时光机穿梭

2020-02-08　　17:38

版本回退　　工作区----git add--- >暂存区----git commit--->版本库

git init 初始化一个Git仓库

git add xx.txt 把文件添加到暂存区

git commit -m "wrote a readme file" 把文件提交到仓库（版本库），-m后面输入的是本次提交的说明

git status 了解仓库当前的状态

git diff xx.txt 查看工作区相较于上一次的修改内容，git diff --cached 查看暂存区修改内容

git log 查看提交历史记录（以确定回退到哪个版本），输出信息太多，可以试试加上--pretty=oneline参数。

HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，往上100个版本写成HEAD~100

git reset --hard HEAD^ 从当前版本回退到上一个版本

git reset --hard *** 从当前版本到未来的版本，***为未来版本的commit id

git reflog （以确定回到未来哪个版本）查看记录的每一个版本

工作区和暂存区

- 电脑里能看到的目录就叫工作区
- 工作区有一个隐藏目录 .git，这个不算工作区，而是Git的版本库存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。
- git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后执行git commit就可以一次性把暂存区的所有修改提交到分支。

管理修改

- Git跟踪并管理的是修改，而非文件。
- 每次修改，如果不用git add到暂存区，那就不会加入到commit中

撤销修改

- git checkout -- file，把file文件在**工作区**的修改全部撤销,让这个文件回到最近一次git commit或git add时的状态,用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。
- git reset HEAD <file>（缓存区的撤销）可以把**暂存区**的修改撤销掉，重新放回**工作区**。git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。

删除文件

- rm file 只删除了工作区的文件，如果想要恢复，用git checkout -- file 命令
-

git rm file 不仅从版本库删除了文件，而且还修改了暂存区，	如果想恢复，需要先git reset HEAD file撤销暂存区的修改，然后再git checkout -- file （出现Unstaged changes after reset时输入git stash）
	如果想从版本库中彻底删除该文件，要 git commit -m "remove file"

- git rm --cached file 将缓存区的file文件从缓存区删除，但是工作区并未删除
- git rm -f file 将add到缓存区的file文件从缓存区和工作区都删除

git update-git-for-windows git更新

远程仓库

2020-02-08　　19:32

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

eg:git remote add origin https://github.com/blaoke/learngit.git

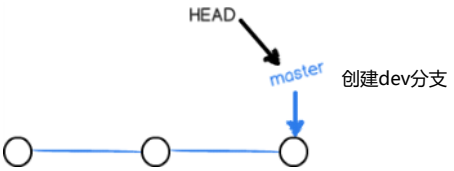
- git push -u origin master 第一次推送master分支的所有内容 origin：远程仓库名称 master：推送分支
- git push origin master 把本地master分支的最新修改推送至GitHub
-

克隆一个远程库到本地库	git clone https://github.com/blaoke/gitskills.git 使用的是http协议
	git clone git@github.com:blaoke/gitskills.git 使用的是ssh协议，速度更快

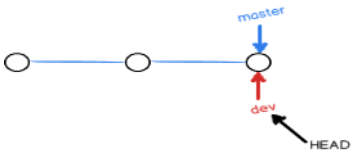
分支管理

2020-02-09　14:55

创建与合并分支



创建dev分支



在多人操作完成项目时，避免不断提交给他人造成麻烦，可以使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。

git checkout -b dev 创建dev分支，然后切换到dev分支。	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 创建分支 git checkout dev 或者git switch dev 切换到dev分支
- git checkout master 切换回master分支
- **最新版本的变化** git switch -c dev 创建dev分支，然后切换到dev分支　git switch master 切换回master分支

- git branch 查看当前分支，当前分支前面会标一个*号。
- git merge dev 把dev分支的工作成果合并到master分支上
- git branch -d dev 删除dev分支

解决冲突

- 当Git无法自动合并分支时，就必须首先解决冲突。解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。
  用 git log --graph --pretty=oneline --abbrev-commit命令可以看到分支合并图。解决冲突后，再提交，合并完成。

分支策略

- 合并分支时，Git默认会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息，看不出来曾经做过合并。如果要强制禁用Fast forward模式改用普通模式合并，使用命令git merge --no-ff -m "merge with no-ff" dev 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来曾经做过合并。
实际开发中分支管理基本原则：	• master分支是非常稳定的，仅用来发布新版本，平时不能在上面干活 • 干活都在dev分支上，dev分支是不稳定的，到某个时候发布新版本时，再把dev分支合并到master上，在master分支发布1.0版本 • 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。 

Bug分支

- 当临时要处理bug时，stash命令 git stash，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
恢复工作现场方法	• git stash apply 但是恢复后，stash内容并不删除(可以用git stash list查看)，需要用git stash drop来删除； • git stash pop，恢复的同时把stash内容删除
- git cherry-pick ******命令，将master分支上修复的bug，“复制”到当前分支，******为master分支修复Bug提交的commit id

Feature分支

- 每添加一个新功能，最好新建一个feature分支，在上面开发，但如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除

多人协作

- 远程仓库的默认名称是origin
- git checkout -b branch-name origin/branch-name，在本地创建和远程分支对应的分支，本地和远程分支的名称最好一致；
- git branch --set-upstream branch-name origin/branch-name ，建立本地分支和远程分支的关联
- git remote：查看远程库的信息，git remote -v显示更详细的信息
	1、试图用git push origin <branch-name>推送自己的修改；
多人协作的工作模式：	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抓取远程的新提交，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-to branch-name origin/branch-name
	3、合并有冲突，则解决冲突，解决的方法和分支管理中的解决冲突完全一样，解决后在本地提交；
	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！
- git remote rm origin 删除已有的GitHub远程库
- git remote add github git@github.com:blaoke/learngit.git 关联GitHub的远程库,远程库的名称叫github

Rebase

- git rebase 把本地未push的分叉提交历史整理成直线，使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

标签管理

2020-02-09 19:25

创建标签

- git tag <tagname> 用于新建一个标签，默认为HEAD，也可以指定一个commit id
- git tag 查看所有标签
- git tag <tagname> ****** 为历史操作打标签，******为commint id，eg:git tag v0.9 f52c633
- git show <tagname> 查看某一标签详细信息
- git tag -a <tagname> -m "blablabla" ****** 创建带有说明的标签，用-a指定标签名，-m指定说明文字

操作标签

- git tag -d <tagname> 删除一个本地标签
- git push origin <tagname> 推送一个本地标签；
- git push origin --tags 推送全部未推送过的本地标签；
- git push origin :refs/tags/<tagname> 删除一个远程标签。

自定义git

2020-02-09　　21:52

忽略特殊文件

- 在Git工作区的根目录下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

配置别名

- git config --global alias.st status 用st就表示status
- git config --global alias.unstage 'reset HEAD' 用unstage(撤销)表示 reset HEAD,即git unstage test.py等于git reset HEAD test.py
- git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit" git log 的美化定义
- 每个仓库的Git配置文件都放在.git/config文件中。当前用户的Git配置文件放在用户主目录下的一个隐藏文件.gitconfig中，通过cat .gitconfig 查看