

登录 | 注册

forlong401的专栏

Android and iOS Now! 多交流技术，多分享，技术只有分享，才会经久不衰。欢迎关注新浪微博：@forlong401。

<http://weibo.com/forlong401>

目录视图 摘要视图 RSS 订阅

个人资料



forlong401

访问: 923128次
积分: 12070
等级: 7
排名: 第346名
原创: 140篇 转载: 789篇
译文: 10篇 评论: 301条

新浪微博: @forlong401
新浪微博: @forlong401

文章分类

- [Game \(1\)](#)
- [Ant \(10\)](#)
- [Android \(457\)](#)
- [android反编译 \(10\)](#)
- [android混淆 \(6\)](#)
- [android反破解 \(9\)](#)
- [Android ListView \(10\)](#)
- [android源码下载 \(3\)](#)
- [Android View \(36\)](#)
- [Android 网络相关 \(22\)](#)
- [Android 面试 \(15\)](#)
- [Android Dalvik \(5\)](#)
- [Android NDK \(13\)](#)
- [android BT \(5\)](#)
- [Android内存相关 \(6\)](#)
- [Android的联通性 \(6\)](#)
- [Android MMS应用 \(15\)](#)
- [Android 破解 \(7\)](#)
- [Android Test \(1\)](#)
- [Android 安全 \(20\)](#)
- [Android Animation \(2\)](#)
- [Objective-C \(10\)](#)
- [Objective-c程序设计 \(第4版\) 学习笔记 \(21\)](#)
- [Objective-C++ \(4\)](#)
- [mac \(18\)](#)
- [ios \(50\)](#)

[博客专家福利](#) [2015年4月微软MVP申请](#) [10月推荐文章汇总](#) [有奖征文--我亲历的京东发展史](#) [参与迷你编程马拉松赢iPhone 6](#)

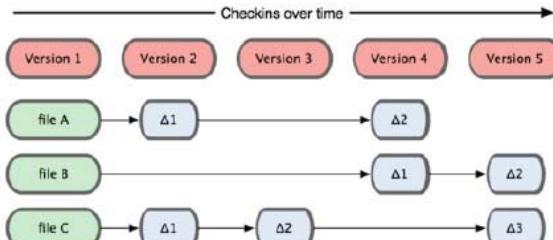
Git使用方法

分类: [Linux](#) 2012-01-29 14:10 5193人阅读 评论(1) 收藏 举报

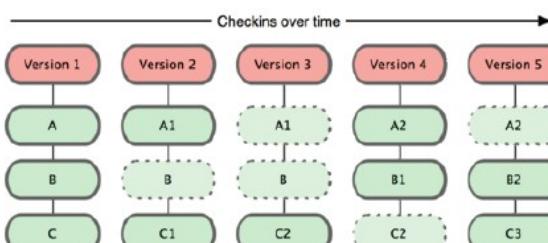
[git](#) [branch](#) [whitespace](#) [merge](#) [版本控制系统](#) [linux内核](#)

Git使用方法(一)

Linux项目开始后，绝大多数的Linux内核维护工作都花在了提交补丁和保存归档的繁琐事物上(1991——2002年间)。到2002年，整个项目组开始启用分布式版本控制系统BitKeeper来管理和维护代码。到2005年的时候，开发BitKeeper的商业公司同Linux内核开发社区的合作关系结束，他们收回了免费使用BitKeeper的权力。这就使得Linux开源社区不得不开发了自己的版本控制软件Git。



其他系统在每个版本中记录着各个文件的具体差异



Git保存每次更新的文件快照

这是Git与其他版本控制系统的主要差别，其他版本控制系统节省了磁盘空间，但增加了计算量；Git是都保存了，因为磁盘原来越便宜。

在保存到Git之前，所有数据都要进行内容的校验和(checksum)计算，并将此结果作为数据的唯一标识和索引。Git使用SHA-1算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个SHA-1哈希值，作为指纹字符串。该字符由40个十六进制字符(0-9及a-f)组成，看起来就像是：

24b9da6552252987aa493b52f8696cd6d3b00373

首先安装git，我是在ubuntu下，所以输入命令：

Java (83)
 Java Debug (1)
 Java 面试 (6)
 eclipse (3)
 C++ (2)
 C面试 (11)
 Perl (12)
 Python (9)
 php (18)
 Linux (39)
 git (9)
 svn (4)
 ubuntu (8)
 FS or OSS (7)
 Life (11)
 Software (38)
 废话 (62)
 Secure (1)
 database (18)
 前端 (15)
 HTML5 (6)
 VI (6)
 JSON (6)
 SOAP (3)
 算法 (8)
 Driver驱动工程师面试 (1)
 NFC (2)
 微信公众平台 (24)
 阿里云服务器 (5)
 Tomcat (1)
 MySQL (4)
 Sublime Text 2 (6)
 JavaScript (1)
 Swift (1)
 cas (4)
 lua (2)
 CentOS (4)
 Nginx (2)

文章存档

2014年12月 (10)
 2014年11月 (7)
 2014年10月 (16)
 2014年09月 (7)
 2014年08月 (1)

阅读排行

POP3、SMTP、IMAP和 (25335)
 Android App集成支付宝 (19651)
 Android动画效果 (16235)
 百度地图如何计算两点之 (13673)
 百度地图API>Android St (8192)
 Ubuntu11.10 64Bit版上 (8109)
 cygwin下安装软件 (7922)
 如何隐藏Android4.0及以 (7788)
 Android开源项目分类汇 (7729)
 Android APK加壳技术方 (7612)

评论排行

幸福时光--养猪的岁月 (58)
 Android开源项目分类汇 (20)
 批评“我拒绝接受的几个

\$sudo apt-get install git-core
 OK!此时应该git成功安装, 查看git版本:
\$git --version

```
hacker@ubuntu:~$ git --version
git version 1.7.0.4
```

一：获取项目的git仓库:

1) 从当前目录初始化:

\$git init
 创建了一个空的git仓库:

```
hacker@ubuntu:~/mygit/dd$ git init
Initialized empty Git repository in /home/hacker/mygit/dd/.git/
如果这个git仓库已经初始化过了, 则提示exist:
```

```
hacker@ubuntu:~/mygit/ggit$ git init
Reinitialized existing Git repository in /home/hacker/mygit/ggit/.git/
```

如果成功创建一个空的git仓库可以看到在当前目录下出现一个.git目录, 这个就是仓库了!

```
hacker@ubuntu:~/mygit/dd$ ls -al
total 12
drwxr-xr-x 3 hacker hacker 4096 2012-01-19 08:14 .
drwxr-xr-x 7 hacker hacker 4096 2012-01-19 08:14 ..
drwxr-xr-x 7 hacker hacker 4096 2012-01-19 08:14 .git
```

现在偷窥一下.git目录下都有什么:

```
hacker@ubuntu:~/mygit/dd$ ls -al .git/
total 40
drwxr-xr-x 7 hacker hacker 4096 2012-01-19 08:14 .
drwxr-xr-x 3 hacker hacker 4096 2012-01-19 08:14 ..
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 branches
-rw-r--r-- 1 hacker hacker 92 2012-01-19 08:14 config
-rw-r--r-- 1 hacker hacker 73 2012-01-19 08:14 description
-rw-r--r-- 1 hacker hacker 23 2012-01-19 08:14 HEAD
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 hooks
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 info
drwxr-xr-x 4 hacker hacker 4096 2012-01-19 08:14 objects
drwxr-xr-x 4 hacker hacker 4096 2012-01-19 08:14 refs
```

2) 从现有仓库克隆:

克隆仓库的命令为git clone [url]。比如, 要克隆Ruby语言的Git代码仓库Grit, 可以用下面的命令:

\$git clone git://github.com/schacon/grit.git

这时在当前目录下创建一个名为“grit”的目录, 其中内含一个.git目录, 并从同步后的仓库中拉出所有的数据, 取出最新版本的文件拷贝。如果想自己指定目录的名字:

\$git clone git://github.com/schacon/grit.git mygrit

获得Linux2.6内核源码:

```
hacker@ubuntu:~/mygit/ggit$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git linux-kernel
Initialized empty Git repository in /home/hacker/mygit/ggit/linux-kernel/.git/
remote: Counting objects: 2330493, done.
remote: Compressing objects: 100% (362020/362020), done.
remote: Total 2330493 (delta 1949532), reused 2326864 (delta 1945982)
Receiving objects: 100% (2330493/2330493), 465.17 MiB | 214 KiB/s, done.
Resolving deltas: 100% (1949532/1949532), done.
Checking out files: 100% (38171/38171), done.
```

如果使用git clone --bare则会只clone .git仓库, 而不会clone working directory和staging area。下面是对比, linux-kernel是使用git clone得到的, my-linux是使用git clone --bare得到的

Ubuntu 11.04 下 Android	(14)
Android App集成支付宝	(7)
Android滑动菜单特效实现	(7)
关于移动通信行业从业者	(7)
如何隐藏Android4.0及以上	(6)
android 和 ios 截屏 方法	(6)
Android APK加壳技术方	(6)
	(6)

RSS1
慕课网
Android开发者
常用javascript例子
HTML CSS文档实例资源
java Tutorial
在线源代码
英语学习
查询各种代码好网站
查询API用法例子
CSS2中文手册

RSS2
苹果开发者 (RSS)
web在线教程 (RSS)
IEEE Transactions on Software Engineering (RSS)
英语读报 (RSS)
开源无界 (RSS)

最新评论
Mac OS X下安装配置Android源码 avoidImpetuous: @forlong401:恩解决了。hdutil detach /Volume/android ...
Mac OS X下安装配置Android源码 forlong401: @x xm282828:rm -rf ~/android.dmg
百度地图如何计算两点之间距离 forlong401: @u010991643:你加 百度sdk没有? sdk的so文件也要 添加。
Mac OS X下安装配置Android源码 avoidImpetuous: 请问博主，这里 挂载后如何删除呢? # hdutil create -type SPARSE -fs ...
Eclipse打包Android项目时用到pri stone_2323: 支持! 写得很到位
百度地图如何计算两点之间距离 糖果莫: 您好,使用这个方法时候会 报 java.lang.UnsatisfiedLinkError: Nati...
Android开源项目分类汇总【畜生 绚丽星空: 不错，点个赞!
去掉 android dialog 白色边框 丁国华: 谢谢分享 学习了、 (*^_~*)
Git使用方法 syw19901001: http://www.ihref.com/read-16369.html 这篇不错，我学习了 一下。和博...
Android开源项目分类汇总【畜生 Duke_Jin: 超赞！

```

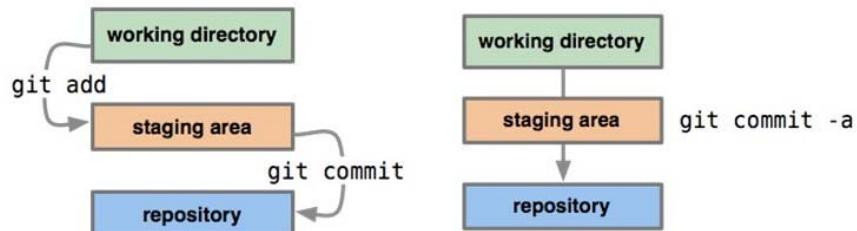
hacker@ubuntu:~/mygit/ggit/linux-kernel$ cd ..
hacker@ubuntu:~/mygit/ggit$ cd linux-kernel/
hacker@ubuntu:~/mygit/ggit/linux-kernel$ ls -al
total 524
drwxr-xr-x 24 hacker hacker 4096 2012-01-24 01:27 .
drwxr-xr-x  6 hacker hacker 4096 2012-01-24 20:01 ..
drwxr-xr-x 29 hacker hacker 4096 2012-01-24 01:27 arch
drwxr-xr-x  3 hacker hacker 4096 2012-01-24 01:27 block
-rw-r--r--  1 hacker hacker 18693 2012-01-24 01:26 COPYING
-rw-r--r--  1 hacker hacker 94984 2012-01-24 01:26 CREDITS
drwxr-xr-x  3 hacker hacker 4096 2012-01-24 01:27 crypto
drwxr-xr-x 94 hacker hacker 12288 2012-01-24 01:26 Documentation
drwxr-xr-x 99 hacker hacker 4096 2012-01-24 01:27 drivers
drwxr-xr-x 37 hacker hacker 4096 2012-01-24 01:27 firmware
drwxr-xr-x 70 hacker hacker 4096 2012-01-24 01:27 fs
drwxr-xr-x  8 hacker hacker 4096 2012-01-24 01:27 .git
-rw-r--r--  1 hacker hacker 1014 2012-01-24 01:26 .gitignore
drwxr-xr-x 22 hacker hacker 4096 2012-01-24 01:27 include
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 01:27 init
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 01:27 ipc
-rw-r--r--  1 hacker hacker 2536 2012-01-24 01:26 Kbuild
-rw-r--r--  1 hacker hacker 252 2012-01-24 01:26 Kconfig
drwxr-xr-x 10 hacker hacker 4096 2012-01-24 01:27 kernel
drwxr-xr-x  9 hacker hacker 4096 2012-01-24 01:27 lib
-rw-r--r--  1 hacker hacker 4320 2012-01-24 01:26 .mailmap
-rw-r--r--  1 hacker hacker 207164 2012-01-24 01:26 MAINTAINERS
-rw-r--r--  1 hacker hacker 53477 2012-01-24 01:26 Makefile
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 01:27 mm
drwxr-xr-x 55 hacker hacker 4096 2012-01-24 01:27 net
-rw-r--r--  1 hacker hacker 17459 2012-01-24 01:26 README
-rw-r--r--  1 hacker hacker 3371 2012-01-24 01:26 REPORTING-BUGS
drwxr-xr-x 10 hacker hacker 4096 2012-01-24 01:27 samples
drwxr-xr-x 13 hacker hacker 4096 2012-01-24 01:27 scripts
drwxr-xr-x  8 hacker hacker 4096 2012-01-24 01:27 security
drwxr-xr-x 22 hacker hacker 4096 2012-01-24 01:27 sound
drwxr-xr-x 12 hacker hacker 4096 2012-01-24 01:27 tools
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 01:27 usr
drwxr-xr-x  3 hacker hacker 4096 2012-01-24 01:27 virt
hacker@ubuntu:~/mygit/ggit/linux-kernel$ cd ../my-linux/
hacker@ubuntu:~/mygit/ggit/my-linux$ ls -al
total 68
drwxr-xr-x 7 hacker hacker 4096 2012-01-24 19:57 .
drwxr-xr-x 6 hacker hacker 4096 2012-01-24 20:01 ..
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 19:16 branches
-rw-r--r--  1 hacker hacker 66 2012-01-24 19:16 config
-rw-r--r--  1 hacker hacker 73 2012-01-24 19:16 description
-rw-r--r--  1 hacker hacker 23 2012-01-24 19:57 HEAD
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 19:16 hooks
drwxr-xr-x  2 hacker hacker 4096 2012-01-24 19:16 info
drwxr-xr-x  4 hacker hacker 4096 2012-01-24 19:16 objects
-rw-r--r--  1 hacker hacker 27805 2012-01-24 19:57 packed-refs
drwxr-xr-x  4 hacker hacker 4096 2012-01-24 19:16 refs

```

二：新加文件到index中，使得git可以跟踪它：

git有3个区域,分别是:

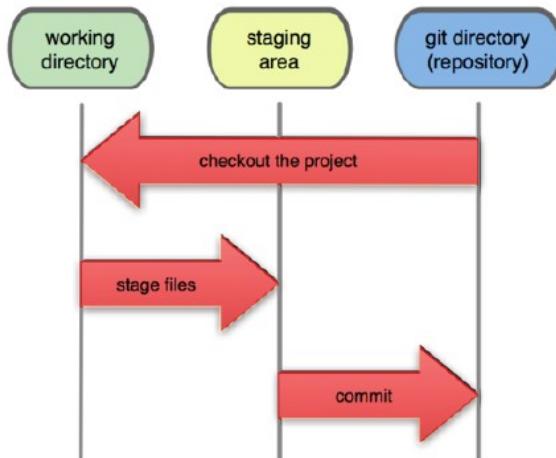
- working directory
- staging area
- repository



任何一个git里的文件都有三种状态:

- 已修改 (修改了某个文件,但是没有提交)
- 已暂存 (把修改的文件放在下次提交时要保存的清单中)
- 已提交 (该文件已经安全地保存在本地数据库中了)

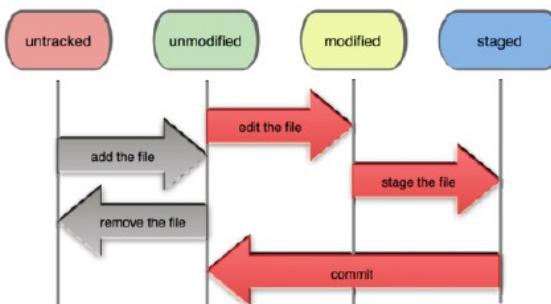
Local Operations



基本的Git工作流如下：

1. 在工作目录中修改某些文件
2. 对这些修改了的文件作快照，并保存到暂存区
3. 提交更新，将保存在暂存区域的文件快照转储到git目录中

File Status Lifecycle



工作目录下面的所有文件都不外乎这两种状态：已跟踪或未跟踪。已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是未更新的，已修改或者已放入暂存区。而所有其他文件都属于未跟踪文件。它们既没有上次更新时的快照，也不在当前的暂存区里。初次克隆某个仓库时，工作目录中的所有文件都属于已跟踪文件，且状态为未修改。

使用git status检查当前文件状态：

\$git status

```
hacker@ubuntu:~/mygit/ggit$ git status
# On branch master
nothing to commit (working directory clean)
```

说明没有跟踪任何文件

用vim修改一下工作目录下的main.c文件，再将文件git add到staging area，然后：

\$git add .

```
hacker@ubuntu:~/mygit/dd$ ls -al .git/
total 44
drwxr-xr-x 7 hacker hacker 4096 2012-01-19 08:30 .
drwxr-xr-x 3 hacker hacker 4096 2012-01-19 08:30 ..
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 branches
-rw-r--r-- 1 hacker hacker 92 2012-01-19 08:14 config
-rw-r--r-- 1 hacker hacker 73 2012-01-19 08:14 description
-rw-r--r-- 1 hacker hacker 23 2012-01-19 08:14 HEAD
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 hooks
-rw-r--r-- 1 hacker hacker 104 2012-01-19 08:30 index
drwxr-xr-x 2 hacker hacker 4096 2012-01-19 08:14 info
drwxr-xr-x 5 hacker hacker 4096 2012-01-19 08:30 objects
drwxr-xr-x 4 hacker hacker 4096 2012-01-19 08:14 refs
```

`git add`+要跟踪文件名，可以看到这里多了一个`index`文件，这就是那个staging area。

再次运行`git status`

```
hacker@ubuntu:~/mygit/ggit$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   main.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```

可以看到`main.c`文件已经被跟踪，并处于暂存状态。

三：提交

然后再`git commit`进行提交，把数据提交到`git`仓库中：

`$git commit -m "this is first commit"`

注意这里通过`-m` 选项加一个注释，这样你就可以提交你的数据到`git`仓库了，也可以把两个步骤合并为一个步骤：

`$git commit -a -m "this is first commit"`

`git`有两个配置文件，一个在`$HOME`下，是全局的，设置时加`--global`，另一个在仓库配置文件里。

`$git config`

设置全局的：

```
hacker@ubuntu:~/mygit/aa$ git config --global user.name 'kernellwp'
hacker@ubuntu:~/mygit/aa$ git config --global user.email 'kernellwp@gmail.com'
hacker@ubuntu:~/mygit/aa$ cat /home/hacker/.gitconfig
[user]
    name = kernellwp
    email = kernellwp@gmail.com
```

本地的：

```
hacker@ubuntu:~/mygit/aa$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
hacker@ubuntu:~/mygit/aa$ git config user.name kernellwp
hacker@ubuntu:~/mygit/aa$ git config user.email kernellwp@gmail.com
hacker@ubuntu:~/mygit/aa$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[user]
    name = kernellwp
    email = kernellwp@gmail.com
```

Git使用方法(二)

一：使用`.gitignore`忽略某些文件

文件`.gitignore`的格式规范如下：

- 所有空行或以注释符号`#`开头的行都会被`Git`忽略
- 可以使用标准的`glob`模式匹配
- 匹配模式最后跟反斜杠`(/)`说明要忽略的是目录
- 要忽略指定模式以外的文件或目录，可以在模式前加惊叹号`(!)`取反

一般时候我们总会有一些文件无需纳入`Git`的管理，也不希望他们总出现在未跟踪文件列表。通常都是些自主生成的文件，像是日志或者

编译过程中创建的等等。我们可以创建一个`.gitignore`的文件，列出要忽略的文件模式，来看一个例子：

```
hacker@ubuntu:~/mygit/ggit$ cat .gitignore
#ignore .test
#.test
.gitignore
#ignore obj and lib file
*[oa]
```

这里忽略了工作目录下的.test文件, 以.o或.a结尾的文件。

```
hacker@ubuntu:~/mygit/ggit$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .test
nothing added to commit but untracked files present (use "git add" to track)
```

如果不使用.gitignore, 可以看到, 显示.test文件未跟踪, 下面添加.gitignore文件

```
hacker@ubuntu:~/mygit/ggit$ git status
# On branch master
nothing to commit (working directory clean)
```

可以看到工作目录很干净的。

如果使用git下载Linux内核源码, 你可以在工作目录下看到.gitignore文件,
cat .gitignore, 截了一段:

```
*.lzo
*.patch
*.gcno

#
# Top-level generic files
#
/tags
/TAGS
/linux
/vmlinux
/vmlinuz
/System.map
/Module.markers
/Module.symvers

#
# Debian directory (make deb-pkg)
#
/debian/

#
# git files that we don't want to ignore even if they are dot-files
#
!.gitignore
!.mailmap

#
# Generated include files
#
/include/config
/include/linux/version.h
/include/generated
/arch/*/.include/generated

# stgit generated dirs
patches-*

# quilt's files
patches
series

# cscope files
cscope.*
ncscope.*

# gnu global files
GPATH
GRTAGS
GSYMS
```

二: git diff 生成patch

git-diff列出自己本地的tree中已修改, 但却未commit的改动, 这也是产生patch的方式。
注意, 使用git-diff产生的patch都应该在patch(1)时

指定-p1, 或者直接使用git-apply打补丁。

```
hacker@ubuntu:~/mygit/ggit$ git diff
hacker@ubuntu:~/mygit/ggit$ echo "11" > main.c
hacker@ubuntu:~/mygit/ggit$ git diff
diff --git a/main.c b/main.c
index 65151e8..b4de394 100644
--- a/main.c
+++ b/main.c
@@ -1,7 +1 @@
-1
-2
-3
-4
-5
-6
-aa
+11
```

选项:

- color diff 语法高亮
- ignore-space-at-eol 忽略行尾的whitespace
- ignore-space-change 忽略行尾的whitespace, 并且认为所有的whitespace都是一样的
- ignore-all-space 比较两行的时候, 完全忽略whitespace。这样, 即使是一行有很多 whitespaces, 另一行文字一样但是没有whitespace, git也认为这两行内容一致。

```
hacker@ubuntu:~/mygit/ggit$ git branch
* master
  testing
hacker@ubuntu:~/mygit/ggit$ cat main.c test.c
1
2
3
4
5
6
aa
this is 1
hacker@ubuntu:~/mygit/ggit$ git checkout testing
Switched to branch 'testing'
hacker@ubuntu:~/mygit/ggit$ cat main.c test.c
1
2
3
4
5
6
ss
hello
1
2
3
4
this is last
this is 2
hacker@ubuntu:~/mygit/ggit$ git diff --color master testing
diff --git a/main.c b/main.c
index 65151e8..30b7b80 100644
--- a/main.c
+++ b/main.c
@@ -4,4 +4,10 @@
-4
-5
-6
-aa
+ss
+hello
+1
+2
+3
+4
+this is last
diff --git a/test.c b/test.c
index 19d6416..c825d65 100644
--- a/test.c
+++ b/test.c
@@ -1 +1 @@
-this is 1
+this is 2
```

这里介绍一下patch文件格式: 补丁头

补丁头是分别由---/+++开头的两行, 用来表示要打补丁的文件。---开头表示旧文件, +++开头表示新文件。

一个补丁文件中的多个补丁

一个补丁文件中可能包含以---/+++开头的很多节, 每一节用来打一个补丁。所以在一个

补丁文件中可以有好多个补丁。

块

块是补丁中要修改的地方。它通常由一部分不用修改的东西开始和结束。他们只是用来表示要修改的位置。他们通常以@@开始，结束于另一个块的开始或者一个新的补丁头。

块的缩进

块会缩进一列，而这一列是用来表示这一行是要增加还是要删除的。

块的第一列

+号表示这一行是要加上的

-号表示这一行是要删除的

没有加号也没有减号表示这里只是引用的而不需要修改

三: git apply打补丁

git-apply相当于patch(1)命令，不过git-apply专门用来apply那些用git-diff生成的补丁选项：

--check 不真正打补丁，而只是检查补丁是否能完美的打上

-v verbose模式

-R reverse模式，也就是拉出这个补丁来

```

hacker@ubuntu:~/mygit/ggit$ git diff master testing
diff --git a/main.c b/main.c
index 65151e8..30b7b80 100644
--- a/main.c
+++ b/main.c
@@ -4,4 +4,10 @@
 4
 5
 6
-aa
+ss
+hello
+1
+2
+3
+4
+this is last
diff --git a/test.c b/test.c
index 19d6416..c825d65 100644
--- a/test.c
+++ b/test.c
@@ -1 +1 @@
-this is 1
+this is 2
hacker@ubuntu:~/mygit/ggit$ git diff master testing > testing.patch
hacker@ubuntu:~/mygit/ggit$ git branch
* master
  testing
hacker@ubuntu:~/mygit/ggit$ git apply testing.patch
hacker@ubuntu:~/mygit/ggit$ cat main.c
1
2
3
4
ss
hello
1
2
3
4
this is last
hacker@ubuntu:~/mygit/ggit$ cat test
cat: test: No such file or directory
hacker@ubuntu:~/mygit/ggit$ cat test.c
this is 2

```

四: git log 查看提交历史记录

运行git log:

```

hacker@ubuntu:~/mygit/ggit$ git log
commit 02945438214ff48cef8ec7cffd394818975ae2c
Author: kernellwp <kernellwp@gmail.com>
Date:   Mon Jan 23 17:45:58 2012 -0800

    fuck

commit b461b3096a355a8fd40cd211f646abc884e07892
Author: kernellwp <kernellwp@gmail.com>
Date:   Mon Jan 23 06:55:43 2012 -0800

    fuck

commit 3311dd387459970b3462b5a1039d1ef6b2807bff
Author: kernellwp <kernellwp@gmail.com>
Date:   Thu Jan 19 18:50:48 2012 -0800

    test

commit 51f9858092dc49ab12d5ab50d54d1499a0f7a5d2
Author: kernellwp <kernellwp@gmail.com>
Date:   Thu Jan 19 02:34:47 2012 -0800

    haha

commit f9bf59a10fa5c74a25346549e616b1250336e698
Author: kernellwp <kernellwp@gmail.com>
Date:   Thu Jan 19 02:27:16 2012 -0800

    modified

commit 41dc862f121ed768a4b623d3fcffae97acb9d5f3
Author: kernellwp <kernellwp@gmail.com>
Date:   Thu Jan 19 01:10:05 2012 -0800

    first commit

```

git会按提交时间列出所有的更新，最近的更新排在最上边。每次更新都会有一个SHA-1校验，作者的名字和电子邮件地址，提交时间，最后一个段落显示提交说明。

选项说明：

-p 按补丁格式显示每个更新之间的差异

--stat 显示每次更新的文件修改统计信息

--shortstat 只显示--stat中最后的行数修改添加移除统计

--pretty 使用其他格式显示历史提交信息。可用的选项包括**online**, **short**, **full**, **fuller**和**format**。

下边是加-p选项，按补丁格式查看每个更新之间的差异

\$git log -p

```

Author: kernellwp <kernellwp@gmail.com>
Date: Thu Jan 19 02:34:47 2012 -0800

    haha

diff --git a/main.c b/main.c
index 8b1072a..b414108 100644
--- a/main.c
+++ b/main.c
@@ -4,7 +4,3 @@
 4
 5
 6
-aa
-bb
-cc
-dd

commit f9bf59a10fa5c74a25346549e616b1250336e698
Author: kernellwp <kernellwp@gmail.com>
Date: Thu Jan 19 02:27:16 2012 -0800

    modified

diff --git a/main.c b/main.c
index b414108..8b1072a 100644
--- a/main.c
+++ b/main.c
@@ -4,3 +4,7 @@
 4
 5
 6
+aa
+bb
+cc
+dd

commit 41dc862f121ed768a4b623d3fcffae97acb9d5f3
Author: kernellwp <kernellwp@gmail.com>
Date: Thu Jan 19 01:10:05 2012 -0800

    first commit

diff --git a/main.c b/main.c
new file mode 100644
index 0000000..b414108
--- /dev/null
+++ b/main.c
@@ -0,0 +1,6 @@
+1
+2
+3
+4
+5
+6

```

Git使用方法(三)

在Git中提交时，会保存一个提交(commit)对象，它包含一个指向暂存内容快照的指针，作者和相关附属信息，以及一定数量(也可能没有)指向

提交对象直接祖先的指针：第一次提交是没有直接祖先的，普通提交有一个祖先，由两个或多个分支合并产生的提交则有多个祖先。现在假设

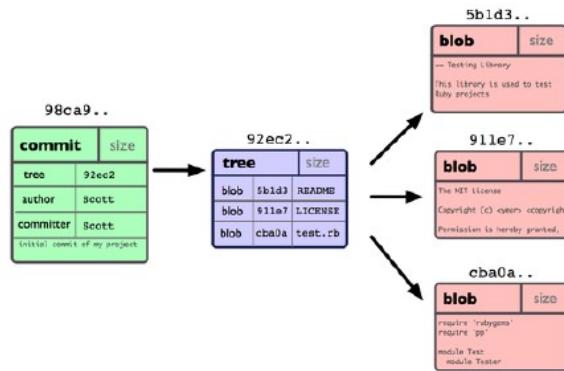
工作目录下有3个文件，准备将他们暂存后提交。暂存操作会对每一个文件计算校验和(即SHA-1哈希字符串)，然后把当前版本控制的文件快照

保存到Git仓库中，并将校验和加入暂存区域。当使用git commit新建一个提交对象前，Git会先计算每一个子目录(本例中就是项目根目录)

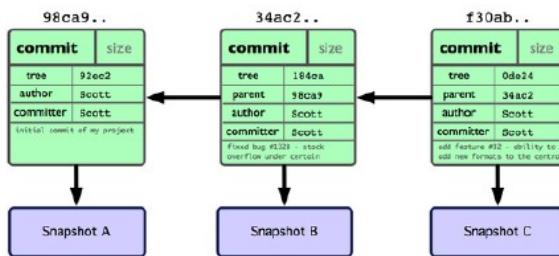
的校验和，然后在Git仓库中将这些目录保存为树(tree)对象。之后Git创建的提交对象

(commit)，除了包含相关提交信息以外，还包含着指向这

个树对象(项目根目录)的指针，如此他就可以在将来需要的时候，重现此次快照的内容。



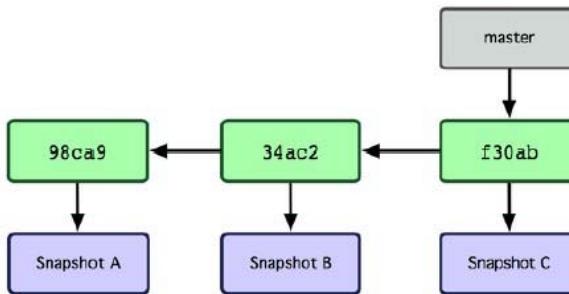
这个是提交一次后仓库里的数据



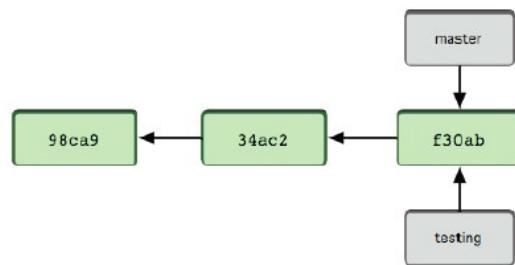
多次提交后Git仓库数据

Git中的分支，其实本质就是个指向commit对象的可变指针。Git会使用master作为分支的默认名字。在若干次提交后，你其实已经有了一个指向最后一次提交对象的master分支，它在每次提交的时候都会自动向前移动。

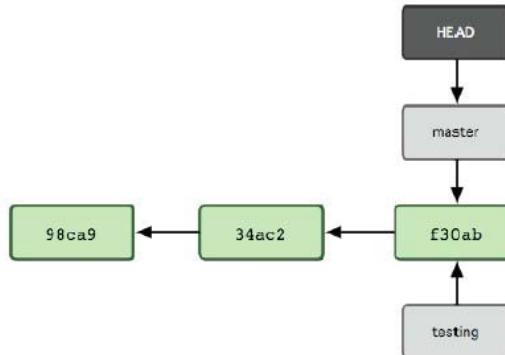
一: git branch



经过多次提交仓库中的情况如上，master指向最新的commit，那么怎样创建一个新的分支呢，可以使用git branch+分知名，这里用git branch testing，就创建了一个基于master的分支。
\$git branch testing



HEAD是一个引用，指向正在使用的分支。

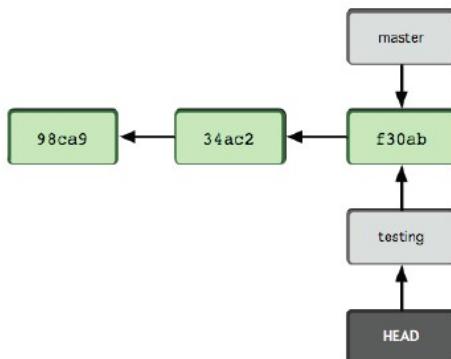


进行查看：

```

hacker@ubuntu:~/mygit/fuck$ cat .git/HEAD
ref: refs/heads/master
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/master
983129d608fb98fb775fb29e568364cc85fa82fd
hacker@ubuntu:~/mygit/fuck$ git cat-file -t 9831
commit
  
```

下面切换到新的分支：



\$git checkout testing

```

hacker@ubuntu:~/mygit/fuck$ git branch
* master
hacker@ubuntu:~/mygit/fuck$ git branch testing
hacker@ubuntu:~/mygit/fuck$ git branch
* master
  testing
hacker@ubuntu:~/mygit/fuck$ git checkout testing
Switched to branch 'testing'
hacker@ubuntu:~/mygit/fuck$ git branch
  master
* testing
  
```

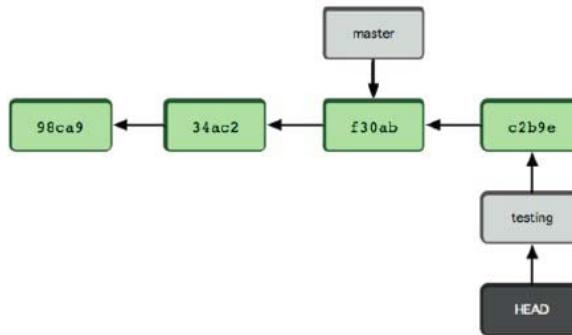
使用git branch可以查看当前都有哪些branch，前面有*的表示，当前所在的branch。

```

hacker@ubuntu:~/mygit/fuck$ git branch
  master
* testing
hacker@ubuntu:~/mygit/fuck$ cat .git/HEAD
ref: refs/heads/testing
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/testing
983129d608fb98fb775fb29e568364cc85fa82fd
hacker@ubuntu:~/mygit/fuck$ git checkout master
Switched to branch 'master'
hacker@ubuntu:~/mygit/fuck$ git branch
* master
  testing
hacker@ubuntu:~/mygit/fuck$ cat .git/HEAD
ref: refs/heads/master
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/master
983129d608fb98fb775fb29e568364cc85fa82fd
hacker@ubuntu:~/mygit/fuck$ git cat-file -t 9831
commit

```

可以看到,此时master与testing都指向同一个commit, 这里的9831*****，一共是20个字节，40个十六进制的字符，这个就是将文件放到暂存区时产生的哈希字符串，它作为文件的名字，这里使用git cat-file -t 查看object的类型，这里至少用前四个字符。

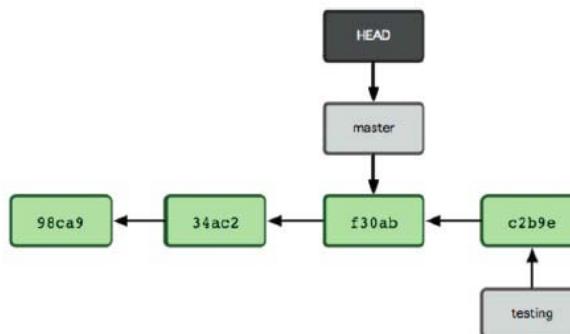


现在切换到testing分支, 修改工作目录下的main.c文件, 然后提交, 再次看HEAD的指向, 发现testing中的HEAD已经指向新的commit了。

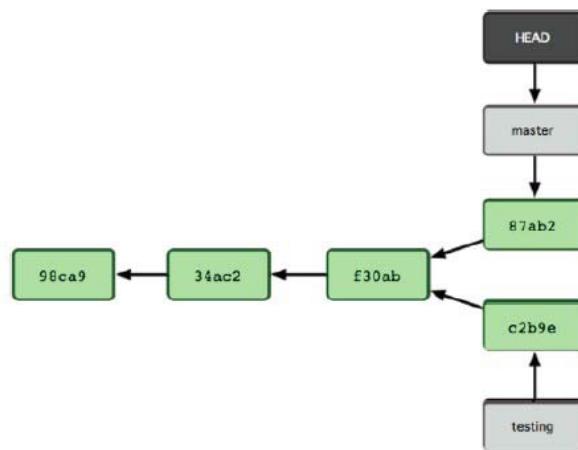
```

hacker@ubuntu:~/mygit/fuck$ git branch
  master
* testing
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/testing
746a2bb6fec0852e90719eeef3885651223adafad
hacker@ubuntu:~/mygit/fuck$ git checkout master
Switched to branch 'master'
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/master
983129d608fb98fb775fb29e568364cc85fa82fd

```



现在对master分支的main.c文件进行修改:



```

hacker@ubuntu:~/mygit/fuck$ git branch
* master
  testing
hacker@ubuntu:~/mygit/fuck$ echo "this branch is master" >> main.c
hacker@ubuntu:~/mygit/fuck$ git commit -a -m "this is master"
[master 1d15035] this is master
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/master
1d150354d172819f7f7851f5ef27042e3c7b9973
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/testing
746a2bb6fec0852e90719eef3885651223adafad

```

可以看到此时master指向的commit已经变化，形成上图的分支。

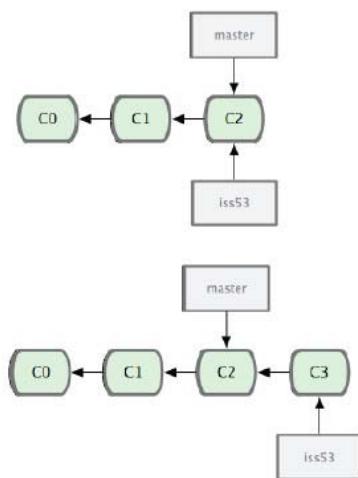
git branch -a 列出所有分支，包括remote和local branches

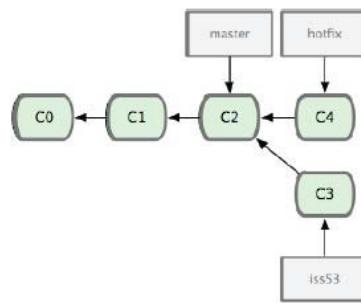
git branch -r 列出remote branches

git branch -d new-branch 删除new-branch

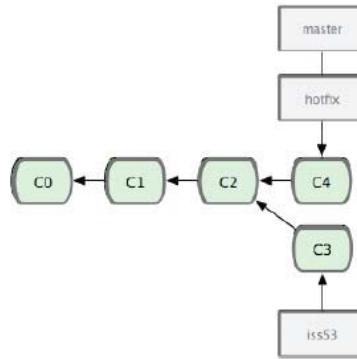
git branch -D new-branch 强制删除new-branch

二: git merge





从上边几张图可以看出先是基于master创建了分支iss53，此时它们指向同一个commit，之后iss32分支进行了commit，然后基于master创建了新的分支hotfix，并进行了commit。现在要把hotfix分支合并到master分支中。



git merge + 要merge的branch，这样就可以把branch merge到当前branch上了。先git checkout master，然后git merge hotfix，这样hotfix branch就可以merge到master分支上了，然后git branch -d hotfix对hotfix分支进行删除。这里的merge其实是比较简单的，由于master分支指向的commit是hotfix分支指向的commit的parent，所以直接移动master指针到hotfix指向的commit就可以了。

```

hacker@ubuntu:~/mygit/fuck$ git branch
  hotfix
  iss53
* master
  testing
hacker@ubuntu:~/mygit/fuck$ git merge hotfix
Updating 1d15035..1000abd
Fast-forward
 main.c |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/fuck$ cat main.c
this is main.c
this branch is master
this is hotfix branch

```

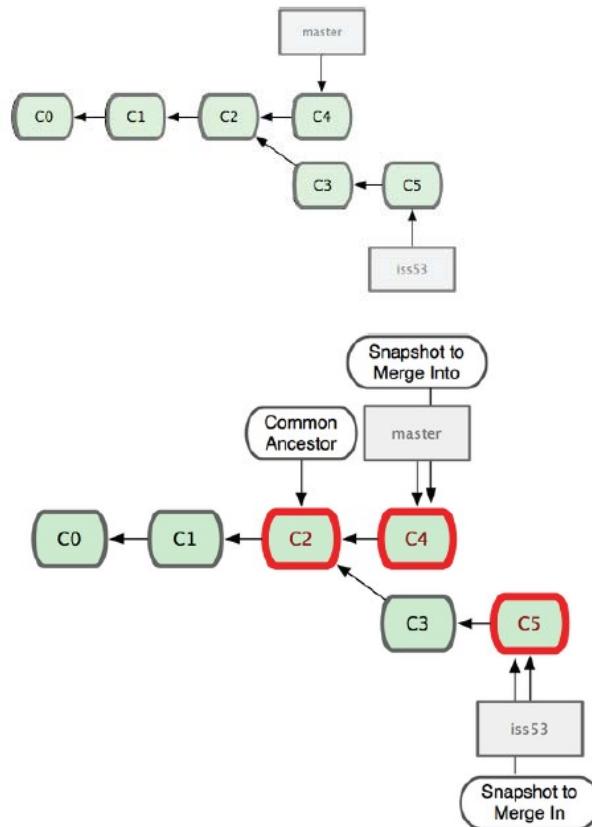
git branch --merged 查看哪些分支已被并入当前分支
git branch --no-merged 查看哪些分支没有被并入当前分支

```

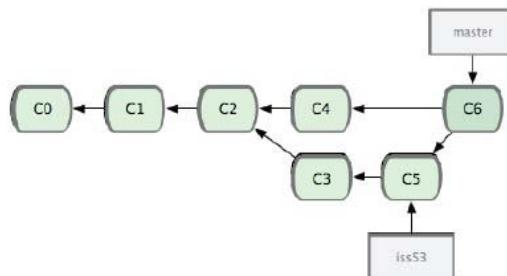
hacker@ubuntu:~/mygit/fuck$ git branch
  hotfix
  iss53
* master
  testing
hacker@ubuntu:~/mygit/fuck$ git branch --merged
  hotfix
* master
hacker@ubuntu:~/mygit/fuck$ git branch --no-merged
  iss53
  testing

```

下面这个是three-way merge



这里要merge c4和c5，此时Git会用两个分支的末端(C4和C5)和他们的共同祖先(C2)进行一次简单的三方合并计算。Git可以自己裁决哪个公共祖先才是最佳合并基础。



三: git checkout

git checkout branch-name 切换到branch-name
 git checkout master 切换到master
 git checkout -b new-branch master 从master建立新的new-branch, 并同时切换过去
 new-branch
 git checkout -b newbranch 由现在的分支为基础, 建立新的branch
 git checkout -b newbranch origin 由origin的基础, 建立新的branch
 git checkout filename 还原档案到Repository状态
 git checkout HEAD 将所有档案都checkout出来(最后一次commit的版本), 注意, 若有
 修改的档案都会被还原到上一版
 git checkout xxxx 将所有档案都checkout出来(xxxx commit的版本, xxxx是commit的
 编号前四位), 注意, 若有修改的档案
 都会被还原到上一版

四: git show

可以使用git show加上commit名称来显示更详细的commit信息:

```
hacker@ubuntu:~/mygit/fuck$ cat .git/refs/heads/master
1000abd76dea3c2ce1f4a245850f6229a4a12a0e
hacker@ubuntu:~/mygit/fuck$ git show 1000
commit 1000abd76dea3c2ce1f4a245850f6229a4a12a0e
Author: kernellwp <kernellwp@gmail.com>
Date: Tue Jan 24 07:02:26 2012 -0800

    this is hotfix branch

diff --git a/main.c b/main.c
index 4f7856a..effea91 100644
--- a/main.c
+++ b/main.c
@@ -1,2 +1,3 @@
    this is main.c
    this branch is master
+this is hotfix branch
```

也可以使用git show加分支名称, 也可显示分支信息:

```
hacker@ubuntu:~/mygit/fuck$ git show master
commit 1000abd76dea3c2ce1f4a245850f6229a4a12a0e
Author: kernellwp <kernellwp@gmail.com>
Date: Tue Jan 24 07:02:26 2012 -0800

    this is hotfix branch

diff --git a/main.c b/main.c
index 4f7856a..effea91 100644
--- a/main.c
+++ b/main.c
@@ -1,2 +1,3 @@
    this is main.c
    this branch is master
+this is hotfix branch
hacker@ubuntu:~/mygit/fuck$ git show testing
commit 746a2bb6fec0852e90719eef3885651223adafad
Author: kernellwp <kernellwp@gmail.com>
Date: Tue Jan 24 05:28:44 2012 -0800

    testing

diff --git a/main.c b/main.c
index 38a3fb4..86d4864 100644
--- a/main.c
+++ b/main.c
@@ -1 +1,2 @@
    this is main.c
+this branch is testing
```

使用HEAD字段可以代表当前分支的头(也就是最近一次commit):

\$git show HEAD

每一次commit都会有“parent commit”，可以使用`^`表示parent:
`$git show HEAD^` 查看HEAD的父母的信息
`$git show HEAD^^` 查看HEAD的父母的父母的信息
`$git show HEAD~5` 查看HEAD上溯5代的信息
 有的时候git merge会产生双父母，比如three-way merge的时候，这种情况这样处理：
`$git show HEAD^1` 查看HEAD的第一个父母
`$git show HEAD^2` 查看HEAD的第二个父母

五: git archive

可以把当前版本(HEAD所处的位置)给export出来

使用git describe可以查看当前的version

```
$mkdir ..linux-2.6.11
$git archive -v v2.6.11 | (cd ..linux-2.6.11/ && tar xf -)
$head -4 ..linux-2.6.11/Makefile
hacker@ubuntu:~/mygit/ggit/linux-kernel$ head -4 ..linux-2.6.11/Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 11
EXTRAVERSION =
```

从本地git仓库中提取某个版本的kernel:

`$git archive -v v2.6.11 | (cd ..linux-2.6.11/ && tar xf -)`

-v表示--verbose，注意'v2.6.11'可以是git tag -l列出的tags中的一个，也可以是其他Rev ID例如HEAD等

这里的“`&&`”类似“；”不过是有区别的，如果每个命令被一个分号“；”所分隔，那么命令会连续的执行下去。如果每个命令

被“`&&`”号分隔，那么这些命令会一直执行下去，如果中间有错误的命令存在，则不再执行后面的命令，没错则执行到完为止。

这里的“`-`”作用是把前边的输出作为这里的输入

导出最新的kernel:

`$git archive -v HEAD | (cd ..linux-HEAD/ && tar xf -)`

或者打成tar包:

`$git archive -v --format=tar v3.0 | bzip2 >..linux-3.0.tar.bz2`

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ ls ...
linux-2.6.11 linux-3.0.tar.bz2 linux-kernel main.c test.c testing.patch
```

Git使用方法(四)

一: git remote

远程仓库是指托管在网络上的项目仓库，可能会有好多个，其中有些你只能读，另外有些可以写。同他人协作开发某个项目时，需要管理这些远程仓库，以便推送或拉取数据，所以在克隆完某个项目后，至少可以看到一个名为origin的远端库，Git默认使用这个名字来标识你所克隆的原始仓库：

`$git remote`

`origin`

也可以加上-v选项(-v为verbose的缩写，取首字母)，显示对应的克隆地址：

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ git remote -v
origin git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git (fetch)
origin git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git (push)
```

添加远程仓库，可以指定一个简单的名字，以便将来引用，运行`git remote add`

[shortname] [url]:

```
$ git remote
origin
$ git remote add pb git://github.com/paulboone/ticgit.git
$ git remote -v
origin git://github.com/schacon/ticgit.git
pb git://github.com/paulboone/ticgit.git
```

使用`git remote show [remote-name]`查看某个远程仓库的详细信息:

```
$ git remote show origin
* remote origin
  URL: git@github.com:defunkt/github.git
  Remote branch merged with 'git pull' while on branch issues
    issues
  Remote branch merged with 'git pull' while on branch master
    master
  New remote branches (next fetch will store in remotes/origin)
    caching
  Stale tracking branches (use 'git remote prune')
    libwalker
    walker2
  Tracked remote branches
    acl
    apiv2
    dashboard2
    issues
    master
    postgres
  Local branch pushed with 'git push'
    master:master
```

它告诉我们, 运行`git push`时缺省推送的分支是什么(最后两行)。它还显示了有哪些远程分支还没有同步到本地(第六行的`caching`分支), 哪些已同步到本地的远端分支在远程服务器上。

远程仓库的删除和重命名

可以用`git remote rename`命令修改某个远程仓库的简短名称, 比如想把AA改成BB, 可以这么运行:

```
$git remote AA BB
```

```
$git remote
```

```
origin
```

```
BB
```

移除远程仓库`git remote rm`命令:

```
$git remote rm BB
```

```
$git remote
```

```
origin
```

二: git fetch

从远程仓库抓取数据, 使用`git fetch [remote-name][branch-name]`。

三: git pull

从远程仓库分支中获得更新,git push [remote-name] [branch-name], pull命令包含两个操作：从远端分支中取出改动，然后合并到当前分支中。相当于git fetch+ git merge。

当git clone之后，直接git pull它会自动匹配一个正确的remote url，因为在.git/config文件里配置了[branch "master"]下面的内容：

```
hacker@ubuntu:~/mygit/ggit/linux-kernels$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
[branch "master"]
    remote = origin
    merge = refs/heads/master
```

1. git处于master这个branch下时，默认的remote就是origin
2. 当在master这个branch下使用指定remote和merge的git pull时，使用默认的remote和merge

\$git pull

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ git pull
remote: Counting objects: 1306, done.
remote: Compressing objects: 100% (316/316), done.
remote: Total 791 (delta 678), reused 543 (delta 474)
Receiving objects: 100% (791/791), 104.76 KiB | 105 KiB/s, done.
Resolving deltas: 100% (678/678), completed with 238 local objects.
From git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6
  claaab02..f8275f9 master      -> origin/master
Updating claaab02..f8275f9
Fast-forward
 Documentation/thermal/sysfs-api.txt          |  2 ++
 arch/powerpc/boot/dts/fsl/mpc8536si-post.dtsi |  4 +
 arch/powerpc/boot/dts/fsl/p1010si-post.dtsi   |  3 ++
 arch/powerpc/boot/dts/fsl/p1020si-post.dtsi   |  4 +
 arch/powerpc/boot/dts/fsl/p1022si-post.dtsi   |  3 ++
 arch/powerpc/boot/dts/fsl/p2020si-post.dtsi   |  4 +
 arch/powerpc/boot/dts/p1020rdb.dtsi           | 13 ++
 arch/powerpc/boot/dts/p1021mds.dts             |  3 ++
 arch/powerpc/boot/dts/p2020ds.dtsi             |  3 ++
 arch/powerpc/boot/dts/p2020rdb.dts             |  3 ++
 arch/powerpc/kernel/crash.c                   |  2 ++
 arch/powerpc/kernel/legacy_serial.c           |  2 +
 arch/powerpc/platforms/85xx/p1022_ds.c       |  1 +
 arch/powerpc/platforms/powernv/pci-ioda.c     | 43 ++
 arch/powerpc/platforms/pseries/Kconfig        |  2 ++
 arch/powerpc/sysdev/fsl_pci.c                 |  5 ++
 arch/x86/net/bpf_jit_comp.c                  | 36 ++
 drivers/acpi/Makefile                         |  1 -
 drivers/acpi/apei/apei-base.c                | 35 +--
 drivers/acpi/apei/einj.c                      | 95 ++++++-
 drivers/acpi/atomicio.c                       | 422 -----
 drivers/acpi/osl.c                           | 152 +++++++-
 drivers/acpi/processor_driver.c               | 154 ++++++-
 drivers/acpi/sleep.c                          |  8 +
 drivers/gpu/drm/gma500/gtt.c                 |  5 ++
 drivers/idle/intel_idle.c                    |  2 ++
 drivers/net/bonding/bond_alb.c               |  2 ++
 drivers/net/dsa/mv88e6060.c                  |  1 +
```

四：git push

如果把本地的master分支推送到origin服务器上(), 可以运行下面命令:推送数据到远程仓库使用git push [remote-name] [branch-name],

```
$git push origin master
```

只有在所克隆的服务器上有写权限, 或者同一时刻没有其他人在推数据, 这条命令才会如期完成任务。如果你在推数据前, 已经有其他人推送了若干更新, 那你的推送操作就会被

五: git reset

使用git reset撤销改动, git reset HEAD^删除最近的commit, 选项:

--mixed staged与committed都会清除, 但是modified不会清除

--hard modified, staged, committed都会清除

--soft committed被清除, modified, staged还在

reset是将当前head的内容重置, 不留任何痕迹。

Sets the current head to the specified commit and optionally resets the index and working tree to match.

git reset --hard HEAD~4

会将最新的4次提交全部重置, 就像没有提交过一样。

使用--mixed选项, 如果你不添加选项, 默认使用的是--mixed选项:

```
hacker@ubuntu:~/mygit/test$ echo "this is second" >> main.c
hacker@ubuntu:~/mygit/test$ git commit -a -m "second"
[master 840f768] second
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/test$ git reset HEAD^
Unstaged changes after reset:
M     main.c
hacker@ubuntu:~/mygit/test$ cat main.c
this is git
this is second
hacker@ubuntu:~/mygit/test$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   main.c
#
no changes added to commit (use "git add" and/or "git commit -a")
hacker@ubuntu:~/mygit/test$ git log
commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
```

这里的“changed but not updated”表示文件被modify, 但是没有stage, 你有两个选择, 一个是stage, git add<file>..., 另一个选择是撤销这次修改git checkout --<file>...

使用--hard选项:

```

hacker@ubuntu:~/mygit/test$ git commit -a -m "second"
[master 6fcfcfd] second
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/test$ git log
commit 6fcfcfd32632398a42bb35ac88b21c8c3980afa8a
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:02:30 2012 -0800

    second

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
hacker@ubuntu:~/mygit/test$ git reset --hard HEAD^
HEAD is now at 684cc14 first
hacker@ubuntu:~/mygit/test$ git log
commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
hacker@ubuntu:~/mygit/test$ cat main.c
this is git
hacker@ubuntu:~/mygit/test$ git status
# On branch master
nothing to commit (working directory clean)

```

使用--soft选项:

```

hacker@ubuntu:~/mygit/test$ echo "this is second" >> main.c
hacker@ubuntu:~/mygit/test$ git commit -a -m "second"
[master db50144] second
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/test$ git status
# On branch master
nothing to commit (working directory clean)
hacker@ubuntu:~/mygit/test$ git log
commit db501448d9e71bb88a44bde82d19e2d0d747e81f
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:04:29 2012 -0800

    second

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
hacker@ubuntu:~/mygit/test$ git reset --soft HEAD^
hacker@ubuntu:~/mygit/test$ cat main.c
this is git
this is second
hacker@ubuntu:~/mygit/test$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   main.c
#
hacker@ubuntu:~/mygit/test$ git log
commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first

```

这里的“Changes to be committed”表示已经modified并且stage，但是没有commit，你可以使用git reset HEAD <file>...进行unstage，或者使用git commit进行commit。

有的时候想要修改提交信息使用git commit --amend

此命令将使用当前的暂存区快照提交。如果刚才提交完没有作任何改动，直接运行此命令的话，相当于有机会重新编辑提交说明，而所提交的文件快照和之前的一样。

```

hacker@ubuntu:~/mygit/test$ echo "this is second" >> main.c
hacker@ubuntu:~/mygit/test$ git commit -a -m "second"
[master 34d7b97] second
 1 files changed, 2 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/test$ git log
commit 34d7b975fb9576867b2eeea4e6adf117ae6c79b7
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:38:27 2012 -0800

    second

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
hacker@ubuntu:~/mygit/test$ git commit -amend
error: did you mean `--amend` (with two dashes ?)
hacker@ubuntu:~/mygit/test$ git commit --amend
[master d10c883] this is second commit
 1 files changed, 2 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/test$ git log
commit d10c883559a9aa823387751901a32929e28d6509
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:38:27 2012 -0800

    this is second commit

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first

```

这里将之前的commit信息“second”改为“this is second commit”。

在git中，除非你运行了git-gc --prune，否则历史是永远不会被擦除的，你可以随意恢复到任何历史状态。下面是一个恢复被git-reset --hard擦除了的commit的例子：

使用git-reflog查看历史：

```

hacker@ubuntu:~/mygit/test$ git log
commit ccf8c4ed4e3847ab9730c8eb8a3812492d6e844f
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:46:22 2012 -0800

    Revert "this is last commit"

    This reverts commit d10c883559a9aa823387751901a32929e28d6509.

commit d10c883559a9aa823387751901a32929e28d6509
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:38:27 2012 -0800

    this is second commit

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
hacker@ubuntu:~/mygit/test$ git reset --hard HEAD
HEAD is now at ccf8c4e Revert "this is last commit"
hacker@ubuntu:~/mygit/test$ git reflog
git: 'relog' is not a git command. See 'git --help'.

Did you mean this?
  reflog
hacker@ubuntu:~/mygit/test$ git reflog
ccf8c4e HEAD@{0}: commit: Revert "this is last commit"
d10c883 HEAD@{1}: commit (amend): this is second commit
34d7b97 HEAD@{2}: commit: second
684cc14 HEAD@{3}: HEAD^: updating HEAD
db50144 HEAD@{4}: commit: second
684cc14 HEAD@{5}: HEAD^: updating HEAD
6fcfcfd3 HEAD@{6}: commit: second
684cc14 HEAD@{7}: HEAD^: updating HEAD
840f768 HEAD@{8}: commit: second
hacker@ubuntu:~/mygit/test$ clear

hacker@ubuntu:~/mygit/test$ git reset --hard HEAD@{0}
HEAD is now at ccf8c4e Revert "this is last commit"

```

六: git revert

git revert只是修改了commit, 修改后再次提交。

git revert与git reset的区别:

- git reset是还原到指定的版本上, 这将扔掉指定版本之后的版本
- git revert是提交一个新的版本将需要revert的版本的内容再反向修改回去, 版本会递增

```
hacker@ubuntu:~/mygit/test$ git log
commit ccf8c4ed4e3847ab9730c8eb8a3812492d6e844f
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:46:22 2012 -0800

    Revert "this is last commit"

    This reverts commit d10c883559a9aa823387751901a32929e28d6509.

commit d10c883559a9aa823387751901a32929e28d6509
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 01:38:27 2012 -0800

    this is second commit

commit 684cc14fed61ac9abfc904644de4abadde8be640
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 00:56:01 2012 -0800

    first
```

Git使用方法(五)

一: git tag

git tag列出已有的标签:

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ git tag
v2.6.11
v2.6.11-tree
v2.6.12
v2.6.12-rc2
v2.6.12-rc3
v2.6.12-rc4
v2.6.12-rc5
v2.6.12-rc6
v2.6.13
v2.6.13-rc1
v2.6.13-rc2
v2.6.13-rc3
v2.6.13-rc4
v2.6.13-rc5
v2.6.13-rc6
v2.6.13-rc7
v2.6.14
v2.6.14-rc1
v2.6.14-rc2
v2.6.14-rc3
v2.6.14-rc4
v2.6.14-rc5
v2.6.15
v2.6.15-rc1
v2.6.15-rc2
v2.6.15-rc3
v2.6.15-rc4
v2.6.15-rc5
v2.6.15-rc6
v2.6.15-rc7
```

使用特定的搜索模式列出匹配的标签:

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ git tag -l 'v3.0*'
v3.0
v3.0-rc1
v3.0-rc2
v3.0-rc3
v3.0-rc4
v3.0-rc5
v3.0-rc6
v3.0-rc7
```

获得某个版本的源码:

```
$mkdir ..linux-2.6.11
```

```
$git archive -v v2.6.11 | (cd ..linux-2.6.11/ && tar xf -)
```

```
$head -4 ..linux-2.6.11/Makefile
```

```
hacker@ubuntu:~/mygit/ggit/linux-kernel$ head -4 ..linux-2.6.11/Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 11
EXTRAVERSION =
```

git使用的标签有两种类型: 轻量级的(*lightweight*)和含注释的(*annotated*)。新建含注释的标签会产生一个标签对象, 而轻量级标签不会产生标签对象。

轻量级的标签建立:

```
$git tag v1.0
```

这样就会给当前的commit打上v1.0这个标签。

```
hacker@ubuntu:~/mygit$ mkdir tag
hacker@ubuntu:~/mygit$ cd tag/
hacker@ubuntu:~/mygit/tag$ git init
Initialized empty Git repository in /home/hacker/mygit/tag/.git/
hacker@ubuntu:~/mygit/tag$ echo "1" > main.c
hacker@ubuntu:~/mygit/tag$ git add .
hacker@ubuntu:~/mygit/tag$ git commit -m "1"
[master (root-commit) 704ffff] 1
 1 files changed, 1 insertions(+), 0 deletions(-)
  create mode 100644 main.c
hacker@ubuntu:~/mygit/tag$ git tag v1.0
hacker@ubuntu:~/mygit/tag$ ls .git/refs/tags/
v1.0
```

此时这个tag是一个引用, 不是对象。

含注释的标签建立:

```
$ git tag -a [name] -m ["xxxx"]
```

```

hacker@ubuntu:~/mygit/tag$ echo "2" >> main.c
hacker@ubuntu:~/mygit/tag$ git commit -a -m "2"
[master dfff3ea] 2
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/tag$ git tag -a v2.0 -m "this is tag v2.0"
hacker@ubuntu:~/mygit/tag$ git log
commit dfff3ea5e5defe84b5a7272ffe61f0e139537bfd
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 04:51:43 2012 -0800

2

commit 704fff4ef0e7e2726d6c3f3b74350cf3c2002efa
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 04:50:01 2012 -0800

1
hacker@ubuntu:~/mygit/tag$ git show v2.0
tag v2.0
Tagger: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 04:52:03 2012 -0800

this is tag v2.0

commit dfff3ea5e5defe84b5a7272ffe61f0e139537bfd
Author: kernellwp <kernellwp@gmail.com>
Date:   Fri Jan 27 04:51:43 2012 -0800

2

diff --git a/main.c b/main.c
index d00491f..1191247 100644
--- a/main.c
+++ b/main.c
@@ -1 +1,2 @@
 1
+2

```

建立含注释的标签会产生一个标签对象:

```

hacker@ubuntu:~/mygit$ mkdir tag3
hacker@ubuntu:~/mygit$ cd tag3/
hacker@ubuntu:~/mygit/tag3$ git init
Initialized empty Git repository in /home/hacker/mygit/tag3/.git/
hacker@ubuntu:~/mygit/tag3$ echo "3" > main.c
hacker@ubuntu:~/mygit/tag3$ git add .
hacker@ubuntu:~/mygit/tag3$ git commit -m "3"
[master (root-commit) 4013769] 3
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 main.c
hacker@ubuntu:~/mygit/tag3$ find .git/objects/ -type f | wc -l
3
hacker@ubuntu:~/mygit/tag3$ git tag -a v3.0 -m "this is tag v3.0"
hacker@ubuntu:~/mygit/tag3$ find .git/objects/ -type f | wc -l
4

```

可以看到在创建标签后对象数增加了一个。

二: git bisect

如果一个项目到某一个版本发现一个错误, 你还知道之前某个版本是好的, 那么可以用git bisect来定位最先出现bug的版本。可以:

git bisect start

git bisect bad 现在这个版本是有bug的

git bisect good good_commit good_commit是好的版本, 你可以用tag表示, 也可以用那20byte的前2个byte表示

```

hacker@ubuntu:~/mygit$ mkdir biseect
hacker@ubuntu:~/mygit$ cd biseect/
hacker@ubuntu:~/mygit/biseect$ git init
Initialized empty Git repository in /home/hacker/mygit/biseect/.git/
hacker@ubuntu:~/mygit/biseect$ echo "1" > main.c
hacker@ubuntu:~/mygit/biseect$ git add .
hacker@ubuntu:~/mygit/biseect$ git commit -m "1"
[master (root-commit) 37e9alc] 1
 1 files changed, 1 insertions(+), 0 deletions(-)
  create mode 100644 main.c
hacker@ubuntu:~/mygit/biseect$ git tag v1.0
hacker@ubuntu:~/mygit/biseect$ echo "2" >> main.c
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "2"
[master 557be9f] 2
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ git tag v2.0
hacker@ubuntu:~/mygit/biseect$ echo "3" >> main.c
hacker@ubuntu:~/mygit/biseect$ git tag v3.0
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "3"
[master 6180a85] 3
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ echo "4" >> main.c
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "4"
[master fa577a2] 4
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ git tag v4.0
hacker@ubuntu:~/mygit/biseect$ echo "5" >> main.c
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "5"
[master 67f41dd] 5
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ git tag v5.0
hacker@ubuntu:~/mygit/biseect$ echo "6" >> main.c
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "6"
[master a0979f0] 6
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ git tag v6.0
hacker@ubuntu:~/mygit/biseect$ echo "7" >> main.c
hacker@ubuntu:~/mygit/biseect$ git commit -a -m "7"
[master 59ca39b] 7
 1 files changed, 1 insertions(+), 0 deletions(-)
hacker@ubuntu:~/mygit/biseect$ git tag v7.0

```

```

hacker@ubuntu:~/mygit/biseect$ git bisect start
hacker@ubuntu:~/mygit/biseect$ git bisect bad
hacker@ubuntu:~/mygit/biseect$ git bisect good v1.0
Bisecting: 2 revisions left to test after this (roughly 2 steps)
[fa577a2bf8c7df734c7698e64781c922df570bb] 4
hacker@ubuntu:~/mygit/biseect$ cat main.c
1
2
3
4
hacker@ubuntu:~/mygit/biseect$ git bisect good
Bisecting: 0 revisions left to test after this (roughly 1 step)
[a0979f02a7de7161ef04b137065cc67cfad091] 6
hacker@ubuntu:~/mygit/biseect$ cat main.c
1
2
3
4
5
6
hacker@ubuntu:~/mygit/biseect$ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[67f41dd91266f4a9f95c85c91c8cb1b66b7ef8f] 5
hacker@ubuntu:~/mygit/biseect$ git bisect bad
67f41dd91266f4a9f95c85c91c8cb1b66b7ef8f is the first bad commit
commit 67f41dd91266f4a9f95c85c91c8cb1b66b7ef8f
Author: kernelwp <kernelwp@gmail.com>
Date:   Fri Jan 27 04:14:04 2012 -0800

5
:100644 100644 94ebaf900161394059478fd88aec30e59092ald7 8a1218a1024a212bb3db30becd860315f9f9ac52 M      main.c

```

我的这个意思是，如果数大于等于5就是bug，这里找到了第一个大于等于5的commit。

三：git format-patch

git format-patch -2 -o ~/patch/

git format-patch是用于把当前的git目录中的commit生成的patch文件，并组织成UNIX mailbox的邮件格式。--cc后指定的是邮件的抄送接收人。-2表示只处理最后两次commit

四：git send-email

**git send-email --to xxx@xxx --to xxx@xx --cc
xxx@xxx --bcc xx@xx ~/patch**

git send-email用于把刚才生成的patch文件直接以email的方式发送出去，要用这个命令

需要保证正确配置了SMTP服务器的相关信息。用git直接生成patch邮件发送到邮件列表是
git config file

[sendmail]

```
smtpencryption = tls
```

```
smtppass      = xxxx
```

```
smtpserver    = smtp.gmail.com
```

```
smtpuser = kernellwp@gmail.com
```

```
smtpserverport = 587
```

<http://blog.csdn.net/woshixingaaa/article/details/7211614>

[上一篇 Android如何防止apk程序被反编译](#)

[下一篇 扩展 JDT 实现自动代码注释与格式化](#)

主题推荐

[linux内核](#) [版本控制系统](#) [版本控制](#) [开源社区](#) [电子邮件](#)

猜你在找

[Git学习笔记二 Git初始化](#)

[Android framework系统默认设置修改](#)

[让jmeter脚本动起来之beanshell](#)

[Tornado getpost请求异步处理框架分析](#)

[64位电脑不能执行aapt](#)

[采用调用链或者函数符号表来定位crash位置](#)

[android的设备永不休眠增加 Settings--Dispaly----](#)

[Ext Js Store Load 返回数据库传出的Message](#)

[网卡DM9000的驱动移植](#)

[查看评论](#)

1楼 [syw19901001](#) 2014-12-03 19:57发表



<http://www.ihref.com/read-16369.html>

这篇不错，我学习了一下。和博主的结合用不错。

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题	Hadoop	AWS	移动游戏	Java	Android	iOS	Swift	智能硬件	Docker			
OpenStack	VPN	Spark	ERP	IE10	Eclipse	CRM	JavaScript	数据库	Ubuntu	NFC		
WAP	jQuery	BI	HTML5	Spring	Apache	.NET	API	HTML	SDK	IIS	Fedor	XML
LBS	Unity	Splashtop	UML	components	Windows Mobile	Rails	QEMU	KDE	Cassandra			
CloudStack	FTC	coremail	OPhone	CouchBase	云计算	iOS6	Rackspace	Web App				
SpringSide	Maemo	Compuware	大数据	apttech	Perl	Tornado	Ruby	Hibernate	ThinkPHP			
HBase	Pure	Solr	Angular	Cloud Foundry	Redis	Scala	Django	Bootstrap				

