Git总结文档

目录

Autor:李亚廷

Date:2018/07/09

[1. Git简介 3](#_Toc2269)

[2. 关键术语 3](#_Toc29644)

[3. Git基础和原理 3](#_Toc32340)

[3.1 Git的特点 3](#_Toc20334)

[3.2 Git保证完整性 4](#_Toc20988)

[4. Git安装 4](#_Toc20719)

[4.1 Linux系统安装 4](#_Toc18356)

[4.2 基于Debian的发行版 4](#_Toc12856)

[4.3 Unix 风格的系统上安装步骤 4](#_Toc18720)

[4.4 mac上安装 4](#_Toc30781)

[4.5 window上安装 4](#_Toc27100)

[5. Git使用前配置 4](#_Toc18641)

[5.1 用户信息 4](#_Toc15976)

[5.2 文本编辑器 4](#_Toc23661)

[5.3 检查配置信息 5](#_Toc9076)

[5.4 检查版本信息 5](#_Toc21197)

[5.5 获取帮助 5](#_Toc16490)

[6. Git快速入门 5](#_Toc10700)

[6.1 什么是远程仓库 5](#_Toc10090)

[6.2 获取Git仓库 5](#_Toc20467)

[6.3 克隆现有Git仓库 5](#_Toc17946)

[6.4 初始化Git仓库 6](#_Toc22669)

[6.5 检查当前文件状态 6](#_Toc13810)

[6.6 文件状态简览 6](#_Toc17722)

[7. Git工作流程 6](#_Toc12270)

[8. Git常用命令 7](#_Toc14688)

[8.1 git config 7](#_Toc18123)

[8.1.1配置文件的存储位置 7](#_Toc19953)

[8.1.2配置用户名和密码 7](#_Toc25663)

[8.1.3配置编辑器 7](#_Toc12359)

[8.1.4配置比较工具 7](#_Toc10436)

[8.1.5检查配置 8](#_Toc19773)

[8.1.6添加/删除配置项 8](#_Toc2803)

[8.1.7获取帮助 8](#_Toc4517)

[8.2 git help 8](#_Toc14976)

[8.3 git init 9](#_Toc21448)

[8.4 git add 9](#_Toc11353)

[8.5 git clone 9](#_Toc21912)

[8.6 git status 9](#_Toc8310)

[8.7 git diff 10](#_Toc3512)

[8.8 git commit 10](#_Toc14164)

[8.9 git reset 10](#_Toc5513)

[8.10 git rm 11](#_Toc7227)

[8.11 git mv 11](#_Toc19678)

[8.12 git branch 11](#_Toc12988)

[8.13 git checkout 12](#_Toc2050)

[8.14 git merge 12](#_Toc26884)

[8.15 git mergetool 12](#_Toc816)

[8.16 git log 12](#_Toc7681)

[8.17 git stash 13](#_Toc28351)

[8.18 git tag 13](#_Toc16743)

[8.19 git fetch 14](#_Toc16762)

[8.20 git pull 15](#_Toc17681)

[8.21 git push 15](#_Toc11015)

[8.22 git remote 16](#_Toc29281)

[8.23 git submodule 16](#_Toc3664)

[8.24 git show 16](#_Toc2940)

[8.25 git shortlog 17](#_Toc27709)

[8.26 git describe 17](#_Toc8093)

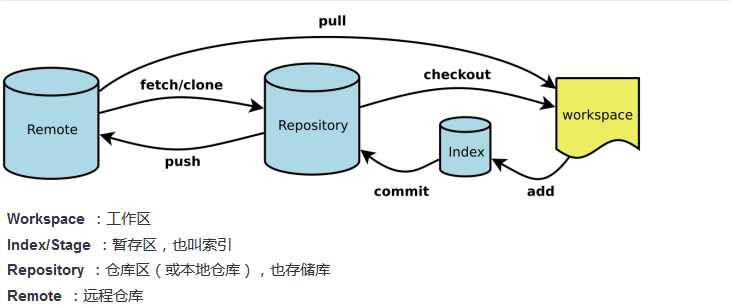
[8.27 git rebase 17](#_Toc10567)

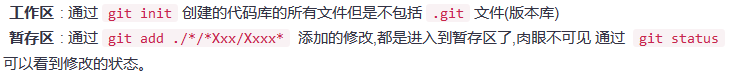
[8.28 git revert 18](#_Toc29398)

1. Git简介

Git是目前很流行的版本控制系统（VCS），**版本控制**是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统,采用版本控制系统(VCS)是个明智的选择。 有了它就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态，可以比较文件的变化细节，查出最后是谁修改了哪个地方，从而找出导致怪异问题出现的原因，又是谁在何时报告了某个功能缺陷等等。 使用版本控制系统通常还意味着，就算你乱来一气把整个项目中的文件改的改，删的删了，这也没有关系，你也照样可以很容易地就恢复到原先的样子。但额外增加的工作量却微乎其微

1. 关键术语





1. Git基础和原理
   1. Git的特点

Git 和其它版本控制系统(包括 Subversion 和近似工具)的主要差别在于 Git 对待数据的方法。 概念上来区分，其它大部分系统以文件变更列表的方式存储信息。 这类系统(CVS、Subversion、Perforce、Bazaar 等等)将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。存储每个文件与初始版本的差异

Git 不按照以上方式对待或保存数据。 反之，Git 更像是把数据看作是对小型文件系统的一组快照。 每次你提交更新，或在 Git 中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。 为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。 Git 对待数据更像是一个 快照流

* 1. Git保证完整性

Git 中所有数据在存储前都计算校验和，然后以校验和来引用。 这意味着不可能在 Git 不知情时更改任何文件内容或目录内容。 这个功能建构在 Git 底层，是构成 Git 哲学不可或缺的部分。 若你在传送过程中丢失信息或损坏文件，Git 就能发现。

1. Git安装

4.1 Linux系统安装

$ sudo yum -y install git

4.2 基于Debian的发行版

$ sudo apt-get install git

4.3 Unix 风格的系统上安装步骤

<https://git-scm.com/download/linux>

4.4 mac上安装

<https://git-scm.com/downloads>

4.5 window上安装

<https://git-scm.com/download/win>

1. Git使用前配置

5.1 用户信息

$ git config --global user.name "yating.li"

$ git config --global user.email "yating.li@hand\_china.com"

如果使用了 --global 选项，那么该命令只需要运行一次，因为之后无论你在该系统上做任何事情， Git 都会使用这些信息

5.2 文本编辑器

既然用户信息已经设置完毕，你可以配置默认文本编辑器了，当 Git 需要您输入信息时会调用它。 如果未配置，Git 会使用操作系统默认的文本编辑器，通常是 Vim。 如果你想使用不同的文本编辑器，例如 Emacs，可以这样做

$ git config --global core.editor emacs

5.3 检查配置信息

$ git config --list

$ git config user.name（某一项的配置）

5.4 检查版本信息

$ git --version

5.5 获取帮助

$ git help

$ git help config（获取config命令手册）

1. Git快速入门

6.1 什么是远程仓库

Repository(仓库)包含的内容 - Git的目标是管理一个工程，或者说是一些文件的集合，以跟踪它们的变化。Git使用Repository来存储这些信息

6.2 获取Git仓库

有两种取得 Git 项目仓库的方法。第一种是从一个服务器克隆一个现有的 Git 仓库。第二种是在现有项目或目录下导入所有文件到 Git 中

6.3 克隆现有Git仓库

首先参考上面的Git使用前配置，如果配置好了，接下来在自己的磁盘里新建一个文件夹，然后右击选择



输入命令：

$ git clone <http://git.oschina.net/yiibai/git-start.git>

这会在当前目录下创建一个名为 “git-start.git” 的目录，并在这个目录下初始化一个 .git 文件夹，从远程仓库拉取下所有数据放入 .git 文件夹，然后从中读取最新版本的文件的拷贝

自定义仓库名称

$ git clone http://git.oschina.net/yiibai/git-start.git mygit-start

6.4 初始化Git仓库

$ git init

6.5 检查当前文件状态

$ git status

6.6 文件状态简览

$ git status -s 或者 $ git status --short

新添加的未跟踪文件前面有 ?? 标记，新添加到暂存区中的文件前面有 A 标记，修改过的文件前面有 M 标记。 你可能注意到了 M 有两个可以出现的位置，出现在右边的 M 表示该文件被修改了但是还没放入暂存区，出现在靠左边的 M 表示该文件被修改了并放入了暂存区

1. Git工作流程

将Git的一个存储库克隆为工作副本。

可以通过添加/编辑文件修改工作副本。

如有必要，还可以通过让其他开发人员一起来更改/更新工作副本。

在提交之前查看更改。

提交更改：如果一切正常，那么将您的更改推送到存储库。

提交后，如果意识到某些错误并修改错误后，则将最后一个正确的修改提交并将推送到存储库。

1. Git常用命令

8.1 git config

git config命令用于获取并设置存储库或全局选项。这些变量可以控制Git的外观和操作的各个方面。

8.1.1配置文件的存储位置

/etc/gitconfig 文件：包含了适用于系统所有用户和所有库的值。如果你传递参数选项’—system’ 给 git config，它将明确的读和写这个文件。

~/.gitconfig 文件 ：具体到你的用户。你可以通过传递—global 选项使Git 读或写这个特定的文件。

位于git目录的config文件 (也就是 .git/config) ：无论当前在用的库是什么，特定指向该单一的库。每个级别重写前一个级别的值。因此，在.git/config中的值覆盖了在/etc/gitconfig中的同一个值。

8.1.2配置用户名和密码

$ git config user.name "yating.li"

$ git config user.email "yating.li@hand-china.com"

8.1.3配置编辑器

可以配置默认的文本编辑器，Git在需要你输入一些消息时会使用该文本编辑器。缺省情况下，Git使用系统的缺省编辑器，这通常可能是 vi 或者 vim 。如果想使用一个不同的文本编辑器，例如：Emacs，可以按照如下操作：

$ git config --global core.editor emacs

8.1.4配置比较工具

你可能需要配置的有用的选项是缺省的比较工具它用来解决合并时的冲突。例如，想使用 vimdiff 作为比较工具:

$ git config --global merge.tool vimdiff

Git可以接受 kdiff3, tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, 和 opendiff 作为有效的合并工具。也可以设置一个客户端的工具

8.1.5检查配置

如果想检查你的设置，可以使用以下命令来列出Git可以在该处找到的所有的设置:

$ git config --list

可能会看到一个关键字出现多次(如这里的:user.name就有两个值)，这是因为Git从不同的文件中(例如：/etc/gitconfig以及~/.gitconfig)读取相同的关键字。在这种情况下，对每个唯一的关键字，Git使用最后的那个值。  
也可以查看Git认为的一个特定的关键字目前的值，使用如下命令

git config {key} 例如：$ git config user.name

8.1.6添加/删除配置项

参数：--add

实例如下：

$ git config --local --add site.name ‘liyating’

参数：--unset

实例如下：

$ git config --local --unset site.name

8.1.7获取帮助

$ git help

$ git help config（获取config命令手册）

8.2 git help

git help命令显示有关Git的帮助信息。

想要查看 help 命令如何使用，可以使用以下命令 -

$ git help help

8.3 git init

git init命令创建一个空的Git仓库或重新初始化一个现有仓库。

命令如下：$ git init

8.4 git add

git add命令将文件内容添加到索引(将修改添加到暂存区)。也就是将要提交的文件的信息添加到索引库

以下是一些示例：

添加documentation目录及其子目录下所有\*.txt文件的内容：

$ git add documentation/\*.txt

将所有 git-\*.sh 脚本内容添加：

$ git add git-\*.sh

$ git add . # 将所有修改添加到暂存区

$ git add \* # Ant风格添加修改

$ git add \*Controller # 将以Controller结尾的文件的所有修改添加到暂存区

$ git add Hello\* # 将所有以Hello开头的文件的修改添加到暂存区 例如:HelloWorld.txt,Hello.java,HelloGit.txt ...

$ git add Hello? # 将以Hello开头后面只有一位的文件的修改提交到暂存区 例如:Hello1.txt,HelloA.java 如果是HelloGit.txt或者Hello.java是不会被添加的

8.5 git clone

git clone命令将存储库克隆到新目录中，命令如下：

git clone -b develop ‘版本库的网址’

克隆岛指定目录 命令如下：

git clone -b develop ‘版本库的网址’ /e/develop\_project/

8.6 git status

git status命令用于显示工作目录和暂存区的状态。使用此命令能看到那些修改被暂存到了, 哪些没有, 哪些文件没有被Git tracked到。git status不显示已经commit到项目历史中去的信息。看项目历史的信息要使用git log

只列出所有已经被git管理的且被修改但没提交的文件，命令如下：

git status -uno

8.7 git diff

git diff命令用于显示提交和工作树等之间的更改。此命令比较的是工作目录中当前文件和暂存区域快照之间的差异,也就是修改之后还没有暂存起来的变化内容。

a）**比较工作区与暂存区**

git diff

b）**比较工作区与最新本地版本库**

git diff HEAD 如果HEAD指向的是master分支，那么HEAD还可以换成master

c）**比较工作区与指定commit-id的差异**

git diff commit-id

d）**比较两个commit-id之间的差异**

git diff commit-id1 commit-id2

8.8 git commit

git commit命令用于将更改记录(提交)到存储库。将索引的当前内容与描述更改的用户和日志消息一起存储在新的提交中。

**git commit . -m “message”**

这种是比较常见的用法，-m 参数表示可以直接输入后面的“message”，如果不加 -m参数，那么是不能直接输入message的，而是会调用一个编辑器一般是vim来让你输入这个message

8.9 git reset

git reset命令用于将当前HEAD复位到指定状态。一般用于撤消之前的一些操作(如：git add,git commit等)。

a）回滚添加操作

git reset

b）回滚最近一次提交

git reset --soft HEAD^

当提交了之后，又发现代码没有提交完整，或者想重新编辑一下提交的信息，可执行此命令，相当于把提交的文件从本地仓库中拿走。

c）回退到上个版本

git reset --hard HEAD^

d）退到/进到 指定commit的版本

git reset --hard commit\_id

e）仅仅只是撤销已提交的版本库，不会修改暂存区和工作区

git reset --soft commit\_id

f）仅仅只是撤销已提交的版本库和暂存区，不会修改工作区

git reset --mixed commit\_id

g）彻底将工作区、暂存区和版本库记录恢复到指定的版本库

git reset --hard commit\_id

8.10 git rm

git rm命令用于从工作区和索引中删除文件。

a）删除文件

git rm text1.txt

b）删除文件夹

git rm -r mydir

c）删除暂存区里面的文件

git rm --cache 文件名

8.11 git mv

git mv命令用于移动或重命名文件，目录或符号链接。

1. 重命名

git mv README.md README

1. 移动文件或者文件夹

git mv b/b.txt a/

8.12 git branch

git branch命令用于列出，创建或删除分支。

**a)查看当前有哪些分支**

git branch

**b)新建一个分支**

git branch dev2

**c)切换分支**

git checkout dev2

**创建远程分支**

git push origin hzb

**d)查看远程分支**

git branch -r

**e)查看本地及远程分支**

git branch -a

**f)修改分支名称**

git branch -m dev2 dev3

**g)删除远程分支**

git push origin --delete dev2

**h)删除本地分支**

git branch -D dev2

**i)查看****本地分支与远程分支的映射关系**

git branch -vv

**j)创建当前分支与远程分支的映射关系**

git branch -u origin/master

**k)创建当前分支与远程分支的映射关系**

git branch --set-upstream-to origin/master

**l)撤销当前分支与远程分支的映射关系**

git branch --unset-upstream

8.13 git checkout

git checkout命令用于切换分支或恢复工作树文件。git checkout是git最常用的命令之一，同时也是一个很危险的命令，因为这条命令会重写工作区。

**a)切换分支**

git checkout dev2

**b)根据指定的分支创建一个新的分支并切换到创建的分支**

git checkout -b branch\_name origin/master

**c)创建分支并拉取指定tag的代码**

git checkout -b branch\_name tag\_name

8.14 git merge

git merge命令用于将两个或两个以上的开发历史加入(合并)一起。

**a)**将分支dev合并到当前分支中，自动进行新的提交

git merge dev

**a)**将分支dev合并到当前分支中，但不要自动进行新的提交

git merge --no-commit dev

8.15 git mergetool

git mergetool命令用于运行合并冲突解决工具来解决合并冲突。

8.16 git log

git log命令用于显示提交日志信息。

**a)**显示整个提交历史记录，但跳过合并

**git log** --no-merges

**b)**显示指定分支下指定文件的提交记录

git log master dev.txt

**c)**显示最近两周指定文件的提交记录

git log --since="2 weeks ago" -- dev.txt

**d)**显示最近三次的提交

git log -3

**e)**显示指定作者的提交

git log --author=maxsu

**f)查看历史命令**

git reflog

8.17 git stash

git stash命令用于将更改储藏在脏工作目录中。

**a)**把所有未提交的修改（包括暂存的和非暂存的）都保存起来，用于后续恢复当前工作目录，需要说明一点，stash是本地的，不会通过git push命令上传到git server上。实际应用中推荐给每个stash加一个message，用于记录版本，使用git stash save取代git stash命令

git stash

git stash save "test-stash"

**b)重新应用缓存stash，**这个指令将缓存堆栈中的第一个stash删除，并将对应修改应用到当前的工作目录，后面可加上名字指定要使用哪个stash

git stash pop

**c)重新应用缓存stash，**这个指令将缓存堆栈中的stash多次应用到工作目录中，但并不删除stash拷贝，后面可加上名字指定要使用哪个stash

git stash apply

**d)查看现用stash**

git stash list

**e)移除stash**

git stash drop 名字

**f)移除所有缓存的stash**

git stash clear

**g)查看指定stash的diff**

git stash show -p 或者 git stash show 名字 -p

Tip:git stash会缓存下列文件：

添加到暂存区的修改（staged changes）

Git跟踪的但并未添加到暂存区的修改（unstaged changes）

但不会缓存以下文件：

在工作目录中新的文件（untracked files）

被忽略的文件（ignored files）

git stash命令提供了参数用于缓存上面两种类型的文件。使用-u或者--include-untracked可以stash untracked文件。使用-a或者--all命令可以stash当前目录下的所有修改。

8.18 git tag

git tag命令用于创建，列出，删除或验证使用GPG签名的标签对象。同大多数 VCS 一样，Git 也可以对某一时间点上的版本打上标签。人们在发布某个软件版本(比如 v1.0 等等)的时候，经常这么做。

**a)列出已有的标签**

git tag

**b)查看标签基本信息**

git show tag\_name

**c)列出指定条件的标签**

git tag -l 'v1.0.2.\*'

**d)创建标签**

git tag -a v1.0 -m 'my version 1.0'

**e)删除标签**

git tag -d v1.0

**f)标签推到远端服务器**

git push origin tag\_name

**g)一次性把所有标签推倒远端服务器**

git push origin --tags

**h)删除远端服务器标签**

git push origin :refs/tags/test\_tag

等同于

git push origin :tag\_name

**i)在某个commit\_id上打上tag**

git tag test\_tag commit\_id

**j)切换到指定的tag**

git checkout tag\_name

**k)创建分支并拉取指定tag的代码**

git checkout -b branch\_name tag\_name

Tip:tag相当于一个快照，是不能直接更改它的代码的，如果要在tag的基础上做修改，需要新建一个分支。命令参考git tag的k)

8.19 git fetch

git fetch命令用于从另一个存储库下载对象和引用。

**a)更新所有远程分支代码到本地仓库**

git fetch

**b)更新指定远程分支代码到本地仓库**

git fetch origin branch\_name

git fetch <远程主机名> <分支名>

fetch更新本地仓库两种方式

**方法一**

a)从远程的origin仓库的master分支下载代码到本地的origin master

git fetch origin master

b)比较本地的仓库和远程参考的区别

git log -p master.. origin/master

c)把远程下载下来的代码合并到本地仓库，远程的和本地的合并

git merge origin/master

**方法二**

a)从远程的origin仓库的master分支下载到本地并新建一个分支temp

git fetch origin master:temp

b)比较master分支和temp分支的不同

git diff temp

c)合并temp分支到master分支

git merge temp

d)删除temp分支

git branch -D temp

Tip:理解 fetch 的关键, 是理解 FETCH\_HEAD，FETCH\_HEAD指的是: 某个branch在服务器上的最新状态’。这个列表保存在 .Git/FETCH\_HEAD 文件中

8.20 git pull

git pull <远程主机名> <远程分支名>:<本地分支名>

git pull命令用于从另一个存储库或本地分支获取并集成(整合)。git pull命令的作用是：取回远程主机某个分支的更新，再与本地的指定分支合并

**a)**要取回origin主机的next分支，与本地的master分支合并

git pull origin next:master

**b)**远程分支(next)要与当前分支合并

git pull origin next

**c)**当前分支与远程分支存在追踪关系，git pull就可以省略远程分支名

git pull origin

**d)**当前分支只有一个追踪分支，连远程主机名都可以省略

git pull

Tip:在某些场合，Git会自动在本地分支与远程分支之间，建立一种追踪关系(tracking)。比如，在git clone的时候，所有本地分支默认与远程主机的同名分支，建立追踪关系，也就是说，本地的master分支自动”追踪”origin/master分支。

8.21 git push

git push <远程主机名> <本地分支名>:<远程分支名>

git push命令用于将本地分支的更新，推送到远程主机

**a)**本地的master分支推送到origin主机的master分支

git push origin master:master

**b)**本地的master分支推送到origin主机的master分支。如果master不存在，则会被新建

git push origin master

**c)删除**远程分支

git push origin :master

等同于

git push origin --delete master

**d)**当前分支与远程分支之间存在追踪关系，本地分支和远程分支都可以省略

git push origin

**e)**当前分支只有一个追踪分支，那么主机名都可以省略

git push

**f)**当前分支与多个主机存在追踪关系，可以使用-u选项指定一个默认主机

git push -u origin master

**g)**不管是否存在对应的远程分支，将本地的所有分支都推送到远程主机

git push --all origin

**h)**查看push的结果(会出现图形界面)

gitk branch\_name

8.22 git remote

git remote命令管理一组跟踪的存储库

**a)列出已经存在的远程仓库**

git remote

等同于

git remote show

**b)显示远程仓库的详细信息**

git remote show remote\_name

**c)**列出远程仓库详细信息，在每一个名字后面列出其远程url

git remote -v

**d)添加远程仓库**

git remote add remote\_name remote\_url

**e)删除远程仓库**

git remote rm remote\_name

8.23 git submodule

git submodule命令用于初始化，更新或检查子模块

8.24 git show

git show命令用于显示各种类型的对象

**a)查看标签基本信息**

git show tag\_name

b**)显示标签指向的树**

git show tag\_name^{tree}

**c)显示标签标签指向的提交的主题**

git show -s --format=%s tag\_name^{commit}

**d)显示指定文件的内容，它们在 next 分支的第10次最后一次提交的内容**

git show branch\_name~10:README.md

**e)将Makefile的内容连接到分支主控的头部**

git show master:Makefile master:t/Makefile

8.25 git shortlog

git shortlog命令用于汇总git日志输出

**a)统计commit的次数**

git shortlog

**b)统计commit的次数，**省略每次 commit 的注释，返回一个简单的统计

git shortlog -s

**c)统计commit的次数，**按照数量进行排序

git shortlog -n

**d)综合统计commit的次数**

git shortlog -s -n

8.26 git describe

git describe命令显示离当前提交最近的标签

**a)列出距离当前最近的标签**

git describe --tags

8.27 git rebase

git rebase命令在另一个分支基础之上重新应用，用于把一个分支的修改合并到当前分支。

**a)执行rebase合并master分支到当前分支**

git rebase master

**b)处理完冲突之后，让rebase继续处理**

git rebase --continue

**c)终止rebase操作**

git rebase --abort

**d)rebase操作之后推送到远端分支不成功**

git push --force-with-lease origin master

Tip:git rebase过程相比较git merge合并整合得到的结果没有任何区别，但是通过git rebase衍合能产生一个更为整洁的提交历史。  
如果观察一个衍合过的分支的历史提交记录，看起来会更清楚：仿佛所有修改都是在一根线上先后完成的，尽管实际上它们原来是同时并行发生的。

一般我们使用衍合的目的，是想要得到一个能在远程分支上干净应用的补丁，比如某个项目你不是维护者，但是想帮点忙，最好使用衍合处理。  
先在自己的一个分支进行开发，当准备向主项目提交补丁的时候，根据最新的orgin/master进行一次衍合操作然后再提交，这样维护者就不需要任何整合工作。

实际为：把解决分支补丁同最新主干代码之间的冲突的责任，划转给由提交补丁的人来解决。  
作为维护项目的人只需要根据你提供的仓库地址做一次快进合并，或者直接采纳你提交的补丁。

衍合的风险，请务必遵循如下准则：  
一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行衍合操作。

8.28 git revert

已经push，对于已经把代码push到线上仓库,你回退本地代码其实也想同时回退线上代码,回滚到某个指定的版本,线上,线下代码保持一致

revert 之后你的本地代码会回滚到指定的历史版本,这时你再 git push 既可以把线上的代码更新.(这里不会像reset造成冲突的问题)

revert 使用,需要先找到你想回滚版本唯一的commit标识代码,可以用 git log 或者在adgit搭建的web环境历史提交记录里查看

**a)回退到指定版本**

git revert commit\_id

**b)删除最后一次提交**

git revert HEAD

git push origin master

git log

**c)删除最后一次提交**

**b)git revert与git reset的区别**

git revert是用一次新的commit来回滚之前的commit，git reset是直接删除指定的commit看似达到的效果是一样的,其实完全不同

第一:上面我们说的如果你已经push到线上代码库, reset 删除指定commit以后,你git push可能导致一大堆冲突（或git push -f强制推送）.但是revert 并不会

第二:如果在日后现有分支和历史分支需要合并的时候,reset 恢复部分的代码依然会出现在历史分支里.但是revert 方向提交的commit 并不会出现在历史分支里

第三:reset 是在正常的commit历史中,删除了指定的commit,这时 HEAD 是向后移动了,而 revert 是在正常的commit历史中再commit一次,只不过是反向提交,他的 HEAD 是一直向前的