Week 2-2: linux shell, Vim & GNU make

Mayrain

2023 年 11 月 25 日

写在文档最前面：

一个有标题的文档需要加上`\maketitle`，否则不会显示标题。我只能说 latex 真是难伺候的主子。虽然其他的也好不到哪里去就是了，笑。

本次学习的内容：
shell，terminal，vim，GNU make

1 Overall

注册表：电脑中的巨大数据库，所有的设置都在注册表中有值。32 年之后，仍然需要更改注册表。
windows 也没有包管理器。windows 是闭源软件。
软件上兼容性还行，但是硬件不行。仅微软在开发。CPU 架构，arm64，x86，x86-64 云云。
windows 不能使用 nul 作为文件名。至今为止仍然如此。

$$linux -> android$$
$$-> arduino$$

2 terminal and shell?

有关 terminal 和 shell terminal，是一个模拟传统终端的东西。他本质上还是个应用程序，提供了窗口和交互功能的应用程序。
只是 terminal 他内部运行的是 shell 而已，shell 才是执行命令的。
cmd 不算 terminal，而算是 shell。windows terminal 才是。很多系统都会自带 terminal 的

2.1 basic knowledge

shell：壳。一个应用程序。它本身也是用户与系统内核交互的界面而已。接受 + 解析输入，交给操作系统执行，并返回输出。比较好用的 shell：zsh（z shell）pwsh 7（问题来了，powershell 和 pwsh 有什么区别?）
对于命令行来说，也可以用命令行中查看网页。他其实就是一种操作计算机的方式。对于 ssh 里链接服务器，那更是如此。

dev cpp -> vscode & gcc

这种转换可以让你熟知编译原理。另外你最好需要知道 json，否则每次都配就很恶心。

插件部分:
oh my zsh, powerlevel10k(p10k), git, sudo, z, zsh-autosuggestions, zsh-syntax-highlighting
可参看 tony crane 的博客。

2.2 command introduction

shell 的用法：
命令与位置非常有关系。rm -rf ./ 这个命令让你可以删除当前目录下的所有文件。 则代表 home。
另外，*nix 系的路径分隔符是/，而 windows 是\。
*nix 系没有分盘，只有一个根目录。，但是 windows 则有盘的概念

prompt：命令提示符，就是跳出来的那一行，告诉我们信息（权限和所在目录）并等待我们输入。

相应的一些命令：
pwd：获取当前路径。
cd：切换路径../代表上一级目录，./代表当前目录，/代表根目录，代表 home 目录。
以根目录（/）为起始的路径叫做绝对路径，而以当前目录为起始的路径叫做相对路径。

ls：列出当前目录下的文件。
-a：列出所有文件，包括隐藏文件。
-l：列出详细信息。包含权限，大小，时间。
我们也可以用-al 和-a -l 来做。

mkdir：创建目录。
touch filename：创建文件。这里可以跟上多个文件名。

cp src dest：复制文件。不加参数是不能复制文件夹的。
这里的 dest 两种情况：
1. dest 为目录，则复制到该目录下。
2. dest 为文件，则复制并重命名，并归入当前目录。本质上和 1 是一样的，这里用到的是相对路径。
-r：递归复制。

mv src dest：移动文件。它也可以被用于重命名
-r：递归移动。

rm filename：删除文件。
-r：递归删除。
-f：强制删除。
-rf：强制递归删除。这命令非常危险。

find path -name filename：在 path 下查找文件名为 filename 的文件。

模糊查找 * 也可以在此处使用。但是要用引号（单双都可以）括起来。

-type：指定类型。

-size：指定大小。

-mtime：指定修改时间。

grep：在文件中查找字符串。

-n：显示行号。

cat filename：查看文件内容。将会在终端中全部打出。

*concatenate 的缩写。

也可以用于拼接文件。但不是真的拼在了一起，只是一起输出了而已。

-n：显示行号。

head filename：查看文件头部。默认显示前 10 行。

-n：显示前 n 行。

tail filename：查看文件尾部。默认显示后 10 行。

-n：显示后 n 行。

less filename：分页显示文件内容。

可以实现查找和更好的翻页，还可以使用滚轮。

more filename：分页显示文件内容。

这个就 just so so 了。

hexdump filename：以 16 进制显示文件内容。

-C：以 16 进制显示文件内容，并且显示 ASCII 码。

-n N：只显示前 N 个字节。

其他文档：

man xxx：查看 xxx 的帮助文档。他的功能强大到 C 语言的函数都能看。

echo xxx：输出 xxx。

clear：清屏。
whoami：查看当前用户。
ps：查看当前进程。
kill pid：杀死进程。
　　pid：进程 id。
　　应该可以和 ps 一起用。
date：查看当前时间。
env：查看环境变量。
export var=value：设置环境变量。
（这样配置出来的环境变量只能在当前执行的 shell 中有效，重开就无用了。可以通过写入配置文件，也就是 shell 在加载时会执行的脚本文件，来使其持久化。插件也是如此实现的。）
unset var：删除环境变量。
　　有关环境变量请参看 week_1 中相关内容
　　我去，这里提醒一下，latex 中下横线也是转义字符，所以要用_来表示下横线。

chmod：修改权限。
　　-R：递归修改。
　　linux 中并不看后缀名确认是否可执行，而是看权限。如果权限上标明并不能执行，那么他就是不能执行。权限的表示是一串字母。
diff：比较两个文件的不同。

curl：发送 http 请求。
wget：下载文件。

2.3 advanced content:redirection and pipe

重定向：

标准输入流 stdin(standard input)
标准输出流 stdout(standard output)
标准错误流 stderr(standard error)

一般所说的是将标准输出流重定向到某个文件，也就是：
echo hello world > a.txt
这样就将 hello world 这个字符串写入到了 a.txt 文件中。注意这是覆盖的。
同时，» 则是追加。它意味着不覆盖源文件，而是添加到末尾。

反过来也可以将文件的内容重定向到标准输入流中：
cat < a.txt
这是一种给程序提供输入的方式。
cat file1 file2 > file3
这则是将 2 个文件拼接，输出的结果存入 file3 的操作。

另外，2> 则是将标准错误流重定向到某个文件。

最后，还可以用 &> 将标准输出流和标准错误流一起重定向。
例如，echo hello world &> a.txt

重定向：将文件输入到命令/将命令输出到文件
管道：将命令的输出作为另一个命令的输入
管道操作符：|
利用管道操作符将数个命令连接起来，形成一个命令序列，以完成较为复杂的任务。

例子：
echo Hello world
这是用 echo 命令输出 hello world。
echo Hello world | cat
这是将 echo 命令的输出作为 cat 命令的输入，最后的输出是由 cat 完成的
实用案例：
ls | tail -n 5
输出前 5 个文件。这就是很巧妙的组合。

ls | head -n 15 | tail -n 5
这就更加巧妙了，先输出头 15 个，然后取后五个，相当于取了 11-15 文件。
ls | less
这就是分页显示文件的方法。真是妙。
ls | grep 'a'
这就是查找文件名称中带 a 的方法。

3 vim

Vi->vim，vim 是所谓的 Vi improved，也就是 Vi 的改进版。
除此之外还有 neovim 和 lunarvim，还可以在 vsc 上搭载插件使用 vim。出于学习目的我们这里只使用 vsc+vim

介绍了一些教程，这里不多说了。

4 GNU make

make：自动化。
只要写一个 makefile，就可以通过一句命令自动完成编译，测试，打包，部署等操作。任何命令都可以执行。

大致的 makefile 结构：

target：prerequisites
command
...
...
...

这里的 target：目标文件/可执行文件/标签
commmand：执行命令，即触发 target 时执行的 shell 命令。
开头加 @：不在终端输出命令

开头加 -：忽略错误，继续执行
.PHONY:clean，phony 定义伪目标。
prerequisites：依赖的 target

用人话来说，就是我们要生成一个 main.out，那么我们需要调用 main.o 和 aux.o。main.out 目前并不存在，但是这个程序执行完了就有了。

```
CFLAGS = -std=c99
main.out: main.o aux.o
    gcc    $ ^-o $@       $(CFLAGS)
```

变量定义：
1. VAR=value：value 在使用时才被展开。
2. VAR:=valude：value 在定义时就被展开了。
3. VAR？=value：只有在 VAR 未定义的时候，才做这样的定义。
4. VAR+=value：将 value 追加到 VAR 末尾。

变量引用：
1. $(VAR) 或者 $VAR：引用变量 VAR 的值。
2. 会以“字符串”形式展开。类似于 C 中的宏展开。
3. $（shell command）可以将 command 的输出结果作为替换值
4. 一些内置变量 $(PATH),$(MAKE)

这一段我们还是另学为好，这里学太杂了，好烦。
资料：GNU make 官方文档，和我一起写 Makefile
另外我们可能要先学编译原理，然后再学 makefile 才能符合课程拓扑。

另外我们还有一些插件和命令行工具。旨在优化命令行的使用体验。甚至有做 C++ 的解释器，我直呼牛逼。以及一些熟人，ffmpeg