operation day 06

================================================

**为b网站虚拟主机添加日志**

**tomcat只为默认的虚拟主机添加了日志功能，如果新建的虚拟主机需要该功能就**

**按下列方式配置：**

打开配置文件，在b网站的配置中添加日志功能<Valve ..... />(该配置从默认的

虚拟主机中复制即可，最后一页)，修改prefix日志名字，suffix日志后缀

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix=" web\_b\_access\_log" suffix=".log"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

之后保存退出，重启tomcat服务，然后访问几次b网站

ls logs //查看tomcat目录下的日志目录，可以看到b网站的日志

------------------------------------------------------------------------------

**使用Varnish加速Web**

**有时网站服务器搭建好了客户会因为距离较远而访问效果不好，出现这种情况就可以用varnish工具在距离客户比较近的地区搭建缓存服务器，然后客户访问缓存服务器即可，缓存服务器会从原始站点获得数据并缓存，随着被访问与被缓存的数据越来越多，客户的访问速度就可以加快了，但由于这种方式不是一般企业可以做到，通常有需求时可以去购买CDN （内容分发网络）服务。**

**以下实验仅用来感受该服务效果，了解CDN服务的理念即可**

在web1主机开启httpd服务

yum -y install httpd

killall nginx

systemctl start httpd

systemctl stop firewalld

在proxy主机测试

[root@proxy ~]# curl 192.168.2.100

[root@proxy ~]#cd /root/lnmp\_soft

[root@proxy lnmp\_soft]# tar -xf varnish-5.2.1.tar.gz

[root@proxy lnmp\_soft]# cd varnish-5.2.1/

yum -y install gcc readline-devel pcre-devel python-docutils

./configure

make

make install

useradd varnish -s /sbin/nologin //创建varnish所需用户

cp etc/example.vcl /usr/local/etc/default.vcl //拷贝配置文件

vim /usr/local/etc/default.vcl //修改配置文件

backend default {

.host = "192.168.2.100"; //原始站点服务器的ip

.port = "80"; //原始站点服务器的端口号

}

killall nginx //如果nginx还在运行就关掉

varnishd -f /usr/local/etc/default.vcl //开启varnish服务

[root@proxy varnish-5.2.1]# curl 192.168.2.5 //测试

httpd-web1~~ //看到web1的页面则成功

**---------------------------------------------------------------**

**程序(文件或其他数据)的版本管理工具**

**svn 集中式 使用该服务时，客户端要与服务器保持**

**在线状态，数据统一保存在服务器端**

**git 分布式 使用该服务时，仅仅传输数据时需要在线**

**，数据保存在git服务器与git客户端**

**1，基本操作**

配置git服务端，这里使用web1

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

[root@web1 ~]# mkdir /var/lib/git //创建git存储数据的目录

[root@web1 ~]# git init --bare /var/lib/git/project //创建仓库，仓

库是用来存储数据的目录

配置git客户端，这里使用web2

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

[root@web2 ~]# git clone 192.168.2.100:/var/lib/git/project //克隆

服务端的仓库

[root@web2 ~]# cd project //进入仓库

[root@web2 project]# echo "web2-01" > web2-01.txt //创建测试文件

git add . //提交到暂存区

git commit -m "web2-01" //将文件保存到仓库中，-m后

面的内容是日志提示信息，首次保存会失败，按照下面2条命令输入邮箱

和用户名即可

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

git commit -m "web2-01" //再次提交文件保存到仓库中即可成功

[root@web2 project]# git push //将本地仓库中的数据推送到远

程服务器，首次推送可以按ctrl+c终止，按提示输入以下习惯配置

git config --global push.default simple //设置使用习惯

[root@web2 project]# git push //再次推送到远程服务器

**2，查看日志记录**

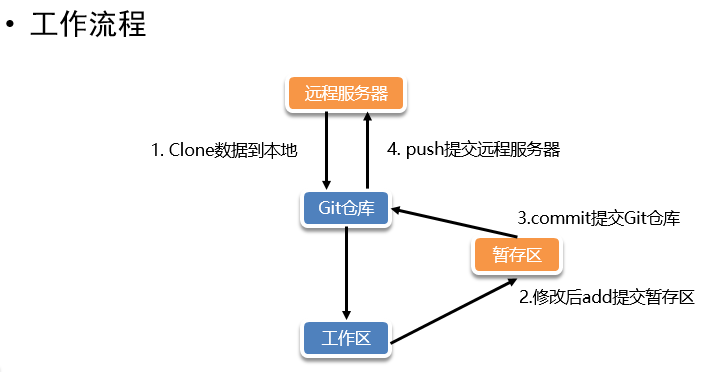
**所有通过commit提交的记录都可以通过日志查看**

[root@web2 project]# git log //查看完整日志

[root@web2 project]# git log --pretty=oneline //查看精简日志

[root@web2 project]# git log --oneline //查看最精简日志

[root@web2 project]# git reflog //查看本机操作记录



git add .

git commit -m "web2-01+"

git push

git log –oneline

----------------------------------------------------------------

**head指针**

**如果需要还原到之前的版本(时间节点)，可以利用head指针对应日志记录**

**中的随机字符串指向需要的版本**

[root@web2 project]# git log --oneline //查看日志，开头的部分就是不同版

本的随机字符串

[root@web2 project]# git reset xxxx --hard //回到过去的某个记录，其

中 xxxx是日志中显示的时间节点信息，要根据实际修改

[root@web2 project]# git reflog //查看回复记录之后的日志记录，head@{0}

代表当前所在版本位置

情景：

先创建文件 web2-03.txt ，写了一些代码,然后git add .再git commit到仓库

再修改web2-03.txt ，继续git add .再git commit ，再创建web2-04.txt，然后

再git add .再git commit ，此时发现需要最初的web2-03.txt的数据，就需要回

到过去挽救这个文件,同时为了避免回到过去丢失web2-04.txt，按照以下方式进行

**回复到过去的时间节点，找回数据思路：**

1，git reflog 查看日志，找到旧数据所在时间节点

2， git reset xxxx --hard 回到过去，xxxx是时间节点的记录

3， 把需要找回的数据，从仓库中拷贝到另外一个目录

4， git reset xxxx --hard 回到现在

5， 在之前的目录找回旧数据

[root@web2 ~]# rm -rf project

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

git常用命令：

git init --bare /var/lib/git/project //创建仓库

git clone 192.168.2.100:/var/lib/git/project //克隆仓库

cd project //进入仓库，所有关于仓库文件的命令都要进入仓库操作

git add . //提交到暂存区

git commit -m "XXXXX" //提交到仓库

git push //提交到远程服务器

git log --oneline //查看最精简日志

git reflog //查看本机操作记录

git reset XXX --hard //回到过去的时间节点

-----------------------------------------------------------------------------

**git分支**

**当项目内容比较多时，可以在git中使用分支，不同分支的文件可以互不**

**干扰而不用创建多个仓库**

[root@web2 ~]# rm -rf project/ //客户端删除仓库

[root@web1 ~]# rm -rf /var/lib/git/project //服务端删除仓库

[root@web1 ~]# git init --bare /var/lib/git/project //重新创建仓库

[root@web2 ~]# git clone 192.168.2.100:/var/lib/git/project //客户端克隆仓库

[root@web2 ~]# cd project/ //进入仓库

[root@web2 project]# echo abc > 项目流程.txt //创建测试文件

[root@web2 project]# git add . //提交到暂存区

[root@web2 project]# git commit -m "001" //提交到本地仓库

[root@web2 project]# git branch hotfix //创建名字是hotfix的分支

[root@web2 project]# git branch //查看分支，\*是所在分支

[root@web2 project]# git checkout hotfix //切换到hotfix的分支

[root@web2 project]# echo abc > hotfix001.txt //创建属于hotfix分支的文件

[root@web2 project]# git add . //提交

[root@web2 project]# git commit -m "hotfix001" //提交

[root@web2 project]# git checkout master //切换到master分支

[root@web2 project]# echo abc > master001.txt //创建属于master分支的文件

[root@web2 project]# git add . //提交

[root@web2 project]# git commit -m "hotfix001" //提交

[root@web2 project]# git merge hotfix //在master分支，将hotfix内容合并，合并

时会进入信息编辑文本，随意写一些合并理由，保存退出即可

**分支中的特殊情况：**

**如果分别在不同分支，创建同名文件，内容不同，再进行合并时，会发生冲**

**突，此时需要手工修改冲突文件，修改完之后，就可以解决冲突**

[root@web2 project]# echo abc > abc.txt //在master分支继续创建文件

[root@web2 project]# git add .

[root@web2 project]# git commit -m "abc"

[root@web2 project]# git checkout hotfix //切换到hotfix的分支

[root@web2 project]# echo xyz > abc.txt //在hotfix分支继续创建文件

[root@web2 project]# git add .

[root@web2 project]# git commit -m "abc"

[root@web2 project]# git checkout master //切换到master的分支

[root@web2 project]# git merge hotfix //再次合并分支，失败

[root@web2 project]#vim abc.txt //编写文件，由人来决定最终合并内容

保存退出即可解决冲突

[root@web2 project]# git add .

[root@web2 project]# git commit -m "abc+" //提交，完成合并