Linux操作系统中GCC的应用介绍 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/181/2021_2022_Linux_E6_93_8D_E4_BD_c103_181681.htm 在为Linux开发应用程序时，绝大多数情况下使用的都是C语言，因此几乎每一位Linux 程序员面临的首要问题都是如何灵活运用C编译器。目前Linux 下最常用的C语言编译器是GCC（GNU Compiler Collection），它是GNU项目中符合ANSI C标准的编译系统，能够编译用C、C 和Object C等语言编写的程序。GCC不仅功能非常强大，结构也异常灵活。最值得称道的一点就是它可以通过不同的前端模块来支持各种语言，如Java、 Fortran、Pascal、Modula-3和Ada等。 开放、自由和灵活是Linux的魅力所在，而这一点在GCC上的体现就是程序员通过它能够更好地控制整个编译过程。在使用GCC编译程序时，编译过程可以被细分为四个阶段： ◆ 预处理（Pre-Processing） ◆ 编译（Compiling） ◆ 汇编（Assembling） ◆ 链接（Linking） Linux程序员可以根据自己的需要让 GCC在编译的任何阶段结束，以便检查或使用编译器在该阶段的输出信息，或者对最后生成的二进制文件进行控制，以便通过加入不同数量和种类的调试代码来为 今后的调试做好准备。和其它常用的编译器一样，GCC也提供了灵活而强大的代码优化功能，利用它可以生成执行效率更高的代码。 GCC提供了30多条警告信息和三个警告级别，使用它们有助于增强程序的稳定性和可移植性。此外，GCC还对标准的C和C 语言进行了大量的扩展，提高程序的执行效率，有助于编译器进行代码优化，能够减轻编程的工作量。 GCC起步 在学习使用GCC之前，下面

的这个例子能够帮助用户迅速理解GCC的工作原理，并将其立即运用到实际的项目开发中去。首先用熟悉的编辑器输入清单1所示的代码： 清单1：hello.c #include int main(void){printf ("Hello world, Linux programming!\\n").return 0.}然后执行下面的命令编译和运行这段程序： # gcc hello.c -o hello# ./helloHello world, Linux programming!从程序员的角度看，只需简单地执行一条GCC命令就可以了，但从编译器的角度来看，却需要完 成一系列非常繁杂的工作。首先，GCC需要调用预处理程序 cpp，由它负责展开在源文件中定义的宏，并向其中插入“ #include ”语句所包含的内容；接着，GCC会调用ccl和as将处理后的源代码编译成目 标代码；最后，GCC会调用链接程序ld，把生成的目标代码链接成一个可执行程序。 为了更好地理解GCC的工作过程，可以把上述编译过程分成几个步骤单独进行，并观察每步的运行结果。第一步是进行预编译，使用-E参数可以让GCC在预处理结束后停止编译过程： # gcc -E hello.c -o hello.i此时若查看hello.cpp文件中的内容，会发现stdio.h的内容确实都插到文件里去了，而其它应当被预处理的宏定义也都做了相应的处理。下一步是将hello.i编译为目标代码，这可以通过使用-c参数来完成： # gcc -c hello.i -o hello.oGCC默认将.i文件看成是预处理后的C语言源代码，因此上述命令将自动跳过预处理步骤而开始执行编译过程，也可以使用-x参数让GCC从指定的步骤开始编译。最后一步是将生成的目标文件链接成可执行文件： # gcc hello.o -o hello在采用模块化的设计思想进行软件开发时，通常整个程序是由多个源文件组成的，相应地也就形成 了多个编译单元，使用GCC能够很好地管理这些编译单元。假设有

一个由foo1.c和foo2.c两个源文件组成的程序，为了对它们进行编译，并最终生成 可执行程序foo，可以使用下面这条命令：# gcc foo1.c foo2.c -o foo如果同时处理的文件不止一个，GCC仍然会按照预处理、编译和链接的过程依次进行。如果深究起来，上面这条命令大致相当于依次执行如下三条命令：# gcc -c foo1.c -o foo1.o# gcc -c foo2.c -o foo2.o# gcc foo1.o foo2.o -o foo在编译一个包含许多源文件的工程时，若只用一条GCC命令来完成编译是非常浪费时间的。假设 项目中有100个源文件需要编译，并且每个源文件中都包含 10000行代码，如果像上面那样仅用一条GCC命令来完成编译工作，那么GCC需要将每个源文件都重新编译一遍，然后再全部连接起来。很显然，这样浪费 的时间相当多，尤其是当用户只是修改了其中某一个文件的时候，完全没有必要将每个文件都重新编译一遍，因为很多已经生成的目标文件是不会改变的。要解决这 个问题，关键是要灵活运用GCC，同时还要借助像Make这样的工具。 警告提示功能 GCC包含完整的出错检查和警告提示功能，它们可以帮助Linux程序员写出更加专业和优美的代码。先来读读清单2所示的程序，这段代码写得很糟糕，仔细检查一下不难挑出很多毛病：◆main函数的返回值被声明为void，但实际上应该是int；◆使用了GNU语法扩展，即使用long long来声明64位整数，不符合ANSI/ISO C语言标准；◆main函数在终止前没有调用return语句。 清单2：illcode.c #include void main(void){long long int var = 1.printf("It is not standard C code!\\n").}下面来看看GCC是如何帮助程序员来发现这些错误的。当GCC在编译不符合ANSI/ISO C语言标准的源代码时，如果加上了-pedantic选项，那么使用了扩展

语法的地方将产生相应的警告信息：# gcc -pedantic illcode.c -o illcodeillcode.c: In function `main:illcode.c:9: ISO C89 does not support `long longillcode.c:8: return type of `main is not `int需要注意的是，-pedantic编译选项并不能保证被编译程序与ANSI/ISO C标准的完全兼容，它仅仅只能用来帮助Linux程序员离这个目标越来越近。或者换句话说，-pedantic选项能够帮助程序员发现一些不符合 ANSI/ISO C标准的代码，但不是全部，事实上只有ANSI/ISO C语言标准中要求进行编译器诊断的那些情况，才有可能被GCC发现并提出警告。 除了-pedantic之外，GCC还有一些其它编译选项也能够产生有用的警告信息。这些选项大多以-W开头，其中最有价值的当数-Wall了，使用它能够使GCC产生尽可能多的警告信息：# gcc -Wall illcode.c -o illcodeillcode.c:8: warning: return type of `main is not `intillcode.c: In function `main:illcode.c:9: warning: unused variable `varGCC给出的警告信息虽然从严格意义上说不能算作是错误，但却很可能成为错误的栖身之所。一个优秀的Linux程序员应该尽量避免产生警告信息，使自己的代码始终保持简洁、优美和健壮的特性。 在处理警告方面，另一个常用的编译选项是-Werror，它要求GCC将所有的警告 当成错误进行处理，这在使用自动编译工具（如Make等）时非常有用。如果编译时带上-Werror选项，那么GCC会在所有产生警告的地方停止编译，迫 使程序员对自己的代码进行修改。只有当相应的警告信息消除时，才可能将编译过程继续朝前推进。执行情况如下：# gcc -Wall -Werror illcode.c -o illcodecc1: warnings being treated as errorsillcode.c:8: warning: return type of `main is not `intillcode.c: In function `main:illcode.c:9:

warning: unused variable `var对Linux程序员来讲，GCC给出的警告信息是很有价值的，它们不仅可以帮助程序员写出更加健壮的程序，而且还是跟踪和调试程序的有力工具。建议在用GCC编译源代码时始终带上-Wall选项，并把它逐渐培养成为一种习惯，这对找出常见的隐式编程错误很有帮助。 库依赖 在Linux 下开发软件时，完全不使用第三方函数库的情况是比较少见的，通常来讲都需要借助一个或多个函数库的支持才能够完成相应的功能。从程序员的角度看，函数库实 际上就是一些头文件（.h）和库文件（.so或者.a）的集合。虽然Linux下的大多数函数都默认将头文件放到/usr/include/目录下，而库 文件则放到/usr/lib/目录下，但并不是所有的情况都是这样。正因如此，GCC在编译时必须有自己的办法来查找所需要的头文件和库文件。 GCC采用搜索目录的办法来查找所需要的文件，-I选项可以向GCC的头文件搜索路径中添加新的目录。例如，如果在/home/xiaowp/include/目录下有编译时所需要的头文件，为了让GCC能够顺利地找到它们，就可以使用-I选项： # gcc foo.c -I /home/xiaowp/include -o foo同样，如果使用了不在标准位置的库文件，那么可以通过-L选项向GCC的库文件搜索路径中添加新的目录。例如，如果在/home/xiaowp/lib/目录下有链接时所需要的库文件libfoo.so，为了让GCC能够顺利地找到它，可以使用下面的命令： # gcc foo.c -L /home/xiaowp/lib -lfoo -o foo值得好好解释一下的是-l选项，它指示GCC去连接库文件libfoo.so。Linux下 的库文件在命名时有一个约定，那就是应该以lib三个字母开头，由于所有的库文件都遵循了同样的规范，因此在用-l选项指定链接的库文件名时可以省去 lib三个字母，也就是说GCC在

对-lfoo进行处理时，会自动去链接名为libfoo.so的文件。
Linux下的库文件分为两大类分别是动态链接库（通常以.so结尾）和静态链 接库（通常以.a结尾），两者的差别仅在程序执行时所需的代码是在运行时动态加载的，还是在编译时静态加载的。默认情况下，GCC在链接时优先使用动态链 接库，只有当动态链接库不存在时才考虑使用静态链接库，如果需要的话可以在编译时加上-static选项，强制使用静态链接库。例如，如果在 /home/xiaowp/lib/目录下有链接时所需要的库文件libfoo.so和libfoo.a，为了让 GCC在链接时只用到静态链接库，可以使用下面的命令：# gcc foo.c -L /home/xiaowp/lib -static -lfoo -o foo 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com