Git/RESTful API/命令行

Git

Git 常用命令

- `git init`
- `git clone`
- `git remote add origin ***.git`
- `git push -u origin master`
- 推送到远程仓库的dev分支：`git push origin dev`
- `git log`
- `git log --graph --pretty=oneline --abbrev-commit`
- `git status`
- `git diff`
- `git add *`
- `git commit -m "message"`
- commit之后又改了一个小bug，但是又不想增加一个commit，可以用：`git commit --amend --no-edit`，直接将改动添加到上一次的commit中
- `git push`
- `git pull`
- `touch .gitignore`

Git 标签管理

- 首先切换到需要打标签的分支上，然后使用`git tag v1.0`就可以在当前commit打上v1.0的标签
- `git tag v1.0 commitID` 对特定commit打标签
- 打标签时加上message：`git tag -a <tagname> -m "message"`
- `git tag` 查看所有标签
- `git show [tagname]` 查看标签详细信息
- `git push origin <tagname>`可以推送一个本地标签到远程仓库
- `git push origin --tags`可以推送全部未推送过的本地标签
- `git tag -d <tagname>`可以删除一个本地标签
- `git push origin :refs/tags/<tagname>`可以删除一个远程标签（先从本地删除）

Git 撤销与回滚

- **暂存区**：`git add`之后commit之前存在的区域；**工作区**：`git commit`之后存在的区域；**远程仓库**：`git push`之后；
- 作了修改，但还没`git add`，撤销到上一次提交：`git checkout -f -- filename`；`git checkout -f -- .`
- 作了修改，并且已经`git add`，但还没`git commit`：
  - 先将暂存区的修改撤销：`git reset HEAD filename`/`git reset HEAD`；此时修改只存在于工作区，变为了 "unstaged changes"；
  - 再利用上面的checkout命令从工作区撤销修改
- `git add`之后，作了修改，想丢弃这次修改：`git checkout -f --filename`会回到最近一次`git add`
- 作了修改，并且已经`git commit`了，想撤销这次的修改：
  - `git revert commitID`. 其实，`git revert`可以用来撤销任意一次的修改，不一定要是最近一次
  - `git reset --hard commitID`/`git reset --hard HEAD^`（HEAD表示当前版本，几个^表示倒数第几个版本，倒数第100个版本可以用HEAD~100）；参数`--hard`：强制将暂存区和工作区都同步到指定的版本
  - `git reset`和`git revert`的区别是：reset是用来回滚的，将HEAD的指针指向了想要回滚的版本，作为最新的版本，而后面的版本也都没有了；而revert只是用来撤销某一次更改，对之后的更改并没有影响
  - 然后再用`git push -f`提交到远程仓库

Git 分支管理

- 创建分支: `git branch test`
- 切换分支: `git checkout test`
- 创建并切换分支：`git checkout -b test`
- 将test分支的更改合并到master分支：先在test分支上commit、push，再：`git checkout master`；`git merge test`
- 如果合并时产生冲突：先手动解决冲突，再合并
- 删除分支：`git branch -d test`
- `git stash`
  - 如果当前分支还有任务没有做完，也不想提交，但此时需要切换或者创建其它分支，就可以使用stash将当前分支的所有修改（包括暂存区）先储藏起来；然后就可以切换到其它分支
  - 在其它分支工作完成之后，首先切换回原来的分支，然后使用`git stash list`命令查看
  - 可以使用`git stash apply <stash number>`恢复之前储藏的工作现场，再使用`git stash drop <stash number>`删除掉储藏的内容
  - 也可以直接用`git stash pop`恢复并删除内容
- 如果在其它分支上做了一个修改（比如修复了一个bug，这次修改有一个commitID），想要将这次修改应用到当前分支上，可以使用：`git cherry-pick commitID`，可以复制一个特定的提交到当前分支

RESTful API

REST指Representational State Transfer，可以翻译为“表现层状态转化”

主要思想

- 对网络上的所有资源，都有一个**统一资源标识符** URI(Uniform Resource Identifier)；
- 这些资源可以有多种表现形式，即REST中的“表现层”Representation，比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现。URI只代表资源的实体，不代表它的形式；
- “无状态(Stateless)”思想：服务端不应该保存客户端状态，只需要处理当前的请求，不需了解请求的历史，客户端每一次请求中包含处理该请求所需的一切信息；
- 客户端使用HTTP协议中的 GET/POST/PUT/DELETE 方法对服务器的资源进行操作，即REST中的”状态转化“

设计原则

- URL设计
    - 最好只使用名词，而使用 GET/POST/PUT/DELETE 方法的不同表示不同的操作；比如使用 `POST /user` 代替 `/user/create`
    - GET：获取资源；POST：新建/更新资源；PUT：更新资源；DELETE：删除资源；
    - 对于只支持GET/POST的客户端，使用 `X-HTTP-Method-Override` 属性，覆盖POST方法；
    - 避免多级URL，比如使用 `GET /authors/12?categories=2` 代替 `GET /authors/12/categories/2`；
    - 避免在URI中带上版本号。不同的版本，可以理解成同一种资源的不同表现形式，所以应该采用同一个URI，版本号可以在HTTP请求头信息的Accept字段中进行区分
- 状态码：服务器应该返回尽可能精确的状态码，客户端只需查看状态码，就可以判断出发生了什么情况。见计算机网络部分 -- HTTP请求有哪些常见状态码？
- 服务器回应：在响应中放上其它API的链接，方便用户寻找

Linux 常用命令

参考

- Git教程 - 廖雪峰的官方网站
- RESTful API 最佳实践 - 阮一峰的网络日志
- GitHub - jlevy/the-art-of-command-line: Master the command line, in one page