

哈尔滨工业大学 计算学部
2024 年秋季学期《开源软件开发实践》
Lab 1: Git 实战

姓名	学号	联系方式
张皓涵	2022111897	2123228124@qq.com/15134623385

目 录

1 实验要求	1
2 安装 Git	1
2.1 本地机器上安装 git	1
2.2 申请 github 帐号	2
3 Git 操作过程	3
3.1 实验场景(1): 仓库创建与提交	3
3.2 实验场景(2): 分支管理	4
3.3 实验场景(3): 在线 Git 练习	11
4 小结	18

[文档全部完成之后, 请更新上述区域]

1 实验要求

熟练掌握 Git 的基本指令和分支管理指令；

掌握 Git 支持软件配置管理的核心机理；

使用 Git/Github 管理自己的项目源代码。

本次实验由个人单独完成。

实现本地仓库创建与提交

完成分支管理

完成在线练习

2 安装 Git

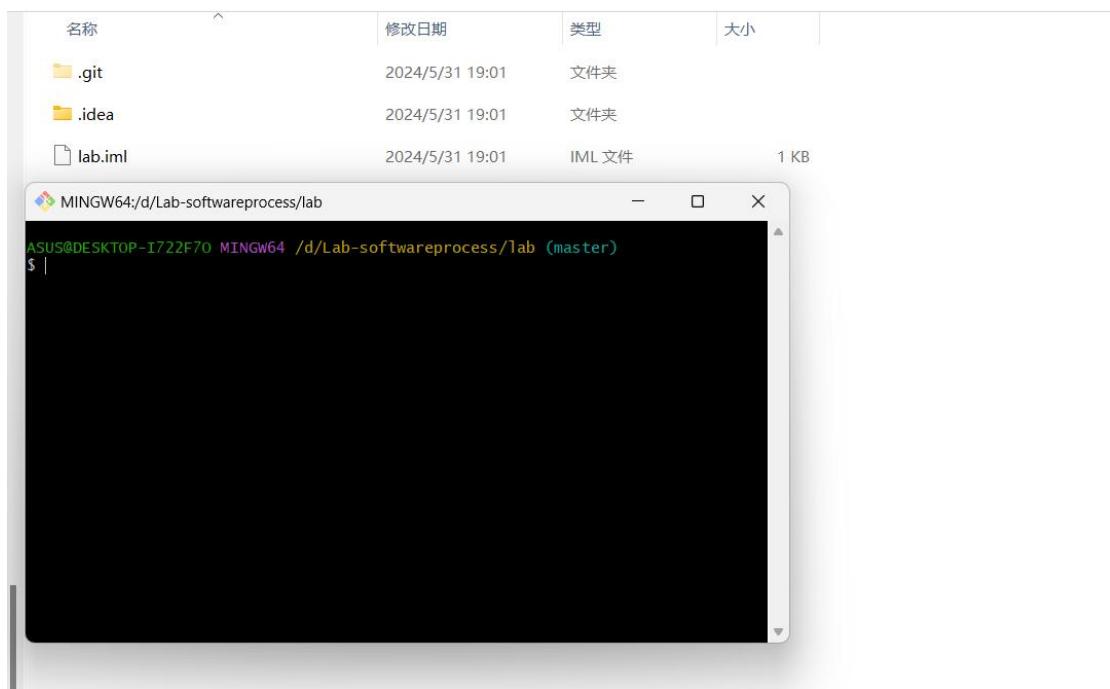
2.1 本地机器上安装 git

Git 版本号：



```
C:\Windows\system32\cmd.exe: x + v
Microsoft Windows [版本 10.0.22631.3593]
(c) Microsoft Corporation。保留所有权利。
C:\Users\ASUS>git --version
git version 2.44.0.windows.1
C:\Users\ASUS>
```

Git 提交界面：



2.2 申请 github 帐号

本人此次实验采用的是 GitHub

账号名称: HIT-2022111897-张皓涵

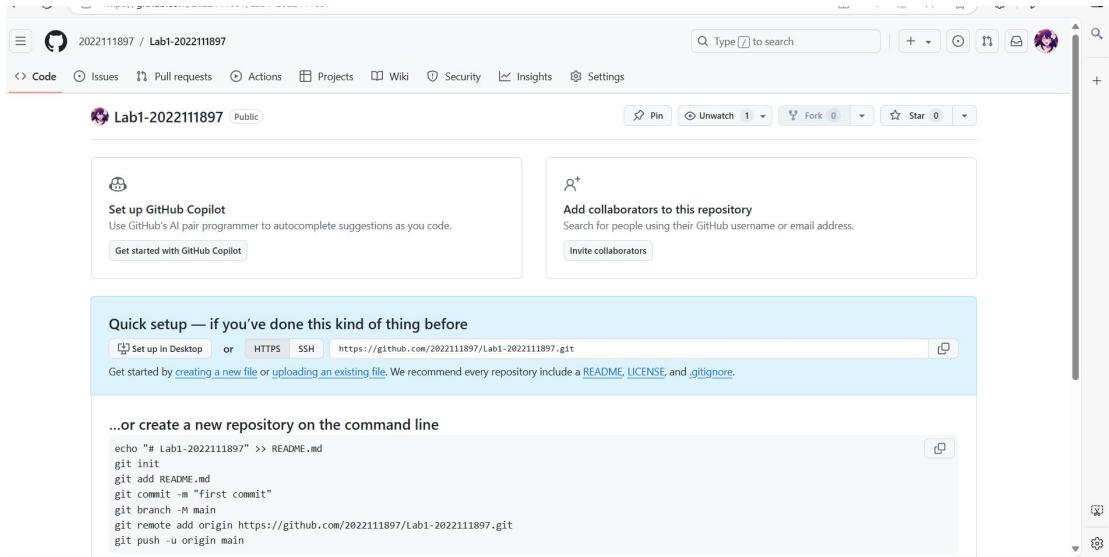
URL 地址:

lab 文件操作的仓库: <https://github.com/2022111897/Lab1-2022111897.git>

账号信息的截图:

A screenshot of a GitHub profile page for the user 'HIT-2022111897-张皓涵'. The profile picture is a purple anime girl. The bio reads 'HIT-2022111897-张皓涵 2022111897'. Below the bio is a 'Edit profile' button. The top navigation bar includes 'Overview', 'Repositories 9', 'Projects', 'Packages', and 'Stars'. The main content area shows 'Popular repositories' with cards for 'Social-Network-system', 'Spring2022_HITCS_SC_Lab1', 'branch-practice', 'test', 'softwareprocess', and 'sns-'. To the right is a 'Customize your pins' section. Below the repositories is a chart titled '27 contributions in the last year' showing activity from November of the previous year to October. A sidebar on the right displays '13.3 GB垃圾' (13.3 GB trash) with a warning about clearing it to prevent slowing down the computer. There are also '360安全大脑' (360 Security Brain) and '立即清理' (Clean Now) buttons.

项目信息截图:



3 Git 操作过程

3.1 实验场景(1): 仓库创建与提交

给出 1.1~1.9 的操作命令，并给出执行界面的截图（命令输入界面和结果界面）

(1) R0: 针对 R1 和 R7，在进行每次 Git 操作之前，随时查看工作区、暂存区、Git 仓库的状态，确认项目里的各文件当前处于什么状态。

指令: git status

说明: 这个指令在未进行本地仓库初始化前是无法生效的，下面分别是本人在初始化前和初始化后执行该指令得到的结果。

初始化前：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git
$ git status
fatal: not a git repository (or any of the parent directories): .git
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$
```

(2) **R1:** 本地初始化一个 Git 仓库，将自己在 Lab1 中所创建项目的全部源文件加入进去，纳入 Git 管理。

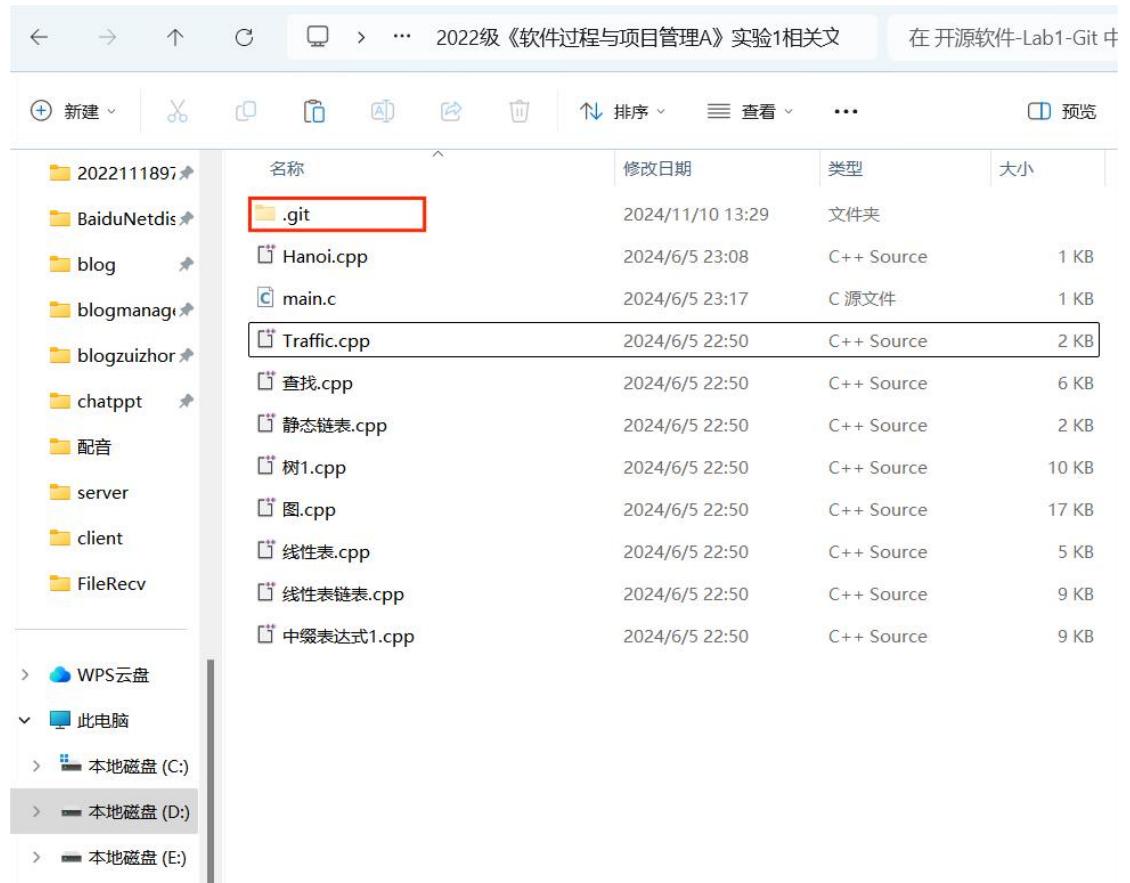
指令: git init

执行命令:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git init
Reinitialized existing Git repository in D:/开源软件-Lab1-Git/.git/

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$
```

执行结果: 在项目文件夹下生成一个名为.git 的隐藏文件夹，需要将 Windows 资源管理器中的“显示隐藏的项目”勾选才能生效。



文件夹中有 10 个文件，故符合要求

(3) R2: 提交

指令：将文件加入暂存区 git add 文件名

将暂存区的文件提交到本地仓库：git commit -m “当前提交备注的信息”

执行结果：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git commit -m "first commit"
[master (root-commit) 787d529] first commit
 10 files changed, 2533 insertions(+)
  create mode 100644 Hanoi.cpp
  create mode 100644 Traffic.cpp
  create mode 100644 main.c
  create mode 100644 "\344\270\255\347\274\200\350\241\250\350\276\276\345\274\21
71.cpp"
  create mode 100644 "\345\233\276.cpp"
  create mode 100644 "\346\237\245\346\211\276.cpp"
  create mode 100644 "\346\240\2211.cpp"
  create mode 100644 "\347\272\277\346\200\247\350\241\250.cpp"
  create mode 100644 "\347\272\277\346\200\247\350\241\250\351\223\276\350\241\25
0.cpp"
  create mode 100644 "\351\235\231\346\200\201\351\223\276\350\241\250.cpp"

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$
```

这里补一下 git add . 的状态：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
nothing to commit, working tree clean

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
```

(4) R3：手工对 lab 的某个文件进行修改，查看上次提交之后都有哪些文件修改、具体修改内容是什么(查看修改后的文件和暂存区域中相应文件的差别)。

首先手动对 lab 的文件 main.c 进行修改，只在原来的文件中添加了一行注释。

```
C main.c 3 ●
D: > 开源软件-Lab1-Git > C main.c > ...
3 void Bubblesort(int a[],int size)
10 {
11     if(a[j]>a[j+1])
12     {
13         temp=a[j];
14         a[j]=a[j+1];
15         a[j+1]=temp;
16         count=1;
17     }
18 }
19 if(count==0)
20 {
21     break;
22 }
23 }
24 }
25 //modified for lab R3
26 int main()
27 {
28     int a[10]={1,5,9,7,8,4,6,3,2,0};
29     printf("1 2 3 4 5 6 7 8 9 10\n");
30     for(int i=0;i<10;i++)
31     {
32         printf("%d ",a[i]);
33     }
34     Bubblesort(a,10);
35     printf("1 2 3 4 5 6 7 8 9 10\n");
36     for(int i=0;i<10;i++)
37     {
38         printf("%d ",a[i]);
39     }
40     return 0;
41 }
```

查看文件修改状况: git status
执行结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   main.c

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

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$
```

查看具体变动内容: git diff
执行结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git diff
diff --git a/main.c b/main.c
index 7ddf9d8..f822ff8 100644
--- a/main.c
+++ b/main.c
@@ -22,16 +22,17 @@ void Bubblesort(int a[],int size)
}
}
}
+//modified for lab R3
int main()
{
    int a[10]={1,5,9,7,8,4,6,3,2,0};
-   printf("<CA><E4><C8><EB>10<B8><F6><CA><FD>I<A3><BA>");
+   printf("*****10*****I***");
    for(int i=0;i<10;i++)
    {
        printf("%d ",a[i]);
    }
    Bubblesort(a,10);
-   printf("0<C5><DD><C5><D0><F2><BA><F3>I<A3><BA>\n");
+   printf("0*****I**\n");
```

可以发现使用 git status 指令后结果中将“main.c”文件被修改的结果标红了，随后使用 git diff 指令，结果中标记了“+ //modified for Lab R3 step”字样，说明之前的修改已经被 Git 工具发现。

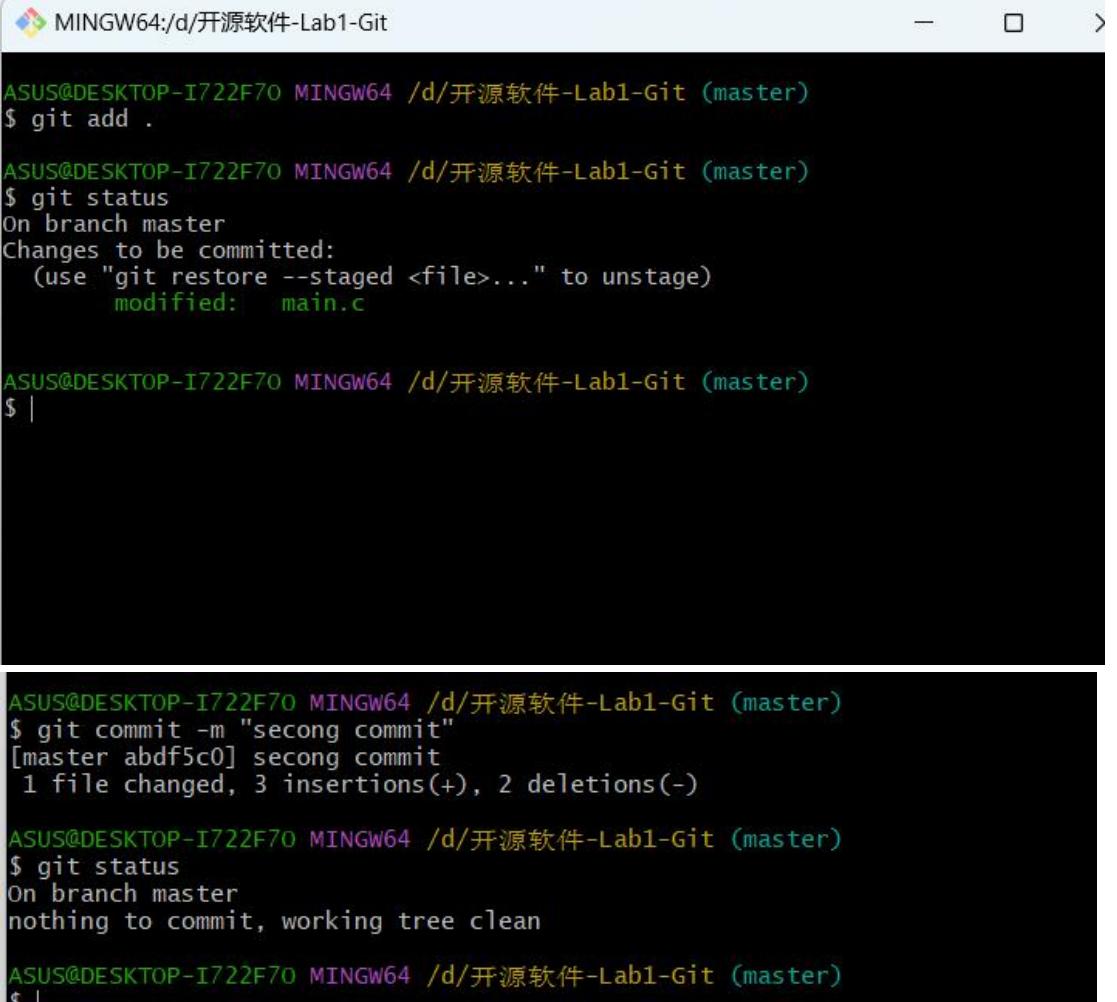
(5)R4: 重新提交

这一步的指令同 R2

将文件加入暂存区: git add 文件名

将暂存区的文件提交到本地仓库: git commit -m “当次提交的备注信息”

执行结果:



```

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   main.c

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git commit -m "secong commit"
[master abdf5c0] secong commit
 1 file changed, 3 insertions(+), 2 deletions(-)

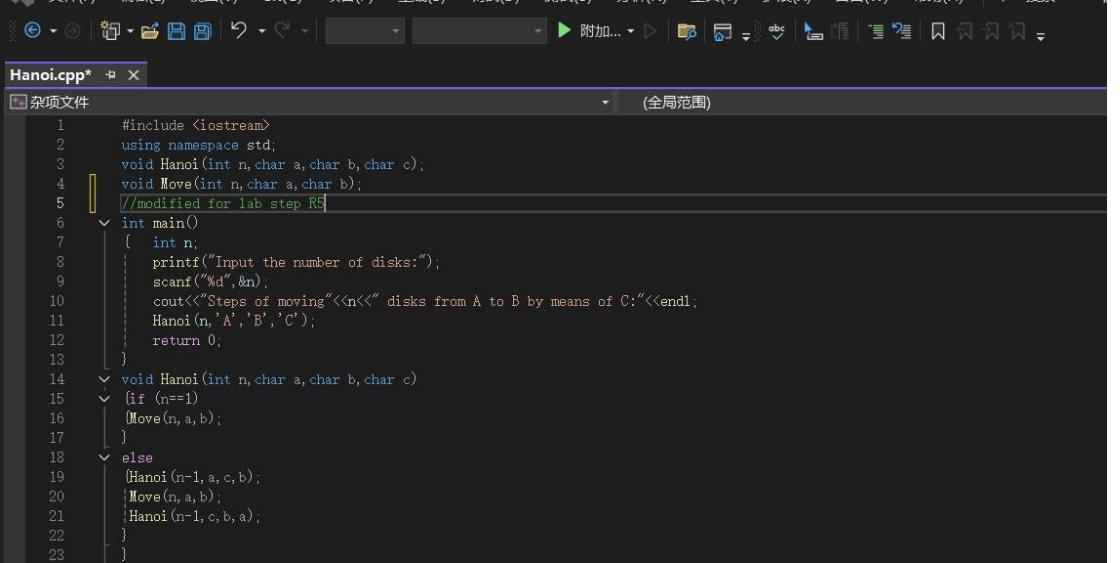
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
nothing to commit, working tree clean

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |

```

(6)R5: 再次对 lab 的一个文件进行修改，重新提交

这一步的指令和前一步没有太大区别，这里本人为了方便，使用“Hanoi.cpp”文件进行展示



```

Hanoi.cpp* ✘
杂项文件 (全局范围)
1 #include <iostream>
2 using namespace std;
3 void Hanoi(int n,char a,char b,char c);
4 void Move(int n,char a,char b);
5 //modified for lab step R5
6 int main()
7 {
8     int n;
9     printf("Input the number of disks:");
10    scanf("%d",&n);
11    cout<<"Steps of moving"<<n<<" disks from A to B by means of C:"<<endl;
12    Hanoi(n,'A','B','C');
13    return 0;
14 }
15 void Hanoi(int n,char a,char b,char c)
16 {
17     if (n==1)
18     {
19         Move(n,a,b);
20     }
21     else
22     {
23         Hanoi(n-1,a,c,b);
24         Move(n,a,b);
25         Hanoi(n-1,c,b,a);
26     }
27 }

```

使用的指令仍然是：git add 文件名和 git commit -m “当次提交的

备注信息”。执行结果如下：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   Hanoi.cpp

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git commit -m "third commit"
[master 78514bb] third commit
 1 file changed, 1 insertion(+)

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git status
On branch master
nothing to commit, working tree clean

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |
```

(7)R6：把最后一次提交撤销

指令：git reset --hard HEAD^

首先可以执行 git reflog 查看历史的提交版本情况，执行结果如下：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git reflog
78514bb (HEAD -> master) HEAD@{0}: commit: third commit
abdf5c0 HEAD@{1}: commit: secong commit
787d529 HEAD@{2}: commit (initial): first commit

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |
```

可以观察到包括初始化的提交一共提交了三次，并且目前的 HEAD 指向的是最后一次提交。

现在执行撤销指令：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git reset --hard HEAD^
HEAD is now at abdf5c0 secong commit
```

再次执行 git reflog 查看版本情况，，特别注意版本的序列号：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git reflog
abdf5c0 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
78514bb HEAD@{1}: commit: third commit
abdf5c0 (HEAD -> master) HEAD@{2}: commit: secong commit
787d529 HEAD@{3}: commit (initial): first commit

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |
```

观察到撤销操作后版本的序列号从“78514bb”变成了“abdf5c0”,这与第二次提交的序列号一致，说明撤销操作成功。查看对应的文件，发现第二次修改后的结果被撤回：

```

1 #include <iostream>
2 using namespace std;
3 void Hanoi(int n, char a, char b, char c);
4 void Move(int n, char a, char b);
5 int main()
6 {
7     int n;
8     printf("Input the number of disks:");
9     scanf("%d", &n);
10    cout<<"Steps of moving"<<n<<" disks from A to B by means of C:"<<endl;
11    Hanoi(n, 'A', 'B', 'C');
12    return 0;
13 }
14 void Hanoi(int n, char a, char b, char c)
15 {
16     if (n==1)
17     {
18         Move(n, a, b);
19     }
20     else
21     {
22         Hanoi(n-1, a, c, b);
23         Move(n, a, b);
24         Hanoi(n-1, c, b, a);
25     }
26 }
27 void Move(int n, char a, char b)
28 {
29     cout<<"Move"<<n<<" from "<<a<<" to "<<b<<endl;
30 }

```

(8)R7: 查询提交记录

指令：git log

执行指令得到的结果：

```

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git log
commit abdf5c003dbd97d25535bbf5ec0492f5e05a738f (HEAD -> master)
Author: zhanghaohan <2123228124@qq.com>
Date:   Sun Nov 10 15:02:58 2024 +0800

    secong commit

commit 787d5295fed00ccf80f7ab60bb515819fbbd50fb
Author: zhanghaohan <2123228124@qq.com>
Date:   Sun Nov 10 13:45:40 2024 +0800

    first commit

```

可以观察到这种查询方式并不会显示撤销前的提交记录，只会显示结果到当前 HEAD 指针指向的版本，及该版本以前的所有提交记录。

(9)R8: 在本地仓库建立同实验远程仓库之间的关联

指令：git remote add origin https://github.com/2022111897/Lab1-2022111897.git
执行指令得到的结果：

```

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git remote add origin https://github.com/2022111897/Lab1-2022111897.git

```

(10)R9: 将之前各步骤得到的本地仓库全部内容推送到 GitHub 远程仓库

指令: git push -u origin master

执行指令得到的结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git push -u origin master
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 16 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 17.37 KiB | 2.17 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:     https://github.com/2022111897/Lab1-2022111897/pull/new/master
remote:
To https://github.com/2022111897/Lab1-2022111897.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
```

3.2 实验场景(2): 分支管理

准备工作: 在你的Gitee/GitLab/GitHub 上, 通过Web 界面建立一个Project, 将不少于 10 个文件(程序代码、文档等)加入进去, 形成初始分支 master; 在 master 基础上建立两个并行的分支B1、B2, 手工对 B2 和B3 上的某些文件进行不同程度的修改并提交。

完成上述的准备:

(1)R1: 获得本地仓库的全部分支, 切换至分支 master;

指令: git branch -a

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git branch -a
  B1
  B2
* master
  remotes/origin/master
```

可以发现当前的分支是本地的 master 分支, 但是远程的 master、B2、B3 都可以在本地被查询到

(2)R2: 在 master 基础上建立两个分支 B1、B2;

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git branch B1
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git branch B2
```

(3) R3: 在 B2 分支基础上创建一个新分支 C4;

这个操作首先要从当前的 master 分支切换到 B2 分支上, 再在这个分支的基

础上创建新的分支。

切换到 **B2** 分支: git checkout -b B2

创建新的分支 **C4** 并切换: git checkout -b C4

执行结果如下:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git checkout B2
Switched to branch 'B2'

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B2)
$ |
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git checkout B2
Switched to branch 'B2'

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B2)
$ git checkout -b C4
Switched to a new branch 'C4'

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (C4)
$ |
```

(4) R4: 在 **C4** 上, 对某个文件进行修改并提交;

指令: git add .

git commit -m “C4”

```
#include <iostream>
using namespace std;
void Hanoi(int n, char a, char b, char c);
void Move(int n, char a, char b);
//modified for lab C4
int main()
{
    int n;
    printf("Input the number of disks:");
    scanf("%d", &n);
    cout<<"Steps of moving" <<n<<" disks from A to B by means of C:"<<endl;
    Hanoi(n, 'A', 'B', 'C');
    return 0;
}
void Hanoi(int n, char a, char b, char c)
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (C4)
$ git add .
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (C4)
$ git commit -m "C4"
[c4 31399d7] C4
 1 file changed, 1 insertion(+)
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (C4)
$ |
```

(5) R5: 在 **B1** 分支上对同样的某个文件做不同修改并提交;

指令: git checkout **B1**

git add .

git commit -m “B1”

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (C4)
$ git checkout B1
Switched to branch 'B1'

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$
```

```
Hanoi.cpp # X
杂项文件 (全局范围)

1 #include <iostream>
2 using namespace std;
3 void Hanoi(int n, char a, char b, char c);
4 void Move(int n, char a, char b);
5 int main()
6 {
7     int n;
8     printf("Input the number of disks:");
9     scanf("%d", &n);
10    cout<<"Steps of moving"<<n<<" disks from A to B by means of C:"<<endl;
11    Hanoi(n, 'A', 'B', 'C');
12    return 0;
13 }
14 void Hanoi(int n, char a, char b, char c)
15 {
16     if (n==1)
17     {
18         Move(n, a, b);
19     }
20     else
21     {
22         Hanoi(n-1, a, c, b);
23         Move(n, a, b);
24         Hanoi(n-1, c, b, a);
25     }
26 }
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git commit -m "B1"
[B1 bbad436] B1
 1 file changed, 1 insertion(+)

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$
```

(6) R6: 将 C4 合并到 B1 分支，若有冲突，手工消解；

合并分支指令：git merge 分支名

需要注意的是上述指令是将分支名指定的分支合并到当前分支上，这里本人所处的分支为上一步的 B1，因此需要使用的指令是 git merge C4，执行后得到如下结果：

```

ASUS@DESKTOP-1ZZF70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3)
$ git merge C4
Auto-merging Hanoi.cpp
CONFLICT (add/add): Merge conflict in Hanoi.cpp
Auto-merging main.c
CONFLICT (add/add): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.

ASUS@DESKTOP-1ZZF70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3|MERGING)

```

Git 提示一共有 2 个文件产生冲突，分别是对 B1、C4 分支改动的 Hanoi.cpp 文件和 main.c 文件，这个是符合预期的，在产生冲突时，Git 的命令行后面会有类似这里"(B1|MERGING)"的提示，下一步需要手动进行合并。

手动合并并没有对应的指令，需要开发者手动打开发生冲突的文件修改，这里以第一个文件"Hanoi.cpp"为例，打开这个文件后发现 Git 已经做了基本的编辑：

```

void Move(int n, char a, char b)
{
    cout << "Move" << n << ": from " << a << " to " << b << endl;
}

//modified for lab branch practice step R11
//modified for lab branch practice step
<<<<< HEAD
//modified for lab branch practice step
=====
>>>>> C4

```

在手动处理冲突后得到如下的内容：

```

void Move(int n, char a, char b)
{
    cout << "Move" << n << ": from " << a << " to " << b << endl;
}

//modified for lab branch practice step R11
//modified for lab branch practice step
//modified for lab branch practice step

```

对于剩下的 1 个文件做同样的操作，就完成了手动解决冲突的问题。

处理完冲突后提交的方式和一般提交文件到本地仓库的操作一致，使用的语句为：git add 文件名 和 git commit -m "提交备注信息"。需要注意的是一般在手动处理冲突时需要开发人员一次性处理完所有的冲突文件，也就是在 git commit 的时候一般不会指定特定的文件名。本人处理完冲突后的结果如下图所示：

```
ASUS@DESKTOP-I722F70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3|MERGING)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3|MERGING)
$ git status
On branch B3
Your branch is ahead of 'origin/B3' by 1 commit.
  (use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified:   Hanoi.cpp

ASUS@DESKTOP-I722F70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3|MERGING)
$ git commit -m "B3 and C4 merging conflict solved"
[B3 ff28fe4] B3 and C4 merging conflict solved

ASUS@DESKTOP-I722F70 MINGW64 /d/Lab-softwareprocess/branch-practice/branch-practice (B3)
```

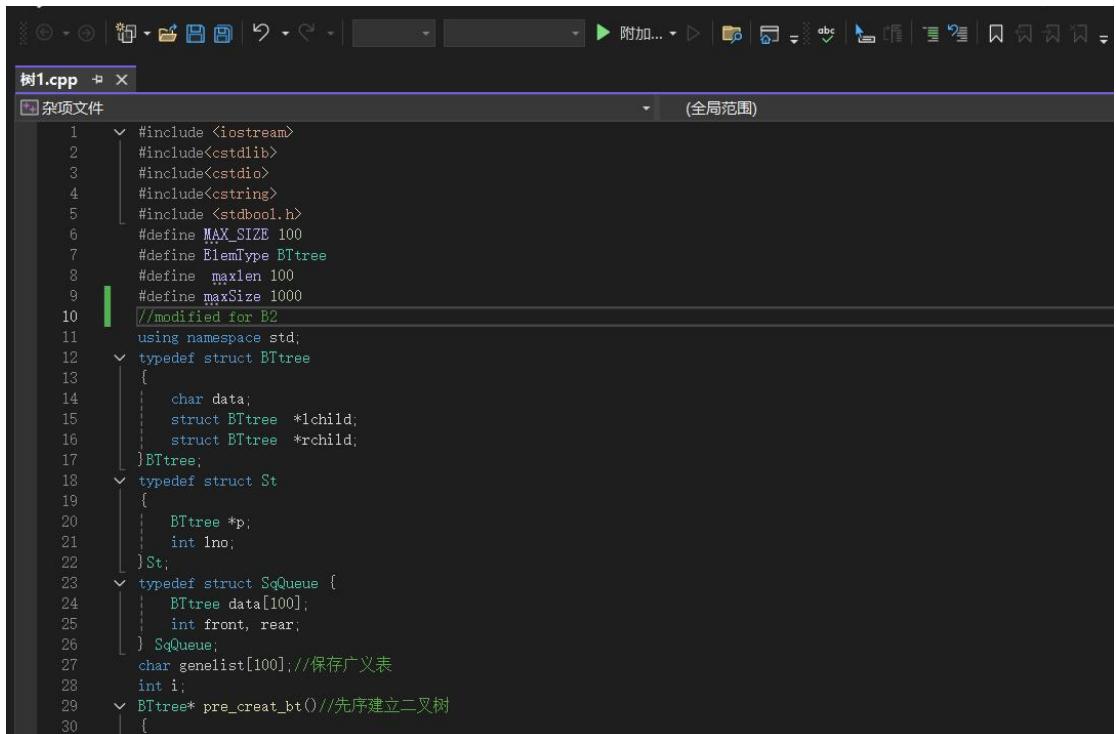
可以发现解决冲突后命令行的提示尾部从“(B3|MERGING)变成了“ (B3)”说明分支合并、冲突处理完毕。

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git merge C4
Already up to date.
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$
```

(7) R7: 在 B2 分支上对某个文件做修改并提交;

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git checkout B2
Switched to branch 'B2'
```



```

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B2)
$ git add .

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B2)
$ git commit -m "B2"
[B2 598aebc] B2
 1 file changed, 1 insertion(+)

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B2)
$ |

```

(8) R8: 查看目前哪些分支已经合并、哪些分支尚未合并；

查看已经合并的分支：git branch --merged

查看尚未合并的分支：git branch --no-merged

分别执行上述两条指令，得到如下的结果：

```

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git branch --merged
* B1
  C4
  master

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (B1)
$ git branch --no-merged
  B2

```

(9) R9: 将已经合并的分支删除，将尚未合并的分支合并到一个新分支上，
分支名字为你的学号；

上述过程一共有 4 个分支分别是 master、B1、B2、C4 分支,这一步删除了
B1 和 C4 合并后的分支，因此需要合并的分支是 B1 和 master 分支

删除分支的指令：git branch -d 分支名

强制删除分支的指令: git branch -D 分支名

执行后得到如下结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git branch -D B1
Deleted branch B1 (was 4be77b5).
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git merge B2
Updating abdf5c0..598aebc
Fast-forward
  "\346\240\2211.cpp" | 1 +
  1 file changed, 1 insertion(+)
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ |
```

对当前的分支改名:

分支改名指令: git branch -m 旧名称 新名称

执行上述指令结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (master)
$ git branch -m master 2022111897
```

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (2022111897)
$ |
```

(10)R10: 将本地以你的学号命名的分支推送到 GitHub 上自己的仓库内;

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (2022111897)
$ git push -u origin master
```

(11) R11: 查看完整的版本变迁树;

指令: git log --graph --oneline --all

结果:

```
ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (2022111897)
$ git log --graph --oneline --all
* 598aebc (HEAD -> 2022111897, B2) B2
| * 31399d7 (C4) C4
|/
* abdf5c0 (origin/master) secong commit
* 787d529 first commit

ASUS@DESKTOP-I722F70 MINGW64 /d/开源软件-Lab1-Git (2022111897)
$ |
```

(12)R12: 在 Github 上以 web 页面的方式查看你的 Lab1 仓库的当前状态。

Lab1-2022111897 (Public)

master had recent pushes 59 minutes ago

Compare & pull request

master ▾ 2 Branches 0 Tags

Go to file Add file ▾ Code ▾

This branch is 2 commits ahead of, 1 commit behind **b1**.

Contribute ▾

2022111897 secong commit abdf5c0 · 1 hour ago 2 Commits

Hanoi.cpp	first commit	2 hours ago
Traffic.cpp	first commit	2 hours ago
main.c	secong commit	1 hour ago
中缀表达式1.cpp	first commit	2 hours ago
图.cpp	first commit	2 hours ago
查找.cpp	first commit	2 hours ago
树1.cpp	first commit	2 hours ago

About
No description

Activity
0 stars
1 watch
0 forks

Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

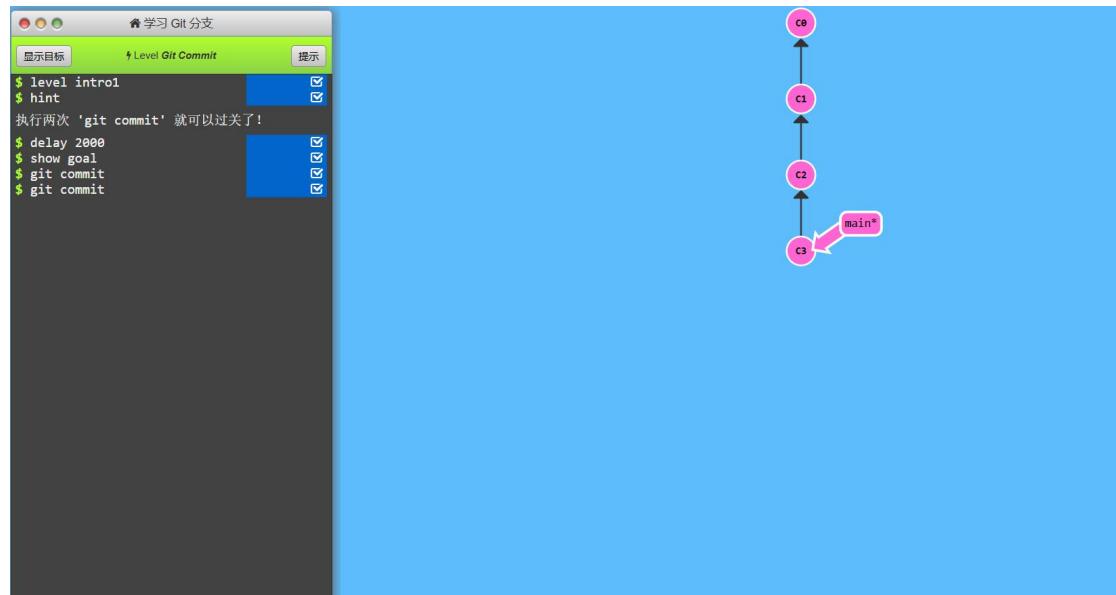
3.3 实验场景(3): 在线 Git 练习

给出完成的所有任务的命令, 格式如下:

(一) 主要页面-基础篇

任务 1:

操作命令集

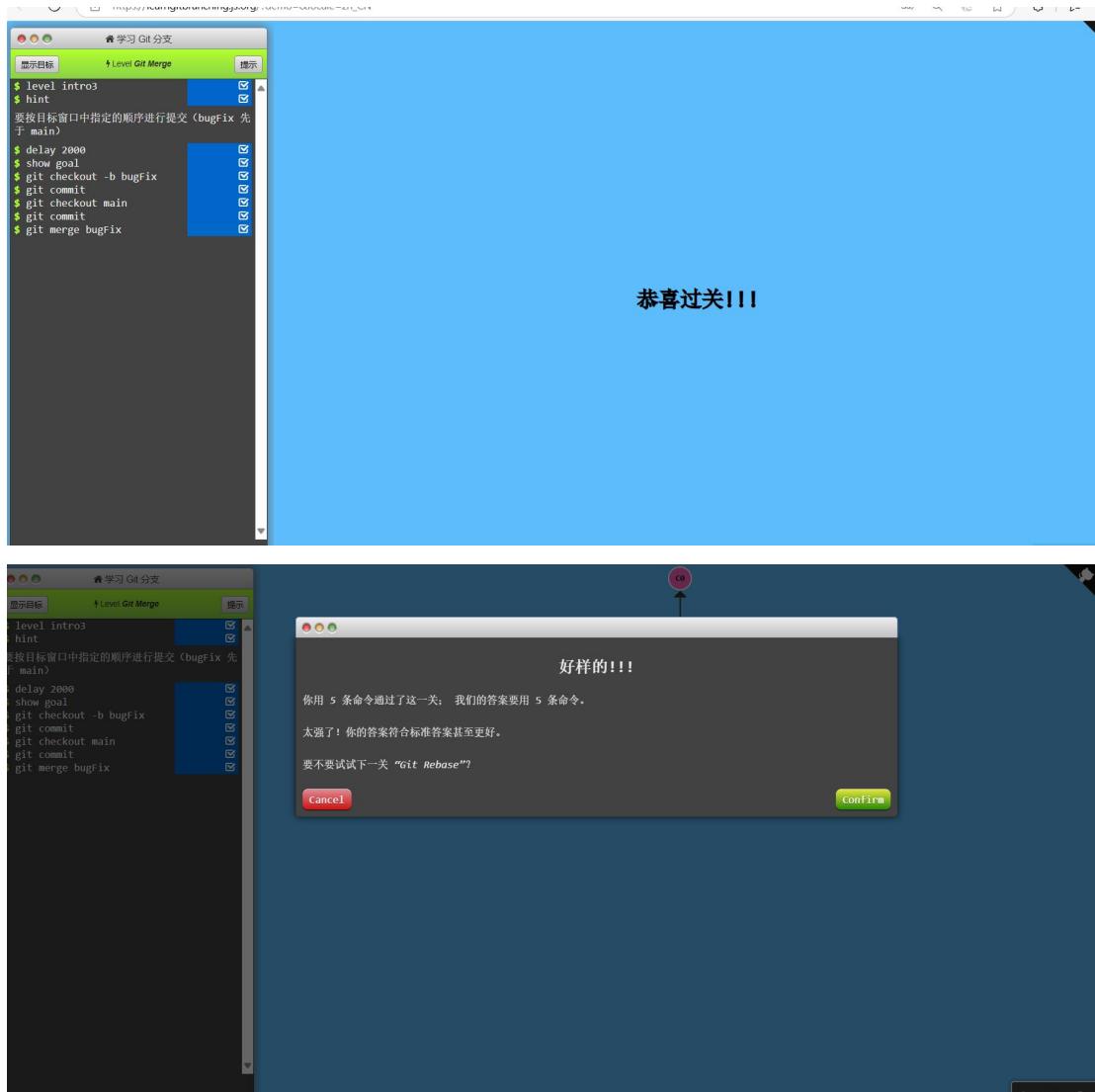




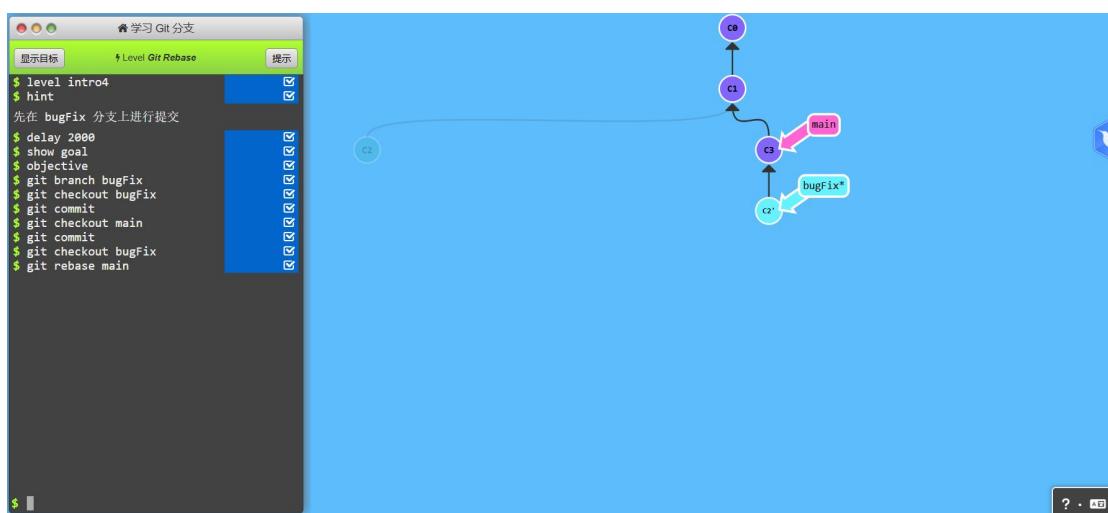
任务 2: 操作命令集



任务 3: 操作命令集



任务 4: 操作命令集

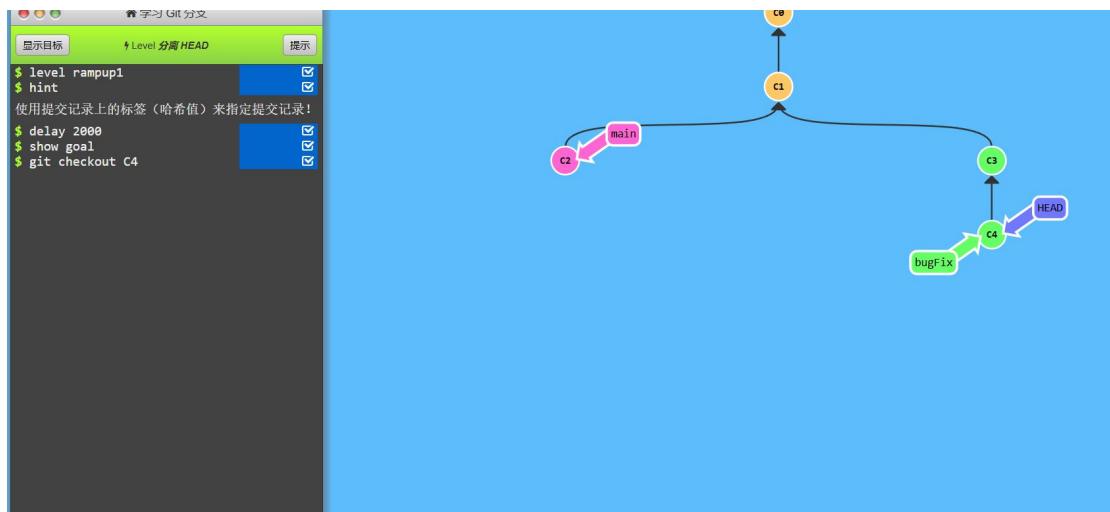




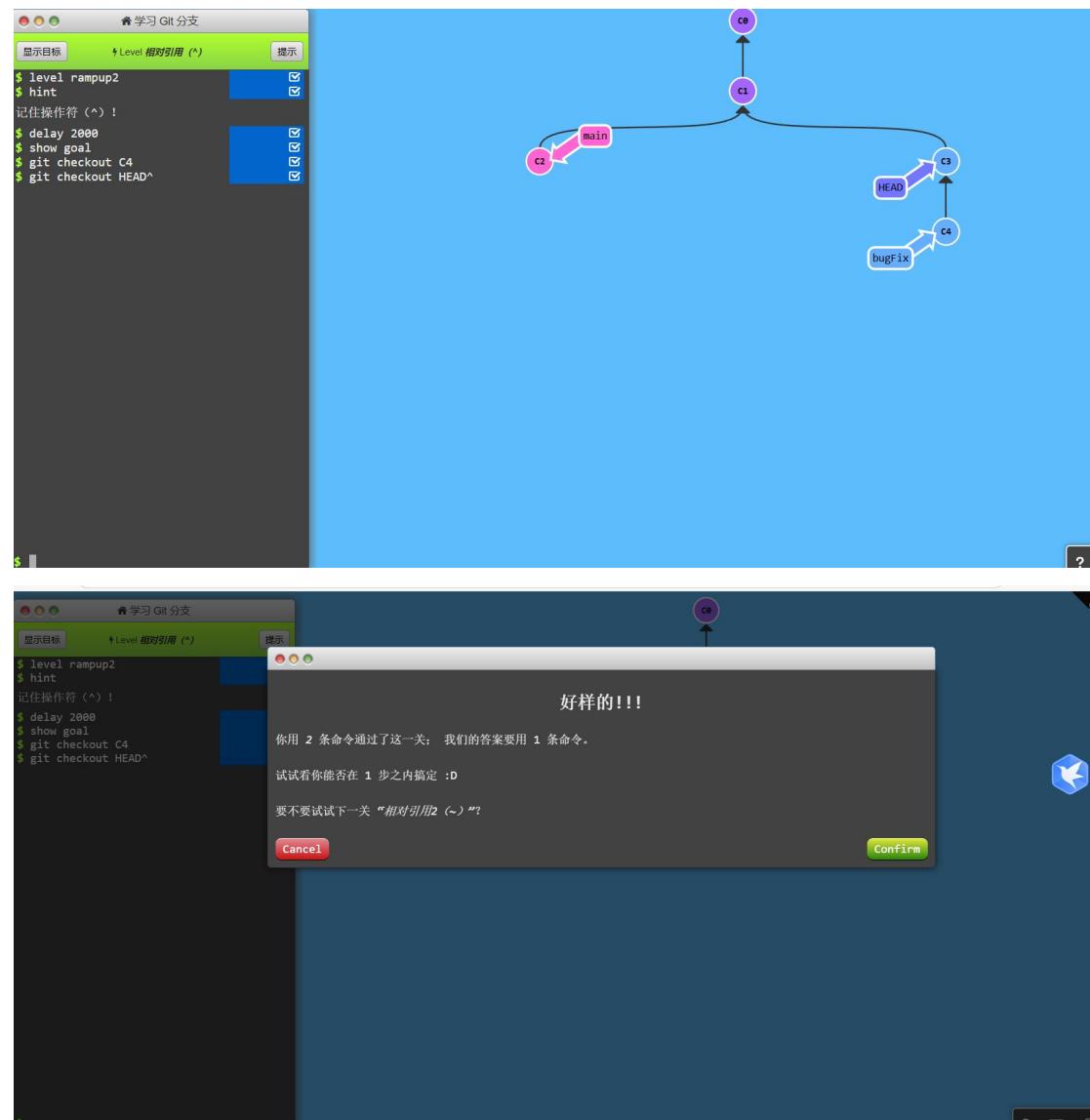
(二) 主要页面-高级篇

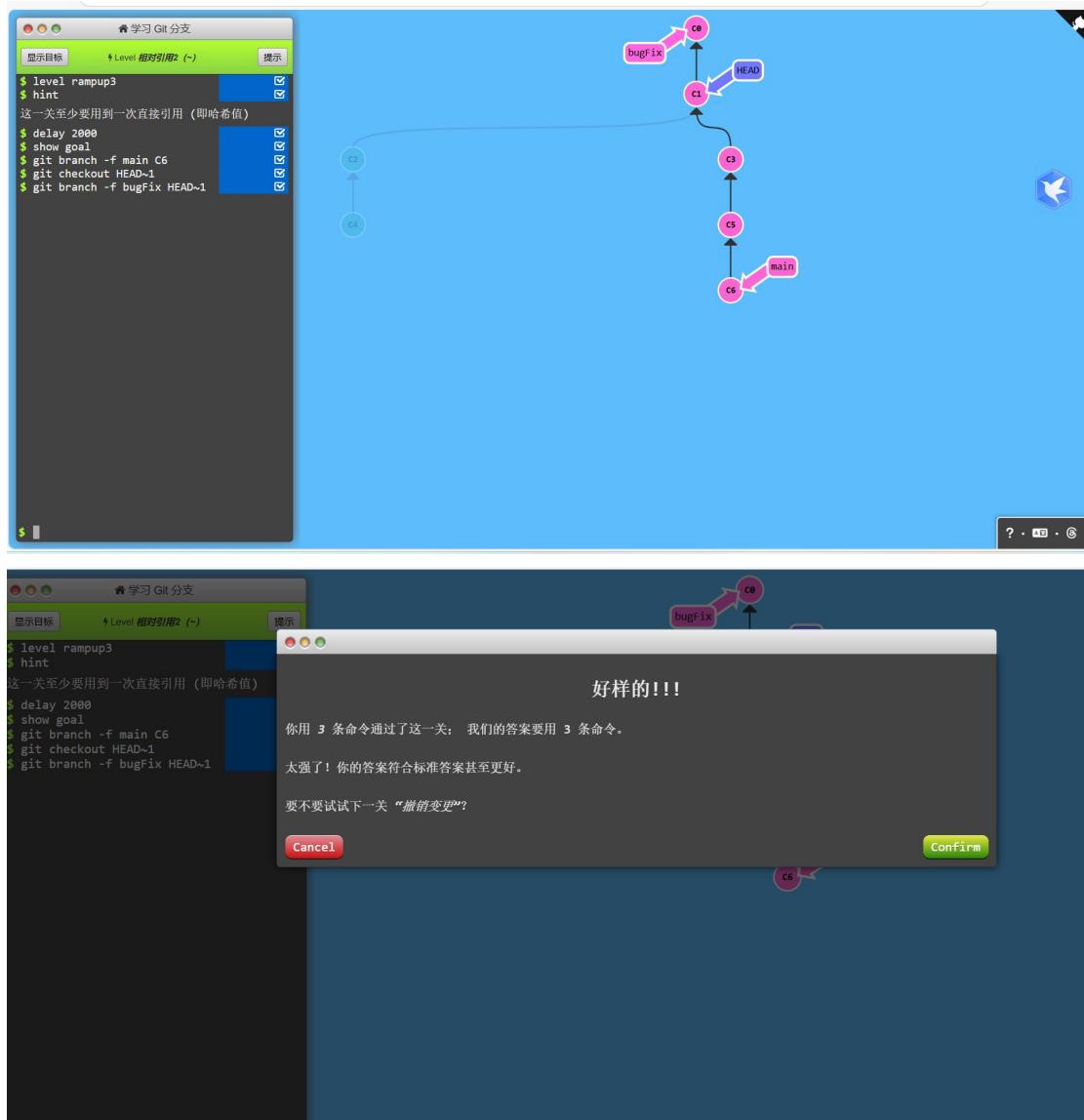
任务 1:

操作命令集:



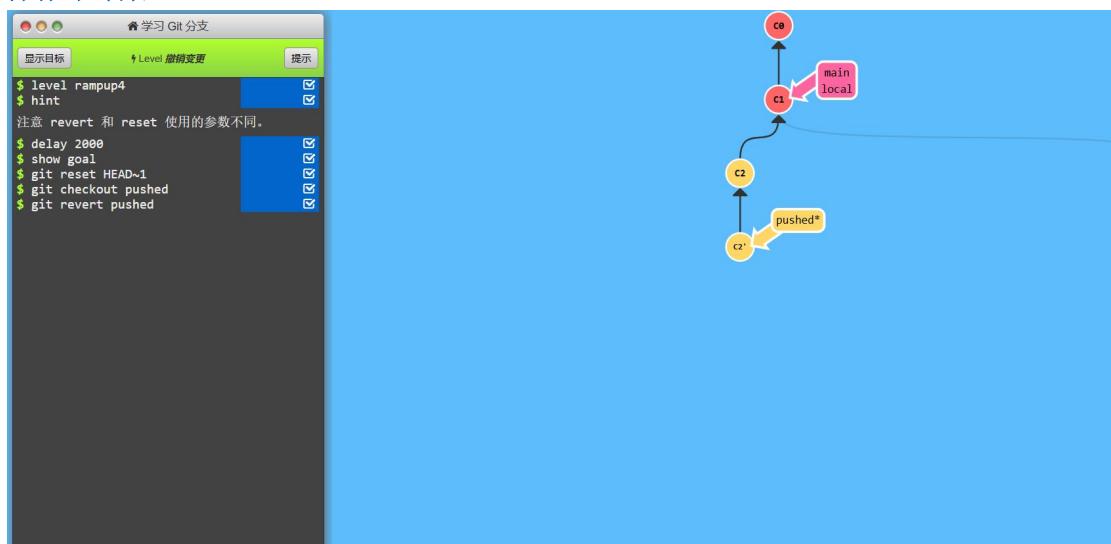
任务 2:

操作命令集:**任务 3:****操作命令集:**



任务 4:

操作命令集:

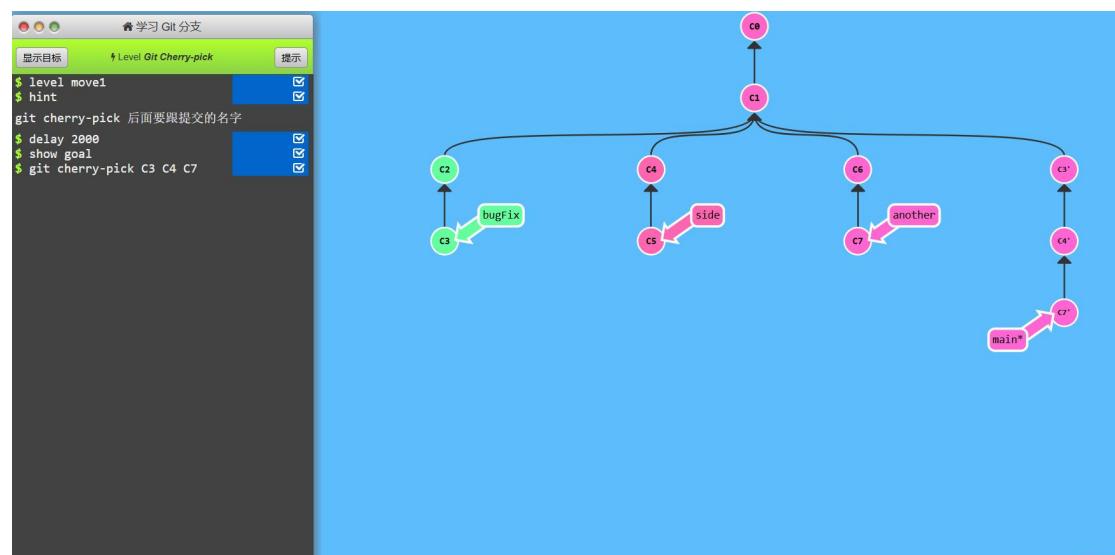


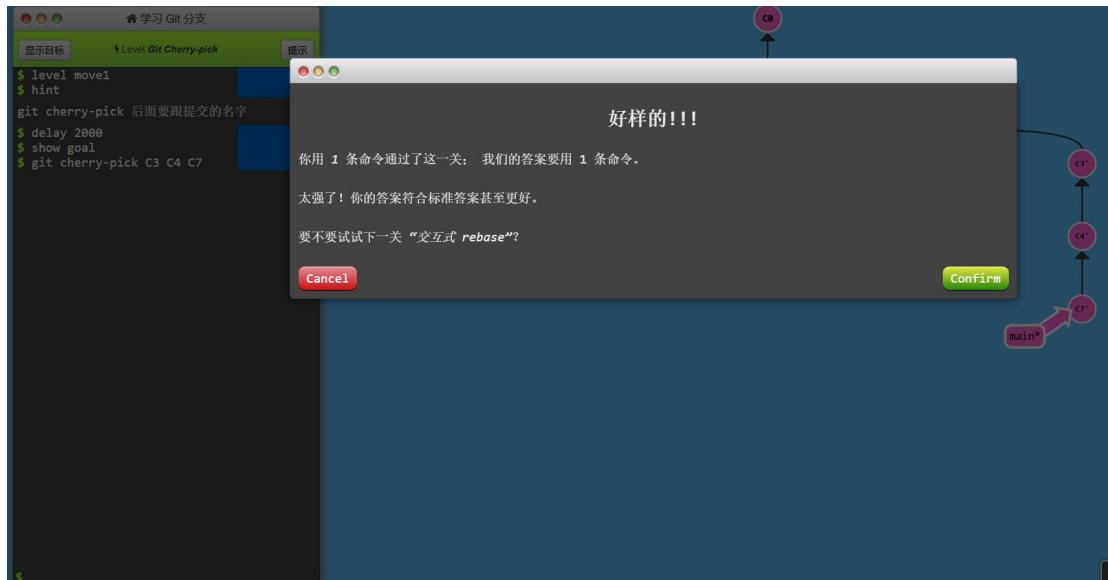


(三) 主要页面-移动提交记录

任务 1:

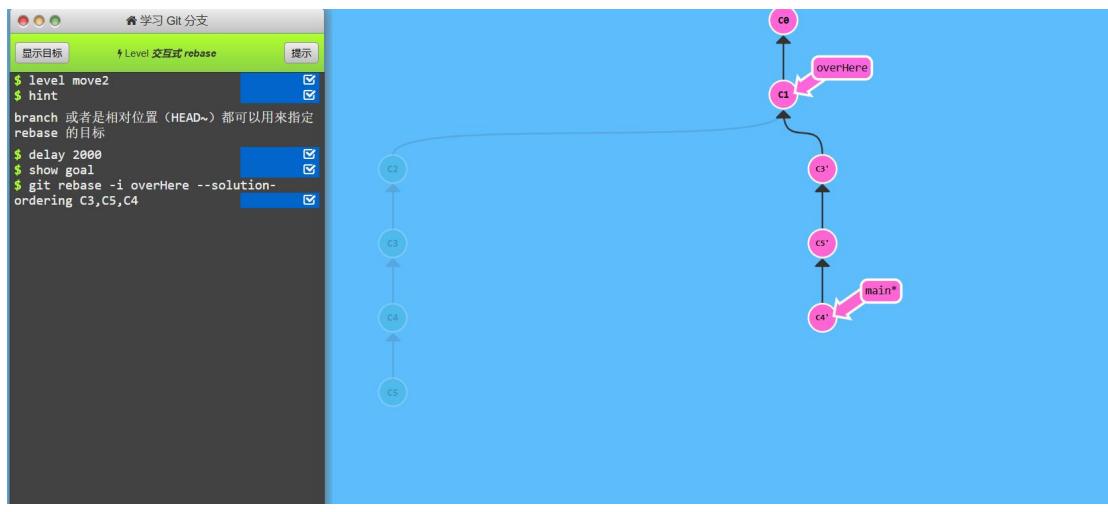
操作命令集:





任务 2:

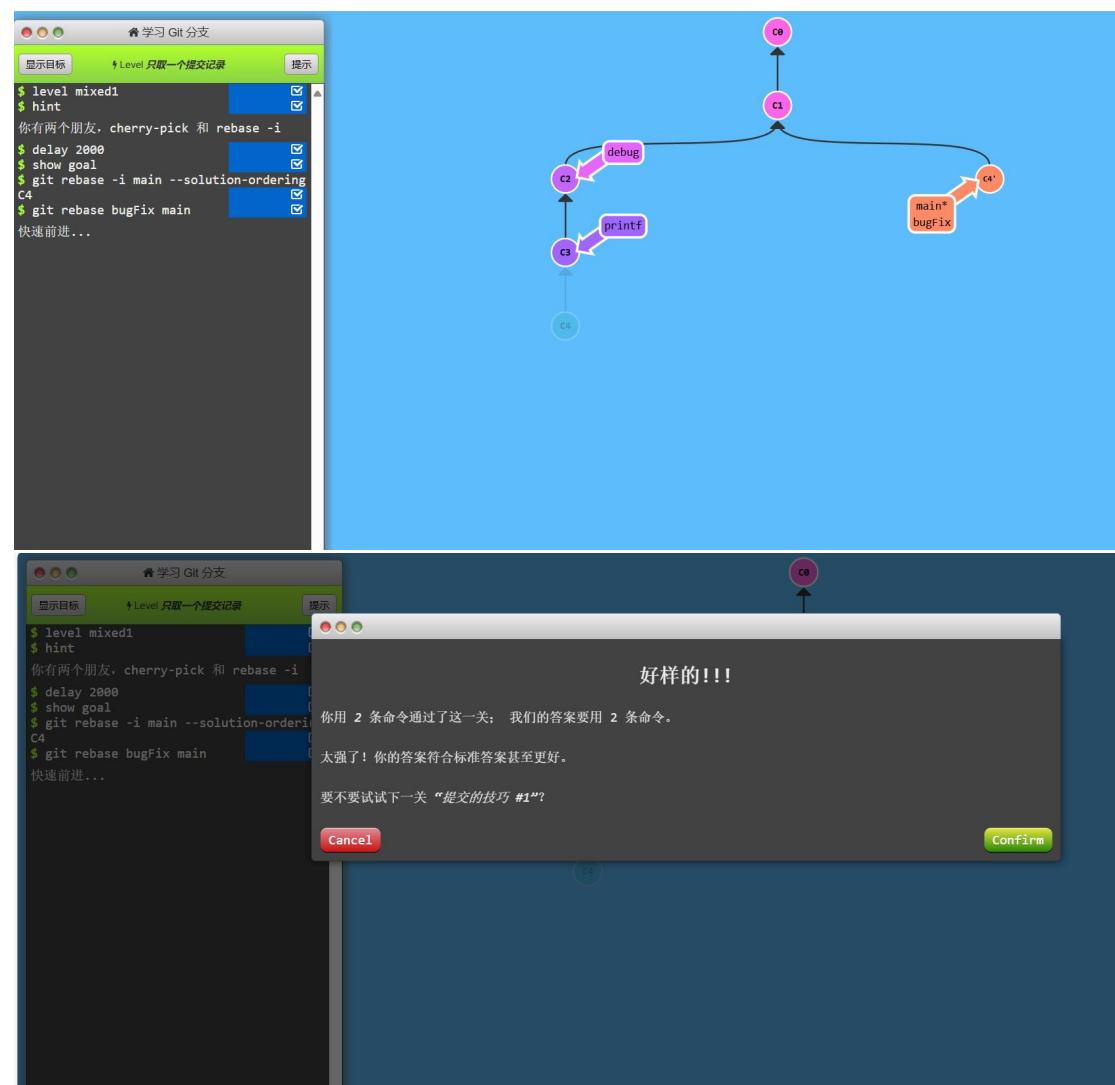
操作命令集:



(四) 主要页面-杂项

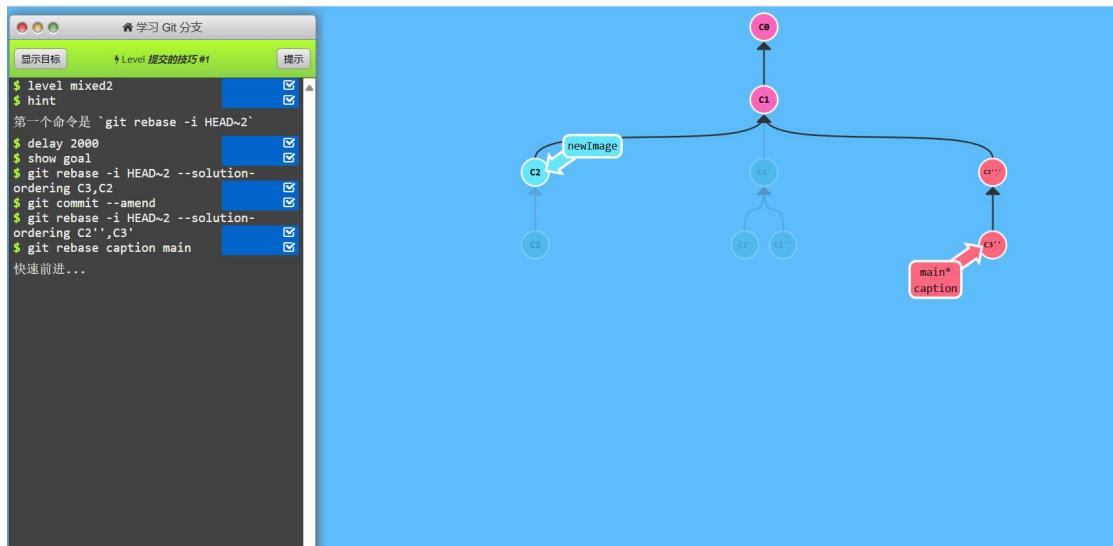
任务 1:

操作命令集:



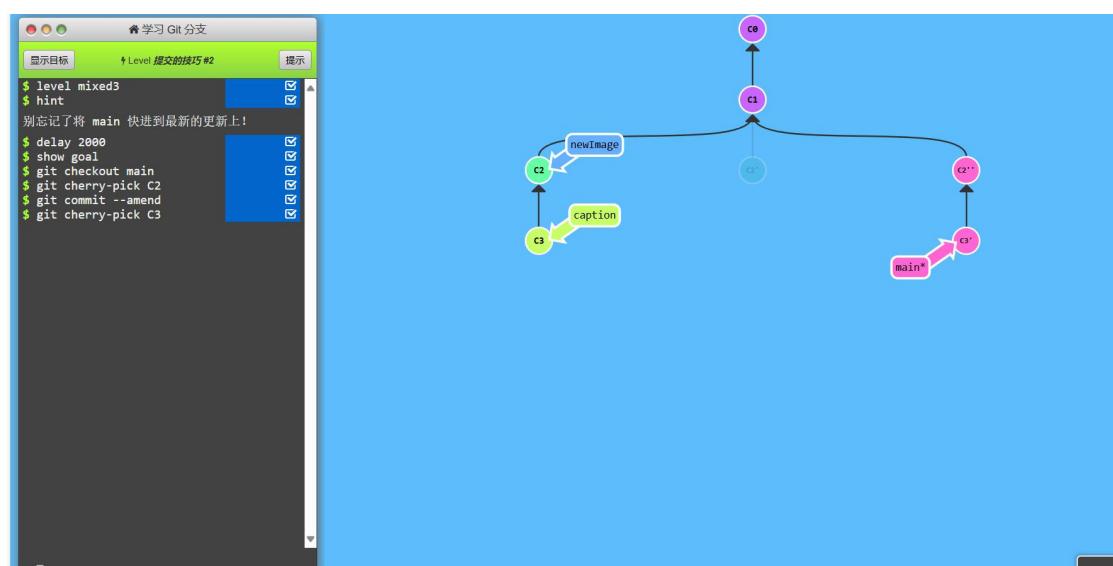
任务 2:

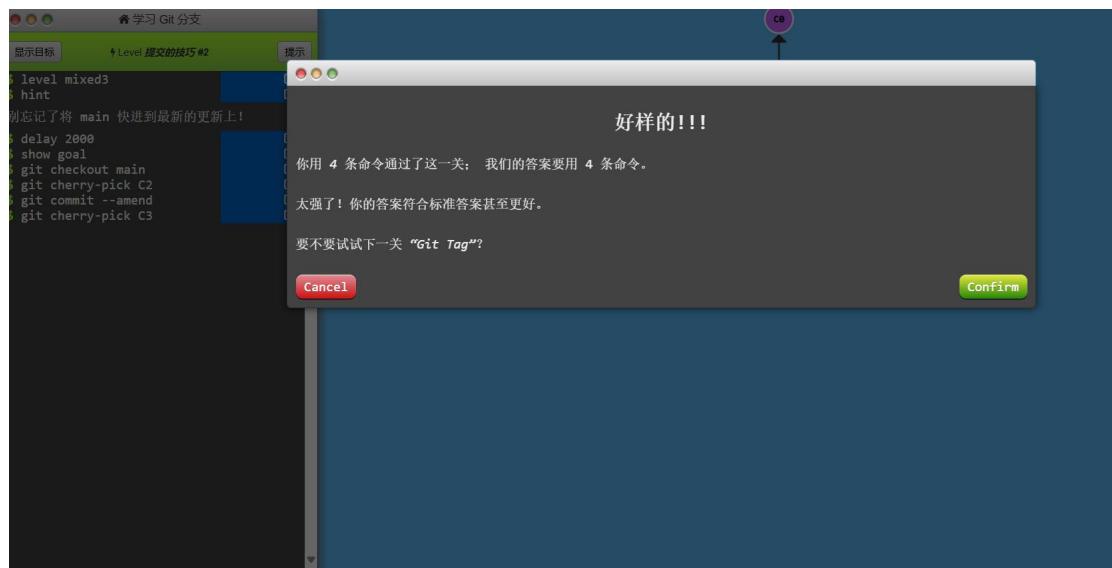
操作命令集:



任务 3:

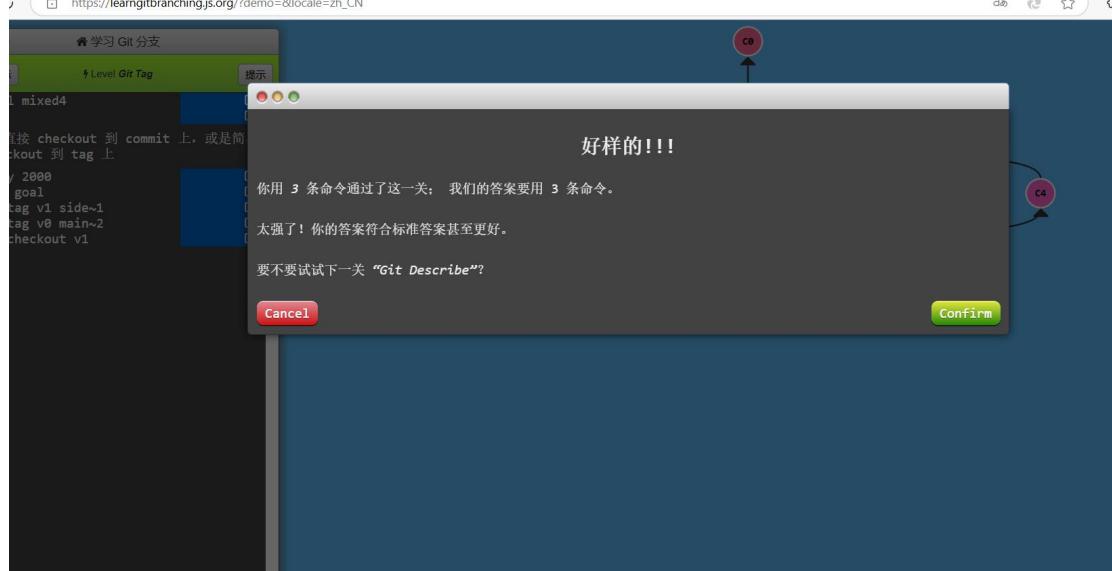
操作命令集:



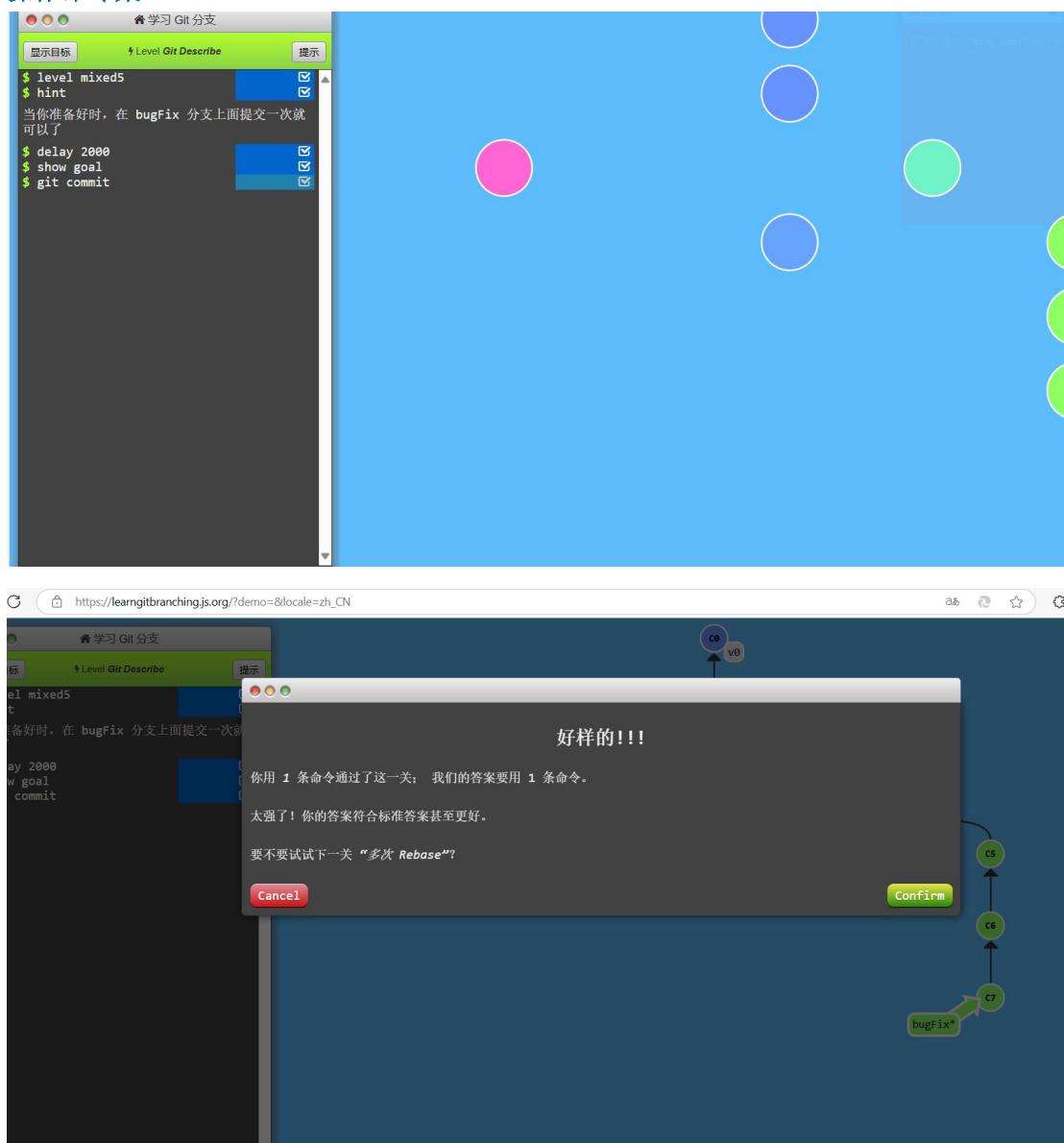


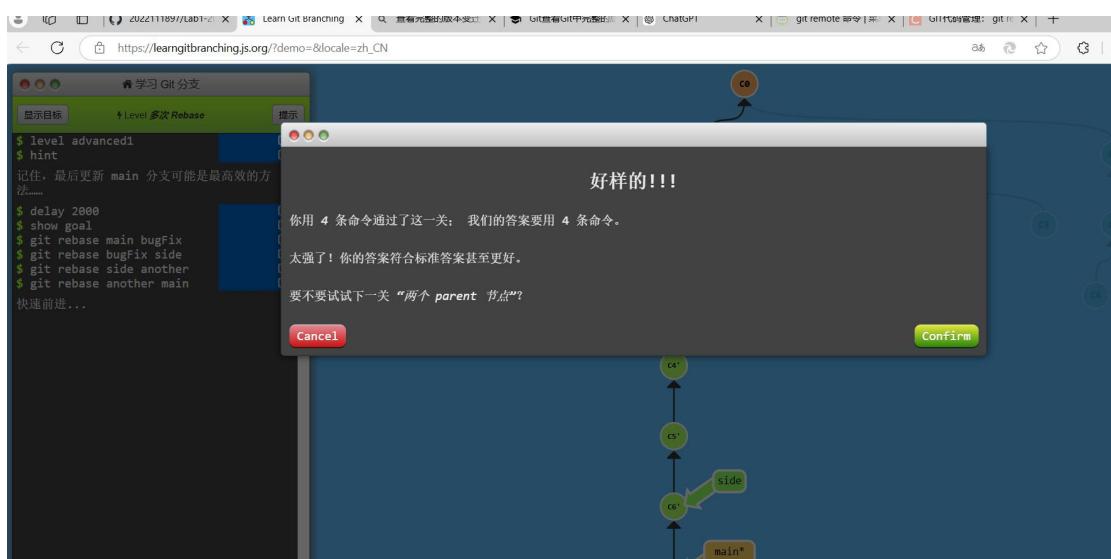
任务 4:

操作命令集:



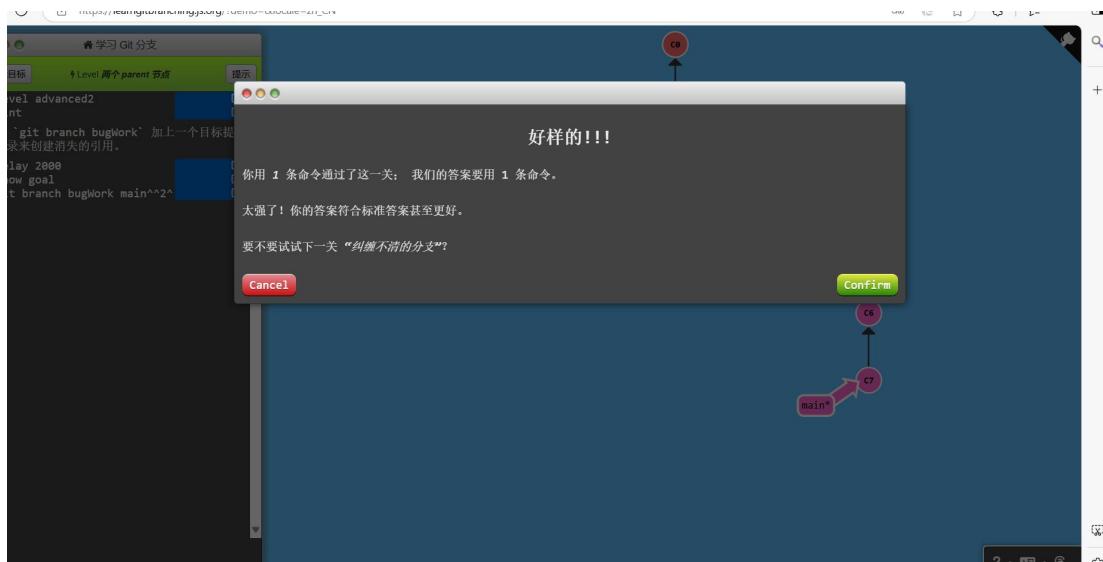
任务 5:

操作命令集:**(五) 主要页面-高级话题*****任务 1:****操作命令集:**

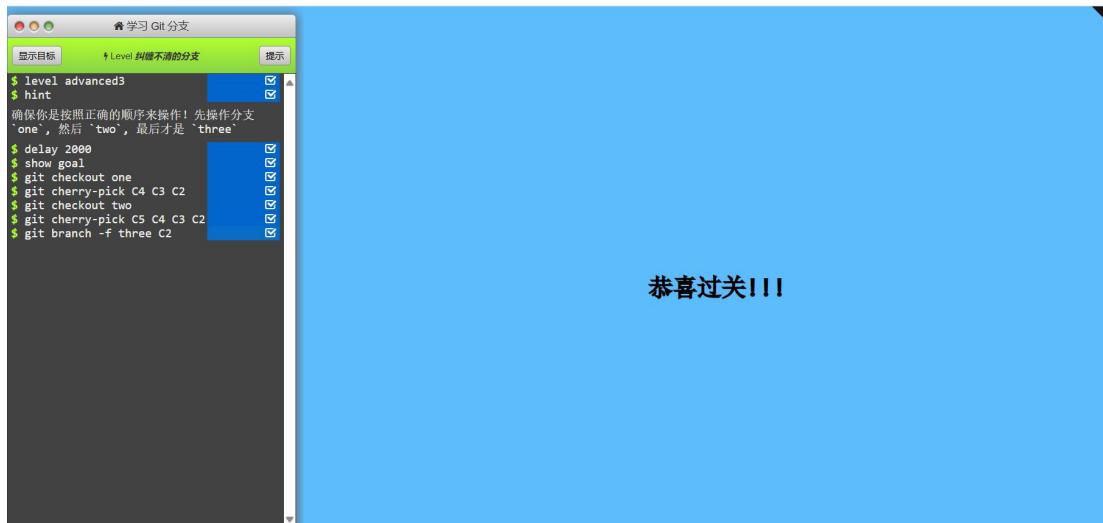


任务 2: 操作命令集：





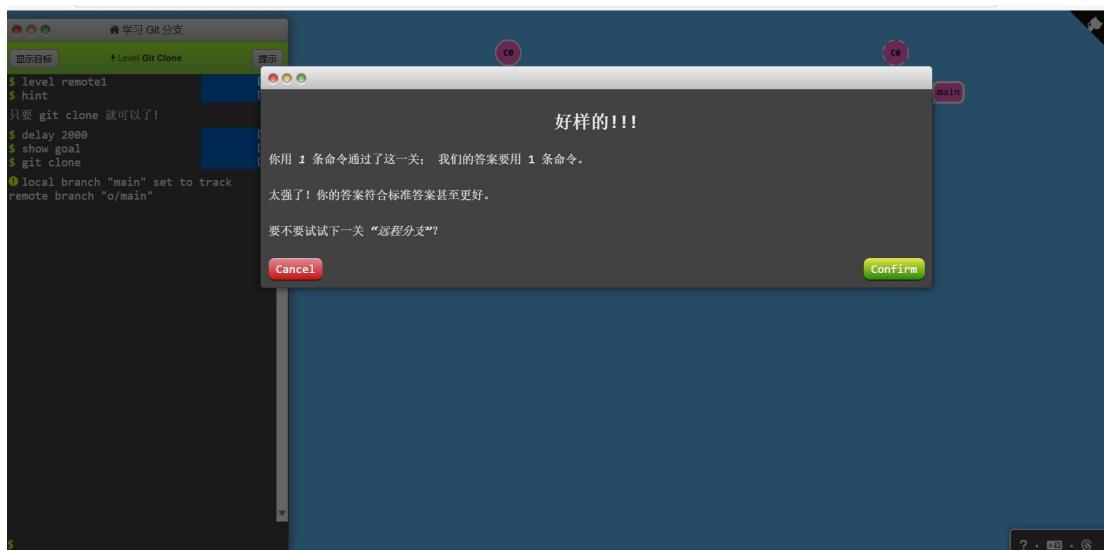
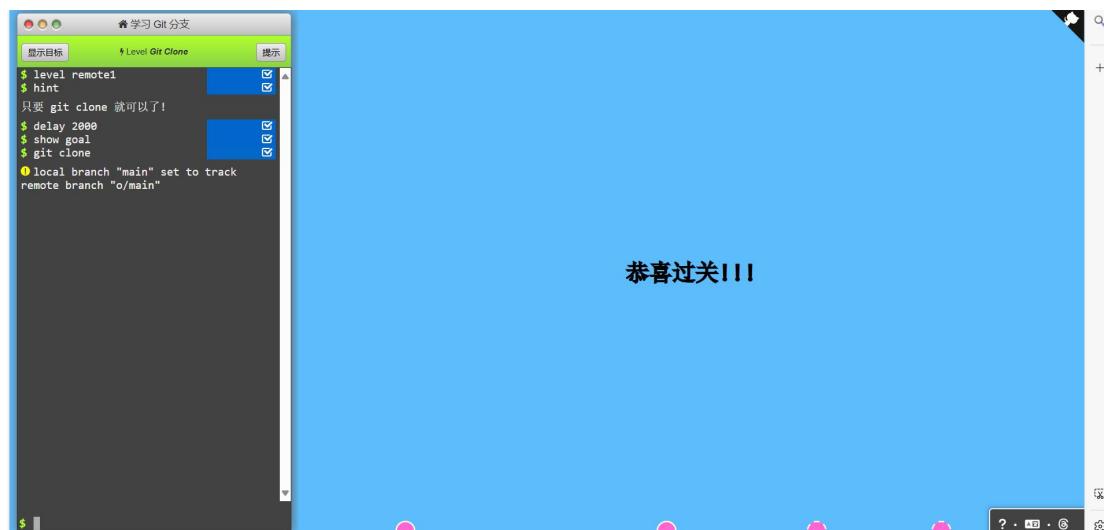
任务 3: 操作命令集:



(六) 远程页面-Git 远程仓库

任务 1:

操作命令集:



任务 2:

操作命令集:





任务 3: 操作命令集:



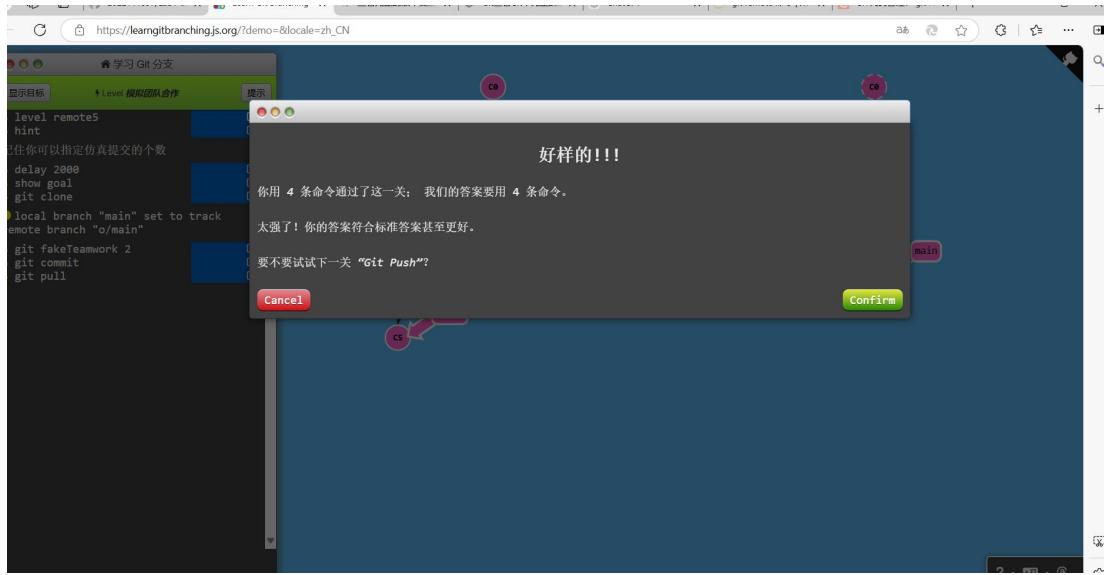
任务 4: 操作命令集:



任务 5：

操作命令集：

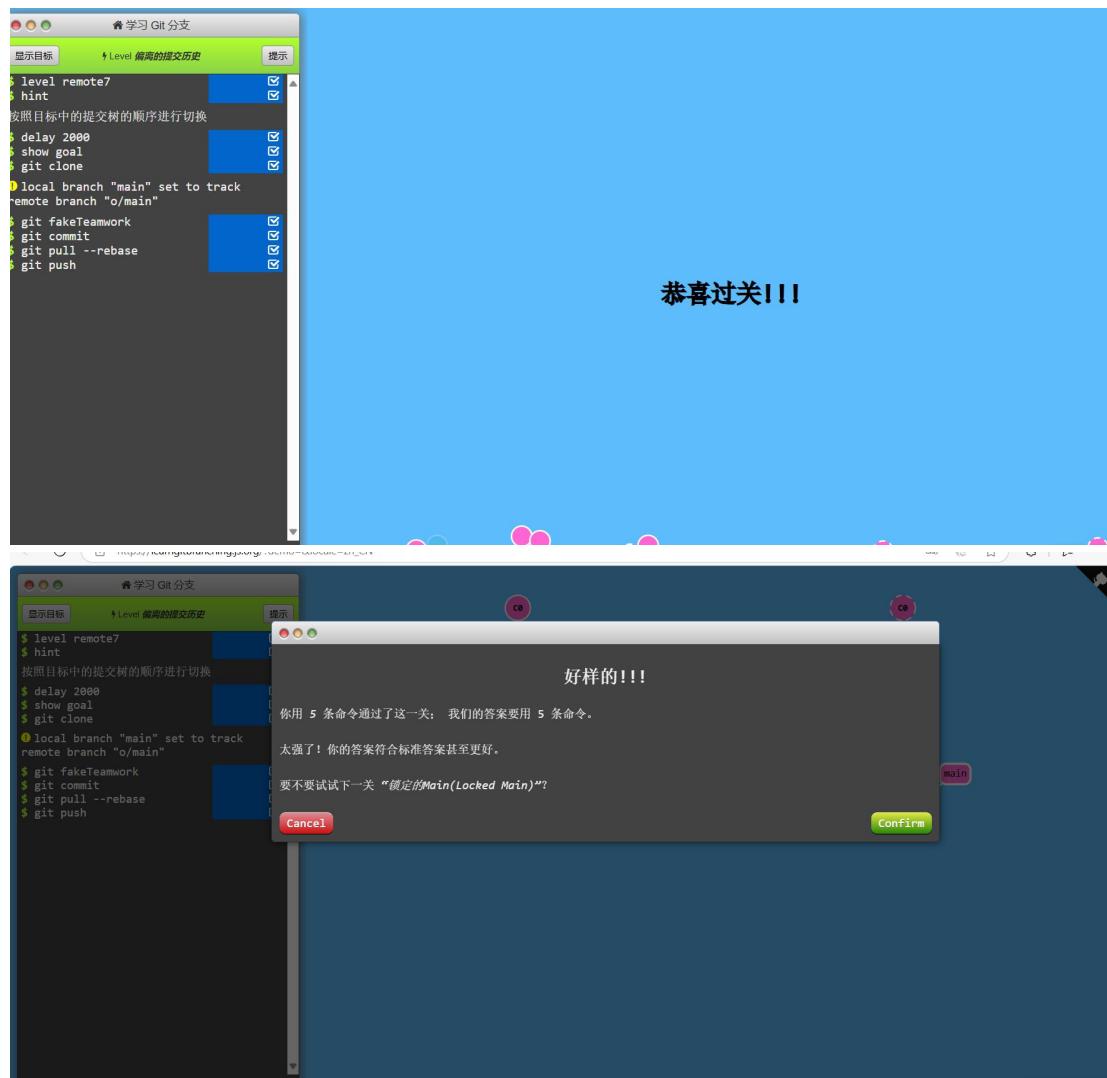


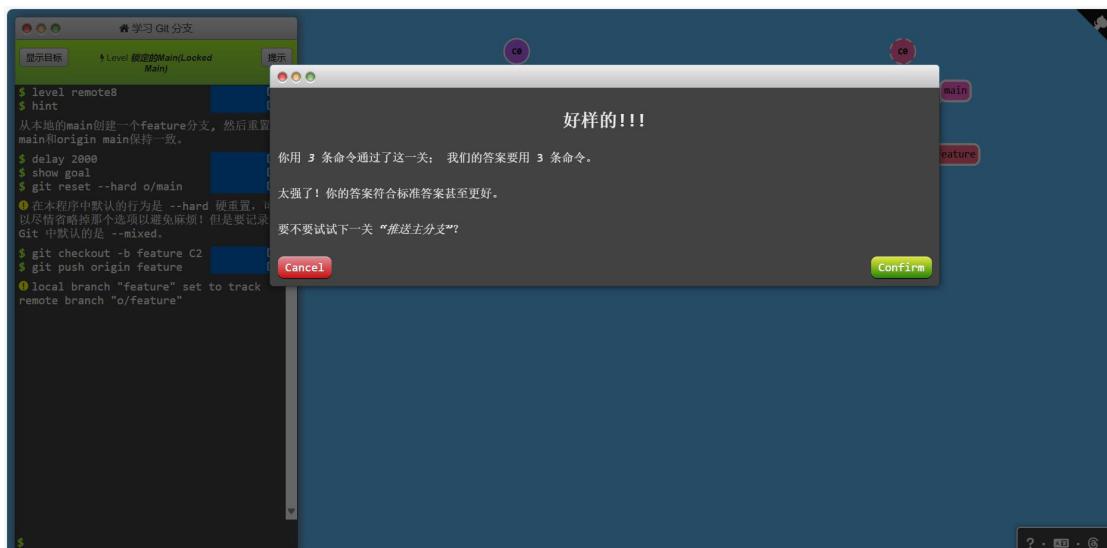


任务 6: 操作命令集:



任务 7:

操作命令集:**任务 8:****操作命令集:**



(七) 远程页面-Git 远程仓库高级操作

任务 1:

操作命令集:

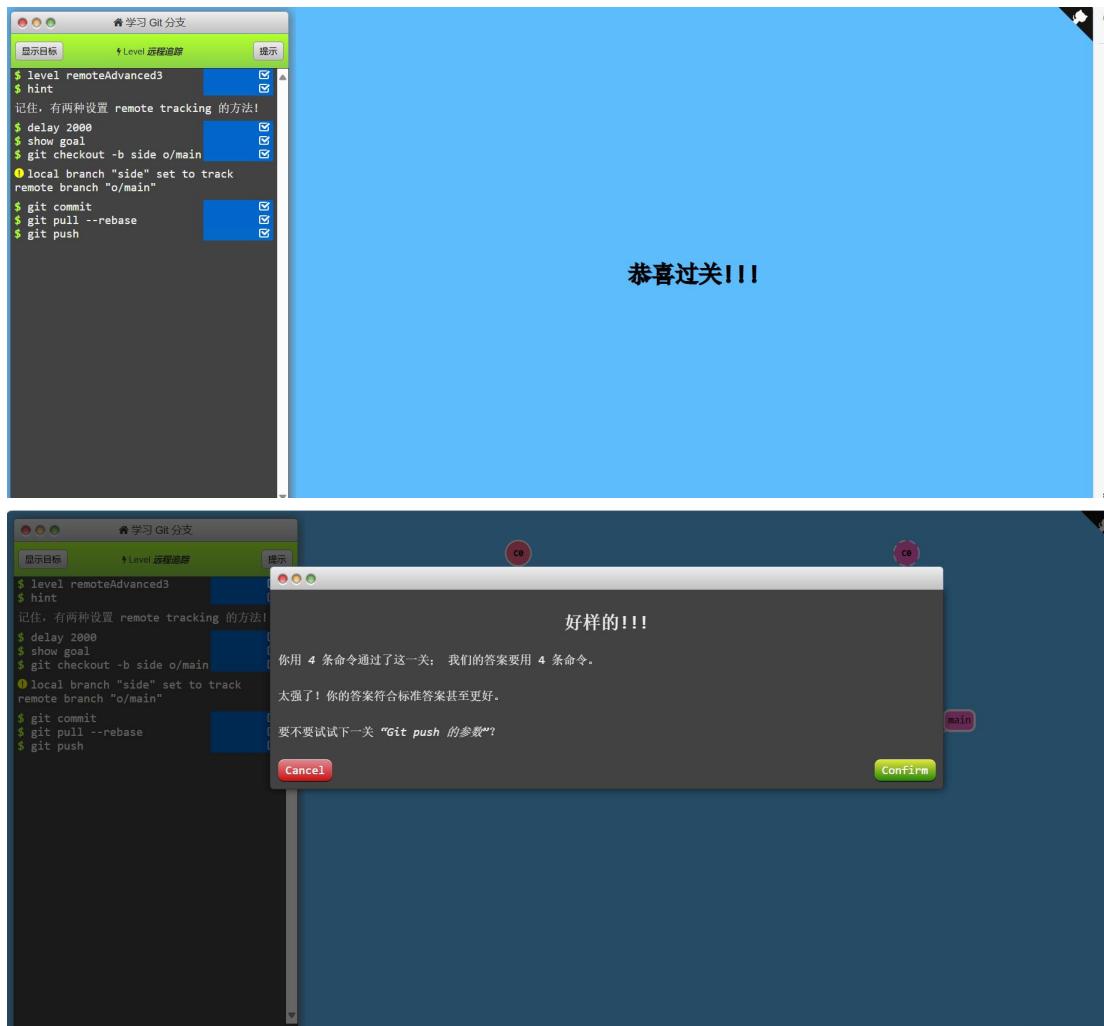




任务 2: 操作命令集:



任务 3: 操作命令集:



任务 4:

操作命令集:





任务 5： 操作命令集：



任务 6： 操作命令集：



任务 7： 操作命令集：





(八) 通关后的主界面截图

完成规定任务后，“主页”和“远程”截图各一张，如下图所示：



4 小结

对本次实验过程和结果的思考：

(1) 比较之前的开发经验，使用 Git 的优点？

使用 Git 作为版本控制系统，相较于早期的开发经验，带来了显著的优点，包括分布式控制、强大的分支和合并功能、高速性能、可靠性、灵活的工作流以及广泛的社区和工具支持。这些优点极大地提升了开发效率、团队协作和项目管理的质量，使得 Git 成为现代软件开发中不可或缺的工具。

(2) 在个人开发和团队开发中，Git 起到的作用有何主要差异？

在个人开发中，Git 主要用于版本控制、备份、分支管理和代码优化，提高开发效率和代码质量。而在团队开发中，Git 的作用更加广泛和复杂，涵盖了协作、代码共享、合并和冲突解决、分支策略、代码审查、持续集成和历史记录等方面。通过这些功能，Git 不仅提升了团队协作的效率和代码质量，还规范了开发流程和工作方式，是现代软件开发中不可或缺的工具。

(3) 之前是否用过其他的版本控制软件？如果有，同 Git 相比有哪些优缺点？如果没有查阅资料对比一下不同版本控制系统的差别。

没有；每种版本控制系统都有其优缺点，选择哪种工具取决于团队的具体需求和开发环境。Git 因其分布式特性、强大的分支管理和合并能力，在现代软件开发中得到了广泛应用和认可。相比之下，SVN 适合需要集中管理的大型企业项目，而 Mercurial 则是追求简单易用、跨平台支持好的项目的选择。CVS 则更多用于历史项目的维护和管理。理解不同版本控制系统的特，有助于在实际开发中做出更好的选择。

(4) 在什么情况下适合使用 Git、什么情况下没必要使用 Git？

Git 适合大多数需要版本控制的项目，特别是分布式团队协作、开源项目和复杂代码库的管理。然而，对于小型项目、集中式控制环境或非代码资产管理，使用 Git 可能显得有些过度。根据项目的具体需求和团队情况选择合适的版本控制工具，才能发挥最佳效果。