Git跟踪并管理的是修改，而非文件。

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的。

Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢。

## 项目创建/获取:

1、通过git init命令把当前目录变成Git可以管理的仓库；

2、要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修

3、要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

## 工作提交：

用命令git add告诉Git，把文件添加到仓库

用命令git commit告诉Git，把文件提交到仓库

git commit –a合并上述两个命令

## 查看区别:

git status命令可以让我们时刻掌握仓库当前的状态

**git diff** 显示已写入缓存与已修改但尚未写入缓存的改动的区别

**git diff --cached** 查看已缓存的改动

**git diff HEAD**查看已缓存的与未缓存的所有改动

**git diff --stat**显示摘要而非整个 diff

上面指令后面加上—filename，可以显示某个文件的改变

## 撤销操作:

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。

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](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

## 回退操作:

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。、

git reset --hard HEAD^。

git reflog用来记录你的每一次命令。

## 分支操作

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

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。

删除分支：git branch -d <name>

## 分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该feature分支。

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

## 多人协作

哪些分支需要推送：

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin branch-name推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin branch-name推送就能成功！

如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream branch-name origin/branch-name。

* 查看远程库信息，使用git remote -v；
* 本地新建的分支如果不推送到远程，对其他人就是不可见的；
* 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
* 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
* 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
* git fetch 会使你与另一仓库同步，提取你本地所没有的数据
* 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

## 标签管理

Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针（但是分支可以移动，标签不能移动），所以，创建和删除标签都是瞬间完成的。

* 命令git tag <name>用于新建一个标签，默认为HEAD，也可以指定一个commit id；
* git tag -a <tagname> -m "blablabla..."可以指定标签信息；
* git tag -s <tagname> -m "blablabla..."可以用PGP签名标签；
* 命令git tag可以查看所有标签。
* 命令git push origin <tagname>可以推送一个本地标签；
* 命令git push origin --tags可以推送全部未推送过的本地标签；
* 命令git tag -d <tagname>可以删除一个本地标签；
* 命令git push origin :refs/tags/<tagname>可以删除一个远程标签。