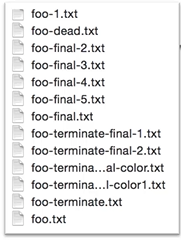
Git笔记

# 版本控制系统

1. 即VCS（version control system）是一种记录若干文件的修订记录的系统，它帮助我们查阅或回到某个历史版本
   1. “人肉”VCS
   2. LVCS本地
   3. CVCS集中式
   4. DVCS分布式

## 人肉VCS



## Local VCS——本地式

1. 举例
   1. RCS（Revision Control System）

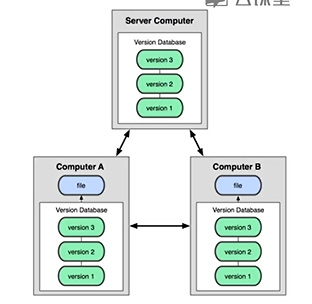
假如，两个用户需要协同开发？

## CVCS-集中式

1. 举例
   1. CVS(Concurrent Versions System)
   2. **SVN(Subversion)**
   3. Perforce

假如中央服务器发生故障？

## DVCS-分布式

1. 举例
   1. **Git**
   2. Mercurial

# 分支模型

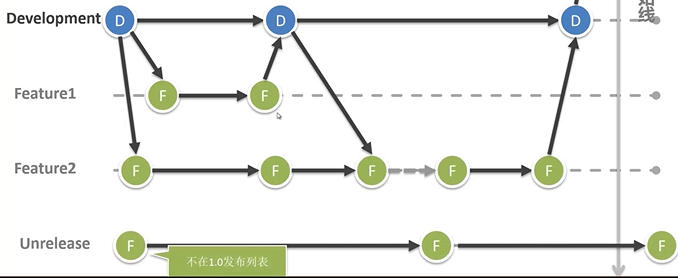
## 分支与分支模型

1. 分支
   1. 从目标仓库获得一份项目拷贝，每条拷贝都有和原仓库功能一样的开发线
2. 分支模型（branching model）/工作流（workflow）
   1. 一个围绕项目开发/部署/测试等工作流程的分支操作（创建，合并等）规范集合

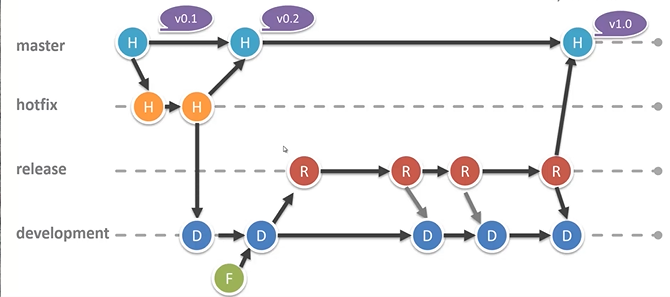
## 产品级的分支模型

1. 常驻分支
   1. Development
      1. 从master创建
   2. Production（master）——都是需要可发布的，一旦生成不可改变
      1. 默认分支
2. 活动分支
   1. Feature——经常用到的分支
      1. 从development创建
   2. Hotfix：如hotfix-36——BUG修复
      1. 从master创建
   3. Release：如release-110——标志开始正式发布
      1. 从development分支创建

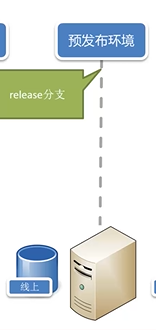
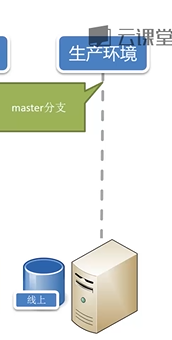
## 分支模型——特性开发

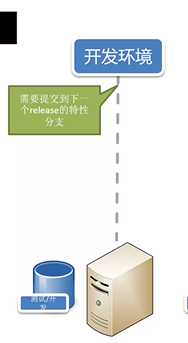
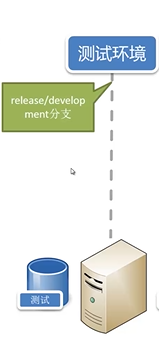


## 分支模型——发布线



## 四个环境





# Git

## Git是什么

1. Git是一个免费开源的的分布式版本控制系统（DVCS）
2. Git是一个基于内容寻址的存储系统

## Git历史

1. Git的出现离不开Linux
   1. 1991-2002：几乎无版本控制（patch包）
   2. 2002-2005：BitKepper
   3. 2005-至今：git



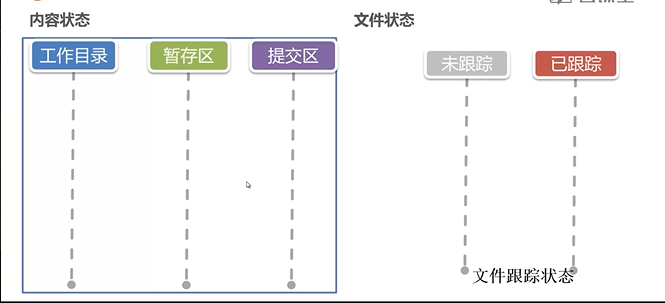
## 优势

1. 快！
2. 完全的分布式
3. 轻量级的分支操作
4. Git已经成为现实意义上的标准
5. 社区成熟活跃

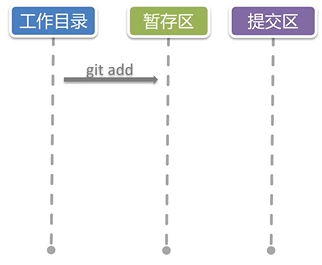
## Git 命令

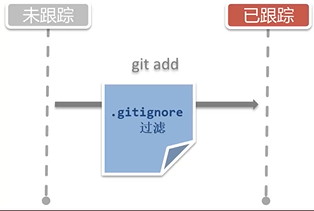
1. Git config——配置git
   1. 用户配置
      1. Git config --global user.name “xxx”
      2. Git config –global user.email “xxx”
   2. 配置级别
      1. –local【默认，高优先级】：只影响本仓库
      2. –global【中优先级】：影响到所有当前用户的git仓库
      3. --system【低优先级】：影响全系统的的git仓库
2. git init——初始化仓库



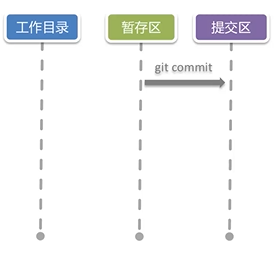
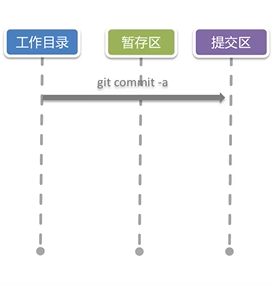
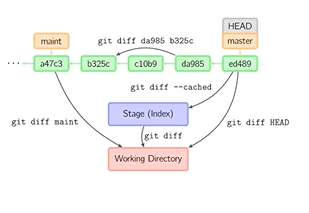
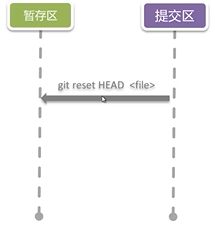
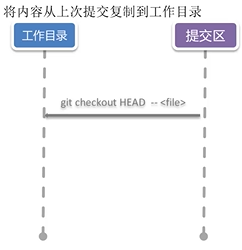
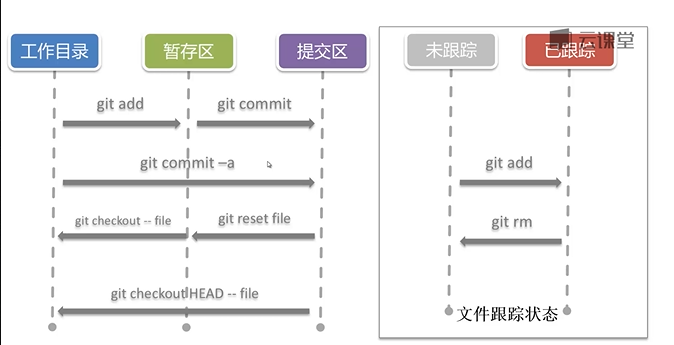
1. git status——对状态的跟踪
2. git add——添加文件内容到**暂存区（同时文件被跟踪）**
   1. git add . ——批量添加



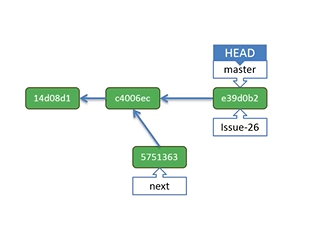
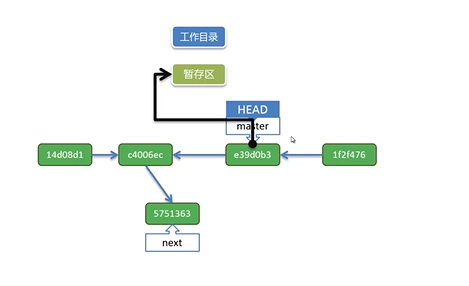
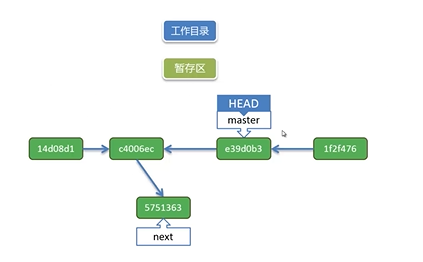
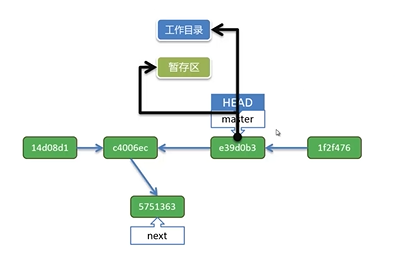


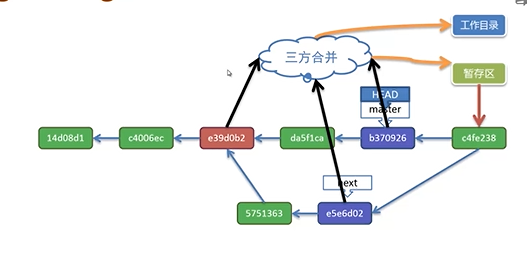
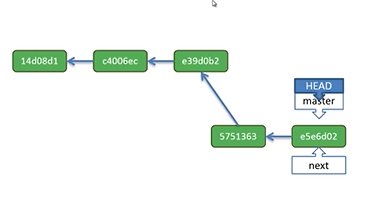
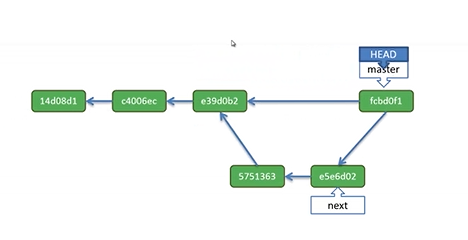
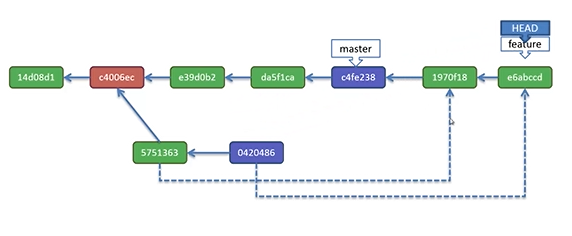
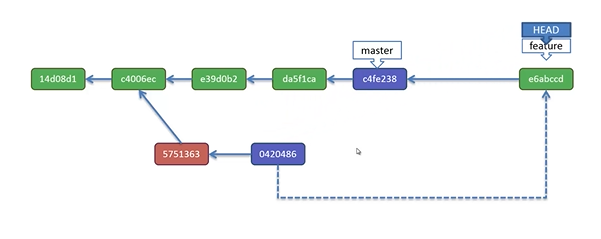
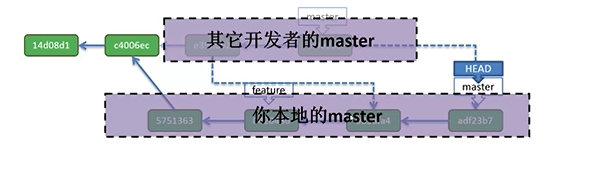
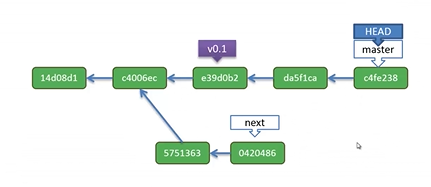
1. 忽略文件
2. .gitignore
   * 1. 在添加是忽略匹配的文件
     2. 仅作用于未跟踪的文件
     3. GitHub有一个官方仓库gitignore存储常见的配置

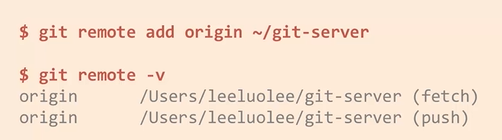


1. git -rm——从暂存区删除
   1. git rm --cached：仅从暂存区删除
   2. git rm：从暂存区与工作目录删除
   3. git rm $(git ls -files --deleted)：删除所有未被跟踪，但是在工作目录被删除的文件（小技巧）
2. git commit——提交
   1. git commit -m “xxx”：描述信息
   2. git commit -a -m “xxx”：直接提交
   3. git log——显示提交历史信息
      1. git log --oneline——简要信息
      2. git log --color --graph --pretty=format: ’%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset’ --abbrev-commit
      3. git中的alias命令——别名
         1. git config alias.shortname <fullcommand>
         2. git config –global alias.lg “log --color --graph --pretty=format: ’%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset’ --abbrev-commit”
         3. 也是配置在~/.gitconfig
3. git diff——显示不同版本的差异
   1. git diff
      1. 工作目录与暂存区的差异
   2. git diff -cached [<reference>]
      1. 暂存区与某次提交的差异，默认为HEAD
   3. git diff <reference>
      1. 工作目录与某次提交的差异
4. 撤销修改
   1. git checkout -- <file>——撤销本地修改
      1. 将文件内容从暂存区复制到工作目录
   2. git reset HEAD <file>——撤销暂存区内容
      1. 将文件内容从上次提交复制到暂存区
   3. git checkout HEAD -- <file>——撤销全部改动
      1. 将内容从上次提交复制到工作目录
   4. 总结
5. 分支操作
   1. git branch——分支的增删改查都靠它
      1. git branch <branchName>——创建分支
      2. git branch -d <branchName>——删除分支
      3. git branch -v——显示所有存在的分支信息
   2. git checkout——通过移动HEAD检出版本，可用于分支切换
      * 1. git checkout <branchName>——切换目标分支
        2. git checkout -b <branchName>——创建并切换到分支
        3. git checkout <reference>——切换到任何引用对象上
        4. git checkout - ——恢复到上一个分支
           1. detached head

HEAD指向非分支

* 1. git reset——将当前分支回退到历史某个版本
     1. get reset --mixed <commit>(默认)
     2. git reset --soft <commit>
     3. git reset --hard <commit>
     4. git reflog——撤回到上一个提交
     5. 使用捷径
        1. A^：回到上一个提交
        2. A~n：回到上n个提交

1. git stash——保存目前的工作目录和暂存区状态，并返回到干净的工作空间
   1. git stash save “xxx”：保存，加上描述信息
   2. git stash list：列出所有记录
   3. git stash apply stash@{x}：恢复到工作目录
   4. git stash drop stash@{x}：删除某条记录
      1. stash pop = stash apply + stash drop
2. git merge——合并分支
   1. 例git merge next master
   2. git cat-file -p HEAD：显示详细信息
   3. git merge fast-forward
   4. git merge next --no-ff
3. git rebase——修改提交历史基线，俗称“变基”
   1. git rebase master
   2. git rebase --onto master 5751363
   3. 勿在共有分支使用rebase
4. git tag——对某个提交设置一个不变的别名
   1. git tag v0.1 e39d0b2
5. 远程操作
   1. Git支持本地协议，所以我们可以初始化一个本地的远程服务器…
      1. git init ~/git-server --bare



* 1. git push——提交本地历史到远程
  2. git remote——远程仓库配置操作
     1. git remote add origin url——添加一个远程仓库别名
     2. git remote -v——查看远程仓库信息
     3. git push origin master
  3. git fetch——获取远程仓库的提交历史
     1. git fetch origin master
     2. git fetch + merge——解决冲突
  4. git pull
     1. = git fetch + git merge

1. git clone——克隆一个远程仓库
   1. = git init + git remote + git pull
   2. git clone url <name>

## 其他参考资料

1. git – 简明指南
   1. 助你入门git的简明指南，没有高深内容
   2. <http://rogerdudler.github.io/git-guide/index.zh.html>
2. Try.github.com
   1. 15分钟的教程
3. Book
   1. Pro Git
      1. 最完整的Git指南
      2. <http://git.scm/book/zh/v1>