码云平台帮助文档 V1.2

码云使用手册

1、什么是码云

1.1、什么是码云（Gitee.com）

1、什么是码云（Gitee.com）

码云（Gitee.com）是专为开发者提供的稳定、高效、安全的云端软件开发协作平台。

无论是个人、团队、或是企业，都能够用码云实现代码托管、项目管理、协作开发。

码云于2013年正式推出，由开源中国基于 Gitlab 所开发，我们在 Gitlab 的基础上做了大量的改进和定制开发，致力于为国内开发者提供优质稳定的托管服务。目前已成为国内最大的代码托管系统。

2、码云主要功能

码云除了提供最基础的 Git 代码托管之外，还提供代码在线查看、历史版本查看、Fork、Pull Request、打包下载任意版本、Issue、Wiki 、保护分支、代码质量检测、PaaS 项目演示等方便管理、开发、协作、共享的功能。

3、码云协作模式

对于一个开源项目，从开发角度讲大体上分为两类人群，一类称为核心开发团队，他们可以向保存源代码的版本库提交，即对源代码的修改具有最终的决定权。另外一类称为贡献者，他们不属于核心开发团队，虽然也能看到源代码，但无权向版本库提交修改。

采用传统的集中式版本控制系统（如 SVN ）的开源项目，这两个群体的用户体验都不是太好。如图1-1所示，项目的贡献者（非核心成员）很不“高兴”，因为他们即便有修改源代码的能力和渴望，也不能直接向版本库提交，要想成为提交者需要一个很长的建立信任的过程。然而即便是核心开发团队的成员，体验也不是太好，因为凡是涉及到版本库的操作（检入、检出、查看日志等）都需要在联网的状态下进行，网络带宽对用户体验影响相当大。

图片地址：https://static.oschina.net/uploads/img/201705/18181722_ePPY.jpg

图1-1：使用集中式版本控制系统

Git 等分布式版本控制系统的出现，彻底颠覆了原有代码管理的组织模式。使用 Git，不再依赖唯一的、集中式的版本库，而是每个开发者本地都拥有一份完整的版本库。Git 并不排斥集中式的使用模式，但更倾向于将集中式版本库称为共享版本库。核心开发团队的成员和贡献者（非核心成员）都可以从共享版本库克隆一份本地版本库，但只有核心团队成员才可以将自己本地版本库的提交

推送到共享版本库上。如图1-2所示。

图片地址：https://static.oschina.net/uploads/img/201705/18181743_LKNF.jpg

图1-2：使用分布式版本控制系统

使用 Git 做版本控制（如图1-2所示），核心开发团队非常“高兴”，因为他们和共享版本库之间不必一直保持连接状态，诸如查看日志、提交、创建分支等几乎全部操作都（脱离网络）在本地的版本库中完成。项目贡献者（非核心成员）也不再那么沮丧，因为版本库人人皆可更改（当然是对本地版本库而言）。稍微让贡献者感到困难的就是如何将自己对项目的改进被核心开发团队所了解并接纳。Git 提供了多种途径，一个方法是先用 git format-patch 命令将本地提交转换为补丁文件或补丁文件序列，再通过邮件发送给核心开发团队。另外一个办法就是搭建一个自己专有的共享版本库，通过邮件创建一个拉拽请求（Pull Request），让核心团队的开发者到自己的版本库来抓取（Pull）。

图片地址：https://static.oschina.net/uploads/img/201705/18181806_8HN7.jpg

图1-3：码云的协同模式

使用码云，无论是项目的核心开发团队，还是普通的项目贡献者都工作得非常“愉快”。创建项目变得非常轻松，创建者只需在码云上点击一下鼠标即可创建一个新版本库，通过简单的 Web 操作即可完成项目授权进而组建项目核心团队。在码云中，非核心团队成员参与项目也很容易。先找到自己希望参与的项目，然后只需在 Web 上点击一下鼠标即可在自己的托管空间下创建一个派生（fork）的项目，并对派生项目的版本库具有读写的完全权限，就好像这个项目原本就是由自己创立的那样。当贡献者完成开发并向自己派生的版本库推送后，可以通过码云的 Web 界面向项目的核心开发团队发送一个 Pull Request，请求审核。项目的核心团队收到 Pull Request 后审核代码，审核通过后可以直接通过 Web 界面执行合并操作接纳贡献者的提交。

1.2、码云亮点

图片地址：https://static.oschina.net/uploads/img/201704/01123654_EHvc.jpg

1.3、探索码云

打开浏览器，访问网址：https://gitee.com。

图片地址：https://static.oschina.net/uploads/img/201707/17191633_7H8o.png

图1-4：码云的首页

作为一个新手，如果想要了解码云上的热门开源项目，那么可以从导航条中的【发现】开始。图1-5显示通过对开源项目数据的分析得到的趋势。

图片地址：https://static.oschina.net/uploads/img/201704/01141935_bVhT.jpg

图1-5：开源项目动态趋势

图1-6显示了码云托管项目编程语言的动态分布，可以帮助程序员在找工作时选择更热门的编程语言，实时了解行业信息。：-)

图片地址：https://static.oschina.net/uploads/img/201704/01142502_1paf.jpg

---

导航栏中【企业服务】，是码云专门为公司提供自托管的代码存取及版本管理服务。**码云企业版**包括了码云免费版的所有主要功能，包括代码浏览、问题追踪、Wiki、质量检测、项目演示、团队协作等，此外，还提供了更强大的安全告警功能以及详细的企业资源、人员贡献、项目进度等的统计功能。

图片地址：https://static.oschina.net/uploads/img/201707/17191202_jrPc.png

码云企业版为了满足不同团队规模的需求，灵活设置了不同的版本，甚至还提供了小额度的企业免费版。

图片地址：https://static.oschina.net/uploads/img/201704/01145325_xFoC.jpg

1.4、码云平台IDEA系列的插件使用

1. 插件安装

方法一

1. 启动 idea，选择 Configure - Plugins

图片地址：https://static.oschina.net/uploads/img/201606/02115644_sY06.png

2. 选择 Browse repositories...

图片地址：https://static.oschina.net/uploads/img/201606/02115659_OIXq.png

3. 搜索 gitosc，安装插件

图片地址：https://static.oschina.net/uploads/img/201606/02115715_OXDe.png

4. 重启 idea 即可

图片地址：https://static.oschina.net/uploads/img/201606/02115732_fb4t.png

方法二

1. 选择 File - Settings

图片地址：https://static.oschina.net/uploads/img/201606/02122358_Ry94.png

2. 选择 Plugins。 其他步骤同上

图片地址：https://static.oschina.net/uploads/img/201606/02122340_VpW7.png

2. 登陆并拉取项目

方法一

1. 启动 idea，选择 Check out from Version Control - 码云

图片地址：https://static.oschina.net/uploads/img/201606/02120258_WPKq.png

2. 输入用户名和密码，登陆

图片地址：https://static.oschina.net/uploads/img/201606/02120306_MS8H.png

3. 点击选框中的向下箭头，会显示当前用户在码云上的所有项目

图片地址：https://static.oschina.net/uploads/img/201606/02120321_H6TO.png

4. 可选择任意项目进行 clone

图片地址：https://static.oschina.net/uploads/img/201606/02120334_8SUv.png

方法二

登陆

1. 选择 File - settings

图片地址：https://static.oschina.net/uploads/img/201606/02123545_hTwI.png

2. 选择 Version Control - 码云

图片地址：https://static.oschina.net/uploads/img/201606/02122846_XuM9.png

如果配置了公钥，可选择 ssh 方式拉取项目。

Test 按钮可测试用户名和密码是否正确。

拉取项目代码

1. 选择 VCS - Checkout from Version Control - 码云

图片地址：https://static.oschina.net/uploads/img/201606/02123042_mnaF.png

2. 其他步骤同上

图片地址：https://static.oschina.net/uploads/img/201606/02123754_WLFj.png

3. push 代码

1. 添加文件

右键点击需要添加的文件、文件夹

图片地址：https://static.oschina.net/uploads/img/201606/02125128_1yZy.png

2. 选择commit

右键点击文件、文件夹

图片地址：https://static.oschina.net/uploads/img/201606/02125200_Mylj.png

3. 填写 commit 信息

图片地址：https://static.oschina.net/uploads/img/201606/02125215_mCBO.png

4. 选择 push

图片地址：https://static.oschina.net/uploads/img/201606/02125233_YqtF.png

5. 将代码 push 到线上

图片地址：https://static.oschina.net/uploads/img/201606/02125248_3yo9.png

4. 托管本地项目到码云

本地项目可以直接托管到码云，不需要在 web 上创建项目。

1. 选择 VCS - Import into Version Control - 托管项目到码云

图片地址：https://static.oschina.net/uploads/img/201606/02121544_480H.png

2. 填写项目信息，可选择公有或私有

图片地址：https://static.oschina.net/uploads/img/201606/02121603_Od6p.png

3. 选择文件，填写 commit 信息。点击OK即可

图片地址：https://static.oschina.net/uploads/img/201606/02121631_xmdK.png

1.5、常用工具及插件教程

- eclipse中egit插件使用-图文并茂-详细(http://my.oschina.net/songxinqiang/blog/192567)
- Visual Studio 2012连接到osc@git(http://my.oschina.net/gal/blog/141442)
- TortoiseGit配合msysGit在Git@OSC代码托管的傻瓜教程 (http://my.oschina.net/icelily/blog/141342)
- 利用eclipse的git插件EGit与git@osc交互(http://my.oschina.net/kzhou/blog/132146)
- Git初体验(http://my.oschina.net/dxqr/blog/134811)
- 在win7系统下使用TortoiseGit(乌龟git)简单操作 Git@OSC(http://my.oschina.net/longxuu/blog/141699)
- Xcode连接git @ osc(http://my.oschina.net/zxs/blog/142544)
- git@osc(git)中team开发、fork和pull request的用法 (http://my.oschina.net/kzhou/blog/150290)
- eclipse的git插件整合Git@OSC(http://my.oschina.net/u/861562/blog/151975)
- Eclipse使用EGit管理git@OSC项目(http://my.oschina.net/China2012/blog/174874)
- 如何导入外部Git仓库到中国源代码托管平台 （Git@OSC）(http://www.oschina.net/question/82993_133520)
- https 方式使用Git@OSC设置密码的方式(http://gitee.com/oschina/git-osc/issues/2586)

2、创建码云账号

2.1、创建码云账号

注册码云账号，只要点击导航条中的“注册”，或者点击首页中那个大大的“加入码云”按钮，即可进入注册页面。

收费吗？目前除企业版外均为免费。开源软件托管是码云的基础，对于开源项目的版本库（无论是公有版本库，还是私有版本库）的托管，码云都是免费的。在码云首页中，就有针对于个人用户的免费托管方案，如图2-1所示。

图片地址：https://static.oschina.net/uploads/img/201704/18155757_fiOb.jpg

图2-1：针对个人用户的免费方案

进入码云的注册界面，依次填入各项，需注意的是：邮箱最好填写国内的邮箱，以免因为众所周知的原因无法接收激活邮件，**个性地址一经选定,无法修改，请慎重填写。**

图片地址：https://static.oschina.net/uploads/img/201704/18171705_6AAU.jpg

图2-2：注册界面

我们会向您的邮箱发送一份激活邮件，请点击其中的链接激活账号，账号激活后,注册流程就算完成了。注册完毕即以新注册的账号登录，登录后即进入用户的控制面板页面。如果图2-3所示。

图片地址：https://static.oschina.net/uploads/img/201704/18172802_RGrN.jpg

图2-3：登录后的码云首页

控制面板页面是用户最重要的页面，因为在这个页面中不仅可以看到组织成员的动态，还可以看到自己的项目信息，代码片段，PR，Issues，以及加入的企业信息，非常方便。

在页面右上方显示当前登录用户的名称和头像。图2-3中显示登录用户为 “不要404”。在页面右上方还有两个图标，从左至右分别是：通知和创建。点击头像图标进入【修改资料】对账号进行进一步设置，如图2-4所示。

图片地址：https://static.oschina.net/uploads/img/201704/18190232_7R3K.jpg

图2-4：账户设置页

点击菜单中的【修改账户】，可以更改私人令牌、更换登陆密码、以及修改注册邮箱，如图2-5所示。

图片地址：https://static.oschina.net/uploads/img/201704/18181420_ix0Y.jpg

图2-5：账户管理

其中私人令牌是和用户密码相关的密钥，当用户密码更改时私人令牌也随之更改。码云的某些应用会使用私人令牌进行身份认证，从而避免直接使用用户密码造成泄露的风险。私人令牌若泄露的危害要远远小于密码泄露，这因为私人令牌不能用于登录码云网站等，而且一旦私人令牌泄露可以很容易通过更改密码的方式更换私人令牌。

码云只允许为一个账号绑定对应一个邮件地址，以便能够将 Git 版本库中的提交正确对应到码云账户。

码云为用户提供两种接受信息的方式：邮件和站内私信，如图2-6所以。

图片地址：https://static.oschina.net/uploads/img/201704/18185015_5LLF.jpg

图2-6：通知方式

码云为托管的 Git 版本库提供 SSH 协议支持，即用户可以用公钥认证的方式连接到码云的 SSH 服务器。点击 生成并部署SSH key(http://git.mydoc.io/?t=154712) 了解具体实现步骤。

码云作为一个优秀程序员的聚集地，期待着你的加入！！

3、代码托管

3.1、创建第一个项目

1、创建项目

点击右上角的
图片地址：https://static.oschina.net/uploads/img/201704/01152036_vITZ.jpg

或者
图片地址：https://static.oschina.net/uploads/img/201704/01152303_3MMo.jpg

，跳转至项目创建页面：

图片地址：https://static.oschina.net/uploads/img/201704/01152603_Zb3l.jpg

依次填写各项信息然后点击创建就可以了，这样我们一个项目就创建好了如果没有意外，你将看到这个页面:

图片地址：https://static.oschina.net/uploads/img/201704/01152726_Wux6.jpg

这样，我们在码云平台就创建好了一个项目

2、本地初始化一个项目

首先，你需要执行下面两条命令，作为 git 的基础配置，作用是告诉 git 你是谁，你输入的信息将出现在你创建的提交中。

```
git config --global user.name "你的名字或昵称"
git config --global user.email "你的邮箱"
```

然后在你的需要初始化版本库的文件夹中执行：

```
git init
git remote add origin <你的项目地址> //注:项目地址形式为
:https://gitee.com/xxx/xxx.git或者 git@gitee.com:xxx/xxx.git
```

这样就完成了一次版本你的初始化。

如果你想克隆一个项目，只需要执行：

```
git clone <项目地址>
```

3、完成第一次提交

进入你已经初始化好的或者克隆项目的目录,然后执行：

```
git pull origin master
<这里需要修改/添加文件，否则与原文件相比就没有变动>
git add .
git commit -m "第一次提交"
git push origin master
```

然后如果需要账号密码的话就输入账号密码，这样就完成了一次提交。

此时，你可以在你的个人面板、项目主页查看到你的提交记录，例如
：https://gitee.com/oschina/git-osc/commit/f3dd1c5bae48fa4244e2595a39e750e5606dd9be](https://gitee.com/oschina/git-osc/commit/f3dd1c5bae48fa4244e2595a39e750e5606dd9be

按照本文档新建的项目时，在码云平台仓库上已经存在 readme 文件，故在提交时可能会存在冲突，这时您需要选择的是保留线上的文件或者舍弃线上的文件，如果您舍弃线上的文件，则在推送时选择强制推送，强制推送需要执行下面的命令：

```
git push origin master -f
```

如果您选择保留线上的 readme 文件,则需要先执行：

```
git pull origin master
```

然后才可以推送,如果发生冲突，则需要先解决冲突,关于如何处理冲突，请参阅如何处理代码冲突(http://git.mydoc.io?v=16912&t=66902)这一小节。

3.2、公钥认证管理

开发者向码云版本库写入最常用到的协议是 SSH 协议，因为 SSH 协议使用公钥认证，可以实现无口令访问，而若使用 HTTPS 协议每次身份认证时都需要提供口令。使用 SSH 公钥认证，就涉及到公钥的管理。

1.如何生成ssh公钥

你可以按如下命令来生成sshkey:

```
ssh-keygen -t rsa -C "xxxxx@xxxxx.com"

# Generating public/private rsa key pair...
# 三次回车即可生成 ssh key
```

查看你的 public key，并把他添加到码云（Gitee.com） SSH key添加地址(http://git.oschina.net/profile/sshkeys)

```
cat ~/.ssh/id_rsa.pub
# ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6eNtGpNGwstc....
```

添加后，在终端（Terminal）中输入

```
ssh -T git@git.oschina.net
```

若返回

```
Welcome to Git@OSC, yourname!
```

则证明添加成功。

•

2.怎么添加用户ssh key?

---

1. 点击右上角的
图片地址：https://static.oschina.net/uploads/img/201604/14154017_ogkd.png

标志,进入个人中心,然后点击左侧的ssh公钥后在下图位置填写你的ssh公钥。

2. 点击确定,然后验证密码(即你的注册账号密码)就完成了ssh公钥添加。

图片地址：https://static.oschina.net/uploads/img/201610/18115822_miTO.png

3.项目的 ssh key 和用户的 ssh key 两处地方有什么不同?

> 项目的 ssh key 只针对项目,且我们仅对项目提供了部署公钥,即项目下的公钥仅能拉取项目,这通常用于生产服务器拉取仓库的代码。

而用户的 key 则是针对用户的,用户添加了 key 就对用户名下的项目和用户参加了的项目具有权限,一般而言,用户的 key 具有推送和拉取的权限,而项目的 key 则只具有拉取权限

3.3、建立主页

很多开源项目托管平台都支持为托管的项目建立主页，但主页的维护方式都没有码云这么酷。大多数托管平台无非是开放一个FTP或类似服务，用户把制作好的网页或脚本上传了事，而在码云用户通过创建特殊名称的 Git 版本库或在 Git 库中建立特别的分支实现对主页的维护。

码云 Pages 是一个免费的静态网页托管服务，您可以使用码云 Pages 托管博客、项目官网等静态网页。如果您使用过 Github Pages 那么您会很快上手使用码云的 Pages服务。

> Jekyll 是一个简单的博客形态的静态站点生产机器。它有一个模版目录，其中包含原始文本格式的文档，通过 Markdown （或者 Textile ）以及 Liquid 转化成一个完整的可发布的静态网站，你可以发布在任何你喜爱的服务器上。Jekyll 也可以运行在 码云 Pages 上，也就是说，你可以使用码云的服务来搭建你的项目页面、博客或者网站，而且是完全免费的。

Jekyll 使用文档：http://jekyll.bootcss.com/docs/home/

1. 使用前须知

1、项目必须有 index.html 才可以正常访问

2、静态网页的样式可以自己写，也可以拿一些静态模版来修改，Pages服务本身不提供任何形式的模版，但我们后续会陆续增加例子，供大家参考使用。

3、如果你想以根目录的形式访问自己的静态网站，只需要建立一个与自己个性地址同名的项目即可，如 https://gitee.com/ipvb 这个用户，想要创建一个自己的站点，但不想以子目录的方式访问，想以ipvb.oschina.io直接访问，那么他就可以创建一个名字为ipvb的项目 https://gitee.com/ipvb/ipvb 部署完成后，就可以以 http://ipvb.oschina.io 进行访问了。

2. 一个小白的 Pages 搭建之旅

A.新建项目 test_pages

图片地址：https://static.oschina.net/uploads/img/201606/01172928_Wv13.png

点击创建完成项目的创建

B.添加文件 index.html (注意名称是index.html哦！)

点击新建文件

图片地址：https://static.oschina.net/uploads/img/201606/01173033_nL18.png

文件名输入 index.html，内容就是简单的 html

图片地址：https://static.oschina.net/uploads/img/201606/01173322_qoin.png

点击提交，将文件提交到仓库

C.选择 pages 服务

图片地址：https://static.oschina.net/uploads/img/201606/01173449_hHY1.png

D.选择需要部署的分支，这里选择 Master 启动服务。

图片地址：https://static.oschina.net/uploads/img/201606/01173541_QDTs.png

E.访问生成的网站地址，即可以查看你部署的静态页面啦！

图片地址：https://static.oschina.net/uploads/img/201606/01173617_VOe9.png

图片地址：https://static.oschina.net/uploads/img/201606/01173639_s4t6.png

3. 已经有Pages项目如何部署到码云的Pages

以jQuery-File-Upload项目为例，项目地址：https://github.com/blueimp/jQuery-File-Upload

它在Github上的Pages地址是：https://blueimp.github.io/jQuery-File-Upload/

如果想把它转移到码云Pages，只需要登录你的码云账户，点击右上角的 + 号，选择新建项目

图片地址：https://static.oschina.net/uploads/img/201606/01170937_93rs.png

图片地址：https://static.oschina.net/uploads/img/201606/01170914_Vb4Q.png

然后点击创建，项目会在后台自动导入，导入成功后，点击菜单栏的 Pages

图片地址：https://static.oschina.net/uploads/img/201606/01171047_x3BN.png

图片地址：https://static.oschina.net/uploads/img/201606/01171113_lM7g.png

这里我们默认的Pages服务分支是osc-pages，但是你也已选择自己静态页面所在的分支，这里jQuery-File-Upload项目的静态页面分支是gh-pages，选择gh-pages并点击启动服务。

图片地址：https://static.oschina.net/uploads/img/201606/01171334_7QAP.png

至此，静态网页已经部署成功，访问提供的地址：http://silentboy.oschina.io/jQuery-File-Upload/ 即可查看到jQuery-File-Upload项目的静态官网。

图片地址：https://static.oschina.net/uploads/img/201606/01171535_B1u8.png

4. Pages 进阶，使用 Jekyll 生成自己的静态网页

Jekyll 究竟是什么？

Jekyll 是一个简单的博客形态的静态站点生产机器。它有一个模版目录，其中包含原始文本格式的文档，通过 Markdown （或者 Textile ）以及 Liquid 转化成一个完整的可发布的静态网站，你可以发布在任何你喜爱的服务器上。Jekyll 也可以运行在 码云Pages上，也就是说，你可以使用码云的服务来搭建你的项目页面、博客或者网站，而且是完全免费的。

Jekyll 使用文档：http://jekyll.bootcss.com/docs/home/

其他帮助:

如何自定义404？

答：在项目根目录新建 404.html 文件即可。

4、协同工作

4.1、Fork + Pull 模式

参与码云中的项目开发，最常用和推荐的首选方式是“Fork + Pull”模式。在“Fork + Pull”模式下，项目参与者不必向项目创建者申请提交权限，而是在自己的托管空间下建立项目的派生（Fork）。至于在派生项目中创建的提交，可以非常方便地利用码云的 Pull Request 工具向原始项目的维护者发送 Pull Request。

1、什么是 Pull Request？

Pull Request 是两个仓库提交变更的一种方式，通常用于 fork 项目与被 fork 项目的差异提交，同

时也是一种非常好的团队协作方式，下面，就来讲解如何在码云平台提交 Pull Request：

PS：码云平台限制 Pull Request 源项目与目标项目需存在 fork 与被 fork 关系，故如果你要提交 Pull Request，必须先 fork 一个项目，然后才能对该项目提交 Pull Request，同时，以该项目为父项目的所有项目，您也均可以提交 Pull Request。

2、如何 fork 项目

fork 项目时非常简单的，进到项目页面，然后找到右上角的 fork 按钮，点击后再次点击确定，等待系统在后台完成仓库克隆操作，你就完成了 fork 操作，如图：

图片地址：https://static.oschina.net/uploads/img/201607/11170038_5Rzf.png

3、如何提交 Pull Request

首先，您的项目与目标项目必须存在差异，这样才能提交,比如这样：

图片地址：https://static.oschina.net/uploads/img/201607/11164020_j8To.png

如果不存在差异，或者目标分支比你提Pull Request的分支还要新，则会得到这样的提示：

图片地址：https://static.oschina.net/uploads/img/201607/11164209_LfgW.png

然后，填入Pull Request的说明，点击提交Pull Request，就可以提交一个Pull Request了，就想下图所示的那样：

图片地址：https://static.oschina.net/uploads/img/201607/11171413_x7Jh.png

4、如何对已经存在的 Pull Request 的进行管理

首先，对于一个已经存在的 Pull Request，如果只是观察者，报告者等权限，那么访问将会受到限制，具体权限限制请参考码云平台关于角色权限的内容，下文涉及的部分，仅针对管理员权限，如果您发现不太一样的地方，请检查您的权限是不是管理员或该 Pull Request 的创建者。

5、如何修改一个已经存在的 Pull Request

点击 Pull Request 的详情界面右上角的编辑按钮，就会弹出编辑框，在编辑框中修改你需要修改的信息，然后点击保存即可修改该 Pull Request，如下图所示：

图片地址：https://static.oschina.net/uploads/img/201607/11182105_veIB.png

请注意，在该界面，可以对 Pull Request 进行指派负责人，指派测试者等等操作，每一个操作均会通知对应的人员

6、对 Pull Request 的 bug 修改如何提交到该 Pull Request 中

对于 Pull Request 中的 bug 修复或者任何更新动作，均不必要提交新的 Pull Request，仅仅只需要推送到您提交 Pull Request 的分支上，稍后我们后台会自动更新这些提交，将其加入到这个 Pull Request 中去

7、Pull Request 不能自动合并该如何处理

在提交完 Pull Request 的后，在这个 Pull Request 处理期间，由原本的能自动合并变成不能自动合并，这是一件非常正常的事情，那么，这时，我们有两种选择，一种，继续合并到目标，然后手动处理冲突部分，另一种则是先处理冲突，使得该 Pull Request 处于可以自动合并状态，然后采用自动合并，一般来讲，我们官方推荐第二种，即先处理冲突，然后再合并。具体操作为：

先在本地切换到提交 Pull Request 的分支，然后拉取目标分支到本地，这时，会发生冲突，参考如何处理代码冲突(http://git.mydoc.io?v=16912&t=83148) 这一小节将冲突处理完毕，然后提交到 Pull Request 所在的分支，等待系统后台完成Pull Request的更新后，Pull Request 就变成了可自动合并状态

8、Pull Request 不小心合并了，可否回退

对于错误合并的 Pull Request，我们提供了回退功能，该功能会产生一个回退 XXX 的 Pull Request，接受该 Pull Request 即可完成回退动作，注意，回退本质上是提交一个完全相反的 Pull Request，所以，你仍然需要进行测试来保证完整性，另，为了不破坏其他 Pull Request，建议只有需回退的 Pull Request 处于最后一次合并操作且往上再无提交时执行回退动作，否则请手动处理。

4.2、组织和团队

首先，在码云平台，组织功能设计上是为了满足大型开发团队的需要，是一堆人员与项目的集合。成员与项目中间并没有直接的权限关系，故组织成员不可以直接管理项目，不对项目具有直接的管理权限，所以，要想让组织成员能访问、管理您组织的中项目，您需要将该成员添加到项目成员中。具体的您可以这样做：

首先点击您的组织中的那个项目，如图：

图片地址：https://static.oschina.net/uploads/img/201605/18111502_0YFX.png

然后点击管理，如图：

图片地址：https://static.oschina.net/uploads/img/201605/18111538_aZH5.png

点击下图所示的任意一个条目：

图片地址：https://static.oschina.net/uploads/img/201605/18111704_RbIm.png

然后点击从组织中添加成员

图片地址：https://static.oschina.net/uploads/img/201605/18111746_7KXj.png

你就能看到下图类似的界面：

图片地址：https://static.oschina.net/uploads/img/201605/18112206_YcFu.png

然后选择好成员，选择好权限，点击添加用户，该用户就可以访问或管理这个组织中的项目了。

【注】：组织拥有者可以创建任意多的团队（Team）即角色，对属于组织的用户进行管理。组织拥有者就是组织中权限最高的角色。组织和用户一样可以创建项目，但是组织没有 SSH 公钥配置，也不能以组织的身份操作版本库。

码云特性

码云SVN支持

现在 **码云** 目前支持使用 Subversion 对仓库进行操作，以下是使用指南和注意事项。

使用前注意

1. 仓库体积超过 300 MB 不建议使用 Subversion 操作仓库，存储库容量达到 400 MB，或者 300 MB 并且存储大量非文本数据时，我们将关闭仓库的 Subversion 支持。

2. 由于 GIT 不支持空目录的提交，在存储机器上，无论是普通仓库还是开启 Subversion 接入的仓库存储时都是 GIT 仓库，Subversion 的 commit 是提交到 git 仓库上的，所以码云的 Subversion 不支持空目录的提交。

3. 第一次开启 Subversion， 操作一个仓库，如果仓库体积较大或者提交次数较多，由于缓存的缘故，响应时间会比较长。

4. 不支持 Subversion 的 Hook 机制，请使用 WebHook 替代。

5. Subversion 属性不完全支持。

6. 客户端需要开启 SASL 支持，不支持的客户端无法访问。

7. 部分 svn 命令不支持。可以查看 **Subversion 客户端的兼容性。**

8. 版本号的映射，目前 Subversion 的版本号计算依据为本分支所有的commit 数目减一 不包括 merge ，如果使用了在 git 中强制回退等操作，请重新检出。

WARNING:

> 由于 git 在设计上就没有考虑空文件 Kernel.org: Git FAQ(https://git.wiki.kernel.org/index.php/GitFaq#Can_I_add_empty_directories.3F)
>
> 我们设计的原则就是不破坏，不主动修改用户的仓库，我们的后端存储的完全是一个 git 仓库，如果我们添加了，一次提交内容也不会一致了，建议你在添加目录的时候添加 .keep 之类的占位文件，空文件即可。
>
> Git 与 SVN 混用时尽量不要使用 Git 强制推送。Git 与 SVN 混用注意事项 (https://git.mydoc.io/?t=122635)

关于改版

Subversion 功能的最终解释权归 OSChina.NET 所有。Subversion 接入的规则可能在下一次改版中发生改变。

开启方式

1. 在项目的设置界面开启

图片地址：https://static.oschina.net/uploads/img/201701/17151858_hFcd.png

1. 如果是空仓库：

图片地址：https://static.oschina.net/uploads/img/201701/17152124_TZe7.png

使用指南

码云 支持的是 svn 协议。 对于 svn 而言，获取一个仓库的代码通常是 checkout，在项目主页我们通常可以获得 URL：

图片地址：http://static.oschina.net/uploads/space/2015/0318/152207_DRMJ_139664.png

这个仓库地址为：

```
svn://git.oschina.net/svnserver/newos
```

1.获取仓库代码：

```
svn checkout svn://git.oschina.net/svnserver/newos newos
```

注意信息 码云的 SVN 接入后端是通过 git 存储库实现，URL 规则为 svn://域名/用户名/项目名 .

使用上述命令，我们将得到项目默认分支的代码。并将本地的工作目录命名为 *newos*

如果最后不带 newos，svn 默认把本地工作目录命名为 项目名

```
svn checkout svn://git.oschina.net/svnserver/newos
```

如果要获得任意分支代码，例如获取 newos 的 dev 分支，请输入近似如下的命令：

• 此时地址为： svn://域名/用户名/项目名/branches/分支名 .

```
svn checkout svn://git.oschina.net/svnserver/newos/branches/dev
```

特别的说明，获取主干分支，也就是 master 分支可以使用下面的分支格式

```
svn checkout svn://git.oschina.net/svnserver/newos/trunk newos
```

svn trunk 分支对应 master 分支 用户应当尽量不使用下面格式

```
svn checkout svn://git.oschina.net/svnserver/newos
```

操作说明

如果部分检出仓库，并且仓库根目录下包含 branches/tags/trunk 这样的目录，请使用完整的路径 layout，如下：

```
svn://git.oschina.net/username/example/trunk/tags/hello
svn://git.oschina.net/username/example/branches/dev/trunk
svn://git.oschina.net/username/example/branches/dev/branches
```

如果没有 master 分支，也就没有 trunk 分支，检出的 URL 不能省略分支名。比如只有一个 dev 分支,必须使用下列格式，否则会提示仓库不存在。

```
svn co svn://git.oschina.net/svnserver/newos/branches/dev  svnserver_dev
```

打开终端,输入上述命令，出现以下下面提示。其中第一个认证领域是用户的密码，这个可以留空。而用户名是用户在 码云（Gitee.com）登陆时使用邮箱地址。密码则是用户登陆 码云 所使用的密

码。

一般而言，svn 会加密缓存用户的用户名密码，所以，对仓库的操作只需要第一次输入用户邮箱和密码。

清除密码缓存，用户目录下的 .subversion/auth/svn.simple 文件夹下的文件。

图片地址：http://static.oschina.net/uploads/space/2015/0318/153828_ptt4_139664.png

下图则是成功的拉取了项目代码。

图片地址：https://static.oschina.net/uploads/img/201701/17152908_GJy9.png

查看本地工作目录信息：

```
svn info
```

图片地址：https://static.oschina.net/uploads/img/201701/17153454_IO8A.png

```
cd helloworld
echo "test" > SVNReadMe.md
#svn add SVNReadMe.md
#svn add * --force类似于git add -A
svn add * --force
svn update .
svn commit -m "first svn commit"
```

Subversion 在提交前建议先使用 svn update 更新工作拷贝。也就相当于 git pull 后再 git push。

Subversion 的提交是在线的，如果机器已经离线，那么提交会失败，这个过程用git的方式理解就是 git commit+git push。

用户使用 svn 提交代码同样会有动态显示。

列出版本库中的目录内容:

```
svn list svn://git.net/svnserver/newos/trunk
```

导出仓库指定分支的所有文件，不含版本控制信息：

```
svn export svn://git.net/svnserver/newos/trunk newos
```

备注

安装 Subversion 客户端

在 Apache 基金会的 Subversion 官网：

http://subversion.apache.org(http://subversion.apache.org)

二进制下载提示页面：

http://subversion.apache.org/packages.html(http://subversion.apache.org/packages.html)

Windows 系统：

与资源管理起集成的 SVN 客户端：TortoiseSVN(http://tortoisesvn.net/downloads.html),通常被叫做"海龟"，为 msi 安装包。可以使用 ExtractMSI(http://pan.baidu.com/s/1szHIn) 解压缩。

很诡异的是，在 Apache 上并没有推荐 TortoiseSVN。

另外还有 SlikSVN，下载地址
：https://sliksvn.com/download/(https://sliksvn.com/download/)

其他的也就不一一介绍了。

Linux 系统

一般而言 Linux 系统自带的包控制软件能够安装 Subversion，如果版本低于1.8，就建议用户下载预编译的二进制或者自己动手编译 Subversion。这里不做过多说明。

OS X

XCode 自带的 Subversion 版本为1.7.x，太老，而 码云（Gitee.com）只支持1.8以上的 SVN 客户端。

如果安装了 Homebrew

```
brew install subversion
```

或者使用 WANdisco 的预编译版本

http://www.wandisco.com/subversion/download#osx(http://www.wandisco.com/subversion/download#osx)

Subversion 客户端的兼容性

我们支持 Apache Subversion 1.8 或者更高的版本，当你安装一个 Subversion 客户端时，如果错误提示是“无法协商验证验证方式” 请确保你的客户端支持 SASL 验证，比如在 Ubuntu 上，你可以安装 libsasl2-dev 然后编译 Subversion, 这样的话客户端是支持 SASL 验证的。

> sudo apt-get install libsasl2-dev

当你使用 svnkit 或者 SubversionJavaHl 这类 IDE 集成客户端，请确保支持 SASL 验证。

关于 GIT 与 SVN 的转换

如果用户存在一个基于 Subversion 托管的项目，要迁移到 码云（Gitee.com），可以使用 git-svn 将项目转变为基于 git 的仓库，然后推送到码云（Gitee.com），这样你依然能够使用SVN对项目进行操作。请记得先在码云（Gitee.com）上新建一个项目

```
git svn clone http://myhost/repo -T trunk -b branches -t tags
git remote add oscgit https://git.oschina.net/user/repo
git push -u oscgit --all
```

通常来说，如果本地存在 SVN 仓库，则可以：

```
git svn clone file:///tmp/svn-repo -T trunk -b branches -t tags
git remote add oscgit https://git.oschina.net/user/repo
git push -u oscgit  --all
```

将项目转移到 GIT@OSC 上以后，使用 svn 命令 checkout 即可对项目进行操作。

高级指南：

http://git-scm.com/book/zh/ch8-2.html(http://git-scm.com/book/zh/ch8-2.html)

安装 git，git-svn

Windows

msysgit 官网 http://msysgit.github.io/(http://msysgit.github.io/),版本比较低。

Github for Windows 提供的 git 工具和 msysgit 一致。

MSYS2 git 下载地址:
http://sourceforge.net/projects/msys2(http://sourceforge.net/projects/msys2)，然后启动终端,安装 git，目前版本为2.4.3。

```
pacman -S git
```

Cygwin git 下载地址: http://www.cygwin.com/(http://www.cygwin.com/),然后使用包管理软件或者直接下载 git 源码编译 git。

```
make configure
./configure --prefix=/usr/local
make
make install
```

Linux

有包管理器的直接用包管理器安装。

如 Ubuntu

```
sudo apt-get install git git-svn
```

也可以手动编译。

Mac OSX

下载地址：http://git-scm.com/download/mac(http://git-scm.com/download/mac)

WebHook

简介：

码云（Gitee.com）钩子功能（callback），是帮助用户 push 了代码后，自动回调一个您设定的 http 地址。 这是一个通用的解决方案，用户可以自己根据不同的需求，来编写自己的脚本程序（比如发邮件，自动部署等）；目前，webhook 支持5种触发方式，支持复选，如图：

图片地址：https://static.oschina.net/uploads/img/201612/15155408_Htaa.png

Old Format? 参数指定了是新的数据格式，还是旧的数据格式，新的数据格式是纯 json，旧的数据格式是 form 参数，具体形式请看下面的数据格式。

其中，Push是指您的项目被您或者该项目的开发者 push 了代码，Tag Push 是指您新建了 tag，Issue 是指您的项目创建、更改内容、关闭、重新打开 issue，评论是指任何用户在您的 issue 下发表了评论，合并请求是指您的项目被人提交了 Pull Request。

说明：

- 请求的方式为 POST 请求
- 格式为 Json
- 超时为5秒（如果程序工作时间较长，建议异步操作。）

Post 数据

为了保证安全以及识别数据来源，我们会在 post 数据中附带您创建 hooks 时填写的密码，请注意，该密码是明文

如何创建 WebHook

关于创建，在项目页面点击管理然后点击图中红色框中部分，如图：

图片地址：http://git.oschina.net/uploads/images/2015/1106/161649_23f28dc9_119968.png

然后再图中所示的地方填写 post 数据接收的地址以及密码：

图片地址：https://static.oschina.net/uploads/img/201612/15155548_qzg9.png

请注意，一定要保证填写的地址是可以访问的，密码也是必须的，不然无法创建成功，如果一切都正常，你将会在刷新后的本页面看到一个类似的东西

图片地址：https://static.oschina.net/uploads/img/201612/15155614_fzp7.png

这样你就创建好了一个 WebHook

数据格式

注意：我们更改了数据的格式，如果您依然想使用较旧的格式，可以在创建时点选Old Format?进行选择，旧的数据格式请往下拉。

以下数据格式，如无特殊说明，均为 json

Note 的数据：


```
{
    "note": "this is note test",
    "noteable_type": "Issue",
    "noteable_id": 23,
    "project": {
        "name": "webhook",
        "path": "webhook",
        "description": "",
        "url": "http://git.oschina.net/oschina/webhook",
        "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
        "git_http_url": "https://git.oschina.net/oschina/webhook.git",
        "git_svn_url": "svn://git.oschina.net/oschina/webhook",
        "namespace": "oschina",
        "path_with_namespace": "oschina/webhook",
        "default_branch": "master"
    },
    "repository": {
        "name": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "description": "",
        "homepage": "http://git.oschina.net/oschina/webhook"
    },
    "url": null,
    "author": {
        "user_name": "a123",
        "email": "123@123.com"
    },
    "hook_name": "note_hooks",
    "password": "pwd"
}
```


PullRequest 的数据：

```
{
    "iid": 1,
    "title": "webhooktest",
    "body": "webhook",
    "state": "opened",
    "merge_status": "unchecked",
    "source_branch": "master",
    "source_repo": {
        "project": {
            "name": "webhook",
            "path": "webhook",
            "description": "",
            "url": "http://git.oschina.net/oschina/webhook",
            "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
            "git_http_url": "https://git.oschina.net/oschina/webhook.git",
            "git_svn_url": "svn://git.oschina.net/oschina/webhook",
            "namespace": "oschina",
            "path_with_namespace": "oschina/webhook",
            "default_branch": "master"
        },
        "repository": {
            "name": "webhook",
            "url": "http://git.oschina.net/oschina/webhook",
            "description": "",
            "homepage": "http://git.oschina.net/oschina/webhook"
        }
    },
    "target_branch": "dev",
    "target_repo": {
        "project": {
            "name": "webhook",
            "path": "webhook",
            "description": "",
            "url": "http://git.oschina.net/oschina/webhook",
            "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
            "git_http_url": "https://git.oschina.net/oschina/webhook.git",
            "git_svn_url": "svn://git.oschina.net/oschina/webhook",
            "namespace": "oschina",
            "path_with_namespace": "oschina/webhook",
            "default_branch": "master"
        },
        "repository": {
```

```
        "name": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "description": "",
        "homepage": "http://git.oschina.net/oschina/webhook"
      }
    },
    "author": {
      "user_name": "a123",
      "email": "123@123.com"
    },
    "hook_name": "merge_request_hooks",
    "password": "123456sdfgsdfg",
    "key1": "gsdfg",
    "key2": "345645"
}
```


Issue 的数据：


```
{
    "iid": 21,
    "title": "Issue test",
    "description": "this is issue description",
    "state": "opened",
    "assignee": null,
    "milestone": null,
    "user": {
      "name": "123",
      "username": "a123"
    },
    "project": {
      "name": "webhook",
      "path": "webhook",
      "url": "http://git.oschina.net/oschina/webhook",
      "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
      "git_http_url": "https://git.oschina.net/oschina/webhook.git",
      "git_svn_url": "svn://git.oschina.net/oschina/webhook",
      "namespace": "oschina",
      "path_with_namespace": "oschina/webhook",
      "default_branch": "master"
    },
    "repository": {
```

```
        "name": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "description": "",
        "homepage": "http://git.oschina/oschina/webhook"
    },
    "hook_name": "issue_hooks",
    "password": "pwd"
}
```

Push 的数据：

```
{
    "before": "fb32ef5812dc132ece716a05c50c7531c6dc1b4d",
    "after": "ac63b9ba95191a1bf79d60bc262851a66c12cda1",
    "ref": "refs/heads/master",
    "user_name": "123",
    "user": {
        "email": "123@123.com",
        "name": "123",
        "time": "2016-12-09T17:28:02 08:00"
    },
    "repository": {
        "name": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "description": "",
        "homepage": "http://git.oschina.net/oschina/webhook"
    },
    "commits": [
        {
            "id": "ac63b9ba95191a1bf79d60bc262851a66c12cda1",
            "message": "1234 bug fix",
            "timestamp": "2016-12-09T17:28:02 08:00",
            "url":
"http://git.oschina.net/oschina/webhook/commit/ac63b9ba95191a1bf79d60bc262851
a66c12cda1",
            "author": {
                "name": "123",
                "email": "123@123.com",
                "time": "2016-12-09T17:28:02 08:00"
            }
        }
```

```
    ],
    "total_commits_count": 1,
    "commits_more_than_ten": false,
    "project": {
        "name": "webhook",
        "path": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
        "git_http_url": "https://git.oschina.net/oschina/webhook.git",
        "git_svn_url": "svn://git.oschina.net/oschina/webhook",
        "namespace": "oschina",
        "path_with_namespace": "oschina/webhook",
        "default_branch": "master"
    },
    "hook_name": "push_hooks",
    "password": "pwd"
}
```


Tag 的数据


```
{
    "ref": "refs/tags/v4",
    "before": "00000000",
    "after": "ac63b9ba95191a1bf79d60bc262851a66c12cda1",
    "project": {
        "name": "webhook",
        "path": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "git_ssh_url": "git@git.oschina.net:oschina/webhook.git",
        "git_http_url": "https://git.oschina.net/oschina/webhook.git",
        "git_svn_url": "svn://git.oschina.net/oschina/webhook",
        "namespace": "oschina",
        "path_with_namespace": "oschina/webhook",
        "default_branch": "master"
    },
    "repository": {
        "name": "webhook",
        "url": "http://git.oschina.net/oschina/webhook",
        "description": "",
        "homepage": "http://git.oschina.net/oschina/webhook"
    },
```

```
    "hook_name": "tag_push_hooks",
    "password": "pwd"
}
```

勾选了 Old Format 的 webhook 数据格式为：

Push 的 Hook 发送的数据类似于：

```
hook=
{
    "password": "123123",
    "hook_name": "push_hooks",
    "push_data": {
        "before": "0000000000000000000000000000000000000000",
        "after": "ec7159240a346fa5988913aa3057b902a4acb126",
        "ref": "refs/heads/master",
        "user_name": "zoker",
        "user": {
            "id": 28,
            "email": "zoker@zoker.com",
            "name": "zoker",
            "time": "2016-12-15T15:42:41+08:00"
        },
        "repository": {
            "name": "MyPro",
            "url": "http://git.com/zoker/MyPro.git",
            "description": "",
            "homepage": "http://git.com/zoker/MyPro"
        },
        "commits": [
            {
                "id": "ec7159240a346fa5988913aa3057b902a4acb126",
                "message": "A Test For WebHooks",
                "timestamp": "2015-11-06T13:21:07+08:00",
                "url":
"http://git.com/zoker/MyPro/commit/ec7159240a346fa5988913aa3057b902a4acb126"
,
                "author": {
                    "name": "zoker",
```

```
                "email": "zoker@zoker.com",
                "time": "2015-11-06T13:21:07+08:00"
            }
        }
    ],
    "project": {
        "name": "MyPro",
        "path": "MyPro",
        "url": "http://git.com/zoker/MyPro",
        "git_ssh_url": "git@git.oschina.net:zoker/MyPro.git",
        "git_http_url": "http://git.com/zoker/MyPro.git",
        "git_svn_url": "svn://git.com/zoker/MyPro",
        "namespace": "zoker",
        "path_with_namespace": "zoker/MyPro",
        "default_branch": "master"
    },
    "total_commits_count": 1,
    "commits_more_than_ten": false
  }
}
```


Tag Push 的 push 数据类似于：


```
hook=
{
  "password": "25b3pzwtxwu8as9",
  "hook_name": "tag_push_hooks",
  "push_data": {
    "ref": "refs/tags/pke1de6qpjbwpz9",
    "before": "00000000",
    "after": "1b0355de43a8e24ef59baa4557d4617732ed1d1b"
  }
}
```


Issue 的 push 数据类似于：


```
hook=
{
  "password": "671vmti2lkhmmqv",
  "hook_name": "issue_hooks",
```

```
    "push_data": {
        "id": 361909,
        "title": "x2tf25k8p0nrkoj",
        "state": "closed",
        "assignee": null,
        "milestone": null
    }
}
```

评论的 push 数据类似于：

```
hook=
{
    "password": "e7i6y15rji72sof",
    "hook_name": "note_hooks",
    "push_data": {
        "id": 212599,
        "note": "_u72b6u6001u66f4u6539u4e3a **u5df2u5173u95ed**_",
        "noteable_type": "Issue",
        "noteable_id": 361909,
        "author": {
            "user_name": "gitlab_test_1",
            "email": "gitlabtest_1@gitlab.com"
        }
    }
}
```

合并请求的 push 数据类似于：

```
hook=
{
    "password": "feds3os4h2f8s4k",
    "hook_name": "merge_request_hooks",
    "push_data": {
        "id": 15339,
        "project": {
            "project_id": 616834,
            "project_name": "test_gitosc_20151106145021322"
        },
        "target_branch": "master",
```

```
        "source_repo": {
            "project_id": 616836,
            "project_name": "test_gitosc_20151106145021322"
        },
        "source_branch": "master",
        "author": {
            "user_name": "gitlab_test_2",
            "email": "gitlabtest_2@gitlab.com"
        },
        "title": "pull_request_test",
        "body": "####################"
    }
}
```


以上就是5种钩子触发时推送的数据范例，为便于理解和使用，以上数据全部经过 json 化。另，本文档中范例数据仅供参考，请以实际收到的数据以及格式为准。

码云Pages

码云 Pages 是一个免费的静态网页托管服务，您可以使用 码云 Pages 托管博客、项目官网等静态网页。如果您使用过 Github Pages 那么您会很快上手使用码云的 Pages服务。

> Jekyll 是一个简单的博客形态的静态站点生产机器。它有一个模版目录，其中包含原始文本格式的文档，通过 Markdown （或者 Textile ）以及 Liquid 转化成一个完整的可发布的静态网站，你可以发布在任何你喜爱的服务器上。Jekyll 也可以运行在 码云 Pages上，也就是说，你可以使用码云的服务来搭建你的项目页面、博客或者网站，而且是完全免费的。

Jekyll 使用文档：http://jekyll.bootcss.com/docs/home/

1. 使用前须知

1、项目必须有 index.html 才可以正常访问

2、静态网页的样式可以自己写，也可以拿一些静态模版来修改，Pages服务本身不提供任何形式的模版，但我们后续会陆续增加例子，供大家参考使用。

3、如果你想以根目录的形式访问自己的静态网站，只需要建立一个与自己个性地址同名的项目即可，如 http://git.oschina.net/ipvb 这个用户，想要创建一个自己的站点，但不想以子目录的方式访问，想以ipvb.oschina.io直接访问，那么他就可以创建一个名字为ipvb的项目 http://git.oschina.net/ipvb/ipvb 部署完成后，就可以以 http://ipvb.oschina.io 进行访问了。

2. 一个小白的Pages搭建之旅

A.新建项目 test_pages

图片地址：https://static.oschina.net/uploads/img/201606/01172928_Wv13.png

点击创建完成项目的创建

B.添加文件 index.html (注意名称是index.html哦！)

点击新建文件

图片地址：https://static.oschina.net/uploads/img/201606/01173033_nL18.png

文件名输入index.html，内容就是简单的html

图片地址：https://static.oschina.net/uploads/img/201606/01173322_qoin.png

点击提交，将文件提交到仓库

C.选择 pages 服务

图片地址：https://static.oschina.net/uploads/img/201606/01173449_hHY1.png

D.选择需要部署的分支，这里选择 Master 启动服务。

图片地址：https://static.oschina.net/uploads/img/201606/01173541_QDTs.png

E.访问生成的网站地址，即可以查看你部署的静态页面啦！

图片地址：https://static.oschina.net/uploads/img/201606/01173617_VOe9.png

图片地址：https://static.oschina.net/uploads/img/201606/01173639_s4t6.png

3. 已经有 Pages 项目如何部署到码云的 Pages

以jQuery-File-Upload项目为例，项目地址：https://github.com/blueimp/jQuery-File-Upload

它在Github上的Pages地址是：https://blueimp.github.io/jQuery-File-Upload/

如果想把它转移到码云Pages，只需要登录你的码云账户，点击右上角的+号，选择新建项目

图片地址：https://static.oschina.net/uploads/img/201606/01170937_93rs.png

图片地址：https://static.oschina.net/uploads/img/201606/01170914_Vb4Q.png

然后点击创建，项目会在后台自动导入，导入成功后，点击菜单栏的Pages

图片地址：https://static.oschina.net/uploads/img/201606/01171047_x3BN.png

图片地址：https://static.oschina.net/uploads/img/201606/01171113_lM7g.png

这里我们默认的Pages服务分支是osc-pages，但是你也已选择自己静态页面所在的分支，这里jQuery-File-Upload项目的静态页面分支是gh-pages，选择gh-pages并点击启动服务。

图片地址：https://static.oschina.net/uploads/img/201606/01171334_7QAP.png

至此，静态网页已经部署成功，访问提供的地址：http://silentboy.oschina.io/jQuery-File-Upload/ 即可查看到jQuery-File-Upload项目的静态官网。

图片地址：https://static.oschina.net/uploads/img/201606/01171535_B1u8.png

4. Pages 进阶，使用 Jekyll 生成自己的静态网页

Jekyll 究竟是什么？

Jekyll 是一个简单的博客形态的静态站点生产机器。它有一个模版目录，其中包含原始文本格式的文档，通过 Markdown （或者 Textile ）以及 Liquid 转化成一个完整的可发布的静态网站，你可以发布在任何你喜爱的服务器上。Jekyll 也可以运行在 码云（ Gitee.com ） Pages 上，也就是说，你可以使用码云的服务来搭建你的项目页面、博客或者网站，而且是完全免费的。

Jekyll 使用文档：http://jekyll.bootcss.com/docs/home/

其他帮助:

如何自定义404？

答：在项目根目录新建 404.html 文件即可。

生成并部署SSH key

1.如何生成ssh公钥

你可以按如下命令来生成 sshkey:

```
ssh-keygen -t rsa -C "xxxxx@xxxxx.com"

# Generating public/private rsa key pair...
# 三次回车即可生成 ssh key
```

查看你的 public key，并把他添加到码云（ Gitee.com ） SSH key添加地址 (http://git.oschina.net/profile/sshkeys)

```
cat ~/.ssh/id_rsa.pub
# ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6eNtGpNGwstc....
```

添加后，在终端（ Terminal ）中输入

```
ssh -T git@git.oschina.net
```

若返回

```
Welcome to Git@OSC, yourname!
```

则证明添加成功。

•

2.怎么添加用户 ssh key?

1. 点击右上角的
图片地址：https://static.oschina.net/uploads/img/201604/14154017_ogkd.png

标志，进入个人中心,然后点击左侧的 ssh 公钥后在下图位置填写你的 ssh 公钥

2. 点击确定,然后验证密码(即你的注册账号密码)就完成了ssh 公钥添加。

图片地址：https://static.oschina.net/uploads/img/201610/18115822_miTO.png

3.项目的 ssh key 和用户的 ssh key 两处地方有什么不同?

> 项目的 sshkey 只针对项目，且我们仅对项目提供了部署公钥，即项目下的公钥仅能拉取项目，这通常用于生产服务器拉取仓库的代码。

而用户的 key 则是针对用户的，用户添加了 key 就对用户名下的项目和用户参加了的项目具有权限，一般而言，用户的key具有推送和拉取的权限，而项目的 key 则只具有拉取权限。

项目成员权限说明

在码云平台,项目成员一共只有4种权限:名字以及对应权限如下:

访客

对于公有项目：

- 创建 issue
- 评论
- Clone 和 Pull 项目
- 打包下载代码
- Fork 项目
- 创建 pull request

报告者

- 继承访客的权限
- 私有项目：不能查看代码

- 私有项目：不能下载代码
- 私有项目：不能 fork 代码

观察者

- 继承报告者权限
- 创建 wiki
- 可以clone下载代码
- 可以 pull
- 不能 push 代码
- 私有项目：可以fork

开发者

- 创建 issue
- 评论
- Clone 和 Pull 项目
- 打包下载代码
- 创建 pull request
- 创建分支
- 推送分支
- 删除分支
- 创建标签（里程碑）
- 创建 wiki

管理员

- 创建 issue
- 评论
- Clone 和 Pull 项目
- 打包下载代码
- 创建 pull request
- 创建分支
- 推送分支
- 删除分支
- 创建标签（里程碑）
- 创建 wiki
- 添加项目成员
- 强制推送分支

• 编辑项目属性

组织管理员

• 编辑组织属性
• 增加成员
• 添加 / 删除项目
• 设置组织管理员
• 更改成员项目权限

Commit关联issue

1. 通过 commit 关闭 issue

1. **close	closes	closing	closed	fixed	fix	resolved #issueid ** ，就可以关闭 issue 了。

1. 效果如下：

图片地址：https://static.oschina.net/uploads/img/201609/21175331_we9I.png

2. 通过 commit 评论 issue

1. **comment #issueid comments** ，就可以评论 issue 了。
2. 效果如下：

图片地址：https://static.oschina.net/uploads/img/201609/21175507_dbqt.png

Pull Request关联issue

通过 Pull Request 关闭 issue

1. 在 pr 的内容里面指定需要关闭的 issue 的 id ,例如：

• 一个 pr 可以关联多个 issue , 例如同时关联 issue1 , issue2 格式为：#1,#2

图片地址：https://static.oschina.net/uploads/img/201611/03183808_VgPB.png

2. 在 issue 详情页可以看到关联关系

图片地址：https://static.oschina.net/uploads/img/201611/03183846_jNaV.png

3. 当 pr 合并之后其关联的 issue 被关闭

图片地址：https://static.oschina.net/uploads/img/201611/03183939_GXBA.png

wiki支持二级目录和TOC

1. wiki 二级目录

• 新建 wiki 的时候将 wiki 的 title 命名为，**xx/yy** 则表示 xx 目录下的 yy。

2. wiki TOC

• wiki 内容里面加上关键字**[TOC]**或者[MENU](关键字不区分大小写)

图片地址：https://static.oschina.net/uploads/img/201611/03190318_H9cD.png

• 效果图如下：

图片地址：https://static.oschina.net/uploads/img/201611/03185801_uBNn.png

码云一键登录功能

码云平台提供一键登录功能，效果图如下：

图片地址：https://static.oschina.net/uploads/img/201611/04104222_8w2M.png

'一键登录' 需要满足2个条件

1. 条件一： 您已经登 录http://www.oschina.net

2. 条件二： 您已经绑定了社区账号， 可以在码云的修改账号的第三方登录处进行绑定即，https://gitee.com/profile/account

图片地址：https://static.oschina.net/uploads/img/201611/04104453_wGaN.png

网页操作文件或者文件夹

1.怎么在线新建文件夹?

> 答：点击新建文件，将文件命名为xx/yy，即表示xx文件夹下的yy文件。

2.怎么在线修改文件或者文件夹？

> 答：选中要修改的文件或者文件夹，点击右键，就可以选择重命名或者删除功能。

什么是 G 币？

什么是 G 币？

G 币是开源中国旗下码云（Gitee.com）平台推出的线上虚拟货币，可以在码云（Gitee.com）平台提出意见、邀请好友注册等动作赚取。

可以用 G币做什么？

可以利用 G 币兑换码云（Gitee.com）定制礼品：码云T恤、手机壳、超大鼠标垫、码客杯等。

如何获取 G 币？

1. 新注册用户获得 **2G** 币
2. 邀请一位用户完成注册并新建项目可得 **1G** 币
3. 改进码云（Gitee.com）的建议被采纳：**5G** 币(被打上标签即视为采纳)
4. 不定期发布的各种形式的 G币 悬赏 活动(http://git.oschina.net/invite)（请关注 **码云微博** 及 **码云微信公众号**）

码云企业版

企业版简介

导言：

码云企业版创建于2016年，是开源中国继个人免费代码托管市场后致力服务于政府企业机构软件管理的代码托管产品业务线。作为中国代码托管服务行业的领导者，开源中国凭借社区超过200万活跃用户、国内首屈一指的软件众包服务和100万+项目托管平台等方面的深厚积累，成功推出各行业用户所急需的更为安全、稳定和快速的企业级代码托管服务的产品。

码云企业版倍受各行各业的欢迎。从政府机构到肩负国家金融使命的金融机构，从顶尖的世界500强到国内普通的中小企业，从CIO、CTO到普通工程师……他们都在使用开源中国码云的产品和服务来提升软件的开发效率，降低开发成本，最大程度上实现资源的共享和协同开发，并在与众多政府机构和企业用户的合作中，赢得了良好的业内口碑。

开源中国码云企业版具有遥遥领先业内的安全服务水准与应急响应能力，助力企业用户决胜工业4.0 时代。

正文

码云企业版致力于为企业提供最优质的产品和服务，包括更充裕的存储空间、更便捷的团队协作、更可靠的企业级安全、更专业的独立管理、更精准的资源统计等全方位服务，充分解决企业对代码仓库、组织、员工等资源的管理问题，如代码仓库的质量分析、操作记录查询；企业组织的增删、指派管理；企业员工帐号的增删屏蔽、代码绩效考察等。更重要的是，进驻企业服务的各企业之间的资源、企业与码云平台之间的资源都是相互隔离的，即码云企业服务面对入驻企业呈现多租户的服务模式。

企业注册码云企业服务后将得到一个企业帐号，这个帐号作为企业在码云的超级管理员。当企业超级管理员登陆码云时便可以直接进入企业空间，在此空间内只会看到企业自身的资源。企业超级管理员进入后台可以编辑企业的相关信息、导入企业的仓库以及员工、组织等资源、也可以指派员工为管理员等。

一、码云企业版增值服务

1、VIP专线客服

让每个企业都有专属于自己的 VIP 客服服务，实现真正高效率解决问题。

客服专线服务时间：（周一至周五） 10：00 至 19：00，法定节假日除外；

2、故障赔付

完善可靠的故障赔付机制来护航，让代码托管零风险。

3、法律保障

签订服务合同，提供服务发票。法律防护，使用更放心。

4、权限隔离

全新多维度权限管控，立体式维护企业资产。例如：

5、企业管理

内部团队和外包团队管理界限模糊？私有项目，内部开源项目，外包项目管理混乱？成员信息、权限难以管理？全新企业版为您提供专属的企业管理功能，让企业工作状态一目了然，最优化企业人力资源。

6、监控告警

关注代码安全，跟踪项目变化，异常状况第一时间告警通知。

7、敏感操作二次验证

管理、删除企业资产需要同时输入密码以及手机验证码，双层认证防止误操作等安全问题，企业管理更安全、更放心。

8、详细统计

项目进度无法掌控？成员贡献无法统计？操作流程无法追溯？你该试一试全新企业版统计解决方案，帮您通通搞定，让企业研发尽收眼底。

二、关于企业版套餐介绍

图片地址：https://static.oschina.net/uploads/img/201704/01145922_M0nb.jpg

【注】：目前已开通在线购买方式：https://gitee.com/enterprises

如果合同到期终止，代码资产怎么处理？

由购买的高级版本降为企业免费版，如果超出免费版的额度（如成员人数、项目容量等），则限制相应的功能（如push、pull等）的使用，直至用户删减至满足企业免费版的额度。

企业免费版会有赔付机制么？

赔付机制只适用于企业标准版及其以上版本，免费版是不会有的。

如何从个人免费版升级到付费企业版？

由企业版生成转移码，通过转移码可以将个人免费版的项目升级到企业版。具体步骤如下所示：

在企业版界面点击【项目管理】->【添加项目】->【导入项目】：

图片地址：https://static.oschina.net/uploads/img/201703/29164459_3cYh.jpg

点击【生成转移码】按钮生成转移码（转移码一小时内有效）：

图片地址：https://static.oschina.net/uploads/img/201703/29164640_1772.jpg

生成转移码如下：

图片地址：https://static.oschina.net/uploads/img/201703/29170041_VXSL.jpg

然后，将此转移码复制到码云项目管理页面的“转移到企业”表单，然后确认即可。

图片地址：https://static.oschina.net/uploads/img/201703/29170511_EKgu.jpg

项目转移至企业的同时也可以将该项目内的所有成员一起转移至企业。

图片地址：https://static.oschina.net/uploads/img/201703/29170910_65cM.jpg

企业成员如何修改手机号？

请在个人页面中的【个人资料】中，进行修改并验证手机号。

图片地址：https://static.oschina.net/uploads/img/201702/13173442_niLJ.png

FAQ

Git 操作管理

新手小白如何快速的在码云平台注册账号并完成第一次提交

在码云平台注册以及创建第一个项目

注册账号

首先是在码云平台注册，注册链接请点解这里(https://git.oschina.net/signup)如下图：

图片地址：https://static.oschina.net/uploads/img/201704/01151716_Q7lu.jpg

依次填入各项，需注意的是：邮箱最好填写国内的邮箱，以免因为众所周知的原因无法接收激活邮件，**个性地址一经选定，无法修改，请慎重填写。**

我们会向您的邮箱发送一份激活邮件，请点击其中的链接激活账号，账号激活后，注册流程就算完成了，接下来就是创建第一个项目了

创建项目

点击右上角的
图片地址：https://static.oschina.net/uploads/img/201704/01152036_vITZ.jpg

或者
图片地址：https://static.oschina.net/uploads/img/201704/01152303_3MMo.jpg

，跳转至项目创建页面：

图片地址：https://static.oschina.net/uploads/img/201704/01152603_Zb3l.jpg

依次填写各项信息然后点击创建就可以了，这样我们一个项目就创建好了如果没有意外，你将看到这个页面:

图片地址：https://static.oschina.net/uploads/img/201704/01152726_Wux6.jpg

这样，我们在码云平台就创建好了一个项目

本地初始化一个项目

首先，你需要执行下面两条命令，作为 git 的基础配置，作用是告诉 git 你是谁，你输入的信息将出现在你创建的提交中。

```
git config --global user.name "你的名字或昵称"
git config --global user.email "你的邮箱"
```

然后在你的需要初始化版本库的文件夹中执行：

```
git init
git remote add origin <你的项目地址> //注:项目地址形式为
:http://git.oschina.net/xxx/xxx.git或者 git@git.oschina.net:xxx/xxx.git
```

这样就完成了一次版本你的初始化。

如果你想克隆一个项目，只需要执行：

```
git clone <项目地址>
```

完成第一次提交

进入你已经初始化好的或者克隆项目的目录，然后执行：

```
git pull origin master
<这里需要修改/添加文件，否则与原文件相比就没有变动>
git add .
git commit -m "第一次提交"
git push origin master
```

然后如果需要账号密码的话就输入账号密码，这样就完成了一次提交。

此时，你可以在你的个人面板、项目主页查看到你的提交记录，例如：http://git.oschina.net/oschina/git-osc/commit/f3dd1c5bae48fa4244e2595a39e750e5606dd9be](http://git.oschina.net/oschina/git-osc/commit/f3dd1c5bae48fa4244e2595a39e750e5606dd9be

按照本文档新建的项目时，在码云平台仓库上已经存在 readme 文件，故在提交时可能会存在冲突，这时您需要选择的是保留线上的文件或者舍弃线上的文件，如果您舍弃线上的文件，则在推送时选择强制推送，强制推送需要执行下面的命令：

```
git push origin master -f
```

如果您选择保留线上的 readme 文件，则需要先执行：

```
git pull origin master
```

然后才可以推送，如果发生冲突，则需要先解决冲突，关于如何处理冲突，请参阅如何处理代码冲突(http://git.mydoc.io?v=16912&t=66902)这一小节。

http(s)方式如何自动记住密码

https 方式每次都要输入密码，按照如下设置即可输入一次就不用再手输入密码的困扰而且又享受 https 带来的极速

按照以下设置记住密码十五分钟：

```
git config --global credential.helper cache
```

如果你想自定义记住的时间，可以这样：

```
git config credential.helper 'cache --timeout=3600' //这里记住的是一个小时，如需其他时间，请修改3600为你想修改的时间，单位是秒
```

你也可以设置长期记住密码：

```
git config --global credential.helper store
```

或修改仓库的地址带上你的账号密码

```
http://yourname:password@git.oschina.net/name/project.git //注意，码云平台同时支持个性地址与邮箱，当使用邮箱时，请对@符号使用%40替换
```

如果你原本使用的 ssh 地址想更换成 http(s) 地址，可以执行以下命令:

```
//删除原本的ssh仓库地址
git remote rm origin //origin 代表你原本ssh地址的仓库的别名
//新增http地址的仓库
git remote add origin http://git.oschina.net/username/project.git
```

如何处理代码冲突

冲突合并一般是因为自己的本地做的提交和服务器上的提交有差异，并且这些差异中的文件改动，Git 不能自动合并，那么就需要用户手动进行合并

如我这边执行git pull origin master

如果 Git 能够自动合并，那么过程看起来是这样的

图片地址：http://git.oschina.net/uploads/images/2016/0226/113507_cca8cd22_62561.gif

拉取的时候，Git 自动合并，并产生了一次提交。

如果 Git 不能够自动合并，那么会提示

图片地址：http://git.oschina.net/uploads/images/2016/0226/113621_dbc985b5_62561.png

这个时候我们就可以知道README.MD有冲突，需要我们手动解决，修改README.MD解决冲突

图片地址：http://git.oschina.net/uploads/images/2016/0226/113823_fffe18cf_62561.png

可以看出来，在1+1=几的这行代码上产生了冲突，解决冲突的目标是保留期望存在的代码，这里保留1+1=2，然后保存退出。

图片地址：http://git.oschina.net/uploads/images/2016/0226/114159_426b8d65_62561.png

退出之后，确保所有的冲突都得以解决，然后就可以使用

```
git add .
git commit -m "fixed conflicts"
git push origin master`
```

即可完成一次冲突的合并。

整个过程看起来是这样的

图片地址：http://git.oschina.net/uploads/images/2016/0226/114058_429e8b54_62561.gif

如何进行版本回退

版本回退有多种方式,下面一一演示:

回退到当前版本(放弃所有修改)

图片地址：https://static.oschina.net/uploads/img/201603/10161457_lf5m.gif

放弃某一个文件的修改

图片地址：https://static.oschina.net/uploads/img/201603/10161707_dstz.gif

回退到某一版本但保存自该版本起的修改

图片地址：https://static.oschina.net/uploads/img/201603/10162127_dLHO.gif

回退到某一版本并且放弃所有的修改

图片地址：https://static.oschina.net/uploads/img/201603/10162634_CKmm.gif

回退远程仓库的版本

先在本地切换到远程仓库要回退的分支对应的本地分支，然后本地回退至你需要的版本，然后执行：

```
git push <仓库名> <分支名> -f
```

如何以当前版本为基础，回退指定个commit

首先，确认你当前的版本需要回退多少个版本，然后计算出你要回退的版本数量，执行如下命令

```
git reset HEAD~X //X代表你要回退的版本数量，是数字！！！！
```

需要注意的是，如果你是合并过分支，那么背合并分支带过来的 commit 并不会被计入回退数量中，而是只计算一个，所以如果需要一次回退多个 commit，不建议使用这种方法

如何回退到和远程版本一样

有时候，当发生错误修改需要放弃全部修改时，可以以远程分支作为回退点退回到与远程分支一样的地方，执行的命令如下

```
git reset --hard origin/master // origin代表你远程仓库的名字，master代表分支名
```

git submodule

我们可能偶尔在一下项目中看到类似下图的东西

图片地址：https://static.oschina.net/uploads/img/201702/16161611_xjmS.png

这其实是 git 的子模块功能,作用是为了在项目中引入另一个项目的部分或全部内容，但是这两个项目是分开开发或管理的,这样直接使用 git submodule 就可以不必要将另一个项目复制一份直接引用即可。

常用 git submodule命令

```
# 添加 git 子模块

git submodule add repourl

# 初次clone带有子模块的仓库如果想查看子模块的内容：

git submodule init
git submodule update
或者  git submodule update --init --recursive

# 更新子模块地址

vi .gitmodules   #修改url地址
git submodule sync

然后提交更改即可。

# 删除子模块

删除 .gitsubmodule中对应submodule的条目
删除 .Git/config 中对应submodule的条目
git rm --cached {submodule_path}  # submodule_path即子模块在仓库下的路径
git commit -m 'xxx'
git push
```

具体的关于 git submodule 的说明以及使用，请点击这里(https://git-scm.com/book/zh/v1/Git-工具-子模块)

为什么在push的时候，出现了413错误，push失败

为了提供更稳定更快的服务，我们的 http 服务器 nginx 设置了单次上传大小限制为 200M，如果您的单次 push 超过 200M，将被服务器拒绝，返回 413 错误。遇到这种问题，可以在 https://gitee.com/keys 页面上传 SSH 公钥，通过 SSH 通道 push 即可。

为什么pull request合并不了

pull request 无法合并是因为您的 pull request 目标分支的提交中存在于您的修改冲突的部分，这时，您需要先在本地将目标分支拉取下来合并到提交 pull request 的项目的分支，解决冲突后重新

推送到提交了 pull request 的项目的分支，我们后台会自动更新，一旦检测到两个仓库不存在冲突时，便会提示可以自动合并，我们建议，在 pull request 合并前，应该处于可以自动合并的状态。

如何对一个项目提交Pull Request

Pull Request 是两个仓库提交变更的一种方式，通常用于 fork 项目与被 fork 项目的差异提交，同时也是一种非常好的团队协作方式，下面，就来讲解如何在码云平台提交 Pull Request：

PS：码云平台限制 Pull Request 源项目与目标项目需存在 fork 与被 fork 关系，故如果你要提交 Pull Request，必须先 fork 一个项目，然后才能对该项目提交 Pull Request，同时，以该项目为父项目的所有项目，您也均可以提交 Pull Request。

如何 fork 项目

fork 项目时非常简单的，进到项目页面，然后找到右上角的 fork 按钮，点击后再次点击确定，等待系统在后台完成仓库克隆操作，你就完成了 fork 操作，如图：

图片地址：https://static.oschina.net/uploads/img/201607/11170038_5Rzf.png

如何提交 Pull Request：

首先，您的项目与目标项目必须存在差异，这样才能提交,比如这样：

图片地址：https://static.oschina.net/uploads/img/201607/11164020_j8To.png

如果不存在差异，或者目标分支比你提 Pull Request 的分支还要新，则会得到这样的提示：

图片地址：https://static.oschina.net/uploads/img/201607/11164209_LfgW.png

然后，填入 Pull Request 的说明，点击提交 Pull Request ，就可以提交一个 Pull Request 了，就想下图所示的那样：

图片地址：https://static.oschina.net/uploads/img/201607/11171413_x7Jh.png

如何对已经存在的 Pull Request 的进行管理

首先，对于一个已经存在的 Pull Request，如果只是观察者，报告者等权限，那么访问将会受到限制，具体权限限制请参考码云平台关于角色权限的内容，下文涉及的部分，仅针对管理员权限，如果您发现不太一样的地方，请检查您的权限是不是管理员或该 Pull Request 的创建者。

如何修改一个已经存在的 Pull Request

点击 Pull Request 的详情界面右上角的编辑按钮，就会弹出编辑框，在编辑框中修改你需要修改的信息，然后点击保存即可修改该 Pull Request，如下图所示：

图片地址：https://static.oschina.net/uploads/img/201607/11182105_veIB.png

请注意，在该界面，可以对 Pull Request 进行指派负责人，指派测试者等等操作，每一个操作均会通知对应的人员

对 Pull Request 的 bug 修改如何提交到该 Pull Request 中

对于 Pull Request 中的 bug 修复或者任何更新动作，均不必要提交新的 Pull Request，仅仅只需要推送到您提交 Pull Request 的分支上，稍后我们后台会自动更新这些提交，将其加入到这个 Pull Request 中去

Pull Request 不能自动合并该如何处理

在提交完 Pull Request 的后，在这个 Pull Request 处理期间，由原本的能自动合并变成不能自动合并，这是一件非常正常的事情，那么，这时，我们有两种选择，一种，继续合并到目标，然后手动处理冲突部分，另一种则是先处理冲突，使得该 Pull Request 处于可以自动合并状态，然后采用自动合并，一般来讲，我们官方推荐第二种，即先处理冲突，然后再合并。具体操作为：

先在本地切换到提交 Pull Request 的分支，然后拉取目标分支到本地，这时，会发生冲突，参考如何处理代码冲突(http://git.mydoc.io?v=16912&t=83148) 这一小节将冲突处理完毕，然后提交到 Pull Request 所在的分支，等待系统后台完成 Pull Request 的更新后，Pull Request 就变成了可自动合并状态。

Pull Request 不小心合并了，可否回退

对于错误合并的 Pull Request，我们提供了回退功能，该功能会产生一个回退 XXX 的 Pull Request，接受该 Pull Request 即可完成回退动作，注意，回退本质上是提交一个完全相反的 Pull Request，所以，你仍然需要进行测试来保证完整性，另，为了不破坏其他 Pull Request，建议只有需回退的 Pull Request 处于最后一次合并操作且往上再无提交时执行回退动作，否则请手动处理。

git clone git@osc项目出错 server aborted the ssl handshake

这是您的网络连接有点问题，建议您更换 http 地址或者 ssh 地址或者过一会再试。

用户账户

个性地址

1. 什么是个性地址？？

> 个性地址即您在码云平台的用户 path，是您在码云平台的唯一身份标识，同时也是您在码云平台创建项目时项目所在的空间的地址，该地址类似于二级域名，每一个用户有且只能有一个个性地址。

- 例如某项目地址为： https://gitee.com/xxx/test.git， xxx即你的个性地址
- 注册的时候请选择自己想要的个性地址 username，因为目前还不允许用户自己修改个性地址。

2.为什么无法修改个性地址？

>

> 个性后缀会作为您项目地址的一部分，这个地址是要配置在您的本地仓库中的，如果修改个性后缀，您的本地项目地址也会更改，而且所有使用这个项目的人的本地项目地址都要更改，非常不便。因为这样的原因，我们不提供自助修改个性后缀的功能，请您在注册的时候谨慎填写。

为什么我收不到激活/通知/重置密码等邮件

> 本站采用SendCloud服务发送邮件，我们尽力保证邮件快速送达，但是发送邮件仍然可能会有延迟，正常情况下都可以在10分钟内收到邮件，如果你没有收到邮件，可查看是否被识别为垃圾邮件，如果垃圾邮件中仍然没有，可通过建议(http://git.oschina.net/oschina/git-osc/issues)，或者在开源中国社区(http://www.oschina.net)上告知我们。另，部分企业邮箱存在反垃圾邮件服务，如您的企业配置了反垃圾邮件服务或者使用了如阿里云之类的企业邮，**请确保将@git.oschina.net的域名设置成白名单**，否则无法接收激活邮件，另外由于众所周知的原因，某些海外邮箱无法接收我们发出的邮件，遇上这种情况，请通过建议或者加入QQ群:655903986 然后联系群管理员。

【注】：

因为特殊原因，如果您使用的是新浪邮箱，暂时是无法收到激活/通知/重置密码等邮件。不过，我们工作人员正在尽快处理该问题，敬请谅解！

怎么更改邮箱

怎么修改账户邮箱？？

1. 如果你的账号已经激活，请到个人中心修改邮箱，地址为：修改账户(http://git.oschina.net/profile/account)

2. 如果未激活，请发邮件到 git#git.oschina.cn申请或者联系码云管理员，在您提供必要的证明信息后，管理员可以协助您修改注册邮箱。

为什么我的网络连接不上码云？

码云为了扛住各种攻击，保证服务正常可用，在真实 IP 前面加了一层高防防护（电信联通单线高防），正是因为加了这层高防防护，因为高仿的链路质量不是特别好，所以会出现有些用户连不上码云的情况，主要有下面几种情况

- 打不开 https 链接
- Git 客户端操作码云链接各种超时 Timeout
- Ping 不通
- 用 https 操作出现can not connect to git.oschina.net:443 No Error

可以通过映射 host 到120.55.226.24来解决这个问题，这是 BGP 高防的 IP

有些用户可能会问为什么我们不把域名直接解析到120.55.226.24，这是因为 BGP 线路的防护流量比较小，如果超过防护上限，那么就会进入黑洞，所有用户都会受影响，而单线高防可防御流量上百 G，所以为了大家的正常使用才这么考虑，目前我们也正在找寻更可靠的防护方案，不论是从防护上还是链路质量上，谢谢大家的支持和理解！

为什么我突然无发访问/登录码云

从2015年6月以来，码云平台一直在遭受攻击，所以码云平台为了提供正常的服务，使用了第三方的防护方案，如高防 ip 等防护手段，所以，如果您发现您无法登录或访问码云，请检查：

• 能否打开网页

• 能否登录

• 能否推送或拉取

以上故障出现的可能情况为：

• 1.码云平台更换了 ip 地址

• 2.码云平台遭受了攻击导致无法访问

• 3.码云平台封锁了您的账号密码

• 4.您的 dns 存在 dns 解析劫持

• 5.码云平台正在维护（一般是周末凌晨）

• 6.您的 ip 地址被第三方防护识别为恶意用户而封禁

• 7.您的 ip 地址过于频繁访问码云平台导致 ip 地址被封禁

• 8.您的项目因为违反码云平台使用条款而屏蔽

• 9.您的账号因为散播不合法信息而遭到封禁

• 10.您的账号或项目因为上级主管政府部门要求而封禁

以上故障的的解决办法为：

• 遇上问题1,4的解决办法为清理自己的 dns 缓存然后重新解析码云平台的 ip 地址

• 遇上问题2,5的解决办法为等待一阵时间，码云平台处理完攻击或完成维护后重新开放即可

• 遇上问题3,8,9的解决办法为主动删除所有不合法信息后重新联系码云的管理员，当账号无法登录时可通过开源中国社区联系码云管理员代为处理

• 遇上问题10的解决办法为没有解决办法，请更换邮箱后重新注册

• 遇上问题6,7的解决办法为联系码云管理员为您解封 ip 地址。一般情况下2小时内可以解决，但如封禁为第三方安全防护封禁，则时间需要得久一点

QQ登录的如何修改密码？

QQ登录的会绑定一个邮箱，通过邮箱修改密码！

代码仓库

为什么大文件推不上去

对于普通用户码云单个仓库限制为1G，单个文件限制 100M。如果超过限制，您的代码将无法推送，我们提供的只是代码托管平台，

不是网盘，二进制文件请勿存放在码云，我们会不定期执行仓库大小扫描，如果发现你您的仓库超过大小超过1G，我们会联系您处理，

如果您15个工作日内没有处理，我们将暂停您的仓库的访问，请注意，我们只是暂停您仓库的访问

，并不是删除您的仓库，

如果您超过半年没有处理，我们将永久屏蔽该仓库的访问.

如果用户仓库提交超过 1GB，请在本地减小仓库体积后清空远程仓库，重新推送到远程服务器即可.这一策略适用与大文件.

企业用户能够支持更大体积的仓库。

配置SSH公钥

Windows下由于SSH配置文件的不匹配，导致的Permission denied (publickey)及其解决方法

很多情况下，Windows 平台由于不原生支持 ssh，只能使用如 git bash openssh，putty 等等工具或连接了不同的平台比如同时连接了码云和 github 等等会因为密钥文件储存位置不一致或者各个工具生成了自己的密钥而导致在连接码云的时候出现 Permission denied (publickey)，当遇上这种问题是一般有两种解决办法，一种是只使用一种工具，这样就能保证密钥位置不会变动，然后在密钥的文件夹下创建 config 文件，然后填写如下内容：

```
Host git.oschina.net
   HostName git.oschina.net
   User git
   IdentityFile ~/.ssh/id_rsa
   IdentitiesOnly yes
```

注意：以上参数中 id*rsa 这一栏请填写绝对地址，并且这一栏指定的私钥文件名字不一定要是 id* rsa，也可以是别的文件名，这样就能保证使用 ssh 连接码云时使用的是指定的密钥而不会被干扰，但请注意，使用这种方法是请保证该密钥不会被覆盖

另一种方法是只使用一个密钥，所有平台均使用同一个密钥，这样就不会出现密钥不匹配的情况。

为什么我添加了公钥后仍然无法推送代码

添加了公钥后仍然无法推送代码原因有一下几点：

- 您本身无权限推送代码
- 您电脑上使用的私钥与您添加的公钥不匹配
- 您添加的是部署公钥不是用户公钥

需要说明的是：公私钥不匹配是很常见的问题，特别是 windows 下，遇上公钥不匹配的问题，请重新添加匹配的公钥。

部署公钥仅仅提供拉取代码的权限，不提供推送代码的权限，故如果是您添加成部署公钥而导致无法推送，请删除该部署公钥后重新添加到个人中心的公钥中。

代码质量管理

我该如何开启代码质量分析

目前码云平台自行搭建了 sonar 代码质量分析平台以及腾讯优测安卓源码分析平台，其中，sonar 支持代码级别的分析，可以对每一行代码进行分析，腾讯优测平台则可以对安卓项目在各个平台运行的效果进行具体测试，以便于适配各种型号的设备。

sonar 代码质量分析

在质量分析页面选择 sonar，如图：

图片地址：https://static.oschina.net/uploads/img/201605/03152727_Q1Jn.png

然后点击分析此项目的代码，在弹出来的对话框中填写相应的参数，如图：

图片地址：https://static.oschina.net/uploads/img/201605/03152844_sbDG.png

请注意：参数不要填写错误，不然无法正确分析。

当你看到如下图所示时，则证明分析任务已经加入后台队列，你可以去喝杯咖啡或者下午茶，然后回来查看结果

图片地址：https://static.oschina.net/uploads/img/201605/03153051_POOa.png

需要说明的是，分析结果以内嵌网页的形式嵌入码云平台，故如果觉得码云平台页面布局不太符合您的要求，您可以自行通过查看网页源码提取出页面地址然后自行打开查看

腾讯优测

腾讯优测的开启方法和 sonar 是一样的，故开启方法请参开 sonar

在提交腾讯优测大约3分钟到半小时后，您的项目分析结果便会已报告的形式呈现给您，报告中会详细的列出您的应用可能遇上的问题，是什么机型，如图：

图片地址：https://static.oschina.net/uploads/img/201605/03153506_k4GC.png

您可以点击具体的项目查看具体的问题，以便于您定位您的问题。需要说明的是，优测仅仅支持安卓平台的项目，并不支持 iOS 或者其他平台，请勿测试其他平台的应用。

Webhook

我该如何利用webhook搭建自动化部署/测试平台

码云平台提供了 webhook 触发钩子,定义了5种触发方法,我们可以根据 webhook 的回调来搭建自动化部署/测试平台,下面给出示例(注:以下给出的是伪代码,并不能实际运行,仅作参考):

首先,我们明确自己的需求:搭建一个自动化部署平台,其需求如下:

- 能做到自动拉取代码
- 自动编译
- 自动更新数据库表结构
- 只更新master分支

这是一个最简单的自动化部署平台需求，下面就来看看怎么实现，需要说明的是,因为是最简单的方案，故没有考虑分布式的架构。

我们来做功能拆分，拆分如下：

代码同步模块:

- 正常代码拉取
- 强制推送代码拉取
- 忽略其他分支的代码推送

编译模块

- 正常编译
- 异常编译回滚

部署模块

- 编译文件自动部署
- 重启服务端
- 重启代理工具

日志模块：

- 正常情况下日志输出
- 代码拉取失败日志输出
- 编译错误日志输出
- 服务端重启日志输出

现在功能需求定好了，开始进行代码编写，因为这是一个最简单的部署，所以数据量非常小，用文本文件足以，故这里不采用数据来增加系统复杂程度

首先，我们我们要写一个接口，以便于接收由 webhook 发出的数据，由于 webhook 的数据发送方式是 post，故需要一个接收 post 数据的地址。例如：

http://xxx.xxx.com/xxx/xxx/git/post.php 这样一个地址，当然，这个 php 文件具体怎么写就不知道了哈，他的作用只有一条：调用执行预定脚本： auto.sh 并将收到的参数传递给脚本。

然后我们剩下的工作就是写 auto.sh 这个脚本了，我们所有的工作都需要在 auto.sh 这个脚本上完成。

首先，我们来搞定自动拉取代码

```
git pull origin master
```

这是正常拉取代码的，但是这条命令需要拉取的分支不存在强制推送状态，故这条命令适用性不够广泛，所以我们要改造一下，改成下面的样子：

```
git reset --hard <commit id>
git pull origin master
puts “时间 拉取代码成功，分支为master” >> "pull.txt"
```

然后我们就搞定了代码拉取的问题，设定好 commit id 任你分支怎么变化怎么强推我都能正常拉取代码。为了正常判断状态，我们将这段代码新建个脚本，命名为 pull_code.sh 然后交给 auto.sh 调

用，这样我们就可以在 auto 中写上判断条件来判断脚本执行成功还是失败，这样便于日志输出

实际上，这里只做到啦拉取 master 分支的代码，我们可以将这段代码改改，将分支名作为参数，然后加上 checkout 语句，就可以拉取任意分支了，比如这样：

```
git checkout 分支名（参数，外部传递）
git reset --hard <commit id>
git pull origin  分支名
```

至于 commit id 从配置文件中读出来就行了，然后就变成了可以部署其他分支的代码了。到这里，拉取代码的部分就算完成了。

接下来是自动编译部分，同样，基于扩展性需求，我们可以将它做成一个脚本，已编译 rails 应用为例，我们一般需要编译资源执行如下命令：

```
bundle exec rake assets:precompile RAILS_ENV=production
```

所以我们将这条命令写成脚本，命名为：compile.sh 同样，编译部分就完成了，其他语言按照其他语言格式将执行的命令写入到脚本中去，然后等待 auto.sh 的调用即可

部署模块同样的操作手法，将需要执行的命令一条一条写入脚本中，等待被调用

然后我们开始写最重要主文件，也就是 auto.sh 脚本（注：以下是伪代码，不具有执行能力，仅提供思路参考）

以下是伪代码：

开始接受参数：params

解析参数，需要：分支信息 commit信息

根据分支信息和 commit 信息判断是否需要更新，如果无需更新，结束

需要更新，开始切换用户，获取更新权限，写入参数到文本文件中，避免因切换参数导致参数丢失

从文本文件中读取参数，调用 pull_code.sh 脚本，同时传递分支参数

接收 pull_code.sh 脚本执行结果，开始根据执行结果打印日志，同时保存执行结果到变量 code

根据变量 code 的值判断代码拉取是否成功，判断是否需要编译资源，同时打印日志

调用部署脚本，开始执行部署操作

日志部分开始工作，对每次执行结果进行记录

汇总日志部分，开始分析是否可以重新启动服务（热启动或其他）

开始清理除日志之外的中间文件，清理工作目录

执行完成

以上就是一个最简单的利用 webhook 进行自动化部署的案例，其原理是先用钩子触发回调，拿到更新信息，然后调用预先定义好的自动化部署脚本进行部署，最后完成操作

SVN 支持

我该如何使用SVN管理我的项目

现在 **码云** 目前支持使用 Subversion 对仓库进行操作，以下是使用指南和注意事项。

使用前注意

1. 仓库体积超过 300 MB 不建议使用 Subversion 操作仓库，存储库容量达到 400 MB，或者 300 MB 并且存储大量非文本数据时，我们将关闭仓库的 Subversion 支持。

2. 由于 GIT 不支持空目录的提交，在存储机器上，无论是普通仓库还是开启 Subversion 接入的仓库存储时都是 GIT 仓库，Subversion 的 commit 是提交到 git 仓库上的，所以码云的 Subversion 不支持空目录的提交。

3. 第一次开启 Subversion， 操作一个仓库，如果仓库体积较大或者提交次数较多，由于缓存的缘故，响应时间会比较长。

4. 不支持 Subversion 的 Hook 机制，请使用 WebHook 替代。

5. Subversion 属性不完全支持。

6. 客户端需要开启 SASL 支持，不支持的客户端无法访问。

7. 部分 svn 命令不支持。可以查看 **Subversion 客户端的兼容性**

8. 版本号的映射，目前 Subversion 的版本号计算依据为本分支所有的 commit 数目减一 不包括 merge ，如果使用了在 git 中强制回退等操作，请重新检出。

WARNING:

> 由于 git 在设计上就没有考虑空文件 Kernel.org: Git FAQ(https://git.wiki.kernel.org/index.php/GitFaq#Can_I_add_empty_directories.3F)
>
> 我们设计的原则就是不破坏，不主动修改用户的仓库，我们的后端存储的完全是一个 git 仓库，如果我们添加了，一次提交内容也不会一致了，建议你在添加目录的时候添加 .keep 之类的占位文件，空文件即可。
>
> Git 与 SVN 混用时尽量不要使用 Git 强制推送。Git 与 SVN 混用注意事项 (https://git.mydoc.io/?t=122635)

关于改版

Subversion 功能的最终解释权归 OSChina.NET 所有。Subversion 接入的规则可能在下一次改版中发生改变。

开启方式

在项目的设置界面开启

图片地址：http://git.oschina.net/uploads/images/2015/0324/182035_d23abb99_62561.png

如果是空仓库：

图片地址：https://static.oschina.net/uploads/img/201506/16185355_IYvK.png

使用指南

码云 支持的是 svn 协议。 对于 svn 而言，获取一个仓库的代码通常是 checkout，在项目主页我们通常可以获得 URL：

图片地址：http://static.oschina.net/uploads/space/2015/0318/152207_DRMJ_139664.png

这个仓库地址为：

```
svn://git.net/svnserver/newos
```

1.获取仓库代码：

```
svn checkout svn://git.net/svnserver/newos newos
```

注意信息 码云的 SVN 接入后端是通过 git 存储库实现，URL 规则为 svn://域名/用户名/项目名 .

使用上述命令，我们将得到项目默认分支的代码。并将本地的工作目录命名为 *newos*

如果最后不带 newos，svn 默认把本地工作目录命名为 SVN-URL 的最后一个目录名，这里是 *newos*

```
svn checkout svn://git.net/svnserver/newos
```

如果要获得任意分支代码，请输入近似如下的命令：

```
svn checkout svn://git.net/svnserver/newos/branches/update newosupdate
```

特别的说明，获取主干分支，也就是 master 分支可以使用下面的分支格式

```
svn checkout svn://git.net/svnserver/newos/trunk newos
```

svn trunk 分支对应 master 分支 用户应当尽量不使用下面格式

```
svn checkout svn://git.net/svnserver/newos
```

操作说明

如果部分检出仓库，并且仓库根目录下包含 branches/tags/trunk 这样的目录，请使用完整的路径 layout，如下：

```
svn://git.net/example/example/trunk/tags/hello
svn://git.net/example/example/branches/dev/trunk
svn://git.net/example/example/branches/dev/branches
```

如果没有 master 分支，也就没有 trunk 分支，检出的 URL 不能省略分支名。比如只有一个 dev 分支,必须使用下列格式，否则会提示仓库不存在。

```
svn co svn://git.net/svnserver/newos/branches/dev  svnserver_dev
```

打开终端,输入上述命令，出现以下下面提示。其中第一个认证领域是用户的密码，这个可以留空。而用户名是用户在码云（Gitee.com）登陆时使用邮箱地址。密码则是用户登陆 码云 所使用的密码。

一般而言，svn 会加密缓存用户的用户名密码，所以，对仓库的操作只需要第一次输入用户邮箱和密码。

清除密码缓存，用户目录下的 .subversion/auth/svn.simple 文件夹下的文件。

图片地址：http://static.oschina.net/uploads/space/2015/0318/153828_ptt4_139664.png

下图则是成功的拉取了项目代码。

图片地址：http://static.oschina.net/uploads/space/2015/0318/154033_aySS_139664.png

查看本地工作目录信息：

```
svn info
```

图片地址：http://static.oschina.net/uploads/img/201503/18161038_vF3E.png

```
cd helloworld
echo "#SVN Options ReadMe.md">SVNReadMe.md
#svn add SVNReadMe.md
#svn add * --force类似于git add -A
svn add * --force
svn update .
svn commit -m "first svn commit"
```

Subversion 在提交前建议先使用 svn update 更新工作拷贝。也就相当于 git pull 后再 git push。

Subversion 的提交是在线的，如果机器已经离线，那么提交会失败，这个过程用git的方式理解就是 git commit+git push。

用户使用 svn 提交代码同样会有动态显示。

列出版本库中的目录内容:

```
svn list svn://git.net/svnserver/newos/trunk
```

导出仓库指定分支的所有文件，不含版本控制信息：

```
svn export svn://git.net/svnserver/newos/trunk newos
```

备注

安装 Subversion 客户端

在 Apache 基金会的 Subversion 官网：

http://subversion.apache.org(http://subversion.apache.org)

二进制下载提示页面：

http://subversion.apache.org/packages.html(http://subversion.apache.org/packages.html)

Windows 系统：

与资源管理起集成的 SVN 客户端：TortoiseSVN(http://tortoisesvn.net/downloads.html),通常被叫做"海龟"，为 msi 安装包。可以使用 ExtractMSI(http://pan.baidu.com/s/1szHIn) 解压缩。

很诡异的是，在 Apache 上并没有推荐 TortoiseSVN。

另外还有 SlikSVN，下载地址
：https://sliksvn.com/download/(https://sliksvn.com/download/)

其他的也就不一一介绍了。

Linux 系统

一般而言 Linux 系统自带的包控制软件能够安装 Subversion，如果版本低于1.8，就建议用户下载预编译的二进制或者自己动手编译 Subversion。这里不做过多说明。

OS X

XCode 自带的 Subversion 版本为1.7.x，太老，而码云（Gitee.com）只支持1.8以上的 SVN 客户端。

如果安装了 Homebrew

```
brew install subversion
```

或者使用WANdisco的预编译版本

http://www.wandisco.com/subversion/download#osx(http://www.wandisco.com/subversion/download#osx)

Subversion 客户端的兼容性

我们支持 Apache Subversion 1.8 或者更高的版本，当你安装一个 Subversion 客户端时，如果错误提示是“无法协商验证验证方式” 请确保你的客户端支持 SASL 验证，比如在 Ubuntu 上，你可以安装 libsasl2-dev 然后编译 Subversion, 这样的话客户端是支持 SASL 验证的。

> sudo apt-get install libsasl2-dev

当你使用 svnkit 或者 SubversionJavaHl 这类 IDE 集成客户端，请确保支持 SASL 验证。

关于 GIT 与 SVN 的转换

如果用户存在一个基于 Subversion 托管的项目，要迁移到码云（Gitee.com），可以使用 git-svn 将项目转变为基于 git 的仓库，然后推送到码云（Gitee.com），这样你依然能够使用SVN对项目进行操作。请记得先在码云（Gitee.com）上新建一个项目

```
git svn clone http://myhost/repo -T trunk -b branches -t tags
git remote add oscgit https://git.oschina.net/user/repo
git push -u oscgit --all
```

通常来说，如果本地存在 SVN 仓库，则可以：

```
git svn clone file:///tmp/svn-repo -T trunk -b branches -t tags
git remote add oscgit https://git.oschina.net/user/repo
```

```
git push -u oscgit  --all
```

将项目转移到 码云（Gitee.com）上以后，使用 svn 命令 checkout 即可对项目进行操作。

高级指南：

http://git-scm.com/book/zh/ch8-2.html(http://git-scm.com/book/zh/ch8-2.html)

安装 git，git-svn

Windows

msysgit 官网 http://msysgit.github.io/(http://msysgit.github.io/),版本比较低。

Github for Windows 提供的 git 工具和 msysgit 一致。

MSYS2 git 下载地址: http://sourceforge.net/projects/msys2(http://sourceforge.net/projects/msys2)，然后启动终端,安装 git，目前版本为2.4.3。

```
pacman -S git
```

Cygwin git 下载地址: http://www.cygwin.com/(http://www.cygwin.com/),然后使用包管理软件或者直接下载 git 源码编译 git。

```
make configure
./configure --prefix=/usr/local
make
make install
```

Linux

有包管理器的直接用包管理器安装。

如 Ubuntu

```
sudo apt-get install git git-svn
```

也可以手动编译。

Mac OSX

下载地址：http://git-scm.com/download/mac(http://git-scm.com/download/mac)

我使用SVN管理我的项目但总是提示输入用户名密码

首先请确认： 1.SVN客户端版本在1.8以上 2.码云的用户名不带有@符号

其次： 使用的账号密码是码云平台的账号密码，并不是社区的账号密码，请确认密码没有使用错误

然后：确认密码不包含特殊字符，如@！#￥%%之类的特殊符号

如果以上三点均达到，请联系管理员或开发人员并请您准备好日志以便于我们开发人员查看分析问题

SVN 常见错误

Repository size is too large

仓库体积超过 300 MB 不建议使用 Subversion 操作仓库，存储库容量达到 400 MB，或者 300 MB 并且存储大量非文本数据时，我们将关闭仓库的 Subversion 支持。

解决方案 使用 Git，或清空项目，重新推送。

Empty directories is not supported

我们SVN管理项目是通过兼容来实现的，所以我们后端仓库仍然是一个git仓库，但是git不允许存在空仓库，故使用SVN管理项目时，使用空文件夹会报错，如果您一定要使用空文件夹，

请您在需要保留的空文件夹内新建一个.keep文件，这样git就会保存这个文件夹。

另外，如果您新建.keep文件后仍然无法推送，请先清空SVN的缓存，然后再次推送。

Can't find node

如果出现了 **Can't find node** 这样的错误，请检查是否是通过默认存储库地址检出的项目，

即：

> svn co svn://example.com/example/example

这样检出的存储库，不能建名字为 trunk， branches,tags 的目录，当然如果是强制指定了分支或者 trunk，

如：

> svn co svn://example.com/example/example/trunk exmaple
>
> svn co svn co svn://example.com/example/example/branches/dev dev

这样检出的存储库可以在根目录下创建 trunk,tags branches 这样名称的目录。

在子目录创建则无影响。

svn://example.com/example/example 这样的路径实际上是检出的 **svn://example.com/example/example/branches/$(default branch)**

此问题只需要检出时强制指定分支名即可。

其他

TBD

Git 与 SVN 混用注意事项

[Deprecated]

已经修复

Git 和 SVN 混合使用时，尽量不要使用 Git 的强制推送。Git 强制推送时，前一次 commit 将会被覆盖，同时，与该 commit 对应的 SVN 版本缺失，造成 SVN 更新代码失败。

具体情况如下：

当 Git 进行强制推送后，SVN 尝试更新代码:

```
$ svn update .
Updating '.':
svn: E200022: svn: E200022: Git Force push miss some object,Please checkout again !
```

如果再次尝试更新，SVN 会进行版本回退：

```
$ svn update .
Updating '.':
svn: E160006: svn: E160006: No such revision 4
$ svn update .
Updating '.':
At revision 3.
```

注意，此时回退是使用本地的数据，该版本是被强制推送覆盖掉的版本！与服务器数据已经不一致！！

此时需要将本地仓库目录删除，然后重新 checkout。

项目演示

我该选择哪种paas平台

目前我们码云平台提供三种paas服务以供您选择,起特点如下

• 百度BAE:

支持java PHP nodejs python .提供数据库mysql mongodb redis 需要注意的是,百度提供急速收录服务,即使用百度的paas内容会非常快速的被百度收录.关于收费:百度BAE是按天收费,具体收费详情为:每64M内存大约为0.1元每天

• Mopaas:

支持Go Python PHP Tomcat JBoss NodeJS Ruby Websphere8 .Net,服务提供Mysql Redis,Memcached,MongoDB,PostgreSQL等等几十种第三方服务.关于收费: 以代金券的方式提供,可以以256M内存运行应用约半年(注:内存指该用户名下总共使用的内存)

• 灵雀云:

提供docker服务,是国内首家真正服务于开发者的docker云平台,提供一年的免费期,每个账户可以创建两个docker(编撰者注:docker大家都懂的,什么都能干,所以就不写具体能干什么了)

以上就是我们码云平台暂时支持的演示服务,各位开发者可以根据自己的需求灵活的选择一个或多个部署自己的演示。

为什么我的paas演示平台打开比较慢

因为我们的 paas 是第三方服务，我们是通过调用接口来获取数据的，因地区差异以及运营商的差异，您的网络速度以及质量可能不太理想，故获取数据的时间可能会稍微长一点，这样就造成了页面打开慢。

为什么我的mopaas演示无法启动

因为 mopaas 的限制，新建的演示必须在部署代码后才能启动，并且，所有的项目必须存在标志性的文件才可以启动成功，比如 php 项目必须存在 index.php，rails 项目必须存在 Gemfile 等等。

为什么我的MoPaaS的项目演示部署失败

根据 mopaas 的规则，部署响应项目必须有项目的该项目类型的标识，比如 php 项目就必须根目录下存在 index.php 文件才可以部署成功，注:Mopaas 已经不再支持静态页面，如果需要部署静态页面，请等待我们即将推出的 pages 服务。

项目管理

如何修改我的git默认的编辑器

nano 是一个字符终端的文本编辑器，有点像 DOS 下的 editor 程序。它比 vi/vim 要简单得多，比较适合 Linux 初学者使用，git 默认的编辑器就是 nano 不过，有些人习惯了使用 vi 或者其他编辑器，如果想要改换默认编辑器的话只需要在仓库目录下执行下面命令就可以了

```
git config --global core.editor vim
```

注意:vim 可以替换成你喜欢的编辑器，比如 emacs。

我该如何才能使我的组织中的成员访问、管理组织中的项目

首先，在码云平台，组织功能设计上只是一堆人员与项目的集合，成员与项目中间并没有直接的权限关系，故组织成员不可以直接管理项目，不对项目具有直接的管理权限，所以，要想让组织成员能访问、管理您组织的中项目，您需要将该成员添加到项目成员中。具体的您可以这样做：

首先点击您的组织中的那个项目，如图：

图片地址：https://static.oschina.net/uploads/img/201605/18111502_0YFX.png

然后点击管理，如图：

图片地址：https://static.oschina.net/uploads/img/201605/18111538_aZH5.png

点击下图所示的任意一个条目：

图片地址：https://static.oschina.net/uploads/img/201605/18111704_RbIm.png

然后点击从组织中添加成员

图片地址：https://static.oschina.net/uploads/img/201605/18111746_7KXj.png

你就能看到下图类似的界面：

图片地址：https://static.oschina.net/uploads/img/201605/18112206_YcFu.png

然后选择好成员，选择好权限，点击添加用户，该用户就可以访问或管理这个组织中的项目了。

为什么有的项目不能提交Issue、Pull Request

> 项目的管理员可以设置项目是否接受 Issue 和 Pull Request，如果该项目设置了不接受 Issue 或者 Pull Request，则您不能提交 Issue 或 Pull Request。

开源指南

开源指南

一、如何贡献于开源项目

1、如何向开源软件贡献自己的力量 (http://www.oschina.net/translate/contribute-to-opensource)

> “我想向开源社区贡献源码，但是该怎么做呢？”我经常看见很多很多学生在很多论坛里问这个问题。有很多种方式可以贡献源码，我列出了很多，希望能够对你们有多帮助。

2、不用写代码，你也能为开源作出贡献(https://my.oschina.net/editorial-story/blog/837992)

> 开源项目在国外已经成为了一股热潮，已经开始影响到日常生活的方方面面，可是在中国，开源项目的使用者不少，贡献者却寥寥无几，但同时有很多人想要参加开源项目，却总是不得其门而入……无论你是久经沙场的编程老手还是初入门道的技术小白，或者压根就不是一个程序猿，都有不写代码而为开源项目作出贡献的方法。

3、开源项目贡献者行为准则(https://my.oschina.net/ljzn/blog/732450)

> 作为该项目的贡献者和维护者，我们希望培养一个开放，友善的社区，我们承诺:尊敬任何通过报告提案，发布特性需求，更新文档，提交pull请求或补丁，以及其它任何活动来进行贡献的人.

二、如何开启开源项目

1、如何开始一个新的开源项目(https://www.oschina.net/translate/starting-open-source-project?lang=chs&page=1#)

> 这一段时间， 好像很多人写了一代代码加上一段开源软件协议， 再把它发布到GitHub， 然后就说: "我把它开源了". 创建一个开源项目并不仅仅是让你的代码可以自由的被访问获取。
>
> 如何开始呢？

2、向开源社区贡献模块的经验分享(https://www.oschina.net/question/554557_120351)

>

> 2007年的时候，我曾经在CPAN上发布了几个Perl模块，至今我的名字一直是Perl模块作者中几十个中文名字的其中一个，在Google中搜索我的名字，CPAN上的页面也一直有较高的关联度。
>
> 向开源社区贡献模块是一个熟悉国外软件开发流程，完善代码规范的过程，对于把自己训练成更专业的软件开发者很有帮助。

3、我如何在OSC上做开源项目(https://my.oschina.net/u/134395/blog/523214)

> - 在 OSC 上如何做开源项目和利用开源项目赚钱？
>
> - 如何做开源软件？
>
> - 大家都明白，所以重点是在如何利用开源软件赚钱。

4、寻找 TODOs：向开源项目贡献的一些建议(https://www.oschina.net/news/26465/find-the-todos-in-opensource-project)

> TODO 和 Bug 不一样。Bug一般是由用户提交的，外部可见的一些问题。而TODO一般是开发者自己标记的，代码中可以改进的地方，由于时间关系没有来得及做。

三、为开源项目寻找参与者

1、吸引学生为你的开源项目贡献代码的九招秘诀(http://os.51cto.com/art/201509/491366.htm)

四、建立开源欢迎社区

1、如何建立开源社区(http://www.kaiyuanshe.cn/article/57.html)

> 社区就是有共同兴趣的一群人。开源项目和闭源项目都有用户社区，大部分用户不会积极地与社区其他成员互动。而另一方面，无论是开源社区还是闭源社区，都会有成员愿意更积极地参与，例如，报告 bug、帮助其他用户、撰写文档或进行推广。

五、如何做好开源项目的维护者

1、怎样维护成功的开源项目(http://www.csdn.net/article/2013-07-03/2816085-How-to-maintain-a-successful-open-source-project)

> 开源项目和普通产品一样，想要取得成功并非易事。流行的开源 Python 库 urllib3 的作者 Andrey Petrov 是总结了自己五年来的经验，写好介绍下文字、对待用户和贡献者的态度、合作和营销意识都至关重要。

2、成为流行开源项目的维护者 你是如何一步步进“坑”的 ？(https://www.sdk.cn/news/4066)

> 我们都清楚，开源的价值越来越高。我谈到了作为一名开源维护人员，你将会经历的各种情绪，但是对于你的职业生涯来说，维护开源项目将会给你带来巨大的回报。

六、如何领导和治理开源社区

1、自由开源社群治理之道(https://linux.cn/article-7436-1.html)

注：以上内容为文章作者

Git 入门篇

Git 是什么

Git 是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。因为在开发过程中需要跟踪代码，文档，项目等信息中的变化，版本控制则成为项目管理中的重中之重，且它对跨平台时遇到的一些问题有很不错的支持，所以在全球范围都广受欢迎。

Git 版本控制入门

Git 版本控制入门

不了解Git请查看权威Git书籍 ProGit（中文版）(http://git.oschina.net/progit/)。

一份很好的 Git 入门教程，点击这里查看(http://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000/001373962845513aefd77a99f4145f0a2c7a7ca057e7570000)。

Git客户端下载地址： 官方Git(http://git-scm.com/downloads) - TortoiseGit(http://tortoisegit.org/download/) - SourceTree(https://www.sourcetreeapp.com/)

Git 常用命令与名词解释

仓库

在 Git 的概念中，仓库，就是你存在.git目录的那个文件夹内的所有文件，包括隐藏的文件，Git程序会再当前目录以及上级目录查找是否存在.git文件，如果存在，则会将.git目录存在的文件夹开始下的所有文件当成你需要管理的文件，所以，我们如果想将某个文件夹当做一个Git仓库，你可以在那个文件夹下通过终端(Window为Cmd或者PoewrShell或者Bash)来执行

```
git init
```

这样，你所期望的那个文件夹就成为了一个Git管理的仓库了

版本

严格来讲，Git并不存在版本的概念，但人们也硬是发展出了这么个玩意，在Git中，计数基础是提交，即我们常说的Commit，我们每做一点更改便可以产生一次提交，当提交累计起来，可以作为产品定型时，就在当前的Commit上打上一个标记，将这个标记我们称之为版本多少多少，那么就算完成了一个版本，标记本身被称之为Tag，请注意，在Git中，版本仅仅只是某一个提交的标签，并没有其他意义，Git本身也仅有打标签的功能，并没有版本功能，版本功能是根据Tag来扩展的，Git本身并没有

分支

这是Git中最重要的也是最常用的概念和功能之一，分支功能解决了正在开发的版本与上线版本稳定性冲突的问题在Git使用过程中，我们的默认分支一般是Master，当然，这是可以修改的，我们在Master完成一次开发，生成了一个稳定版本，那么当我们需要添加新功能或者做修改时，只需要新建一个分支，然后在该分支上开发，完成后合并到主分支即可

提交

提交在Git中同样是非常重要的概念，Git对于版本的管理其实是对于提交的管理，在整个Git仓库中，代码存在的形式并不是一分一分的代码，而是一个一个的提交，Git使用四十个字节长度的16进制字符串来标识每一个提交，这基本保证了每一个提交的标识是唯一的，然后通过组织一个按照时间排序的提交列表，就组成了我们所说的分支，请注意，分支在本质上只是一个索引，所以，我们可以任意回退，修正，即使因为某些原因丢失了，也可以重建另外，关于Git的储存方式:Git是仅仅只储存有修改的部分，并不会储存整个文件，所以，请不要删除文件夹整个文件夹的内容，除非你确定你不再需要他，否则请勿删除

同步

同步，也可以称之为拉取，在Git中是非常频繁的操作，和SVN不同，Git的所有仓库之间是平等的，所以，为了保证代码一致性，尽可能的在每次操作前进行一次同步操作，具体的为在工作目录下执行如下命令:

```
git pull origin master
```

其中origin代表的是你远程的仓库，可以通过命令 git remote -v 查看，master是分支名，如果你本地是其他分支，请换成其他分支的名字，另，因为远程仓库与你本地仓库可能存在冲突，故当存在冲突时，请参考进阶篇的如何处理冲突

推送

和拉取一样，也是一个非常频繁的操作，当你代码有更新时，你需要更新到远程仓库，这个动作被称之为推送，执行的命令与拉取一样，只是将其中的pull这个单词改成push，同样，如果远程仓库存在你本地仓库没有的更新，则在推送前你需要先进行一次同步，如果你确定你不需要远程的更新，则在推送时加上 -f 选项，则可以强制推送，注:在协同开发中，我并不建议这么做，因为这样很可能覆盖别人的代码

推送代码示例:

```
git push origin master
```

强制推送代码示例:

```
git push origin master -f
```

冲突

在使用Git开发时，如果只是一个人使用，那么基本不会产生冲突，但是在多人合作开发的情况下，产生冲突是很正常的一件事情，关于如何处理冲突，请参考进阶篇的如何处理代码冲突(http://git.mydoc.io?v=16912&t=83148) 这一小节

合并

合并这个命令通常情况下是用于两个分支的合并，一般用于本地分支，远程分支多用Pull命令，该命令的功能是将待合并分支与目标分支合并在一起，注意，这个命令只会合并当前版本之前的差异，两个分支的提交历史会根据提交时间重新组织索引，故只可能会产生一次冲突但是会生成一个提交，如果你不想生成这次提交，加上 --base参数即可

暂存

这个既是一个概念也是一个命令，其含义就是字面上的，作用就是可以将你当前正在进行的工作暂时存起来，然后在此基础上干别的事情，等你别的事情干完后，再转回来继续，注意，暂存只是针对你最后一次改动而言，即针对当前所在的版本的所有改动都算具体执行命令为:

将当前改动暂存起来:

```
git stash
```

恢复最后一次暂存的改动

```
git stash pop
```

查看有多少暂存

```
git stash list
```

撤销

撤销命令使用是非常频繁的，因为某些原因，我们不再需要我们的改动或者新的改动有点问题，我们需要回退到某个版本，这时就需要用到撤销命令，或者说这个应该翻译成重置更加恰当。具体命令如下:

撤销当前的修改:

```
git reset --hard
```

请注意:以上命令会完全重置你的修改，如果你想保留某些文件，请使用checkout +文件路径 命令来逐一撤销修改

如果你想重置到某一版本，可以将 --hard 改为具体的Commit的id如:

```
git reset 1d7f5d89346
```

请注意，这时你的修改仍然存在，只是你的最近一次提交的版本号变成了你要重置的版本，如果说你想完全丢弃修改，只需要加上 --hard参数就可以

这里解释的只是一些工作中经常用到的git的基本概念，名词，并不包含Git的所有名词，概念解释，故本文未提到之处请自行使用搜索引擎搜索;另:文档编撰者尽可能的寻找标准的解释，因结果来自于互联网，如果解释有错漏之处，烦请指出并给出正确的解释，谢谢

另，本文仅提到部分常使用的概念以及命令，但Git远远不止这些东西，所以，在本文找不到的内容请自行百度，这里推荐 Git的官方文档(中文和英文均有，由官方以及志愿者维护):<http://git-scm.com/book/zh/v2> ，廖雪峰博客:<http://www.liaoxuefeng.com/wiki/001373951630592960dd18361248578c67b8067c8c017b000>

Git 安装

1.Git 的安装

Window 下的安装

从 http://git-scm.com/download 上下载window版的客户端，然后一直下一步下一步安装git即可，请注意，如果你不熟悉每个选项的意思，请保持默认的选项

Ubuntu 下安装

```
在终端下执行 apt-get install git
```

Centos/Redhat 安装

```
在终端下执行 yum install git
```

Fedora23 安装

```
在终端下执行 dnf install git 或者 yum install git
```

Fedora22/21 安装

```
在终端下执行 yum install git
```

SUSE/OPENSUSE安装

```
在终端下执行 sudo zypper install git
```

Mac OS X 安装

```
在终端下执行brew install git (注:请自行解决环境变量以及Brew工具的问题)
```

编译安装(注:仅适合非window系统)

从 https://github.com/git/git/releases 上选取一个版本下载，解压缩后进入到 Git 的目录然后依次执行以下代码:

```
make configure
./configure
make all
sudo make install
```

注意:如果遇上无法编译的问题，请自行通过搜索引擎来查找 Git 所需的依赖

如果以上一切正常，打开终端(Window下请打开安装git时一并安装的bash) 输入 git --version 应该会显示如下类似的信息

```
git version 2.5.0
```

Git 使用前的配置

配置自己的用户名和邮箱

在使用git前，我们需要告诉git自己是谁以及自己的邮箱是什么，所以我们需要对git进行一些基本设置。打开终端（Windows打开安装git时安装的git bash）执行如下命令

```
git config --global user.name "你的名字或昵称"
git config --global user.email "你的邮箱"
```

这样就配置好了这台电脑的git 如果您没有安装git bash或者使用其他工具，建议您查找该工具的设置教程，本文档建议使用命令行终端操作git，这样遇上的兼容性问题最小。

关于如何配置sshkey问题，请参阅ssh key相关问题(http://git.mydoc.io/?t=83157)

如何克隆一个项目？

以克隆项目git@git.oschina.net:zxzllyj/sample-project.git为例(注:本处使用的是ssh地址，因为演示机已经配置好ssh公钥，故可以使用ssh地址，如果您没有配置公钥，请使用https地址)

图片地址：https://static.oschina.net/uploads/img/201603/10160653_BHzv.gif

注：上图的方法虽然将仓库完整的拉取了下来，但是仅仅只会是显示默认分支，如果需要直接到指定的分支，可以在仓库地址后面加上分支名

Git 仓库的基本操作

1.修改仓库名

一般来讲，默认情况下，在执行clone或者其他操作时，仓库名都是 origin 如果说我们想给他改改名字，比如我不喜欢origin这个名字，想改为 oschina 那么就要在仓库目录下执行命令:

```
git remote rename origin oschina
```

这样 你的远程仓库名字就改成了oschina，同样，以后推送时执行的命令就不再是 git push origin master 而是 git push oschina master 拉取也是一样的

2.添加一个仓库

在不执行克隆操作时，如果想将一个远程仓库添加到本地的仓库中，可以执行

```
git remote add origin  仓库地址
```

注意: 1.origin是你的仓库的别名 可以随便改，但请务必不要与已有的仓库别名冲突 2. 仓库地址一般来讲支持 http/https/ssh/git协议，其他协议地址请勿添加

3.查看当前仓库对应的远程仓库地址

```
git remote -v
```

这条命令能显示你当前仓库中已经添加了的仓库名和对应的仓库地址，通常来讲，会有两条一模一样的记录，分别是fetch和push，其中fetch是用来从远程同步 push是用来推送到远程

Git 进阶篇

如何处理代码冲突

冲突合并一般是因为自己的本地做的提交和服务器上的提交有差异，并且这些差异中的文件改动，Git不能自动合并，那么就需要用户手动进行合并

如我这边执行git pull origin master

如果Git能够自动合并，那么过程看起来是这样的

图片地址：http://git.oschina.net/uploads/images/2016/0226/113507_cca8cd22_62561.gif

拉取的时候，Git自动合并，并产生了一次提交。

如果Git不能够自动合并，那么会提示

图片地址：http://git.oschina.net/uploads/images/2016/0226/113621_dbc985b5_62561.png

这个时候我们就可以知道README.MD有冲突，需要我们手动解决，修改README.MD解决冲突

图片地址：http://git.oschina.net/uploads/images/2016/0226/113823_fffe18cf_62561.png

可以看出来，在1+1=几的这行代码上产生了冲突，解决冲突的目标是保留期望存在的代码，这里保留1+1=2，然后保存退出。

图片地址：http://git.oschina.net/uploads/images/2016/0226/114159_426b8d65_62561.png

退出之后，确保所有的冲突都得以解决，然后就可以使用

```
git add .
git commit -m "fixed conflicts"
git push origin master`
```

即可完成一次冲突的合并。

整个过程看起来是这样的

图片地址：http://git.oschina.net/uploads/images/2016/0226/114058_429e8b54_62561.gif

如何进行版本回退

版本回退有多种方式,下面一一演示:

回退到当前版本(放弃所有修改)

图片地址：https://static.oschina.net/uploads/img/201603/10161457_lf5m.gif

放弃某一个文件的修改

图片地址：https://static.oschina.net/uploads/img/201603/10161707_dstz.gif

回退到某一版本但保存自该版本起的修改

图片地址：https://static.oschina.net/uploads/img/201603/10162127_dLHO.gif

回退到某一版本并且放弃所有的修改

图片地址：https://static.oschina.net/uploads/img/201603/10162634_CKmm.gif

回退远程仓库的版本

先在本地切换到远程仓库要回退的分支对应的本地分支，然后本地回退至你需要的版本，然后执行：

```
git push <仓库名> <分支名> -f
```

如何以当前版本为基础，回退指定个commit

首先，确认你当前的版本需要回退多少个版本，然后计算出你要回退的版本数量，执行如下命令

```
git reset HEAD~X //X代表你要回退的版本数量，是数字！！！！
```

需要注意的是，如果你是合并过分支，那么背合并分支带过来的commit并不会被计入回退数量中，而是只计算一个，所以如果需要一次回退多个commit，不建议使用这种方法

如何回退到和远程版本一样

有时候，当发生错误修改需要放弃全部修改时，可以以远程分支作为回退点退回到与远程分支一样的地方，执行的命令如下

```
git reset --hard origin/master // origin代表你远程仓库的名字，master代表分支名
```

如何进行分支合并

分支合并分为两种情况一种是本地分支合并，一种是远程分支合并到本地分支,下面,分别用GIF动画演示

本地合并分支:

图片地址：https://static.oschina.net/uploads/img/201603/11110502_Puw4.gif

远程分支合并

图片地址：https://static.oschina.net/uploads/img/201603/11105933_WdrB.gif

如何从众多提交中保留只需要的提交

如果说在众多提交中,已某个提交为基准,只保留上游众多提交中的某个或者某几个,可以使用 cherry-

pick命令,具体是:

```
git cherry-pick <commit id>
```

如果没有冲突,则回显示如下:

```
Finished one cherry-pick.
# On branch dev
# Your branch is ahead of 'origin/dev' by 3 commits.
```

如果存在冲突,则需要解决冲突然后继续,关于如何冲突,请查看如何处理代码冲突小节

如何进行减少提交历史数量以及修改自己的commit中的邮箱

注:本节中内容来自 https://git-scm.com/book/zh/v2/Git-工具-重写历史 最终解释权归该页面编撰者所有,本页面仅引用以及对内容进行一定的排版,本文档编撰者对本页面内容无版权

许多时候，在使用 Git 时，可能会因为某些原因想要修正提交历史。 Git 很棒的一点是它允许你在最后时刻做决定。 你可以在将暂存区内容提交前决定哪些文件进入提交，可以通过 stash 命令来决定不与某些内容工作，也可以重写已经发生的提交就像它们以另一种方式发生的一样。 这可能涉及改变提交的顺序，改变提交中的信息或修改文件，将提交压缩或是拆分，或完全地移除提交 - 在将你的工作成果与他人共享之前。

在本节中，你可以学到如何完成这些非常有用的工作，这样在与他人分享你的工作成果时你的提交历史将如你所愿地展示出来。

修改最后一次提交

修改你最近一次提交可能是所有修改历史提交的操作中最常见的一个。 对于你的最近一次提交，你往往想做两件事情：修改提交信息，或者修改你添加、修改和移除的文件的快照。

如果，你只是想修改最近一次提交的提交信息，那么很简单：

```
 git commit --amend
```

这会把你带入文本编辑器，里面包含了你最近一条提交信息，供你修改。 当保存并关闭编辑器后，编辑器将会用你输入的内容替换最近一条提交信息。

如果你已经完成提交，又因为之前提交时忘记添加一个新创建的文件，想通过添加或修改文件来更改提交的快照，也可以通过类似的操作来完成。 通过修改文件然后运行 git add 或 git rm 一个已追踪的文件，随后运行 git commit --amend 拿走当前的暂存区域并使其做为新提交的快照。

使用这个技巧的时候需要小心，因为修正会改变提交的 SHA-1 校验和。 它类似于一个小的变基 - 如果已经推送了最后一次提交就不要修正它。

修改多个提交信息

为了修改在提交历史中较远的提交，必须使用更复杂的工具。 Git 没有一个改变历史工具，但是可

以使用变基工具来变基一系列提交，基于它们原来的 HEAD 而不是将其移动到另一个新的上面。 通过交互式变基工具，可以在任何想要修改的提交后停止，然后修改信息、添加文件或做任何想做的事情。 可以通过给 git rebase 增加 -i 选项来交互式地运行变基。 必须指定想要重写多久远的历史，这可以通过告诉命令将要变基到的提交来做到。

例如，如果想要修改最近三次提交信息，或者那组提交中的任意一个提交信息，将想要修改的最近一次提交的父提交作为参数传递给 git rebase -i命令，即 HEAD~2^ 或 HEAD~3。 记住 ~3 可能比较容易，因为你正尝试修改最后三次提交；但是注意实际上指定了以前的四次提交，即想要修改提交的父提交：

```
git rebase -i HEAD~3
```

再次记住这是一个变基命令 - 在 HEAD~3..HEAD 范围内的每一个提交都会被重写，无论你是否修改信息。 不要涉及任何已经推送到中央服务器的提交 - 这样做会产生一次变更的两个版本，因而使他人困惑。

运行这个命令会在文本编辑器上给你一个提交的列表，看起来像下面这样：

```
pick f7f3f6d changed my name a bit
pick 310154e updated README formatting and added blame
pick a5f4a0d added cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

需要重点注意的是相对于正常使用的 log 命令，这些提交显示的顺序是相反的。 运行一次 log 命令，会看到类似这样的东西：

```
git log --pretty=format:"%h %s" HEAD~3..HEAD
a5f4a0d added cat-file
310154e updated README formatting and added blame
f7f3f6d changed my name a bit
```

注意其中的反序显示。 交互式变基给你一个它将会运行的脚本。 它将会从你在命令行中指定的提交（HEAD~3）开始，从上到下的依次重演每一个提交引入的修改。 它将最旧的而不是最新的列在上面，因为那会是第一个将要重演的。

你需要修改脚本来让它停留在你想修改的变更上。 要达到这个目的，你只要将你想修改的每一次提交前面的 ‘pick’ 改为 ‘edit’。 例如，只想修改第三次提交信息，可以像下面这样修改文件：

```
edit f7f3f6d changed my name a bit
pick 310154e updated README formatting and added blame
pick a5f4a0d added cat-file
```

当保存并退出编辑器时，Git 将你带回到列表中的最后一次提交，把你送回命令行并提示以下信息：

```
 git rebase -i HEAD~3
Stopped at f7f3f6d... changed my name a bit
You can amend the commit now, with

      git commit --amend

Once you’ re satisfied with your changes, run

      git rebase --continue
```

这些指令准确地告诉你该做什么。 输入

```
 git commit --amend
```

修改提交信息，然后退出编辑器。 然后，运行

```
 git rebase --continue
```

这个命令将会自动地应用另外两个提交，然后就完成了。 如果需要将不止一处的 pick 改为 edit，需要在每一个修改为 edit 的提交上重复这些步骤。 每一次，Git 将会停止，让你修正提交，然后继续直到完成。

重新排序提交

也可以使用交互式变基来重新排序或完全移除提交。 如果想要移除 “added cat-file” 提交然后修改另外两个提交引入的顺序，可以将变基脚本从这样：

```
pick f7f3f6d changed my name a bit
pick 310154e updated README formatting and added blame
pick a5f4a0d added cat-file
```

改为这样：

```
pick 310154e updated README formatting and added blame
```

```
pick f7f3f6d changed my name a bit
```

当保存并退出编辑器时，Git 将你的分支带回这些提交的父提交，应用 310154e 然后应用 f7f3f6d，最后停止。 事实修改了那些提交的顺序并完全地移除了 “added cat-file” 提交。

压缩提交

通过交互式变基工具，也可以将一连串提交压缩成一个单独的提交。 在变基信息中脚本给出了有用的指令：

```
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

如果，指定 “squash” 而不是 “pick” 或 “edit” ，Git 将应用两者的修改并合并提交信息在一起。 所以，如果想要这三次提交变为一个提交，可以这样修改脚本：

```
pick f7f3f6d changed my name a bit
squash 310154e updated README formatting and added blame
squash a5f4a0d added cat-file
```

当保存并退出编辑器时，Git 应用所有的三次修改然后将你放到编辑器中来合并三次提交信息：

```
# This is a combination of 3 commits.
# The first commit's message is:
changed my name a bit

# This is the 2nd commit message:

updated README formatting and added blame
```

```
# This is the 3rd commit message:

added cat-file
```

当你保存之后，你就拥有了一个包含前三次提交的全部变更的提交。

拆分提交

拆分一个提交会撤消这个提交，然后多次地部分地暂存与提交直到完成你所需次数的提交。 例如，假设想要拆分三次提交的中间那次提交。 想要将它拆分为两次提交：第一个 “updated README formatting”，第二个 “added blame” 来代替原来的 “updated README formatting and added blame”。 可以通过修改 rebase -i 的脚本来做到这点，将要拆分的提交的指令修改为 “edit”：

```
pick f7f3f6d changed my name a bit
edit 310154e updated README formatting and added blame
pick a5f4a0d added cat-file
```

然后，当脚本将你进入到命令行时，重置那个提交，拿到被重置的修改，从中创建几次提交。 当保存并退出编辑器时，Git 带你到列表中第一个提交的父提交，应用第一个提交（f7f3f6d），应用第二个提交（310154e），然后让你进入命令行。 那里，可以通过 git reset HEAD^ 做一次针对那个提交的混合重置，实际上将会撤消那次提交并将修改的文件未暂存。 现在可以暂存并提交文件直到有几个提交，然后当完成时运行 git rebase --continue：

```
git reset HEAD^
git add README
git commit -m 'updated README formatting'
git add lib/simplegit.rb
git commit -m 'added blame'
git rebase --continue
```

Git 在脚本中应用最后一次提交（a5f4a0d），历史记录看起来像这样：

```
git log -4 --pretty=format:"%h %s"

1c002dd added cat-file
9b29157 added blame
35cfb2b updated README formatting
f3cc40e changed my name a bit
```

再一次，这些改动了所有在列表中的提交的 SHA-1 校验和，所以要确保列表中的提交还没有推送到共享仓库中。

核武器级选项：filter-branch

有另一个历史改写的选项，如果想要通过脚本的方式改写大量提交的话可以使用它 - 例如，全局修

改你的邮箱地址或从每一个提交中移除一个文件。 这个命令是 filter-branch，它可以改写历史中大量的提交，除非你的项目还没有公开并且其他人没有基于要改写的工作的提交做的工作，你不应当使用它。 然而，它可以很有用。 你将会学习到几个常用的用途，这样就得到了它适合使用地方的想法。

从每一个提交移除一个文件

这经常发生。 有人粗心地通过 git add . 提交了一个巨大的二进制文件，你想要从所有地方删除它。 可能偶然地提交了一个包括一个密码的文件，然而你想要开源项目。 filter-branch 是一个可能会用来擦洗整个提交历史的工具。 为了从整个提交历史中移除一个叫做 passwords.txt 的文件，可以使用 --tree-filter 选项给 filter-branch：

```
 git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)
Ref 'refs/heads/master' was rewritten
```

--tree-filter 选项在检出项目的每一个提交后运行指定的命令然后重新提交结果。 在本例中，你从每一个快照中移除了一个叫作 passwords.txt 的文件，无论它是否存在。 如果想要移除所有偶然提交的编辑器备份文件，可以运行类似 git filter-branch --tree-filter 'rm -f *~' HEAD 的命令。

最后将可以看到 Git 重写树与提交然后移动分支指针。 通常一个好的想法是在一个测试分支中做这件事，然后当你决定最终结果是真正想要的，可以硬重置 master 分支。 为了让 filter-branch 在所有分支上运行，可以给命令传递 --all 选项。

使一个子目录做为新的根目录

假设已经从另一个源代码控制系统中导入，并且有几个没意义的子目录（trunk、tags 等等）。 如果想要让 trunk 子目录作为每一个提交的新的项目根目录，filter-branch 也可以帮助你那么做：

```
 git filter-branch --subdirectory-filter trunk HEAD
Rewrite 856f0bf61e41a27326cdae8f09fe708d679f596f (12/12)
Ref 'refs/heads/master' was rewritten
```

现在新项目根目录是 trunk 子目录了。 Git 会自动移除所有不影响子目录的提交。

全局修改邮箱地址

另一个常见的情形是在你开始工作时忘记运行 git config 来设置你的名字与邮箱地址，或者你想要开源一个项目并且修改所有你的工作邮箱地址为你的个人邮箱地址。 任何情形下，你也可以通过 filter-branch 来一次性修改多个提交中的邮箱地址。 需要小心的是只修改你自己的邮箱地址，所以你使用 --commit-filter：

```
git filter-branch --commit-filter '
      if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];
      then
            GIT_AUTHOR_NAME="Scott Chacon";
            GIT_AUTHOR_EMAIL="schacon@example.com";
```

```
        git commit-tree "$@";
    else
        git commit-tree "$@";
    fi' HEAD
```

这会遍历并重写每一个提交来包含你的新邮箱地址。 因为提交包含了它们父提交的 SHA-1 校验和，这个命令会修改你的历史中的每一个提交的 SHA-1 校验和，而不仅仅只是那些匹配邮箱地址的提交。

如何减小仓库体积

因为我们码云平台目前仅提供 1G 的仓库大小，且单文件限制在 100M，如果您的项目中不小心打包进来了比较大的二进制文件，那么仓库很快就会超过我们规定的大小，这时，您需要精简您的仓库以免因为仓库大小超过规定而导致该仓库停止访问，这里给出精简仓库大小的命令:

改写历史，去除大文件

```
git filter-branch --tree-filter 'rm -f path/to/large/files' --tag-name-filter cat -- --all
git push origin --tags --force
git push origin --all --force
```

并告知所有组员，push 代码前需要 pull rebase，而不是 merge，否则会从该组员的本地仓库再次引入到远程库中，导致项目在此被码云系统屏蔽。

更加具体的操作可以点击这里
(http://my.oschina.net/jfinal/blog/215624?fromerr=ZTZ6c38X)查看

版权以及使用条款

本站使用说明以及用户条款

感谢您访问 git.oschina.net（以下简称“本网站”）。

本网站由深圳市奥思网络科技有限公司（以下简称“奥思网络”）运行维护。本网站向公众开放，提供基于 git 的代码托管和项目管理服务。 在使用本网站前，敬请您仔细阅读以下各项使用条款（以下简称“本使用条款”）。您对本网站的使用(包括但不限于对本网站的访问、登录，对本网站内容的浏览和使用)，将被视为您自愿承诺接受本声明的约束。如果您对本使用条款的内容不能接受，您应当立即停止使用本网站并迅速离开。

本网站的使用

奥思网络有限许可您在您的计算机设备上浏览（使用）本网站展示的内容（由本使用条款第2条规定）。

知识产权声明

本网站，包括(但不仅限于) 文字，内容，软件，录像，音乐，声音，图形，照片，图表，美术设计，图片，名称，标识，商标和/或服务标志（包括已注册和未注册的），以及其他资料（以下简称网站内容）都受到版权法，商标法和一切知识产权公约的保护。本网站内容包括奥思网络所有或控制下的内容和第三方所有或控制下，并授权奥思网络使用的内容。所有代码，文章，讨论等一切构成本网站的元素都可能是受版权保护的作品。您同意遵守所有适用本网站的版权保护法律法规，以及所有本网站包含的补充性的版权说明或限制。本网站的内容均由相应的机构/个人上传、维护。对于本站内容的任何使用请遵守内容所附带的授权协议。如不清楚相应的授权协议请询问上传该内容的机构/个人。

任何在 git.oschina.net 上注册的账号上传的内容的版权均归上传者所有，上传者承担所有被上传内容的版权责任。奥思网络有权在其拥有的网站上，展示上传者上传到本网站公共区域的内容。

个人信息的保护

奥思网络尊重访问本网站的任何个人的隐私信息。当您访问本网站的时候可能被要求提供您个人的基本资料(如姓名、电子邮件、电话号码等)，您可以自行选择是否提供。对于您提供的个人信息，奥思网络将根据中华人民共和国相关法律进行保密并严格保管，不会将这些信息以任何方式提供或展示给任何第三方，但下述情况除外：

(一) 当司法机关或行政机关依照法定程序和法定职权要求本网站披露个人资料时，我们将依法提供相关资料。对于此情况下的任何披露，本网站均得免责；

(二) 对于任何由黑客攻击、计算机病毒侵入或发作、或政府管制而造成的暂时性关闭等影响网站正常运营的不可抗力事件所造成的个人资料的泄露、丢失、被盗用或被篡改等，本网站均得免责；

(三) 对于用户将个人密码告知他人或与他人共享注册帐号所导致的任何个人资料泄露，丢失、被盗用或被篡改等,本网站不负任何责任；

(四) 对于与本网站链接的任何其它网站的个人资料泄露，丢失、被盗用或被篡改等事件，本网站不负任何责任。

网站的更新与变更

奥思网络提供的本网站上的内容仅为方便您获取信息，奥思网络有权单方面在任何时间，对本网站的内容进行变更或更新，此种变更或更新无需以通知您或任何第三方为前提。您承认并接受上述变更或更新。我们建议您定期访问本网站以尽快获知有关的更新或变更的信息。

无担保声明

在法律允许的最大限度内，本网站及本网站的内容不包括任何明确或暗示的有关（但不限于）所有权，适销性，质量满意程度，特定用途的适用性，准确性、时效性和完整性，不侵犯知识产权和其它方权利的承诺。

奥思网络尽其合理的商业努力维护本网站的安全和功能，但不保证本网站上的所有功能正常运作，也不保证本网站连续性地或无故障运行或缺陷被及时更正。

奥思网络不保证本网站系统适合您的电脑操作系统性能，也不保证本网站或其服务器永不产生故障或没有病毒、木马程序和其他有害程序。奥思网络亦不承担您因上述问题而产生的任何电脑的损坏的责任。

奥思网络不为您所使用的访问本网站的网络线路和设备的可靠性以及连续性承担责任。

禁止盗链

奥思网络禁止擅自利用本网站内容的网页构架系统的行为。奥思网络保留断开任何未经授权指向本网站的超文本链接或网页构架系统的权利。

项目大小

奥思网络允许用户上传项目，规定每个项目的大小上限为1G，禁止大量的外链请求到本网站，并且不允许存放与代码无关的文件，如果代码实在有需要更大，请发邮件(git#oschina.cn)联系我们，说明使用场景和原因进行申请，我们邮件同意后方可上传更大的项目，如果未经允许私自上传大项目、与代码无关的文件或者大量的外链请求，奥思网络保留停止项目的使用以及通知作者备份并删除的权利。

终止使用

如果您的账户违反当地相关法律法规或网站使用条款，在未经事先通知的情况下，奥思网络有权自行判断并立即终止您对本网站的使用，或者采取措施禁止您对本网站的再次访问。

准据法和管辖权

本网站内容及本使用条款的解释和适用均适用中华人民共和国法律，由于使用本网站而产生的一切纠纷，除另有约定，均由中华人民共和国(奥思网络所在地)人民法院管辖。

继续有效原则

如果本使用条款中的部分条款被有关机关认定为无效，则此无效部分并不影响本使用条款其他部分的效力，其他部分仍然有效。

版权声明

本文档引用互联网内容部分版权归属为作者本人，其他部分版权归属码云平台。码云平台对本文档内容不做限制，任何个人或单位或组织均可以引用本文档内容而无需经过码云平台同意，但需要保留出处。任何人对本文档进行拷贝，修改而生成自己的文档均无需码云平台的同意，但需要保留出处。

鸣谢以及关于文档的维护

本文档git相关部分编撰参考廖雪峰先生的git的教程，具体请参考廖雪峰先生的官方博客(http://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000) 本文档已经覆盖了日常工作中绝大部分的使用场景，如有未覆盖的地方，请自行通过搜索引擎搜索并联系文档管理员将您遇上的问题以及解决办法添加到FAQ中，谢谢

继续阅读

你可以点击这里阅读权威的 git 书籍ProGit(http://git.oschina.net/progit)

以下为收集的开源中国社区热心网友制作的码云（Gitee.com）跟各种 IDE，软件的集成办法，可

以点击查看：

- git官方中文文档(https://git-scm.com/book/zh/v2) 推荐阅读!
- eclipse中egit插件使用-图文并茂-详细(http://my.oschina.net/songxinqiang/blog/194203)
- Visual Studio 2012连接到osc@git(http://my.oschina.net/gal/blog/141442)
- TortoiseGit配合msysGit在Git@OSC代码托管的傻瓜教程 (http://my.oschina.net/icelily/blog/141342)
- 利用eclipse的git插件EGit与git@osc交互(http://my.oschina.net/kzhou/blog/132146)
- Git初体验(http://my.oschina.net/dxqr/blog/134811)
- 在win7系统下使用TortoiseGit(乌龟git)简单操作 Git@OSC(http://my.oschina.net/longxuu/blog/141699)
- Xcode连接git @ osc(http://my.oschina.net/zxs/blog/142544)
- git@osc(git)中team开发、fork和pull request的用法 (http://my.oschina.net/kzhou/blog/150290)
- eclipse的git插件整合Git@OSC(http://my.oschina.net/u/861562/blog/151975)
- Eclipse使用EGit管理git@OSC项目(http://my.oschina.net/China2012/blog/174874)
- 如何导入外部Git仓库到中国源代码托管平台 （Git@OSC）(http://www.oschina.net/question/82993_133520)
- https 方式使用Git@OSC设置密码的方式(http://git.oschina.net/oschina/git-osc/issues/2586)
- sourcetree使用git@osc教程(http://blog.csdn.net/riven_wn/article/details/45332713)
- Netbeans使用git教程(https://netbeans.org/kb/docs/ide/git_zh_CN.html?print=yes)