基础

切换分支 : **git** **checkout** -**b** **dev** **origin**/**dev**

拉取代码与远程库同步 : **git** **pull**

查状态 : **git** **status**

查提交日志：**git log**

提交暂存区 : **git** **add** -**A**全部 **git** **add** **x**.**txt** 单个文件 **git add \*.java**某类文件

提交本地仓库 : **git** **commit** -**m** '注释'

切换分支 : **git checkout master**

合并代码 : **git** **merge** **dev**

提交远程分支 : **git** **push** **origin** **head**:**refs**/**for**/**master**

同步当前分支 : **git** **merge master**

git本地新建一个分支后，必须要做远程分支关联。先新建远程分支再关联。

修改注释：# **git commit --amend** 然后在出来的编辑界面，直接编辑注释的信息,保存退出 (**:wq!**)

**生成秘钥：**

依次运行以下命令

**git config --global user.name** XXXX #请将XXXX替换为你的gerrit中的“ Username”

**git config --global user.email** YYYY #将YYYY替换为你的公司邮箱

**ssh-keygen** （设置密码处按enter即可）

**eval `ssh-agent`; ssh-add;**

**cat ~/.ssh/id\_rsa.pub** #复制显示出的文件内容

打开ssh-key设置页面，点击"Add key"将上一步骤中复制的内容粘贴并追加到秘钥列表

克隆

**clone**地址 : **git** **clone** **ssh**://**zhangsan**@**gerrit**.**dev**.**aixuexi**.**com**:29418/**bridge**.**git**

第一次**clone**一个项目(**Clone** **with** **commit**-**msg** **hook**)

**git** **clone** **ssh**://**zhangsan**@**gerrit**.**dev**.**aixuexi**.**com**:29418/**IronMan** && **scp** -**p** -**P** 29418 **zhangsan**@**gerrit**.**dev**.**aixuexi**.**com**:**hooks**/**commit**-**msg** **IronMan**/.**git**/**hooks**/

拉取代码

**git** **pull**命令的作用是，取回远程主机某个分支的更新，再与本地的指定分支合并。它的完整格式稍稍有点复杂。

$ **git** **pull** <远程主机名> <远程分支名>:<本地分支名>

比如，取回**origin**主机的**next**分支，与本地的**master**分支合并，需要写成下面这样。

$ **git** **pull** **origin** **next**:**master**

如果远程分支是与当前分支合并，则冒号后面的部分可以省略。

$ **git** **pull** **origin** **next**

上面命令表示，取回**origin**/**next**分支，再与当前分支合并。实质上，这等同于先做**git** **fetch**，再做**git** **merge**。

$ **git** **fetch** **origin**

$ **git** **merge** **origin**/**next**

分支管理

1、创建分支

创建分支很简单：**git** **branch** <分支名>

2、切换分支

**git** **checkout** <分支名>

该语句和上一个语句可以和起来用一个语句表示：**git** **checkout** -**b** <分支名>

创建新分支：**git** **branch** **branchName**

切换到新分支：**git** **checkout** **branchName**

上面两个命令也可以合成为一个命令：**git** **checkout** -**b** **branchName**

3、分支合并

比如，如果要将开发中的分支（**develop**），合并到稳定分支（**master**），

首先切换的**master**分支：**git** **checkout** **master**。

然后执行合并操作：**git** **merge** **develop**。

如果有冲突，会提示你，调用**git** **status**查看冲突文件。

解决冲突，然后调用**git** **add**或**git** **rm**将解决后的文件暂存。

**所有冲突解决后，git commit 提交更改**。

4、分支衍合

分支衍合和分支合并的差别在于，分支衍合不会保留合并的日志，不留痕迹，而 分支合并则会保留合并的日志。

要将开发中的分支（**develop**），衍合到稳定分支（**master**）。

首先切换的**master**分支：**git** **checkout** **master**。

然后执行衍和操作：**git** **rebase** **develop**。

如果有冲突，会提示你，调用**git** **status**查看冲突文件。

解决冲突，然后调用**git** **add**或**git** **rm**将解决后的文件暂存。

所有冲突解决后，**git** **rebase** --**continue** 提交更改。

5、删除分支

【git 删除本地分支】

执行**git** **branch** -**d** <分支名>

如果该分支没有合并到主分支会报错，可以用以下命令强制删除**git** **branch** -**D** <分支名>

【git 删除远程分支】

**git push origin :dev**  (origin 后面有空格)

【指定本地dev分支与远程origin/dev分支的链接】

$ git branch **--set-upstream dev origin/dev**

**1.创建并切换到新分支：**

$ git checkout –b dev

**2.创建远程分支：**

$ git push origin dev

3.关联本地与远程的分支：（查看.git/config文件是否被修改）

$ git branch **--set-upstream dev origin/dev**

Branch dev **set** up **to** track remote branch dev **from** origin.

.git/config文件：

…

[branch "master"]

remote = origin

merge = refs/heads/master

[branch "dev"]

remote = origin

merge = refs/heads/dev

6.分支合并

[**git**](http://lib.csdn.net/base/git)**merge** 用来做分支合并，将其他分支中的内容合并到当前分支中。

7. 查看本地和远程分支 -a。前面带\*号的代表你当前工作目录所处的分支

**$ git branch -a**

撤销

【恢复某个已修改的文件（撤销未提交的修改）】：

**git** **checkout** **file**-**name**

如：**git** **checkout** \*.**java**

撤销所有修改**git** **checkout** .

【还原已提交的修改：**revert】**

还原最近一次提交的修改：$ **git** **revert** **HEAD**

还原指定版本的修改：$ **git** **revert** **commit**-**id**

**git** **checkout** . #本地所有修改的。没有的提交的，都返回到原来的状态

**git** **stash** #把所有没有提交的修改暂存到**stash**里面。可用**git** **stash** **pop**回复。

**git** **reset** --**hard** **HASH** #返回到某个节点，不保留修改。

**git** **reset** --**soft** **HASH** #返回到某个节点。保留修改

git reflog

【撤销一个“已公开”的改变】

场景: 你已经执行了 **git** **push**, 把你的修改发送到了 **GitHub**，现在你意识到这些 **commit** 的其中一个是有问题的，你需要撤销那一个 **commit**.

方法: **git** **revert** <**SHA**>

原理: **git** **revert** 会产生一个新的 **commit**，它和指定 **SHA** 对应的 **commit** 是相反的（或者说是反转的）。如果原先的 **commit** 是“物质”，新的 **commit** 就是“反物质” — 任何从原先的 **commit** 里删除的内容会在新的 **commit** 里被加回去，任何在原先的 **commit** 里加入的内容会在新的 **commit** 里被删除。

这是 **Git** 最安全、最基本的撤销场景，因为它并不会改变历史 — 所以你现在可以 **git** **push** 新的“反转” **commit** 来抵消你错误提交的 **commit**。

【修正最后一个 **commit** 消息】

场景: 你在最后一条 **commit** 消息里有个笔误，已经执行了 **git** **commit** -**m** "**Fxies** **bug** #42"，但在 **git** **push** 之前你意识到消息应该是 “**Fixes** **bug** #42″。

方法: **git** **commit** --**amend** 或 **git** **commit** --**amend** -**m** "**Fixes** **bug** #42"

原理: **git** **commit** --**amend** 会用一个新的 **commit** 更新并替换最近的 **commit** ，这个新的 **commit** 会把任何修改内容和上一个 **commit** 的内容结合起来。如果当前没有提出任何修改，这个操作就只会把上次的 **commit** 消息重写一遍。

【撤销“本地的”修改（无法恢复！）】

场景: 一只猫从键盘上走过，无意中保存了修改，然后破坏了编辑器。不过，你还没有 **commit** 这些修改。你想要恢复被修改文件里的所有内容 — 就像上次 **commit** 的时候一模一样。

方法: **git** **checkout** -- <**bad** **filename**>

原理: **git** **checkout** 会把工作目录里的文件修改到 **Git** 之前记录的某个状态。你可以提供一个你想返回的分支名或特定 **SHA** ，或者在缺省情况下，**Git** 会认为你希望 **checkout** 的是 **HEAD**，当前 **checkout** 分支的最后一次 **commit**。

记住：你用这种方法“撤销”的任何修改真的会完全消失。因为它们从来没有被提交过。

【在撤销“本地修改”之后再恢复】

场景: 你提交了几个 **commit**，然后用 **git** **reset** --**hard** 撤销了这些修改（见上一段），接着你又意识到：你希望还原这些修改！

方法: **git** **reflog** 和 **git** **reset** 或 **git** **checkout**

**或者**

**git** **log**和**git** **reset** --**hard** 3628164(**commit**\_**id**不必写全)

原理: **git** **reflog** 对于恢复项目历史是一个超棒的资源。你可以恢复几乎 任何东西 — 任何你 **commit** 过的东西 — 只要通过 **reflog**。

你可能已经熟悉了 **git** **log** 命令，它会显示 **commit** 的列表。 **git** **reflog** 也是类似的，不过它显示的是一个 **HEAD** 发生改变的时间列表.

【一些注意事项】：

它涉及的只是 **HEAD** 的改变。在你切换分支、用 **git** **commit** 进行提交、以及用 **git** **reset** 撤销 **commit** 时，**HEAD** 会改变，但当你用 **git** **checkout** -- <**bad** **filename**> 撤销时（正如我们在前面讲到的情况），**HEAD** 并不会改变 — 如前所述，这些修改从来没有被提交过，因此 **reflog** 也无法帮助我们恢复它们。

**git** **reflog** 不会永远保持。**Git** 会定期清理那些 “用不到的” 对象。不要指望几个月前的提交还一直躺在那里。

你的 **reflog** 就是你的，只是你的。你不能用 **git** **reflog** 来恢复另一个开发者没有 **push** 过的 **commit**。

查看日志

git log --graph命令可以看到分支合并图。

git log --oneline --graph –decorate可以看到一个仓的直线提交图标。

git log –stat查看提交了哪些文件

git reflog可以查看所有分支的所有操作记录，包括已经被删除的commit记录

对比

本地与远程对比

git diff <local branch> <remote>/<remote branch>

统计文件的改动

git diff --stat local\_branch origin/remote\_branch

覆盖更新（慎用！！）

git fetch --all

git reset --hard origin/master