**第一章 Git的基本操作方法**

**一.初始化仓库**

*git init*

此时会在当前目录＊假如为($WORK）生成一个.git的目录文件．这个.git目录就是Ｇit仓库．其中存放的是我们所提交的文档索引内容，Git 可基于文档索引内容对其所管理的文档进行内容追踪，从而实现文档的版本控制。工作树是包含 .git 的目录$WORK.如图所示

[](http://blog.haohtml.com/wp-content/uploads/2011/04/git-ware-worktree.bmp)

**二.生成快照，并将快照存放到索引区域**

*git add .*

此命令将当前目录下所有文件生成快照．此命令通常在忘记哪些文档更新，编辑或者删除的情况下使用.也可以指定一个或者多个文件，多个文件之间用空格隔开．

*git add a.txt b.java*

在我们使用的过程中．有时候编码器产生的一些临时文件是没有必要提交git管理的，可以用以下方法来进行忽略.

*echo 'zh/' >> .gitignore*

这样对于zh目录及内容是不会被git所管理的.

注意如果这里使用 > 定向符的话，则.gitignore文件只能保留最后一次的内容的．

**三．提交索引到仓库中**

*git commit*

此时系统会自动调用系统默认的文本编辑器，让用户输入版本更新备注信息，对于简单的备注信息要，可以用

*git commit -m "这里是版本更新信息"*

另外对于git commit命令还有一个-a的参数.此选项可以将所有被修改的文档或者删除的文档的当前状态提交到仓库中．

注意：对于只是修改或者删除了已被git管理的文档，是没有必要使用git add命令的．

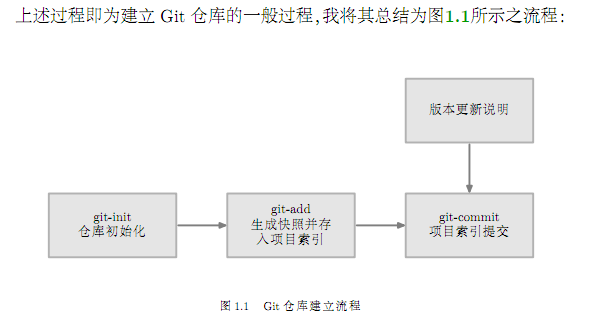
实例一：

*cp -R $WORK/.git /tmp/m2doc.git  
cd /tmp  
git clone m2doc.git m2doc-copy*

这样就实现了通过git复制一份内容到m2doc-copy目录里了．如果省略工作树名，则会使用与工作树同名的工作树名．对于clone的源git也可以是各种协议网络地址，如[https://github.com/joyent/node.git](https://github.com/joyent/node.git" \t "_blank)

**说明：** 在使用git之前，需要对进行个人信息登记一下，这样就可以划分相应的责任

*git config --global user.name "Your Name Comes Here"  
git config --global user.email you@yourdomain.example.com*

上述过程即为建立 Git 仓库的一般过程，我将其总结为图1.1所示之流程： [](http://blogstatic.u.qiniudn.com/wp-content/uploads/2011/04/git-floower.png)

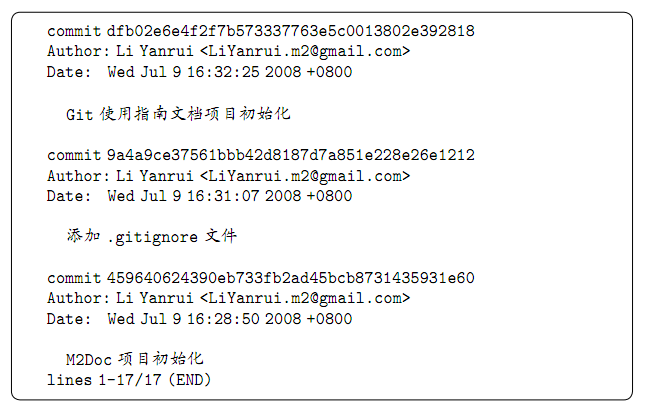
**四.查看版本历史**

*git log*

如果想看一下每一次版本的大致变动情况，可以使用

*git log --stat --summary*

返回结果以下图所示：

[](http://blog.haohtml.com/wp-content/uploads/2011/04/git-log-start-summary.bmp)

上图中每行commit后面的那一串数字为项目版本号．这些都是唯一的．默认由git自动生成.用以标示项目的某一次更新．我们可以根据此版本号用git show命令查看．

*$ git show dfb02e6e4f2f7b573337763e5c0013802e39281*

此方法可能使用的时候感觉特别的麻烦，其实可以用另外一种方便的方法．

*$ git show dfb02 # 一般只使用版本号的前几个字符即可  
$ git show HEAD # 显示当前分支的最新版本的更新细*

每一个项目版本号通常都对应存在一个父版本号，也就是项目的前一次版本状态。可使用如下命令查看当前项目版本的父版本更新细节：

*$ git show HEAD^ # 查看 HEAD 的父版本更新细节  
$ git show HEAD^^ # 查看 HEAD 的祖父版本更新细节  
$ git show HEAD~4 # 查看 HEAD 的祖父之祖父的版本更新细节*

你可以对项目版本号进行自定义，然后就可以使用自定义的版本号查看对应的项目版本更新细节：

*$ git tag v0.1 dfb02  
$ git show*

实际上，上述命令并非是真正的进行版本号自定义，只是制造了一个 tag 对象而已，这在进行项目版本对外发布时比较有用。本文档后续章节会对 tag 的一些细节进行介绍。

**撤销与恢复**

git-reset 命令有三个选项：--mixed 、 --soft 和 --hard 。我们在日常使用中仅使用前两个选项；第三个选项由于杀伤力太大，容易损坏项目仓库，需谨慎使用。 --mixed 是 git-reset 的默认选项，它的作用是重置索引内容，将其定位到指定的项目版本，而不改变你的工作树中的所有内容，只是提示你有哪些文件还未更新 --soft 选项既不触动索引的位置，也不改变工作树中的任何内容，但是会要求它们处于一个良好的次序之内。该选项会保留你在工作树中的所有更新并使之处于待提交状态。 如果执行过上面的命令，想查看实际效果，可以使用命令

*git log  
git reset --hard SHA1\_HASH  
git-status*

**第2章 基于Git的团队协同开发**

**1.两人的协同开发**

修改一个项目m2ge.原来由Lyr一个人开发,后来由于时间过于紧张,于是邀请他的地友Tzc来参与M2GE库. Lyr的M2GE工作树为/work/m2ge,Tzc可通过以下命令获取同Lyr同样的工作树

*$cd work  
$git clone lyr@192.168.0.7:~/work/m2ge m2ge*

于是两个人各自在自己的机器上开始了项目的开发,这里于是用到了我们上面讲的内容了.等一段时间后,他们感觉着很有必要将各自所做的工作合并起来,然后再进行新的开发阶段.由于Lyr为主要开发者,对开发的项目比较的熟悉,这里我们将把Tzc的工作合并到Lyr的工作树中.执行以下操作

*$cd ~/work/m2ge  
$git pull tzc@192.168.0.5:~/work/m2ge*

这里我们使用了git pull命令,可以将远程的项目信息拉取到本地工作树中 .在本机进行合并.与其相反的命令就是git push了.操作正好相反,是将本地的工作树内容传送到远程的工作树中,在远程工作树中进行合并操作. 在合并的过程中,如果两个人修改的是不同的文件的话,合并的比较的顺利的.如果他人共同对同一个文件进行了修改,在合并的时候,系统会提示修改冲突,这个时候需要人工手动进行合并一下,然后再将合并的结果提交到仓库中.

**2.三人及更多人如何协同操作**

实际工作中,许多大些的项目都是由三个人以上来完成的.在合并的过程上,如果我们仍然采取上面的合并方法.所有的合并工作交给一个人来处理的话,那可能需要很长的时间和精力的.那有没有办法将合并的工作减小呢,或者分解一下呢.答案是有的.那就是让每个人从远程工作树取的信息合并到自己本地的工作树中.下面我们来介绍一下这种方法. 利用 git-pull 与 git-push 命令，那么在一个协同周期之内，除了 Lyr 之外(因为其它人是从他这里取的工作树,所以他不用进行任何操作)，其余三人的项目开发流程大致如下：

*$ git clone lyr@192.168.0.7:~/work/m2ge ... 项目开发 ...  
$ git add 改动的文件  
$ git commit  
$ git pull ... 解决版本合并问题 ...  
$ git push*

上一节所给出的三人及三人以上的协同工作模式有些不合理，譬如 Lyr 过于特殊，别人都要 git-pull 与 git-push，唯独他不需要。现在要剥夺他的这一特权，最有效的办法就是将 M2GE 仓库建立在实验室的服务器上。

首先， Lyr 通过 SSH 登录到服务器，寻找合适位置，建立 m2ge.git 目录，譬如/project/m2ge.git ，然后初始化一个空仓库，以此作为 M2GE 仓库：

*$ mkdir -p ~/project/m2ge.git  
$ cd ~/project/m2ge.git  
$ git --bare init --shared*

上述操作中， git-init 命令的 --bare 选项可以让 m2ge.git 目录等价于一个仓库。也就是说，m2ge.git 本来是一个工作树，但是 --bare 选项将本应当存放在m2ge.git/.git 中的仓库内容全部放置在 m2ge.git 目录下，就好像仓库完全的裸露在工作树中，所以称之为赤裸的仓库。

然后， Lyr 将自己机器上已经接受 Git 管理的 m2ge 仓库推送到服务器端的m2ge.git 仓库：

*$ cd ~/work/m2ge  
$ git push m2@192.168.0.2:~/project/m2ge.git master*

上述 git-push 命令中出现的 master 参数的含义将在下一章讲述，此处可略过不谈。现在，大家已经得到了 M2GE 仓库的最初版本，并且可以使用 git-clone 命令在本地创建工作目录：

*$ git clone m2@192.168.0.2:~/project/m2ge.git*

之后，我们就可以开始一个又一个协同周期，服务器上的 m2ge.git 仓库将会逐次记录着每位协同开发者的版本更新提交，此基本过程可参考上一节所述内容来理解。

2.5 总结本章讲述了基于 Git 最基本的多人协同工作模式，并引入了三个新的 Git 命令： git-clone、git-pull 与 git-push。基于这三个命令并配合上一章所讲述 Git 基本操作，足以实现 M2GE 初级阶段的协同开发。

**第 3 章 项目分支管理**

Git 最为世人称道的就是它那强大的项目分支管理功能，现在较为流行的版本控制系统，诸如 CVS、SVN 等，虽然也提供了项目分支管理功能，但是可用性极低。对于 Git 而言，管理多个项目分支如探囊取物耳。本章主要讲述 Git 的项目分支管理的基本知识以及如何利用这一功能形成更有章法的项目协同开发模式。

**1.如何创建项目分支**

Git默认只有一个分支的，那就是master分支，此分支称为主分支．并由Ｇit自动产生.我们在上面用的其实就是一个主分支的．

要产生一个分支，操作如下:

*git branch local*

默认初始化其它分支时，分支内容完全赞同于主分支的．但在local分支进行的所有操作都不会影响到主分析.所以可以在local分支进行任意操作.

要查看项目有多少个分支．使用命令:

*$git branch  
local  
\*master*

其中master为当前分支，前面有一个\*号．所谓的主分支其实相对于当前用户所在的分支来说，各分支之间是没有主次之分的．只是人们习惯上将默认的分支称作主分支而已．  
上面我们虽然创建了local分支，但Git不会自动将当前分支切换到local分支，此时我们使用命令git checkout来实现，如下：

*$git checkout local*

另外也可以用git checkout filename 来恢复改动的文件到git的当前版本,见：<http://blog.haohtml.com/archives/14167>

**2.分支的合并**

当我们在一个分支下(如local)进行了一系统的添加,编辑，删除及版本的变更操作后．需要将此分支合并到master分支里.合并分支可以用git merge命令实现,如下:

*$git checkout master # 将当前分支切换为master  
$git merge local # 将local分支与当前分支合并*

当一个分支与master合并后，此分支就没有什么用了，我们可以将其删除：

*$git branch -d local*

注意：上面的-d参数只能删除已经合并过的分支，对于未合并的分支是无法删除的．如果确实想实现删除未合并的分支的话，可以用-D参数

*$git branch -D localtc*

**3.项目新的协同开发模式**

这里我们还以上面的项目M2GE为例,在实验室服务器上已经建立了M2GE仓库，以供所有开成员同步本地使用．现在以Lyr为主，看看他在此项目上开发工作的一天中的工作过程.

首先，Lyr 需要更新自己机器上的工作树，并查看实验室其他成员的版本更新信息.

*$git pull  
$git log*

然后，Lyr 开始建立一个新的项目分支，将其命名为 lyr(此分支内容与master完全一样，可以理解成复制了一份内容，但并不是真正的复制)，并将当前分支切换为该分支．

*$git branch lyr  
$git checkout lyr*

然后这一天中剩余的大部分时间，Lyr 都在自己所建立的项目分支lyr上工作，譬如增加了 3 个新的接口及相关测试程序，并对原有接口做了一些修改。一天的工作完成后，他有必要将这一天的工作与 本人机器上M2GE 仓库的 master 分支进行合并，然后删除 lyr 分支：

*$git checkout master  
$git merge lyr  
$git branch -d lyr*

现在，Lyr 已经将这一天的工作反映到自己机器上的 M2GE master 分支上了，他最后要做的是将其它开发成员的工作内容git pull到本地，与本地的master合并，然后将其master推送到服务器端的 M2GE 仓库，以使项目其他成员能够分享他的工作。(这里要注意，在推送版本更新之前，需要使用 git pull 命令将这一天中其他成员对服务器端的 M2GE 的更新拉过来合并到自己的 master 分支，然后才可以将自己的版本更新推送到服务器上的 M2GE 仓库，)具体操作如下：

*1. 使用 git pull 命令将其它开发者对服务端的更新拉过来（由于其它成员都同时对服务端的M2GE的进行了更新，所以这里只需要从服务端拉取数据即可），与本地的master分支合并,更新本地工作树；  
2. 若出现版本合并冲突，并且 Git 无法自动合并，需要手工合并，然后将合并结果提交到本地 master 分支；  
3. 使用 git-push 命令将本地 master 分支更新推送到服务器 M2GE 仓库中。*

上面这个操作，其实也可以直接push到服务端进行操作的．但现在是以Lyr主要开发者，所以一般情况下是由主开发者在本地更新，然后再提交到服务端的公共仓库.

目前，对于我们而言，在基于 Git 的 M2GE 协同开发过程中，引入分支管理功能，可有效防止因个人操作不当而导致向服务器 M2GE 仓库提交太多的脏数据。另外，也有效保持了本地项目主分支的干净，避免了频繁 git-clone 服务器端的M2GE 仓库来恢复本地的项目主分支。

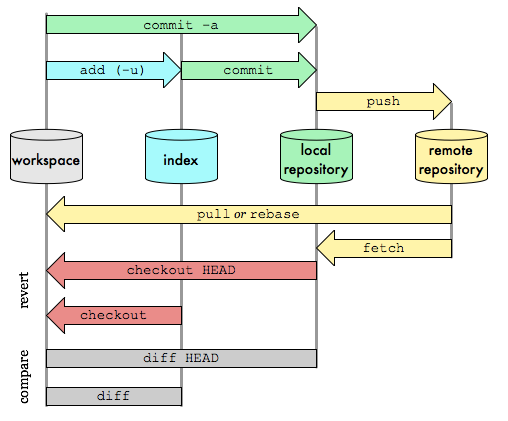
**常见问题：**

*warning: Your console font probably doesn't support Unicode. If you experience strange characters in the output, consider switching to a TrueType font such as Lucida Console!*

解决办法：

*$ git config --global core.autocrlf true*

**git工作流程图解：**

[](http://blogstatic.u.qiniudn.com/wp-content/uploads/2011/06/3.git_architecture.png)

另一篇git介绍的比较清楚：<http://nettedfish.sinaapp.com/blog/2013/08/05/deep-into-git-with-diagrams/>

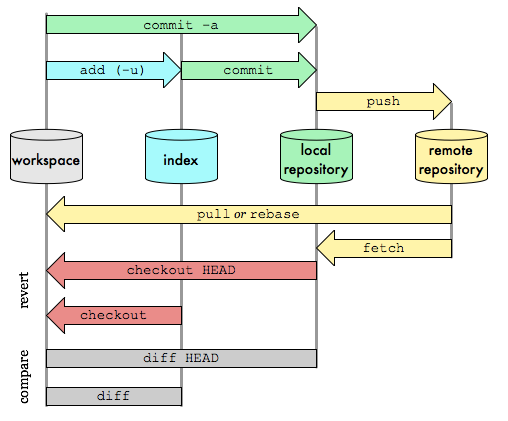
### 相关文章

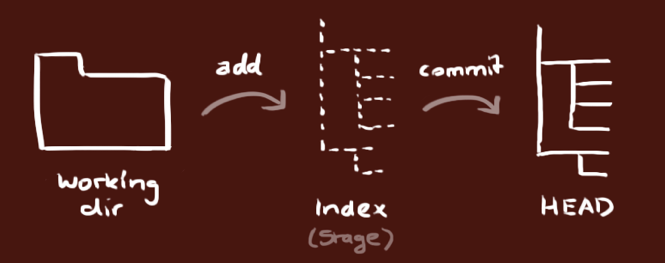
* [gitflow 开发流程](http://blog.haohtml.com/archives/15317)
* [Git Flow印象(转)](http://blog.haohtml.com/archives/15170)
* [利用git中的tab标签发布新系统](http://blog.haohtml.com/archives/15160)
* [Git分支管理策略(转)-推荐](http://blog.haohtml.com/archives/15158)
* [git stash和git stash pop的用法](http://blog.haohtml.com/archives/14865)

# 图解GIT

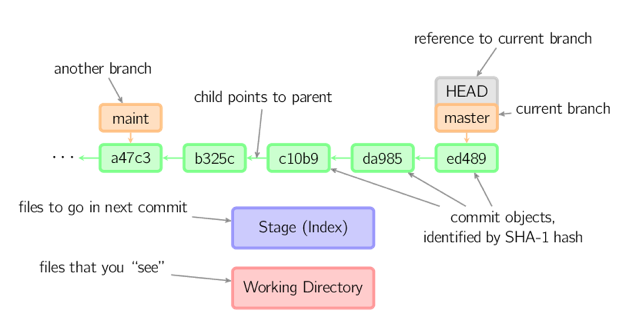
GIT是非常优秀的版本控制工具，但是苦于git那晦涩难懂的man pages，还有众多的命令选项和怪异的用法，git有点难学。这篇文章分享我学习过程中收藏的一些好图，并围绕这些图讲讲我对git的理解，希望对大家有所帮助。

### GIT工作流程

了解git，首先要弄清楚对象在被git管理过程中所处的4个阶段，分别是：工作目录、index(又称为暂存区)、本地仓库和远程仓库。从时间先后来讲，工作目录的内容是你当前看到的，也是最新的；index区标记了你当前工作目录中，哪些内容是被git管理的；而本地仓库保存了对象被提交过的各个版本，比起工作目录和暂存区的内容，它要更旧一些；远程仓库是本地仓库的异地备份，远程仓库的内容可能被分布在多个地点的处于协作关系的本地仓库修改，因此它可能与本地仓库同步，也可能不同步，但是它的内容是最旧的。任何对象都是在工作目录中诞生和被修改；任何修改都是从进入index区才开始被版本控制；只有把修改提交到本地仓库，该修改才能在仓库中留下痕迹；而要与协作者分享本地的修改，可以把它们push到远程仓库来共享。图最上方的add、commit、push等，展示了git仓库的产生过程。反过来，我们可以从远程历史仓库中获得本地仓库的最后一个版本，clone到本地，从本地检出对象的各个版本到index暂存区或工作目录中，从而实现任何对象或整个仓库的任意阶段状态的”回滚”。当正向和反向都能自由切换后，git就强大到无所不能了。

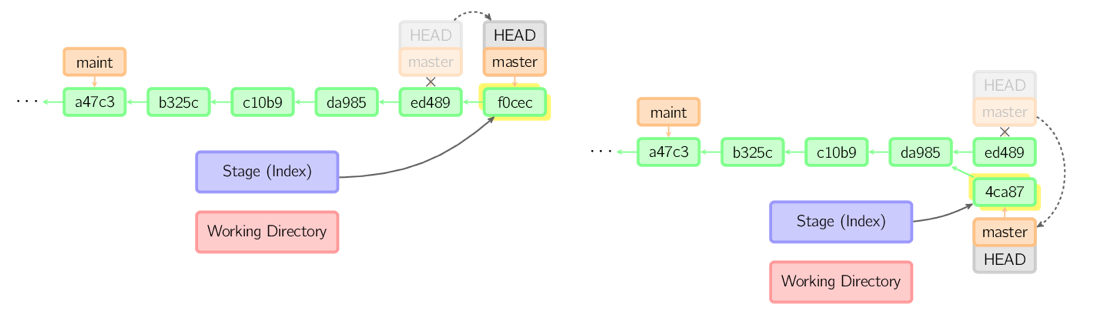
一开始接触这些概念可能比较绕，其实在git入门阶段，可以先抛开远程仓库不看，只了解管理本地仓库的”3棵树”就够了。如下图：   


**常用GIT命令**

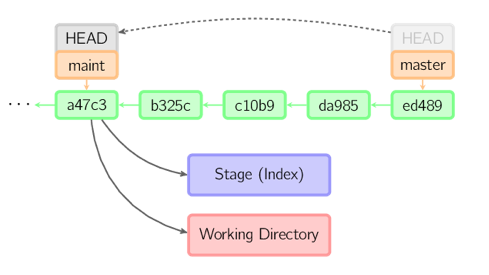
在开始之前，我们需要把下面的图看懂：  


* HEAD，头，它始终指向当前所处分支的最新的提交点。你所处的分支变化了，或者产生了新的提交点，HEAD就会跟着改变。
* working directory，它是你的工作目录，也是当前你看到的东西。你的工作目录是与版本、分支相关的。
* stage的东西虽然看不见，但是执行git status就会看到哪些对象的修改将在下一次commit的时候被放进本地仓库。这些东西称为stage。

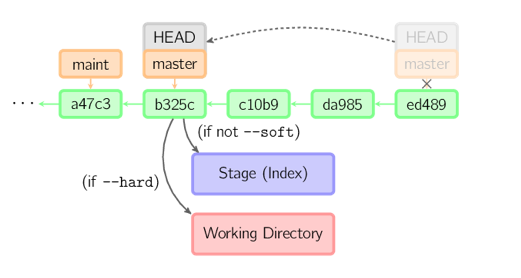
#### commit

commit把暂存区的内容存入到本地仓库，并使得当前分支的HEAD向后移动一个提交点。如果对最后一次commit不满意，可以使用git commit --amend来进行撤销，修改之后再提交。如图所示的，ed489被4ca87取代，但是git log里看不到ed489的影子，这也正是amend的本意：原地修改，让上一次提交不露痕迹。   


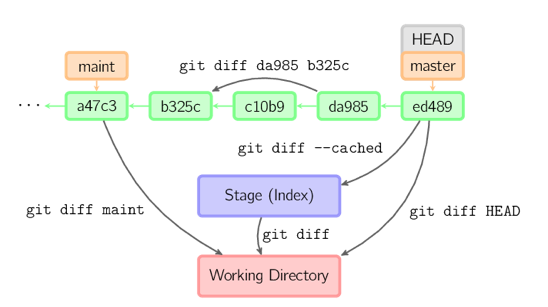
#### checkout

checkout用来检出并切换分支。checkout成功后，HEAD会指向被检出分支的最后一次提交点。对应的，工作目录、暂存区也都会与当前的分支进行匹配。下图是执行git checkout maint后的结果：   


#### reset

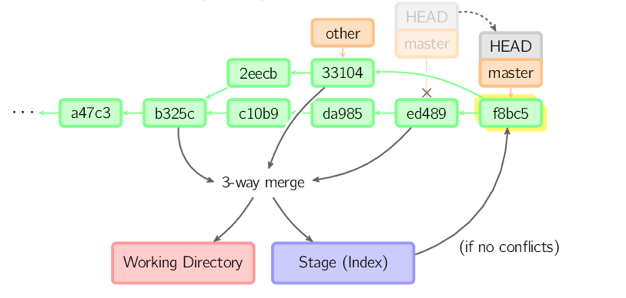
reset命令把当前分支指向另一个位置，并且相应的变动工作目录和索引。如下图，执行git reset HEAD~3后，当前分支相当于回滚了3个提交点，由ed489回到了b325c：   
   
reset有3种常用的模式：  
—soft，只改变提交点，暂存区和工作目录的内容都不改变  
—mixed，改变提交点，同时改变暂存区的内容。这是默认的回滚方式  
—hard，暂存区、工作目录的内容都会被修改到与提交点完全一致的状态

#### diff

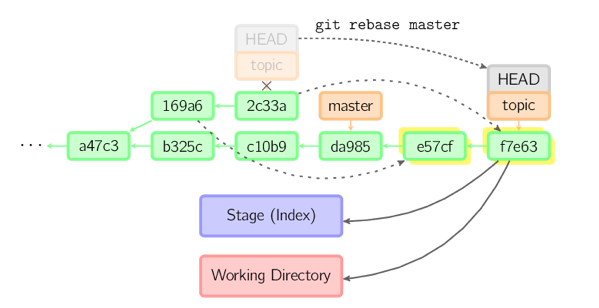
我们在commit、merge、rebase、打patch之前，通常都需要看看这次提交都干了些什么，于是diff命令就派上用场了：  
   
来比较下上图中5种不同的diff方式：

* 比较不同的提交点之间的异同，用git diff 提交点1 提交点2
* 比较当前分支与其他分支的异同，用git diff 其他分支名称
* 在当前分支内部进行比较，比较最新提交点与当前工作目录，用git diff HEAD
* 在当前分支内部进行比较，比较最新提交点与暂存区的内容，用git diff --cached
* 在当前分支内部进行比较，比较暂存区与当前工作目录，用git diff  
  看起来有点复杂？是的，记不住的时候就看看这些图吧。

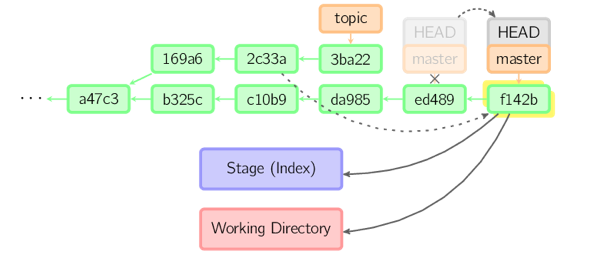
#### merge

merge命令把不同的分支合并起来。如下图，HEAD处于master分支的ed489提交点上，other分支处于33104提交点上，项目负责人看了下觉得other分支的代码写的不错，于是想把代码合并到master分支，因此直接执行git merge other，如果没有发生冲突，other就成功合并到master分支了。   


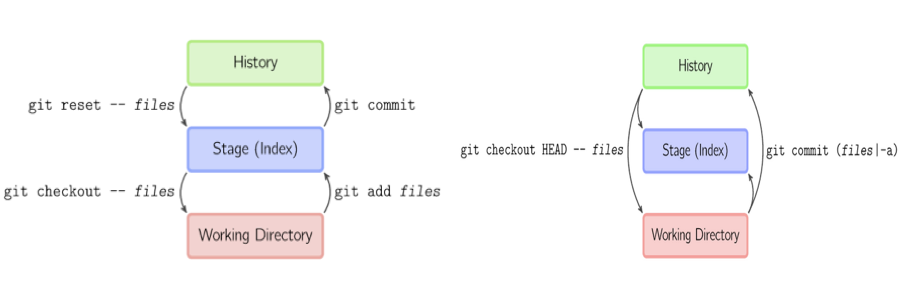
#### rebase

rebase又称为衍合，是合并的另外一种选择。merge把两个分支合并到一起进行提交，无论从它们公共的父节点开始(如上图，other分支与master分支公共的父节点b325c)，被合并的分支(other分支)发生过多少次提交，合并都只会在当前的分支上产生一次提交日志，如上图的f8bc5。所以merge产生的提交日志不是线性的，万一某天需要回滚，就只能把merge整体回滚。而rebase可以理解为verbosely merge，完全重演下图分支topic的演化过程到master分支上。如下图：  
  
在开始阶段，我们处于topic分支上，执行git rebase master，那么169a6和2c33a上发生的事情都在master分支上重演一遍，分别对应于master分支上的e57cf和f7e63，最后checkout切换回到topic分支。这一点与merge是一样的，合并前后所处的分支并没有改变。git rebase master，通俗的解释就是，topic分支之前在master分支的HEAD上进行开发，过了一段时间，topic分支发现master分支有了新的提交，于是，topic分支想把这些新的提交拿过来使用，此时topic可以选择merge，也可以选择rebase，而rebase就相当于是移植，或者嫁接，这样master分支的最新修改就等于是通过插队的方式进入topic分支了。我在研究redis源码的时候，经常会加一些注释，每当上游作者对redis源代码有了更新，我就会采用rebase的方式来保持我的代码是最新的，同时，我的注释也不会丢失。

#### cherry-pick

cherry-pick命令复制一个提交点所做的工作，把它完整的应用到当前分支的某个提交点上。rebase可以认为是自动化的线性的cherry-pick。   
例如执行git cherry-pick 2c33a :   


#### 正反过程对比

理解了上面最晦涩的几个命令，我们来从正反两个方向对比下版本在本地的3个阶段之间是如何转化的。如下图(history就是本地仓库)：   
  
如果觉得从本地工作目录到本地历史库每次都要经过index暂存区过渡不方便，可以采用图形右边的方式，把两步合并为一步。

### Some Tips

#### 如何管理空文件夹

git本身不会对空文件夹进行版本控制，如果希望项目被clone后自带一些空目录，那么可以：

* 在要被管理的空目录下创建.gitignore文件。
* 在.gitignore文件内写入如下代码。第一行忽略所有文件。第二行除了.gitignore文件不被忽略。   
  \*  
  !.gitignore

#### 用git实现远程备份

git本地仓库的初始化采用git init . 即可。如果需要搭建异地备份，除了可以使用开源的gitlab等来搭建服务器外，还可以git --bare init . 来初始化远服务器仓库，然后把本地仓库推送到服务器仓库。这样可以轻松实现代码或文档的异地备份，即使把代码备份到本地也是可以的(这或许可以避免rm -rf \*的悲剧哦)：  
git remote add origin git://192.168.1.2/abc.git   
然后建立到远程的ssh信任关系，enjoy yourself！

#### 无密码登陆git服务器

调试技巧：ssh -v [git@github.com](mailto:git@github.com)

### GIT与知识管理

日常工作中，我习惯用git来管理github和本地的代码；也用git来管理各种配置文件，例如.vimrc、.bashrc等，以及各种技术文档。我的博客用vim+markdown来编写，也用git来做版本控制，很方便。感谢linus大神为我们写了这么好的工具！

欢迎批评指正！

注：本文的图片绝大部分来自于[visual-git-guide](http://marklodato.github.io/visual-git-guide/index-en.html)，这也同时是一篇非常好的git教程。

nettedfish 原创文章，转载请注明出处 Aug 5th, 2013  [tools](http://nettedfish.sinaapp.com/blog/categories/tools/)