|  |
| --- |
| 文章标题 |
| **作者：**  **归档：学习笔记** |
| **快捷键：**  Ctrl + 1 标题1  Ctrl + 2 标题2  Ctrl + 3 标题3  Ctrl + 4 实例  Ctrl + 5 程序代码  Ctrl + 6 正文 |
| **格式说明：**  蓝色字体：注释  黄色背景：重要  绿色背景：注意 |

****

**老男孩教育教学核心思想6重：重目标、重思路、重方法、重实践、重习惯、重总结**

**学无止境，老男孩教育成就你人生的起点！**

# GIT介绍

## 9常见版本的管理工具

### SVN介绍

集中式的版本控制系统，只有一个中央数据仓库，如果中央数据仓库挂了或者不可访问，所有的使用者无法使用SVN，无法进行提交或备份文件。





### Git

分布式的版本控制系统，在每个使用者电脑上就有一个完整的数据仓库，没有网络依然可以使用Git。当然为了习惯及团队协作，会将本地数据同步到Git服务器或者GitHub等代码仓库。



## GIT安装配置

### 环境准备

[root@oldboyedu ~]# cat /etc/redhat-release

CentOS Linux release 7.2.1511 (Core)

[root@oldboyedu ~]# uname -r

3.10.0-327.el7.x86\_64

[root@oldboyedu ~]# getenforce

Disabled

[root@oldboyedu ~]# systemctl status firewalld.service

● firewalld.service - firewalld - dynamic firewall daemon

Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)

Active: inactive (dead)

[root@oldboyedu ~]# hostname -I

172.16.1.200 10.0.0.200 192.168.122.1

### 安装Git

#### 最小安装已经存在Git ，如果没有使用yum 安装即可

[root@oldboyedu ~]# rpm -qa git

git-1.8.3.1-5.el7.x86\_64

#### yum 安装

[root@oldboyedu ~]# yum -y install git

[root@oldboyedu ~]# rpm -qa git

git-1.8.3.1-6.el7\_2.1.x86\_64

### Git全局配置

全局变量即配置用户和邮箱，自定义即可

git config --global user.name "zy" #配置git使用用户

git config --global user.email "zhangyao@oldboyedu.com" #配置git使用邮箱

git config --global color.ui true #语法高亮

git config --list # 查看全局配置

查看配置

[root@oldboyedu ~]# cat .gitconfig

[user]

name = zy

email = zhangyao@oldboyedu.com

[color]

ui = true

# GIT使用

## 初始化工作目录

### 创建git工作目录

[root@oldboyedu ~]# mkdir git\_data

[root@oldboyedu ~]# cd git\_data

### 初始化

[root@oldboyedu git\_data]# git init

初始化空的 Git 版本库于 /root/git\_data/.git/

[root@oldboyedu git\_data]# ls

[root@oldboyedu git\_data]# ls -a

. .. .git

## 查看工作目录状态

### 查看工作区域状态

[root@oldboyedu git\_data]# git status

# 位于分支 master

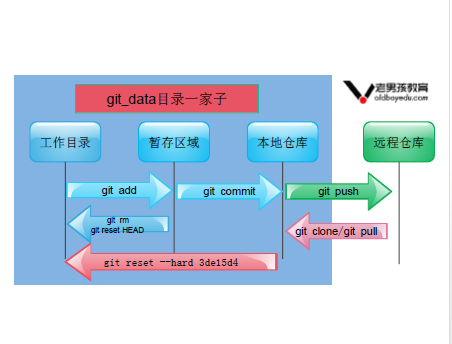
#

# 初始提交

#

无文件要提交（创建/拷贝文件并使用 "git add" 建立跟踪）

## 创建文件并提交



### 创建文件

[root@oldboyedu git\_data]# touch README

#### 查看工作区域状态

[root@oldboyedu git\_data]# git status

# 位于分支 master

#

# 初始提交

#

# 未跟踪的文件:

# （使用 "git add <file>..." 以包含要提交的内容）

#

# README

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）

### 将文件添加到暂存区域

[root@oldboyedu git\_data]# git add README

#### 查看状态

[root@oldboyedu git\_data]# git status

# 位于分支 master

#

# 初始提交

#

# 要提交的变更：

# （使用 "git rm --cached <file>..." 撤出暂存区）

#

# 新文件： README

#

### 将文件提交到本地库

-m 后跟注释，用于解释

[root@oldboyedu git\_data]# git commit -m "this is first commit"

[master（根提交） 16f3c94] this is first commit

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 README

#### 查看状态

[root@oldboyedu git\_data]# git status

# 位于分支 master

无文件要提交，干净的工作区

# -a 表示直接提交

### 修改后直接提交到本地库

-a 直接提交到本地

用于修改已经提交过的文件，直接提交到本地库，无需先提交暂存区

[root@oldboyedu git\_data]# git commit -a -m 'second commit'

[master adeb3bf] second commit

1 file changed, 1 insertion(+)

## 删除文件

### 没有添加到暂存区的数据直接rm删除即可。

[root@oldboyedu git\_data]# touch oldboy

[root@oldboyedu git\_data]# git status

# 位于分支 master

# 未跟踪的文件:

# （使用 "git add <file>..." 以包含要提交的内容）

#

# oldboy

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）

[root@oldboyedu git\_data]# rm -f oldboy

[root@oldboyedu git\_data]# git status

# 位于分支 master

无文件要提交，干净的工作区

### 已经添加到暂存区数据：

将文件从git暂存区域的追踪列表移除(并不会删除当前工作目录内的数据文件)，使用 "git reset HEAD <file>..." 撤出暂存区或git rm --cached database

[root@oldboyedu git\_data]# touch oldboy

[root@oldboyedu git\_data]# git status

# 位于分支 master

# 要提交的变更：

# （使用 "git reset HEAD <file>..." 撤出暂存区）

#

# 新文件： oldboy

#

#### 移除暂存区

[root@oldboyedu git\_data]# git reset HEAD oldboy

[root@oldboyedu git\_data]# git status

# 位于分支 master

# 未跟踪的文件:

# （使用 "git add <file>..." 以包含要提交的内容）

#

# oldboy

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）

### 将文件数据从git暂存区和工作目录一起删除，git rm -f database

[root@oldboyedu git\_data]# git add oldboy

[root@oldboyedu git\_data]# git status

# 位于分支 master

# 要提交的变更：

# （使用 "git reset HEAD <file>..." 撤出暂存区）

#

# 新文件： oldboy

#

[root@oldboyedu git\_data]# git rm -f oldboy

rm 'oldboy'

[root@oldboyedu git\_data]# git status

# 位于分支 master

无文件要提交，干净的工作区

[root@oldboyedu git\_data]# ls

README

## 重命名暂存区数据

### 没有添加到暂存区的数据直接mv/rename改名即可。

### 已经添加到暂存区数据：

git mv README NOTICE

## 查看历史记录 --查看所有版本信息

git log #→查看提交历史记录

git log -2 #→查看最近几条记录

git log -p -1 #→-p显示每次提交的内容差异,例如仅查看最近一次差异

git log --stat -2 #→--stat简要显示数据增改行数，这样能够看到提交中修改过的内容，对文件添加或移动的行数，并在最后列出所有增减行的概要信息

git log --pretty=oneline #→--pretty根据不同的格式展示提交的历史信息

git log --pretty=fuller -2 #→以更详细的模式输出提交的历史记录

git log --pretty=fomat:"%h %cn" #→查看当前所有提交记录的简短SHA-1哈希字串与提交着的姓名，其他格式见备注。

### #→还可以使用format参数来指定具体的输出格式，这样非常便于后期编程的提取分析哦，常用的格式有：

%s 提交说明。

%cd 提交日期。

%an 作者的名字。

%cn 提交者的姓名。

%ce 提交者的电子邮件。

%H 提交对象的完整SHA-1哈希字串。

%h 提交对象的简短SHA-1哈希字串。

%T 树对象的完整SHA-1哈希字串。

%t 树对象的简短SHA-1哈希字串。

%P 父对象的完整SHA-1哈希字串。

%p 父对象的简短SHA-1哈希字串。

%ad 作者的修订时间。

## 还原数据（切换到某个版本）

Git服务程序中有一个叫做HEAD的版本指针，当用户申请还原数据时，其实就是将HEAD指针指向到某个特定的提交版本，但是因为Git是分布式版本控制系统，为了避免历史记录冲突，故使用了SHA-1计算出十六进制的哈希字串来区分每个提交版本，另外默认的HEAD版本指针会指向到最近的一次提交版本记录，而上一个提交版本会叫HEAD^，上上一个版本则会叫做HEAD^^，当然一般会用HEAD~5来表示往上数第五个提交版本。

gitreset --hard HEAD^ #→还原历史提交版本上一次

gitreset --hard 3de15d4 #→找到历史还原点的SHA-1值后，就可以还原(值不写全,系统会自动匹配)

[root@oldboyedu git\_data]# git reset --hard 16f3

HEAD 现在位于 16f3c94 this is first commit

[root@oldboyedu git\_data]# git log

commit 16f3c947c5b69b28922f502027d2efec332eb368

Author: zy <zhangyao@oldboyedu.com>

Date: Wed Nov 8 09:48:09 2017 +0800

this is first commit

### 还原未来数据

什么是未来数据？就是你还原到历史数据了，但是你后悔了，想撤销更改，但是gitlog已经找不到这个版本了。

gitreflog#→查看未来历史更新点

[root@oldboyedu git\_data]# git reflog

16f3c94 HEAD@{0}: reset: moving to 16f3

c283f22 HEAD@{1}: commit: four commit

223deee HEAD@{2}: commit: third commit

adeb3bf HEAD@{3}: commit: second commit

16f3c94 HEAD@{4}: commit (initial): this is first commit

[root@oldboyedu git\_data]# git reset --hard c283f22

HEAD 现在位于 c283f22 four commit

## 标签使用

前面回滚使用的是一串字符串，又长又难记。

### 指定标签，

git tag v1.0 #→当前提交内容打一个标签(方便快速回滚)，每次提交都可以打个tag。

### 查看标签

git tag #→查看当前所有的标签

git show v1.0 #→查看当前1.0版本的详细信息

git tag v1.2 -m "version 1.2 release is test" #→创建带有说明的标签,-a指定标签名字，-m指定说明文字

git tag -d v1.0 #→我们为同一个提交版本设置了两次标签,删除之前的v1.0

#### 切换标签

[root@centos7 git\_data]# git reset --hard 0bdf2e7

HEAD is now at 0bdf2e7 modified README file

[root@centos7 git\_data]# git reset --hard V1.0

HEAD is now at a66370a add oldboy dir

#### 查看标签，并切换

[root@oldboyedu git\_data]# git tag v20171104

[root@oldboyedu git\_data]# git tag

v20171104

[root@oldboyedu git\_data]# git reset --hard 16f3

HEAD 现在位于 16f3c94 this is first commit

[root@oldboyedu git\_data]# git reset --hard v20171104

HEAD 现在位于 c283f22 four commit

## 分支结构

在实际的项目开发中，尽量保证master分支稳定，仅用于发布新版本，平时不要随便直接修改里面的数据文件。

那在哪干活呢？干活都在dev分支上。每个人从dev分支创建自己个人分支，开发完合并到dev分支，最后dev分支合并到master分支。

所以团队的合作分支看起来会像下图那样。



### 创建分支

#### 创建分支

[root@oldboyedu git\_data]# git branch linux

#### 查看分支

[root@oldboyedu git\_data]# git branch

linux

\* master

#### 切换分支

[root@oldboyedu git\_data]# git checkout linux

切换到分支 'linux'

[root@oldboyedu git\_data]# git branch

\* linux

master

[root@oldboyedu git\_data]# ls

README test test1

### 在分支linux编辑代码

[root@oldboyedu git\_data]# echo linux >> README

[root@oldboyedu git\_data]# git commit -a -m 'this is linux branch commit'

[linux 0e5ff51] this is linux branch commit

1 file changed, 1 insertion(+)

[root@oldboyedu git\_data]# cat README

1234

linux

[root@oldboyedu git\_data]# git log

commit 0e5ff512473d3531e8b569043f9883f9d1892131

Author: zy <zhangyao@oldboyedu.com>

Date: Wed Nov 8 11:59:40 2017 +0800

this is linux branch commit

commit c283f22b8ffca4e19785cdf0e7bc50c1b49484bb

Author: zy <zhangyao@oldboyedu.com>

Date: Wed Nov 8 10:43:21 2017 +0800

four commit

commit 223deeee24c6b7fa65b9b61add8da7a7342dbd4e

Author: zy <zhangyao@oldboyedu.com>

Date: Wed Nov 8 10:42:53 2017 +0800

[root@oldboyedu git\_data]# git checkout master

切换到分支 'master'

[root@oldboyedu git\_data]# cat README

1234

### 自动合并代码

git merge 合并代码

[root@oldboyedu git\_data]# git merge linux

更新 c283f22..0e5ff51

Fast-forward

README | 1 +

1 file changed, 1 insertion(+)

[root@oldboyedu git\_data]# cat README

1234

linux

[root@oldboyedu git\_data]# git log -1

commit 0e5ff512473d3531e8b569043f9883f9d1892131

Author: zy <zhangyao@oldboyedu.com>

Date: Wed Nov 8 11:59:40 2017 +0800

this is linux branch commit

## 分支冲突

#### 切换分支，并合并

[root@oldboyedu git\_data]# git checkout master

换到分支 'master'

[root@oldboyedu git\_data]# git merge linux

自动合并 README

冲突（内容）：合并冲突于 README

自动合并失败，修正冲突然后提交修正的结果。

#### 查看文件内容

[root@oldboyedu git\_data]# cat README

1234

linux

<<<<<<< HEAD

master branch

=======

linux branch

>>>>>>> linux

#### 删除其中一个冲突分支

[root@oldboyedu git\_data]# vim README # 保留一个

1234

linux

master branch

[root@oldboyedu git\_data]# git commit -a -m 'this is master branch two commit'

[master d33d077] this is master branch two commit

## gitdiff可以对比当前文件与仓库已保存文件的区别，知道了对README作了什么修改后，再把它提交到仓库就放⼼多了。

gitdiff README

# GIT服务器

## windows客户端登录

前面讲的都是linux客户端，在讲讲windows客户端使用，安装Git-2.10.0-64-bit。

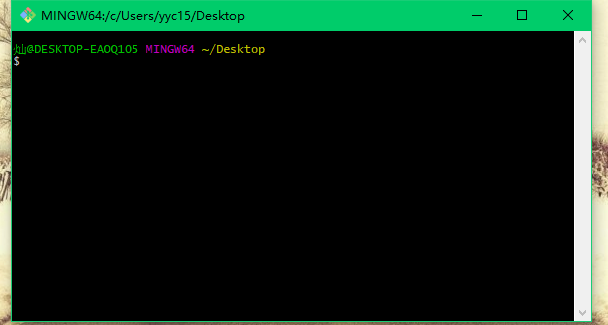


•windows的git，本质是windows上的linux系统

•TortoiseGit-2.2.0.0-64bit 给git加外壳，svn客户端的git版本

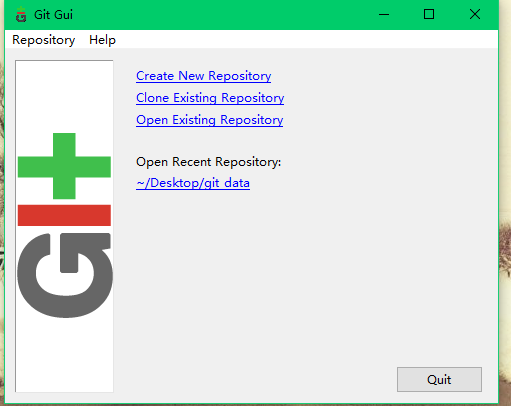
### Git bash here

安装完成桌面右键打开即可



通过命令行模式管理GIT，具体操作与linux一致

### Git Gui here



页面版，通过页面管理Git

## 搭建私有的仓库gitlab

前面我们已经知道Git人人都是中心，那他们怎么交互数据呢？

•使用GitHub或者码云等公共代码仓库

•使用GitLab私有仓库

### rpm包下载地址

https://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el7/

### 安装

yum localinstall gitlab-ce-9.1.4-ce.0.el7.x86\_64.rpm

### 初始化

gitlab-ctl reconfigure

### 登录

浏览器访问10.0.0.200

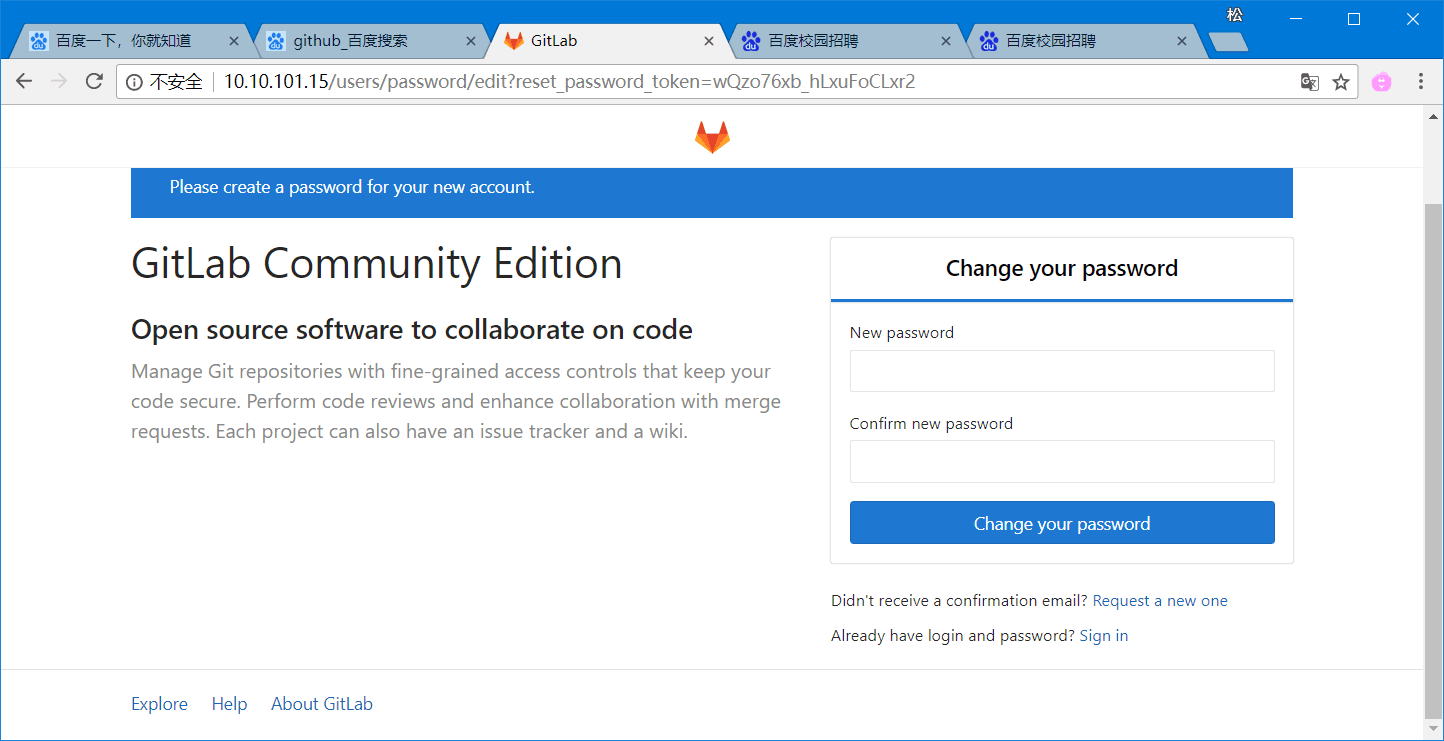
#### 查看状态

gitlab-ctl status/stop/start

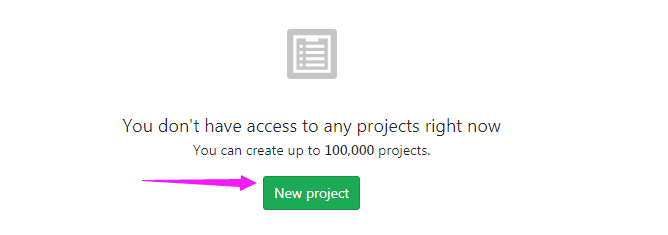
## 网页操作

### 通过浏览器访问页面，设置初始密码，其他操作类似GitHUB。

账户：root 密码自己设置为12345678

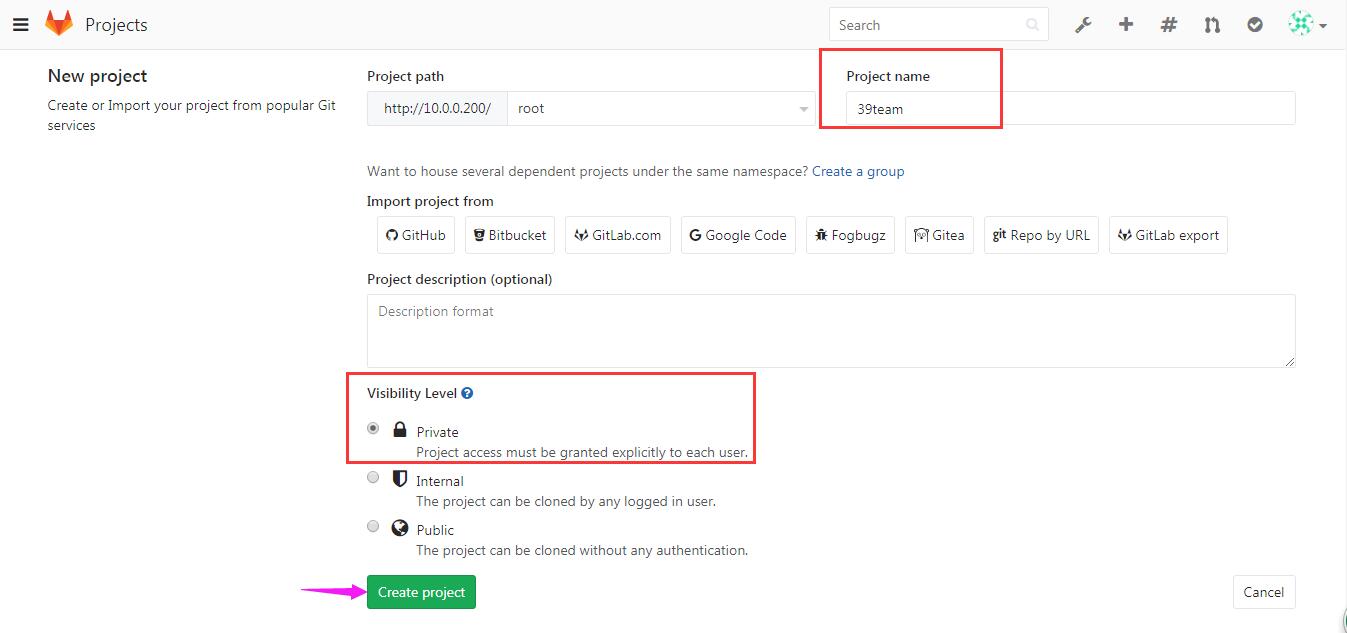


### 创建新的项目



### 添加项目名

选择私人模式



### 生成秘钥，加入网页

#### 在一台客户端生成秘钥

[root@lb01 ~]# ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id\_rsa):

Created directory '/root/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /root/.ssh/id\_rsa.

Your public key has been saved in /root/.ssh/id\_rsa.pub.

The key fingerprint is:

bc:9c:53:35:66:b1:9e:0e:22:f7:93:85:8e:47:96:37 root@lb01

The key's randomart image is:

+--[ RSA 2048]----+

| . |

| o |

| \* |

| . B o |

| . S B E |

| + @ \* . |

| \* \* . |

| o . |

| |

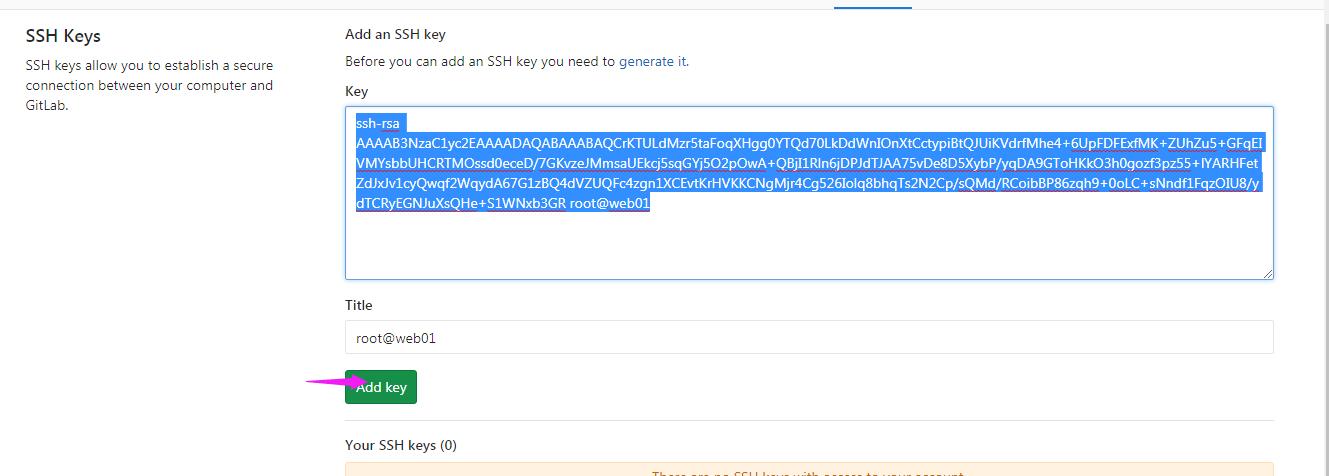
+-----------------+

#### 查看公钥

[root@lb01 ~]# cat /root/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC+37lwhdAS/vZ+eGozeU74AV9HdYEUmHWdajVbuEf7zYi+GKHjZL0MZB5wLN/Lq45nyWL1daGzyZu1aVxILRMhvFw+cKtzKlIAbfDSwPwTJ+mMphgofy9xoHHnIi5rsY2uKIJoesE6cqg/F7dSihshPAJkwhedk/AwrYE+Rj4I/9yHAK7RaIBvIrvvPeWLOa5OxeOw/4232SKbtYwmwG7hHkQ7BPUgqJYcvGn811x9xq+Tktumx74hKy6w1HqyRX3OQI5ICsN94sdQnQxslhINhv2F7Iw6q+WOIw6i8E9hsaakS4x+BMTI3o8Qe/zflcoQJ64MtXDKl+XAyHbXqiQv root@lb01

#### 将公钥复制到网页



### 配置全局变量

#### 客户端配置全局变量

git config --global user.name "yyc"

git config --global user.email "yyc1503@163.com"

git config --global color.ui true

git clone [git@10.0.0.200/39team.git](mailto:git@10.0.0.200/39team.git)

### 按照网页流程，创建新的仓库

git clone git@gitlab.example.com:root/39team.git 如果没有本地解析就这直接使用IP地址

git clone [git@10.0.0.200:root/39team.git](mailto:git@10.0.0.200:root/39team.git) 使用IP地址

#### 会自动生成39team/目录，这个目录是在网页创建的

[root@lb01 ~]# cd 39team/

[root@lb01 39team]# ls

[root@lb01 39team]# touch README.md

[root@lb01 39team]# git add README.md

[root@lb01 39team]# git commit -m "add README"

[master（根提交） c602907] add README

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 README.md

[root@lb01 39team]# git push -u origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 208 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To git@10.0.0.200:root/39team.git

\* [new branch] master -> master

分支 master 设置为跟踪来自 origin 的远程分支 master。

#### 刷新网页页面

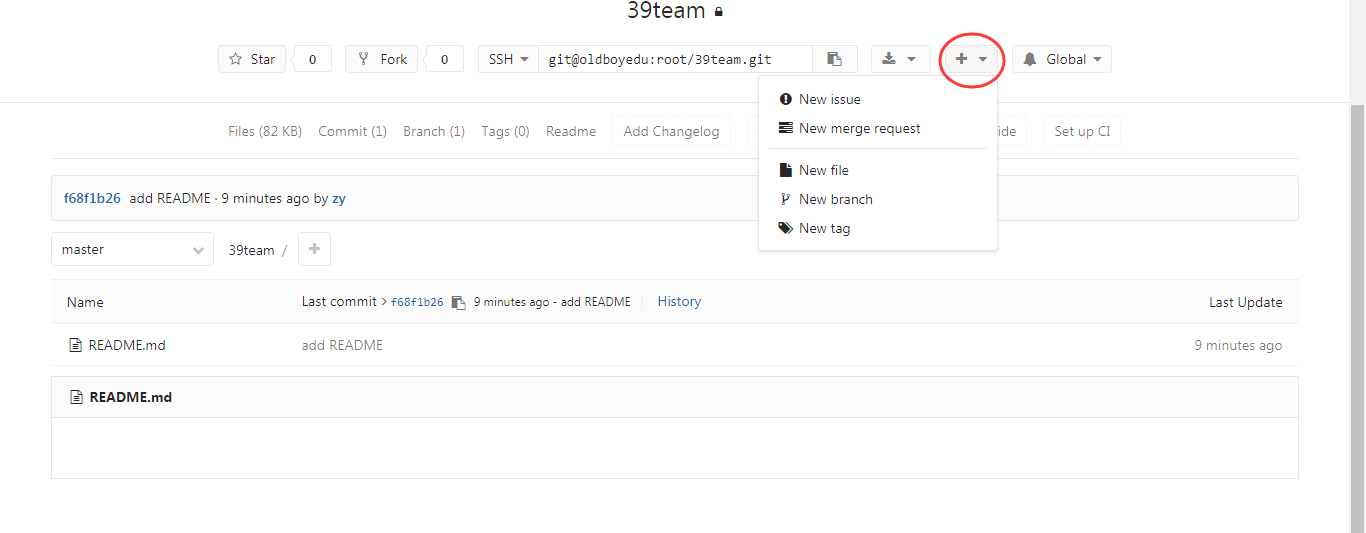
创建的文件就生成出来了

每次修改文件内容后需要添加到本地库

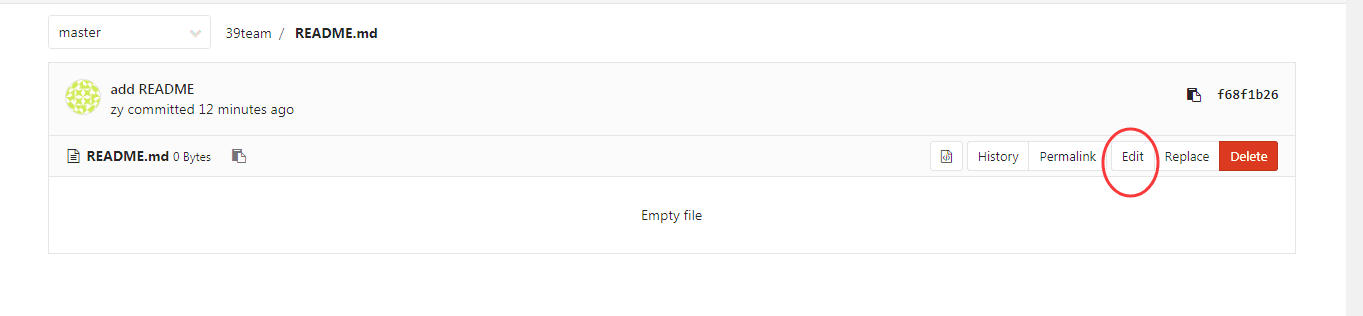
git commit - a -m "add README"

上传代码

git push -u origin master



### 编辑，输入代码





此处用到markdown语法

<http://www.zyops.com/markdown-syntax>

### 下载代码

[root@lb01 39team]# git pull

remote: Counting objects: 6, done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 6 (delta 1), reused 0 (delta 0)

Unpacking objects: 100% (6/6), done.

来自 10.0.0.200:root/39team

c602907..4ff9f65 master -> origin/master

更新 c602907..4ff9f65

Fast-forward

README.md | 33 +++++++++++++++++++++++++++++++++

1 file changed, 33 insertions(+)

[root@lb01 39team]# ll

总用量 4

-rw-r--r-- 1 root root 678 11月 9 19:02 README.md

[root@lb01 39team]# cat README.md

#Command line instructions

# 代码上线

## 小型公司代码上线

### 小型企业现状

小型公司一般只有几个开发人员，并且网站核心程序大多数都是PHP语言开发，为了方便，会直接通过FTP直接上传代码到线上服务器，随时随地上线更新。

### 上述代码的优缺点

#### 优点：

发布快，及时，随时随地就可以发布代码

#### 缺点：

开发人员发布的代码不经过测试人员的测试，用户刷新网页可能发现页面改变，也可能刷新瞬间程序在更新，到时无法访问，对网站用户的体验比较差，如果开发写错了代码，造成的影响就更大了，这是拿用户做测试的上线方案。

据统计，网站中大概50%以上的故障是和开发程序代码有关的，比如，开发写错了一个循环代码，导致了死循环，此时大量用户访问这个程序，就能把服务器资源耗尽，搞死服务器。

在中小公司网站出了问题一般是运维人员的问题（例如网站宕机），但这种情况下，问题大多可能由开发人员或代码引起的，这里比较好的策略是开发项目责任制。

### 小型企业上线架构方案建议

开发人员需在个人电脑搭建LNMP环境测试开发好的网站代码，并且在办公室或IDC机房的测试环境通过，最好有专职测试人员。

程序代码上线规定时间，例如：三天上线一次，如网站需经常更新可每天下午17点上线，这个看网站业务性质而定，原则就是影响用户体验最小。

代码上线之前需备份，网站程序出了问题方便回退，另外，从上线技巧上讲，上传代码时尽可能先传到服务器网站临时目录，传完整后一步mv过去，或者通过ln做软连接。

尽量由运维人员管理上线，对于代码的功能性，开发人员更在意，而对于代码的性能优化和上线后服务器的稳定，运维更在意服务器的稳点性，因此，如果网站宕机问题归运维管，就要让运维上线，这样更规范科学。否则，开发随意更新，出了问题运维负责，这样就错了，运维就永远无法抬头。

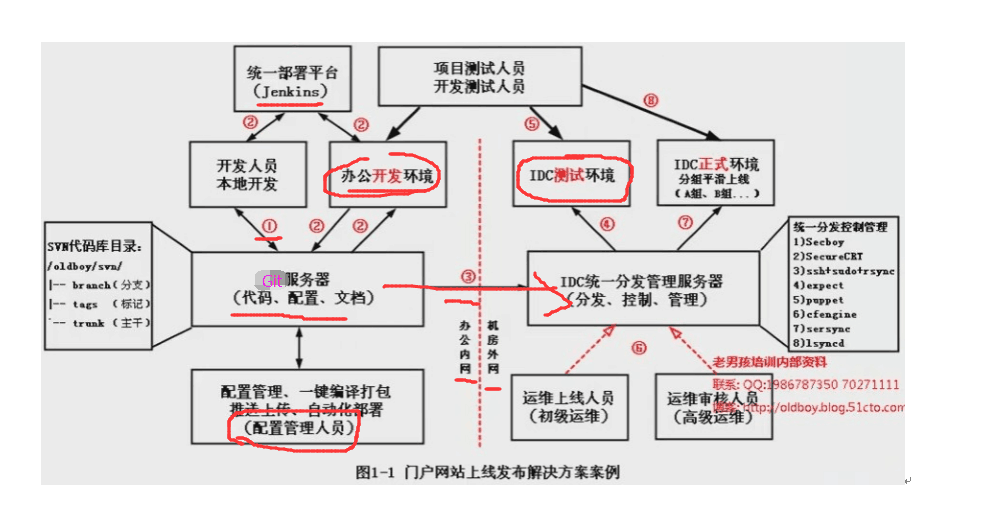
## 中型公司代码上线

中型企业上线，一般是规范运维人员操作步骤，指定统一的上线操作脚本、备份文件名称、备份文件路径，使操作人性化、统一化、自动话。

## 大型企业上线解决方案

大型企业上线一般制度和流流程控制比较多，比较严谨，下面是某大型企业上线解决方案：

### 架构图



### PHP程序代码上线注意事项

对于PHP上线方法：发布代码时（也需要测试流程）可以直接发布到正式线临时目录，然后mv或更改link的方式发布到 正式线目录，不需要重启http/php服务。

### JAVA程序代码上线注意事项

对于JAVA上线方法：需要分组平滑上线。例如，首先从负载均衡服务器上摘掉一半的服务器，发布代码后，重启服务器测试，没问题后，挂载上线完成的一半，再卸载另一半。如果前端有DNS智能解析，上线还可以分地区上线若干服务器，逐渐普及到全国的服务器，这个被称为灰度发布。

### AB组上线

#### 准备3套配置文件

nginx.conf

nginxA.conf

nginxB.conf

nginxall.conf

## 大型集群架构代码发布方案注意事项

上线的流程里：办公室测试环境—》IDC测试环境—》正式生产环境，所有环境中的所有软件均应版本统一，其次，尽量单一，否则将后患无穷（开发测试成功，IDC测试可能有问题）。例如：操作系统、web服务器、jdk、php、tomcat等版本）

采用全量升级。

参考资料：

https://mp.weixin.qq.com/s?\_\_biz=MzAxOTE5NjQwOA==&mid=2650113445&idx=1&sn=ca4231f30a39872db9fb6893d5740d49&chksm=83cb9532b4bc1c242bba69d52c96e188c43117c3c29fc9b47830e9d390514a3ff26f812e19a6&mpshare=1&scene=23&srcid=0718al9lUYSe5sPlxbcnTn7t#rd