**git 基础操作**

git bash 基本操作：

clear 清屏

ll 删除记录查询

history 查看历史

help 帮助

rm 删除

exit 退出

# 注释

建仓

1 mkdir <name>

2 cd <name>

3 git--bare init

------------------------

加入索引 git add .

检查状态 git status （ modified：☞修改未提交，）

提交更新 git commit -m （提交更新后，将检查不到status，

因为这之前（上一版本后）所有

的status更新为一个新的版本提

交至log中）

查询记录 git log

忽略处理：

.gitignore 忽略档案

<文件名>.html 忽略全部HTML档案

forder/ 忽略 folder

<说白了，按照路径和文件名去在 .gitignore文档中对需要忽略更新的路径或文件

进行保护即可>

# 取消索引<git add 之后，git commit之前>

全部档案取消索引

git reset HEAD

取消单一档案

git reset HEAD <档案名>

# 还原档案

恢复单一档案到最新 commit状态

git checkout <档案名>

恢复目录与索引，与最后一次commit保持一致

git reset --hard

pwd linux(github) 绝对路径获取命令

git config 配置文件的优先级

--local 最高级别

--global 次级

--system 优先级最低

-----------------------

查看操作：

ll|less 查看历史操作记录

ls -lA|less 查看包含隐藏文件的历史记录

<进入上述状态下的vim界面后，退出时，首先esc键退出输入状态，然后shift+“.”,

最后按q<no saving>退出 or 按wq<saving written>保存退出>

git commit “<operation\_name>” <text\_name>

<进入vim界面后，仍然esc键退出编辑，再切换到大写输入，连续两次输入Z，

即可退出>

补充： git log --pretty=oneline 用于单行显示每一条记录，字面意思即可理解

git log --oneline 同样单行显示，但是哈希索引仅显示前7位（0x进制）

git reflog 比上一条多了commit说明 和 HEAD@{<返回到该版本所

需移动步数>}

查看txt类文件内容：cat <文件名.txt>

(.docx 类的摁键无法查看会报错，考虑utf-8转码格式？)

历史版本操作git reset：

本质：类似链表的head指针移动过程，指针始终默认指向最新版本哈希值

三种方法：

I 基于索引值【推荐】

git reset --hard <索引前7位>

II 基于 ^ 符号

git reset --hard ^^^ 后退三个版本

III 基于 ~ 符号

git reset --hard~n n为后退版本数

<II和III 只能后退>

三参数 --hard –soft--mixed 比较：

--soft 仅在本地库移动HEAD

--mixed 本地库移动HEAD指针，重置暂存区

--hard 本地库移动HEAD，重置暂存区，重置工作区

关于删除找回：

前提： 删除前，文件存在时的状态提交到本地库。

操作：git reset --hard [指针位置]

删除操作提交到了本地库：指针位置指向历史记录

删除操作未提交到本地库：指针位置指向HEAD

比较文件差异：

git diff [doc\_name] 将工作区中的文件和暂存区文件进行比较

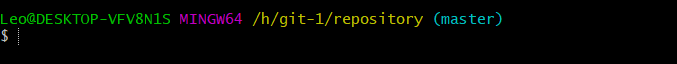
git diff [local\_repo\_load] [doc\_name] 将工作去中文件与本地库历史记录比较

分支管理：

分支：

版本控制过程中，使用多条线同时推进多个任务。

<每当进入仓库时，可以看到文件绝对路径后有一个(master)，其实就是主干分支的意思>



可以同时并行推进多个功能的开发，提高开发效率

各分支开发过程中开发失败不会对其他分支产生任何影响，失败分支删除重开即可。

创建新分支的操作是以复制主干的操作实现的

<

tips:

热修复与冷修复：

属于修复过程分支

>

分支操作：

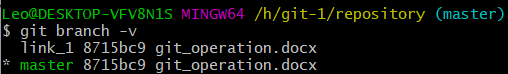
i 创建新分支

git branch <branch\_name>

<创建新分支无需保证原分支的工作树是干净的>

ii 查看分支：

git branch -v



<查看时，前面标\*的为当前所在分支>

iii 切换分支

git checkout <分支名>

iv 合并分支

step1： 要合并分支，必须要切换到接受合并的分支<主干分支>上。

step2： git merge <有新内容的分支>

v 合并过程产生冲突的原因与解决办法

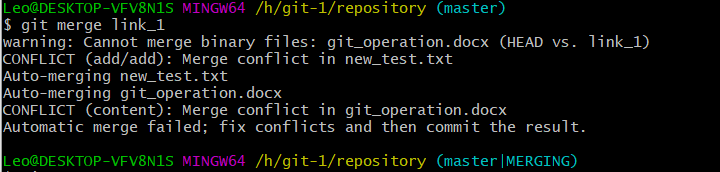
1 产生的原因

当两个分支同步推进时，这时进行分支操作，git本身不能判断究竟时谁向谁进行

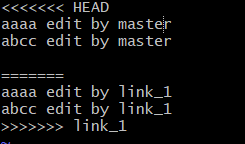
合并，因此需要进行手动的合并.

冲突的实例<master中的new\_test.txt和link\_1中的new\_test.txt具有相同的修改内

容，此时无法进行直接的合并，转为手动merge>：



此时打开当前branch下的该文件会发现内容有所改变，这是冲突的实际结果



HEAD到后面的“======”处，是当前分支的内容

“======”到分支名处，是另一分支的内容

2 解决冲突

直接vim进入后，与协作者进行协商并决定如何完成。

修改后查看状态会出现You have unmerged paths.等字样，

此时再次add到暂存区内，查看状态会出现All conflicts fixed but you are still merging.

用git commit -m “日志名”即可

4 远程库的相关操作：

git remote -v 查看远程库信息

git remote add <创建名> 新建远程库链接信息

推送：git push <创建名> <branch\_name>

正确步骤：

git init //初始化仓库

git add .(文件name) //添加文件到本地仓库

git commit -m “first commit” //添加文件描述信息

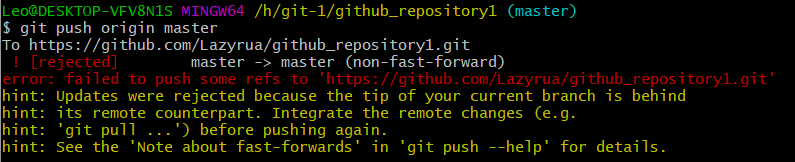
git remote add origin + 远程仓库地址 //链接远程仓库，创建主分支

git pull origin master // 把本地仓库的变化连接到远程仓库主分支

git push -u origin master //把本地仓库的文件推送到远程仓库

Tip1：

出现如下截图错误：



解决办法：

使用强制命令git pull origin master --allow-unrelated-histories

把两段不相干的分支进行强行合；这里可能会进入一个编辑页面，只需要先ESC，然后敲 q! 即可， 或者根据它的页面提示如何操作可退出编辑页面即可

之后在进行push：git push -u origin master