**01-Bash命令**

https://mp.weixin.qq.com/s/suoqkDTZtEJIBfs\_iE0t2Q

1. **bash定义**

1.1 **bash 执行的操作**

预留关键字：for、history–自己操作

命令程序：ls、free–去环境变量中找命令程序交由他们来执行

环境变量PATH中记录了bash 找程序的路径

set -e： 它的作用是让Shell在执行脚本过程中，如果任何一条命令的返回值不是0（表示执行失败），则立即退出脚本，不再继续执行后续的命令

1.2 **shell脚本单步**

-n 只读取shell脚本，但不实际执行

-x 进入跟踪方式，显示所执行的每一条命令

-c "string" 从strings中读取命令

1.3  **目录相关**

~ 与 $HOME相同

Linux下~、/home与/root的区别：

1、在root用户下，~等同于/root

2、在普通用户下，~等同于/home/当前的普通用户名

1.4 **history命令**

记录在.bash\_history

默认记录条数

echo $HISTSIZE

1.5 **命令格式**

1、脚本开头， 告诉系统这个脚本应该通过 Bash

|  |
| --- |
| C++ #!/bin/bash |

1.6 **tmux终端命令**

1.7 **shc命令脚本加密**

shc命令

执行下面的命令，编译 func.sh 脚本

|  |
| --- |
| C++ shc -rf func.sh -o func.bin |

2. **基本命令**

2.1 **正则匹配**

2.1.1 **正则表达式说明**

https://c.biancheng.net/view/5124.html

**正则查找开头和提取部分**

整个正则表达式^-task\_executor\_timemonitor-(f[0-9]+)$会匹配以“-task\_executor\_timemonitor-f”开头，后面跟着至少一个数字，并且是整个字符串的结尾的任意参数

|  |
| --- |
| C++ #!/bin/bash  # 判断参数是否匹配-task\_executor\_timemonitor-前缀 if [[ $1 =~ ^-task\_executor\_timemonitor-(f[0-9]+)$ ]]; then  # 提取匹配的部分 (-f36, -f37, ..., -f41)  case\_arg=${BASH\_REMATCH[1]}  # 如果需要执行带有额外参数的命令，可以这样做  if [[ $case\_arg = "f40" || $case\_arg = "f41" ]]; then  /ota/aarch64\_Linux\_gnu\_release/scheduler/FT/taskexec\_oss3/task\_executor\_timemonitor -case $case\_arg -p  else  /ota/aarch64\_Linux\_gnu\_release/scheduler/FT/taskexec\_oss3/task\_executor\_timemonitor -case $case\_arg  fi else  echo "The parameter does not match the expected format." fi |

* ~：这是Bash中执行正则表达式匹配操作的操作符。
* ^-task\_executor\_timemonitor-：这部分指定了字符串的开头（^是正则表达式中匹配开头的特殊字符）。接下来的-task\_executor\_timemonitor-是要检查的确切字符序列，表明参数必须以此序列开始。
* (f[0-9]+)：这是一个捕获组，用于从匹配到的表达式中提取信息。 - f：参数中紧跟-task\_executor\_timemonitor-必须有一个小写的“f”字符。 - [0-9]：这表示数字0到9中的任意一个数字。 - +：这表明前面的数字（[0-9]）可以连续出现一次或多次。这意味着“f”后面可以跟随一个或多个数字。
* $：这表示字符串的结尾。确保字符串在匹配成功后立即结束，没有额外的字符。

BASH\_REMATCH数组中将存储匹配的结果。BASH\_REMATCH[0]包含整个匹配的字符串，而BASH\_REMATCH[1]则包含第一个捕获组匹配的字符串，即“f”后面跟随的数字（例如"f36"）

**是否存在特定子字符串**

|  |
| --- |
| C++ if [[ $1 == \*task\_executor\_performance\* ]]; then  echo "1" elif [[ $1 =~ ^-task\_executor\_timemonitor-(f[0-9]+)$ ]]; then  # Rest of your code goes here fi |

**以特定字符串开头并去除特定字符串执行**

|  |
| --- |
| C++ if [ $1 == task\_executor\_performance\* ]; then  argument=${1#task\_executor\_performance}  /ota/aarch64\_Linux\_gnu\_release/scheduler/PT/taskexec\_oss3/task\_executor\_performance $argument & |

2.2 **xargs命令**

command 的输出会被作为输入传递给 xargs，-I 选项后面跟随的 {} 是替换字符串。每次 xargs 读取一个输入值，它都会用这个值来替换 another-command 中出现的 {}，然后执行 another-command。

|  |
| --- |
| C++ command | xargs -I {} another-command {} |

2.3 **管道、重定向**

重定向输出是将程序的标准输出（stdout）从显示在终端上改为输出到文件或其他设备。

重定向输入是将程序的标准输入（stdin）从键盘输入改为从文件中读取输入数据。

my\_program为一个程序，需要输入或者输出命令

|  |  |
| --- | --- |
| | 管道 | A | B 将A命令输出作为B命令的输入 |
| >重定向输出 | ps > file.txt |
| >>追加到末尾 |  |
| < 重定向输入 | my\_program < file.tx |

2.4 **普通命令**

2.4.1 **文件表达式**

序号 参数 说明

1 -e filename  如果 filename存在，则为真

2 -d filename 如果 filename为目录，则为真

3 -f filename 如果 filename为常规文件，则为真

4 -L filename 如果 filename为符号链接，则为真

5 -r filename 如果 filename可读，则为真

6 -w filename 如果 filename可写，则为真

7 -x filename 如果 filename可执行，则为真

8 -s filename 如果文件长度不为0，则为真

9 -h filename 如果文件是软链接，则为真

10 filename1 -nt filename2 如果 filename1比 filename2新，则为真

11 filename1 -ot filename2 如果 filename1比 filename2旧，则为真

2.4.2 **整数变量表达式**

序号 参数 说明

1 -eq 等于（equal）

2 -ne  不等于（not equal）

3 -gt  大于（greater than）

4 -ge 大于等于（greater than or equal）

5 -lt  小于（less than）

6 -le 小于等于（less than or equal）

2.4.3 **字符串变量表达式**

序号 参数 说明

1 $a = $b 如果string1等于string2，则为真

2 $string1 !=  $string2 如果string1不等于string2，则为真

3 -n $string 如果string 非空(非0），返回0(true)  
4 -z $string 如果string 为空，则为真

5 $string 如果string 非空，返回0 (和-n类似)  
6 ! 表达式 条件表达式的相反[逻辑非]  
7 表达式1  –a  表达式2 条件表达式的并列[逻辑与]  
8 表达式1  –o 表达式2 条件表达式的或[逻辑或]  
样例：  
[ -z "$pid" ] 单对中括号变量必须要加双引号

[[ -z $pid ]] 双对括号，变量不用加双引号  
 

2.4.4 **特殊变量**

序号 参数 说明  
1 $0 当前脚本的文件名

2 $n 传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是1  
3 $# 传递给脚本或函数的参数个数

4 $\* 传递给脚本或函数的所有参数  
5 $@ 传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $\* 稍有不同  
6 $? 上个命令的退出状态，或函数的返回值

7 $$ 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID  
8 $! Shell最后运行的后台Process的PID(后台运行的最后一个进程的进程ID号)

参考：Linux的shell中 if 的-e,-d,-f的说明

2.4.5 **文本操作**

2.4.5.1 **awk命令**

|  |
| --- |
| C++ awk 'pattern { action }' file |

pattern：是一个模式，用于匹配文本行。

action：是在模式匹配时执行的操作。

file：是要处理的输入文件。

|  |
| --- |
| C++ your\_command | awk -F 'total duration=' '{ if ($2 > 1.5) print $0 }' |

* your\_command 是产生原始输出的命令，你需要将其替换为你实际使用的命令。
* | 是管道，它会将前一个命令的输出传输到后面的命令。
* awk 是文本处理工具，我们使用它来筛选和打印数据。
* -F 'total duration=' 设置awk的字段分隔符为 total duration=。
* ' { if ($2 > 1.5) print $0 }' 是awk的动作，它会检查第二个字段（$2——数字1.5之后的部分）是否大于1.5，如果是则打印整行（$0）。

|  |
| --- |
| C++ # 打印每一行的第一个字段 awk '{print $1}' input\_file  # 分隔符改为冒号，打印每一行的第二个字段 awk -F':' '{print $2}' input\_file  # 只打印包含 "pattern" 的行 awk '/pattern/' input\_file  # 模式匹配，行号大于2的行，打印第一个字段和第三个字段 awk 'NR > 2 {print $1, $3}' input\_file  # 以逗号分隔输出，打印第二和第四字段 awk -F, '{print $2, $4}' input\_file  # 使用 BEGIN 和 END 块，分别在处理输入之前和之后执行动作 awk 'BEGIN {print "Start Processing."} {print $0} END {print "End Processing."}' input\_fi |

参数

$0 整行， $1、2、3...n后面的第n个分段

2.4.5.2 **sed命令**

2.4.5.3 **expr命令**

2.4.6 **查找命令**

|  |  |
| --- | --- |
| **grep命令** |  |
|  | grep命令 grep -option（参数） ‘word’（关键词） file（文本文件）  查字符 grep -r "xxx" file  查进程 ps -e | grep xxxx  grep参数： 过滤 ifconfig |grep -w inet |
|  | -a 不要忽略二进制数据。  -A <显示列数> 除了显示符合范本样式的那一行之外，并显示该行之后的内容。  -b 在显示符合范本样式的那一行之外，并显示该行之前的内容。  -c 计算符合范本样式的列数。  -C <显示列数>或-<显示列数> 除了显示符合范本样式的那一列之外，并显示该列之前后的内容。  -d <进行动作> 当指定要查找的是目录而非文件时，必须使用这项参数，否则grep命令将回报信息并停止动作。  -e <范本样式> 指定字符串作为查找文件内容的范本样式。  -E 将范本样式为延伸的普通表示法来使用，意味着使用能使用扩展正则表达式。  -f <范本文件> 指定范本文件，其内容有一个或多个范本样式，让grep查找符合范本条件的文件内容，格式为每一列的范本样式。  -F 将范本样式视为固定字符串的列表。  -G 将范本样式视为普通的表示法来使用。  -h 在显示符合范本样式的那一列之前，不标示该列所属的文件名称。  -H 在显示符合范本样式的那一列之前，标示该列的文件名称。  -i 忽略字符大小写的差别。  -l 列出文件内容符合指定的范本样式的文件名称。  -L 列出文件内容不符合指定的范本样式的文件名称。  -n 在显示符合范本样式的那一列之前，标示出该列的编号。  -q 不显示任何信息。  -R/-r 此参数的效果和指定“-d recurse”参数相同。  -s 不显示错误信息。  -v 反转查找。  -w 只显示全字符合的列。  -x 只显示全列符合的列。  -y 此参数效果跟“-i”相同。  -o 只输出文件中匹配到的部分。 |
| **find命令** |  |
|  | find命令 find 目录 -选项 动作[-print -exec -ok ...]  找文件 find / -name "xxx"  常用选项： -newer file1 ! newer file2 查找更改时间比文件file1新但比文件file2旧的文件 |
|  | -a: and 必须满足两个条件才显示  -o: or 只要满足一个条件就显示  -name： 按照文件名查找文件  -iname： 按照文件名查找文件(忽略大小写)  -type： 根据文件类型进行搜索  -perm： 按照文件权限来查找文件  -user 按照文件属主来查找文件。  -group 按照文件所属的组来查找文件。  -fprint 文件名：将匹配的文件输出到文件。  常用动作：  -print 默认动作，将匹配的文件输出到标准输出  -exec 对匹配的文件执行该参数所给出的命令。相应命令的形式为 'command' { } \;，注意{ }和\；之间的空格。  -ok 和-exec的作用相同，只不过以一种更为安全的模式来执行该参数所给出的命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。  -delete 将匹配到的文件删除 |

2.4.7 **压缩命令**

|  |  |
| --- | --- |
| tar.gz |  |
| 压缩 | tar -zcvf xxx.tar.gz xxx |
| 解压 | tar -zxvf xxx.tar.gz |
| tar.xz |  |
| 压缩 | xz -z filename |
| 解压 | xz -d filename |
|  | tar -xvJf xxx.tar.xz |
| zip命令 |  |
| 解压 | unzip file.zip |
| 压缩 | zip -r file.zip file |
| gzip命令 |  |
| 压缩 | gzip file |
| 解压 | gzip -dv file.gz |

3. **Linux 常用命令**

3.1 **常用命令**

|  |  |
| --- | --- |
| ls | 查看目录与文件  #显示当前目录下所有文件的详细信息： ls -la |
| mkdir | 创建目录  #在当前目录下创建 test 目录： mkdir test  #在/opt/test 目录下创建目录 img， 若无 test 目录， 先创建 test 目录： mkdir -p/opt/test/img |
| cat | 查看文件内容  #查看 desc.txt 的内容： cat desc.txt |
| more | 分页查看文件内容  #分页查看 desc.txt 文件的内容： more desc.txt |
| tail | 查看文件尾部内容  #查看 desc.txt 的最后 100 行内容： tail -100 desc.txt |
| cp | 拷贝命令  #拷贝 desc.txt 文件到/mnt 目录下： cp desc.txt /mnt/  #拷贝 test 目录到/mnt 目录下： cp -r test /mnt  这里注意拷贝目录时， 通过-r 指明一下 |
| mv | 剪切或者改名  #剪切文件 desc.txt 到目录/mnt 下： mv desc.txt /mnt/  这里注意， 这个 Linux 命令有两种用法， 剪切和改名  1、 当源文件所在的目录跟目标目录不同时， 就是剪切操作；  2、 当源文件所在目录跟目标目录相同时， 就是改名操作。 |
| rm | 删除命令  #删除 test 目录， -r 递归删除， -f 强制删除。 数据无价， 删除有风险，删前需谨慎， 切记！： rm -rf test |
| find | 搜索文件命令  #在 opt 目录下查找以.txt 结尾的文件： find /opt -name '.txt' |
| ln | 创建链接文件  #创建目录/opt/test 的符号链接： ln -s /opt/test ./link2test |
| man | 命令帮助  #对你不熟悉的命令提供帮助和解释： man ls 就可以查看 ls 相关的用法 |
| cd | 改变当前目录  #进入 netseek 目录： cd netseek |
| pwd | 查看当前所在目录完整路径： pwd |
| shutdow n | 关机或重启命令  #立刻关机： shutdown -h now  #60s 后重启： shutdown -r -t 60 |
| netstat | 显示网络相关信息  #列出所有端口： netstat -a |
| du | 查看目录使用情况  #查看/opt/test 目录的磁盘使用情况： du -h /opt/test |
| top | 显示系统当前进程信息 |
| kill | 杀死进程  #杀死进程号为 27810 的进程， 强制终止， 系统资源无法回收： kill -s 9 27810 |
| chmod | 改变文件或目录的访问权限  #权限范围： u(拥有者)g(群组)o(其他用户)， 权限代号： r(读权限)w(写权限)x(执行权限)#给文件拥有者增加 test.sh 的执行权限： chmod u+x test.sh  #给文件拥有者增加 test 目录及其下所有文件的执行权限： chmod u+x -R test |
| tar + vf | 压缩和解压缩  #打包 test 目录为 test.tar.gz 文件， -z 表示用 gzip 压缩： tar -zcvf test.tar.gz ./test  #解压 test.tar.gz 文件： tar -zxvf test.tar.gz |
| vim | 文本编辑  #vim 三种模式： 命令模式、 插入模式、 编辑模式。 使用 ESC 或 i 或： 来切换模式  #命令模式下，  :q 退出 :q!强制退出 :wq 保存退出  :set number 显示行号  /java 在文档中查找 java  yy 复制 p 粘贴  #编辑 desc.txt 文件： vim desc.txt |

3.2  **其他命令**

|  |  |
| --- | --- |
| clear 命令 | 清屏， 相当与 DOS 下的 cls |
| date 命令 | 显示当前时间 |
| mount命令 | 加载一个硬件设备  mount [参数] 要加载的设备 载入点  mount /dev/cdrom  cd /mnt/cdrom //进入光盘目录 |
| su命令 | 在不退出登陆的情况下， 切换到另外一个人的身份  su -l 用户名(如果用户名缺省， 则切换到 root 状态)su -l netseek (切换到 netseek 这个用户， 将提示输入密码) |
| whoami，  whereis，  which， id | //whoami:确认自己身份  //whereis:查询命令所在目录以及帮助文档所在目录  whereis bin 显示 bin 所在的目录， 将显示为： /usr/local/bin  //which:查询该命令所在目录(类似 whereis)  //id:打印出自己的 UID 以及 GID。 (UID:用户身份唯一标识。 GID:用户组身份唯一标识。每一个用户只能有一个唯一的 UID 和 GID) |
| grep | 命令该命令常用于分析一行的信息， 若当中有我们所需要的信息， 就将该行显示出来， 该命令通常与管道命令一起使用， 用于对一些命令的输出进行筛选加工等等， 它的简单语法为  grep:文本内容搜索;  grep success \* /\*查找当前目录下面所有文件里面含有 success 字符的文件 |
| passwd命令 | 可以设置口令 |
| history 命令 | 可以显示用户过去使用的命令 |
| finger命令 | 可以让使用者查询一些其他使用者的资料  finger root //查看 root 的资料 |
| file | 命令该命令用于判断接在 file 命令后的文件的基本数据， 因为在 Linux 下文件的类型并不是以后缀为分的， 所以这个命令对我们来说就很有用了， 它的用法非常简单， 基本语法如下：  file filename |
| gcc | 命令对于一个用 Linux 开发 C 程序的人来说， 这个命令就非常重要了， 它用于把 C 语言的源程序文件， 编译成可执行程序， 由于 g++的很多参数跟它非常相似， 所以这里只介绍 gcc 的参数， 它的常用参数如下：  -o ： output 之意， 用于指定生成一个可执行文件的文件名  -c ： 用于把源文件生成目标文件(.o)， 并阻止编译器创建一个完整的程序-I ： 增加编译时搜索头文件的路径  -L ： 增加编译时搜索静态连接库的路径  -S ： 把源文件生成汇编代码文件  -lm： 表示标准库的目录中名为 libm.a 的函数库  -lpthread ： 连接 NPTL 实现的线程库  -std= ： 用于指定把使用的 C 语言的版本  例如：  把源文件 test.c 按照 c99 标准编译成可执行程序 test  gcc -o test test.c -lm -std=c99  #把源文件 test.c 转换为相应的汇编程序源文件 test.s  gcc -S test.c |

3.3 **编译命令**

|  |  |
| --- | --- |
| file 命令：file 命令可用于查看文件的类型，包括执行文件和动态库 | file your\_file |
| ldd 命令：ldd 命令用于查看动态共享库的依赖关系，即哪些库被一个可执行文件或其他动态库所使用 | ldd your\_executable\_or\_shared\_library |
| readelf 命令：readelf 命令是 GNU 工具链的一部分，它可以用来查看 ELF（可执行和链接格式）文件的详细信息，包括头部、段、节、符号表等 | readelf -h your\_file # 显示文件头  readelf -S your\_file # 显示节头  readelf -s your\_file # 显示符号表 |
| objdump 命令：objdump 命令也是 GNU 工具链的一部分，它可用于反汇编可执行文件和动态库，以及查看符号表等 | objdump -d your\_file # 反汇编文件  objdump -t your\_file # 显示符号表 |
| nm 命令：nm 命令用于查看 ELF 文件的符号表，它列出了文件中定义的符号 | nm your\_file |
| strace 命令：strace 命令可用于跟踪执行文件或动态库的系统调用，以帮助调试和分析应用程序的行为 | strace your\_executable\_or\_shared\_library |
| ldconfig 命令：ldconfig 命令用于配置系统的动态链接器和运行时连接器缓存，以查找和管理动态共享库 | ldconfig -v -N -X |

3.3.1 **可执行文件命令**

3.3.1.1 **Objdump工具**

|  |
| --- |
| C++ objdump -s -d /bin/bash |

3.3.1.2 **Readelf工具**

|  |
| --- |
| C++ readelf -h /bin/bash |

3.3.1.3 **c++filt工具**

c++filt 分析c++函数名称修饰

|  |
| --- |
| C++ c++filt \_ZN1N1C4funcEi N::C::func(int) |