1. 解释web中什么是静态资源和动态资源？

<https://zhuanlan.zhihu.com/p/351238623>

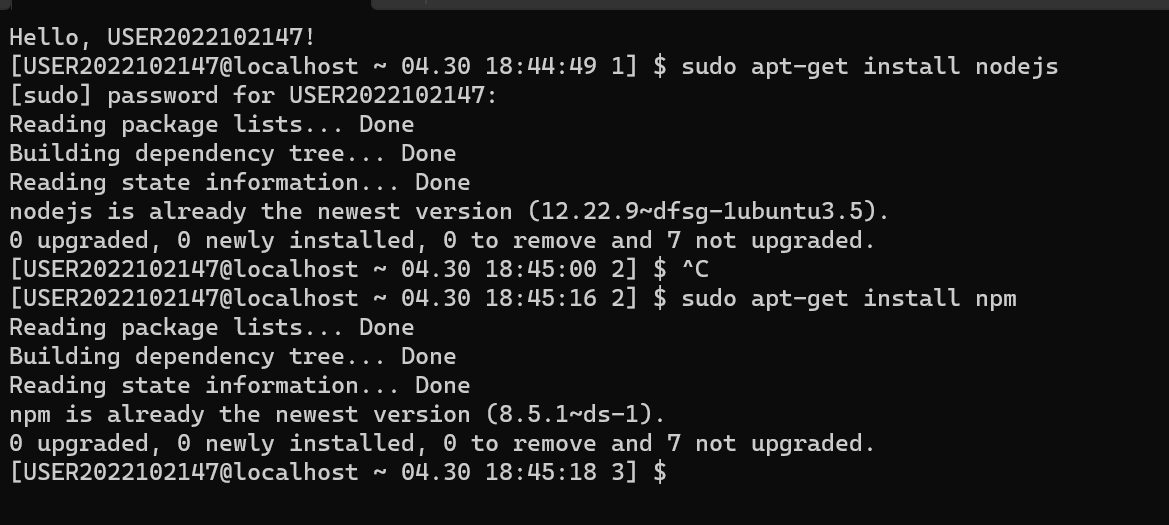
**静态资源**：使用静态网页开发技术（**HTML、CSS、JavaScript**）发布的资源

**动态资源：**使用动态网页开发技术（**jsp/servlet,php,asp...**）发布的资源用nodejs了解前端代码与后端代码。

1)、安装nodejs, npm（之前安装过了，系统有就可以不用重新安装了）

$ sudo apt-get install nodejs

$ sudo apt-get install npm



2）、安装express

<https://www.expressjs.com.cn/starter/installing.html>

安装

首先假定你已经安装了 Node.js，接下来为你的应用创建一个目录，然后进入此目录并将其作为当前工作目录。

$ mkdir my\_app

$ cd my\_app

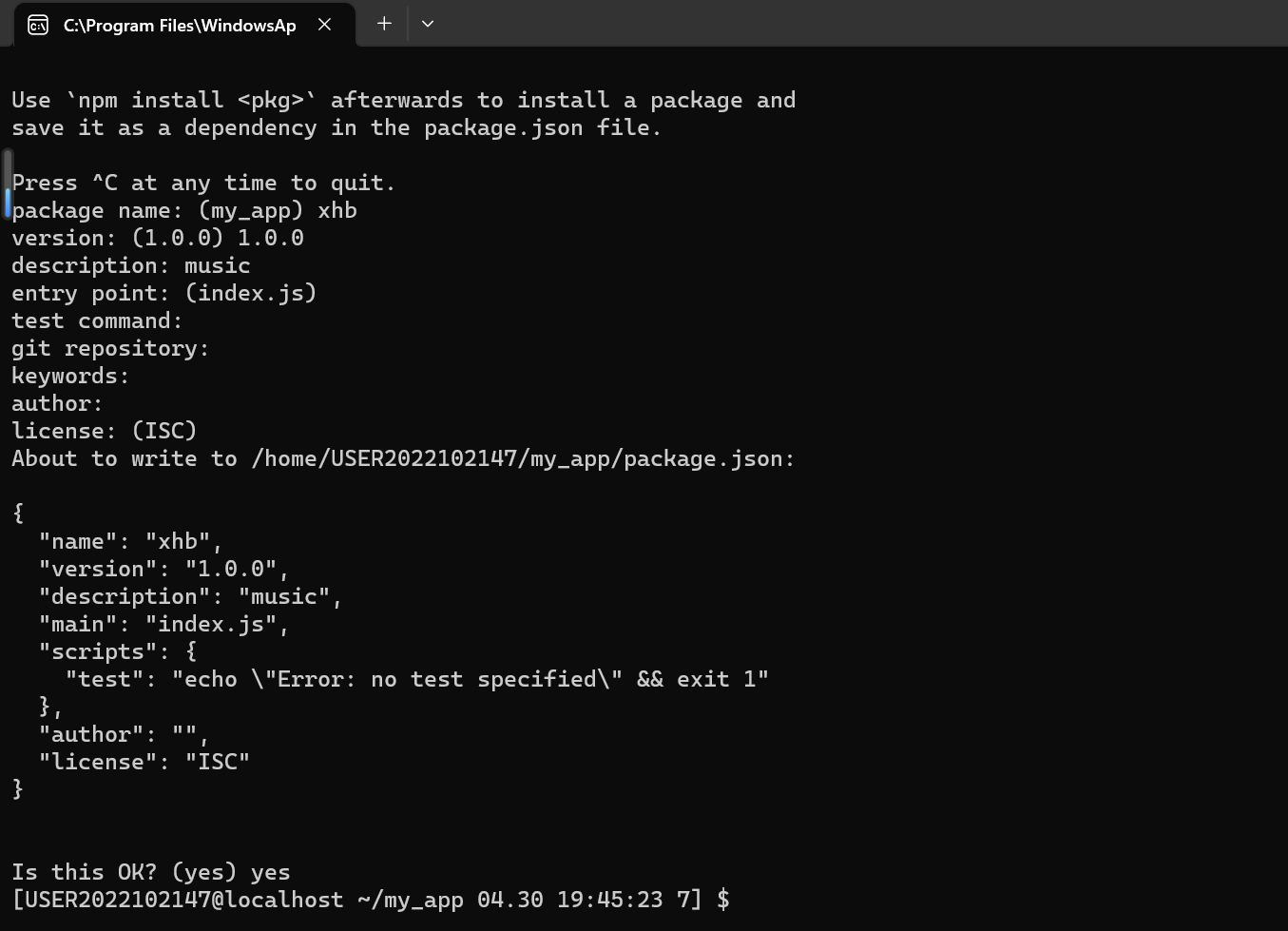
通过 npm init 命令为你的应用创建一个 package.json 文件。 欲了解 package.json 是如何起作用的，请参考 [Specifics of npm’s package.json handling](https://docs.npmjs.com/files/package.json)

$ npm init

此命令将要求你输入几个参数，例如此应用的名称和版本。 你可以直接按“回车”键接受大部分默认设置即可，下面这个除外：

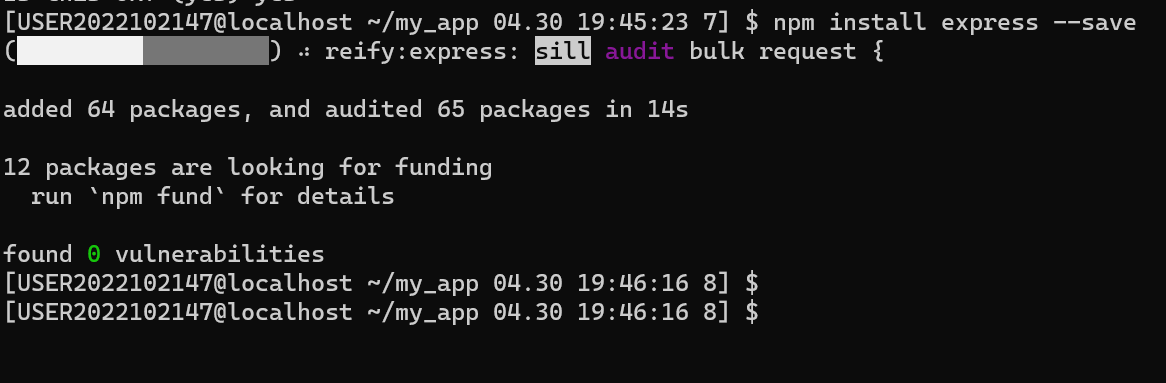
entry point: (index.js)

这是当前应用的入口文件(可以改为你所希望的名称，)。如果你希望采用默认的 index.js 文件名，只需按“回车”键即可。



接下来在 my\_app 目录下安装 Express 并将其保存到依赖列表中。如下：

$ npm install express --save （最好用管理员身份进去下载）



3)、测试 hello world ,

a).查看服务器IP地址：

$ sudo ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 1500

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0xfe<compat,link,site,host>

loop (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wifi0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet *192.168.1.4* netmask 255.255.255.0 broadcast 192.168.1.255

inet6 fe80::d58f:2a44:cf71:81ad prefixlen 64 scopeid 0xfd<compat,link,site,host>

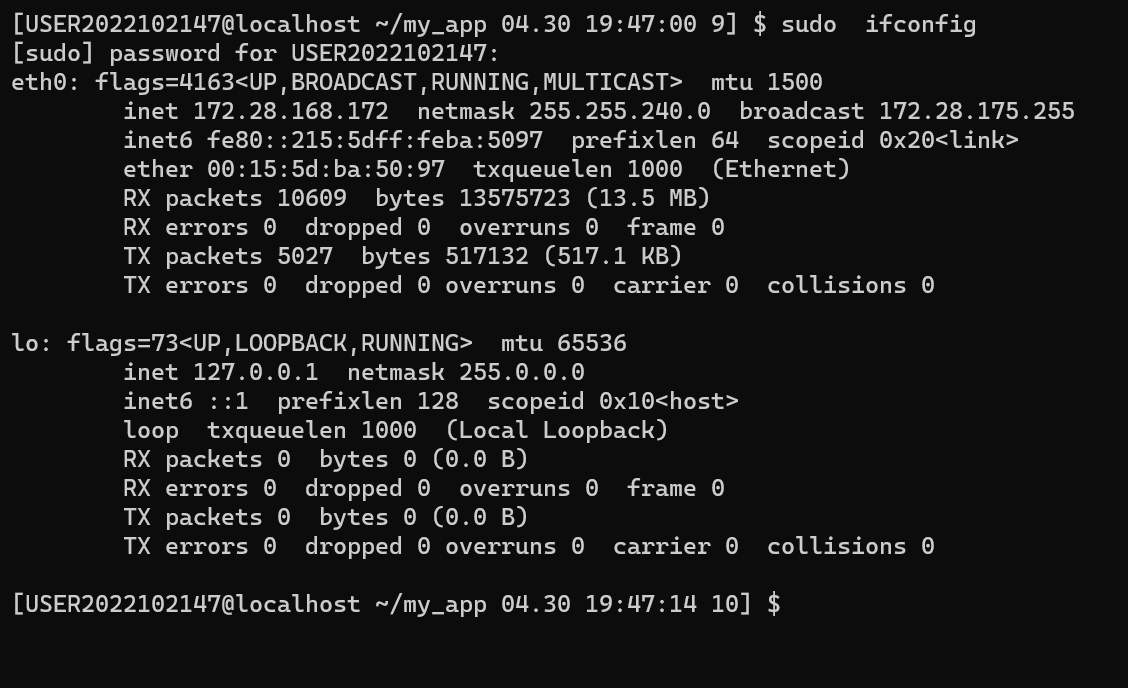
ether d4:d2:52:3e:06:66 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0



b). 将以下程序命名为 index.js

const express = require('express')

// 得到server（服务器）实例

const app = express()

// 绑定服务器接受请求事件，并且添加处理回调函数

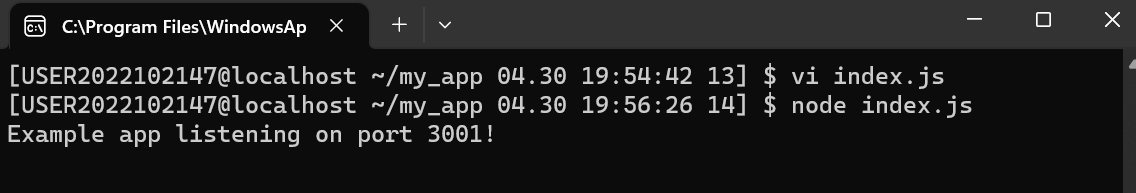
app.get('/', (req, res) => res.send('Hello World from express!'))

// 绑定服务端口，启动服务器

app.listen(3001, () => console.log('Example app listening on port 3001!'))

c）启动服务程序1（使用 **node** 命令执行以上的代码：）

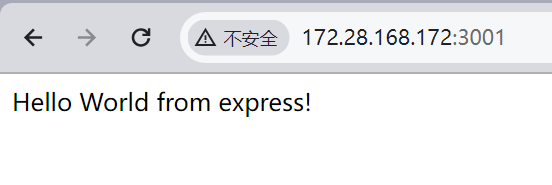
$ node index.js



d)、 在客户端测试：

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3001

[http://*192.168.1.4*:3001/](http://192.168.1.4:3001/)



4)、测试静态资源

a). 将以下内容命名为 add-front.html

<html>

<head>

    <title>加法计算器 </title>

</head>

<body>

    <h1>加法计算器</h1>

    <form onsubmit="return false;">

        <input type="number" id="num1" placeholder="第一个数字">

        <input type="number" id="num2" placeholder="第二个数字">

        <input type="submit" value="计算" onclick="calculate()">

    </form>

    <div id="result"></div>

    <script>

        function calculate() {

            var num1 = parseFloat(document.getElementById('num1').value);

            var num2 = parseFloat(document.getElementById('num2').value);

            var result = num1 + num2;

            document.getElementById('result').innerHTML = "结果：" + result;

        }

    </script>

</body>

</html>

b). 将以下程序命名为 index1.js

const express = require('express')

// 得到server（服务器）实例

const app = express()

// 静态资源在当前目录下

app.use(express.static('.'));

// 绑定服务器接受请求事件，并且添加处理回调函数

app.get('/', (req, res) => res.send('Hello World from express static!'))

// 绑定服务端口，启动服务器

app.listen(3002, () => console.log('Example app listening on port 3002!'))

c）启动服务程序1（使用 **node** 命令执行以上的代码：）

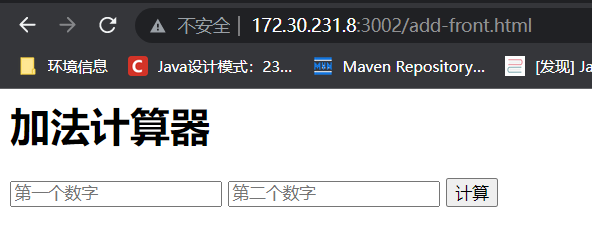
$ node index1.js

d)、 在客户端测试：

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3002

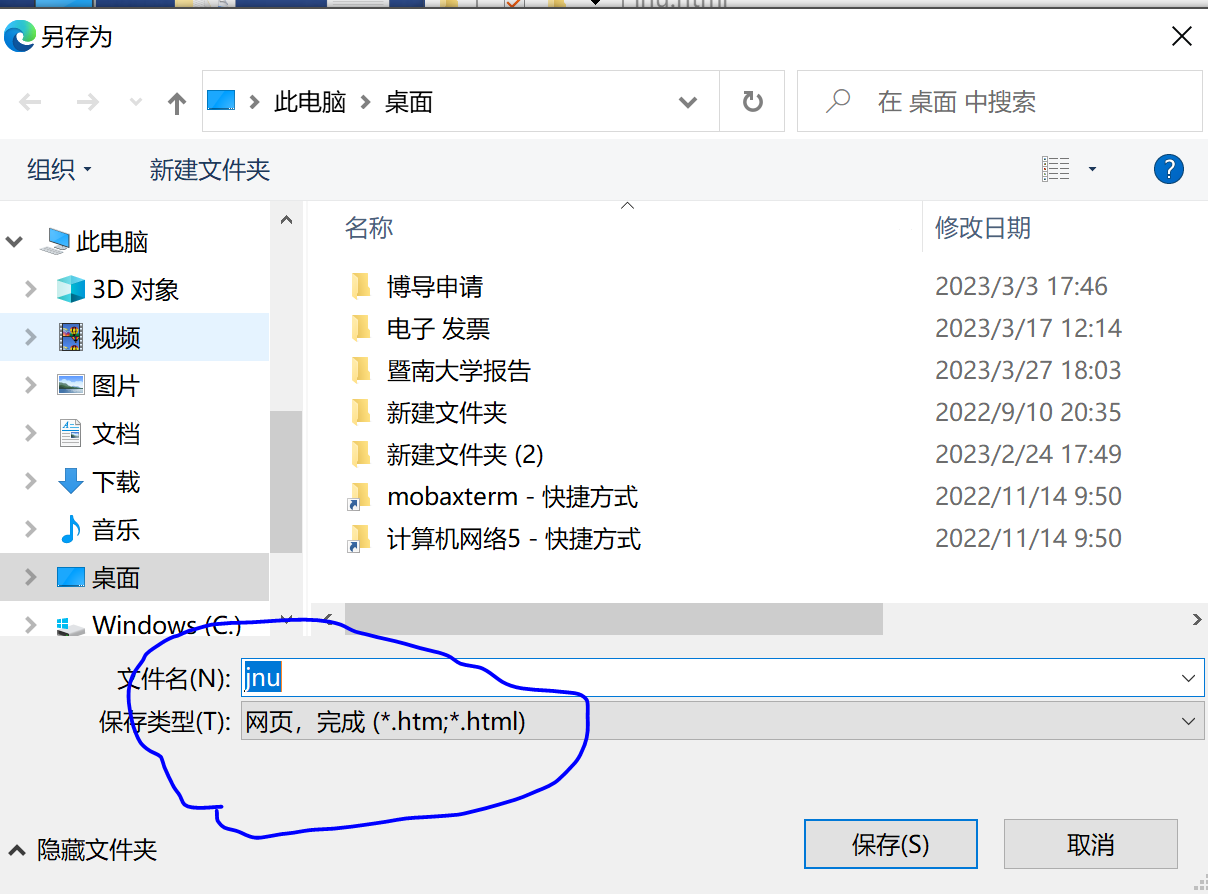
[http://*192.168.1.4*:3002/](http://192.168.1.4:3002/)add-front.html





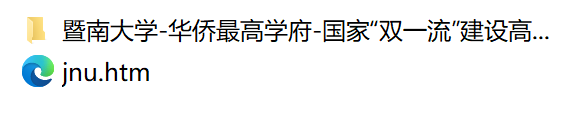
e). 静态WEB网页测试

用浏览器打开网页： <https://www.jnu.edu.cn/main.htm>



另存为文件名 jnu.htm （类型选网页）

将jnu.htm 文件及其子目录下的 文件复制到 myapp 当前目录下。



打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3002

[http://*192.168.1.4*:3002/](http://192.168.1.4:3002/)jnu.htm

5)、测试动态资源

a). 将以下内容命名为 add-back.html

<!DOCTYPE html>

<html>

<head>

    <title>提交两个数字</title>

</head>

<body>

    <h1>提交两个数字</h1>

    <form onsubmit="return false;">

        <input type="number" id="num1" placeholder="第一个数字">

        <input type="number" id="num2" placeholder="第二个数字">

        <input type="submit" value="提交" onclick="submitNumbers()">

    </form>

    <div id="result"></div>

    <script>

        function submitNumbers() {

            var num1 = parseFloat(document.getElementById('num1').value);

            var num2 = parseFloat(document.getElementById('num2').value);

            var xhr = new XMLHttpRequest();

            xhr.onreadystatechange = function() {

                if (xhr.readyState === 4 && xhr.status === 200) {

                    document.getElementById('result').innerHTML = xhr.responseText;

                }

            }

            xhr.open('GET', `/back-end?num1=${num1}&num2=${num2}`, true);

            xhr.send();

        }

    </script>

</body>

</html>

b). 将以下程序命名为 index2.js

const express = require('express')

const url = require('url');

const app = express()

// 静态资源在当前目录下

app.use(express.static('.'));

app.get('/', (req, res) => res.send('Hello World from express with back end!'))

// 后端处理代码

app.get('/back-end', (req, res) => {

const query = url.parse(req.url, true).query;

const num1 = parseFloat(query.num1);

const num2 = parseFloat(query.num2);

const result = num1 + num2;

result\_str = `${num1} + ${num2} = ${result}`

// 发送HTTP响应

res.send(result\_str);

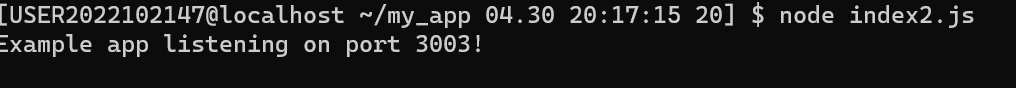
} )

// 绑定服务端口，启动服务器

app.listen(3003, () => console.log('Example app listening on port 3003!'))

c）启动服务程序（使用 **node** 命令执行以上的代码：）

$ node index2.js



d)、 在客户端测试1：

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3003

[http://*192.168.1.4*:3003/](http://192.168.1.4:3003/)add-back.html



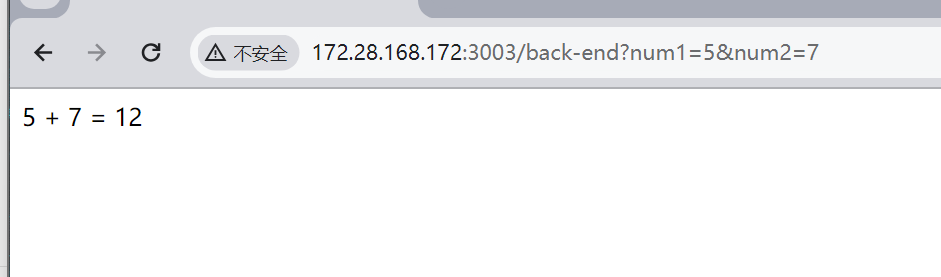


e)、 在客户端测试2：

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3003

用chrome开发者工具，观察http的请求与响应过程。

[http://*192.168.1.4*:3003/](http://192.168.1.4:3003/)back-end?num1=3&num2=5



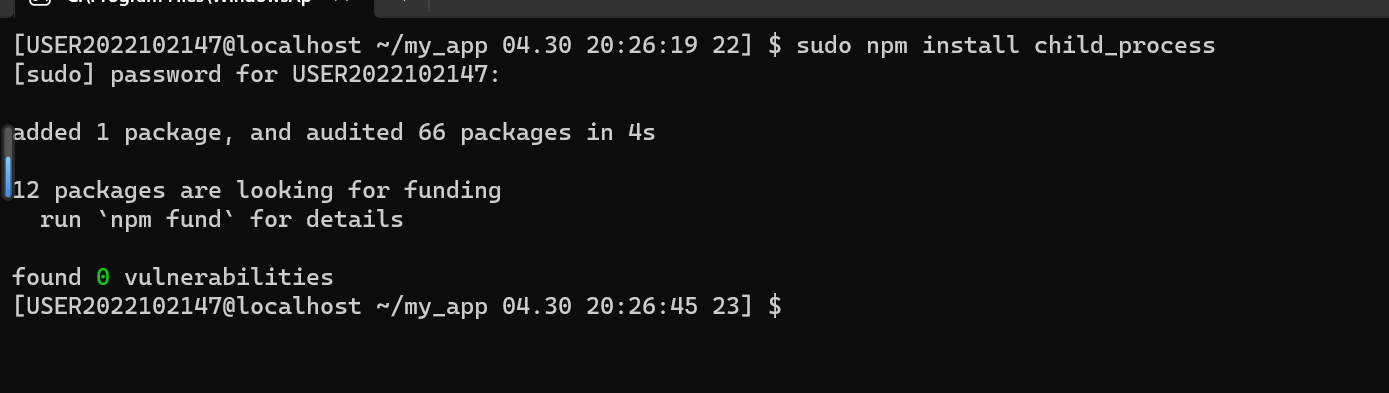
6)、测试动态资源

a). 安装child\_process

<https://juejin.cn/post/6844903612842246157>

参考以上网页，安装child\_process

sudo npm install child\_process



b). 将以下程序命名为 index3.js

const express = require('express')

const url = require('url');

const {exec} = require('child\_process');

const app = express()

// 静态资源在当前目录下

app.use(express.static('.'));

app.get('/', (req, res) => res.send('Hello World from express with back end!'))

// 后端处理代码

app.get('/back-end', (req, res) => {

const query = url.parse(req.url, true).query;

const num1 = parseFloat(query.num1);

const num2 = parseFloat(query.num2);

const result = num1 + num2;

result\_str = `${num1} + ${num2} = ${result}`

// 发送HTTP响应

res.send(result\_str);

} )

app.get('/ls', (req, res) => {

// 输出当前目录（不一定是代码所在的目录）下的文件和文件夹

exec('ls -l', (err, stdout, stderr) => {

    if(err) {

        console.log(err);

        return;

    }

    console.log(`stdout: ${stdout}`);

   // console.log(`stderr: ${stderr}`);

    res.send(`stdout: ${stdout}`);

});

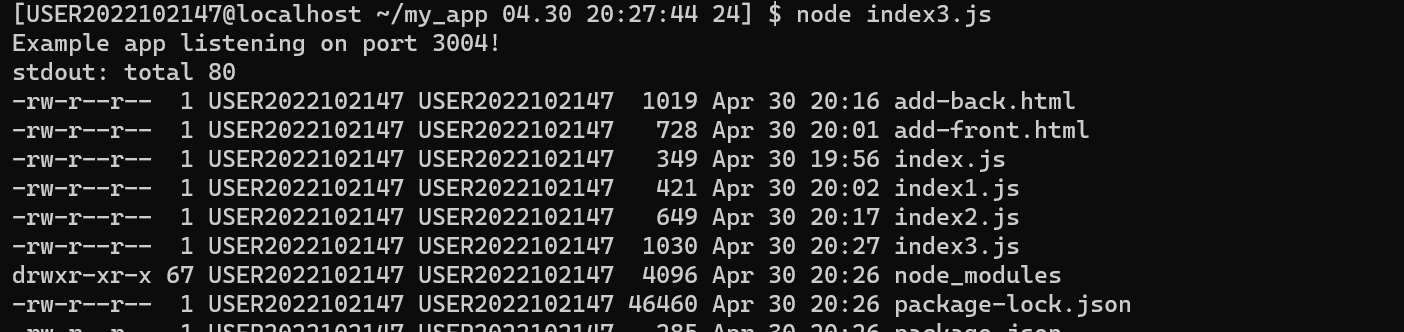
} )

// 绑定服务端口，启动服务器

app.listen(3004, () => console.log('Example app listening on port 3004!'))

c）启动服务程序（使用 **node** 命令执行以上的代码：）

$ node index3.js

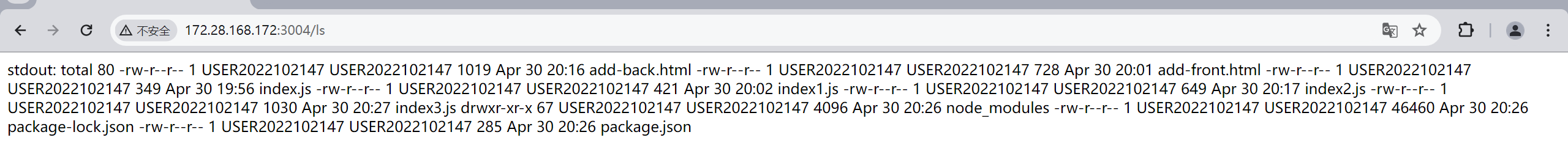


d)、 在客户端测试1：

打开浏览器，输入URL：（IP地址为前面通过 ifconfig 查到的地址），注意端口是 3004

用chrome开发者工具，观察http的请求与响应过程。

[http://*192.168.1.4*:3004/](http://192.168.1.4:3004/)ls



1. 参考网页：

<https://blog.csdn.net/weixin_45730130/article/details/124502960>

<https://blog.csdn.net/weixin_48994377/article/details/108575318>

编写回声服务器，及对应的客户机。

1. TCP服务器：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

typedef struct sockaddr\* SP;

int main(int argc,const char\* argv[])

{

    // 创建socket

    int sockfd = socket(AF\_INET,SOCK\_STREAM,0);

    if(0 > sockfd)

    {

        perror("socket");

        return EXIT\_FAILURE;

    }

    // 准备地址

    struct sockaddr\_in addr = {};

    addr.sin\_family = AF\_INET;

    addr.sin\_port = htons(3031);

    addr.sin\_addr.s\_addr = INADDR\_ANY;

    socklen\_t addrlen = sizeof(addr);

    // 绑定

    if(bind(sockfd,(SP)&addr,addrlen))

    {

        perror("bind");

        return EXIT\_FAILURE;

    }

    // 监听

    if(listen(sockfd,5))

    {

        perror("listen");

        return EXIT\_FAILURE;

    }

    // 等待连接

    int clifd = accept(sockfd,(SP)&addr,&addrlen);

    if(0 > clifd)

    {

        perror("accept");

        return EXIT\_FAILURE;

    }

    char buf[4096] = {};

    size\_t buf\_size = sizeof(buf);

    for(;;)

    {

        // 接收/返回数据

        memset(buf,0,sizeof(buf));

        size\_t ret\_size = read(clifd,buf,sizeof(buf));

        if(0 == strcmp("quit",buf))

        {

            printf("通信结束!\n");

            break;

        }

        if(0 == ret\_size)

        {

            printf("连接断开!\n");

            break;

        }

        printf("recv:[%s] bytes:%d\n",buf,(int)ret\_size);

        strcat(buf,":return");

        write(clifd,buf,strlen(buf)+1);

    }

    // 关闭socket

    close(clifd);

}

将以上程序命令为 echo-s.c，编译并运行服务程序

gcc echo-s.c –o echo-s

./echo-s

1. 客户机

import socket

# 创建一个socket对象

s1 = socket.socket()

s1.connect(('127.0.0.1', 3031))

# 不断发送和接收数据

while 1:

    send\_data = input("客户端要发送的信息：")

    # socket传递的都是bytes类型的数据,需要转换一下

    s1.send(send\_data.encode("utf-8"))

    # 接收数据，最大字节数1024,对返回的二进制数据进行解码

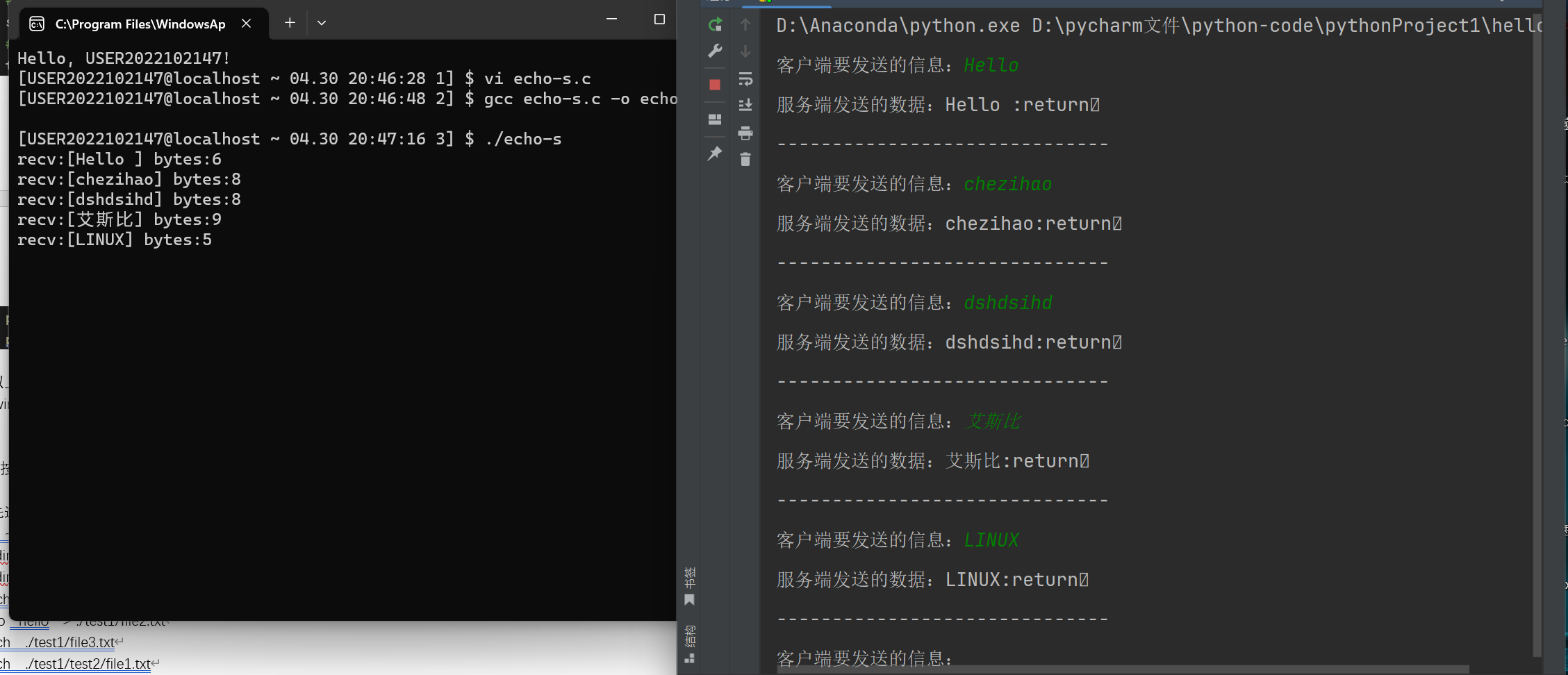
    text = s1.recv(1024).decode("utf-8")

    print("服务端发送的数据：{}".format(text))

    print("------------------------------")

将以上程序命名为 client.py

在windows 端， 用python运行以上程序 。



1. 按要求编写脚本，给出在您的环境下的运行过程与结果。

首先运行以下命令，创建测试需要的文件

cd ~

mkdir test1

mkdir ./test1/test2

touch ./test1/file1.txt

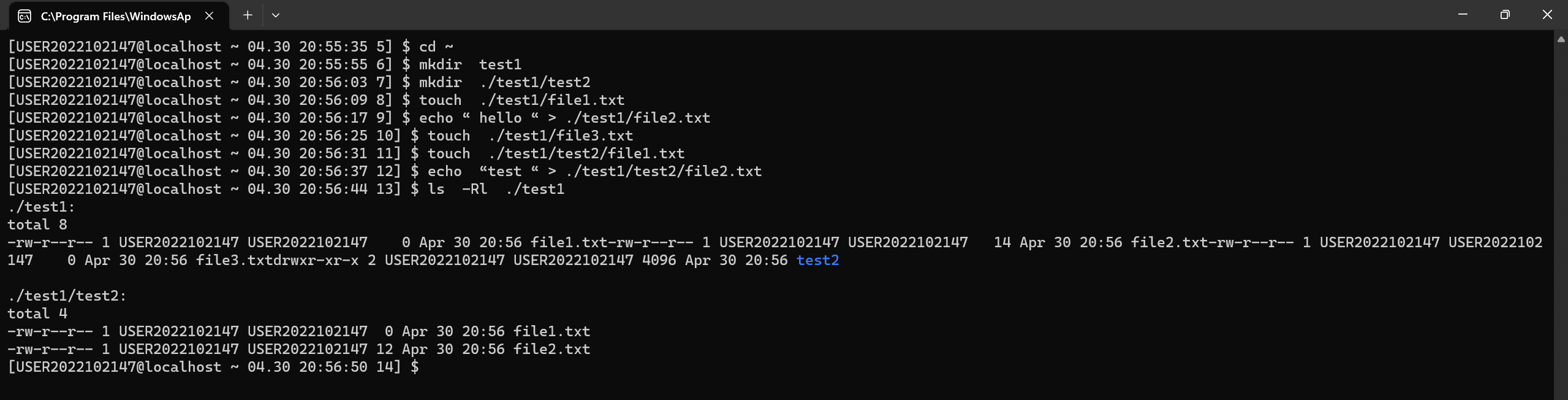
echo “ hello “ > ./test1/file2.txt

touch ./test1/file3.txt

touch ./test1/test2/file1.txt

echo “test “ > ./test1/test2/file2.txt

ls -Rl ./test1

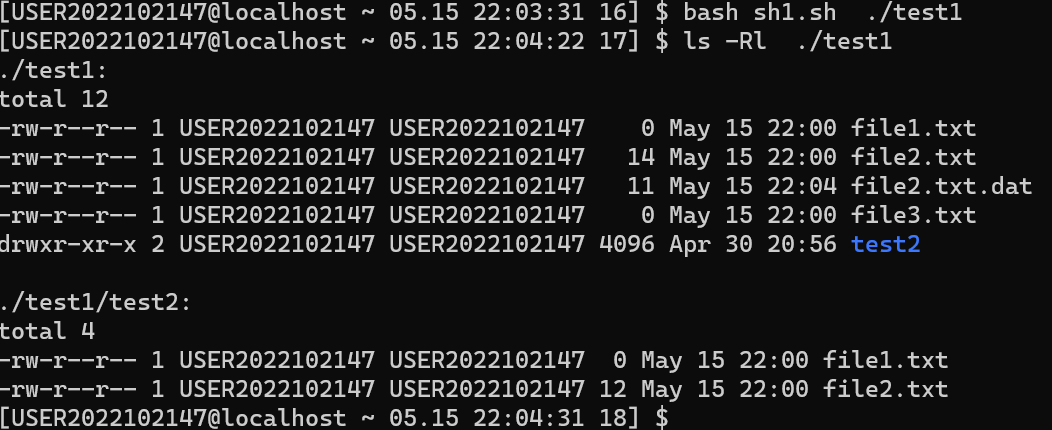


1）、编写脚本sh1 ，将指定目录下的文件中，对含有“hello”的文件进行处理，将该文件中的“hello” 替换为 “hi”，（保留原文件）新生成替换后的文件名在原文件名后加“.dat”。（即 file2.txt 🡺 file2.txt.dat）。提示：用sed命令。

cd ~

ls -Rl ./test1

bash sh1 ./test1

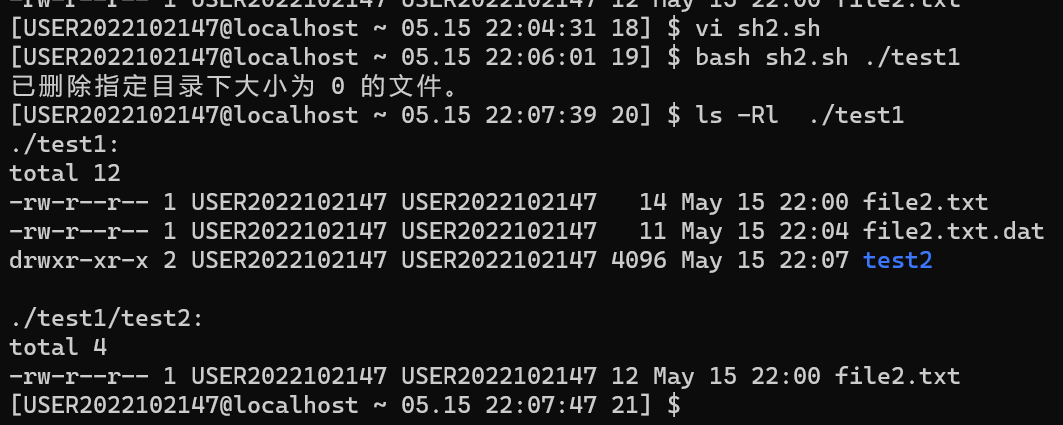


2）、编写脚本sh2, 删除指定目录下，文件大小为 0 的文件

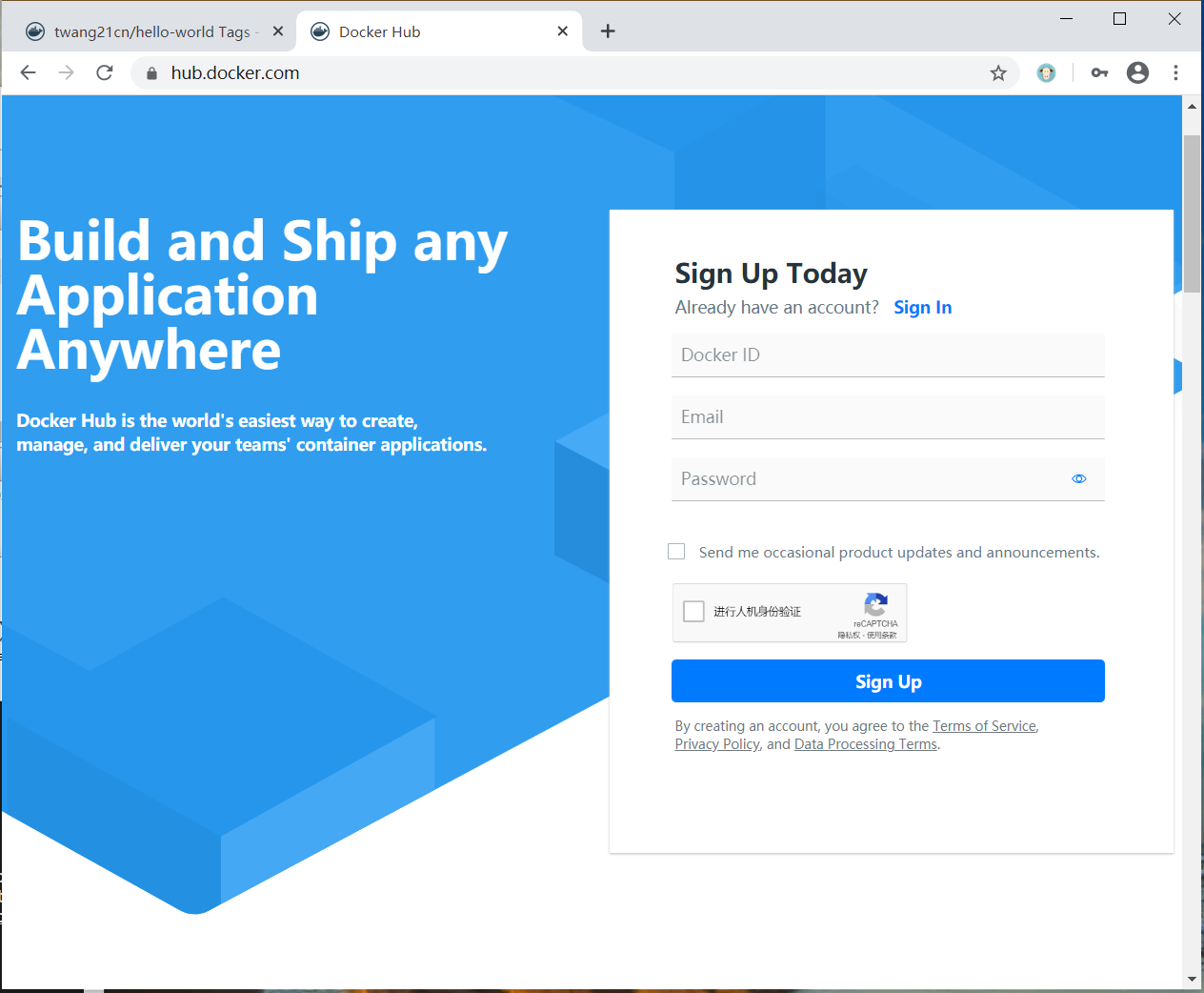
cd ~

bash sh2.sh ./test1

ls -Rl ./test1

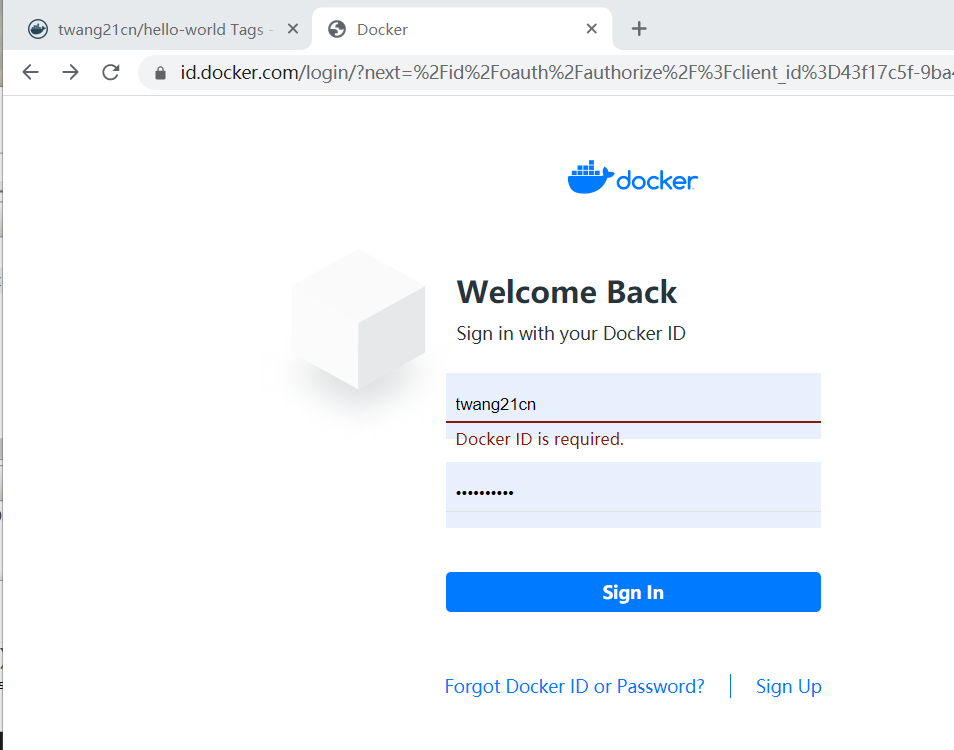


1. 注册docker仓库帐号，安装docker，将hello-world保存到自己的docker仓库中。
2. 注册 hub.docker.com

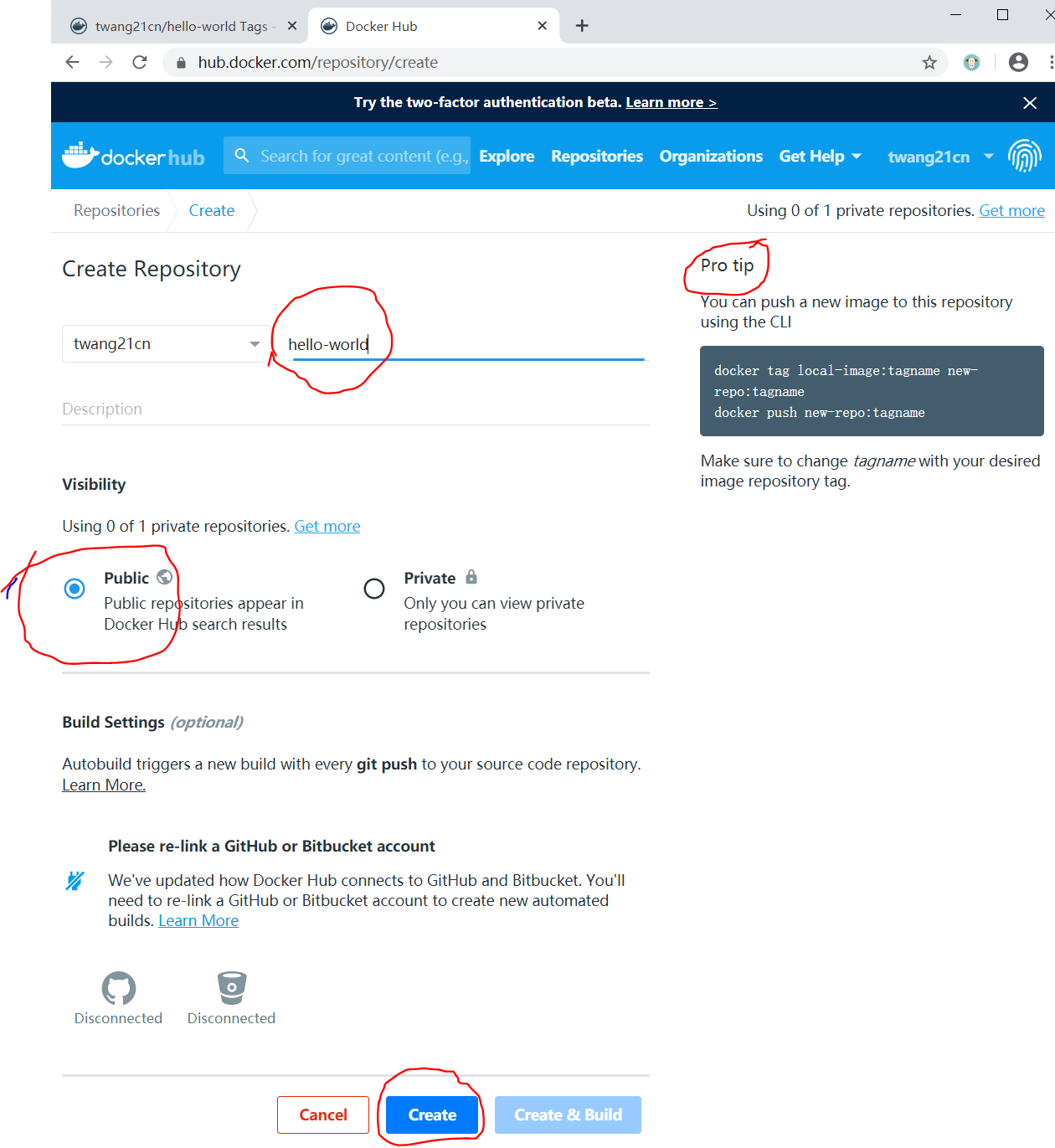


通过邮件 注册, 激活 一个用户 twang21cn (后续请用你自己的名字替代 twang21cn)

1. 登录 hub.docker.com



1. 创建一个仓库



1. ubuntu下安装docker

来源于官方文档

<https://docs.docker.com/engine/install/ubuntu/>

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

注：（换源过的）下载的过程中，如果出现网络问题，可以试试进行以下操作：

sudo vi /etc/resolv.conf

在打开的文件中添加以下两行，分别指定首选DNS服务器和备用DNS服务器：

nameserver 8.8.8.8

nameserver 8.8.4.4

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install \

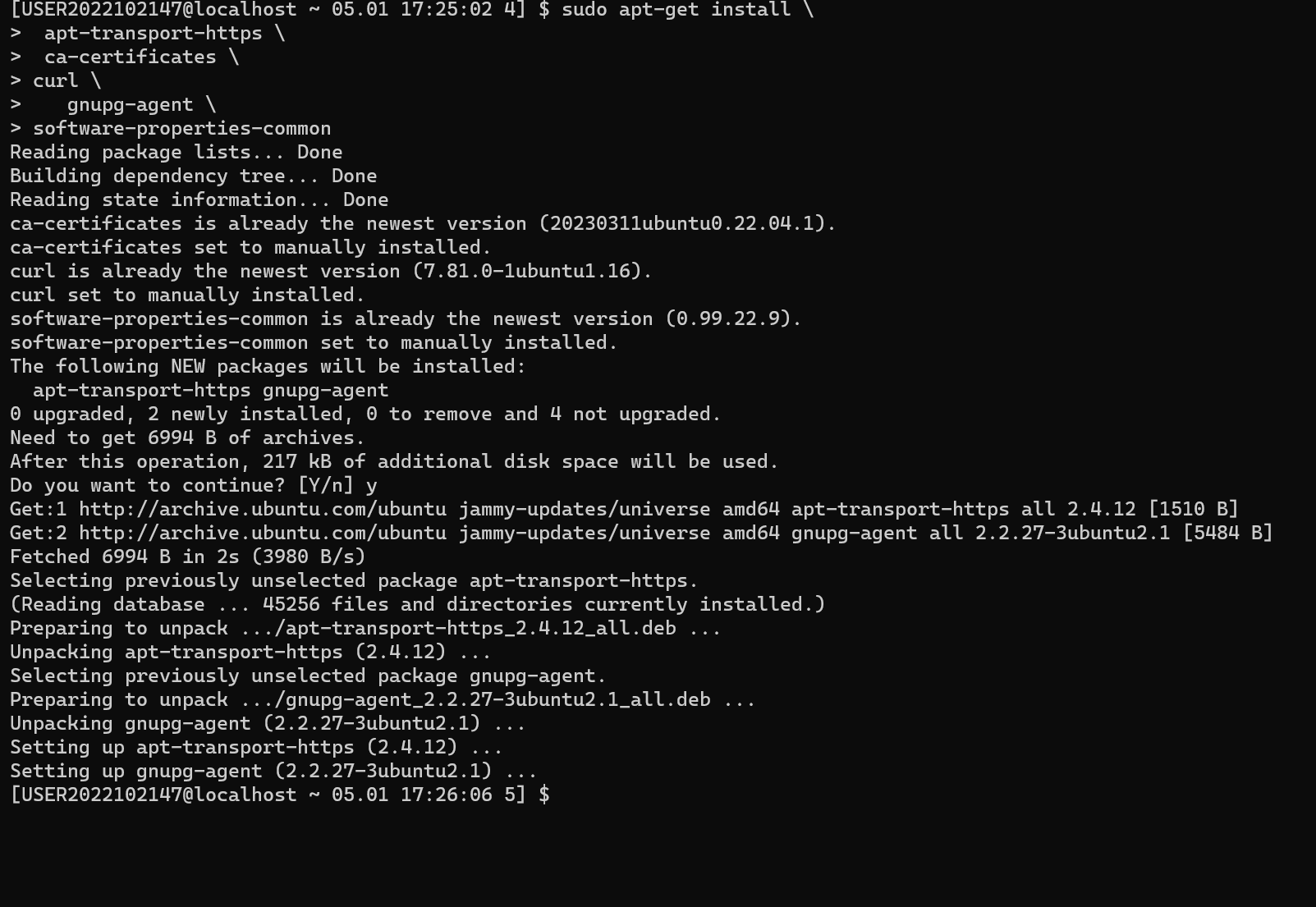
apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common



1. Add Docker’s official GPG key:

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -



Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

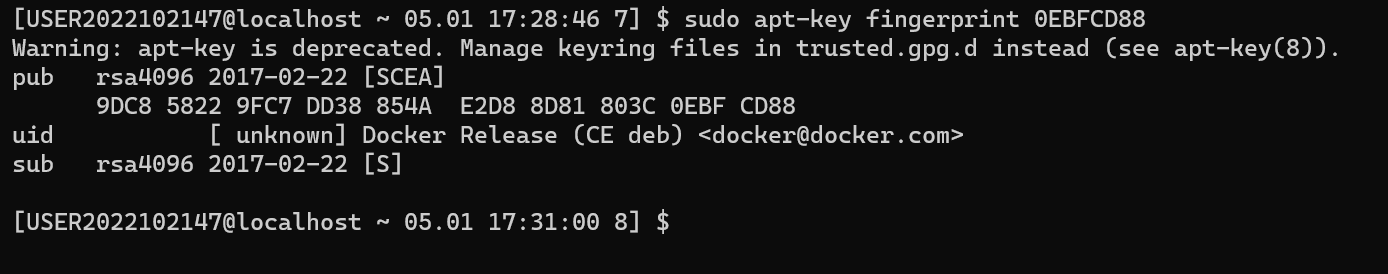
$ sudo apt-key fingerprint 0EBFCD88

pub rsa4096 2017-02-22 [SCEA]

9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid [ unknown] Docker Release (CE deb) <docker@docker.com>

sub rsa4096 2017-02-22 [S]



1. Use the following command to set up the **stable** repository. To add the **nightly** or **test** repository, add the word nightly or test (or both) after the word stable in the commands below. [Learn about **nightly** and **test** channels](https://docs.docker.com/engine/install/).

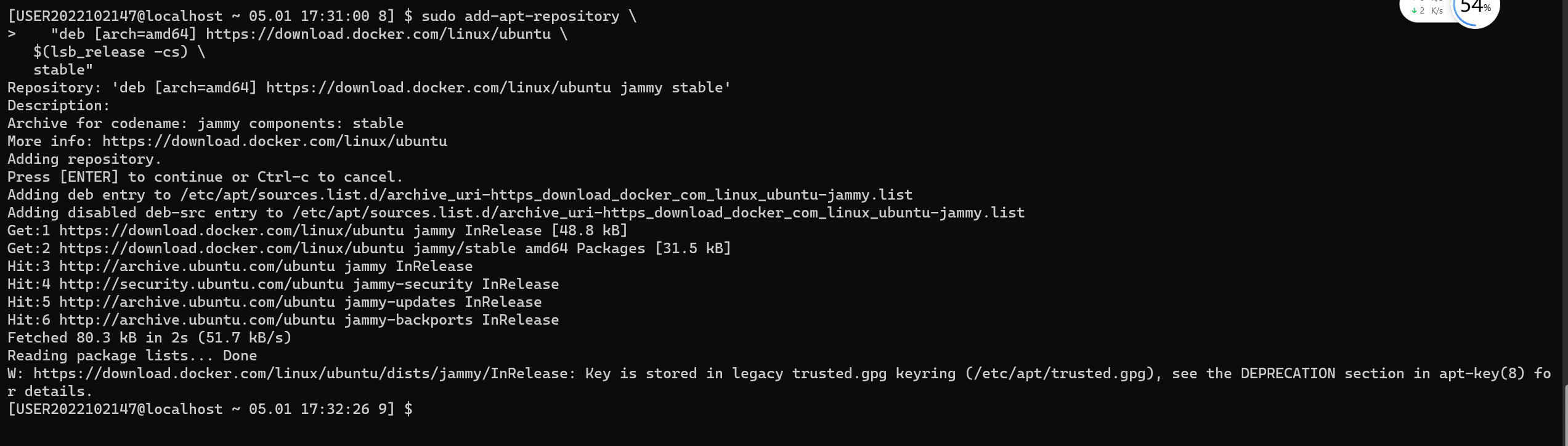
**Note**: The lsb\_release -cs sub-command below returns the name of your Ubuntu distribution, such as xenial. Sometimes, in a distribution like Linux Mint, you might need to change $(lsb\_release -cs) to your parent Ubuntu distribution. For example, if you are using Linux Mint Tessa, you could use bionic. Docker does not offer any guarantees on untested and unsupported Ubuntu distributions.

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

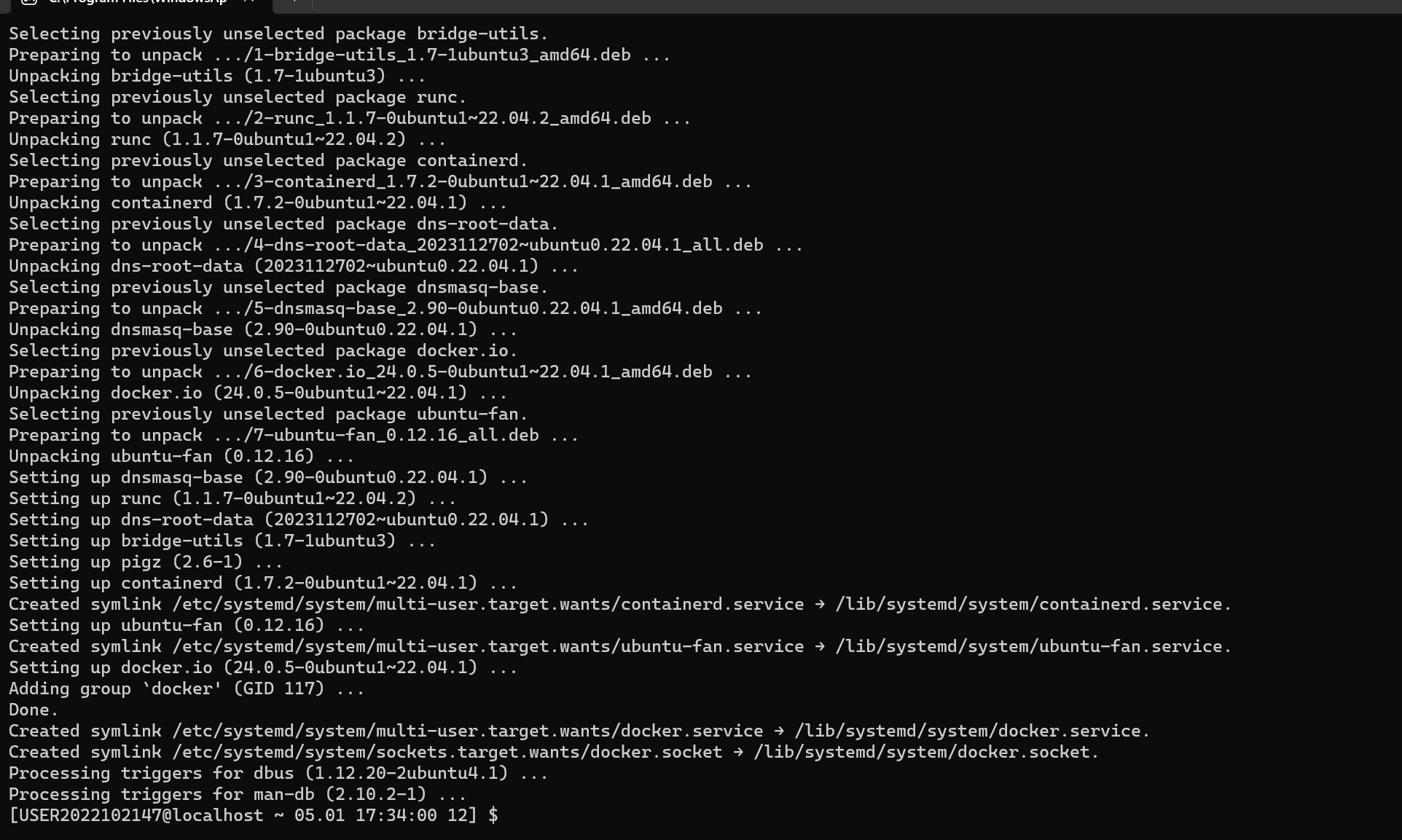
stable"



1. INSTALL DOCKER ENGINE
2. Inform the apt package index, and install the latest version of Docker Engine and containerd, or go to the next step to install a specific version:

$ sudo apt-get update

$ sudo apt-get install docker.io

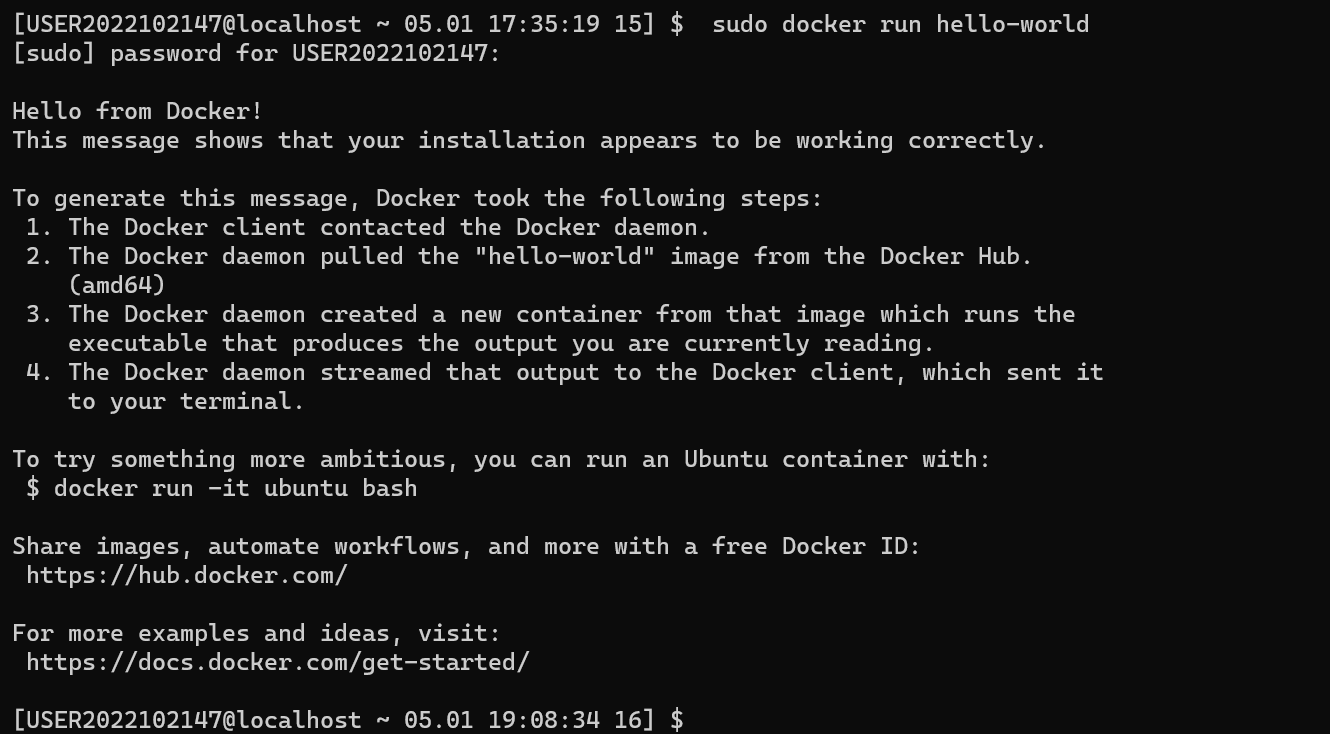


1. Verify that Docker Engine is installed correctly by running the hello-world image.

$ sudo service docker start

$ sudo docker run hello-world

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

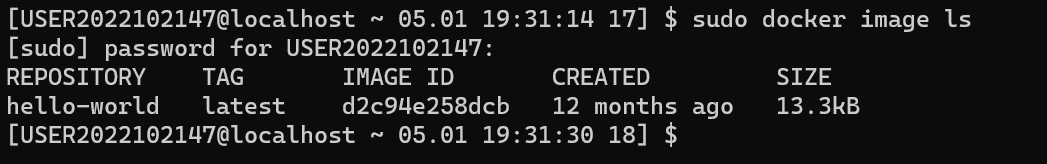


1. 将镜像推送到自己的仓库，及下载自己的镜像
2. 列出镜像

#### wang@wang-Virtual-Machine:~$ sudo docker image ls

#### REPOSITORY TAG IMAGE ID CREATED SIZE

#### hello-world latest bf756fb1ae65 4 months ago 13.3kB



1. 登录docker仓库

#### wang@wang-Virtual-Machine:~$ sudo docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: twang21cn

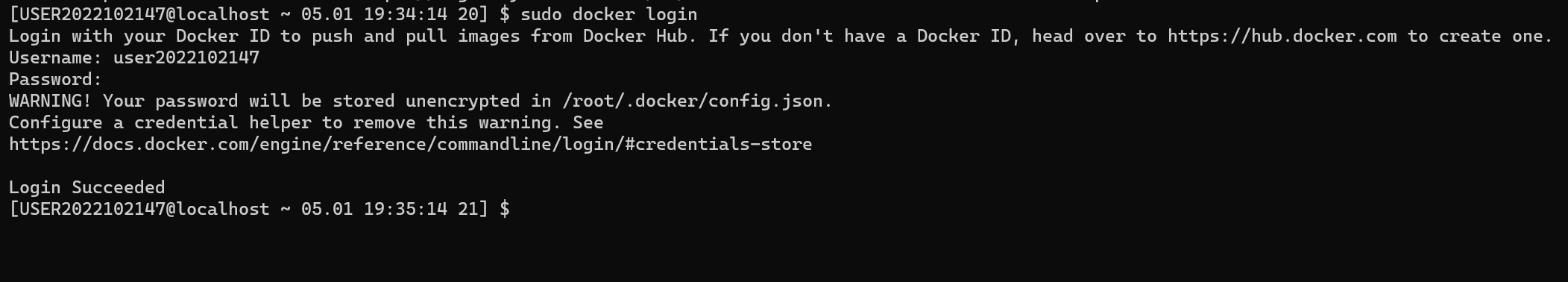
Password:

WARNING! Your password will be stored unencrypted in /home/wang/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded



1. 将镜像设置标签：用户名/镜像名

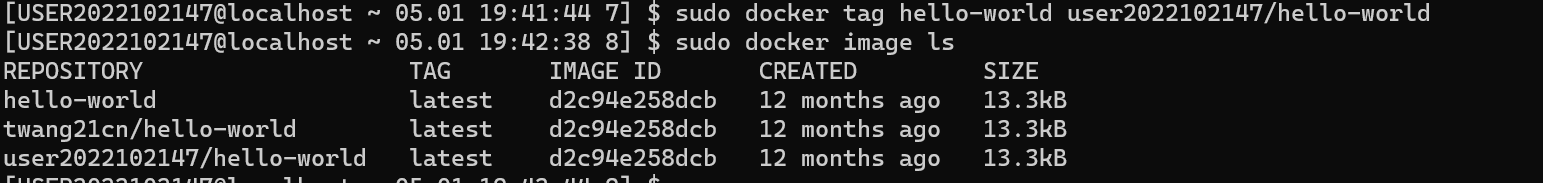
#### wang@wang-Virtual-Machine:~$ sudo docker tag hello-world twang21cn/hello-world

#### wang@wang-Virtual-Machine:~$ sudo docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

twang21cn/hello-world latest bf756fb1ae65 4 months ago 13.3kB

hello-world latest bf756fb1ae65 4 months ago 13.3kB



1. 将镜像推送到自己的仓库：用户名/镜像名

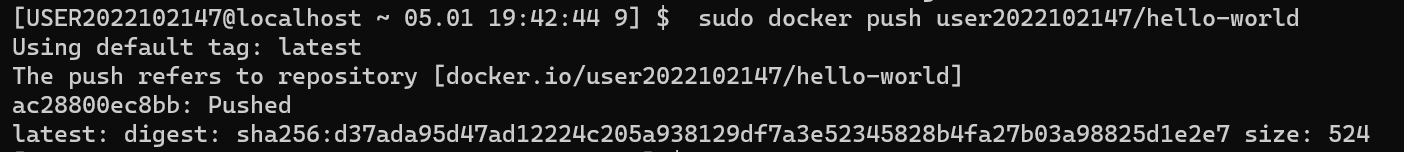
#### wang@wang-Virtual-Machine:~$ sudo docker push twang21cn/hello-world

The push refers to repository [docker.io/twang21cn/hello-world]

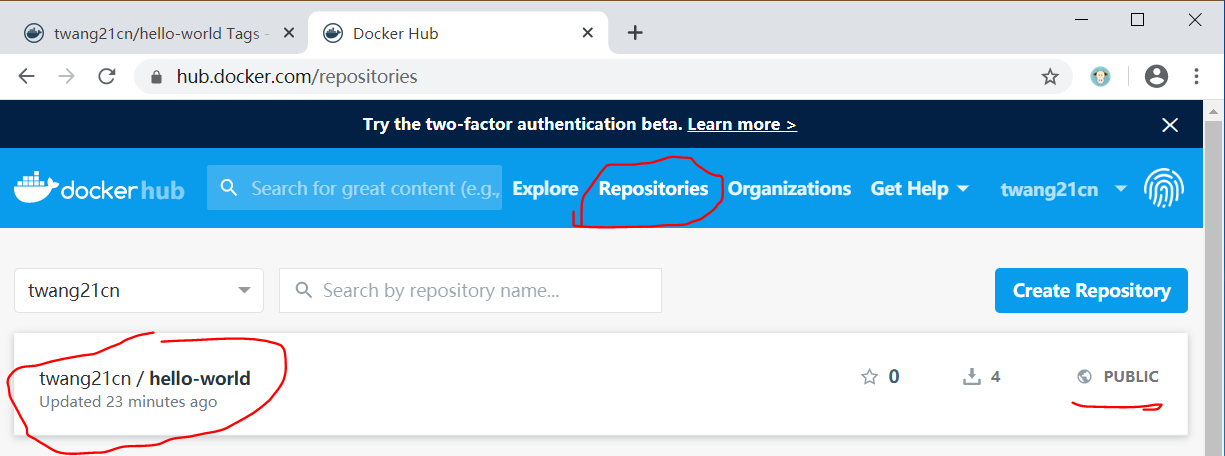
9c27e219663c: Mounted from library/hello-world

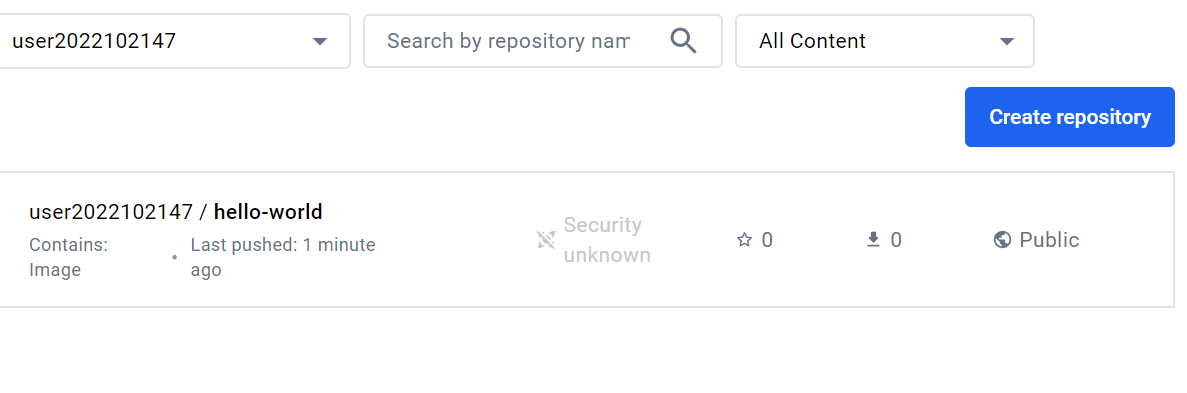
latest: digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042 size: 525

wang@wang-Virtual-Machine:~$



1. 可以在hub.docker.com下的自己仓库中看到这个镜像。





1. 下载自己镜像：用户名/镜像名

#### wang@wang-Virtual-Machine:~$ sudo docker pull twang21cn/hello-world

[sudo] password for wang:

Using default tag: latest

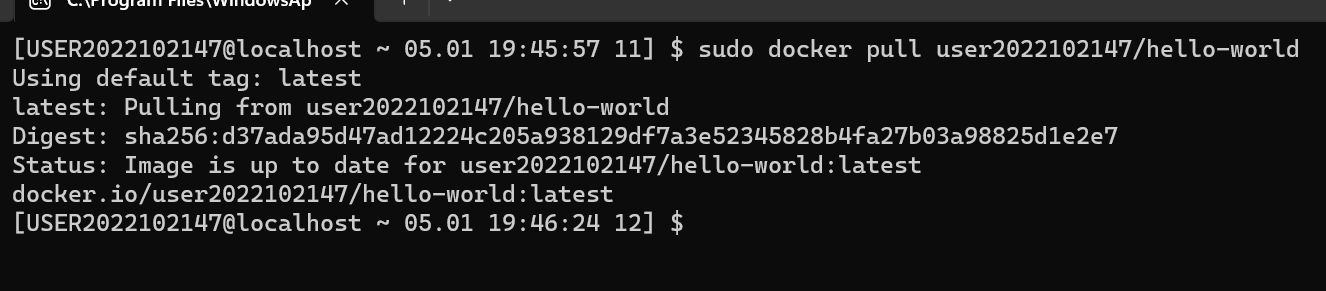
latest: Pulling from twang21cn/hello-world

Digest: sha256:90659bf80b44ce6be8234e6ff90a1ac34acbeb826903b02cfa0da11c82cbc042

Status: Image is up to date for twang21cn/hello-world:latest

docker.io/twang21cn/hello-world:latest

wang@wang-Virtual-Machine:~$



6、编写简单mygrep程序,

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 1024

int main(int argc, char \*argv[]) {

    // Check if filename and string to search are provided as arguments

    if (argc != 3) {

        printf("Usage: %s <string> <filename> \n", argv[0]);

        return 1;

    }

    FILE \*file = fopen(argv[2], "r"); // Open the file

    if (file == NULL) {

        printf("Error: File '%s' not found\n", argv[2]);

        return 1;

    }

    char line[MAX\_LINE\_LENGTH];     // Read each line from the file

    while (fgets(line, MAX\_LINE\_LENGTH, file) != NULL)

        // Search for the string in the line

        if (strstr(line, argv[1]) != NULL)   printf("%s", line);

    fclose(file);      // Close the file

    return 0;

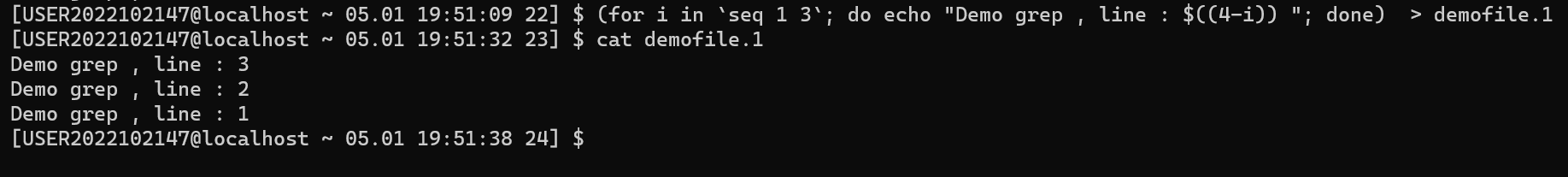
}

将以上程序命名为 mygrep.c

1）、运行以下命令，生成测试用的数据。

(for i in `seq 1 3`; do echo "Demo grep , line : $((4-i)) "; done) > demofile.1

cat demofile.1

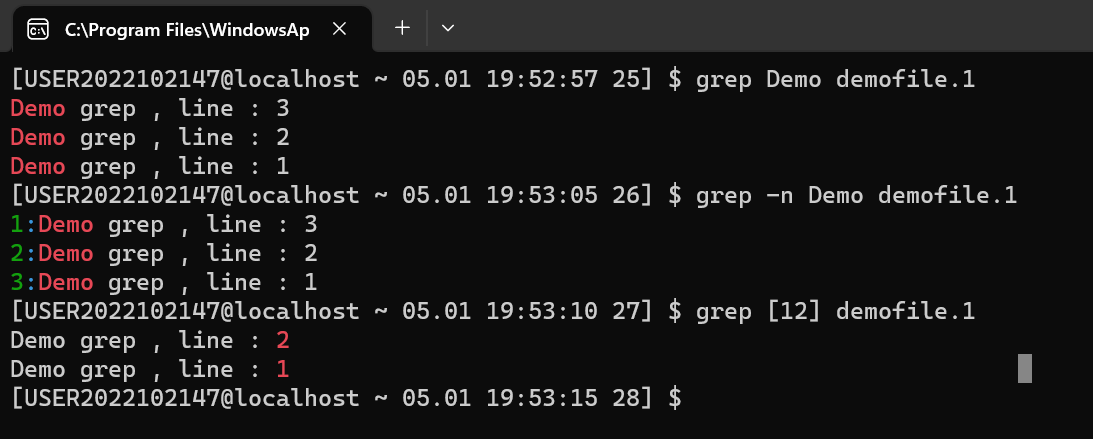


2）、运行以下命令，查看grep命令的运行结果

grep Demo demofile.1

grep -n Demo demofile.1

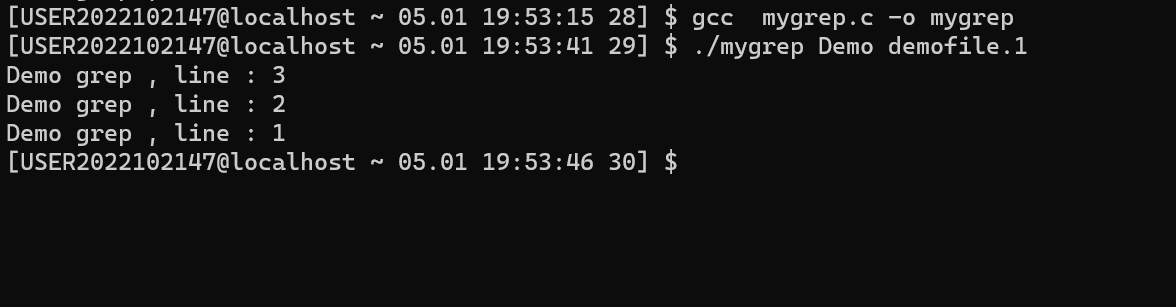
grep [12] demofile.1



3）、运行以下命令，查看mygrep的运行结果

gcc mygrep.c -o mygrep

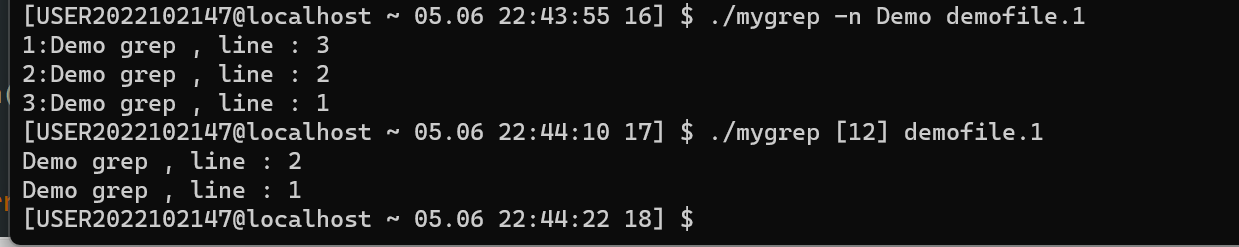
./mygrep Demo demofile.1

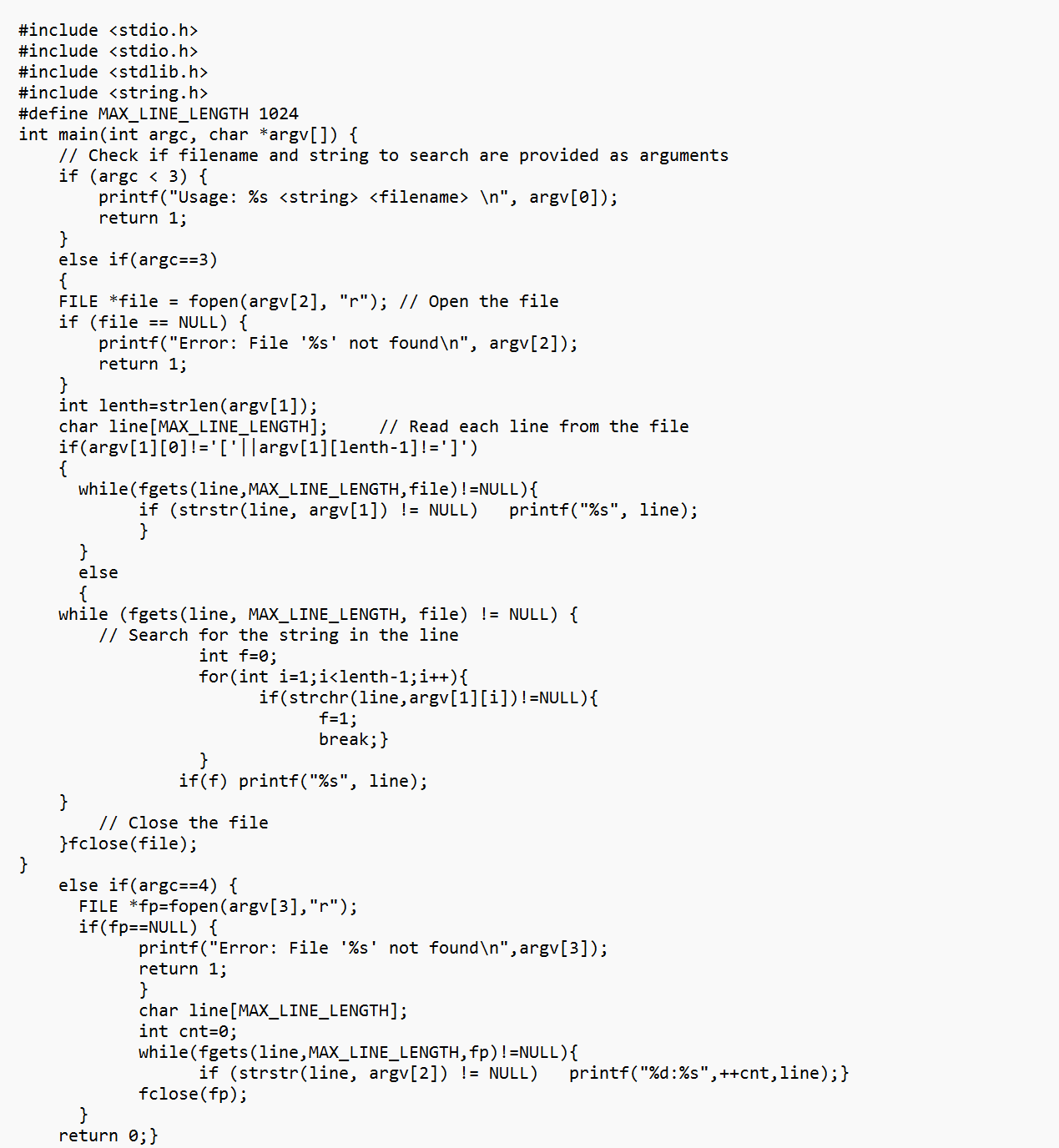


4)、改写mygrep.c 程序， 增加 -n 选项及支持正则静态式查找功能。

./mygrep -n Demo demofile.1

./mygrep [12] demofile.1





正则表达式请参考：

<https://www.cnblogs.com/stlong/p/6289107.html>

1. 理解管道操作。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE\_LENGTH 1024

int main(int argc, char \*argv[]) {

    // Check if filename and string to search are provided as arguments

    FILE \*file ;

    if (argc < 2) {

        printf("Usage: %s <string> <filename> \n", argv[0]);

        return 1;

    }

    if(argc==3)

    {

    file = fopen(argv[2], "r"); // Open the file

    if (file == NULL) {

        printf("Error: File '%s' not found\n", argv[2]);

        return 1;

    }

    }

    else

     file = stdin;

    char line[MAX\_LINE\_LENGTH];     // Read each line from the file

    while (fgets(line, MAX\_LINE\_LENGTH, file) != NULL)

        // Search for the string in the line

        if (strstr(line, argv[1]) != NULL)   printf("%s", line);

    fclose(file);      // Close the file

    return 0;

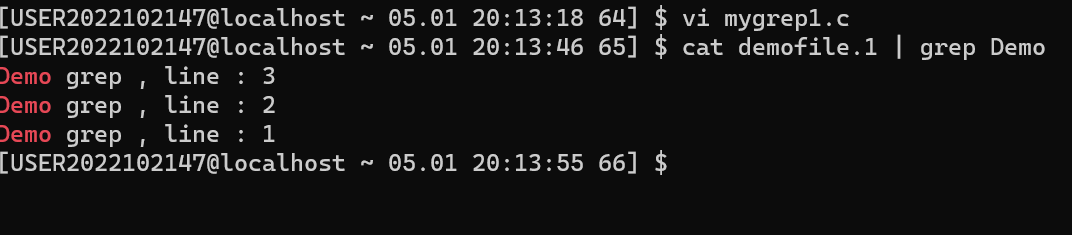
}

将以上程序命名为 mygrep1.c

用上一题目的测试数据。

1）、运行以下命令，查看grep命令的运行结果。

cat demofile.1 | grep Demo

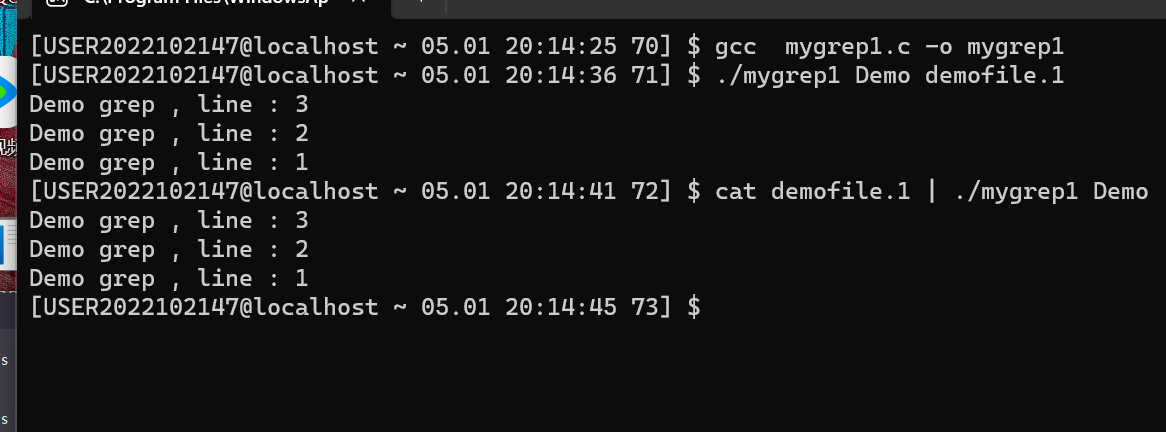


2）、运行以下命令，查看mygrep2的运行结果。解释本题程序与上一题的差异？

gcc mygrep1.c -o mygrep1

./mygrep1 Demo demofile.1

cat demofile.1 | ./mygrep1 Demo



管道 | 命令使用详解：

<https://blog.csdn.net/xhtchina/article/details/112931829>

Linux命令中的重定向(>)和管道(|)

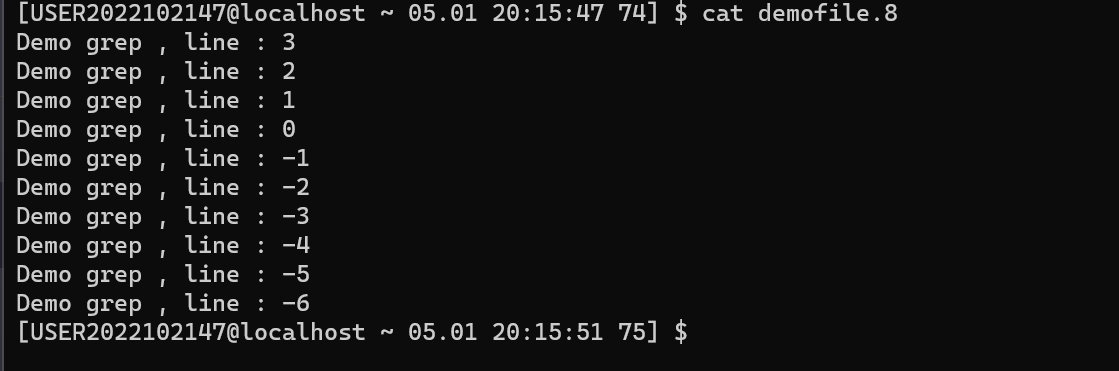
<https://murphypei.github.io/blog/2018/04/linux-redirect-pipe>

1. 理解head命令，编写实现head功能的程序

1）、运行以下命令，生成测试用的数据。

(for i in `seq 1 10`; do echo "Demo grep , line : $((4-i)) "; done) > demofile.8

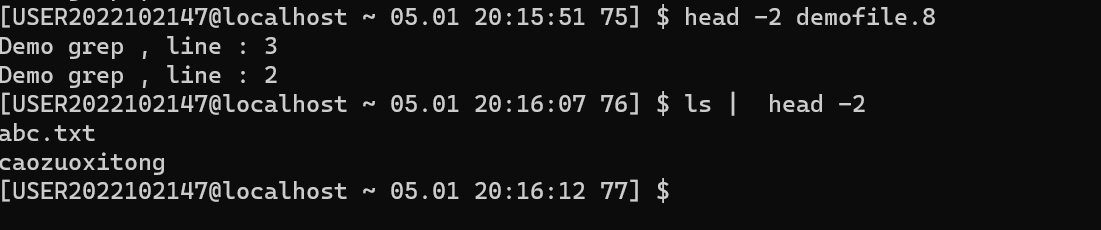
cat demofile.8



2）、运行以下命令，查看命令的运行结果。

head -2 demofile.8

ls | head -2



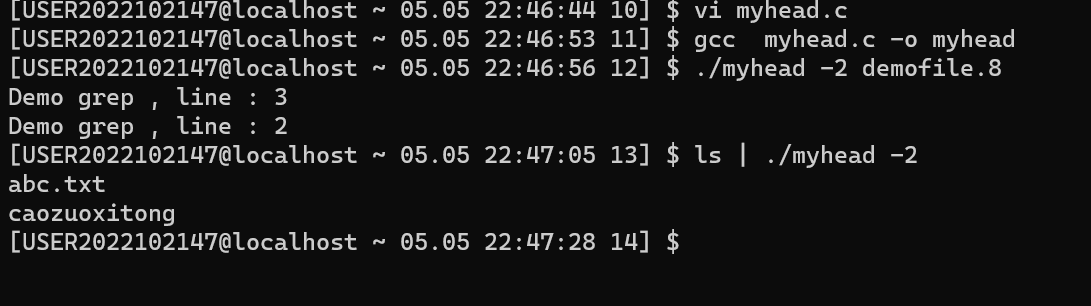
3)、编写myhead.c 实现上述head简单功能。

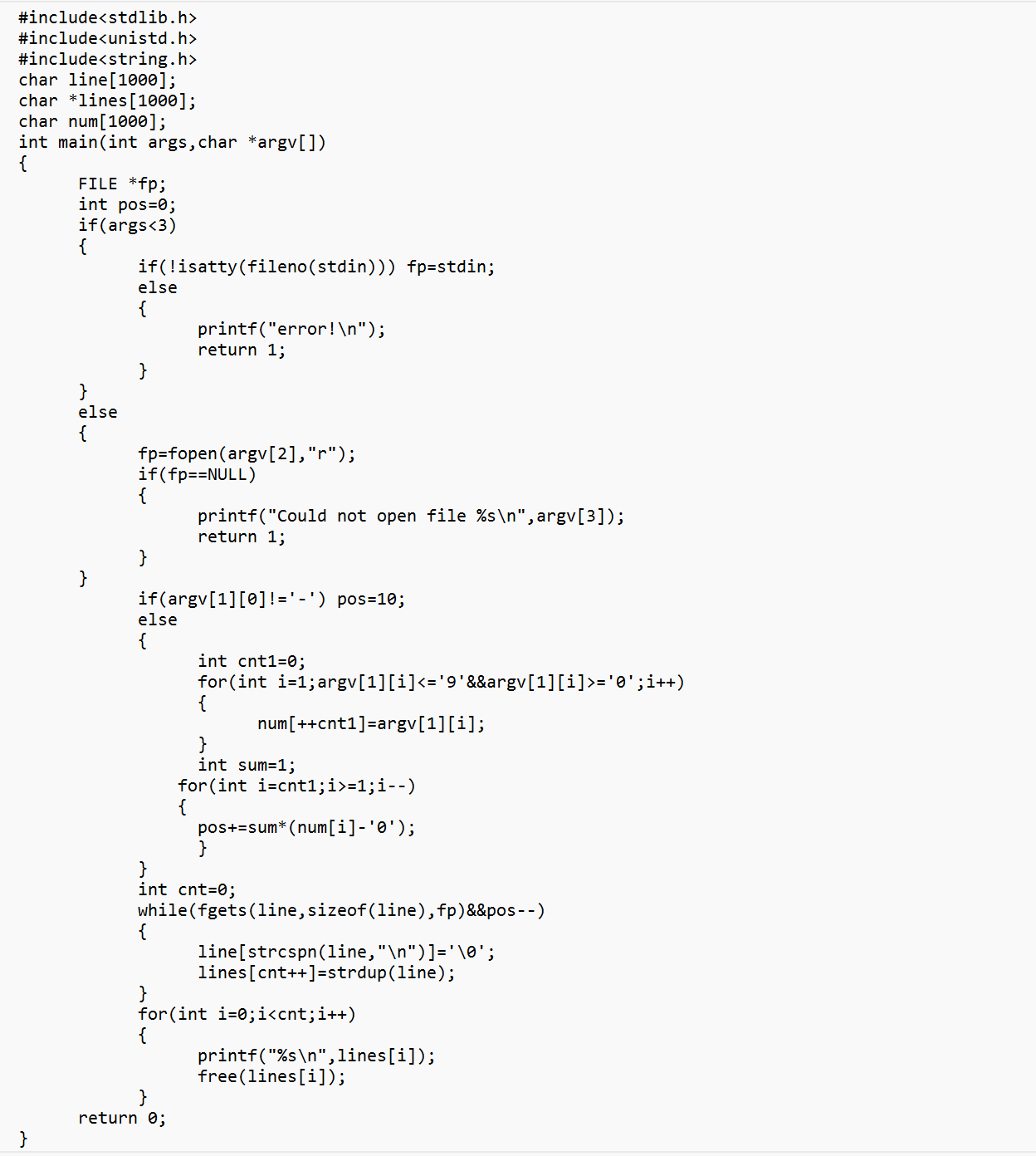
以下命令的结果，应该与2）中的输出结果一样。

gcc myhead.c -o myhead

./myhead -2 demofile.9

ls | ./myhead -2





head命令：

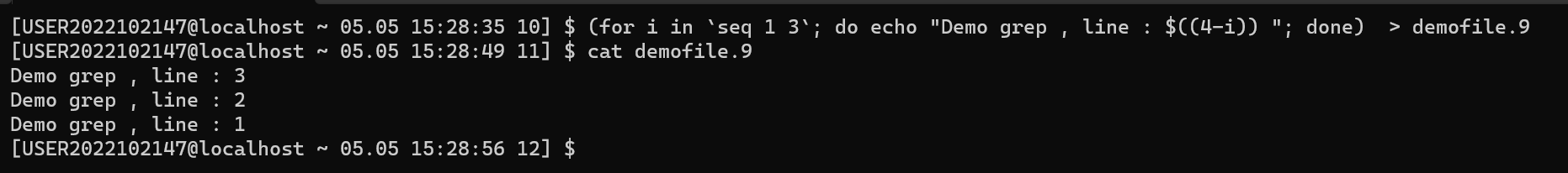
<https://www.cnblogs.com/peida/archive/2012/11/06/2756278.html>

1. 理解sort，编写实现sort功能程序。

1）、运行以下命令，生成测试用的数据。

(for i in `seq 1 3`; do echo "Demo grep , line : $((4-i)) "; done) > demofile.9

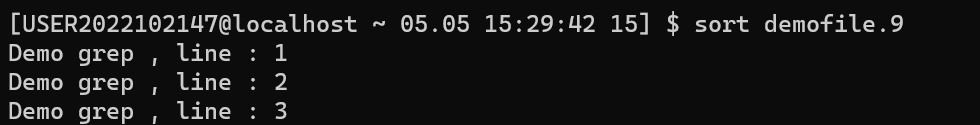
cat demofile.9

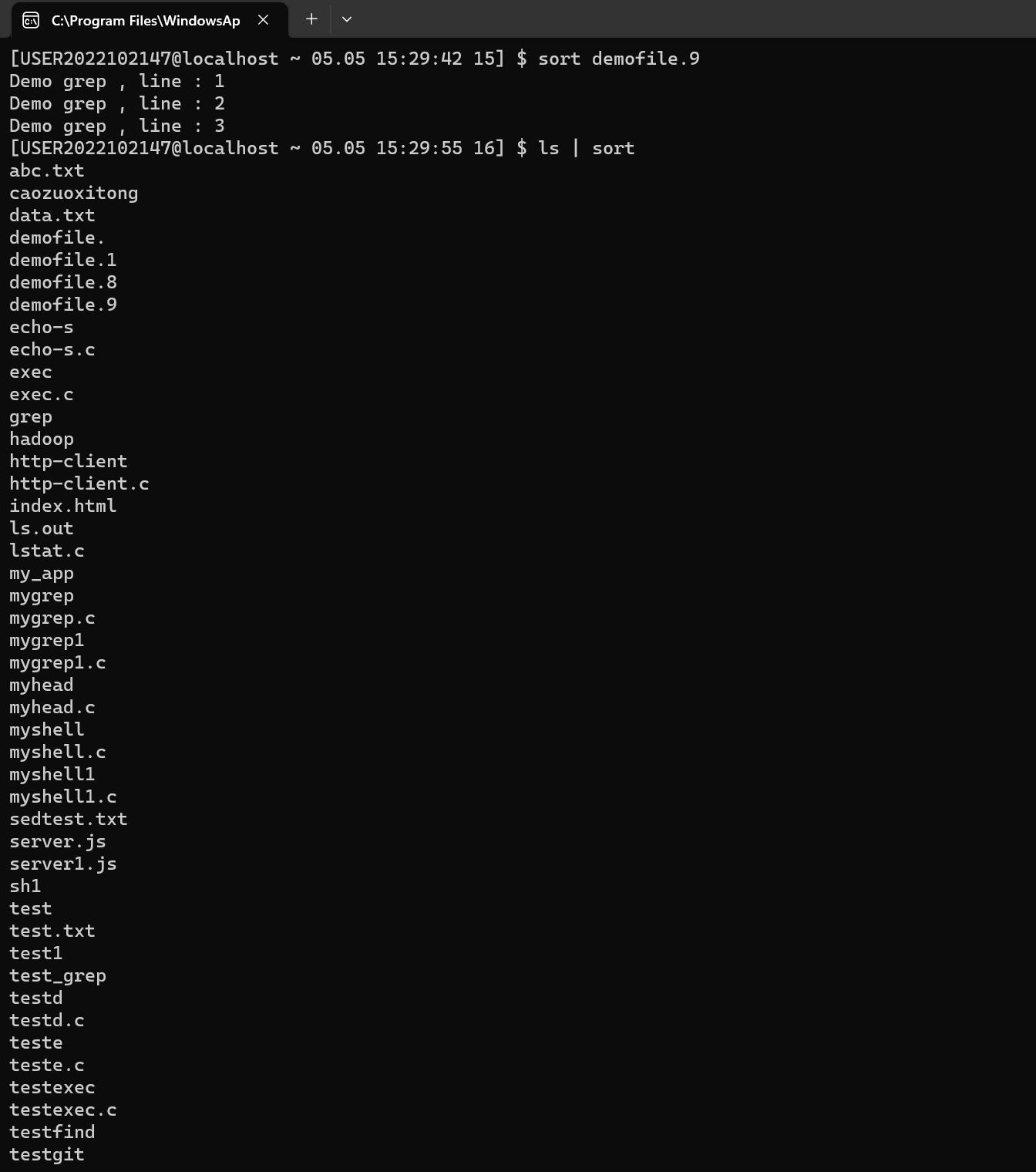


2）、运行以下命令，查看命令的运行结果。

sort demofile.9

ls | sort





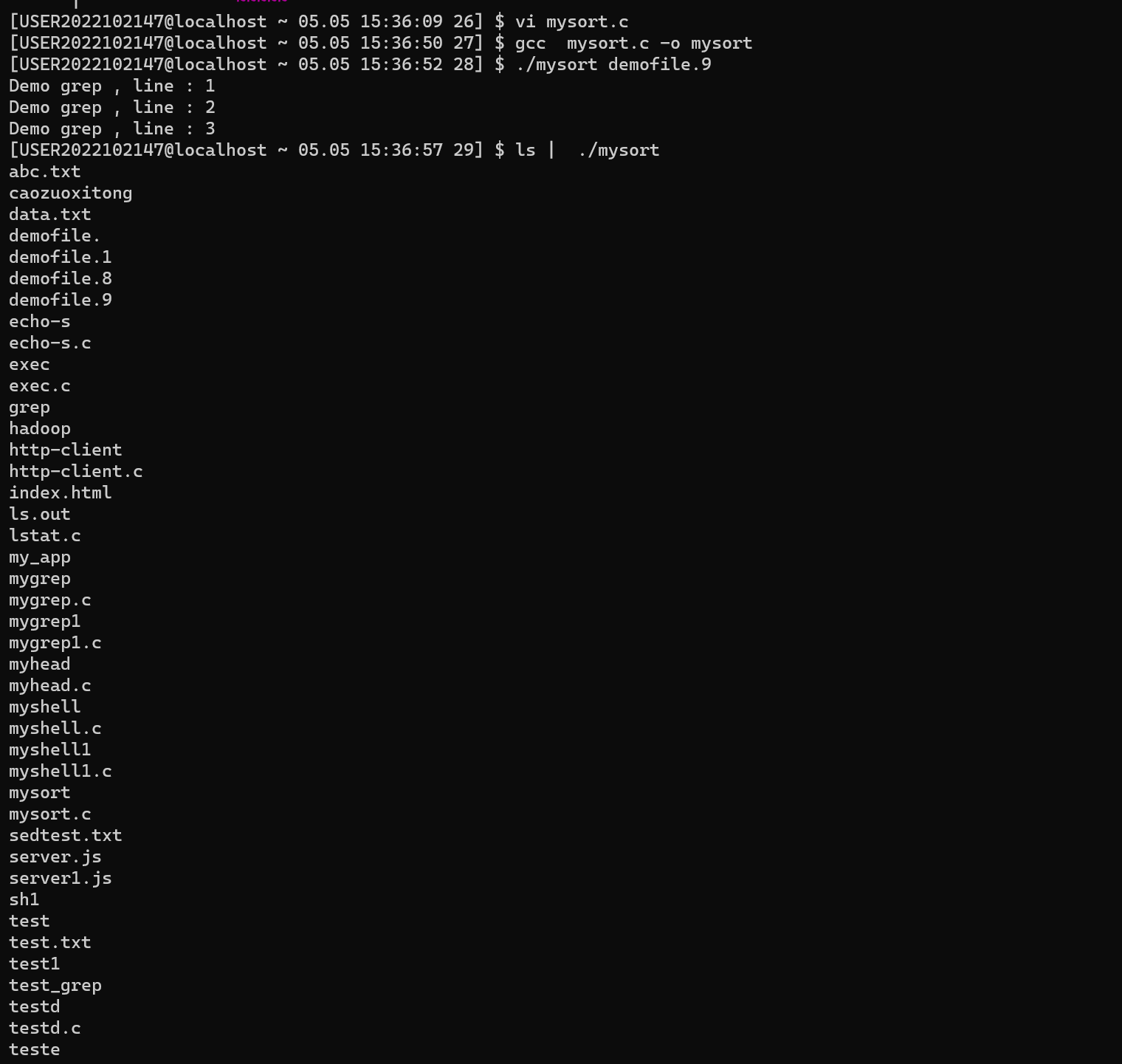
2)、编写mysort.c 实现上述简单的文本文件的排序功能。

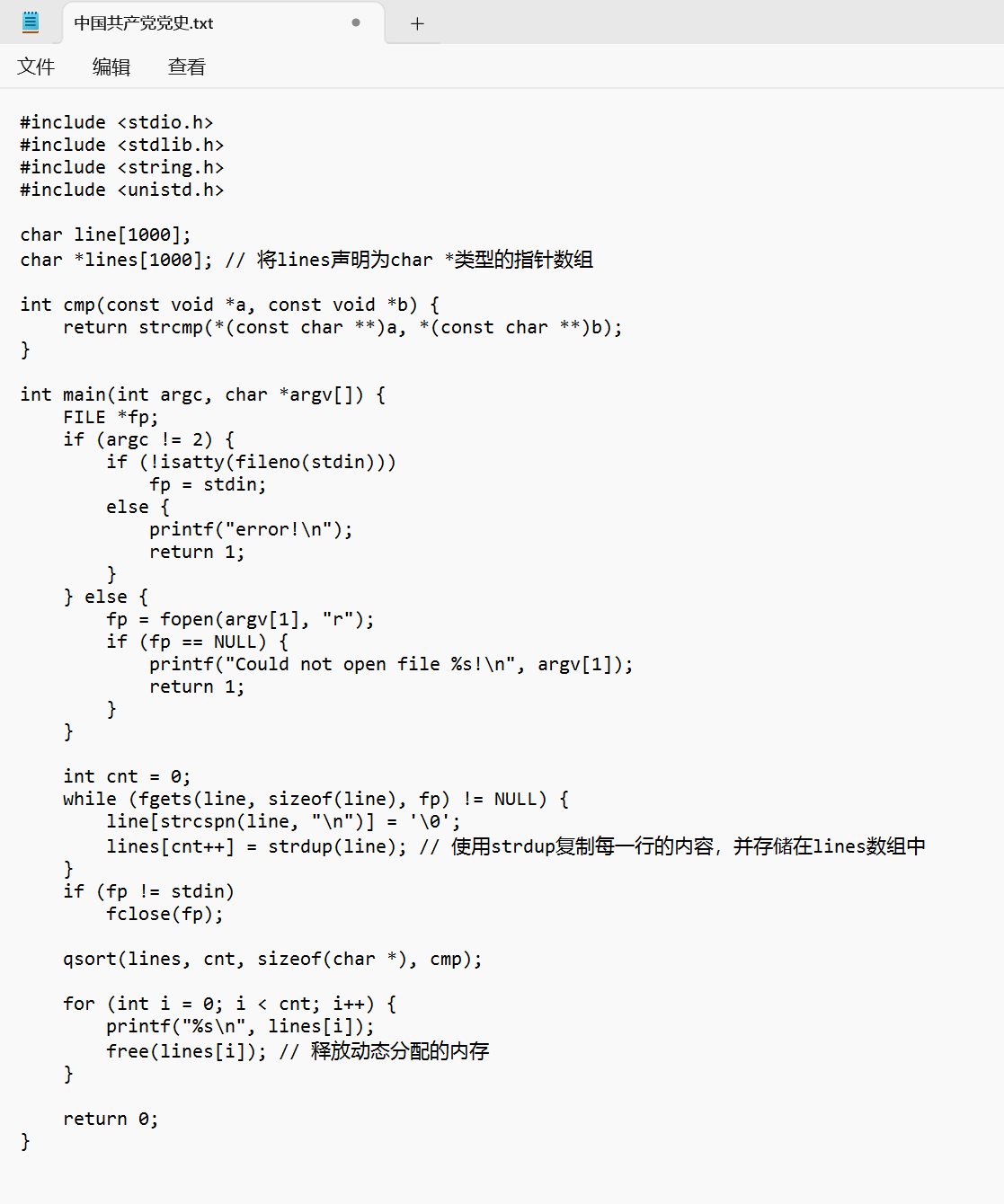
以下命令的结果，应该与1）中的输出结果一样。

gcc mysort.c -o mysort

./mysort demofile.9

ls | ./mysort





排序算法系统调用，参考：

<https://www.cnblogs.com/sggggr/p/16992007.html>

sort命令：

<https://www.runoob.com/linux/linux-comm-sort.html>