提交到服务器步骤

git add .

git commit -m 'message'

git fetch

git merge origin/263

git branch

git push origin 263

# 修改本地配置

|  |  |
| --- | --- |
| git config | 配置相关信息. 例如:  git config --global user.name "jinwanlin"  git config --global user.email "jinwanlin123@gmail.com" |

# 创建库

|  |  |
| --- | --- |
| git init | 创建数据库 |

# 分支管理

|  |  |
| --- | --- |
| git branch temp | 创建分支 |
| git branch | 看分支 |
| git branch -d temp | 删除分支 |
| git checkout master | 切换分支 |
| git merge temp | 合并分支, 把目标分支合并到当前分支 |
| git whatchanged | 查看两个分支的变化 |
| git co origin/web2 -b web2 | checkout服务器上的分支（本地没有的） |

|  |  |
| --- | --- |
| git diff | working tree(源代码文件) 和index file(索引)之间的区别 |
| git diff --cached | index file(索引)和仓库之间的区别 |
| git status | 整体改动信息 |

本地创建新分支，并切换到刚创建的分支

|  |  |
| --- | --- |
|  |  |
|  | git checkout tag1  修改分支内文件内容  git checkout -b tag2 //以tag1为源创建分支tag2，修改后的内容在tag2中 |

本地分支提交到远程分支

|  |  |
| --- | --- |
| $ git push origin test:master         // 提交本地test分支作为远程的master分支 $ git push origin test:test              // 提交本地test分支作为远程的test分支  $ git push origin :test              //远程的test将被删除，但是本地还会保存的 |  |
|  |  |

# 从服务器上下载到本地

|  |  |
| --- | --- |
| git clone | 复制一个数据库到指定文件夹 |
| git pull | 从服务器下载数据, 并和本地数据库合并  pull = fetch + merge |
| git fetch | 从服务器上下载数据, 并放到新分支, 不跟自己的数据库合并 |

# push

|  |  |
| --- | --- |
| git push | 提交到本地 |
| git push -u origin master | 提交到远程服务器 |

# 提交到远程服务器

|  |  |
| --- | --- |
| git add | 把想提交的文件制作一个快照 |
| git commit | 将快照里登记的内容永久写入本地git仓库中 |
| git commit -a | 提交所有修改, 省去git add 和git diff和git commit  但无法把新增文件和文件夹加进来 |

# 查看历史

|  |  |
| --- | --- |
| git log | 查看历史版本 |
| git show | 查看版本号对应版本的历史, 如果参数是HEAD则查看最新版本 |
| git log -p filename | 查看某个文件的修改历史 |

# 恢复到以前版本

|  |  |
| --- | --- |
| ght reset --mixed默认参数 | 重置索引内容, 将其定位到指定的项目版本, 而不改变你的工作树中的所有内容, 只是提示你有哪些文件还未更新. |
| ght reset --soft  相当于  ght reset --mixed后git add | 不触动索引位置, 也不改变工作树中的任何内容.  保留你在工作树中的所有更新, 并使之处于待提交状态 |
| git reset --hard | 把整个目录还原到一个版本, 包括所有文件 |
| git checkout -- hello.rb | git 恢复单个文件 |
| git reset 版本号 文件名 |  |

# 发补丁给上级

|  |  |
| --- | --- |
| git diff master temp > patch |  |

# 打补丁

|  |  |
| --- | --- |
| git status | 显示当前的状态  快照(已暂存staged): 作了修改并已放入暂存区域  已修改(modified): 自上次取出后, 作了修改但还没作快照(放到暂存区域)  已提交(committed): git目录中保存着特定的版本, 文件以保存在本地数据库中 |

# 文件管理

|  |  |
| --- | --- |
| git mv filename | 重命名文件或文件夹 |
| git rm filename | 删除文件或文件夹 |

# 其它

|  |  |
| --- | --- |
| git tag | 标定版本号 |
| git help | 查看帮助 |

# 常见日志:

|  |  |
| --- | --- |
| changed but not updated | 修改了, 但还未add |
| changes to be commited | add了, 但还未commited |
| untracked files | 你增加了文件或文件夹 |

# 写开发日志

|  |
| --- |
| 第一行: 少于50字的概括信息, 因为很多版本系统中的email机制会选择第一行作标题  第二行: 必须是空行  第三方: 细致描述开发信息 |

# 查看日志

|  |  |
| --- | --- |
| git log |  |
| git log -p |  |

# 基本守则:

1. git diff --check

一般开发者的角度:

1. 从服务器上克隆数据库到本机上;

2.在本机创建分支, 修改代码;

3. 提交到本机自己创建的分支上;

4.在本机上合并分支

5.新建一个分支, 把服务器上最新版的代码fetch下来, 然后跟自己的主分支合并;

6.生成补丁(patch), 把补丁发送给主开发者.

7.看组开发者的反馈,

解决冲突的方法:

使用pull命令, 解决完冲突之后再向组开发者提交补丁.

git diff的用法

|  |
| --- |
| **这三大部分中：** **working tree：**就是你所工作在的目录，每当你在代码中进行了修改，working tree的状态就改变了。 index file：是索引文件，它是连接working tree和commit的桥梁，每当我们使用git-add命令来登记后，index file的内容就改变了，此时index file就和working tree同步了。 commit：是最后的阶段，只有commit了，我们的代码才真正进入了git仓库。我们使用git-commit就是将index file里的内容提交到commit中。 |
| **总结一下：** git diff：是查看working tree与index file的差别的。 git diff --cached：是查看index file与commit的差别的。 git diff HEAD：是查看working tree和commit的差别的。（你一定没有忘记，HEAD代表的是最近的一次commit的信息） |
| * git diff one\_commit\_hash another\_commit\_hash 看新老两个版本之间的 diff NOTE: 这里的 commit\_hash 就是所谓的“版本号”，其本质是 git 中 **commit** 这种 object 的40位 sha1 哈希。 * git diff filename 或者 git diff dirname/ 查看指定文件或目录的修改 * git diff one\_branch\_name another\_branch\_name 对比两个分支之间的差异 |