

使用git在本地创建一个项目的过程

```
$ mkdir ~/hello-world //创建一个项目hello-world
$ cd ~/hello-world //打开这个项目
$ git init //初始化
$ touch README
$ git add README //更新README文件
$ git commit -m 'first commit' //提交更新，并注释信息“first commit”
$ git remote add origin git@github.test:hellotest.git //连接远程github项目
$ git push -u origin master //将本地项目更新到github项目上去
```

git设置关闭自动换行

```
$ git config --global core.autocrlf false
```

为了保证文件的换行符是以安全的方法，避免windows与unix的换行符混用的情况，最好也加上这么一句

```
$ git config --global core.safecrlf true
```

git tag 使用

```
git tag # 列出当前仓库的所有标签
git tag -l 'v0.1.*' # 搜索符合当前模式的标签
git tag v0.2.1-light # 创建轻量标签
git tag -a v0.2.1 -m '0.2.1版本' # 创建附注标签
git checkout [tagname] # 切换到标签
git show v0.2.1 # 查看标签版本信息
git tag -d v0.2.1 # 删除标签
git tag -a v0.2.1 9fbc3d0 # 补打标签
git push origin v0.1.2 # 将v0.1.2标签提交到git服务器
git push origin --tags # 将本地所有标签一次性提交到git服务器
git tag # 查看当前分支下的标签
```

git pull问题

```
You asked me to pull without telling me which branch you
want to merge with, and 'branch.content_api_zhangxu.merge' in
your configuration file does not tell me, either. Please
specify which branch you want to use on the command line and
try again (e.g. 'git pull <repository> <refspec>').
See git-pull(1) for details.
If you often merge with the same branch, you may want to
use something like the following in your configuration file:
[branch "content_api_zhangxu"]
```

```
remote = <nickname>
merge = <remote-ref>
[remote "<nickname>"]
url = <url>
fetch = <refspec>See git-config(1) for details.
```

```
git pull origin new_branch
```

怎样遍历移除项目中的所有 .pyc 文件

```
sudo find /tmp -name "*.pyc" | xargs rm -rf
```

替换/tmp目录为工作目录

```
git rm *.pyc
```

这个用着也可以

避免再次误提交，在项目新建.gitignore文件，输入*.pyc过滤文件

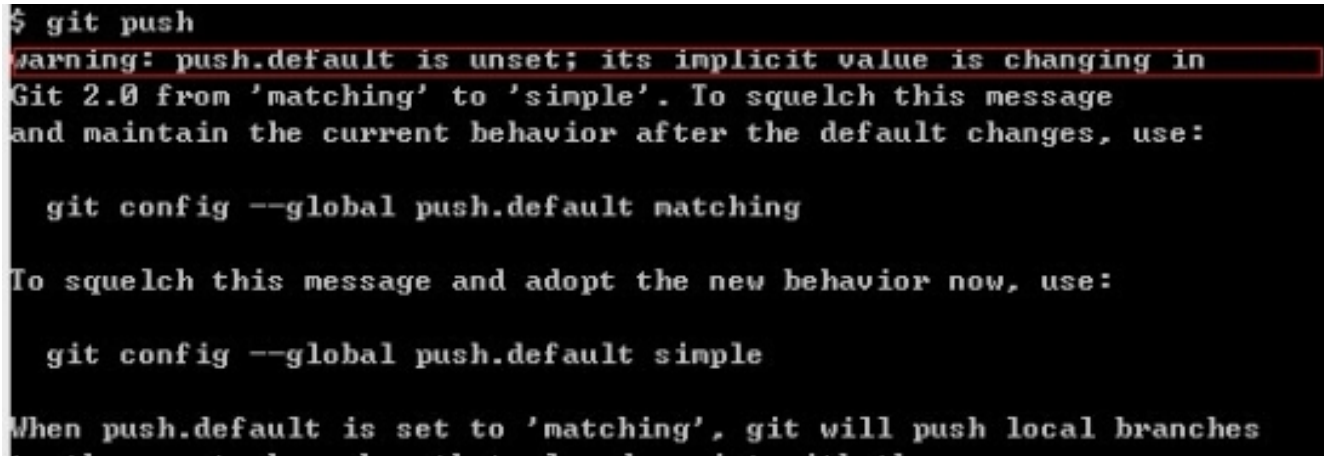
git变更项目地址

```
git remote set-url origin git@192.168.6.70:res_dev_group/test.git
git remote -v
```

查看某个文件的修改历史

```
git log --pretty=oneline 文件名 # 显示修改历史
git show 356f6def9d3fb7f3b9032ff5aa4b9110d4cca87e # 查看更改
```

git push 时报错 warning: push.default is unset



```
$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote even when the push is not a fast-forward.
```

‘matching’ 参数是 Git 1.x 的默认行为，其意是如果你执行 git push 但没有指定分支，它将 push 所有你本地的分支到远程仓库中对应匹配的分支。而 Git 2.x 默认的是 simple，意味着执行 git push 没有指定分支时，只有当前分支会被 push 到你使用 git pull 获取的代码。根据提示，修改git push的行为:

```
git config -global push.default matching
```

再次执行git push 得到解决

git submodule的使用拉子项目代码

开发过程中，经常会有一些通用的部分希望抽取出来做成一个公共库来提供给别的工程来使用，而公共代码库的版本管理是个麻烦的事情。今天无意中发现了git的git submodule命令，之前的问题迎刃而解了。

添加

为当前工程添加submodule，命令如下：

```
git submodule add 仓库地址 路径
```

其中，仓库地址是指子模块仓库地址，路径指将子模块放置在当前工程下的路径。
注意：路径不能以 / 结尾（会造成修改不生效）、不能是现有工程已有的目录（不能顺利Clone）

命令执行完成，会在当前工程根路径下生成一个名为“.gitmodules”的文件，其中记录了子模块的信息。添加完成以后，再将子模块所在的文件夹添加到工程中即可。

删除

submodule的删除稍微麻烦点：首先，要在“.gitmodules”文件中删除相应配置信息。然后，执行git rm –cached命令将子模块所在的文件从git中删除。

下载的工程带有submodule

当使用git clone下来的工程中带有submodule时，初始的时候，submodule的内容并不会自动下载下来的，此时，只需执行如下命令：

```
git submodule update --init --recursive
```

即可将子模块内容下载下来后工程才不会缺少相应的文件。

一些错误

“pathspec ‘branch’ did not match any file(s) known to git.”错误

```
git checkout master
git pull
git checkout new_branch
```

使用git提交比较大的文件的时候可能会出现这个错误

```
error: RPC failed; result=22, HTTP code = 411
fatal: The remote end hung up unexpectedly
fatal: The remote end hung up unexpectedly
Everything up-to-date
```

这样的话首先改一下git的传输字节限制

```
git config http.postBuffer 524288000
```

然后这时候在传输或许会出现另一个错误

```
error: RPC failed; result=22, HTTP code = 413
fatal: The remote end hung up unexpectedly
fatal: The remote end hung up unexpectedly
Everything up-to-date
```

这两个错误看上去相似，一个是411，一个是413

下面这个错误添加一下密钥就可以了

首先key-keygen 生成密钥

然后把生成的密钥复制到git中自己的账号下的相应位置

```
git push ssh://192.168.64.250/eccp.git branch
```

git add文件取消

在git的一般使用中，如果发现错误的将不想提交的文件add进入index之后，想回退取消，则可以使用命令：

```
git reset HEAD <file>...
```

同时git add完毕之后，git也会做相应的提示。

[git reset \(回退add操作 \)](#)

git删除文件

删除文件跟踪并且删除文件系统中的文件file1

```
git rm file1
```

提交刚才的删除动作，之后git不再管理该文件

```
git commit
```

删除文件跟踪但不删除文件系统中的文件

```
file1git rm --cached file1
```

提交刚才的删除动作，之后git不再管理该文件，但是文件系统中还是有file1

```
git commit
```

版本回退

版本回退用于线上系统出现问题后恢复旧版本的操作，回退到的版本。

```
git reset --hard 248cba8e77231601d1189e3576dc096c8986ae51
```

回退的是所有文件，如果后悔回退可以git pull就可以了。

历史版本对比

查看日志git log

查看某一历史版本的提交内容，这里能看到版本的详细修改代码。

```
git show 4ebd4bbc3ed321d01484a4ed206f18ce2ebde5ca
```

对比不同版本

```
git diff c0f28a2ec490236caa13dec0e8ea826583b49b7a 2e476412c34a63b213b735e5a6d90cd05b014c33
```

Git常用命令和场景(三)--版本差别查看

分支的意义与管理

创建分支可以避免提交代码后对主分支的影响，同时也使你有了相对独立的开发环境。分支具有很重要的意义。

创建并切换分支，提交代码后才能在其它机器拉分支代码

```
git checkout -b new_branch
```

查看当前分支

```
git branch
```

切换到master分支

```
git checkout master
```

合并分支到当前分支，合并分支的操作是从new_branch合并到master分支，当前环境在master分支。

```
git merge new_branch
```

删除分支

```
git branch -d new_branch
```

git冲突文件编辑

冲突文件冲突的地方如下面这样

```
a123
<<<<<<< HEAD
b789
=====
b45678910>>>>>>> 6853e5ff961e684d3a6c02d4d06183b5ff330dccc
```

冲突标记<<<<<<<（7个<）与=====之间的内容是我的修改，=====与>>>>>>>之间的内容是别人的修改。
此时，还没有任何其它垃圾文件产生。
你需要把代码合并好后重新走一遍代码提交流程就好了。

不顺利的代码提交流程

在git push后出现错误可能是因为其他人提交了代码，而使你的本地代码库版本不是最新。
这时你需要先git pull代码后，检查是否有文件冲突。
没有文件冲突的话需要重新走一遍代码提交流程add —> commit —> push。
解决文件冲突在后面说。

git顺利的提交代码流程

查看修改的文件

```
git status
```

为了谨慎检查一下代码

```
git diff
```

添加修改的文件，新加的文件也是直接add就好了

```
git add dirname1/filename1.py dirname2/filename2.py
```

添加修改的日志

```
git commit -m "fixed:修改了上传文件的逻辑"
```

提交代码git push，如果提交失败的可能原因是本地代码库版本不是最新。

理解github的pull request

有一个仓库，叫Repo A。你如果要往里贡献代码，首先要Fork这个Repo，于是在你的Github账号下有了一个Repo A2,。然后你在这个A2下工作，commit，push等。然后你希望原始仓库

Repo A合并你的工作，你可以在Github上发起一个Pull Request，意思是请求Repo A的所有者从你的A2合并分支。如果被审核通过并正式合并，这样你就为项目A做贡献了。

github的pull request是指什么意思

创建和使用git ssh key

首先设置git的user name和email

```
git config --global user.name "xxx"
git config --global user.email "xxx@gmail.com"
```

查看git配置

```
git config --list
```

然后生成SHH密匙

查看是否已经有了ssh密钥：cd ~/.ssh
如果没有密钥则不会有此文件夹，有则备份删除
生成密钥

```
ssh-keygen -t rsa -C "noogel666@gmail.com"
```

按3个回车，密码为空这里一般不使用密钥。
最后得到了两个文件：id_rsa和id_rsa.pub
注意：密匙生成就不要改了，如果已经生成到~/.ssh文件夹下去找。