git 使用

一、git配置

配置用户名和邮箱

$ git config --global user.name "yeshen"

$ git config --global user.email 569242715@qq.com

配置git的文本编辑器

$ git config --global core.editor vim

检测已有的配置信息

$ git config --list

配置钥匙

$ ssh-keygen -t rsa -C 569242715@qq.com

二、git仓库创建

远程仓库创建

通常实际项目中会使用一台独立的机器作为git服务器，然后在git服务器中建立一个远程仓库，然后项目中所有开发人员通过局域网在自己的电脑上访问该git服务器。但由于条件限制，这里使用同一台本地仓库的机器来搭建git服务器。

$ cd /home/work

$ mkdir git.source

$ cd git.source

$ mkdir origin

$ cd origin

$ git --bare init

最后的 git --bare init 命令在/home/work/git.source/origin目录下创建了一个空的远程仓库，通过ls -a看到目录多了branches config description HEAD hooks info objects refs。

本地仓库创建

$ cd /home/work/git.source

$ mkdir user1

$ cd user1

$ vim test.txt

在/home/work/git.client目录下创建一个test.txt文件然后写点内容保存。

$ git init

git init 命令在/home/work/git.client/目录下创建了一个本地仓库。通过ls -a看到目录下多了一个.git目录,进入.git然后ls -a发现里面的内容和之前创建远程仓库一样多了branches config description HEAD hooks info objects refs。

$ git remote add origin ssh://work@192.168.45.130:/home/work/git.source/origin

最后添加绑定远程仓库地址。

用同样的方法创建第二个叫user2的用户。

三、git基础命令

一个仓库由**工作区**，**缓存区（Index）**，**提交区（HEAD）**三部分组成。工作区保持着实际的项目代码文件，缓存区临时保存改动，提交区保存最后一次提交的结果。以下简图是他们的关系。

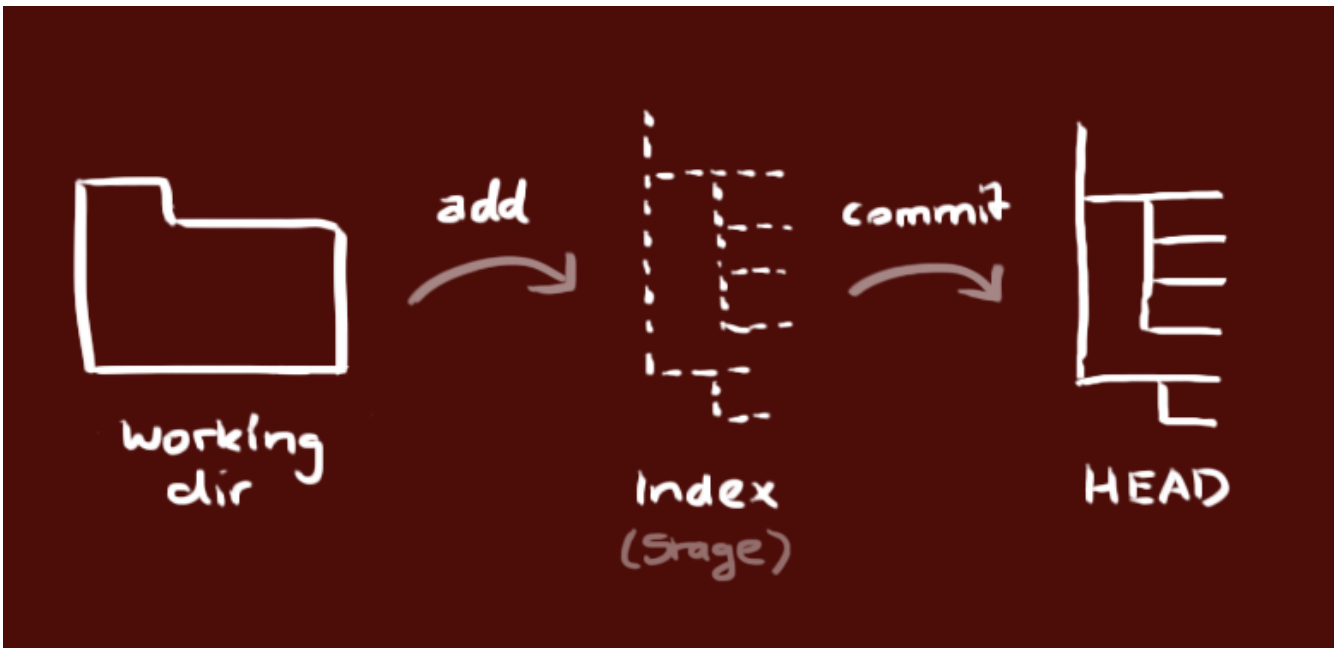

图1

查看工作区的状态

进入仓库目录中，比如cd /home/work/git.source/user1，然后 git status会显示工作区的代码文件及相应信息。 当新建文件test.c或修改时，未git add之前git status会显示以下信息:

```
work@work-virtual-machine:~/git.source/user1$ git status
位于分支 master

尚无提交

未跟踪的文件：
  （使用 "git add <文件>..." 以包含要提交的内容）

        test.txt

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）
```

图2

上图说明工作区新建了文件或新修改了文件test.txt，而没有使用git add命令添加到缓存区来建立跟踪。

当git add test.c之后git status：

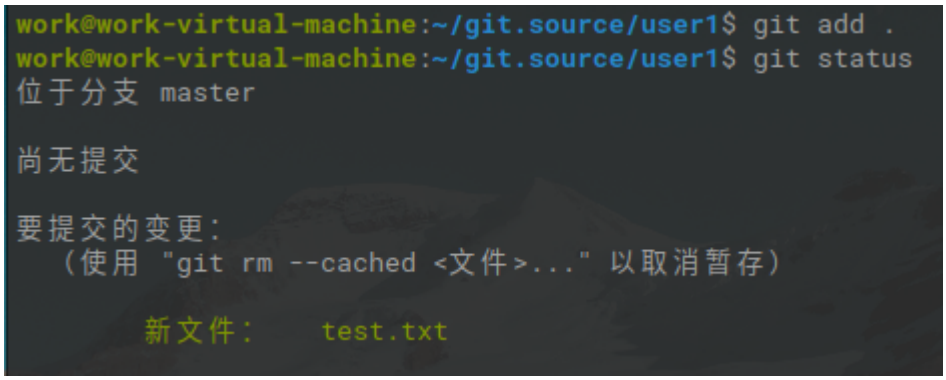

图3

通过git commit -m "1st user1 master commit"提交后git status：

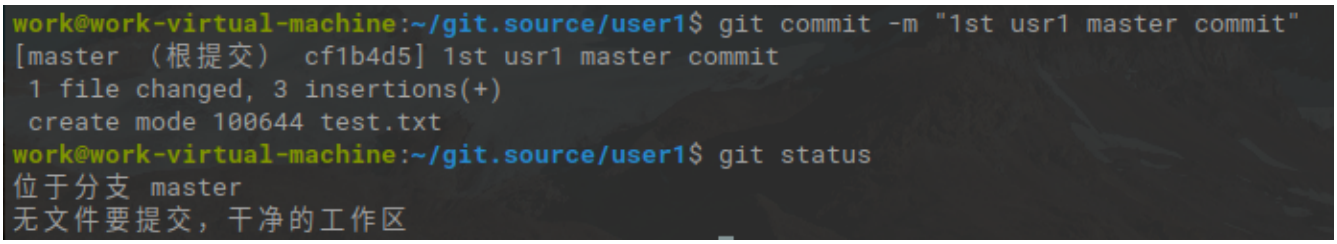

图4

将工作区的某个文件添加到缓存区

比如将test.c添加到缓存区 git add test.c，然后通过git status可以查看状态的变化，如图3。如果想将工作区所有文件添加到缓存区，则git add *。

将缓存区的某个文件删除

比如将已经添加到缓存区中的 test.c从缓存区删除 git rm --cached test.c，然后通过git status可以查看状态变化，此时会变回图2的状态。如果想将缓存区的所有文件从缓存区删除，则git rm --cached *。

将缓存区的所有文件提交到版本库

$ git commit -m "提交注释信息"

在修改本地代码之前如果没有和服务器远程代码同步，提交的时候可能会产生**冲突**，这时需要解决冲突问题。比如，先git pull获取最新的代码，然后再修改、提交、推送。

$git commit --amend -m "新提交注释信息"

以上命令将最近一次提交的注释修改为 "提交注释信息"合并。

查看提交版本记录

$ git log

将版本库的改动推送到远程仓库

推送格式为git push <远程仓库名> <本地分支名>:<远程分支名>

$ git push origin master:master

将远程版本库的拉取到本地仓库

拉取格式为git pull <远程仓库名> <远程分支名>:<本地分支名>

强制拉取格式为git pull -f <远程仓库名> <远程分支名>:<本地分支名>

$ git pull origin master:master

$ git pull -f origin master:master

在git pull后git log会多一个被推送到远程的最新的提交记录。

clone本地仓库

git clone操作会将目标的版本库最新版本复制一份到当前目录。

$ git clone /home/work/git.source/user1

clone远程仓库

和clone本地仓库有点不一样，地址中需要网络地址不只是本机目录地址。

$ git clone ssh://work@192.168.45.130:/home/work/git.source/origin

clone仓库中特定的分支

默认情况下clone一个仓库只会clone该仓库的master分支，通过git clone -b <指定分支名> <远程仓库地址>就可以clone指定分支。

四、分支管理

分支原理

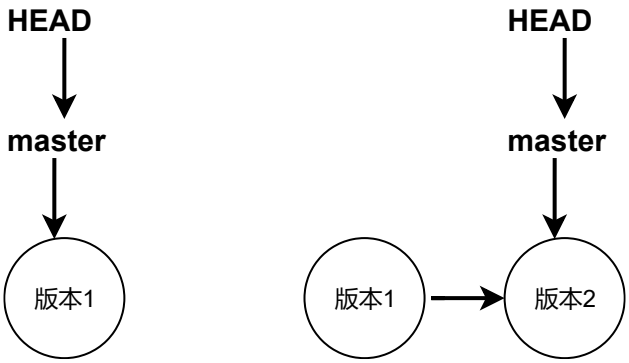

图5

分支是由一个指针来表示的，在不考虑版本穿梭的情况下该指针指向该分支最新一次提交，而HEAD指针指向当前分支。如图5所示，在第一次提交时只有一个master分支和一个版本提交，然后做些文件修改第二次提交，此时master分支指针指向最新提交，HEAD指针随着移动指向master。

查看分支

$ git branch

上述命令显示本地分支，通过上述命令后显示* master，说明只有一个master分支，*表示当前分支。

$ git branch -a

上述命令显示本地分支和远程分支。

创建分支

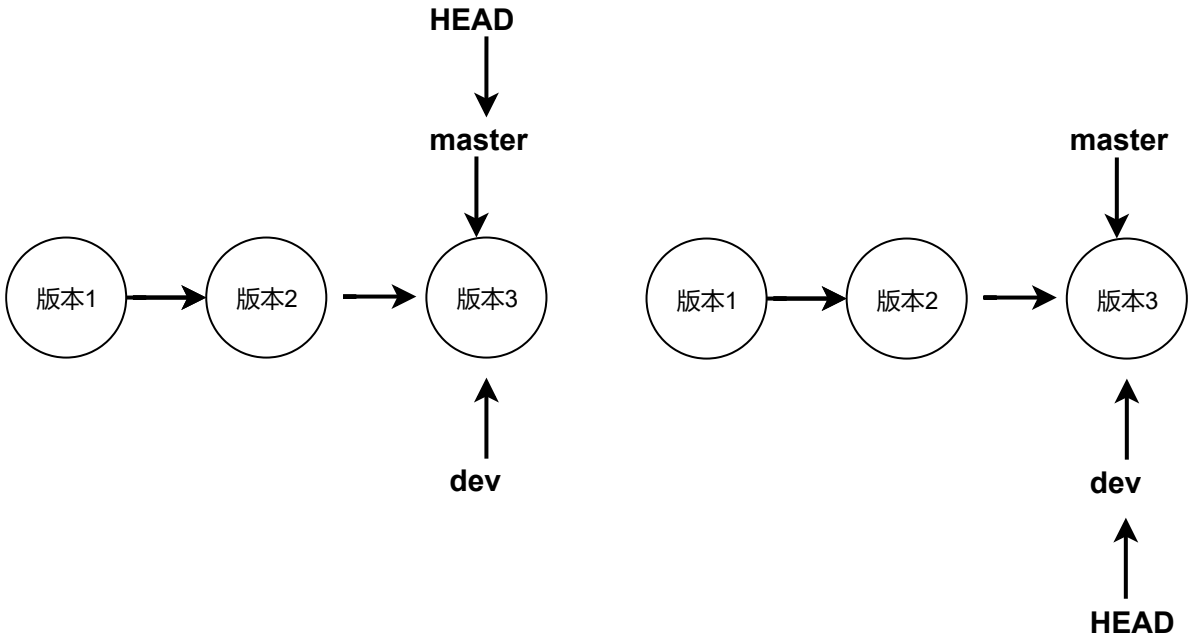

图6

创建分支会创建一个分支指针和当前分支指针指向同一版本提交。如图6左图所示。

$ git branch dev

通过上述命令创建一个dev分支。我们在提交三个版本后创建dev分支，然后git log显示master分支的提交记录：

```
work@work-virtual-machine:~/git.source/user1$ git log
commit cf8761236511cb8dbb6f52f3266f04ae8e990c0f (HEAD -> master)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 16:26:52 2020 +0800

    3st user1 master commit

commit ab2910a6d75438178aa2502b6fc37a0f04d5db2a
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 16:25:50 2020 +0800

    2st user1 master commit

commit cf1b4d5208fcd864cd0b33307ec3579e6030b833 (origin/master)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 14:47:21 2020 +0800

    1st usr1 master commit
```

图7

切换分支

切换分支只是将HEAD指针从之前指向别的分支到指向新分支的过程。如图6右图所示。

$ git checkout dev

使用上述命令来切换到dev分支,然后git log显示dev分支的提交记录：

```
work@work-virtual-machine:~/git.source/user1$ git checkout dev
切换到分支 'dev'
work@work-virtual-machine:~/git.source/user1$ git log
commit cf8761236511cb8dbb6f52f3266f04ae8e990c0f (HEAD -> dev, master)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 16:26:52 2020 +0800

    3st user1 master commit

commit ab2910a6d75438178aa2502b6fc37a0f04d5db2a
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 16:25:50 2020 +0800

    2st user1 master commit

commit cf1b4d5208fcd864cd0b33307ec3579e6030b833 (origin/master)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 14:47:21 2020 +0800

    1st usr1 master commit
```

图8

从上图可以看出dev分支和master分支的提交记录一样，因为他们还没有新的提交，在各自有新提交后才真正的分支。

删除分支

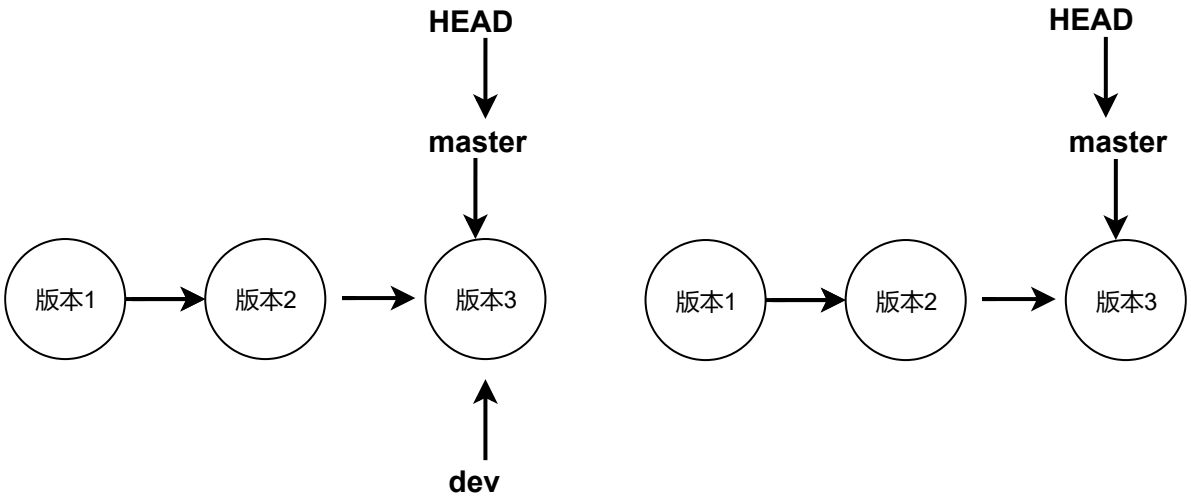

图9

删除分支只是单纯的删除该分支的指针，但要注意的是不能删除当前分支，因此先要切换别的分支为当前分支再删除该分支。如图6右图，我们想删除dev分支，但此时dev是当前分支，因此我们先切换master分支为当前分支，如图9左，然后再删除dev分支，如图9右。

$ git branch -d dev

上述命令删除dev分支。

推送分支

git push <远程仓库名> <本地分支名>:<远程分支名>

$ git push origin dev:dev

上述命令将本地的dev分支推送到远程的dev分支，如果远程不存在dev分支会自动创建。注意的是如果推送之前没有git commit则clone得到的文件是创建该分支之前的分支的文件，无论在推送之前在新分支做了什么改动。

合并某个分支到当前分支

$ git merge dev

上述命令将dev分支合并到当前分支master。

(1) 在dev分支做新提交，master分支不做新提交，然后合并dev分支到master分支，这种情况没有冲突。

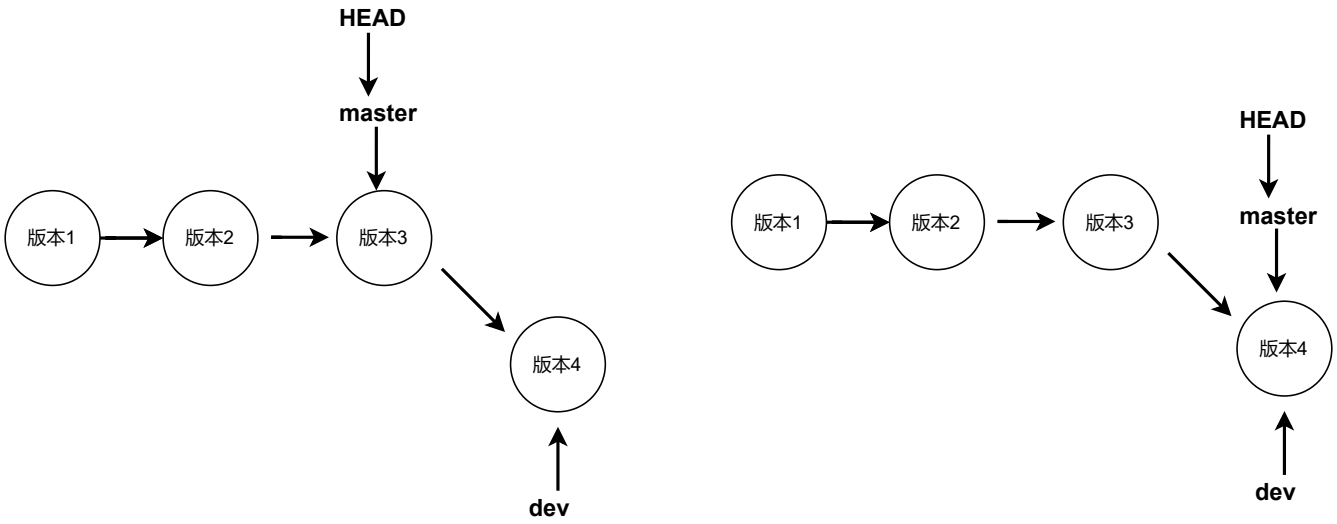

图10

如图10所示，在版本3时创建了dev分支，然后在dev分支做了一次新提交，而master分支不做新提交，此时分支结构如图左所示，然后将当前分支切换到master后再通过git merge dev将dev分支合并到master分支，此时的合并结果只是将master和HEAD指针移动到dev的位置。

(2) master分支和dev分支各自做自己的提交，然后合并dev分支到master分支，这种情况一般有冲突。

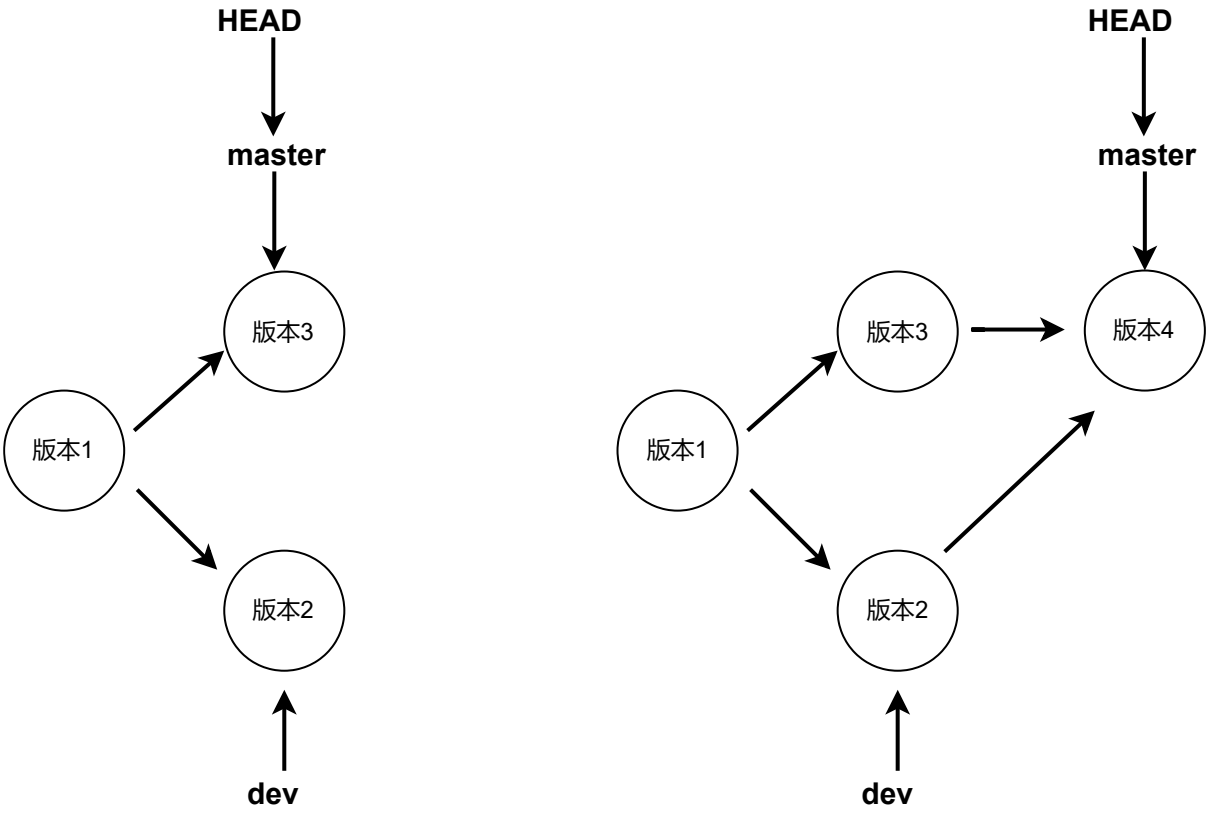

图11

如图11所示，在版本1创建了dev分支，然后dev分支做了版本2提交，然后切换到master分支做了版本3提交，此时再把dev分支合并到master分支，合并结果如图11右所示，此时会创建一个新的合并提交版本，然后把master指针和HEAD指针移动到该新的合并提交，不过在合并提交之前一般需要我们先解决文件冲突问题，下一小节讲解冲突解决问题。

解决冲突

```
work@work-virtual-machine:~/git.source/user4$ git log
commit 528298c5f7399ba0ab9ca16f41ba474469a99375 (HEAD -> dev)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:12:03 2020 +0800

    1st dev commit

commit a2484358f456fb42be4e01850003b3e6a1586a37 (master)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:09:16 2020 +0800

    1st master commit
```

图12

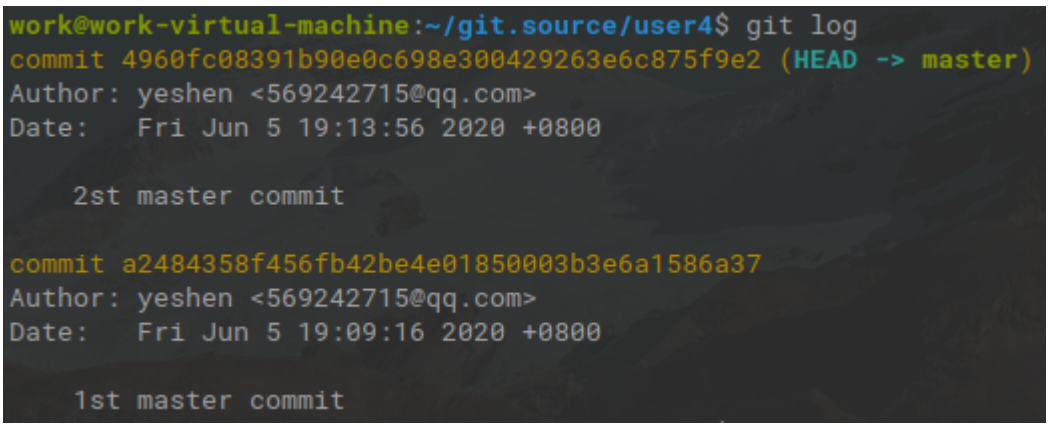

图13

图14

如图11，12和13，提交记录1st master commit对应的是版本1，1st dev commit对应的是版本2，2st master commit对应的是版本3。然后尝试通过git merge dev将dev分支合并到master分支，此时产生冲突的文件内容如图14所示。

```
work@work-virtual-machine:~/git.source/user4$ git add .
work@work-virtual-machine:~/git.source/user4$ git merge --continue
[master 2b2a724] Merge branch 'dev' merge by 2st master and 1st dev
work@work-virtual-machine:~/git.source/user4$ git log
commit 2b2a724c65e10ba6a1069389822def55f973ecc1 (HEAD -> master)
Merge: 4960fc0 528298c
Author: yeshen <569242715@qq.com>
Date:   Sat Jun 6 08:46:45 2020 +0800

    Merge branch 'dev'
    merge by 2st master and 1st dev

commit 4960fc08391b90e0c698e300429263e6c875f9e2
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:13:56 2020 +0800

    2st master commit

commit 528298c5f7399ba0ab9ca16f41ba474469a99375 (dev)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:12:03 2020 +0800

    1st dev commit

commit a2484358f456fb42be4e01850003b3e6a1586a37
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:09:16 2020 +0800

    1st master commit
```

图15

```
work@work-virtual-machine:~/git.source/user4$ git checkout dev
切换到分支 'dev'
work@work-virtual-machine:~/git.source/user4$ git log
commit 528298c5f7399ba0ab9ca16f41ba474469a99375 (HEAD -> dev)
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:12:03 2020 +0800

    1st dev commit

commit a2484358f456fb42be4e01850003b3e6a1586a37
Author: yeshen <569242715@qq.com>
Date:   Fri Jun 5 19:09:16 2020 +0800

    1st master commit
work@work-virtual-machine:~/git.source/user4$ cat test.txt
1st
dev
work@work-virtual-machine:~/git.source/user4$ git checkout master
切换到分支 'master'
work@work-virtual-machine:~/git.source/user4$ cat test.txt
2st
master
1st
dev
```

图16

在修改了冲突文件后，先git add .，然后git merge --continue继续合并，此时合并成功如图15和16所示，查看提交记录在master分支多处了一个新的合并产生的版本提交，而切换到dev分支查看提交记录和合并前一致，分别查看dev分支和master分支的文件，dev分支和合并前内容一致，而master分支是合并产生冲突后经过人为修改后得到的内容。

版本穿梭

版本穿梭主要有如下3种方式，第一种方式将HEAD和HEAD指向的分支指针向前移动一次，如果要向前移动两次则git reset --hard HEAD^^，如此类推；第二种方式可以将HEAD和HEAD指向的分支指针向前移动n次;前两种方式只能向前移动，不能向后移动，而第三种方式可以任意方向穿梭，只要知道版本提交id，此时需要git reflog获得之前的提交信息，然后再通过git reset来穿梭。

(1) $ git reset --hard HEAD^

(2)$ git reset --hard HEAD~n

(3) $ git reset --hard commitid前几位

五、和gitlab联合使用

5.1 初始化

本地git clone获取gitlab服务器的master（不会获取其他分支）

gitlab服务器新建分支然后创建合并请求

本地git branch 分支名 创建本地分支（或者git checkout -b 分支名创建+切换）

本地git rebase master 将master合并到当前分支

5.2 运行过程

本地git pull <远程主机名> <远程分支名>:<本地分支名> （一般是先拉master分支到本地master，然后git rebase）

git add 和git commit

git push

gitlab合并请求