**6\_Operation06版本控制git基础及进阶rpm打包**

**一 版本控制**

**1.1 版本库:典型的客户/服务器系统**

版本库是版本控制的核心,支持任意数量的客户端,客户端通过写数据库分享代码

**1.2 分布式版本控制**

1.2.1 集中式版本控制系统,代表SVN(Subversion)

开发者之间公用一个仓库(repository),所有操作需要联网

1.2.2 分布式版本控制系统,代表git

每个开发者都是一个仓库的完整克隆,每个人都是服务器;支持断网操作

**二 GIT基本**

**2.1 GIT基本概念**

2.1.1 GIT仓库:保存所有数据的地方

2.1.2 工作区:从仓库中提取出文件,放在磁盘上供使用或修改的区域

2.1.3 暂存区:就是一个文件,索引文件,保存了下次将要提交的文件列表信息

**2.2案例：Git基本操作**

**2.2.1 问题**

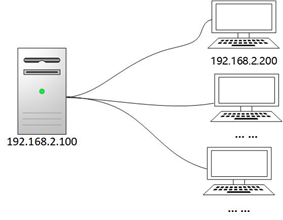
本案例要求先快速搭建好一台Git服务器，并测试该版本控制软件，要求如下：

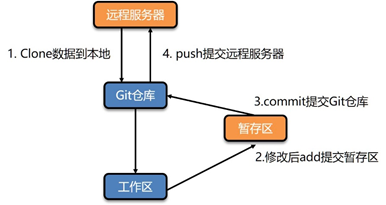
安装Git软件 创建版本库 客户端克隆版本仓库到本地 本地工作目录修改数据

提交本地修改到服务器

**2.2.2方案**

实验拓扑如图所示，Git工作流如图所示。





**2.2.3步骤**

**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。**

web1 ~]# yum -y install git

web1 ~]# git --version

**2)初始化一个空仓库。**

web1 ~]# mkdir /var/git #建立1个空目录

web1 ~]# **git init --bare** /var/git/project

git init --bare /路径/仓库名

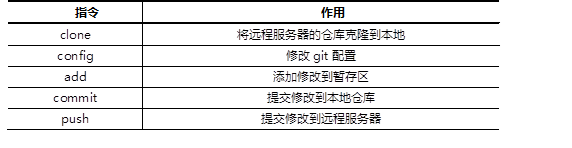
#初始化空的 Git 版本库于 /var/git/project/

web1 ~]# ls /var/git/project

branches 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)

**git常用指令列表**



**1) clone克隆服务器仓库到本地。**

web2 ~]# yum -y install git

web2 ~]# **git clone** root@192.168.2.100:/var/git/project

web2 ~]# cd project #本地project目录为工作区

web2 ~]# ls -a

. .. .git #.git目录为本地仓库

**2) 修改git配置。**

web2 project]# **git config --global user.email "you@example.com"**

web2 project]# **git config --global user.name "Your Name"**

web2 project]# cat ~/**.gitconfig**

[user]

    email = you@example.com

    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**)。

web2 project]# echo "init date" > init.txt

web2 project]# mkdir demo

web2 project]# cp /etc/hosts demo

**4） 查看仓库中数据的状态。**

web2 project]# **git status**

**5） 将工作区的修改提交到暂存区**。

web2 project]# **git add .**

**6) 将暂存区修改提交到本地仓库。**

web2 project]# **git commit -m "注释，可以为任意字符"**

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)。

web2 project]# **git config --global push.default simple**

web2 project]# **git push**

root@192.168.2.100's password: 输入服务器root密码

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。

web2 project]# **git pull**

**9) 查看版本日志**。

web2 project]# git log #显示详细版本日志

web2 project]# git log --pretty=oneline #1行显示版本日志

web2 project]# git log --oneline #精简显示版本日志

web2 project]# git reflog #带指针显示版本日志

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。

**三 git进阶**

**3.1 HEAD指针**

3.1.1 **HEAD指针是一个可以在任何分支和版本移动的指针,通过移动指针可以将数据还原至任何版本.指向当前分支的最新版本.**

**3.2案例：HEAD指针操作**

**3.2.1 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

**3.2.2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。每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。

**3.2.3步骤**

**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）。

web2 project]# echo "new file" > new.txt

web2 project]# git add .

web2 project]# git commit -m "add new.txt"

web2 project]# echo "first" >> new.txt

web2 project]# git add .

web2 project]# git commit -m "new.txt:first line"

web2 project]# echo "second" >> new.txt

web2 project]# git add .

web2 project]# git commit -m "new.txt:second"

web2 project]# echo "third" >> new.txt

web2 project]# git add .

web2 project]# git commit -m "new.txt:third"

web2 project]# **git push**

web2 project]# echo "123" > num.txt

web2 project]# git add .

web2 project]# git commit -m "num.txt:123"

web2 project]# echo "456" > num.txt

web2 project]# git add .

web2 project]# git commit -m "num.txt:456"

web2 project]# echo "789" > num.txt

web2 project]# git add .

web2 project]# git commit -m "num.txt:789"

web2 project]# git push

**2) 查看Git版本信息。**

web2 project]# git reflog

web2 project]# git log --oneline

04ddc0f num.txt:789

7bba57b num.txt:456

301c090 num.txt:123

b427164 new.txt:third

0584949 new.txt:second

ece2dfd new.txt:first line

e1112ac add new.txt

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}。**

web2 project]# **git reset --hard** 301c0

web2 project]# git reflog

301c090 HEAD@{0}: reset: moving to 301c0

04ddc0f HEAD@{1}: commit: num.txt:789

7bba57b HEAD@{2}: commit: num.txt:456

301c090 HEAD@{3}: commit: num.txt:123

b427164 HEAD@{5}: commit: new.txt:third

0584949 HEAD@{6}: commit: new.txt:second

ece2dfd HEAD@{7}: commit: new.txt:first line

e1112ac HEAD@{8}: commit: add new.txt

1a0d908 HEAD@{9}: commit (initial): 初始化

web2 project]# cat num.txt #查看文件是否为123

123

web2 project]# git reset --hard 7bba57b

web2 project]# cat num.txt #查看文件是否为456

456

web2 project]# git reflog #查看指针移动历史

7bba57b HEAD@{0}: reset: moving to 7bba57b

301c090 HEAD@{1}: reset: moving to 301c0

… …

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。**

web2 project]# git rm init.txt #删除文件

rm 'init.txt'

web2 project]# git commit -m "delete init.txt"

#提交本地仓库

web2 project]# git reflog #查看版本历史

0dc2b76 HEAD@{0}: commit: delete init.txt

7bba57b HEAD@{0}: reset: moving to 7bba57b

301c090 HEAD@{1}: reset: moving to 301c0

… …

web2 project]# git reset --hard 04ddc0f #恢复数据

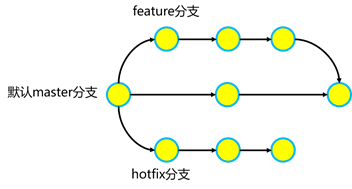
web2 project]# ls

demo init.txt new.txt num.txt

**四 分支**

**4.1 分支概念**

4.1.1分支可以让开发多条主线同时进行,每条主线互不影响.



4.1.2分支可以按功能模块分支,或按版本分支;分支可可以合并.

4.1.3常见的分支规范

MASTER分支 主分支,是代码的核心

DEVELOP分支 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

**4.2管理多分支**

4.2.1 查看当前分支

**git status 查看git状态; git branch 查看当前所在分支**

**git branch -v 查看所有分支, \*号代表所在的分支**

4.2.2 创建分支

**git branch 分支名**

4.2.3 分支切换

**git checkout 分支名**

4.2.4 分支合并

**切换到master分支: git checkout master**

**执行merge命令合并分支: git merge 要被合并的分支名**

**4.2.5 解决分支合并冲突**

查看有冲突的文件,手动修改为最终需要的文件内容

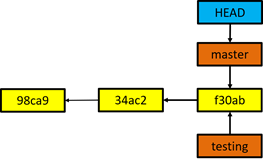
修改完成后,正常add,commit,解决冲突

**4.3 分支与HEAD指针的关系**

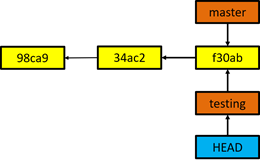
创建分支的本质是在当前提交上创建一个可以移动的指针

如何判断当前分支呢？答案是根据HEAD这个特殊指针

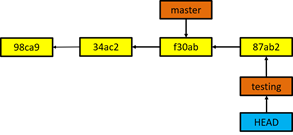
分支操作流程如图-6，图-7，图-8，图-9，图-10所示。



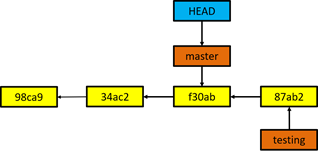
HEAD指针指向master分支



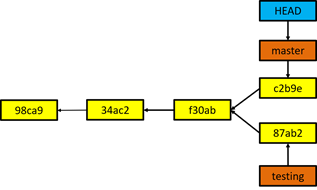
切换分支，HEAD指针指向testing分支



在testing分支中修改并提交代码



将分支切换回master分支



在master分支中修改数据，更新版本

**4.4案例：Git分支操作**

**4.4.1 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

**4.4.2 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。

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。

web2 project]# **git status** #查看git状态

# On branch master

nothing to commit, working directory clean

web2 project]# **git branch -v**  #查看git分支状态

\* master 0dc2b76 delete init.txt

2）创建分支。

web2 project]# **git branch** hotfix #创建分支

web2 project]# git branch feature #创建分支

web2 project]# **git branch -v** #查看git分支状态

feature 0dc2b76 delete init.txt

hotfix 0dc2b76 delete init.txt

\* master 0dc2b76 delete init.txt

**步骤二：切换与合并分支**

1）切换分支。

web2 project]# **git checkout** hotfix #切换分支

web2 project]# git branch -v #查看git分支状态

feature 0dc2b76 delete init.txt

\* hotfix 0dc2b76 delete init.txt # \*号代表所在分支

master 0dc2b76 delete init.txt

**2）在新的分支上可以继续进行数据操作（增、删、改、查）。**

web2 project]# echo "fix a bug" >> new.txt

web2 project]# git add .

web2 project]# git commit -m "fix a bug"

**3）将hotfix修改的数据合并到master分支。**

#合并前必须要先切换到master分支，然后再执行merge命令。

web2 project]# git checkout master #切换到master分支

web2 project]# cat new.txt

#默认master分支中没有hotfix分支中的数据

web2 project]# git merge hotfix #合并hotfix分支到master

Updating 0dc2b76..5b4a755

Fast-forward

new.txt | 1 ++

1 file changed, 1 insertions(+)

**4）将所有本地修改提交远程服务器。**

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。

web2 project]# git checkout hotfix

web2 project]# echo "AAA" > a.txt

web2 project]# git add .

web2 project]# git commit -m "add a.txt by hotfix"

web2 project]# git checkout master

web2 project]# echo "BBB" > a.txt

web2 project]# git add .

web2 project]# git commit -m "add a.txt by master"

web2 project]# git merge hotfix

自动合并 a.txt

冲突（添加/添加）：合并冲突于 a.txt

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。

web2 project]# cat a.txt #该文件中包含有冲突的内容

<<<<<<< HEAD

BBB

=======

AAA

>>>>>>> hotfix

web2 project]# vim a.txt

#修改该文件，为最终需要的数据，解决冲突

BBB

web2 project]# git add .

web2 project]# git commit -m "resolved"

**五 git服务器**

**案例：Git服务器**

**5.1 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 (可读写)

创建Git协议服务器 (只读:只能clone到本地,不可push)

创建HTTP协议服务器 (只读:只能clone到本地,不可push)

**5.2 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）。

web1 ~]# git init --bare /var/git/base\_ssh

Initialized empty Git repository in /var/git/base\_ssh/

2)客户端访问的方式（web2主机操作）。

web2 ~]# git clone root@192.168.2.100:/var/git/base\_ssh

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）。

web2 ~]# **ssh-keygen** -f /root/.ssh/id\_rsa -N ''

# -f 指定密码存储路径和文件名,-N ‘’ 指定密码为空

web2 ~]# **ssh-copy-id** 192.168.2.100 #将密码传递给git的ssh服务器

web2 ~]# git clone **root@192.168.2.100:/var/git/base\_ssh**

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）。

web1 ~]# yum -y install **git-daemon**

2）创建版本库（web1主机操作）。

web1 ~]# git init --bare /var/git/base\_git

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）。

web1 ~]# vim **/usr/lib/systemd/system/git@.service**

修改前内容如下：

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

--export-all --user-path=public\_git --syslog --inetd –verbose

修改后内容如下：

ExecStart=-/usr/libexec/git-core/git-daemon **--base-path=/var/git**

--export-all --user-path=public\_git --syslog --inetd –verbose

web1 ~]# **systemctl start git.socket** #起服务

4）客户端访问方式（web2主机操作）

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）。

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）。

web1 ~]# vim +11 **/etc/gitweb.conf** #直接跳到第11行

**$projectroot = "/var/git";**  #添加一行,注意空格与分号

3) 创建版本仓库（web1主机操作）

web1 ~]# git init --bare /var/git/base\_http

4）启动httpd服务器

web1 ~]# **systemctl start httpd** #启动http服务

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。

web2 ~]# **firefox http://192.168.2.100/git/**

**六 RPM打包**

**6.1 应用场景**

官方未提供RPM包;官方RPM无法自定义;大量源码包,希望提供统一的软件管理机制

**6.2 打包流程**

**6.2.1 安装rpm-build**

web1 ~]# yum -y install rpm-build #安装软件

web1 ~]# rpmbuild -ba xxx.spec #在家目录下生成rpmbuild目录及子目录

web1 ~]# cd rpmbuild/ #进入rpmbuild目录

web1 rpmbuild]# ls #查看rpmbuild子目录

BUILD BUILDROOT RPMS SOURCES SPECS SRPMS

**6.2.2 准备源码软件**(tar包复制到SOURCE目录下)

web1 lnmp\_soft]# cp nginx-1.12.2.tar.gz /root/rpmbuild/SOURCES/

web1 lnmp\_soft]# cd /root/rpmbuild/SOURCES/

**6.2.3 编写编译配置文件**(新建 文件名.spec文件并用vim打开)

web1 SOURCES]# vim nginx.spec

**描述信息:**

**Name: nginx #软件名称,必须和tar包一致**

**Version: 1.12.2 #软件版本,必须和tar包一致**

Release: 1 #自行编辑的版本,随便写

Summary:这是个网站服务器软件 #描述摘要,随便写

#Group: #软件组,注释掉

License: GPL #协议,随便写

URL: [www.douniwan.com](http://www.douniwan.com) #网址,随便写

**Source0: nginx-1.12.2.tar.gz #源码文件,必须和tar包一致**

#BuildRequires: #编译时依赖关系,注释掉

#Requires: #安装时依赖关系,注释掉

%description 这还是一个网站服务器软件 #详细描述,随便写

**安装信息:**

%prep #安装RPM包前需要执行的脚本

%setup -q #相当于tar解压,并进入解压后的目录

**%post #安装RPM包后需要执行的脚本,手动添加**

**useradd -s /sbin/nologin nginx #%post的脚本,手动添加**

**%build #编译需要执行的命令**

**./configure #修改此行为./configure,并添加相应的nginx参数**

make %{?\_smp\_mflags}

%install

make install DESTDIR=%{buildroot}

%files

**%doc**

**/usr/local/nginx/\* #需要打包的目录与文件**

6.2.4 编译RPM包,安装编译出的RPM包并测试

**6.3 案例：制作nginx的RPM包**

**步骤一：安装rpm-build软件**

**1）安装rpm-build软件包**

web1 ~]# yum -y install rpm-build

**2）生成rpmbuild目录结构**

web1 ~]# rpmbuild -ba nginx.spec #会报错，没有文件或目录

web1 ~]# ls /root/rpmbuild #自动生成的目录结构

BUILD BUILDROOT RPMS SOURCES SPECS SRPMS

**3）准备工作，将源码软件复制到SOURCES目录**

web1 ~]# cp nginx-1.12.2.tar.gz /root/rpmbuild/**SOURCES**/

**4）创建并修改SPEC配置文件**

web1 ~]# vim /root/rpmbuild/**SPECS/nginx.spec**

**Name:nginx**  #源码包软件名称

**Version:1.12.2** #源码包软件的版本号

Release:    10 #制作的RPM包版本号

Summary: Nginx is a web server software. #RPM软件的概述

License:GPL #软件的协议

URL:[www.test.com](http://www.test.com) #网址

**Source0:nginx-1.12.2.tar.gz** #源码包文件的全称

**#**BuildRequires: #制作RPM时的依赖关系

**#**Requires: #安装RPM时的依赖关系

%description

nginx [engine x] is an HTTP and reverse proxy server.

#软件的详细描述

**%post**

**useradd nginx**  #非必需操作：安装后脚本(创建账户),手动编写

%prep

%setup -q #自动解压源码包，并cd进入目录

%build

**./configure** #修改此行为./configure

make %{?\_smp\_mflags}

%install

make install DESTDIR=%{buildroot}

%files

%doc

**/usr/local/nginx/\*** #对哪些文件与目录打包

%changelog

**步骤二：使用配置文件创建RPM包**

**1）安装依赖软件包**

web1 ~]# yum -y install gcc pcre-devel openssl-devel

**2）rpmbuild创建RPM软件包**

web1 ~]# rpmbuild -ba /root/rpmbuild/SPECS/nginx.spec

web1 ~]# ls /root/rpmbuild/RPMS/x86\_64/nginx-1.12.2-10.x86\_64

.rpm

**步骤三：安装、卸载软件**

web1 ~]# rpm -ivh /root/rpmbuild/RPMS/x86\_64/nginx-1.12.2-10.

x86\_64.rpm

web1 ~]# rpm -qa |grep nginx #检测安装结果

web1 ~]# /usr/local/nginx/sbin/nginx #启动nginx

web1 ~]# curl <http://127.0.0.1/> #测试nginx默认页面