NSD Python1 DAY01

1 案例1：准备python开发环境

1.1 问题

1. 下载最新版本的python3
2. 下载pycharm社区版
3. 安装python3，使其支持Tab键补全
4. 配置pycharm，使其符合自己的习惯

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：下载最新版python3

首先去python官网下载python3的源码包，网址：https://www.python.org/

进去之后点击导航栏的Downloads，也可以鼠标放到Downloads上弹出菜单选择Source code，表示源码包，这里选择最新版本3.6.4，这里选择第一个下载即可，下载的就是源码包：Python-3.6.4.tar.gz，下载好之后上传到linux系统，准备安装，如图-1所示：

Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
Windows help file	Windows	
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86 embeddable zip file	Windows	
Windows x86 executable installer	Windows	

图-1

步骤二：Linux下安装python3

1)python安装之前需要一些必要的模块，如果没有这些模块后来使用会出现一些问题，输入以下命令提前预装依赖包：

Top

```
01. [root@localhost ~]# yum install -y gcc gcc-c++ zlib-devel openssl-devel readline-devel lib
```

释放文件：

```
[root@localhost python]# tar -xzf Python-3.6.4.tar.gz
```

2)进入Python-3.6.4目录：

```
[root@localhost python]# cd Python-3.6.4
[root@localhost Python-3.6.4]#ls        #此时Python-3.6.4文件夹中没有makefile文件
```

3)配置安装目录：

configure是用来进行用户个性配置的工具，--prefix是说软件安装目录设置在哪里，=/usr/local就是你给出的安装目录

```
[root@localhost Python-3.6.4]# ./configure --prefix=/usr/local
[root@localhost Python-3.6.4]# ls        #此时Python-3.6.4文件夹中生成了makefile文件
aclocal.m4     Doc             Makefile        PCbuild         python-config.py
build          Grammar         Makefile.pre    Programs        python-gdb.py
config.guess   Include         Makefile.pre.in pybuilddir.txt  README.rst
config.log     install-sh      Misc            pyconfig.h      setup.py
config.status  Lib             Modules         pyconfig.h.in   Tools
config.sub     libpython3.6m.a Objects         python
configure      LICENSE         Parser          Python
configure.ac   Mac             PC              python-config
```

4)接下来编译源码：

```
[root@localhost Python-3.6.4]# make
```

5)执行安装：

```
[root@localhost Python-3.6.4]# make install
```

Top

整个过程大约5-10分钟，安装成功

步骤三：下载并安装Pycharm社区版

网址：https://www.jetbrains.com/pycharm/download，这里选择下图红框下载即可，下载好之后上传到linux系统，准备安装，如图-2所示：

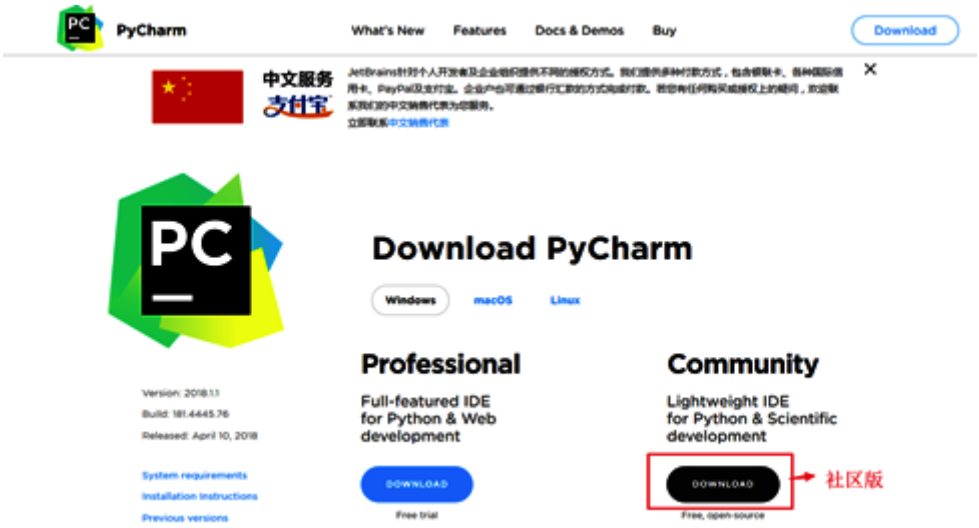


图-2

2)释放文件：

```
[root@localhost ~]# tar -xzf pycharm-community-2018.1.1.tar.gz
```

3)运行下面的命令进入PyCharm 目录：

```
[root@localhost pycharm-community-2018.1.1]# cd pycharm-community-2018.1.1/bin
```

4)通过运行下面的命令来运行PyCharm进入图形化安装界面：

```
[root@localhost bin]# sh pycharm.sh &
```

5)Pycharm打开后，如果你需要导入之前安装版本的设置的话，可以选择第一个选项，如果没有的话，选择(Do not import settings)默认不导入设置，点击/同意，就可以进入pycharm进行配置，如图-3所示：

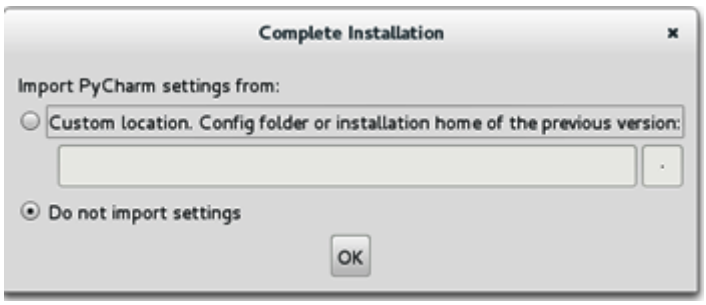


图-3

Top

6)激活Pycharm：在弹出的激活窗口中，选择“License serveer”输入激活服务器地址“http://127.0.0.1:1017”，之后点击‘Activate’，完成pycharm激活，如图-4所示：

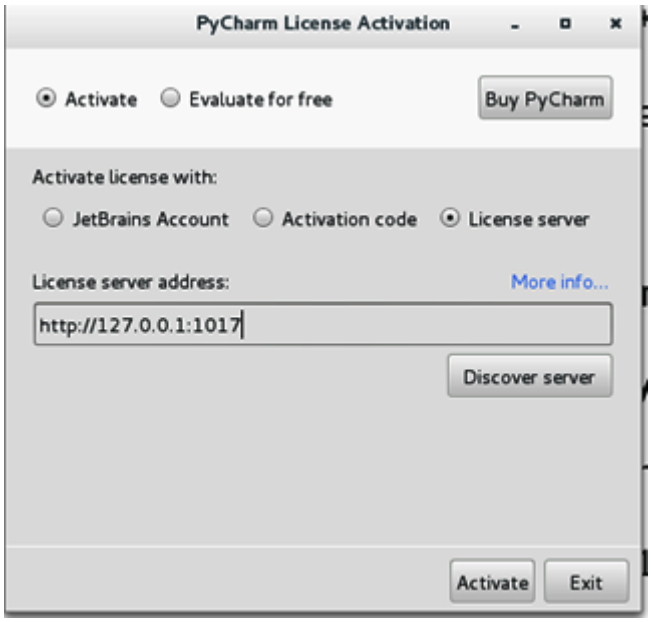


图-4

7)启动完成进入欢迎界面，如图-5所示：

图-5

2 案例2：配置git

2.1 问题

1. 安装git版本控制软件
2. 设置用户信息，如用户名、email等
3. 设置默认编辑器为vim
4. 查看用户配置

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装git版本控制软件

Top

```
[root@localhost ~]# yum install -y git
已安装:
  git.x86_64 0:1.8.3.1-11.el7

作为依赖被安装:
  perl-Error.noarch 1:0.17020-2.el7          perl-Git.noarch 0:1.8.3.1-11.el7
  perl-TermReadKey.x86_64 0:2.30-20.el7

完毕！
[root@localhost ~]# git --version       #查看版本
git version 1.8.3.1
```

步骤二：设置用户信息

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

```
[root@localhost ~]# git config --global user.name "Mr.Zhang"
[root@localhost ~]# git config --global user.email zhangzg@tedu.cn
```

步骤三：设置默认编译器为vim

```
[root@localhost ~]# git config --global core.editor vim
```

步骤四：查看配置

```
[root@localhost ~]# git config --list
user.name=Mr.Zhang
user.email=zhangzg@tedu.cn
core.editor=vim
```

3 案例3：git本地操作

3.1 问题

1. 创建devops目录
2. 为devops创建git仓库
3. 新建文件hello.py，并将文件初始化到仓库中
4. 修改hello.py并将其更新到仓库

Top

5. 从他库中删除hello.py

3.2 方案

Git 使用 git init 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以git init是使用 Git 的第一个命令。添加文件第一步使用git add是将文件添加进暂存区，第二部git commit提交更改，实际上将暂存区的所有内容提交到仓库。

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建目录初始化

此时创建了一个空仓库，可以发现当前目录下有一个隐藏的目录.git，此目录为Git来跟踪管理版本库，建议不要修改内部文件，以免Git仓库遭到破坏。

```
[root@localhost ~]# mkdir devops
[root@localhost ~]# cd devops/
[root@localhost devops]# git init        #通过命令把目录变成Git可以管理的仓库
初始化空的 Git 版本库于 /root/devops/.git/
[root@localhost devops]# git init devops
初始化空的 Git 版本库于 /root/devops/devops/.git/
[root@localhost devops]# ls -a
. .. devops .git
```

步骤二：新建文件hello.py，并将文件初始化到仓库中

1)添加指定文件hello.py到暂存区

```
[root@localhost devops]# echo 'print( "hello world! ") ' > hello.py
[root@localhost devops]# git add hello.py        #将文件添加到暂存区
[root@localhost devops]# git status        #查看状态
# 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：    hello.py
```

Top

2)将暂存区文件初始化到仓库中

```
[root@localhost devops]# git commit -m "初始化仓库"      #把暂存区所有内容提交到分
[master 8e6e22a] 初始化仓库
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
[root@localhost devops]# git status
# 位于分支 master
无文件要提交，干净的工作区
```

步骤三：修改hello.py并将其更新到仓库

```
[root@localhost devops]# echo 'print("done.")' >> hello.py
[root@localhost devops]# git commit -am "向hello.py添加新行"
[master 1ca03d5] 向hello.py添加新行
 1 file changed, 1 insertion(+)
```

步骤四：从库中删除hello.py文件

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除

```
[root@localhost devops]# git ls-files       #查看版本库中文件
hello.py
niha.py
[root@localhost devops]# git rm hello.py
rm 'hello.py'
[root@localhost devops]# git commit -m '删除hello.py'
[master a37ff34] 删除hello.py
 1 file changed, 2 deletions(-)
 delete mode 100644 hello.py
```

4 案例4：使用自建gitlab服务器

4.1 问题

1. 通过docker搭建gitlab服务器
2. 新建群组devops
3. 新建项目core_py
4. 新建用户，他/她在devops组中是主程序员
5. 新用户上传版本库到gitlab
6. 熟悉git远程操作方法

Top

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：通过docker搭建gitlab服务器

1)从ftp://172.40.50.116/pub/docker/images/处获取gitlab_zh.tar镜像文件，导入中文版gitlab镜像

```
[root@localhost devops]# docker load < /path/to/gitlab_zh.tar
a94e0d5a7c40: Loading layer [==================================================>]
88888b9b1b5b: Loading layer [==================================================>]
52f389ea437e: Loading layer [==================================================>]
52a7ea2bb533: Loading layer [==================================================>]
db584c622b50: Loading layer [==================================================>]
62786ff6a243: Loading layer [==================================================>]
71bc04f4b7c7: Loading layer [==================================================>]
26e083d332d8: Loading layer [==================================================>]
2c02e58e96b8: Loading layer [==================================================>]
589c7a23de2a: Loading layer [==================================================>]
44474d2cdcd1: Loading layer [==================================================>]
41c94e16b901: Loading layer [==================================================>]
04cafa6a1534: Loading layer [==================================================>]
Loaded image: gitlab_zh:latest
```

2)将物理主机ssh端口改为2022后，输入如下命令起动容器：

```
[root@localhost devops]# docker run -d -h gitlab --name gitlab -p 443:443 -p 80:80 -p 2
b9dc65e0def51a4d09d2a597b2b929490e972a34f3de993439d2f7cc22039b77
此时端口成功启动
```

步骤二：新建群组

1)在浏览器地址栏中输入启动容器ip地址即可登录GitLab的界面，第一次登录使用的用户名为root，首次登录会强制用户修改密码。密码修改成功后，输入新密码进行登录，如图-6所示：

Top

图-6

2)进入网站后点击菜单栏-工具图标打开管理区域，创建群组，使用群组管理项目和成员，如图-7、图-8所示：

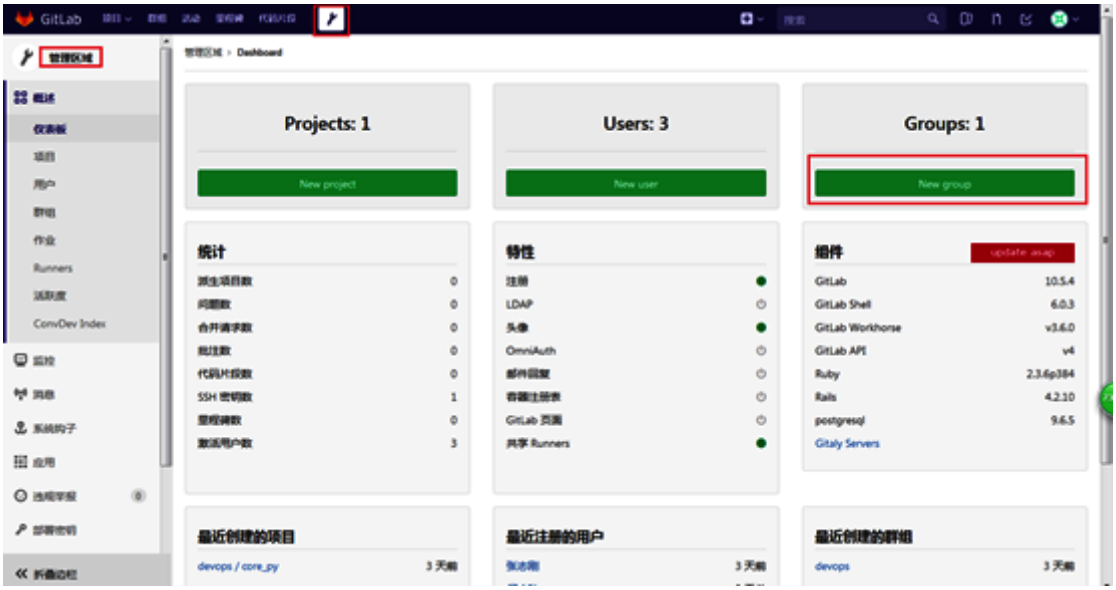
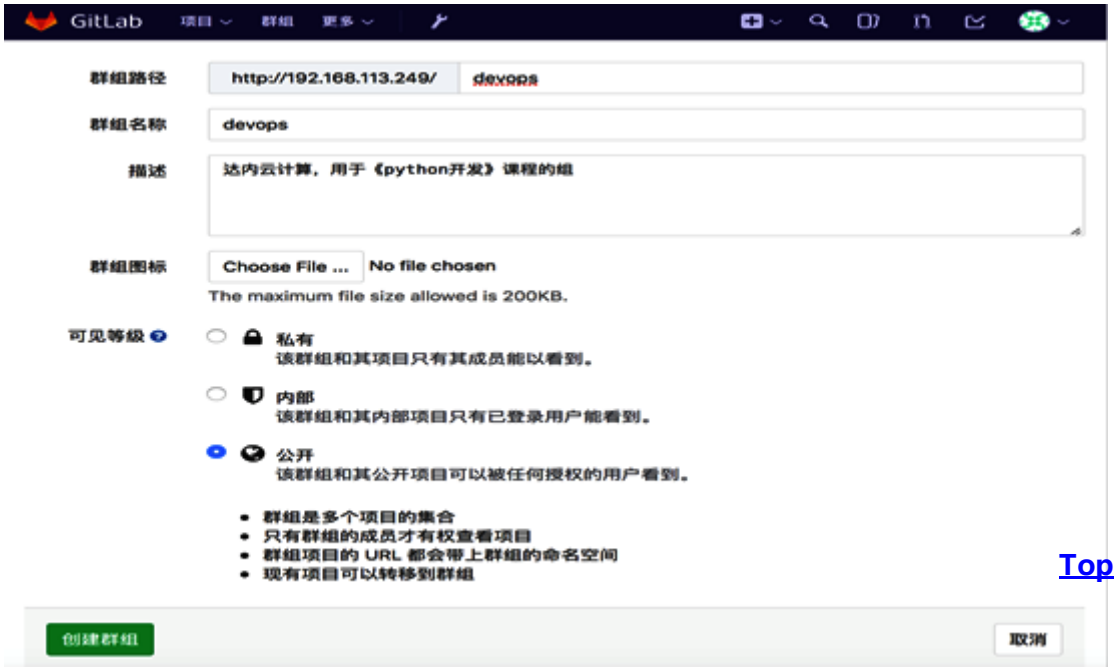


图-7



Top

图-8

步骤三：在Gitlab主页中新建一个项目

显示如图-9所示：

图-9

步骤四：在Gitlab主页中新建一个用户

1)在Gitlab主页中新建一个用户，如图-10所示：

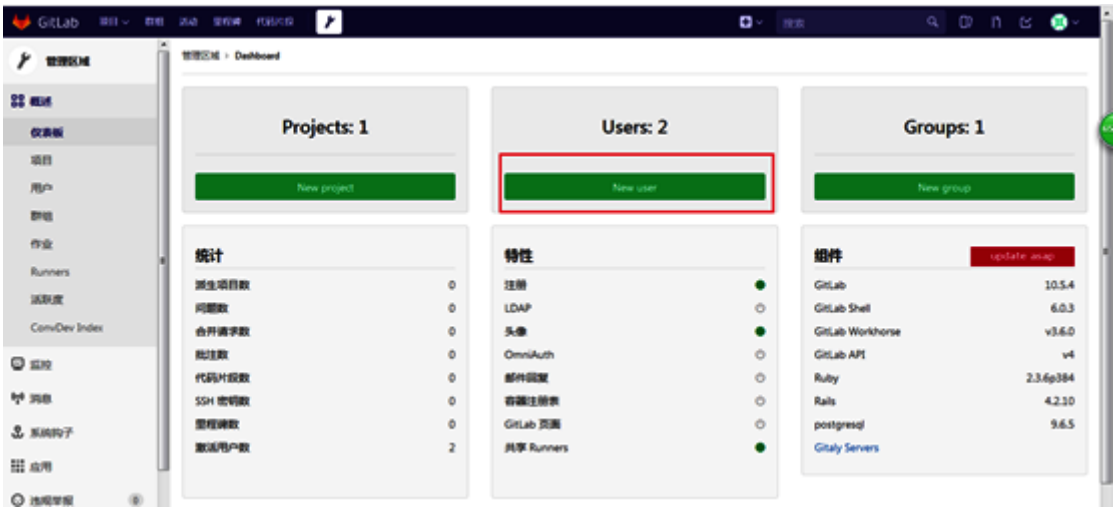


图-10

2)设置账号基本信息，其他均可为默认，如图-11所示：

Top

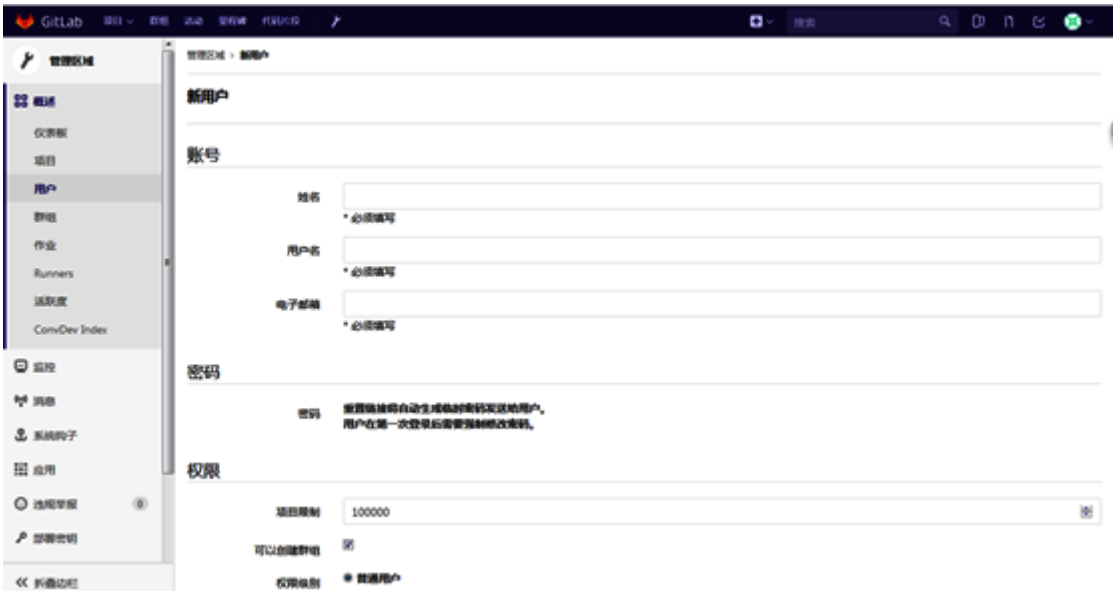


图-11

3)创建用户后，再次编辑可设置密码，如图-12所示：

图-12

4)root用户将新用户加入组中，点击devops进入群组，设置管理权限，在群组中添加成员并设置新成员为“主程序员”，如图-13、图-14、图-15所示：

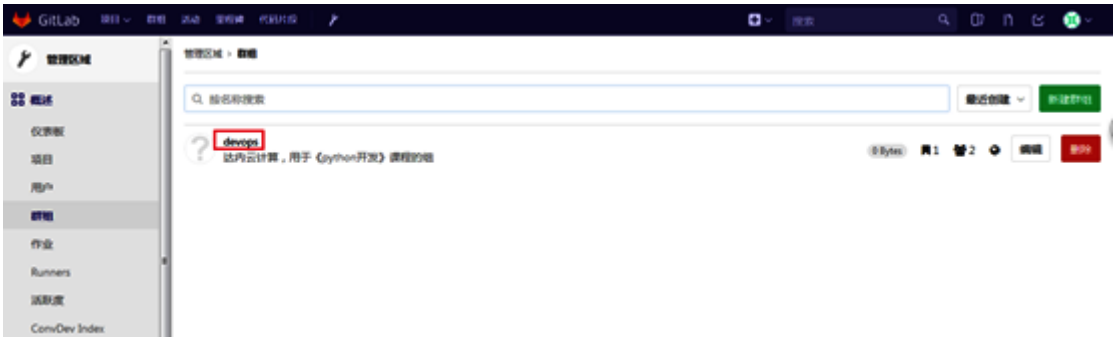


图-13

Top

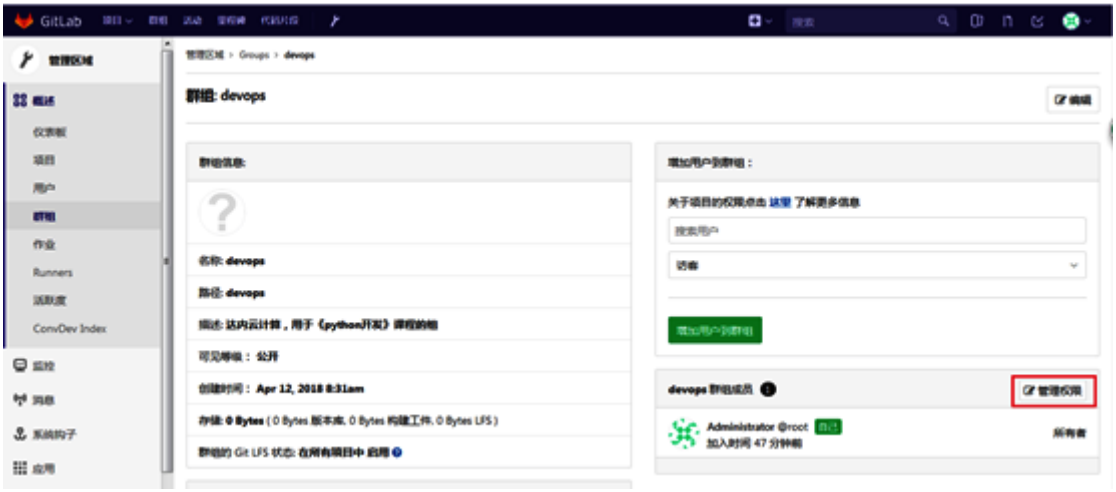


图-14

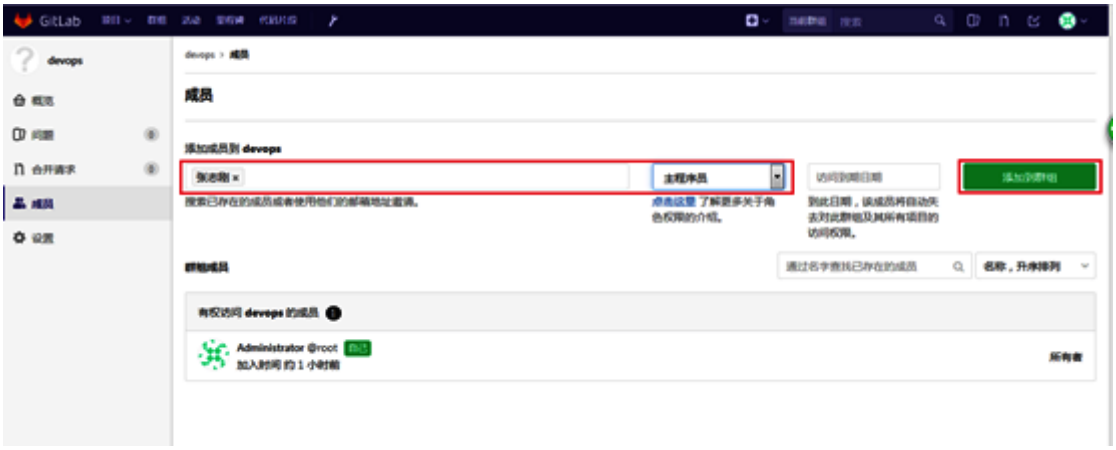


图-15

5)在终端中输入ssh-keygen命令，一路enter，可以生成缺省的rsa方式的sshkey，将/root/.ssh/id_rsa.pub中生成的rsa公钥内容拷到gitlab中，如图-16所示，进入设置页面对ssh进行配置：

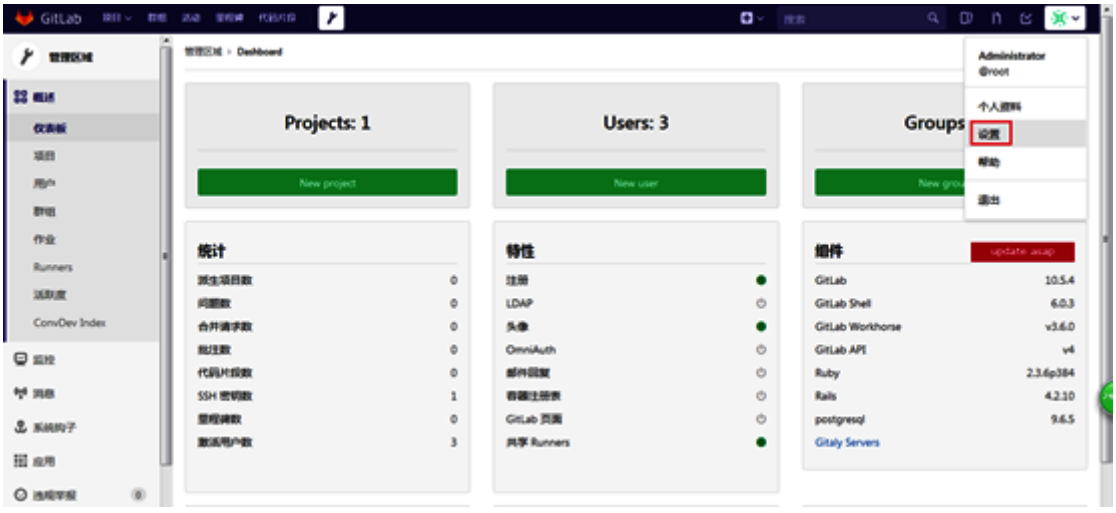


图-16

1. SSH是一种安全协议，在你的电脑与GitLab服务器进行通信时，我们使用SSH密钥（SSH Keys）认证的方式来保证通信安全。你可以在网络上搜索到关于SSH密钥的更多介绍；下面我们重点讲解如何创建 SSH密钥，并将密钥中的公钥添加到GitLab，以便我们通过SSH协议来访问Git仓库。
2. 显示如图-17所示：

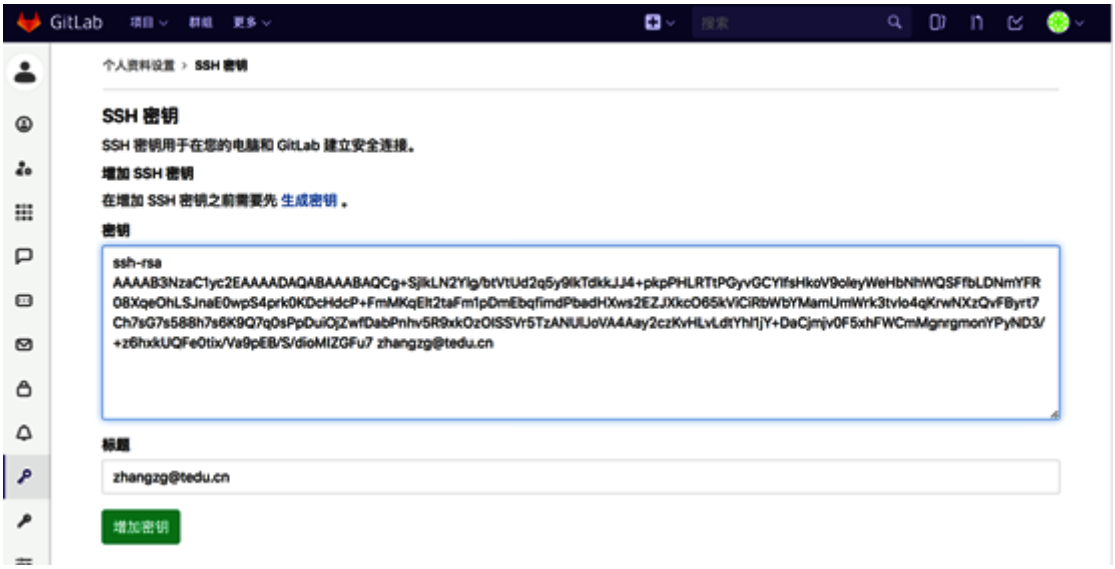


图-17

步骤五：简单远程操作及新用户上传版本库到gitlab

1)克隆远程库代码到本地

```
01.  # git clone git@192.168.113.249:devops/core_py.git
```

2)创建一个文件

```
01.  # cd /root/whsir/whsir
02.  # echo "Hello" > hello.py
```

3)将文件添加到仓库

```
01.  # git add hello.py
```

4)提交文件到仓库，输出信息如图-18所示：

```
01.  # git commit -m "hello文件"
```

Top

```
[root@vm2 whsir]# git commit -m "1"
[master (root-commit) b498a0e] 1
 Committer: root <root@vm2.(none)>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

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

If the identity used for this commit is wrong, you can fix it with:

    git commit --amend --author='Your Name <you@example.com>'

 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

图-18

5)关联远程库

```
01.    # git remote add origin git@192.168.113.249:root/whsir.git
```

6)最后推送到gitlab上，输出信息如图-19所示：

```
01.    # git push origin master
```

```
[root@vm2 whsir]# git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 203 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@192.168.0.80:root/whsir.git
 * [new branch]      master -> master
```

图-19

5 案例5：模拟用户登陆

5.1 问题

编写login.py脚本，实现以下目标：

1. 创建名为login.py的程序文件
2. 程序提示用户输入用户名
3. 用户输入用户名后，打印欢迎用户

5.2 方案

编写程序时，很多情况下都需要程序与用户交互。在python3中，主要通过input()获取用户输入信息，使用print()打印信息。

通常当想看变量内容时，会在代码中使用print()语句输出。不过在交互式解释器中，可以用print语句显示变量的字符串表示，或者仅使用变量名查看该变量的原始值。

Top

从用户那里得到数据输入的最容易的方法是使用input()内建函数。它读取标准输入，并将读取到的数据赋值给指定的变量。需要注意的是，input()函数读入的数据全部是以字符串的方式存储的。如果用户输的是数字，那么python也将其保存为字符串，当将字符串与数字做数学运算是将会出现TypeError的错误。

初学者在需要显示信息或得到用户输入时，很容易想到使用print()语句和input()内建函数。不过在此建议函数应该保持其清晰性，也就是它只应该接受参数，返回结果。从用户那里得到需要的数据， 然后调用函数处理， 从函数得到返回值，然后显示结果给用户。这样你就能够在其它地方也可以使用你的函数而不必担心自定义输出的问题。这个规则的一个例外是，如果函数的基本功能就是为了得到用户输出，或者就是为了输出信息，这时在函数体使用print()语句或input()也未尝不可。更重要的，将函数分为两大类，一类只做事，不需要返回值（比如与用户交互或设置变量的值）， 另一类则执行一些运算，最后返回结果。如果输出就是函数的目的，那么在函数体内使用print()语句也是可以接受的选择。

5.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

本次练习的脚本文件是/root/bin/login.py。

```
[root@localhost day01]# vim login.py
#! /usr/bin/env python3

username = input('username: ')       #使用变量username接收用户输入的字符

print('Welcome', username)              #输出欢迎信息，字符串和变量名之间用逗号
                                    #隔开，两者之间自动会加上空格
```

步骤二：测试脚本执行

```
[root@localhost day01]# python3 login.py
username: bob                       #输入用户名
Welcome bob
```

Top