[一、 Git代码状态转换图 1](#_Toc398128314)

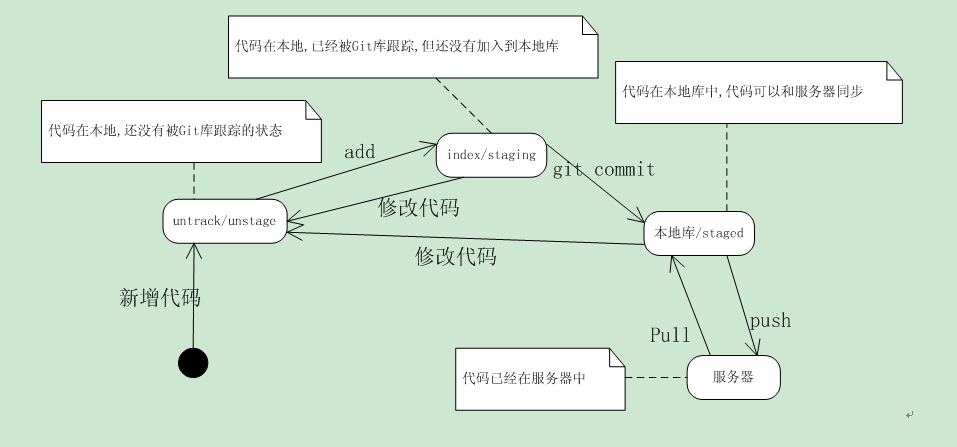
[二、 git的个人本地使用及操作 1](#_Toc398128315)

[三、 git的团队开发及操作 4](#_Toc398128316)

[四、 git的分支管理 5](#_Toc398128317)

[五、 git 忽略机制 5](#_Toc398128318)

1. Git代码状态转换图



其中：未被Git跟踪的状态为unstage状态

已经被Git跟踪的状态为stage状态，因此包括staging状态和staged状态

untrack files是指尚未被git所管理的文件；changed but not updated是指文件被git管理，并且发生了改变，但改动还没被git管理；这两种状态，都可以看成是改动还没被git管理的状态，我们这里称非stage状态。

changes to be commited是指进入stage状态的文件，stage是commit和未管理之间的一个状态，也有别名叫index状态，也就是git已经管理了这些改动，但是还没完成提交。

.gitignore中的文件，不会出现在以上三个状态中。

1. git的个人本地使用及操作

1，  创建Git库

cd 源码目录

git init #初始化 在源码目录内生成一个.git的目录

2，  注册用户信息

git config --global user.name XXX 用户名

git config --global user.email XXX 用户邮箱

git config –list #查看用户信息

3，  向git库中添加或删除文件

git add XX #加单个文件

git add . #加所有

git add [path］会把对应目录或文件，添加到stage状态

git add . 会把当前所有的untrack files和changed but not updated添加到stage状态

4，  向版本库提交变化

git commit –m “XXXX” #直接添加简单提交信息,添加注释

[注：直接调用git commit命令，会提示填写注释。通过如下方式在命令行就填写提交注释：git commit -m "Initial commit of gittutor reposistory"。 注意，和CVS不同，git的提交注释必须不能为空，否则就会提交失败。   
       git commit还有一个 -a的参数，可以将那些没有通过git add标识的变化一并强行提交，但是不建议使用这种方式。   
每一次提交，git就会为全局代码建立一个唯一的commit标识代码，用户可以通过git reset命令恢复到任意一次提交时的代码。   
       git commit –-amend –m “message” （在一个commit id上不断修改提交的内容） ]

git status #查看当前代码库的状态

git log #查看版本信息

git log –p #查看版本信息并显示每次修改的diff

git show sdjf974654dd…. #查看指定版本信息

#(show后面为每次提交系统自动生成的一串哈希值)[注：git show sdji97也会显示本版本修改的diff ]

git show sdji97 #一般只使用版本号的前几个字符即可

 [git log参数注：**-p 按补丁格式显示每个更新之间的差异（具体改动）**。

**--stat 显示每次更新的文件修改统计信息（变更概要）**。

--shortstat 只显示 --stat 中最后的行数修改添加移除统计。

--name-only 仅在提交信息后显示已修改的文件清单。

--name-status 显示新增、修改、删除的文件清单。

--abbrev-commit 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。

--relative-date 使用较短的相对时间显示（比如，“2 weeks ago”）。

--graph 显示 ASCII 图形表示的分支合并历史。

--pretty 使用其他格式显示历史提交信息。可用的选项包括 oneline，short，full，fuller 和 format（后跟指定格式）。

-<n>（<n>为数字），显示最近的<n>条日志。]

5，  撤销与恢复

git reset

git reset --hard #回到原来编辑的地方,改动会丢失。

#（同样适用于团队对于其他人的修改恢复）

git reset --hard sdv143kvf…... #可回到指定的版本

#(hard后面为每次提交系统自动生成的一串哈希值)

git reset [path] 会改变path指定的文件或目录的stage状态，到非stage状

git reset 会将所有stage的文件状态，都改变成非stage状

回退1个change的写法就是git reset HEAD^，2个为HEAD^^，3个为HEAD~3，以此类推。

git revert

git revert是**撤销**某次操作，此次操作之前和之后的commit都会**被保留**，并且 会把这次 撤销 作为一次**最新的提交**；

* git revert HEAD                  撤销前一次 commit
* git revert HEAD^               撤销前前一次 commit
* git revert commit （比如：fa042ce57ebbe5bb9c8db709f719cec2c58ee7ff）撤销指定的版本，撤销也会作为一次提交进行保存。

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，**不影响之前提交的内容**。

[git reset](http://blog.csdn.net/wh_19910525/article/details/7439915)是撤销某次提交，但是此次之后的修改都会被退回到暂存区；

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

## [注：撤消操作

接下来，我们会介绍一些基本的撤消操作相关的命令。请注意，**有些操作并不总是可以撤消的，所以请务必谨慎小心，一旦失误，就有可能丢失部分工作成果**修改最后一次提交

有时候我们提交完了才发现漏掉了几个文件没有加，或者**提交信息写错了**。想要撤消刚才的提交操作，**可以使用--amend 选项重新提交**：

$ **git commit --amend -m"修改 提交 说明"**

此命令将使用当前的暂存区域快照提交。**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**，但将要提交的文件快照和之前的一样。

启动文本编辑器后，会看到上次提交时的说明，编辑它确认没问题后保存退出，就会使用新的提交说明覆盖刚才失误的提交。

如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 --amend 提交：

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

上面的三条命令最终只是产生一个提交，第二个提交命令修正了第一个的提交内容。

**取消已经暂存的文件**

接下来的两个小节将演示如何取消暂存区域中的文件，以及如何取消工作目录中已修改的文件。不用担心，查看文件状态的时候就提示了该如何撤消，所以不需要死记硬背。来看下面的例子，**有两个修改过的文件，我们想要分开提交，但不小心用git add . 全加到了暂存区域**。该如何撤消暂存其中的一个文件呢？其实，git status 的命令输出已经告诉了我们该怎么做：

$ git add .

$ git status

# On branch master

# Changes to be committed:

# (use "**git reset HEAD <file>**..." to unstage)

#

# modified: README.txt

# modified: benchmarks.rb

#

就在 “Changes to be committed” 下面，括号中有提示，**可以使用git reset HEAD <file>... 的方式取消暂存**。好吧，我们来试试取消暂存 benchmarks.rb 文件：

$ git reset HEAD benchmarks.rb

benchmarks.rb: locally modified

$ git status

# On branch master

# Changes to be committed:

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

#

# modified: README.txt

#

# 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: benchmarks.rb

#

这条命令看起来有些古怪，先别管，能用就行。现在 benchmarks.rb 文件又回到了之前已修改未暂存的状态。

**取消对文件的修改，之前已经暂存**

如果觉得刚才对 benchmarks.rb 的修改完全没有必要，该如何取消修改，回到之前的状态（也就是修改之前的版本）呢？git status 同样提示了具体的撤消方法，接着上面的例子，现在未暂存区域看起来像这样：

# 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: benchmarks.rb

#

在第二个括号中，我们看到了抛弃文件修改的命令（至少在 Git 1.6.1 以及更高版本中会这样提示，如果你还在用老版本，我们强烈建议你升级，以获取最佳的用户体验），让我们试试看：

$ git checkout -- benchmarks.rb

$ git status

# On branch master

# Changes to be committed:

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

#

# modified: README.txt

#

可以看到，该文件已经恢复到修改前的版本。**你可能已经意识到了，这条命令有些危险，所有对文件的修改都没有了，因为我们刚刚把之前版本的文件复制过来重写了此文件。所以在用这条命令前，请务必确定真的不再需要保留刚才的修改。**如果只是想回退版本，同时保留刚才的修改以便将来继续工作，可以用下章介绍的 stashing 和分支来处理，应该会更好些。

记住，**任何已经提交到 Git 的都可以被恢复**。即便在已经删除的分支中的提交，或者用 --amend 重新改写的提交，都可以被恢复（关于数据恢复的内容见第九章）。所以，**你可能失去的数据，仅限于没有提交过的**，对 Git 来说它们就像从未存在过一样。]

6，  向服务器提交变化

git push #向服务器提交

7, 暂存改动

git stash可以把当前的改动（stage和unstage，但不包括untrack的文件）暂存。然后通过git stash list查看。并通过git stash apply重新取出来。但apply之前要保证worktree是干净的。

[注：git stash 可用来暂存当前正在进行的工作， 比如想pull 最新代码， 又不想加新commit， 或者另外一种情况，为了fix 一个紧急的bug,  先stash, 使返回到自己上一个commit, 改完bug之后再stash pop, 继续原来的工作。  
基础命令：  
**$git stash  
$do some work  
$git stash pop**

git stash save "work in progress for foo feature"

**当你多次使用’git stash’命令后**，你的栈里将充满了未提交的代码，这时候你会对将哪个版本应用回来有些困惑，

’**git stash list**’ 命令可以将当前的Git栈信息打印出来，你只需要将找到对应的版本号，例如使用’**git stash apply stash@{1}’就可以将你指定版本号为stash@{1}的工作取出来**，当你将所有的栈都应用回来的时候，可以使用’**git stash clear**’来将栈清空。]

8，Git rm

从当前的工作目录中和索引中删除文件。   
可以递归删除，即如果后面跟的是一个目录做为参数，则会递归删除整个目录中的所有子目录和文件。例如：   
       git rm –r \* （进入某个目录中，执行此语句，会删除该目录下的所有文件和子目录）   
       git rm f1    (删除文件f1，包含本地目录和index中的此文件记录)   
       git rm --ached f1 (删除文件f1，不会删除本地目录文件，只删除index中的文件记录；将已经git add的文件remove到cache中,这样commit的时候不会提交这个文件, 适用于一下子添加了很多文件, 却又想排除其中个别几个文件的情况.)

6.差异比较：git diff

比较里程碑B和里程碑A：git diff B A

比较工作区和里程碑A：git diff A

比较暂存区和里程碑A：git diff –-cached A

比较工作区和暂存区：git diff

比较暂存区和HEAD：git diff –cached

比较工作区和HEAD：git diff HEAD

1. git的团队开发及操作

1，  获取项目

cd 本地工作目录（自定）

git clone 服务器帐户@IP：项目.根路经

这里具体操作为：

git clone [git@192.168.20.22](mailto:git@192.168.20.22)：android2.2.git

说明：这里假定服务器的用户名为git，服务器IP为192.168.20.22，项目名为android2.2，根路经为git的home（即根路径）

[注：$ **git clone** git://github.com/schacon/**grit**.git

这会在 **当前目录下 创建一个名为 “grit” 的目录**，其中**包含一个 .git 的目录**，用于保存下载下来的所有版本记录，然后从中取出**最新版本** 的文件拷贝。如果进入这个新建的 grit 目录，你会看到项目中的所有文件已经在里边了，准备好后续的开发和使用。如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

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

唯一的差别就是，现在新建的目录成了 mygrit，其他的都和上边的一样。

Git 支持许多数据传输协议。之前的例子使用的是**git:// 协议**，不过你也可以用 http(s):// 或者user@server:/path.git 表示的 SSH 传输协议。]

2，  团队开发的基本流程

git add 改动的文件

git commit #（提交至本地）

git pull #（将服务器项目与本地项目合并）

[注：从服务器的仓库中获取代码，和本地代码合并。（与服务器交互，从服务器上下载最新代码，等同于： Git fetch + Git merge）   
从其它的版本库（既可以是远程的也可以是本地的）将代码更新到本地，例如：“git pull origin master ”就是将origin这个版本库的代码更新到本地的master主分支。   
       git pull可以从任意一个git库获取某个分支的内容。用法如下：   
git pull [username@ipaddr](mailto:username@ipaddr):远端repository名远端分支名 本地分支名。这条命令将从远端git库的远端分支名获取到本地git库的一个本地分支中。其中，如果不写本地分支名，则默认pull到本地当前分支。   
需要注意的是，git pull也可以用来合并分支。 和git merge的作用相同。 因此，如果你的本地分支已经有内容，则git pull会合并这些文件，如果有冲突会报警。 ]

git push #（将本地项目上传至服务器）（在提交前要git pull --rebase一下，确保当前的本地的代码为最新。）[注：git push好像不会自动合并文件。因此，如果git push时，发生了冲突，就会被后push的文件内容强行覆盖，而且没有什么提示。 这在合作开发时是很危险的事情。]

git fetch 从服务器的仓库中下载代码。（与服务器交互，从服务器上下载最新代码） 相当于从远程获取最新版本到本地，不会自动merge，比Git pull更安全些。 使用此方法来获取服务器上的更新。   
例如：如果使用git checkout nov/eclair\_rocket （nov/eclair\_rocket为服务器上的分支名），则是获取上次使用git fetch命令时从服务器上下载的代码；如果先使用 git fetch ，再使用git checkout nov/eclair\_rocket，则是先从服务器上获取最新的更新信息，然后从服务器上下载最新的代码。

git blame 文件追溯，逐行显示文件，在每一行的行首显示此行最早是什么版本引入的，由谁引入

只想查看某几行，使用-L n,m参数，如：git blame –L 6，+5 README

1. git的分支管理

git分支操作在本地建立分支，然后与本地主枝合并，最终提交到服务器。有效的避免了因个人操作不当向服务器提交过多脏数据，避免频繁git clone服务器来更新本地库。

分支操作指令：

1，  建立分支

git branch AAA #建立分支AAA

2，分支切换

git checkout AAA #从当前分支切换到AAA分支

3，  将分支与主枝master合并

git checkout master #（首先切换回主枝）

git merge AAA #（将分支AAA与主枝合并）

4，  当前分支查看

git branch #默认有master（也称为主枝）

git branch –a 查看当前所有分支

5，  删除分支

git branch –d AAA #删除分支AAA

## [注：创建分支

**git branch**

　　没有参数，显示本地版本库中所有的本地分支名称。

　　当前检出分支的前面会有星号。

**git branch newname**

　　在当前检出分支上新建分支，名叫newname。

**git checkout newname**

　　检出分支，即切换到名叫newname的分支。

**git checkout –b newname master**

　　这个命令将上面两个命令合并：在master分支上创建分支newname分支并检出到该分支。

**合并分支间的修改 Merge**

　　合并操作将两条或多条分支合并到一起，实际上有好几种分支合并方法，下面介绍主要的三种：

**1.直接合并(straight merge)**：

　　把两条分支上的历史轨迹合并，交汇到一起。

　　比如要把dev分支上的所有东东合并到master分支：

　　首先先到master分支：**git checkout master**

　　然后把dev给合并过来：**git merge dev**

　　注意没参数的情况下merge是**fast-forward**的，即Git将master分支的指针直接移到dev的最前方。

　　换句话说，如果顺着一个分支走下去可以到达另一个分支的话，那么Git在合并两者时，只会简单移动指针，所以这种合并成为**快进式(Fast-forward)**。

**2.压合合并(squashed commits)：**

　　将一条分支上的若干个提交条目压合成一个提交条目，提交到另一条分支的末梢。

　　把dev分支上的所有提交压合成主分支上的一个提交，即压合提交：

**git checkout master**

**git merge --squash dev**

　　此时，dev上的所有提交已经合并到当前工作区并暂存，但还没有作为一个提交，可以像其他提交一样，把这个改动提交到版本库中：

**git commit –m “something from dev”**

**3.拣选合并(cherry-picking)：**

　　拣选另一条分支上的某个提交条目的改动带到当前分支上。

　　每一次提交都会产生一个全局唯一的提交名称，利用这个名称就可以进行拣选提交。

　　比如在dev上的某个提交叫：321d76f

　　把它合并到master中：

**git checkout master**

**git cherry-pick 321d76f**

　　要拣选多个提交，可以给git cherry-pick命令传递-n选项，比如：

**git cherry-pick –n 321d76f**

　　这样在拣选了这个改动之后，进行暂存而不立即提交，接着可以进行下一个拣选操作，一旦拣选完需要的各个提交，就可以一并提交。

**冲突处理**

　　当两条分支对同一个文件的同一个文本块进行了不同的修改，并试图合并时，Git不能自动合并的，称之为冲突(conflict)。解决冲突需要人工处理。

　　比如当前在master分支，想把dev分支merge过来，结果产生了一个冲突，打开文件内容可以看到这么一个冲突：

[复制代码](javascript:void(0);)

<<<<<<< HEAD

test in master

=======

test in dev

>>>>>>> dev

[复制代码](javascript:void(0);)

**<<<<<<<**标记冲突开始，后面跟的是当前分支中的内容。

**HEAD**指向当前分支末梢的提交。

**=======**之后，**>>>>>>>**之前是要merge过来的另一条分支上的代码。

**>>>>>>>**之后的dev是该分支的名字。

**对于简单的合并，手工编辑，然后去掉这些标记，最后像往常的提交一样先add再commit即可。**

**删除分支**

　　有些分支没有必要长期保存，比如分支中的代码已经打了标签并已发布，或者实验分支已经成功完成工作或中途废弃等等。

**注意**：打了标签的分支，Git在删除该分支时，从版本树起始到此标签间的全部历史轨迹均会保留，此时删除分支操作只是删除分支本身的名称，因此可以说该分支没有必要长期保存。

　　而在其他版本控制工具中，删除分支通常意味着删除分支上的所有历史轨迹，所以不能因为打了标签就认为其没有必要保存。

　　删除一个分支dev2：

**git branch –d dev2**

　　注意不能删除当前所在分支，需要转到别的分支上。

　　如果要删除的分支已经成功合并到当前分支，删除分支的操作会直接成功。

　　如果要删除的分支没有合并到当前所在分支，则会出现提示，如果确定无须合并而要直接删除，则执行命令：

**git branch –D dev2**

　　进行强删。

**分支重命名**

　　重命名分支：

**git branch –m oldname newname**

**-m**不会覆盖已有分支名称，即如果名为newname的分支已经存在，则会提示已经存在了。

　　如果改成**-M**就可以覆盖已有分支名称了，即会强制覆盖名为newname的分支，这种操作要谨慎。]

1. [git 忽略机制](http://blog.csdn.net/wh_19910525/article/details/7554555)

**在生成文档内容快照时，工作树中有一些文档是你不希望接受Git 管理的**，譬如**程序编译时生成的 中间文件**，对于这样的文件如何避免为之生成快照？

**Git 提供了 文档忽略机制，可以将工作树中你不希望接受Git 管理的文档信息写到 同一目录 下的.gitignore文件中**。

1.在**项目根目录下建立 .gitignore 文件**

2.   .gitignore文件过滤有两种模式，**开放模式 和 保守模式**

2.1开放模式负责设置**过滤**哪些文件和文件夹

eg：

过滤**文件夹**设置：

**mtk/       表示过滤这个文件夹**

过滤文件设置

指定过滤某种类型的文件：  
\*.zip  
\*.rar  
\*.via  
\*.tmp  
\*.err

指定过滤某个文件：  
/mtk/do.c

/mtk/if.h

2.2 b保守模式负责设置哪些文件**不被过滤**，也就是哪些文件要被跟踪。

跟踪某个文件夹

!/plutommi/mmi

跟踪某类文件

!\*.c

!\*.h

跟踪某个指定文件

!/plutommi/mmi/mmi\_features.h

3.配置.gitignore 的简易原则

采用共享模式与保守模式结合配置的办法。eg：一个文件夹下有很多文件夹和文件，而我只想跟踪其中的一个文件，这样设置就可以满足这种情况，先用共 享模式把整个目录 都设置为不跟踪，然后再用保守模式把这个文件夹中想要跟踪的文件设置为被跟踪，配置很简单，就可以跟踪想要跟踪的文件。

**需要注意的是：.gitignore 中的规则只对 未提交 且 未缓存 的文件有效。因此，如果在"未缓存的改动"中发现有需要忽略的文件， 马上修改 .gitignore，再重新扫描就会发现该文件被忽略了。**