一、正则表达式

二、markdown

二、环境变量

Git

最新使用：

①最开始，先初始化本地库，并设置邮箱和用户名(随便设，只作为标识，若不设，则无法提交)

git init

git config –global user.email “…”

git config –global user.name “…”

注意1：

github将master改为了main，所以本地库也改成main

git branch –m main

注意2：

github修改了push的提交方式，只能用ssh公钥：

（1）先在自己电脑上创建ssh公钥，复制到github的设置中（百度）

（2）git remote add origin [git@github.com:ligengxin549/](mailto:git@github.com:ligengxin549/)仓库名.git

②以后的每一次修改：

git add . #添加该目录所有文件到暂存区

git commiy –m “描述” #添加到本地库

git pull origin main #建议在push之前先pull下来，得到别人的最新修改

git push origin main

注意1：

git add的警告warning: LF will be replaced by CRLF

原因：Linux用LF代表换行和回车，而windows的LF是换行，CRLF是回车，若不处理将会将LF转为CRLF，跨平台时就会出错

解决：git config --global core.autocrlf false

注意2：

git add 无法添加空文件夹，若想添加这个文件夹，只能在这个空文件夹放一个任意文件。

注意3：

git clone 可以拷贝github上的项目

pull与clone区别：

clone是没有本地库的情况下，完全克隆远程库的内容

oull是在有本地库的情况下，pull下来保存到本地库，但不覆盖本来内容。

注意4：

Git clone 从github拷贝vue项目后，需要执行npm install

若有不需要保存到本地库的文件，可以在.gitignore文件中声明

如想忽略node\_modules文件夹

/node\_modules

空的文件夹不会被提交到本地库，若想保存该空文件夹，可以在里面放一个隐藏文件占位，命名一般为 .gitkeep

//git结构

工作区：写代码的地方

暂存区：暂存代码，准备提交

本地库：保存提交的代码

远程库：github托管代码

//初始化

git init 初始化本地库，生成.git文件夹（隐藏）

ls -a 查看隐藏文件

//设置签名,签名只是作为说明信息，没有其他功能，有了签名才能提交

#项目级别签名，表明这个项目

git config user.name ...

git config user.email ...

git status 查看工作区i，暂存区状态该

cat .git/config 查看项目签名

#系统用户级别签名，表明这个系统用户

git config --global user.name ...

git config --global user.eamil ...

cd ~ 在家目录中找.git文件中的config

//提交

git status 查看工作区，暂存区状态,红字表示工作区有文件可提交到暂存区，绿字表示暂存区有文件可提交到本地库

vim xxx.txt 新建vim文件

（i进入编辑模式，esc退出模式，:wq保存退出）

cat ... 直接打印文件内容

git add ... 提交到暂存区

git commit ... 提交到本地库（会进入vim输入说明信息）

git commit -m "..." ... 直接写说明信息提交（不进入vim）

//查看历史记录，前进到新版本，后退到老版本

查看历史记录,只会显示当前以及更后面的版本

#以下依次显示的更简洁

git log

git log --pretty=oneline

git log --oneline

#最常用

git reflog

HEAD是指向当前版本的指针，{n}表示从当前版本到此版本的移动步数，16进制数据是跳转版本的索引

//前进，后退版本

git reset --hard 16进制索引 //直接到指定版本呢

git reset --hard HEAD^ //后瑞n个^步数的版本

git reset --hard HEAD~n //后退n个版本

--hard 移动本地库HEAD，并将暂存区，工作区替换为相应版本

--mixed 移动本地库HEAD，并将暂存区替换版本，工作区不变

--soft 移动本地库HEAD，暂存区，工作区不变

#sort,mixed后git status的显示

soft本地库移动了，暂存区不变，但是暂存区相对于本地库是移动了的，所以会显示绿字

mixed是工作区相对移动了，所以显示红字

#提交到本地库后，工作区删除的文件可找回

rm ... 删除工作区的文件

再次提交到暂存区-本地库（只暂不本地也行）

--hard将指针移动到相应版本就可恢复

//有时候修改了工作区，想和原先进行比较

-表示删除的内容，+表示新增的内容

git diff ... 或 git diff 与暂存区比较

git diff HEAD ... 或 git diff HEAD 与本地库比较

不带文件名是将工作区所有文件都比较

本地库比较可以指定版本 如HEAD^

//分支

在不同的分支中开发，互不影响，各分支开发完后与主干合并。若有bug则新建一个hot\_fix临时分支测试。

git branch -v 查看所有分支和当前所在分支

git branch name 复制master的形式创建分支

git checkout -b xxx 直接创建并切换到xxx分支

git checkout name 切换分支

git merge name 将name分支和主干合并（必须在master分支下操作），将分支修改的内容更新到主干

git合并分支，合并的是该分支的本地库，所以main合并分支后不需要add和commit就能直接push

若push时远程库没有该分支，则远程库会自动创建该分支

开发完成该分支后，应该push该分支，而后合并到主干，再push主干

分支冲突：

当合并的两个分支，都对同一个文件的同一个位置进行了修改，此时git就不知道如何合并，发生冲突

解决：

1、编辑文件，修改到梦遗的程度

2、git add ... 表明已解决冲突

3、git commit -m "说明" 不带文件名，合并成功。

//哈希

明文通过加密算法变密文，git使用的加密算法是哈希

只要使用相同的哈希算法：

不管明文有多大，加密后的都是相同的固定长度

任何时候，算法和明文确定，密文就确定

明文有细微变化，密文可能有很大变化

哈希算法不可逆