**-- 使用**

**1. git init (建本地git仓库)-> git add . -> git commit –m “commit message”**

**2. github上建立项目（同名）**

**3. git remote add origin https://github.com/zqunor/lamp.git**

**4. git push origin master**

**-- Experiment about push a few projects into github,and then clone on another computer**

**first: create demo2**

**second: create demo3**

**and then cut demo2 and demo3 into the same directory**

**then push this directory into the remote**

**then clone the remote into a new directory**

**put demo2 and demo3 into newspace**

**import -> existing maven projects -> choose newspace directory -> and then you can choose which project to import**

git add <file1> <file2>… 添加指定文件到暂存区

git commit -m “提交暂存”

* Branch

git鼓励大量使用分支

git branch

查看本地分支

git branch –a

查看所有分支（本地以及线上）

git branch branchName

创建一个本地分支

git branch –d branchName

删除一个本地分支

git checkout <name>

切换分支

git checkout -b <name>

创建+切换分支

git merge <name>

合并某分支到当前分支

**远程分支**

1. **创建远程分支**

**git checkout -b dev**

**git push <远程主机名> 本地分支名:远程分支名**

**例：git push origin dev:dev (一般把远程分支和本地分支同名便于管理)**

1. **删除远程分支**

**方法一：git push origin :dev 推送空分支到远程分支**

**方法二：git push origin --delete dev**

1. **pull、push远程分支(<来源地>:<目的地>)**

**git pull <远程主机名> <远程分支名>:<本地分支名>**

**git pull origin dev:dev**

**也可以 git pull origin dev 拉取与本地分支有追踪关系的远程分支**

**git push <远程主机名> <本地分支名>:<远程分支名>**

**git push origin dev:dev**

**git push orgin dev 推送到与本地分支有追踪关系的远程分支**

* 版本回退（已经commit，还没有push）

git log 查看提交的日志

git log --pretty=oneline 查看提交日志（漂亮点）

git log --graph --pretty=oneline --abbrev-commit 查看分支合并情况

git reset --hard HEAD^ 返回上一个版本

git reset --hard HEAD^^ 返回上上个版本

git reset --hard HEAD~100 返回上100个版本

返回上一个版本后，后悔了，我要恢复，如果bash没有关

则找到git log 中的那个commit-id（**不需要写全，只要前几位就可以，git会自动去找，一般写个5~6位吧**）,使用下面的命令：

git reset --hard commit-Id

如果bash关掉了，则需要使用下面的命令：

git reflog 查看记录的每一次命令找到需要返回的版本的commit\_id

git reset --hard commit\_id

* 撤销修改

1. 没有add到暂存区（stage）

git checkout -- filename **(. 这是撤销所有) 注意：-- 和fileName之间是有空格的**

1. 已经add添加到暂存区，没有提交
2. 先将暂存区的回退到工作区

git reset HEAD filename

1. git checkout -- fileName

* 远程仓库

git remote -v 查看远程仓库版本

$ git pull

There is no tracking information **for** the current branch.

Please specify which branch you want to merge **with**.

See git-pull(1) **for** details.

git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

git branch --set-upstream-to=origin/<branch> dev

git pull也失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

$ git branch *--set-upstream-to=origin/dev dev*

Branch 'dev' **set** up **to** track remote branch 'dev' **from** 'origin'.

//方法一

$ git fetch origin master //从远程的origin仓库的master分支下载代码到本地的origin master

$ git log -p master.. origin/master//比较本地的仓库和远程参考的区别

$ git merge origin/master//把远程下载下来的代码合并到本地仓库，远程的和本地的合并

//方法二

$ git fetch origin master:temp //从远程的origin仓库的master分支下载到本地并新建一个分支temp

$ git diff temp//比较master分支和temp分支的不同

$ git merge temp//合并temp分支到master分支

$ git branch -d temp//删除temp

* 比较

git diff

比较的是工作区和暂存区文件的区别（上一次git add后的内容）

git diff --cached

比较的是暂存区的文件和仓库分支（上次git commit后的内容）里的文件的区别

git diff HEAD -- filename**（当只有一个文件的时候可以 -- fileName不写）**

比较的是工作区和版本库里最新版本的区别

测试：

修改三次分别放在：未add、add、commit

* 管理修改

**Git 管理的是修改而不是文件**

测试：

第一次修改 - > add

第二次修改

Commit

git status 发现第二次的修改没有提交

git diff HEAD -- filename**（当只有一个文件的时候可以 -- fileName不写）**发现第二次修改确实没有被提交

* 删除文件

1. 直接在工作区手动删除
2. 或者在bash中使用 rm test.txt删除

删除后需要 add，commit

* 合并解决冲突

测试：

1. 新建dev分支，修改、add、commit
2. 切换到master分支，修改相同处、add、commit
3. Master分支下，合并dev，git merge dev

|  |
| --- |
| Auto-merging src/test/java/com/example/demo/DemoApplicationTests.java  CONFLICT (content): Merge conflict in src/test/java/com/example/demo/DemoApplicationTests.java  Automatic merge failed; fix conflicts and then commit the result. |

有冲突产生

1. git status 查看冲突产生的文件
2. 回到工作区（开发工具）发现，冲突文件冲突处内容为：

<<<<<<< HEAD

Creating a new branch is quick & simple.

=======

Creating a new branch is quick AND simple.

>>>>>>> dev

6．将head行、===行、dev行删掉，冲突内容修改成最终要提交的

7. 最后再add、commit

8. 查看分支的合并情况

git log --graph --pretty=oneline --abbrev-commit

合并分支时如果使用git merge --no-ff -m “commit message” dev（该方式会重新生成一个commit，可以在log中看出分支信息）如下：

|  |
| --- |
| \* bffe7d3 (HEAD -> master) no-ff merge  |\  | \* 4a93ec9 (dev) no-ff解决冲突-dev  \* | e894677 no-ff master conflict merge  |/ |

1. **实际开发中，master是稳定的，仅用来发布新版本，不能再上面干活**
2. **干活都是在dev（自定义）分支上，dev分支是不稳定的，要发布版本的时候把dev分支合并到master上，你和你的小伙伴是在dev分支上干活的**

* Bug分支

我个人觉得场景是这样的。设A为游戏软件

1、master 上面发布的是A的1.0版本

2、dev 上开发的是A的2.0版本

3、这时，用户反映 1.0版本存在漏洞，有人利用这个漏洞开外挂

4、需要从dev切换到master去填这个漏洞，正常必须先提交dev目前的工作，才能切换。

5、而dev的工作还未完成，不想提交，所以先把dev的工作stash一下。然后切换到master **（git stash）**

6、在master建立分支issue101并切换.

7、在issue101上修复漏洞。

8、修复后，在master上合并issue101 9、切回dev，恢复原本工作，继续工作。

**git stash list 查看stash内容**

**git stash pop 恢复stash到工作现场并删除stash内容**

或者

git stash apply 恢复stash但是stash内容不删除

git stash drop 删除stash内容

**多次stash的时候：（如果stash了两次，则最后一次为stash@{0},第一次为stash@{1},相当于堆栈，先入后出）**

1. **git stash list查看**
2. **git stash apply stash@{0} 恢复指定stash**
3. **git stash drop stash@{0} 删除指定stash**

* tag标签使用

git tag --help : 查看tag的使用

git tag <tagname> : 创建标签（tag默认是打在HEAD上-及最新提交的commit上）

git tag <tagname> <commit\_id> : 创建标签，打在指定的commit上

git tag : 查询所有标签（**标签是指向commit的死指针，不管切换到那个分支，查询到的tag都是所有的tag，标签不是按照时间顺序排序的，而是按照字母顺序排序的**）

git show <tagname> : 显示标签信息

git tag -a <tagname> -m “commit message” <commit\_id> : 创建带有说明的标签，-a 指定标签名，-m 指定说明文字

**注意：标签总是和commit挂钩，如果某个commit既出现在master又出现在dev分支，那么这两个分支都能看到这个标签（查看log的时候能看到）**

**操作tag:**

创建的tag只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。

推送到远程：

git push origin <tagname> : 推送指定标签到远程

git push origin --tags : 推送所有标签到远程

删除标签（已经推送到远程）：

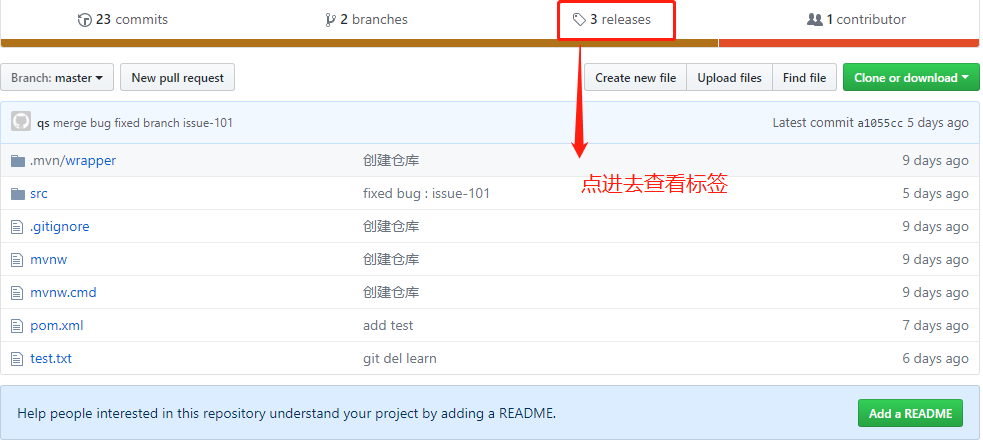
1. 删除本地标签

git tag -d <tagname>

1. 删除远程标签

git push origin :refs/tags/<tagname>

远程看标签：



**查询某次历史提交的修改内容**

**git log**

**git log -3 --stat : 查询最近3条日志，并且显示涉及的文件**

git show <commit-id> [filename] ：filename可选参，指定提交的修改

git show <commit-id> --stat : 显示指定提交的涉及的文件