



Hello, The Git World

吴章金

wuzhangjin@gmail.com

Tiny Lab – Embedded Geeks
<http://tinylab.org>

2013年3月1日



+++ git



内容概要

- 1 Git简介
- 2 基本用法
- 3 SVN转Git
- 4 协同开发
- 5 在线浏览
- 6 相关技巧
- 7 相关资料



快速入门

以Linux内核开发为例：

► 开发人员

```
$ git clone git://git.kernel.org/.../linux-2.6.git
$ cd linux-2.6
  (edit files)
$ git add (files)
$ git commit -s -m "modification_description"
  (build, boot and the other necessary test)
$ git format-patch origin/master
$ git send-email --to="Maintainer's_email"
  --cc="mailing_list" (patch files)
```

► 维护人员

```
$ git am (patch files)
```



Git+Linux二三事

- ▶ 1991-2002: tar, diff & patch
- ▶ 2002-2005: Proprietary Bitkeeper
- ▶ 2005-: Linus开始写Git
 - April 3: 启动
 - June 16: Linux 2.6.12
- ▶ Git主要特点
 - 开源：可以自由使用，无专利限制
 - 非线性开发：支持多个并行开发分支
 - 完全分布式：Client也是Server，默认备份
 - 离线、速度快：本地跟远程操作独立，可以到后期再同步
 - 兼容各种协议：git, ssh, rsync, http, ftp



Git安装与配置

- ▶ 下载 : <http://git-scm.com/download>
- ▶ 安装: msysgit or cygwin+git(Windows), git(Linux), git-email
- ▶ 配置 : ~/.gitconfig
 - git config --global user.name "Your Name"
 - git config --global user.email "Your Email"
- ▶ 配置范例

```
[user]
  name = Wu Zhangjin
  email = wuzhangjin@gmail.com
  editor = vim

[core]
  pager = less -FRSX

[color]
  ui = auto

[merge]
  tool = vimdiff

[sendemail]
  confirm = auto
  smtpserver = smtp.gmail.com
  smtpencryption = tls
  smtpuser = wuzhangjin
  smtpserverport = 587
  suppresscc = all
```



+++ git



把项目导入Git管理

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



+++ git



Git仓库简介

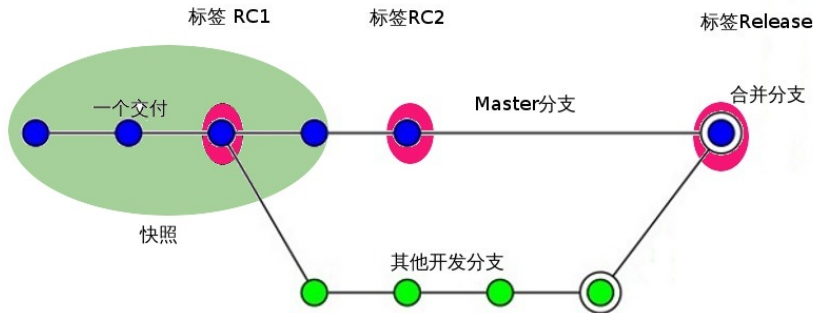
- ▶ Git仓库管理文件: .git
 - config: 仓库特定的配置
 - description: 仓库的描述
- ▶ 工作目录: 除.git之外的内容
- ▶ 交付: 某个存入到.git管理的修改; 有全球唯一id: SHA-1
 - git commit
- ▶ 分支: 主分支和其他分支, 开发过程中的不同并行任务
 - git branch
- ▶ 标签: 某个具有里程碑意义的交付
 - git tag
- ▶ 快照: 某个交付之前的所有历史修改



+++ git



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)

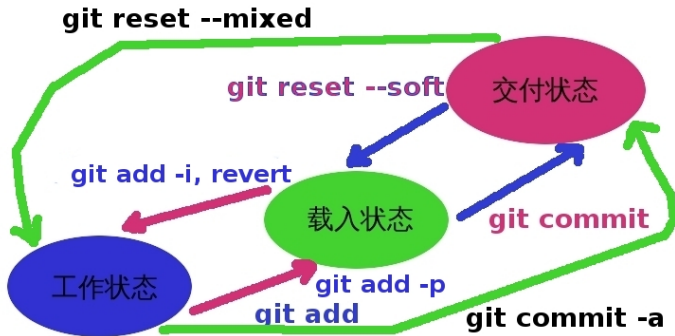


Figure: Git仓库的状态转换



+++ git



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 diff "@{1 second}"
 - git show
 - git whatchanged
 - git log
 - git rev-list



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 add -i, revert
 - 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 -d
 - git clean -x
 - git clean -X



Git交付(commit)管理

- ▶ 提交 : `git commit -s -m "修改记录"`
- ▶ 撤销 : `git revert commit`
- ▶ 删除 : `git rebase -i commit^, pick1,pick2->pick2`
- ▶ 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`
- ▶ 把补丁作为邮件发送出去
 - `git send-email (patch files)`
- ▶ 应用补丁
 - `patch -p1 < (patch file)`
 - `git apply (patch file)`
 - `git am` (邮件格式的patch)
- ▶ 外置补丁管理工具
 - `guilt`



+++ git



Git分支(branch)管理

- ▶ 查看
 - `git branch [-a|分支名]`
- ▶ 创建
 - `git branch 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`



SVN转Git

▶ 从SVN仓库Clone并创建Git仓库

```
mkdir git-repo && cd git-repo
git svn init svn://172.16.11.122/svn_src/meizu_mx_master/linux-2.6.35-mx-rtm --no-metadata
git config svn.authorsfile /path/to/authorslist.txt
git svn fetch
git gc --prune=yes
```

▶ 从SVN仓库自动产生一份authorlist

```
#!/bin/bash
authors=$(svn log -q | grep -e '^r' | awk 'BEGIN {FS = "|"}; {print $2}' | sort | uniq)
for author in ${authors}; do
    echo "${author}_=_NAME_<USER@DOMAIN>"
done
```

▶ 同步后续更新

```
git svn fetch
git rebase --onto git-svn --root
OR
git reset --hard remotes/git-svn
OR
git merge git-svn
```



+++ git



SVN转Git

- ▶ 忽略一些无关文件
 - 诸如.o, .cmd等等
 - 在相关文件和目录下添加.gitignore文件
- ▶ 提交修改到SVN仓库
 - `git svn dcommit`



SVN转Git

SVN v.s. Git

- ▶ 状态
 - `svn status <-> git status`
- ▶ 添加
 - `svn add <-> git add`
- ▶ 交付
 - `svn commit <-> git commit (本地) + git push (远程)`
- ▶ 更新
 - `svn update <-> git pull = git fetch + git merge`
- ▶ 历史
 - `svn log <-> git log`
- ▶ 变更
 - `svn diff -r rev1:rev2 <-> git diff commit1:commit2`
 - `svn diff -c rev1 <-> git show commit1`



创建多用户Git仓库

- ▶ `mkdir proj.git && cd proj.git`
- ▶ `git init --bare --shared`
 - `--bare`: 不包含工作目录(只有.git)
 - `--shared`: 多用户共享
- ▶ 创建用户组
 - `adduser --system --shell /bin/sh --gecos 'Git Version Control' --group --disabled-password --home /home/git git`
- ▶ 修改git仓库所属组
 - `chgrp -R git proj.git`
- ▶ 添加用户到git组
 - `usermod -a -G git 新用户`



多用户访问控制: Git协议

- ▶ 安装git-daemon-run
- ▶ 修改可访问的Git仓库路径
 - /etc/service/git-daemon/run
 - 默认/var/cache/git, 基准目录/var/cache
- ▶ 允许仓库可访问
 - cp -r proj.git /var/cache/git/
 - touch /var/cache/git/proj.git/git-daemon-export-ok
- ▶ 访问该仓库
 - git clone git://localhost/git/proj.git



+++ git



多用户访问控制：ssh与其他

▶ ssh协议

- `ssh://user@localhost/path/to/proj.git`
- 无密码访问：`ssh-keygen`产生密钥，上传公钥到服务器

▶ 其他辅助工具

- gitosis
- gitolite



Git远程仓库管理

- ▶ 添加
 - `git remote add origin git://localhost/git/proj.git`
- ▶ 删除
 - `git remote rm origin`
- ▶ 重命名
 - `git remote rename` 旧名字 新名字
- ▶ 显示远程所有分支
 - `git remote show` 仓库名
- ▶ `.git/config`

```
[remote "origin"]
  url = git://localhost/git/proj.git
  fetch = +refs/heads/*:refs/remotes/origin/*
```



+++ git



Git本地仓库和远程仓库交互：下载

- ▶ 复制仓库
 - `git clone git://localhost/git/proj.git myproj`
- ▶ 复制仓库并切换到指定分支
 - `git clone git://localhost/git/proj.git myproj -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



Web浏览（持久）

- ▶ 安装gitweb, apache2和libapache2-mod-perl2
- ▶ 修改仓库描述
 - .git/description
- ▶ 配置仓库路径
 - /etc/gitweb.conf
 - \$projectroot = "/var/cache/git";
- ▶ Web访问路径
 - http://localhost/gitweb/
- ▶ 相关配置
 - /etc/apache2/conf.d/gitweb



Web浏览（临时）

▶ 创建服务

- `cd proj.git && git instaweb -httpd apache2`
- 配置文件：`.git/gitweb/httpd.conf`
- 启动进程：`apache2 -f /path/to/proj.git/.git/gitweb/httpd.conf`

▶ 浏览地址

- `http://localhost:1234`
- 端口见配置文件`listen`所在行



技巧汇总

- ▶ 让Git无视某些文件和目录
 - 把相关文件和目录添加到.gitignore中：例如.rej, .bak
 - 记得把.gitignore也加入到仓库中：git add .gitignore
- ▶ 暂存/恢复当前的工作状态
 - git stash save
 - (做其他工作)
 - git stash pop
- ▶ 查看最近的历史修改
 - git diff HEAD^; git diff HEAD^^; ...
- ▶ 查看某个历史版本的文件
 - git show HEAD^:arch/mips/kernel/cssrc-r4k.c



技巧汇总(cont.)

► 修改后交付：合并Stage和Commit

```
$ git add . && git commit -s -m 'commit log'
$ git commit -a -s -m 'commit log'
```

► 抽取多个交付

```
$ git rev-list --reverse commit1..commit2 | xargs -i git cherry-pick {}
```

► 全局性地更换电子邮件地址

```
$ git filter-branch --commit-filter '
    if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];
    then
        GIT_AUTHOR_NAME="Scott Chacon";
        GIT_AUTHOR_EMAIL="schacon@example.com";
        git commit-tree "$@";
    else
        git commit-tree "$@";
    fi' HEAD
```

► 从所有提交中删除一个文件

```
$ git filter-branch --tree-filter 'rm -f *.rej' HEAD
```



技巧汇总(cont.)

- ▶ 仅提交文件中的部分修改: `git add -p`
- ▶ 清理仓库、加速仓库: `git gc`
 - 压缩文件交付并清理过期的数据
- ▶ 打包某个分支
 - `git archive --format=tar --prefix=proj-0.1/ develop | gzip > proj-0.1.tar.gz`
- ▶ Git图形化工具: `gitk`
- ▶ Git子仓库管理: `git-submodule`
- ▶ 查看文件特定行的修改记录: 为什么会修改这行? 谁改了?

```
$ git blame -L 122,122 arch/mips/kernel/ftrace.c  
538f1952 (Wu Zhangjin 2009-11-20 20:34:32 +0800 122) int ftrace_make_nop(struct module *mod,
```



技巧汇总(cont.)

通过Git bisect二法定位引入Bug的第一个交付。

- ▶ Git bisect本质上是实现了一个二分法的算法
- ▶ 与顺序查找或随机查找相比平均效率会更高
- ▶ 基本用法
 - git bisect start
 - 找出一个有问题交付，标记为bad: git bisect bad commit1
 - 找出一个好的交付标记为good: git bisect good commit2
 - Git bisect自动checkout出中间交付，测试并进行合适标记
 - 重复上述步骤直到提示找出The first bad commit
- ▶ 自动运行
 - 如果问题容易通过脚本重现，那么可自动进行bisect
 - git bisect bad_commit good_commit start
 - git bisect run /path/to/autotest.script



技巧汇总(cont.)

定义别名节省时间

▶ 定义别名

- `git config --global alias.br branch`

▶ 配置记录：`~/.gitconfig`

```
[alias]
br = branch
st = status
cm = commit
co = checkout
pom = push origin master
```

▶ 查看配置

- `git config -l | grep alias`

▶ 用法实例

- `git branch` → `git br`
- `git status` → `git st`
- `git commit` → `git cm`



技巧汇总(cont.)

► Git流程开发模型: git flow

```
$ wget --no-check-certificate -q -O -  
  https://github.com/nvie/gitflow/raw/develop/contrib/gitflow-installer.sh | sudo sh  
$ git flow init  
$ git branch  
* develop  
  master  
$ git flow feature start test  
$ git branch  
  develop  
* feature/test  
  master  
$ git flow feature finish test  
$ git branch  
* develop  
  master
```



Git相关资料

▶ Git @ wikipedia

- [http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))

▶ Git 首页

- <http://git-scm.com/>

▶ Git 文档

- <http://git-scm.com/documentation>

▶ 免费Git仓库托管平台

- <https://git.wiki.kernel.org/index.php/GitHosting>



项目管理相关资料

- ▶ 免费邮件管理系统: mailman, thunderbird
 - <http://www.gnu.org/software/mailman/index.html>
- ▶ Patch管理工具: patchwork
 - <http://ozlabs.org/~jk/projects/patchwork/>
- ▶ Bug跟踪工具: bugzilla
 - <http://www.bugzilla.org/>
- ▶ 开源信息/项目管理工具: Trac, Redmine
 - <http://www.cyberciti.biz/tips/open-source-project-management-software.html>
 - <http://softwareforgoodconstruction.info/55-open-source-replacements-for-informationproject-management-tools/>
- ▶ 免费博客/CMS管理平台: wordpress, xoops, drupal
 - <http://wordpress.com/>
 - <http://www.scriptol.com/cms/list.php>
- ▶ 免费文档管理系统: didiwiki, twiki, MediaWik, DokuWiki
 - http://www.siteground.com/compare_best_wiki.htm