

Git/Repo/Gerrit 工具

主题

- † 版本控制系统
- † git 介绍
- † repo 介绍
- † Gerrit 介绍
- † Jenkins 介绍

版本控制系统

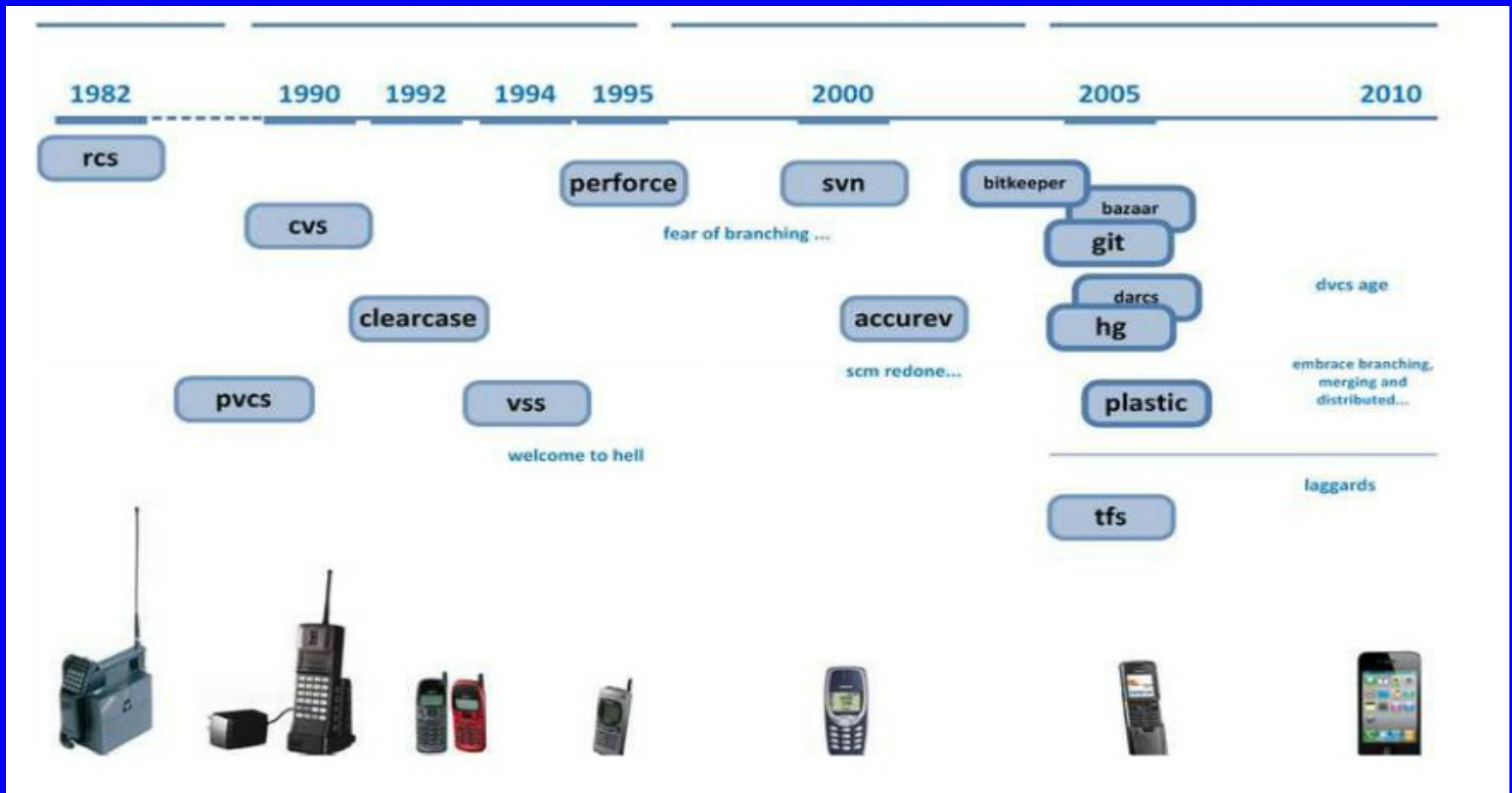
✦ 什么是版本控制系统？

- 是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统

✦ 常见的版本控制系统有哪些？

- 本地版本控制系统 (RCS)
- 集中式版本控制系统 (CVS、SVN、Perforce)
- 分布式版本控制系统 (Git、Mercurial、Bazaar、Darcs、BitKeeper)

版本控制系统 (VCS) 的发展史



Git 诞生

- † Git 有着显赫的身世：Git 是 Linux 之父 Linus Torvalds 的伟大作品。在 Linux 早期，Linus 顶着开源社区一大波精英们的口诛笔伐，选择的是一个商业版本控制系统：BitKeeper 作为 Linux 内核的代码管理工具。
- † 转折：2005 年一件大事导致了 git 的诞生。Samba 的作者（澳大利亚人 Andrew Tridgell）试图对 BitKeeper 反向工程，希望可以开发出一个能与 BitKeeper 交互的开源工具。于是商业软件公司决定收回对 Linux 开源社区免费使用 BitKeeper 的授权。
- † 于是 Linus 盛怒之下开发出了分布式版本控制系统：git。
 - 问题：git 的源码使用什么来管理源码的呢？

Git 优点

- † 开源：可以自由使用，无需授权、无专利费用
- † 非线性开发：支持多个并行开发分支
- † 完全分布式：既是客户端也是服务端
- † 离线、速度快：本地和远程独立操作，可以后期再同步
- † 兼容各种协议：git、ssh、http 等
- † 时刻保证数据的完整性：所有数据都要进行内容计算和校验，并将结果作为数据的唯一标识和索引
- † 有能力高效管理类似 Linux 内核一样的超大规模的项目（速度和数据量）
- † git 源码：<https://github.com/git/git>

Git 的安装和配置

+ Git 的安装 (ubuntu)

- `sudo apt-get install git`

+ 初次运行 git 前的配置

- `git config --global user.name "Your Name"`
- `git config --global user.email "Your Email"`
- 配置范例

```
[user]
  name = tangjj1112
  email = tangjj1112@thundersoft.com
[color]
  ui = auto
[core]
  editor = vim
```

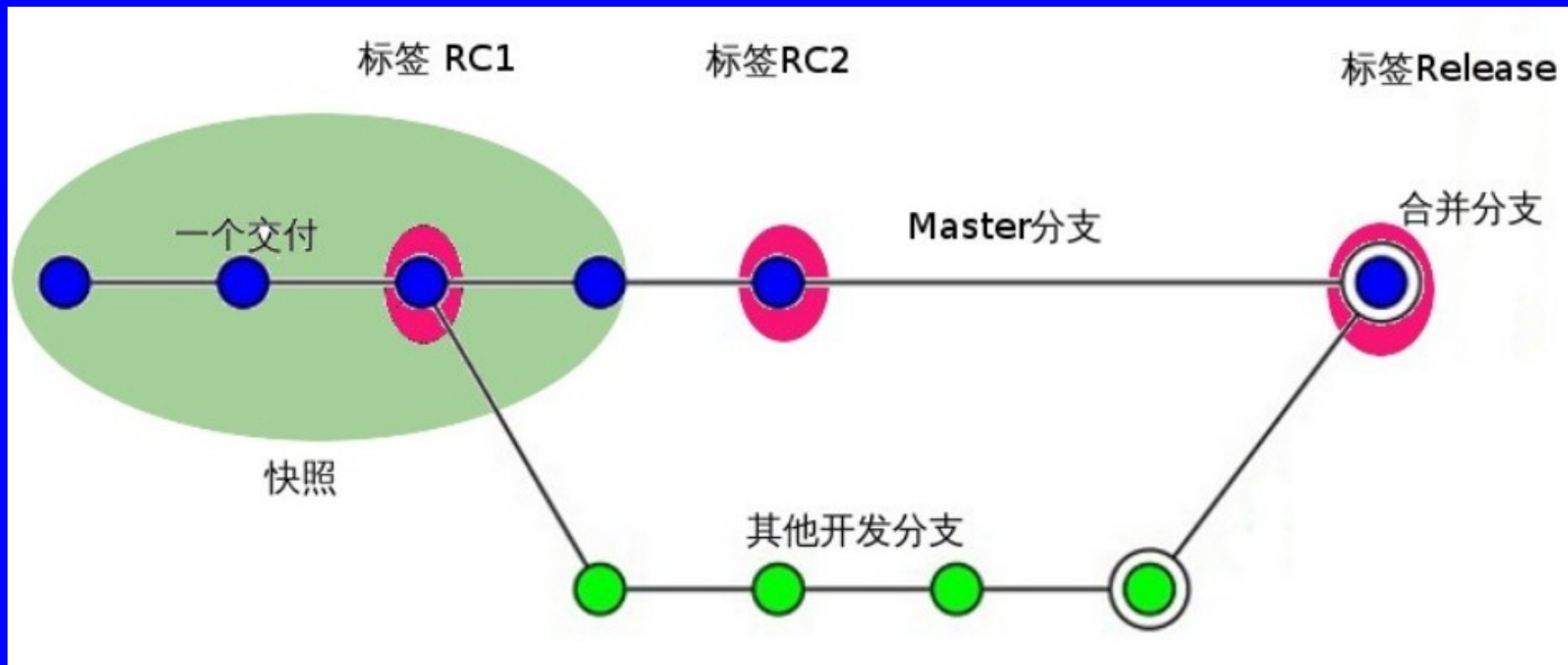
使用 git 管理项目

- † `cd project-directory`
- † `git init`
- † `git add .`
- † `git commit -s -m " 项目简介 "`

Git 仓库简介

- † Git 仓库管理文件：.git
 - config: 仓库特定的配置
 - description: 仓库的描述
- † 工作目录：除 .git 之外的内容
- † 交付：某个存入到 .git 管理的修改；有全球唯一 id:SHA-1
 - git commit
- † 分支：主分支和其他分支，开发过程中的不同并行任务
 - git branch
- † 标签：某个具有里程碑意义的交付
 - git tag
- † 快照：某个交付之前的所有历史修改

Git 仓库图示

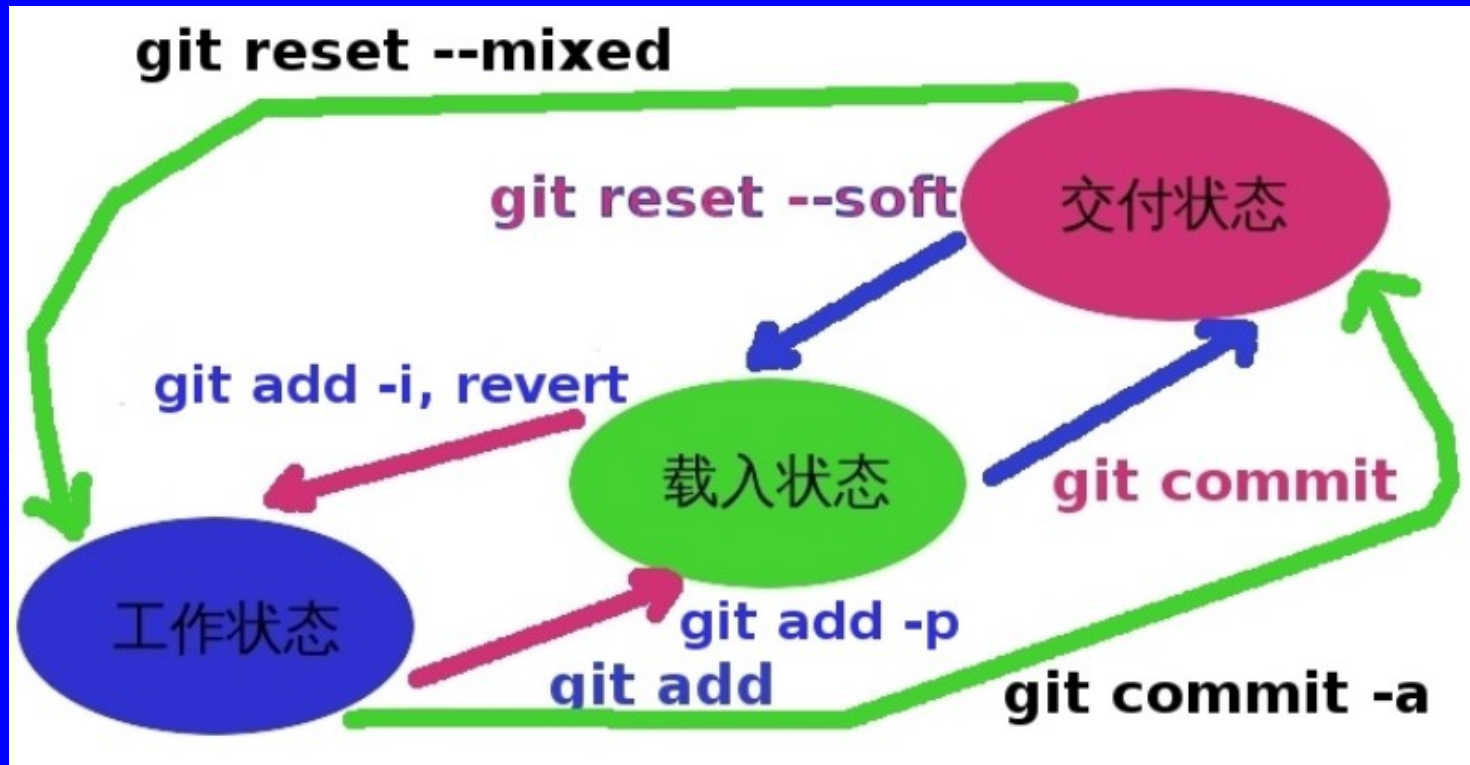


Git 基本工作流程

- ✦ 创建或切换工作目录
 - `git checkout [branch|tag|commit]`
- ✦ 日常工作 :working
 - 日常目录和文件操作
- ✦ 载入某些工作 :staged/cached
 - 添加 : `git add`
 - 删除 : `git rm`
 - 重命名 : `git mv`
- ✦ 交付已经载入的工作 :committed
 - `git commit -s -m " 工作描述 "`

Git 仓库的三种状态以及之间的转换

† 工作状态 (working) + 载入状态 (staged/cached) + 交付状态 (committed)



查看 Git 工作状态

✚ 日常工作

- git status
- git diff

✚ 已经载入的工作 : staged, cached

- git status
- git diff --staged

✚ 已经交付到 Git 仓库的工作

- git diff commit1..commit2
- git show
- git log --graph

Git 纠错机制：撤销或者恢复

✦ 日常工作

- `git checkout – (files)`

✦ 已经载入的工作：staged, cached

- `git rm –cached`

✦ 已经交付到 Git 仓库的工作

- `git revert` 某个交付
- `git reset [–mixed|–soft|–keep|–hard]` 某个快照
- `git rebase -i` 某个历史交付到最新交付

✦ 清理非 Git 管理的文件和目录

- `git clean -dfx` 强制删除 (恢复为干净的仓库)

Git 交付 (commit) 管理

- † 提交 :git commit -s -m " 修改记录 "
- † 撤销 :git revert commit
- † 修订
 - 最新交付 HEAD(.git/HEAD):git commit --amend
 - git rebase -i commit^, pick->edit, reword
- † 合并 :git rebase -i commit ,pick1,pick2->pick1,squash
- † 重排 :git rebase -i commit^, pick1,pick2->pick2,pick1
- † 抽取 :git cherry-pick commit

Git 补丁 (patch) 管理

✚ 生成补丁

- `git format-patch commit1..commit2`
- `git format-patch HEAD^`
- `git format-patch -1 commit`

✚ 应用补丁

- `patch -p1 < (patch file)`
- `git apply (patch file)`
- `git am (邮件格式的 patch)`

Git 分支 (branch) 管理

✚ 查看

- `git branch [-a] 分支名`

✚ 创建

- `git branch 分支名 commit`
- `git checkout -b 分支名 commit`

✚ 删除

- `git branch [-D|-d] 分支名`

✚ 合并

- `git merge 分支名`

Git 标签 (tag) 管理

+ 查看标签

- `git tag`

+ 创建标签

- `git tag -m " 标签描述 " 标签名 commit`

+ 删除标签

- `git tag -d 标签名`

Git 本地仓库和远程仓库交互：下载

✦ 复制仓库

- `git clone ssh://example@host/proj.git localproj`

✦ 复制仓库并切换到指定分支

- `git clone ssh://example@host/proj.git --branch 分支名`

✦ 下载分支

- `git fetch origin 远程分支名`
- `FETCH_HEAD:..git/FETCH_HEAD`
- 可作为分支直接引用：`git merge FETCH_HEAD`
- 下载并创建本地分支：`git fetch origin 远程分支名 : 本地分支名`

✦ 下载分支并合并到当前分支：fetch & merge

- `git pull origin 远程分支名`

Git 本地仓库和远程仓库交互：上传

† 上传分支到远程仓库

- 同名 :git push origin 本地分支名
- 改名 :git push origin 本地分支名 : 远程分支名

† 删除远程分支

- git push origin : 远程分支名

† 标签 (tag) 操作

- 同名 :git push origin 标签名
- 改名 :git push origin 本地标签名 : 远程标签名
- 删除标签 : git push origin : 远程标签名
- Fetch 和 Push: git fetch --tags, git push --tags

Repo 简介

- † 从 Android 诞生以来，其用 git 管理的代码仓库从未少于 400 个。那么如何能方便的管理 Android 项目中这么多源码库呢？一个一个的 git clone ？
- † Google 自然不会允许这么 low 的存在，于是 repo 就诞生了。
- † Repo 本身就是一个 python+bash 的结合体 -- 一个脚本文件，但正是这个脚本可以很方便的管理 AOSP 如此巨大的项目源码。
- † `curl https://android.git.kernel.org/repo > ~/bin/repo`

Repo 常用命令 (1)

† repo status

- 查看整个 worktree 的状态：是否 clean, 哪些库有什么样的变动等

† repo branch[es]

- 查看本地 worktree 上的工作分支情况

† repo abandon <branch_name> {<git 库名> | --all}

- 删除指定分支 (可以是特定 git 库, 也可以是全体), 可视为 repo start 的逆操作

† repo upload [<project> ...] | [--replace <project>]

- 向 Gerrit 提交本地的修改

† repo checkout <branchname> [<project>...]

- 把指定 git 库都切换为某个分支 (如不指定具体 git 库则对所有 git 库)

Repo 常用命令 (2)

- † repo list [<project>...]
 - 列出指定或所有库的关联路径，格式是“目录：库名”
- † repo download {project change[/patchset]}...
 - 从 Gerrit 下载一个或多个 Patch 并反映到工作目录中
- † repo diff [<project>...]
 - 比较当前工作目录中的代码和库中最新代码的区别
- † repo grep {pattern | -e pattern} [<project>...]
 - 在指定或所有库中查找关键字
- † repo forall [<project> ...] -c ‘命令’
 - 对指定定 git 库执行“命令”（如不指定具体 git 库则对所有 git 库）

项目中真实的 repo

- † `repo init -u example@host:29418/manifest.git -b example`
- † `repo sync`
- † `repo start local-branch <project|--all>`
- † `cd your_project_dir`
- † `# ...edit...edit...edit...`
- † `git status`
- † `git add <changed file1> <changed file2>...`
- † `git status`
- † `git commit`
- † `repo upload .`

Gerrit 简介

- † Gerrit 是 Google 开发的一个代码审核工具。
- † 它是一个 Web 工具，它靠 git 来存放代码，靠 repo 这个接口来提交和下载修改。
- † 提交到 Gerrit 时，每个 Git 库的修改都会变成一次提交，每个提交可以有一个或多个人来 review 和 verify。
- † 当你的修改被批准之后，Gerrit 会把修改真正提交到指定的分支中。

代码审核系统 Gerrit

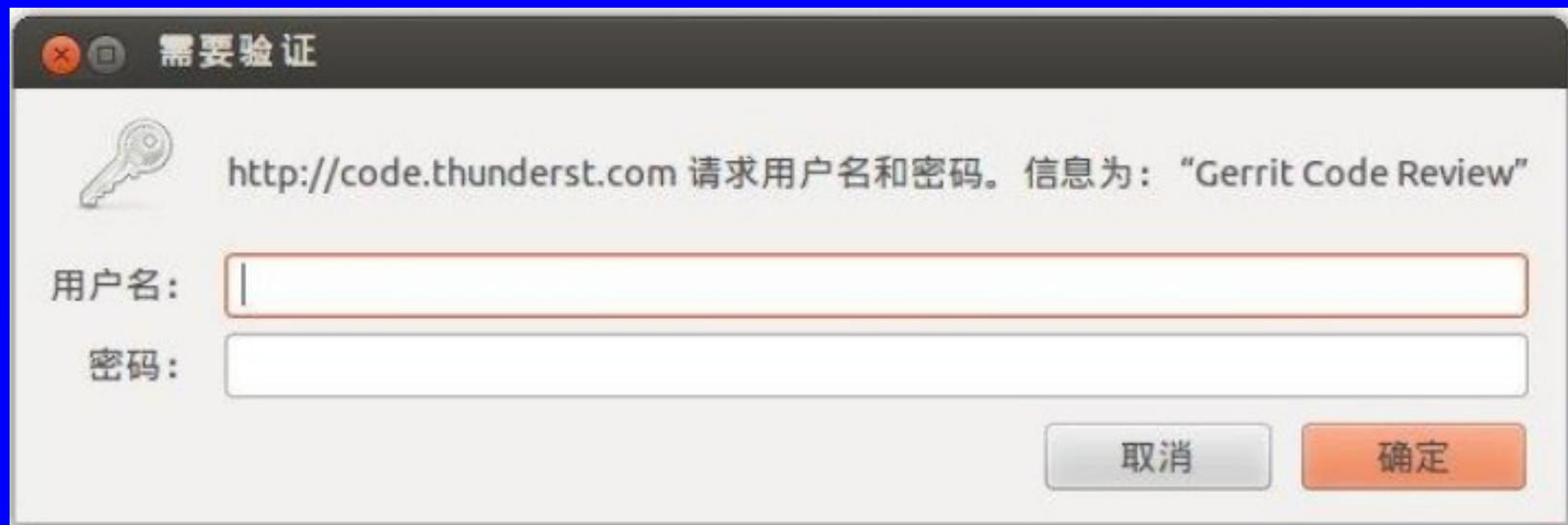
- † 谷歌的又一个重要创新
- † 利用 git 的 branch 和 merge 机制
- † 具有强制性
- † 除非特别的授权否则不能入库
 - 向 Git 版本库的推送 (Push) 必须要经过 Gerrit 服务器
 - 修订必须经过代码审核流程才可以经批准纳入正式代码中

代码审核系统的工作流程

- † 开发者提交代码至 Gerrit 服务器
- † Code Review
 - 检查代码，确定是否合理
- † Verify
 - 应用代码，确认是否能正确编译、执行
- † Submit
 - 与已发布版本合并（入库）
- † Rollback
 - 以上每个步骤都可能失败，并进行相应的流程回退


项目中真实的 Gerrit- 登陆

- † Gerrit 有账户管理，需要输入正确的用户名和密码才可以登录
- † 不同项目的 Gerrit 可能是不同的，需要向项目 PM 申请账户和权限
- † 帐号一般都会预先创建好



A screenshot of a Windows-style dialog box titled "需要验证" (Need Verification). The dialog box has a key icon on the left. The main text reads: "http://code.thunderst.com 请求用户名和密码。信息为: 'Gerrit Code Review'". Below this text are two input fields: "用户名:" (Username) and "密码:" (Password). At the bottom right, there are two buttons: "取消" (Cancel) and "确定" (OK).

需要验证

 http://code.thunderst.com 请求用户名和密码。信息为: "Gerrit Code Review"

用户名:

密码:

取消 确定

项目中真实的 Gerrit-SSH key

- † 在自己电脑上执行 `ssh-keygen`
- † `cat ~/.ssh/id_rsa.pub`, 将其内容粘贴到 Gerrit settings 的 SSH Public Keys 中, 通过 ssh 协议来同步 / 提交代码



项目中真实的 Gerrit- 权限

- ✦ 在项目初始，除了向 PM 申请帐号还需要申请 Gerrit 权限
- ✦ Gerrit 权限说明 (一个人不能同时拥有 super_review、verify、submit 三种权限)

Project admin	该项目管理员，可以为用户添加 Gerrit 身份；
Project_read	拥有 gerrit 用户名密码，设置了全名和邮箱的用户；拥有下载代码，提交代码，Code Review +1 的权限；
Project_super_review	负责查看该项目代码 Review，拥有 Code Review +2 的权限；
Project_verify	负责该项目编译验证代码，拥有 Verified +1 的权限
Project_submit	当该次提交代码 Code Review +2 并且 Verified +1 后，负责将代码 Merge 到代码库中

项目中真实的 Gerrit- 代码提交状态

† Gerrit 上代码提交的三种状态

- Open、Merged、Abandoned
- Open 状态的代码需要经过 Review, Verify, Submit 操作后才会真正入库，即成为 Merged 状态
- Merged 状态的代码已经入库，不能再 Abandoned, 只能 Revert.
- Open 状态的代码由于各种原因不能入库的可以放弃，即 Abandoned 状态。
- Abandoned 状态的代码不能再入库，如有需要，可以“Restore”。

项目中真实的 Gerrit- 提交详细信息

- ✦ 代码提交之后，登录 Gerrit, 可以看到此次提交已经显示在列表中
- ✦ 绿箭头所指是目前查看的代码状态，即 Open 状态的列表。

All

My

Projects

Groups

Plugins

Documentation

Open

Merged

Abandoned

status:open

Search for status:open

ID	Subject	Owner	Project
<div> <div></div> <div>I5e084ef2</div> </div>	[R17424]Porting framework wifi change [LC]		fujitsu/platform/frameworks/base
<div> <div></div> <div>I703e75ce</div> </div>	[R17360] Porting feature "CellBroadcast" [Tele]		fujitsu/platform/packages/apps/CellBroadcastReceiver

项目中真实的 Gerrit- 提交详细信息

ID	Subject	Owner	Project	Branch	Updated	CR	V
Change Id	提交信息的第一行	提交人	库名	分支名	最后变更的时间	Review 的状态	Verify 的状态

- ✚ CR 的状态有四种，分别为 ✕ (-2), -1, +1, √ (+2)
 - 其中 -1 并不影响入库，但 -2 则需要重新提交新的 Patch Set, 或者设置 -2 的成员批准才能入库。
- ✚ V 的状态有两种，分别是 ✕ (-1), √ (+1)
 - 其中 -1 的状态不能入库，需要重新编译验证 (Verify) 通过才可以入库。

项目中真实的 Gerrit- 提交详细信息

息

- † 两个 Patch Set 表示该提交更新过一次修改
- † 可以查看修改的文件
- † 文件修改的对比版本可以自由选择是与 Base 或是与某一个 Patch Set 作对比
- † Comments 信息是 Review 代码时填写的信息

Dependencies

Old Version History: Base

Patch Set 1 34bf497d3d4d495cc0dc2068c981815756a76493

Patch Set 2 7e15f72e39468293c6466bf5971dfec4803eb63

Author: [redacted]@thundersoft.com> May 9, 2013 9:17 PM

Committer: [redacted]@thundersoft.com> May 9, 2013 9:58 PM

Parent(s): d9a314cc42ddfa19820e5ebb57a93382e511a199 branch out packages/apps/Phone

Download: [repo download](#) [checkout](#) [pull](#) [cherry-pick](#) [patch](#)

[Review](#) [Abandon Change](#)

File Path [Commit Message](#)

M default.xml

Comments

huan Patch Set 1: I would prefer the

huan Patch Set 1: path should inclu

huan Uploaded patch set 2.

huan Patch Set 2: Verified; Looks good to me, approved

huan

Change has been successfully merged into the git repository.

May 9 9:58 PM

May 9 10:18 PM

May 9 10:19 PM

项目中真实的 Gerrit- 代码 review

- + 可以邀请 reviewer
- + 点击 Review 按钮可以对当前提交做 Review 的操作
- + 不同权限，可以看到左右不同的页面
- + 信息框中可以填写此次 Review 的更详细的信息

The image displays two side-by-side screenshots of the Gerrit code review interface, illustrating different user permissions or settings.

Left Screenshot:

- Reviewer Table:** Shows 'jenkins' with a score of -1 and 'scm' with a score of -1. Both have 'Verified' checkboxes marked with an 'X'.
- Buttons:** 'Need Verified' and 'Need Code-Review' are both selected (indicated by dots).
- Code Review:** The '0 No score' option is selected.
- Cover Message:** A large text area for the review message.
- Bottom:** The 'Publish Comments' button is circled in red.

Right Screenshot:

- Reviewer Table:** Identical to the left screenshot.
- Buttons:** Only 'Need Code-Review' is selected.
- Code Review:** The '0 No score' option is selected.
- Cover Message:** A large text area for the review message.
- Bottom:** The 'Publish Comments' button is circled in red.

项目中真实的 Gerrit- 代码 merge

✚ 代码经过 Review +2 和 Verify +1 后就可以 Submit 入库

Reviewer: [huajie](#) [X] [✓] [✓] [Green Arrow]

Name or Email or Group Add Reviewer

► Dependencies

Old Version History: Base ▼

► Patch Set 1 e7a1d761fdddcacd6591db5e44dd9207c4c6f8d5

▼ Patch Set 2 38f6c02fa60b9cf1b3db1169071091ec7353f0fc

Author: [xingchun@thundersoft.com](#)> May 9, 2013 9:13 PM

Committer: [xingchun@thundersoft.com](#)> May 9, 2013 10:02 PM

Parent(s): b5e31e3d31f8809482c76880a7e96038d1f7f389 branch out packages/apps/Phone

Download: [repo download](#) | [checkout](#) | [pull](#) | [cherry-pick](#) | [patch](#) | [SSH](#) | [HTTP](#)

git fetch ssh://huajie@192.168.8.186:29418/fujitsu/manifest refs/changes/03/503/2 && git cherry-pick FETCH_HEAD

Review **Submit Patch Set 2** Abandon Change Rebase Change Diff All Side-by-Side Diff All Unified

	File Path	Comments	Size	Diff	Reviewed
►	Commit Message			Side-by-Side Unified	
M	default.xml		+30, -30	Side-by-Side Unified	✓
			+30, -30		

一些不很常用但很有用的命令

† 忽略某些文件

■ .gitignore 文件

- † # 此为注释— 将被 Git 忽略
- † *.so # 忽略所有 .so 结尾的文件
- † *.[oa] # 忽略所有以 .o 或 .a 结尾的文件
- † !lib.a # 但 lib.a 除外
- † /TODO # 仅仅忽略项目根目录下的 TODO 文件, 不包括 subdir/TODO
- † build/ # 忽略 build/ 目录下的所有文件
- † doc/*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

† 生成 git 格式的 patch

■ git format-patch

一些不很常用但很有用的命令

† git log 的一些参数

- --oneline 单行显示 log, 只显示短提交号和标题
- --stat 显示每次更新的文件修改统计信息
- --name-only 仅在提交信息后显示已修改的文件清单
- --name-status 显示新增、修改、删除的文件清单
- -n 只显示最近的 n 次提交
- --author 仅显示指定作者相关的提交
- --committer 仅显示指定提交者相关的提交
- --since, --after 仅显示指定时间之后的提交
- --until, --before 仅显示指定时间之前的提交

一些不很常用但很有用的命令

✚ 一些关于远程仓库的命令

- `git remote [-v]` 查看当前配置的远程仓库
- `git remote show [remote-name]` 查看远程仓库信息
- `git ls-remote [remote-name]` 查看远程仓库中各分支最新更新
- `git remote add <name> < 远程仓库 URL>` 添加远程仓库
- `git remote rm <name>`
- `git remote rename <old> <new>` 远程仓库重命名
- `git push [--tags] [remote-name] [本地分支 : 远程分支]`
 - 推送代码到远程库的指定分支

一些不很常用但很有用的命令

- † `git reflog/git log -g`
 - 可以查看 git 的引用日志
 - 可以通过这些记录找回丢失的提交
- † `git add -i`
 - 交互式暂存
 - 在修改文件多且不希望一次都提交的时候很有用
- † `git blame`
 - 查看文件每行的 git 信息
 - -C 参数可以找出文件的拷贝来源

谢谢大家