# shell脚本学习笔记

## shell常见问题总结(自用，后有基础知识)

**1) 如何向脚本传递参数？**

`./script argument` `bash script argument`

**2) 如何在脚本中使用参数？**

第一个参数：`$1`，第二个参数：`$2`，第n个参数：`$n`

**3) 如何计算传递进来的参数个数？**

`$#`

**4) 如何在脚本中获取脚本名称？**

`$0`：

**5) 如何检查之前的命令是否运行成功？**

`$?`成功则返回0否则返回其他值

**6) 如何获取文件的最后一行？**

`tail -1 file–name`获取从后往前第n行：`tail -n file-name`

**7) 如何获取文件的第一行？**

`head -1 file-name`获取从前往后第n行：`head -n file-name`

**8) 如何获取一个文件每一行的第三个元素？**

`awk '{print $3}' 'filenames'` 获取每一行第n个元素`awk '{print $n}' 'filenames'`注意只能用在#!/bin/awk中

**9) 假如文件中每行第一个元素是 FIND，如何获取第二个元素**

`awk'{ if ($1 == "FIND") print $2}'`

**10) 如何调试 bash 脚本**

将 `-xv` 参数加到 `#!/bin/bash` 后

例子：

`#!/bin/bash –xv`

**11) 举例如何写一个函数？**

```
function name {

}
name() {

}
function name() {

}
```

**12) 如何向连接两个字符串？**

例：

```
string1="Just for "
string2="fun"
string3=${string1}${string2}
echo $string3
```

输出`Just for fun`

**13) 如何进行两个整数相加？**

```
a=1
b=2
let c=$a+$b
(( d=a+b ));
echo $c
echo $d
echo $(($a+$b))
echo $[$a+$b]
expr $a+$b
echo $a+$b | e
```

**14) 如何检查文件系统中是否存在某个文件？**

```
if [ -f ~/file-way/file-name ] #判断文件系统中是否存在某个文件
if [ -d ~/file-way/file-name ] #判断目录是否存在
if [ -e ~/file-way/file-name ] #判断是否存在
if [ -r ~/file-way/file-name ] #判断是否可读
if [ -x ~/file-way/file-name ] #判断是否可执行
```

**15) 写出 shell 脚本中常用语法？**

**if判断：**

```
if [ test ]
then
    #statements
fi

if [[ condition ]]; then
    #statements
fi

if [[ condition ]]; then
    #statements
    else
        #statements
fi

if [[ condition ]]; then
    #statementts
elif [[ condition ]]; then
    #statements
elif [[ condition ]]; then
    #statements
    else
    #statements
fi
```

**for 循环：**

```
for word in `ls`; do
done
```

**while 循环：**

```
while [[ condition ]]; do
    #statements
done
```

**until 循环：**

```
until [[ condition ]]; do
    #statements
done
```

**流程控制CASE：**

```
case word in
    pattern)
        ::
esac
```

**16) 每个脚本开始的 #!/bin/sh 或 #!/bin/bash 表示什么意思？**

这一行说明要使用的 `shell`。`#!/bin/bash` 表示脚本使用 `/bin/bash`。对于 python 脚本，就是 `#!/usr/bin/python`。

**17) 如何获取文本文件的第 10 行？**

`head -10 file|tail -1`

**18)chomd作用?**

`chmod a+x file`给予文件可执行权限

**19) 数组操作**

`declare -a a` `name[subscript]=value` `name=(value1 value2 ...)`创建数组

输出数组内容:

`${arry[*]}` `${arry[@]}`

确定数组元素个数:

`${#arry[@]}`

数组追加:

`arry+=(a b c)`

数组排序:

`sort`

删除数组与元素:

`unset`

**20) 命令 “export” 有什么用？**

使变量在子 `shell` 中可用。

**21) 如何在后台运行脚本？**

在脚本后面添加 `“&”`。

**22) "chmod 500 script" 做什么？**

使脚本所有者拥有可执行权限。

**23) ">" 做什么？**

重定向输出流到文件或另一个流。 `>>`是追加不覆盖的重定向

**24) & 和 && 有什么区别**

- `&` - 希望脚本在后台运行的时候使用它
- `&&` - 当前一个脚本成功完成才执行后面的命令/脚本的时候使用它

**25) 什么时候要在 [ condition ] 之前使用 “if”？**

当条件满足时需要运行多条命令的时候。例：

```
if [ condition ] && [ condition ]
then
fi
```

**26) ";" 表示什么?**

`command1;command2` 顺序执行命令

**27) bash shell 脚本中哪个符号用于注释？**

`#`

**28) 命令: echo ${new:-variable} 的输出是什么**

`variable`

**29) ' 和 " 引号有什么区别？**

- `'` - 当我们不希望把变量转换为值的时候使用它。
- `"` - 会计算所有变量的值并用值代替。

**30) 如何在脚本文件中重定向标准输出和标准错误流到 log.txt 文件？**

在脚本文件中添加 `"exec >log.txt 2>&1"` 命令。

**31) 如何只用 echo 命令获取字符串变量的一部分？**

```
echo ${variable:x:y} #x是起始位置，y是截取的长度。本句句意为从左侧第x位开始截取y位
```

**32) 如果给定字符串 variable="User:123:321:/home/dir"，如何只用 echo 命令获取 home_dir？**

`echo ${variable#*:*:*:}` 输出变量从#后内容后从前向后开始输出

或

`echo ${variable##*:}` 输出变量从##后内容不重复的那个开始从前向后输出

**33) 如何从上面的字符串中获取 “User”？**

`echo ${variable%:*:*:*}` 输出变量从%后内容从开始从后向前输出

或

`echo ${variable%%:*}` 输出变量从%%后内容从不重复的那个开始从后向前输出

**34) 如何使用 awk 列出 UID 小于 100 的用户？**

`awk -F: '$3<100' /etc/passwd`

**35)5**36) 如何在 bash shell 中更改标准的域分隔符为 ":" ?**

`IFS=":"`

**37) 如何获取变量长度？**

`${#variable}`

**38) 如何打印变量的最后 5 个字符？**

`echo ${variable: -5}`注意程序报红但不是错误

**39) `${variable:-10}`和 `${variable: -10}` 有什么区别?**

- `${variable:-10}` - 如果之前没有给 variable 赋值则输出 10；如果有赋值则输出该变量
- `${variable: -10}` - 输出 variable 的最后 10 个字符

**40) 如何只用 echo 命令替换字符串的一部分？**

`echo ${variable//pattern/replacement}`

**41) 哪个命令将命令替换为大写？**

`tr '[:lower:]' '[:upper:]'`

**42) 如何计算本地用户数目？**

`wc -l /etc/passwd|cut -d" " -f1 或者 cat /etc/passwd|wc -l`

**43) 不用 wc 命令如何计算字符串中的单词数目？**

```
set ${string}
echo $#
```

**44) export**

设置或显示环境变量, export可新增，修改或删除环境变量，供后续执行的程序使用。export的效力仅及于该次登录操作

export PATH=”$PATH:/home/user/bin” 增加 export -p 显示shell赋予程序的环境变量

**45) 如何列出第二个字母是 a 或 b 的文件？**

`ls -d [ab]*`

**46) 如何将整数 a 加到 b 并赋值给 c？**

`c=$((a+b))`

或

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

或

`c=`echo "$a+$b"|bc

**47) 如何去除字符串中的所有空格？**

```
echo $string|tr -d " "
```

**48) 重写这个命令，将输出变量转换为复数: item="car"; echo "I like $item" ?**

```
item="car"; echo "I like ${item}s"
```

**49) 写出输出数字 0 到 100 中 3 的倍数(0 3 6 9 ...)的命令？**

```
for i in {0..100..3}; do echo $i; done
```

或

```
for (( i=0; i<=100; i=i+3 )); do echo "Welcome $i times"; done
```

**50) 如何打印传递给脚本的所有参数？**

```
echo $*
```

或

```
echo $@
```

**51) `[ $a == $b ]` 和 `[ $a -eq $b ]` 有什么区别**

- `[ $a == $b ]` - 用于字符串比较
- `[ $a -eq $b ]` - 用于数字比较

其他可以用man手册命令`man test`查看

**52) = 和 == 有什么区别**

- `=` - 用于为变量赋值
- `==` - 用于字符串比较

**53) 写出测试 $a 是否大于 12 的命令？**

```
[ $a -gt 12 ]
```

**54) 写出测试 $b 是否小于等于 12 的命令？**

```
[ $b -le 12 ]
```

**55) 如何检查字符串是否以字母 "abc" 开头？**

```
[[ $string == abc* ]]
```

**56) `[[ $string == abc* ]]` 和 `[[ $string == "abc*" ]]` 有什么区别**

- `[[ $string == abc* ]]` - 检查字符串是否以字母 abc 开头
- `[[ $string == "abc" ]]` - 检查字符串是否完全等于 abc

**57) 如何列出以 ab 或 xy 开头的用户名？**

```
egrep "^ab|^xy" /etc/passwd|cut -d: -f1
```

**58) bash 中 $! 表示什么意思？**

后台最近执行命令的 PID.

**59) $? 表示什么意思？**

上一个命令是否执行成功

**60) 如何输出当前 shell 的 PID？**

```
echo $$
```

**61) 如何获取传递给脚本的参数数目？**

```
echo $#
```

**62) `$\*` 与 `$@` 有什么区别**

- `$*` - 以一个字符串形式输出所有传递到脚本的参数
- `$@` - 以 $IFS 为分隔符列出所有传递到脚本中的参数

**63) 如何在 bash 中定义数组？**

```
array=("Hi" "my" "name" "is")
```

**64) 如何打印数组的第一个元素？**

```
echo ${array[0]}
```

**65) 如何打印数组的所有元素？**

```
echo ${array[@]}
```

**66) 如何输出所有数组索引？**

```
echo ${!array[@]}
```

**67) 如何移除数组中索引为 2 的元素？**

```
unset array[2]
```

**68) 如何在数组中添加 id 为 333 的元素？**

```
array[333]="New_element"
```

**69) shell 脚本如何获取输入的值？**

a) 通过参数

```
./script param1 param2
```

b) 通过 `read` 命令

```
read -p "Destination backup Server : " desthost
```

**70)关于cut和tr**

`cut -d "word" -f num` 截取以word为分割的第num个字符串

`tr "word1" "word2"` 字符串中的Word1替换为Word2

**71)cat正向连续读 tac反向连续读nl输出行号显示文件more一页一页显示文件内容head只看头几行tail只看后几行**

**72)按行输出文件**

`sed -n 'a,bp' file` 输出从a到b行

**73)关于文件权限**

`chmod option file`给文件赋予权限

其中t是除创建者外其他用户不能删除

s是临时赋予root权限

**74)关于用户组**

`groupadd name` 创建一个用户组

`groups name`查看用户所有组

`chgrp name name` 将文件加入用户组，或将用户组加入文件

**75)关于命令替换**

`alias name=“order”可以将命令替换为名为name的命令执行`

写在`~/.bashrc`中`source ~/.bashrc`后可用

**76)关于管道的使用**

`|`可以将上一个命令的结果作为下一项命令的输入

例

```
cpu_1=`cat /proc/stat | awk '{if(NR==1) for(i=2;i<=NF;i++)printf("%s ",$i)}'`
```

获取了ｃｐｕ的温度

# shell基础知识

# 用户管理

## /etc/passwd

用户名　密码位　用户编号　归属组编号　姓名　$HOME　Shell

## /etc/shadow

用户名　以加密密码　密码改动信息　密码策略

## /etc/group

群组名　密码位　群组编号　组内用户

## /etc/gshadow

群组密码相关文件

# /etc/sudoers

用户名　权限定义　权限

# 用户管理相关命令

## su

切换用户

-|-l:重新登录

-m|-p:不更改环境变量

-c comand:切换后执行命令，并退出

## passwd

设定用户密码

-d:清除密码

-l:锁定账户

-e:使密码过期

-:S显示密码认证信息

-x days:密码过期后最大使用天数

-n days:密码冻结后最小使用时间

-s:更改登录shell

-f:更改用户信息

## chsh

更改用户ｓｈｅｌｌ

chsh -s Shell

## useradd

新建用户

-d:指定$HOME

-m:自动建立$ HOME

-M:不自动建立$HOME

-s shell:设置用户登录shell

-u uid:设置用户编号

-g groupname:设定用户归属群组

-G groupname:设置用户归属附加群组

-n:不建立以用户名称为群组名称的群组

-e days:缓冲时间，days天后关闭账号

-c string:设置用户备注

-D[表达式]:更改预设值

## id

显示用户信息

-g:下属所属组ID

-G:显示附加组ID

-n:显示用户，所属组或附加群组的名称

-u:显示用户ID

-r:显示实际ID`

## sudo

临时切换为ｒｏｏｔ用户

-s:切换为root shell

-i:切换到root shell,并初始化

-u username|uid:执行命令的身份

-l:显示自己的权限

## gpasswd

设定群组密码

-a username:将用户加入数组

-d username:将用户从群组中删除

-r:删除密码

-A username: 将用户设置为群组管理员

-M username1,username2...:设置群组成员

## usermod

修改用户账号

-c string:修改备注信息

-d dir:修改$HOME

-e days:密码期限

-f days:密码过期后宽限的日期

-g groupname:修改用户所属群组

-G groupname:修改用户所属附加群组

-l username:修改用户账号名称

-L:锁定用户密码，使密码无效

-s shell:修改用户登录后所使用的shell

-u uid:修改用户ID

-U:接触密码锁定

## userdel

删除用户

-r:删除用户相关文件和目录

# 数据提取操作

# TR对标准输入的字符替换，压缩，删除

`tr [csdt] <字符集> <字符集>`

c取代所有不属于第一字符集的字符

d删除所有属于第一字符集的字符

s将连续重复的字符以单独一个字符表示

t先删除第一字符集较第二字符集多出的字符

# cut切分

-d c：以c字符分割

-f num：显示num字段的内容

## -b num： 字节

-c num： 字符

# GREP检索

`grep [-acinv] <string> <file>`

-a: 将二进制文件以普通文件的形式搜索数据

-c：统计搜寻到的次数

-i：忽略大小写

-n：顺序输出行号

-v：反向输出

## SORT排序

`sort [-fbMnrtuk] <sile_or_stdio>`

-f：忽略大小写

-b：忽略最前面的空格

-M：以月份名称排序

-n：以纯数字方式排序

-r：反向排序

-u：uniq

-t：分割符，默认[TAB]

-k：以那个区间排序

## WC统计字符，字数，行数

`wc [-lwm] <file_or_stdin>`

-l:仅列出行号

-w：仅列出多少字

-m：仅列出多少字符

## UNIQ

`uniq [-ic]`

-i：忽略大小写字符的不同

-c：进行计数

## TEE双向重导项

`tee [-a] file`

-a：append

## SPLIT文件切分

`split [-bl] <file> PREFIX`

-b SIZE：且分为SIZE大小的文件

-l num：以num行为大小切分

## XARGS参数代换

`xargs [-0pne] <command>`

-0：将特殊字符还原为普通字符

-eEOF：当xargs读到EOF时停止

-p：执行指令前询问

-n num：每次执行command时需要的参数个数

# 进程管理

## free

打印系统情况和内存情况

-b|k|m|g:以字节,KB,M,G显示

-o:忽略缓冲区调节列

-t seconds:每隔seconds执行一次

-h:以可读形式显示

## dstat

实现监控磁盘，ＣＰＵ，网络等

## pstree

以树状显示进程派生关系

-a:显示每个程序的完整指令

-n:使用ＰＩＤ排序

-p:显示ＰＩＤ

-u:显示用户名

-l:使用长列格式显示树状

## kill

删除执行中的程序和工作

-a:处理当前进程时，不限制命令名和进程号的对应关系

-l 信号ID:不加信号ＩＤ，则列出全部信号

-p pid:给ｐｉｄ的进程只打印相关进程号，而不发送任何信号

-s 信号ID|信号 name: 指定要发出的信号

-u:　指定用户

# top

显示当前系统进程情况，内存，ＣＰＵ等信息

-b:以批处理模式操作

-c:显示完整的命令

-d seconds: 屏幕刷新间隔时间

-s:以安全模式运行

-S:累计模式

-u uname:指定username

-p pid:指定PID

-n nums:循环显示次数

-q:root时，给尽可能高的优先级

# ps

报告当前进程状态

ps -aux

ps -ef

# pgrep

查找进程ＩＤ

-o:起始进程号

-n:结束进程号

-l:显示进程名称

-p pid:指定父进程

-g gid:指定进程组

-t tty:指定开启的进程终端

-u uid:指定uid

# pkill

批量按照进程名杀死进程

-o:起始pid

-n:结束pid

-p pid:指定父进程发送信号

-g:指定进程组

-t tty:指定终端

# Linux数据提取操作

## TR对标准输入的字符替换，压缩，删除

```
tr [csdt] <字符集> <字符集>
```

c取代所有不属于第一字符集的字符

d删除所有属于第一字符集的字符

s将连续重复的字符以单独一个字符表示

t先删除第一字符集较第二字符集多出的字符

## cut切分

-d c：以c字符分割

-f num：显示num字段的内容

### -b num： 字节

-c num： 字符

## GREP检索

```
grep [-acinv] <string> <file>
```

-a: 将二进制文件以普通文件的形式搜索数据

-c：统计搜寻到的次数

-i：忽略大小写

-n：顺序输出行号

-v：反向输出

## SORT排序

```
sort [-fbMnrtuk] <sile_or_stdio>
```

-f：忽略大小写

-b：忽略最前面的空格

-M：以月份名称排序

-n：以纯数字方式排序

-r：反向排序

-u：uniq

-t：分割符，默认[TAB]

-k：以那个区间排序

# WC统计字符，字数，行数

```
wc [-lwm] <file_or_stdin>
```

-l:仅列出行号

-w：仅列出多少字

-m：仅列出多少字符

# UNIQ

```
uniq [-ic]
```

-i：忽略大小写字符的不同

-c：进行计数

# TEE双向重导项

```
tee [-a] file
```

-a：append

# SPLIT文件切分

```
split [-bl] <file> PREFIX
```

-b SIZE：且分为SIZE大小的文件

-l num：以num行为大小切分

# XARGS参数代换

```
xargs [-0pne] <command>
```

-0：将特殊字符还原为普通字符

-eEOF：当xargs读到EOF时停止

-p：执行指令前询问

-n num：每次执行command时需要的参数个数