[Git学习笔记](http://blog.csdn.net/liuhaoyutz/article/details/40919081)

分类： [学习记录](http://blog.csdn.net/liuhaoyutz/article/category/2651029)2014-11-08 11:36 1042人阅读 [评论](http://blog.csdn.net/liuhaoyutz/article/details/40919081#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/liuhaoyutz/article/details/40919081#report)

目录[(?)[+]](http://blog.csdn.net/liuhaoyutz/article/details/40919081)

作者：刘昊昱

博客：<http://blog.csdn.net/liuhaoyutz>

推荐学习网址

<http://git-scm.com/doc>

<http://lostechies.com/joshuaflanagan/2010/09/03/use-gitk-to-understand-git/>

1、commit树不是线性的，不是一条直线，而是tree，是树状的，可以有很多分支。

2、一个commit可以理解为是一个对repository的snapshot。

·        **创建一个新的版本库**

liuhaoyu@EMGD:~/work$ mkdir git\_study

liuhaoyu@EMGD:~/work$ cd git\_study/

liuhaoyu@EMGD:~/work/git\_study$ pwd

/home/liuhaoyu/work/git\_study

liuhaoyu@EMGD:~/work/git\_study$ git init

Initialized empty Git repository in/home/liuhaoyu/work/git\_study/.git/

liuhaoyu@EMGD:~/work/git\_study$

可以看到，我创建了/home/liuhaoyu/work/git\_study目录，然后在该目录下执行gitinit命令，只用这一个命令，即新建了一个空的git版本库。

·        **向Git版本库中添加一个新文件**

liuhaoyu@EMGD:~/work/git\_study$ echo "hello, gitstudy" >> Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

liuhaoyu@EMGD:~/work/git\_study$ git add Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "add areadme file"

[master (root-commit) b21ab70] add a readme file

 1 file changed, 1insertion(+)

 create mode 100644Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git log

commit b21ab7072128a930d697143c5d2dcd4afb4b0a06

Author: Liu Haoyu <haoyux.liu@intel.com>

Date:   Sun Aug 1018:04:25 2014 +0800

    add a readme file

liuhaoyu@EMGD:~/work/git\_study$

可以看到，向Git版本库中添加一个新文件需要两步，一个是执行：git add filename，然后再执行：gitcommit -m “提交说明”。

·        **修改一个文件的内容，并提交**

修改Readme.txt文件的内容，添加一行：

liuhaoyu@EMGD:~/work/git\_study$ echo "add a line">> Readme.txt

然后执行git status命令，查看当前工作区的状态：

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "git add<file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       modified:   Readme.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

可以看到，git status命令显示文件Readme.txt的内容改变了，但是还没有准备commit的changes。

虽然git status命令告诉我们Readme.txt文件被修改了，但是没有告诉我们该文件具体发生了什么样的改变，有时我们修改文件后，时间久了会忘记自己做了哪些修改，此时，我们可以用”gitdiff文件名”命令来看指定文件发生了哪些修改。

liuhaoyu@EMGD:~/work/git\_study$ git diff Readme.txt

diff --git a/Readme.txt b/Readme.txt

index 23ecf33..bee7231 100644

--- a/Readme.txt

+++ b/Readme.txt

@@ -1 +1,2 @@

 hello, git study

+add a line

liuhaoyu@EMGD:~/work/git\_study$

通过” git diff Readme.txt”命令我们可以看到，Readme.txt文件增加了一行，内容为”adda line”

知道了对Readme.txt文件做了什么样的修改，我们把修改后的状态提交给版本库就放心了，提交修改后的状态和提交新文件过程是一样的，即gitadd filename和git commit -m “注释”。

liuhaoyu@EMGD:~/work/git\_study$ git add Readme.txt

执行完git add filename命令后，在执行git commit命令之前，我们先用gitstatus来看看当前工作区的状态：

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes to be committed:

#   (use "gitreset HEAD <file>..." to unstage)

#

#       modified:   Readme.txt

#

liuhaoyu@EMGD:~/work/git\_study$

可以看到，Readme.txt文件的changes已经准备好被commit。

下面我们提交修改后的状态：

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "add aline"

[master 8f34dac] add a line

 1 file changed, 1insertion(+)

liuhaoyu@EMGD:~/work/git\_study$

提交后，我们再来用git status命令看一下工作区当前状态：

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

nothing to commit (working directory clean)

liuhaoyu@EMGD:~/work/git\_study$

可以看到，没有需要commit的变化，工作区是干净的。

·        **回退到以前的版本**

上一节我们学会了怎样修改一个文件，并提交修改后的状态，下面我们再给Readme.txt文件增加一行，然后提交修改后的状态：

liuhaoyu@EMGD:~/work/git\_study$ echo "add secondline" >> Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git add Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "addsecond line"

[master 2e7c6f8] add second line

 1 file changed, 1insertion(+)

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

add second line

liuhaoyu@EMGD:~/work/git\_study$ git log

commit 2e7c6f86f733392403192f356db76f0ec2248509

Author: Liu Haoyu <haoyux.liu@intel.com>

Date:   Sun Aug 1019:16:51 2014 +0800

    add second line

commit 8f34dacd3f588226adc1ab81921e02b167247c80

Author: Liu Haoyu <haoyux.liu@intel.com>

Date:   Sun Aug 1019:00:40 2014 +0800

    add a line

commit 684929115f347724447573cc0cfe4ccbbed4c078

Author: Liu Haoyu <haoyux.liu@intel.com>

Date:   Sun Aug 1018:34:33 2014 +0800

    add a readme file

liuhaoyu@EMGD:~/work/git\_study$ git log --pretty=oneline

2e7c6f86f733392403192f356db76f0ec2248509 add second line

8f34dacd3f588226adc1ab81921e02b167247c80 add a line

684929115f347724447573cc0cfe4ccbbed4c078 add a readme file

liuhaoyu@EMGD:~/work/git\_study$

现在，如果我想回到上一次提交的状态，即”add a line”的状态，应该怎么办呢？

首先，Git必须知道当前版本是哪个版本，在Git中，使用HEAD表示当前版本，上一个版本用HEAD^表示，上上个版本用HEAD^^表示，那当然，往上100个版本写100个^不容易写，所以写成HEAD~100。

现在我们要回退到上一次提交的状态，可以使用如下命令：

liuhaoyu@EMGD:~/work/git\_study$ git reset --hard HEAD^

HEAD is now at 8f34dac add a line

liuhaoyu@EMGD:~/work/git\_study$ git log --pretty=oneline

8f34dacd3f588226adc1ab81921e02b167247c80 add a line

684929115f347724447573cc0cfe4ccbbed4c078 add a readme file

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

liuhaoyu@EMGD:~/work/git\_study$

可以看到，我们已经回退到上一次提交的状态了。

现在如果我又想回到” add second line”状态，该怎么办呢？

只要你知道” add second line”状态的commit ID即可：

liuhaoyu@EMGD:~/work/git\_study$ git reset --hard2e7c6f86f733392403192f356db76f0ec2248509

HEAD is now at 2e7c6f8 add second line

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

add second line

liuhaoyu@EMGD:~/work/git\_study$ git log --pretty=oneline

2e7c6f86f733392403192f356db76f0ec2248509 add second line

8f34dacd3f588226adc1ab81921e02b167247c80 add a line

684929115f347724447573cc0cfe4ccbbed4c078 add a readme file

·        **提交对多个文件的多处修改**

下面我们再给Readme.txt文件添加一行文字，然后再创建一个新的文件file1.txt：

liuhaoyu@EMGD:~/work/git\_study$ echo "file1" >file1.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "git add<file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       modified:   Readme.txt

#

# Untracked files:

#   (use "git add<file>..." to include in what will be committed)

#

#       file1.txt

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$

通过git status命令可以看到，Readme.txt文件被修改了，而file1.txt文件则没有被track，也就是说没有添加过file1.txt文件。

下面我们通过两次调用git add，添加文件Readme.txt和file1.txt，然后用gitstatus来看工作区当前状态：

liuhaoyu@EMGD:~/work/git\_study$ git add Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git add file1.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes to be committed:

#   (use "gitreset HEAD <file>..." to unstage)

#

#       modified:   Readme.txt

#       new file:   file1.txt

#

liuhaoyu@EMGD:~/work/git\_study$

可以看到，两个changes准备好被commit。

下面执行git commit，提交对两个文件的changes：

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "two filechanges commit"

[master bbf2322] two file changes commit

 2 files changed, 2insertions(+)

 create mode 100644file1.txt

liuhaoyu@EMGD:~/work/git\_study$

·        **撤销修改**

下面我们修改文件file1.txt的内容，添加一行文字：

liuhaoyu@EMGD:~/work/git\_study$ echo "add oneline" >> file1.txt

liuhaoyu@EMGD:~/work/git\_study$ cat file1.txt

file1

add one line

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "git add<file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       modified:   file1.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$

现在我们想撤销对file1.txt所做的修改，恢复到file1.txt之前的状态，应该怎么办呢？我们可以手工把修改的内容撤销，但是如果修改的地方比较多，手工撤销可能很困难，此时，正如gitstatus命令打印的信息所提示的那样，我们可以用如下命令来撤销修改：

liuhaoyu@EMGD:~/work/git\_study$ git checkout -- file1.txt

liuhaoyu@EMGD:~/work/git\_study$ cat file1.txt

file1

liuhaoyu@EMGD:~/work/git\_study$

再考虑一种情况：如果我们修改了file1.txt的内容，添加了一行文字，并且用git addfile1.txt提交到了暂存区，然后我们想撤销对file1.txt的修改，应该怎么做呢？

liuhaoyu@EMGD:~/work/git\_study$ echo "add oneline" >> file1.txt

liuhaoyu@EMGD:~/work/git\_study$ cat file1.txt

file1

add one line

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "git add<file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       modified:   file1.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$ git add file1.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes to be committed:

#   (use "gitreset HEAD <file>..." to unstage)

#

#       modified:   file1.txt

#

liuhaoyu@EMGD:~/work/git\_study$

正如git status提示的那样，我们可以用如下命令撤销暂存区对文件file1.txt的修改：

liuhaoyu@EMGD:~/work/git\_study$ git reset HEAD file1.txt

Unstaged changes after reset:

M       file1.txt

liuhaoyu@EMGD:~/work/git\_study$

再用git status查看一下，暂存区对file1.txt的修改已经被撤销，但是工作区对file1.txt的修改依然存在，所以我们可以像前面一样，用gitcheckout -- filename命令撤销对工作区中文件的修改：

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "git add<file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       modified:   file1.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$ git checkout -- file1.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

nothing to commit (working directory clean)

liuhaoyu@EMGD:~/work/git\_study$ cat file1.txt

file1

liuhaoyu@EMGD:~/work/git\_study$

·        **删除文件**

在Git中，删除也是一种修改。我们实战一下删除操作：首先新建一个文件并commit：

liuhaoyu@EMGD:~/work/git\_study$ echo file2 > file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git add file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "addfile2.txt"

[master 4be0d22] add file2.txt

 1 file changed, 1insertion(+)

 create mode 100644file2.txt

liuhaoyu@EMGD:~/work/git\_study$

假设经过一段时间工作后，文件file2.txt被删除了，执行如下命令：

liuhaoyu@EMGD:~/work/git\_study$ rm file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "gitadd/rm <file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       deleted:    file2.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$

可以看到，因为工作区文件file2.txt被删除了，但版本库在还保存着该文件的信息，所以Git会检测到这种区别，gitstatus命令会提示工作区和版本库的这种差别，即工作区的file2.txt被删除了。

这又分为两种情况，一是file2.txt确实不需要了，我们主动删除它；还有一种情况是file2.txt有用，被误删除了。

如果是我们主动删除的，那么我们就要执行git rm filename命令，从版本库中也删除file2.txt的信息，并且执行gitcommit提交这个改变：

liuhaoyu@EMGD:~/work/git\_study$ git rm file2.txt

rm 'file2.txt'

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes to be committed:

#   (use "gitreset HEAD <file>..." to unstage)

#

#       deleted:    file2.txt

#

liuhaoyu@EMGD:~/work/git\_study$ git commit -m"file2.txt deleted"

[master 2a433a3] file2.txt deleted

 1 file changed, 1deletion(-)

 delete mode 100644file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

nothing to commit (working directory clean)

liuhaoyu@EMGD:~/work/git\_study$

如果是我们误删除了file2.txt文件：

liuhaoyu@EMGD:~/work/git\_study$ rm file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

# Changes not staged for commit:

#   (use "gitadd/rm <file>..." to update what will be committed)

#   (use "gitcheckout -- <file>..." to discard changes in working directory)

#

#       deleted:    file2.txt

#

no changes added to commit (use "git add" and/or"git commit -a")

liuhaoyu@EMGD:~/work/git\_study$

因为版本库中有保存的file2.txt文件的信息，所以正如git status所提示的那样，我们可以从版本库中把file2.txt恢复回来：

liuhaoyu@EMGD:~/work/git\_study$ git checkout -- file2.txt

liuhaoyu@EMGD:~/work/git\_study$ git status

# On branch master

nothing to commit (working directory clean)

liuhaoyu@EMGD:~/work/git\_study$ cat file2.txt

file2

liuhaoyu@EMGD:~/work/git\_study$

·        **将本机上的Git版本库备份到GitHub上**

1、申请GitHub账号并上传SSH Key。

2、在GitHub上创建一个新的代码仓库（New repository）,填写Repositoryname，这里我就起名叫git\_study。其它选项使用默认设置，点Createrepository创建。

3、创建完新的代码仓库后，GitHub会提示你下一步怎么操作：

如果要创建一个新的本地仓库，并上传到GitHub上去，执行如下命令：

touchREADME.md

git init

git addREADME.md

git commit-m "first commit"

git remoteadd origin https://github.com/liuhaoyutz/git\_study.git

git push-u origin master

如果要将一个已经存在的本地仓库上传到GitHub上去，执行如下命令：

git remote add origin https://github.com/liuhaoyutz/git\_study.git

git push -u origin master

其中，

git remote add originhttps://github.com/liuhaoyutz/git\_study.git

用于指定远程仓库名为origin，其URL地址为<https://github.com/liuhaoyutz/git_study.git>

Git push -u origin master

用于将本地master分支上传到远程仓库origin中。第一次用gitpush命令向远程仓库推送master分支时，加上-u参数，这样不但完成推送，还将本地master分支与远程master分支关联起来，以后再推送master分支时，只要执行gitpush origin master即可。

·        **clone远程仓库**

首先要知道远程仓库的地址，例如我在GitHub上创建的git\_study仓库，其地址是<https://github.com/liuhaoyutz/git_study.git>

知道了远程仓库的地址，我们就可以clone远程仓库了，使用如下命令：

git clone https://github.com/liuhaoyutz/git\_study.git

Git远程仓库的地址支持https协议、ssh协议、git协议，但是许多在公司内部，只开放了https协议，所以只能使用https地址。

·        **创建和操作分支**

我们在本地的git\_study仓库下创建一个新的分支development，并切换到该分支：

liuhaoyu@EMGD:~/work/git\_study$ git branch -a

\* master

 remotes/origin/master

liuhaoyu@EMGD:~/work/git\_study$ git branch development

liuhaoyu@EMGD:~/work/git\_study$ git branch -a

  development

\* master

 remotes/origin/master

liuhaoyu@EMGD:~/work/git\_study$ git checkout development

Switched to branch 'development'

liuhaoyu@EMGD:~/work/git\_study$ git branch -a

\* development

  master

 remotes/origin/master

创建分支用命令：git branch 新建分支名

切换分支用命令：git checkout分支名

git branch -a 显示的分支中，前面有星号的表示当前操作的分支。

所以以后我们的操作都是在development分支上进行了：

liuhaoyu@EMGD:~/work/git\_study$ echo "add a newbranch" >> Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

add second line

add third line

add a new branch

liuhaoyu@EMGD:~/work/git\_study$ git add Readme.txt

liuhaoyu@EMGD:~/work/git\_study$ git commit -m "add anew branch"

[development 2ef19f6] add a new branch

 1 file changed, 1insertion(+)

liuhaoyu@EMGD:~/work/git\_study$ git checkout master

Switched to branch 'master'

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

add second line

add third line

liuhaoyu@EMGD:~/work/git\_study$

我们来修改Readme.txt文件并提交，是提交到development分支上，master分支不受影响，切换到master分支，Readme.txt还是原来的样子。

下面我们将development分支合并到master分支：

liuhaoyu@EMGD:~/work/git\_study$ git branch

  development

\* master

liuhaoyu@EMGD:~/work/git\_study$ git merge development

Updating e2edfe0..2ef19f6

Fast-forward

 Readme.txt |    1 +

 1 file changed, 1insertion(+)

liuhaoyu@EMGD:~/work/git\_study$ cat Readme.txt

hello, git study

add a line

add second line

add third line

add a new branch

liuhaoyu@EMGD:~/work/git\_study$

git merge 分支名，该命令用于将指定分支合并到当前分支。

最后，我们不再需要development分支，可以通过如下命令删除它：

liuhaoyu@EMGD:~/work/git\_study$ git branch -d development

Deleted branch development (was 2ef19f6).

liuhaoyu@EMGD:~/work/git\_study$ git branch -a

\* master

 remotes/origin/master

liuhaoyu@EMGD:~/work/git\_study$

git branch -d 分支名，删除一个已经合并过的分支。

git branch -D 分支名，强制删除一个未合并过的分支。

·        **使用Tag**

创建一个Tag用如下命令：git tag tagname

例如：git tag 1.0.0

如果要在某个commit上创建Tag，使用如下命令：gittag tagname commit\_id

例如：git tag 0.1.5 684929115f347724447573cc0cfe4ccbbed4c078

检出某个Tag指定的版本用如下命令：

git checkout tagname

**·        查找问题的利器 - Git Blame**

如果你要查看文件的每个部分是谁修改的, 那么 [git blame](http://www.kernel.org/pub/software/scm/git/docs/git-blame.html) 就是不二选择. 只要运行'git blame [filename]', 你就会得到整个文件的每一行的详细修改信息:包括SHA串,日期和作者:

译者注: Git采用SHA1做为hash签名算法, 在本书中,作者为了表达方便,常常使用SHA来代指SHA1. 如果没有特别说明, 本书中的SHA就是SHA1的代称.

$ git blame sha1\_file.c

...

0fcfd160 (Linus Torvalds  2005-04-18 13:04:43 -0700    8)  \*/

0fcfd160 (Linus Torvalds  2005-04-18 13:04:43 -0700    9) #include "cache.h"

1f688557 (Junio C Hamano  2005-06-27 03:35:33 -0700   10) #include "delta.h"

a733cb60 (Linus Torvalds  2005-06-28 14:21:02 -0700   11) #include "pack.h"

8e440259 (Peter Eriksen   2006-04-02 14:44:09 +0200   12) #include "blob.h"

8e440259 (Peter Eriksen   2006-04-02 14:44:09 +0200   13) #include "commit.h"

8e440259 (Peter Eriksen   2006-04-02 14:44:09 +0200   14) #include "tag.h"

8e440259 (Peter Eriksen   2006-04-02 14:44:09 +0200   15) #include "tree.h"

f35a6d3b (Linus Torvalds  2007-04-09 21:20:29 -0700   16) #include "refs.h"

70f5d5d3 (Nicolas Pitre   2008-02-28 00:25:19 -0500   17) #include "pack-revindex.h"628522ec (Junio C Hamano              2007-12-29 02:05:47 -0800   18) #include "sha1-lookup.h"

...

如果文件被修改了(reverted),或是编译(build)失败了; 这个命令就可以大展身手了.

你也可以用"-L"参数在命令(blame)中指定开始和结束行:

$>git blame -L 160,+10 sha1\_file.c

ace1534d (Junio C Hamano 2005-05-07 00:38:04 -0700       160)}

ace1534d (Junio C Hamano 2005-05-07 00:38:04 -0700       161)

0fcfd160 (Linus Torvalds 2005-04-18 13:04:43 -0700       162)/\*

0fcfd160 (Linus Torvalds 2005-04-18 13:04:43 -0700       163) \* NOTE! This returns a statically allocate

790296fd (Jim Meyering   2008-01-03 15:18:07 +0100       164) \* careful about using it. Do an "xstrdup()

0fcfd160 (Linus Torvalds 2005-04-18 13:04:43 -0700       165) \* filename.

ace1534d (Junio C Hamano 2005-05-07 00:38:04 -0700       166) \*

ace1534d (Junio C Hamano 2005-05-07 00:38:04 -0700       167) \* Also note that this returns the location

ace1534d (Junio C Hamano 2005-05-07 00:38:04 -0700       168) \* SHA1 file can happen from any alternate

d19938ab (Junio C Hamano 2005-05-09 17:57:56 -0700       169) \* DB\_ENVIRONMENT environment variable if i

·        **撤消对某个文件的修改**

git checkout -- <file\_name>

例如：

git checkout -- src/core/gem.c

·        **使用Git查看某个文件的修改历史**

1、执行git log --pretty=oneline 文件名

# git log --pretty=oneline src/core/git\_test.c

72354e1eb423456d29f2a33ee4342349c36e968bc Fix klockworkissue by add NULL check

ac2345ae927e91b9986d342432535d5df4c2fd0f5c Combined twoPatches for the bug\_123

8842f12215bb567243432f5be6e86d632e55491efb Use newbind/unbind

2、执行git show commit\_id命令显示次修改的具体内容

# git show 72354e1eb423456d29f2a33ee4342349c36e968bc

·        **使用gitblame命令查看某文件每一行是谁/怎样修改的**

这个命令非常有用，会显示每一行是谁做了怎样的修改。

git blame file\_name

以上命令会显示每一行的修改对应的commit号，有了commit号，再执行如下命令：

git show commit\_id

即可查看具体修改内容。

如果不想查看整个文件的修改历史，而只是关注具体的某几行，可以加上-L参数

git blame -L 1000,+20file\_name

·        **使用git am命令批量打patch**

**实际上git apply才是打patch的命令“Apply a patch to files and/or to the index”。而git am实际上是“Apply a series ofpatches from a mailbox”**

这篇文章主要介绍一下git-am 和 format-patch 的使用。 因为在git使用当中，会有很多时候别人（供应商或者其他的开发人员）发过来一系列的**patch**，这些patch通常的是类似这样的名字：

0001--JFFS2-community-fix-with-not-use-OOB.patch 0002--Community-patch-for-Fix-mount-error-in.patch 0003--partial-low-interrupt-latency-mode-for-ARM113.patch 0004--for-the-global-I-cache-invalidation-ARM11.patch 0005--1-arm-Add-more-cache-memory-types-macr.patch 0006--2-Port-imx-3.3.0-release-to-2.6.28.patch 0007--3-Add-MX25-support.patch 0008--Move-asm-arch-headers-to-linux-inc-dir.patch 0009--1-regulator-allow-search-by-regulator.patch

里面包含了提交的日志，作者，日期等信息。你想做的是把这些patch引入到你的代码库中，最好是也可以把日志也引入进来， 方便以后维护用。传统的打patch方式是

patch-p1 < 0001--JFFS2-community-fix-with-not-use-OOB.patch

这样来打patch，但是这样会把这些有用的信息丢失。由于这些patch显然是用gitformat-patch来生成的，所以用git的工具应该就可以很好的做好。git-am就是作这件事情。

在使用git-am之前， 你要首先git am –abort一次，来放弃掉以前的am信息，这样才可以进行一次全新的am。  
不然会遇到这样的错误。  
               .git/rebase-apply still exists but mbox given.

git-am 可以一次合并一个文件，或者一个目录下所有的patch，或者你的邮箱目录下的patch.

下面举两个例子：

1. 你现在有一个code base： small-src,你的patch文件放在~/patch/0001-trival-patch.patch

cdsmall-src git-am~/patch/0001-trival-patch.patch

如果成功patch上去， 你就可以去喝杯茶了。如果失败了， git 会提示错误， 比如：

error:patch failed: android/mediascanner.cpp:452 error: android/mediascanner.cpp:patch does not apply

这样你就需要先看看patch， 然后改改错误的这个文件，让这个patch能够patch上去。

1. 你有一堆patch， 名字是上面提到的那一堆patch， 你把他们放在~/patch-set/目录下（路径随意）

cdopencore git am~/patch-set/\*.patch

(这里git就会按照文件名的顺序一次am这些patch）如果一切顺利， 你所有的patch都OK了， 你又Lucky了。不过不顺利的时候十有八九，如果git am中间遇到了patch,am就会停到打这个patch的地方， 告诉你是哪个patch打不上去。

比如我现在有一个文件file,有两个patch.  
file 的内容是

thetext more text

两个patch分别是：

0001-add-line.patch:

From8869ccbced494e05738090afa5a54f2a261df0f Mon Sep 1700:00:00 2001 From: abc [abc@abc-desktop.(none)](mailto:zhangjiejing@zhangjiejing-desktop.%28none%29) Date: Thu, 22 Apr2010 13:04:34 +0800 Subject:[PATCH 1/2] add line --- file | 2 ++ 1files changed, 2 insertions(+), 0 deletions(-) diff --git a/file b/file index 067780e..685f0fa 100644 --- a/file +++ b/file @@ -3,3 +3,5 @@ file: some text more text + +add line --1.6.3.3

0002-change-line.patch:

Fromf756e1b3a87c216b7e0afea9d15badd033171578 Mon Sep 17 00:00:00 2001 From: abc [abc@abc-desktop.(none)](mailto:zhangjiejing@zhangjiejing-desktop.%28none%29) Date: Thu, 22 Apr2010 13:05:19 +0800 Subject:[PATCH 2/2] change line --- file | 2 +- 1files changed, 1 insertions(+), 1 deletions(-) diff --git a/file b/file index 685f0fa..7af7852 100644 --- a/file file: -some text +Change line text more text -- 1.6.3.3

运行  
git am \*.patch

来merge这些patch， 报错， Patch failed at 0001 addline这样我们看0001这个patch,原来patch需要的是some text, 而file里面是the text, 所以我们用编辑器把这行改成sometext,

**git apply --reject 0001-add-line.patch;  fix file;  git add file; git am --resolved;**

在解决完冲突以后， 比如用git add来让git知道你已经解决完冲突了。

如果你发现这个冲突是无法解决的， 要撤销整个am的东西。 可以运行git am –abort，

如果你想只是忽略这一个patch，可以运行git am –skip来跳过这个patch.

**Deal with git am failures**

Posted on [2011/10/31](http://www.pizzhacks.com/bugdrome/2011/10/deal-with-git-am-failures/) by [bugdromer](http://www.pizzhacks.com/bugdrome/author/bugdromer/)

Applying a git-formatted patch is as simple as running:

$ git am PATCH

If all goes well you should find the patch integrated into yourrepo.

In case of failure this gets a bit more complicated.

If you tryto google for “git am fail” you’ll find this [post](http://kerneltrap.org/mailarchive/git/2007/2/22/239371) which unfortunately is quite outdated, and I couldn’t findreference to the problem in the git manual (or maybe I didn’t stare enough intoit).

So you just tried git am PATCH, and get this:

$ git am PATCH

Applying: PACTH DESCRIPTION

error: patch failed: file.c:137

error: file.c: patch does not apply

error: patch failed: Makefile:24

error: libavfilter/Makefile: patch does not apply

Patch failed at 0001 PATCH DESCRIPTION

When you have resolved this problem run "git am --resolved".

If you would prefer to skip this patch, instead run "git am --skip".

To restore the original branch and stop patching run "git am --abort".

In these cases git will just complain and stop. No bits from thepatch are applied if a single conflict is found.

The simplest way for dealing with it would be to git am --abort,apply the patch manually by patch -p1 < PATCH, resolve the conflict by hand,and finally commit with git commit -a, but in this case you'll have to rewritethe commit message, which is not very nice. There is a more clever way.

You can find the corresponding patch file stored in.git/rebase-apply, and named "0001" (the name of the dir where thepatch is stored changed recently, this is tested with 1.7.4.1).  
At this point you can use git apply for applying the patch, which is the gitequivalent of the patch command, and fix the conflicting files the usual way(you check the .rej files, compare them with the conflicting files and finallyadd the fixed files to the index):

$ git apply PATCH --reject

$ edit edit edit

$ git add FIXED\_FILES

$ git am --resolved

and you're done!  
In other words, since git am didn't change the index, you need to git apply--reject the patch (stored in .git/rebase-apply, fix conflicts by hand, add thechanged files and finally tell git that you resolved the trouble. The advantagein this case is that you don't need to re-edit the commit message, and in thecase you're applying a set of patches (that is you're using git am PATCHES,where PATCHES is a mailbox) you don't have to git abort and run git am again.

·        **使用gitapply命令打patch**

**实际上git apply才是打patch的命令“Apply a patch to files and/or to the index”。而git am实际上是“Apply a series ofpatches from a mailbox”**

git apply是patch命令的git版本，例如，打一个patch，使用如下命令：

# git apply patch\_file

撤销一个patch，使用如下命令：

# git apply -R patch\_file

git apply可以加一些参数，常用的参数如下：

-R, --reverse

Apply the patch inreverse.

--check

Instead of applying thepatch, see if the patch is applicable to the current working tree and/or theindex file and detects errors.

--reject

For atomicity, git applyby default fails the whole patch and does not touch the working tree when someof the hunks do not apply. This option makes it apply the parts of the patchthat are applicable, and leave the rejected hunks in corresponding \*.rej files.

·        **使用gitreset将当时分支恢复到某个commit的状态**

首先，Git必须知道当前版本是哪个版本，在Git中，使用HEAD表示当前版本，上一个版本用HEAD^表示，上上个版本用HEAD^^表示，那当然，往上100个版本写100个^不容易写，所以写成HEAD~100。

现在我们要回退到上一次提交的状态，可以使用如下命令：

# git reset --hard HEAD^

如果要回退到某个commit指定的版本，可以使用如下命令：

# git reset --hard 2e7c6f86f733392403192f356db76f0ec2248509

git reset可以带一些参数，man手册有如下信息：

git reset --<mode> [<commit>]

This form resets the current branch head to <commit>and possibly updates the index (resetting it to the tree of <commit>) andthe working tree depending on <mode>, which must be one of the following:

--soft

Does not touch the index file nor the working tree at all(but resets the head to <commit>, just like all modes do). This leavesall your changed files "Changes to be committed", as git status wouldput it.

--mixed

Resets the index but not the working tree (i.e., the changedfiles are preserved but not marked for commit) and reports what has not beenupdated. This is the default action.

--hard

Resets the index and working tree. Any changes to trackedfiles in the working tree since <commit> are discarded.

--merge

Resets the index and updates the files in the working treethat are different between <commit> and HEAD, but keeps those which aredifferent between the index and working tree (i.e. which have changes whichhave not been added). If a file that is different between <commit> andthe index has unstaged changes, reset is aborted. In other words, --merge doessomething like a git read-tree -u -m <commit>, but carries forwardunmerged index entries.

--keep

Resets index entries and updates files in the working treethat are different between <commit> and HEAD. If a file that is differentbetween <commit> and HEAD has local changes, reset is aborted.

If you want to undo a commit other than the latest on abranch, git-revert(1) is your friend.

·        **使用gitcherry-pick将其它分支上的commit应用到当前分支上**

Apply the changes introduced by some existing commits。用法举例：

# git cherry-pick commit\_id

如果git cherry-pick冲突，会有如下提示：

Automatic cherry-pick failed.  Afterresolving the conflicts,

mark the corrected paths with 'git add<paths>' or 'git rm <paths>'

and commit the result with:

        git commit -c 15a2b6c61927e5aed6718de89ad9dafba939a90b

根据提示，发生git cherry-pick冲突后，我们需要执行如下操作：

1. 手动解决冲突

2. git add / git rm

3. git commit -c  15a2b6c61927e5aed6718de89ad9dafba939a90b

·        **git merge**

将分支”branch\_name”合并到当前分支：

# git merge branch\_name

如果merge时发生冲突，需要手动解决，可以执行

#git status

查看哪些文件发生了冲突。

参考：

http://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E6%96%B0%E5%BB%BA%E4%B8%8E%E5%90%88%E5%B9%B6

·        **使用gitpush将本地修改上传到远程服务器**

# git push origin test:master // 提交本地test分支作为远程的master分支  
# git push origin test:test   // 提交本地test分支作为远程的test分支  
# git push origin :test // 刚提交到远程的test将被删除，但是本地还会保存的

格式：

git push <options> <repository> <refspec>

<repository>

The "remote" repository that is destination of apush operation. This parameter can be either a URL (see the section GIT URLSbelow) or the name of a remote (see the section REMOTES below).

<refspec>

The format of a <refspec> parameter is an optionalplus +, followed by the source ref <src>, followed by a colon :, followedby the destination ref <dst>. It is used to specify with what <src>object the <dst> ref in the remote repository is to be updated.

The <src> is often the name of the branch you wouldwant to push, but it can be any arbitrary "SHA-1 expression", such asmaster~4 or HEAD (see gitrevisions(7)).

The <dst> tells which ref on the remote side isupdated with this push. Arbitrary expressions cannot be used here, an actualref must be named. If :<dst> is omitted, the same ref as <src> willbe updated.

The object referenced by <src> is used to update the<dst> reference on the remote side, but by default this is only allowedif the update can fast-forward <dst>. By having the optional leading +,you can tell git to update the <dst> ref even when the update is not afast-forward. This does not attempt to merge <src> into <dst>. SeeEXAMPLES below for details.

tag <tag> means the same asrefs/tags/<tag>:refs/tags/<tag>.

Pushing an empty <src> allows you to delete the<dst> ref from the remote repository.

The special refspec : (or +: to allow non-fast-forwardupdates) directs git to push "matching" branches: for every branchthat exists on the local side, the remote side is updated if a branch of thesame name already exists on the remote side. This is the default operation modeif no explicit refspec is found (that is neither on the command line nor in anyPush line of the corresponding remotes file---see below).

·        **使用gitlog命令显示commitlog**

1、显示commit log：

# git log

2、以一行的形式显示commit log：

# git log --pretty=oneline

或

# git log --oneline

This is a shorthand for "--pretty=oneline--abbrev-commit" used together.

3、显示作用于某个特定文件上的commit log：

# git log file\_name

4、以一行的形式显示作用于某个特定文件上的commit log：

# git log --pretty=oneline file\_name

·        **gitcheckout**

git-checkout - Checkout a branch or paths to the workingtree

这里的branch好理解，但是paths怎样理解呢？答案是一个具体文件或目录的路径，即我们不但可以使用gitcheckout命令将一个branch检出到workingtree，还可以通过指定具体路径，使用gitcheckout命令将某个目录下的所有文件或某个具体文件检出到workingtree。

1、检出(check out)某个分支(branch)到当前工作树(workingtree)：

# git checkout branch\_name

2、创建一个新的分支并切换过去

# git checkout -b branch\_name

3、检出某个具体目录或文件

格式：git checkout <commit> < -- > <path>

commit代表一个分支，是可选的，如果不指定commit，则会从暂存区中检出目录或文件，如果指定了commit，则会用提交的目录或文件内容覆盖暂存区和工作区对应内容。

# git checkout directory\_path

# git checkout file\_path

如果目录或文件名与一个branch同名，则上面的命令会出错，因为git不知道是要检出branch还是目录/文件，此时，可以用下面的命令表示要检出目录/文件：

# git checkout -- directory\_path

# git checkout -- file\_path

4、检出到某个commit的状态

# git checkout commit\_id

注意此时会提示此后的操作不在任何一个分支上，此时处于“分离头指针”状态，在“分离头指针”状态下的提交不能被引用关联到，从而可能丢失。

5、执行状态检查

# git checkout

只是执行git checkout命令而不带任何参数，会进行一些状态检查，并给出提示信息。

6、强制检出

# git checkout -f commit

加上-f参数，表示进行强制检出，即使index或working tree的内容有改动，所以强制检出会抛弃所有本地修改的内容。

When switching branches, proceed even if the index or theworking tree differs from HEAD. This is used to throw away local changes.

7、强制恢复当前工作分支到原始内容

# git checkout -f

只是执行git checkout -f命令，而不加分支名，则会强制恢复当前工作的分支到原始内容。

·        **gitstatus、gitcheckout、gitdiff等命令结果标志**

执行man git diff可以查到这些标志信息，如下：

       ·  A: addition of a file

       ·   C: copy of a file into a new one

       ·   D: deletion of a file

       ·   M: modification of the contents or mode of afile

       ·   R: renaming of a file

       ·   T: change in the type of the file

       ·   U: file is unmerged (you must complete themerge before it can be committed)

       ·   X: "unknown" change type (mostprobably a bug, please report it)

·        **使用gitformat-patch命令取得从指定commit开始的n个patch**

格式：

# git format-patch commit\_id -n

例如：

# git format-patch 3e12233405270961cd52155b4a8b436472814748-5

表示获取从commit 3e12233405270961cd52155b4a8b436472814748开始的5个patch。

·        **使用gitsend-email发送patch到指定邮件**

# git send-email -to xxx@126.com \*patch

默认情况下git send-email会自动添加path中包含的所有邮件到CC列表中，如果要禁止git自动添加CC列表，可以使用如下参数:

# git send-email --suppress-cc=all -to xxx@126.com \*patch