

第二章 Git远程库(4课时)

★ 本章目标

- 了解：什么是<远程库>？
- 掌握：针对<远程库>数据交互的<协议实现>
- 掌握：针对<远程库>的各项基本操作

1. 了解：什么是<Git远程库>

1.1. 了解：<Git远程库>的作用和特点

★ <Git远程库>的作用：

- <Git远程库>是一个<Git版本库>的集合，主要是为了实现<Git版本库>的资源共享。

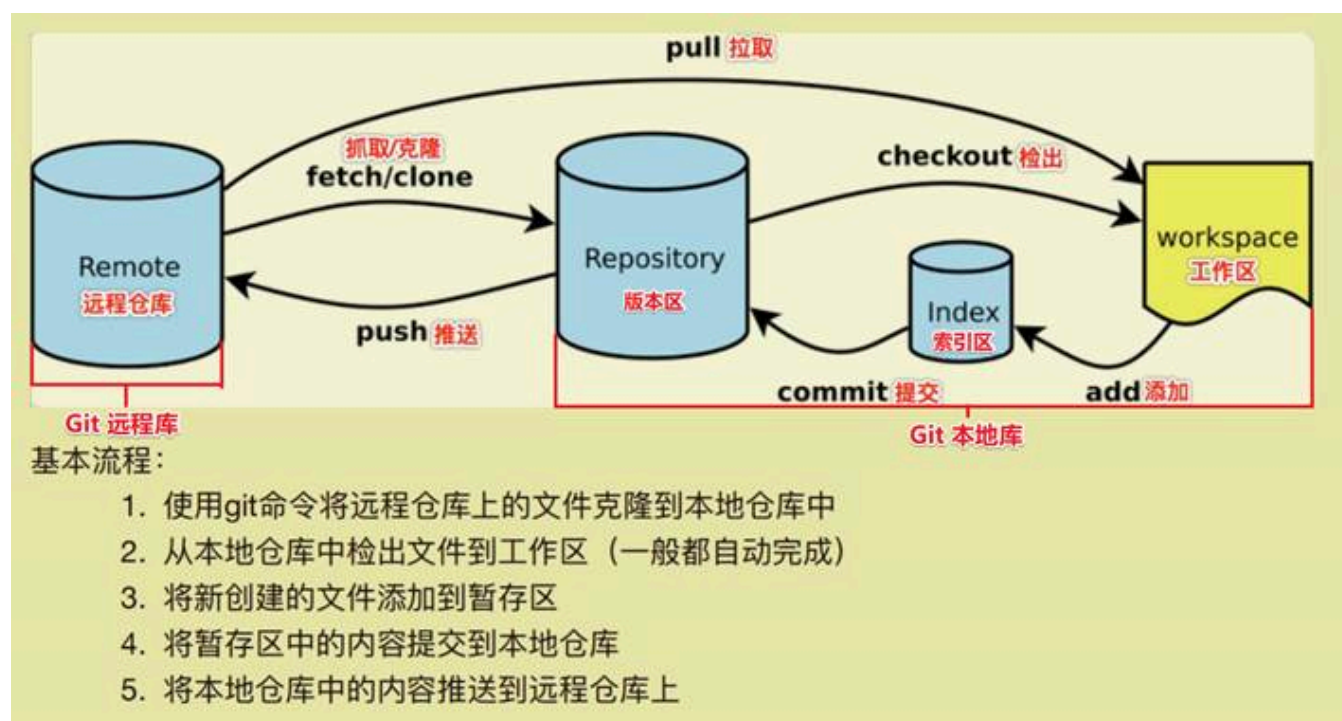
★ <Git远程库>的特点：

- <Git远程库> 通常是一个<裸库>，它只有<Git版本区>，它无需<Git工作区>和<Git索引区>。
- <Git远程库> 可以在<本地>，但通常在<远程>，必须通过<网络协议>进行交互。

<Git远程库> 支持<远程写>，<Git本地库>不支持<远程写>。

1.2. 了解：<Git远程库>的交互方法

★ <Git远程库>的交互方法：



★ <Git远程库>的交互协议：

- git协议

◆ 优点：方便

◆ 缺点：不支持<用户身份验证>

不支持<写入>，仅支持<读取>

• ssh协议

◆ 优点：<Git服务端>默认支持，执行<安全加密传输>，必须执行<用户身份验证>

既支持<写入>，也支持<读取>

◆ 缺点：须在<Git服务端>创建<OS用户>来执行<身份验证>，属于不安全的<用户管理>方式

• http协议

◆ 优点：同时支持<无用户身份验证>和<用户身份验证>，可执行<SSL安全加密传输>

◆ 缺点：须在<Git服务端>部署和配置<Web服务>

既支持<写入>，也支持<读取>

2. 管理：Git远程库

2.1. 构建：<Git远程库>

```
yum install -y git

## <---- /gits 作为: <Git版本库>集合的<基础目录>
mkdir -p /gits/{git01,git02}

cd /gits/git01

## <---- 初始化构建: Git远程库/裸库, --bare选项 表示: 仅构建一个<裸库>
git init --bare

cd /gits/git02

## <---- 初始化构建: Git远程库/裸库, --bare选项 表示: 仅构建一个<裸库>
git init --bare

cd ~
```

2.2. 配置：与<远程库>的<网路协议 交互机制>

2.2.1. 配置：git协议机制

(1) 安装：<git守护进程>

★ YUM 安装 Git 的软件包 简介

- git: Git的最小化依赖关系的核心包
- git-all: Git所有工具的元数据
- git-cvs: 导入CVS版本库的工具
- git-daemon: Git协议的守护进程
- git-email: Git发送email的工具
- git-gui: Git的GUI工具
- git-svn: 导入SVN版本库的工具
- gitk: Git版本树的可见性工具
- gitweb: Git简单的Web界面

```
yum install -y git-daemon
rpm -ql git-daemon
```

```
/usr/lib/systemd/system/git.socket
/usr/lib/systemd/system/git@.service
/usr/libexec/git-core/git-daemon ← 这是：<git 协议>守护进程程序
/usr/share/doc/git-daemon-1.8.3.1
/usr/share/doc/git-daemon-1.8.3.1/git-credential-cache--daemon.html
/usr/share/doc/git-daemon-1.8.3.1/git-credential-cache--daemon.txt
/usr/share/doc/git-daemon-1.8.3.1/git-daemon.html
/usr/share/doc/git-daemon-1.8.3.1/git-daemon.txt
/usr/share/man/man1/git-credential-cache--daemon.1.gz
/usr/share/man/man1/git-daemon.1.gz
/var/lib/git
```

```
cat > /etc/profile.d/git.sh <<EOF
export PATH=/usr/libexec/git-core:$PATH
EOF

source /etc/profile
```

(2) 启动：<git守护进程>

```
git-daemon --verbose --export-all --base-path=/gits --reuseaddr --user=nobody &
```

备注：

- --verbose

- 详细显示：传入的连接和请求的文件的日志信息。
- --export-all
 - 允许访问：该Git远程库，Git协议 不支持 用户身份验证。
 - 如果不带 --export-all选项，则需在 Git远程库 中创建 git-daemon-export-ok空文件，方可访问。
- --base-path=/gits
 - 设置：基础目录路径，该选项，则表示：
 - 允许：Git客户端 仅使用 URL相对路径 来访问 Git远程库
 - 例如：git clone git://Git远程库 服务器 IP地址/**git01**
 - 如果不带 --base-path选项，则表示：
 - 要求：Git客户端 须使用 URL绝对路径 来访问Git远程库
 - 例如：git clone git://Git远程库 服务器 IP地址/**gits/git01**
- --reuseaddr
 - 表示：当我们要重启 Git守护进程 时，不必等待 旧连接 超时，就可以立即重启。
- --user=nobody
 - 表示：采用 nobody用户 来运行 git-daemon守护进程。
- &
 - 表示：在后台运行。

(3) 禁用：SELinux，以避免干扰

```
if ! [[ $(getenforce) == "Disabled" ]]; then
    sed -r -i '/^[ \t]*SELINUX=/c\SELINUX=disabled' /etc/selinux/config
    reboot
fi
```

(4) 配置：开机自启动<git守护进程>

```
yum install -y xinetd
gitBasePath="/gits"

cat > /etc/xinetd.d/git <<EOF
service git
{
    disable = no
    type = UNLISTED
    port = 9418
    socket_type = stream
    wait = no
    user = nobody
    server = /usr/libexec/git-core/git-daemon
    server_args = --base-path=$gitBasePath --export-all --syslog --inetd --verbose
    log_on_failure += USERID
}
EOF
```

```
systemctl enable xinetd
systemctl start xinetd
```

(5) 配置：防火墙

git守护进程 采用：9418/tcp端口

```
netstat -tunlp | grep xinetd
```

```
tcp6      0      0 :::9418          :::*              LISTEN     1301/xinetd
```

```
firewall-cmd --permanent --zone=public --add-port=9418/tcp
firewall-cmd --reload
```

(6) 测试：<Git客户端10.0.0.101>向<Git远程库10.0.0.121>采用<SSH协议>git push推送上传

```
cd ~/myProjectDir
git add -A
git commit -m "这是第一次提交至<我的Git本地库>"
## -----

ssh-keygen
ssh-copy-id root@10.0.0.121
## 注意：使用绝对路径
git remote add gitssh_root ssh://root@10.0.0.121/gits/git01
git remote -v
```

```
gitssh_root      ssh://root@10.0.0.121/gits/git01 (fetch)
gitssh_root      ssh://root@10.0.0.121/gits/git01 (push)
```

```
git push gitssh_root master:master
```

```
Counting objects: 3, done.      基于<SSH 协议>的<git push>推送上传 OK
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 274 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://root@10.0.0.121/gits/git01
 * [new branch]      master -> master
```

(7) 测试：<Git客户端10.0.0.101>从<Git远程库10.0.0.121>采用<Git协议>git clone克隆获取

```
cd ~
rm -rf ~/myProjectDir

## 注意：这里使用的是<相对路径>
git clone -o git_git01 git://10.0.0.121/git01 ~/myProjectDir
```

```
正克隆到 '/root/myProjectDir'... 基于<Git 协议> git clone 克隆获取 OK
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
接收对象中: 100% (3/3), done.
```

2.2.2. 配置：ssh协议机制

(1) 安装：SSHD服务 (无需安装, 哈哈)

(2) 配置：<指定OS用户>针对<Git版本库 目录>具备<读/写能力>

```
## 注意: <OS用户>需要有<自己的home家目录>
useradd user01
echo 'user01:a123456!' | chpasswd
setfacl -R -m u:user01:rwX /gits
setfacl -R -d -m u:user01:rwX /gits
```

(3) 测试：<Git客户端10.0.0.101>向<Git远程库10.0.0.121>采用<SSH协议>git push推送上传

```
cd ~/myProjectDir
echo "222" > 1.html
git add -A
git commit -m "这是第二次提交至<我的Git本地库>"
```

```
ssh-keygen
ssh-copy-id user01@10.0.0.121
```

注意：使用 绝对路径

```
git remote add gitssh_user01 ssh://user01@10.0.0.121/gits/git01
git remote -v
```

```
git_git01      git://10.0.0.121/git01 (fetch)
git_git01      git://10.0.0.121/git01 (push)
gitssh_user01  ssh://user01@10.0.0.121/gits/git01 (fetch)
gitssh_user01  ssh://user01@10.0.0.121/gits/git01 (push)
```

```
git push gitssh_user01 master:master
```

```
Counting objects: 5, done. 基于<SSH 协议>的<git push>推送上传 OK
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 332 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To ssh://user01@10.0.0.121/gits/git01
5be0282..2a088ad master -> master
```


(4) 测试: <Git客户端10.0.0.101>从<Git远程库10.0.0.121>采用<SSH协议>git clone克隆获取

```
cd ~
rm -rf ~/myProjectDir
## 注意: 使用绝对路径
git clone -o gitssh_user01 ssh://user01@10.0.0.121/gits/git01 ~/myProjectDir
```

```
正克隆到 '/root/myProjectDir'...
remote: Counting objects: 6, done. 基于<SSH 协议> git clone 克隆获取 OK
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 1), reused 0 (delta 0)
接收对象中: 100% (6/6), done.
处理 delta 中: 100% (1/1), done.
```

2.2.3. 配置: http协议机制

(1) 安装: HTTPD Web服务

```
yum install -y httpd mod_ssl
systemctl enable httpd
systemctl start httpd
```

(2) 禁用: SELinux, 以避免干扰

```
if ! [[ $(getenforce) == "Disabled" ]]; then
    sed -r -i '/^[ \t]*SELINUX=/c\SELINUX=disabled' /etc/selinux/config
    reboot
fi
```

(3) 创建: HTTP验证用户

```
htpasswd -m -b -c /etc/httpd/conf.d/git.htpasswd test01 123456

htpasswd -m -b /etc/httpd/conf.d/git.htpasswd test02 123456

chown apache:apache /etc/httpd/conf.d/git.htpasswd
```

(4) 设置: <Git远程库>的<虚拟主机>

```
cat > /etc/httpd/conf.d/git.conf <<EOF
Listen 88
<VirtualHost *:88>
    ServerName 10.0.0.121
    DocumentRoot "/gits"
    ## 设置: 允许导出<所有的Git版本库>
    SetEnv GIT_HTTP_EXPORT_ALL
    ## 设置: <所有的Git版本库>的<基础目录>
```

```

SetEnv GIT_PROJECT_ROOT /gits
## 设置: 别名路径
## git-http-backend 是: 一个简单的<CGI程序>, 它被用来实现基于<HTTP协议>
## 对<Git远程库>执行<读/写>操作。
ScriptAlias /gits/ /usr/libexec/git-core/git-http-backend/
<Location />
Options Indexes FollowSymLinks
AllowOverride None
# Require all granted
## 设置: httpd用户身份验证
AuthType Basic
AuthName "Please enter your Git username and password ..."
AuthUserFile /etc/httpd/conf.d/git.htpasswd
Require valid-user
</Location>
</VirtualHost>
EOF

setfacl -R -m u:apache:rwX /gits
setfacl -R -d -m u:apache:rwX /gits
systemctl restart httpd
firewall-cmd --permanent --zone=public --add-port=88/tcp
firewall-cmd --reload

```

(5) 测试: <Git客户端10.0.0.101>向<Git远程库10.0.0.121>采用<HTTP协议>git push推送上传

```

cd ~/myProjectDir
echo "333" > 1.html
git add -A
git commit -m "这是第三次提交至<我的Git本地库>"

```

```

git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
git remote -v

```

```

githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 (fetch)
githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 (push)
gitssh_user01 ssh://user01@10.0.0.121/gits/git01 (fetch)
gitssh_user01 ssh://user01@10.0.0.121/gits/git01 (push)

```

```

git push githttp_test01 master:master

```

```

Counting objects: 5, done. 基于<HTTP 协议>的<git push>推送上传 OK
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 331 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To http://test01:123456@10.0.0.121:88/gits/git01
2a088ad..3ba7cfb master -> master

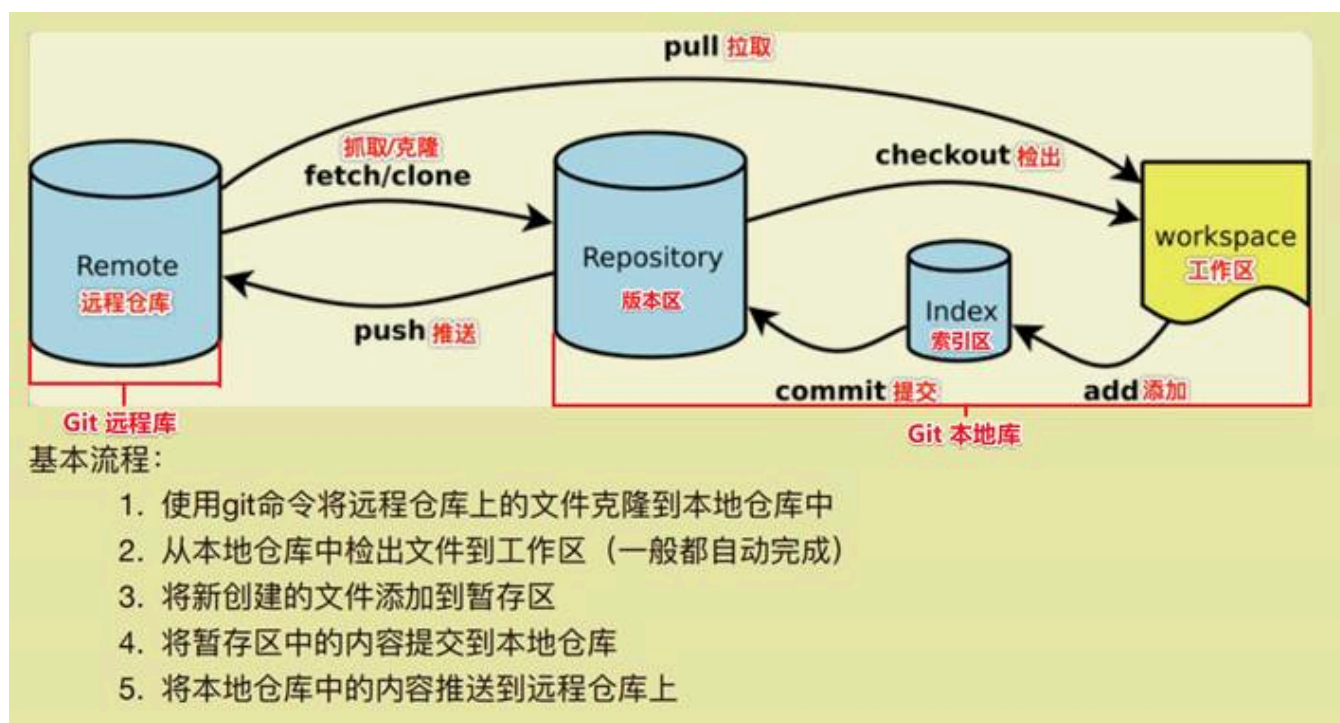
```


(6) 测试: <Git客户端10.0.0.101>从<Git远程库10.0.0.121>采用<HTTP协议>git clone克隆获取

```
cd ~  
rm -rf ~/myProjectDir  
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88**/gits/git01** ~/myProjectDir
```

```
正克隆到 '/root/myProjectDir' ... 基于<HTTP 协议> git clone 克隆获取 OK  
remote: Counting objects: 9, done.  
remote: Compressing objects: 100% (6/6), done.  
remote: Total 9 (delta 2), reused 0 (delta 0)  
Unpacking objects: 100% (9/9), done.
```

2.3. 掌握: <Git远程库>的<基本管理>



2.3.1. Git远程库别名 (git remote)

👉 远程库别名:

- 它是一个<远程库>的<url访问路径>, 从而方便<Git客户端>管理<被跟踪的远程库>。

👉 注意事项:

- 只有在<Git本地库>内, 方可管理<远程库别名>。

👉 默认别名:

- 如果没有设置<指定别名>, 则<Git客户端>会自动采用<origin别名>。

👉 创建别名:

- 语法: `git remote add <别名>`

- 举例:

```
git remote add git_git01 git://10.0.0.121/git01
git remote add gitssh_user01 ssh://user01@10.0.0.121/gits/git01
git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
```

👉 建立: <本地分支>与<远程跟踪分支>之间<默认关联关系>

- 语法: `git branch {-u, --set-upstream-to} <远程跟踪分支名> <本地分支名>`

- 执行时机:

如果<本地库>不是git clone而来, 则当我们git push推送之后, 建议执行!

- 执行目的:

当我们执行 `git pull <别名>` (不带<远程分支名>) 时, 不会出现以下错误提示:

```
You asked to pull from the remote 'githttp_test01', but did not specify
a branch. Because this is not the default configured remote
for your current branch, you must specify a branch on the command line.
[root@server01 myProjectDir]#
[root@server01 myProjectDir]# 提示: 由于没有<默认的关联关系>, 因此需要明确指定<pull 拉取>的<远程分支名>
```

- 举例:

```
cd ~/myProjectDir
git add -A; git commit -m "这是第一次master提交"
git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
git push githttp_test01 --all ## 执行: 第一次git push推送
git branch -a
```

```
* master
remotes/githttp_test01/master
[root@server01 myProjectDir]# ← 这就是: 建立的<远程库 跟踪分支名称>
```

```
git branch -u githttp_test01/master master
```

```
分支 master 设置为跟踪来自 githttp_test01 的远程分支 master。
[root@server01 myProjectDir]#
[root@server01 myProjectDir]# 表明: <远程跟踪分支> 已与 <本地分支> 建立关联关系
```

```
git pull githttp_test01 ## 测试: git pull(不带<选项参数>), OK
```

👉 取消: <本地分支>与<远程跟踪分支>之间<默认关联关系>

- 语法: `git branch --unset-upstream <本地分支名>`

- 举例:

```
git branch --unset-upstream master
git pull githttp_test01
```

```
You asked to pull from the remote 'githttp_test01', but did not specify
a branch. Because this is not the default configured remote
for your current branch, you must specify a branch on the command line.
[root@server01 myProjectDir]#
[root@server01 myProjectDir]# 提示: 由于没有<默认的关联关系>, 因此需要明确指定<pull 拉取>的<远程分支名>
```

👉 查看别名:

- `git remote -v`

- 举例:

```
git remote -v
```

```
git_git01      git://10.0.0.121/git01 (fetch)
git_git01      git://10.0.0.121/git01 (push)
githttp_test01 http://test01:123456@10.0.0.121/gits/git01 (fetch)
githttp_test01 http://test01:123456@10.0.0.121/gits/git01 (push)
gitssh_user01  ssh://user01@10.0.0.121/gits/git01 (fetch)
gitssh_user01  ssh://user01@10.0.0.121/gits/git01 (push)
```

👉 修改别名:

- 语法: `git remote set-url <别名> <老URL>`

- 举例:

```
git remote set-url git_git01 git://10.0.0.121/git02
git remote set-url gitssh_user01 ssh://user01@10.0.0.121/gits/git02
git remote set-url githttp_test01 http://test01:123456@10.0.0.121/gits/git02
```

👉 更名别名:

- 语法: `git remote rename <老别名> <新别名>`

- 举例:

```
git remote rename git_git01 git_xxx
git remote rename gitssh_user01 gitssh_xxx
git remote rename githttp_test01 githttp_xxx
```

👉 删除别名:

- 语法: `git remote remove <别名>`
- 举例:

```
git remote remove git_xxx
git remote remove gitssh_xxx
git remote remove githttp_xxx
```

2.3.2. 推送/删除 (git push)

👉 什么情况下, 需要执行 git push 推送操作?

- 情况1: 我们刚刚创建了一个 Git 远程库, 该 Git 远程库 还是一个 空库。
- 情况2: 我们在工作中, 需要我们的工作成果, 上传到 Git 远程库, 以共享给他人使用。

👉 功能 及 语法:

- (1) 向 Git 远程库 推送: 指定的 分支
 - `git push [-u, --set-upstream] 别名 本地分支名[:远程分支名]`
 - 备注:
 - `-u, --set-upstream`
 - 自动建立: 远程分支 与 本地分支 的 默认关联 关系
 - 使用场景: 当我们尚未建立 本地分支 与 远程跟踪分支 之间 默认关联关系 的时候
- (2) 向 Git 远程库 推送: 所有的 分支
 - `git push 别名 --all`
- (3) 向 Git 远程库 镜像推送: 所有的 分支, 会自动删除 不存在的分支
 - `git push 别名 --mirror`
- (4) 删除 Git 远程库: 指定的 非当前分支
 - `git push 别名 --delete 远程库的非当前分支名`

👉 举例:

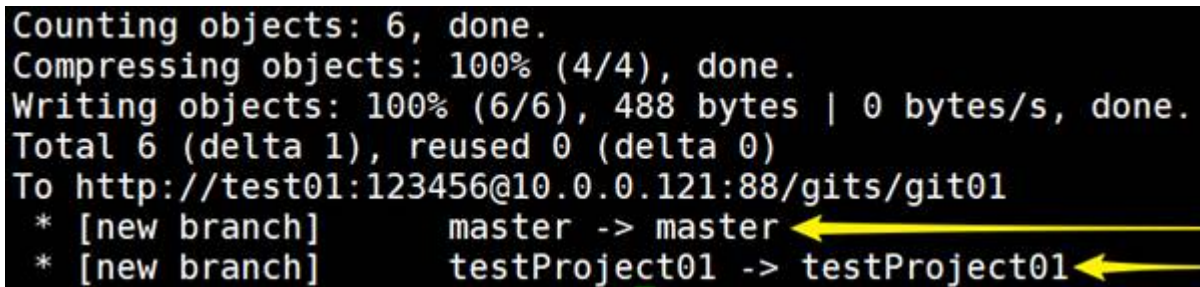
```
cd ~/myProjectDir
git add -A
git commit -m "这是第一次提交至<我的Git本地库>"
```

```
git branch testProject01
git checkout testProject01
echo "123" > 123.txt
git add -A
git commit -m "123.txt"
git checkout master
```

```
git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
git remote -v
```

向 Git 远程库 推送: 所有的 分支, 不会自动删除 不存在的分支

```
git push githttp_test01 --all
```



```
Counting objects: 6, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 488 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
To http://test01:123456@10.0.0.121:88/gits/git01
 * [new branch]      master -> master
 * [new branch]      testProject01 -> testProject01
```

建立: 本地分支 与 远程跟踪分支 之间 默认的关联关系

```
git branch -u githttp_test01/master master
```

向 Git 远程库 推送: 指定的 分支

```
git push githttp_test01 master:master
```

向 Git 远程库 推送: 所有的 分支, 但会自动删除 不存在的分支

```
git push githttp_test01 --mirror
```

删除 Git 远程库: 指定的 非当前分支

```
git push githttp_test01 --delete testProject01
```

2.3.3. 克隆新建 (git clone)

👉 什么情况下, 需要执行 git clone 克隆操作?

- 如果你现在没有<Git本地库>, 则可以使用git clone从无到有的新建一个<Git本地库>。

👉 特点:

- git clone可以克隆获取<Git远程库数据>, 从无到有的新建<Git本地库>或<Git裸库>。
- git clone只能在一个<空目录>中克隆新建<Git本地库>或<Git裸库>。
- 即便<远程库>是一个<空库>, 也可以执行git clone, 从而新建一个<空的本地库>。

👉 功能及语法:

• (1) 首次获取: <远程库 所有分支>

克隆<远程库: 所有分支>, 在<客户端>新建: <Git本地库>, 包含<版本区、索引区、工作区>

```
git clone [-o <自定义别名>] [-b 远程分支名] <远程库URL> <本地库空目录>
```

- 备注:
 - -o <自定义别名> ## 如果不使用 -o选项, 则默认创建<origin别名>
 - -b 远程分支名 ## 如果使用 -b选项, 则克隆之后, 就git checkout切换到<远程库指定分支名>
 - 如果不用 -b选项, 则克隆之后, 就git checkout切换到<远程库当前分支名>
- 注意: <远程库>通常是一个<裸库>, 因此<远程库当前分支名>为master

• (2) 冗余备份: <远程库 所有分支>

克隆<远程库: 所有分支>, 在<客户端>新建: <Git**裸库**>, 仅含<版本区>

```
git clone [--bare | --mirror] [-b 远程分支名] <远程库URL> <本地库空目录>
```

- 备注:
 - --bare ## 表示: 克隆<完整裸库>, 来新建本地的<Git裸库>
 - --mirror ## 表示: 镜像<完整裸库>, 来新建本地的<Git裸库>, 比 --bare 更加完整
 - -b 远程分支名 ## 如果使用 -b选项, 则克隆之后, 就git checkout切换到<远程库指定分支名>
 - 如果不用 -b选项, 则克隆之后, 就git checkout切换到<远程库当前分支名>
- 注意: <远程库>通常是一个<裸库>, 因此<远程库当前分支名>为master

👉 举例:

```
## -----
```

```
## 示例1: 首次获取<远程库>
```

```
## 克隆<远程库: 所有分支>, 新建<Git本地库>, 自动git checkout切换到<远程库当前分支名>
```

```
##
```

```
cd ~
rm -rf ~/myProjectDir
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 ~/myProjectDir
```

```
正克隆到 '/root/myProjectDir'...
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. OK
```

```
cd ~/myProjectDir/
git branch -a
```



```

* master ← 本地分支
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master
remotes/githttp_test01/testProject01 ← 跟踪分支

```

示例2: 首次获取<远程库>

克隆<远程库: 所有分支>, 新建<Git本地库>, 自动git checkout切换到<远程库指定分支名>

##

```

cd ~
rm -rf ~/myProjectDir
git clone -b testProject01 -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
~/myProjectDir
cd ~/myProjectDir/
git branch -av

```

```

* testProject01 ←
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master
remotes/githttp_test01/testProject01 ←

```

示例3: 冗余备份<远程库>

克隆<完整裸库>, 新建<本地Git裸库>, 自动git checkout切换到<远程库当前分支名>

##

```

cd ~
rm -rf ~/myProjectDir
git clone --bare http://test01:123456@10.0.0.121:88/gits/git01 ~/myProjectDir
cd ~/myProjectDir/
git branch -a

```

```

* master
testProject01 | ← 本地<裸库>中的<分支>

```

示例4: 冗余备份<远程库>

克隆<完整裸库>, 新建<本地Git裸库>, 自动git checkout切换到<远程库指定分支名>

##

```
cd ~
rm -rf ~/myProjectDir
git clone --bare -b testProject01 http://test01:123456@10.0.0.121:88/gits/git01
~/myProjectDir
cd ~/myProjectDir/
git branch -a
```

```
master
* testProject01 | ← 本地<裸库>中的分支
```

示例5: 冗余备份<远程库>

镜像<完整裸库>, 新建<本地Git裸库>, 自动git checkout切换到<远程库当前分支名>

##

```
cd ~
rm -rf ~/myProjectDir
git clone --mirror http://test01:123456@10.0.0.121:88/gits/git01 ~/myProjectDir
cd ~/myProjectDir/
git branch -a
```

```
* master
testProject01 | ← 本地<裸库>中的<分支>
```

示例6: 冗余备份<远程库>

镜像<完整裸库>, 新建<本地Git裸库>, 自动git checkout切换到<远程库指定分支名>

##

```
cd ~
rm -rf ~/myProjectDir
git clone --mirror -b testProject01 http://test01:123456@10.0.0.121:88/gits/git01
~/myProjectDir
cd ~/myProjectDir/
git branch -a
```

```
master
* testProject01 | ← 本地<裸库>中的分支
```

2.3.4. 抓取分支, 不合并 (git fetch)

👉 什么情况下, 需要执行 git fetch 抓取操作?

- 如你已拥有一个<Git本地库>, 则你可通过git fetch来获取<Git远程分支>的<更新代码>, 但不立即合并使用。

👉 特点:

1. git fetch不能新建<Git本地库>或<Git裸库>, 只能获取<Git远程库分支>。
2. git fetch不会将<远程分支>自动合并到<本地分支>, 只能后期手动执行<分支合并>。

👉 工作机制:

- (1) git fetch抓取<远程库分支>完整的<版本数据>, 保存到<本地库>的<版本区>。
- (2) git fetch会在<.git/FETCH_HEAD指针文件>中, 记录<远程库分支>的<当前快照哈希值>。
- <本地库>使用<.git/HEAD 指针文件>, 在本地保存: <本地分支>的<当前快照哈希值>。
 - <本地库>使用<.git/FETCH_HEAD指针文件>, 在本地保存: <跟踪分支>的<当前快照哈希值>。
- (3) git fetch不会将<远程分支>自动合并到<本地分支>, 只能后期手动执行<分支合并>。

👉 功能及 语法:

- (1) 出于<更新代码>的<目的>, 抓取: <所有跟踪分支>的<远程库: 所有分支>

```
git fetch --all [-p]
```

备注:

--all ## 依据<所有的remote远程库别名>, 执行<抓取>操作

-p ## 根据<远程库分支>的<存在与否>, 来增/删<跟踪分支名 及 版本数据>, 否则只增不删

- (2) 出于<更新代码>的<目的>, 获取: <指定跟踪分支>的<远程库: 所有分支>

```
git fetch [<别名>|<远程库URL>] [-p]
```

备注:

-p ## 根据<远程库分支>的<存在与否>, 来增/删<跟踪分支名 及 版本数据>, 否则只增不删

- (3) 出于<更新代码>的<目的>, 获取: <指定跟踪分支>的<远程库: 指定分支>

```
git fetch [<别名>|<远程库URL>] <远程库指定分支名>[:<本地库指定分支名>]
```

👉 举例:

👉 环境准备: 克隆<远程库当前分支>, 来新建<Git本地库>

##

```
cd ~
```

```
rm -rf ~/myProjectDir
```

```
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 ~/myProjectDir
```

```
cd ~/myProjectDir
```

```
git branch -a
```

```
* master
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master
remotes/githttp_test01/testProject01
```

← 这是克隆过程中, 所建立的<跟踪分支>

示例1：出于<更新代码>的<目的>，抓取：<所有跟踪分支>的<远程库：所有分支>

##

cd ~/myProjectDir

git branch -a

```
* master ← 本地分支
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master ← 跟踪分支
remotes/githttp_test01/testProject01
```

git fetch --all

cat .git/FETCH_HEAD

```
35b5733bd331cf127e292606c93e4988e96a4d0e |  | branch 'master' of http://10.0.0.121:88/gits/git01
ef914c7b06ba864017c0e6624db7c468376b41d9 |  | not-for-merge branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
显示表明：保存着<远程库：所有分支>的<当前快照：哈希值>
```

##

git checkout master

git merge FETCH_HEAD ## <---- 如果需要，可以手动合并，如有冲突，就通过git mergetool进行解决

##

git checkout testProject01

git merge FETCH_HEAD ## <---- 如果需要，手动合并，如有冲突，就通过git mergetool进行解决

git checkout master

示例2：出于<更新代码>的<目的>，获取：<指定跟踪分支>的<远程库：所有分支>

##

cd ~/myProjectDir

git fetch githttp_test01

cat .git/FETCH_HEAD

```
35b5733bd331cf127e292606c93e4988e96a4d0e |  | branch 'master' of http://10.0.0.121:88/gits/git01
ef914c7b06ba864017c0e6624db7c468376b41d9 |  | not-for-merge branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
显示表明：保存着<远程库：所有分支>的<当前快照：哈希值>
```

##

git checkout master

git merge FETCH_HEAD ## <---- 如果需要，可以手动合并，如有冲突，就通过git mergetool进行解决

##

git checkout testProject01

git merge FETCH_HEAD ## <---- 如果需要，手动合并，如有冲突，就通过git mergetool进行解决

```
git checkout master
```

```
## -----
```

示例3：出于<更新代码>的<目的>，获取：<指定跟踪分支>的<远程库：指定分支>

```
cd ~/myProjectDir
```

```
git checkout master
```

```
git fetch githttp_test01 master
```

```
cat .git/FETCH_HEAD
```

```
f2337e84e3adbc723a1c3e2a94d06ecfbe6450a0      branch 'master' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[root@server02 myProjectDir]#
```

这就是：<远程库 master 分支>的<当前快照哈希值>

```
git merge FETCH_HEAD      ## <---- 手动合并，如有冲突，就通过git mergetool进行解决
```

```
##
```

```
git fetch githttp_test01 testProject01:testProject01
```

```
cat .git/FETCH_HEAD
```

```
af66833b61ba69163242b8e8b682ad35808bddff      branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[root@server02 myProjectDir]#
```

这就是：<远程库 testProject01 分支>的<当前快照哈希值>

```
git checkout testProject01
```

```
git merge FETCH_HEAD      ## <---- 手动合并，如有冲突，就通过git mergetool进行解决
```

```
git checkout master
```

2.3.5. 拉取分支，并合并 (git pull)

👉 什么情况下，需要执行拉取操作？

- 如你已拥有一个<Git本地库>，则你可通过git pull来获取<Git远程分支>的<更新代码>，并立即合并使用。

👉 特点：

1. git pull不能新建<Git本地库>或<Git裸库>，只能获取<Git远程库分支>。
2. git pull会将<远程分支>自动合并到<本地分支>，类似于 git fetch + git merge。

👉 工作机制：

- (1) git pull抓取<远程库分支>完整的<版本数据>，保存到<本地库>的<版本区>。
- (2) git pull会在<.git/FETCH_HEAD指针文件>中，记录<远程库分支>的<当前快照哈希值>。

👉 <本地库>使用<.git/HEAD 指针文件>本地保存：<本地分支>的<当前快照哈希值>。

👉 <本地库>使用<.git/FETCH_HEAD指针文件>本地保存：<跟踪分支>的<当前快照哈希值>。

- (3) git pull会将<远程分支>自动合并到<本地分支>，类似于 git fetch + git merge。

👉 功能及语法:

- (1) 出于<更新代码>的<目的>, 抓取: <所有跟踪分支>的<远程库: 所有分支>

```
git pull --all [-p]
```

备注:

--all ## 依据<所有的remote远程库别名>, 执行<抓取>操作

-p ## 根据<远程库分支>的<存在与否>, 来增/删<跟踪分支名 及 版本数据>, 否则只增不删

- (2) 出于<更新代码>的<目的>, 获取: <指定跟踪分支>的<远程库: 所有分支>

```
git pull [<别名>|<远程库URL>] [-p]
```

备注:

-p ## 根据<远程库分支>的<存在与否>, 来增/删<跟踪分支名 及 版本数据>, 否则只增不删

- (3) 出于<更新代码>的<目的>, 获取: <指定跟踪分支>的<远程库: 指定分支>

```
git pull [<别名>|<远程库URL>] <远程库指定分支名>[:<本地库指定分支名>]
```

👉 举例:

👉 环境准备: 克隆<远程库当前分支>, 来新建<Git本地库>

##

```
cd ~
```

```
rm -rf ~/myProjectDir
```

```
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 ~/myProjectDir
```

```
cd ~/myProjectDir
```

```
git branch -a
```

```
* master
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master
remotes/githttp_test01/testProject01
```

← 这是克隆过程中, 所建立的<跟踪分支>

示例1: 出于<更新代码>的<目的>, 抓取: <所有跟踪分支>的<远程库: 所有分支>

##

```
cd ~/myProjectDir
```

```
git branch -a
```

```
* master ← 本地分支
remotes/githttp_test01/HEAD -> githttp_test01/master
remotes/githttp_test01/master ← 跟踪分支
remotes/githttp_test01/testProject01
```



```
git pull --all
```

```
cat .git/FETCH_HEAD
```

```
35b5733bd331cf127e292606c93e4988e96a4d0e | not-for-merge branch 'master' of http://10.0.0.121:88/gits/git01
ef914c7b06ba864017c0e6624db7c468376b41d9 | branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[显示表明：保存着<远程库：所有分支>的<当前快照：哈希值>]
```

```
## -----
```

示例2：出于<更新代码>的<目的>，获取：<指定跟踪分支>的<远程库：所有分支>

```
##
```

```
cd ~/myProjectDir
```

```
git pull githttp_test01
```

```
cat .git/FETCH_HEAD
```

```
35b5733bd331cf127e292606c93e4988e96a4d0e | not-for-merge branch 'master' of http://10.0.0.121:88/gits/git01
ef914c7b06ba864017c0e6624db7c468376b41d9 | branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[显示表明：保存着<远程库：所有分支>的<当前快照：哈希值>]
```

```
## -----
```

示例3：出于<更新代码>的<目的>，获取：<指定跟踪分支>的<远程库：指定分支>

```
cd ~/myProjectDir
```

```
git checkout master
```

```
git pull githttp_test01 master
```

```
cat .git/FETCH_HEAD
```

```
f2337e84e3adbc723alc3e2a94d06ecf6e6450a0 | branch 'master' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[这就是：<远程库 master 分支>的<当前快照哈希值>]
```

```
##
```

```
git pull githttp_test01 testProject01:testProject01
```

```
cat .git/FETCH_HEAD
```

```
af66833b61ba69163242b8e8b682ad35808bddff | branch 'testProject01' of http://10.0.0.121:88/gits/git01
[root@server02 myProjectDir]#
[这就是：<远程库 testProject01 分支>的<当前快照哈希值>]
```

2.3.6. 恢复：分离的分支

👉 语法：git branch <自定义分支名> <分离分支的HEAD快照哈希值>

👉 应用场景：

甲乙都通过git clone克隆新建了相同的<Git本地库>，随着项目工作的开展，甲新建了<Git分支1>，并git push --all上传，乙新建的<Git分支2>，并git push --all上传。

现在甲乙均通过git pull获取最新的<Git版本库数据>，甲发现了乙创建的<Git分支2>，但是处于<分离状态>，乙也发现了甲创建的<Git分支1>，但是同样处于<分离状态>。

通常情况下，<Git分支>属于<测试代码>，一旦稳定后，就应该合并的<master主干>，甲乙双方可以不必理睬这些<分离的分支>。

如果甲乙双方均需参与针对<所有Git分支>的<团队协助>开发，那么就需要恢复<分离的分支>！

👉 举例：

甲：

```
cd ~
rm -rf /git01
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 /git01
```

乙：

```
cd ~
rm -rf /git01
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 /git01
```

甲：

```
cd /git01
echo "甲 111" > app01
git add -A
git commit -m "甲：这是master主干的第一次提交"
git branch test01
git checkout test01
echo "甲 111 测试" > test01_app01
git add -A
git commit -m "甲：这是test01分支的第一次提交"
git checkout master
git push githttp_test01 --all
```

乙：

```
cd /git01
echo "乙 222" > app02
git add -A
git commit -m "乙：这是master主干的第一次提交"
git branch test02
git checkout test02
echo "乙 222 测试" > test02_app01
git add -A
git commit -m "乙：这是test02分支的第一次提交"
git checkout master
git pull githttp_test01
```

开始恢复：甲创建的<test01分支>

```
cat .git/FETCH_HEAD
```

```
c889855576970305a778cb51b3ddddd3f7a1219a      branch 'master' of http://10.0.0.121:88/gits/git01
27b081af0997bbc5f5290783e343d2c8a9b90d1b ← not-for-merge branch 'test01' of http://10.0.0.121:88/gits/git01
04ad3874acbe60f43cedaa3e1470e88ecbec7756      not-for-merge branch 'test02' of http://10.0.0.121:88/gits/git01
```

```
git checkout -b test01 27b081af0997bbc5f5290783e343d2c8a9b90d1b
git checkout master
git branch -a
```

```
* master
test01 ←
test02
remotes/githttp_test01/master
remotes/githttp_test01/test01
remotes/githttp_test01/test02
```

```
git push githttp_test01 --all
```

```
## -----
```

```
## 甲:
```

```
##
```

```
git pull githttp_test01
## 开始恢复: 乙创建的<test02分支>
cat .git/FETCH_HEAD
```

```
fca68a8e89aebd5d31394b6628bbcb456dc93911      branch 'master' of http://10.0.0.121:88/gits/git01
27b081af0997bbc5f5290783e343d2c8a9b90d1b      not-for-merge branch 'test01' of http://10.0.0.121:88/gits/git01
04ad3874acbe60f43cedaa3e1470e88ecbec7756 ← not-for-merge branch 'test02' of http://10.0.0.121:88/gits/git01
```

```
git checkout -b test02 04ad3874acbe60f43cedaa3e1470e88ecbec7756
git checkout master
git branch -a
```

```
* master
test01
test02 ←
remotes/githttp_test01/master
remotes/githttp_test01/test01
remotes/githttp_test01/test02
```

2.3.7. 列出: <远程库>的<分支>

👉 语法: `git ls-remote [<别名>|<远程库URL>]`

👉 举例:

```
git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
git ls-remote githttp_test01
```

```
14a746700607d8e30f15169787e252cdfc6324d3      HEAD
14a746700607d8e30f15169787e252cdfc6324d3      refs/heads/master
```

★ 本章习题

1. 请问：什么是<裸库>？为什么<远程库>通常是<裸库>？<远程库>可以不是<裸库>吗？
2. 请列出：与<远程库>进行数据交互的常用<协议>？并分别说明各个<协议实现>的特点。
3. 请问：<远程库的别名>有什么作用？默认别名是什么？可以自定义别名吗？
4. 现在你需要将你的代码分享给其他人使用，你如何创建<远程库>？采用什么<协议实现>方式最为方便？
5. 现在你刚刚新建了一个<远程库>，目前还没有任何数据，你应该如何操作，才能对其他人分享代码？
6. 现在你已经部署好一台<远程库服务器>，为了实现数据冗余，你应该如何操作，来实现数据冗余？
7. 请阐述：git clone、git fetch、git pull这三个基本命令的作用和彼此之间的区别？
8. 公司现在需要开展一个项目，项目开发成员有甲、乙、丙三人，为了实现彼此之间的协助开发，因此部署了一台gitserver远程库服务器，大家可以通过<http://test01:123456@10.0.0.121:88/gits/git01>来共享gitserver远程库服务器上的项目代码库。

当前工作流程如下：

(1) 甲通过以下命令，已在</git01目录>中，建立了<git01本地库>，并推送到<远程库>：

```
mkdir /git01
cd /git01
git init
echo "程序代码01" > app01
git add -A
git commit -m "甲：本项目master主干的第一次提交"
git remote add githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01
git push githttp_test01 --all
git pull githttp_test01
```

```
You asked to pull from the remote 'githttp_test01', but did not specify
a branch. Because this is not the default configured remote
for your current branch, you must specify a branch on the command line.
[root@server01 git01]#
[root@server01 git01]#
[root@server01 git01]#
[root@server01 git01]#
```

出现这样的提示，说明什么问题？该如何解决？

请回答：

(2) 乙和丙通过以下命令，获得完整的<git01本地库>，进而开展<项目开发>工作。

```
rm -rf /git01
git clone -o githttp_test01 http://test01:123456@10.0.0.121:88/gits/git01 /git01
```

(3) 乙在后续工作中，执行了如下命令：

```
cd /git01
echo "程序代码01的更新代码01" >> app01
echo "程序代码02" > app02
git add -A
git commit -m "乙：本项目master主干的第一次提交"
git pull githttp_test01
git push githttp_test01 --all
```

(4) 丙在后续工作中，执行了如下命令：

```
cd /git01
git branch test01
git checkout test01
echo "测试程序代码01" > testApp01
echo "测试程序代码02" > testApp02
echo "测试程序代码03" > testApp03
git add -A
git commit -m "丙：本项目test01分支的第一次提交"
git checkout master
git pull githttp_test01
git push githttp_test01 --all
```

(5) 甲在后续工作中，执行了如下命令：

```
cd /git01
echo "程序代码01的更新代码02" >> app01
echo "程序代码02的更新代码01" >> app02
echo "程序代码03" > app03
git add -A
git commit -m "甲：本项目master主干的第二次提交"
git pull githttp_test01
```

```
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
来自 http://10.0.0.121:88/gits/git01
 904d703..066c782 master -> githttp_test01/master
* [新分支] test01 -> githttp_test01/test01
自动合并 app02
冲突（添加/添加）：合并冲突于 app02
自动合并 app01
冲突（内容）：合并冲突于 app01
自动合并失败，修正冲突然后提交修正的结果。
```

这个问题，该如何解决？

请回答：

```
git push githttp_test01 --all
git branch -a
```

```
* master
remotes/githttp_test01/master 请问：甲 git pull 拉取了 test01 分支，为什么没有<test01 本地分支>？
remotes/githttp_test01/test01 请问：甲需要参与<test01 分支>的协助开放，该如何恢复<test01 本地分支>？
[root@server01 git01]#
[root@server01 git01]#
```

请回答：