**Git版本控制工具**

* Git 创建仓库：

1. 使用当前目录作为git仓库:git init
2. 使用指定目录作为仓库：git init newdir

* 基本概念：

1. 工作区：
2. 暂存区：
3. 版本库：

* 基本操作：

设置全局变量：

git config --global user.name "your name"

git config --global user.email "your\_email@youremail.com"

查看仓库状态：git status

查看文件区别：git diff 文件

追加的代码用绿色表示，删除的代码用红色表示。

Git 查看提交历史:

git log

git log --oneline

git log --oneline --graph

版本穿越恢复搭配指定的提交点（版本）：

git reset –hard 版本

git checkout file（检出指定文件，并覆盖工作区和暂存区）,用于撤销已经提交到暂存区但还未提交到版本库中的文件。

git checkout(检出全部文件，此操作十分危险，将覆盖工作区和暂存区)

删除文件：del read.txt->git add read.txt->git commit read.txt –m ‘删除文件read.txt’

* GitHub

1. 简介

* GitHub是个大型的远程仓库，可有用户自己创建
* 公共项目（免费）：所有人可以获取你的代码
* 私有项目（收费）：用于公司项目

1. 删除项目操作：setting->Delete this repository(在最底部)
2. 配置远程仓库

新建仓库->配置SSHKey；

配置SHHKey信息：

* 生成一个RSA的认证的key的信息ssh-keygen-t rsa

ssh-keygen -t rsa -C [your\_email@youremail.com](mailto:your_email@youremail.com)

以上命令成功后会在目录下/Users/用户/.生成.ssh文件夹，打开id\_rsa\_pub，复制里面的key.

* 取得生成的公钥和私钥(id\_rsa\_pub里的key)，将公钥配置在GitHub中
* 验证是否配置成功，在git bash:

ssh –T your\_email@youremail.com

1. 测试远程仓库是否可以连接：
2. 执行命令ssh –T [your\_email@youremail.com](mailto:your_email@youremail.com)

若提示SHH 22端口连接超时，则需要在.ssh目录下修改或新建config文件，添加配置如下：

Host qq.com

User xxxx@qq.com

Hostname ssh.github.com

PreferredAuthentications publickey

IdentityFile ~/.ssh/id\_rsa

Port 443

之后按提示一定要输入yes，否则Host key verification failed.

若出现Warning: Permanently added '[ssh.github.com]:443,[192.30.253.122]:443' (RSA) to the list of known hosts.

**这个问题查了许久，发现是因为有时候防火墙会拒绝 SSH连接**（Sometimes, firewalls refuse to allow SSH connections entirely. ）在github的帮助中可以看到该问题。他们也给出了解决方案[https://help.github.com/articles/using-ssh-over-the-https-port/](https://link.jianshu.com?t=https://help.github.com/articles/using-ssh-over-the-https-port/" \t "_blank)

通过命令：ssh -T -p 443 [git@ssh.github.com使用克隆过的SSH连接HTTPS端口](mailto:git@ssh.github.com使用克隆过的SSH连接HTTPS端口)。

添加远程地址：

git remote add origin <https://github.com/SmileBitter/first_git.git>

本地文件修改后提交到版本库，实质上远程仓库没有更新，需要执行合并操作：

git push –u origin master

5、仓库克隆

所谓仓库克隆就是将远程服务器的仓库内容抓取到本地：

* SSH连接：[git@github.com](mailto:git@github.com):账号/仓库.git
* HTTPS：<https://github.com/>账户/仓库.git

若执行不成功，那么可能是本地仓库环境有问题，可以有两种处理方式:

* 删除本地的远程仓库:gitremove rm oringin;
* 重新与远程仓库建立一个连接:gir remote add origin 地址

6、创建和合并分支

查看仓库中可用分支：git branch,当前在使用的分支用“\*”标记

创建新分支：git branch 分支名

切换分支：git checkout 分支名(本质上是HEAD指针的切换)

分支合并：

快速合并：先切回要要发布使用的分支上，在合并分支。执行

git merge dev

快速合并（Fast-forward）：HEAD指针将指向dev分支。

如果此时原本的远程仓库已经被删除了，那么建立一个新的远程超链接：

git remote set-url origin [git@github.com:账号/仓库名.git](mailto:git@github.com:账号/仓库名.git)

删除分支：

1、删除本地仓库分支：git branch –d dev

2、删除远程仓库：

（1）直接删除:git push origin –-delete dev

(2) 直接推送一个空的分支：

git push origin :dev

7、冲突解决

两个分支修改了同一个文件，并且同时都进行了提交。这种情况会产生冲突。

需手工修改文件，并提交处理。

8、分支合并模式

快速合并模式(FF)：Fast-forward,直接改变了head的指向，不会有提交点，对于历史版本回退相当麻烦。

NO-FF:提交之后默认追加一个提交点，记录每个分支的记录情况。

git merge –no-ff –m ‘Use No-ff Merge dev’ dev

dev分支为开发总分支，

9、bug分支

1)、考虑到开发中要解决bug问题，需要将此时代码暂存，执行git stash.

2)、由于需要修改bug， 并且保证master的分支可用，所以要将当前分支切换回master分支，git checkout master.

3)、创建并切换到bug分支上：

git checkout –b bug

4)在bug分支上将代码进行修改同属将修改后的bug分支与master分支合并,并删除bug分支。

5)、查看挂起的工作区，git stash list

6)、进行恢复操作：

（1）分步进行：恢复暂时挂起的工作区，执行git stash apply;清楚挂起 的工作区,执行git stash drop。

(2) 一步完成：从暂存区恢复同时进行清除：git stash pop.

10、feature分支(扩展分支)

1）、创建并且切换到feature分支:

git checkout –b feature.

2)、需要进行删除时无法删除，因为此时分支上存在未合并的内容。

3）、需要执行git branch –D feature

11、补丁

将两个不同分支的文件不同部分做个差别化的标记，这个文件叫做补丁文件。

1. 创建并切换到dev分支：

git checkout –b dev

1. 在dev分支上修改文件：
2. 这时候不能讲整个文件提交，可以不同部分作出标记git diff 文件。。
3. 将修改后的代码提交，
4. 将dev与master文件内容进行比较，并且将结果保存在补丁文件中：

将比较的结果保存在“patch”文件中。git diff master > patch

1. 切换回到master分支上，随后应用此补丁文件：

git checkout master

git apply patch

1. 创建并且切换到pat分支上：

git checkout –b pat

1. 在pat分支上进行代码的部分修改，并且将其进行提交：
2. 生成格式化补丁：

git format-patch –M master

1. 应用补丁文件

git checkout master

xxx.patch

1. 查看远程仓库：

git remote –v

多人开发

1. 创建并切换到dev分支中：

git checkout –b dev

1. 在dev分支上对一些代码进行修改，随后将其提交到仓库中：

git add.

git commit –m “”

1. 将两个分支一起提交给远程的GitHub上：

git push origin master

git push origin dev

1. 查看远程仓库git remote –v
2. [第二个开发者] 打开一个新的窗口，表示第二个开发者，

创建一个新文件夹，克隆远程仓库 git clone 远程地址

1. 第二个开发者]将远程仓库的dev分支抓取到本地的yootk分支。

目的是模拟本地仓库名称与远程不同，

git checkout –b yootk origin/dev

1. 想继续抓取远程仓库中的分支：git pull,如果抓取不了则执行git branch –set-upstream-to=origin/dev yootk,再重新抓取。如果远程分支与本地名称相同则不用执行git branch –set-upstream-to=origin/dev yootk，直接抓取即可。
2. 第一个开发者修改文件并提交，并推送内容。
3. 第二个开发者，也修改了本地仓库文件，提交并推送。结果推送了一个新的分支。此时应该先抓取下数据git pull，发现存在冲突。所以第二个开发者需手动解决。然后在提交，推送。
4. 第一个开发者抓取数据，发现第二个开发者提交的yootk分支抓取不到。需要设置git branch –set-upstream-to=origin/yootk dev，然后再次执行抓取操作。
5. 以上操作发现不同开发者建立的分支不一样，相当麻烦，所以最好使用相同名称比较好：

第二个开发者要删除远程的yootk分支，建立并切换到dev分支，这样就和远程仓库内容相同，删除前内容必须先推送到远程仓库；

git checkout mster

git push origin –delete yootk

git branch –D yootk

git checkout –b dev

第二个开发者抓取数据

git checkout –b dev origin/dev

这样修改提交的时候就会很方便。

Git 标签管理：

1. 表示对提交点的一个名称定义
2. 范例：将当前的提交点设置为v1.0版本

git tag v1.0

如果现在需要查询有哪些标签，则可以使用“git tag”命令查看。

1. 查看所有的标签

git tag

那么是不是只能为最后一个提交点设置标签，这是不一定的，也可以为历史提交点设置。

1. 查询历史提交点

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

1. 分别设置三个标签

git tag v0.1 提交点

git tag v0.2 提交点

git tag v0.3 提交点

1. 查看标签

git show v0.6

1. 范例：重新建立“v0.1”标签,同时为其设置说明文字

girl tag –d v0.1

git tag v0.1 提交点 –m “注释”

1. 将标签推送到远程仓库

git push oringin v1..0 单个上传

git push origin –tags 批量上传标签

1. 删除v0.1标签

git tag –d v0.1

git push origin :refs/tags/v0.1