## [参考文献](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000)

## Git远程分支本地分支介绍

首先要明确一点，对git的操作是围绕3个大的步骤来展开的（其实几乎所有的SCM都是这样）

1.从git取数据（git clone）

2.改动代码

3.将改动传回git（git push）

这3个步骤又涉及到两个repository，一个是remote repository，再远程服务器上，一个是local repository，再自己工作区上。其中

1, 3两个步骤涉及到remote server/remote repository/remote branch，2涉及到local repository/local branch。git clone 会根据你指定的remote server/repository/branch，拷贝一个副本到你本地，再git push之前，你对所有文件的改动都是在你自己本地的local repository来做的，你的改动(local branch)和remote branch是独立（并行）的。Gitk显示的就是local repository。

在clone完成之后，Git 会自动为你将此远程仓库命名为origin（origin只相当于一个别名，运行git remote –v或者查看.git/config可以看到origin的含义）(guyue: 即origin是代表远程仓库的一个默认别名, 可以在项目工程目录中.git/config文件中查看或修改)，并下载其中所有的数据，建立一个指向它的master 分支的指针，我们用(远程仓库名)/(分支名) 这样的形式表示远程分支，所以origin/master指向的是一个remote branch（从那个branch我们clone数据到本地），但你无法在本地更改其数据。

同时，Git 会建立一个属于你自己的本地master 分支，它指向的是你刚刚从remote server传到你本地的副本。随着你不断的改动文件，git add, git commit，master的指向会自动移动，你也可以通过merge（fast forward）来移动master的指向。

$git branch -a (to show all the branches git knows about)

\* master

  remotes/origin/HEAD -> origin/master

  remotes/origin/master

$git branch -r (to show remote branches git knows about)

  origin/HEAD -> origin/master

  origin/master

可以发现，master就是local branch，origin/master是remote branch（master is a branch in the local repository. remotes/origin/master is a branch named master on the remote named origin）

$git diff origin/master master （show me the changes between the remote master branch and my master branch).

需要注意的是，remotes/origin/master和origin/master的指向是相同的

$git diff origin/master remotes/origin/master

git push origin master

origin指定了你要push到哪个remote

master其实是一个“refspec”，正常的“refspec”的形式为”+<src>:<dst>”，冒号前表示local branch的名字，冒号后表示remote repository下 branch的名字。注意，如果你省略了<dst>，git就认为你想push到remote repository下和local branch相同名字的branch。听起来有点拗口，再解释下，push是怎么个push法，就是把本地branch指向的commit push到remote repository下的branch，比如

**$git push origin master:master (在local repository中找到名字为master的branch，使用它去更新remote repository下名字为master的branch，如果remote repository下不存在名字是master的branch，那么新建一个)**,

$git push origin master （省略了<dst>，等价于“git push origin master:master”）

我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的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 cat-file -p :filename 查看文件在暂存区的内容

git branch --set-upstream master origin/next Git也允许手动建立追踪关系

git push <远程主机名> <本地分支名>:<远程分支名>

git pull <远程主机名> <远程分支名>:<本地分支名>

git push origin --delete master 删除origin主机的master分支

git fetch 更新所有分支,用于从另一个存储库下载对象和引用

git checkout -b newBrach origin/master 在origin/master的基础上，创建一个新分支:newBrach

git remote -v 列出详细信息，在每一个名字后面列出其远程url

## Git branch基本操作命令

一般用于分支的操作，比如创建分支，查看分支等等，

**1.1 git branch**

　　　　　　不带参数：列出本地已经存在的分支，并且在当前分支的前面用"\*"标记

**1.2 git branch -r**

　　　　　　查看远程版本库分支列表

**1.3 git branch -a**

　　　　　　查看所有分支列表，包括本地和远程

**1.4 git branch dev**

　　　　　　创建名为dev的分支，创建分支时需要是最新的环境，创建分支但依然停留在当前分支

**1.5 git branch -d dev**

　　　　　　删除dev分支，如果在分支中有一些未merge的提交，那么会删除分支失败，此时可以使用 git branch -D dev：强制删除dev分支，

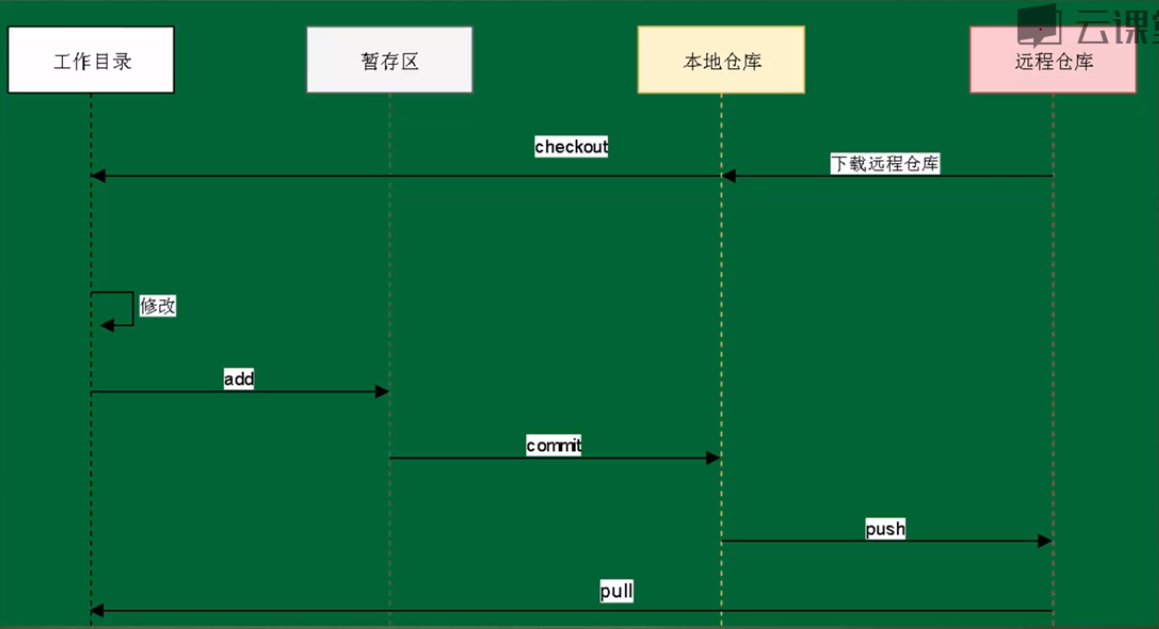
**1.6 git branch -vv**

　　　　　　可以查看本地分支对应的远程分支

**1.7 git branch -m oldName newName**

　　　　　　给分支重命名

## Git工作流程



## Git文件状态

touch test>>git add .>>git commit –m “create file test”>>git push origin master 创建文件

echo “111” >> test 第一次修改文件

git add . 将111加入暂存区

echo “222” >> test 第二次修改文件

git status 文件会有两种状态，一种是111在暂存区，一种是222刚刚修改

git commit –m “change 111” 将111提交到本地，此时222还在修改状态，暂存区没有内容

git status 文件会有两种状态，一种是111在本地仓库，一种是222刚刚修改

**(git diff工作目录和暂存区的区别) (git diff HEAD工作目录和本地仓库的区别) (git diff -- cached 暂存区和本地仓库的区别)**

git add . 将222加入暂存区

git status 文件会有两种状态，一种是111在本地仓库，一种是222在暂存区

git echo “333” >> test 第三次修改文件

git status 文件会有三种状态，一种是111在本地仓库，一种是222在暂存区, 一种是333刚刚修改

## Git的rm --cached的使用

    •    新建文件1.txt, 未被跟踪(Untracked files)，提交到暂存区(Changed to be committed)，未提交到版本库。 使用git rm —cached 1.txt，是把文件恢复到未被跟踪的状态，即删除暂存区中的1.txt;

    •    文件1.txt，已经提交到版本库，工作区，暂存区都是干净的.   使用git rm —cached 1.txt,

    ◦    把工作区的文件1.txt置为了”未跟踪”状态，即Untracked files。

    ◦    暂存区生成一个deleted 1.txt的记录，如果提交了，就是把版本库中的1.txt删除。

    ◦    不影响工作区中的文件。

    •    文件1.txt, 已经提交到版本库，修改1.txt，并且提交到了暂存区。   使用git rm —cached 1.txt,

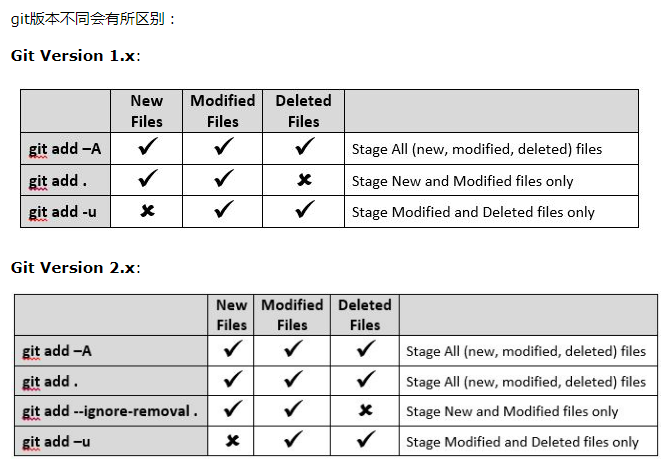
    ◦    把工作区的文件1.txt置为了”未跟踪”状态，即Untracked files。

    ◦    暂存区生成一个deleted 1.txt的记录，如果提交了，就是把版本库中的1.txt删除。

    ◦    不影响工作区中的文件。还是修改后的1.txt

    •    文件1.txt, 已经提交到版本库，修改1.txt，提交到暂存区，继续修改1.txt。   使用git rm —cached 1.txt   会报错，不能执行操作。

## Git的add用法



## Git 撤销修改

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file(git status 有提示)。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "_blank)一节，不过前提是没有推送到远程库。

git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就**回到和版本库一模一样的状态**；

（原本空白文件，写入11字符串，放到暂存区，撤销修改后还是空白文件）

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就**回到添加到暂存区后的状态**。

（原本空白文件，写入11字符串，放到暂存区，再写入22字符串，撤销修改后只剩下暂存区的11字符串）

git reset HEAD filename **放弃**暂存区某个文件的修改，是放弃，不是还原

**(在暂存区放弃只能用 git reset HEAD filename，git checkout -- readme.txt 用于工作区) 命令行有提示**

git reset --hard HEAD 回退到当前版本

## Git reset版本回退

git log 显示最近commit日志

git log --oneline 最近日志的简化展示，可以清楚地看到提交历史的时间线

git reset HEAD 放弃暂存区的修改

git re**set** --hard HEAD 回退到当前版本（是当前的某一个版本，不是和远程仓库一样的版本，适用于放弃暂存区和工作区的所有修改，如果修改内容已经commit到本地仓库，那么）

git re**set** --hard HEAD^ 回退到上一个版本

git re**set** --hard HEAD^^ 回退到上上一个版本

git re**set** --hard HEAD~100 回退到上100个版本

git re**set** --hard 版本号 回退到某个版本号

**这里的HEAD指的是本地仓库，远程仓库不变，回退是指回退到本地仓库的某一个版本a，回退后，head指向的是某一个版本a，这个某个版本a可能是和远程仓库一样的最新版本，也可能是其他版本。**

git reset --hard origin/R2 如果当前分支是R2并且对应的远程分支是origin/R2，那么使**本地仓库**和**当前版本**回退到和**远程分支**一样。

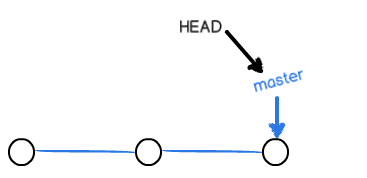
如果本地分支是R3，却使用了git reset --hard origin/R2，是把origin/R2的内容覆盖R3的工作目录内容，

可以使用git branch –vv查看本地分支对应的远程分支。

## Git创建与合并分支

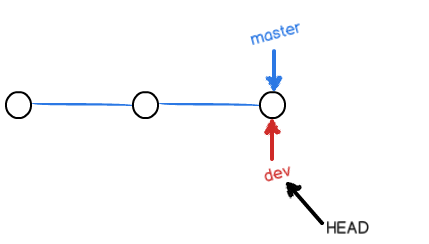
在[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)里，你已经知道，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



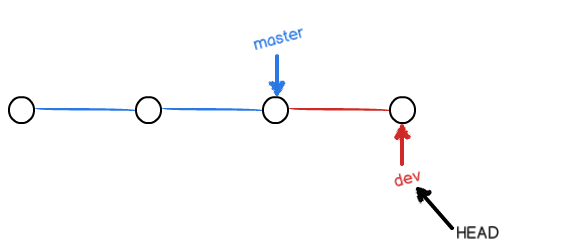
每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：

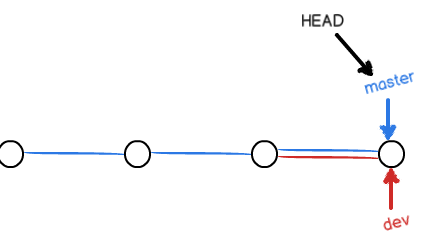


你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：

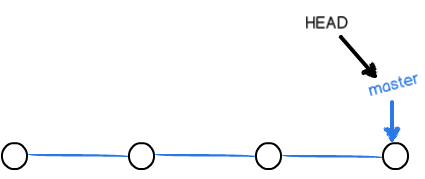


假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



## Git fetch

要更新所有分支，命令可以简写为：

$ git fetch

上面命令将某个远程主机的更新，全部取回本地。默认情况下，git fetch取回所有分支的更新。如果只想取回特定分支的更新，可以指定分支名,如下所示 -

$ git fetch <远程主机名> <分支名>

比如，取回origin主机的master分支。

$ git fetch origin master

所取回的更新，在本地主机上要用”远程主机名/分支名”的形式读取。比如origin主机的master分支，就可以用origin/master读取。

git branch命令的-r选项，可以用来查看远程分支，-a选项查看所有分支。

$ git branch -r

origin/master

$ git branch -a

\* master

remotes/origin/master

上面命令表示，本地主机的当前分支是master，远程分支是origin/master。

取回远程主机的更新以后，可以在它的基础上，使用git checkout命令创建一个新的分支。

$ git checkout -b newBrach origin/master

上面命令表示，在origin/master的基础上，创建一个新分支:*newBrach*。

## [Git pull](https://www.yiibai.com/git/git_pull.html)

$ git pull <远程主机名> <远程分支名>:<本地分支名>

比如，要取回origin主机的next分支，与本地的master分支合并，需要写成下面这样 -

$ git pull origin next:master

如果远程分支(next)要与当前分支合并，则冒号后面的部分可以省略。上面命令可以简写为：

$ git pull origin next

上面命令表示，取回origin/next分支，再与当前分支合并。实质上，这等同于先做git fetch，再执行git merge。

$ git fetch origin

$ git merge origin/next

在某些场合，Git会自动在本地分支与远程分支之间，建立一种追踪关系(tracking)。比如，在git clone的时候，所有本地分支默认与远程主机的同名分支，建立追踪关系，也就是说，本地的master分支自动”追踪”origin/master分支。

Git也允许手动建立追踪关系。

$ git branch --set-upstream master origin/next

上面命令指定master分支追踪origin/next分支。

如果当前分支与远程分支存在追踪关系，git pull就可以省略远程分支名。

$ git pull origin

上面命令表示，本地的当前分支自动与对应的origin主机”追踪分支”(remote-tracking branch)进行合并。

如果当前分支只有一个追踪分支，连远程主机名都可以省略。

$ git pull

上面命令表示，当前分支自动与唯一一个追踪分支进行合并。

如果合并需要采用rebase模式，可以使用–rebase选项。

$ git pull --rebase <远程主机名> <远程分支名>:<本地分支名>

**git fetch和git pull的区别**

1.git fetch：相当于是从远程获取最新版本到本地，不会自动合并。

$ git fetch origin master

$ git log -p master..origin/master

$ git merge origin/master

以上命令的含义：

* 首先从远程的origin的master主分支下载最新的版本到origin/master分支上
* 然后比较本地的master分支和origin/master分支的差别
* 最后进行合并

上述过程其实可以用以下更清晰的方式来进行：

$ git fetch origin master:tmp

$ git diff tmp

$ git merge tmp

2. git pull：相当于是从远程获取最新版本并merge到本地

git pull origin master

上述命令其实相当于git fetch 和 git merge  
在实际使用中，git fetch更安全一些，因为在merge前，我们可以查看更新情况，然后再决定是否合并。

## Git push

$ git push <远程主机名> <本地分支名>:<远程分支名>

$ git push origin master

上面命令表示，将本地的master分支推送到origin主机的master分支。如果master不存在，则会被新建。

如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支。

$ git push origin :master

# 等同于

$ git push origin --delete master

上面命令表示删除origin主机的master分支。如果当前分支与远程分支之间存在追踪关系，则本地分支和远程分支都可以省略。

$ git push origin

上面命令表示，将当前分支推送到origin主机的对应分支。如果当前分支只有一个追踪分支，那么主机名都可以省略。

$ git push

如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机，这样后面就可以不加任何参数使用git push。

$ git push -u origin master

上面命令将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了。

## Git stash

git stash/git stash save "message..."

备份当前的工作区的内容（备份全部内容，不是修改内容）

git stash list

显示Git栈内的所有备份，可根据这个列表来决定从备份版本恢复

git stash pop [stash@{index}]

不加stash版本号，是将当前工作区内容和stash区的第一个内容合并，不会移除stash列表的第一个缓存内容（百度说会移除）

加stash版本号，是将当前工作区内容和stash区的某个内容合并，不会移除stash列表的某个缓存内容（百度说会移除）

git stash clear

清空Git栈，即删除所有备份

git stash show [stash@{index}]

git stash drop [stash@{index}]

删除某个备份之后，stash备份会重新排序

## 用stash解决冲突

本地仓库和远程仓库文本test.txt内容：aa

修改远程仓库文本test.txt内容并提交保存：aa，bb

修改本地仓库文本test.txt内容还未add：aa，cc

因为在add之前会git pull origin master来确认没有冲突

git pull origin master，即git fetch origin master & git merge origin master操作的合并，但是git pull是原子操作

此时git pull origin master会有冲突产生

因为没有add，所以将工作区的所有内容stash备份git stash save "message..."，是所有内容，不是修改内容

此时git pull origin master，是将远程仓库代码拉到本地，并更新工作目录

git stash pop [stash@{index}] 将备份内容和工作区最新内容合并

然后打开文件，修改冲突