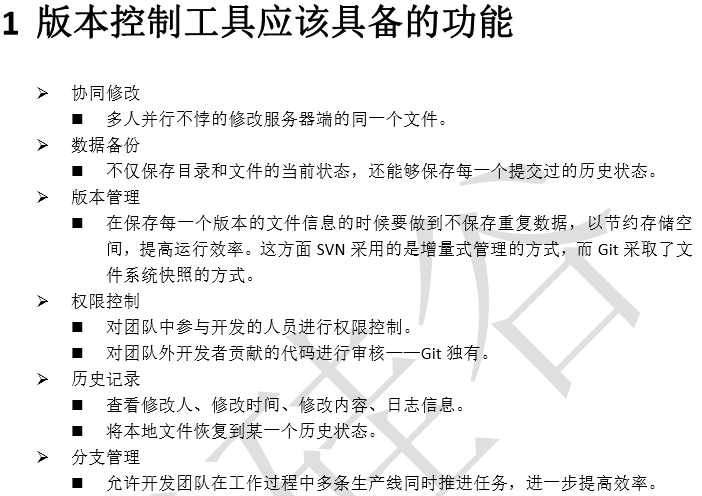
# Git

## 一、版本控制



**版本控制工具分类**：

**①集中式版本控制工具**： CVS、**SVN**、VSS……

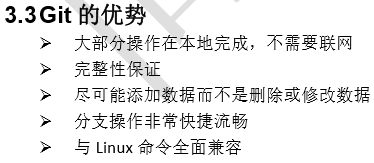
**②分布式版本控制工具**： **Git**、Mercurial、Bazaar、Darcs……

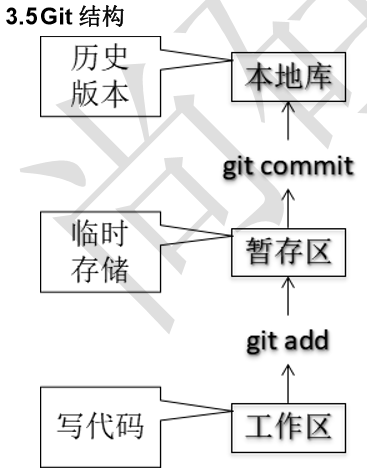
集中式版本控制：每个开发人员是一个客户端，文件和版本信息是存储在服务器中的，开发人员直接跟服务器进行数据交互。**缺点**：若服务器宕机或损坏，则所有的历史数据都会丢失，客户端保存的只是当前的状态，若要恢复只能恢复当前状态的内容，这种也称为**单点故障**。

分布式版本控制：每个开发人员在本地就能进行完整的版本控制，本地就是完整的版本历史记录，每个开发人员都可以很好的将版本内容传给别人，从而避免了单点故障。

## 二、Git

### 1.简介

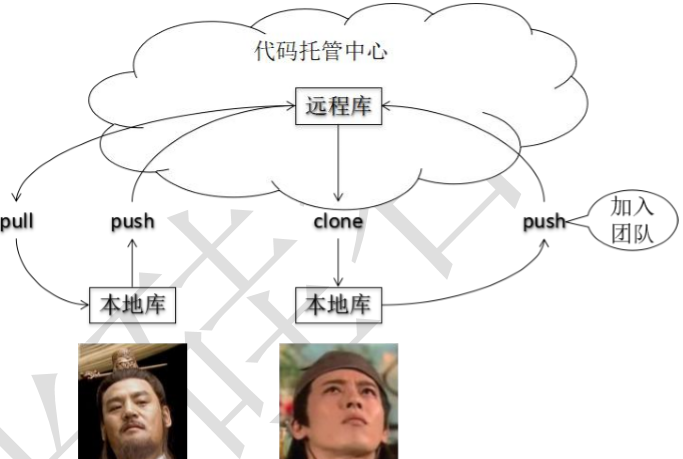




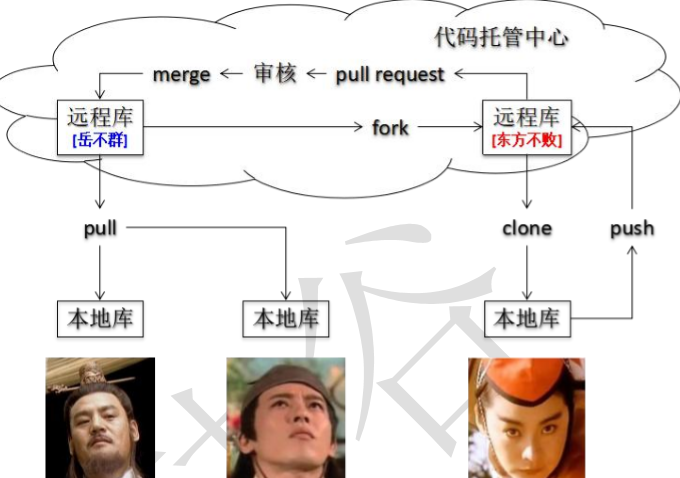
代码托管中心的任务：维护远程库。在局域网情况下的代码托管中心：GitLab 服务器；外网环境下：GitHub、码云。

**本地库和远程库之间交互的方式**：

**①团队内部协作**：



**②跨团队协作**：



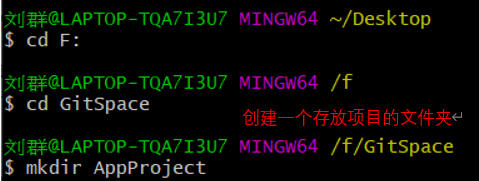
### 2.命令行操作

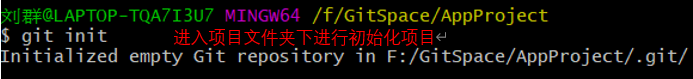
Git中的命令是跟Linux中的命令兼容的。

#### 2.1本地库的初始化

**命令**：**git init**，初始化本地库的命令。

桌面右键Git Bash Here，进入F://GitSpace，这个自己专门用来存放Git项目的文件夹。







**注意**：.git目录中存放的是本地库相关的子目录和文件，不要删除，也不要胡乱修改。

#### 2.2设置签名

**形式**：

用户名：tom

Email 地址：goodMorning@atguigu.com

**作用**：区分不同开发人员的身份。

**辨析**：这里设置的签名和登录远程库(代码托管中心)的账号、密码没有任何关系。

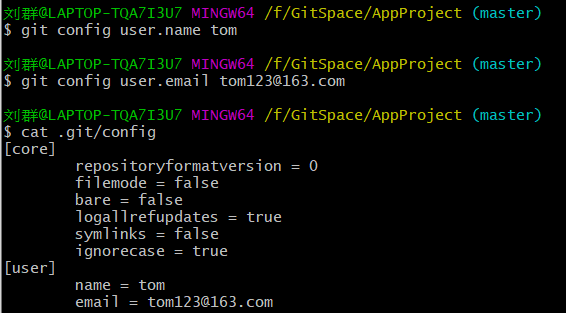
**命令**：

**①项目级别/仓库级别**：仅在当前本地库范围内有效。

**git config** user.name tom；

**git config** user.email [tom123@163.com](mailto:tom123@163.com)。

**信息保存位置**：当前项目的.git文件夹下的config文件中。

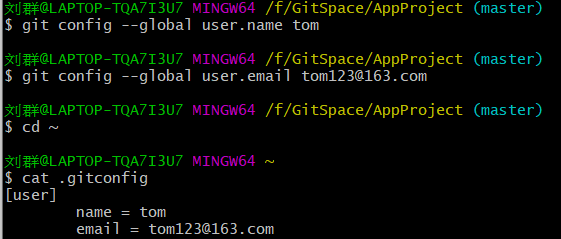


**②系统用户级别**：登录当前操作系统的用户范围。

**git config** **--global** user.name tom；

**git config** **--global** user.email [tom123@163.com](mailto:tom123@163.com)。

**信息保存位置**：在系统根目录下的.gitconfig文件中。



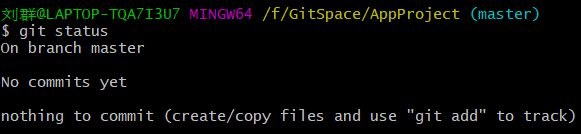
**级别优先级**：

**就近原则**：项目级别优先于系统用户级别，二者都有时采用项目级别的签名；如果只有系统用户级别的签名，就以系统用户级别的签名为准；二者都没有不允许。

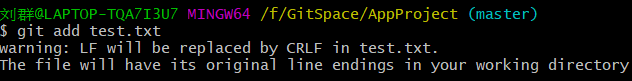
实际开发中，通常设置系统用户级别的就够了。

#### 2.3基本操作

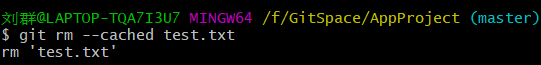
**查看状态**：**git status**命令。查看工作区和暂存区的状态。



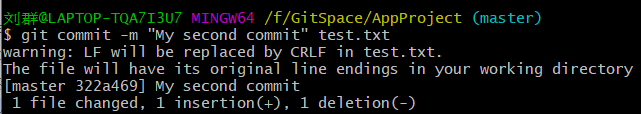
**添加**：**git add** [fileName]。将工作区中新建或修改后的文件添加到暂存区。



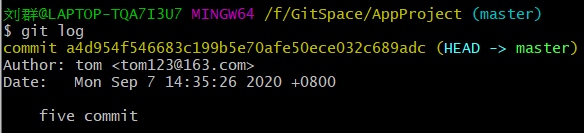
**取消添加**：**git rm –cached** [fileName]。将暂存区中的文件返回到工作区中。



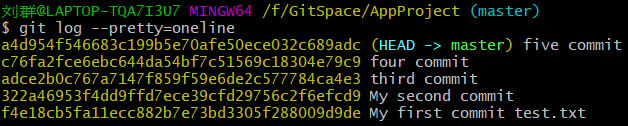
**提交**：**git commit -m** "message"[filename]。将暂存区的内容提交到本地库。



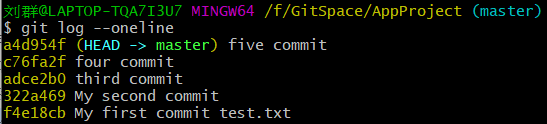
**查看日志记录**：**git log**。当日志非常多时，多屏显示控制方式： 空格向下翻页，b向上翻页，q退出。



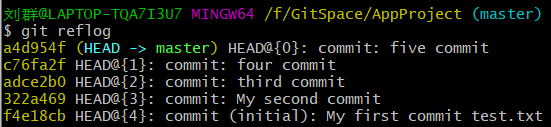
**git log –pretty=oneline**：将日志信息以每行的形式显示。



**git log –oneline**。

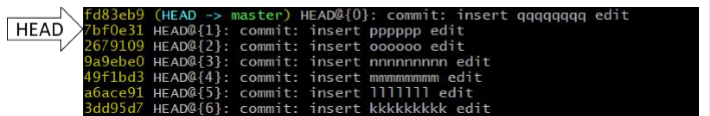


**git reflog**。HEAD@{移动到当前版本需要多少步}。



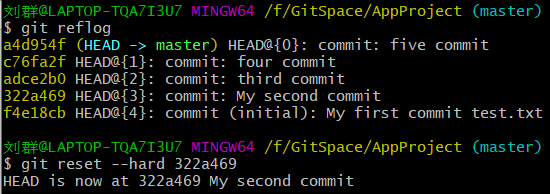
#### 2.4版本的前进后退

**本质**：



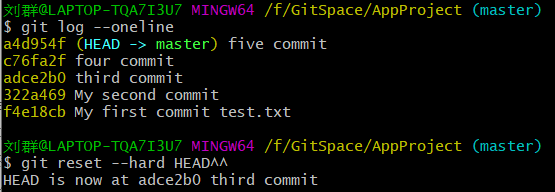
**①基于索引值操作**[推荐]：可前进可后退。

**git reset –hard** [局部索引值]。



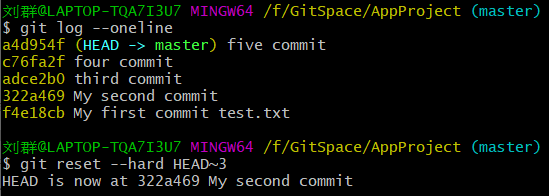
**②使用^符号**：只能后退。

**git reset –hard HEAS^**。注：一个^表示后退一步，n 个表示后退 n 步。



**③使用~符号**：只能后退。

**git reset –hard HEAD~n**。注：n表示后退 n 步。



#### 2.5reset命令的参数

执行**git help reset**命令，会直接在网页中打开reset命令的文档。



**①--soft参数**：仅仅在本地库移动HEAD指针。相当于仅将本地库向后移动了一步。

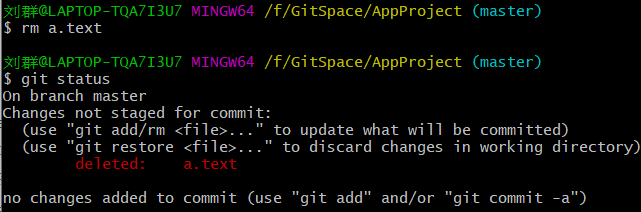
**②--mixed参数**：在本地库移动 HEAD指针，重置暂存区。相当于仅将本地库和暂存区都向后移动了一步。

**③--hard参数**：在本地库移动HEAD指针，重置暂存区，重置工作区。相当于仅将本地库、暂存区、工作区都向后移动了一步。

#### 2.6删除的文件恢复

**rm** [**fileName**]：删除文件。

要想将删除的文件进行恢复前提：删除前，文件存在时的状态提交到了本地库。



**操作**：**git reset --hard**[指针位置]。删除操作已经提交到本地库，指针位置指向历史记录；删除操作尚未提交到本地库，指针位置使用 HEAD。

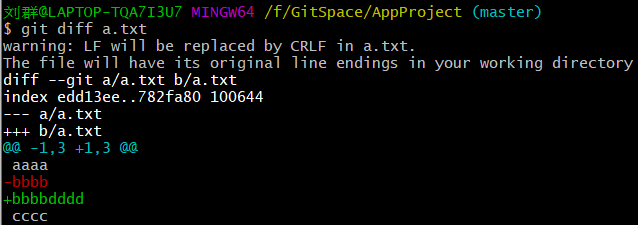


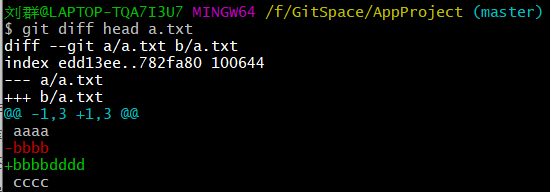
#### 2.7比较文件差异

**git diff**[文件名]：将工作区中的文件和暂存区进行比较。

**git diff** [本地库中历史版本][文件名]：将工作区中的文件和本地库历史记录比较。

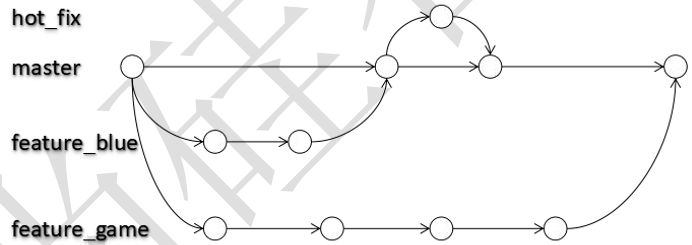
不带文件名比较多个文件。

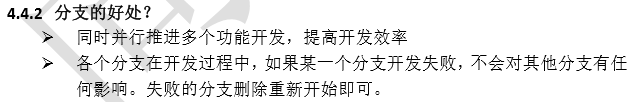




### 3.分支管理

分支：在版本控制过程中，使用多条线同时推进多个任务。



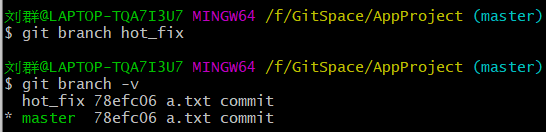


#### 3.1分支操作

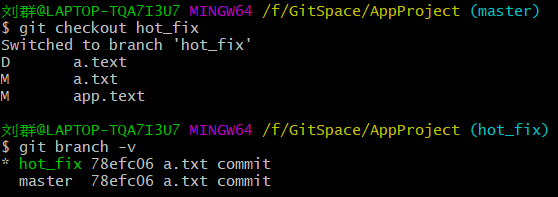
**查看分支**：**git branch -v**。



**创建分支**：**git branch** [**分支名**]。



**切换分支**：**git checkout** [**分支名**]。

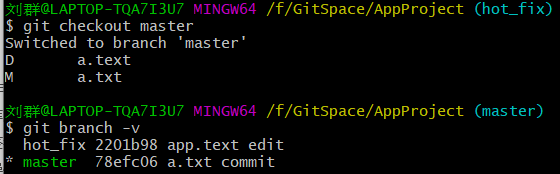


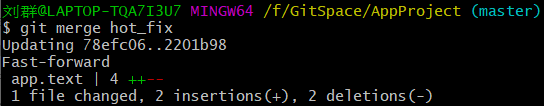
**合并分支**：

第一步，先切换到主分支上：**git checkout** [**branchName**]；

第二步，执行 merge 命令：**git merge** [**branchName**]。

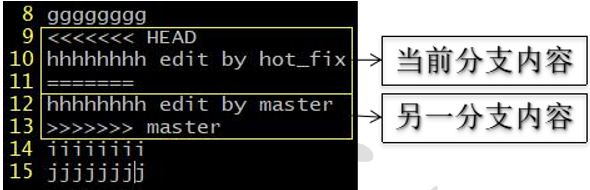
如hot\_fix分支上的内容需要合并到master分支上，需要先切换到master分支上。





**解决合并产生的冲突问题**：

**冲突的表现**：



**冲突的解决**：

第一步：编辑文件，删除特殊符号；

第二步：把文件修改到满意的程度，保存退出；

第三步：**git add[文件名]**；

第四步：**git commit -m**"日志信息"。注意：此时 commit 一定不能带具体文件名，即git commit -m “xx” xx.txt等是错误的。