Operation 006

GIT

版本控制软件:

ftp (覆盖关系)(共享仓库) --类似 版本控制软件(共享仓库): a1 a2(备份了a1)

Client client

主要作用:开发(更新) -企业存放代码地方(必需品)

Dos -->图像-->

共享: ftp nfs samba httpd

One people :one software wps office

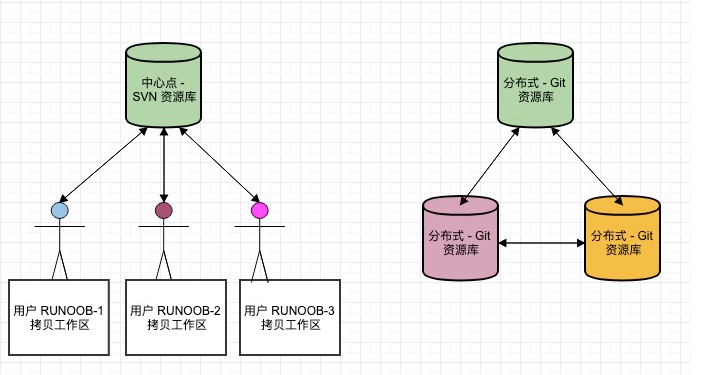
100 people :one software million people :linux

版本库: --版本控制核心

集中式:SVN subversion

分布式 GIT: 安卓 谷歌 微软 Linux 领英

Linus(linux基金会审核人 git )

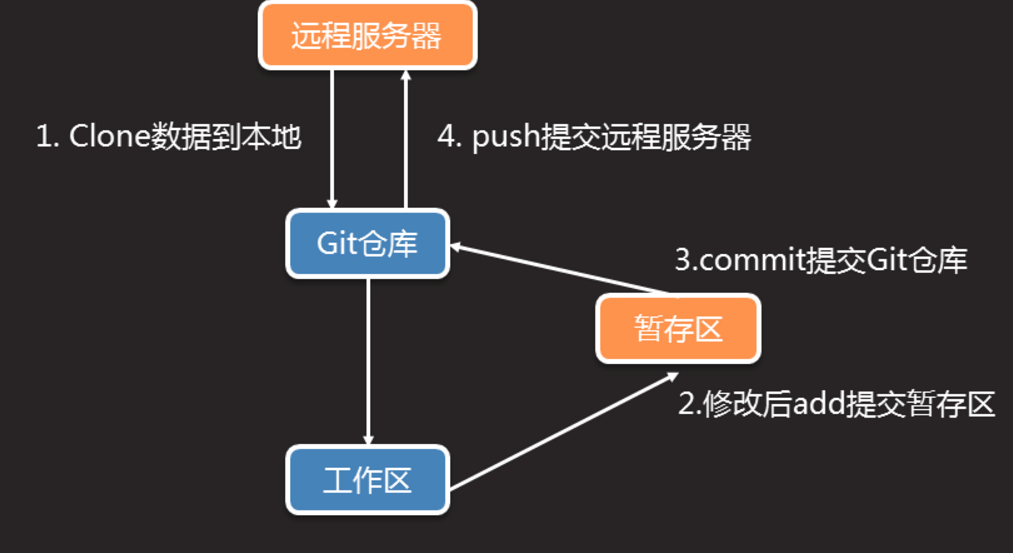


工作区：就是你在电脑里能看到的目录。

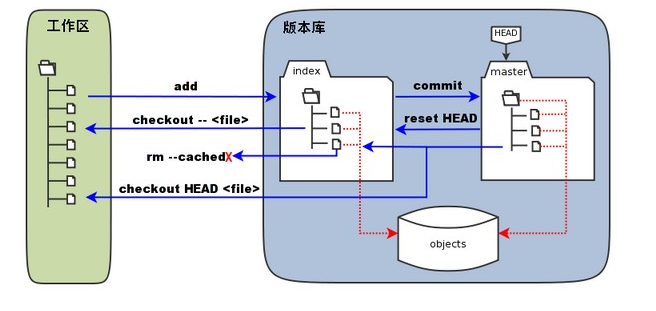
暂存区：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。

版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：



--查看版本历史



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset --hard 码 " 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

服务端:web1)

1装包: git

2 初始化一个空仓库。

mkdir /var/git

Git init /var/git/project --bare

Cd /var/git

ls //var/git

客户端(web2)

1 clone克隆服务器仓库到本地

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

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

[root@web2 ~]# cd project

[root@web2 ~]# ls

2 git add --提交到暂存区

3 git commit --提交到版本库

4 git push --提交远程服务器

5 将服务器上的数据更新到本地（web1的数据更新到web2

Git pull

6 git log \ refog

HEAD指针操作

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。没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。

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）。

[root@web2 project]# echo "new file" > new.txt

[root@web2 project]# git add .

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

2) 查看Git版本信息。

[root@web2 project]# git reflog --查看版本历史

[root@web2 project]# git log --oneline

04ddc0f num.txt:789

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}。

[root@web2 project]# git reset --hard 301c0

[root@web2 project]# git reflog

4)模拟误删后的数据还原操作。

[root@web2 project]# git rm init.txt #删除文件

rm 'init.txt'

[root@web2 project]# git commit -m "delete init.txt" #提交本地仓库

[root@web2 project]# git reflog #查看版本历史

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

[root@web2 project]# ls

demo init.txt new.txt num.txt

GIT分支管理

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。

有人把 Git 的分支模型称为必杀技特性，而正是因为它，将 Git 从版本控制系统家族里区分出来。

常见的分支规范如下：

MASTER分支（MASTER是主分支，代是码的核心）。

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）

GIT分支

查看分支

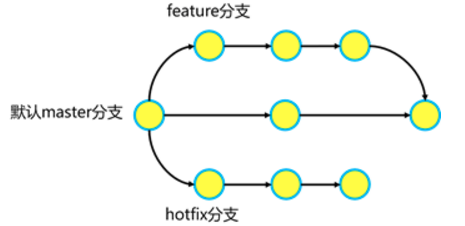
创建分支

切换分支

合并分支

解决分支的冲突

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，分支效果如图所示



1）查看当前分支。

[root@web2 project]# git status --查看当前分支

2)

[root@web2 project]# git branch -v --查看分支

\* master 0dc2b76 delete init.txt

2）创建分支。

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

[root@web2 project]# git branch -v

3)切换分支。

[root@web2 project]# git checkout hotfix

[root@web2 project]# git branch -v

feature 0dc2b76 delete init.txt

4)在新的分支上可以继续进行数据操作（增、删、改、查）。

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

[root@web2 project]# git add .

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

5)将hotfix修改的数据合并到master分支。

注意，合并前必须要先切换到master分支，然后再执行merge命令。

[root@web2 project]# git checkout master

[root@web2 project]# cat new.txt #默认master分支中没有hotfix分支中的数据

[root@web2 project]# git merge hotfix --合并分支

Updating 0dc2b76..5b4a755

6)将所有本地修改提交远程服务器。

[root@web2 project]# git push

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。

[root@web2 project]# git checkout hotfix

[root@web2 project]# echo "AAA" > a.txt

[root@web2 project]# git add .

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

[root@web2 project]# echo "BBB" > a.txt

[root@web2 project]# git add .

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

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。

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

<<<<<<< HEAD

BBB

=======

AAA

>>>>>>> hotfix

[root@web2 project]# vim a.txt #修改该文件，为最终需要的数据，解决冲突

BBB

[root@web2 project]# git add .

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

Git checkout hotfix

修改AAA >new.txt add . Commit

Git checkout master

修改BBB >new.txt add . Commit

Git merge hotfix 提示冲突

Cat

Vim new.txt

四 GIT 服务器

创建SSH协议服务器

创建Git协议服务器

创建HTTP协议服务器

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。

SSH协议服务器（支持读写操作）

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）。

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

[root@web2 ~]# ssh-copy-id 192.168.2.100

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

[root@web2 ~]# git push

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）。

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

2）创建版本库（web1主机操作）。

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

3）修改配置文件，启动git服务（web1主机操作）。

[root@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

4）客户端访问方式（web2主机操作）

[root@web2 ~]# git clone git://192.168.2.100/base\_git

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）。

[root@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）。

[root@web1 ~]# vim +11 /etc/gitweb.conf

$projectroot = "/var/git"; #添加一行

3) 创建版本仓库（web1主机操作）

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

4）启动httpd服务器

[root@web1 ~]# systemctl start httpd

五 制作NGINX RPM包

./configure

Make

Make install

/usr/local/nginx

吧/usr/local/nginx 打包压缩nginx.rpm

Rpm -ivh nginx.rpm

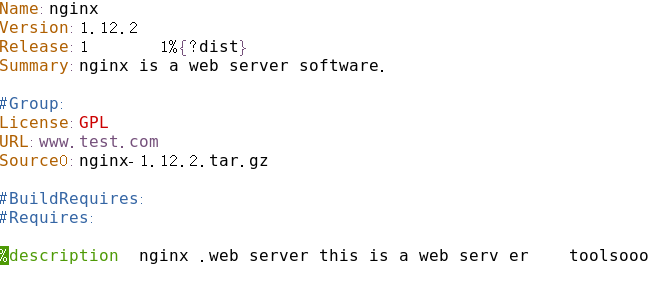
/usr/local/nginx

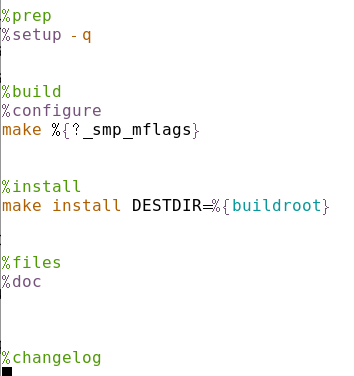
Yum -y install

GPL;1随意下载

2 修改

3 发布





%post

:安装后脚本

%setup -q (tar + cd )

./configure

Git 与 SVN 区别

Git 不仅仅是个版本控制系统，它也是个内容管理系统(CMS)，工作管理系统等。

如果你是一个具有使用 SVN 背景的人，你需要做一定的思想转换，来适应 Git 提供的一些概念和特征。

Git 与 SVN 区别点：

1、Git 是分布式的，SVN 不是：这是 Git 和其它非分布式的版本控制系统，例如 SVN，CVS 等，最核心的区别。

2、Git 把内容按元数据方式存储，而 SVN 是按文件：所有的资源控制系统都是把文件的元信息隐藏在一个类似 .svn、.cvs 等的文件夹里。

3、Git 分支和 SVN 的分支不同：分支在 SVN 中一点都不特别，其实它就是版本库中的另外一个目录。

4、Git 没有一个全局的版本号，而 SVN 有：目前为止这是跟 SVN 相比 Git 缺少的最大的一个特征。

5、Git 的内容完整性要优于 SVN：Git 的内容存储使用的是 SHA-1 哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

Git 安装配置

在使用Git前我们需要先安装 Git。Git 目前支持 Linux/Unix、Solaris、Mac和 Windows 平台上运行。

Git 各平台安装包下载地址为：http://git-scm.com/downloads

Linux 平台上安装

Git 的工作需要调用 curl，zlib，openssl，expat，libiconv 等库的代码，所以需要先安装这些依赖工具。

Linux:

1、安装和登录命令：login、 shutdown、 halt、 reboot 、mount、umount 、chsh

2、文件处理命令：file、 mkdir、 grep、dd、 find、 mv 、ls 、diff、 cat、 ln

3、系统管理相关命令： df、 top、 free、 quota 、at、 lp、 adduser、 groupadd kill、 crontab、 tar、 unzip、 gunzip 、last

4、网络操作命令：ifconfig、 ip 、ping 、 netstat 、telnet、 ftp、 route、 rlogin rcp 、finger 、mail 、nslookup

5、系统安全相关命令： passwd 、su、 umask 、chgrp、 chmod、chown、chattr、sudo、 pswho

1、TCP/IP网络基础

2、Linux企业常用服务（如dns、http、ftp、mail、nfs等）

3、Linux企业级安全原理和防范技巧（系统性能/安全、安全威胁模型和保护方法）

4、 加密/解密原理及数据安全、系统服务访问控制及服务安全基础

5、iptables安全策略构建

6、shell脚本进阶（主要是结合一些应用，写一些案例）

7、MySQL应用原理及管理入门

、http服务代理缓存加速（主要学习varnish、nginx缓存系统，了解CDN）

2、企业级负载集群（主要学习nginx、haproxy、lvs熟练掌握，熟悉负载均衡算法）

3、企业级高可用集群 （对keepalived，heartbeat等深入讲解）

4、运维监控zabbix详解（这一项还是需要花时间学习的）

5、运维自动化学习（一些开源运维自动化工具的使用如ansible、puppet、cobbler等运维自动化工具）

新闻

Hi, 请登录 我要注册 找回密码

看完Linux运维发展路线？想下自己想做技术还是走管理？

网络编辑 05月22日 9:09

概述

Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

Linux发行版：

Red Hat，CentOS，Ubuntu，Debian GNU/Linux，Fedora，Gentoo，MandrivaLinux，PCLinuxOS，openSUSE，ArchLinux，Puppylinux等

从事Linux运维行业，必须要了解和掌握Linux运维的各项技能，这是一个非常庞大的技术领域，下面分成几个层次来介绍下这一块内容。

01

初级入门（从基础命令开始入手）

Linux命令有许多强大的功能：从简单的磁盘操作、文件存取到复杂的服务器管理运行。下面把其中比较重要的和使用频率最多的命令，按照它们在系统中的作用分成几个部分介绍给大家，通过这些基础命令的学习进一步理解Linux系统：

1、安装和登录命令：login、 shutdown、 halt、 reboot 、mount、umount 、chsh

2、文件处理命令：file、 mkdir、 grep、dd、 find、 mv 、ls 、diff、 cat、 ln

3、系统管理相关命令： df、 top、 free、 quota 、at、 lp、 adduser、 groupadd kill、 crontab、 tar、 unzip、 gunzip 、last

4、网络操作命令：ifconfig、 ip 、ping 、 netstat 、telnet、 ftp、 route、 rlogin rcp 、finger 、mail 、nslookup

5、系统安全相关命令： passwd 、su、 umask 、chgrp、 chmod、chown、chattr、sudo、 pswho

学习shell脚本

Linux中的shell是一个很好用的工具。通过shell编写脚本文件，实现工作的自动化，来减轻在Linux系统上面所做的重复劳动，这样部署服务器时，能够减少时间，高效的工作。同样通过shell可以减少服务器资源的浪费，把可使用的资源最大化的给应用软件，实现资源最优的分配。

1、 大数据方向（需要对hadoop、storm等常见开源大数据系统需要深入了解）

2、 云计算方向（主要是openstack这套东西，像一些kvm、docker也是需要掌握的）

3、 运维开发（主要是python运维开发）

4、 自动化运维（在之前自动化基础上做深入）

5、 运维架构师（主要需要广度）