本页不打印

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 开发/优化者 | 时间 | 审核人 | 开发类型（新开发/优化） | 更新说明 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程编码 | 适用产品 | 产品版本 | 课程版本ISSUE |
|  |  |  |  |

修订记录

Git常用操作指南



华为技术有限公司

目录

[1 Git参考资料 2](#_Toc177053482)

[1.1 Git各工作区之间的交互关系 2](#_Toc177053483)

[1.1.2 Git参考资料 2](#_Toc177053484)

[2 Git配置管理 3](#_Toc177053485)

[2.1 配置查询命令 3](#_Toc177053486)

[2.2 第一次使用Git，配置用户信息命令 3](#_Toc177053487)

[2.3 其它配置命令 3](#_Toc177053488)

[3 暂存区上的操作命令 4](#_Toc177053489)

[3.1 提交文件到版本库命令 4](#_Toc177053490)

[3.2 查看信息命令 4](#_Toc177053491)

[4 本地库和远程库 5](#_Toc177053492)

[4.1 新建仓库 5](#_Toc177053493)

[4.2 从远程库克隆 5](#_Toc177053494)

[5 常用查看指令 7](#_Toc177053495)

[5.1 常用查看命令 7](#_Toc177053496)

[6 常用修改指令 8](#_Toc177053497)

[6.1 常用修改命令 8](#_Toc177053498)

[7 撤销修改和版本回退 9](#_Toc177053499)

[7.1 撤销修改和版本回退命令 9](#_Toc177053500)

[8 分支管理 10](#_Toc177053501)

[8.1 创建和合并分支命令 10](#_Toc177053502)

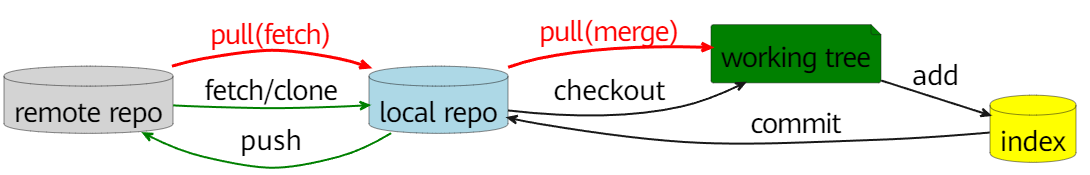
[8.2 解决冲突 10](#_Toc177053503)

[8.3 Bug分支 11](#_Toc177053504)

[8.4 远程分支 11](#_Toc177053505)

# Git参考资料

## Git各工作区之间的交互关系

Git各工作区之间的交互关系

Git各工作区之间的交互关系图

### Git参考资料

<https://progit.bootcss.com/git>

<https://learngitbranching.js.org/>

交互式的Git分支教程，直接在浏览器中练习Git的常用命令。

# Git配置管理

## 配置查询命令

git config --list 列出当前配置

git config --local --list 列出repository配置

git config --global --list 列出全局配置

git config --system --list 列出系统配置

## 第一次使用Git，配置用户信息命令

git config --global user.name "your name" 配置用户名

git config --global user.email "youremail@xxx.com" 配置用户邮箱

## 其它配置命令

git config --global merge.tool vimdiff 配置解决冲突时使用哪种差异分析工具，Ex:vimdiff

git config --global color.ui auto 配置git命令输出为彩色

git config --global core.editor vim 配置git使用的文本编辑器

# 暂存区上的操作命令

## 提交文件到版本库命令

git commit -m "commit info" 将暂存区中的文件提交到本地仓库中

git commit -a -m "commit info" 配将所有已经使用git管理过的文件暂存后一并提交，跳过add到暂存区的过程

git commit --amend 提交文件时，发现注释写错了，可以修改上一次提交

git reset --soft HEAD^ 撤销上一次提交

## 查看信息命令

git diff --cached 比较暂存区与上一版本的差异

git diff <file-name> --cached 指定文件在暂存区和本地仓库的不同

git log -p -2 查看提交历史，-p展开每次提交的内容差异，-2显示最近的2次更新

git tag 列出现在所有的标签

git tag -l "v1.8\*" 使用特定的搜索模式列出符合条件的标签

# 本地库和远程库

## 新建仓库

* 建立远程库(为空，不要加README.md，不然后面会push不上去)
* 本地新建文件夹
* git init初始化仓库，可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的。
* 远程库的名字就是origin，这是Git默认的叫法
* git remote add origin git@xxx.com:xxx/xxx.git 这个命令是在本地的learngit仓库下执行的。这两个地方的*仓库名不需要相同*，因为会通过在本地的仓库目录下执行这条命令（命令中包含远程库的名字）已经将两者建立了联系
* git push -u origin master 把本地库的所有内容推送到远程库上。把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。由于远程库是空的，我们第一次推送master分支时，加上了-u参数(推送和关联)，Git不但会把本地的master分支内容推送到远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。
* git push origin master每次本地提交后,推送最新修改到远程库

## 从远程库克隆

假设testhub.com上面已经有一个远程库，但是本地没有，需要克隆到本地，远程库的名字叫gittest

* git clone git@test.com:xxx/gittest.git 克隆一个本地库,则在当前文件夹下会多一个gittest的文件夹。
* cd gittest进入克隆下来的本地库，默认的名字是和testhub上的一样的
* git push origin master 推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上

# 常用查看指令

## 常用查看命令

git status 查看仓库当前的状态

git diff 文件名查看对文件做什么修改

git diff 版本号1 版本号2 --stat查看两个版本的差异的文件列表，包括被修改行数和增删图。参数改为--name-status前面显示修改说明字母(A,M等)，无行数

git log显示从最近到最远的提交日志

git log --pretty=oneline 简化日志输出的显示信息，commit id很长,详细显示见[这里](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "_blank)

git reflog 记录你的每一次命令，最先显示的是这个命令执行之后的版本的版本号的前七位，这样就算你清屏了或者重启了，也能找到某个版本的版本号，就可以轻松回退到那个版本

git branch 查看当前所在的分支。git branch -a命令会列出所有分支，当前分支前面会标一个\*号

git log --graph --pretty=oneline --abbrev-commit用带参数的git log可以看到分支的合并情况。用git log --graph命令可以看到分支合并图

git remote 查看远程库的信息

git remote -v 显示更为详细的信息

# 常用修改指令

## 常用修改命令

git add readme.txt添加，但是不提交

git commit -m "提交描述"提交，**只有add后提交才有效**。*“改文件->add文件->再改->提交”*，则第二次修改无效,不会被提交，只会成功提交第一次的修改

# 撤销修改和版本回退

## 撤销修改和版本回退命令

git checkout -- 文件名把没暂存(即没add)的干掉，或者说，丢弃工作区，回到到暂存状态

git reset HEAD 文件名把暂存的状态取消，工作区内容不变，但状态变为“未暂存”。

简单来说，没有add过的修改，只需要git checkout -- 文件名即可撤销；add 过的修改，先git reset HEAD 文件名变成没add 过的修

git checkout -- 文件名撤销。

git reset --hard HEAD^ 会回退到上一个版本

git reset --hard 某版本号前几位通过命令行上的历史信息（假如你没清屏的话），找到某版本的版本号回到指定版本。不一定要全部的版本号，就像这个命令的例子，只要前面的约7、8位这样就可以。

# 分支管理

## 创建和合并分支命令

git checkout -b dev创建一个新的分支：dev，并且会切换到dev分支。所以这条命令有两个作用。git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：git branch dev和git checkout dev

git branch dev，新建分支是新建指针,指向当前commit

git checkout dev切换到dev分支

git checkout ${commit-id} -b dev 根据commit id创建分支dev

git checkout masterdev分支的工作完成，我们就可以切换回master分支(**此时在dev分支的修改在master上是看不到的**)

git merge dev 这是在master分支上执行的命令，作用是：把dev分支上的工作成果合并到master分支上

git branch -d dev 删除已合并的分支。删除分支就是删除指针

git branch -D devGit友情提醒，dev分支还没有被合并，如果删除，将丢失掉修改，如果要强行删除，需要使用git branch -D dev命令

git rebase master变基。在当前分支(非master)下执行该命令，则相当于把当前分支和mater分支合并，和merge操作类似，但提交历史不同，rebase操作的log更干净。具体可参考[Git 分支 - 变基](https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E5%8F%98%E5%9F%BA" \t "_blank)

git checkout -b ${newbranch} ${tag} How to create a new branch from a tag

## 解决冲突

假设在master分支和feature1分支对同一文件做了修改

git merge feature1 在master分支上执行该命令，与feature1分支合并。这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，果然冲突了！Git告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。git status也可以告诉我们冲突的文件。合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息

git merge --no-ff -m "merge with with no-ff" dev准备合并dev分支，注意--no-ff参数表示禁用Fast forward，因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去

## Bug分支

Git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

* git stash保存工作现场
* git stash list 查看工作现场
* git stash apply恢复工作现场，但是恢复后，stash内容并不删除，有多个工作现场时可以git stash apply stash@{0}恢复特定的现场
* git stash drop删除stash的内容
* git stash pop恢复的同时也把stas内容删除了

## 远程分支

这部分只介绍常用的几个操作

* git fetch origin 这个命令查找 “origin” 是哪一个服务器，从中抓取本地没有的数据，并且更新本地数据库，移动 origin/master指针指向新的、更新后的位置
* git push (remote) (branch)推送本地的分支来更新远程仓库上的 同名分支。如前文提到的git push origin master就是将本地master分支推送到远程master分支；复杂一点的，git push origin serverfix:awesomebranch将本地的 serverfix分支推送到远程仓库上的awesomebranch分支
* git push origin --delete serverfix或者git push origin :remotebranch,删除远程的serverfix分支
* git pull在大多数情况下它的含义是一个git fetch紧接着一个git merge命令。具体可参考[Git远程操作详解](http://www.ruanyifeng.com/blog/2014/06/git_remote.html)和[Documentation git-pull](https://git-scm.com/docs/git-pull)