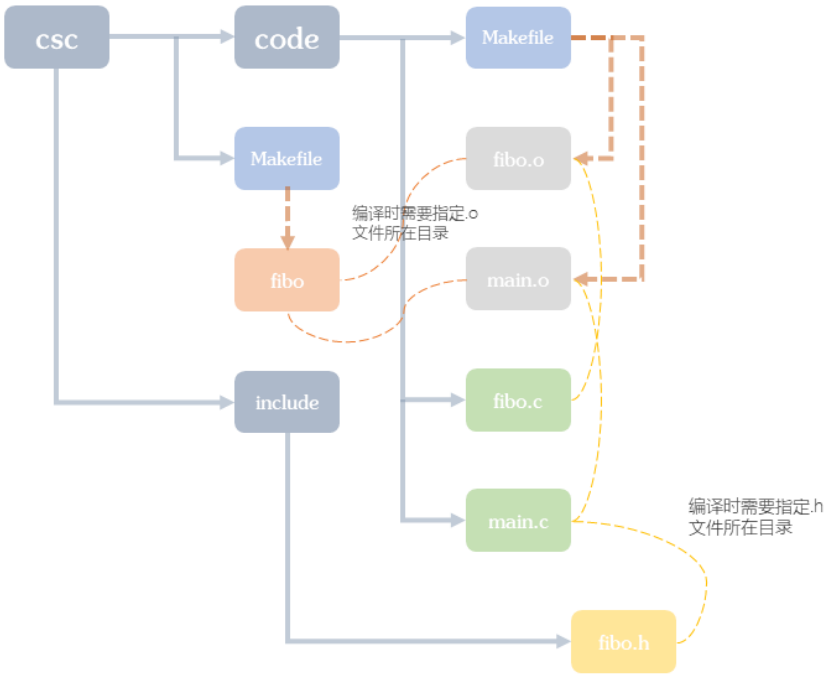
实验目的

- 认识操作系统实验环境
- 掌握操作系统实验所需的基本工具
    - 在本章中，我们需要去了解实验环境，熟悉 Linux 操作系统（Ubuntu），了解控制 终端，掌握一些常用工具并能够脱离可视化界面进行工作。

实验难点

个人认为，实验的前三个任务只需要了解指导书第一单元中基本的shell命令行操作即可完成，难点在于Exercise 0.4的第二个要求——在其他目录编译生成指定文件，并且由于c程序的特性，还要实现对位于其他目录的.h文件等的调用。



Exercise

*已于课下测试中获得100分

```
mote: [ make clean can delete main.o ]
mote: [ find your fibo after make clean!  ]
mote: [ You got 100 (of 100) this time. Thu Mar 10 00:56:01 CST 2022 ]
mote:
mote: Autotest: End at Thu Mar 10 00:56:01 CST 2022
```

Exercise 0.1

- 补全回文数判断程序

```c
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int test = n, tmp = 0;
    while (test > 0) {
        tmp = test % 10 + tmp * 10;
        test /= 10;
    }

    if(tmp == n){
        printf("Y");
    }else{
        printf("N");
    }
    return 0;
}
~
```

- 补全Makefile文件

```
make:  palindrome.c
    gcc palindrome.c -o palindrome
~
```

- 补全hello_os.sh

```
#!/bin/bash

#Create a document named as $2
touch $2
sed -n '8p' $1 > $2
sed -n '32p' $1 >> $2
sed -n '128p' $1 >> $2
sed -n '512p' $1 >> $2
sed -n '1024p' $1 >> $2
~
```

在这里使用`sed`工具来对指定行数的内容进行提取，并写到新建文件中。在这里需要注意的是，`>`将覆盖文件中的内容，而`>>`则会在文件末尾拼接内容，因此我的处理方法是先覆盖再拼接，以完成覆写已存在文件的实验要求。

- 复制文件到指定路径

使用`cp`命令即可复制文件夹到另一路径中，格式为`cp src dest`

Exercise 0.2

- 补完changefile.sh

```
#!/bin/bash
a=1
while [ $a -le 100 ]
do
    if [ $a -gt 70 ]      #if loop variable is greater than 70
    then
        rm -rf file$a

    elif [ $a -gt 40 ]        # else if loop variable is great than 40
    then
        mv file$a newfile$a
    fi
    a=$[$a+1]
            #don't forget change the loop variable
done
~
```

Exercise 0.3

- 补完search.sh

```
#!bin/bash
#First you can use grep (-n) to find the number of lines of string.
#Then you can use awk to separate the answer.

#$1 file
#$2 int
#$3 result

touch $3
grep -n "$2" $1 | awk -F: '{print $1}' > $3
~
```

Exercise 0.4

- 补完modify.sh

```
#!/bin/bash

#$1 fibo.c
#$2 char
#$3 int
sed -i "s/$2/$3/g" $1
~
```

- 修改fibo.c

```c
int fibo(int n)
{
    int a;
    int b;
    int c;
    if (n==1){
        a=1;
    }else{
        int i;
        c=0;
        b=1;
        for(i=2;i<=n;i++){
            a=b+c;
            c=b;
            b=a;
        }
    }
    return a;
}
~
```

- 补全csc/Makefile

```
O_PATH = --directory="code"

make :
    $(MAKE) $(O_PATH)
    gcc ./code/main.o ./code/fibo.o -o fibo

clean :
    $(MAKE) clean $(O_PATH)
~
```

- 补全csc/code/Makefile

```
make : main.c ../include/fibo.h fibo.c
    gcc main.c -c -I "../include/"
    gcc fibo.c -c
clean :
    rm main.o
    rm fibo.o

~
```

Thinking

Thinking 0.1

使用了`cat Modified.txt`后，发现其与第一次add之前的status不一样：

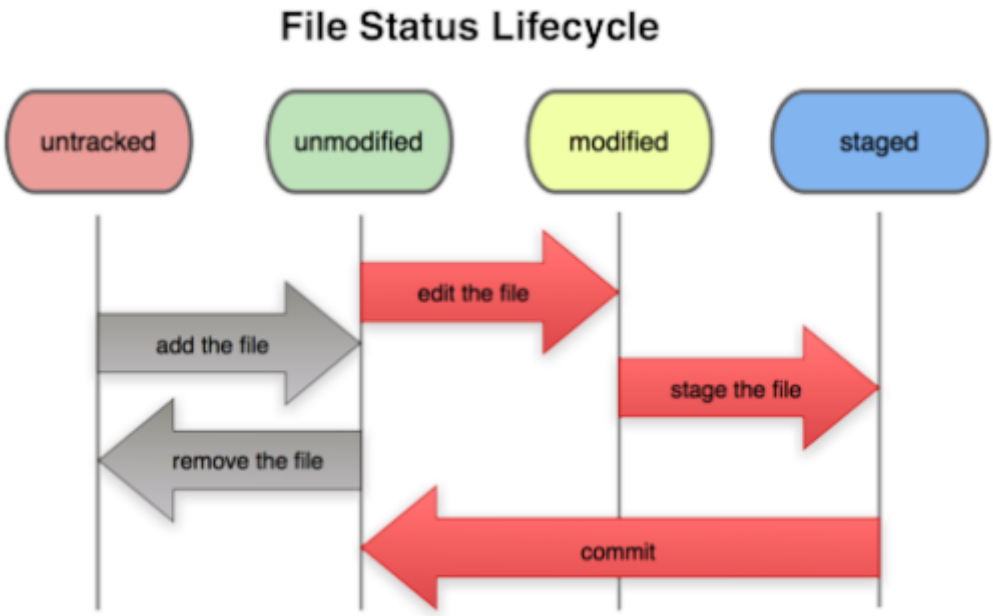
```
##----------------git status > Modified.txt----------------
#On branch master
#Changes not staged for commit:
#  (use "git add <file>..." to update what will be committed)
#  (use "git restore <file>..." to discard changes in working directory)
#        modified:   README.txt
#
#Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#        Modified.txt
#        Stage.txt
#        Untracked.txt
#
#no changes added to commit (use "git add" and/or "git commit -a")

##----------------git status > Untracked.txt----------------
#On branch master
#
#No commits yet
#
#Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#        README.txt
#        Untracked.txt
#
#nothing added to commit but untracked files present (use "git add" to track)
```

第一次查看`Untracked.txt`的status时，报“Untracked files...”，因为此时只在工作区有file，暂存区还没有；

而查看`Modified.txt`时，报“Changes not staged for commit”，说明工作区、暂存区都存在文件，此时我们正在工作区进行对此文件的修改或删除，但没有将此文件add到暂存区中。

Thinking 0.2



- add the file 对应 `git add`
- Stage the file 对应 `git add`
- commit 对应 `git commit`

Thinking 0.3

- 恢复printf.c：`git checkout printf.c`
- 恢复被小红`rm`掉的printf.c

  `git reset HEAD printf.c`
  `git checkout printf.c`
- 返回根目录，执行`touch .gitigore`，然后使用vim命令在.gitigore中写入 /mtk/Tucao.txt

Thinking 0.4

- 第三次提交的提交日志没了
- 第二次提交的提交日志也消失了，只保留了提交说明为1那一次的提交日志
- 我们每一次在git中执行提交时，除了提交作者、日期和存储的数据等所有明显的信息外，每个提交还包含上一个提交的hash值——这正是生成提交历史记录的方式。每个提交都知道它前面的提交的哈希值。这个独一无二的哈希值映射到每一次提交的行为及其内容，因此，通过对这个哈希值进行操作，就可以实现对过去内容的“恢复”等操作，也就是指导书中“时光机”的描述。

Thinking 0.5

- 1错误。一般来说，如果直接执行`git clone`而不带相应参数`-b`，只会克隆远程库的master分支；若克隆后不执行git checkout命令，HEAD指向的分支也无法被检出。
- 2正确。
- 3正确。使用`cat .git/HEAD`验证即可
- 4正确。使用`git branch`验证即可。

Thinking 0.6

Thinking 0.7

- command文件内容

```
#!/bin/bash
touch test.sh
echo echo Shell Start... > test.sh
echo echo set a = 1 >> test.sh
echo a=1 >> test.sh
echo echo set b = 2 >> test.sh
echo b=2 >> test.sh
echo echo set c = a+b >> test.sh
echo 'c=$[$a+$b]' >> test.sh
echo 'echo c = $c' >> test.sh
echo echo save c to ./file1 >> test.sh
echo 'echo $c>file1' >> test.sh
echo echo save b to ./file2 >> test.sh
echo 'echo $b>file2' >> test.sh
echo echo save a to ./file3 >> test.sh
echo 'echo $a>file3' >> test.sh
echo echo save file1 file2 file3 to file4 >> test.sh
echo 'cat file1>file4' >> test.sh
echo 'cat file2>>file4' >> test.sh
echo 'cat file3>>file4' >> test.sh
echo echo save file4 to ./result >> test.sh
echo 'cat file4>>result' >> test.sh
~
~
~
~
~
~
~
~
~
"command.sh" 22L, 705C
```

- result文件内容

从test文件的内容可以看到，该脚本程序的作用是：

1. 先进行赋值a=1和b=2，然后执行计算操作得到c=a+b=3；
2. 分别保存a、b、c的值到file3、file2、file1；
3. 将file1、file2、file3文件中的内容依次输出到file4；
4. 保存file4文件内容到result文件。

因此result文件中的内容依次为3、2、1.

- 思考
  - echo echo Shell Start 与 echo ‘echo Shell Start’ 效果没有区别
  - echo echo $c>file1 与 echo ‘echo $c>file1’ 效果有区别，因为在.sh文件里会把$c识别成参数，如果需要将其作为字符串处理就要使用单引号括起来

实验体会与反思

- **实验体会**

  1. 完成Lab0课下实验用时为Wed Mar 9 21:30 ~ Thu Mar 10 00:56
  2. 总体来说实验难度不高，初步了解了操作系统实验环境与一些基本的指令。

- **实验反思**

  1. 文件中对$_的字符串可能会有识别参数的操作，一方面，我们可以利用它来实现带参数的脚本运行（在文件中设定好$1、$2等）；另一方面，如果我们只希望将其作为字符串输出，则需要使用单引号括起；

     1. 值得注意的是，双引号可以识别变量，并实现转义；
     2. 需要把命令执行结果赋给变量时使用反引号；

  2. shell的一些基础文件操作命令；
  3. Git的基本操作与概念理解；
  4. 编写脚本程序时首行必须记得写上`#!/bin/bash`告诉系统后面的参数是执行文件的程序；
  5. 编写完脚本程序后记得`chmod +x [scriptName.sh]`修改程序的执行权限，否则会报Permission denied的错；
  6. c语言编译的一些内容：

编译过程	执行操作	生成文件
预处理	展开头文件/宏替换/去掉注释/条件编译	.i
编译	检查语法，生成汇编	.s
汇编	汇编代码转换机器码	.o
链接	链接到一起生成可执行程序	.out

     // 编写编译脚本时，`-I`可以指定所在目录，`-i`指定文件

  7. 基本的Makefile编写方法；
  8. 基本的重定向与管道操作。