Android Repo 源码管理系统

风灵创景科技有限公司 2010 年 11 月

简介

* 基于 **Git** 构建
* 管理多个 **Git** 库
* 自动化版本管理工作流程
* 提供一致的分支操作
* 基于 **Python** 实现

原理

组件

* **repo** 脚本　　**repo** 入口，基础功能实现
  - 存放于用户可执行目录
* **.repo** 目录　　**repo** 实现及管理目录
  - 存放于 **Android** 项目源码根目录
    * **manifest**　　项目描述文件库
    * **repo**　　**repo** 实现
    * **projects**　　**repo** 所管理的 **git** 库

组件 – Manifest

* Android 源码管理的核心
* 存放 Android 项目描述文件
* 有多个分支，每个分支可能多个描述文件
* manifest.xml 指定其中一个描述文件
* 包含 manifest.git, manifest 目录 , manifest.xml
  * manifest/.git ----> manifest.git
  * manifest.xml ----> manifest/default.xml (或其他 xml)

组件 – Manifest

XML 文件结构

```
<remote  name="korg"
        fetch="git://git.source.dianxinos.com/android2.2/"
        review="git.source.dianxinos.com:1234" />
```

远程库相关配置

name: 远程库名，任意指定

fetch: 远程库路径 ，指定远程库的根目录

review:gerrit 路径，用于连接 **review**

组件 – Manifest

XML 文件结构

```
<default revision="froyo-dev"
        remote="korg" />
```

默认 checkout 版本配置

revision: 默认 checkout 版本，分支或 Tag

----revision 会引起在所有 git 库中创建名为 m/revision 的分支，指向这里指定的版本。

remote: 默认 checkout 库名，一般跟 remote/name 相同即可

组件 – Manifest

XML 文件结构

```
<project path="packages/wallpapers/MusicVisualization"
  name="platform/packages/wallpapers/MusicVisualization"
  revision="863cb2e0b8f3e07e1143a9910045c1735ddfb5ab"
  />
```

Git project 定义

path: 本地 checkout 相对路径

name: 在远程库中相对路径

revision: 默认 checkout 版本，该属性会覆盖 default 中的 revision ， Git 库中创建的 m/revision 依然以 default 中的 revision name 命名，但指向这里定义的 revision

组件 – repo 实现 /projects

repo 实现

* **python** 源码开发，可直接修改
* **subcmd** 实现在 **subcmd** 目录

projects

* 被各 **Git** 库的 **.git** 目录所指向，集中管理
* 布局与 **checkout** 出的 **project** 一致

主要操作

* **init** 初始化
* **sync** 同步代码
* **status/diff** 查看修改状态 / 查看详细修改
* **branches** 查看分支状态
* **start/checkout/abandon/prune** 创建 / 切换 / 删除 / 整理分支
* **forall** 对所有 **project** 执行统一操作

主要操作 -init

repo init 完成 **Android** 项目初始化。

-u URL: 指定远程 **manifest** 库路径。

-b: 指定 **manifest** 库中的分支名。

-m: 指定 **manifest** 库中 **xml** 文件。默认为 **default.xml** 。

初始化完成：

* 确定 **manifest.xml** ，并下载 **manifest** 库
* 下载 **repo** 实现 源码

主要操作 -sync

repo sync 完成 **Android** 项目代码下载，更新及自动 **checkout**

-j: 并行下载

[project name]: 单独 **sync** 某个 **project** ，需要填写本地相对路径

* 优先根据 **manifest** 中 **project revision** 确定版本，其次是 **default revision**

* 默认 **checkout** 出的版本，都在 **(no branch)**

* 需要保证本地工作目录干净再做 **sync** ，否则会失败

主要操作 -sync

特别注意：

repo sync 会尽一切办法将本地正在使用的 **branch** 更新到指定的版本。

* **repo sync** 首先会更新本地库 **(git fetch)**

* 如果当前分支在 no branch ，或未 track 指定 revision（非 repo start 创建）

    * 自动切换到 no branch ，并更新到指定 revision

* 如果当前分支 track 指定 revision（repo start 创建）

    * 若该分支无改动，直接 fast-forward 到指定 revision

    * 若该分支有改动，自动跟指定 revision 进行 merge

主要操作 -status/diff

* **repo status** 查看当前源码修改状态 (封装 **git status)**

列出信息：

- 修改项目
- 修改文件
- 当前分支
- 与 **m/revision** 差异

* **repo diff** 查看当前源码详细修改状态 (封装 **git diff)**

列出信息：

- 修改项目
- 文件详细修改

主要操作 - 分支管理

* **repo branches** 查看当前项目分支情况

当前若无任何分支 **(Clean)** 状态，返回 **no branch**

否则返回当前分支状态 **(** 分支 **----** 项目 **)**

* **repo start** 新建分支。分支自动 track manifest 中指定 revision

* **repo checkout** 切换另一分支

* **repo prune** 进行分支整理。自动删除提交回远程库的分支，保留其他。

* **repo abandon** 强制删除指定的分支

主要操作 - 分支管理

定义状态：

Clean: repo status 查看没有修改， **repo branches** 返回 **no branch** 。推荐在此状态进行 **repo sync** 。

Dirty: repo status 查看有修改， **repo branches** 返回 **no branch** 。 此状态应避免出现，修改应先建立开发分支再修改。

Clean-Dev: repo status 查看无修改， **repo branches** 返回目前分支状态。 开发状态，所有修改已提交到本地，可能已经提交到远程库。

Dirty-Dev: repo status 查看有修改， **repo branches** 返回目前分支状态。 开发进行状态，正在进行修改。

主要操作 - 分支管理

简单开发流程：

Clean → Clean-Dev → Dirty-Dev → Clean-Dev → Clean

1. 在 **Clean** 状态进行 **repo sync** ，更新代码

2. Clean → Clean-Dev, 通过 **repo start** 创建开发分支

3. 进行开发，导致 **Clean-Dev → Dirty-Dev**

4. 将开发结果进行本地提交，恢复 **Clean-Dev**

5. 确保远程提交后，通过 **repo prune** 尝试恢复 **Clean** 状态。如果失败，检查是否有改动需要提交到远程，提交之；否则通过 **repo abandon** 强制恢复 **Clean** 状态。如果失败，考虑手工清理不需要的分支，恢复 **Clean** 状态。

该操作流程可以保证基于 **repo** 开发简单可靠进行，仅供参考。

主要操作 - 分支管理

技巧：

* **Clean** 状态可以确保 **repo sync** 成功完成。熟练后，完成远程提交的情况下，在 **Clean-Dev** 也可以使用 **repo sync** 。

* **repo prune** 能够自动清理已经提交回远程库的分支。一般请在 **Clean-Dev** 状态使用该命令，可以快速找出自己残留未提交改动的分支。

* **repo abandon** 同样在 **Clean-Dev** 下使用，强制清除指定分支，不管是否已经提交回远程库。

主要操作 -forall

* **repo forall** 对所有项目执行同一操作，可以灵活使用

repo forall [<project>...] -c <command> [<arg>...]

command 请使用单引号包含

可用环境变量：

- **pwd** 指向当前工作目录
- **REPO_PROJECT** 是当前 **project** 在服务器库中的相对路径 (也称 **project name**)
- **REPO_PATH** 是当前 **project** 本地工作目录的相对路径
- **REPO_REMOTE** 是当前 **project** 的库名
- **REPO_LREV** 是当前 **project revision** 版本的 **SHA1**
- **REPO_RREV** 是当前 **project** 的 **revision**