operation day 06

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

**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。**

**1，首先使用web1主机开启httpd服务**

[root@web1 ~]# ss -ntulp | grep :80 //可以先查询80端口是否被占用

killall nginx //如果nginx占用就杀掉

[root@web1 ~]# systemctl start httpd //仅仅启动httpd

使用proxy测试web1的页面，web1的防火墙需要关闭

[root@proxy ~]# curl 192.168.2.100

[root@proxy ~]# ss -ntulp | grep :80 //检查自身80端口是否被占用,如果有

就把相关服务关掉

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

./configure

make

make install

useradd varnish -s /sbin/nologin //创建varnish所需账户

**2，修改配置**

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17、18行

.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

varnishd -f /usr/local/etc/default.vcl //指定配置文件路

径并启动varnish服务

**3，测试效果**

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

问proxy的页面，可以看到原始服务器web1的内容

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

程序(文件或者其他数据)版本管理工具:

svn 集中式 用户使用该服务时,需要与服务器保持在线状态, 数据统一

保存在服务端

git 分布式 用户使用该服务时,不需要与服务器保持在线状态, 仅仅传递

数据时需要联网,数据保存在git服务器与git客户端

**1，基本操作**

首先在web1与web2安装git工具

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

[root@web1 ~]# mkdir /var/lib/git

[root@web1 ~]# git init /var/lib/git/project --bare //创建空仓库，叫Project

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

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

克隆服务器的仓库

[root@web2 ~]#cd project //进入仓库

[root@web2 project]# echo "web2\_01" > web2\_01.txt //创建测试文件

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

[root@web2 project]# git commit -m "web2\_01.txt" //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"

[root@web2 project]# git commit -m "web2\_01.txt" //再次提交文件保存到仓库中

[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

[root@web2 project]# git config --global push.default simple //配置使用习惯

[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**

git log //查看完整日志

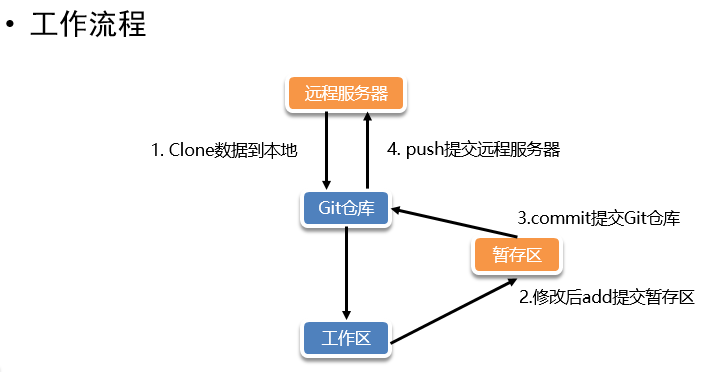
git log --pretty=oneline //查看精简日志

git log --oneline //查看最精简日志

git reflog //查看本机操作记录

[root@web2 project]# git status //查看仓库状态，如果提交任务完成，则显示干净的

工作区



[root@web2 project]# vim web2\_01.txt //编辑现有文件，可以添加些内容

[root@web2 project]# git status //再次查看仓库状态，提示有文件被修改

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

[root@web2 project]# git commit -m "web2\_01.txt+" //提交到本地仓库

[root@web2 project]# git status //再次查看仓库状态，显示干净的工作区

[root@web2 project]#git push //将本地仓库数据推送到远程git服务器中

git rm web2\_02.txt //删除文件

git add . //提交到暂存区

git commit -m "web2\_02.txt--" //提交到本地仓库

git push //提交到远程仓库

git log --oneline //查看最精简日志

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

**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:

先创建了文件abc01,写了一些代码进入,然后git add .再git commit到仓库

然后再修改abc01,继续git add .再git commit到仓库, 然后再创建了abc02

文件, 继续git add .再git commit到仓库,此时发现当初的abc01不该修改,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abc02文件,需要按

照下面方式进行

**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

git init /var/lib/git/project --bare

git clone 192.168.2.10: /var/lib/git/project

cd project

git add .

git commit -m "XXXXXXX"

git push

git status

git log --oneline

git reflog

git reset

**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 project]# git branch //查看当前分支，\*是所在位置

[root@web2 project]# git branch hotfix //创建hotfix分支

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

[root@web2 project]# echo "hotfix\_01" > hotfix\_01.txt //编写测试文件

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

[root@web2 project]# git commit -m "hotfix\_01.txt" //提交到仓库保持

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

[root@web2 project]# echo "master\_01" > master\_01.txt //编写测试文件

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

[root@web2 project]# git commit -m " master\_01.txt " //提交到仓库保持

[root@web2 project]# git checkout master //如果没在master就回到master分支

[root@web2 project]# git merge hotfix //将hotfix分支中的文件与master分支合并

然后会进入vim编辑状态,填写一些日志信息即可,说明本次合并的理由

[root@web2 project]#ls //可以查看到所有文件

下课休息 17:25回