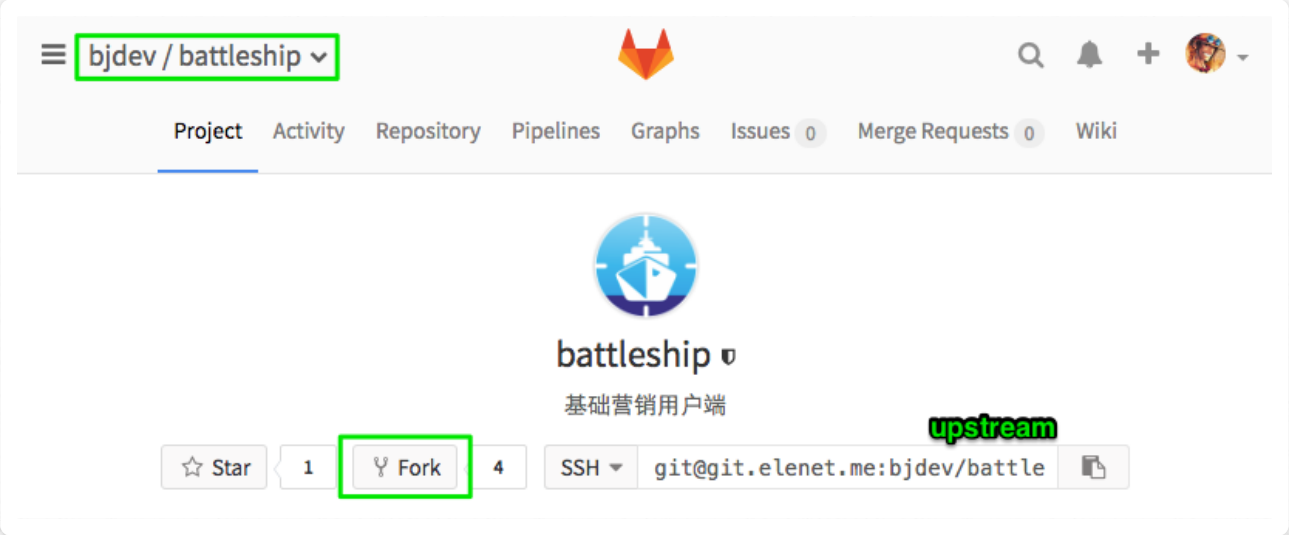
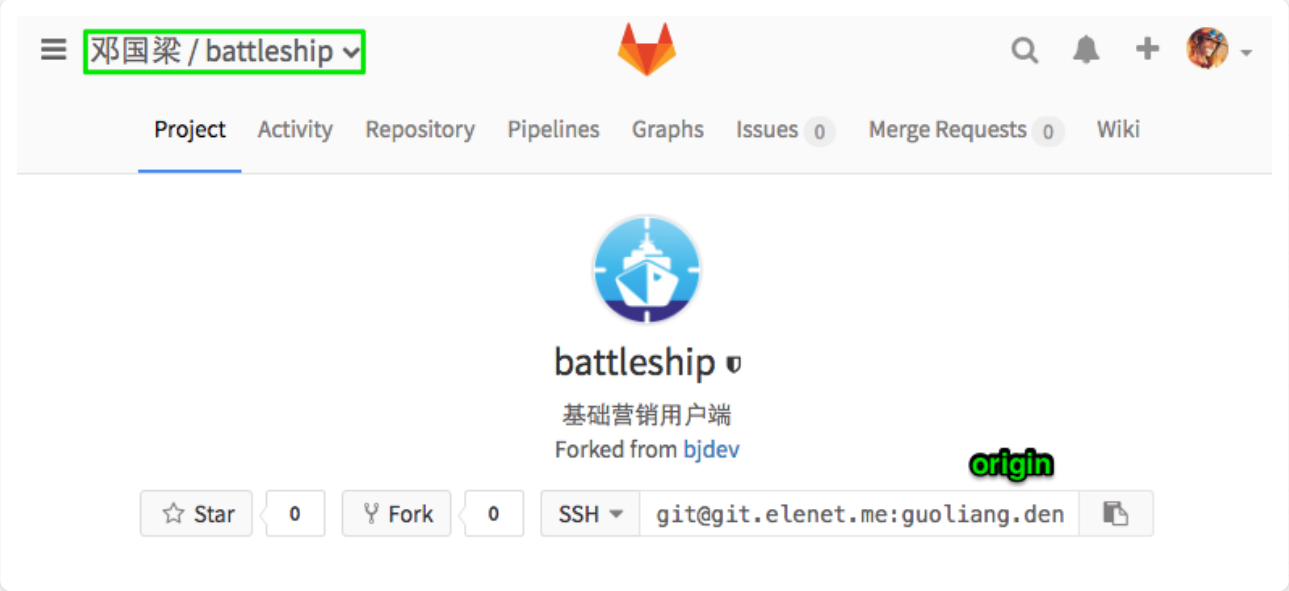
Git Workflow & Code Review Guide

fork 之后再进行开发





推荐阅读：https://git.elenet.me/help/workflow/forking_workflow.md

SSH keys

方便push代码不用每次都输入密码

生成 SSH key

在终端键入：

```
cat ~/.ssh/id_rsa.pub
```

如果**有输出**，说明之前生成过，直接copy输出内容。如果没有请使用如下命令生成：

```
ssh-keygen -t rsa -C "your_name@ele.me"
```

一路回车 (这里有详细说明：https://git.elenet.me/help/ssh/README)

添加 SSH key

访问： https://git.elenet.me/profile/keys

讲上面生成的 `id_rsa` 内容粘贴到 `Key` 文本框中, `Title` 随意

添加 git remote

remote是指托管在网络上的项目仓库，可能会有好多个，其中有些你只能读，另外有些可以写。同他人协作开发某个项目时，需要管理这些远程仓库，以便推送或拉取数据，分享各自的工作进展

```
git clone git@git.elenet.me:guoliang.deng/battleship.git
```

clone 自己的 fork，git 会自动添加一个名为 `origin` 的remote， 使用如下命令可查看已有的remote：

```
git remote -v
```

```
Robin@localhost ~/Code/battleship master git remote -v
origin  git@git.elenet.me:guoliang.deng/battleship.git (fetch)
origin  git@git.elenet.me:guoliang.deng/battleship.git (push)
upstream        git@git.elenet.me:bjdev/battleship.git (fetch)
upstream        git@git.elenet.me:bjdev/battleship.git (push)
```

需要再添加一个 `upstream remote` 来方便跟上线上版本

```
git remote add upstream git@git.elenet.me:bjdev/battleship.git
```

再次使用 `git remote -v` 查看

```
 Robin@localhost  ~/Code/battleship  master  git remote -v
origin  git@git.elenet.me:guoliang.deng/battleship.git (fetch)
origin  git@git.elenet.me:guoliang.deng/battleship.git (push)
upstream        git@git.elenet.me:bjdev/battleship.git (fetch)
upstream        git@git.elenet.me:bjdev/battleship.git (push)
```

`origin` 作为自己的fork进行日常开发工作
`upstream` 作为'上游'用来同步最稳定的生产环境代码

保证 upstream master 随时是可以发布的稳定版本

由一下两点保证:
1. 不能向 upstream master 直接push代码 （gitlab 里设置保护分支）
2. 每次将merge request合入master时确保该mr是测试通过的 （mr的评论里需要有测试人员的 `test pass` 回复）

每次开发之前，发起PR之前都要先 merge upstream master

每次开发之前用如下命令来跟上上游，防止遗漏 commit:

```
git pull upstream master
```

如何发起 merge request

1. 少量，频繁
2. 提供有 用的上下文（commit）
3. 主动请求

* Code Review 都 review 什么?

1. 限制风险：typo，低级错误，边界判断，逻辑漏洞...
2. 提高代码质量
3. 增加互动与交流
4. 熟悉项目

git 使用技巧

添加 alias

编辑 `~/.gitconfig` 来进行全局git配置，添加/修改如下内容：

```
[user]
  name = Your Name
  email = your name@ele.me
[alias]
  co = checkout
  ci = commit
  st = status
  br = branch
  lg = log --all --color --graph --pretty=format:'%Cred%h%Creset -%C(ye
llow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-comm
it
[color]
  ui = true
```

alias 可以简化git命令，如

```
git ci 代替 git commit
git st 代替 git status
```

分支的使用

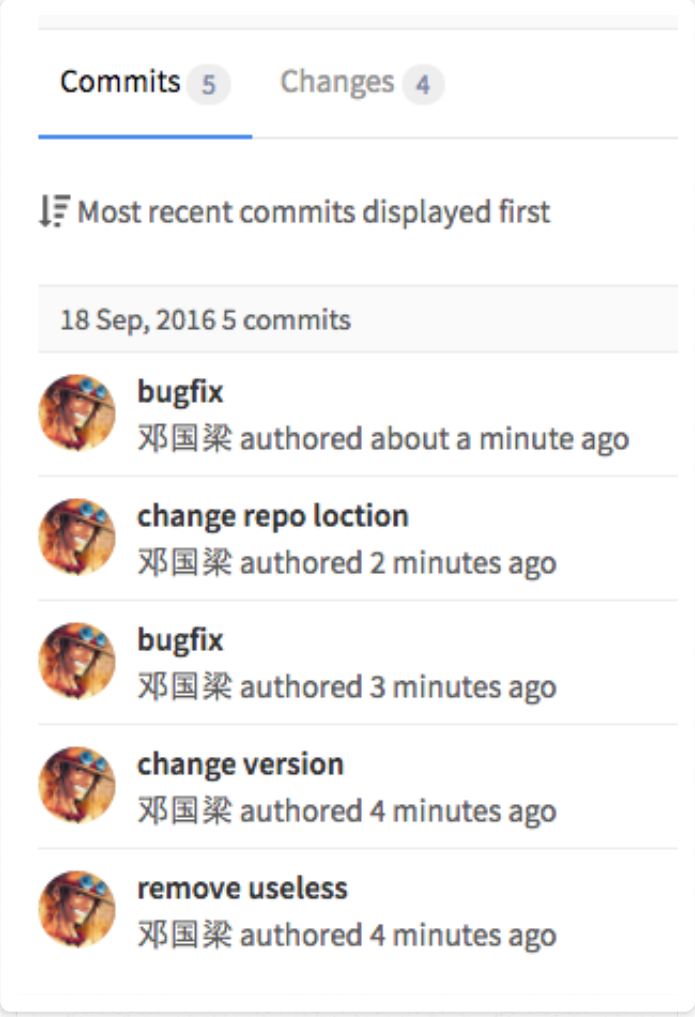
```
git co -b 'branch_name'
```

1. 分支命名：使用当前PRD版本号（如：v2.3.0），如果是hotfix或者CR，没有对应的PRD版本，则使用当前进行PRD版本号的末尾（如：v2.3.1）
2. 可以同时多开几个分支进行不同功能的修改，但不相关的修改需要在不同的分支处理，不能放在同一分支
3. 分支创建都应该基于 `upstream master`

4. 推荐阅读： http://iissnan.com/progit/html/zh/ch3_1.html

git rebase

rebase 目前主要用来修剪commit，删除无用的commit message，或者合并多个。使得你的commit变的干净，便于review。例如，多次修改代码并commit之后的结果：



有很多类似 `bugfix` 无意义的commit信息，我们需要通过rebase来修剪掉

```
git rebase upstream/master -i
```

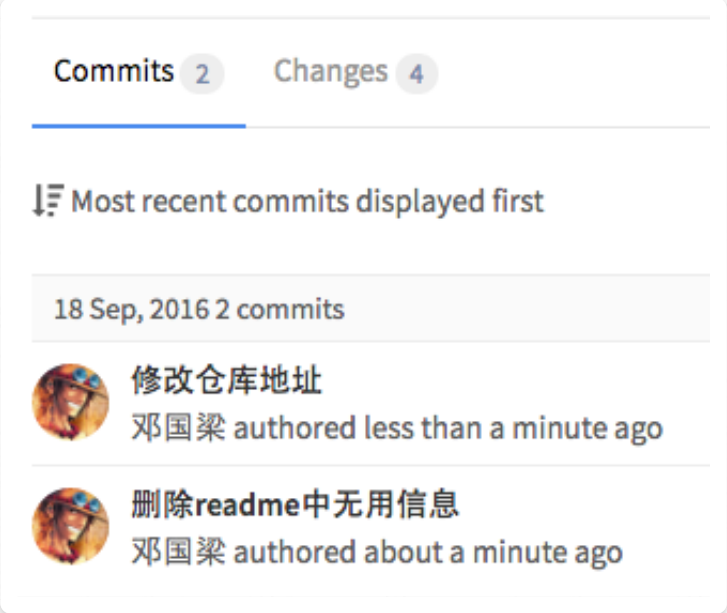
`-i` 使用交互界面进行rebase，这样可以进一步操作

```
pick 7a50b77 remove useless
s 0bce016 change version
s 684f5c0 bugfix
pick afd4f78 change repo loction
s 5d5e477 bugfix
```

修改完成后保存并退出，使用如下命令再次推送到远端，并发起merge request:

```
git push origin branch_name -f
```

必须使用 `-f` 参数来强制改写commit记录



git stash

临时存储你不想commit的工作状态:

```
# 保存工作状态
git stash

# 查看已暂存的工作状态列表
git stash list

# 拿出暂存
git stash apply

# 拿出指定暂存
git stash apply stash@{2}

# 拿出最近暂存，并将其从暂存列表中移除
git stash pop
```

推荐阅读：http://iissnan.com/progit/html/zh/ch6_3.html