Introduction of Linux 

汪杰
51255901002@stu.ecnu.edu.cn

Press Space for next page →

Linux

首先，要有一个能跑Linux的运行环境（任选一个）

- 虚拟机：
  - 【建议】VMware（需要激活码）
  - VirtualBox
  - WSL2 (Windows Subsystem for Linux)
- 物理机：
  - 一台Linux电脑 ...
  - Mac （大概可以）
  - 云服务器
  - 双系统
  - ...

Ubuntu

Linux有许多不同的发行版 (distribution)，Ubuntu是一个常用的Linux发行版

请在互联网上搜索安装教程（加上对应的运行环境作为前缀）

- 例如：“VMware Ubuntu 22.04 安装教程”

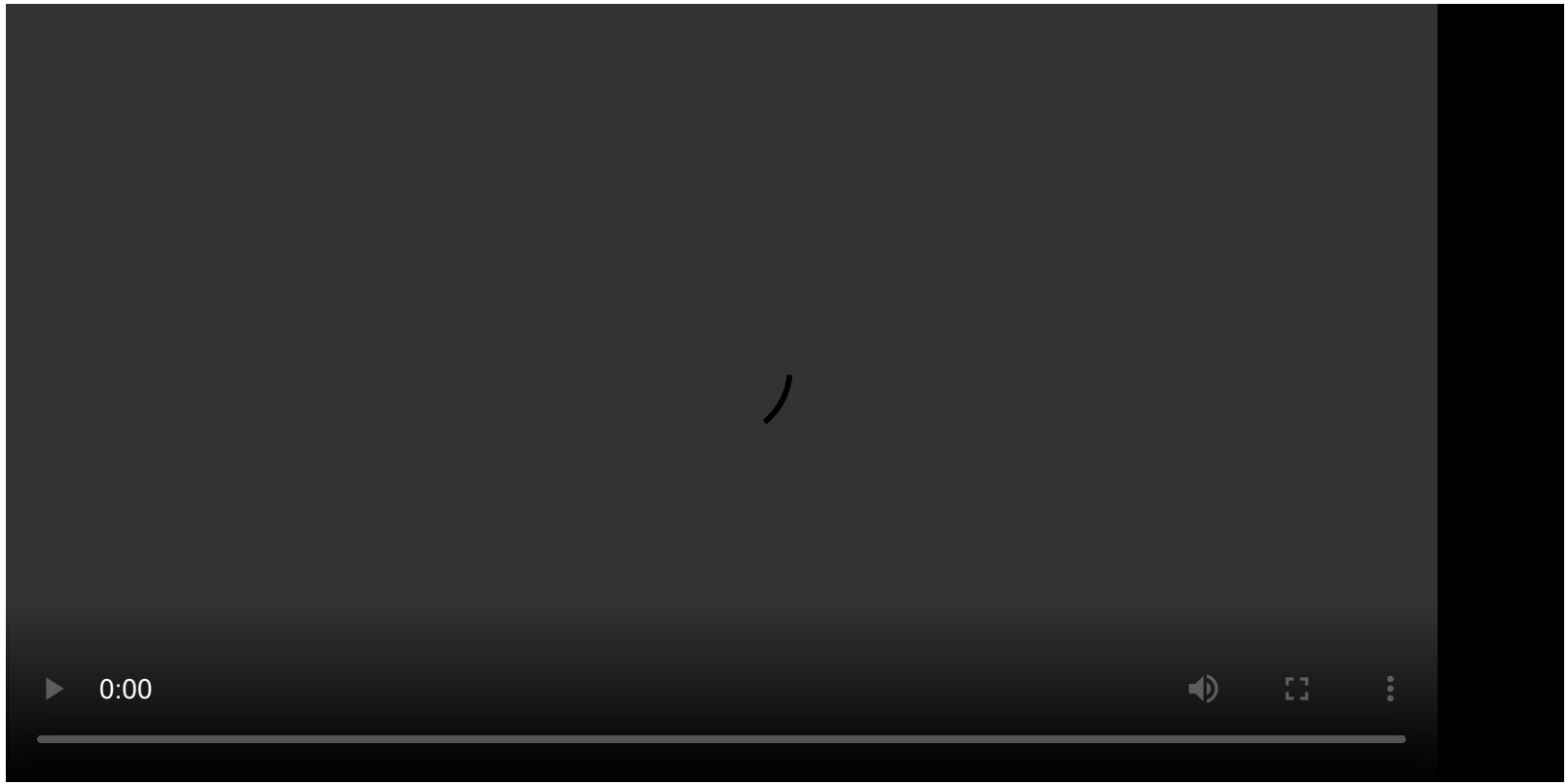
> WSL会自动下载 Ubuntu，请参考官方教程安装WSL

【重要】 在VMware或VirtualBox中成功安装系统后，请**立即**保存快照

【重要】 不要强制关闭虚拟机！请等待正常关机流程全部结束。

【重要】 这里每出现一个红字，就有一位无辜的同学重装了系统。

Shell 入门



https://www.bilibili.com/video/BV1ah411R7W6

1. Shell 入门 
    1. Hello Shell !
    2. 重定向/管道
    3. 任务管理
2. Shell 脚本
    1. 变量
    2. 分支 & 循环
    3. 条件表达式
    4. 函数
    5. 静态代码检查
3. 命令行环境
    1. 环境变量
    2. 权限
    3. Shell 是什么
4. 推荐阅读

Hello Shell !

echo & Quoting

```
echo Hello shell
echo  hello      shell  with  a   l o t   of    spaces
echo "Hello      shell"    'with correct spaces'
```

Shell Variables

```
foo=bar
# foo = bar will NOT work (what is this?)
echo $foo
echo "$foo" '$foo'  # bar $foo

foo=sun
echo $fooshine     # $fooshine is undefined
echo ${foo}shine   # displays the word "sunshine"
```

Command Substitution

```
foo=$(pwd)
echo $foo
echo "We are in $(pwd)"
```

重定向/管道

Redirection/Pipeline

- 输入重定向：`< file`
- 输出重定向（覆盖/追加）：`> file_overwrite`, `>> file_append`

```
echo $(pwd) >> file.txt  # append working directory to file

gcc -E test.c > test.i  # or `gcc -E test.c -o test.i`

cat file1.txt - file3.txt < file2.txt > output.txt  # This is the real `concatenate`
```

`-` 是什么文件？ `man cat`

- 管道：`command1 | command2`

```
ls | wc  # Well, it's tricky here (does not match `ls`), try `ls | cat`

python train.py | tee train.log  # print to screen and save to file at the same time

history | tail  # last 10 commands
```

More pipelines

gen_rnd.sh： 批量生成随机数

```
./gen_rnd.sh | sort -n | uniq  # unique number

./gen_rnd.sh > data.txt
cat data.txt | paste -sd+ | bc | factor  # sum all numbers and find prime factors
echo "print($(cat data.txt | paste -sd+))" | python | factor  # another approach
# pure python seems more readable
cat data.txt | python -c "import sys; print(eval('+'.join(sys.stdin.read().split())))"
```

```
# cd xv6

grep -rn "sleep(" --color=always | less  # all lines containing "sleep("
# also, try vscode

find . -name "*.[ch]" | ls -l | awk '{ print $5 }' > data.txt  # 5th column of ls, source code file size

find . -name "*.[ch]" | xargs wc -l  # source code line count
find . | grep "\.[ch]$" | xargs wc -l  # another approach
diff <(find . -name "*.[ch]") <(find . | grep "\.[ch]$")  # compare two approaches
```

Linux 三剑客： `awk` `sed` `grep`

STFW (Search The Friendly Web)

坚持查阅英文资料：

1. 问题翻译成英文
2. 仅保留关键字 (search engine friendly)
3. Google

例子：

- 问答网站
  - compare output of two commands bash
  - python print variable name and value
- “精选摘要”（虽然不一定完全正确）
  - size of long
  - mac address collision probability

任务管理

Job control

- 终止进程：
  - 按下 `Ctrl+C`
  - `pkill proc_name`
  - 查看进程 pid `ps aux | grep "proc name"` 然后发送SIGTERM `kill -TERM <pid>`
- 后台运行：在命令末尾加 `&` 后缀，即 `command &`
- 暂停进程：按下 `Ctrl+Z` （放到后台，且暂停 suspend ）
- 让后台进程来到前台： `fg` （如果有多个： `fg %2`）
- `bg` ：恢复一个暂停的进程，在后台运行（`Ctrl+Z` 后 `bg` 相当于给命令加上 `&`）
- 查看所有后台进程： `jobs`
- 等待后台进程结束： `wait`
- “任务管理器”： `top` `htop` … （列出系统中的所有进程，及相关信息）

1. Shell 入门 
    1. Hello Shell !
    2. 重定向/管道
    3. 任务管理
2. Shell 脚本
    1. 变量
    2. 分支 & 循环
    3. 条件表达式
    4. 函数
    5. 静态代码检查
3. 命令行环境
    1. 环境变量
    2. 权限
    3. Shell 是什么
4. 推荐阅读

Shell 脚本

把一堆 shell 命令放到文件里（批处理）

run.sh：

```
mkdir -p log

python train.py --log_dir=log --epoch=100 --lr=0.005
```

运行脚本：

```
bash run.sh
```

变量

```
name=value
$name
${name}  # Parameter Expansion
```

Special Parameters :

$0	filename of the script
$1 to $9	Arguments to the script. $1 is the first argument and so on.
$#	the number of the arguments
$?	Return error code of the previous command
$@	all command line arguments
$$	Process identification number (PID) for the current script

`man sh` `man bash`

分支 & 循环

```
if  [condition1]; then
    # ...
elif [condition2]; then
    # ...
else
    # ...
fi
```

```
for i in 1 2 3 4 5; do # or: for i in {1..5}; do
    echo $i
done

# C type for loop, won't work in some POSIX shells
for (( i = 1; i <= 5; i++ )); do
    echo $i
done

while [ condition ]; do
    # ...
done

# do while loops, until loops, etc.
```

`man sh` Flow-Control Constructs / `man bash` Compound Commands

条件表达式

Conditional Expressions

[-f FILE] [-d FILE]	FILE exists and is file / dir
[-r FILE] [-w FILE] [-x FILE]	FILE readable / writable / executable
[-z STRING]	STRING length zero (empty STRING)
[-n STRING] [STRING]	non-empty STRING
[STRING1 == STRING2]	STRING equal
[INTEGER1 -eq INTEGER2]	INTEGER equal
[INTEGER1 -ge INTEGER2]	INTEGER >=

`man test` or STFW (eg. "bash file exists")

> 在 `bash` 中, `[[ condition ]]` 更好, 但是这种写法不兼容 `sh`

函数

Shell Functions

mcd.sh：

```
mcd () {
    mkdir -p $1
    cd $1
}
```

```
source mcd.sh

mcd foo
# now in foo
```

一些例子 

gen_rnd.sh： 批量生成随机数

- `{1..5}` 是什么？试试 `echo {1..5}` 。答案在 `man bash` Brace Expansion，或者 STFW
- `$(())` ？ bash 竟然可以做数学运算！ `man bash` Arithmetic Expansion

example.sh

- `/dev/null` `2>&1` ? STFW
- 顺带一提， `2>&1` 的顺序是有影响的（涉及到 bash 的实现）

install.sh (oh-my-zsh 安装脚本) ~~你已经学会1+1了，来看这个积分~~

- 考虑各种兼容性 `command_exists()` `user_can_sudo()` ...
- 彩色输出（用户体验） `man console_codes` , STFW: ANSI escape code

run-tests.sh： 作业中的测试脚本

- 处理 `-h -v` 等 shell command option

静态代码检查

Static code analysis / Lint

`shellcheck` 可以给出一些 Error 和 Warning， 以及著名的 "Did you mean" ~~(教你写代码)~~

```
sudo apt install shellcheck
shellcheck mcd.sh
```

```
In mcd.sh line 1:
mcd () {
^-- SC2148: Tips depend on target shell and yours is unknown. Add a shebang.

In mcd.sh line 2:
    mkdir -p $1
             ^-- SC2086: Double quote to prevent globbing and word splitting.
Did you mean:
    mkdir -p "$1"

In mcd.sh line 3:
    cd $1
    ^---^ SC2164: Use 'cd ... || exit' or 'cd ... || return' in case cd fails.
       ^-- SC2086: Double quote to prevent globbing and word splitting.
Did you mean:
    cd "$1" || exit
```

如何运行脚本

- `bash script.sh`： `bash` 程序读取脚本内容并解释执行 (interpret)
- `./sript.sh`：让 shell （根据文件内容）自己决定如何运行
  - 脚本的第一行声明解释器 (interpreter) `#!/bin/bash`，这一行也叫做 Shebang
  - 需要脚本有可执行权限 (executable)，添加权限：`chmod +x script.sh`
- `source script.sh`：在当前 shell 环境下运行脚本
- `. script.sh`：等同于 `source script.sh`

1. Shell 入门 
    1. Hello Shell !
    2. 重定向/管道
    3. 任务管理
2. Shell 脚本
    1. 变量
    2. 分支 & 循环
    3. 条件表达式
    4. 函数
    5. 静态代码检查
3. 命令行环境
    1. 环境变量
    2. 权限
    3. Shell 是什么
4. 推荐阅读

命令行环境

```
$ which ls
/usr/bin/ls
$ ls -l $(which sh)
lrwxrwxrwx 1 root root 4 Aug 22  2022 /usr/bin/sh -> dash*
$ type cd  # why?
cd is a shell builtin
$ type mcd
mcd is a function
```

`man sh` Search and Execution & Path Search：

- 如果是相对路径或绝对路径，直接执行
- 否则，遍历 `$PATH` 搜索可执行程序
- 执行的程序（eg. `cat`）是 shell 的子进程

`man builtins`：`cd fg bg jobs wait ...` 都是内置命令

环境变量

- 使用 `env` 命令查看
  - PATH : 可执行文件搜索路径
  - PWD : 当前路径
  - HOME : home 目录 (vs. `man sh` Tilde Expansion)
  - PS1 : shell 的提示符
  - ...
- `export` : 告诉 shell 在创建子进程时设置环境变量
- `name=value command` 为命令单独设置环境变量

破案了， `CUDA_VISIBLE_DEVICES=0,1 python train.py` ： 通过环境变量指定 GPU

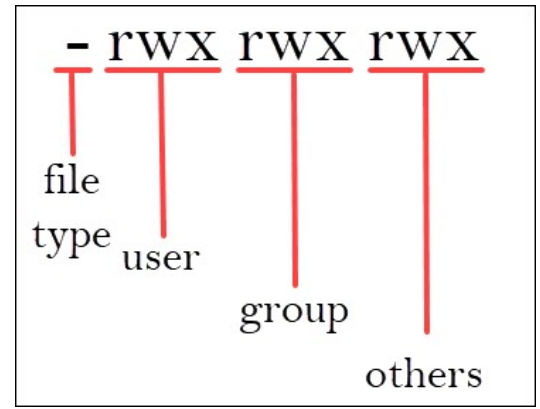
权限

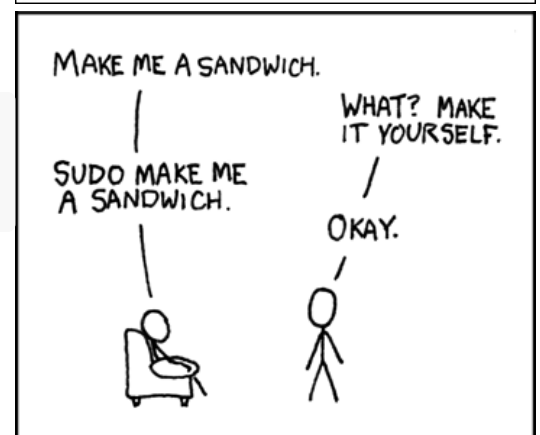
- 查看权限：`ls -l`
  - file type : 常见的有 regular file (–), directory (d), link (l) ...
  - r, w, x : read, write, execute
- 修改权限：`chmod`
- 根用户 root （超级用户，管理员...）拥有所有权限
  - 【不建议】`su` : switch user，切换为 root 用户
  - `sudo command` ： 以 root 权限运行命令

```
apt install some_package

sudo !!  # repeat last command with sudo
```

`man bash` `\!\!` （需要转义）





权限

```
$ echo hello > /etc/a.txt
bash: /etc/a.txt: Permission denied
$ sudo echo hello > /etc/a.txt
bash: /etc/a.txt: Permission denied
```

```
$ echo hello | sudo tee /etc/a.txt   # WTF?
hello
$ echo hello > a.txt && sudo mv a.txt /etc
$ sudo bash -c 'echo hello > /etc/a.txt'
```

Good practice: 最小权限原则

```
# cd /sys/class/leds/input1::capslock
echo 1 | sudo tee brightness
echo 0 | sudo tee brightness
```

于是 ... blink.sh —— 可以作为工具链的一环，甚至定时触发(crontab)

```
./blink.sh &

pkill blink
```

Shell 是什么

What is "the Shell"?

- Shell 连接了用户和操作系统
- Shell 是 “自然语言”、“机器语言” 之间的边缘地带！
- 程序员的角度：Shell 是一个程序
  - `sh` : Bourne shell
  - `bash` : Bourne Again shell
  - `zsh` : Z shell
  - `fish` : the friendly interactive shell
  - `ksh` : Korn shell
  - `csh` : C shell
  - ...

RTFM (Read The ~~F**king~~ Friendly Manual)

“Unix is user-friendly; it’s just choosy about who its friends are.” ——*The Art of UNIX Programming*

【推荐】`man sh`

更实用的 `man` : tldr (`pip install tldr` or `npm install -g tldr`)

推荐阅读

- The Missing Semester of Your CS Education and [video on bilibili]
- Linux入门教程 by jyy
- 命令行的艺术 (the art of command line)