C语言程序每日构建

实现方案

**修订记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 修订描述 | 作者 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**修订内容：**

|  |  |
| --- | --- |
| 版本 | 修订内容 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**目录**

[1. 引言 1](#_Toc528596160)

[1.1. 范围 1](#_Toc528596161)

[1.2. 预期读者 1](#_Toc528596162)

[1.3. 术语表 1](#_Toc528596163)

[2. C语言项目每日构建框架概述 1](#_Toc528596164)

[3. 日构建工具介绍与安装 3](#_Toc528596165)

[3.1. 日构建工具 3](#_Toc528596166)

[3.2. 下载地址和使用说明 4](#_Toc528596167)

[3.3. 相关工具软件安装和配置 4](#_Toc528596168)

[3.3.1. 安装Git 5](#_Toc528596169)

[3.3.2. 安装check 6](#_Toc528596170)

[3.3.3. 安装lcov 8](#_Toc528596171)

[3.3.4. 安装openssl 8](#_Toc528596172)

[3.3.5. 安装curl 9](#_Toc528596173)

[4. 搭建项目每日构建框架 10](#_Toc528596174)

[4.1. 配置git 10](#_Toc528596175)

[4.1.1. 配置TFS git仓库的SSH公钥 10](#_Toc528596176)

[4.1.2. 配置http/https拉取 13](#_Toc528596177)

[4.2. 使用自动测试框架封装测试用例 14](#_Toc528596178)

[4.3. 配置测试用例构建脚本 15](#_Toc528596179)

[4.4. 编写每日构建驱动脚本 15](#_Toc528596180)

[4.5. 每日构建结果展示 15](#_Toc528596181)

[4.6. 配置TFS定时任务 16](#_Toc528596182)

[A.搭建每日构建展示网站 16](#_Toc528596183)

# 引言

## 范围

本文描述了一种实现C语言项目每日构建的具体方案，给出了为Linux环境下一般C语言项目实现每日构建的详细方法。

本文假定准备实现每日构建的是Linux环境下、用C语言编写的、代码通过TFS的Git仓库保管的项目。

## 预期读者

希望实现项目每日构建的项目开发人员和系统配置管理员。

## 术语表

TFS：Visual Studio Team Foundation Server 2015

# C语言项目每日构建框架概述

按照软件工程的理论，在项目中引入每日构建机制可以带来多种好处，是现代软件开发中通行的做法。

本文核心内容分为两部分：前一部分是第三章，内容是介绍如何安装和配置每日构建环境所需的相关软件；第二部分是第四章，内容是介绍一个项目如何搭建自己的每日构建框架。

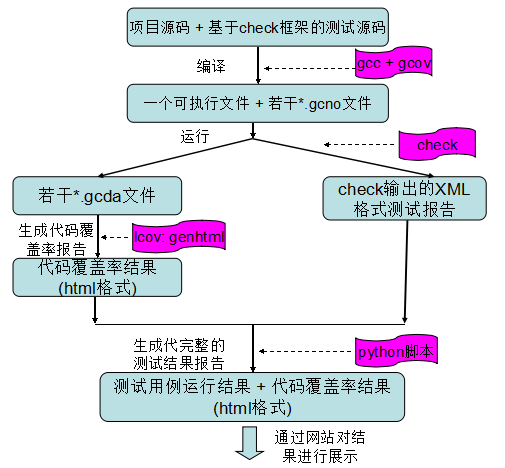


图 2-1 每日构建框架执行流程示意图

本文介绍的C语言项目每日构建框架的基本执行流程包括如下5步，如上图2-1所示。

Step1：获取最新版本的项目源码到日构建环境下，本文默认行内项目源代码均保管在通过TFS管理的Git仓库上；

Step2：编译项目源代码和测试用例，生成测试用例的二进制可执行文件及用于后续生产代码覆盖率的\*.gcno文件；

Step3：运行测试用例的二进制可执行文件，获得XML格式的测试日志和测试用例代码覆盖率的\*.gcda元数据文件；

Step4：生成测试用例覆盖率数据结果的\*.html文件，使用代码覆盖率检测工具；

Step5：合并XML格式的测试日志和HTML格式的代码覆盖率结果文件，得到最终的日构建展示文件。

# 日构建工具介绍与安装

## 日构建工具

这里所说的日构建工具主要指在Linux环境下，实现C语言日构建所依赖的、需要用户自行安装的第三方开源软件，按照用途的不同，可以分为自动化测试框架、展示工具以及依赖工具和依赖库，共三类，其中前两类是实现每日构建的核心工具。

* 自动化测试框架——check，一款开源的C语言编写的测试框架。它提供测试用例的统一包装接口和多种断言宏。check最重要的特性是支持用户对测试用例分组，并为每一组测试用例单独建立一个执行进程，保证即使组内测试用例出现Core dump也不会影响后续其他组测试用例的正常执行和测试报告的生成。支持自动导出XML格式的测试结果。
* lcov是基于GNU gcc的gcov模块的第三方开源的代码覆盖率展示工具，它可以基于gcov生成的\*.gcda文件生成格式清晰、易于阅读的HTML形式的测试程序代码覆盖率检测结果。

详细信息见下表：

* 依赖工具和依赖库，包括git、zlib、libtools、cmake。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 名称 | 功能 | 版本 | 备注 |
| 1 | check | C语言自动化测试框架 | 0.12.0 | 核心模块 |
| 2 | lcov | 日构建结果展示工具 | 1.13 | 核心模块 |
| 3 | git | 从TFS的Git仓库获取项目最新源码 | 2.19.1 | 核心模块 |
| 4 | libtool | 辅助安装工具 | 2.4.6 | 可选 |
| 5 | cmake | 辅助安装工具 | 3.4.1 | 可选 |
| 6 | zlib |  | 1.2.11 | 依赖 |
| 7 | openssl | 开源加密算法库 | 1.1.0h | 依赖 |
| 8 | curl | http通讯库 | 7.59.0 | 依赖 |

表格 3.1-1 搭建日构建框架需要依赖的工具

上述软件中，搭建每日构建框架需要使用的核心模块包括：Git、check和lcov，其他软件为安装核心模块时需要依赖的辅助工具或依赖库。详细的依赖关系如下：

1. check依赖于cmake和libtool；
2. lcov依赖于GNU的gcov（gcov是GNU GCC的组件之一，Linux下不需要单独安装）；
3. Git依赖于zlib。

## 下载地址和使用说明

上一节中给出的所有第三方开源软件，推荐用户及时升级到最新版本，确保获得最新的安全更新和BUG修复。下表给出了上述软件的官方网站地址：

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 官方网址 |
| 1 | check | <https://libcheck.github.io/check/> |
| 2 | lcov | <http://ltp.sourceforge.net/coverage/lcov.php> |
| 3 | git | <https://git-scm.com/> |
| 4 | libtool | <https://www.gnu.org/software/libtool/> |
| 5 | cmake | <https://cmake.org/> |
| 6 | zlib | <http://www.zlib.net/> |
| 7 | openssl |  |
| 8 | curl |  |

表格 3.2-1 日构建工具软件的官方网址

表格2-2中的check的官方网址中都包含有详细的使用教程和接口参考，lcov的官方网站提供有Quick Start例子，可以帮助使用者快速掌握工具的使用方法。

## 相关工具软件安装和配置

本节的各个小节中将按照依赖关系顺序逐个介绍相关软件的安装和配置方式。在此首先简单介绍Linux/Unix下C/C++软件编译安装的基础知识。

1. .bashrc文件和.profile：Linux环境下.bashrc文件位于用户主目录（~）下，在该文件中的各个配置环境变量的命令会在用户打开一个新的shell时自动执行。如果在一个打开的shell中修改该文件中的内容，保存关闭后必须执行source ~/.bashrc刷新，才能使修改对当前shell生效。AIX环境下，.profile相当于Linux下.bashrc的效果；
2. Linux环境下C/C++语言的常用环境变量及作用：

C\_INCLUDE\_PATH：gcc命令的默认头文件搜索路径之一；

CPLUS\_INCLUDE\_PATH：g++命令的默认默认头文件搜索路径之一；

LIBRARY\_PATH：编译期gcc/g++的默认搜索库目录之一；

LD\_LIBRARY\_PATH：运行期gcc/g++的默认搜索动态库目录之一；

### 安装Git

Git依赖zlib库，安装Git之前需要先安装zlib。

1. 安装依赖库zlib

# （1.1）解压缩和编译安装

tar -zxvf zlib-1.2.11.tar.gz

cd zlib-1.2.11

./configure --prefix=/path/to/your/zlib

make all && make install

# （1.2）设置环境变量，在文件 ~/.bashrc中添加如下内容, 之后保存退出执行source ~/.bashrc：

export ZLIB\_HOME=/path/to/your/zlib

export C\_INCLUDE\_PATH=$ZLIB\_HOME/include:$C\_INCLUDE\_PATH

export CPLUS\_INCLUDE\_PATH=$ZLIB\_HOME/include:$CPLUS\_INCLUDE\_PATH

export LIBRARY\_PATH=$ZLIB\_HOME/lib:$LIBRARY\_PATH

export LD\_LIBRARY\_PATH=$ZLIB\_HOME/lib:$LD\_LIBRARY\_PATH

export MANPATH=$ZLIB\_HOME/share/man:$MANPATH

1. 安装Git

# （2.1）解压缩和安装

tar -zxvf git-2.13.0.tar.gz

cd git-2.13.0

# 生成configure

make configure

# 使用configure安装

./configure --prefix=/path/to/your/git

make all && make install

# （2.2）添加环境变量，在~/.bashrc中添加如下内容后，保存退出执行source ~/.bashrc

export GIT\_HOME=path/to/your/git

export PATH=$GIT\_HOME/bin

export MANPATH=$GIT\_HOME/man:$MANPATH

# （2.3）验证安装

git --version

### 安装check

安装check有两种方式，一种是通过cmake安装，另一种是通过configure直接安装。需要注意的是，用户编译安装的check程序库的位数应该与其准备测试的C程序的位数相一致。本文主要介绍编译安装check的32位库的方法。64位库的编译安装方法可以参照执行，只需要去掉命令中的linux32和-m32参数即可。

方法一：通过cmake安装check，此方法依赖cmake和libtool，只适用于Linux平台。

# （1.1）解压缩和编译安装cmake

tar -zxvf cmake-3.4.1.tar.gz

cd cmake-3.4.1

./configure --prefix=/path/to/your/cmake

make all && make install

# （1.2）修改~/.bashrc，加入以下几行后保存退出并执行source ~/.bashrc

export CMAKE\_HOME=/path/to/your/cmake

export PATH=$PATH:$CMAKE\_HOME/bin

# （1.3）安装32位libtools

tar -zxvf libtool-2.4.6.tar.gz

cd libtool-2.4.6

env CFLAGS='-m32' ./configure --prefix=/path/to/your/libtool

linux32 make all

make install

# （1.4）修改~/.bashrc，加入以下几行后保存

export LIBTOOL\_HOME=/path/to/your/libtool

export PATH=$PATH:$LIBTOOL\_HOME/bin

export C\_INCLUDE\_PATH=$LIBTOOL\_HOME/include:$C\_INCLUDE\_PATH

export CPLUS\_INCLUDE\_PATH=$LIBTOOL\_HOME/include:$CPLUS\_INCLUDE\_PATH

export LIBRARY\_PATH=$LIBTOOL\_HOME/lib:$LIBRARY\_PATH

export LD\_LIBRARY\_PATH=$LIBTOOL\_HOME/lib:$LD\_LIBRARY\_PATH

# 保存退出后，刷新当前shell环境变量

source ~/.bashrc

# （1.5）使用cmake安装check

tar -zxvf check-0.11.0.tar.gz

cd check-0.11.0

mkdir build

cd build

cmake .. -DCMAKE\_INSTALL\_PREFIX=XXX -DCMAKE\_C\_CFLAGS=-m32 -DCMAKE\_CXX\_FLAGS=-m32

make && make install

# （1.4）在~/.bashrc中添加如下内容，保存退出后用source ~/.bashrc刷新当前shell环境变量

export $CHECK\_HOME=安装目录

export $LIBRARY\_PATH=$LIBRARY\_PATH:$CHECK\_HOME/lib

export $LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$CHECK\_HOME/lib

export C\_INCLUDE\_PATH=$C\_INCLUDE\_PATH:$CHECK\_HOME/include

export CPLUS\_INCLUDE\_PATH=$CPLUS\_INCLUDE\_PATH:$CHECK\_HOME/include

# 注意1：测试时，某些Linux平台可能需要在源码目录的src/check\_run.c里添加对<sys/wait.h>的包含才能通过编译；

# 注意2: 在64位Linux下编译，可能会由于系统中的libtool的config.guess和config.sub文件与check自带的同名文件不一致而在运行./configure时发生错误, 解决方法如下: 从libtools安装目录复制这两个文件到软件目录覆盖一下重新./configure就可以正常安装。

方法二：用configure安装check，适用于AIX/Linux。

tar -zxvf check-0.11.0.tar.gz

cd check-0.11.0

# Linux环境编译32位库：

linux32 ./configure CFLAGS=-m32 CXXFLAGS=-m32 --prefix=/path/to/your/check

# Linux环境编译64位库和AIX环境下编译32位库：

./configure --prefix=/path/to/your/check

make && make install

完成后与方法一中的（1.4）相同，在~/.bashrc文件中添加check的环境变量。

安装完成后，应该到安装目录的lib子目录下找到libcheck.a，验证其位数。

Linux下可以使用objdump -a libcheck.a来验证，注意图3.3.2左侧中绿框中的“elf32-i386”，说明这是一个32位库，右侧绿框中的“elf64-x86-64”说明这是一个64位库。

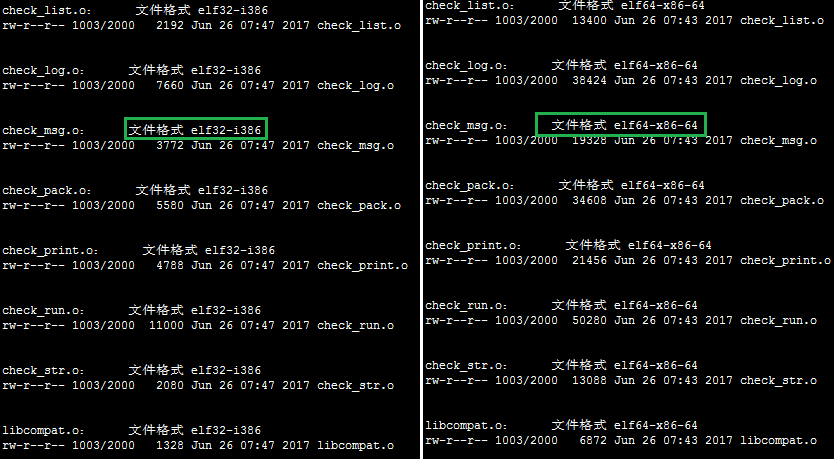


图 3.3.2-1 Linux下32位库用objdump命令显示

AIX环境下静态库位数可以通过nm -X32 libcheck.a和nm -X64 libcheck.a命令来检查。

### 安装lcov

# 解压和安装

tar -zxvf lcov-1.13.tar.gz

cd lcov-1.13

make install PREFIX=/path/to/your/lcov

# 在~/.bashrc中添加如下内容，保存退出后执行source ~/.bashrc

export LCOV\_HOME=/path/to/your/lcov

export PATH=$PATH:$LCOV\_HOME/bin

export MANPATH=$MANPATH:$LCOV\_HOME/share/man

# 验证安装

lcov --version

genhtml --version

### 安装openssl

# 解压和安装

tar -zxvf openssl-1.1.0h.tar.gz

cd openssl-1.1.0h

./config --prefix=/path/to/your/openssl

# 在~/.bashrc中添加如下内容，保存退出后执行source ~/.bashrc

export OPENSSL\_HOME=/path/to/your/openssl

export PATH=$OPENSSL\_HOME/bin:$PATH

export C\_INCLUDE\_PATH=$OPENSSL\_HOME/include:$C\_INCLUDE\_PATH

export CPLUS\_INCLUDE\_PATH=$OPENSSL\_HOME/include:$CPLUS\_INCLUDE\_PATH

export LIBRARY\_PATH=$OPENSSL\_HOME/lib:$LIBRARY\_PATH

export LD\_LIBRARY\_PATH=$OPENSSL\_HOME/lib:$LD\_LIBRARY\_PATH

export MANPATH=$OPENSSL\_HOME/share/man:$MANPATH

# 验证

Openssl version -a

### 安装curl

# 解压和安装

tar -zxvf curl-7.59.0.tar.gz

cd curl-7.59.0

./configure --prefix=/path/to/your/curl

# 在~/.bashrc中添加如下内容，保存退出后执行source ~/.bashrc

export OPENSSL\_HOME=/path/to/your/curl

export PATH=$PATH:$CURL\_HOME/bin

export C\_INCLUDE\_PATH=$C\_INCLUDE\_PATH:$CURL\_HOME/include

export CPLUS\_INCLUDE\_PATH=$CPLUS\_INCLUDE\_PATH:$CURL\_HOME/include

export LIBRARY\_PATH=$LIBRARY\_PATH:$CURL\_HOME/lib

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$CURL\_HOME/lib

export MANPATH=$CURL\_HOME/share/man:$MANPATH

# 验证

curl --version

# 搭建项目每日构建框架

## 配置git

在自动构建中，git的作用是从远程仓库自动拉取代码。目前主流git平台（比如：github，TFS git）都支持http/https和ssh两种方式，用户可以任选其一，但是每一种拉取模式都需要进行专门配置。

### 配置TFS git仓库的SSH公钥

每日构建开始时需要首先从TFS的git仓库中拉取项目最新源码，TFS收到客户端基于SSH协议的访问请求后会通过预先配置好的RSA公钥对请求方进行身份核验，只有通过核验才允许客户端获取代码。因此必须首先在TFS的项目git库中正确配置请求方（每日构建环境）的RSA公钥。

第一步，在项目每日构建环境下生成RSA公钥和私钥。

# 在命令行执行如下命令，之后出现的所有提示项一律回车

ssh-keygen -t rsa

执行后的效果如下图所示：

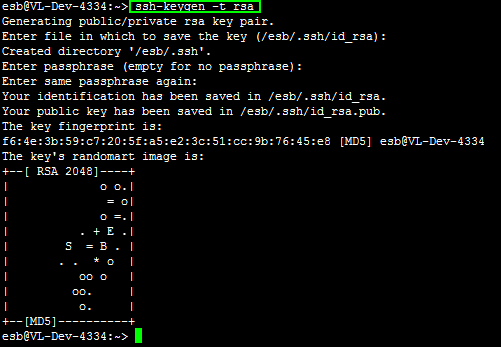


图 4.1-1 用ssh生成RSA公钥和私钥

完成后，在当前用户的根目录下会生成一个.ssh目录，可以用ls -al查看，如下图所示：

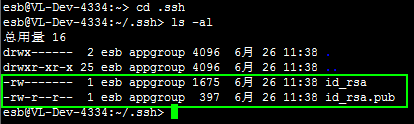


图 4.1-2 本地RSA公钥和私钥

打开id\_rsa.pub文件，其中只有一行内容，这行内容被空格分成三个部分，选择第二个部分的字符串（下图中绿色方框中的内容）并复制到剪切板，如下图所示：

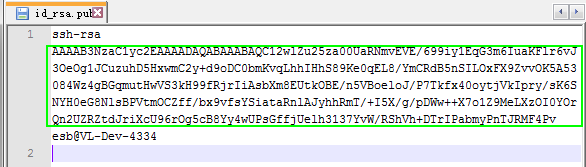


图 4.1-3 需要复制的RSA公钥文件的内容

第二步，将RSA公钥加入到TFS的git仓库的SSH密钥管理配置文件中。

登录项目的TFS页面，依次点击左上方的“代码” -> “资源管理器”，再点击右侧的：“克隆” 按钮，再在弹出的小窗口中依次点击“SSH” -> “管理SSH密钥”两个按钮，过程如下图所示：

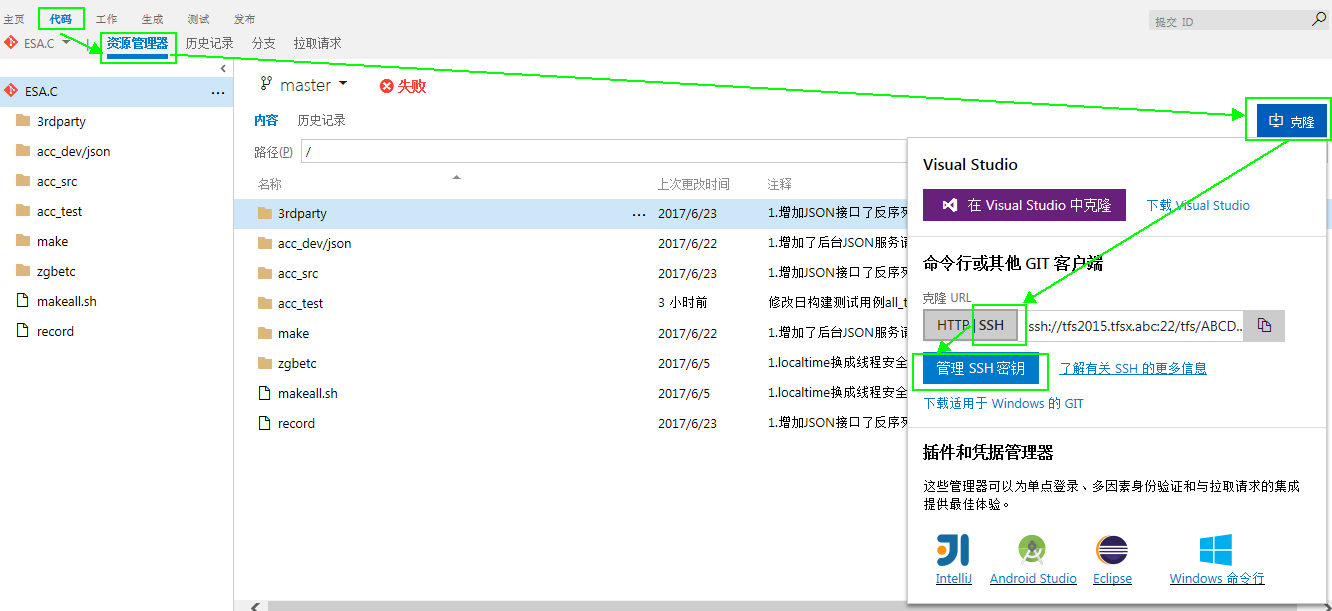


图 4.1-4 在TFS上添加RSA公钥操作步骤1

在弹出的新窗口上点击“添加SSH公钥按钮”，如下图左侧所示。之后把第一步中复制的公钥字符串粘贴在弹出新窗口的“密钥数据”一栏，如下图右侧的绿色方框中所示，“说明”一栏可以填写自定义内容，填好后点击右下角的“保存更改”按钮。



图 4.1-5 在TFS上添加RSA公钥操作步骤2

至此，在TFS的项目git仓库的密钥管理器里，添加项目每日构建环境的公钥的工作完成。后续需要在每日构建环境下获取项目最新源代码，只需要：

1. 首先从项目的TFS页面上依次点击：“代码” -> “资源管理器” -> “克隆” -> “SSH” -> 复制图标（下入中蓝色方框框住的按钮），将项目源码的git地址复制到剪切板；

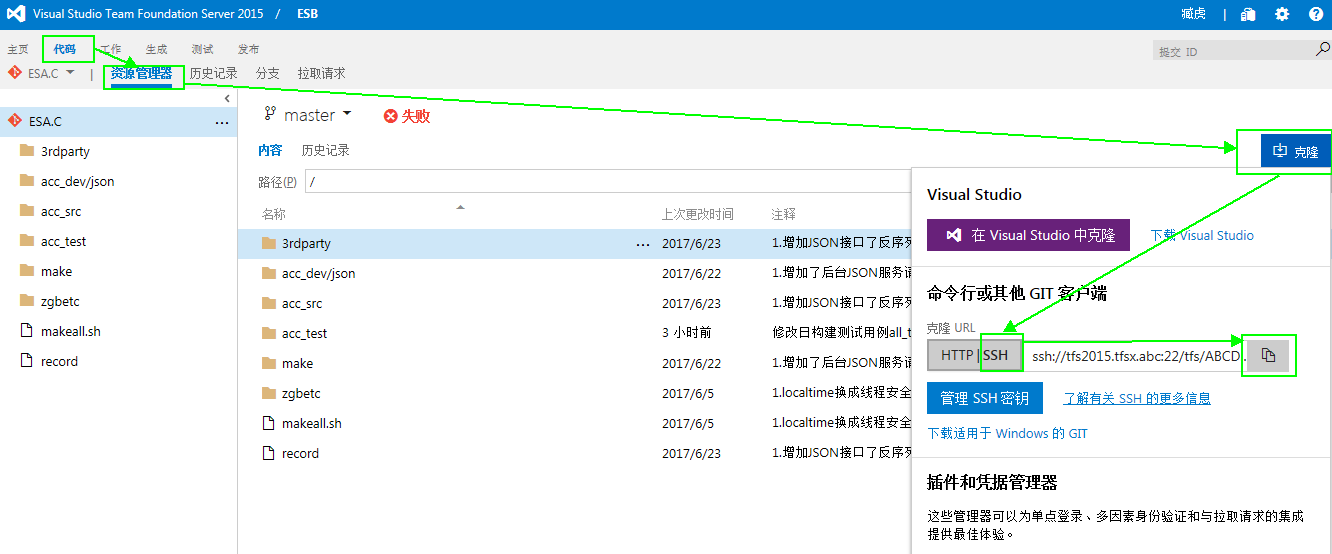


图 4.1-6 从TFS获得项目代码的git地址

1. 在每日构建环境的shell终端或shell脚本中执行

git clone 上一步复制到剪切板的项目源码git地址

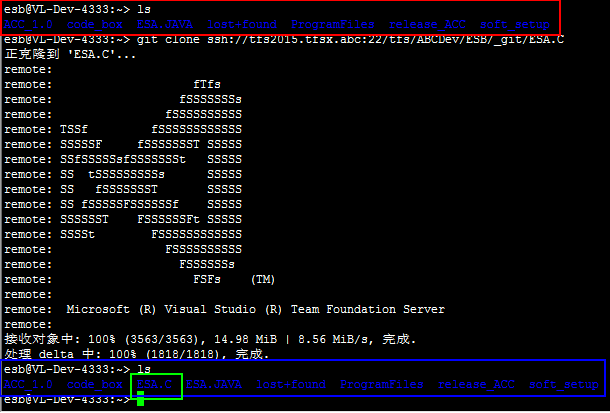


图 4.1-7 从项目的TFS的git 仓库clone最新代码

即可将TFS的git仓库中保存的项目源码的最新版本拉取到项目的每日构建环境下。注意对比上面的图2-7中红色方框和蓝色方框显示的由ls命令得到的当前目录下的子目录列表，容易看出，git clone命令执行后当前目录下多出了ESA.C目录（绿色方框括住的内容）。

这里还需要注意的是：获得项目代码的git地址一部中，请在“分支”选项中选择自己希望获取的分支，再拷贝地址。

### 配置http/https拉取

Git源码本身不包含对HTTP/HTTPS的支持，因此如果希望采用http/https模式获取代码，那么在编译git源码时应该确保本地已安装openssl和curl，这样编译出的git才能通过http/https拉取代码。

http/https方式获取代码时，可以直接通过git clone命令将用户名和密码一起上送，避免使用自动化脚本执行代码拉取时需要人工输入账号和密码，执行方式如下：

git clone http://<用户名>:<密码>@<代码库http地址>

上面的命令中，如果用户名或密码中存在不合法的url字符，需要转译成对应的合法url字符串。例如：’@’是不合法的url字符，需要转为”%40”。

## 使用自动测试框架封装测试用例

自动化测试框架是实现项目持续集成必不可少的工具，经过调研、对比和试用，本文推荐使用第三方开源的C语言项目自动测试框架check。Check作为C语言项目自动化测试框架具有以下优点：

1. 能够处理测试中出现的程序意外中止（例如：段错误等异常信号导致的终止），保证测试过程的得到完整执行并生成测试报告。check支持使用者对测试用例进行分组，在执行测试用例时，check默认为每一组测试用例单独建立一个执行进程。该特性保证整个测试过程中，即使某测试用例组内的某个用例出现Core dump，其测试结果仍然会被父进程接收，同时不会影响后续其他组测试用例的正常执行。测试用例编写人员可以通过合理控制测试用例分组的粒度来将测试进程总数控制在合理的范围，避免反复创建、销毁进程带来过多额外开销；
2. 源代码具备可移植性，在AIX环境和Liunx环境下均可编译运行；
3. 提供了统一的测试用例包装接口和多种用于检查测试用例运行结果的断言宏；
4. 支持自动导出XML格式的测试报告。

Check测试框架的使用请参考官网资料：

Tutorial: <https://libcheck.github.io/check/doc/check_html/check_3.html>

API: <https://libcheck.github.io/check/doc/doxygen/html/check_8h.html>

具体项目使用的例子请参考本文附录中的AIR平台1.0项目的ACC模块C语言版每日构建实现代码。

注意：编译基于check框架的测试用例时，需要链接静态库libcheck.a，同时需要确保链接时所用的libcheck.a的位数和项目编译所需的位数一致。

## 配置测试用例构建脚本

项目开发人员需要为项目的测试用例编写自动构建脚本，该编译脚本必须满足以下两个要求：

1. 在测试用例自动构建脚本中，必须为gcc开头的编译命令加上参数“-fprofile-arcs -ftest-coverage”，这样gcc在编译时会自动使用gcov工具；
2. 为了通过lcov获得测试用例对项目源码的代码覆盖率统计结果，必须在编译测试用例时采用直接从源码编译的方式。

执行满足上述两点要求的测试用例自动构建脚本后，会自动在构建脚本所在目录下生成多个用于检测代码覆盖率的\*.gcno文件。

## 编写每日构建驱动脚本

项目的每日构建驱动脚本是由项目组编写的用来控制项目每日构建过程的启动和终止的控制程序，将项目每日构建流程中的各个阶段连接起来，使之按照正确的顺序执行。

项目每日构建驱动脚本是本文所介绍的每日构建框架的总控制器，是TFS远程定时调用的直接对象，项目配置管理员通过在TFS配置实现对指定机器上的驱动脚本的定是调用，来达到每天定时执行项目日构建的目的（这一点将在4.6节中介绍）。

每日构建驱动脚本的实现方式有很多种，编写语言可以选择shell、Python、Perl等等，只需要保证控制逻辑的正确即可。一种具体的实现思路可以参考附录中AIR1.0平台的ACC模块C语言版的自动构建驱动脚本。

## 每日构建结果展示

选择每日构建的展示方式时，应以方便项目组成员观察、发现和跟踪项目中各个模块存在的问题，便于准确了解项目测试用例的行覆盖率、分支覆盖率等信息为原则，结合项目特点进行选择。

考虑到lcov可以直接生成HTML格式的代码覆盖率展示结果，check也可以生成XML格式的测试报告，因此本文推荐采用WEB站点作为项目每日构建的展示方式。基本实现思路是：

第一步，将check生成的XML格式测试报告转为HTML格式的文件，集成到lcov生成的代码覆盖率文件index.html中。

第二部，将代码覆盖率展示文件目录作为一个子目录移动到指定的每日构建结果保存目录；

第三步，搭建一个WEB站点。

具体实现的例子请参考附录C中的介绍。

## 配置TFS定时任务

到本节为止，项目组每日构建环境上的日构建系统的搭建工作已经全部完毕，只要有一方主动执行每日构建驱动脚本，就可以自动进行一次完整的项目日构建。构建结果可以通过项目组搭建的展示网站查看。

目前尚缺少的是日构建驱动脚本的定时启动功能，目前这一功能的实现需要项目组联系项目管理办公室负责相关工作的同事在TFS上进行配置。项目组需要将以下信息告知TFS配置人员：

1. 项目在TFS上的地址；
2. 项目组搭建的每日构建环境的IP；
3. 项目组搭建的每日构建环境的用户名和密码；
4. 每日构建驱动脚本的绝对路径；

如果项目组希望TFS每日构建完成后在日志中给出构建结果的展示网址，还需要

1. 约定每日构建结果的展示URI的获取方式，例如：作为构建脚本执行的终端输出的最后一行；

## A.搭建每日构建展示网站

ACC-C模块项目在选择每日构建结果展示方式时，采用了搭建WEB网站进行展示的方案。具体方法是使用screen命令 + python在Linux环境下搭建了一个可以实现7x24小时服务的基于HTTP的WEB站点，步骤如下：

1. 用screen命令打开一个新的shell，切换到每日构建结果保存目录；
2. 在shell中执行“python -m SimpleHTTPServer 8006”，启动一个监听本地8006端口的HTTP服务；
3. 关闭当前shell。

至此，展示网站搭建完成。任何人如果希望查看项目的每日构建结果，只需要打开浏览器，在地址栏输入“http://每日构建环境IP:展示用WEB服务端口/”即可查看。

下图B-1展示的事每日构建结果展示网站的首页面。图中首页面包含多个超链接，每个超链接对应于一次每日构建结果，超链接的名字是该每日构建的发起时间。

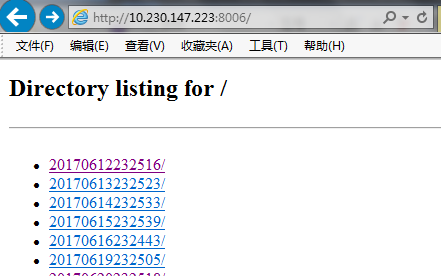


图 A-1 ACC-C的每日构建结果展示网站

下图B-2展示了点击图B-1所示的每日构建网站上的一个超链接后，打开的页面的内容。该页面展示的是一次日构建的结果。图中页面上的表格是对项目源码各个源文件（或目录）的测试覆盖率的图形化展示。

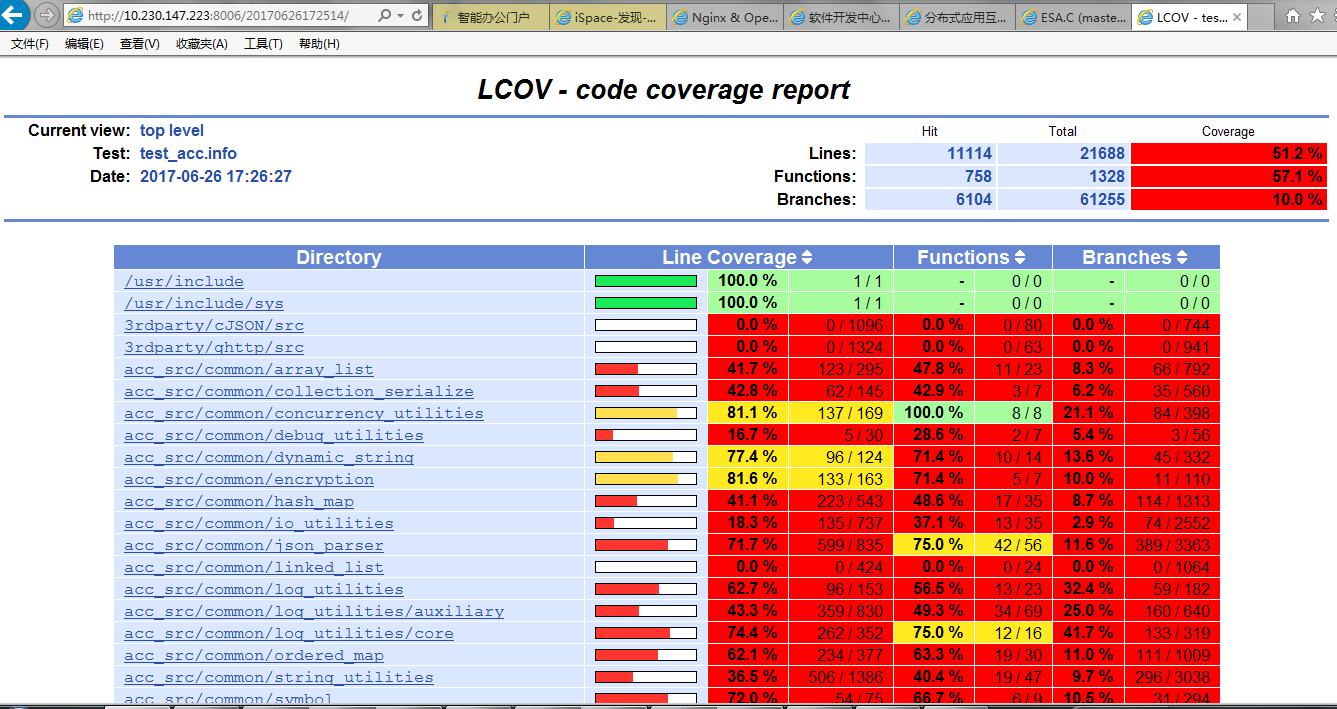


图 A-2 ACC-C的一次日构建结果展示主页面