Git的知识点整理

目录

[Git的知识点整理 1](#_Toc910)

[1.git的基本概念 1](#_Toc3295)

[1.1什么是Git？ 2](#_Toc10175)

[1.2 Git的作用 2](#_Toc1304)

[1.3为什么要做版本控制 2](#_Toc11214)

[1.4安装GIT 2](#_Toc24922)

[1.5 git做版本控制的步骤 2](#_Toc20958)

[1.6git运行的注意事项 2](#_Toc25166)

[1.7.git的三大区域 2](#_Toc2745)

[1.8 Git工作流 2](#_Toc17715)

[1.9 Github/Gitlab 2](#_Toc30653)

[2.git的基本操作 3](#_Toc683)

[2.1 git的基本命令 3](#_Toc24820)

[2.2忽略文件（ignore掉数据库文件） 3](#_Toc21714)

[2.3删除 3](#_Toc26938)

[2.4回滚 3](#_Toc18277)

[2.5分支以及命令补充 3](#_Toc22634)

[2.6 Git rebase变基（使提交记录变得及简洁）及其命令 4](#_Toc27069)

[3.基于github的代码管理 4](#_Toc21457)

[3.1 github的代码管理（两地开发）的部分命令 4](#_Toc14815)

[3.2代码合并（冲突） 4](#_Toc24456)

[3.3 beyond compare软件快速解决冲突 5](#_Toc22784)

[3.4协同开发（git flow工作流） 5](#_Toc5994)

[3.5 Tag版本管理 5](#_Toc23848)

[3.6代码review 5](#_Toc17581)

[3.7给开源项目贡献代码 5](#_Toc28882)

[3.8 Git配置文件的存放位置 5](#_Toc23925)

[3.9 Git的免密登录（三种方式） 6](#_Toc14532)

[3.10 Github任务管理 6](#_Toc9692)

[4. 部分概念问题与图 6](#_Toc31093)

[4.1 git rebase 与git merge的区别： 6](#_Toc9785)

[4.2 git几大区域的关系图 6](#_Toc176)

[4.3工作流图 7](#_Toc2229)

1.git的基本概念

1.1什么是Git？

分布式版本控制软件

1.本地版本控制，实现回滚（把所有版本存放在本地）

2.集中式版本控制（把所有版本存放在中心）

3.分布式版本控制工具（把所有版本都放在了本地和中心）

分布式版本控制工具与集中式版本控制的区别

1.集中式版本控制只把版本都放在中心，中心数据丢失则所有丢失；分布式版本控制把所有版本不仅存放在中心，还存放在本地。

2.分布式提交版本先提交到本地，再同步到中心，集中式只能提交到中心。

1.2 Git的作用

1.实现协同开发软件。

2.保存源码。

3.版本控制。

1.3为什么要做版本控制

1，防止线上代码出问题而做回滚。

2，版本更替减少工作量。

1.4安装GIT

1.5 git做版本控制的步骤

1，进入要管理的文件夹

2，初始化

3，管理

4，生成版本

工作区：新增的文件或者修改过的文件。

暂存区：已被管理起来版本。

版本库：保存生成的版本。

1.6git运行的注意事项

个人信息配置

1.7.git的三大区域

1，工作区

2，暂存区

3，版本库

1.8 Git工作流

1.默认分支master

2.预发布分支releas

3.研发分支development

4.功能或个人分支XX

5.修改bug分支debug

1.9 Github/Gitlab

1.Gitub：代码仓库网站/Gitlab：代码仓库软件

2.注册账号

3.创建仓库

4.本地代码传递

2.git的基本操作

2.1 git的基本命令

1.git init //初始化

2.git status //检测当前目录下文件的状态

3.git status -s 和 git status -short //检测当前目录下文件的状态获得更简短的输出

4.git add.(或文件名) //将此文件添加到暂存区

5.git commit -m ‘版本名称’ //生成版本

6.git commit -a -m ‘版本名称’ //跳过暂存区，直接上传到本地版本库

7.git log//查看版本记录

8.git config --global user.email “” //配置个人邮箱

9.git config --global user.name ”” //配置个人姓名

10.git reset --hard 版本号 //回滚到之前版本

11.git reflog //查找所有版本号

12.git diff //显示添加和删除的确切行

2.2忽略文件（ignore掉数据库文件）

可以创建文件列表模式以匹配名为.gitignore的文件。

.gitignore文件中的模式的规则如下：

1.空行或以#开头的行的文件将被忽略。

2.可以使用\*来进行匹配。

3.使用正斜杠（/）结束模式以指定目录，忽略文件夹下所有文件。

4.以感叹号（!）开头来否定模式。

2.3删除

将文件，目录和文件全局模式传递给git rm命令，

1.git rm //删除文件

2.git rm log/\\*.log //删除log/目录下的所有后缀有.log的文件

3.git rm \\*XX //删除文件名以XX结尾的所有文件

2.4回滚

1.git reflog //查找所有版本（回滚到之前版本后，git log不会再显示后面版本）

2.从后往前回滚 git reset --hard 版本号 //回滚到XXX版本号

2.5分支以及命令补充

版本间有指针，指针指向上一个版本未变动的内容，线上代码出bug，创建新的bug分支，修改完成后合并到master分支中去，Master分支为主路线。

1.git branch //查找分支

2.git branch xxx //创建xxx分支

3.git checkout 分支名 //切换分支

4.git merge XX //分支合并，先切换到合并到的分支上

5.git branch -d 分支名 //删除分支

6.git log --graph //以图的形式显示版本记录

7.git log --graph --pretty=format:”%h %s” //以图的形式简化限时版本记录

冲突：master和另一个分支进行合并后。如果两个分支都对master分支里面同名文件的某一行做了修改。会产生冲突，此时需要手动修改。

2.6 Git rebase变基（使提交记录变得及简洁）及其命令

1.把多个提交记录整合为一个提交记录

-整合记录时不要合并已push到远程仓库的记录

-git rebase -i HEAD~数字 //合并包含本版本在内的几个版本记录

-git rebase -i 版本号 //合并当前版本到版本号之间的版本记录

2.把两个分支合并成一条分支

-git rebase 分支名 //两个分支合并为一条分支

3.防止产生分支

-git fetch origin dev

-git rebase origin/dev

变基时遇到冲突时的步骤

-手动解决冲突

-git add 冲突文件名

-git rebase continue

3.基于github的代码管理

3.1 github的代码管理（两地开发）的部分命令

1.git remote add origin 远程仓库地址 //给远程仓库命名为origin

2.git push -u origin 分支 //向远程仓库传递代码

3.git clone 远程仓库地址 //克隆远程仓库代码（内部已实现git remote origin 远程仓库地址）已克隆所有分支

4.git checkout 分支名 //切换分支

5.git push origin 分支 //将代码传送到远程仓库

6.git pull origin 分支 //将远程仓库代码拉下来

在dev分支上开发时代码要与master里最新版本一致

3.2代码合并（冲突）

1.git pull origin dev等同于git fetch origin dev + git merge origin/dev

2.git fetch origin dev //在远程仓库把代码拉到版本库

3.git merge origin/dev //把版本库代码拉到工作区

为什么是origin/dev？

因为从远程仓库拉到版本库时，为了表示代码来自远程仓库，会加一个origin的前缀。

3.3 beyond compare软件快速解决冲突

**1.安装beyond compare**

**2.在git中配置，配置命令如下**

-git config --local merge.tool bc3

-git config --local merge.path ‘beyond compare路径’

-git config --local mergetool.keepBackup false //不保留备份（--local 只在当前文件内有效）

3.遇到冲突时使用beyond compare解决冲突

-git mergetool //解决冲突

3.4协同开发（git flow工作流）

1.每个功能或者人员有自己的分支，git merge dev，只在自己功能分支开发。

2.每个功能写完之后进行code review，git brench dev；git merge 功能分支合并到一起

3.在release分支进行测试，git brench release； git merge dev，测试无误修改bug后提交到master分支进行发布，提交到dev分支进行继续开发

4.线上代码出bug，创建bug分支，使用bug分支修改bug，修改完毕后切换到master分支，git merge bug

3.5 Tag版本管理

1.git tag -a v1 -m “描述信息” //管理版本号

2.git checkout -b dev //创建并切换到dev分支

3.git push origin dev //上传dev分支

4.git push origin --tags //上传版本

3.6代码review

使用pull request把自己开发完毕的功能交给项目leader

注意：产生冲突，解决冲突，在进行合并

3.7给开源项目贡献代码

1.fork源代码，把别人代码拷贝到自己园区内从仓库

2.git clone到本地

3.pull request到开源项目

3.8 Git配置文件的存放位置

**1.存放到当前项目，配置文件只对当前项目有效**

git config --local user.name “用户名”

git config --local user.email “用户邮箱”

**2.存放到全局，配置文件对所有项目有效**

git config --global user.name “用户名”

git config --global user.email “用户邮箱”

**3.存放到系统,配置文件对系统项目有效，需要root权限**

git config --system user.name “用户名”

git config --system user.email “用户邮箱”

Git remote add origin默认配置在当前项目配置文件中

3.9 Git的免密登录（三种方式）

**1.url中体现**

在初始时 git remote add origin https://用户名：密码XXXXX.com

git push origin master

**2.ssh密钥**

-生成公钥和私钥（默认放在~/.ssh目录下）,id\_psa.pub为公钥,id\_psa为私钥

ssh-keygen

-拷贝公钥内容。并设置到github中。

-在git本地配置ssh地址 git remote add origin 公钥地址

-git push origin master

**3.git自动管理凭证**

3.10 Github任务管理

1.Issues：文档及任务管理

2.Wiki：项目文档

1. 部分概念问题与图
   1. git rebase 与git merge的区别：

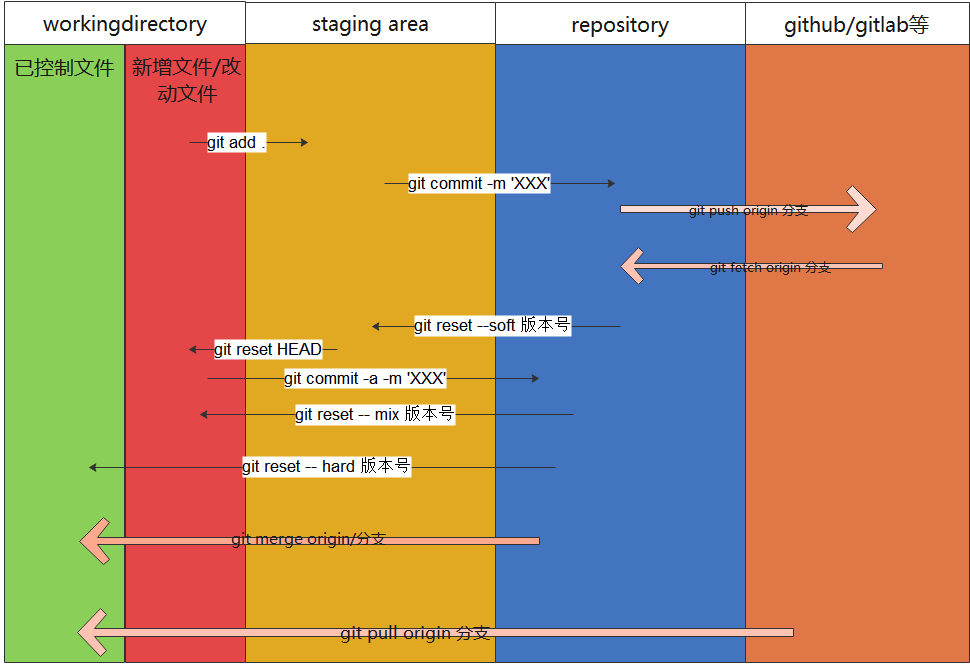
**1.采用merge和rebase后，merge命令不会保留merge的分支的commit。**

即git merge是在master分支中新建了一个版本，版本指针指向dev分支版本，而git rebase 是两个分支合并，两个分支变成一个分支，执行后dev分支不再有版本。

**2.解决冲突不同**

使用merge命令合并分支，解决完冲突，执行git add .和git commit -m 'fix conflict'。这个时候会产生一个新的commit。使用rebase命令合并分支，解决完冲突，执行git add .和git rebase --continue，不会产生额外的commit。更加干净。分支上不会有无意义的解决分支的commit；坏处是如果合并的分支中存在多个commit，需要重复处理多次冲突。

4.2 git几大区域的关系图



4.3工作流图

