**git分布式管理数据的工具**

**典型的客户/服务器系统**

-版本库是版本控制的核心

-任意数量客户端

-客户端通过写数据库分享代码

分布式版本控制

集中式版本控制系统

-开发者之间共用一个仓库（repository）

-所有操作需要联网

**分布式版本控制系统**

-每个开发者都是一个仓库的完整克隆 每个人都是服务器

-支持断网工作 只有在上传或者下载仓库信息的时候需要断网

分布式版本控制

-git基本概念

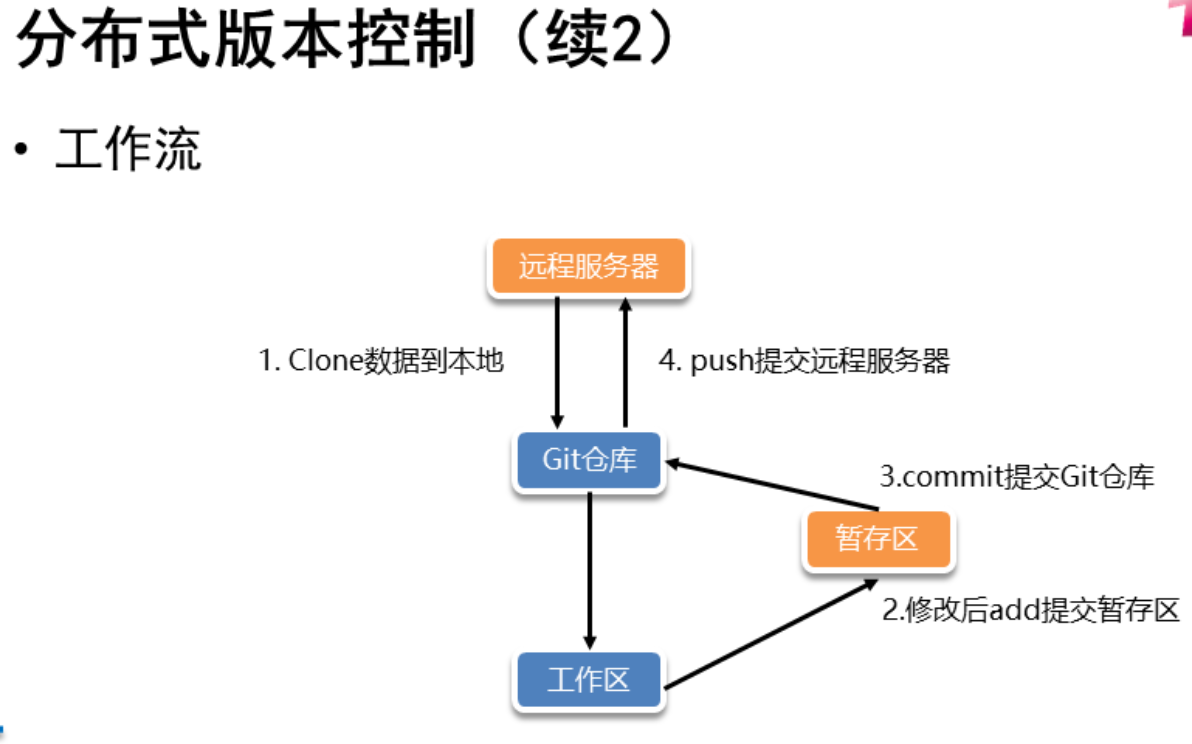
Git仓库：保存所有数据的地方

-工作区

从仓库提取出来的文件 放在磁盘上供你使用/修改

-暂存区

就是一个文件 索引文件 保存了下次将提交的文件列表



**部署安装git环境 部署服务端环境**

* 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

**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。

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

2)初始化一个空仓库。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web1 ~]# mkdir -p /var/lib/git
2. [root@web1 ~]# git init /var/lib/git/project --bare
3. [root@web1 ~]# ls /var/lib/git/project
4. config description HEAD hooks info objects refs

**客户端测试(192.168.2.200作为客户端主机)**



1) clone克隆服务器仓库到本地。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 ~]# yum -y install git
2. [root@web2 ~]# git clone root@192.168.2.100:/var/lib/git/project
3. [root@web2 ~]# cd project
4. [root@web2 ~]# ls

2) 修改git配置。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git config --global user.email "you@example.com"
2. [root@web2 project]# git config --global user.name "Your Name"
3. [root@web2 project]# cat ~/.gitconfig
4. [user]
5. email = you@example.com
6. 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)。

1. [root@web2 project]# echo "init date" > init.txt
2. [root@web2 project]# mkdir demo
3. [root@web2 project]# cp /etc/hosts demo

4） 查看仓库中数据的状态。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git status

5） 将工作区的修改提交到暂存区。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git add .

6) 将暂存区修改提交到本地仓库。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git commit -m "注释，可以为任意字符 在将来查看日志的时候看的"
2. [root@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)。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git config --global push.default simple
2. [root@web2 project]# git push
3. root@192.168.2.100's password: 输入服务器root密码
4. [root@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。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git pull

9) 查看版本日志。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git log
2. [root@web2 project]# git log --pretty=oneline
3. [root@web2 project]# git log --oneline
4. [root@web2 project]# git reflog

HEAD指针概述

HEAD指针概述

-HEAD指针是一个可以在任何分支和版本移动的指针

-通过移指针我们可以将数据还原至任何版本

* 查看Git版本信息
* 移动指针
* 通过移动HEAD指针恢复数据

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。

**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）。

1. [root@web2 project]# echo "new file" > new.txt
2. [root@web2 project]# git add .
3. [root@web2 project]# git commit -m "add new.txt"
5. [root@web2 project]# echo "first" >> new.txt
6. [root@web2 project]# git add .
7. [root@web2 project]# git commit -m "new.txt:first line"
9. [root@web2 project]# echo "second" >> new.txt
10. [root@web2 project]# git add .
11. [root@web2 project]# git commit -m "new.txt:second"
13. [root@web2 project]# echo "third" >> new.txt
14. [root@web2 project]# git add .
15. [root@web2 project]# git commit -m "new.txt:third"
16. [root@web2 project]# git push
18. [root@web2 project]# echo "123" > num.txt
19. [root@web2 project]# git add .
20. [root@web2 project]# git commit -m "num.txt:123"
22. [root@web2 project]# echo "456" > num.txt
23. [root@web2 project]# git add .
24. [root@web2 project]# git commit -m "num.txt:456"
26. [root@web2 project]# echo "789" > num.txt
27. [root@web2 project]# git add .
28. [root@web2 project]# git commit -m "num.txt:789"
29. [root@web2 project]# git push
30. 查看Git版本信息。
31. [root@web2 project]# git reflog
33. [root@web2 project]# git log --oneline
34. 04ddc0f num.txt:789
35. 7bba57b num.txt:456
36. 301c090 num.txt:123
37. b427164 new.txt:third
38. 0584949 new.txt:second
39. ece2dfd new.txt:first line
40. e1112ac add new.txt
41. 1a0d908 初始化

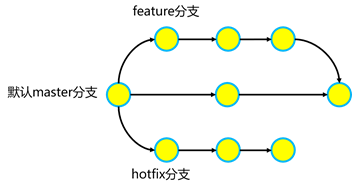
3）移动HEAD指针，将数据还原到任意版本。

提示：当前HEAD指针为HEAD@{0}。

1. [root@web2 project]# git reset --hard 301c0
2. [root@web2 project]# git reflog
3. 301c090 HEAD@{0}: reset: moving to 301c0
4. 04ddc0f HEAD@{1}: commit: num.txt:789
5. 7bba57b HEAD@{2}: commit: num.txt:456
6. 301c090 HEAD@{3}: commit: num.txt:123
7. b427164 HEAD@{5}: commit: new.txt:third
8. 0584949 HEAD@{6}: commit: new.txt:second
9. ece2dfd HEAD@{7}: commit: new.txt:first line
10. e1112ac HEAD@{8}: commit: add new.txt
11. 1a0d908 HEAD@{9}: commit (initial): 初始化
12. [root@web2 project]# cat num.txt                #查看文件是否为123
13. 123
14. [root@web2 project]# git reset --hard 7bba57b
15. [root@web2 project]# cat num.txt                #查看文件是否为123，456
16. 123
17. 456
19. [root@web2 project]# git reflog                #查看指针移动历史
20. 7bba57b HEAD@{0}: reset: moving to 7bba57b
21. 301c090 HEAD@{1}: reset: moving to 301c0
22. … …
23. [root@web2 project]# git reset --hard 04ddc0f    #回到最后一次修改的版本

## **Git分支操作**

* 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-5所示。

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）。

**查看并创建分支**

1）查看当前分支。

1. [root@web2 project]# git status
2. # On branch master
3. nothing to commit, working directory clean
5. [root@web2 project]# git branch -v
6. \* master 0dc2b76 delete init.txt

2）创建分支

1. [root@web2 project]# git branch hotfix
2. [root@web2 project]# git branch feature
3. [root@web2 project]# git branch -v
4. feature 0dc2b76 delete init.txt
5. hotfix 0dc2b76 delete init.txt
6. \* master 0dc2b76 delete init.txt

**切换与合并分支**

1）切换分支。

1. [root@web2 project]# git checkout hotfix
2. [root@web2 project]# git branch -v
3. feature 0dc2b76 **delete** init.txt
4. \* hotfix 0dc2b76 **delete** init.txt
5. 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）。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# echo "fix a bug" >> new.txt
2. [root@web2 project]# git add .
3. [root@web2 project]# git commit -m "fix a bug"

3）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。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git checkout master
2. [root@web2 project]# cat new.txt        #默认master分支中没有hotfix分支中的数据
3. [root@web2 project]# git merge hotfix
4. Updating 0dc2b76..5b4a755
5. Fast-forward
6. new.txt | 1 ++
7. 1 file changed, 1 **insertions**(+)

**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。

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

1. [root@web2 project]# git checkout hotfix
2. [root@web2 project]# echo "AAA" > a.txt
3. [root@web2 project]# git add .
4. [root@web2 project]# git commit -m "add a.txt by hotfix"
6. [root@web2 project]# git checkout master
7. [root@web2 project]# echo "BBB" > a.txt
8. [root@web2 project]# git add .
9. [root@web2 project]# git commit -m "add a.txt by master"
10. [root@web2 project]# git merge hotfix
11. 自动合并 a.txt
12. 冲突（添加/添加）：合并冲突于 a.txt
13. 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。

1. [root@web2 project]# cat a.txt                #该文件中包含有冲突的内容
2. <<<<<<< HEAD
3. BBB
4. =======
5. AAA
6. >>>>>>> hotfix
7. [root@web2 project]# vim a.txt #修改该文件，为最终需要的数据，解决冲突
8. BBB
10. [root@web2 project]# git add .
11. [root@web2 project]# git commit -m "resolved"

分支指针与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。

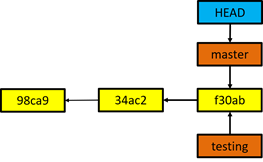


图-6 HEAD指针指向master分支

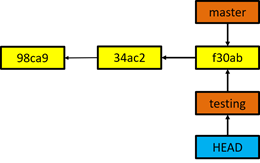


图-7 切换分支，HEAD指针指向testing分支

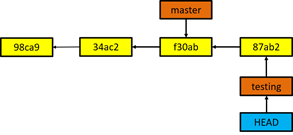


图-8 在testing分支中修改并提交代码

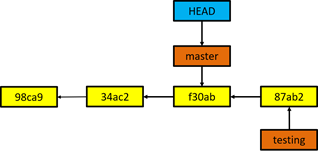


图-9 将分支切换回master分支

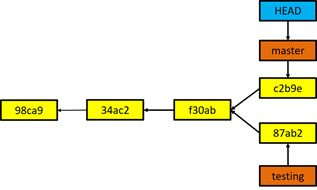


图-10 在master分支中修改数据，更新版本

**Git服务器**

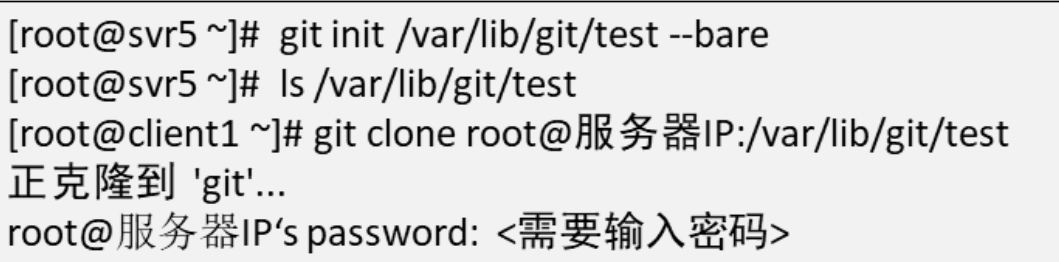
Ssh协议

密码认证访问

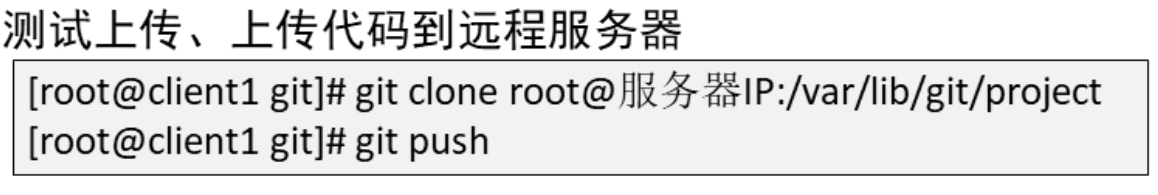
-服务器安装git

-使用git命令初始化仓库版本

-客户端使用ssh远程访问（可读可写）



Ssh协议 免密吗远程git 服务器需要制作ssh-keygen ssh-copy-id传输密钥



* 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）

1）创建基于密码验证的SSH协议服务器（web1主机操作）。

1. [root@web1 ~]# git init --bare /var/lib/git/base\_ssh

2)客户端访问的方式（web2主机操作）。

1. [root@web2 ~]# git clone root@192.168.2.100:/var/lib/git/base\_ssh
2. [root@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）。

1. [root@web2 ~]# ssh-keygen -f /root/.ssh/id\_rsa -N ''
2. [root@web2 ~]# ssh-copy-id 192.168.2.100
3. [root@web2 ~]# git clone root@192.168.2.100:/var/git/base\_ssh
4. [root@web2 ~]# git push

**git协议 访问支持无授权访问（只读）**

**-服务器安装git-daemon软件包**

**Git协议服务器（只读操作的服务器）**

安装git-daemon软件包（web1主机操作

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

2）创建版本库（web1主机操作

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

3）修改配置文件，启动git服务（web1主机操作

1. [root@web1 ~]# cat /usr/lib/systemd/system/git@.service
2. #仅查看即可
4. [root@web1 ~]# systemctl start git.socket

#备注:默认git配置为只读服务器，如果需要读写，

[则需要修改git@.service配置文件](mailto:则需要修改git@.service配置文件)

在git-daemon命令后面添加--enable=receive-pack参数即可。

4）客户端访问方式（web2主机操作

**http协议 支持浏览器访问**

**-服务器需要安装gitweb软件包**



**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）。

1. [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）。

1. [root@web1 ~]# vim +11 /etc/gitweb.conf
2. $projectroot = "/var/lib/git";

3) 创建版本仓库（web1主机操作）

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

4）启动httpd服务器

1. [root@web1 ~]# systemctl start httpd

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。

5. 命令行操作（需要联网的主机，如真实机）

[root@pc001 ~]# yum -y install git

[root@pc001 ~]# git clone https://github.com/账户名称/仓库名称

#clone指令用于将服务器仓库中的资料打包下载到本地

[root@pc001 ~]# cd 仓库名称

[root@pc001 ~]# 任意修改文件，或新建文件

[root@pc001 ~]# git add .

#add添加新文件

[root@pc001 ~]# git commit -m "test"

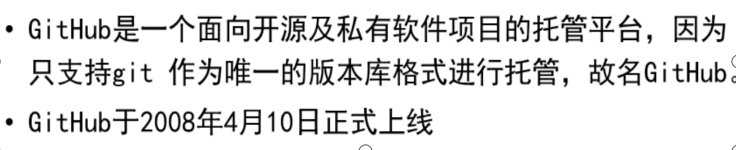
[root@pc001 ~]# git push

#commit和push实现提交代码的功能

[root@pc001 ~]# git pull

#pull可以从githuab服务器拉取数据到本地

**GitHub**



**Git 命令总结**

Git基本命令

Git软 git

Git init /var/lib/git/新建仓库名称 --bare 新建一个空的git仓库

Clone 将远程服务器的仓库克隆到本地

语法：git clone root@192.168.2.100:/var/lib/git/服务端的仓库名称（客户端第一次使用）

Config 修改git配置

Add (./文件名) 将工作区的修改提交到暂存区

Commit -m “修改文件描述信息” 提交暂存区修改到本地仓库

Push 提交需改到远程服务器

Git status 查看仓库中的数据状态

Git push 将本地仓库中的数据推送到远程服务器

git config --global push.default simple 定义如何推送（更安全的推送）

Git pull 可以将服务器上新的数数据更新到本地

log查看历史日志

Git log 不传入任何参数的默认情况下，

git log 会按时间先后顺序列出所有的提交，

最近的更新排在最上面

其中一个比较有用的选项是 -p 或 --patch

你也可以限制显示的日志条目数量，例如使用

-2 选项来只显示最近的两次提交

Git log --pretty=oneline

另一个非常有用的选项是 --pretty。 这个选项可以使用不同于默认格式的方式展示提交历史。 这个选项有一些内建的子选项供你使用。 比如 oneline 会将每个提交放在一行显示，在浏览大量的提交时非常有用。 另外还有 short，full 和 fuller 选项，它们展示信息的格式基本一致，

Git reflog

git log --oneline

**修改git配置**

[copytextpop-up](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN202001/OPERATION/DAY06/CASE/01/index.html)

[root@web2 project]# git config --global user.email "you@example.com"

[root@web2 project]# git config --global user.name "Your Name"

[root@web2 project]# cat ~/.gitconfig

[user]

    email = you@example.com

    name = Your Name

HEAD指针操作

Git log --oneline 查看日志信息

Git reset --hard (每一个日志消息的id号缩写) 回到某一个时光

Git reflog 查看指针移动历史

Git分支操作

Git status 查看当前的状态

Git branch -v 查看所有分支

Git branch 分支名称 创建分支

Git checkout 分支名称 切换到分支 在新的分支上可以继续修改代码 修改文件创建文件等操作 正常add commit提交版本库

合并分支

Git checkkout master 合并前一定切换到master分支

Git marge 分支名 执行marge命令合并分支