<https://www.yiibai.com/git/git_pull.html>

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375840038939c291467cc7c747b1810aab2fb8863508000>

<https://www.cnblogs.com/lianghe01/p/5806518.html>

<https://blog.csdn.net/qq_35161540/article/details/74626000>

<https://blog.csdn.net/runfan1014/article/details/54584448>

git图表展示：



git状态显示：

**本地仓库：**

git init    #初始化当前文件夹为本地仓库

**远程仓库：**

git remote     #查看远程仓库

git remote -v     #查看远程仓库详细信息

**关联本地仓库到远程仓库：**

git remote add origin [git@git.yonyou.com](mailto:git@git.yonyou.com):iuap\_ap/iuap\_cloud\_basedoc.git

这个命令的意思是，我给远程仓库（就是[git@git.yonyou.com](mailto:git@git.yonyou.com):iuap\_ap/iuap\_cloud\_basedoc.git）起了个别名，叫origin（其实叫别的名字也可以，你只要知道这是别名就行了，以后就用别名来操作远程库）。

**克隆分支**

git clone [git@git.yonyou.com](mailto:git@git.yonyou.com):iuap\_ap/iuap\_cloud\_bd.git                   #克隆工程的默认分支

git clone -b develop [git@git.yonyou.com](mailto:git@git.yonyou.com):iuap\_ap/iuap\_cloud\_bd.git        #克隆工程的指定分支

**新建、查看和转换分支：**

git branch --all     #查询所有本地和远程分支

git branch     #查看本地分支

git branch -r    #查看远程分支

git branch -vv    #查看本地分支以及追踪的远程连接

git branch dev    #创建本地dev分支

git checkout -b dev     创建并转到dev分支

git checkout dev    #本地当前分支转换到dev分支

git checkout -f dev    #强制转换到dev分支

**手动建立当前分支和远程分支的追踪关系：**

git branch --set-upstream master origin/next

上面的命令是指master分支追踪origin/next分支

**注意：如果没有手动建立追踪关系，则所有的默认追踪关系为名字一致的。如果远程不存在则自动创建。**

**pull拉取代码：**

git pull的语法为：git pull <远程主机名> <远程分支名>:<本地分支名>

描述：将远程存储库中的更改合并到当前分支中。

git pull origin dev    #从远程分支dev分支拉取代码并合并到到本地当前分支(省略本地分支，默认为当前分支)

git pull origin dev:dev    #从dev分支拉取代码合并到本地dev分支

**提交并push代码：**

git push的语法为：git push <远程主机名> <本地分支>:<远程分支名>

git push origin dev    #提交dev分支代码到与之追踪的远程分支（即dev远程分支）（省略了远程分支，如果远程没有dev分支，则创建dev远程分支，名字自动保持一致）

git push origin dev:dev    #强制提交dev分支代码到远程dev分支

实例：

git push origin master

该语法表示把本地分支名push到远程主机名的远程分支。我们可以看到刚才执行的语句并没有远程分支名（被省略了），如果省略远程分支名，则表示将本地分支推送到与之存在”追踪关系”的远程分支(通常两者同名)，如果该远程分支不存在，则会被新建。所以刚才执行的语句的意思是，把本地的master分支推送到远程主机的master分支。

关于git push命令以后还需要注意的地方：

a.如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支。例如：

$ git push origin :master

该命令等同于如下指令：$ git push origin --delete master，表示删除远程主机的master分支。

b.git push 有时候本地分支和远程分支名都可以省略，有时候连主机名也可以省略，这个现在就不细说了。

c.如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机，这样后面就可以不加任何参数使用git push:

$ git push -u origin master

**合并分支（在本地分支进行合并然后push到远程分支）：**

git merge dev     #把本地dev分支合并到当前分支

git merge origin/master     #将远程origin/master分支合并到当前本地分支

实例：

git checkout dev    #切换到dev分支进行开发

#开发功能完成后

git checkout master    #切换到master分支

git merge dev    #把dev分支合并到当前master分支

git push origin master     #提交主分支到远程master

git checkout dev    #切换到dev远程分支

git push origin dev    #提交到远程分支dev

**合并分支冲突后撤销合并**

当合并分支时遇到错误或者冲突，分支旁边会多出“|MERGING”这个东西

所以需要先取消这次合并，使用“git merge --abort”命令

git merge --abort

**删除分支：**

git push origin :master    #删除远程分支master

等价于：git push origin --delete master

删除本地分支：

git checkout master     #切换到master分支

git branch -d dev    #删除本地dev分支

**删除文件**

**方法一、直接删除**

git rm <file\_path>

git commit -m 'delete'

git push

**方法二、间接删除**

首先手动把本地文件删了(git status后会发现在删除的文件变红)

git add <file\_path>

git commit -m 'delete'

git push

**git rm与git rm --cached的区别**

当我们需要删除暂存区或分支上的文件，同时工作区也不需要这个文件了，可以使用

git rm <file\_path>

git commit -m 'delete somefile'

git push

当我们需要删除暂存区或分支上的文件，但本地又需要使用，只是不希望这个文件被版本控制，可以使用

git rm --cached <file\_path>

git commit -m 'delete remote somefile'

git push

**查看工作空间的状态**

git status

#红色代表本地修改但未添加到暂存区的代码（工作区）

    git checkout <file>    #可以放弃/撤销指定的修改/删除的文件

    git checkout \*            #可以放弃/撤销所有的修改/删除的文件

    git add <file>            #可以添加指定修改/删除的文件到暂存区

    git add \*                    #可以添加所有修改/删除的文件到暂存区

#绿色代表已经添加到暂存区但未提交的代码（暂存区）

    git reset HEAD <file> #可以将暂存区中指定的文件退回

    git reset HEAD \*          #可以将暂存区所有的文件退回

    git commit -'信息'    # 将暂存区里的改动给提交到本地的版本库

**暂存命令stash使用**

git stash    #将工作现场存储起来

git stash apply    #恢复工作现场

git stash stop    #删除存储的内容

git stash pop    #恢复工作现场并删除存储内容

git stash list    #查看工作现场列表

git stash apply stash@{0}    #恢复指定的工作现场

git stash clear        #清空暂存栈

**查看日志**

git log <无参数>   #显示出所有的日志，按向下键查看更多，按q键退出日志查看

git log --oneline    #每条日志为一行的显示所有的日志，简洁显示

git log --pretty=raw    #显示每条日志更多的信息

git log -[length]    #指定显示多少条日志

git log --skip=[length]    #跳过前几条日志

git log --graph    #显示日志时，绘制提交的线索

git log --author [yourname]     #通过作者搜索显示日志

git log --grep [keywords]    #可以通过提交的commit描述中搜索日志，

git log --after="2014-7-1"    #显示2014年7月1日（包括）之后commit的日志

git log --before="2014-7-7"    #显示2014年7月7日（包括）之前commit的日志

**恢复版本**

git reset --hard 560378461832b872cfc60bc0e59e442724c7cd01    #回退到560378461832b872cfc60bc0e59e442724c7cd01版本

git reset --hard HEAD^                                        #回退到上一个版本（HEAD表示当前版本，也就是最新提交的版本）

以下有疑问：

git revert HEAD                                               #撤销前一次commit

git revert HEAD^                                               #撤销前前一次commit

git revert

git reset 与git revert的区别

git reset是指针向前移

git revert是重新一次commit，但作用是与指定commit作用相反

git push -f origin release   强制推送

**比较版本之间的差异**

git diff HEAD     #与上次commit之间的差别（爸爸）

git diff HEAD^    #与上上次（爷爷）

git diff HEAD^^    #与上上上次（曾祖父）

git diff HEAD~5        #与前面第五次commit（好吧，祖先吧）

git diff HEAD^..HEAD    #中间是两个点比较（爸爸）和（爷爷）的差别

git diff f5fdjsalfjdskaf..4fdklsajfdksaf    #比较 两个不同hash值记录之间的不同

git diff master dev    #比较 分支之间的不同

**做备份分支**

git branch back\_test            #创建一个本地分支

git push origin back\_test        #将分支提交到远程分支（会自动创建一个back\_test远程分支）

git fetch



