Git各个区



* Workspace：工作区
* Index / Stage：暂存区
* Repository：仓库区（或本地仓库）
* Remote：远程仓库

**一、新建代码库**

# 在当前目录新建一个Git代码库

$ git init

# 新建一个目录，将其初始化为Git代码库

$ git init [project-name]

# 下载一个项目和它的整个代码历史

**$ git clone [url]**

## 二、配置

Git的设置文件为.gitconfig，它可以在用户主目录下（全局配置），也可以在项目目录下（项目配置）。

# 显示当前的Git配置

$ git config --list

# 编辑Git配置文件

$ git config -e [--global]

# 设置提交代码时的用户信息

$ git config [--global] user.name "[name]"

$ git config [--global] user.email "[email address]"

## 三、增加/删除文件

# 添加指定文件到暂存区

**$ git add [file1] [file2] ...**

# 添加指定目录到暂存区，包括子目录

$ git add [dir]

# 添加当前目录的所有文件到暂存区

**$ git add .**

# 添加每个变化前，都会要求确认

# 对于同一个文件的多处变化，可以实现分次提交

$ git add -p

# 删除工作区文件，并且将这次删除放入暂存区

$ git rm [file1] [file2] ...

# 停止追踪指定文件，但该文件会保留在工作区

$ git rm --cached [file]

# 改名文件，并且将这个改名放入暂存区

$ git mv [file-original] [file-renamed]

## 四、代码提交

# 提交暂存区到仓库区

**$ git commit -m [message]**

# 提交暂存区的指定文件到仓库区

**$ git commit [file1] [file2] ... -m [message]**

# 提交工作区自上次commit之后的变化，直接到仓库区

$ git commit -a

# 提交时显示所有diff信息

$ git commit -v

# 使用一次新的commit，替代上一次提交

# 如果代码没有任何新变化，则用来改写上一次commit的提交信息

$ git commit --amend -m [message]

# 重做上一次commit，并包括指定文件的新变化

$ git commit --amend [file1] [file2] ...

## 五、分支

# 列出所有本地分支

**$ git branch**

# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

**$ git branch -a**

# 新建一个分支，但依然停留在当前分支

**$ git branch [branch-name]**

# 新建一个分支，并切换到该分支

**$ git checkout -b [branch]**

# 新建一个分支，指向指定commit

$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系

$ git branch --track [branch] [remote-branch]

# 切换到指定分支，并更新工作区

**$ git checkout [branch-name]**

# 切换到上一个分支

**$ git checkout -**

# 建立追踪关系，在现有分支与指定的远程分支之间

$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支

$ git merge [branch]

# 选择一个commit，合并进当前分支

$ git cherry-pick [commit]

# 删除分支

**$ git branch -d [branch-name]**

# 删除远程分支

$ git push origin --delete [branch-name]

$ git branch -dr [remote/branch]

## 六、标签

# 列出所有tag

$ git tag

# 新建一个tag在当前commit

$ git tag [tag]

# 新建一个tag在指定commit

$ git tag [tag] [commit]

# 删除本地tag

$ git tag -d [tag]

# 删除远程tag

$ git push origin :refs/tags/[tagName]

# 查看tag信息

$ git show [tag]

# 提交指定tag

$ git push [remote] [tag]

# 提交所有tag

$ git push [remote] --tags

# 新建一个分支，指向某个tag

$ git checkout -b [branch] [tag]

## 七、查看信息

# 显示有变更的**文件**

**$ git status**

# 显示当前分支的版本历史

**$ git log**

# 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

# 搜索提交历史，根据关键词

$ git log -S [keyword]

# 显示某个commit之后的所有变动，每个commit占据一行

$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名

$ git log --follow [file]

$ git whatchanged [file]

# 显示指定文件相关的每一次diff

$ git log -p [file]

# 显示过去5次提交

$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序

$ git shortlog -sn

# 显示指定文件是什么人在什么时间修改过

$ git blame [file]

# 显示暂存区和工作区的**代码**差异

**$ git diff**

# 显示暂存区和上一个commit的差异

$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 显示两次提交之间的差异

$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码

$ git diff --shortstat "@{0 day ago}"

# 显示某次提交的元数据和内容变化

**$ git show [commit]**

# 显示某次提交发生变化的文件

$ git show --name-only [commit]

# 显示某次提交时，某个文件的内容

$ git show [commit]:[filename]

# 显示当前分支的最近几次提交

$ git reflog  
  
# 从本地master拉取代码更新当前分支：branch 一般为master  
**$ git rebase [branch]**

## 八、远程同步

**$ git remote update --更新远程仓储**

# 下载远程仓库的所有变动

$ git fetch [remote]

# 显示所有远程仓库

$ git remote -v

# 显示某个远程仓库的信息

$ git remote show [remote]

# 增加一个新的远程仓库，并命名

$ git remote add [shortname] [url]

# 取回远程仓库的变化，并与本地分支合并

**$ git pull [remote] [branch]**

# 上传本地指定分支到远程仓库

**$ git push [remote] [branch]**

# 强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] --force

# 推送所有分支到远程仓库

$ git push [remote] --all

## 九、撤销

# 恢复暂存区的指定文件到工作区

$ git checkout [file]

# 恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file]

# 恢复暂存区的所有文件到工作区

$ git checkout .

# 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

$ git reset [file]

# 重置暂存区与工作区，与上一次commit保持一致

**$ git reset --hard**

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit]

# 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit]

# 重置当前HEAD为指定commit，但保持暂存区和工作区不变

$ git reset --keep [commit]

# 新建一个commit，用来撤销指定commit

# 后者的所有变化都将被前者抵消，并且应用到当前分支

$ git revert [commit]

# 暂时将未提交的变化移除，稍后再移入

**$ git stash**

**$ git stash pop**

## 十、其他

# 生成一个可供发布的压缩包

$ git archive

# git cherry-pick 小结

**git cherry-pick**可以选择某一个分支中的一个或几个commit(s)来进行操作。例如，假设我们有个稳定版本的分支，叫v2.0，另外还有个开发版本的分支v3.0，我们不能直接把两个分支合并，这样会导致稳定版本混乱，但是又想增加一个v3.0中的功能到v2.0中，这里就可以使用cherry-pick了。

就是对已经存在的commit 进行 再次提交；

git cherry-pick <commit id>

**注意：当执行完 cherry-pick 以后，将会 生成一个新的提交；这个新的提交的哈希值和原来的不同，但标识名 一样；**

**例如：**  
$ git checkout old\_cc  
$ git cherry-pick 38361a68    # 这个 38361a68 号码，位于：

$ git log

commit 38361a68138140827b31b72f8bbfd88b3705d77a  
Author: Siwei Shen <siwei.shen@focusbeijing.com>  
Date:   Sat Dec 10 00:09:44 2011 +0800

**1. 如果顺利，就会正常提交**。结果：

Finished one cherry-pick.  
# On branch old\_cc  
# Your branch is ahead of 'origin/old\_cc' by 3 commits.

**2. 如果在cherry-pick 的过程中出现了冲突**

Automatic cherry-pick failed.  After resolving the conflicts,  
mark the corrected paths with 'git add <paths>' or 'git rm <paths>'  
and commit the result with:   
  
        git commit -c 15a2b6c61927e5aed6718de89ad9dafba939a90b

就跟普通的冲突一样，手工解决：  
2.1 $ git status    # 看哪些文件出现冲突

both modified:      app/models/user.rb

2.2 $ vim app/models/user.rb  # 手动解决它。   
2.3 $ git add app/models/user.rb

2.4 git commit -c <新的commit号码>

我们实际开发中肯定遇到过这样的场景，我在develop分支开发了一段时间，我们develop开发中假如提交了三次，我们分别实现了不同的模块功能1，2，3，这时候正赶上老大“心血来潮”，需要把其中模块2提前上线，这时候我们怎么办？？难道在建分支，然后回退，合并吗，不是说不能实现，但有没有更好的解决方案呢？其实git早已考虑到这一点。这就是cherry-pick。

cherry-pick就是从不同的分支中捡出单独的commit，并把它和你当前的分支合并。但有一点需要注意，如果你pick第二个commit，第一个commit是不会被pick的。

需要说明的是：cherry-pick不但可以用在不同分支之间，也可以用在同一个分支上。简单说一个场景吧：比如今天你新增了一个功能，commit1，第二天这个功能又不需要了，你又不情愿的注释掉,甚至删除了代码，commit2. 然后过了几天，产品过来了，来来来，老弟，还是需要加这个功能，这时候你拿出来菜刀，但产品经理这次却不吃那一套，好吧，那就找回来吧。这时候就可以用cherry-pick，重新找回来了。

另外，也可以cherry-pick多个，最好是最早提交的放在最上边，不同commitID之间，用空格分开。