        就C＋＋开发工具而言，与Windows下微软（VC， VS2005等）一统天下相比，Linux/Unix下C＋＋开发，可谓五花八门，各式各样。Emacs, vi, eclipse, anjuta，kdevelop等层出不穷。

        Windows下，开发工具多以集成开发环境IDE的形式展现给最终用户。例如，VS2005集成了编辑器，宏汇编ml，C /C++编译器cl，资源编译器rc，调试器，文档生成工具, nmake。它们以集成方式提供给最终用户，对于初学者而言十分方便。但是，这种商业模式，直接导致用户可定制性差，不利于自动化，集成第三方工具的能力弱。

例如，无法定制一些宏来处理一些重复操作；体会不到自动化makefile一步到位快感；无法远程登录到服务器上进行开发；无法使用某种粘合剂来把第三方工具（例如，文本工具，字符串工具）有效地调用起来。可以说，良好的商业支持和傻瓜式开发，是它们主要的优点。

        在linux下，开发工具被切割成一个个独立的小工具。各自处理不同的问题。例如:

编辑器（emacs, vim）                     用来进行编辑程序的  
调试器（gdb）                                用来调试程序  
编译器（GCC）                              用来编译和链接程序的  
性能分析工具（gcov, gprof）           用来优化程序的  
文档生成器（doxygen）                用来生成文档的

         同时，还有一些系统工具和系统知识，我们是很有必要了解的：程序自动化机制 makefile，系统粘合剂shell，系统查找工具grep, locate, find。其它的工具（例如ctags， OCI公司的MPC等等），一旦熟练掌握，它们将成为你手中的利器。

        本文主要是一些针对LINUX下开发工具使用的经验之谈。由于，工具品种繁多，我们没有能力也没有必要一一介绍。对于LINUX下IDE工具，例如 eclipse, anjuta等，它们虽然也很实用，但是使用起来比较简单，而且目前还算不上主流。所以，它们将不被着重介绍。同时，本文也不打算写成各个工具的操作手册，只着眼于介绍各个工具的想要解决的问题、运行机理和主要特性。

    
        **编辑器**   
        要进行开发，第一件事情就是选择一个合适的编辑器。编辑器选择有几个要素：   
        **1）**减少不必要的编辑动作，减少编辑的时间。   
        一切能够无二义性描述出来的编辑任务，都可以而且应该能被自动化。例如，每一个C＋＋程序都会有一个main函数；我们在定义.h文件时，都希望加入一些预处理指令＃define来帮我们解决重复引用同一个头文件而带来的麻烦。鼠标操作总是比键盘操作要慢的。这方面 **EMACS** 做得可算是到了极致。所以， EMACS用户经常会吹嘘：他们编辑的速度等同于他们思考的速度。

=============================================================================

**链表处理语言 (有专贴)**

**LISP**（全名LISt Processor，即链表处理语言），由[约翰·麦卡锡](http://baike.baidu.com/view/2154954.htm)在1960年左右创造的一种基于[λ演算](http://baike.baidu.com/view/1179241.htm)的函数式编程语言。

　　LISP有很多种方言，各个实现中的语言不完全一样。各种LISP方言的长处在于操作符号性的数据和复杂的数据结构。1980年代Guy L. Steele编写了Common Lisp试图进行标准化，这个标准被大多数[解释器](http://baike.baidu.com/view/592974.htm)和[编译器](http://baike.baidu.com/view/487018.htm)所接受。在[Unix](http://baike.baidu.com/view/8095.htm)/[Linux](http://baike.baidu.com/view/1634.htm)系统中，还有一种和[Emacs](http://baike.baidu.com/view/113220.htm)一起的Emacs Lisp（而Emacs正是用Lisp编写的）非常流行，并建立了自己的标准。

　　LISP的祖先是1950年代Carnegie-Mellon大学的Newell、Shaw、Simon开发的[IPL](http://baike.baidu.com/view/1580777.htm)语言。

　　LISP语言的主要现代版本包括Common Lisp和[Scheme](http://baike.baidu.com/view/459697.htm)。

　　lisp拥有理论上最高的运算能力

　　lisp在cad绘图软件上的应用非常广泛，普通用户均可以用lisp编写出各种定制的绘图命令。

=============================================================================

        **2）**可扩展性高。   
        程序员预期的编辑器应该能提供一些编程的帮助，例如，语法高亮，自动补齐，自动排版，语法检查等等。留心观察一下gedit, vim, emacs, ultraEdit，就会发现它们提供的远不是windows 记事本，写字板提供的那么简陋的功能。对于一种新的语言，新的语法，它们应该能很方便地提供支持，而不停留在一种或几种固定的语言上。

        **3）**用户可定制性高。   
        如果想长期从事研发， 特别是linux/unix下研发的话，那么你很有必要学好一个功能足够的编辑器。有这么一句话：Linux下程序员分为三种，使用emacs的，使用vi的，还有其它。   
        EMACS是Stallman用lisp语言写的一个GPL的编辑器。我们这里所说的emacs指的是GNU emacs，而非Xemacs。由于它的开放性，我们可以把它打造成一个功能强大的IDE。我们在安装好CGYwin之后，也可以在Windows系统下使用 EMACS。CGYwin和MINGW是第三方写的一个在Windows系统上模拟POSIX系统的工具。   
        EMACS与其说的是一个编辑器，倒不如说它是一个操作系统。我们可以用它来写编程，写wiki，收发邮件等等。EMACS主要是通过两种方式来进行扩展：el脚本(elisp是lisp的一种方言）和第三方扩展包。**EMACS的入门成本很高**。由于是纯键盘操作，所以需要记忆大量的快捷键；功能强大是通过用户添加一些扩展包，lisp脚本来实现的。如何正确配置和修改是很需要耐心和技巧的。

        **编译器**   
        编译器首选GCC（GNU COMPILER COLLECTION)。原因有两个，它是GNU开源的，同时它对标准C++的支持度高达96.15%。而VC++6.0的支持度只有83.43%。 GCC不仅是通常意义上的C或C++的编译器，它还可以编译java等其它语言。gcc是GUN c的编译器, **g++** 是GUN c++的编译器, 而EGCS(Enhanced GNU Compiler Suite)可以认为是gcc的改进版。   
        编译语言从源程序到目标代码会经过如下几个阶段：源程序－＞汇编程序－＞编译成obj程序－&gt;链接成最终可执行程序。我们可以通过一条编译指令来完成所有步骤。也可以分步执行。

        gcc有三个重要选项**-E(只进行预处理), -S(生成汇编代码), -g(生成带原代码调试符号的可执行文件**，如果想用gdb调试的话，就应该在编译时打开这个选项）。   
        GCC可以看作一个软件包，除了编译工具，它还集成了**调试器gdb**，**性能分析工具gcov, gprof**。只要我们装好了GCC，这些强大工具就可以直接使用了。   
        通过**gcov**，我们可以查看一个程序，源代码中每行代码的运行次数。我们优化运行次数最多的代码，那么就可以大大优化程序。使用gcov时，需要打开 GCC的 fprofile-arcs 和 ftest-coverage 两个选项。gcov中常用的选项有-b分支统计信息。   
        通过**gprof**工具，我们可以查看函数之间的调用顺序，及各个函数运行的时间。我们可以将gprof理解为linux/unix自带工具time的加强版。使用gprof时，需要打开GCC的pg选项。   
        gcov 和 gprof 的共同点是在编译程序时，加入自己的一些辅助信息，由此来进行程序诊断。除了，这些优化手段，我们还可以使用一些内存泄漏工具，来减少野指针，未释放的内存空间。

         **调试器**   
        GDB即GNU的调试器，它是GCC附带的一个性能优质的调试器。通过GDB和脚本结合，我们可以很好的**实现回归测试**。   
        GDB可以运行于CLI和GUI两种模式（命令行和图形模式） 。默认GDB是CLI模式的，我们可以去下载和安装GUI模式的GDB，例如xxgdb， ddd等。一个更好的方式是在 EMACS中使用GDB。GDB包括visual studio工具的所有调试功能，还包括它没有的功能。它除了支持，我们一般的设置断点，单步跟踪，step in, step out, step over等，还有一些强大的功能。在gdb中，我们可以有以下几种暂停方式：断点（BreakPoint）、观察点（WatchPoint）、捕捉点（CatchPoint）、信号（Signals）、线程停止（Thread Stops）。   
        下面列举几个让我印象深刻的功能。1）通过 watch指令，可以让程序在某个变量的值发生变化时，暂停下来。2）通过print指令，在程序运行时，设置变量的值，运行一个程序自身支持的一个方法。3）通过until指令，我们可以让程序在运行到某个程序时暂停下来。4）通过break.. if指令，使得程序在满足某个bool表达式时，暂停下来。

         **粘合剂**   
        我想通过粘合剂这个词来表达将多个工具粘合起来的胶水。例如，通过shell脚本，我们可以把OS命令，sed指令，awk指令，其它脚本文件等串联起来，发挥它们的合力。在linux C++编程中，我们不可避免地会使用makefile文件。通过它我们可以把编译指令，生成文档操作，清除操作等等串联起来。从某种意义上来看，它也相当于一个粘合剂。   
        makefile的出发点是，维护好一个项目中众多文件的依赖关系，由此得到一个源程序的拓扑图。当我们只修改图中某个结点时，重新编译时就只需要将拓扑图中关联的链路进行编译就好了。由此，大大缩短了编译的时间。make有两大概念：dependencies和rules。规则rule即针对每一个依赖关系 dependency定义一个操作规则。这个细粒度的分离，就可以使我们可以定制软件构建的行为。例如，修改使用的编译器，修改includepath, 修改libpath, 修改编译选项等等。我们常见的VC中的nmake，功能和make是类似的。   
        make使用的重点和难点是编写Makefile文件。Makefile的语法相对其它语言来说是很不一样的，我们要特别注意TAB键和空格键的区别。有很多工具可以用来帮助我们生成Makefile。最出名的就是GNU的**autoconf**了。一个GNU程序的编写，需要autoscan, aclocal, autoconf, automake这四个工具。   
        我们知道GNU软件安装的三步曲是：./configure, make, make install。其中./configure就是根据autoconf, alocal等工具生成一个makefile文件。make指令就是调用make指令来根据makefile文件的规则来编译源程序。而make install就是执行makefile中的install规则指出的操作（一般是copy操作）。而make clean就是执行makefile中的clean规则指出的操作（一般是rm操作）。我们用Eclispe+CDT开发Managed C++ Project时，它就是通过objects.mk,subdir.mk,sources.mk三个文件来生成Makefile。我们注意观察编译时的输出信息，就可以看到显示的Makefile文件的内容。   
        可以说，如果想编译出跨平台的C＋＋程序，那么makefile是一种最方便的机制。   
        OCI公司为Douglas C.Schmidt的ACE，TAO开源社区编写了一段伟大的perl脚本－－MPC。它由平台信息，一个规则文件，源代码，生成用户想要的工程文件，例如Make, Nmake, Visual C++ 6, Visual C++ 7等等。Google Web Tookit, Celtix做的事情与之类似，不过它们是针对JAVA的，而MPC是针对C＋＋的。

         **结束语**   
        国内资料太多的低层次的重复，经常是一个网页被多次转载，而且回答问题时深度不够。个人找资料的顺序是：

        查看quick start或how to文档－＞自带的帮助（如果看起来不太吃力话）－＞百度查一下中文网页，来理清一下基本概念－＞google查一下－＞几个大的，相关的网站查下资料－＞看自带帮助。再者，面对面的交流是十分重要的，大家可以相互理一下概念，交流一下心得。可惜，我身边这种氛围还是不够。 // 这种气氛现在想来是无比的难得  
        Linux下开发还可以更友好一些。个人感觉， linux开发要在国内普通程序员中大规模普及，还有一段很长的路要走。ubuntu火爆的主要原因，就是它帮用户搭好一些默认配置。用户如果想新加一些服务，它们提供了良好，有效地支持。所以，我想我们可以在IDE和现在linux这种一个个小部件的这两种状态之间，取一个折衷。也就是针对几种主要需求的用户，发布一些配置好的环境。尤其是emacs的各种el脚本（例如界面主题的color-theme， C/C++语言编辑策略，代码样式设置脚本，各种emacs系统设置）， 第三方扩展包（模板template, 编程支持包cedet等）。每一个用户浪费时间来进行这些配置是十分没有意义的！工具始终只是工具，我们不能沦为工具的奴隶，不能把一大部分精力浪费在配置工具上。   
        值得注意的是现在方兴未艾的eclipse有向这方面迈进的趋势。但是，现在emacs已经做得很好了，ecilpse能否超越它，我们还有待观察。我们可以通过在eclipse上安装SDT插件来进行C++开发。但是，它目前还不支持调试功能，而且不太稳定，功能不够强。例如，有时会无故死掉；如果想通过eclipse来转向函数原型的话，那么迎接你的将是一个漫长、焦急的等待。   
        由于本人缺少在linux下进行实际大规模程序的开发经验，对很多工具和机制的理解还比较肤浅。对它们的熟悉程度离真正实用，还有一段很长的路要走。