

# 230704 c高级

## 1. linux 的发展

```
1  贝尔实验室 、麻省理工 -----> multics 计划      失败
2  汤姆森.肯      -----> unics
3  -----> 贝尔实验室推广 -----> unix{ sys 5      BSD}
4
5  莱纳斯.托瓦斯 -----> 类unix -----> linux
6
7  linux 发行版本:
8      ubuntu -----> “人道主义”
9
10  GUN & GPL
11      GUN is not linux
12  GPL 通用协议
13
```

## 2. linux 的体系结构

```
1  应用层 -----命令、程序、shell 脚本-----
2
3      |
4      |shell \      系统调用
5      | /
6  内核 -----
7      内核功能:
8          文件管理
9          设备管理
10         进程管理
11         内存管理
12         网络管理
13  硬件 -----
14         摄像头、鼠标。。。
15  内核:
16         内核是Linux系统的最底层，提供了系统的核心功能并允许进程以一种
17         有序
18         的方式访问硬件。
19         用于控制进程、输入、输出设备、文件系统操作、管理内存；
20
21  linux 系统支持多任务、多用户的模式运行;
```

## 3.shell 命令

```
1 shell是一种命令行解释器，
2 shell --> "贝壳" 对操作系统起到保护的作用；
3 用户 和 内核的交互：
4 用户在命令行提示符下键入命令文本，开始与Shell进行交互。
5 接着，Shell将用户的命令或按键转化成内核所能够理解的指令
6 控制操作系统做出响应，直到控制相关硬件设备。
7 然后，Shell将输出结果通过Shell提交给用户。
8
9 复习常见的shell命令：
10 ls 、 cd 、 pwd 、 whoami
11 ls 选项 参数
12 选项：
13     -a: 查看以点开头的隐藏文件；
14     -l:查看文件的详细属性；
15     -R: 递归查询
16     -i: 查询inode号
17 eg:
18     -rw-rw-r-- 1 jiangcx jiangcx 903 Jul 23 22:48
19     tel.c
20     -: 文件的类型
21     linux 下的七大文件类型：(linux 一切皆文件)
22     - 普通文件: xxx.c xxx.h xxx.txt
23     d 目录文件: 相当于windows的文件夹
24     b 块设备文件:磁盘(/dev/sda ...)
25     c 字符设备文件:摄像头("/dev/video0")、串口
26     (/dev/ttyUSB0)
27     p 管道文件:(mkfifo fifo)用于本地进程间的通信；
28     l 链接文件：(软链接:相当于windows的快捷方式)
29     s 套接字文件：(socket)用于网络间通信；
30     rw- rw- r--: 文件的权限 (0664)
31     文件的三种权限：
32         r --> 读权限 4
33         w --> 写权限 2
34         x --> 执行权限 1
35     文件的权限包含：
36         用户的权限
37         组的权限
38         其他权限
39     一个权限可以用八进制表示：
40     一个文件的最高权限：0777
41     1: 硬链接数
42     jiangcx:用户名
43     jiangcx:组名
44     903: 文件的大小
45     Jul 23 22:48: 文件最后修改的时间
46     tel.c 文件名
47
48 路径：
49     相对路径:从当前出发的一条连续不断的路径；
50     绝对路径:从根目录出发的一条连续不断的路径；
```

## 3.2 cd

```
1 cd 目录的路径 去到某个目录下
2 cd (cd ~) 去到家目录 (/home/hqyj)
3 cd / 去到根目录
4 cd .. 返回上一级
5 cd - 返回上一步;
```

## 3.3 目录的创建

```
1 mkdir dirname
2 mkdir -p demo1/demo2/demo3 递归创建(创建多级目录)
3
4 删除:
5 rm -r dirname 删除一个目录
6 rmdir dirname 删除一个空目录
```

## 3.4 文件

```
1 文件的创建:
2 ①touch filename 文件存则更新时间戳, 不存在则创建一个空文件;
3 ②vi filename 文件存在则打开不存在则创建;
4 文件的删除:
5 rm filename
6 文件的查看:
7 cat filename 文件的查看
8 cat -n filename 查看文件自带行号
9 vi编辑器:
10 三种模式:
11 命令行模式:刚打开文件进入的模式;
12 在任意模式下按 Esc 切换到命令行模式:
13 nyy --> n代表行数 复制多行
14 ndd --> 剪切多行
15 p --> 粘贴
16 u --> 撤销
17 ctrl + r 反向撤销
18
19 gg=G 代码缩进对齐
20
21 /str 查找
22 n:向下查找
23 N:向上查找
24
25 1G: 光标去到第一行
26 G: 光标去到最后一行
27 插入模式:
28 在命令行模式的基础上, 按 a / i / o;
29 底行模式:
30 在命令行模式的基础上, 输入 : (shift + :)
31 w: 保存
```

```

32      q: 退出
33      wq: 保存并退出
34      Q!: 强制退出
35      WQ!: 强制保存并退出
36
37      设置行号: set number
38      取消行号: set nonumber
39
40      :num 去到第num行
41
42      vsp filename 水平分屏
43      ctrl + w + w 切换光标到分屏;
44
45      替换:
46      . ---> 当前
47      $ ---> 末尾
48      g ---> 每行的所有
49      % 全文
50
51      范围 s/str1/str2
52
53      1, $ s/str1/str2
54      //将第一行到最后一行的每行第一个str1替换为str2;
55      1, $ s/str1/str2/g
56      //将第一行到最后一行的所有str1替换为str2
57
58      块复制, 块剪切
59      范围 y
60      范围 d
61      eg:
62      . , $ y 复制当前到最后一行
63
64

```

### 3.5 文件的复制

```

1  cp 文件的路径1 文件的路径2
2  cp 文件的路径1 目录的路径2
3      eg:
4          cp test.c ../ 将test.c 拷贝到上一级的目录
5          cp test.c ../my.c 将test.c 拷贝到上一级并重命名为my.c
6
7  cp -a 目录的路径1 目录的路径2
8      cp demo1 demo2

```

### 3.6 文件移动

```

1  mv 路径1 路径2
2      mv test.c my.c 重命名

```

## 3.7 其他常用命令

```
1 clear 清屏
2 exit 退出终端
3 su 切换用户
4 sudo 临时超级用户权限
5 diff
6     diff file1 file2 比较两个文件是否一样
7 eog 图片查看器
8     eog 1.jpg
9 history 查看历史输入记录
```

## 3.8 快捷方式

```
1 创建终端：
2     ctrl + shift + n
3 终端的切换：
4     Alt + Tab
5 Tab：
6     按一下Tab 补全内容
7     按两下：显示待输入的命令
8     ctrl + l 清屏
9     ctrl + c 结束
10 ↑:向上查看历史输入记录
11 ↓:向下查看历史输入记录
```

## 4. 软件包的管理

### 4.1 dpkg 离线安装

```
1 软件包名的构成：
2  s1    _5.02-1    _amd64    .deb
3  软件名 版本号  修订版本号  体系架构    后缀
4
5                      i386: 32位
6                      amd64(x64): 64位
6  运行软件：s1 (软件名)
7
8  软件安装：
9     sudo dpkg -i 软件包名
10 卸载：
11     sudo dpkg -r 软件名
12 完全卸载：
13     sudo dpkg -P 软件名
14 查看软件版本号：
15     sudo dpkg -l 软件名
16 查看软件包的安装清单
17     sudo dpkg -L 软件名
18 查看软件安装的状态
19     sudo dpkg -s 软件名
```

## 4.2 apt 在线安装

```
1  软件源：
2      /etc/apt/sources.list (服务器镜像地址)
3  创建服务器的索引（清楚镜像地址上有哪些软件资源）
4      sudo apt-get update
5  软件包的下载缓存地址：
6      /var/cache/apt/archives
7  软件的安装：
8      sudo apt-get install 软件名；
9  软件的卸载：
10     sudo apt-get remove 软件名
11  清空软件包
12     sudo apt-get clean
13  检查系统中依赖关系的完整性
14     sudo apt-get check
15  将系统中的所有包升级到最新版本
16     sudo apt-get upgrade
17  重新安装：
18     sudo apt-get --reinstall install 软件名
19  完全卸载：
20     sudo apt-get --purge remove 软件名
21  只下载不安装：
22     sudo apt-get -d install 软件名
23  获取软件包的安装状态：
24     sudo apt-cache policy 软件名
25
26  2.配置网络 ----> 桥接模式
27     ping www.baidu.com
28
```

## day2

### 1.文件互传（windows 和ubuntu）

#### 1.1 重装安装vm-tools

```
1  1.确保有安装包
2      VMwareTools-10.3.22-15902021.tar.gz
3      设置 ----> 重新安装vmware-tools ,点击就可以找到安装包；
4  2.解压：
5      sudo tar -xvf VMwareTools-10.3.22-15902021.tar.gz
6      会生成一个目录： vmware-tools-distrib
7  3.cd vmware-tools-distrib
8  4.sudo ./vmware-install.pl
9      一直回车，如果在安装的过程中出错(.....[No])，就把相应已经存
    在的
10     文件删除(新建终端删除)，接着安装完然后再重复第4步重新安装；
```

```
11
12 重新安装Vm --- tools 是灰色的解决办法：
13 1.将虚拟机关机 ---> 找到设置 -->CD/DVD -->选择使用ISO映像
14 文件；去Vmware的安装路径下浏览，找到linux.iso选择；
15 2.打开虚拟机 ---> CD/DVD 里面就有Vmware的安装包；
16
```

## 1.2 安装共享文件夹

```
1 1.将虚拟机关机 --> 设置 --> 选项 ---> 共享文件夹 --> 总是启
  用
2  ----> 添加 --> 主机路径： windows的路径 ---->
3  设置在ubuntu 上此文件夹的名称 --> 应用；
4 2.打开虚拟机，去到 /mnt/hgfs里面就能找到共享文件夹；
5
6 1.可以将文件拷贝到共享文件夹，那么两边系统都能看的；
7 2.可以将windows（ubuntu）上的文件直接鼠标复制，
8  去到ubuntu(windows)的图形界面 鼠标粘贴；
9 3.对于普通文件从windows传到ubuntu，可以拷贝然后到ubuntu的终
  端，
10 右键粘贴文件名，使用命令mv 拷贝；
11  eg：
12  mv '/tmp/VMwareDnD/6RuSze/非递归先序.png' ./
```

## 2.文件相关的命令

### 2.1 文件的压缩

```
1 在对文件压缩的时候，压缩文件会把源文件覆盖，解压也是一样；
2 .gz： 压缩率最低，压缩速度最快
3  压缩：
4      gzip file.txt ---> file.txt.gz
5  解压：
6      gunzip file.txt.gz ---> file.txt
7 .bz2： 压缩率次之， 压缩速度次之
8  压缩：
9      bzip2 file.txt ---> file.txt.bz2
10 解压：
11      bunzip2 file.txt.bz2 ----> file.txt
12 .xz： 压缩率最高，压缩速度最慢
13  压缩：
14      xz file.txt ---> file.txt.xz
15  解压：
16      unxz file.txt.xz ---> file.txt
```

### 2.3 文件的归档

```
1 归档： 对批量文件进行打包，归档的对象是目录；
2 归档之后源文件依旧在，生成新的归档文件；
```

```

3   拆包之后，归档的文件依旧在，会重新生成源文件；
4   归档、压缩：
5       tar 选项 归档后的文件名 目录
6   拆包、解压：
7       tar 选项 归档后的文件名
8   选项：
9       归档并压缩：
10      -c 创建归档(打包)
11      -v 显示归档的过程
12      -f f后面固定跟随归档后的文件名
13      -z 归档并压缩为.gz的文件
14      -j 归档并压缩为 .bz2的文件
15      -J 归档并压缩为 .xz的文件
16      注意:选项中f放在最后，其他选项位置随意；
17   拆包并解压：
18      -x 创建归档(打包)
19      -v 显示归档的过程
20      -f f后面固定跟随归档后的文件名
21      -z 拆包并解压.gz的文件
22      -j 拆包并解压 .bz2的文件
23      -J 拆包并解压 .xz的文件
24
25   万能拆包：
26       tar -xvf 归档后的文件名；

```

## 2.4 文件查看

```

1   cat filename 查看文件的内容
2   cat -n filename 查看的内容会自带行号；
3
4   head -num filename 查看文件的前num行；
5   tail -num filename 查看文件末尾的num行；
6
7   more / less 以一种比例的方式查看文件，适合较大的文件查看；
8
9   file filename 查看文件类型；

```

## 2.5 重定向和追加

```

1   >:
2   重定向
3   eg:
4       cat /etc/passwd > test.txt
5   注意: test.txt 不存在则会创建，并把/etc/passwd里面的
   内容
6       重定向到test.txt中，如果test.txt存在，那里面的所有内
   容会
7       被替换；
8   >>: 追加
9   eg:
10      cat /etc/passwd >> test.txt

```



```

11 注意: test.txt 不存在则会创建, 并把/etc/passwd里面的内
    容
12 追加到test.txt中, 如果test.txt存在, 也会将内容从文件
    的
13 末尾追加写入;
14 eg:
15 将/etc/passwd 里的第45行显示在终端上;
16 方法1:
17 head -45 /etc/passwd > test.txt
18 tail -1 test.txt
19 方法2:
20 head -45 /etc/passwd | tail -1
21 | 管道:
22 将前面的结果作为后面的输入;
23
24 路径:
25 绝对路径: 从根目录出发的一条连续不断的路径
26 相对路径: 从当前出发的一条连续不断的路径

```

## 2.6 在文件中搜索字符串

```

1  grep "string" 文件名 选项
2  -n 搜索的结果待行号
3  -i 不区分大小写
4  -R 递归搜索, 给个目录, 搜索整个目录下的所有文件
5  -w 搜索更加严谨, 字符串的前后有其他字符就搜索不到;
6  选项可以多个一起用;
7  eg:
8  grep "str" test.txt -ni
9  表示是在test.txt 中搜索 str 这个串, 不区分大小写以及
10  搜索结果显示行号
11  grep "^str" ./ -niR
12  表示: 在当前目录的所有文件中搜索以str开头的串, 不区分大
    小写
13  以及搜索结果显示行号
14  grep "str$" test.txt -n
15  表示搜索以str结尾的串, 结果显示行号;
16  grep "str" test.txt -wni
17  表示: 搜索 str前后没有其他字符的串, 不区分大小写以及搜
    索结果
18  显示行号
19  grep "arr\[5\]" ./ -nR
20  表示搜索数组 arr[5];
21

```

## 2.7 搜索文件

```

1  find 搜索路径 -name filename

```

## 2.8 字符的提取

```

1 cut
2 eg:
3     cut -d ':' -f 1,5 /etc/passwd
4     : 是分隔符
5     1,5 提取的位置, 第一位置和第5位置;
6     /etc/passwd 文件名
7 eg:
8 ① 提取当前用户的组id 和用户id // 1000:1000
9     head -45 /etc/passwd | tail -1 | cut -d ':' -f 3,4
10
11 ② 在终端上显示当前用户行号 // 45
12     grep "hxyj" /etc/passwd -n | cut -d ':' -f 1
13

```

## 2.9 通配符

```

1 * 代表所有字符
2 ? 通配一个字符
3 [ab] 通配中括号里面的其中一个字符
4 [^ab] 通配一个字符, 除了a或者b的其他字符;
5 [a-z] 通配这个范围里面的一个字符
6 例如:
7     test1.c test2.c testa.c testb.c testab.c testc.c
8 命令:
9     ls *.c
10 输出:
11     test1.c test2.c testa.c testb.c testab.c testc.c
12 命令:
13     ls test?.c
14 输出:
15     test1.c test2.c testa.c testb.c testc.c
16 命令:
17     ls test[ab].c
18 输出:
19     testa.c testb.c
20 命令:
21     ls test[^ab].c
22 输出:
23     test1.c test2.c testc.c
24
25 eg:
26     char buf[32] = {0};
27
28     scanf("%s", buf);
29
30     puts(buf);
31 输入:
32     hello world
33 输出:
34     hello world
35

```



## 2.硬链接

```
1  硬链接的创建：（相当于对文件取别名）
2      ln 源文件 目标文件
3      硬链接是根据inode创建的文件，硬链接文件和源文件inode号一样；
4      当硬链接文件被修改，源文件也会被修改；
5      创建硬链接硬链接数会加一；
6      当原文件啊被删除，硬链接数会减一；
7      源文件被移动，硬链接数不变， 硬链接正常使用；
```

## 3.man 手册

```
1  man 函数名 （默认查看第一页）
2  man num 函数名 （查看第num页）
3
4  一般第一页查询 命令
5      第二页      系统调用函数的接口（open）
6      第三页      库函数
7
8  man -a 函数名（q退出当前页，回车进入下一页）
9
```

## 4.用户管理

### 4.1 adduser

```
1  1.创建用户
2      sudo adduser 用户名
3
4      /etc/skel 存放用户的模板
5      /etc/passwd 存放用户信息
6      /etc/group 存放用户的信息
7      /etc/shadow 存放用户密码的文件
8
9      sudo su 去到root用户下
10     su 用户名 去到某个普通用户
11
12     关机命令：
13     sudo shutdown -r now(时间) 重启
14     sudo reboot 重启
15     sudo shutdown -h now(时间) 关机
16
17     新添加的用户用不了sudo,解决办法：
18     （切换到root用户或者本地用户）
19     sudo vi /etc/sudoers
20     添加一行代码：
21     linux ALL=(ALL:ALL) ALL
22     （新用户名）
23     修改用户密码：
24     sudo passwd 用户名
```

## 4.2 usermod

```
1 1.
2 sudo usermod -aG linux test
3 将test追加到linux 这个组里面
4 2.
5 sudo usermod -c LINUX linux
6 将linux这个用户在/etc/passwd里的第五个参数修改为LINUX
7 //修改登录界面名称
8 3.sudo usermod -g test linux
9 //将linux 这个用户的组改为test;
10 4.sudo usermod -d /home/test linux
11 //将linux 这个用户的家目录改为/home/test
12 5.sudo usermod -l Linux linux
13 //将linux这个用户的用户名改为Linux
14
```

## 4.3 用户的删除及组的删除

```
1 sudo deluser 用户名 删除用户
2 sudo deluser --remove-home 用户名 删除用户及家目录
3 sudo delgroup 组名 删除组
```

## 5. 磁盘相关的命令

```
1 /dev/sda开头 磁盘相关的文件
2 /dev/sdb开头 移动硬盘相关的文件
3 /dev/sr0 光驱 (CD/DVD)
4 sudo fdisk -l 查看磁盘
5 sudo df -h 查看分区
6
7 挂载光驱:
8 sudo mount /dev/sr0 ~/myDVD
9 取消挂载:
10 sudo umount /dev/sr0
11 挂载优盘:
12 sudo mount /dev/sdb ~/myDVD
13 取消挂载
14 sudo umount /dev/sdb
15
```

## 6. 环境变量

```
1 env 查看环境变量
2
3 HOME: 存放家目录
4 PATH: 保存可执行文件的路径
5 如何输入可执行文件名就能运行:
6 ①导入临时的环境变量:
7 export PATH=$PATH:./
8 ②让环境变量长久有效
```

```

9      将export PATH=$PATH:./ 放在家目录下的.bashrc文件
    中
10      让脚本生效:
11      source .bashrc
12      ③打开/etc/environment 文件
13      sudo vi /etc/environment
14      PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:
15          /usr/bin:/sbin:/bin:/usr/games:/usr/local
16          /games:/snap/bin:./" (加上./)
17      source /etc/environment
18
19

```

## 7. 网络相关的命令

```

1  ifconfig 查询ip地址
2  ping (ping 两台主机之间是否能通信)
3
4  解决ubuntu网络图标消失的问题:
5  1.删除文件:
6  sudo rm /var/lib/NetworkManager/NetworkManager.state
7  2.将文件里的false改为true
8  sudo vi /etc/NetworkManager/NetworkManager.conf
9  3.重启网络服务
10  sudo service network-manager restart
11

```

## 8. 进程相关的命令

```

1  1.进程:动态的概念, 一个程序的执行过程;
2  进程号: pid
3
4  kill : 向内核发送信号;
5  kill -l 查看信号
6  eg:
7      kill -9 pid 向一个进程发送SIGKILL 的信号
8  ps -aux 查看进程
9  top 动态查看进程的状态

```

# day4

---

## 1.shell 脚本

```

1  shell 是命令行解释器
2  ubuntu 默认使用bash
3  #! /bin/bash 表示此脚本由bash 解释;
4  sh

```

```

5    csh
6    ksh
7 shell 脚本
8    shell命令的有序集合;
9    shell 是一门解释性的语言,不需要编译;
10   创建脚本的三步:
11       第一步: 创建一个xxx.sh的文件
12           vi xxx.sh
13       第二步: 给初始文件赋权限
14           chmod 0777 xxx.sh
15       第三步: 执行shell脚本
16           ./xxx.sh
17   eg:
18       1.在家目录下创建一个目录 mydir
19       2.将/etc/passwd 和 /etc/groff 拷贝到mydir
20       3.再将mydir 归档并压缩为 mydir.tar.gz
21   执行脚本的三种方法:
22       第一种: ./xxx.sh
23       第二种: bash xxx.sh
24       第三种: source xxx.sh
25
26       ./xxx.sh 适用于所有脚本的运行,需要赋权限: bash xxx.sh
27   默认
28       由/bin/bash解释
29       ./xxx.sh 和 bash xxx.sh 执行的时候都是创建一个子终端执
30   行,
31       再将结果返回到当前终端;
32   source 是在当前终端执行的,一般用于让脚本生效;

```

## 2.shell 变量

```

1   自定义变量:
2       1.命名规则: 见名知意
3       2.字母数字下划线不以数字开头
4       3.变量的前面不需要数据类型
5       4.变量内容都视为字符串
6       5.变量不需要提前定义,可以直接使用
7       6.变量的后面不需要给分号结束
8       7.在对变量赋值的时候,等号两边不能有空格
9   位置变量(相当于命令行参数)
10      $0 $1.....
11   预定义变量
12      $? $$
13   环境变量
14      PATH HOME ...

```

### 2.1 变量的引用

```

1   在引用变量的时候,需要在变量前面加上 $
2   eg:
3       $var 或者 ${var}

```

## 2.2 变量的赋值

```
1 var=123
2 var=hello //变量的重新赋值
3 当字符串有空格的时候，赋值的时候需要加引号，双引号单引号都可以；
4 如果引号里面有引用变量的情况，必须是双引号；
5 eg:
6   var1='hello world'
7   var2="hello world"
8   var="$var world" //变量的追加赋值
9 eg:
10 编写一个shell脚本，定义两个变量保存环境变量 HOME 和 PATH 的
    值；
11 将两个变量的值交换；
```

## 2.3 变量的取消赋值

```
1 unset var
```

## 2.4 命令置换

```
1 将一个命令的结果赋值给一个变量
2 eg:
3   var=`ls` (注意： ` 是Esc下面的按键 )
4   或者
5   var=$(ls)
```

## 2.5 位置变量

```
1 命令行参数
2   $0 $1 $2 $3.....${10} ${11}.....
3   $0表示脚本文件名，如果用source执行$0为bash
4   @$ * 表示所有的命令行参数
5   $$ 脚本执行的进程号 （预定义变量）
6
7   set 将命令置换的结果赋值给命令行参数
8   eg:
9   #ret=`grep "hello" test.txt -ni`
10  // test.txt 一个已经创建好的文件；
11   ret=`grep $1 $2 -ni` //将grep搜索的结果赋值给ret变量
12   //终端执行的时候应该加上命令行参数 如：
13   // bash xxx.sh "str" 文件名
14   echo $ret
15   //打印grep搜索的结果
16
17   set $ret //将命令置换的结果设置给命令行参数
18
19   echo $1 //搜索的第一条结果
20   echo $2//搜索的第二条结果
21   echo $3 //.....
22   echo $4
```



## 2.5 echo 输出

```
1 echo str 直接输出 并且自带换行
2 echo -n str 输出不换行
3 echo -e "hello\n" 识别特殊字符'\n'
```

## 2.6 字符串

### 2.6.1 字符串的拷贝

```
1 str1=hello
2 str2=world
3 str1=$str2
```

### 2.6.2 字符串的连接

```
1 str1=hello
2 str2=world
3 str1="$str1$str2"
```

### 2.6.3 求字符串的长度

```
1 ${#str}
```

### 2.6.4 字符串的提取

```
1
2 str="www.baidu.comwww.baidu.com"
3 1. ${str:start:len}
4 从左往右数到第start的位置，提取后面的len个字符；
5 eg:
6 ${str:4:3} --> bai
7 2. ${str:start}
8 从左往右数到start的位置，提取后面的所有字符；
9 eg:
10 ${str:4} ----> baidu.comwww.baidu.com
11 3. ${str:0-start:len}
12 从右往左数到start的位置，提取该位置及后面的len-1个字符
13 eg:
14 ${str:0-5:4} ----> u.co
15 4. ${str:0-start}
16 从右往左数到start的位置，提取该位置及后面的所有字符
17 eg:
18 ${str:0-5} --> u.com
19 查找子串
20 5. ${str#*sub}
21 从左往右数，提取第一个子串后面的所有字符
22 eg:
23 ${str#*bai} ----> du.com www.baidu.com
```

```

24 6.${str##*sub}
25 从左往右提取最后一个子串后面的所有字符
26 eg:
27 ${str##*bai} ---->du.com
28 7.${str%sub*}
29 从右往左数，提取第一个子串前面的所有内容
30 eg:
31 ${str%bai*} --->www.baidu.com www.
32 8.${str%%sub*}
33 从右往左数，提取最后一个子串前面的所有内容
34 eg:
35 ${str%%bai*} ---> www.
36

```

### 3.注释

```

1 # 单行注释
2 :<<!
3 注释的内容
4 !
5
6 :<<"EOF"
7 注释的内容
8 EOF
9

```

### 4.数组

#### 4.1 数组的初始化

```

1 数组只支持一维数组，里面元素依旧是字符串
2 1.根据下标依次赋值
3 arr=(1 2 3 4 5)
4 2.不按照顺序赋值
5 arr=([0]=10 [2]=20 [3]=30 [5]=50 [10]=100)
6
7 没有被赋值到的元素值为空；数组的长度只会统计有效的长度为 5
8

```

#### 4.2 数组的引用

```

1  ${数组名[下标]} 下标依旧是从0开始
2  eg:
3      arr[1] 表示数组的第二个元素
4      ${arr[1]} 表示数组的第二个元素的值
5      ${arr[@]} 和 ${arr[*]} 表示数组的所有元素
6
7      arr[1]=100 //对数组的第二个元素赋值
8

```

### 4.3 求数组的长度

```

1  数组的长度:${#arr[@]} ${#arr[*]}
2
3  单个元素的长度: ${#arr[下标]}

```

### 4.4

```

1  在数组的后面追加赋值
2      arr=(${arr[@]} "AAA")
3
4  在数组的前面插入赋值
5      arr=("AAA" ${arr[@]})
6  注意:
7      顺序初始化和跳跃初始化, 在数组的整体前后赋值,
8      数组里的元素都会重新排列, 有空格的字符串会被
9      拆分开作为多个元素, 下标是跳跃的也会变得有顺序;

```

### 4.5 取消赋值

```

1  unset arr[下标]
2  unset arr

```

## day5

---

### 1.read 输入

```

1  read 变量1 变量2 ....
2      //当输入多个字符串的时候, 遇到空格就会分割字符串
3  read -p "提示的内容" 变量
4  read -a 数组名 //输入的数据依次对数组赋值
5  read -n num 变量 //输入的字符个数等于num就自动结束输入
6  read -s 变量 静默输入 //比如在输入密码的时候不会显示输入的字符
7  read -t 秒数 变量 //在规定时间内正常输入, 否则结束输入

```

### 2.shell运算

```

1 shell 变量保存的是字符串，要进行运算需要加特定的符号；
2   + - * / %
3   += -= *= /=
4   > < == !=
5   && || !
6   & | ~ ^ >> <<

```

## 2.1 (())

```

1 1.支持整数的运算，不支持浮点数的运算；
2 2.在运算的时候，取变量的值可以加$也可以不加
3 3.可以将计算结果整体赋值给一个变量，需要加$
4 eg:
5     ret=$((var1 + var2))
6     $? 表示上一条语句的执行情况，执行成功返回0，失败返回非0
7 4.可以像C语言一样进行复杂的运算；
8 5.((表达式1, 表达式2, 表达式3))
9     以表达式3作为整个表达式的运算结果

```

## 2.2 \$[ ]

```

1 1.支持整数运算，不支持浮点运算
2 2.在运算的时候，取变量的内容可以加$也可以不加
3 3.运算结果一定要赋值给一个变量保存
4 4.基本上和(( ))用法一样
5 5.ret=$[表达式1, 表达式2, 表达式3]
6     //以表达式3作为整个表达式的运算结果

```

## 2.3 expr

```

1 1.支持整数运算不支持浮点运算
2 2.在运算的时候需要加上空格
3 eg:
4     expr $var1 + $var2
5 3.引用变量的时候必须加上$
6 4.expr 可以输出结果
7 5.如果需要将结果赋值给一个变量，那么需要命令置换
8     eg:
9     var=`expr $var1 + $var2`
10 6.expr 的算术运算
11     + - \* /
12     关系运算
13     \> \< \>= \<= ==
14     如果需要加上括号运算，括号需要转义
15     \ ( \)
16 7.expr 不能进行幂运算

```

```

1  expr 处理字符串
2      str=hello
3      1.匹配字符串
4      expr match $str "string"
5      从第一个字符开始匹配，如果第一个不一样返回0，
6          第一个一样返回一样的个数
7      2.提取字符
8      expr substr $str start len
9      从第start 的位置开始提取len个字符
10     3.求字符串的长度
11     expr length $str

```

### 3.shell中的选择结构

```

1  if 判断式
2  then
3      语句块;
4  fi
5
6  if 判断式
7  then
8      语句块1
9  else
10     语句块2
11  fi
12
13  阶梯结构
14  if 判断式1
15  then
16     语句块1;
17  elif 判断式2
18  then
19     语句块2;
20     .....
21  elif 判断式n
22  then
23     语句块n;
24  else
25     语句块n+1
26  fi
27
28  嵌套:
29      if 判断式1
30      then
31          if 判断式2
32          then
33              语句3 //判断式1 和 判断式2 都成立执行语句3
34          else
35              语句2 //判断式1成立， 判断式2不成立
36          fi
37

```

```
38     else
39         语句块1 //判断式1不成立
40     fi
```

## 4. shell 中的判断式

```
1  第一种：
2      [ 判断式 ] [ 和 ] 左右必须有空格；
3  第二种：
4      test 判断式 test 左右也必须有空格
```

## 5. shell整数的判断

```
1  -gt 大于
2  -ge 大于等于
3  -lt 小于
4  -le 小于等于
5  -eq 等于
6  -ne 不等于
```

## 6.shell 与 或非

```
1  &&  -a
2  ||  -o
3  !    !
```

```
1  1.输入一个成绩输出对应的等级；
2  2.输入一个年份判断是否是闰年
3      （能被4整除但不能被100整除 或者 能被400整除）
```

## 7.结束脚本

```
1  exit 0 正常结束
2  exit 1 不正常结束
3
4  重新执行此脚本
5      bash $0
```

## 8 shell 字符串的判断

```

1  -z 判断字符串是否为空
2  -n 判断字符串不为空
3  = 判断字符串相等
4  != 判断字符串不等
5  /> 判断字符串大于
6  /< 判断字符串小于
7      对字符串的判断，在引用字符串内容的时候加上引号；
8  eg:
9      "$str"
10 eg:
11     从终端输入两个字符串，比较两个字符串的大小

```

## 9.shell 对文件的判断

### 9.1 对文件类型的判断

```

1  -f 判断文件是否存在并判断是否为普通文件
2  -d 判断文件是否存在，并判断是否为目录文件
3  -L 判断文件是否存在，并判断是否为链接文件
4  -b 判断文件是否存在，并判断是否为块设备文件
5  -c 判断文件是否存在，并判断是否为字符设备
6  -p 判断文件是否存在，并判断是否为管道文件
7  -s 判断文件是否存在，并判断是否为套接字文件
8  -s 判断文件是否存在，文件长度大于0为真
9  -e 判断文件是否存在

```

```

1  输入一个文件判断类型

```

### 9.2对文件权限的判断

```

1  -w 判断文件是否具备写权限
2  -r 判断文件是否具备读权限
3  -x 判断文件是否具备执行权限
4
5  eg:
6      输入一个文件，判断文件是否存在，如果存在那么再判断文件是否具备
7      写权限，如果写权限就往这个文件写入一句“hello world”，如果
8      不具备写权限，那么赋上写权限再写入hello world;

```

### 9.3 对文件时间戳的判断

```

1  -nt 判断文件更新
2  -ot 判断文件更旧

```

## 10 case ....in ....esac

```

1  case 表达式 in
2      模式1)

```

```

3      shell 语句块1
4      ;;
5      模式2)
6      shell 语句块2
7      ;;
8      .....
9      *)
10     shell 表达式n;
11     ;;
12 esac
13
14 注意:模式可以通配
15     * 通配所有, 以上模式和表达式都不匹配就执行*下面的内容
16     [a-zA-Z]通配a-z 或者A-Z
17     "str1" | "str2" | .... 通配str1或者 .....strn
18
19     请输入一个软件名, 输入yes 下载, 输入no拒绝下载

```

## day6

---

### 1.循环结构

#### 1.1 while

```

1  while 判断式 //表达式成立执行循环体
2  do
3      shell 语句
4  done
5
6  辅助控制语句
7  break 跳出当前循环
8  continue 结束本次循环继续下一次循环
9
10 实现从1加到100求和

```

#### 1.2 until

```

1  while 判断式 //表达式不成立执行循环体
2  do
3      shell 语句
4  done

```

#### 1.3 for循环

```

1  1.一般用法
2  //      初值      判断      递增递减

```



```

3   for((表达式1; 表达式2; 表达式3))
4   do
5       循环体
6   done
7  2. 常规用法
8   for 变量 in 单词列表
9   do
10      循环体
11  done
12
13
14  1. 散列的单词列表
15  2. 连续的单词列表 {start..end}
16  3. 把一个命令经过命令置换作为单词列表
17
18
19  eg:
20  <<!
21  for var in apple huawei sanxing oppo vivo xiaomi meizu
22  do
23      ((i++))
24      echo "i = $i ---> var: $var"
25  done
26  !
27
28  :<<!
29  for var in {1..100}
30  do
31      ((sum += $var))
32  done
33  echo "sum = $sum"
34  !
35
36  for var in `ls $1`
37  do
38      ((i++))
39      echo "$var"
40  done
41
42  echo "i = $i"
43
44
45  练习:
46  1. 输入两个目录名称 mydir myfile, 判断这两个目录是否
47     存在, 如果存在输入【yes/no】询问要不要删除,
48     如果删除则再创建出来;
49  2. 再输入一个已经存在的目录, 将这个目录里的所有文件
50     拷贝到myfile中, 所有目录拷贝到mydir
51  3. 统计文件的个数, 目录的个数;

```

## 2. shell 函数

```

1 一般形式
2  function 函数名 () {
3      函数体
4  }
5  shell 函数 function 是shell函数的标配
6  shell 函数没有数据类型
7  shell 函数没有参数说明列表
8  shell 里的变量默认为全局变量
9  shell 中的局部变量 需要加上local
10
11 函数调用:
12  函数名
13  或者
14  函数名 arg1 arg2 ....(参数)
15  $0 $1 $2
16  在函数体内部的位置变量通过调用函数的时候传参;
17  在函数体外部的的位置变量通过命令行参数传递
18  eg:
19  :<<!
20  function add(){
21      ret=$((10+20))
22  }
23
24  add
25
26  echo "ret: $ret"
27  !
28
29  :<<"!"
30  function add(){
31      ret=$((1+2))
32  }
33
34  #add 20 20
35  add $1 $2 # 执行脚本:  bash 08_func.sh 10 20
36
37  echo "ret: $ret"
38  !
39
40
41  function add(){
42      local ret=$((10+20))
43
44      return $ret #返回结果
45  }
46  add #函数调用
47  echo ret: $? #结束返回值, 注意只能接收255以内

```

## 4. C语言

## 4.1 虚拟4G内存

```
1  c语言的本质：内存
2  c语言是如何分配内存的：
3      1.根据数据类型，自动分配内存
4      2.在堆区上手动分配空间
5
6  虚拟4G内存：
7      内核层 3G-4G
8      用户层 0-3G
9      堆区 手动申请 手动释放 (malloc / free)
10     (映射区 、mmap)
11     栈区 自动申请、自动释放
12
13     静态数据区
14         dada 段 全局或者static 修饰的 初始化的变量
15         bss 段 全局或者static 修饰的未被初始化的变量
16         test 段
17         .ro / rodata段 只读 const 修饰、字符串常量
18         const char *p = "hello world"
19
```

## 4.1 类型

```
1  存储类型：
2      自动存储 auto 一般默认为自动存储
3      寄存器存储 register 运行速率快
4      为什么不都定义成寄存器存储？
5          不一定能申请到，寄存器数量是有限制的；
6      外部存储：extern 一般用作外部说明
7      静态存储：static
8          修饰局部变量 没有初始化初值为0
9          修饰局部变量 延长生命周期
10         修饰全局变量或者函数 限制作用域,仅能在本文件使用
11         static 和 extern 不能一起使用；
12     头文件：
13         <include.h> 默认在系统目录下搜索这个头文件
14         "include.h" 默认在当前目录下搜索头文件
15
16     数据类型：
17         基本类型
18             char 1字节
19             short 2字节
20             int 4字节
21             long 8字节
22             longlong 8字节
23             float 4字节
24             double 8字节
25             enum 4字节
26         构造类型
27             数组 结构体 共用体
28         指针类型
```

```

29     char * 、int * 、struct stu *....
30     无类型 void
31

```

## 4.2 const

```

1  1.const 修饰常量吗？
2      const 修饰的是变量
3  2.const 修饰的变量一定不能修改吗？
4      const 修饰的局部变量可以用指针间接修改；
5  3.
6  1) const 数据类型 *指针变量名 ；
7      指针变量不能修改目标的值，指针变量本身的值可以修改；
8  eg:
9      int var1 = 10, var2 = 20;
10     const int *p = &var1;
11     *p = 100; // false read-only
12     p = &var; // true
13 2) 数据类型 * const 指针变量名；
14     指针变量可以修改目标的值，指针变量本身的值不能修改；
15  eg:
16     int var1 = 10, var2 = 20;
17     int * const p = &var1; // 只能初始化
18     p = &var2 // flase
19     *p = 100; // true
20
21 3) const 数据类型 * const 指针变量名；
22     指针变量既不能修改目标的值，指针变量本身的值也不能修改；
23  eg:
24     int var1 = 10, var2 = 20;
25     int * const p = &var1; // 只能初始化
26     p = &var2 // flase
27     *p = 100; // false
28

```

## 4.3 malloc 动态分配内存

```

1  #include <stdlib.h>
2
3  void *malloc(size_t size);
4  {
5      功能： 在堆区上申请一片空间；
6      参数： size: 表示空间的大小
7      返回值：返回空间首地址，void *类型
8      因为工程师可能需要不同类型的空间，所有返回void *类型的地址，
9      可以强制类型转换为其他任意类型的地址；
10     申请失败返回NULL；
11
12 }

```

```

13 void free(void *ptr);
14 {
15     功能：释放空间
16     参数：空间的首地址
17     注意：malloc 申请的空间需要手动释放，如果不释放可能会造成
        宕机
18     把空间释放之后，指向申请的这片空间的指针依旧指向这片空间，需
        要将
19     它指向零地址(待指状态);
20 }
21
22 输入一片数据在堆区上存储;
23
24 什么是段错误?
25     访问了非法内存;
26 如何规避段错误?
27     不能使用野指针;
28     malloc 释放了空间之后将指针指向NULL;

```

## 5.二维数组

```

1  可以由多个一维数组组成:
2      int arr[2][3] = {1, 2, 3, 4, 5, 6}
3      可以看成由 两个一维数组组成;
4      arr[0]
5      arr[1]
6
7      arr[2][3] --> {arr[0], arr[1]}
8  元素的访问:
9      arr[0] == *(arr + 0)
10     arr[1] == *(arr + 1)
11
12     arr[0] 一维数组 ---> {1, 2, 3}
13 元素的访问:
14     *(arr[0] + 0) == 1 == (*(arr + 0) + 0) == **arr
15     *(arr[0] + 1) == 2 == (*(arr + 0) + 1)
16     *(arr[0] + 2) == 3 == (*(arr + 0) + 2)
17
18     第i行的第j列:
19     (*(arr + i) + j)
20     == arr[i][j]
21     == (*(arr + i))[j]
22     == *(arr[i] + j)
23
24

```

## 6.指针访问二维数组

```

1  二级指针可不可以访问二维数组?
2      不能，二维数组的数组名是行指针，二级指针和二维数组名的类型不
        一样;

```

```

3 行指针的一般形式:
4  <存储类型> <数据类型> (*行指针变量名) [常量表达式];
5                                     //列数
6  eg:
7  int arr[2][3] = {0};
8  int (*p)[3] = arr;
9
10 第i行的第j列:
11  (*(p + i) + j)
12  == p[i][j]
13  == (*(p + i))[j]
14  == *(p[i] + j)
15
16  // char * arr[32]
17

```

## day7

---

### 1. 指针数组

```

1 一般形式:
2  <存储类型> <数据类型> * 指针数组名[元素个数];
3  eg:
4  int * parr[5]; //表示一个可以存储5个整型地址的数组
5
6  int var1, var2, var3;
7  parr[0] = &var1;
8  parr[1] = &var2;
9  parr[2] = &var3;
10
11 int *p = &var1, *q = &var2, *k = &var3;
12 parr[0] = p;
13 parr[1] = q;
14 parr[2] = k;
15
16 sizeof(parr) == 8 * 5
17 用指针数组保存二维数组:
18 int a[2][3];
19 parr[0] = a[0];
20 parr[1] = a[1];
21
22 a[i] == *(parr + i)
23 a[i][j] == (*(parr + i) + j)
24
25 parr++// 数组名地址常量
26
27 用二级指针访问指针数组:
28 //int a[32] = {0};

```

```

29      //int *p = a;
30
31      int **p = parr;
32      a[i][j] == (*(p + i) + j)
33
34

```

```

1  思考:
2  char *str = "12345678";
3  char p[5] = {0};
4  short *t = (short *)&p[2];
5  *t = atoi(str + 4); // atoi("12345") == 12345;
6  求:
7      p[0]:0
8      p[1]:0
9      p[2]:0x2e
10     p[3]:0x16
11     p[4]:0
12
13     小端:数据的低位存在低地址, 高位存在高地址
14     大端:数据的高位存在低地址, 低位存在高地址
15

```

## 2.函数

### 2.1 函数的传参

```

1  1.函数的传参:
2      复制传递: 将实参的值赋值给形参, 形参不能修改实参的值
3      地址传递: 将实参的地址赋值给形参, 形参可以修改实参的值;
4      全局变量:

```

### 2.2 数组作为参数传递

```

1  一维数组:
2      ①地址传递:
3          函数原型:
4              void func(int *p, int size);
5              void func(char *p);
6          一般在传递数组的时候, 传递数组的首地址和大小;
7          在传递字符串数组的时候, 可以不用传大小,
8          因为知道首地址就能找到结尾的 '\0';
9      ②复制传递:
10         函数原型:
11             void func(int p[100], int size);
12             void func(int q[], int size);
13             sizeof(p) = ? //8
14             sizeof(q) = ? //8

```

```

15  数组的复制传递个地址传递只是表现形式不同，实质上都是将数组的首地
    址
16  赋值给形参，形参可以修改实参的值；
17
18  二维数组：
19      int arr[3][2] = {1, 2, 3, 4, 5, 6};
20      void func(int (*p)[2], int c);
21  指针数组：
22      int * parr[3] = {arr[0], arr[1], arr[2]};
23      void func(int **p, int c, int l);
24

```

## 2.3 指针函数

```

1  本质： 返回值为一个地址的函数；
2  一般形式：
3      <数据类型> * <函数名>(形式参数说明列表)
4      {
5          函数体；
6
7          return 地址量；
8      }
9  eg:
10     char * strcpy(char *dest, const char *src);
11     返回值返回dest；
12     char str1[32] = "hello";
13     char str2[32] = "world";
14     /*  strcpy(str1, str2);
15         puts(str1);*/
16     puts(strcpy(str1, str2));
17

```

## 2.4 函数指针

```

1  本质： 是一个指针，一个可以指向函数的指针；
2  函数名： 入口地址；
3
4  函数指针的一般形式：
5      数据类型 (*函数指针变量名)(形参);
6      数据类型和形参应该和所指向的函数保持一致；
7  函数指针数组的一般形式：
8      数据类型 (*函数指针变量名[元素个数])(形参);
9
10     int add(int x, int y)
11     {
12         return x + y;
13     }
14
15     int sub(int x, int y)
16     {
17         return x - y;

```



```

18 }
19
20 int testfunc(int x, int y, int (*pfunc)(int, int))
21 {
22
23     return pfunc(x, y);
24 }
25
26 int main()
27 {
28     int a = 10, b = 20;
29     printf("%d\n", add(a, b)); // 30
30     printf("%d\n", sub(a, b)); // -10
31
32     //通过函数指针来访问函数
33     int (*pfunc)(int, int) = add;
34     printf("%d\n", pfunc(a, b)); // 30
35     pfunc = sub;
36     printf("%d\n", pfunc(a, b)); //-10
37     //函数指针数组
38     int (*parrfunc[3])(int, int) = {add, sub};
39     printf("%d\n", parrfunc[0](100, 200)); //300
40     printf("%d\n", parrfunc[1](100, 200)); //-100
41
42     //回调函数
43     printf("%d\n", testfunc(10, 20, add));
44
45     return 0;
46 }
47

```

### 3. gdb 调试

```

1 编译: gcc file.c -g
2 调试:
3     gdb a.out
4
5  l  查看代码默认10行
6  r  运行
7  b  行号 设置断点
8  info b 查看断点
9  c  继续运行
10 n  单行运行
11 p var 查看变量的值
12 q  退出

```

### 4. typedef 和 define

```

1 1.define 宏定义，宏替换 在预处理的阶段进行的；
2 将后面的内容用前面的内容来替换；
3 不需要加分号；

```

```
4 2.typedef 取别名，在编译的阶段进行的；
5 将前面的内容去个别名放在后面；
6 语句结束需要加分号；
7
8 eg:
9     #define INT_t int
10     INT_t var;
11     typedef int int_t;
12
13 eg:
14
15
16 #if 0
17     typedef int * int_p;
18     int var = 10;
19     int_p p = &var;
20     printf("*p = %d\n", *p);
21 #endif
22 #if 0
23     typedef int Arr_t[10];
24     Arr_t arr={0};
25     arr[0] = 100;
26     arr[9] = 200;
27     int i = 0;
28     for(; i < 10; i++){
29         printf("%-4d", arr[i]);
30     }
31     puts("");
32     printf("sizeof(arr) = %ld\n", sizeof(arr));
33
34 #endif
35
36 #if 0
37     typedef int *Parr_t[10];
38     Parr_t parr;
39     printf("sizeof(parr) = %d\n", sizeof(parr));
40
41 #endif
42
43 #if 1
44     typedef int (*Pfunc_t)(int);
45     Pfunc_t pfunc;
46     pfunc = func;
47
48     printf("%d\n", pfunc(10));
49     typedef int (*PfuncArr_t[3])(int);
50     PfuncArr_t pfuncarr = {func};
51     printf("%d\n", pfuncarr[0](20));
52
53 #endif
54
55
```

## 5.枚举

```
1  用于对有序数据的罗列，比如月份、星期；
2  一般形式：
3      enum week{
4          Mon,
5          Tue,
6          Wed,
7          Thu,
8          Fri,
9          Sat,
10         Sun
11     };
12
13     1. 枚举里面的成员是常量；
14     2. 对枚举里面的成员赋值，那么下面的成员会依次递增，
15         如果枚举里的成员没有赋值第一个默认为0；
16     3. 枚举定义的变量用枚举里面的成员来赋值；
17     4. 枚举是基本类型占4个字节；
18     5. 枚举的成员之间用, 隔开；
19
20     enum week day = Sun;
21     //enum week day = 7; // 一般不这样赋值
22
```

## 6.结构体

```
1  1. 结构体属于构造类型，它的成员可以是相同数据类型的集合，
2     也可以是不同数据类型的集合；
3  2. 结构体里面的成员是变量；
4  3. 结构体里面的成员一定是固定大小的成员；
5  4. 结构体的成员之间用 ; 隔开；
6  5. 只有在定义了结构体变量的时候系统才分配空间；
7  6. 结构体变量之间可以直接赋值；
8  7. 结构体的关键字是struct
9  8. 结构体的访问：结构体变量名.成员 如果是指针，结构体指针->成员
```

```
1  结构体的一般形式：
2      struct 结构体名{
3          成员列表1;
4          成员列表2;
5          ....
6      };
7  定义结构体变量：
8      struct 结构体名 结构体变量名；
```

### 6.1 结构体的定义方式

```
1  ① struct STU{
2      char name[32];
```

```

3     int id;
4     float score;
5 };
6
7 ② struct STU{
8     char name[32];
9     int id;
10    float score;
11 }stu1, stu2;
12
13 ③ struct{
14     char name[32];
15     int id;
16     float score;
17 }stu1, stu2; //不能单独定义结构体变量
18
19 ④ typedef struct STU{
20     char name[32];
21     int id;
22     float score;
23 }stu_t;
24
25 ⑤ typedef struct{
26     char name[32];
27     int id;
28     float score;
29 }stu_t;
30

```

## 7.结构体赋值

```

1 ① 结构体变量.成员直接赋值;
2     strcpy(stu.name, "TOM");
3     stu.id = 1;
4     stu.score = 98.5;
5
6 ② 定义结构体变量的时候整体赋值作为初始化;
7     struct STU stu3 = {"xiaoming", 1, 97, 'B'};
8 ③ 对变量赋值的时候, 整体赋值那么需要强转类型换行
9     stu2 = (struct STU){ "xiaofang", 3, 100, 'G'};
10 ④部分赋值:
11     struct STU stu4 = {
12         .name = "小张",
13         .score= 70
14     };
15

```

## 8.结构体数组

```

1 eg:
2     typedef struct{

```

```
3     char name[32];
4     int id;
5     float score;
6     char sex;
7 }stu_t;
8
9 stu_t stu[3];
10 1. 逐个赋值:
11 strcpy(stu[0].name, "TOM")
12
13 2. 完全初始化
14 stu_t stu[3] = {
15     {"小明", 1, 90, 'B'},
16     {"小红", 2, 83, 'B'},
17     {
18         .name = "小芳",
19         .sex = 'G'
20     }
21 };
22 //跳跃赋值
23 3. stu_t stu[3] = {
24     [0] = {"小明", 1, 90, 'B'},
25     [2] = {"小红", 2, 83, 'B'},
26 };
27 作业:
28 定义一个结构体数组, 实现对学生信息的存储以及增删改查;
```