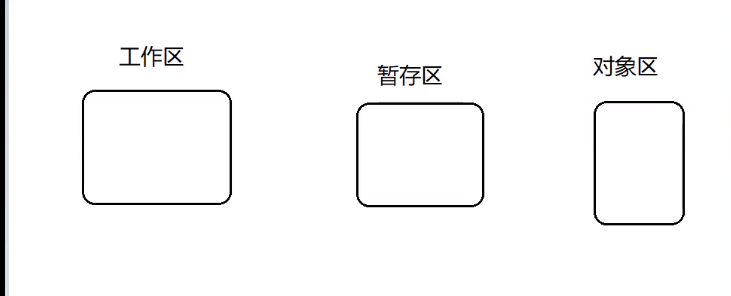
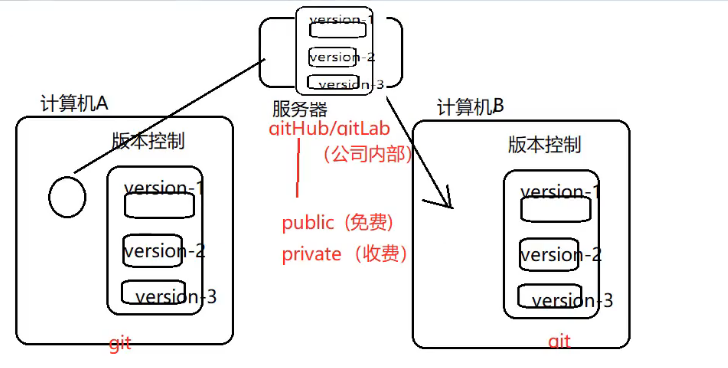
GIT





三种状态：

1. 已修改:modified
2. 已暂存:staged
3. 已提交:commited

命令

初始化：git init （初始化仓库）

清屏：Ctrl+l clear

touch hello.txt 创建文件

git add . 添加到暂存区

git commit -m ‘desc’: 提交

vi 退出：shift +zz

git rm --cached ‘文件’: 暂存区--工作区

git log 查看提交日志

git log --pretty=online （日志简略版）

git log --pretty=format:’%h - %an, %ar : %s’

commit ： 91765d2e16918df96cafa2ea1335a19268cfc58d

sha1计算的结果 sha1, md5、加密算法、随机数、用于区分是哪一次提交（并且不重复）

分布式id生成器

ctrl + a ctrl + e 调到头和尾

ctrl + c 放弃输入

echo ‘world’ > hello.txt 输出字符串到hello.txt文件夹

1、git config --global (基本不用，给整个计算机一次性设置)

2、git config --system （推荐，给当前用户一次性设置）

3、git config --local （给当前项目一次性设置）

3》2》1 优先级

设置 用户名 邮箱

git config --local user.name ‘andy’

git config --local user.email [‘andy@email.com’](mailto:‘andy@email.com’)

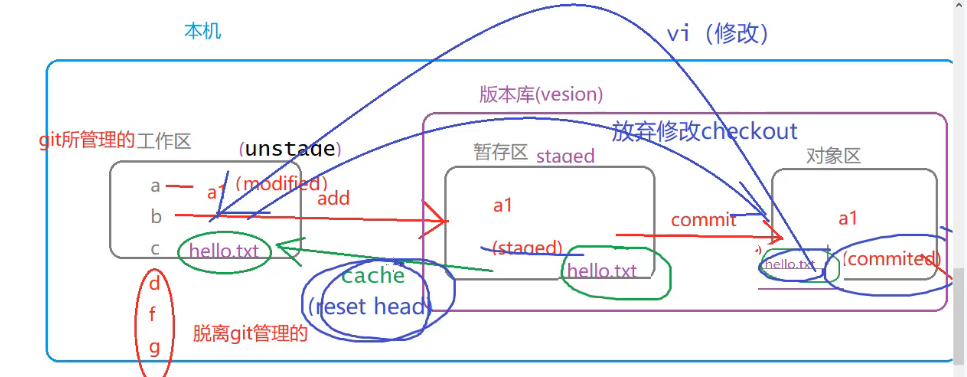
删除 git config --local --unset user.name

git config --local --unset user.email

.git 目录下

cat config 查看文件

撤销工作区改变git checkout -- hello.txt 放弃修改（还原到已提交状态）



git reset head hello.txt 撤销暂存区

提交的问题：

只对修改之后的提交有效。修改之前的提交仍然使用的是之前的配置

删除已提交的文件：

git rm hello.txt

删除之后 文件被放到 暂存区

彻底删除： git commit -m ‘’

后悔：1、git reset HEAD hello.txt 恢复到工作区

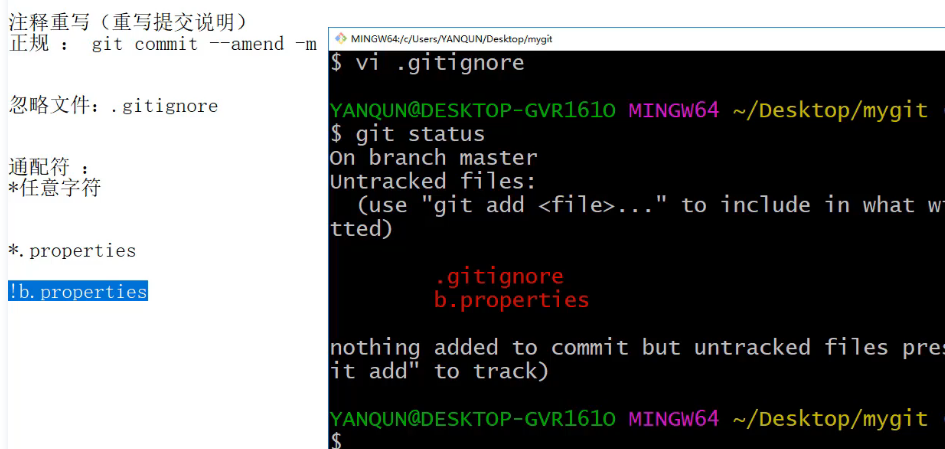
1. git checkout -- hello.txt

重写注释：

git commit --amend -m ‘修正’

忽略文件：

通过配置文件： .gitignore



分支

查看分支：git branch

创建分支：git branch new\_branch

切换分支：git checkout 分支名

删除分支：git branch -d 分支名（不能删除当前分支）为合并的内容，删除分支之前，建议先合并

创建新分支并切换：git checkout -b new\_branch

合并：git merge new\_branch（当前在master分支，把new\_branch合并过来）

强行删除修改的分支：git branch -D 分支名

细节：

如果在分支A中进行了写操作，但此操作局限于工作区中（没add， commit）。在master中能够看到该操作。如果在分支A中从commit，则在master不可见

分支：一个commit链，一条工作记录线

分支名（master）：指向当前的提交（commit）

HEAD：指向当前的分支（HEAD->分支名）

如果一个分支靠前（dev），另一个落后（master）。则如果不冲突，master可以通过merge直接追赶上dev，称为fast forward

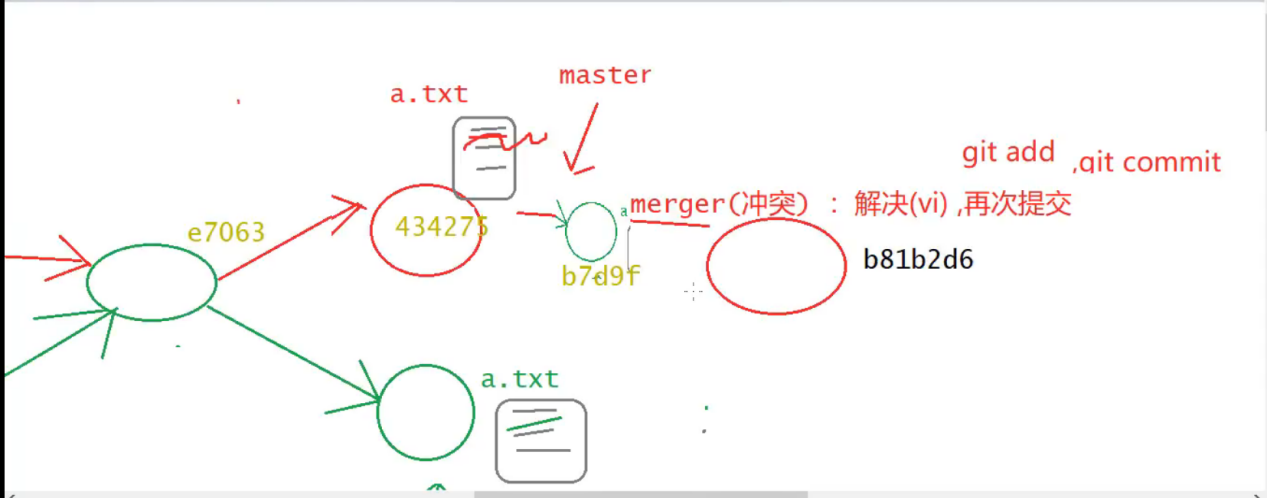
fast forward 本质就是 分支指针的移动，注意：跳过的中间commit，仍然会保存。

fast forward 1、两个分支fast forward 归于一点的commit

1. 没有分支信息

no fast forward 时：默认使用fast forward ；也可以禁止：git merge --no-ff

1. 两个分支 fast forward ， 不会归于一点commit （主动合并的分支，会前进一步）
2. 分支信息完整（不丢失分支信息）



合并：如果冲突，需要解决冲突。

解决完冲突， git add xxx , git commit -m ‘xxx’

git add xxx 告知git，已解决冲突

注意：master在merge时，如果遇到冲突，并解决，则解决冲突，会进行2次提交：1次是最终提交，一次是将对方dev的提交信息commit也拿来了

如果一方落后，另一方 前进，则落后放可以直接通过merge合并到前进方。

git log --graph 图形界面

git log --graph --pretty=oneline --abbrev-commit

合并add和commit：

git commit -am ‘注释’；

版本穿梭：在多个commit之间 进行穿梭。回退、前进

回退到上二次commit： git reset --hard HEAD^^

回退到上N次commit： git reset --head HEAD~n

回退到前XX次commit：通过sha1值 直接回退 git reset --hard sha1 值的前几位

git reflog：查看任意一次记录，记录所有操作。可以帮助我们实现“后悔操作”，需要借助良好的日志习惯

checkout :放弃修改。放弃的是 工作区中的修改，相对于暂存区或对象区

reset：将之增加到暂存区中的内容，回退到工作区

git log -1 :查看最后一次日志

git checkout sha1值：穿梭到某个节点

1. 版本穿梭（游离状态）
2. 修改后必须提交
3. 创建分支好时机 git branch branch sha1

分支重命名：

Git branch –m master master2

Stash:保存现场

1. 建议（规范）：在功能没有开发完毕前，不要commit
2. 规定（必须）：在没有commit之前，不能切换分支
3. 如果还没有将某一个开发完毕，就要切换分支：建议1、保存现场（临时保存 stash）
4. 切换

保存现场：git stash

查看现场（默认是最近一次）：git stash list:查看临时文件

指定恢复现场：git stash apply ‘{stash{0}}’

还原现场：git stash pop：恢复临时文件（将原来保存的删除）

Git stash apply（还原内容，不删除原保存的内容）

手动删除：git stash drop ‘stash{0}’

Tag 标签

git tag xxx

git tag –a xxxx –m ‘xxxx’

查看标签:git tag

删除标签：git tag -d v1.0

模糊查询标签： git tag –l ‘xxx’ -- git tag –l ‘v\*’

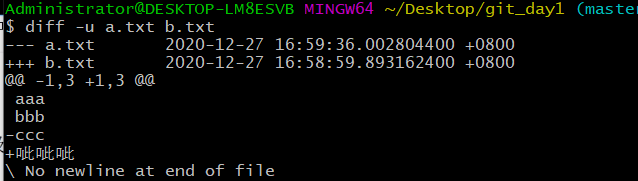
blame:责任

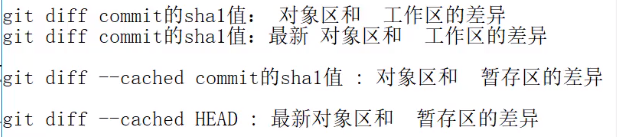
git blame a.txt 查看a.txt的所有提交commit 的sha1值，以及每一行的作者

差异性 diff

diff a.txt b.txt

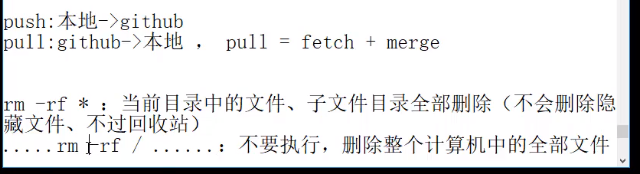
diff –u a.txt b.txt





Push：本地 -🡪 github

Pull:github->本地， pull=fetch + merge



Github

Git add .

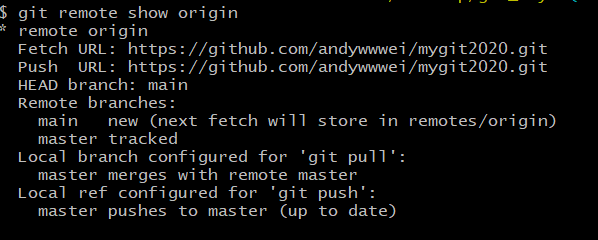
Git commit –m ‘aaa’

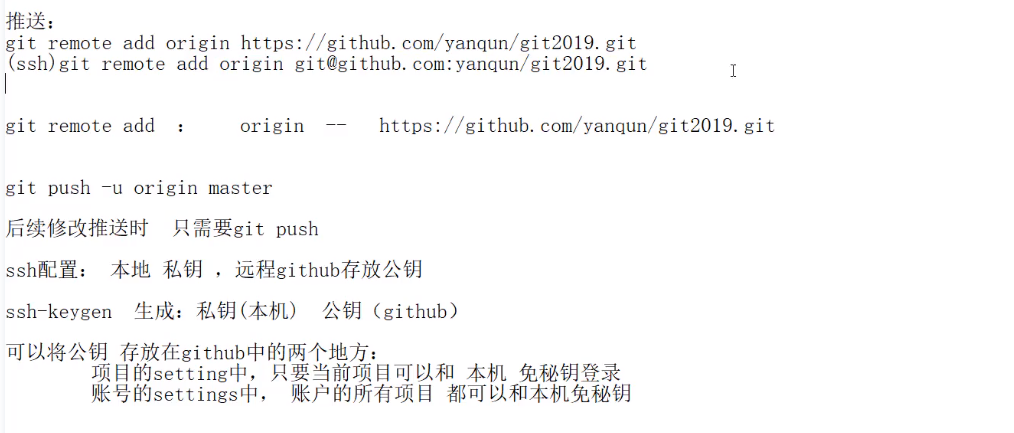
Git remote add origin <https://github.com/andywwwei/mygit2020.git>

git push -u origin master

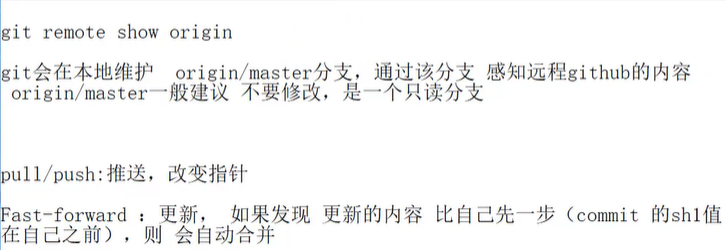
git remote show

git remote show origin





cd – 上一次



解决冲突方法：

Fetch first

Git pull

Pull = fetch + merge

Merge: vi 解决冲突 –》 git add . git commit –m ‘’ -> push

Git图形化工具

Gui gitk

