# Git

## 全局配置

* **git config --global user.name crperlin #git的用户名**
* **git config --global user.email crper@outlook.com #git的登录账号**
* **git config --global core.editor vim #设置默认编辑器**
* **git config --global merge.tool vimdiff #设置默认的对比文件差异工具**
* **git config --global color.status auto #显示颜色信息**
* **git config --global color.branch auto #显示颜色信息**
* **git config --global color.interactive auto #显示颜色信息**
* **git config --global color.diff auto #显示颜色信息**
* **git config --global push.default simple #简化提交**
* **git config --list#查看配置的信息**
* **git help config#获取帮助信息**

## 配置SSHKEY

配置这货的好处就是可以省去提交的时候每次都输入账号密码；减少重复工作！

不管是github还是一些基于gitlab的托管社区，配置这个都是大同小异，在个人账户那里找到SSH-KEYGEN

ssh-keygen -t rsa -C crper@outlook.com #生成密钥，也可以通过图形工具生成，看个人喜好

ssh -T git@github.com #测试链接github是否成功，其他社区域名不同罢了

## 初始化本地仓库/提交/链接远程仓库

git init #初始化

git status #获取状态,很实用的一个功能，对文件做了操作都能看到各种提示信息

git add <file> # . 或 \* 代表全部添加

git commit -m "注释文本" #添加注释

git remote add origin git@github.com:crperlin/S-NEXT.git #本地链接远程仓库

git push -u origin master #推送到主分支

## 克隆远程仓库

一般的git托管社区都提供两种，一种是基于https，一种是基于ssh

前者需要账号密码提交，后者可以配置SSH-KEYGEN

git clone git@github.com:crperlin/S-NEXT.git #克隆到本地，仓库名就是文件夹的名字

git clone xx.git <自定义文件夹名字> #不需要带括号，只是区别

## git仓库操作

git add \*#跟踪新文件

git rm -f \* #强制删除所有文件

git rm --cached \* #取消跟踪

git mv file\_from file\_to #重命名跟踪文件，与linux一致

git log#查看提交记录

git commit#提交更新

git commit -m "注释文本" #添加注释

git commit -a #跳过使用暂存区域，把所有已经跟踪过的文件暂存起来一并提交

git commit --amend #修改最后一次提交

git reset HEAD \*#取消已经暂存的文件

git checkout -- file#取消对文件的修改（从暂存区去除file）

git checkout branch|tag|commit -- file\_name#从仓库取出file覆盖当前分支

git checkout -- .#从暂存区去除文件覆盖工作区

## 分支管理

git branch #列出本地分支

git branch -r #列出远端分支

git branch -a #列出所有分支

git branch -v #查看各个分支最后一个提交对象的信息

git branch --merge #查看已经合并到当前分支的分支

git branch --no-merge #查看为合并到当前分支的分支

git branch test #新建test分支

git checkout test #切换到test分支

git checkout -b test #新建+切换到test分支

git checkout -b test dev#基于dev新建test分支，并切换

git branch -d test #删除test分支

git branch -D test #强制删除test分支

git merge test #将test分支合并到当前分支

git rebase master #将master分之上超前的提交，变基到当前分支

## 分支数据推送更新

git fetch origin branch #获取远端上指定分支

git merge origin branch #合并远端上指定分支

git push origin branch #推送到远端上指定分支

git push origin localbranch:serverbranch #推送到远端上指定分支

git checkout -b test origin/dev #基于远端dev新建test分支

git push origin :server#删除远端分支[推送空分支，目前等同于删除]

## 标签版本管理

git tag#列出现有标签

git tag v1.0.0 #新建标签

git tag -a v0.1 -m '注释文本' #新建带注释标签

git tag v2.0 9fceb02 #给指定的指向添加版本

git show <version> #显示指定版本的详细信息

git checkout tagname#切换到标签

git push origin v1.5#推送分支到源上

git push origin --tags#一次性推送所有分支

git tag -d v0.1#删除标签

git push origin :refs/tags/v0.1#删除远程标签

## alias【别名系统，基本和linux一致】

### 一. 到配置文件配置，最傻瓜化的写法，等值赋予

[alias]

logs = log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit

st = status

ci = commit

br = branch

co = checkout

df = diff

lg = log -p

### 二. 命令行追加

git config --global alias.co checkout

git config --global alias.ci commit

git config --global alias.br branch

...

## 文本搜索

git grep '查询文本' #对全局的字符串查询

git grep '查询文本' v1.0.0 #针对版本的字符串查询

## 文件修改详情跟踪

git blame <file> #谁，在什么时间，修改了文件的什么内容

## 撤销

#放弃工作目录下的所有修改：

git reset --hard HEAD

#移除缓存区的所有文件（i.e. 撤销上次git add）:

git reset HEAD

#放弃某个文件的所有本地修改：

git checkout HEAD <file>

#重置一个提交（通过创建一个截然不同的新提交）

git revert <commit>

#将HEAD重置到指定的版本，并抛弃该版本之后的所有修改：

git reset --hard <commit>

#将HEAD重置到上一次提交的版本，并将之后的修改标记为未添加到缓存区的修改：

git reset <commit>

#将HEAD重置到上一次提交的版本，并保留未提交的本地修改：

git reset --keep <commit>