

一、码云

码云(Git@OSC)是开源中国社区团队推出的基于Git的快速的、免费的、稳定的在线代码托管平台,不限制私有库和公有库数量.

github大伙总该听说过的吧, 码云就是中国版的github, 由于GitHub服务器在国外, 国内访问比较慢, 经常出问题, 抽起风来, 啥都搞不来. 所以还是国内的环境好, 码云还是很适用于大部分国人的, 虽然里面的内容不多, 但是相较于github的国外服务器, 码云在国内的速度快. 码云官网: <https://gitee.com/>, 后面我们做项目将用使用git做版本控制, 码云网站托管

所以先注册码云账号:



1、git安装

在ubuntu下安装git:

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

安装成功之后在shell终端git命令能看到下面信息说明已经安装成功。

```
python@python:~$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]
```

这些是各种场合常见的 Git 命令:

配置:

```
vim .gitconfig
```

#填入下面内容

```
[user]
```

```
email = 'huanggui0915@foxmail.com' # 修改为注册码云的邮箱,
```

```
name='hgf' # 填写用户名, 要求协同开发人员的用户名不能重复
```

在实际项目开发中, 按照如下步骤使用git进行代码管理

1.开发之初, 创建好仓库, 上传项目的框架、组员分支 2.组员克隆项目框架, 同步分支, 按分工开发, 在分支提交代码 3.在需要发布时, 将各分支合并到dev上, 再合并到master上

git将代码开发分成了工作区、暂存区、仓库区, 为了能够交换代码还需要有服务器, 我们这里使用码云。

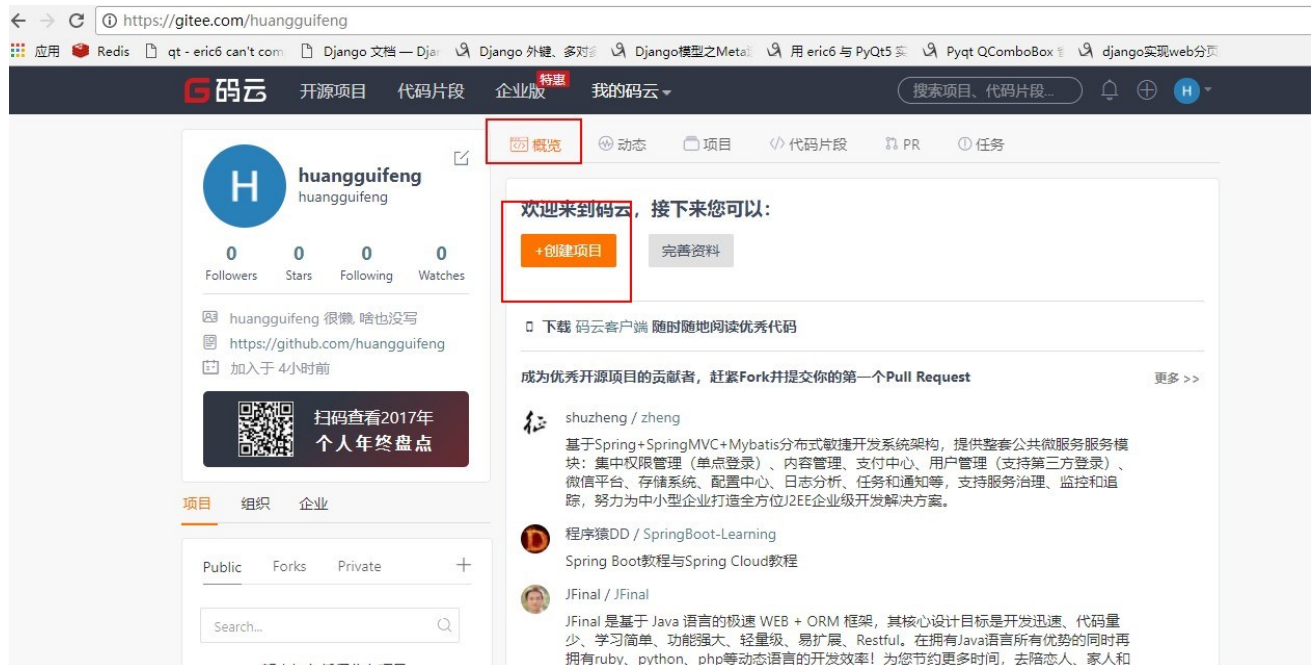


2、git常用命令

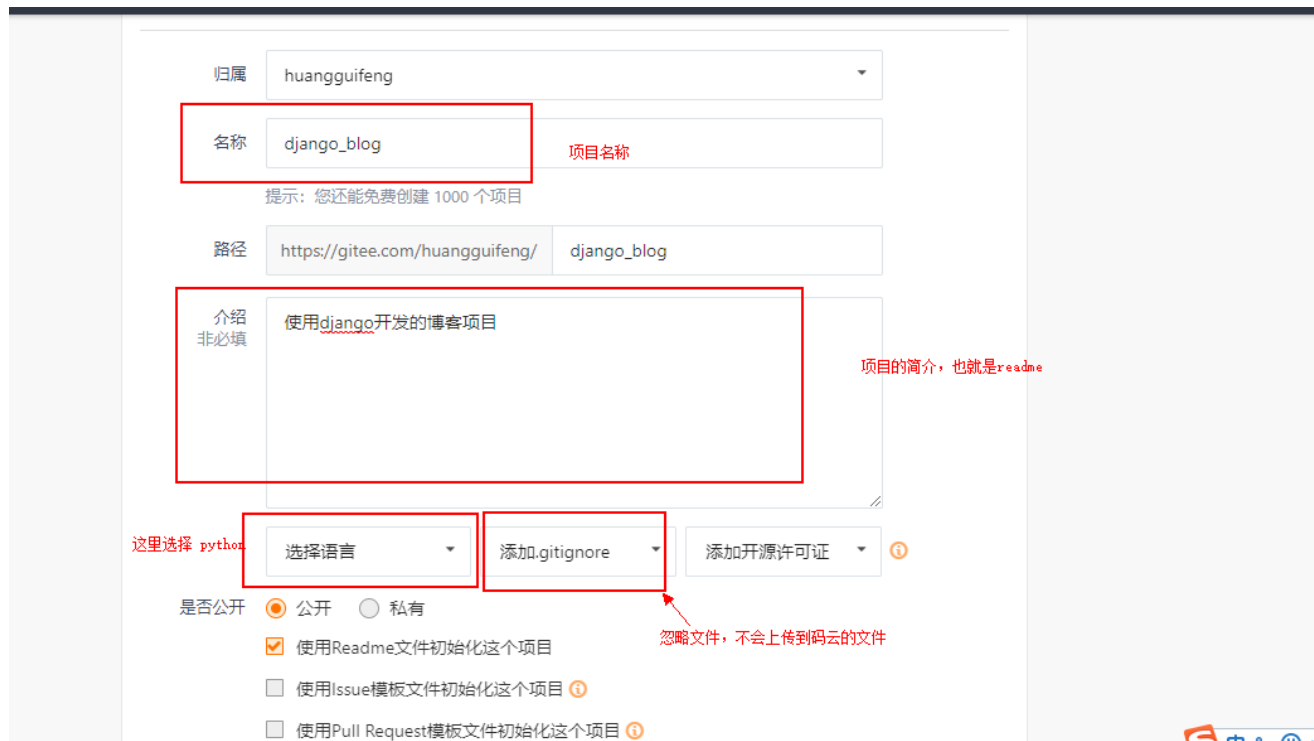
```
git clone git地址 #克隆项目
git add 文件或目录 # 添加
git rm 文件或目录 # 删除暂存区文件
git checkout -- 文件 # 恢复文件
git commit -m '备注说明' # 提交到仓库区
git reset HEAD或版本号 # 回退版本
git relog # 查看版本日志
git log # 查看详细日志
git status # 查看暂存区文件
git branch 分支名称 # 创建分支
git branch --set-upstream-to=origin/分支名称 分支名称 #跟踪分支
git checkout 分支名称 # 切换分支
git checkout -b 分支名称 origin/分支名称 #创建分支同时切换到新分支
git diff 版本1 版本2 # 对比两个版本的不同
git merge 分支名称 # 合并分支代码
git pull #拉取服务器代码
git push origin 分支名称 #推送代码到服务器
git tag 标签名称 # 打标签
```

创建仓库

登录码云，点击概览---创建项目



创建项目



3、添加SSH账户

如果某台机器需要与码云上的仓库交互，那么就要把这台机器的ssh公钥添加到这个码云账户上 点击账户头像后的下拉三角，选择'设置'



点击ssh公钥:



生成git密钥:

ubuntu中 删除~/.ssh目录, 这里可能存储了旧的密钥

```
rm -r ~/.ssh
```

执行下面命令:

```
ssh-keygen -t rsa -C "码云账号, 邮箱地址"
```

```
python@python:~$ ssh-keygen -t rsa -C "huanggui0915@foxmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/python/.ssh/id_rsa):
Created directory '/home/python/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/python/.ssh/id_rsa.
Your public key has been saved in /home/python/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:1/oCI1TUt2XIMZxH2xkvBjsM0lqlcN7M6AISDIwFVJM huanggui0915@foxmail.com
The key's randomart image is:
+---[RSA 2048]---+
|**+O..0.00+=..|
|...E.+.0.0*++0+|
|. . .B+.+=+.0|
|. . .+..0. .|
|.O S. .|
|.O. .|
|.O. .|
|. .|
|. .|
|. .|
+---[SHA256]-----+
```

图中1：可以填写保存密钥的目录，留空默认生成在家目录下的 `./ssh`

图中2：可以填写密码，如果填写，一般为项目的名称，后续操作时会要求填写此密码

图中3：重复上一次密码

4、查看公钥

```
python@python:~$ cd .ssh/
python@python:~/ssh$ ls
id_rsa id_rsa.pub
```

公钥名称为 `id_rsa.pub` 私钥名称为

`id_rsa`复制这一段公钥信息

```
python@python:~/ssh$ cat id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDa6+yqYFaGAL9rs9R7+HKXWsKBD7STiuARsCMr/Jp4lawBjpmBZGr3AzTOSoUiezWA
YRz8iYQrDojj1FXdAeS6Gg5t5WR8E5JokXqbsj2COTSa8vbClBA/oOPnmZlznNvIypEUso6irtLawBi/5jxtRz/96dNCDzto
3Icp69hRpHJKU/fEBNh7k1wX34rIhANC+KaGIidqg9h31mZD2MZZJxb/u7NBtitjyQPKeaV2lJY4EUveCc3YpC+iretfjqj
CaICh6PYGalgaN42gYXculSLA+4J20FZ1+laQv5S3FgvnocaAv0koL95PY6LaFZgwMe7fv7LnoweM9CiicYp
huanggui0915@foxmail.com
```

将复制的公钥填写到码云的ssh公钥信息中

H

huangguifeng

加入时间 4小时前

个人资料

修改账户

通知设置

SSH公钥

代码风格

第三方应用

升级为组织

升级为企业版

捐赠管理

SSH公钥

使用SSH公钥可以让你在你的电脑和码云通讯的时候使用安全连接（Git的Remote要使用SSH地址）

您当前的SSH公钥数: 0

你还没有添加任何SSH公钥

添加公钥

标题

huanggui0915@foxmail.com

公钥

把你的公钥粘贴到这里，查看 怎样生成公钥

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDA6+yqYFaGAL9rs9R7+HKXWsKBD7STiuARsCMr/Jp4lawBjp
mBZGr3AzTOSoUiezWAYRz8iYQrDojj1FxdAeS6Gg5t5WR8E5JokXqbsj2COTSa8vbCIBA/oOPnmZlZrNvlypEUso6ir
tLawBi/5jxtRz/96dNCDzto3lcp69hRpHJKU/fEBNh7k1wX34rlhANC+KaGlidqg9h31mZD2MZZJxb/u7NBtitjyQPK
eaV2IJY4EUveCc3YpC+iretfqjCalCh6PYGalgaN42gYXculSLA+4J20FZ1+laQv5S3FgvnocaAvOkol95PY6LaFZgw
Me7fv7LnoweM9CiicYp huanggui0915@foxmail.com

确定

如果在windows或者其他环境安装请参考：[参考git使用文档](#)

[windows生成秘钥参考](#)

5、克隆项目

添加好ssh公钥之后，回到码云的个人主页--概览：

安全 | <https://gitee.com/huangguifeng>

应用 Redis qt - eric6 can't com Django 文档 - Django 外键、多对 Django模型之Meta 用 eric6 与 PyQt5 Pyqt QComboBox django实现web分页

码云 开源项目 代码片段 企业版 我的码云 搜索项目、代码片段...

H

huangguifeng

huangguifeng

0 Followers 0 Stars 0 Following 1 Watches

huangguifeng 很懒，啥也没写

<https://github.com/huangguifeng>

加入于 4小时前

扫码查看2017年个人年终盘点

项目 组织 企业

Public Forks Private

Search...

概览 动态 项目 代码片段 PR 任务

热门项目

自定义精选项目

huangguifeng / django_blog

使用django开发的博客项目

1 0 0

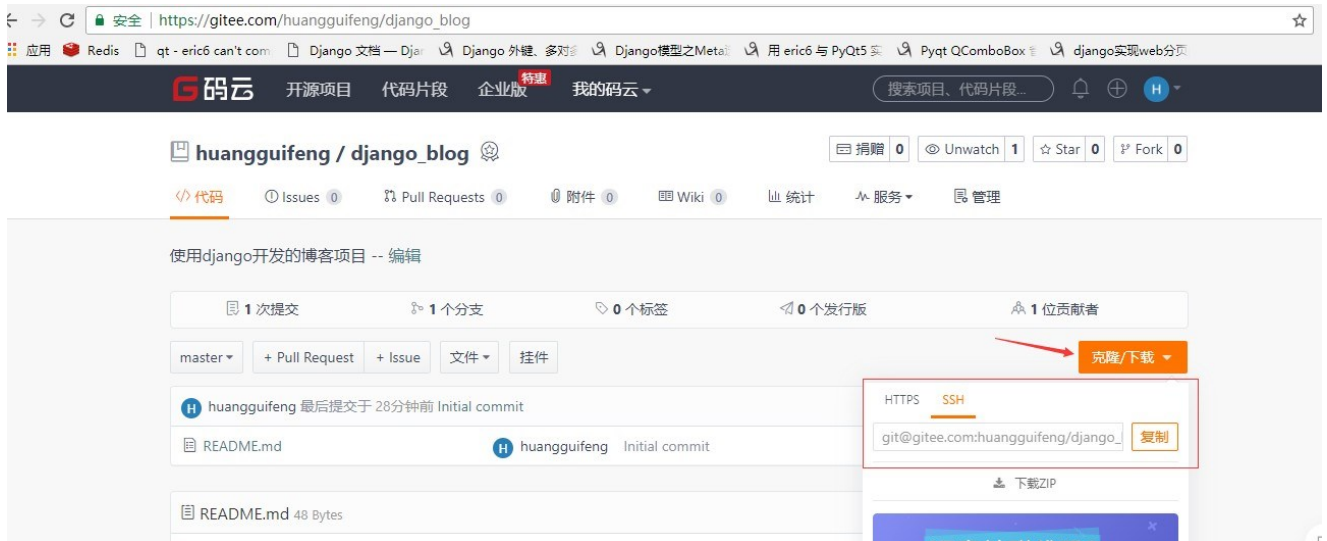
贡献度

1 2 3 4 5 6 7

最近一年贡献: 2次 最长连续贡献: 1日 最近连续贡献: 1日

贡献度的统计数据包括代码提交、创建任务 / Pull Request、合并 Pull Request，其中代码提交的次数需本地配置的 git 邮箱与码云账号邮箱一致才会被统计。

可以看到我们刚才创建的项目，点击项目进到项目详情页，点击克隆可以看到有项目地址，将ssh地址复制下来。



将码云仓库中的项目克隆到本地

```
git clone 项目地址
```

创建一个 django_web文件夹保存项目

```
python@python:~$ mkdir django_web
python@python:~$ cd django_web/
python@python:~/django_web$ git clone git@gitee.com:huangguifeng/django_blog.git
```

第一次连接会提示我们是否连接 输入yes, 可以看到刚才在码云上创建的项目文件已经克隆下来。

```
python@python:~/django_web$
python@python:~/django_web$ ls
django_blog
python@python:~/django_web$
```

6、创建项目分支

- 如果是多人协作开发为了代码互不干扰, 并行开发, 则每人使用一条分支
- 项目开发中公用分支包括master、dev
 - 分支master用于发布, 默认分支, 当需要发布时将dev分支合并
 - 分支dev开发阶段性的代码合并, 每个阶段的工作完成后需要进行一次, 控制项目的进度成员分支用于每个项目成员的代码开发, 实现不交叉

进入到项目目录 django_blog

```
cd django_blog
```

查看当前分支以及所有分支

```
git branch
```

前面标 * 表示当前分支

```
python@python:~/django_web/django_blog$ git branch
* master
python@python:~/django_web/django_blog$
```

创建分支

```
git branch 分支名
称例:
git branch dev
```

将分支推送到服务器

```
git push origin 分支名
称例:
git push origin dev
```

```
python@python:~/django_web/django_blog$ git push origin dev
Total 0 (delta 0), reused 0 (delta 0)
To git@gitee.com:huangguifeng/django_blog.git
 * [new branch]      dev -> dev
python@python:~/django_web/django_blog$
```

将本地分支跟踪服务器分支,跟踪服务器分支后才能push

```
git branch --set-upstream-to=origin/分支名称 分支名
称例:
git branch --set-upstream-to=origin/dev dev
```

创建并切换分支

```
git checkout -b 分支名
称例:
git checkout -b dev
```

删除分支, 这个删除了就找不回来的, 慎用。

```
git checkout -d 分支名称
```

在项目中我们一般会创建一个dev分支, 还有一个自己的分支。刚才已经创建了dev, 这里我们在创建一个自己的分支。

```
git branch it
```


7、搭建项目框架

当前项目分支一共有3个，分别为master、dev、it，当前在it分支上工作

```
python@python:~/桌面/django项目/single_blog$ git branch
dev
* it
master
python@python:~/桌面/django项目/single_blog$
```

在克隆的目录下创建django项目。 如果使用虚拟环境需要先进入虚拟环境

```
django-admin startproject single_blog
```

```
(py3) python@python:~/django_web/django_blog$
(py3) python@python:~/django_web/django_blog$
(py3) python@python:~/django_web/django_blog$
(py3) python@python:~/django_web/django_blog$
(py3) python@python:~/django_web/django_blog$
(py3) python@python:~/django_web/django_blog$ django-admin startproject single_blog
(py3) python@python:~/django_web/django_blog$ ls
README.md  single_blog
(py3) python@python:~/django_web/django_blog$ a
```

将文件代码添加到暂存区， ./表示目录下的所有文件也可以指定单独文件

```
git add ./
```

将暂存区提交到仓储区

```
git commit -m '搭建django博客框架'
```

如果提交出现下面错误那是git用户信息没有配置：

```
(py3) python@python:~/django_web/django_blog$ git commit -m '项目创建'

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'python@python.(none)')
```

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

```
$ git config --global user.name "it" #你的名字
$ git config --global user.email it@example.com #码云账号
```

合并分支，dev分支一般用来合并代码，开发一般在自己的分支，那么就需要将dev的内容合并到it分支
it分支同步dev分支

```
git checkout it

git merge dev
```

推送it分支

```
git push origin it
```

二、协作开发

1、添加ssh账户

一个项目需要多人协同开发的话，其他组员想要向服务器push，也必须先添加ssh公钥到码云。

点击账户头像后的下拉三角，选择'设置'



点击ssh公钥：



生成git密钥:

ubuntu中 删除~/.ssh目录, 这里可能存储了旧的密钥

```
rm -r ~/.ssh
```

执行下面命令:

```
ssh-keygen -t rsa -C "码云账号, 邮箱地址"
```

```
python@python:~$ ssh-keygen -t rsa -C "huanggui0915@foxmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/python/.ssh/id_rsa):
Created directory '/home/python/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/python/.ssh/id_rsa.
Your public key has been saved in /home/python/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:1/oCI1TUt2XIMZxH2xkvBjsM0lqLcN7M6AISDIwFVJM huanggui0915@foxmail.com
The key's randomart image is:
+---[RSA 2048]---+
|**+O..0.00+=..|
|...E.+.0.0*++0+|
|. . .B+.+=+.0|
|. . .+ . . .O .|
|. . .O S . .|
|. . .O . .|
|. . .O . .|
|. . .O . .|
|. . .O . .|
+---[SHA256]-----+
```

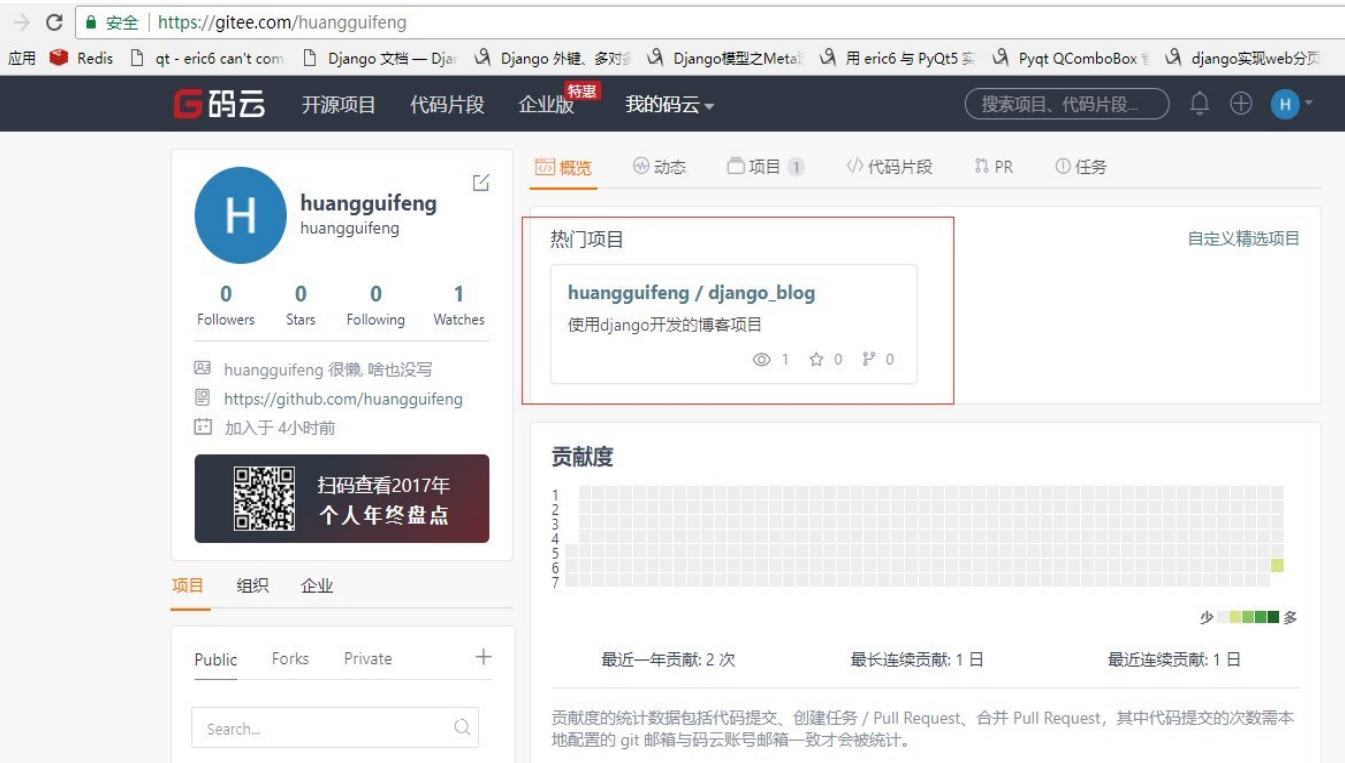
图中1: 可以填写保存密钥的目录, 留空默认生成在家目录下的 ~/.ssh

图中2：可以填写密码，如果填写，一般为项目的名称，后续操作时会要求填写此密码

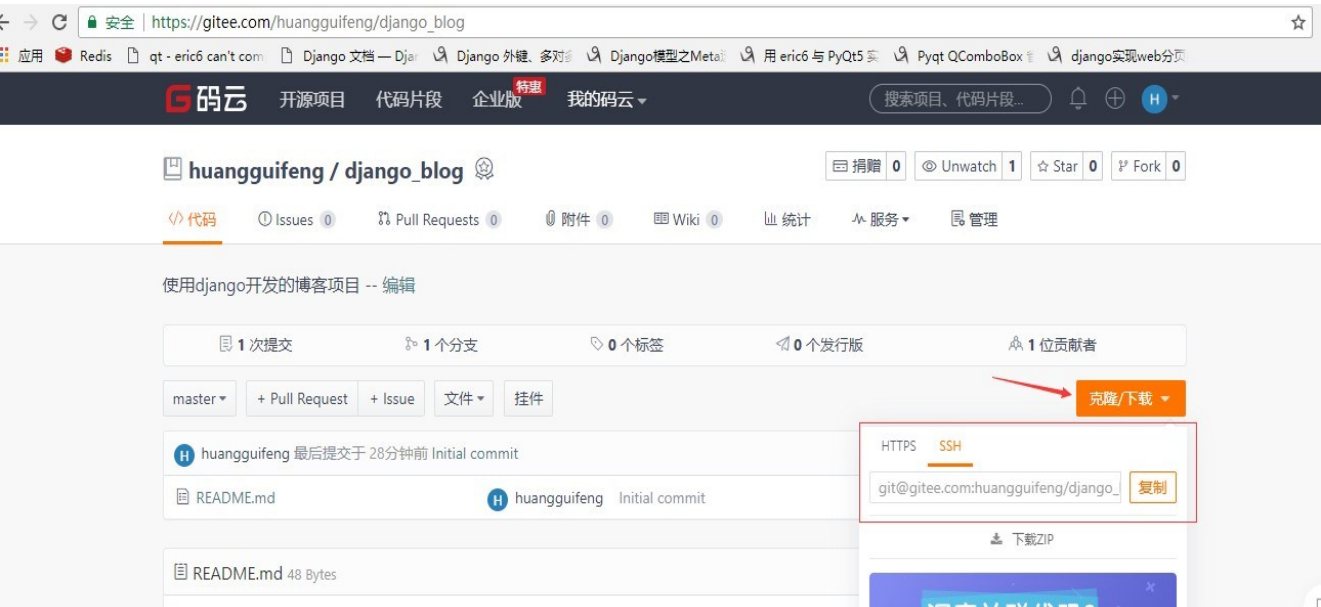
图中3：重复上一次密码

3、克隆项目

添加好ssh公钥之后，回到码云的个人主页--概览：



可以看到我们刚才创建的项目，点击项目进到项目详细页，点击克隆可以看到有项目地址，将ssh地址复制下来。



将码云仓库中的项目克隆到本地

```
git clone 项目地址
```

创建一个 django_web文件夹保存项目

```
python@python:~$ mkdir  
r django_web  
python@python:~$ cd  
django_web/  
python@python:~/django_web$ git clone git@github.com:huangguifeng/django_blog.git
```

第一次连接会提示我们是否连接 输入yes, 可以看到刚才在码云上创建的项目文件已经克隆下来。

```
python@python:~/django_web$  
python@python:~/django_web$ ls  
django_blog  
python@python:~/django_web$
```

2、查看公钥

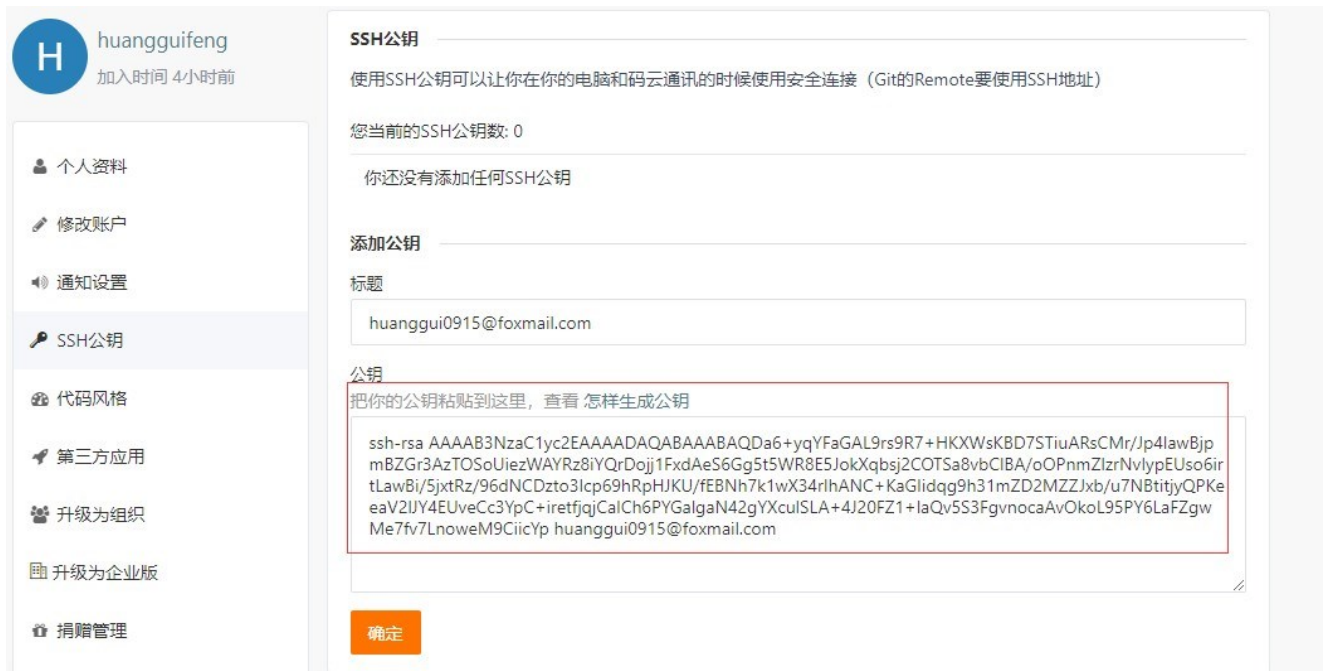
```
python@python:~$ cd .ssh/  
python@python:~/.ssh$ ls  
id_rsa id_rsa.pub
```

公钥名称为id_rsa.pub 私钥名称为

id_rsa复制这一段公钥信息

```
python@python:~/.ssh$ cat id_rsa.pub  
ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQDa6+yqYFaGAL9rs9R7+HKXWsKBD7STiuARsCMr/Jp4lawBjpmBZGr3AzTOSoUiezWA  
YRz8iYQrDojj1FxdAeS6Gg5t5WR8E5JokXqbsj2COTSa8vbC1BA/oOPnmZlZrNvIypEUso6irtLawBi/5jxtRz/96dNCDzto  
3Icp69hRpHJKU/fEBNh7k1wX34rIhANC+KaGIidqg9h31mZD2MZZJxb/u7NBtitjyQPKeeaV2lJY4EUveCc3YpC+iretfjqj  
CaICh6PYGalgaN42gYXculSLA+4J20FZ1+laQv5S3FgvnocaAv0koL95PY6LaFZgwMe7fv7LnoweM9CiicYp  
huanggui0915@foxmail.com
```

将复制的公钥填写到码云的ssh公钥信息中



如果在windows或者其他环境安装请参考：[参考git使用文档](#)

[windows生成秘钥参考](#)

4、同步服务器分支

创建自己的分支，不能与服务
器上的重复。创建agou分支，

```
git checkout -b
```

并切换到agou分支

将本地分支推送到服务器

```
git push origin
```

将本地分支跟踪服务器分支

```
git branch --set-upstream-to=origin/分  
支名称 分支名称例：  
git branch --set-upstream-to=origin/agou agou
```

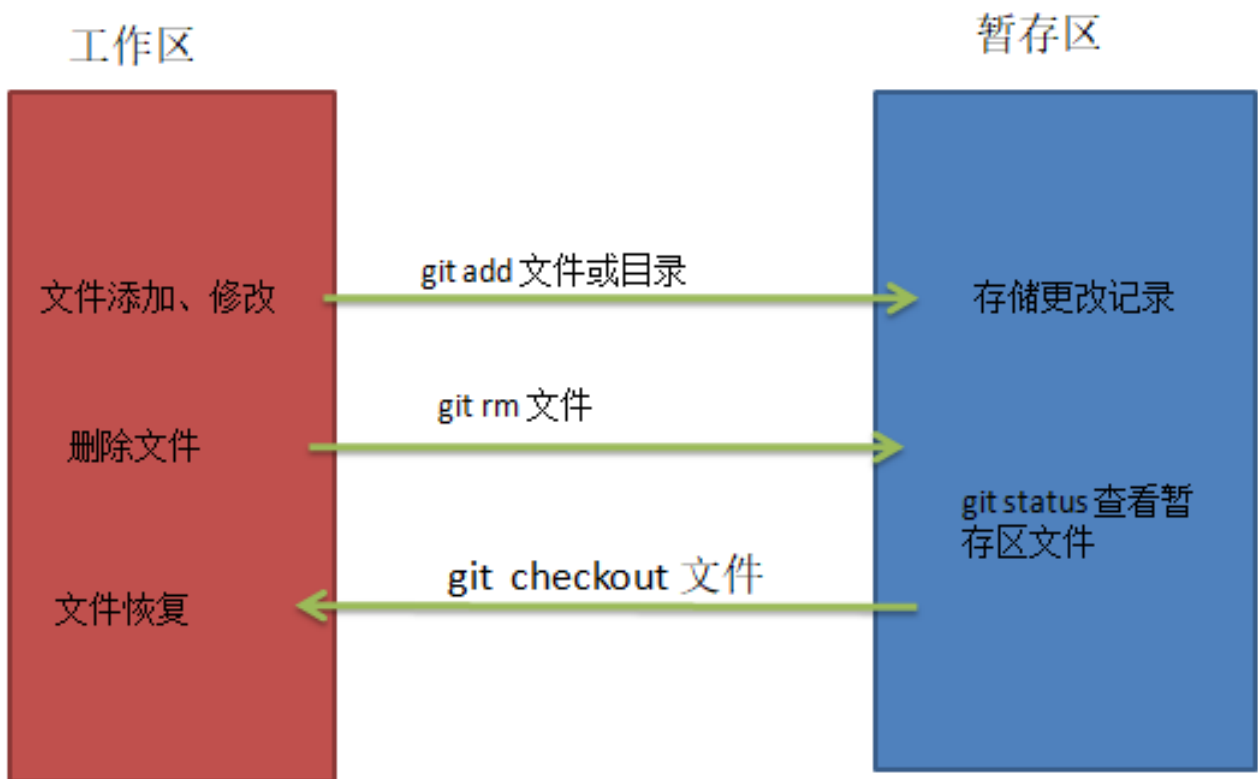
将码云上的dev分支同步到本地，因为开发过程中，所有组员都向这个分支上提交阶段性代码，并从这个分支获取最新代码

```
git checkout -b dev origin/dev
```


5、工作区与暂存区

本地仓库分为三部分：工作区，暂存区，仓库区，其中暂存区、仓库区是版本库部分 对于添加、修改、删除文件的操作，都发生在工作区中

工作区与暂存区交互的方式及命令如下



添加:

创建一个文件git.py写入一个方法，此时文件位于工作区，

```
def add():  
    print('打狗棒法')  
  
def add1():  
    print('打狗棒法')
```

将git.py文件添加到暂存区

```
git add 文  
件1 文件  
2 ... git  
add 目录  
例:
```

撤销 使用暂存区的内容恢复工作区的内容，放弃工作区的更改 将 git.py文件中的 add1方法删除

```
def add():  
    print('git大法好')
```

这个时候工作区add1方法是删除了，暂存区还存在add1方法，如果想回到暂存区的状态，则

```
git  
checkout  
-- 文件名  
例:  
git checkout -- git.py
```

查看git.py文件，可以看到删除的add1方法恢复了。

6、暂存区与仓库区



仓库区存储每个阶段开发提交的记录，仓库区中记录的各版本是可以查看并回退的，但是在暂存区的版本一旦提交暂存区就没有了。

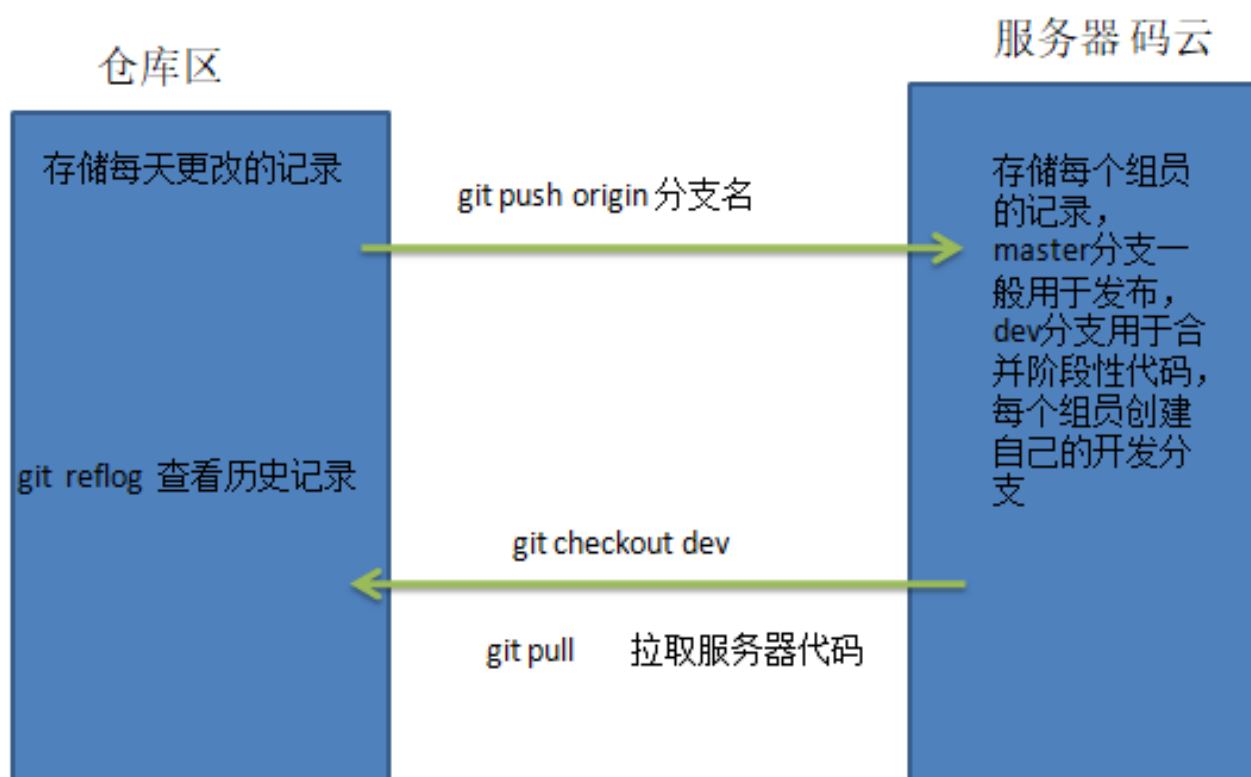
查看未提交的文件

```
git
```

将暂存区的记录提交到仓库区

```
git commit -m '创建两个add方法'
```

7、本地与服务器



推送指将特定分支在本地仓库区的记录发送到服务器上

```
git push origin 分支名称
```

获取指将服务器特定分支向本地工作区同步

```
git checkout dev # 先切换到需要同步的分支
git pull         # 拉取服务器代码
```

同步dev分支到自己的分支 agou 开发

```
git
checkout
agou git
```

建议每天下班前将当前开发的内容推送到服务器，这样确保本机出现意外情况，还有服务器备份代码。要推送前需要先跟踪服务器分支

```
git branch --set-upstream-to=origin/分支名
```

如果要推送自己分支以外的分支，需要先获取，再解决冲突，然后再推送，比如dev分支。

8、分支合并

当一个应用开发完成之后，需要合并到dev

1.切换到dev分支

```
git checkout
```

2.获取代码，如果dev分支上有更新的记录则会同步到本地

```
git
```

3.合并

```
git merge
```

4.添加、提交并推送

```
git add ./
git commit -m
'agou合并分支' git
push origin dev
```

5.切换回工作分支

```
git checkout
```

6.同步dev上最新代码，合并到agou分支，继续开发

```
git merge dev
```

9、合并冲突

在修改公共文件的时候可能会产生冲突，比如大家都可以在dev操作，你修改了git.py 文件，另外一个同事也修改了git.py 文件，就会出现冲突的情况。

比如：开发人员A在dev修改了git.py 先切换到 dev 分支

```
git branch
```

1、添加 add2 方法

```
def add():  
    print('git大法好')  
  
def add1():  
    print('打狗棒法')  
  
def add2():  
    print('打狗棒法真的好耶')
```

2、添加到暂存区

```
git add
```

3、提交到仓库区

```
git commit -m '添加add2方法'
```

4、同步到服务器

```
git push origin dev
```

开发人员B也在dev分支修改了 git.py

1、先克隆 项目

```
git clone git@gitee.com:huangguifeng/django_blog.git
```

2、进入到项目目录

```
cd cd
```

3、创建dev分支

```
git branch
```

4、跟踪服务器上的dev分支

```
git branch --set-upstream-to=origin/dev
```

5、同步服务器上的代码

```
git
```

6、修改git.py

```
def add():  
    print('git大法好')  
  
def add1():  
    print('打狗棒法')  
  
def add2():  
    print('打狗棒法真的好耶')  
  
def add3():  
    print('少林功夫好')
```

7、添加到暂存区

```
git add
```

8、提交到仓库区

```
git commit -m '添加add2方法'
```

9、同步到服务器

```
git push origin dev
```

这里可能都不会产生冲突，但是如果A发现刚才改的方法好像有点问题，回去重新修改下，再提交就会提示：更新拒绝，原因是有人修改过了，需要先同步。提示我们使用 `git pull`


```

python@python:~/django_web/django_blog$ git push origin dev
To git@gitee.com:huangguifeng/django_blog.git
! [rejected]        dev -> dev (fetch first)
error: 无法推送一些引用到 'git@gitee.com:huangguifeng/django_blog.git'
提示: 更新被拒绝, 因为远程仓库包含您本地尚不存在的提交。这通常是因为另外
提示: 一个仓库已向该引用进行了推送。再次推送前, 您可能需要先整合远程变更
提示: (如 'git pull ...')。
提示: 详见 'git push --help' 中的 'Note about fast-forwards' 小节。
python@python:~/django_web/django_blog$

```

同步服务器dev分支:

```
git
```

同步之后就会提示我们那个文件有冲突。我们需要找到有冲突的文件手动修改冲突。

```

python@python:~/django_web/django_blog$ git pull
自动合并 git.py
冲突 (内容): 合并冲突于 git.py
自动合并失败, 修正冲突然后提交修正的结果。
python@python:~/django_web/django_blog$

```

打开文件git.py

```

def add():
    print('git大法好')

def add1():
    print
('打狗棒法
') def
add2():
<<<<<<< HEAD
    print('打狗棒法真的好耶, 真的')
=====
    print('打狗棒法真的好耶')

def add3():
    print('少林功夫')
\>>>>>>> 8f81b29cf10eb82211a68b5c9810e58470240e20

```

<<<<<<< HEAD 表示当前版本的内容, ===== 后面, 表示 >>>>>>> ae79e1fd93d0d9e7f8ca36481c611a2b4a38a9db版本的内容, 发现两句代码并不冲突, 都需要保留, 但add2方法有两句print, 如果不是自己写的, 不能确定是否保留, 可以与编写该语句的人员沟通, 当前代码更改后如下

```
def add():
    print('git大法好')

def add1():
    print('打狗棒法')

def add2():
    print('打狗棒法真的好耶，真的')
    print('打狗棒法真的好耶')

def add3():
    print('少林功夫')
```

冲突解决完成再次提交推送服务器，就不会出错了。

```
git add ./
git commit -m '修改
add2方法，解决突出'
```

10、历史版本

查看历史版本记录

```
git
```

```
python@python:~/django_web/django_blog$ git reflog
8a79db5 HEAD@{0}: commit (merge): 修改add2方法，解决突出
15a237e HEAD@{1}: commit: 修改add2方法
b9f37d3 HEAD@{2}: commit: 添加add2方法
469cec7 HEAD@{3}: checkout: moving from dodi to dev
469cec7 HEAD@{4}: merge dev: Fast-forward
858661b HEAD@{5}: checkout: moving from dev to dodi
469cec7 HEAD@{6}: commit: 创建项目
858661b HEAD@{7}: checkout: moving from dodi to dev
858661b HEAD@{8}: checkout: moving from dodi to dodi
858661b HEAD@{9}: checkout: moving from dev to dodi
858661b HEAD@{10}: checkout: moving from master to dev
858661b HEAD@{11}: clone: from git@gitee.com:huangguifeng/django_blog.git
python@python:~/django_web/django_blog$
python@python:~/django_web/django_blog$
python@python:~/django_web/django_blog$
```

HEAD表示当前版本，也就是最新的提交的版本，上一个版本就是HEAD^，再上一个版本就是HEAD^^，也可以使用 HEAD~2，接数字表示第几个版本。

对比工作区和仓库区中某版本某文件的不同：

```
git
diff
HEAD --
文件名
例:
git diff HEAD^ -- git.py # 对比上一个版本, 与当前工作区的内容的区别
```

```
python@python:~/django_web/django_blog$ git diff HEAD^ git.py
diff --git a/git.py b/git.py
index fdad015..bb13d25 100644
--- a/git.py
+++ b/git.py
@@ -5,3 +5,7 @@ def add1():
    print('打狗棒法')
    def add2():
        print('打狗棒法真的好耶, 真的')
+       print('打狗棒法真的好耶')
+
+def add3():
+    print('少林功夫')
```

版本回退

```
git
reset
HEAD^或版
本号例:
git reset HEAD^
或者
```

```
python@python:~/django_web/django_blog$ git reset HEAD^
重置后取消暂存的变更:
M       git.py
python@python:~/django_web/django_blog$
```

git reset 不是直接回退到工作区, 而是先回退到暂存区。可以使用 git status 查看暂存区内容。

```
python@python:~/django_web/django_blog$ git status
位于分支 dev
您的分支落后 'origin/dev' 共 2 个提交, 并且可以快进。
(使用 "git pull" 来更新您的本地分支)
尚未暂存以备提交的变更:
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

    修改:       git.py

修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
python@python:~/django_web/django_blog$
```

暂存区的内容恢复到工作区

```
git checkout -- git.py
```

恢复之后查看git.py的内容就是前一个版本的内容。

```
def add():  
    print('git大法好')  
  
def add1():  
    print  
( '打狗棒法  
' ) def  
add2():  
    print('打狗棒法真的好耶，真的')
```

三、项目发布

1、项目合并

当项目开发完成之后，需要进行项目的合并与发布，每组员将开发的分支逐个合并到dev分支，如果有冲突则解决冲突，在dev上的代码经过测试没有问题后，合并到master分支，完成发布

逐个合并

组员将自己分支上开发的代码，合并到dev分支上

前题：已经完成了自己分支代码的开发并完成添加、提交及推送 1.切换到dev分支

```
git checkout dev
```

2.同步服务器dev代码

```
git pull
```

3.合并，如果有冲突则与上一个组员商量解决冲突

```
git merge agou
```

4.添加git根目录下。

```
git add ./
```

5.提交 git commit -m 'agou合并'

6.推送

```
git push origin dev
```

所有组员都合并完成之后，在dev测试代码没问题则可以同步到master分支

1.切换到dev分支

```
git checkout dev
```

2.获取项目代码，即所有组员合并完成之后的代码。

```
git pull
```

3.切换到master分支

```
git checkout master
```

4.合并dev分支到master分支

```
git merge dev
```

5.将文件添加到暂存区

```
git add .
```

6.提交

```
git commit -m '发布'
```

7、打标签 版本号 v1.0, 起一个容易记住的名字

```
git tag 标签名称
```

8、同步到服务器

```
git push origin master
```