**Git-分布式版本控制系统**

集中式：版本库集中放在中央服务器，需联网才能工作。

分布式：没有中央服务器，版本库就在自己的电脑上

**安装后的设置**

$ git config –global user.name “Galium”

$ git config –global user.email “896662351@qq.com”

**版本库（repository）**：一个目录——里面所有文件可以被Git管理，每一次操作都能被跟踪，还原

创建版本库

**创建目录：**

|  |  |
| --- | --- |
| $ mkdir learngit | 创建新目录 |
| $ cd learngit | 进入目录主分支 |
| $ git init | 初始化目录 |
| $ git add “filename” | 把文件添加到stage区 |
| $ git commit –m“注释” | 将修改提交到所处分支上 |
| $ git status | 掌握仓库当前的状态 |
| $ git diff | 如果文件被修改过，这个命令可以查看修改内容 |

**回退**

HEAD表示当前版本，上一个版本就是HEAD^ ,由此类推，往上100个版本可以写成HEAD~100

|  |  |
| --- | --- |
| $ git log | 从近到远显示提交日志 |
| $ git log –pretty=oneline | 输出信息排列整齐，简洁化 |
| $ git reflog | 显示每一次的命令，可显示commit id |
| $ git reset –hard commit\_id/HEAD | 回退到版本 |

**工作区和暂存区**

工作区：电脑直接可以看到的目录

版本库：分stage区（暂存区）和分支区，当使用add命令，修改被提交到stage区，commit之后提交到所处分支上，每次修改，如果不add到暂存区，commit时就不会被提交到分支中

|  |  |
| --- | --- |
| **撤销修改：** | |
| $ git checkout –file（文件名） | 撤销工作区 |
| $ git reset HEAD file $ git checkout –file | 撤销暂存区 |
| $ git reset –hard commit\_id | 撤销分支 |
| **删除文件** | |
| $ git rm filename |  |
| $ git commit –m” ” | 从版本库中删除文件，用rm命令删除，并提交 |

Github

注册账号

**添加远程库**

|  |
| --- |
| Create a new repository |
| 创建一个新的Git仓库 |
| $ git remote add origin [git@github.com:Galium/learngit.git](mailto:git@github.com:Galium/learngit.git) |
| 关联本地与远程仓库，远程库的名字默认origin |
| $ git push –u origin master |
| 第一次推送master分支，加上了-u参数，把本地内容推送远程，新的分支同时关联起来，之后不用-u参数 |

**从远程库克隆**

|  |
| --- |
| Create a new repository:gitskills |
| $ git clone [git@github.com:Galium/gitskills.git](mailto:git@github.com:Galium/gitskills.git) |

**分支管理**

|  |  |
| --- | --- |
| $ git checkout –b dev | 创建并切换到dev分支，相当于git branch dev+git checkout dev |
| $ git branch | 查看当前所有分支，当前所在分支前会有一个\*号 |
| $ git merge dev | 合并分支 |
| $ git branch –d dev | 删除分支 |

**解决冲突**

|  |  |
| --- | --- |
| $ git log –graph –pretty=oneline –abbrev-commit | 查看分支合并图 |

**分支管理策略**

通常合并分支时，如果可能，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。

如果要禁用fast-forward模式，Git就会在merge的时候生成一个新的commit，这样，从分支历史上就可以看出分支信息。

|  |
| --- |
| $ git merge –no-ff –m”注释” dev. |
| $ git log –graph –pretty=oneline –abbrev-commit（重新查看分支情况） |

**Bug分支**

当手头有工作时候却需要修复bug，则需储存当前分支

|  |  |
| --- | --- |
| $ git stash | 把当前工作现场储藏起来 |
| $ git checkout master |  |
| $ git checkout –b issue-01 | 创建一个分支修复bug |
| $ git readme.txt |  |
| $ git commit –m “fix bug 101” | 修复bug后提交 |
| $ git checkout master |  |
| $ git merge –no-ff –m”merged bug fix 101” issue-101 | |
| $ git branch –d issue-101 | 删除bug分支 |
| 回到dev |  |
| $ git checkout dev | 工作区是干净的 |
| $ git status |  |
| $ git stash list | Git把工作现场存在某个地方了 |
| $ git stash apply+$ git stash drop（删除stash内容）/$ git stash pop（直接删除stash内容） | |
| $ git stash list | 此时看不到内容 |

**Feature分支**

开发一个新的feature，最好新建一个分支

**多人协作**

当你从远程库克隆时，实际上git自动地把master 分支和远程的master分支对应起来了，并且，远程仓库默认origin

解决冲突后再push

|  |  |
| --- | --- |
| $ git remote | 查看远程库的信息 |
| $ git remote –v | 更仔细 |
| $ git push origin master | 远程推送分支 |
| $ git pull | 抓取远程的新提交 |
| $ git checkout –b branch-name origin/branch-name | 在本地创建和远程对应的分支，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 |

**标签**

**1.标签管理**

发布一个版本时，现在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来，所以标签也是版本库的一个快照。

相当于某个指向commit的指针，不可移动，创建删除瞬间完成

**2.创建标签**

|  |  |
| --- | --- |
| $ git checkout master | 切换到分支上 |
| $ git tag v1.0 | 打标签 |
| $ git tag | 查看所有标签 |
| //默认标签打在最新提交的commit上，也可通过commit id打标签 | |
| $ git log –pretty=oneline –abbrev-commit |  |
| $ git tag v0.9 6224b3a |  |
| //标签不是按时间顺序列出，而是按字母顺序 | |
| $ git show (v0.9) |  |
| /$ git tag –a v0.1 –m”version 0.1 released” 3628164 | 带有说明的标签 |
| $ git tag –s<tagname> -m “balabala” | 可以用PGP签名标签 |

**3.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 –tag | 推送所有未推送过的标签 |
| $ git push origin: refs/tags/<tagname> | 可删除一个远程标签 |