# Git操作手册

## 安装与配置

### 1.1 下载安装

<https://git-scm.com/download/>

### 1.2 配置

* + git config --global credential.helper store
  + git config --system --unset credential.helper
  + git config –-global --add user.name “xxx”
  + git config –-global --add user.email [“xxx@qq.com”](mailto:“xxx@qq.com”)
  + git config --global --list
  + git config --global --unset-all user.name

设置credential helper  
凭据可以设置为全局，也可以设置为仅仅在当前的项目下起作用。  
如果想使用全局，则设置为：  
git config –global credential.helper store –file=git\_credentails.  
如果仅仅设置当前项目有效，则进入项目代码目录下，运行：  
git config credential.helper store –file=.git\_credentails.  
上面的–file=.git-credentials表示你的用户名密码存储的文件目录结构，即存放在哪里由你指定。

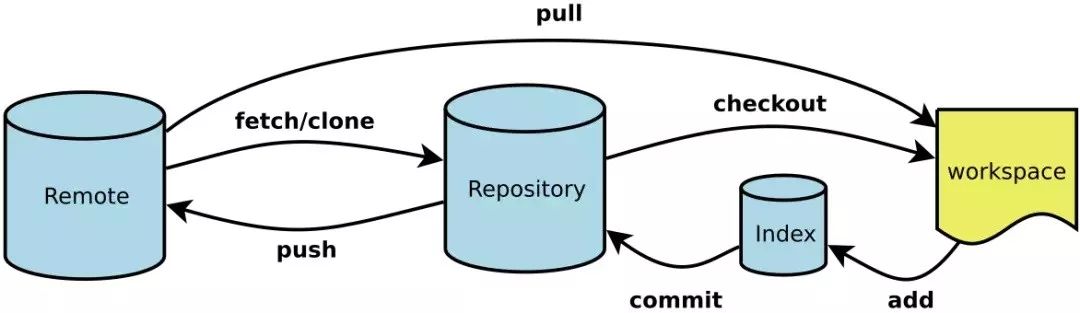
### 1.3 工作原理/流程

Workspace：工作区

Index / Stage：暂存区

Repository：仓库区(或本地仓库)

Remote：远程仓库



## 2 常用操作介绍

### 2.1 版本管理

### 2.1.1 创建版本库（Master）

git init

git status

git diff “文件”

git add “文件”

git commit –m “注释”

### 2.1.2 版本回退

git log --pretty=online

git reflog --pretty=online

git reset --hard HEAD^

git reset --hard HEAD~100

git reset --hard 版本号

### 2.1.3 撤销修改

git checkout –文件名

### 2.1.4 管理远程仓库

git remote add origin “远程仓库地址”

git push [–u] origin “分支名称”

git clone “远程仓库地址”

### 2.1.5 创建本地分支

git branch “分支名字”

git checkout “分支名字”

git checkout –b “分支名字” [origin/远程分支名称] no-mapping

git fetch origin “远程分支名：本地分支名”

git branch –d “分支名字”

### 2.1.6 查看分支

git branch –a

git remote -v

### 2.1.7 分支合并

git merge [--no-ff] “分支名称” –m

git pull origin “分支名称”

git pull --all

git push origin “分支名称”

冲突处理：

找到：<<<<<<<，=======，>>>>>>> 删除不要的内容后，删除这些符号。

## 3 分支管理规范

### 3.1 分支命名规范

#### 3.1.1 master 分支

master 为主分支，也是用于部署生产环境的分支，确保master分支稳定性

master 分支一般由develop以及hotfix分支合并，任何时间都不能直接修改代码

#### 3.1.2 develop 分支

develop 为开发分支，始终保持最新完成以及bug修复后的代码

一般开发的新功能时，feature分支都是基于develop分支下创建的

#### 3.1.3 feature 分支

开发新功能时，以develop为基础创建feature分支

分支命名：feature/开头的为特性分支，命名规则: feature/abs、 feature/mysql

#### 3.1.4 release分支

release 为预上线分支，发布提测阶段，会以release分支代码为基准提测。当有一组feature开发完成，首先会合并到develop分支，进入提测时，会创建release分支。如果测试过程中若存在bug需要修复，则直接由开发者在release分支修复并提交。当测试完成之后，合并release分支到master和develop分支，此时master为最新代码，用作上线。

#### 3.1.5 hotfix 分支

分支命名: hotfix/ 开头的为修复分支，它的命名规则与 feature 分支类似

线上出现紧急问题时，需要及时修复，以master分支为基线，创建hotfix分支，修复完成后，需要合并到master分支和develop分支

### 3.2 日志规范

#### 3.2.1 日志规范目标

* 加快review的流程
* 帮助我们编写良好的版本发布日志
* 让之后的维护者了解代码里出现特定变化和feature被添加的原因

#### 3.2.2 日志内容

##### 3.2.2.1 内容要求

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

type: 本次 commit 的类型，诸如 bugfix docs style 等

scope: 本次 commit及的范围

subject: 简明扼要的阐述下本次 commit 的主旨，在原文中特意强调了几点

1. 使用祈使句

2. 首字母不要大写

3. 结尾无需添加标点

body: 同样使用祈使句，在主体内容中我们需要把本次 commit 详细的描述一下，比如此次变更的动机，如需换行，则使用 |

footer: 描述下与之关联的 issue 或 break change

Type的类别说明：

* feat: 添加新特性
* fix: 修复bug
* docs: 仅仅修改了文档
* style: 仅仅修改了空格、格式缩进、都好等等，不改变代码逻辑
* refactor: 代码重构，没有加新功能或者修复bug
* perf: 增加代码进行性能测试
* test: 增加测试用例
* chore: 改变构建流程、或者增加依赖库、工具等

##### 3.2.2.2 格式要求

# 标题行：50个字符以内，描述主要变更内容  
#  
# 主体内容：更详细的说明文本，建议72个字符以内。 需要描述的信息包括:  
#  
# \* 为什么这个变更是必须的? 它可能是用来修复一个bug，增加一个feature，提升性能、可靠性、稳定性等等  
# \* 他如何解决这个问题? 具体描述解决问题的步骤  
# \* 是否存在副作用、风险?  
#  
# 如果需要的话可以添加一个链接到issue地址或者其它文档

## 4 结合gitlab的codeReview

