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

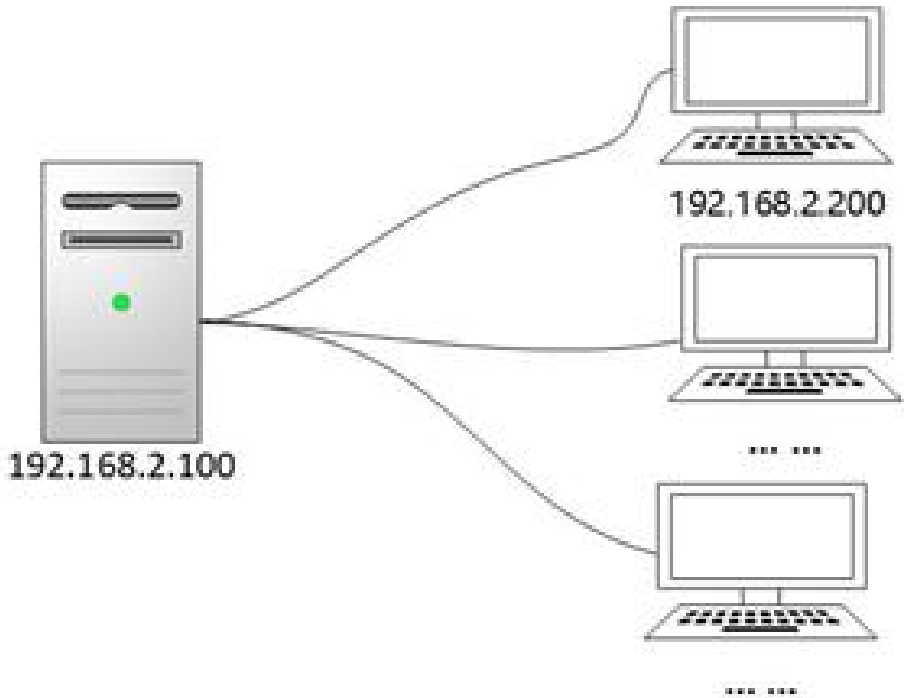
2.2.1 问题

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

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。

```
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。

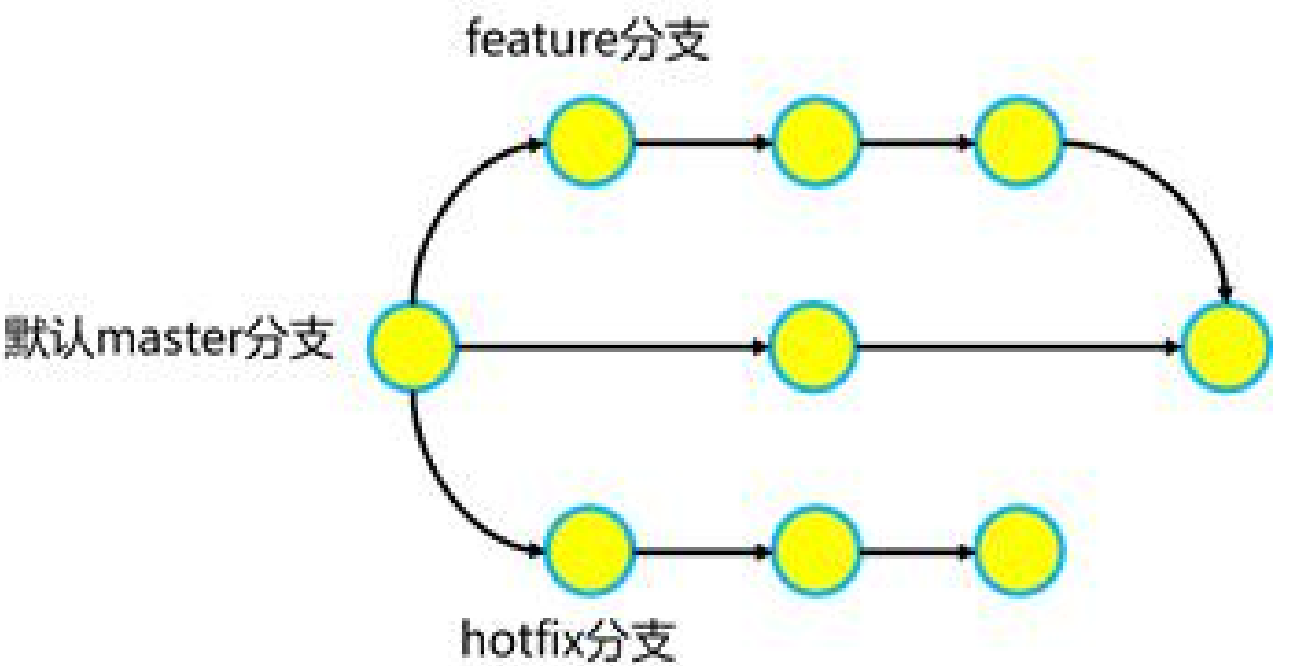
```
web2 project]# git rm init.txt  #删除文件
rm 'init.txt'
web2 project]# git commit -m "delete init.txt"
      #提交本地仓库
```

```
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

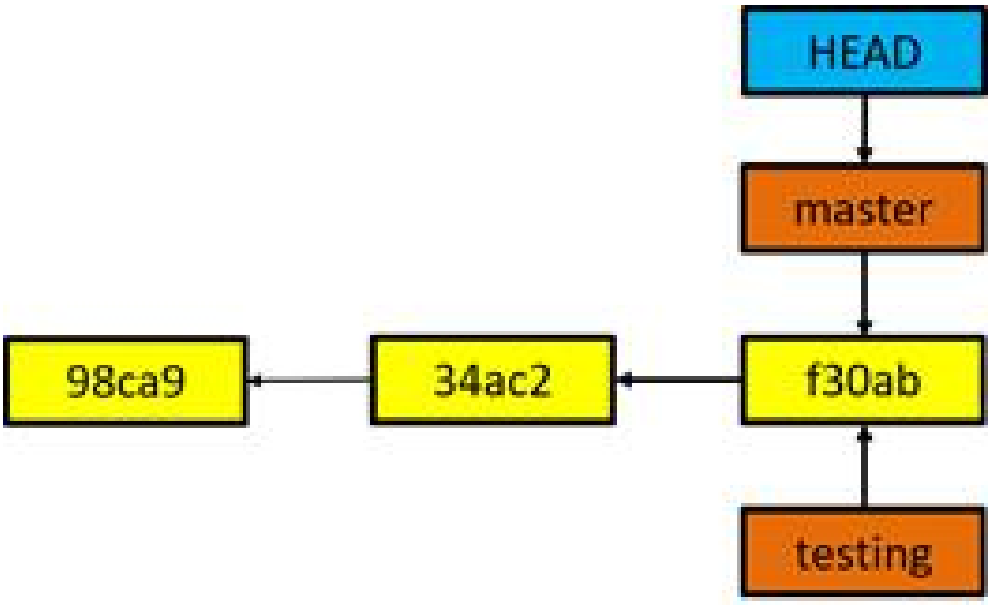
修改完成后,正常 add,commit,解决冲突

4.3 分支与 HEAD 指针的关系

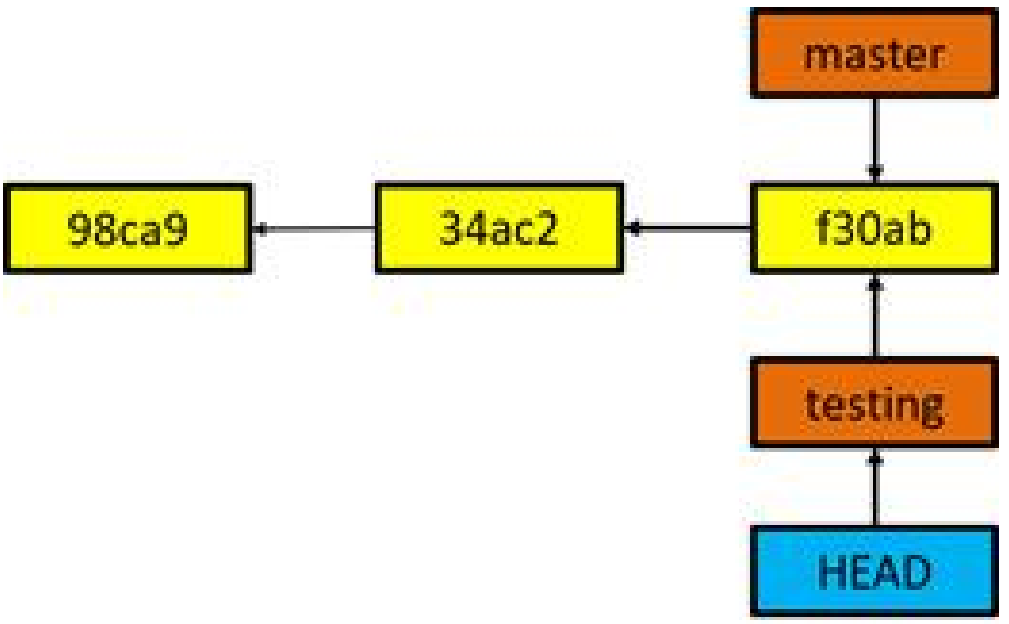
创建分支的本质是在当前提交上创建一个可以移动的指针

如何判断当前分支呢？答案是根据 HEAD 这个特殊指针

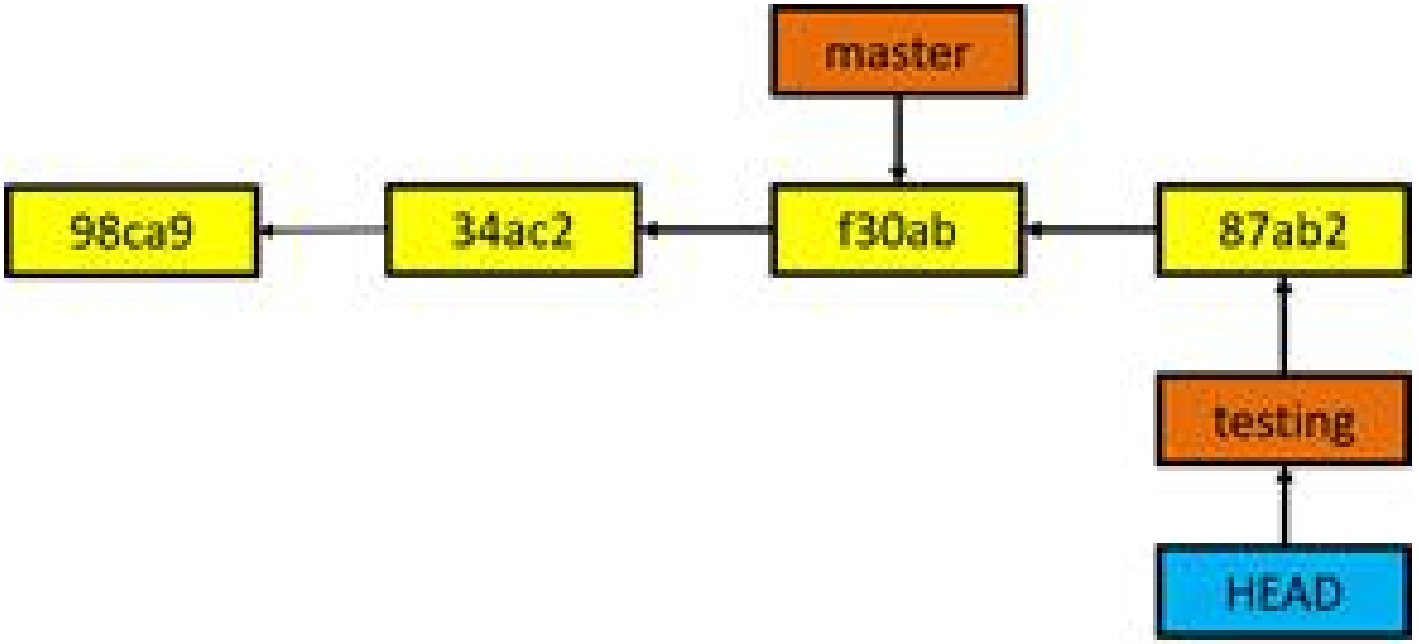
分支操作流程如图-6，图-7，图-8，图-9，图-10 所示。



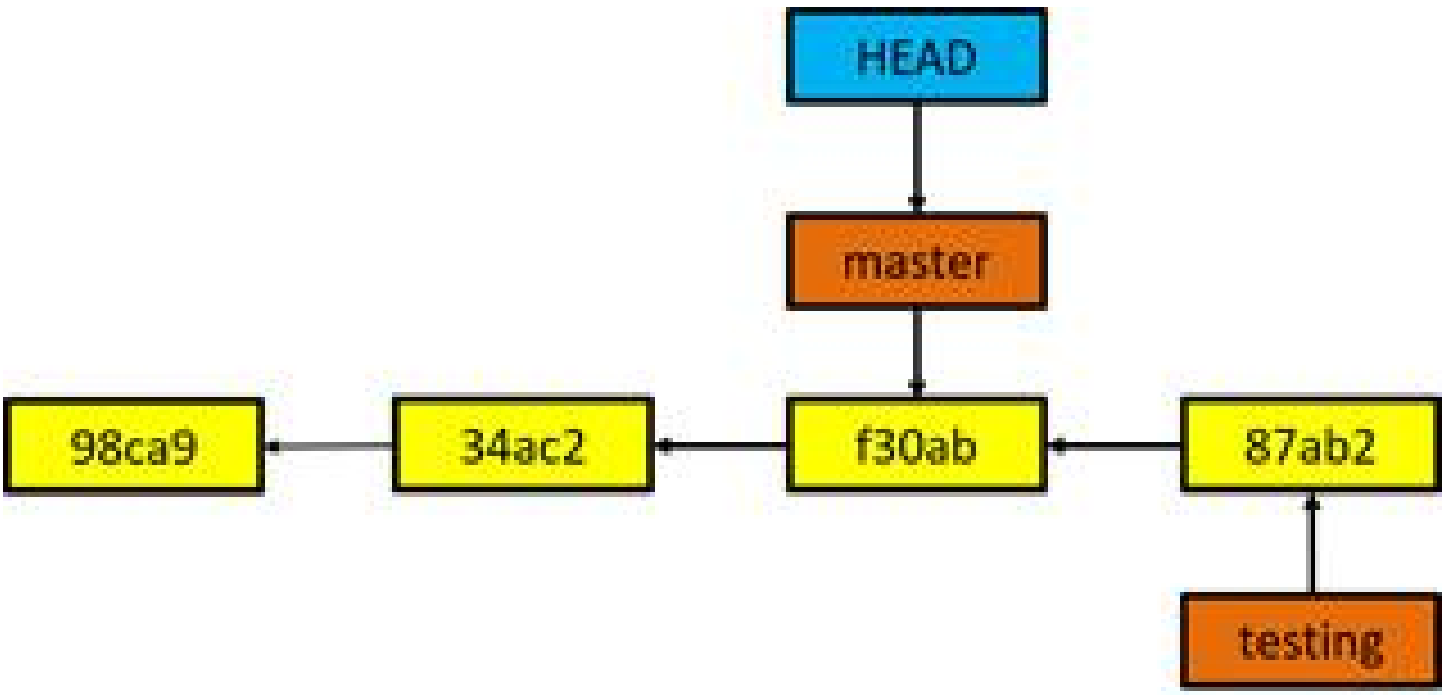
HEAD 指针指向 master 分支



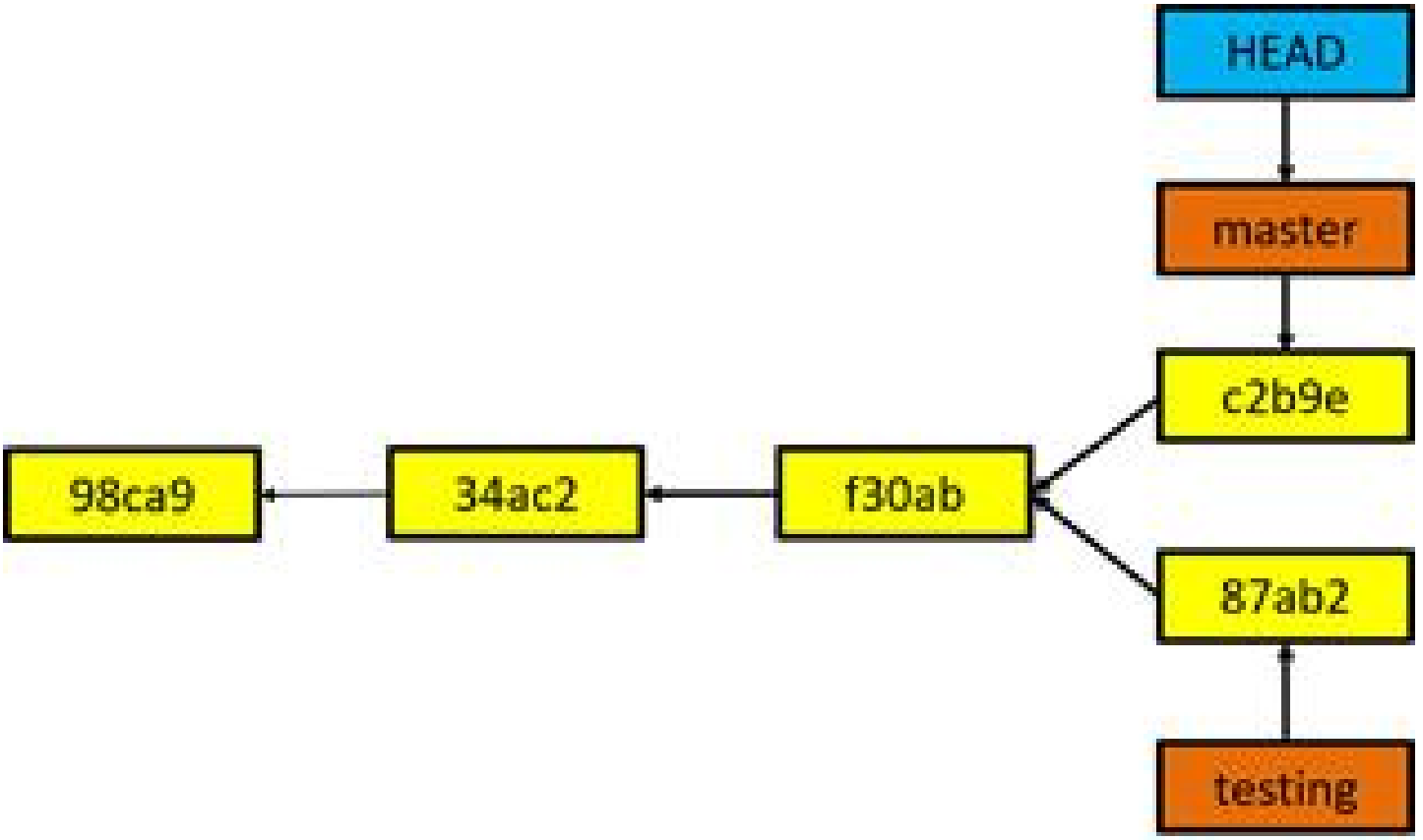
切换分支，HEAD 指针指向 testing 分支



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 #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                     #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
```