跟我一起学 OpenFOAM(2206)

作者: 汪洋

2022 年 07 月 01 日

目录

Chapter 1

课程概述

使用计算流体力学 CFD 理论，结合现代计算机硬件以及编程语言 `C++` ， `Fortran` ，对普遍存在的流动、传热等自然现象进行数值模拟，已成显学。因此许多同学也想较好的掌握 CFD 理论，更好的服务科研生产工作。但是由于 CFD 理论难度较大，使用商业软件很难理解其中真谛，并且商业软件版权费用昂贵一般个人用户难以承受，直到 OpenFOAM 软件出现。OpenFOAM 是上世纪英国帝国理工学院在 Linux 平台使用现代 `C++` 语言开发，并于新千年开源，时至今日已衍生出诸多版本。因此种种，使用者只有具备良好的理论功底，了解 Linux 操作系统，掌握基础 `C++` 语法，才有可能通过 OpenFOAM 解决实际问题。通过自学，学习曲线较为陡峭，时间成本较大，容易导致从入门到放弃。课程目的就是希望通过十周时间解决 OpenFOAM 入门难的问题

1.1 关于本课程的思考

1. 课程拟通过十周时间，通过 OpenFOAM 开源软件来讲解不可压缩 CFD 相关理论知识，希望同学们具备今后在各自相关领域初步科研工作能力。
2. 课程分为五个模块。其中第一个模块 Linux 与后续四个 OpenFOAM 模块表面看去，并不那么紧密，然而这是学好、用好 OpenFOAM 工具至为关键的先修内容。
3. 学习到底是宽度搜索还是深度搜索更为合适。个人观点，在入门阶段，应该以宽度为准。在选题结束后，就应该义无反顾，直达彼岸，深度搜索更合适。

Chapter 2

命令行基本命令

2.1 一个人

```
Linus Torvalds
Richard Stallman
Dennis Ritchie
```

2.2 千里之行始于足下

看看 OpenFOAM 目录下，安装文件脚本 `Allwmake`

```
#!/bin/sh
# Run from OPENFOAM top-level directory only
cd "${0%/*}" || exit
wmake -check-dir "$WM_PROJECT_DIR" 2>/dev/null || {
    echo "Error (${0##*/}) : not located in \$WM_PROJECT_DIR"
    echo "    Check your OpenFOAM environment and installation"
    exit 1
}
if [ -f "$WM_PROJECT_DIR"/wmake/scripts/AllwmakeParseArguments ]
then  . "$WM_PROJECT_DIR"/wmake/scripts/AllwmakeParseArguments || \
    echo "Argument parse error"
else
    echo "Error (${0##*/}) : WM_PROJECT_DIR appears to be incorrect"
    echo "    Check your OpenFOAM environment and installation"
    exit 1
fi

#------------------------------------------------------------------------------
# Preamble. Report tools or at least the mpirun location
if [ -f "$WM_PROJECT_DIR"/wmake/scripts/list_tools ]
```

(下页继续)

(续上页)

```
then sh "$WM_PROJECT_DIR"/wmake/scripts/list_tools || true
else
    echo "mpirun=$(command -v mpirun || true)"
fi
echo
# Report compiler information. First non-blank line from --version output
compiler="$(wmake -show-path-cxx 2>/dev/null || true)"
if [ -x "$compiler" ]
then
    echo "compiler=$compiler"
    "$compiler" --version 2>/dev/null | sed -e '/^$/d;q'
else
    echo "compiler=unknown"
fi

echo
echo ========================================
date "+%Y-%m-%d %H:%M:%S %z" 2>/dev/null || echo "date is unknown"
echo "Starting compile ${WM_PROJECT_DIR##*/} ${0##*/}"
echo "  $WM_COMPILER ${WM_COMPILER_TYPE:-system} compiler"
echo "  ${WM_OPTIONS}, with ${WM_MPLIB} ${FOAM_MPI}"
echo ========================================
echo

# Compile tools for wmake
"${WM_DIR:-wmake}"/src/Allmake

# Compile ThirdParty libraries and applications
if [ -d "$WM_THIRD_PARTY_DIR" ]
then
    if [ -e "$WM_THIRD_PARTY_DIR"/Allwmake.override ]
    then
        if [ -x "$WM_THIRD_PARTY_DIR"/Allwmake.override ]
        then    "$WM_THIRD_PARTY_DIR"/Allwmake.override
        fi
    elif [ -x "$WM_THIRD_PARTY_DIR"/Allwmake ]
    then      "$WM_THIRD_PARTY_DIR"/Allwmake
    else
        echo "Skip ThirdParty (no Allwmake* files)"
    fi
else
    echo "Skip ThirdParty (no directory)"
fi

# OpenFOAM libraries
src/Allwmake $targetType $*
```

(下页继续)

(续上页)

```
# OpenFOAM applications
applications/Allwmake $targetType $*


# Additional components/modules
case "$FOAM_MODULE_PREFIX" in
(false | none)
    echo ========================================
    echo "OpenFOAM modules disabled (prefix=${FOAM_MODULE_PREFIX})"
    echo
    ;;
(*)
    # Use wmake -all instead of Allwmake to allow for overrides
    ( cd "$WM_PROJECT_DIR/modules" 2>/dev/null && wmake -all )
esac

# Count files in given directory. Ignore "Test-*" binaries.
_foamCountDirEntries()
{
    (cd "$1" 2>/dev/null && find . -mindepth 1 -maxdepth 1 -type f 2>/dev/null) |\
        sed -e '\@/Test-@d' | wc -l
}

# Some summary information
echo
date "+%Y-%m-%d %H:%M:%S %z" 2>/dev/null || echo "date is unknown"
echo ========================================
echo "  ${WM_PROJECT_DIR##*/}"
echo "  $WM_COMPILER ${WM_COMPILER_TYPE:-system} compiler"
echo "  ${WM_OPTIONS}, with ${WM_MPLIB} ${FOAM_MPI}"
echo
echo "  api   = $(etc/openfoam -show-api 2>/dev/null)"
echo "  patch = $(etc/openfoam -show-patch 2>/dev/null)"
echo "  bin   = $(_foamCountDirEntries "$FOAM_APPBIN") entries"
echo "  lib   = $(_foamCountDirEntries "$FOAM_LIBBIN") entries"
echo
echo ========================================

#------------------------------------------------------------------------------
```

> **警告：** 这是 OpenFOAM 目录下的文件，Allwmake ，无论如何我们也应该能大致读懂

2.3 Linux 目录

文件夹	内容
/	根目录，所有的起点
/bin	系统权限程序
/boot	包含了 linux 内核、启动镜像和 boot loader
/dev	所有设备文件都在这里
/etc	所有系统范围内的配置文件
/home	家目录
/lib	系统核心库文件
/lost+found	一般为空，现在 ext4 的 linux 系统都包含
/media	一般对应 USB 和 CD-ROM
/mnt	老版本加载外设，手动加载
/opt	option 的含义，部分软件会安装在此
/proc	Linux 内核维护的虚拟文件系统
/root	这是根目录用户的家目录
/sbin	superuser 的可执行文件放在此处
/tmp	临时文件
/usr	可能是 linux 系统中最大的文件夹，所有很用户有关的都保存在此处
/usr/bin	用户可执行文件
/usr/lib	用户的库文件
/usr/local	通常用户编译的程序此
/usr/sbin	包含更多的系统级程序
/usr/share	包含默认的配置文件，图标，桌面和声音文件等
/usr/share/doc	一些软件安装之后有文档，安装在此
/var	包含一些静态不变内容
/var/log	包含一些系统日志在内内容

2.4 echo 命令

- Print given arguments

```
$echo "OpenFOAM"
OpenFOAM
$echo hello\ wangyang
hello wangyang
$echo "hello wangyang"
hello wangyang
$echo $PATH
/home/cfd-wy/Software/nvim-linux64/bin:
/home/cfd-wy/.local/lib:
/home/cfd-wy/bin:/home/cfd-wy/.local/lib:
```

(下页继续)

(续上页)

```
/usr/local/bin:
/home/cfd-wy/anaconda3/envs/sphinx/bin:
/home/cfd-wy/Software/nvim-linux64/bin:
/home/cfd-wy/anaconda3/condabin:
/home/cfd-wy/.local/lib:
/home/cfd-wy/bin:
/usr/local/bin:
/usr/local/sbin:
/usr/local/bin:
/usr/sbin:
/usr/bin:
/sbin:
/bin:
/usr/games:
/usr/local/games:
/snap/bin
$which echo
/usr/bin/echo
$pwd
/home/cfd-wy/Desktop/2022/may/myst_learning
$cd /home/cfd-wy/
$whoami
$echo $((3 + 2))
$echo $(($(5**2) + 3))
```

2.4.1 括号扩展

```
$echo OpenFOAM2206_case_{0..20}
$echo a{A{1,2},B{3,4}}b
$mkdir {2022..2030}-{01..12}
```

2.5 date 命令

- Set or print the system date

```
$date
```

2.6 cd 命令

```
$cd ..
$cd ~
$cd ~/Desktop
$cd -
```

2.7 ls 命令

显示制定参数目录下的文件名，有许多参数可供选择

```
$ls --help
$ls -l
total 48K
drwxrwxr-x 4 cfd-wy cfd-wy 4.0K 6月  10 09:29 .
drwxrwxr-x 6 cfd-wy cfd-wy 4.0K 5月  25 15:56 ..
-rw-rw-r-- 1 cfd-wy cfd-wy 7.4K 6月  10 09:29 1.txt
drwxrwxr-x 4 cfd-wy cfd-wy 4.0K 5月   6 21:03 build
-rw-rw-r-- 1 cfd-wy cfd-wy  804 5月   6 21:02 make.bat
-rw-rw-r-- 1 cfd-wy cfd-wy  638 5月   6 21:02 Makefile
drwxrwxr-x 4 cfd-wy cfd-wy 4.0K 5月   9 07:41 source
-rw-rw-r-- 1 cfd-wy cfd-wy  13K 5月   9 07:47 wealth_dynamics_md.md
$ls -h
$ls -a
```

```
参数是flages或者options, Usage: ls [OPTION] ... [FILE] ...
```

备注:

- 注意最前面一列，d 表示目录。
- 后面每 3 个字母一组。3 个字母分别表示读、写和可执行。- 表示没有。
- 3 组从左至右，分别为用户、分组和其他对该文件有读、写和可执行权限的用户

2.8 touch 命令

新建文件，有一些技巧

```
$touch OpenFOAM.txt
$touch OpenFOAM{1..100}.txt
```

2.9 mv 命令

移动命令，move

```
$mv wangyang.md daijuan.md
```

2.10 cp 命令

拷贝命令，copy

```
$cp -r wangyang.md daijuan.md
```

2.11 rm 命令

remove，删除

```
$rm -r OpenFOAM
```

警告： 特别小心，在删除之前想明白，如果不确定可以使用 `-i` flag，表示交互删除的意思

2.12 man 命令

manual pages

```
$man echo
$man rm
$man 5 proc
```

备注： 大部分命令都可以默认用 man 获得帮助信息，但是偶尔有些命令，需要第三条 `man 5 proc`

2.13 clear 命令

清除屏幕

```
$clear
```

备注： 快捷键是 `Ctrl+l` 。

2.14 cat 命令

清除屏幕

```
$ls > OpenFOAM.txt
$cat OpenFOAM.txt
```

备注: 这里出现了一个新的符号，> 。它的作用是什么呢？有没有类似的符号？

2.15 重定向和管道

Linux 哲学就是一切都是文件，所以 ls 命令执行的的结果给到一个特殊文件，标准输出

2.15.1 标准输入、输出和错误

```
0. stdin
1. stdout
2. stderr
```

```
$ls /usr/bin > ls-output.txt
$ls /bin/usr > ls-output.txt
$ls /usr/bin > ls-output.txt
$ls /usr/bin >> ls-output.txt # append
$ls /bin/usr 2> ls-error.txt
$cat ls-error.txt
```

警告:

- 理解文件描述符，file descriptor ， stdin 0, stdout 1 stderr 2 。
- 重定向符号 > >> 使用之后，并没有更好的符号表示标准错误。
- 重点体会 ls /bin/usr 2> ls-error.txt

2.15.2 如何将标准输出和标准错误指向同一个文件

两种方式可以实现这个目标

```
$ls -l /bin/usr > ls-output.txt 2>&1
$ls -l /bin/usr &> ls-output.txt
```

2.15.3 简单理解 /dev/null

```
$ls -l /bin/usr 2> /dev/null
```

警告:

- 这里简单理解 /dev/null 就是垃圾桶，什么反映都不会有的。具体可以参考

  维基

2.16 管道

从标准输入读取数据然后将数据传递给标准输出就是管道，管道符号为 |

```
command1 | command2
```

```
$ls -l /usr/bin | less
```

```
重定向和管道的不同点，最简单的看法，管道是不同命令间的，重定向是针对文件的。
```

警告:

- 千万千万不要将两个不同命令进行重定向
- 千万千万不要将两个不同命令进行重定向
- 千万千万不要将两个不同命令进行重定向

```
$ls /bin /usr/bin | sort | uniq | less
$ls /bin /usr/bin | sort | uniq -d | less
$wc ls-output.txt
2  18 112 ls-output.txt
$wc -l ls-output.txt
$ls /bin /usr/bin | sort | uniq | less | grep zip
$ ls /usr/bin | tee ls.txt | grep zip
```

备注:

1. sort 命令，排序
2. uniq 命令，排除重复
3. less 命令，交互阅读
4. grep 命令，可以发现文件名具备的内容。

5. tee 命令，非常好的中间命令，可以讲标准输出转换为标准输入

2.17 环境变量

```
$1 $(which $(echo $PAGER))
```

2.17.1 常用环境变量

变量名	内容
DISPLAY	显示
EDITOR	
SHELL	
HOME	
LANG	
OLDPWD	
PAGER	
PATH	
PS1	
PWD	
TERM	
TZ	
USER	

2.17.2 环境如何建立

login shell sessions

文件	内容
/etc/profile	全局
~/.bash_profile	用户定义的，可以覆盖上面的系统设置
~/.bash_login	如果第二个文件没有找到，系统试图找它
~/.profile	如果第二、三个都没有，系统找他

non login shell sessions

文件	内容
/etc/bash.bashrc	全局
~/.bashrc	用户定义的，可以覆盖上面的系统设置

2.18 包管理

Linux 主流分支和发行版本众多，所以安装方式也不禁相同。

2.18.1 包管理主流分支

主流分支	发现版本
Debian Style(.deb)	Debian, Ubuntu, Linux Mint, Rasphbian
Red Hat Sytle(.rpm)	Fedora, CentOS, Red Hat, OpenSUSE

包管理工具

发行版本	低阶工具	高阶工具
Debian style	dpkg	apt, apt-get
Fedora, CentOS	rpm	yum, dnf
Red Hat		

如何在库中找包

分支	命令
Debian	apt update; apt-get update; apt search xxxxx
Red Hat	yum search xxxxx

从库中安装文件

```
$apt update
$apt install package_name
$apt-get update
$apt-get install package_name
```

Low-level 安装

```
$dpkg -i package_file
$rmp -i package_file
```

2.19 查找文件

Linux 下面，很多时候查找内容非常重要，如果掌握一些相关操作帮助会极大

2.19.1 常用命令

```
* locate: 通过名字查找文件
* find: 递归搜索文件，通常和其他命令组合
* xargs: 从标准输入构建和执行命令
* touch: 改变文件时间
* stat: 显示文件或文件系统状态
```

```
$locate bin/zip
$locate zip | grep bin

$find ~ -type d | wc -l
$find ~ -type f | wc -l
$find ~ -type f -iname "*.jpg" -size +1M | wc -l
$find ~ -type f -name "*.bak" -delete
$find ~ -type f -name "*.log" -print
```

备注:

1. locate 有些时候不能用，而这个更新程序 `locatedb` 是系统自动定时运行的
2. find 常用的类型，`d`，`f`，`l` 分别表示文件夹，文件和链接
3. find 可以与逻辑运算符一起使用，`-and`，`-or` `-not`
4. find 最后接 `-delete` 要特别小心

用户定义行为

用法

```
-exec command {} ;
xargs command
```

```
$find ~ -type f -name "foo*" -exec ls -l '{}' ';'
$find ~ -type f -name "foo*" | xargs ls -l
```

备注:

1. -delete 就等于 -exec rm '{}' ';'
2. 两句话是等效的
3. xargs 会有使用长度限制

2.20 打包和备份

2.20.1 常用命令

```
* gzip: 压缩或者扩展文件 ``gunzip``
* tar: 打包工具
* zip: 打包或者压缩文件 ``unzip``
* rsync: 远程文件和文件夹同步
```

2.21 正则表达式

正则表达式，是非常强悍的工具，通过匹配达到检索目的。regular expression 掌握后对今后工作帮助极大。

本文档基于 POSIX 标准。不同系统、编程语言会有 轻微区别。

2.21.1 grep

grep: global regular expression print. 该工具通过正则表达式来匹配搜索文件。通过标准输出输出所有符合匹配的内容。

```
grep [options] regex [file...]
```

选项	描述
-i	忽略大小写
-v	匹配不包含的内容
-c	打印匹配的数字
-l	打印匹配到的文件名字
-L	打印没有匹配到的文件名字
-n	前缀匹配
-h	对于多文件搜索，压制输出文件名

```
$ls /bin > dirlist-bin.txt
$ls /usr/bin > dirlist-usr-bin.txt
$ls /sbin > dirlist-sbin.txt
$ls /usr/sbin > dirlist-usr-sbin.txt
$ls dirlist*.txt
$grep bzip dirlist*.txt
$grep -l bzip dirlist*.txt
$grep -L bzip dirlist*.txt
```

元字符

metacharacaters

```
^ $ . [ ] { } - ? * + ( ) | \
```

```
$grep -h '.zip' dirlist*.txt
$grep -h '^zip' dirlist*.txt
$grep -h 'zip$' dirlist*.txt
$grep -h '^zip$' dirlist*.txt
$grep -i '^..j.r$' /usr/share/dict/words
$grep -h '[bg]zip' dirlist*.txt
$grep -h '^[bg]zip' dirlist*.txt
$grep -h '^[A-Z]zip' dirlist*.txt
$grep -h '^[bg]zip' dirlist*.txt
$grep -h '^[bg]zip' dirlist*.txt
$grep -h '^[A-Z0-9a-z]' dirlist*.txt
$grep -h '[-AZ]' dirlist*.txt
```

关于正则表达式，这里面有 basic regular expression (BRE) 和 extended regular expression (ERE) 之分。

```
$grep -Eh "^(gz|bz|zip)" disrlist*
$grep -Eh "^gz|bz|zip" dirlist*
```

? 匹配一个元素 0 或者 1 次

```
电话号码
(nnn)-nnnnnnnn
nnn-nnnnnnnn
```

```
^\(?[0-9][0-9])[0-9]\)?-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]$
$echo "027-87531765" | grep -E '^\(?[0-9][0-9][0-9]\)?-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
↪$'
```

* 匹配一个元素 0 或者多次

```
[[:upper:]][[:upper:][:lower:] ]*\.
```

```
echo "This Works." | grep -E '[[:upper:]][[:upper:][:lower:] ]*\.'
```

+ 匹配一个元素 0 或者多次

```
^([[:alpha:]]+ ?)+$
```

```
$echo "This that" | grep -E '^([[:alpha:]]+ ?)+$'
```

{} 匹配一个元素给定次数

描述	含义
{n}	匹配前述元素 n 次
{n, m}	匹配前述元素至少 n 次, 不超过 m 次
{n, }	
{, m}	

```
^\(?[0-9][0-9])[0-9]\)?-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]$
'^\(?[0-9]{3}\)?-[0-9]{8}$'
```

See Shotts [1] for an introduction to non-standard and a non-standard analysis if fun [1] .

Chapter 3

Shell 脚本

最简单的理解，shell 脚本就是由一系列命令组成。shell 本身是系统命令的接口又是脚本语言的解释器。interpreter 和 compiler 的区别。

```
1. 写脚本，使用合理的工具，文本编辑器：vscode，vim，kate 等。
2. 脚本权限，改变权限，通常使用 chmod +x xxx.sh
3. 放对路径，把脚本放在shell能找到的地方
```

3.1 引言关于编程

1. 通过观察，解释型语言和编译型语言异同?[1]
2. 这些异同会对你今后工作生活带来那些影响?[2]
3. 思考一个问题，编程语言最基本的要素是什么?[3]

3.2 脚本文件格式

```
#!/bin/bash

# This is our first script

echo 'Hello OpenFOAM'
```

```
$ls -l Hello_OpenFOAM
$chmod 755 Hello_OpenFOAM
$./Hello_OpenFOAM
$mkdir ~/wangyang
```

(下页继续)

[1] 编译型语言速度快，性能好，难写。
[2] 个人感悟是，针对不同问题使用不同语言。
[3] 三个基本内容，顺序、分支、循环。

(续上页)

```
$export PATH=~/wangyang:"$PATH"
$. ~/.bashrc # . = source
$Hello_OpenFOAM
```

备注:

1. 这里第一行，起始的两个字符组合在一起名字叫 shebang. 它的作用是告诉系统后面所跟路径的解释器用来执行下面的语句。
2. 755 怎么来的，read=1, write=2, exectute=4. 三个数字的组合。任何脚本要可执行一定要可读。
3. 倒数第二句话，这样做可以把 wanngyang 添加的路径中去，shell 自然就可找到 Hello_OpenFOAM 脚本。
4. 最好讲自己编译或者脚本放在，/usr/local 下面。也可以放在 ~/.local 目录下面。

写个报告生成器

3.3 第一步，最小文档

```
#!/bin/bash

# Program to output a system information page

echo "<html>
        <head>
                <title>Page Title</title>
        </head>
        <body>
                Page body
        </body>
</html>"
```

3.4 第二步，添加小内容

```
#!/bin/bash

# Program to output a system information page

echo "<html>
        <head>
                <title>System Information Report</title>
        </head>
```

(下页继续)

(续上页)

```
        <body>
                <h1>System Information Report</h1>
        </body>
</html>"
```

```
#!/bin/bash

# Program to output a system information page

title="System Information Report"
echo "<html>
        <head>
                <title>$title</title>
        </head>
        <body>
                <h1>$title</h1>
        </body>
</html>"
```

3.5 变量和常量

1. 变量名字有字母、下划线和数字组成
2. 变量必须以字母或者下划线开头
3. 空格和标点符合是非法的
4. 通常约定，常量用全大写，变量用小写

```
declare -r TITLE="Page TITLE"
```

3.5.1 变量赋值和常量

```
variable=value
```

1. 与python等脚本语言类似，所以不用制定类型
2. 语法规则等号两边不能有空格
3. 当字符串用比较多

```
$a=z
$b="a string"
$c="a string and $b"
$d="$(ls -l /usr/bin)"
$e=$((5 * 7))
$f="\t\ta string\n"
```

备注:

1. 用双引号包裹变量和命令会非常有效

3.6 从上到下设计

程序设计对于大多数而言，都非常麻烦。当程序变大之后，难以维护。通常，我们希望把大的问题分解为小的问题，也就是最常见的思想，分而治之，`divided and conquer` 。

```
把大象关进冰箱里面，拢共分几步？
```

这就是经典的分而治之思想

3.7 函数

shell function 可以理解为，脚本的脚本或者更小的脚本

```
system_info()
{
        echo "<h2>System release info</h2>"
        echo "<p>Function not yet implemendt</p>"
}
```

3.8 条件，分支

程序一般都是顺序执行，但是我们可以通过一些语句改变程序执行顺序

3.8.1 if

语法规则

```
if commands; then
        commands
[elif commands; then
        commands]
[else
        commands]
fi
```

3.8.2 Exit status

退出状态，在 linux 下面，一般为 0-255 的整数，通常 0 表示正常，非 0 表示错误。可以使用 $? 来检查上一次操作是否正确

3.8.3 test

```
# First form

test expression

# Second form

[ expression ]
```

```
if [ -f .bashrc ]; then
        echo "You have a .bashrc."
else
        echo "You haven't .bashrc."
fi
```

表达式	描述
-d file	如果是文件夹
-e file	如果存在该文件
-f file	文件存在且是正常的
-L file	是链接文件
-r file	文件可读
-W file	文件可写
-x file	文件可执行
file1 -nt file2	文件 1 是否新于文件 2
file1 -ot file2	文件 1 是否旧于文件 2
-z string	字符串是否为空
-n string	字符串是否不为空
string1 = string2	两个字符串相等
string1 != string2	两个字符串不相等

3.8.4 exit

```
exit 0
exit 1
```

3.8.5 测试用户权限

```
if [ "$(id -u)" = 0 ]; then
        echo "not superuser"
        exit 1
fi

if [ "$(id -u)" != 0 ]; then
        echo "你应该拥有超级用话权限" >&2
        exit 1
fi
```

3.8.6 处理问题

```
trouble.sh
```

```
$./trouble.sh
./trouble.sh: line 5: [: =: unary operator expected
Number does not equal 1
```

3.9 键盘输入和基本运算

3.9.1 read

```
read_demo.sh
-p: 可以让你制定输入参数赋值给谁
-t: 可以在制定时间内输入
-s:
arithmetic.sh
arithmetic.c
```

3.9.2 case

```
case.sh
case.c
case word in
    patterns ) commands ;;
esac
```

3.9.3 loops

```
while
until
for
```

while

```
loops.sh
```

until

```
#!/usr/bin/bash

number=0
until [ "$number" -ge 10 ]; do
    echo "Number = $number"
    number=$((number + 1))
done

$enhance_loops.sh
```

3.10 位置参数

```
$ some_program
$ some_program_c
$ dollar_sharp.sh
$ dollar_sharp.c
参考 ``some_program`` 和 ``some_program_c``
$0-$9，相当与C语言的 ``argv[]`` ，字符串数组. $#相当与 ``argc - 1`` ，参数个数
$?，以前将过，表示上一次命令或者脚本执行的正确与否
```

```
for variable in words; do
    commands;
done

for.sh
for.c
for_bashrc.sh
for_bash.c
for_dollar_at.sh
for_dollar_at.c

$bash for_bashrc.sh
$bash for_dollar_at.sh *
```

Chapter 4

VIM 使用入门

4.1 引言

Gilles Castel 他的故事

VIM 哲学就是在不同的模式干不同的事情。

1. normal 模式，普通模式
2. insert 模式，插入模式
3. visual 模式，可视模式

vim 需要一定量针对性的训练，才能提高效率，切记针对性训练。

4.2 vim 基本操作

4.2.1 保存退出

在普通模式下，敲击键盘

```
:w，写入，也即保存
:q，退出，如果没有保存，会有提示并不能退出
:i，从普通模式进入插入模式，可以输入内容
ESC，从其他模式切换进普通模式

:a，从光标后添加
:A，从行尾添加
:o，从下一行开始输入
:O，从上一行开始输入
```

4.2.2 VIM 配置文件

```
syntax on
set nu
set cursorline

map s <nop>
map S :w<CR>
map Q :wq<CR>
map R :source $MYVIMRC<CR>
```

4.2.3 理解 VIM 的方式

```
<operation> <motion>
```

例子: OpenFOAM wangyang

d 是一个删除操作，后面要接动作

```
x
d->
d<-
dd
```

p 是一个粘贴操作

y 是一个复制操作，yank

```
y->
y3->
y3w
```

c 是一个改变操作，可以理解为 d 的动作之后再 i

```
c->
c3->
c3w
```

两个重要介词, in and around

```
#include <iostream>
#include <vector>
#include <string>

int main(int argc, char* argv[]) {
    std::vector<std::string> wangyang = { "Hello", "OpenFOAM" };
```

(下页继续)

(续上页)

```
    for (const auto &e : wangyang) {
        std::cout << e << " ";
    }

    std::cout << std::endl;

    return 0;
}
```

可以以这段代码, 理解一下动作 in 和 around

```
可以试一下，怎么用 ``ci)``，同理，你可以用其他操作
```

f 动作, motion

vim is a best editor

```
f)
df)
```

搜索

```
/ 用来搜索
```

继续添加 VIM 配置文件

```
noremap K 5k
noremap J 5j

set hlsearch
exec "nohlsearch"
set incsearch
set ignorecase
set smartcase
set scrolloff=10


noremap <LEADER><CR> :nohlsearch<CR>
```

分屏

split 和 vsplit

4.3 配色

默认使用 :color ， 可以尝试一下。我们使用一个插件管理器 vim-plug

```
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

安装完成后，打开你的配置文件 .vimrc , 添加一下语句，要安装一下，再重启 vim，看看效果

```
call plug#begin('~/.vim/plugged')

Plug 'vim-airline/vim-airline'

Plug 'connorholyday/vim-airline'

Plug 'ghifarit53/tokyonight-vim'

call plug#end()

colorsheme snazzy
let g:SnazzyTransparent = 1
```

给出了两个主题 vim-snazzy 和 tokyonight

4.4 advaned contents

4.4.1 可视模式

三种方式， visual , line-visual, block-visual

如何直接加呢？块选择如何操作？例如将 OpenFOAM 的首字母改成小写改如何呢？

```
openFOAM-a.out.py
openFOAM-build.py
openFOAM-Makefile.py
openFOAM-make.sh.py
openFOAM-OpenFOAM.txt.py
openFOAM-README.rst.py
openFOAM-requirements.txt.py
openFOAM-source.py
```

在 C-v 块选择下，使用 “S-i“，然后输入 o 再按 ESC 就行了

如何用 S-v 方法来实现呢?[1]

[1] :normal ^xio<CR>

4.4.2 正常模式

分屏

```
map sh :split<CR>
map sv :vsplit<CR>
```

分屏幕之后，怎么在屏幕间切换呢?

```
map <LEADER>h <C-w>h
map <LEADER>j <C-w>j
map <LEADER>k <C-w>k
map <LEADER>l <C-w>l
```

那窗口大小可否调整呢？答案是可以的。

标签页

```
map tu :tabe<CR>
map tn :+tabnext<CR>
map tp :-tabnext<CR>

set nocompatible
filetype on
filetype indent on
filetype plugin on
filetype plugin indent on

set mouse=a
set encoding=utf-8
set laststatus=2
```

一些比较推荐的插件

1. NerdTree
2. youcompleteme

```
map tt :NERDTreeToggle<CR>
let NERDTreeMapOpenExpl = ""
let NEDTreeMapUpdir
let NEDTreeMapUpdirKeepOpen = "l"
let NEDTreeMapOpenSplit = ""
let NEDTreeOpenVSplit = ""
let NEDTreeMapActivateNode = "i"
let NEDTreeMapOpenInTab = "o"
let NEDTreeMapPreview = ""
```

(下页继续)

(续上页)

```
let NEDTreeMapCloseDir = "n"
let NEDTreeMapChangeRoot = "y"
```

以上配置，是 NERDTree 插件的快捷键方式，仅供参考！还可以参考:!help NERDTree

4.5 写在本章最后

1. VIM 是否是你需要的，没有答案。但是还是希望大家能用起来。
2. 上面推荐的某些插件，可能比较老了，但是教学目的够用了。
3. 接上面的话，代码补全，可能用 lsp - language server protocol 更好
4. neovim 更好。感觉想套娃，但是你可能要会一点点 lua 。关于 lua 是什么，大家有兴趣可以看看，也很轻量。
5. 还是多加练习，平时工作生活中就可以不停摊销掉学习成本。

Chapter 5

LaTeX 入门

5.1 latex 简介

1. Donald Kunth 先生为了自己的书，计算机编程的艺术花了 10 年时间把 TeX 做出来的。
2. Leslie Lamport 为了方便大家使用，封装了 Tex 一下，做出了 LaTeX 。

   二位大佬都是计算机最高奖项图灵奖获得者。

5.2 Tex/LaTeX 能做什么

1. 写大小论文
2. 可以写书
3. 写作业、做笔记

5.3 常见使用工具或平台

5.3.1 texlive 2022

安装参考第一周周末答疑视频。只要注意不要装在 /usr/local/ 下面，尽量安装在 ~/.local/ 文件夹下面。

5.3.2 overleaf

overleaf

5.3.3 texpage

texpage

5.4 文档的组成

1. 标题
2. 前言/摘要
3. 目录
4. 正文 a. 篇、章、节、小节和小段

   文字、公式表图
5. 文献
6. 索引和词汇表

写文章不要在意格式，尽量不要造轮子。

5.5 LaTeX 的基本结构

以 document 环境为界，之前为导言部分 (preamble)。环境内部是正文部分。通常在导言区进行格式设置，正文部分就套用格式。

```
\documentclass{ctexart}
\usepackage[A4paper]{geometry}
\begin{document}
 通过使用\emph{\LaTeX{}}来学习OpenFOAM
\end{document}
```

5.5.1 文档构成

1. 标题, title
2. 摘要, abstract
3. 目录, tableofcontens
4. 章节, chapter, section
5. 附录, appendix
6. 文献, bibliography

7. 索引, printindex

5.6 磁盘文件组成

1. documentclass, .cls
2. usepackage, .sty
3. include, .tex
4. input, .tex

5.7 理解命令和环境

5.7.1 命令

```
\frac{1}{2}
```

5.7.2 环境

```
\begin{env}
...
\end{env}
```

5.7.3 正文文本

直接输入正文文本。用空格分开单词。多个空格等与一个空格。自然分段是空一行。注意转义符号的使用。要用 `\textbackslash` 输出。有很多奇怪字体，可以参考 symbols 库。

5.8 数学模式

5.8.1 行内

```
$a + b = c$
```

5.8.2 显示

上标，下标，^ _
上下画线和花括号，\overline \underline，\overbrace，\underbrace
分式，\frac{}{}
根式，\sqrt[次数]{根号下}
矩阵，amsmath下，matrix，bmatrix，pmatrix

Chapter 6

Makefile 入门

在 Linux 环境下会不会写 makefile 是判断一个人是否具备大型工程能力的标准。

makefile 对于一个工程而言，管理了整个工程的编译规则，编译的先后顺序，外部库的链接管理等。它带来的最大好处就是可以实现 自动化，一旦规则确定，仅仅需要寥寥数行命令就可以完全一切工作，显著提高了生产效率。在很多 IDE 中都会有它的身影，会有不同，但是不会改变本质。

本文基于 kubuntu 20.04 LTS 操作系统，使用 GNU Make 4.2.1 ，会涉及到一些例如 C/C++，fortran，tex 等内容，不涉及 windows 编程环境。

6.1 编译和链接

在 Linux 中，常见的可执行程序是二进制文件。通过 C/C++ 本质上是先将源文件编译成 .o 文件[1] ，然后将多个 .o 文件已经第三方库文件 (.a)[2] 合并生成执行文件。前述动作为编译，后续动作为链接。

6.2 makefile 简介

使用 make 命令时，需要一个 makefile 文件，告诉 make 命令按照什么样的约定执行编译和链接的过程。

6.2.1 例子

```
git clone https://github.com/modern-fortran/tsunami
cd scr/ch02
l
vim Makefile
make
./tsunami > tsunami_v2.txt
```

(下页继续)

[1] object 文件

[2] archive 文件

(续上页)

```
chmod 755 plot_height_multipanel.py
./plot_height_multipanel.py tsunami_v2.txt
```

从这里就可以粗略的感受到，make 的强大

bash 脚本是否也能实现类似功能？答案是可以的

可以现场显示一下，那又何不足呢？

要非常复杂，才能实现多任务，显然没有 make 方便。

也可以在上面的 makefile 文件内添加 python plot_xxx.py tsunami_v2.txt 自动画图。

这个时候可以把 make 当成增强型的脚本。

用 make 管理 latex 文档会方便一些

6.2.2 一个例子

完整的 makefile

```
MAIN = make_latex
TEX_SOURCES = makefile\
      *.tex \
      references.bib \
      mystyle.sty

IMAGES = $(shell find images/* -type f)
DATE = $(shell date +"%s")

.PHONY: clean

all:
  xelatex $(MAIN)
  makeindex $(MAIN)
  bibtex $(MAIN)
  xelatex $(MAIN)
  xelatex $(MAIN)

once:
  xelatex $(MAIN)

final:
  pdftk cover.pdf $(MAIN).pdf cat output final.pdf

zip:
  zip -q $(MAIN)_$(DATE).zip $(TEX_SOURCES) $(IMAGES) cover.pdf

clean:
  ls -I "*.tex" | grep $(MAIN) | xargs rm -r
```

这里并没有涉及到 make 里面较为高级的内容，例如，函数，$^ $@ $< 等自动化变量

6.3 fonts

6.3.1 有衬线 Serif

宋体

6.3.2 无衬线 Sans Serif

黑体

6.3.3 等款字体 Mono

6.3.4 变体

意大利体
斜体
自重
小型大写

6.4 写在本章最后

1. 本章内容是make最简单的部分，非常少，但是是个良好的起步
2. 自动化我们的工作，是追求的目标，永远在路上
3. 不要嫌弃自己工作的不完美，用起来。
4. 如果打开视野，你会看到更多、更好的程序，比如：fluid engin dev
5. 加油加油加油！

Chapter 7

VSCode

7.1 简介

1. Micorsoft出品，2015年。
2. github上星星最多的开源项目。
3. 每个月都跟新，对开源社区的反馈非常及时。
4. 跨操作系统和硬件平台。

下载地址

快捷方式

7.2 基本入门

1. 菜单栏
2. 侧边栏
3. 状态栏

7.3 常用插件

1. 主题和图标主题 ayu
2. C/C++
3. CMAKE
4. Doxygen
5. Docker
6. Git graph
7. Hex

8. Jupyter
9. Live
10. Python
11. Remote-SSH

7.4 整体的应用场景

1. 新建文件夹，gitHello
2. 用 vscode 打开目录
3. 新建文件，gitHello.cpp 并用 github 初始化文件夹

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main(int argc, char* argv[]) {
    std::vector<std::string> hello = {"Hello", "World", "OpenFOAM", "and", "git"};

    for(string& s : hello) {
        std::cout << s << " ";
    }

    std::cout << std::endl;

    return 0;

}
```

4. 讲解，task.json，c_cpp_properties.json，launch.json，这里已 GCC``为主，``clang msvc 不涉及！

   intellisense，智能补全，信息提示。

```
{
"configurations": [
    {
        "name": "Linux",
        "includePath": [
            "${workspaceFolder}/**"
        ],
        "defines": [],
        "compilerPath": "/usr/bin/g++-9",
        "cStandard": "c17",
```

（下页继续）

(续上页)

```
        "cppStandard": "c++14",
        "intelliSenseMode": "linux-gcc-x64"
    }
],
"version": 4
}
```

这里可以两种设置方式，一种 UI 一种 “JSON“

task.json，它的模式都是从任务来的，通过 terminal -> configure default build task 来写入

```
{
"version": "2.0.0",
"tasks": [
    {
        "type": "shell",
        "label": "C/C++: g++ build active file",
        "command": "/usr/bin/g++-9",
        "args": [
            "-std=c++14",
            "-g",
            "${file}",
            "-o",
            "${fileDirname}/${fileBasenameNoExtension}"
        ],
        "options": {
            "cwd": "${workspaceFolder}"
        },
        "problemMatcher": [
            "$gcc"
        ],
        "group": {
            "kind": "build",
            "isDefault": true
        }
    }
]
}
```

这个是编译程序，但是注意 -g 如果没有这个，就很难断点调试。

launch.json，它的模式都是从任务来的，通过 terminal -> configure default build task 来写入

```
{
"version": "0.2.0",
"configurations": [
```

(下页继续)

(续上页)

```
    {
        "name": "g++-9 - Build and debug active file",
        "type": "cppdbg",
        "request": "launch",
        "program": "${fileDirname}/${fileBasenameNoExtension}",
        "args": [],
        "stopAtEntry": false,
        "cwd": "${workspaceFolder}",
        "environment": [],
        "externalConsole": false,
        "MIMode": "gdb",
        "preLaunchTask": "C/C++: g++ build active file"
    }
]
}
```

5. 通过断点调试功能，使得程序跑起来，没有逻辑错误。
6. 这时，由于能力有限，需要远程帮助，通过 `live share` 插件，邀请远方的大佬给予支持！
7. 在大佬的帮助下，调试顺利通过，这时可以将程序推送到远程库方便保存，例如：github
8. vscode 推送完毕后，同样演示在命令行演示推送，这里最好方式，直接先 clone 下来测试库 “git“，然后随便添加一些内容，然后在 “git push“

7.5 远程开发 SSH

7.5.1 使用命令行

```
$ ssh ubuntu@124.222.97.12
$ pwd
$ sudo apt install build-essential
$ mkdir CPP
$ cd CPP
$ vim hello.cpp
$ g++ hello.cpp
$ ./a.out
```

这相当与一台 `linux` 设备，多说一点，可以作为自己博客、个人技术网站等地址

如果要将本地内容拷贝到远程该如何？使用 `scp`

```
$ scp gitHello.cpp ubuntu@124.222.97.12:/home/ubuntu
```

那反过来呢？

7.5.2 使用 vscode

这里用到插件，Remote-SSH 来连接上面的地址，可以在上面写代码的。

7.5.3 可否用 vscode 跑 OpenFOAM 算例呢?

时间足够可以演示一下!

7.6 参考学习资料:

菜鸟

参考文献

[1] William Shotts. *The Linux Command Line*. A LinuxCommand.org Book, 2019.