## git参考资料

http://www.ruanyifeng.com/blog/2012/07/git.html

http://www.cnblogs.com/lee0oo0/archive/2013/06/29/3161883.html

http://www.cnblogs.com/wish123/p/3937851.html

## 为什么使用git而不是svn

一些使用者的总结:

从使用者的角度:

1、svn下载源码很慢，因为svn下载的是一个一个单独的文件，而git传输时进行了压缩。

2、svn随时需要与服务器交互，无论是查看log还是查看以往的版本必须跟服务器相连。

3、各个分支之间的补丁迁移比较麻烦，在git上只要两三个命令就可以完成，但在svn上必须要下载每个分支的代码，然后比较修改，再上传。

从服务器的角度：

1、git版本库占用空间小，尤其是涉及到多个分支的情况，svn的每个分支都是一份代码的copy，而git每个分支只是各个提交点的hash值的集合。分支几乎不占用什么空间。

2、git是分布式管理系统，我们完全可以不对代码进行备份，而svn需要做灾备处理，如果是单点，一旦硬盘坏掉，真个代码库就不能使用了。而git的每一个客户端都相当与一个备份。

## git仓库的特性

git和其他vcs版本管理软件不同，通过git获取一个项目时，会将该项目的所有历史版本都下载到本地。

## git三个工作区域

git将工作区域分为三个：本地数据目录、工作目录、暂存区域

本地数据目录中保存着git仓库的所有信息,包括历史提交记录,分支信息,标签信息等

工作目录中存放着某个版本的所有数据,在工作目录中可以进行对文件的操作,然后提交到git仓库中

暂存区域只不过是个简单的文件，保存着一次提交的文件列表信息,一般都放在 git 目录中。有时候人们会把这个文件叫做索引文件，不过标准说法还是叫暂存区域。暂存区的作用是确定本次要提交的文件

## git工作目录下的文件

git工作目录下的文件分为两种，一种是已跟踪文件，另一种是未跟踪文件。只有处于已跟踪状态的文件才会纳入git的管理范围内。

通过git add 文件名 命令可以将未跟踪状态的文件进行暂存

## git仓库目录下的四个核心文件或目录

在.git目录下有四个重要的文件或目录:HEAD及index文件,objects及refs目录.



objects目录存储所有数据内容,包括提交到git仓库中的所有文件,

refs目录存储指向提交对象的分支指针

HEAD文件存储当前分支名

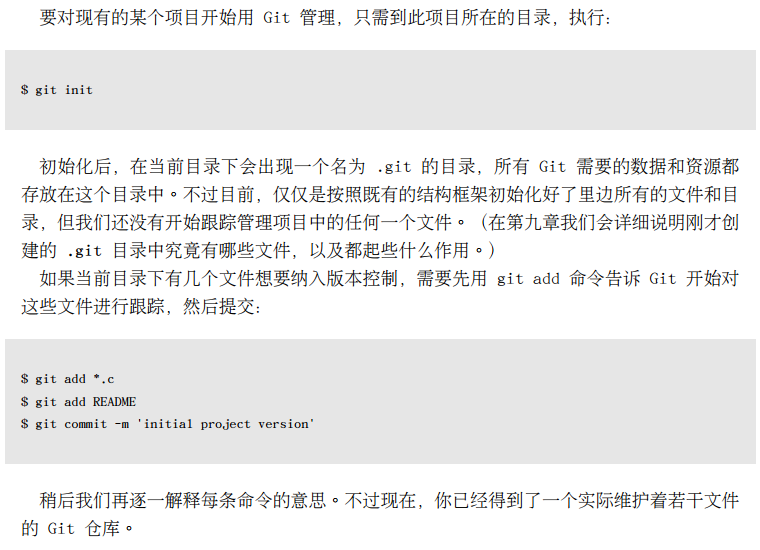
index文件保存了暂存区域信息

## git仓库获取

获取一个git仓库有两种方式

### 将一个已有的项目初始化为一个git仓库

#### 创建一个工作仓库



#### 创建一个纯仓库

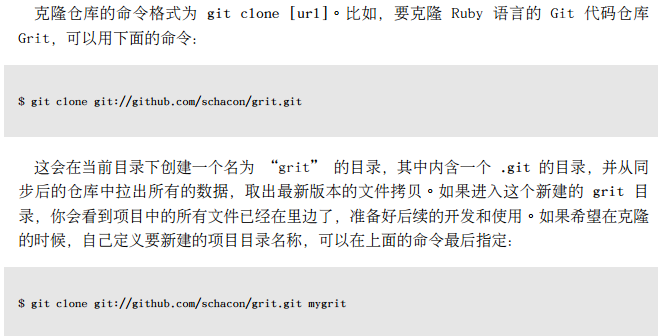
如果想初始化一个仓库作为远程仓库用，那么可以在调用init命令时加上--bare参数，这样创建出来的仓库没有工作目录，.git目录下的文件直接放在git仓库根目录下.





### 复制一个已有的仓库

#### 复制一个仓库作为工作仓库

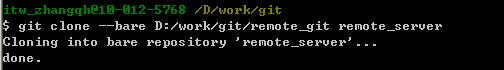


##### 在git路径上添加认证信息

git clone http://itw\_zhangqh:El250814@10.137.199.44/octopus/TestSpringBoot.git

#### 复制一个仓库作为纯仓库

有时候我们可能向建设一个远程仓库,并且将现有仓库中的内容作为仓库的内容,可以在调用clone命令时加--bare参数.



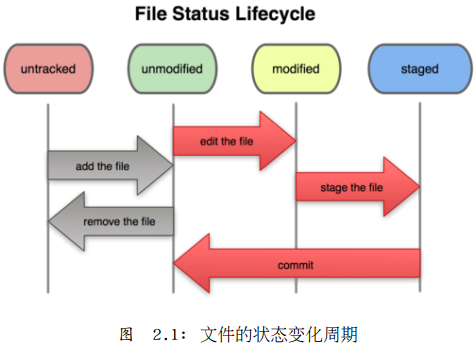
克隆后的仓库



## git文件状态

未跟踪、未修改、已修改、已暂存

### git文件状态周期图



### 已修改-->已暂存

当修改了一个已跟踪的文件后，需要将该文件从已修改状态转为已暂存状态然后再提交到git仓库中。

使用【git add 文件名】 命令可以将文件从已修改转为已暂存状态。

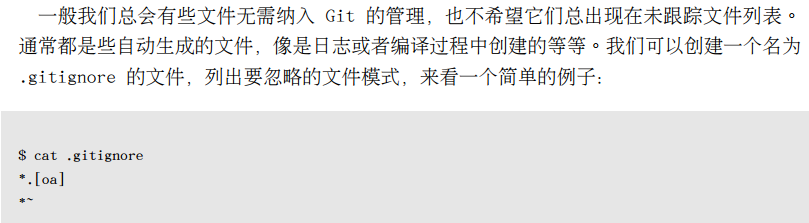
说明：如果执行git add命令后，没有提交又对该文件进行了修改，那么需要重新将该文件进行暂存操作才能将第二次做的修改纳入本次提交中。

### 已暂存-->未修改

工作目录中的文件只有和git仓库中当前分支最新提交的文件保持一致才能算是处于未修改状态。

使用【git commit】命令可以将暂存的文件提交到git仓库中。

### 忽略检查某些文件



经测试,需要将.gitignore写入.gitignore文件中,否则git会将.gitignore文件作为未跟踪文件处理

## 分支

在git中分支本质上就是一个指向commit对象的分支指针。由HEAD指针指向正在工作的分支。

git会使用master作为分支的默认名称。当前工作的分支指针会随着每次提交自动指向最新的commit对象。

## 跟踪分支

### 理解跟踪分支

从远程分支检出的本地分支称为跟踪分支。其实就是跟远程分支绑定的本地分支。

### 跟踪分支的特性

1、在跟踪分支里输入git push，git会自行推断应该向那个服务器的那个分支推送数据，而不需要指定服务器和分支。

2、在跟踪分支里运行git pull会获取所有远程分支，并把他们的数据自动合并到本地分支中。

### 创建跟踪分支

#### git checkout -b 分支名 [远程名]/[分支名]

新建一个跟踪分支,并切换到该分支中。

#### git checkout --track [远程名]/[分支名]

新建一个和远程分支名相同的本地跟踪分支,并切换到新建的分支中。

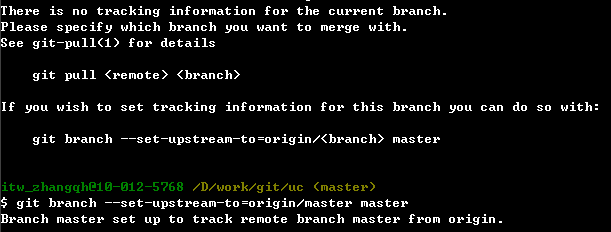
### 删除远程分支

#### git push [远程名] :[分支名]

删除远程仓库中的分支

### 将本地分支与远程分支绑定

git branch --set-upstream-to=origin/<branch> <local-branch>



## 标签

### 分类

#### 轻量级的(lightweight)

轻量级标签就像时个不会变化的分支,实际上他就是个指向特定提交对象的引用.

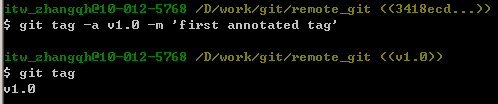
#### 含附注的(annotated)

含附注的标签,实际上是存储在仓库中的一个独立对象,他有自身的校验和信息,包含着标签的名字,电子邮件地址和日期,以及标签说明

### 创建标签

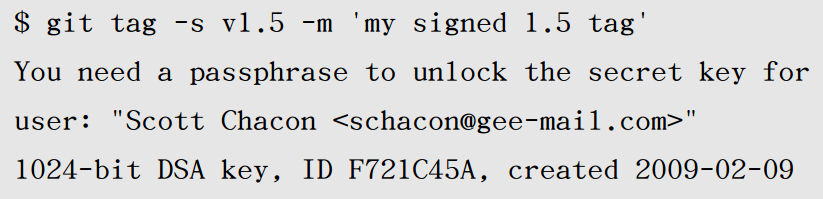
#### 创建含附注的标签

创建一个含附注类型的标签需要使用-a 和 -m参数分别指定标签名和标签附注.-a取annotated的首字母.



#### 创建签署标签

如果你有自己的私钥,还可以用GPG来签署标签,只需要把之前的-a改为-s(取signed的首字母)即可:



#### 创建轻量级标签

如果要创建轻量级标签不需要使用-a 和 -m参数,直接指定标签名即可:



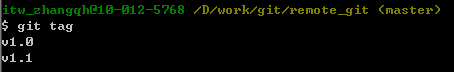
#### 为指定提交打标签

使用 【git tag 标签名 提交名】命令为指定的提交打标签

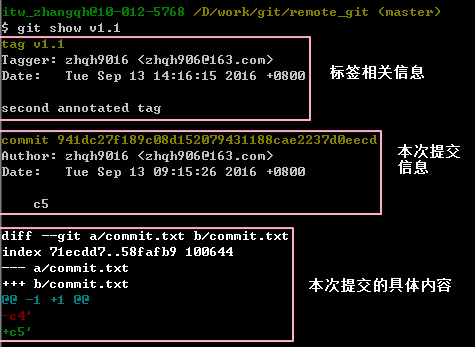
### 查看标签

#### 标签列表

使用git tag命令可以展示出标签列表



#### 展示指定标签的具体内容



### 分享标签

默认情况下，git push并不会把标签传送到远端服务器上，只有通过显示命令才能分享标签到远端仓库。格式命令和推送分支类似，运行 git push origin 【tagname】即可

## Git对象

### 理解git对象

git对象是git存储文件的一种方式。git会为每一个被提交的文件(一次提交可能包含多个文件,这样就会创建多个对象)创建一个git对象,该对象被存入.git/objects文件夹下。

### git对象生成时机

git对象是在执行暂存操作时生成的。并且，对一个修改的文件暂存后会生成了一个git对象，如果没有提交又对该文件进行了修改并进行暂存，那么会生成另一个git对象，之前生成的git对象不会删除。

### 创建对象的方法

生成该对象是使用git的底层命令hash-object命令实现的.

该命令的语法为:git hash-object -w 文件名。其中-w是write的意思，如果不加-w那么只会根据文件内容计算一个校验和并返回而不会将生成的对象存储起来。

例如:



该命令执行完之后会返回一个40位的SHA-1哈希值.该校验和是根据存储的内容和头信息生成的。如下图:



### git对象存储的目录结构

在生成git对象时会生成一个40位的SHA-1哈希值，去前两位作为文件夹名，剩下的38为作为git对象文件的文件名。如上图中的校验和为3dff953473a06d4fcd5c6a85ff8d4619088b1a6d，那么该git对象将会以ff953473a06d4fcd5c6a85ff8d4619088b1a6d作为文件名，存储在.git/objecs/3d文件夹下。

### blob

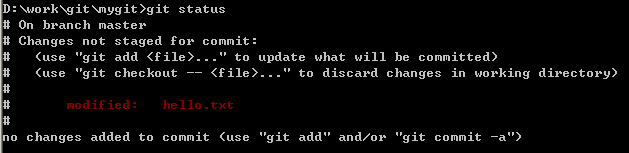
git对象不存储文件名，而仅仅是文件内容，这种对象类型称为blob。

## 高层命令

### 查看状态相关命令

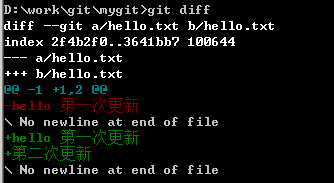
#### git status

该命令可以列出哪些文件做出了修改，哪些文件已被暂存，哪些文件未被跟踪



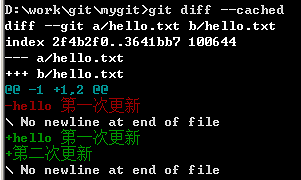
#### git diff

该命令列出所有已修改未暂存的文件，并列出修改的具体细节



#### git diff --cached

该命令列出所有已暂存的文件，并列出已暂存的信息和上次提交版本的具体差异细节



### 提交命令

#### git commit

运行该命令后会提交已暂存的快照，并会弹出版本说明信息输入界面。

#### git commit -m “备注信息”

该命令可以直接提交已暂存的快照信息，并直接指定备注信息

#### git commit -a

该命令允许跳过暂存操作，将已修改的内容自动暂存，然后提交

### 移除命令

#### git rm 文件名

该命令会从工作目录中删除指定的文件，并从暂存区域删除。

#### git rm -f 文件名

强制移除命令，当我们要移除的文件已被修改，或已被暂存，那么需要使用-f来强制移除。

#### git rm -cached 文件名

该命令不会删除工作目录中的文件,只会从暂存区中将该文件删除.

### 移动命令

#### git mv file\_from file\_to

该命令可以实现移动文件的动作,也可以实现重命名的功能.

注意:如果先修改了一个文件,然后再调用git mv命令,那么调用git status会显示两条提示,一是rename提示,一个modified提示,如果不管modified提示直接提交,那么提交时相当于rename操作,如果先将modified暂存再提交,那么提交时会得到一个delete和一个create的操作.

#### 重命名

删除一个文件,添加一个文件,如果这两个文件的内容完全相同,那么git会自动认定为这是一个重命名的动作.如果两个文件的内容不完全相同,那么就认为是删除了一个文件,添加了一个文件.

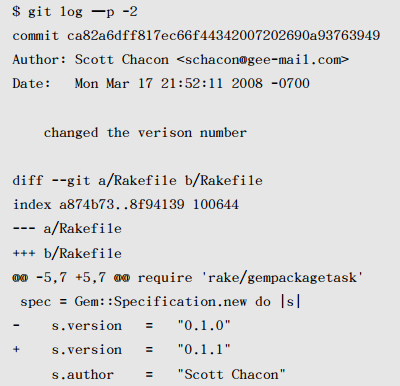
### 历史命令

#### git log

该命令可以查看历史提交列表,包括SHA-1校验和、作者和邮箱地址、提交时间，最后一行展示提交说明。后提交的会排在列表前面

#### git log -p

该命令在基本信息的基础上会显示出每次提交中的具体差异信息

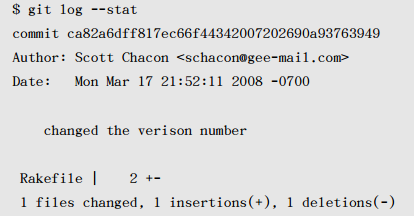


#### git log -(number)

该命令可以指定查看的历史数量

#### git log --stat

该命令会比只用git log多简要的增改行数统计信息

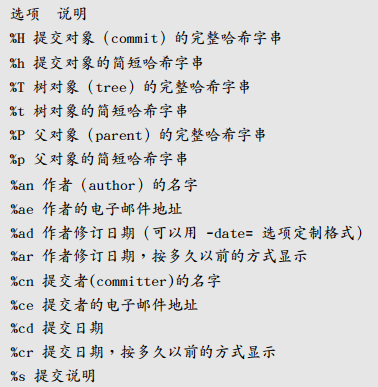


#### git log --pretty

该命令可以指定历史列表的输出格式



常用占位符:

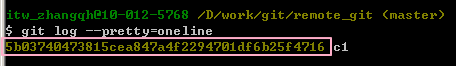


#### git log --abbrev-commit

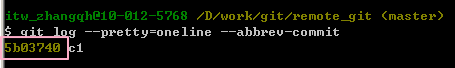
如果在调用git log 的时候传递abbrev-commit参数,输出结果里就会使用简短且唯一的值,默认使用七个字符来表示.

例如:

未使用abbrev-commit参数的输出结果:



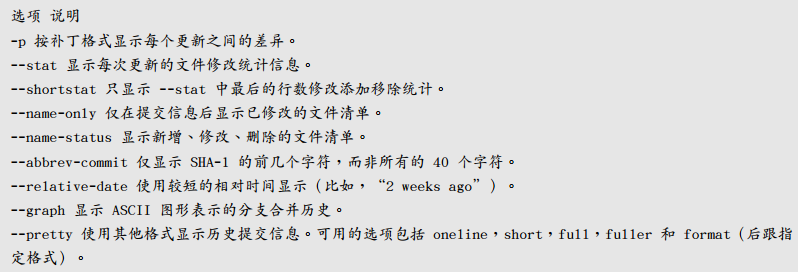
使用abbrev-commit参数后的输出结果:



#### 展示分支和衍合信息的命令

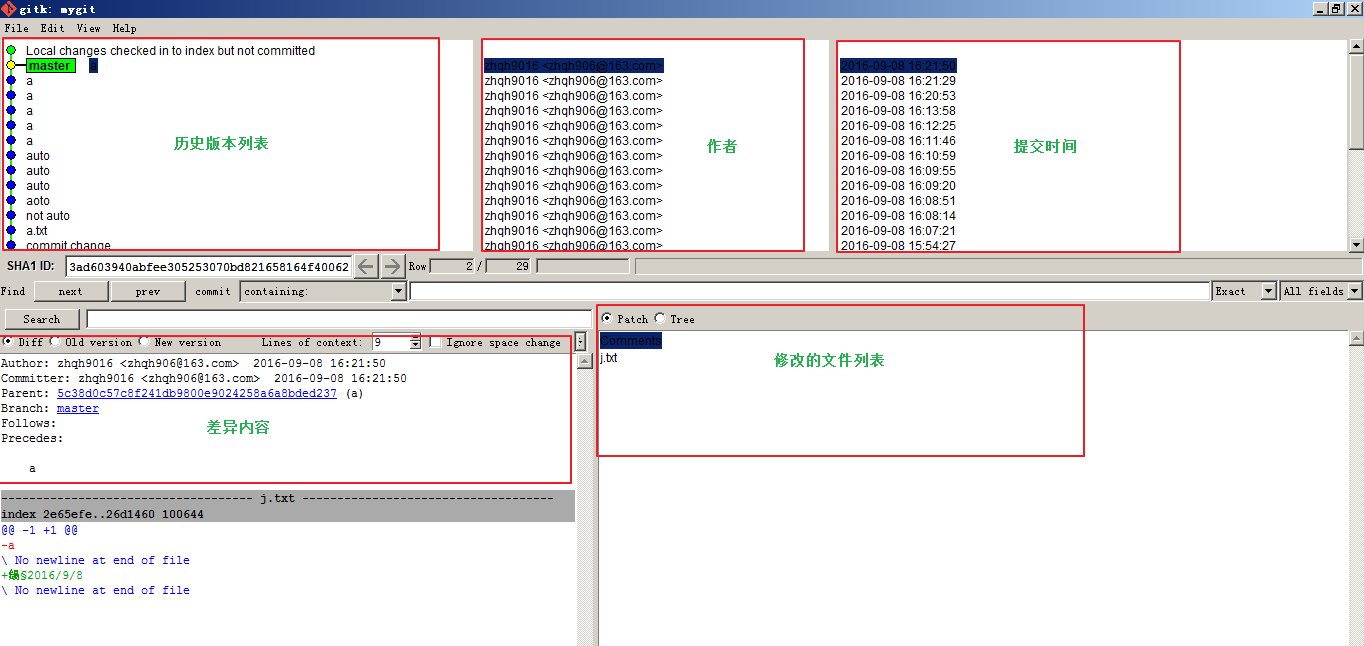


#### 其他历史信息查看选项



### 历史记录可视化工具

在git工作目录中执行gitk命令即可打开可视化工具.



### 撤销命令

#### git commit --amend

该命令重新提交暂存区的快照,并覆盖上一次的提交

#### git reset HEAD <file>

该命令撤销已经暂存的文件,比如说我们执行了git add \*将所有文件都添加到了暂存区,但我们发现一些文件我们并不希望添加进去,这时可以使用git reset HEAD <file>命令取消指定文件的暂存.

#### git checkout -- <file>

该命令可以将指定文件恢复到上一次提交后的状态.

### 分支命令

#### 创建分支

使用【git branch 分支名】命令可以创建一个指向当前commit对象的分支。

注意：运行该命令后只会创建一个新的分支，并不会自动切换到新建的分支中去。

使用【git checkout -b 分支名】命令可以创建一个指向当前commit对象的分支，并切换到新建的分支中。

#### 切换分支

使用【git checkout 分支名】命令可以切换到指定的分支上进行工作。

注意：如果在当前分支中做了修改，未提交就切换到其他分支，那么git会阻止这一操作，必须保证工作目录中的文件处于未修改或未跟踪状态才能切换分支。

#### 删除分支

1、使用 【git branch -d 分支名】命令可以删除分支。

注意：使用该命令时git会检测当前分支有没有和其他分支进行合并，如果没有合并，git会阻止本次删除操作。

2、使用【git branch -D 分支名】命令可以强制删除分支。

注意：使用该命令不会检测被删除的分支是否和其他分支进行了完全合并，直接将分支删除。

#### 合并分支

使用【git merge 分支名】命令可以将指定分支合并到当前分支上。如果没有冲突那么合并后会自动提交，如果有冲突，那么不会提交，需要解决冲突后手动提交。

#### 分支列表

1、使用【git branch】命令可以查看分支列表，效果如下图：



当前工作的分支前面会有\*号标记。

2、使用【git branch -v】命令可以查看分支列表，并展示出各分支最后一次提交的信息



3、使用【git branch --merge】命令可以查看已经跟当前分支合并的分支列表，即所在commit是当前分支直接上游的分支。



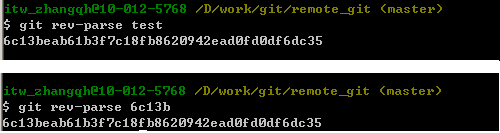
4、使用【git branch --no-merged】命令可以查看尚未和当前分支合并的分支列表。



#### 查看分支对应的commit的SHA值

使用【git rev-parse 分支名】命令可以查看指定分支名对应的提交对象的SHA值。

使用该命令也可以查看一个SHA缩写的全拼

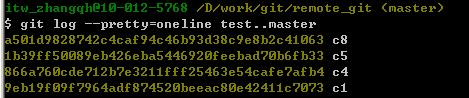


### 提交范围

对提交历史做范围查询

#### 双点

通过双点语法来指明查找范围,可以查找出从一个分支中可以获得,而从另一个分支中不能获得的提交.例如【git log --pretty=oneline test..master】:

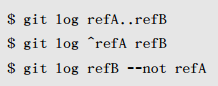


这样就可以查出在test中没有但在master中存在的提交列表.在需要查看实验分支上哪些没有被合并到主分支上的提交时很有用。

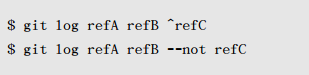
#### 多点

如果想查看一些提交被包含在某些分支中,而不被包含在另一些分支中,可以使用git提供的多点范围查询命令.可以使用^字符或--not指明不希望提交被包含在结果中的分支.

下面三条命令的效果是一样的:

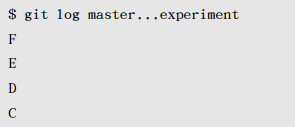


也可以像下面这样使用:

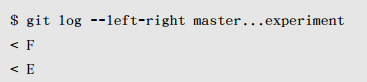


#### 三点

这种范围查询方法可以实现这样的查询效果,可以指定两个分支,查询出所有不被两者同时包含的提交.



使用--left-right参数可以显示出每个提交到底处于那一侧的分支.



### 抓取数据命令

#### git fetch [remote-name]

从远程库抓取数据,但不会自动合并,只是下载到本地库

#### git pull

如果设置了某个分支用于跟踪某个远端仓库的分支,可以使用git pull命令自动抓取数据下来,然后将远端分支自动合并到本地仓库中当前分支.在默认情况下git clone命令本质上就是自动创建了本地的master分支用于跟踪远程仓库中的master分支.

### 推送命令

#### git push [remote-name] [branch-name]

该命令可以将指定的分支推送到远程库中,执行该命令有一个前提,你推送之前没有其他人推送更新,如果在之前有人推送了更新,那么你的推送操作就会被驳回.你必须先把他们的更新抓取到本地,合并到自己的项目中,然后才能推送.

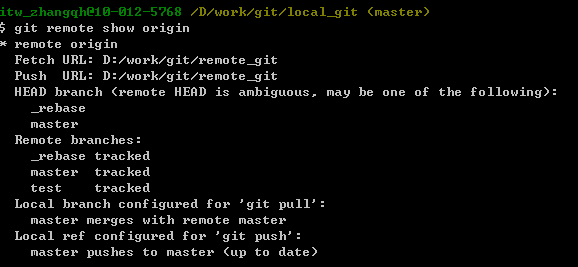
#### git push [remote] local-branch:remote-branch

该命令可以将指定的分支推送到远程库中分支名不相同的分支中。前提和上面的命令相同。

### 查看远程仓库信息

#### git remote show [remote-name]

该命令会列出远程仓库中存在的分支,当前工作分支,git pull和git push的默认配置.



## 底层命令

### 创建git对象

#### git hash-object

该命令可以向git仓库中插入任意类型的内容,并返回一个键值,通过该键值可以在任何时候再取出该内容.



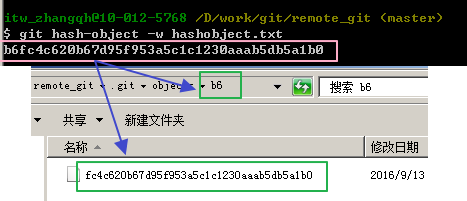
但是这样使用该命令,并不会将上图中的hashobject.txt文件存储到git仓库中,只是根据该文件计算出一个hash值.下图是git仓库存放git对象的目录,在该目录中没有出现b6文件夹:



使用参数-w可以实现将文件写入git仓库的目的.

#### git hash-object -w

加上-w参数后在git仓库中生成了对应的git对象.



### 查看git对象

#### git cat-file -p (git对象)

查看指定git对象的具体内容

#### git cat-file -t (git对象)

查看指定git对象的类型,如blob、tree、commit、parent

#### git cat-file -p {blob对象)

列出blob的具体内容



#### git cat-file -p (tree对象)

列出tree对象的具体内容



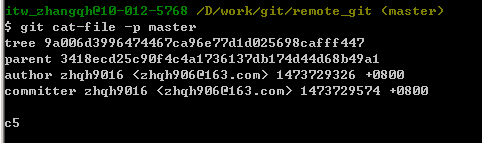
可以使用commit对象^{tree}这样的形式指定tree对象,比如

git cat-file -p master^{tree}

此处master代表所指向的commit对象.

#### git cat-file -p (commit对象)

列出commit对象的具体内容,包括对应的tree对象,parent等



## 命令别名

### 命令别名

有些命令我们可能经常要用,但拼写起来又比较麻烦,我们可以使用git config 为命令设置别名



使用上面的命令就可以为checkout命令起一个别名ck,下次可以使用ck代替checkout来使用,例如:



### 自定义命令

使用命令别名可以创造出新的命令,比方说取消暂存文件时的输入比较繁琐,可以自己设置一个:

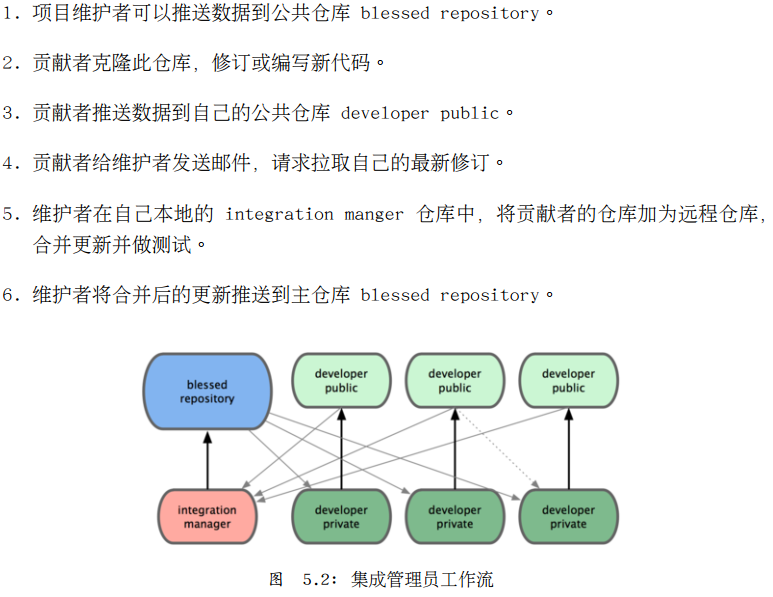


这样下次可以直接使用 git unstage 文件名执行取消暂存操作.

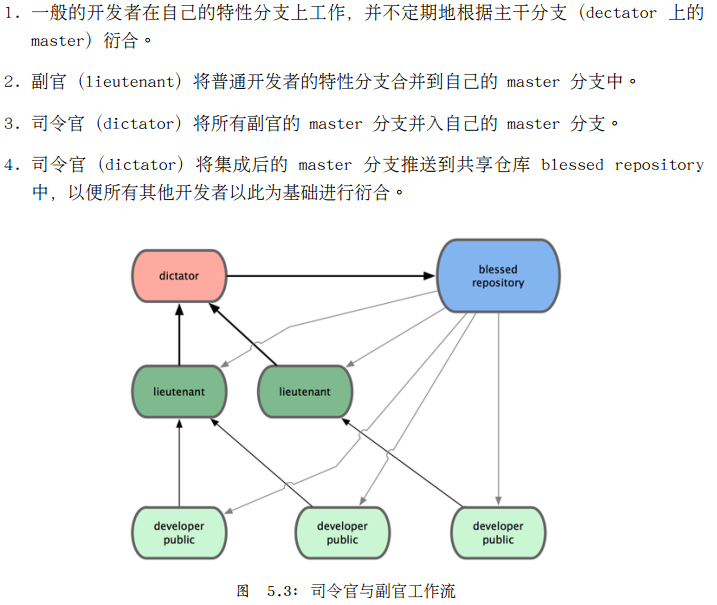


## 工作流程

### 集成管理员工作流

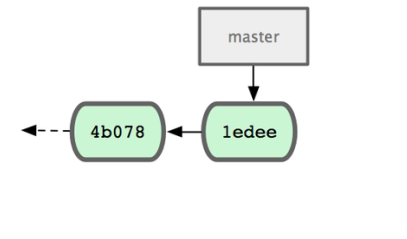


### 司令官与副官工作流

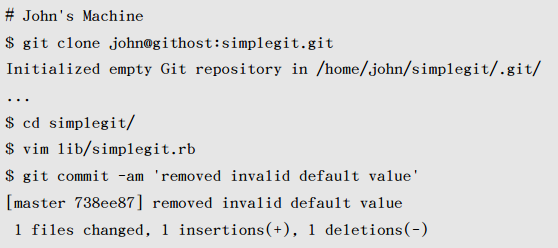


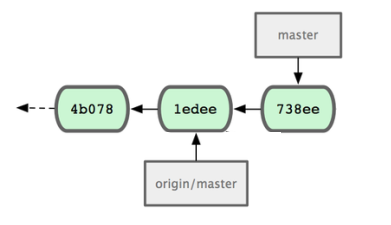
## 一个协作项目的简单流程

现在John和Jessica共同开发一个项目.下图是初始项目的历史结构



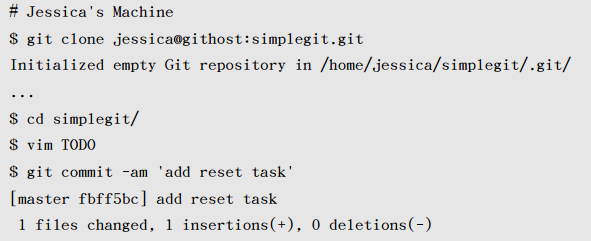
1、John克隆了仓库,做了些更新,在本地提交.



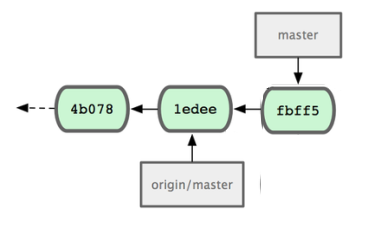


John做出更改后本地的历史提交图

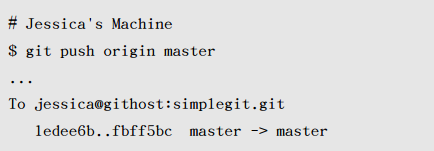
2、Jessica也克隆了仓库,做了些更新,在本地提交.



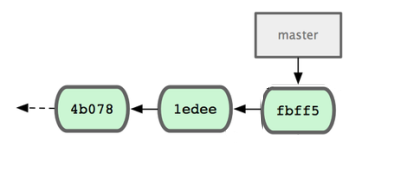
Jessica做出改变后本地的历史提交图如下:



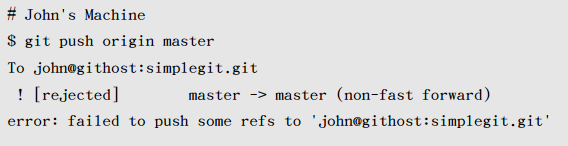
3、Jessica将本地更新推送到服务器上



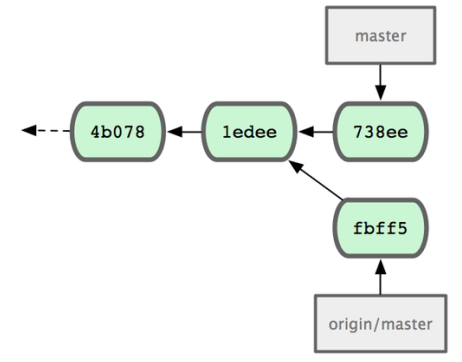
Jessica推送更新后远程仓库中的历史提交图为:



4、John也尝试将更新推送到服务器上。

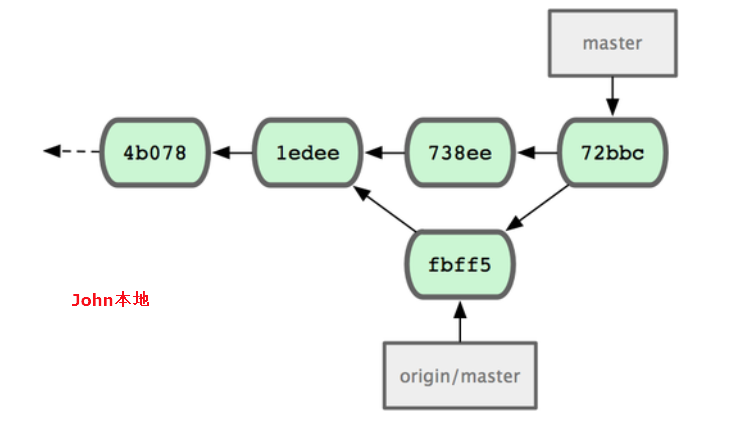


显示John的推送失败了，并且显示远程仓库中的master分支和本地的master分支不能执行快进操作。因为Jessica已经推送了新的数据到服务器上。远程仓库中的master分支并不是本地master分支的直接上游，所以需要先合并再推送。拉取远程仓库的更新后John本地的历史提交结构图应该如下图所示：

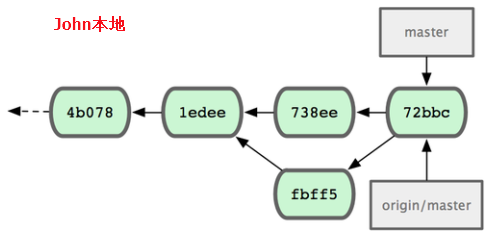


John本地历史结构图

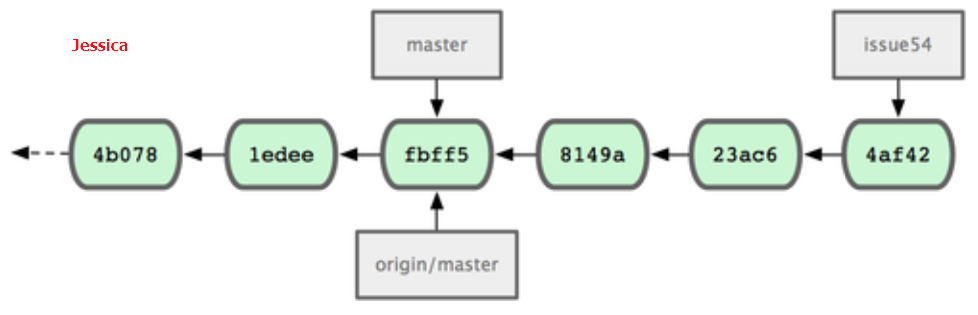
5、John将本地master分支与origin/master分支进行合并



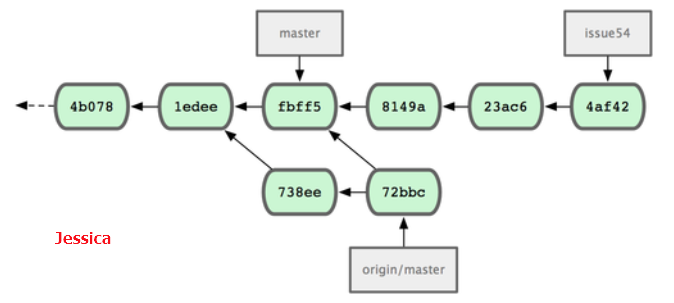
6、John将合并后的提交推送到远程库,推送后John本地的历史提交图为:



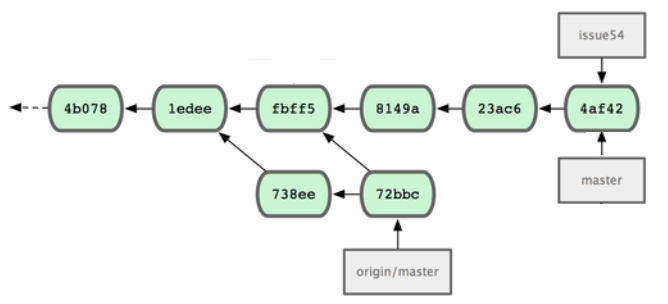
7、在John在做一些处理的过程中,Jessica又做了一些工作,Jessica又开启了另一个特性分支issue54,并做了三次更新,历史提交结构如下:



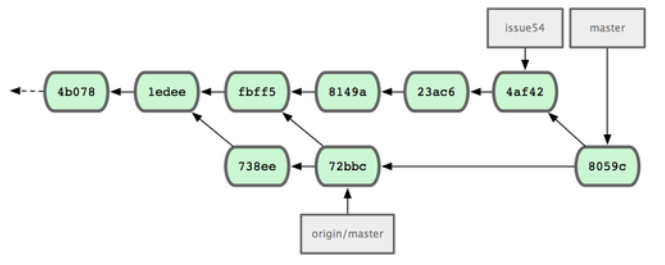
8、Jessica在推送更新之前,先拉取了远程库的数据,此时Jessica本地的历史提交结构图如下:



9、Jessica将特性分支上的工作做完之后将特性分支并到master分支上,此时Jessica本地的历史提交结构为:



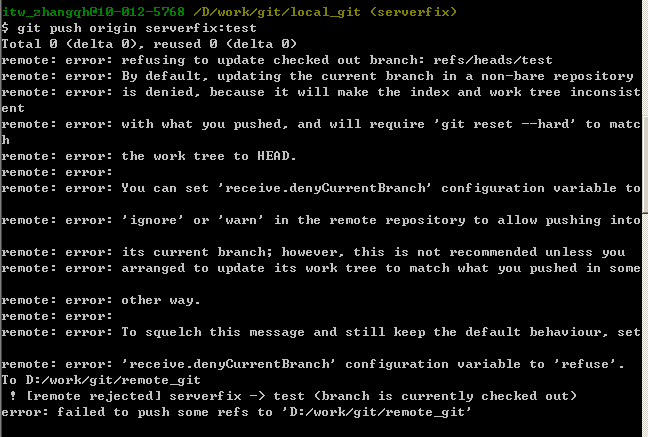
10、最后Jessica将master分支和orgin/master分支进行合并，合并完之后历史提交结构图为：



## 错误

git push origin serverfix:test

当向一个远程库推送时,如果推送的分支在远程库中为当前的工作分支,那么会报如下错误



## 创建一个干净的历史提交历史

在本地可能有很多提交,但是不想将本地的提交历史提交到远程库中,可以使用下面的方法实现

* 在本地新建一个工作分支
* 平时的提交都提交到工作分支上
* 当要推送到远端库时切换到master分支上
* 将远端库中的代码pull到本地
* 切换到工作分支
* 将master分支合并到工作分支上
* 切换到master分支
* 执行git checkout --patch 工作分支名
* 在master上做一次提交
* 将master推到远端库中

## 数据恢复

数据恢复分两种,一是将分支指向一个历史提交,这样恢复的是所有文件,另一种是将指定文件从历史提交中恢复到工作目录中

### 恢复到历史提交

#### git reset --hard <commit>

执行后工作目录、index、HEAD都会和指定的commit一致

#### git reset --soft <commit>

执行后分支和HEAD会指向指定的commit，工作目录和index和命令执行前保持一致

#### git reset --mixed <commit>

执行后分支和HEAD会指向指定的commit，index也会和指定的commit保持一致，工作目录和米宁执行前保持一致

### 恢复指定文件

#### git checkout <commit> -- <file>

从指定的提交中检出指定的文件