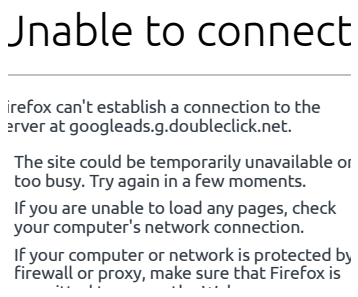
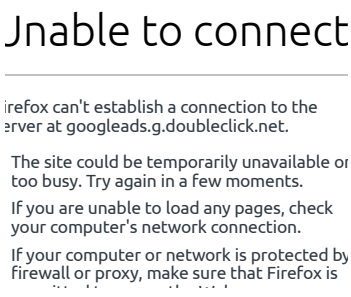
Linux下Google Test （GTest）测试环境搭建步骤

[日期：2015-05-01] 来源：Linux社区 作者：xuning [字体：大 中 小]





本文将介绍单元测试工具Google Test（GTest）在Linux操作系统中测试环境的搭建方法。本文属于Google Test使用的基础教程。在Linux中使用Google Test之前，需要对如下知识有一些了解：

（1）C/C++编程方法

（2）makefile的编写

（3）Linux命令行操作

（4）GTEST背景知识：它是用来干什么的？它用什么语言编写的？它的一些基本语法？等等

以上知识不需要做到熟能生巧，只要达到入门级别，使用GTEST不会费任何力气，毕竟GTEST也不是一个高门槛的工具。言归正传，我们来按照步骤逐一讲解：

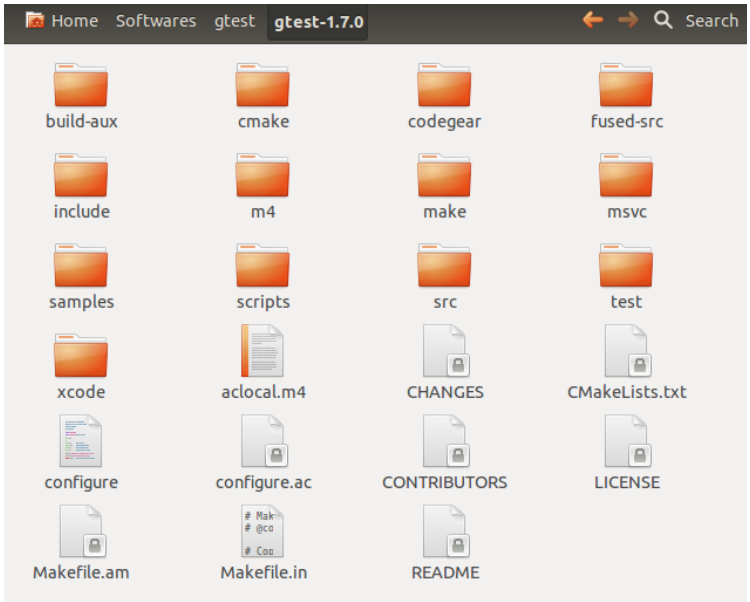
Step 1：下载GTEST

下载链接为：https://code.google.com/p/googletest/downloads/list

目前GTEST的最新版本为gtest-1.7.0.zip，因此我们在本文中将以这个版本作为例子讲解。

Step 2：认识文件夹

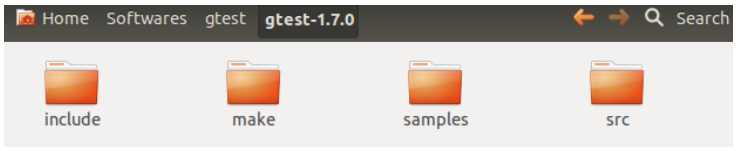
下载并解压之后，就可以看到gtest-1.7.0文件夹了，里面的内容如下图所示



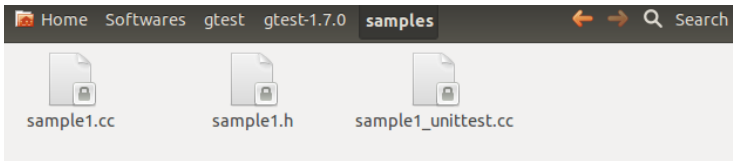
这么多文件乍一看确实有点让人摸不到头脑。其实，GTEST提供了对于多个不同平台的支持，例如msvc文件夹是用在微软Visual Studio中，xcode文件夹是用于Mac Xcode，codegear文件夹是用于Borland C++ Builder，在Linux环境中，我们用的内容就是make文件夹了。

Step 3：清除不需要的文件

前一步中我们已经讲到，很多文件是为了支持不同平台的，为了保持程序的简洁，避免混淆试听，我们在这一步把所有不需要的文件全部删除，只保留我们需要的。其实我们只需要4个文件夹，如下图所示。其余的文件以及文件夹全部删除（这一步并不是必须，但对于我们有洁癖的程序员来讲，容不得一堆不用的代码放在那~）。



好了，只剩下了四个文件夹，看上去是不是清爽了很多？其实打开make文件夹，你会发现里面只有一个Makefile文件。查看Makefile文件内容，得知这是系统给出的编译samples文件夹中的第一个sample的命令。但是打开sample文件夹，又看到里面一大坨源文件。在本入门教程中，我们先不考虑那些复杂的例子。因此，打开samples文件夹，开始删文件，删到只剩下如图所示的三个文件为止。

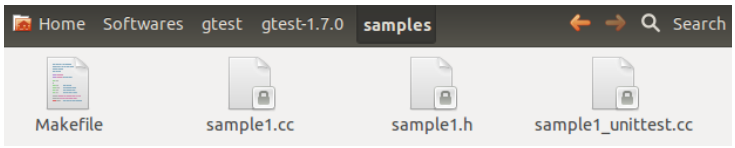


我们的程序越来越清爽了^_^。

Step 4：改写Makefile文件

此时如果你到make文件夹下，通过命令行执行 $ make && ./sample1_unittest 命令，可以看到测试的执行结果。不过如果打开Makefile查看一下，就会发现这个makefile只适用于编译sample1，如果我再增加一个被测的源文件呢？又要重新写makefile，太麻烦了。于是，在这一步，我们改写一下Makefile。

上一步我们讲到，现在只剩下4个文件夹（include，make，samples，src），既然make里面的唯一一个文件也要被改写，那也没必要留这个文件夹了。于是，在Step 4，你要做的第一件事情就是，把make文件夹，连同里面的Makefile文件全部删除……然后，进入samples文件夹，自己创建一个文件，名为Makefile，如图所示

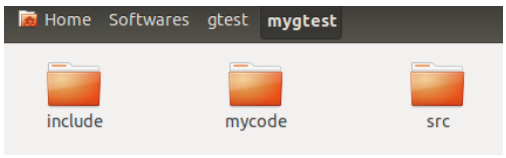


然后，打开Makefile文件，写入以下内容，如图所示（不要把图中的行号也写进去哦~）。这个新的Makefile是由刚才被我们删除的Makefile改写而来的，如果你好奇的话可以比较一下它们之间的差别，里面涉及到一些makefile的语法和函数，如果不熟的话，你可能需要花费几分钟查一下资料去了解他们。

注意下图中改写的Makefile第32行，我们编译的是后缀名为cpp的文件，而原来给的例子却以cc结尾。因此，你还要做一件事情，就是把sample1.cc的文件名改为sample1.cpp，把sample1_unittest.cc的文件名改为sample1_unittest.cpp，就大功告成了。

```
GTEST_DIR = ..
USER_DIR = .
CPPFLAGS += -isystem $(GTEST_DIR)/include
CXXFLAGS += -g -Wall -Wextra -pthread
TESTS = run_test
GTEST_HEADERS = $(GTEST_DIR)/include/gtest/*.h \
                $(GTEST_DIR)/include/gtest/internal/*.h
FILES = $(foreach d, $(USER_DIR), $(wildcard $(d)/*.cpp))
OBJS = $(patsubst %.cpp, %.o, $(FILES))


all : $(TESTS)

clean :
    rm -f $(TESTS) gtest_main.a *.o

.PHONY : clean

GTEST_SRCS_ = $(GTEST_DIR)/src/*.cc $(GTEST_DIR)/src/*.h $(GTEST_HEADERS)

gtest-all.o : $(GTEST_SRCS_)
    $(CXX) $(CPPFLAGS) -I$(GTEST_DIR) $(CXXFLAGS) -c \
            $(GTEST_DIR)/src/gtest-all.cc

gtest_main.o : $(GTEST_SRCS_)
    $(CXX) $(CPPFLAGS) -I$(GTEST_DIR) $(CXXFLAGS) -c \
            $(GTEST_DIR)/src/gtest_main.cc

gtest_main.a : gtest-all.o gtest_main.o
    $(AR) $(ARFLAGS) $@ $^

%.o : %.cpp
    $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $< -o $@

$(TESTS) : $(OBJS) gtest_main.a
    $(CXX) $(CPPFLAGS) $(CXXFLAGS) -lpthread $^ -o $@
```

现在我们的文件夹有三个（include，src，samples），我们自己被测的程序放在sample文件夹中。这个文件夹的名字看着也比较不爽，你可以把它改为mycode或者testcode，然后GTEST根目录的文件夹名称gtest-1.7.0也可以改为mygtest之类，用以满足我们完美主义者的需求。如图所示：



现在，进入命令行进行编译执行操作： $ make && ./run_test，就可以看到结果了，如图所示：

```
Running main() from gtest_main.cc
[==========] Running 6 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 3 tests from FactorialTest
[ RUN      ] FactorialTest.Negative
[       OK ] FactorialTest.Negative (0 ms)
[ RUN      ] FactorialTest.Zero
[       OK ] FactorialTest.Zero (0 ms)
[ RUN      ] FactorialTest.Positive
[       OK ] FactorialTest.Positive (0 ms)
[----------] 3 tests from FactorialTest (0 ms total)

[----------] 3 tests from IsPrimeTest
[ RUN      ] IsPrimeTest.Negative
[       OK ] IsPrimeTest.Negative (0 ms)
[ RUN      ] IsPrimeTest.Trivial
[       OK ] IsPrimeTest.Trivial (0 ms)
[ RUN      ] IsPrimeTest.Positive
[       OK ] IsPrimeTest.Positive (0 ms)
[----------] 3 tests from IsPrimeTest (0 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 2 test cases ran. (0 ms total)
[  PASSED  ] 6 tests.
```

Step 5：添加自己的测试函数

到上面一步，其实我们的测试环境已经搭建完成。如果你现在有一个函数想要被测试一下，可以继续阅读Step 5。

假设我们现在有一个待测函数sqrt.cpp以及它的头文件sqrt.h，他们的内容如下：

(sqrt.cpp)

```cpp
#include "sqrt.h"

int sqrt(int x) {
    if(x<=0) return 0;
    if(x==1) return 1;

    int small=0;
    int large=x;
    int temp=x/2;

    while(small<large){
        int a = x/temp;
        int b = x/(temp+1);

        if (a==temp) return a;
        if (b==temp+1) return b;

        if(temp<a && temp+1>b){
            return temp;
        }
        else if(temp<a && temp+1<b){
            small=temp+1;
            temp = (small+large)/2;
        }else {
            large = temp;
            temp = (small+large)/2;
        }
    }
    return -1;
}
```

(sqrt.h)

```cpp
#ifndef _SQRT_H_
#define _SQRT_H_

int sqrt(int x);

#endif //_SQRT_H_
```

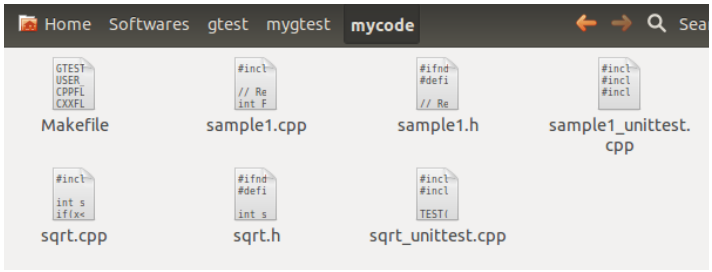
(sqrt_unittest.cpp)

```cpp
#include "sqrt.h"
#include "gtest/gtest.h"

TEST(SQRTTest, Zero){
    EXPECT_EQ(0, sqrt(0));
}

TEST(SQRTTest, Positive){
    EXPECT_EQ(100, sqrt(10000));
    EXPECT_EQ(1000, sqrt(1000009));
    EXPECT_EQ(99, sqrt(9810));
}

TEST(SQRTTest, Negative){
    int i=-1;
    EXPECT_EQ(0, sqrt(i));
}

```

这个被测文件的作用是计算任意一个正整数的平方根，算法复杂度在log(n)级别。将以上三个文件放在mycode文件夹中，然后 make && ./run_test进行编译运行，就可以看到结果了：

```
Running main() from gtest_main.cc
[==========] Running 9 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 3 tests from FactorialTest
[ RUN      ] FactorialTest.Negative
[       OK ] FactorialTest.Negative (0 ms)
[ RUN      ] FactorialTest.Zero
[       OK ] FactorialTest.Zero (0 ms)
[ RUN      ] FactorialTest.Positive
[       OK ] FactorialTest.Positive (0 ms)
[----------] 3 tests from FactorialTest (0 ms total)

[----------] 3 tests from IsPrimeTest
[ RUN      ] IsPrimeTest.Negative
[       OK ] IsPrimeTest.Negative (0 ms)
[ RUN      ] IsPrimeTest.Trivial
[       OK ] IsPrimeTest.Trivial (0 ms)
[ RUN      ] IsPrimeTest.Positive
[       OK ] IsPrimeTest.Positive (0 ms)
[----------] 3 tests from IsPrimeTest (0 ms total)

[----------] 3 tests from SQRTTest
[ RUN      ] SQRTTest.Zero
[       OK ] SQRTTest.Zero (0 ms)
[ RUN      ] SQRTTest.Positive
[       OK ] SQRTTest.Positive (0 ms)
[ RUN      ] SQRTTest.Negative
[       OK ] SQRTTest.Negative (0 ms)
[----------] 3 tests from SQRTTest (1 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 3 test cases ran. (1 ms total)
[  PASSED  ] 9 tests.
```

总结：环境搭建完成之后，每次测试一个文件xxx.cpp以及它的xxx.h文件，就把这俩放入mycode文件夹，然后编写xxx_unittest.cpp测试文件，也放进去。然后到这个目录下用命令行 make && ./run_test就可以了，应该比最开使的时候方便了许多吧？要测试时，我们只需要三个文件放入mycode，然后命令行进入这个目录 make && ./run_test 即可完成测试。下面的图就是我现在的mycode文件夹了。xxx_unittest.cpp这个文件名并不是固定，取成别的也无所谓，只是这样更容易辨认哪一个文件是在测哪个函数而已。



当然，Google Test是一个非常强大的工具，以上所讲解的只是使用了它最最基本的功能，以及一个简单环境的搭建。其实需要用的文件就是Step 4中提到的那剩下的三个文件夹内容，进行适当的改写，就可以实现更加强大的功能。这些等到以后有机会再写啦。

更多Ubuntu相关信息见Ubuntu 专题页面 http://www.linuxidc.com/topicnews.aspx?tid=2

本文永久更新链接地址：http://www.linuxidc.com/Linux/2015-05/116894.htm

Unable to connect

Firefox can't establish a connection to the server at googleads.g.doubleclick.net.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Try Again

猜你喜欢

自动化安装工具cobbler简易安装配置
ubuntu配置 boa服务器
ubuntu 16.04 gtk扁平化主题:vimixdark-gtk-theme
hadoop2.2.0中dfsinputstream类的read方法浅析
ubuntu下实现用python开机自动更新壁纸为bing壁
android studio单刷《第一行代码——android》系列
linux内核实现中断和中断处理
android widget桌面数字时钟(digtalclockwidget)实例
oracle 10.2.0.5 64位rman迁移到11.2.0.3x64
使用git进行版本控制--在多台pc上同步源代码
busybox 1.23.2发布下载,unix常用工具包
android核心分析

百度推荐>

• 搓澡毛刷上长蘑菇 • 女孩身体注水银 • 女生受骗钱退一半 • 学生与雇工冲突

Git与Github入门学习笔记
安装完最小化 RHEL/CentOS 7 后需要做的 30 件事情（三）

相关资讯 Gtest Google Test

Linux下Gtest测试框架应用实例 (03/19/2012 17:19:54)

本文评论 查看全部评论（0）

表情: 姓名: 匿名 匿名 字数 0

同意评论声明 发表

评论声明

- 尊重网上道德，遵守中华人民共和国的各项有关法律法规
- 承担一切因您的行为而直接或间接导致的民事或刑事法律责任
- 本站管理人员有权保留或删除其管辖留言中的任意内容
- 本站有权在网站内转载或引用您的评论
- 参与本评论即表明您已经阅读并接受上述条款