提交



<https://blog.csdn.net/qq_21108099/article/details/121508185>

<https://git-scm.com/book/zh/v2>

git clone

<https://github.com/guoxingjin/gavin_android_study.git>

sudo gedit /etc/hosts

140.82.112.4 [github](https://so.csdn.net/so/search?q=github&spm=1001.2101.3001.7020).com

# 工作区暂存区和仓库区

https://www.cnblogs.com/EF1355/p/14631572.html

# 服务器端Git仓库的创建（Ubuntu）

https://www.linuxidc.com/Linux/2009-06/20421.htm

http://www.linuxidc.com/Linux/2011-02/32542.htm

Linux下Git简单使用教程（以Android为例）

<https://www.linuxidc.com/Linux/2010-11/29883.htm>

Linux源码

[git@github.com:torvalds/linux.git](mailto:git@github.com:torvalds/linux.git)

查看分支的树形结构？

配置Github

<https://blog.csdn.net/tichimi3375/article/details/79844514>

官方教程

[https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF](https://git-scm.com/book/zh/v2/Git-分支-远程分支)

创建版本

[https://git-scm.com/book/zh/v2/Git-%E4%B8%8E%E5%85%B6%E4%BB%96%E7%B3%BB%E7%BB%9F-%E8%BF%81%E7%A7%BB%E5%88%B0-Git](https://git-scm.com/book/zh/v2/Git-与其他系统-迁移到-Git)

<https://www.liaoxuefeng.com/wiki/896043488029600/896827951938304>

命令

<https://www.jianshu.com/p/46ffff059092>

android studio 配置git

<https://www.jianshu.com/p/059ed1e01229>

<https://www.liaoxuefeng.com/wiki/896043488029600/899998870925664>

图形界面的配置

<https://blog.csdn.net/xiaobin_hlj80/article/details/10953701>

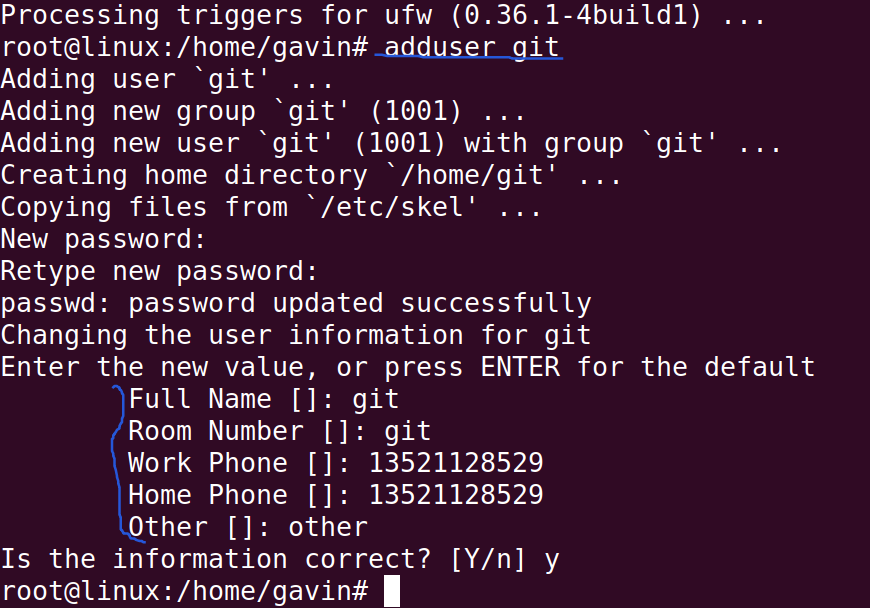
# 搭建Git服务器

第一步，安装git和openssh-server

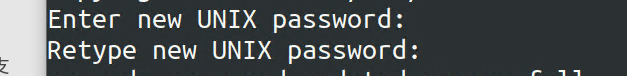
$ sudo apt-get install git openssh-server

第二步，创建一个git用户，用来运行git服务：

$ sudo adduser git



~~Unix密码干啥的~~



~~哈哈，干啥用的~~

### 创建登录证书

创建路径.ssh，然后创建authorized\_keys文件，这里路径和文件名不能任取，该文件用于存放各个git客户端的RSA公钥，只有在这个列表里的客户端才能连接git服务。公钥在下一步创建。

cd /home/git/

sudo mkdir .ssh

sudo touch .ssh/authorized\_keys

第三步，创建本地证书：

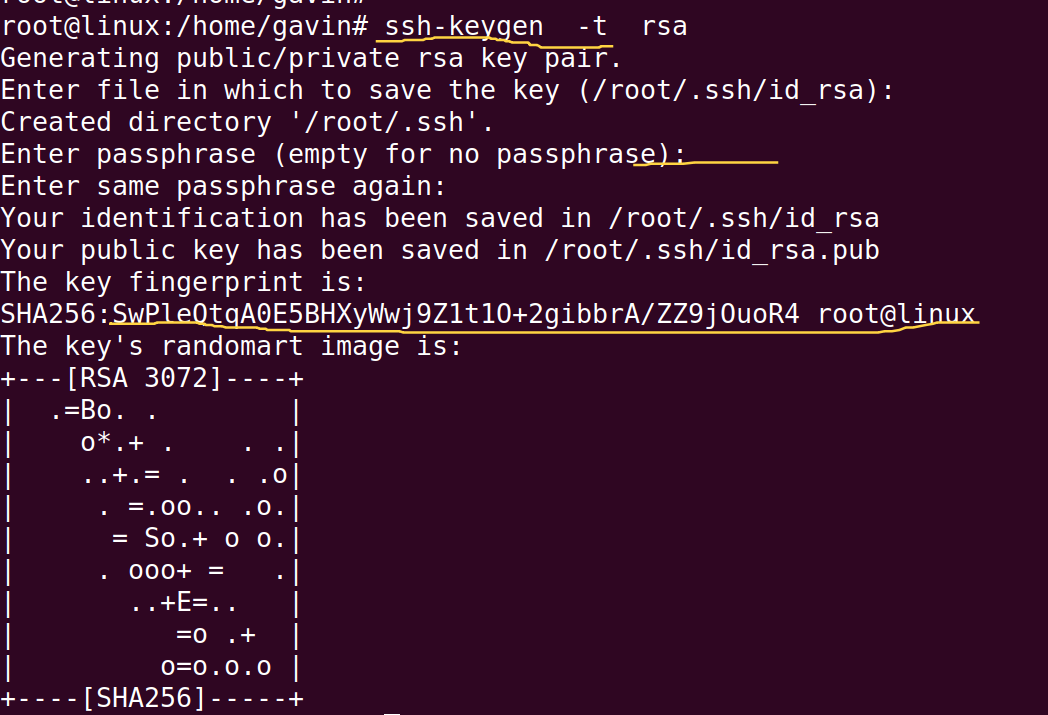
这一步在本地端进行。不同的本地用户有不同的KEY

$  ssh-keygen  -t  rsa       # 这里会提示输入密码，我们不输入直接回车即可

~~ssh-keygen -t rsa –C~~ [~~your\_email@youremail.com~~](mailto:your_email@youremail.com)

github

ssh-keygen -t rsa

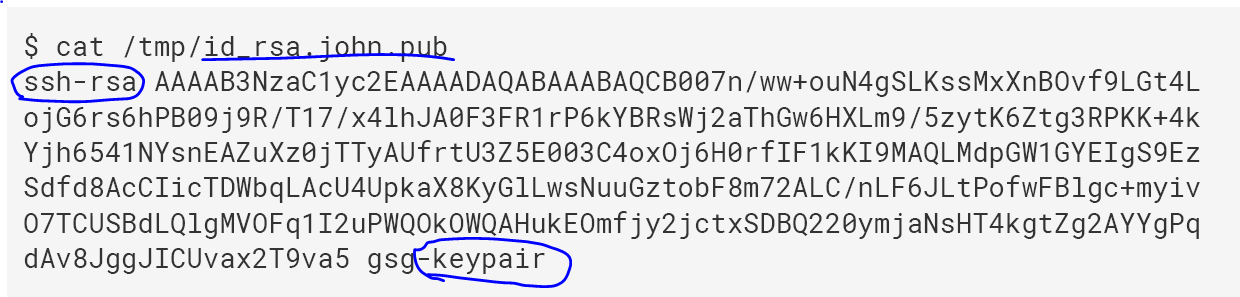


~~SHA256:SwPleQtqA0E5BHXyWwj9Z1t1O+2gibbrA/ZZ9jOuoR4 root@linux~~

SHA256:VeKrXMe5tirh1oMTrNJVbrq4MAk3cuq6U/sbcJbmALM gavin@linux

添加到

公钥内容如下图：



将这些公钥加入git的系统用户的.ssh目录下authorized\_keys文件的末尾

cat -n /home/gx/.ssh/id\_rsa.pub > /home/gx/.ssh/authorized\_keys

cat -n /home/gavin/13521128529.pub > /home/git/.ssh/authorized\_keys

home/gavin：是客户端

/home/git： 服务端

如果想确认是否成功，可以输入命令查看：

head -n 5 authorized\_keys

命令的意思是查看authorized\_keys文件前5行的内容

第四步，初始化Git仓库：

先选定一个目录作为Git仓库，假定是/home/gx/guoxing/git\_server/test\_service，在/ test\_service目录下输入命令：

$ sudo git init --bare gavin\_service.git

Git就会创建一个裸仓库，没有工作区，因为服务器上的Git仓库纯粹是为了共享，所以不让用户直接登录到服务器上去改工作区，并且服务器上的Git仓库通常都以.git结尾。

然后，把 gavin\_service .git 的 owner 改为 git ：

$ su**do** chown -R git:git gavin\_service.git

第五步，禁用shell登录：

出于安全考虑，第二步创建的git用户不允许登录shell，这可以通过编辑/etc/passwd文件完成。找到类似下面的一行：

git:x:1001:1001:,,,:/home/git:/bin/bash

改为：

git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell

这样，git用户可以正常通过ssh使用git，但无法登录shell，因为我们为git用户指定的git-shell每次一登录就自动退出。切换到root权限下

第六步，克隆远程仓库：

现在，可以通过git clone命令克隆远程仓库了，在各自的电脑上运行：

$ git clone [git@server:/srv/sample.git](mailto:git@server:/srv/sample.git) 我的路径如下

git clone git@localhost: /home/gx/guoxing/test\_git/git\_server/gavin\_service.git

Cloning into 'sample'...

warning: You appear to have cloned an empty repository.

# git提交本地代码到远程服务器

1、提交代码

命令版本操作步骤：

第一步：git add .

第二步：git commit -m "提交代码的注释内容" （将代码提交到本地仓库中）

第三步：git pull（防止和别人代码冲突，如果服务器的文件发生变化，这里出现冲突）

第四步：git status （查看当前工作目录树的工作修改状态）

第五步：git push （将代码推送到服务器），如果推送到dev上，是不是push origin dev？

git push origin HEAD:ivicar-lorry-6 公司服务器，推送到ivicar-lorry-6分支上

客户端版本操作步骤：

第一步：右键-->TortoiseGit--> pull（防止和别人代码冲突，每天第一件事就是这个了）

第二步：右键-->Git Commit --> "master"（将代码提交到本地仓库中）

第三步：右键-->TortoiseGit-->push（将代码推送到服务器）

2、修改本地代码后想恢复到修改前的版本

客户端操作步骤：右键-->TortoiseGit-->Revert

# 在已存在目录中初始化仓库

# git配置用户名和邮箱

### 用户名和邮箱的作用

    用户名和邮箱地址是本地git客户端的一个变量 . 用户每次提交代码都会记录用户名和邮箱 .

### 设置

设置用户名

git config --global user.name "username"

git config --global user.name gx

设置邮箱 (没有双引号)

git config --global user.email [useremail@qq.com](mailto:useremail@qq.com)

git config --global user.email [gx\_jin@163.com](mailto:gx_jin@163.com)

查看用户名和密码

git config user.name

git config user.email

### 取消http和https代理

//取消http代理

git config --global --unset http.proxy

//取消https代理

git config --global --unset https.proxy

https://blog.csdn.net/good\_good\_xiu/article/details/118567249

### 查看其他配置信息(git设置列表)

git config –list

### 下载本地和推送到服务器

https://\*\*\*/gavin\_android\_study.git -b study\_frameworks

git push origin HEAD:study\_frameworks

# 创建版本库

什么是版本库呢？版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

第二步，通过git init命令把这个目录变成Git可以管理的仓库：

$ git init

Initialized empty Git repository in /Users/michael/learngit/.git/

瞬间Git就把本地仓库建好了，而且告诉你是一个空的仓库，这个目录是Git来跟踪管理版本库的。如果你没有看到.git目录。

# 把文件添加到版本库

一定要放到learngit目录下（子目录也行），因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。

第一步，用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

执行上面的命令，没有任何显示，这就对了，Unix的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令git commit告诉Git，把文件提交到仓库：

$ git commit -m "wrote a readme file"

[master (root-commit) eaadf4e] wrote a readme file

1 file changed, 2 insertions(+)

create mode 100644 readme.txt

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

## git log

如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数

## git reset

我们要把当前版本append GPL回退到上一个版本add distributed，就可以使用git reset命令：

$ git reset --hard HEAD^

HEAD is now at e475afc add distributed

## git reflog

使用reset向后切换，使用reflog向前切换

# 工作区和暂存区

Workspace：工作区

Index / Stage：暂存区

Repository：仓库区（或本地仓库）

Remote：远程仓库

## 版本库（Repository）

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage暂存区，还有Git自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

你可以简单理解为，需要提交的文件修改通过放到暂存区，然后，一次性提交暂存区的所有修改。

## git status

查看状态

# 管理修改

工作区的每次修改，都要添加git add到暂存区，git commit只负责把暂存区的修改提交，

# 撤销修改

1. 修改的文件尚未执行add

工作区是需要修改的，而暂存区是干净的，因此这一步需要使用git checkout命令

执行checkout ,全部撤回，执行checkout .;对于执行单个文件，执行checkout 文件名

注意：如果想要撤销新增的文件，这种方法不起作用！

1. 修改的文件已经add 尚未[commit](https://so.csdn.net/so/search?q=commit&spm=1001.2101.3001.7020)

使用git reset命令，git reset命令有三种模式：

* mixed模式

git reset即为该模式：会**清除暂存区，保留工作区**，即撤销“add”这步操作，让其回到add之前（**注意：对工作区的修改仍然存在，撤销的仅仅只是add的过程**），因此，还需要再执行一次git checkout才行

* soft模式

git reset --soft

* hard模式

git reset –hard HEAD^[ok,^ 是6的上键]

放弃最后一次的[commit](https://so.csdn.net/so/search?q=commit&spm=1001.2101.3001.7020)而回复到再上一次commit的指令

1. 修改的文件已经commit

使用git reset命令，但是不同的是，需要使用soft模式或者hard模式

对于soft模式，会变换版本，但是保留暂存区，换言之，仅仅是撤销了"commit"这步操作，“add”所施加的影响还在。

hard模式，直接变换版本，清空暂存区、工作区，换言之，是彻彻底底地擦出了修改的痕迹，最直接地复原了最初的版本。

一般使用hard的情况会比较多；

### git add 添加多余文件 撤销操作

git reset HEAD 后面什么都不跟的，就是上一次add 里面的内容全部撤销  
git reset HEAD XXX 后面跟文件名，就是对某个文件进行撤销

### ****git commit** 撤销操作**

git reset --soft HEAD^ //撤销了[commit](https://so.csdn.net/so/search?q=commit&spm=1001.2101.3001.7020)操作，代码仍然保留

**git reset** --mixed //不删除工作空间改动代码，撤销commit，并且撤销git add . 操作这个为默认参数,git reset --mixed HEAD^ 和 git reset HEAD^ 效果是一样的

**git reset** --soft  不删除工作空间改动代码，撤销commit，不撤销git add .

**git reset** --hard 删除工作空间改动代码，撤销commit，撤销git add . 完成这个操作后，就恢复到了上一次的commit状态。

git commit –amend // 修改注释，进入默认vim，修改注释完保存就好

git checkout 丢弃工作区的修改，

命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区（即没有执行add），现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到git checkout命令

git reset HEAD <file>

可以把暂存区的修改撤销掉（unstage），重新放回工作区：

# 删除文件

命令git rm删掉，并且git commit：如下：

# 添加远程仓库

，，，

<https://blog.csdn.net/u012308586/article/details/103256578>

<https://www.runoob.com/git/git-remote-repo.html>

可以理解为远程分支

# 分支

## 创建与合并分支

在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

创建dev切换到dev分支：

$ git checkout -b dev

Switched to a new branch 'dev'

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

然后，用git branch命令查看当前分支：

$ git branch

\* dev

master

git branch命令会列出所有分支，当前分支前面会标一个\*号。

现在，把dev分支的工作成果合并到master分支上：

**git merge命令用于合并指定分支到当前分支。**合并后，再查看readme.txt的内容【怎么查看， gavin】，就可以看到，和dev分支的最新提交是完全一样的。

注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。

合并完成后，就可以放心地删除dev分支了：

branch -d dev

因为创建、合并和删除分支非常快，所以Git鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。

Git鼓励大量使用分支，总结如下：

查看：git branch

查看远程项目所有分支 “git branch -a”

创建：git branch <name>

切换：git checkout <name>或者git switch <name>

创建+切换：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除：git branch -d <name>

### 创建分支 并 提交到远程分支

<https://www.cnblogs.com/bluestorm/p/6252900.html>

### 建立

git remote add <branch\_name> 仓库地址，这些远程仓库就是的，

git remote add gavin1 git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git

查看使用git remote –v，这里只是建立远程，没有和本地建立联系，所有使用git branch -r查看不到

获取到本地  git fetch gavin1

其中origin为远程分支。

推送到服务器

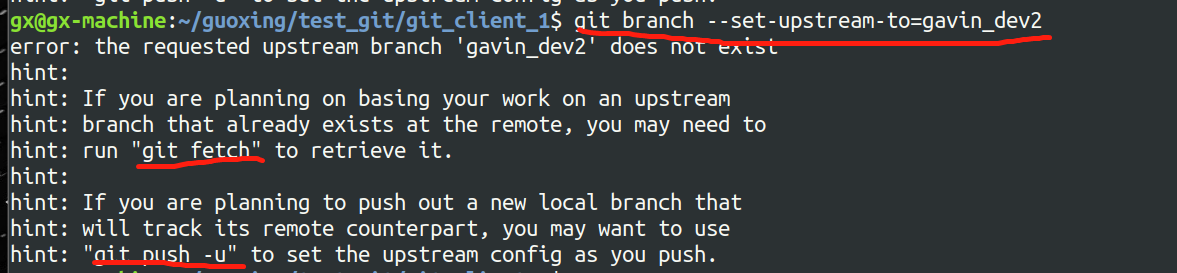
<https://www.cnblogs.com/Camier-myNiuer/p/5558884.html>

## 服务端及本地建立分支，推数据到服务器

1. ~~查看本地当前分支~~
2. ~~建立远程分支~~
3. ~~本地分支和远程分支建立连接~~

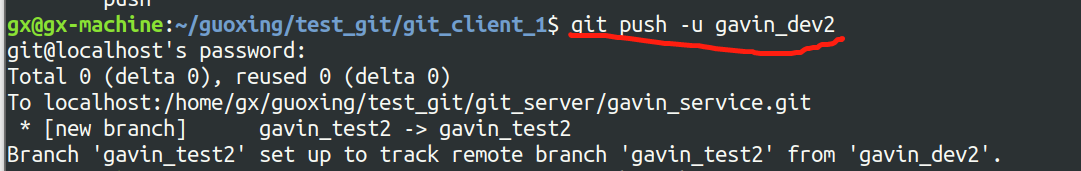
~~这里有三个方法，只有三成功了，暂时没有搞懂，先把流程跑通~~

git branch --set-upstream-to=origin/dev

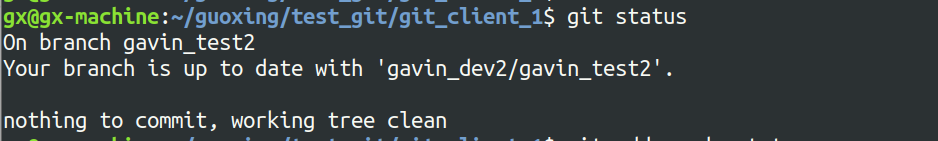


1. Push 命令

将当前的本地分支和远程分支关联



1. 查看关联状态



Add建立分支后，代码文件自接过来，在此基础上修改后执行Commit即可。

git log --oneline --graph --decorate

## 从当前分支创建自己的分支1

1. 在本地创建自己的分支

git branch dev\_lsw

2. 同步分支到github上面

git push origin dev\_lsw

3. 本地当前应该是master分支，切换到自己的分支上，然后把本地分支和远程分支做关联

git checkout dev\_lsw，不切换就是当前分支

git branch --set-upstream-to origin/dev\_lsw

4. 使用git branch -vv命令可以看到

dev\_lsw 28d9b82 [origin/dev\_lsw] Change Chinese comments into English

  master  86f52f7 [origin/master] Merge pull request #2 from shiwei-li/master

表面分支关联成功

## 本地建立分支后然后推到远程2

<https://www.cnblogs.com/silencey/p/10794784.html>

git checkout -b dev2

git remote add dev2 git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git

git push -u dev2

git log --oneline --graph –decorate

## 删除文件夹

* 查看本地分支下的文件

Ls –la

* 删除raindow文件夹及其下所有的文件

git rm raindow -r -f

* 同步删除操作到远程分支：

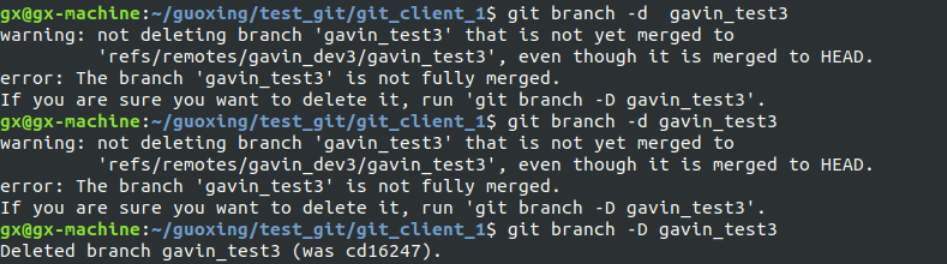
git commit -m "delete raindow"

## 删除分支

git branch -r //查看远程所有分支

合并前

git branch -d bug\_xzx



合并后

删除本地的bug\_xzx分支 git branch -d bug\_xzx

删除远程的bug\_xzx分支 git push origin --delete bug\_xzx

### 删除本地分支

git branch –d branch\_name

### 删除远程

git remote remove dev2 git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git

为啥这个删除命名，不能执行成功

git remote remove dev2 可以成功；

另外

使用如下操作方法，查看远程仓库名称以及路径相关信息，可以删除错误的远程仓库名称，重新添加新的远程仓库；

**git remote -v：                查看远程仓库详细信息，可以看到仓库名称**

**git remote remove orign： 删除orign仓库**

**git remote add origin 仓库地址：              重新添加远程仓库地址**

**gti push -u origin master：                 提交到远程仓库的master主干**

下面的命令请谨慎操作

git push orgin :dev # 删除远程的dev分支

git checkout master # 切换到master分支

git branch -d dev # 删除本地的dev分支

## 删除分支2

从服务器拉回分支： git branch feature\_test

删除本地分支 git branch -D feature\_test

就把本地的feature\_test分支给删除了，但是这样删除的仅仅只是本地的分支，远端的分支还一直保存着，那么远端的分支如何删除呢？使用如下命令：

git push origin --delete feature\_test

**git remote remove**

这两个命令啥区别？？？

git push origin --delete 分支

git remote remove 分支

git branch -d –r

## 本地分支、远程分支关联

git查看本地分支关联（跟踪）的远程分支之间的对应关系，本地分支对应哪个远程分支

git branch -vv

将本地分支同远程分支进行关联，可以分为以下2种情形：

* 情形1：本地已经创建了分支dev（以dev为例，下同），而远程没有

可以通过以下2种方法在远程创建分支dev，并与本地分支进行关联：

方法1： git push -u origin dev

方法2： git push --set-upstream origin dev

* 情形2：远程已经创建了分支dev,而本地没有

在本地创建分支并与远程分支进行关联，也有2种方法：

* 方法1分为两步：

step1:先将远程分支pull到本地 git pull origin dev

step2:再在本地创建分支并与之关联，又有2种方法

(1)git checkout -b dev origin/dev 【成功，gavin】

(2)git checkout -b dev --track origin/dev #可以简写为git checkout --track origin/dev

* 方法2：可以在pull远程分支的同时，创建本地分支并与之进行关联

git pull origin dev:dev-------两个dev分别表示远程分支名：本地分支名【单独执行，没有成功，gavin】

* 本地和远程都存在，切换到dev下，执行下面的命令表示关联

git branch --set-upstream-to=origin/dev

## 分支（branch）的基本操作

git remote show origin

git remote prune origin

git log --oneline --graph --decorate

git branch //查看本地所有分支

git branch -r //查看远程所有分支

git branch -a //查看本地和远程的所有分支

git checkout branchName//分支的切换

git checkout –b branchName//创建并切换分支

git branch <branchname> //新建分支

git branch -d <branchname> //删除本地分支

git branch -d -r <branchname> //删除远程分支，删除后还需推送到服务器

git push origin:<branchname> //删除后推送至服务器

git branch -m <oldbranch> <newbranch> //重命名本地分支

/\*\*

\*重命名远程分支：

\*1、删除远程待修改分支

\*2、push本地新分支到远程服务器

\*/

//git中一些选项解释:

-d

--delete：删除

-D

--delete --force的快捷键

-f

--force：强制

-m

--move：移动或重命名

-M

--move --force的快捷键

-r

--remote：远程

-a

--all：所有

remote show origin查看remote地址，远程分支，还有本地分支与之相对应关系等信息

git remote prune origin  远程仓库已经不存在的分支，删除了那些远程仓库不存在的分支

## Bug分支

Git还提供了一个stash功能，可以把当前工作现场“储藏”起来

$ git stash

Saved working directory and index state WIP on dev: f52c633 add merge

刚才的工作现场存到哪去了？用git stash list命令

$ git stash list

stash@{0}: WIP on dev: f52c633 add merge

工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

### 小结

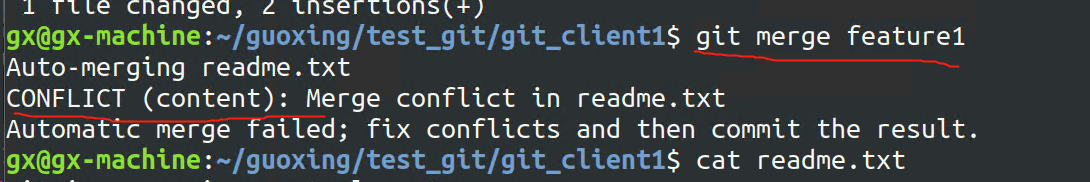
修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场；

在master分支上修复的bug，想要合并到当前dev分支，可以用git cherry-pick <commit>命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

# 解决冲突

我们在一个分支dev上完成后，修改正常提交操作后，新建一个分支test，然后提交，再在dev上修改后，执行提交，就会产生冲突，



用带参数的git log也可以看到分支的合并情况

git log --graph --pretty=oneline --abbrev-commit

### 小结

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

## 分支管理策略

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

$ git merge --no-ff -m "merge with no-ff" dev

Merge made by the 'recursive' strategy.

readme.txt | 1 +

1 file changed, 1 insertion(+)

本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

合并后，我们用git log看看分支历史

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

# 多人协作

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin。

查看远程库的信息，用git remote或者git remote -v显示更详细的信息：

## 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

$ git push origin master

如果要推送其他分支，比如dev，就改成：

git push origin div

并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

## 抓取分支

多人协作时，大家都会往master和dev分支上推送各自的修改。

git clone git@localhost:michaelliao/learngit.git

当你的小伙伴从远程库clone时，默认情况下，你的小伙伴只能看到本地的master分支

现在，你的小伙伴要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是他用这个命令创建本地dev分支：

git checkout -b dev origin/dev

现在，他就可以在dev上继续修改，然后，时不时地把dev分支push到远程：

### 解决版本冲突

先用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送：

如果文件发生变化，执行pull时，就发生冲突了，合并后，在

#### 查看不同

Git show

### 本地分支和远程分支连接

指定本地dev分支与远程origin/dev分支的链接

git branch --set-upstream-to=origin/dev dev

再pull

因此，多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

## 远程仓库地址修改，如何更新呢？

方式一：直接修改

git remote set-url origin git新地址

方式二：先删除后添加

git remote rm origin

git remote add origin git新地址

方式三：直接修改config文件

位置：.git->config文件-》url = git新地址（注意：.git默认是隐藏目录）

## 小结

* 查看远程库信息，使用git remote -v；
* 本地新建的分支如果不推送到远程，对其他人就是不可见的；
* 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
* 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
* 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
* 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

# rebase的操作

* rebase操作可以把本地未push的分叉提交历史整理成直线；
* rebase的目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

# 创建标签

Gavin 今天先看到这里吧

首先，切换到需要打标签的分支上；

敲命令git tag <name>就可以打一个新标签

$ git tag v1.0

用命令git tag查看所有标签：

默认标签是打在最新提交的commit上的。有时候，如果忘了打标签，方法是找到历史提交的commit id，

git log --pretty=oneline --abbrev-commit

要对add merge这次提交打标签，它对应的commit id是f52c633，敲入命令：

$ git tag v0.9 f52c633

再用命令git tag查看标签：

用git show <tagname>查看标签信息：

还可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

小结

* 命令git tag <tagname>用于新建一个标签，默认为HEAD，也可以指定一个commit id；
* 命令git tag -a <tagname> -m "blablabla..."可以指定标签信息；
* 命令git tag可以查看所有标签。

# 操作标签

删除标签

git tag -d v0.1

创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。如果要推送某个标签到远程，使用命令git push origin <tagname>：

或者，一次性推送全部尚未推送到远程的本地标签：

git push origin --tags

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除：

git tag -d v0.9

然后，从远程删除。删除命令也是push，但是格式如下：

$ git push origin :refs/tags/v0.9

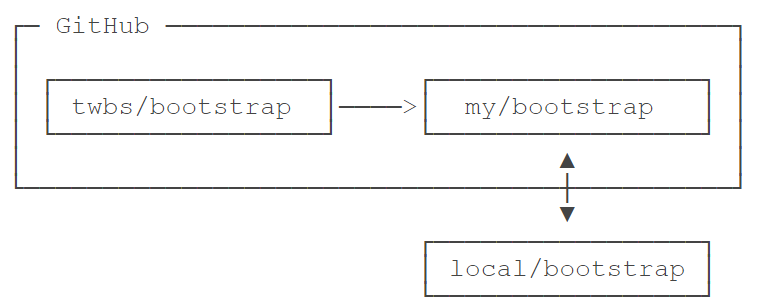
### 小结

* 命令git push origin <tagname>可以推送一个本地标签；
* 命令git push origin --tags可以推送全部未推送过的本地标签；
* 命令git tag -d <tagname>可以删除一个本地标签；
* 命令git push origin :refs/tags/<tagname>可以删除一个远程标签。

# 使用GitHub

git clone git@github.com:michaelliao/bootstrap.git

Bootstrap的官方仓库twbs/bootstrap、你在GitHub上克隆的仓库my/bootstrap，以及你自己克隆到本地电脑的仓库，他们的关系就像下图显示的那样：



如果你想修复bootstrap的一个bug，或者新增一个功能，立刻就可以开始干活，干完后，往自己的仓库推送。

如果你希望bootstrap的官方库能接受你的修改，你就可以在GitHub上发起一个pull request。当然，对方是否接受你的pull request就不一定了。

使用Gitee

# 忽略特殊文件

在Git工作区的根目录下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

忽略文件的原则是：

* 忽略操作系统自动生成的文件，比如缩略图等；
* 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
* 忽略自己的带有敏感信息的配置文件，比如存放口令的配置文件。

最后一步就是把.gitignore也提交到Git，就完成了

你确实想添加该文件，可以用-f强制添加到Git：

git add -f App.class

# 配置别名

有没有经常敲错命令？比如git status？status这个单词真心不好记。

如果敲git st就表示git status那就简单多了。

我们只需要敲一行命令，告诉Git，以后st就表示status：

git config --global alias.st status

另外

$ git config --global alias.co checkout

$ git config --global alias.ci commit

$ git config --global alias.br branch

$ git config --global alias.unstage 'reset HEAD'

--global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

配置一个git last，让其显示最后一次提交信息：

$ git config --global alias.last 'log -1'

Lg

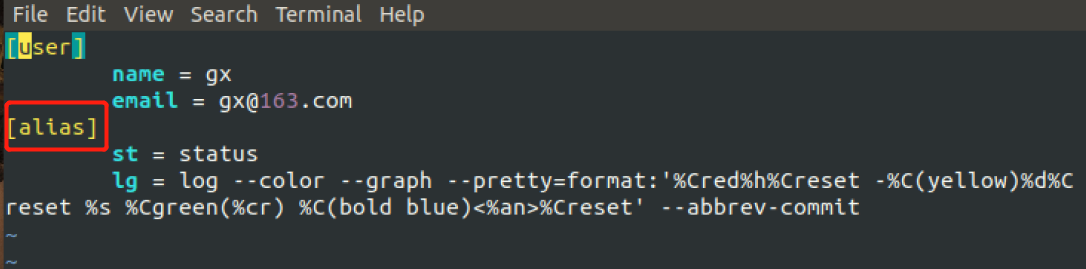
git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

### 配置文件

配置Git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。Git配置文件都放在.git/config文件中：

别名就在[alias]后面，要删除别名，直接把对应的行删掉即可。

而当前用户的Git配置文件放在用户主目录【在桌面上进入终端，pwd就是该路径】下的一个隐藏文件.gitconfig中：



# 问题

## 查看本地+远程所有分支的全部提交以及关系

git log --graph --all

gitk –all

<https://blog.csdn.net/wq6ylg08/article/details/89052225>

## git如何在某个分支下创建分支

<https://blog.csdn.net/RollingMikuzn/article/details/81134293>

## 远程仓库克隆dev分支到本地的实现

新建文件夹后

* 设置用户名、密码
* 初始化一个本地仓库 ，git init
* 查看远程分支 git remote -v
* 与远程仓库建立连接 ：git remote add origin XXXXX.git（

git remote add origin [git@localhost:/Auto/diiacarservice.git](mailto:git@localhost:/Auto/diiacarservice.git)），

git remote add origin git@github.com:guoxingjin/TorqueManageSystem.git

注意替换

* 查看本地是否具有dev分支：git branch
* 如果没有dev分支：git fetch origin dev
* 在本地创建分支dev并切换到该分支  git checkout -b dev origin/dev
* dev分支上的内容都拉取到本地  git pull origin dev
* 检查是否是配置的两个分支git log --oneline --graph --decorate
* 修改代码后如何更新dev呢：git add . ；git commit -m "这是备注信息" ；git push origin dev
* git远程仓库地址修改，如何更新呢

## 远程master到本地master

* clone
* 看看能不能提交

git clone [git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git](mailto:git@localhost:/home/gx/guoxing/test_git/git_server/gavin_service.git)

## remote HEAD refers to nonexistent ref, unable to checkout.

git remote add origin [git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git](mailto:git@localhost:/home/gx/guoxing/test_git/git_server/gavin_service.git)

## git clone fatal: unable to access '\*\*\*': OpenSSL SSL\_read: Connection was reset, errno 10054

### 原因分析

git在拉取或者提交时，中间有http或者https代理，但是本地已经存在[SSL](https://so.csdn.net/so/search?q=SSL&spm=1001.2101.3001.7020)协议了，所以只要取消这两者的代理即可

### 解决方案

//取消http代理

git config --global --unset http.proxy

//取消https代理

git config --global --unset https.proxy

<https://github.com/guoxingjin/gavin_android_study.git>

1. 下载后，初始化

**Git init**

1. 配置用户名和邮箱

## 作用：

用户每次用git 提交代码时都会记录用户名和邮箱

## 设置用户名和邮箱的命令

git config --global user.name gx1

git config --global user.email gx\_jin@163.com

## 查看用户名和密码

git config user.name

git config user.email

1. 分支

查看分支：$ git branch

查看状态：$ git status

1. Dd

## 新建远程分支

* 新建一个本地分支：

$ git checkout -b 分支名

* 查看一下现在的分支状态：

$ git branch //星号(\*)表示当前所在分支。现在的状态是成功创建的新的分支并且已经切换到新分支上。

* 把新建的本地分支push到远程服务器，远程分支与本地分支同名（当然可以随意起名）：

$ git push origin 分支名:分支名

//使用git branch -a查看所有分支，会看到remotes/origin/localbranch这个远程分支，说明新建远程分支成功。

//两个分支名字与冒号中间不能存在空格

## 删除远端分支

* 查看分支

git branch -a

* 切换到删除的分支

git checkout xxx

* 删除该分支

git push origin --delete xxx

建立远端分支

<https://blog.csdn.net/qq_43679405/article/details/111580590>

https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E6%96%B0%E5%BB%BA%E4%B8%8E%E5%90%88%E5%B9%B6

# SourceTree

破解

<https://blog.csdn.net/qq_24445853/article/details/93855377>

廖雪峰

<https://www.liaoxuefeng.com/wiki/896043488029600/900004111093344>

备案

<https://www.cnblogs.com/OrgCheng/p/11765373.html>

<https://www.jianshu.com/p/0cf8590ab9fd>

[https://www.git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF](https://www.git-scm.com/book/zh/v2/Git-分支-远程分支)

git fetch & pull详解

<https://www.cnblogs.com/runnerjack/p/9342362.html>

[Git常用命令](https://www.cnblogs.com/isky0824/p/12103214.html)

<https://www.cnblogs.com/isky0824/p/12103214.html>

<https://www.cnblogs.com/bluestorm/p/6252900.html>

git push dev gx\_temp:gx\_temp

查看远程分支

问题

1： 在本地执行git init后，无法找到本地的master分支，也无法找到远程分支，

[git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git](mailto:git@localhost:/home/gx/guoxing/test_git/git_server/gavin_service.git)

添加远程库（后面）

<https://www.liaoxuefeng.com/wiki/896043488029600/898732864121440>

git init

git touch READMEgit

git add README

git commit -m 'first commit'

git remote add origin [git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git](mailto:git@localhost:/home/gx/guoxing/test_git/git_server/gavin_service.git)

git push origin gavin

当前目录dev分子下：

git remote add dev git@localhost:/home/gx/guoxing/test\_git/git\_server/gavin\_service.git

git push -u

git checkout -b test2

git push --set-upstream origin dev

问题