**Git分布式版本控制系统**

**一、Git应用背景：**

**二、Git的概述：**

**三、案例：搭建实现GIT分布式版本控制系统；**

**一、Git应用背景；**

**Git的背景：**Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符；

到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统；安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了，于是BitMover公司怒了，要收回Linux社区的免费使用权；

于是Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等；

**二、GIT的概述：**

**概述：**Git是一个开源的分布式版本控制系统，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；

**集中式版本控制器：**

集中式版本控制系统：版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆；

集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊。



**分布式版本控制器：**

分布式版本控制器：分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了；

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了；

在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已；



**分支结构：**

在 SVN 这类的版本控制系统上，分支（branch）是一个完整的目录，且这个目录拥有完整的实际文件。如果工作成员想要开启新的分支，那将会影响“全世界”！每个人都会拥有和你一样的分支。如果你的分支是用来对系统模块进行安全检查测试的，那将会像传染病一样，你改一个分支，还得让其他人重新切分支重新下载，而且这些代码很可能对稳定版本还是具有破坏性的。

在 Git上，每个工作成员可以任意在自己的本地版本库开启无限个分支。举例：当我想尝试破坏自己的程序（安检测试），并且想保留这些被修改的文件供日后使用，我可以开一个分支，做我喜欢的事。完全不需担心妨碍其他工作成员。只要我不合并及提交到主要版本库，没有一个工作成员会被影响。等到我不需要这个分支时，我只要把它从我的本地版本库删除即可，无痛无痒；

**理解Git与SVN：**<https://www.cnblogs.com/dazhidacheng/p/7478438.html>

**三、案例：搭建实现GIT分布式版本控制系统；**

**案例环境：**

|  |  |  |  |
| --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 |
| Centos 7.4 1708 64Bit | 192.168.100.101 | git.linuxfan.cn | git-2.18.0.tar.gz |

**案例步骤：**

* 安装git程序：
* 创建git管理代码的版本库：
* 上传测试文件到git版本库中：
* 测试修改代码并且上传到git：
* 将版本库中代码进行回滚：
* 理解git的工作区与缓存区：
* 代码删除管理：
* git分支管理：
* **安装git程序：**

[root@git ~]# git

...

[root@git ~]# git --version

git version 1.8.3.1

[root@git ~]# rpm -e git --nodeps

[root@git ~]# ls

anaconda-ks.cfg git-2.18.0.tar.gz

[root@git ~]# tar zxvf git-2.18.0.tar.gz

[root@git ~]# cd git-2.18.0

[root@git git-2.18.0]# ./configure --prefix=/usr/local/git

[root@git git-2.18.0]# make

GIT\_VERSION = 2.18.0

\* new build flags

CC credential-store.o

In file included from credential-store.c:1:0:

cache.h:19:18: 致命错误：zlib.h：没有那个文件或目录

#include <zlib.h>

^

编译中断。

make: \*\*\* [credential-store.o] 错误 1

[root@git git-2.18.0]# yum -y install zlib-devel

[root@git git-2.18.0]# make && make install

[root@git git-2.18.0]# cd

[root@git ~]# ls /usr/local/git/

bin libexec share

[root@git ~]# cat <<END>> /etc/profile

export PATH=$PATH:/usr/local/git/bin

END

[root@git ~]# source /etc/profile

[root@git ~]# git --version

git version 2.18.0

* **创建git管理代码的版本库：**

[root@git ~]# cd /tmp

[root@git tmp]# mkdir repository

[root@git tmp]# cd repository

[root@git repository]# pwd

/tmp/repository

[root@git repository]# git init

已初始化空的 Git 仓库于 /tmp/repository/.git/

[root@git repository]# ls -a

. .. .git

* **上传测试文件到git版本库中：**

[root@git repository]# cat <<END >>1.txt

[www.linuxfan.cn](http://www.linuxfan.cn/)

END

[root@git repository]# cat 1.txt

[www.linuxfan.cn](http://www.linuxfan.cn/)

[root@git repository]# git add 1.txt ##添加文件

[root@git repository]# git commit -m "1" ##提交文件

[master（根提交） ae97fb0] 1

Committer: root <root@git.linuxfan.cn>

您的姓名和邮件地址基于登录名和主机名进行了自动设置。请检查它们正确

与否。您可以对其进行设置以免再出现本提示信息。运行如下命令在编辑器

中编辑您的配置文件：

git config --global --edit

设置完毕后，您可以用下面的命令来修正本次提交所使用的用户身份：

git commit --amend --reset-author

1 file changed, 2 insertions(+) ##1 file changed：1个文件被改动（我们新添加的1.txt文件）

2 insertions： ##插入了两行内容（1.txt有两行内容）

create mode 100644 1.txt ##-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的

[root@git repository]# git config --global --edit ##更改默认身份

# This is Git's per-user configuration file.

[user]

# Please adapt and uncomment the following lines:

# name = linuxfan

# email = admin@linuxfan.cn

[root@git repository]# touch {2..6}.txt ##使用git一次性提交多个文件

[root@git repository]# git add {2..6}.txt

[root@git repository]# git commit -m "11"

...

* **测试修改代码并且上传到git：**

[root@git repository]# vi 1.txt

[www.baidu.com](http://www.baidu.com/)

:wq

[root@git repository]# git status ##查看修改的文件

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

修改： 1.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@git repository]# git diff 1.txt ##查看文件的修改内容

diff --git a/1.txt b/1.txt

index 0c4853c..5822db3 100644

--- a/1.txt

+++ b/1.txt

@@ -1 +1 @@

-[www.linuxfan.cn](http://www.linuxfan.cn/)

+www.baidu.com

[root@git repository]# git add 1.txt ##提交文件到git版本库

[root@git repository]# git status

位于分支 master

要提交的变更：

（使用 "git reset HEAD <文件>..." 以取消暂存）

修改： 1.txt

[root@git repository]# git commit -m "2"

...

[root@git repository]# git status

位于分支 master

无文件要提交，干净的工作区

[root@git repository]# git log ##查看版本库中代码改动日志

commit f4636d3380cbc91248f2e79febb1d2c5712aab6c (HEAD -> master)

Author: root <root@git.linuxfan.cn>

Date: Tue Sep 25 23:59:46 2018 +0800

2

commit 8774ec286fdd2771ba7072c1fee9b6f95608f9d9

Author: root <root@git.linuxfan.cn>

Date: Tue Sep 25 23:57:08 2018 +0800

11

commit 70c4e23fe6ef68c8011e29d4902241f501ffe04c

Author: root <root@git.linuxfan.cn>

Date: Tue Sep 25 23:55:42 2018 +0800

1

[root@git repository]# git log --pretty=oneline ##简化输入上一条命令

f4636d3380cbc91248f2e79febb1d2c5712aab6c (HEAD -> master) 2

8774ec286fdd2771ba7072c1fee9b6f95608f9d9 11

70c4e23fe6ef68c8011e29d4902241f501ffe04c 1

* **将版本库中代码进行回滚：**

[root@git repository]# git reset --hard HEAD^ ##回滚到上一个版本

HEAD 现在位于 3d3ffb3 1

[root@git repository]# cat 1.txt

[www.linuxfan.cn](http://www.linuxfan.cn/)

[root@git repository]# git log --pretty=oneline ##查看代码版本

8774ec286fdd2771ba7072c1fee9b6f95608f9d9 (HEAD -> master) 11

70c4e23fe6ef68c8011e29d4902241f501ffe04c 1

[root@git repository]# git log ##查看git的日志

...



[root@git repository]# git reset --hard f463 ##将代码反向回滚

HEAD 现在位于 f4636d3 2

[root@git repository]# cat 1.txt

[www.baidu.com](http://www.baidu.com/)

* **理解git的工作区与缓存区：**

工作区（Working Directory）：就是你在电脑里能看到的目录；

版本库（Repository）：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。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD；



[root@git repository]# ls

1.txt 2.txt 3.txt 4.txt 5.txt 6.txt

[root@git repository]# vi 1.txt ##修改工作区现有文件

[www.sina.com](http://www.sina.com/)

[root@git repository]# vi 1.file ##工作区新增文件

好好学习

[root@git repository]# git status ##查看git状态，发现两个文件区分开了

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

修改： 1.txt

未跟踪的文件:

（使用 "git add <文件>..." 以包含要提交的内容）

1.file

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@git repository]# git add 1.txt ##将两个文件提交

[root@git repository]# git add 1.file

[root@git repository]# git status

位于分支 master

要提交的变更：

（使用 "git reset HEAD <文件>..." 以取消暂存）

新文件： 1.file

修改： 1.txt



[root@git repository]# git commit -m "111" ##所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支；

[master 2ef3aee] 111

Committer: root <root@git.linuxfan.cn>

您的姓名和邮件地址基于登录名和主机名进行了自动设置。请检查它们正确

与否。您可以对其进行设置以免再出现本提示信息：

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

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

设置完毕后，您可以用下面的命令来修正本次提交所使用的用户身份：

git commit --amend --reset-author

2 files changed, 2 insertions(+), 1 deletion(-)

create mode 100644 1.file

[root@git repository]# git status

位于分支 master

无文件要提交，干净的工作区

[root@git repository]# git log --pretty=oneline

2ef3aee23c0e31423f75d0146be7e51e426cfa41 (HEAD -> master) 111

f4636d3380cbc91248f2e79febb1d2c5712aab6c 2

8774ec286fdd2771ba7072c1fee9b6f95608f9d9 11

70c4e23fe6ef68c8011e29d4902241f501ffe04c 1



* **代码删除管理：**

[root@git repository]# ls

1.file 1.txt 2.txt 3.txt 4.txt 5.txt 6.txt

[root@git repository]# rm -rf 1.file

[root@git repository]# git status

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add/rm <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

删除： 1.file

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@git repository]# git rm 1.file

rm '1.file'

[root@git repository]# git commit -m "1111"

[master bf75860] 1111

Committer: root <root@git.linuxfan.cn>

您的姓名和邮件地址基于登录名和主机名进行了自动设置。请检查它们正确

与否。您可以对其进行设置以免再出现本提示信息：

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

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

设置完毕后，您可以用下面的命令来修正本次提交所使用的用户身份：

git commit --amend --reset-author

1 file changed, 1 deletion(-)

delete mode 100644 1.file

[root@git repository]# git status

位于分支 master

无文件要提交，干净的工作区

[root@git repository]# ls

1.txt 2.txt 3.txt 4.txt 5.txt 6.txt

[root@git repository]# rm -rf 1.txt

[root@git repository]# git status

位于分支 master

尚未暂存以备提交的变更：

（使用 "git add/rm <文件>..." 更新要提交的内容）

（使用 "git checkout -- <文件>..." 丢弃工作区的改动）

删除： 1.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）

[root@git repository]# git checkout -- 1.txt

[root@git repository]# ls

1.txt 2.txt 3.txt 4.txt 5.txt 6.txt

* **git分支管理：**

分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。



当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



[root@git repository]# git checkout -b dev

切换到一个新分支 'dev'

##git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

git branch dev

git checkout dev

[root@git repository]# git branch ##列出所有分支，当前分支前面会标一个\*号。

\* dev

Master

[root@git repository]# git status

位于分支 dev

无文件要提交，干净的工作区