，main.o就作为下一次的目标，main.o依赖的文件是main.c

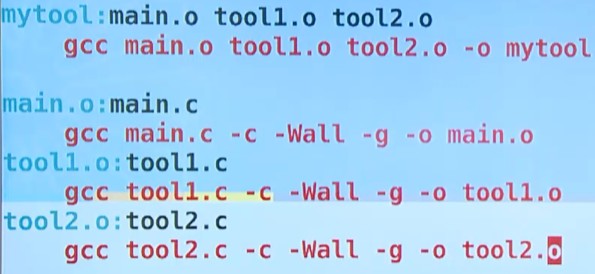
下面一句，tab之后，gcc main.c -c -o main.o，这里你可以加上一些调试，比如我想看所有的警告，我还想加上-g选项来方便以后调试，gcc main.c -c -Wall -g -o main.o

继续，tool1.o要依赖于tool1.c

下面，tab之后，gcc tool1.c -c -Wall -g -o tool1.o

继续，同样的道理，tool2.o就依赖于tool2.c

下面，tab之后，gcc tool2.c -c -Wall -g -o tool2.o



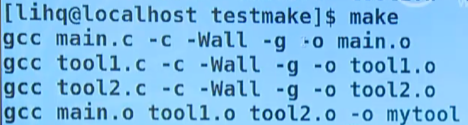
这就是一个最基本的makefile了，来看怎么执行

这是所有的源文件



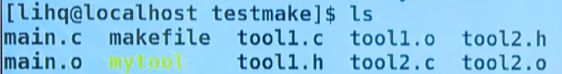
.h实际上从来不参与我们的编译，都是include进去的

在这里我们执行make，它就会执行我们makefile中指定的内容，因为我们makefile当中没有别的语法标记，所有它默认执行最上面的一个target

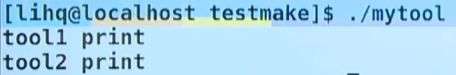


相当于从叶子到根的一棵树，先去产生各个.o，然后再产生我们的可执行文件

ls看一下，有很多中间过程的文件，.c，.h，.o都在这



然后./mytool，就可以看到我们的执行效果



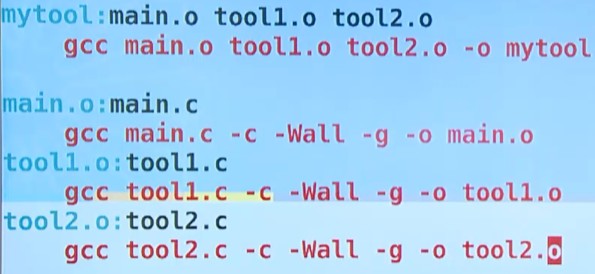
当然，我如果再去make，它就会告诉我，mytool是最新的



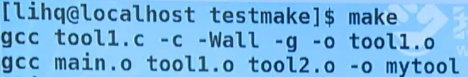
怎么会这么说，我们make中的依赖与被依赖写在这，比如



当前mytool文件是否被重新生成取决于，当前的某一个文件的时间戳怎么样，比如说，如果mytool的时间戳跟main.o tool1.o tool2.o这些文件比起来都是最新的，那么mytool一定是最新的，但如果mytool和它们几个一比，发现这里面有一个或多个文件的时间戳比mytool要新，说明mytool是旧的，那就要重新再更改一下

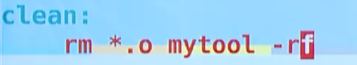


对着makefile看，如果我们改变了tool1.c，那就牵涉到tool1.o要变，而tool1.o一变，那就牵涉到mytool要变，而其中main.o和tool2.o实际上是没有发生变化的，所以它们不用重新编译，所以make这个工程管理器是会检查某一个文件和它所依赖的文件的时间戳的关系来确定要不要重新进行编译，这个时候我们再去make看看结果



由于tool1.c变了，就会导致出现新的tool1.o，而tool1.o变了就会导致出现新的mytool，其中main.o和tool2.o是再也没有被重新去实现的

如果makefile里只是这样的话，那么再次make时将看不到过程了，所以我们习惯性在makefile文件底下加上clean

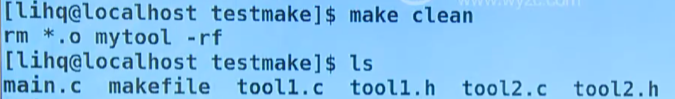


再重新make



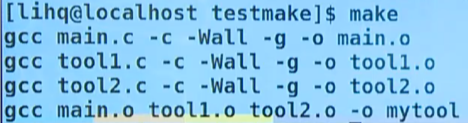
看到mytool是最新的，看不到过程

再make clean

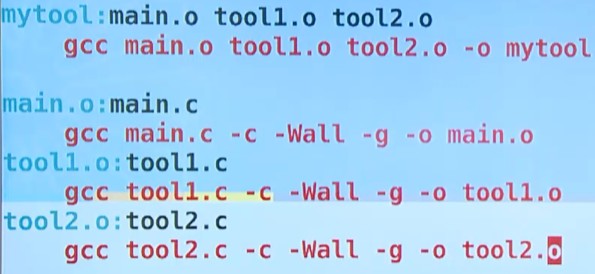


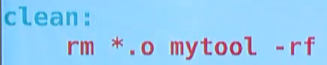
可以看到.o都被删除了，没有了

我再重新make



就能看到过程了，后面我还想看过程，就可以先执行make clean，再make就可以了



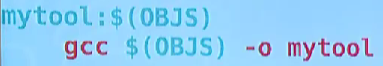


以上是makefile的第一个版本，这个makefile还是太繁琐了，可以精简

精简makefile，用变量替换，发现main.o tool1.o tool2.o重复度较高，所以我定义一个变量叫OBJS，指的是对象，对象是main.o tool1.o tool2.o



那下面完全不用直接写了，可以用变量值代替，变量值是$(OBJS)



还有一个什么可以改，gcc实际上，当前makefile当中会有一个编译器叫CC，实际上它的默认值就是gcc，你不写也行，当然你也可以假装没有默认值，重复一下

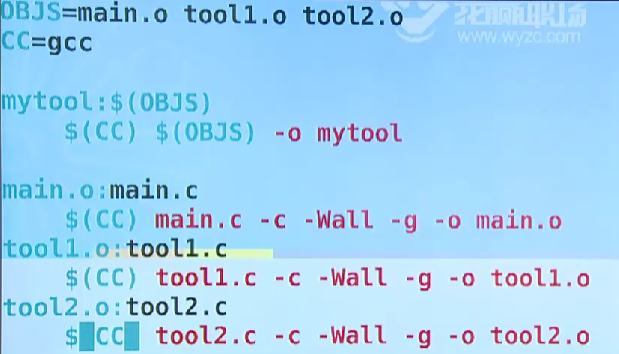


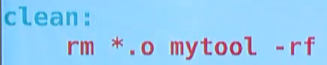
那底下的gcc就可以用$(CC)来代替，实际上就相当于gcc



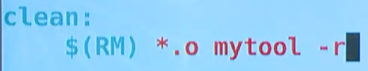
写到这，你应该反应过来，当前的make不止可以帮我们管理c程序，只要我们安装了特定的编译器，它就可以帮我们管理各种程序，比如说，当我们CC的值是g++的时候，那我们写C++的工程，它就可以帮我们编译了

底下都改了，makefile变成这样





clean这块也可以改



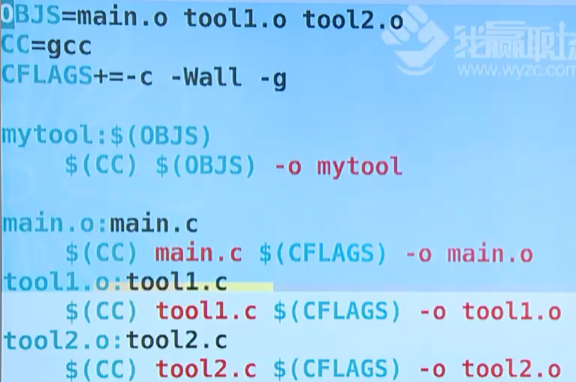
这个RM相当于是rm -f

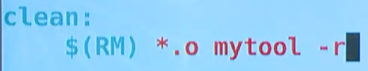


上面那就就是第二个版本了，目前来说够基本使用，如果你还需要编译选项和链接选项，你可以继续定义变量，比如说-c -Wall -g都是属于编译选项，这些内容我们就可以来一个CFLAGS+=，+=的意思就是在原来的基础上再加上一些功能，-c是我需要进行编译，-Wall是我要查看所有的警告，-g是我要加调试选项



底下的都改了，makefile变成这样





继续简化，在当前某一行，比如mytool那一行的实现上



后面的内容$(OBJS)，在下一行引用的时候，我们其实可以用一个符号来代替，那个符号叫$^，代表的意思是在上一句依赖关系中，被依赖的所有文件$(OBJS)

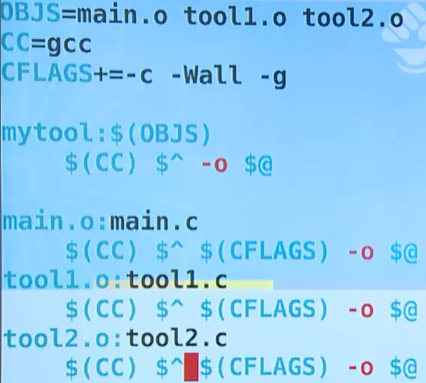


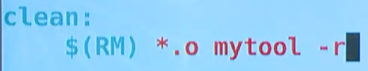
-o后面跟的是目标名，每一句里面的目标，实际上都是相对来讲的，它可以用一个$@符号来代表



这句话的意思就是，gcc把上述要依赖的文件来进行-o，生成上一句当中的目标文件

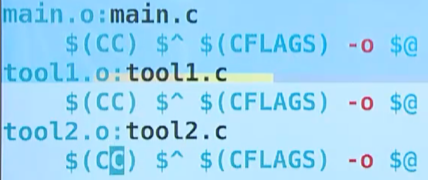
依葫芦画瓢，下面都改了，makefile变成





当然，这还不是最简单的，还能再简单

你会发现这几句特别像



我们可以提取出一个公式，就是我想要生成一个某种名字的.o，那它就依赖于某个名字的.c



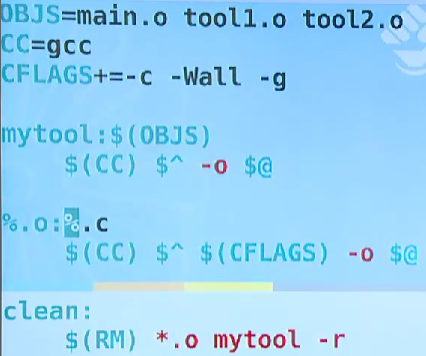
底下这句话都是一样的



我就可以利用gcc把上述依赖的这些文件，通过编译选项CFLAGS，-o生成目标文件，底下就可以都删了，上面当然不能写XX，要写%，%相当于通配符，在同一句话当中，%代表的是同一个名字，所以应该写成



最后的makefile就是



整体的意思是，你想要生成mytool这样一个文件，它依赖于main.o tool1.o tool2.o这堆.o，而这堆.o现在没有，每一个.o都依赖于和它相同名字的.c，然后利用



这样一个command给它运行出来

对makefile最基本的理解，其实就是一个工程管理器的使用，把当前工程（mytool）作为最终极的一个目标，一层一层剥洋葱似的，或者像嵌套调用一样，一层一层地把它反映出来