# 遇到问题

进入Unix模式退出

ESC + ZZ（大写）

## 从dev分支上更新代码

1.与远程仓库建立连接：git remote add origin XXXXX.git

2.使用git branch 查看本地是否具有dev分支

3.如果没有 git fetch origin dev

4.git checkout -b dev origin/dev在本地创建分支dev并切换到该分支

5.git pull origin dev就可以把gitLab上dev分支上的内容都拉取到本地了

6. 本地分支重命名

git branch -m xxx yyy：将名字为xxx的分支重命名为yyy

7.远程分支重命名

git push --delete origin dev

git branch -m dev develop

git push origin develop

## 上传代码

git add .

（取消add -----[git reset 文件名]）

4.git commit -m "自己想写的提交备注"

（取消commit –[git reset –soft HEAD^]）



\*\*\*在进行这个操作之前需要自己登陆git，用户名为自己登陆gitlab的账号，邮箱为自己使用的邮箱。

5.第一次创建develop分支请执行 git branch develop 今后就不用再执行了

6.git branch -a 查看本地和远程主机的全部分支（可不执行）

7.git checkout develop 切换到develop分支(默认是在master分支)

8.git pull --rebase origin develop 从远程主机的develop分支下拉代码并合并，如果没报错那么执行9，否则自行百度解决冲突。

pull需要输入账号密码，即自己登陆gitLab的账号和密码。

9.git push -u origin develop 把代码推到远程主机的develop分支

## 合并分支

假如我们现在在dev分支上，刚开发完项目，执行了下列命令：git add .

git commit -m '提交的备注信息'

git push -u origin dev

想将dev分支合并到master分支，操作如下：

1、首先切换到master分支上

git checkout master

2、如果是多人开发的话 需要把远程master上的代码pull下来

git pull origin master

//如果是自己一个开发就没有必要了，为了保险期间还是pull

3、然后我们把dev分支的代码合并到master上

git merge dev

4、然后查看状态及执行提交命令

git status

On branch master

Your branch is ahead of 'origin/master' by 12 commits.

(use "git push" to publish your local commits)

nothing to commit, working tree clean

//上面的意思就是你有12个commit，需要push到远程master上

> 最后执行下面提交命令

git push origin master

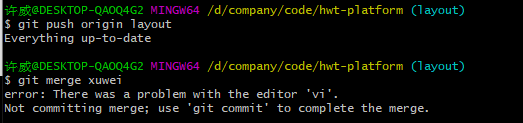
## 冲突

git merge –abort 保存本地

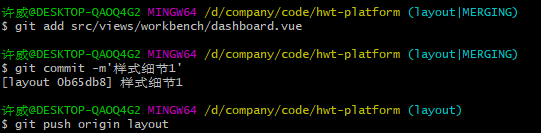
git reset –merge

<https://www.cnblogs.com/qinbb/p/5972308.html>

如果有冲突先



1. 先查看代码冲突的文件，然后提交git add src/views/workbench/dashboard.vue



## 回退

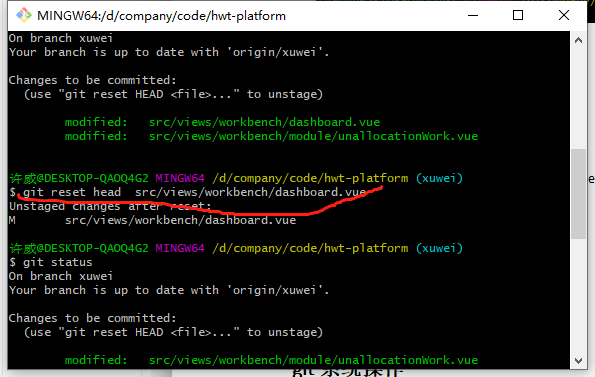
1、Git log (查看版本

2、git reset --hard I4e1fe8ace2036cd99baadfc9366f5a3e9c76f5eb (commit ID)

3. 推送到远程服务器：git push -f -u origin master  
4. 重新拉代码：git pull

Add错内容回退

Git reset head (文件)



# git相关命令

### 暂存

1. git 代码暂存指令：git stash（不推荐使用）
2. git 代码暂存指令：git stash push -m’’

2、git 代码暂存列表信息：git stash list

3、git 代码应用暂存代码：git stash apply stash@{1}

4、git 代码清空暂存：git stash clear

5、git stash pop 恢复之前缓存的工作目录。这个指令将缓存堆栈中的第一个stash删除并将对应修改应用到当前工作目录下

6、git stash drop 命令，后面可以跟stash名字移除某一项的暂存

7、git stash -u 可以stash untracked文件。

8、git stash -a 可以暂存当前目录下的所有修改

## git系统操作

gitlab-ctl start  //启动gitlab服务

gitlab-ctl status  //查看gitlab状态

gitlab-ctl stop  //停止gitlab服务

gitlab-ctl restart  //重启gitlab服务

gitlab-ctl reconfigure  //让配置生效

作者：250ZH  
链接：https://www.jianshu.com/p/4ef7aa6677eb  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 分支操作

查看本地分支：$ git branch

查看远程分支：$ git branch -r

创建本地分支：$ git branch [name] ----注意新分支创建后不会自动切换为当前分支

切换分支：$ git checkout [name]

创建新分支并立即切换到新分支：$ git checkout -b [name]

删除分支：

1 先切换到别的分支: git checkout dev20180927

2 删除本地分支： git branch -d dev20181018

3 如果删除不了可以强制删除，git branch -D dev20181018

4 有必要的情况下，删除远程分支**(慎用)**：git push origin --delete dev20181018

我从master分支创建了一个issue5560分支，做了一些修改后，使用git push origin master提交，但是显示的结果却是'Everything up-to-date'，发生问题的原因是git push origin master 在没有track远程分支的本地分支中默认提交的master分支，因为master分支默认指向了origin master 分支，这里要使用git push origin issue5560：master 就可以把issue5560推送到远程的master分支了。 如果想把本地的某个分支test提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支，那么可以这么做。

$ git push origin test:master // 提交本地test分支作为远程的master分支 //好像只写这一句，远程的github就会自动创建一个test分支

$ git push origin test:test // 提交本地test分支作为远程的test分支

## 版本相关操作

查看版本：$ git tag

创建版本：$ git tag [name]

删除版本：$ git tag -d [name]

查看远程版本：$ git tag -r

创建远程版本(本地版本push到远程)：$ git push origin [name]

删除远程版本：$ git push origin :refs/tags/[name

## 与远程仓库相关操作

检出仓库：$ git clone git://github.com/jquery/jquery.git

查看远程仓库：$ git remote -v

添加远程仓库：$ git remote add [name] [url]

删除远程仓库：$ git remote rm [name]

修改远程仓库：$ git remote set-url --push[name][newUrl]

拉取远程仓库：$ git pull [remoteName] [localBranchName]

推送远程仓库：$ git push [remoteName] [localBranchName]

# Git多人协助规范

## Git使用规范

团队开发中，遵循一个合理、清晰的 Git 使用流程，是非常重要的。

否则，各种不清晰的分支结构，后续产品迭代或维护都会让人很头疼，再如果每个程序员都提交一堆杂乱无章的commit，后续的快速查找定位问题只能通过阅读代码，也是很低效的。

### 分支规范

主分支：master、develop、release

功能分支：hotfix、feature(根据项目实际需要新增)

#### 分支类型详细说明

| **分支** | **名称** | **用途** |
| --- | --- | --- |
| master | 主分支 | 稳定版本 |
| release | 发布分支 | 发布新版本 |
| develop | 开发分支 | 最新版本 |
| hotfix | 热修复分支 | 修复线上 Bug |
| feature | 特性（功能）分支 | 实现新特性（功能） |

##### master 分支

master 为主分支，用于部署到正式环境（PRO），一般由 release 或 hotfix 分支合并，任何情况下不允许直接在 master 分支上修改代码。

仅master分支承担发布任务，每次发布完成后追加tag。

##### release 分支

release 为预上线分支，用于部署到预上线环境（PRE），始终保持与 master 分支一致，一般由 develop 或 hotfix 分支合并，不建议直接在 release 分支上直接修改代码。

##### develop 分支

develop 为测试分支，用于部署到测试环境（DEV），始终保持最新完成以及 bug 修复后的代码，可根据需求大小程度确定是由 feature

一定是满足测试的代码才能往上面合并或提交。

##### hotfix 分支

hotfix 为紧急修复分支，命名规则为hotfix-功能关键字， hotfix-i18n。

当线上出现紧急问题需要马上修复时，需要基于 release 或 master 分支创建 hotfix 分支，修复完成后，再合并到 release 或 develop 分支，一旦修复上线，便将其删除。

##### feature 分支

feature 为需求开发分支，命名规则为 feature-功能关键字 ，例如feature-i18n，一旦该需求上线，便将其删除。

这么说可能有点不容易理解，接下来举几个开发场景。

## Commit 日志规范

提交信息一定要认真填写！

建议参考规范：<type>(scope)：<subject>

比如：fix(首页模块)：修复弹窗 JS Bug。

type 表示 动作类型，可分为：

* feat: 新的功能
* fix: 修复 bug
* docs: 只是文档的更改
* style: 不影响代码含义的更改 (例如空格、格式化、少了分号)
* refactor: 重构功能或方法：既不是修复 bug 也不是添加新功能的代码更改
* perf: 提高性能的代码更改
* test: 添加或修正测试
* chore: 对构建或者辅助工具的更改，例如生成文档

scope 表示 影响范围，可分为：模块、类库、方法等。

subject 表示 简短描述，最好不要超过 60 个字，如果有相关 Bug 的 Jira 号，建议在描述中加上。

参考链接：

https://juejin.im/post/6844904069652299784

https://juejin.im/post/6844904111414968333

https://juejin.im/post/6844903945245048846

## 分支规范

#### 远程分支

master、pre、dev、feature、hotfix

#### 本地分支

* 必存分支：master、pre、dev、feature、hotfix
* 临时分支：feature-功能关键字（n个）、hotfix-功能关键字（n个）

#### feature、hotfix必存分支与临时分支的关联

1. 维护上：必存分支需要一直维护，而临时分支对应的功能代码上线后，便可删除；
2. 作用上：feature（hotfix)必存分支是feature（hotfix)临时分支的汇总分支；
3. 命名上：必存分支命令固定，临时分支命令根据功能模块关键字命令；
4. 数量上：必存分支1个，临时分支n个。

##### feature必存分支

开发者功能总分支，用来合并开发者负责的feature临时分支。以开发者名字命令，例如：liaosp

##### feature临时分支

功能分支，按照功能模块名在本地创建/维护，存在n个。都是从最新的master分支checkout出来。各功能分支相互独立，代码互不融合，功能上线后，可删除。

命名规则为feature-功能关键字： feature-i18n。

本地功能分支可以创建对应的远程分支，用来备份。功能上线，需要删除本地/线上的远程功能分支

##### hotfix必存分支

紧急修复分支，修改上线后的紧急bug。用来合并各模块的hotfix临时分支。

##### hotfix临时分支

紧急修复分支，按照功能模块名在本地创建/维护，存在n个。都是从最新的master分支checkout出来。各分支相互独立，代码互不融合，功能上线后，可删除。

命名规则为hotfix-功能关键字： hotfix-i18n

## 分支操作流程

### 新版本上线流程

1. **新功能开发前**：所有远程分支、本地分支更新为线上(master分支）最新代码；
2. **新功能开发**：统一从远程master分支，checkout一个新feature临时分支。（命名规则为feature-功能关键字）

执行命令：git checkout feature-key origin/masetr；

1. **功能开发完**：将本地feature临时分支合并到feature必存分支，再推送到feature远程分支（目前每个开发者都有一个远程feature分支，是以开发者名字命令）；
2. **功能测试**：远程feature分支合并到远程dev分支（需要线上请求Merge Requests），代码部署到测试环境；
3. **测试环境BUG修改**：在本地feature临时分支修复后，重复步骤3
4. **上预发环境**：管理员人将远程dev分支合并远程pre分支；
5. **预发布环境BUG修改**：在本地feature临时分支修复后，重复步骤3
6. **上正式环境**：管理员人将远程pre分支推到远程master分支；
7. **补丁版上线后**：所有远程分支、本地分支更新为线上(master分支）最新代码

### 补丁版上线流程

1. **功能开发前**：远程分支、本地分支更新为线上(master分支）最新代码
2. **功能开发**：统一从远程master分支，checkout一个新hotfix临时分支。（命令规则为hotfix-功能关键字）
3. **功能开发完**：将本地hotfix临时分支合并到hotfix必存分支，再push到hotfix远程分支(Only one)
4. **功能测试**：确认测试环境是否有正在测试的功能，如果没有，直接打包部署至测试环境进行测试，如果有，hotfix远程分支合并到pre远程分支，打包至预发布环境测试。
5. **BUG修复**：在本地hotfix临时分支修复后，重复步骤3
6. **上预发环境**：管理员人将远程hotfix分支合并远程pre分支；
7. **上正式环境**：管理员人将远程pre分支推到远程master分支；
8. **补丁版上线后**：所有远程分支、本地分支更新为线上(master分支）最新代码

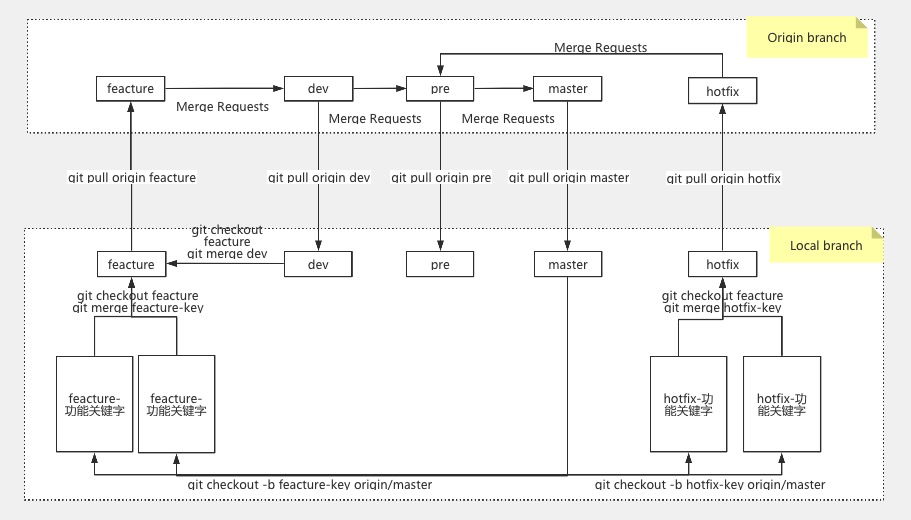
#### 注意事项

1. 新功能开发前，所有远程分支、本地分支更新为线上(master分支）最新代码。
2. 所有feature、hotfix分支都从线上master分支checkout出来，不要从远程dev切，确保本地各模块开发分支绝对纯净，不掺合其他未上线、无关联的分支代码
3. 测试环境、预发布环境、正式环境的BUG修改，都修改回到各模块功能分支修改后，再一步步合并到dev、pre、master，进行功能测试
4. 本地功能分支可以创建对应的远程分支，用来备份。功能上线，需要删除本地/线上的远程功能分支。本地功能分支及时推送到远程功能分支，确保每天下班前都将代码push到远程分支。防止本地电脑故障，代码丢失。
5. 如果两个开发者之间某一功能存在关联，某一功能需使用同一分支进行管理。
6. 碰到功能临时不上的情况。重新合并需要上线功能分支，合并到dev->pre->master。

操作步骤：

* + （首次）开发者删除并重新创建本地开发者分支（从远程master切 git checkout -b liaosp origin/master）, 本地开发者分支重新合并待上线的功能分支，并强制推送到远程个人分支(git push origin liaosp -f)；首次开发者不需要merge request。
  + （首次）负责人在本地拉取（包含待上线功能的）开发者远程分支，在本地解决冲突（有冲突需要和开发员确认）。强行推送到远程dev分支。
  + (非首次) 如果修改本地功能分支代码需再次合并到 远程Dev分支，操作步骤：本地功能分支 -> 本地开发者分支（合并本地功能分支，远程dev分支） -> 远程开发者分支 -> 远程dev分支。

## git多人协助流程图

<https://www.processon.com/view/link/5a1e2a4de4b04f355d323c71>

## 问题讨论