

# 产品需求

产品：编写一个go计算器

- 加法
- 减法
- 乘法
- 除法

项目划分开发阶段：

P1:阶段

时间：2019年10 ~ 2019年12月31

功能：

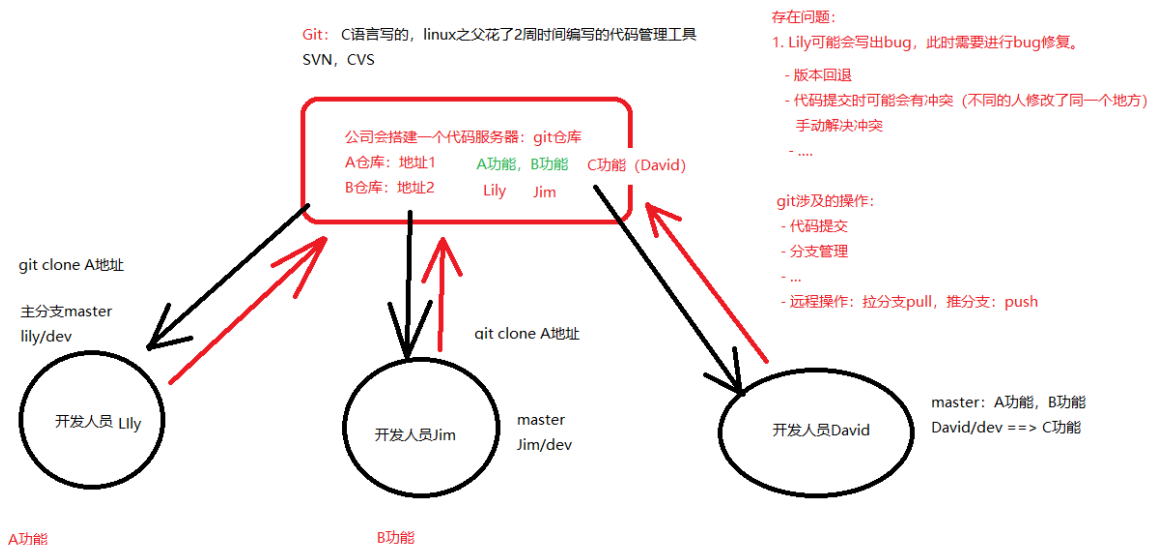
1. 加法
2. 减法

P2:阶段

时间：2020年2 ~ 2020年4月

功能：

1. 乘法
2. 除法



# 一、基本命令

---

## 1. 加法功能

---

calc/add.go

```
package calc

func Add(a, b int) int {
    return a + b
}
```

main.go

```
package main

import (
    "fmt"
    "go5期/gitTest/calc"
)

func main() {
    res := calc.Add(10, 20)

    fmt.Println("Add(10 ,20) :", res)
}
```

## 2. 创建仓库

---

```
git init
```

1. 将当前的目录变成一个代码仓库
2. 可以有很多个仓库，多个仓库之间是独立的，无法相互提交代码

.git文件夹

## 3. 查看当前状态

---

```
git status
```

此时，会看到main.go，calc是红色标识的，说明需要处理

## 4. git追踪代码（暂存区）

```
git add main.go calc
```

## 5.查看当前状态

```
git status
```

此时，main.go，calc会变成绿色的，说明已经添加到暂存区

## 6. 提交代码到本地仓库

```
git commit
```

第一次提交，可能会遇到下面的提示

问题1：设置用户信息：

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

请替换为自己的名字和邮箱，告知git系统提交人的信息

问题2：编辑器不是vim，需要执行如下命令，配置成vim后，重新commit

```
git config --global core.editor "vim"
或
export GIT_EDITOR=vim
```

此时会弹出vim界面，需要添加本次的注释，保存退出

## 7.查看当前状态

```
git status
```

main.go和calc不见了，红、绿都不见了，说明本次提交成功了。

## 8.查看提交日志

```
git log
```

代码问责查看

```
git blame main.go
```

```
^dc73475 (Your Name 2019-11-07 09:00:18 +0800 7)
^dc73475 (Your Name 2019-11-07 09:00:18 +0800 8) func main() {
d453b7ab (duke 2019-11-07 10:48:09 +0800 9)     fmt.Println("calc.Add called!")
^dc73475 (Your Name 2019-11-07 09:00:18 +0800 10)     res := calc.Add(10, 20)
d453b7ab (duke 2019-11-07 10:48:09 +0800 11)     fmt.Println("Add(10 ,20) :", res)
d453b7ab (duke 2019-11-07 10:48:09 +0800 12)
d453b7ab (duke 2019-11-07 10:48:09 +0800 13)     fmt.Println("calc.Sub called!")
d453b7ab (duke 2019-11-07 10:48:09 +0800 14)     res = calc.Sub(30, 20)
d453b7ab (duke 2019-11-07 10:48:09 +0800 15)     fmt.Println("Sub(10 ,20) :", res)
d453b7ab (duke 2019-11-07 10:48:09 +0800 16)     //fmt
^dc73475 (Your Name 2019-11-07 09:00:18 +0800 17) }
```

## 当前问题：

1. git命令是可以自动补全，如果不能补全，需要配置一下（git-bash-complete.sh，自己查找）
  1. 不要全部手巧
  2. 慢，容易出错
2. 如果第一次提交，会要求配置提交人的信息，这样后续可以跟踪某个代码时谁提交的，用于问责
  1. git blame <文件>

```
git config --global user.email "duke@itcast.cn"
git config --global user.name "duke"
```

3. git一般使用vim作为commit时的编辑器，如果不是，请配置一下：

```
git config --global core.editor "vim"
或
export GIT_EDITOR=vim
```

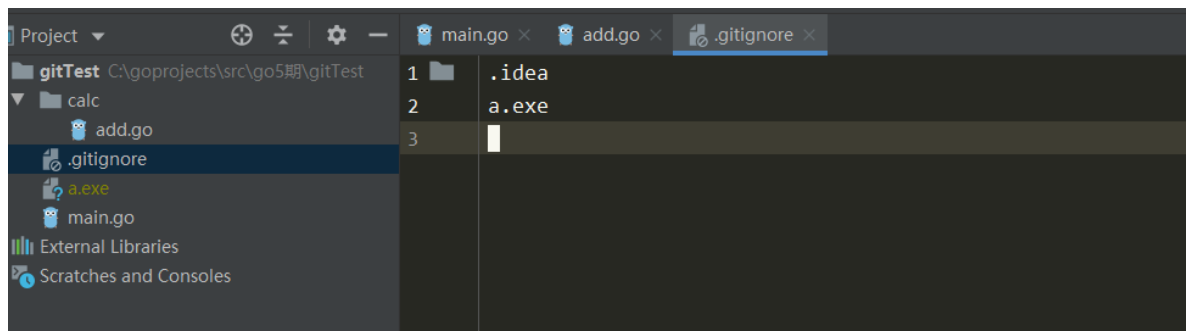
4. 执行git init时，他会在当前的目录创建一个代码仓库，不同的文件夹仓库不同。多个仓库之间是相互独立的
  1. 前端仓库
  2. 后端的仓库
    1. 模块1仓库 ==》 A开发人员
    2. 模块2仓库 ==》 B开发人员

## 9. .gitignore忽略文件

在当前文件夹下存在.idea的文件夹，它并不是我们的代码，我们不想提交，也不想总看见它的提示，为了避免误提交，可以将这个文件（夹）添加到一个特定的文件中：.gitignore

```
.idea
```

如图：



需要将.gitignore添加到仓库

1. git add .gitignore
2. git commit .gitignore

两种添加commit备注的方式:

1. git commit ,回车, 进入vim模式, 填写描述 ==》 适合于注释较多的情况
2. git commit -m "添加.gitignore文件" ==》 适合备注较少的情况

修改后的文件: 提交/丢弃

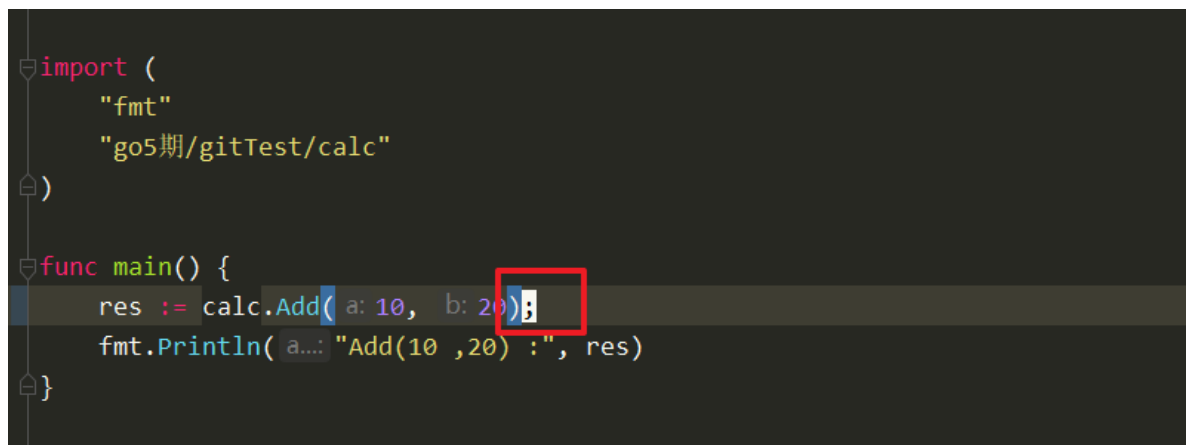
1. git add <文件> ==>提交本次修改
2. git checkout <文件> ==》 丢弃本次修改

在.gitignore中修改内容, 查看文件状态, 会发现提示文件被修改:

```
git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore
```

如果这是一个误操作, 使用status发现文件被修改了。



```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore
        modified:   main.go

no changes added to commit (use "git add" and/or "git commit -a")
```

## 10.如何知道修改了哪些内容

```
git diff ==> 查看所有被修改的文件，打印对比信息
git diff main.go ==>指定文件查看
```

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git diff main.go
warning: LF will be replaced by CRLF in main.go.
The file will have its original line endings in your working directory
diff --git a/main.go b/main.go
index 5c99f17..42fc9fc 100644
--- a/main.go
+++ b/main.go
@@ -6,7 +6,6 @@ import (
 )

 func main() {
-     res := calc.Add(10, 20)
-
+     res := calc.Add(10, 20);
     fmt.Println("Add(10 ,20) :", res)
 }

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
```

## 11.添加/丢弃修改

如果想提交本次的修改，在使用git add 提交。

```
git add main.go
git commit -m "xxxxx"
```

如果这是误操作，或者想将当前的修改丢弃掉，使用git checkout <文件名字>

```
git checkout main.go
```

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
```

main.go的修改已经被丢弃，还原为原始状态

## 二、分支管理

### 1.增加加法功能

基于master分支创建一个新的分支

一个新功能，一个新分支 duke/add, duke/sub, duke/multi, duke/div

一个bug，一个解决分支 duke/bug-1137

```
git branch duke/dev //创建分支
git checkout duke/dev //切换分支
```

或者，创建的时候，直接切换

```
git checkout -b duke/dev1 // <== 推荐
```

添加代码：

```
git checkout -b duke/sub
```

calc/sub.go

```
package calc

func Sub(a, b int) int {
    return a - b
}
```

```
package main

import (
    "fmt"
    "go5期/gitTest/calc"
)

func main() {
    fmt.Println("calc.Add called!")
    res := calc.Add(10, 20)
```

```

    fmt.Println("Add(10 ,20) :", res)

    fmt.Println("calc.Sub called!")
    res = calc.Sub(30, 20)
    fmt.Println("Sub(10 ,20) :", res)
    //fmt
}

```

提交代码最基本的原则：

- 确保编译通过

提交代码：

1. 一次性将当前目录的所有文件提交

```
git add . //<<== 强烈不推荐，可能会错误的提交一些不必要的文件
```

2. 建议单个文件，指定名字提交

```

git add calc/sub.go main.go //<== 推荐
git commit -m "添加sub函数"

```

如果不小心提交了不想提交的文件，可以使用git reset HEAD <文件>进行撤销

示例：

```

$ git status
On branch duke/dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   calc/sub.go
    new file:   gitTest.exe
    modified:   main.go
git add . 错误的提交了gitTest.exe文件

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/dev)
$ git reset HEAD gitTest.exe
使用reset命令，将gitTest.exe撤销提交

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/dev)
$ git status
On branch duke/dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   calc/sub.go
    modified:   main.go

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    gitTest.exe
gitTest.exe
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/dev)
$

```

## 2. 合并分支

将duke/dev分支合并到主分支



然后执行合并动作

```
git checkout master  
git merge duke/dev //此时，将duke/dev的代码合并到master
```

查看当前已经提交的内容，

```
git log -p //<== 强烈推荐
```

### 3. 删除分支

---

如果想删除duke/dev，必须切换到其他分支，合并到master之后（2周之后再删除）

```
git branch -d duke/dev //由于duke/dev已经合并到了master，所以删除成功
```

如果有一个新分支duke/dev1，我们在其中做了修改和提交，但是没有合并到其他分支。那么如果想删除，需要使用-D才可以，否则报错。git系统对我们误操作的防护。

```
git branch -D duke/dev1 //-D用于未合并过的分支删除。
```

只有合并到了master之后，才能够使用-d删除，否则都是-D

### 4. 删除文件

---

两种方式：

方式1：

1. rm gitTest.exe文件
2. git rm gitTest.exe 或 git add gitTest.exe
3. git commit -m "rm gitTest.exe"

```

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ rm gitTest.exe

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    gitTest.exe

no changes added to commit (use "git add" and/or "git commit -a")

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git rm gitTest.exe
rm 'gitTest.exe'

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    gitTest.exe

此时已经可以进行commit了

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git reset HEAD
Unstaged changes after reset:
D    gitTest.exe

```

方式2:

1. git rm gitTest.exe
2. git commit -m "rm gitTest.exe"

```

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git rm gitTest.exe
rm 'gitTest.exe'

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

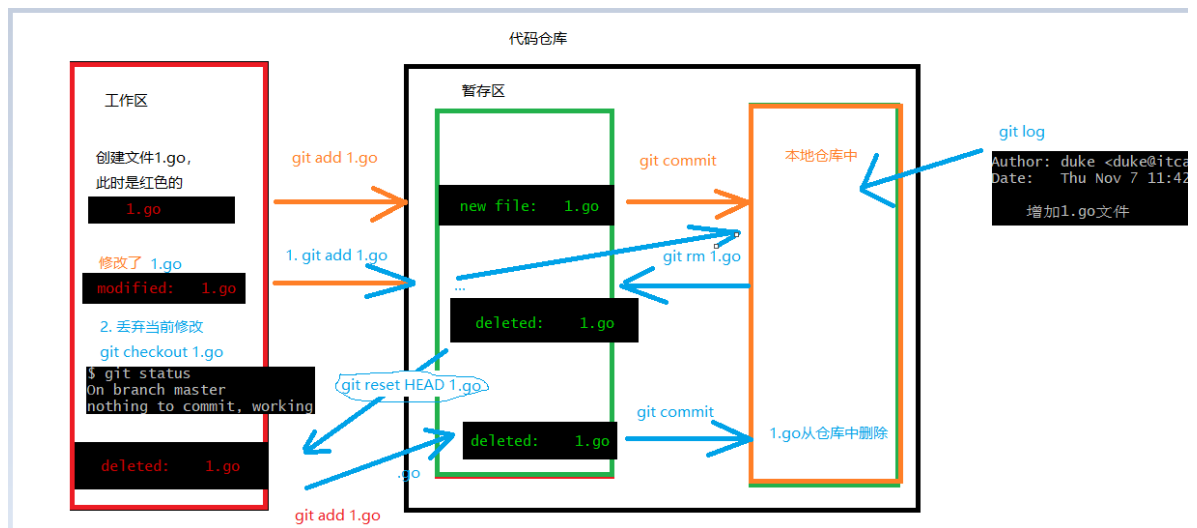
        deleted:    gitTest.exe

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git commit -m "delete gitTest.exe"
[master e849ffd] delete gitTest.exe
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 gitTest.exe

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git log
commit e849ffdc5612aa4884d1248a429e9af22d9f4e8f (HEAD -> master)

```

## 5. 工作区、暂存区介绍 (重要)



## 6. 合并冲突问题merge

duke: 乘法

lily :除法

基于master创建两个分支

```
git branch duke/multi
git branch lily/div
```

切换到duke/multi分支

```
git checkout duke/multi
```

实现代码:

```
package calc

func Multi(a, b int) int {
    return a * b
}
```

main.go

```
package main

import (
    "fmt"
    "go5期/gitTest/calc"
)

func main() {
```

```

fmt.Println("calc.Add called!")
res := calc.Add(10, 20)
fmt.Println("Add(10 ,20) :", res)

fmt.Println("calc.Sub called!")
res = calc.Sub(30, 20)
fmt.Println("Sub(10 ,20) :", res)
//fmt

fmt.Println("calc.Multi called!")
res = calc.Multi(30, 20)
fmt.Println("Multi(10 ,20) :", res)
}

```

提交!

切换到lily/div分支

```

package calc

func Div(a, b int) int {
    if b == 0 {
        panic("除数不应为0")
    }
    return a / b
}

```

main.go

```

package main

import (
    "fmt"
    "go5期/gitTest/calc"
)

func main() {
    fmt.Println("calc.Add called!")
    res := calc.Add(10, 20)
    fmt.Println("Add(10 ,20) :", res)

    fmt.Println("calc.Sub called!")
    res = calc.Sub(30, 20)
    fmt.Println("Sub(10 ,20) :", res)
    //fmt

    fmt.Println("calc.Div called!")
    res = calc.Div(100, 20)
    fmt.Println("Div(100 ,20) :", res)
}

```

提交!

此时，两个分支修改main函数中的同一处代码。

切换到master分支

分别合并两个分支

```
git merge duke/multi
```

合并成功

```
git merge lily/div
```

此时，会发生冲突：

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git merge duke/multi
Auto-merging main.go
CONFLICT (content): Merge conflict in main.go
Automatic merge failed; fix conflicts and then commit the result.

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  new file:   calc/mult.go

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   main.go
```

查看冲突文件：

```
<<<<<< HEAD
    fmt.Println(a...: "calc.Div called!")
    res = calc.Div(a: 100, b: 20)
    fmt.Println(a...: "Div(100 ,20) :", res)
=====
    fmt.Println(a...: "calc.Multi called!")
    res = calc.Multi(a: 30, b: 20)
    fmt.Println(a...: "Multi(10 ,20) :", res)
>>>>>> duke/multi
}
```

当前分支的代码

合并分支的代码

人工判断哪些该保留：全部保留

```

fmt.Println(a...: Sub(10,20) :", res)
//fmt

fmt.Println(a...: "calc.Div called!")
res = calc.Div(a: 100, b: 20)
fmt.Println(a...: "Div(100,20) :", res)

fmt.Println(a...: "calc.Multi called!")
res = calc.Multi(a: 30, b: 20)
fmt.Println(a...: "Multi(10,20) :", res)
}

```

两部分全部保留

修改之后，重写提交文件

```

git add main.go
git commit

```

合并完成!

```

commit 0925ed4fd290a17dc773773c726bf643c90cd6d2 (HEAD -> master)
Merge: 45bee68 feb5c78
Author: duke <duke@itcast.cn>
Date: Thu Nov 7 15:00:06 2019 +0800

    Merge branch 'duke/multi'

commit 45bee68a54027bc0bbba91c61abff5d94d6b5b23 (lily/div)
Author: duke <duke@itcast.cn>
Date: Thu Nov 7 14:56:41 2019 +0800

    实现除法Div

commit feb5c782c3dc829c4f9b007d7f1acdead6cc40d1 (duke/multi)
Author: duke <duke@itcast.cn>
Date: Thu Nov 7 14:52:28 2019 +0800

```

## 7.合并方式2: rebase

当多个分支合并时，尽量使用rebase命令，

1. 可以使得提交log按时间线排列
2. 减少Merge branch "duke/multi"等log日志出现，更加简洁

操作流程：

1. git rebase duke/multi
2. 如果发生冲突，需要手动解决（同merge冲突）
3. 如果不想提交，可以使用git rebase ----abort
4. 如果想提交，解决冲突后，使用git rebase --continue即可，不需要使用git commit

## 8. 暂存stash

当开发过程中，还没有办法编译的时候，可能临时出现一个需要紧急处理的bug：

1. 当前代码不能提交
2. bug还需要创建新的分支解决。

需要，将当前分支的代码暂存起来，去解决bug，解决之后，再唤醒之前的存放代码，继续开发。

git stash命令可以帮忙解决

增加mod.go

```
package calc

import "fmt"

func mod() {
    fmt.Println("mod called")
    fmt
}
```

使用stash进行暂存

```
git stash
```

当前分支的所有暂存区的数据会被临时存储起来，分支就像没有修改过一样。

可以基于分支解决bug，当bug解决后使用git stash pop 唤醒暂存数据

```
git stash pop
```

查看当前stash数据

```
git stash list
```

如果确定不要的stash，可以使用clear清理掉(慎用)

```
git stash clear
```

## 9. 版本回退

1. git reset <哈希值>

1. 每一个提交都会有一个哈希值：1c9b43f23f48777454f288c3cddcdee122fede1d

2. 此时会将版本回退到指定哈希版本，但是回退中间提交的代码会保留下来

3. 推荐使用这个模式

2. git reset --hard <哈希值>

1. 此时会直接会推到指定版本，但是中间提交的记录，会之前删除掉。

3. git reflog，此时会显示出这个分支所有的操作

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/multi)
$ git reflog
dc73475 (HEAD -> duke/multi) HEAD@{0}: reset: moving to dc734759613b7bdbd980f770c
9a7c82e HEAD@{1}: reset: moving to 9a7c82ee8fdbf773384dfc596301dc67edf0e926
faabacf (duke/bug-1137) HEAD@{2}: reset: moving to faabacf95e8f44026217d95b8d4b21
1c9b43f HEAD@{3}: commit: 增加mod方法
faabacf (duke/bug-1137) HEAD@{4}: checkout: moving from master to duke/multi
b35cf18 (master) HEAD@{5}: checkout: moving from duke/multi to master
faabacf (duke/bug-1137) HEAD@{6}: reset: moving to HEAD
faabacf (duke/bug-1137) HEAD@{7}: checkout: moving from duke/bug-1137 to duke/multi
faabacf (duke/bug-1137) HEAD@{8}: checkout: moving from duke/multi to duke/bug-1137
faabacf (duke/bug-1137) HEAD@{9}: checkout: moving from master to duke/multi
b35cf18 (master) HEAD@{10}: rebase finished: returning to refs/heads/master
b35cf18 (master) HEAD@{11}: rebase: 在master中增加Div注释
e4b0f91 HEAD@{12}: rebase: 实现除法Div
5ae403a HEAD@{13}: rebase: 在mult中增加注释
b2da3ef HEAD@{14}: rebase: 实现mult函数
d3eb7eb (lily/div) HEAD@{15}: rebase: checkout lily/div
8b4abbb HEAD@{16}: rebase: updating HEAD
d3eb7eb (lily/div) HEAD@{17}: rebase: checkout lily/div
8b4abbb HEAD@{18}: rebase finished: returning to refs/heads/master
8b4abbb HEAD@{19}: rebase: 在master中增加Div注释
d99eb4c HEAD@{20}: rebase: 实现除法Div
faabacf (duke/bug-1137) HEAD@{21}: rebase: checkout duke/multi
e2236b7 HEAD@{22}: commit: 在master中增加Div注释
0925ed4 HEAD@{23}: checkout: moving from lily/div to master
d3eb7eb (lily/div) HEAD@{24}: commit: 增加Div注释
45bee68 HEAD@{25}: checkout: moving from duke/multi to lily/div
faabacf (duke/bug-1137) HEAD@{26}: commit: 在mult中增加注释
feb5c78 HEAD@{27}: checkout: moving from lily/div to duke/multi
45bee68 HEAD@{28}: checkout: moving from master to lily/div
0925ed4 HEAD@{29}: checkout: moving from lily/div to master
45bee68 HEAD@{30}: checkout: moving from duke/multi to lily/div
feb5c78 HEAD@{31}: checkout: moving from master to duke/multi
0925ed4 HEAD@{32}: commit (merge): Merge branch 'duke/multi'
```

版本回退是，都是对当前的分支进行处理的。

## 10.基于历史哈希创建新分支

有时，基于当前的分支，指定某一个版本号，创建一个新的分支


场景1：临时使用

```
git checkout e849ffdc5612aa4884d1248a429e9af22d9f4e8f
```

此时，会创建一个临时的分支，当前切换为其他分支后，这个临时分支消失



```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest ((9a7c82e...))
$ git branch
* (HEAD detached at 9a7c82e)
  duke/bug-1137
  duke/multi
  lily/div
  master
```



场景2：基于这个分支创建一个作为后续开发使用的分支

```
git checkout -b duke/test e849ffdc5612aa4884d1248a429e9af22d9f4e8f
```

```
duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git checkout -b duke/test c38cf00bd5a1816d5a671aa03c2abe7fff1e0e7a
Switched to a new branch 'duke/test'

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/test)
$ git branch
  duke/bug-1137
  duke/multi
* duke/test
  lily/div
  master

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (duke/test)
$ git checkout master
Switched to branch 'master'

duke@DUKEDU51C6 MINGW64 /c/goprojects/src/go5期/gitTest (master)
$ git branch
  duke/bug-1137
  duke/multi
  duke/test
  lily/div
* master
```

