

_Rad 的专栏

不要生气，不要失望，不要欣喜若狂，一切都会过去。

目录视图

摘要视图

RSS 订阅

个人资料



self001

访问：16469次

积分：275分

排名：千里之外

原创：9篇

转载：10篇

译文：0篇

评论：4条

文章搜索

文章分类

C++ (7)

C (3)

Visual C++ (6)

汇编 (0)

随笔 (1)

Linux (2)

Git (4)

设计模式 (1)

文章存档

2012年11月 (1)

2012年10月 (6)

2012年06月 (1)

2012年03月 (7)

2012年02月 (1)

展开

阅读排行

windows下git bash显示! (4993)

Git团队协作使用规范 以 (2360)

Git merge 合并分区详解 (2188)

汉诺塔问题详解 (递归) (1644)

Git下的冲突解决 (851)

Fedora 下安装codeblocl (661)

三种方法实例XP风格按 (473)

有奖征资源，博文分享有内涵

人气博主的资源共享：老罗的Android之旅

微软Azure•英雄会编程大赛题

关注CSDN社区微

信，福利多多

社区问答：叶劲峰 游戏引擎架构

Git团队协作使用规范 以及一些常用命令详解

分类：Git

2012-03-08 15:34

2362人阅读

评论(0)

收藏

举报

git

branch

工作

file

subversion

服务器

目录(?)

[+]

Git 使用规范

特别提醒：

• 使用Git过程中，必须通过创建分支进行开发，坚决禁止在主干分支上直接开发。review的同事有责任检查其他同事是否遵循分支规范。

• 在Git中，默认是不会提交空目录的，如果想提交某个空目录到版本库中，需要在该目录下新建一个.gitignore 的空白文件，就可以提交了

• 【代码回溯注意】把外部文件纳入到自己的 Git 分支来的时候一定要记得是先比对，确认所有修改都是自己修改的，然后再纳入。不然，容易出现代码回溯

• 【代码回溯注意】多人协作时，不要各自在自己的 Git 分支开发，然后发文件合并。正确的方法应该是开一个远程分支，然后一起在远程分支里协作。不然，容易出现代码回溯（即别人的代码被覆盖的情况）

• 【代码回溯注意】每个人提交代码是一定要 git diff 看提交的东西是不是都是自己修改的。如果有不是自己修改的内容，很可能就是代码回溯

• 【代码回溯注意】review 代码的时候如果看到有被删除掉的代码，一定要确实是否是写代码的同事自己删除的。如果不是，很可能就是代码回溯

格式：[分支名称]+message

例如：[www2011072501]前台商品列表按价格排序需求实现。

比如有一个客户留言功能，被我们拆成了很多子块，那么提交注释如下：

git commit -m "[manage2011072501]完成客户留言的添加、修改功能" git commit -m "[manage2011072501]完成客户留言的删除及手动排序功能" git commit -m "[manage2011072501]完成客户留言的回复功能" git commit -m "[manage2011072501]把旧客户留言的功能删除"

分支合并及上线

步骤

克隆代码

创建分支

在分支中开发

review代码

Git 操作

git clone 远程代码

git checkout -b branch_name

无

无

http://blog.csdn.net/self001/article/details/7333096

1/11

c++ auto_ptr智能指针	(253)
vi的使用方法	(248)
windows消息机制[图]	(223)

评论排行	
windows下git bash显示	(3)
WSAEventSelect模型	(0)
WSAAsyncSelect模型 实	(0)
Select模型学习	(0)
简单的客户端和服务流	(0)
大小端详解	(0)
机选彩票代码	(0)
1.Factory（工厂）模式	(0)
windows消息机制[图]	(0)
三种方法实例XP风格按	(0)

推荐文章	
------	--

最新评论	
windows下git bash显示中文	huhuint: 恩恩，很好用
windows下git bash显示中文	霜之咏叹调: 我已经搞懂了，你应该说，只需要加载文件的最后面一行就可以了
windows下git bash显示中文	蜗牛向前冲: @chuck_lu:加在C:\Program Files\Git\etc\git-completi...
windows下git bash显示中文	霜之咏叹调: 1、C:\Program Files\Git\etc\git-completion.bash: ali...
工作后的第一个项目	wangjieest: 正需要做一个基于某某协议的串口控制程序...望指点:1.协议的指令是变长的...要根据头4个字节判断...
第一份工作.一个星期后的感受!	wangjieest: 楼主这是...公司也敢只给这么多?后面肯定会涨滴

第一轮测试	无
添加代码到分支的暂存区	git add somefile
提交代码到分支	git commit -m "本次提交的注释"
切换到主版本	git checkout master
获取远程最新代码	git pull origin master
合并某分支到master分支	git merge branch_name
解决合并时产生的冲突 请参考分支合并时冲突的解决	无
第二轮测试	无
准备上线文档	无
获取远程最新代码	git pull origin master
推送master分支	git push origin master
通知上线	无
没有问题了删除本地分支	git branch -d branch_name

三种状态

对于任何一个文件，在 Git 内都只有三种状态

中文 英文 含义

已提交 committed 已提交表示该文件已经被安全地保存在本地数据库中了

已修改 modified 已修改表示修改了某个文件，但还没有提交保存

已暂存 staged 已暂存表示把已修改的文件放在下次提交时要保存的清单中

目录 用法

git 目录 它是 Git 用来保存元数据和对象数据库的地方。该目录非常重要，每次克隆镜像仓库的时候，实际拷贝的就是这个目录里面的数据。
工作目录 从项目中取出某个版本的所有文件和目录，用以开始后续工作的叫做工作目录。这些文件实际上都是从 git 目录中的压缩对象数据库中提取出来的，接下来就可以在工作目录中对这些文件进行编辑
暂存区域 所谓的暂存区域只不过是简单的文件，一般都放在 git 目录中。有时候人们会把这个文件叫做索引文件，不过标准说法还是叫暂存区域。

基本的 Git 工作流程

- 1、在工作目录中修改某些文件。
- 2、对这些修改了的文件作快照，并保存到暂存区域。
- 3、提交更新，将保存在暂存区域的文件快照转储到 git 目录中。

安装 Git

以下命令安装的为git 1.7.1

sudo apt-get install -y git-core

配置 Git

以下命令为配置 Git 相关信息，以下两项必须要配置，会出现在每次提交的信息里。

git config --global user.name "caowlong" #规定为姓名全拼
git config --global user.email "caowlong163@163.com" #规定为公司邮箱
git config --global merge.tool "meld"
git config --global color.ui true # 使用git默认的配色方案，推荐
git config --global --list # 查看配置信息

```
git config --global user.name # 查看 user.name 的配置信息
```

让 Git 用 meld 比较文件差异

```
gedit ~/git-meld.sh  
#输入下面两行内容保存并退出
```

```
#!/bin/sh  
meld $2 $5
```

执行

```
chmod +x ~/git-meld.sh  
git config --global diff.external ~/git-meld.sh
```

以下为使用示例

```
git diff readme.txt  
git diff --cached readme.txt
```

获取帮助

格式 git help <verb>

示例

```
git help commit # 按 q 退出帮助
```

取得项目的 Git 仓库

有两种取得 Git 项目仓库的方法

在现存的目录下，通过导入所有文件来创建新的 Git 仓库从已有的 Git 仓库克隆出一个新的镜像仓库来

一、从当前目录初始化

要对现有的某个项目开始用 Git 管理，只需到此项目所在的目录，执行

```
cd 某个目录  
git init
```

初始化后，在当前目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。

不过目前，仅仅是按照既有的结构框架初始化好了里边所有的文件和目录，但我们还没有开始跟踪管理项目中的任何一个文件

命令	含义
git add *.php	把所有的php文件放入暂存区
git add readme.txt	把名为readme.txt的文件放入暂存区
git add dir/	把名为dir的目录里的所有文件放入暂存区
git add *	把当前目录的所有文件都放入暂存区
git rm --cached dir/ -r	把名为dir的目录里的所有文件取消暂存
git rm --cached readme.txt	把名为readme.txt的文件取消暂存
git commit -m '初始化版本库'	提交代码到本地仓库 “初始化版本库”为本次提交的注释信息

这样我们就创建了一个新的 Git 仓库

二、从现有仓库克隆

这个适合我们在公司环境下用，我们可以先把某个项目的 Git 仓库复制一份出来，这就需要用到 git clone 命令。如果你熟悉其他的 VCS 比如 Subversion，你可能已经注意到这里使用的是 clone 而不是 checkout。

Git 收取的是项目历史的所有数据（每一个文件的每一个版本），服务器上有的数据克隆之后本地也都有了。实际上，即便服务器的磁盘发生故障，用任何一个克隆出来的客户端都可以重建服务器上的仓库，回到当初克隆时的状态。

命令格式为：**git clone [url] [projectName]** 如下两行均为克隆 Ruby 语言的 Git 代码仓库 Grit。

```
git clone git://github.com/schacon/grit.git # projectName 省略时以 grit 为默认目录
```

这会在当前目录下创建一个名为“grit”的目录，其中内含一个 .git 的目录，并从同步后的仓库中拉出所有的数据，取出最新版本的文件拷贝。如果进入这个新建的 grit 目录，你会看到项目中的所有文件已经在里边了，准备好后续的开发和使用。如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令最后指定：

```
git clone git://github.com/schacon/grit.git mygrit # 以 mygrit 为默认目录
cd mygrit
git gc
```

记录每次更新到仓库

工作目录下面的所有文件的这两种状态

已跟踪 已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是未更新，已修改或者已放入暂存区。初次克隆某个仓库时，工作目录中的所有文件都属于已跟踪文件，且状态为未修改。**未跟踪** 而所有其他文件都属于未跟踪文件。它们既没有上次更新时的快照，也不在当前的暂存区域。比如：一个全新的文件。

在编辑过某些文件之后，Git 将这些文件标为已修改。我们逐步把这些修改过的文件放到暂存区域，然后等最后一次性提交暂存区域的所有文件更新，如此重复。如下图

检查当前文件状态

现在我们可以再命令行下回到刚才的git目录。要确定哪些文件当前处于什么状态，可以用 git status 命令

```
git status
# On branch master
nothing to commit (working directory clean)
```

新建一个test.txt的文件，再用git status来看

```
vi test.txt # 按i输入内容然后 按ESC 按shift+z+z保存并退出
git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       test.txt
nothing added to commit but untracked files present (use "git add" to track)
```

跟踪新文件

使用命令 git add 开始跟踪一个新文件，如跟踪刚才建立的test.txt文件

```
git add test.txt
git status
# On branch master
# Changes to be committed:
```

```
# (use "git reset HEAD <file>..." to unstage)
#
# new file: test.txt
#
```

只要在“Changes to be committed”这行下面的，就说明是已暂存状态。

暂存已修改文件

我们修改一下刚才git目录下已存在的文件，如benchmarks.txt

```
vim benchmarks.txt # 按i输入内容然后 按ESC 按shift+z+z保存并退出
git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# new file: test.txt
#
# Changed but not updated:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified: benchmarks.txt
#
```

文件 benchmarks.txt 出现在“Changed but not updated”这行下面，说明已跟踪文件的内容发生了变化，但还没有放到暂存区。要暂存这次更新，需要运行 git add 命令，这是个多功能命令，根据目标文件的状态不同，此命令的效果也不同

- 可以用它开始跟踪新文件
- 把已跟踪的文件放到暂存区
- 合并时把有冲突的文件标记为已解决状态

```
git add benchmarks.txt
git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# modified: benchmarks.txt
# new file: test.txt
#
```

现在两个文件都已暂存，下次提交时就会一并记录到仓库。

假设此时，你想要在 benchmarks.txt 里再加一行，重新编辑存盘后，准备好提交。不过稍等一下，再运行 git status 看看：

```
vim benchmarks.txt
git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# modified: benchmarks.txt
# new file: test.txt
#
```

```
# Changed but not updated:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   benchmarks.txt
#
```

现在我们发现 benchmarks.txt 文件出现了两次！一次算未暂存，一次算已暂存，这怎么可能呢？好吧，实际上 Git 只不过暂存了你运行 git add 命令时的版本，如果现在提交，那么提交的是添加注释前的版本，而非当前工作目录中的版本。所以，运行了 git add 之后又作了修订的文件，需要重新运行 git add 把最新版本重新暂存起来：

```
git add benchmarks.txt
git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   benchmarks.txt
#   new file:   test.txt
#
```

忽略某些文件

非重点 有兴趣的同学可以参考 <http://progit.org/book/zh/ch2-2.html>

查看已暂存和未暂存的更新

```
git diff --staged # 已经暂存起来的文件和上次提交时的快照之间的差异 也可以用 git diff --cached
git diff # 直接使用此命令是 工作目录中当前文件和暂存区域快照之间的差异
```

提交更新

现在的暂存区域已经准备妥当可以提交了。在此之前，请一定要确认还有什么修改过的或新建的文件还没有 git add 过，否则提交的时候不会记录这些还没暂存起来的变化。所以，每次准备提交前，先用 git status 看下，是不是都已暂存起来了，然后再运行提交命令 git commit：

```
git status
git commit -m "提交备注" # 可以直接提交
这样我们就完成了一次git提交
```

跳过使用暂存区域

不推荐 尽管使用暂存区域的方式可以精心准备要提交的细节，但有时候这么做略显繁琐。Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤：
git commit -a -m "强制提交备注"

移除文件

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

```
git rm somefile #
如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 -f
git rm -f somefile
```

我们想把文件从 Git 仓库中删除（亦即从暂存区域移除），但仍然希望保留在当前工作目录中。换句话说，仅是从跟踪清单中删除。

```
git rm --cached readme.txt
批量移除
git rm log/*.log
```

移动文件

要在 Git 中对文件改名，可以这么做：

```
git mv file_from file_to
相当于执行以下的3个命令
mv file_from file_to
git rm file_from
git add file_to
```

示例如下

```
git mv test.txt testtest.txt
git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   testtest.txt
```

查看提交历史

在提交了若干更新之后，又或者克隆了某个项目，想回顾下提交历史，可以使用 git log 命令。

```
git clone git://github.com/schacon/simplegit-progit.git
cd simplegit-progit/
```

然后在此项目中运行 git log，应该会看到下面的输出：

```
git log
```

```
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gmail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

changed the verison number

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700
```

removed unnecessary test code

```
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
```

first commit

更多参数说明

参数 说明

-p 按补丁格式显示每个更新之间的差异。
--stat 显示每次更新的文件修改统计信息。
--shortstat 只显示 --stat 中最后的行数修改添加移除统计。
--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。
--abbrev-commit 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。
--relative-date 使用较短的相对时间显示（比如，“2 weeks ago”）。
--grap 显示 ASCII 图形表示的分支合并历史。
--pretty 使用其他格式显示历史提交信息。可用的选项包括 oneline, short, full, fuller 和 format（后跟指定格式）。
-n 仅显示最近的 n 条提交
--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。
--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。

```
git log --pretty=oneline
```

```
git log --pretty=format:"%h - %an, %ar : %s"
```

#如果要查看 Git 仓库中，2008 年 10 月期间，Junio Hamano 提交的但未合并的测试脚本（位于项目的 t/ 目录下的文件）

```
git log --pretty="%h:%s" --author=gitster --since="2008-10-01" \  
--before="2008-11-01" --no-merges -- t/
```

选项 说明

%H 提交对象（commit）的完整哈希字符串

%h 提交对象的简短哈希字符串

%T 树对象（tree）的完整哈希字符串

%t 树对象的简短哈希字符串

%P 父对象（parent）的完整哈希字符串

%p 父对象的简短哈希字符串

%an 作者（author）的名字

%ae 作者的电子邮件地址

%ad 作者修订日期（可以用 -date= 选项定制格式）

%ar 作者修订日期，按多久以前的方式显示

%cn 提交者（committer）的名字

%ce 提交者的电子邮件地址

%cd 提交日期

%cr 提交日期，按多久以前的方式显示

%s 提交说明

撤消操作

修改最后一次提交

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交：

```
git commit -m 'first'
```

```
git add test.txt
```

```
git commit --amend -m 'first too'
```

取消已经暂存的文件

```
git reset HEAD filename
```

取消对文件的修改

这里指未git add到的暂存区的文件

在用这条命令前，请务必确定真的不再需要保留刚才的修改。

```
git checkout -- filename
```


远程仓库的使用

同他人协作开发某个项目时，需要管理这些远程仓库，以便推送或拉取数据，分享各自的工作进展。管理远程仓库的工作，包括添加远程库，移除废弃的远程库，管理各式远程库分支，定义是否跟踪这些分支，等等。

查看当前的远程库

使用git remote 查看当前配置有哪些远程仓库，至少可以看到一个名为 origin 的远程库，Git 默认使用这个名字来标识你所克隆的原始仓库：

```
git clone git://github.com/schacon/ticgit.git
cd ticgit
git remote # 列出每个远程库的简短名字
origin
git remote -v # 显示对应的克隆地址
origin  git://github.com/schacon/ticgit.git (fetch)
origin  git://github.com/schacon/ticgit.git (push)
```

添加远程仓库

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用格式：git remote add [shortname] [url]：

```
cd ticgit/
git remote -v
origin  git://github.com/schacon/ticgit.git (fetch)
origin  git://github.com/schacon/ticgit.git (pus
git remote add pb git://github.com/paulboone/ticgit.git
git remote -v
origin  git://github.com/schacon/ticgit.git (fetch)
origin  git://github.com/schacon/ticgit.git (push)
pb      git://github.com/paulboone/ticgit.git (fetch)
pb      git://github.com/paulboone/ticgit.git (push)
现在可以用字符串 pb 指代对应的仓库地址了
git fetch pb
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 44 (delta 23), reused 38 (delta 17)
Unpacking objects: 100% (44/44), done.
From git://github.com/paulboone/ticgit
* [new branch]    master    -> pb/master
* [new branch]    ticgit    -> pb/ticgit
```

从远程仓库抓取数据

命令 git pull [remote-name]

此命令会到远程仓库中拉取所有你本地仓库中还没有的数据。运行完成后，你就可以在本地访问该远程仓库中的所有分支，将其中某个分支合并到本地，或者只是取出某个分支。如果是克隆了一个仓库，此命令会自动将远程仓库归于 origin 名下。所以，git pull origin 会抓取从你上次克隆以来别人上传到此远程仓库中的所有更新（或是上次 pull 以来别人提交的更新）。有一点很重要，需要记住，pull 命令只是将远端的数据拉到本地仓库，并不会自动合并到当前工作分支，只有当你确实准备好了，才能手工合并。

推送数据到远程仓库

项目进行到一个阶段，要同别人分享目前的成果，可以将本地仓库中的数据推送到远程仓库。实现这个任务的命令很简单：git push [remote-name] [branch-name]。如果要把本地的 master 分支推送到 origin 服务器上（再次说明下，克隆操作会自动使用默认的 master 和 origin 名字），可以运行下面的命令：

```
git push origin master
```

只有在所克隆的服务器上有写权限，或者同一时刻没有其他人在推数据，这条命令才会如期完成任务。如果你推数据前，已经有其他人推送了若干更新，那你的推送操作就会被驳回。你必须先把他们的更新抓取到本地，并到自己的项目中，然后才可以再次推送。

远程仓库的删除和重命名

在新版 Git 中可以用 git remote rename 命令修改某个远程仓库的简短名称，比如想把 pb 改成 paul，可以这么运行：

```
git remote rename pb paul
git remote
origin
paul
```

移除远程仓库

```
git remote rm paul
git remote
origin
```

更多 4

上一篇 c++ auto_ptr智能指针详解
下一篇 windows下git bash显示中文

主题推荐 团队 电子邮件 版本控制 服务器 数据库

猜你在找

- CentOS6.4安装samba后windows不能访问
- php实现hash值计算（浅谈php的高精度计算）
- mac os x 安装 PCRE
- 使用socket的Linux上的C语言文件传输顺序服务器和客户
- 软件自动化测试—代码覆盖率
- Pro Android学习笔记（八四）：了解Package（3）：包间
- zookeeper原理与安装
- Spring Data JPA 简单介绍(转)
- IOS 利用Model 反射属性 自动 创建表 插入 查询 修改
- GNU makefile 指南



银座佳驿连锁酒店烟台开发区长江路店

Qunar 去哪儿
低至 CN¥114.00
经济型

查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题
- Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
- VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
- BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
- Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
- FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
- Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
- Angular Cloud Foundry Redis Scala Django Bootstrap

