

Git配置方法

安装

```
1 yum -y install git
```

检查版本

```
1 git version
```

配置

为git添加单独的用户

```
1 groupadd git
2 useradd git -g git # -g 参数后是组名
3 passwd git # 需要设置git用户的密码，比如密码设置为git123456，也可以直接按回车密码设空。
```

禁用shell登录

```
1 vi /etc/passwd
```

修改文件中下面这行的内容

```
1 git:x:500:502::/home/git:/bin/bash
2 # 改为
3 git:x:500:502::/home/git:/usr/bin/git-shell
```

创建共享仓库

```
1 mkdir -p /home/git/woniuboss.git
2 git init --bare /home/git/woniuboss.git # 初始化一个裸仓库
3 chown -R git:git /home/git/woniuboss.git # 改变仓库目录的拥有者
```

登陆证书

- 服务器

避免客户端每次提交时都要输入密码

```
1 mkdir -p /home/git/.ssh
2 chmod 700 /home/git/.ssh
3 touch /home/git/.ssh/authorized_keys
4 chmod 600 /home/git/.ssh/authorized_keys
5 chown -R git:git /home/git/.ssh
```

将客户端公钥id_rsa.pub文件的内容写到服务器端 /home/git/.ssh/authorized_keys 文件里

```
1 vi /etc/ssh/sshd_config
```

修改sshd-config配置文件内容，打开下面内容的注释

```
1  RSAAuthentication yes
2  PubkeyAuthentication yes
3  AuthorizedKeysFile .ssh/authorized_keys
```

重启sshd服务

```
1  service sshd restart
```

• 客户端

生成证书

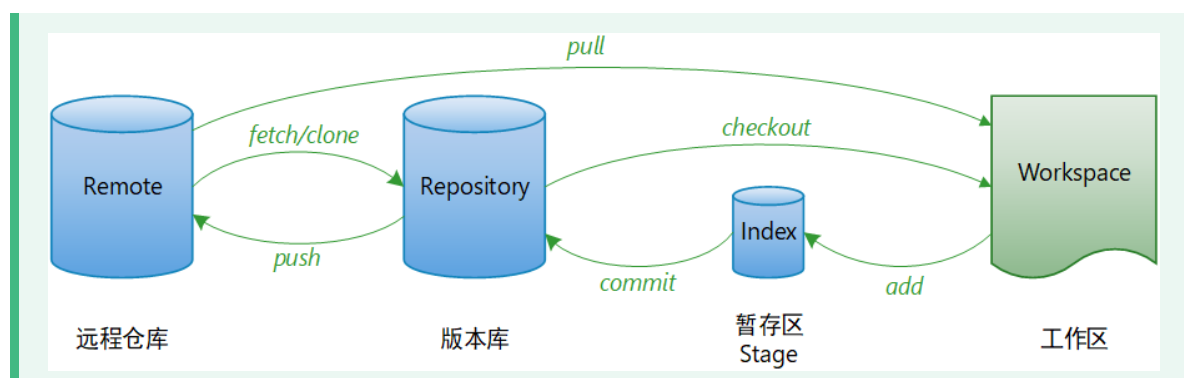
```
1  ssh-keygen -t rsa -C "XXX@XXX.com"
```

执行结果如下

```
1  Generating public/private rsa key pair.
2  Enter file in which to save the key (/c/Users/XXX/.ssh/id_rsa):
3  Enter passphrase (empty for no passphrase):
4  Enter same passphrase again:
5  Your identification has been saved in /c/Users/XXX/.ssh/id_rsa.
6  Your public key has been saved in /c/Users/XXX/.ssh/id_rsa.pub.
7  The key fingerprint is:
8  SHA256:79kTL322MCX5Y/KNC6wr65YPVbo301BSZeBs/WSic/k XXX@XXX.com
9  The key's randomart image is:
10 +---[RSA 2048]---+
11 |                 .oo  |
12 |                 o.o  |
13 |                 .= o o|
14 |                 .+...* |
15 |                S ooooo..|
16 |                 o.o.o+. |
17 |                ..o.=B +E|
18 |                 +o =++0.=|
19 |                oo+*.o+o*o|
20 +---[SHA256]-----+
21
```

其中id_rsa是私钥，id_rsa.pub是公钥，这个需交给git服务器端用于配置。

工作原理



使用方法

- 本地克隆仓库

```
1 git clone git@jacky-vpc:/home/git/woniuboss.git # url的格式: 用户名@服务器地址:仓库路径
2 git remote add origin git@jacky-vpc:/home/git/woniuboss.git # 将本地已经存在的git仓库与刚刚创建的远程仓库关联
```

执行结果如下

```
1 Cloning into 'woniuboss'...
2 git@jacky-vpc's password:
3 warning: You appear to have cloned an empty repository.
```

- 本地提交代码前的信息配置

```
1 git config --global user.name "Your Name" # 配置git用户名字
2 git config --global user.email "email@example.com" # 配置git用户邮箱
```

- WINDOWS下需要再添加一个配置, 否则可能后续Git命令可能会报warning

```
1 warning: LF will be replaced by CRLF in readme.txt. # 这行是windows可能显示的警告信息
2 git config --global core.autocrlf false # 要添加的配置命令
```

- 更新代码

```
1 cd woniuboss # 切换到本地仓库目录
2 touch demo.txt # linux环境在本地仓库中添加一个文件
3 echo text > demo.txt # windows环境在本地仓库中添加一个文件, linux环境添加内容到指定文件中
4 git add -A # 添加未提交的文件到暂存区。后面可以跟文件名, 多个文件用空格分开
5 git commit -m "test commit" # 提交文件
6 git push origin master # 推送文件到远程服务器。第一次推送master分支时, 加上了-u参数, Git不但会把本地的master分支内容推送到远程新的master分支, 还会把本地的master分支和远程的master分支关联起来, 在以后的推送或者拉取时就可以简化命令, git push就行
7 git pull origin master # git pull <远程主机名> <远程分支名>:<本地分支名> 相当于git fetch和git merge两条指令
8 git fetch origin master
9 git merge origin/master # 这两个指令等同于上述git pull指令
```

- 查看仓库状态

```
1 git status # 查看仓库状态
```

- 检查历史

```
1 git log # 查看本地仓库
2 git log origin/master # 查看服务器仓库
3 git reflog # 查看命令历史
```

- 文件比较

```
1 git diff demo.txt # 查看文件修改内容
2 git diff HEAD "HEAD^" # 比较两个版本的不同, 后面可以跟文件名参数
```

- 版本回退

```
1 git reset --hard "HEAD^" # 回退到上一个版本。git是用HEAD来表示当前分支中的当前版本，HEAD^表示上一个版本，HEAD^^表示上上一个版本，以此类推，如果要回退很早的版本就用HEAD@{版本号}
```

- -mixed

默认项。**修改**版本库，**修改**暂存区，**保留**工作区。会清空暂存区的修改。

- -soft

修改版本库，**保留**暂存区，**保留**工作区。只需要commit即可恢复版本回退前的状态。

- -hard

修改版本库，**修改**暂存区，**修改**工作区。回退最彻底的一种。

- 撤销修改

分三种情况

- 场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout --file **或者** 手动修改。
- 场景2：当你不但改乱了工作区某个文件的内容，并且还添加到了暂存区（即已经git add了）时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。
- 场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，可以用版本回退，**不过前提是没有推送到远程库。**

- 分支

- 列出本地分支

```
1 git branch
```

- 查看远程分支

```
1 git branch -r
```

- 查看全部分支

```
1 git branch -a
```

- 创建分支

```
1 git branch dev # dev为新分支的名字，此命令创建分支后仍会停留在当前分支。
```

- 删除分支

```
1 git branch -d dev # 删除分支。如果在分支中有一些未merge的提交，那么会删除分支失败，此时可以使用 git branch -D dev：强制删除dev分支。
```

- 分支改名

```
1 git branch -m oldName newName
```

- Checkout

- 操作文件

```
1  git checkout filename # 放弃单个文件的修改
2  git checkout . # 放弃当前目录下的修改
```

- 操作分支

```
1  git checkout master # 将分支切换到master
2  git checkout -b master # 如果分支存在则只切换分支，若不存在则创建并切换到master分支。
```

- 合并分支

```
1  git merge dev # 将分支dev合并到当前分支
```

- 快速合并

- 非快速合并

