

Git

=====报错:

git push后报错

```
No refs in common and none specified; doing nothing.  
Perhaps you should specify a branch such as 'master'.  
fatal: The remote end hung up unexpectedly  
error: failed to push some refs to 'git@xxxxxxx'
```

此报错意思为git不知道您当前提交的版本

解决办法:

```
#git push origin master //远程存储库上还没有主服务器
```

=====

1 案例1: Git基本操作

1.1 问题

本案例要求先快速搭建好一台Git服务器，并测试该版本控制软件，要求如下：

- 安装Git软件
- 创建版本库
- 客户端克隆版本仓库到本地
- 本地工作目录修改数据
- 提交本地修改到服务器

1.2 方案

实验拓扑如图-1所示，Git工作流如图-2所示。

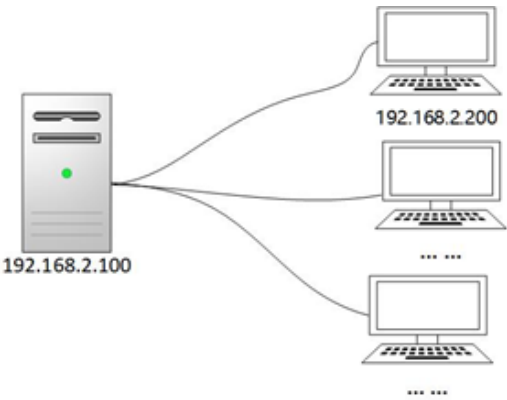


图-1

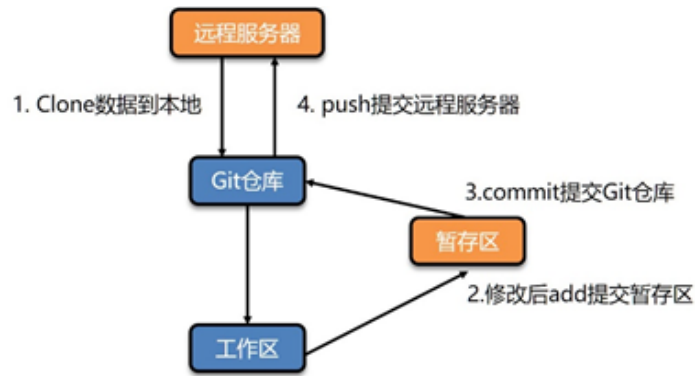


图-2

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：部署Git服务器（192.168.2.100作为远程git服务器）

1) YUM安装Git软件。

1. `[root@web1 ~]# yum -y install git`

2. `[root@web1 ~]# git --version`

2) 初始化一个空仓库。

1. `[root@web1 ~]# mkdir /var/git`

2. `[root@web1 ~]# git init /var/git/project --bare`

3. `[root@web1 ~]# ls /var/git/project`

4. `config description HEAD hooks info objects refs`

步骤二：客户端测试（192.168.2.200作为客户端主机）

使用git常用指令列表如表-1所示。

表-1 git常用指令列表

指令	作用
clone	将远程服务器的仓库克隆到本地
config	修改 git 配置
add	添加修改到暂存区
commit	提交修改到本地仓库
push	提交修改到远程服务器

1) clone克隆服务器仓库到本地。

1. `[root@web2 ~]# yum -y install git`

2. `[root@web2 ~]# git clone root@192.168.2.100:/var/git/project`

3. `[root@web2 ~]# cd project`

4. `[root@web2 ~]# ls`

2) 修改git配置。

1. `[root@web2 project]# git config --global user.email "you@example.com"`

2. `[root@web2 project]# git config --global user.name "Your Name"`

3. `[root@web2 project]# cat ~/.gitconfig`

4. `[user]`

5. `email = you@example.com`
6. `name = Your Name`
- 3) 本地工作区对数据进行增删改查(必须要先进入仓库再操作数据)。

1. `[root@web2 project]# echo "init date" > init.txt`
2. `[root@web2 project]# mkdir demo`
3. `[root@web2 project]# cp /etc/hosts demo`

- 4) 查看仓库中数据的状态。

1. `[root@web2 project]# git status`

- 5) 将工作区的修改提交到暂存区。

1. `[root@web2 project]# git add .`

- 6) 将暂存区修改提交到本地仓库。

1. `[root@web2 project]# git commit -m "注释, 可以为任意字符"`
2. `[root@web2 project]# git status`

- 7) 将本地仓库中的数据推送到远程服务器(web2将数据推送到web1)。

1. `[root@web2 project]# git config --global push.default simple`
2. `[root@web2 project]# git push`
3. `root@192.168.2.100's password: 输入服务器root密码`
4. `[root@web2 project]# git status`

- 8) 将服务器上的数据更新到本地(web1的数据更新到web2)。

备注: 可能其他人也在修改数据并提交服务器, 就会导致自己的本地数据为旧数据, 使用 pull 就可以将服务器上新的数据更新到本地。

1. `[root@web2 project]# git pull`

- 9) 查看版本日志。

1. `[root@web2 project]# git log`
2. `[root@web2 project]# git log --pretty=oneline`
3. `[root@web2 project]# git log --oneline`
4. `[root@web2 project]# git reflog`

备注: 客户端也可以使用图形程序访问服务器。

Windows需要安装git和tortoiseGit。如图-3、图-4所示。



图-3



图-4

2 案例2： HEAD指针操作

2.1 问题

沿用练习一，学习操作HEAD指针，具体要求如下：

- 查看Git版本信息
- 移动指针
- 通过移动HEAD指针恢复数据

2.2 方案

HEAD指针是一个可以在任何分支和版本移动的指针，通过移动指针我们可以将数据还原至任何版本。没做一次提交操作都会导致git更新一个版本，HEAD指针也跟着自动移动。

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：HEAD指针基本操作

1) 准备工作（多对数据仓库进行修改、提交操作，以产生多个版本）。

1. `[root@web2 project]# echo "new file" > new.txt`
2. `[root@web2 project]# git add .`
3. `[root@web2 project]# git commit -m "add new.txt"`
4. `[root@web2 project]# echo "first" >> new.txt`

5. [root@web2 project]# git add .
6. [root@web2 project]# git commit -m "new.txt:first line"
7. [root@web2 project]# echo "second" >> new.txt
8. [root@web2 project]# git add .
9. [root@web2 project]# git commit -m "new.txt:second"
10. [root@web2 project]# echo "third" >> new.txt
11. [root@web2 project]# git add .
12. [root@web2 project]# git commit -m "new.txt:third"
13. [root@web2 project]# git push
14. [root@web2 project]# echo "123" > num.txt
15. [root@web2 project]# git add .
16. [root@web2 project]# git commit -m "num.txt:123"
17. [root@web2 project]# echo "456" > num.txt
18. [root@web2 project]# git add .
19. [root@web2 project]# git commit -m "num.txt:456"
20. [root@web2 project]# echo "789" > num.txt
21. [root@web2 project]# git add .
22. [root@web2 project]# git commit -m "num.txt:789"
23. [root@web2 project]# git push

2) 查看Git版本信息。

1. [root@web2 project]# git reflog
2. [root@web2 project]# git log --oneline
3. 04ddc0f num.txt:789
4. 7bba57b num.txt:456
5. 301c090 num.txt:123
6. b427164 new.txt:third
7. 0584949 new.txt:second
8. ece2dfd new.txt:first line
9. e1112ac add new.txt
10. 1a0d908 初始化

3) 移动HEAD指针，将数据还原到任意版本。

提示：当前HEAD指针为HEAD@{0}。

1. [root@web2 project]# git reset --hard 301c0
2. [root@web2 project]# git reflog
3. 301c090 HEAD@{0}: reset: moving to 301c0
4. 04ddc0f HEAD@{1}: commit: num.txt:789
5. 7bba57b HEAD@{2}: commit: num.txt:456
6. 301c090 HEAD@{3}: commit: num.txt:123

```

7. b427164 HEAD@{5}: commit: new.txt:third
8. 0584949 HEAD@{6}: commit: new.txt:second
9. ece2dfd HEAD@{7}: commit: new.txt:first line
10. e1112ac HEAD@{8}: commit: add new.txt
11. 1a0d908 HEAD@{9}: commit (initial): 初始化
12. [root@web2 project]# cat num.txt          #查看文件是否为123
13. 123
14. [root@web2 project]# git reset --hard 7bba57b
15. [root@web2 project]# cat num.txt          #查看文件是否为123, 456
16. 123
17. 456
18. [root@web2 project]# git reflog          #查看指针移动历史
19. 7bba57b HEAD@{0}: reset: moving to 7bba57b
20. 301c090 HEAD@{1}: reset: moving to 301c0
21. ... ...
22. [root@web2 project]# git reset --hard 04ddc0f #恢复num.txt的所有数据

```

4) 模拟误删后的数据还原操作。

```

1. [root@web2 project]# git rm init.txt      #删除文件
2. rm 'init.txt'
3. [root@web2 project]# git commit -m "delete init.txt" #提交本地仓库
4. [root@web2 project]# git reflog          #查看版本历史
5. 0dc2b76 HEAD@{0}: commit: delete init.txt
6. 7bba57b HEAD@{0}: reset: moving to 7bba57b
7. 301c090 HEAD@{1}: reset: moving to 301c0
8. ... ...
9. [root@web2 project]# git reset --hard 04ddc0f #恢复数据
10. [root@web2 project]# ls
11. demo init.txt new.txt num.txt

```

3 案例3: Git分支操作

3.1 问题

沿用练习二，学习操作Git分支，具体要求如下：

- 查看分支
- 创建分支
- 切换分支
- 合并分支
- 解决分支的冲突

3.2 方案

Git支持按功能模块、时间、版本等标准创建分支，分支可以让开发分多条主线同时进行，每条主线互不影响，分支效果如图-5所示。

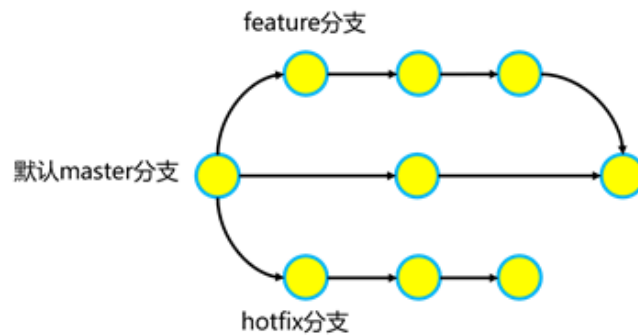


图-5

常见的分支规范如下：

MASTER分支（MASTER是主分支，是代码的核心）。

DEVELOP分支（DEVELOP最新开发成果的分支）。

RELEASE分支（为发布新产品设置的分支）。

HOTFIX分支（为了修复软件BUG缺陷的分支）。

FEATURE分支（为开发新功能设置的分支）。

步骤一：查看并创建分支

1) 查看当前分支。

1. `[root@web2 project]# git status`
2. `# On branch master`
3. `nothing to commit, working directory clean`
4. `[root@web2 project]# git branch -v`
5. `* master 0dc2b76 delete init.txt`

2) 创建分支。

1. `[root@web2 project]# git branch hotfix`
2. `[root@web2 project]# git branch feature`
3. `[root@web2 project]# git branch -v`
4. `feature 0dc2b76 delete init.txt`
5. `hotfix 0dc2b76 delete init.txt`
6. `* master 0dc2b76 delete init.txt`

步骤二：切换与合并分支

1) 切换分支。

1. `[root@web2 project]# git checkout hotfix`
2. `[root@web2 project]# git branch -v`
3. `feature 0dc2b76 delete init.txt`
4. `* hotfix 0dc2b76 delete init.txt`
5. `master 0dc2b76 delete init.txt`

2) 在新的分支上可以继续进行操作（增、删、改、查）。

1. `[root@web2 project]# echo "fix a bug" >> new.txt`
2. `[root@web2 project]# git add .`
3. `[root@web2 project]# git commit -m "fix a bug"`

3) 将hotfix修改的数据合并到master分支。

注意，合并前必须先切换到master分支，然后再执行merge命令。

1. `[root@web2 project]# git checkout master`
2. `[root@web2 project]# cat new.txt` #默认master分支中没有hotfix分支中的数据
3. `[root@web2 project]# git merge hotfix`
4. Updating 0dc2b76..5b4a755
5. Fast-forward
6. `new.txt | 1 ++`
7. 1 file changed, 1 insertions(+)

4) 将所有本地修改提交远程服务器。

1. `[root@web2 project]# git push`

步骤二：解决版本分支的冲突问题

1) 在不同分支中修改相同文件的相同行数据，模拟数据冲突。

1. `[root@web2 project]# git checkout hotfix`
2. `[root@web2 project]# echo "AAA" > a.txt`
3. `[root@web2 project]# git add .`
4. `[root@web2 project]# git commit -m "add a.txt by hotfix"`
5. `[root@web2 project]# git checkout master`
6. `[root@web2 project]# echo "BBB" > a.txt`
7. `[root@web2 project]# git add .`
8. `[root@web2 project]# git commit -m "add a.txt by master"`
9. `[root@web2 project]# git merge hotfix`
10. 自动合并 a.txt
11. 冲突（添加/添加）：合并冲突于 a.txt
12. 自动合并失败，修正冲突然后提交修正的结果。

2) 查看有冲突的文件内容，修改文件为最终版本的数据，解决冲突。

1. `[root@web2 project]# cat a.txt` #该文件中包含有冲突的内容
2. `<<<<<< HEAD`
3. BBB
4. `=====`
5. AAA
6. `>>>>>> hotfix`
7. `[root@web2 project]# vim a.txt` #修改该文件，为最终需要的数据，解决冲突
8. BBB
9. `[root@web2 project]# git add .`

10. [root@web2 project]# git commit -m "resolved"

总结：分支指针与HEAD指针的关系。

- 创建分支的本质是在当前提交上创建一个可以移动的指针
- 如何判断当前分支呢？答案是根据HEAD这个特殊指针

分支操作流程如图-6，图-7，图-8，图-9，图-10所示。



图-6 HEAD指针指向master分支



图-7 切换分支，HEAD指针指向testing分支



图-8 在testing分支中修改并提交代码

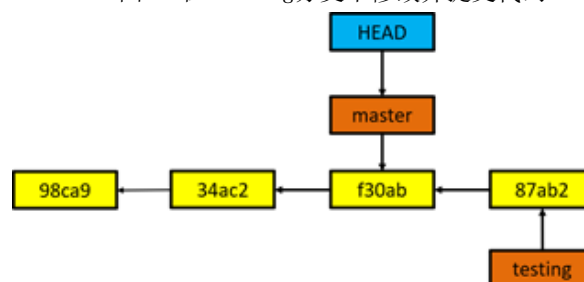


图-9 将分支切换回master分支

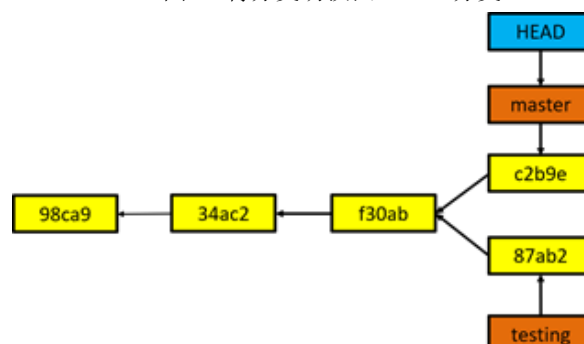


图-10 在master分支中修改数据，更新版本

4 案例4: Git服务器

4.1 问题

沿用练习三，学习Git不同的服务器形式，具体要求如下：

- 创建SSH协议服务器
- 创建Git协议服务器
- 创建HTTP协议服务器

4.2 方案

Git支持很多服务器协议形式，不同协议的Git服务器，客户端就可以使用不同的形式访问服务器。创建的服务器协议有SSH协议、Git协议、HTTP协议。

步骤一：SSH协议服务器（支持读写操作）

1) 创建基于密码验证的SSH协议服务器（web1主机操作）。

1. `[root@web1 ~]# git init --bare /var/git/base_ssh`

2. Initialized empty Git repository in `/var/git/base_ssh/`

2) 客户端访问的方式（web2主机操作）。

1. `[root@web2 ~]# git clone root@192.168.2.100:/var/git/base_ssh`

2. `[root@web2 ~]# rm -rf base_ssh`

3) 客户端生成SSH密钥，实现免密码登陆git服务器（web2主机操作）。

`[root@web2 ~]# ssh-keygen -f /root/.ssh/id_rsa -N ""` //非交互，-f是指定路径，-N 是指密码为空

1. `[root@web2 ~]# ssh-copy-id 192.168.2.100`

2. `[root@web2 ~]# git clone root@192.168.2.100:/var/git/base_ssh`

3. `[root@web2 ~]# git push`

步骤二：Git协议服务器（只读操作的服务器）

1) 安装git-daemon软件包（web1主机操作）。

1. `[root@web1 ~]# yum -y install git-daemon`

2) 创建版本库（web1主机操作）。

1. `[root@web1 ~]# git init --bare /var/git/base_git`

2. Initialized empty Git repository in `/var/git/base_git/`

3) 修改配置文件，启动git服务（web1主机操作）。

1. `[root@web1 ~]# vim /usr/lib/systemd/system/git@.service`

2. 修改前内容如下：

3. `ExecStart=-/usr/libexec/git-core/git-daemon --base-path=/var/lib/git`

4. `--export-all --user-path=public_git --syslog --inetd -verbose`

5. 修改后内容如下：

6. `ExecStart=-/usr/libexec/git-core/git-daemon --base-path=/var/git`

7. `--export-all --user-path=public_git --syslog --inetd -verbose`

8. `[root@web1 ~]# systemctl start git.socket`

4) 客户端访问方式 (web2主机操作)

1. `[root@web2 ~]# git clone git://192.168.2.100/base_git`

步骤三: HTTP协议服务器 (只读操作的服务器)

1) 安装gitweb、httpd软件包 (web1主机操作)。

1. `[root@web1 ~]# yum -y install httpd gitweb`

2) 修改配置文件, 设置仓库根目录 (web1主机操作)。

1. `[root@web1 ~]# vim +11 /etc/gitweb.conf`

2. `$projectroot = "/var/git";` #添加一行

3) 创建版本仓库 (web1主机操作)

1. `[root@web1 ~]# git init --bare /var/git/base_http`

4) 启动httpd服务器

1. `[root@web1 ~]# systemctl start httpd`

5) 客户端访问方式 (web2主机操作)

注意: 调用虚拟机中的firefox浏览器, 需要在远程时使用ssh -X 服务器IP, 并且确保真实主机的firefox已经关闭。

1. `[root@web2 ~]# firefox http://192.168.2.100/git/`

步骤四: 课外扩展知识: 注册使用Github

1. 登陆网站<https://github.com>, 点击Sign up (注册), 如图-11所示。

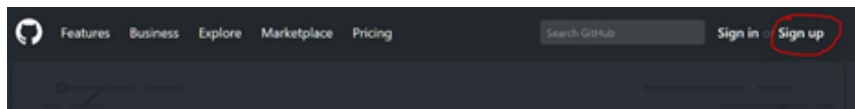


图-11

2. 填写注册信息 (用户名, 邮箱, 密码), 如图-12所示。

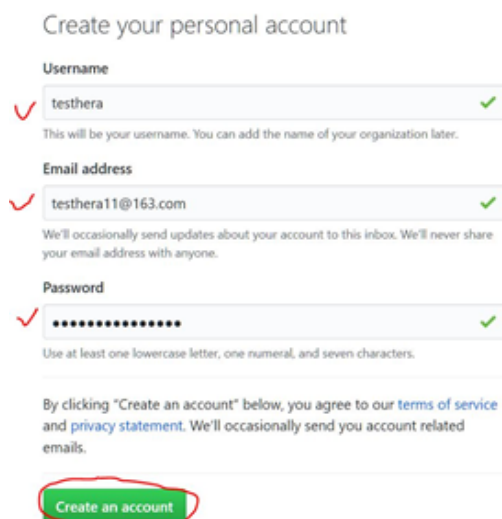


图-12

3. 初始化操作, 如图-13和图-14所示。

Welcome to GitHub

You've taken your first step into a larger world, @tes

✓ Completed

Set up a personal account

🔧 Step 2:

Choose your plan

Choose your personal plan

☒ Unlimited public repositories for free.

☐ Unlimited private repositories for \$7/month. [\(view in CNY\)](#)

Don't worry, you can cancel or upgrade at any time.

☐ Help me set up an organization next

Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees. [Learn more about organizations](#)

☐ Send me updates on GitHub news, offers, and events

Unsubscribe anytime in your email preferences. [Learn more](#)

Continue

图-13

You'll find endless opportunities to learn, code, and create, @testhera.

✓ Completed

Set up a personal account

🔧 Step 2:

Choose your plan

⚙️ Step 3:

Tailor your experience

How would you describe your level of programming experience?

☐ Totally new to programming

☐ Somewhat experienced

☐ Very experienced

What do you plan to use GitHub for? (check all that apply)

☐ Designs

☐ School projects

☐ Research

☐ Project Management

☐ Development

☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a student

☐ I'm a professional

☐ I'm a hobbyist

☐ Other (please specify)

What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

Submit skip this step

不需要选择任何内容, 直接提交即可 (submit)

图-14

注意，初始化完成后，到邮箱中去激活Github账户。

4. 创建仓库、使用仓库

点击Start a project（如图-15所示），

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide

Start a project

图-15

填写项目名称（项目名称任意），如图-16所示。

图-16

往仓库中上传文件或新建文件，如图-17所示

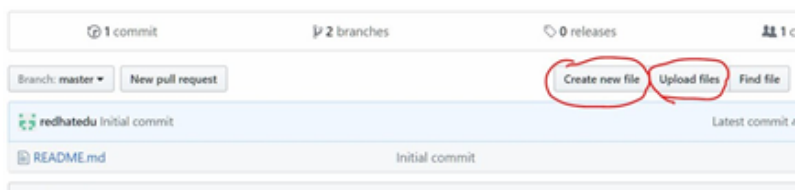


图-17

下载仓库中的代码，如图-18所示。

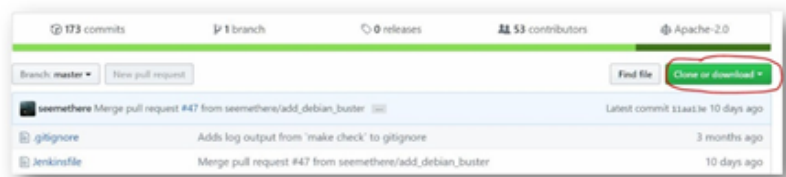


图-18

5. 命令行操作（需要联网的主机，如真实机）

```
[root@pc001 ~]# yum -y install git
```

```
[root@pc001 ~]# git clone https://github.com/账户名称/仓库名称
```

#clone指令用于将服务器仓库中的资料打包下载到本地

```
[root@pc001 ~]# cd 仓库名称
```

```
[root@pc001 ~]# 任意修改文件，或新建文件
```

```
[root@pc001 ~]# git add .
```

#add添加新文件

```
[root@pc001 ~]# git commit -m "test"
```

```
[root@pc001 ~]# git push
```

#commit和push实现提交代码的功能

```
[root@pc001 ~]# git pull
```

#pull可以从github服务器拉取数据到本地

