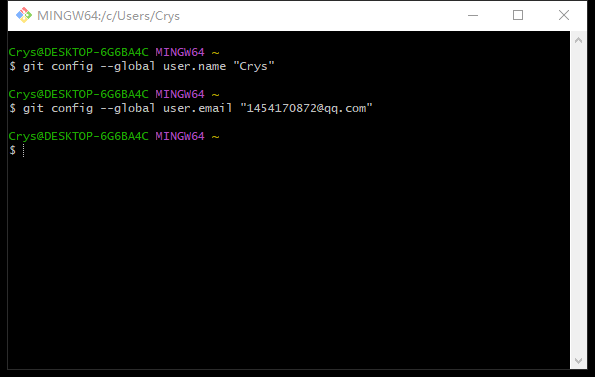
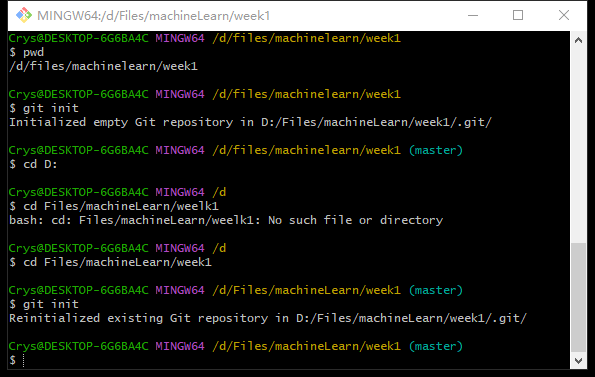
1．git config --global 参数，有了这个参数，表示这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定的不同的用户名和邮箱。

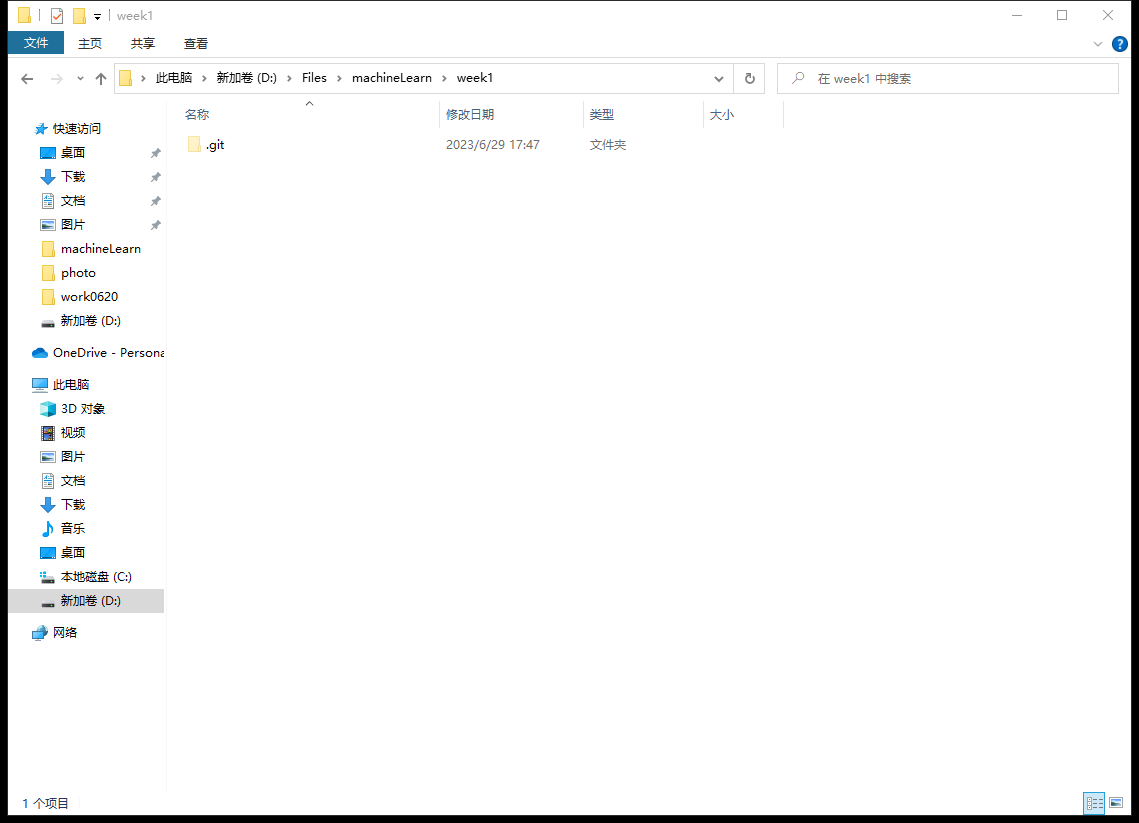


2．创建版本库

版本库又名仓库，英文名repository,简单的理解为一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改，删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻还可以将文件”还原”。

1. 进入文件使用
2. 命令pwd可以显示当前的目录
3. 命令 git init 把这个目录变成git可以管理的仓库

当前week1目录下会多了一个.git的目录

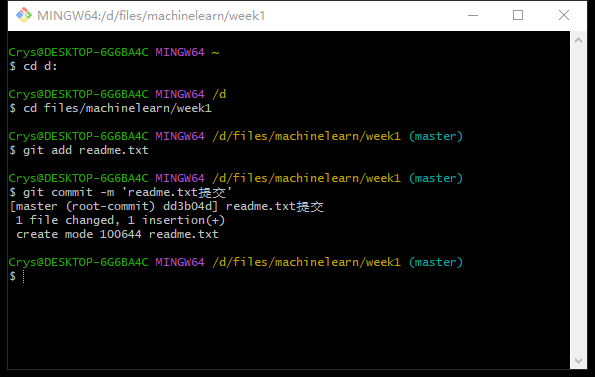


1. 命令**git add**将内容写入暂存区，**git commit** 命令将暂存区内容添加到本地仓库中。

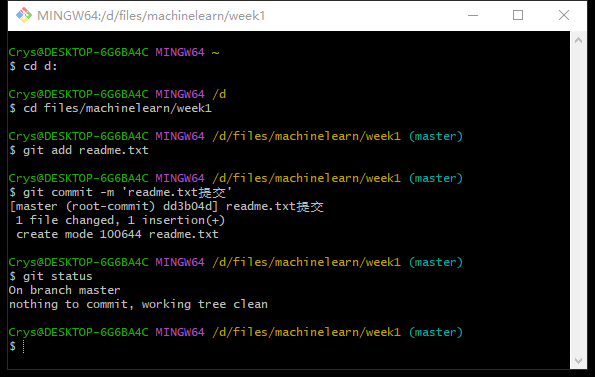
提交暂存区到本地仓库中:，[message] 可以是一些备注信息。

提交暂存区的指定文件到仓库区：

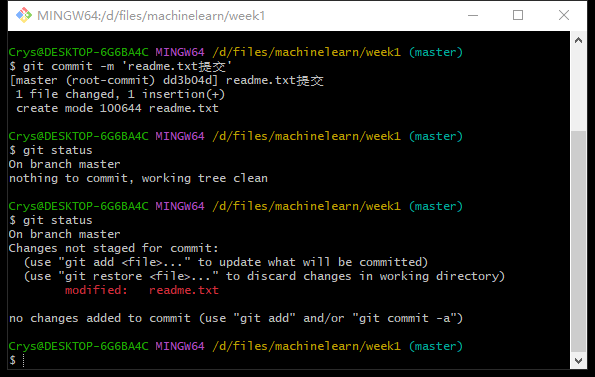




1. 命令git status来查看是否还有文件未提交：

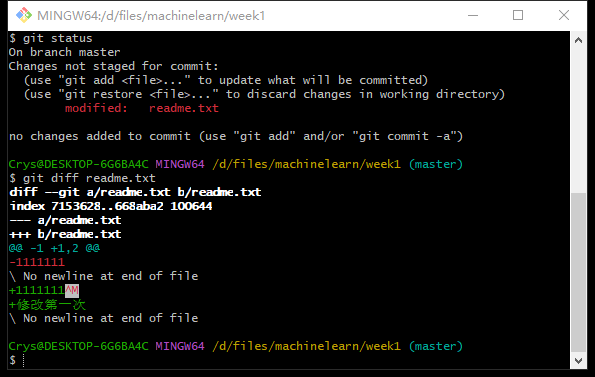


1. 修改readme.txt里的内容再使用git status：



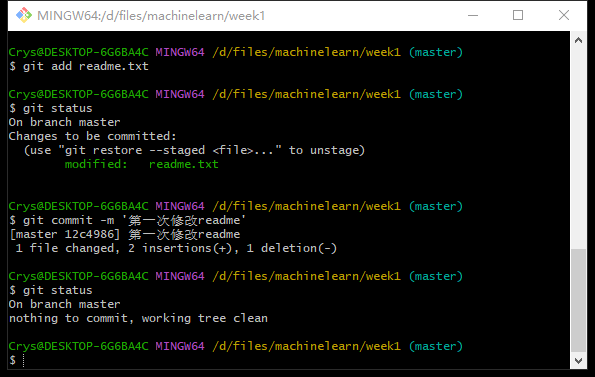
命令告诉 readme.txt文件已被修改，但是未被提交的修改。

1. 命令git diff readme.txt展示文件到底改了什么内容：

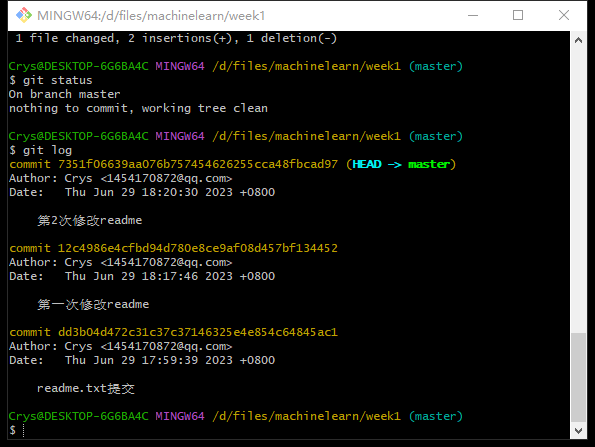


1. 将文件提交到仓库使用命令

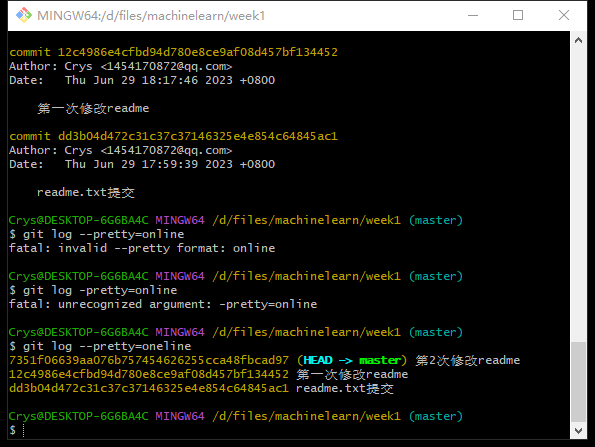




1. 查看对readme.txt修改的历史记录：

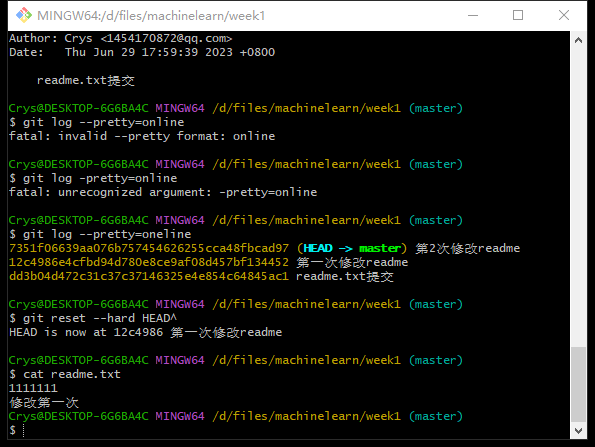


：简洁显示修改历史

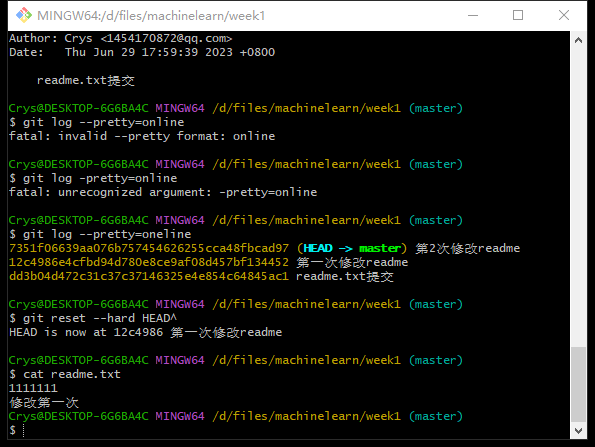


1. 想把文件当前的版本回退到上一个版本：

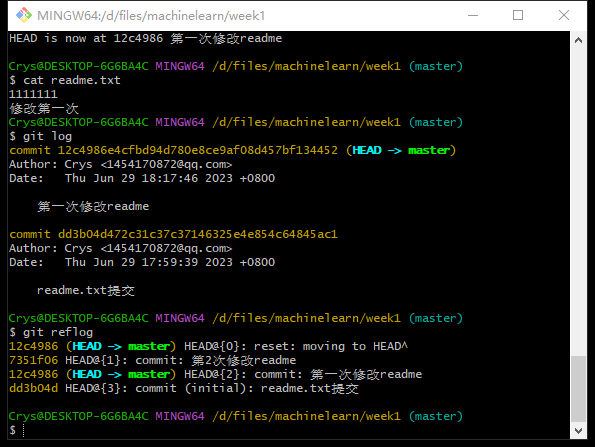




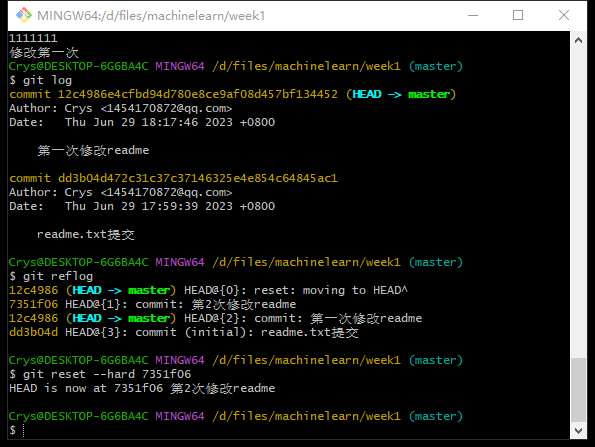
1. 查看文件的内容：



1. 命令即可获取到版本号：



恢复特定版本：：



3．工作区：在电脑上看到的目录，比如目录下week1里的文件(.git隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

版本库(Repository)：工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是stage(暂存区)，还有Git为我们自动创建了第一个分支master,以及指向master的一个指针HEAD。

使用Git提交文件到版本库有两步：

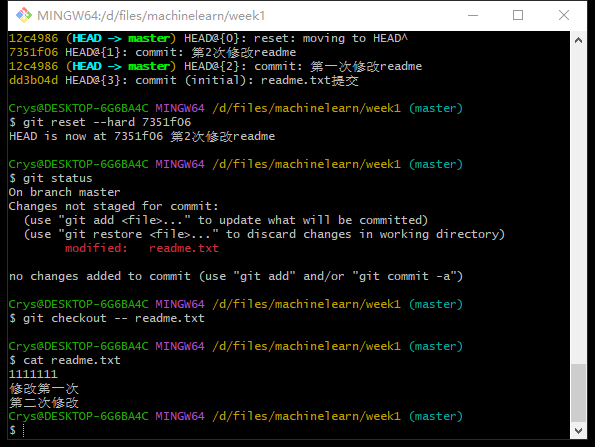
第一步：是使用 git add 把文件添加进去，实际上就是把文件添加到暂存区。

第二步：使用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

4．撤销修改方法：①知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后add添加到暂存区，最后commit掉。

②按以前的方法直接恢复到上一个版本。使用 git reset --hard HEAD^。

③git checkout -- filename 可以丢弃工作区的修改。

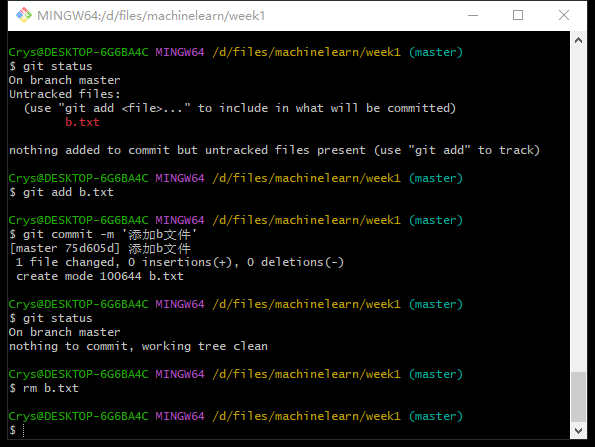


命令 git checkout --readme.txt 意思就是，把readme.txt文件在工作区做的修改全部撤销，这里有2种情况，如下：

1.readme.txt自动修改后，还没有放到暂存区，使用 撤销修改就回到和版本库一模一样的状态。

2.另外一种是readme.txt已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。

删除文件方法：命令:



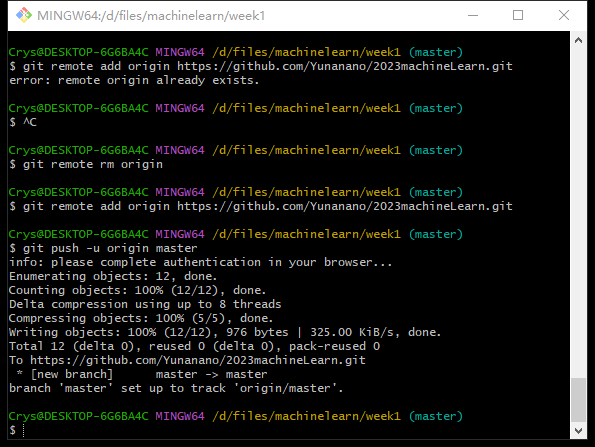
一般情况下，可以直接在文件目录中把文件删了，或者使用如上rm命令：rm b.txt ，如果我想彻底从版本库中删掉了此文件的话，可以再执行commit命令提交掉。只要没有commit之前，可以使用命令 git checkout -- b.txt恢复b.txt文件。

5．远程仓库。

（1）将本地仓库连接到远程仓库：

在当前目录下把一个已有的本地仓库与github上面的2023machineLearn关联。

$ git remote add origin <https://github.com/Yunanano/2023machineLearn.git>



把本地库的内容推送到远程，使用 git push命令，实际上是把当前分支master推送到远程。

从现在起，只要本地作了提交，就可以通过如下命令：git push origin master

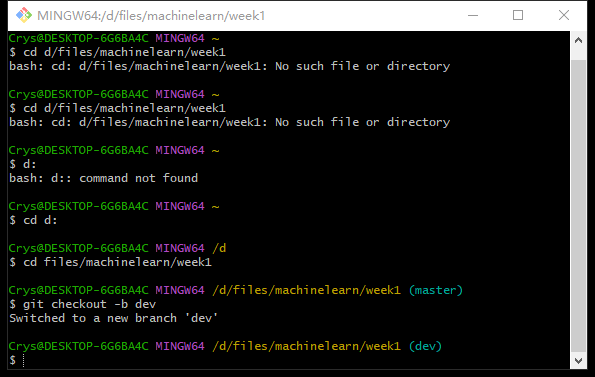
把本地master分支的最新修改推送到github上了，现在就拥有了真正的分布式版本库了。

（2）将远程库克隆至本地：

git clone <https://github.com/Yunanano/2023machineLearn.git>

（3）创建与合并分支：

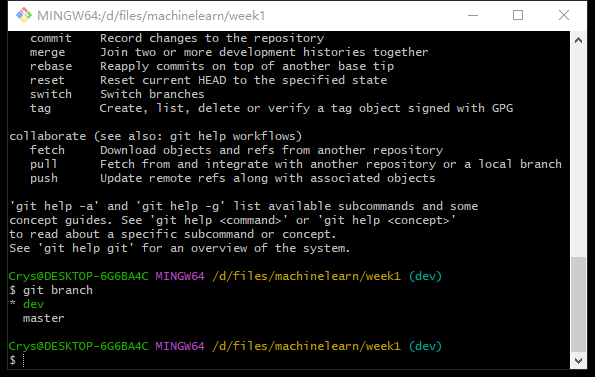
git checkout -b dev：创建dev分支并切换到dev分支上



git branch dev：创建dev分支

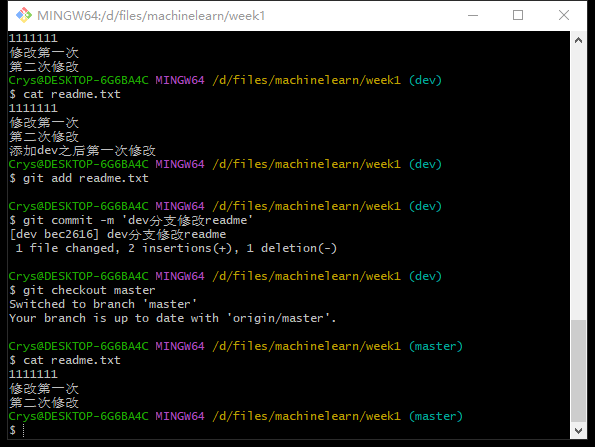
git checkout dev：切换到dev分支

git branch：查看当前分支

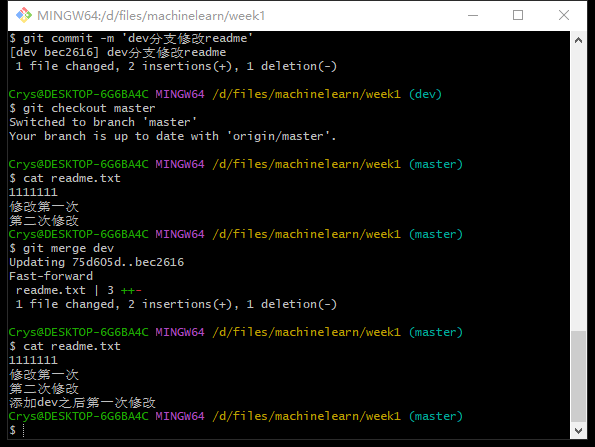


在分支dev上修改readme.txt里的内容不会更新到master上的readme.txt内容

里。

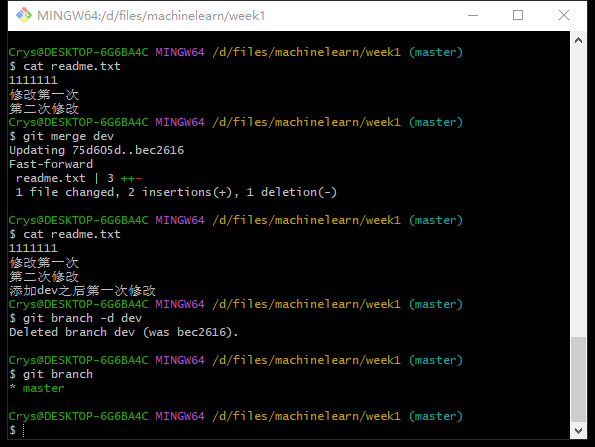


把dev分支上的内容合并到分支master上了，可以在master分支上，使用如下命令 git merge dev 如下所示：



git merge命令用于合并指定分支到当前分支上。

删除dev分支：git branch -d dev



查看分支：git branch

创建分支：git branch name

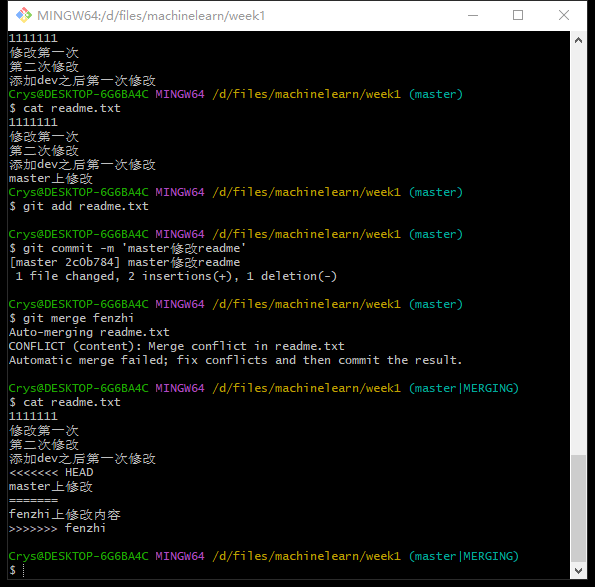
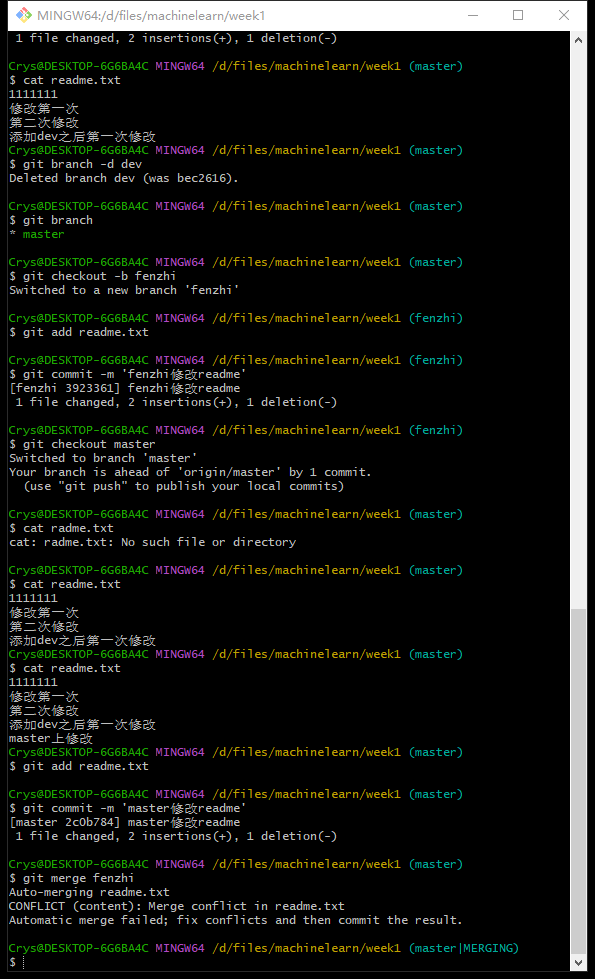
切换分支：git checkout name

创建+切换分支：git checkout –b name

合并某分支到当前分支：git merge name

删除分支：git branch –d name

(4)不同分支对同一文件内容合并的冲突：



Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，其中<<<HEAD是指主分支修改的内容，>>>>>fenzhi1 是指fenzhi上修改的内容。可以直接在txt文件里修改。

（5）分支管理策略。

通常合并分支时，git一般使用”Fast forward”模式，在这种模式下，删除分支后，会丢掉分支信息，现在我们来使用带参数 –no-ff来禁用”Fast forward”模式。

Demo：

创建一个dev分支。

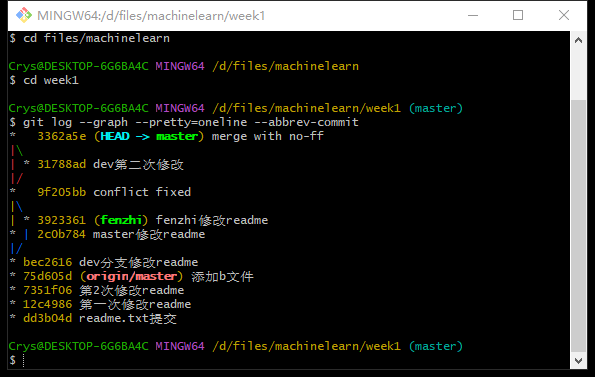
修改readme.txt内容。

添加到暂存区。

切换回主分支(master)。

合并dev分支，使用命令 git merge –no-ff -m “注释” dev

查看历史记录。



被删除的分支信息

分支策略：首先master主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的dev分支上干活，干完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

（6）bug分支

在开发中，会经常碰到bug问题，那么有了bug就需要修复，在Git中，分支是很强大的，每个bug都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时的分支删除掉。

git stash：将当前的工作现场—dev分支隐藏起来。

之后在创建新的分支--issue-404修复bug。修复完之后合并到其他分支--master上。删除新的分支—issue-404。

git stash list：查看隐藏起来的工作现场。

Git把stash内容存在某个地方了，但是需要恢复一下：

①git stash apply恢复，恢复后，stash内容并不删除，你需要使用命令git stash drop来删除。

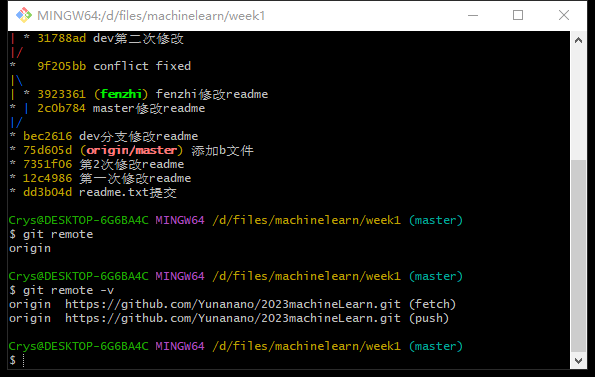
②另一种方式是使用git stash pop,恢复的同时把stash内容也删除了。

6．多人协作

当你从远程库克隆时候，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程库的默认名称是origin。

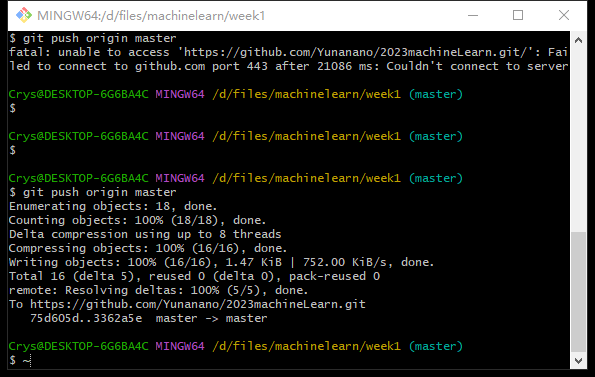
要查看远程库的信息 使用 git remote

要查看远程库的详细信息 使用 git remote –v



1. 推送分支。

推送分支就是把该分支上所有本地提交到远程库中，推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：命令 git push origin master



1. 抓取分支。

多人协作工作模式：

git push origin branch-name推送自己的修改。

如果推送失败，则因为远程分支比你的本地更新早，需要先用git pull试图合并。

如果合并有冲突，则需要解决冲突，并在本地提交。再用git push origin branch-name推送。

一、新建代码库

# 在当前目录新建一个Git代码库

$ git init

# 新建一个目录，将其初始化为Git代码库

$ git init [project-name] # 下载一个项目和它的整个代码历史

$ git clone [url]

二、配置

# 显示当前的Git配置

$ git config --list

# 编辑Git配置文件

$ git config -e [--global] # 设置提交代码时的用户信息

$ git config [--global] user.name "[name]"

$ git config [--global] user.email "[email address]"

三、增加/删除文件

# 添加指定文件到暂存区

$ git add [file1] [file2] ... # 添加指定目录到暂存区，包括子目录

$ git add [dir] # 添加当前目录的所有文件到暂存区

$ git add . # 添加每个变化前，都会要求确认 # 对于同一个文件的多处变化，可以实现分次提交

$ git add -p

# 删除工作区文件，并且将这次删除放入暂存区

$ git rm [file1] [file2] ... # 停止追踪指定文件，但该文件会保留在工作区

$ git rm --cached [file] # 改名文件，并且将这个改名放入暂存区

$ git mv [file-original] [file-renamed]

四、代码提交

# 提交暂存区到仓库区

$ git commit -m [message] # 提交暂存区的指定文件到仓库区

$ git commit [file1] [file2] ... -m [message] # 提交工作区自上次commit之后的变化，直接到仓库区

$ git commit -a

# 提交时显示所有diff信息

$ git commit -v

# 使用一次新的commit，替代上一次提交 # 如果代码没有任何新变化，则用来改写上一次commit的提交信息

$ git commit --amend -m [message] # 重做上一次commit，并包括指定文件的新变化

$ git commit --amend [file1] [file2] ...

五、分支

# 列出所有本地分支

$ git branch

# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

$ git branch -a

# 新建一个分支，但依然停留在当前分支

$ git branch [branch-name] # 新建一个分支，并切换到该分支

$ git checkout -b [branch] # 新建一个分支，指向指定commit

$ git branch [branch] [commit] # 新建一个分支，与指定的远程分支建立追踪关系

$ git branch --track [branch] [remote-branch] # 切换到指定分支，并更新工作区

$ git checkout [branch-name] # 切换到上一个分支

$ git checkout - # 建立追踪关系，在现有分支与指定的远程分支之间

$ git branch --set-upstream [branch] [remote-branch] # 合并指定分支到当前分支

$ git merge [branch] # 选择一个commit，合并进当前分支

$ git cherry-pick [commit] # 删除分支

$ git branch -d [branch-name] # 删除远程分支

$ git push origin --delete [branch-name]

$ git branch -dr [remote/branch]

六、标签

# 列出所有tag

$ git tag

# 新建一个tag在当前commit

$ git tag [tag] # 新建一个tag在指定commit

$ git tag [tag] [commit] # 删除本地tag

$ git tag -d [tag] # 删除远程tag

$ git push origin :refs/tags/[tagName] # 查看tag信息

$ git show [tag] # 提交指定tag

$ git push [remote] [tag] # 提交所有tag

$ git push [remote] --tags

# 新建一个分支，指向某个tag

$ git checkout -b [branch] [tag]

七、查看信息

# 显示有变更的文件

$ git status

# 显示当前分支的版本历史

$ git log

# 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

# 搜索提交历史，根据关键词

$ git log -S [keyword] # 显示某个commit之后的所有变动，每个commit占据一行

$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名

$ git log --follow [file]

$ git whatchanged [file] # 显示指定文件相关的每一次diff

$ git log -p [file] # 显示过去5次提交

$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序

$ git shortlog -sn

# 显示指定文件是什么人在什么时间修改过

$ git blame [file] # 显示暂存区和工作区的差异

$ git diff

# 显示暂存区和上一个commit的差异

$ git diff --cached [file] # 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 显示两次提交之间的差异

$ git diff [first-branch]...[second-branch] # 显示今天你写了多少行代码

$ git diff --shortstat "@{0 day ago}" # 显示某次提交的元数据和内容变化

$ git show [commit] # 显示某次提交发生变化的文件

$ git show --name-only [commit] # 显示某次提交时，某个文件的内容

$ git show [commit]:[filename] # 显示当前分支的最近几次提交

$ git reflog

八、远程同步

# 下载远程仓库的所有变动

$ git fetch [remote] # 显示所有远程仓库

$ git remote -v

# 显示某个远程仓库的信息

$ git remote show [remote] # 增加一个新的远程仓库，并命名

$ git remote add [shortname] [url] # 取回远程仓库的变化，并与本地分支合并

$ git pull [remote] [branch] # 上传本地指定分支到远程仓库

$ git push [remote] [branch] # 强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] --force

# 推送所有分支到远程仓库

$ git push [remote] --all

九、撤销

# 恢复暂存区的指定文件到工作区

$ git checkout [file] # 恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file] # 恢复暂存区的所有文件到工作区

$ git checkout . # 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

$ git reset [file] # 重置暂存区与工作区，与上一次commit保持一致

$ git reset --hard

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit] # 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit] # 重置当前HEAD为指定commit，但保持暂存区和工作区不变

$ git reset --keep [commit] # 新建一个commit，用来撤销指定commit # 后者的所有变化都将被前者抵消，并且应用到当前分支

$ git revert [commit] # 暂时将未提交的变化移除，稍后再移入

$ git stash

$ git stash pop