## Git命令

|  |  |  |
| --- | --- | --- |
| **类别** | **命令** | **说明** |
| 配置一个用户 | git config --global user.name “username” | 配置用户名为username |
| git config --global user.emal xxx@yy.com | 配置邮箱为xxx@yy.com |
| git config --global -l | 查看用户信息 |
| 查看状态 | git status | 查看当前分支的状态 |
| 追踪文件 | git add readme.txt | 追踪文件readme.txt |
| git rm --cached readme.txt | 取消对reamde.txt的追踪 |
| 查看更改内容 | git diff | 查看所有更改内容 |
| git diff readme.txt | 仅查看readme.txt的更改 |
| 确认提交 | git commit -m “version 1.1” | 提交所有，并备注: version 1.1 |
| git commit readme.txt -m “readme v1.2” | 仅提交readme.txt，并备注: readme v1.2 |
| 回滚 | git log | 查看日志，以得到commit ID |
| git reset --soft 139dcfaa558e3276b30b6b2e5cbbb9c00bbdca96 | 软回滚到ID为139…ca96的提交状态 |
| git reset --hard 139dcfaa558e3276b30b6b2e5cbbb9c00bbdca96 | 硬回滚，删除目标版本之后的所有版本 |
| 分支 | git branch | 查看分支信息 |
| git branch readme | 新建一个叫readme的分支 |
| git branch -d readme | 删除readme分支 |
| git checkout readme | 切换到readme分支 |
| git merge readme | 将readme的内容合并到当前分支 |
| 远程仓库 | git clone <http://192.168.100.197/test.git> | 从服务器192.168.100.197上下载test项目 |
| git remote -v | 查看远程仓库 |
| git fetch [remote-name] (如git fetch origin) | 获取origin仓库的内容，不自动合并 |
| git pull [remote-name] (如git pull tyl) | 获取tyl仓库的内容并自动合并 |
| git push [remote-name] [branch-name] (如git push tyl master) | 推送分支到上游 |
| git remote add tyl http://192.168.100.197/test | 向test项目添加tyl仓库 |

## 附录1 分支的新建与合并

<https://git-scm.com/book/zh/v2>

让我们来看一个简单的分支新建与分支合并的例子，实际工作中你可能会用到类似的工作流。 你将经历如下步骤：

1. 开发某个网站。
2. 为实现某个新的需求，创建一个分支。
3. 在这个分支上开展工作。

正在此时，你突然接到一个电话说有个很严重的问题需要紧急修补。 你将按照如下方式来处理：

1. 切换到你的线上分支（production branch）。
2. 为这个紧急任务新建一个分支，并在其中修复它。
3. 在测试通过之后，切换回线上分支，然后合并这个修补分支，最后将改动推送到线上分支。
4. 切换回你最初工作的分支上，继续工作。

**新建分支**

首先，我们假设你正在你的项目上工作，并且已经有一些提交。



Figure 18. 一个简单提交历史

现在，你已经决定要解决你的公司使用的问题追踪系统中的 #53 问题。 想要新建一个分支并同时切换到那个分支上，你可以运行一个带有 -b 参数的 git checkout 命令：

$ git checkout -b iss53

Switched to a new branch "iss53"

它是下面两条命令的简写：

$ git branch iss53

$ git checkout iss53



Figure 19. 创建一个新分支指针

你继续在 #53 问题上工作，并且做了一些提交。 在此过程中，iss53 分支在不断的向前推进，因为你已经检出到该分支（也就是说，你的 HEAD 指针指向了 iss53 分支）

$ vim index.html

$ git commit -a -m 'added a new footer [issue 53]'



Figure 20. iss53 分支随着工作的进展向前推进

现在你接到那个电话，有个紧急问题等待你来解决。 有了 Git 的帮助，你不必把这个紧急问题和 iss53 的修改混在一起，你也不需要花大力气来还原关于 53# 问题的修改，然后再添加关于这个紧急问题的修改，最后将这个修改提交到线上分支。 你所要做的仅仅是切换回 master 分支。

但是，在你这么做之前，要留意你的工作目录和暂存区里那些还没有被提交的修改，它可能会和你即将检出的分支产生冲突从而阻止 Git 切换到该分支。 最好的方法是，在你切换分支之前，保持好一个干净的状态。 有一些方法可以绕过这个问题（即，保存进度（stashing） 和 修补提交（commit amending）），我们会在 [储藏与清理](https://git-scm.com/book/zh/v2/ch00/_git_stashing) 中看到关于这两个命令的介绍。 现在，我们假设你已经把你的修改全部提交了，这时你可以切换回 master 分支了：

$ git checkout master

Switched to branch 'master'

这个时候，你的工作目录和你在开始 #53 问题之前一模一样，现在你可以专心修复紧急问题了。 请牢记：当你切换分支的时候，Git 会重置你的工作目录，使其看起来像回到了你在那个分支上最后一次提交的样子。 Git 会自动添加、删除、修改文件以确保此时你的工作目录和这个分支最后一次提交时的样子一模一样。

接下来，你要修复这个紧急问题。 让我们建立一个针对该紧急问题的分支（hotfix branch），在该分支上工作直到问题解决：

$ git checkout -b hotfix

Switched to a new branch 'hotfix'

$ vim index.html

$ git commit -a -m 'fixed the broken email address'

[hotfix 1fb7853] fixed the broken email address

1 file changed, 2 insertions(+)



Figure 21. 基于 master 分支的紧急问题分支 hotfix branch

你可以运行你的测试，确保你的修改是正确的，然后将其合并回你的 master 分支来部署到线上。 你可以使用 git merge 命令来达到上述目的：

$ git checkout master

$ git merge hotfix

Updating f42c576..3a0874c

Fast-forward

index.html | 2 ++

1 file changed, 2 insertions(+)

在合并的时候，你应该注意到了"快进（fast-forward）"这个词。 由于当前 master 分支所指向的提交是你当前提交（有关 hotfix 的提交）的直接上游，所以 Git 只是简单的将指针向前移动。 换句话说，当你试图合并两个分支时，如果顺着一个分支走下去能够到达另一个分支，那么 Git 在合并两者的时候，只会简单的将指针向前推进（指针右移），因为这种情况下的合并操作没有需要解决的分歧——这就叫做 “快进（fast-forward）”。

现在，最新的修改已经在 master 分支所指向的提交快照中，你可以着手发布该修复了。



Figure 22. master 被快进到 hotfix

关于这个紧急问题的解决方案发布之后，你准备回到被打断之前时的工作中。 然而，你应该先删除 hotfix分支，因为你已经不再需要它了 —— master 分支已经指向了同一个位置。 你可以使用带 -d 选项的 git branch 命令来删除分支：

$ git branch -d hotfix

Deleted branch hotfix (3a0874c).

现在你可以切换回你正在工作的分支继续你的工作，也就是针对 #53 问题的那个分支（iss53 分支）。

$ git checkout iss53

Switched to branch "iss53"

$ vim index.html

$ git commit -a -m 'finished the new footer [issue 53]'

[iss53 ad82d7a] finished the new footer [issue 53]

1 file changed, 1 insertion(+)



Figure 23. 继续在 iss53 分支上的工作

你在 hotfix 分支上所做的工作并没有包含到 iss53 分支中。 如果你需要拉取 hotfix 所做的修改，你可以使用 git merge master 命令将 master 分支合并入 iss53 分支，或者你也可以等到 iss53 分支完成其使命，再将其合并回 master 分支。

**分支的合并**

假设你已经修正了 #53 问题，并且打算将你的工作合并入 master 分支。 为此，你需要合并 iss53 分支到 master 分支，这和之前你合并 hotfix 分支所做的工作差不多。 你只需要检出到你想合并入的分支，然后运行 git merge 命令：

$ git checkout master

Switched to branch 'master'

$ git merge iss53

Merge made by the 'recursive' strategy.

index.html | 1 +

1 file changed, 1 insertion(+)

这和你之前合并 hotfix 分支的时候看起来有一点不一样。 在这种情况下，你的开发历史从一个更早的地方开始分叉开来（diverged）。 因为，master 分支所在提交并不是 iss53 分支所在提交的直接祖先，Git 不得不做一些额外的工作。 出现这种情况的时候，Git 会使用两个分支的末端所指的快照（C4 和 C5）以及这两个分支的工作祖先（C2），做一个简单的三方合并。



Figure 24. 一次典型合并中所用到的三个快照

和之前将分支指针向前推进所不同的是，Git 将此次三方合并的结果做了一个新的快照并且自动创建一个新的提交指向它。 这个被称作一次合并提交，它的特别之处在于他有不止一个父提交。



Figure 25. 一个合并提交

需要指出的是，Git 会自行决定选取哪一个提交作为最优的共同祖先，并以此作为合并的基础；这和更加古老的 CVS 系统或者 Subversion （1.5 版本之前）不同，在这些古老的版本管理系统中，用户需要自己选择最佳的合并基础。 Git 的这个优势使其在合并操作上比其他系统要简单很多。

既然你的修改已经合并进来了，你已经不再需要 iss53 分支了。 现在你可以在任务追踪系统中关闭此项任务，并删除这个分支。

$ git branch -d iss53

**遇到冲突时的分支合并**

有时候合并操作不会如此顺利。 如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git 就没法干净的合并它们。 如果你对 #53 问题的修改和有关 hotfix 的修改都涉及到同一个文件的同一处，在合并它们的时候就会产生合并冲突：

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

此时 Git 做了合并，但是没有自动地创建一个新的合并提交。 Git 会暂停下来，等待你去解决合并产生的冲突。 你可以在合并冲突后的任意时刻使用 git status 命令来查看那些因包含合并冲突而处于未合并（unmerged）状态的文件：

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: index.html

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

任何因包含合并冲突而有待解决的文件，都会以未合并状态标识出来。 Git 会在有冲突的文件中加入标准的冲突解决标记，这样你可以打开这些包含冲突的文件然后手动解决冲突。 出现冲突的文件会包含一些特殊区段，看起来像下面这个样子：

<<<<<<< HEAD:index.html

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>> iss53:index.html

这表示 HEAD 所指示的版本（也就是你的 master 分支所在的位置，因为你在运行 merge 命令的时候已经检出到了这个分支）在这个区段的上半部分（======= 的上半部分），而 iss53 分支所指示的版本在 ======= 的下半部分。 为了解决冲突，你必须选择使用由 ======= 分割的两部分中的一个，或者你也可以自行合并这些内容。 例如，你可以通过把这段内容换成下面的样子来解决冲突：

<div id="footer">

please contact us at email.support@github.com

</div>

上述的冲突解决方案仅保留了其中一个分支的修改，并且 <<<<<<< , ======= , 和 >>>>>>> 这些行被完全删除了。 在你解决了所有文件里的冲突之后，对每个文件使用 git add 命令来将其标记为冲突已解决。 一旦暂存这些原本有冲突的文件，Git 就会将它们标记为冲突已解决。

如果你想使用图形化工具来解决冲突，你可以运行 git mergetool，该命令会为你启动一个合适的可视化合并工具，并带领你一步一步解决这些冲突：

$ git mergetool

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge

Merging:

index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

如果你想使用除默认工具（在这里 Git 使用 opendiff 做为默认的合并工具，因为作者在 Mac 上运行该程序）外的其他合并工具，你可以在 “下列工具中（one of the following tools）” 这句后面看到所有支持的合并工具。 然后输入你喜欢的工具名字就可以了。

|  |  |
| --- | --- |
| Note | 如果你需要更加高级的工具来解决复杂的合并冲突，我们会在 [高级合并](https://git-scm.com/book/zh/v2/ch00/_advanced_merging) 介绍更多关于分支合并的内容。 |

等你退出合并工具之后，Git 会询问刚才的合并是否成功。 如果你回答是，Git 会暂存那些文件以表明冲突已解决： 你可以再次运行 git status 来确认所有的合并冲突都已被解决：

$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: index.html

如果你对结果感到满意，并且确定之前有冲突的的文件都已经暂存了，这时你可以输入 git commit 来完成合并提交。 默认情况下提交信息看起来像下面这个样子：

Merge branch 'iss53'

Conflicts:

index.html

#

# It looks like you may be committing a merge.

# If this is not correct, please remove the file

# .git/MERGE\_HEAD

# and try again.

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: index.html

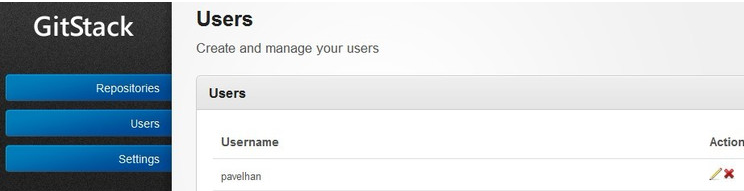
#

如果你觉得上述的信息不够充分，不能完全体现分支合并的过程，你可以修改上述信息，添加一些细节给未来检视这个合并的读者一些帮助，告诉他们你是如何解决合并冲突的，以及理由是什么。

## 附录2 windows平台下git服务器搭建

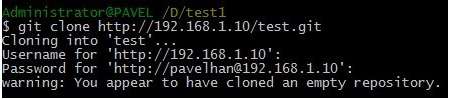
目前在windows平台上的git服务器大多数采用CopSSH+MsysGit的方式来实现，当然这种方式也是最原汁原味的在windows平台上搭建git服务器的方式，提供了最高程度的安全性和灵活性。但是缺点在于搭建仍然比较麻烦，要是能有一个Git版本的类似VisualSVN的服务器搭建包相信会更有利于git在windows平台上的使用。  
我在这里推荐一个目前来说部署和搭建最方便的windows版本Git服务器---gitstack。gitstack内部采用Apache HTTP server作为服务器，使用http协议对git进行封装，整合的已经比较完备，安装后简单配置一下即可使用。但目前所实现的功能还非常简单，期待能在后续开发中一步步完善起来。  
  
**1.下载gitstack**  
下载地址：[http://gitstack.com/download/](http://gitstack.com/download/" \t "_blank)  
同时在[http://gitstack.com/getting-started/](http://gitstack.com/getting-started/" \t "_blank)也提供了一个简单的安装和使用说明。  
软件包超过了80M，这是因为在里面同时整合了apache http server和msysgit。

**2.安装**  
与安装其他的windows软件别无二致。  
注意其安装路径：  
由于目前的gitstack版本（V1.2）的git仓库只能保存在git安装目录的repositories目录下（如果GitStack安装在C:\GitStack下，所有的git仓库都将位于C:\GitStack\repositories目录下），从代码库和数据的安全性考虑，最好把GitStack安装在非系统盘或者使用云同步软件随时同步在云端。  
  
**3.配置**  
安装完成后在浏览器中访问http://localhost/gitstack即可访问gitstack的配置界面，第一次登陆时以admin:admin进入，进去以后可以在Setting界面修改admin的登录密码。  
然后在User界面创建开发团队成员的账户：

  
现在就可以去Respositorys界面创建工程了，创建完成后还需要在Action中的Users域中选取可以访问该仓库的用户名：



 此时就可以在\GitStack\repositories目录下看到一个test.git目录，里面保存了test仓库的所有元数据。  
  
**4.访问git仓库**  
git仓库创建完成后就可以使用git客户端（如git bash）进行访问了：



  可以看到此时的test仓库中什么都没有，在其中创建一个文件后提交并push到服务器中。（注意这里服务器的ip：如果是局域网，用百度看的ip是不行的，正确的ip是cmd中ipconfig显示的ip。）



在git服务器的test仓库中立即可以看到新的提交：



**总结：**  
从目前的简单使用来看，gitstack虽然实现的功能还比较简单，但是作为一个有效版本管理的工具还是足够了，部署和使用也很方便，没有繁琐的配置和安装流程，非常适合于不需要繁琐的权限访问模型的中小型开发团队。