Shell用途：



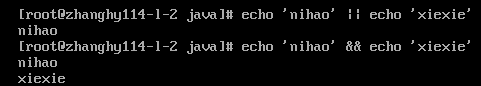
1. Ping一个网站通的话打印通，不通打印不通

Ping –c1 ‘www.baidu.com’ && echo “通” || echo ‘不通’

&&左边执行成功才继续执行右边

||表示左边执行成功右边不执行

C1表示执行一次



如果想ping 一个数据不想返回结果

Ping –c1 [www.baidu.com](http://www.baidu.com) &>/dev/null即可

# 1.执行脚本

编写的shell脚本默认是用bash这个命令行解释器，所以执行

./ \*.sh(但是要修改权限)

或者 bash \*.sh也行(无需修改权限)

# 2.查找

Find查找文件

Find / -iname nginx –type f

Iname不区分大小写

Whereis 查找程序位置

Whereis nginx

Which +命令

查找命令位置

Which bash

which cat

# 3.编写脚本

**#！shell命令解释器的路径，放在开头**

eg：#！/bin/bash

当然默认解释器就是bash

执行python脚本：  
python \*\*.py文件

# 4.脚本执行方式（升级）

c语言 编译型执行——转换成二进制码（其中根据CPU不同，编译的类型不同，如果CPU是X86那么会是复杂指令集，是power是精简指令集）

shell 解释型执行

python 既有编译也有解释，编译（转化为字节码，然后由PYTHON虚拟机来执行）

java 编译型执行——转换成字节码（由JAVA虚拟机来执行这个字节码） tomcat和hadoop都是java语言，所以运行要先安装JDK

perl 解释型执行

except 解释型执行（一般没安装）

程序是由数据和逻辑组成

执行脚本：

bash编辑器执行

bash a.sh

python 执行Py脚本

python a.py

**但是也可以直接执行脚本**

./a.sh 或者source a.sh

./a.sh有二点很重要

./表示当前目录，执行一个脚本是要用绝对路径的，但是./表示当前目录，在当前目录下是可以执行的

所以执行root下脚本可以使用/root/a.sh 也可以cd root 然后在这个目录下用相对路径 ./a.sh也可以

# 5.输入重定向

之前一直没弄清楚输入重定向，只知道输出重定向

输出重定向是将之前的命令执行内容打印到后面的文件里面

例: ls –al >> a.txt

这样cat a.txt文件内容就是ls –al的内容了

但是输入重定向就更牛逼了：是将输入的内容作为之前的命令来使用，总之就是输入作为命令,直到输入后面同样内容就停止

例： ls << EOF

-al

EOF

我输入了-al内容，他提示我还要继续输入，我想让他停止，就输入EOF，然后这个就相当于执行ls –al了

再举一例：cat << EOF

Nihao

Xiexie

EOF

Nihao

Xiexie

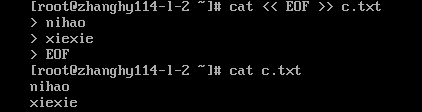
这个就相当于查看NIHAO xiexie这个内容，打印出来内容。

如果再将打印内容输出到文件呢

没错，输入输出一起用，我们输入的内容就写入到文件了

cat << EOF >> a.txt

这个命令就是将输入内容输入到a.txt文件中



# 6.如果一个脚本文件执行多个解释器内容

比如一个文件既有cat a.txt 又有print(‘hello world’)那么该怎么办？

这时候就需要用输入重定向了

a.sh文件

#!/bin/bash

cat a.txt

python << EOF

print(‘hello world’)

EOF

# 7.当前shell与子shell

当我们执行一个shell,比如bash \*.sh就会打开一个新的子shell

但是当我们直接执行，比如. \*.sh或者 source \*.sh就在当前执行

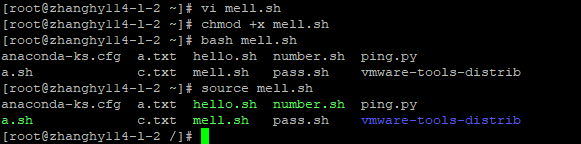
eg. vi mell.sh

ls

cd ..

然后修改权限

执行



可以发现，当我们用bash mell.sh执行，虽然执行结果，但是没有回到/目录

但是用source mell.sh就回到原始目录/

因为之前是新开了一个bash子shell中执行的，而第二个就是在当前shell执行的（类似于直接敲命令了）

但是还是有区别的，子shell有时候无法运行一些配置生效

比如：在 vi ~/.bashrc

加入 alias zhu=’rm -rf’

然后 bash ~/.bashrc

发现根本不生效，除非重启

但是执行 source ~/.bashrc就可以马上生效

# 8.环境变量

~/.bashrc

/etc/profile

/etc/.bash\_profile

~/.bash\_profile

这四个环境脚本文件

.文件，是因为这是隐藏文件，只有用 ls –al才看得见

~是用户目录，如果是root就是root目录，其他用户就是home/用户/

~/就是用户目录下

当我们任何一个用户进入家目录，ls -al都会有.bashrc 和.bash\_profile二个隐藏文件

其中.bash\_profile是当前用户下，只有在bash解释器下的设置一些环境变量和配置，

要让配置生效，必须是这个用户

而.bashrc也是在当前用户下，一些设置环境变量个人配置，这个用户设置了，其他用户只要以该用户身份运行一样可以

/etc/profile则是环境变量，任何用户都可以生效配置

/etc/.bash\_profile则是在bash解释器下面执行。任何用户配置都生效

# **9.用户shell四个文件**

当我们创建新用户，在每一个用户文件夹下面都有四个隐藏文件

.bashrc 开机就会运行，只对当前用户有效

.bash\_profile 用户环境配置文件，里面有执行.bashrc文件命令，所以先执行这个，就可以执行.bashrc

.bash\_logout 只要退出就会执行脚本的命令

.bash\_history 只要退出，就会把系统执行过的命令保存到这个文件，作为history，这样下次登录就能看见history

这四个shell文件非常重要

例1：有些黑客可以通过我们的history来查看我们的命令

我们想每次退出登录就删history文件内容怎么办？

在 .bash\_logout 添加这个命令就可以了

rm -f $HOME/.bash\_history  ，每次退出执行删除这个history文件命令

例2：

我们想配置一些命令，或者代理：  
vi ～/.bashrc

添加 ipconfig = ‘ip addr’

这样每次输入命令就可以查看IP地址了

例3：

我想把我一个安装包安装文件加入PATH路径，怎么办？

假如这个文件是/home/zhanghy/python/

因为用户目录～可以这样表示 $HOME，当然必须是在zhanghy用户下

所以在～/.bash\_profile 添加

PATH=$HOME/path

我直接把系统搞崩溃了，bash 未找到命令

网上找到解决方法:

<https://blog.csdn.net/chenht8/article/details/67645649>

这是deepin系统，文件是.profile，我测试了centos完全没问题

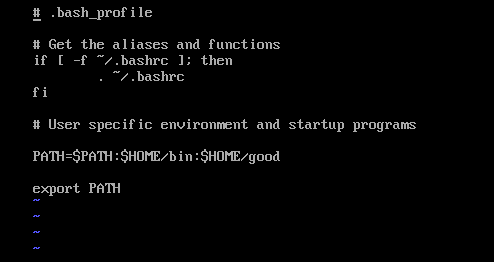
下面在root用户下添加PATH路径：

首先进入root用户,编辑.bash\_profile文件

在PATH=$PATH:$HOME/bin(添加了二个变量，一个是系统路径，一个是/root/bin文件夹,不同路径之间用：隔开)

后面增加:/home/zhanghy/python即可

测试：



在good文件下加入一个脚本文件a.sh

加入内容： I take it

然后修改权限

最后随便进入一个路径：



成功。

10.non-login shell 与login shell

**这二个其实是不同的shell编辑器**

non-login是不是直接登录命令，比如我们一开始进入centos7图形界面，然后打开终端，这个shell就是non-shell，用户登录 su –zhanghy su –用户名

login就是直接登录命令行登录，在命令行输入账号密码，登录 su 用户名

但这二种登录区别很大，第一种上面四个文件

~/.bashrc

/etc/profile

/etc/.bash\_profile

~/.bash\_profile

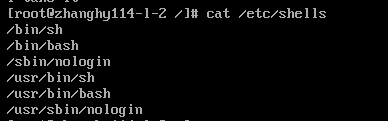
.bash\_history都可以执行，第二种是直接读取下面文件

~/.bash\_profile

~/.bashrc

11.查看所有的shell编辑器

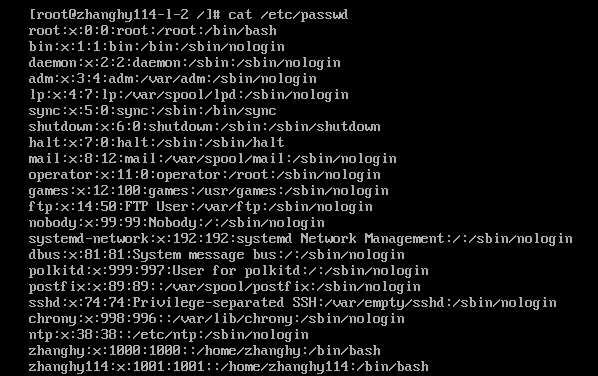
cat /etc/shells



如何查看默认的shell  
echo $SHELL

还可以

cat /etc/passwd



最最下面就是/bin/bash

如何修改为其他解释器为默认shell解释器

命令：

chsh –l 查看

chsh –s /bin/sh

然后查看，发现修改成功

12.bash的一些基本操作

查看历史记录：

上下键

！为查看历史记录的关键标记（写脚本必备）

!number，会默认打印history的数字命令然后执行



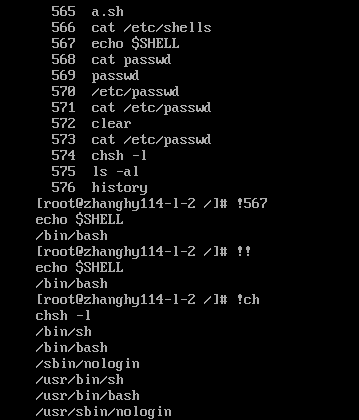
！！



打印上一个命令

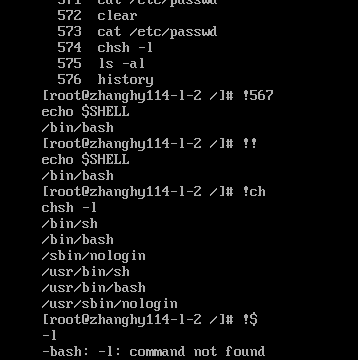
！加命令前缀（当命令不全的时候会补全）

会打印最近一个符和的并执行



！$

这也是打印上一个命令，但是打印的是上一个命令的后缀



别名：

alias 查看所有别名

新加别名： alias zhu=’ls -al’

取消unalias zhu

但是只是临时生效，永久是vi ~/.bashrc

添加 这个，然后 在shell 下执行 source ~/.bashrc

快捷键：

ctrl + a 敲命令时候，光标移到命令最前面ahead

ctrl + e敲命令时候，光标移到命令最后面 end

ctrl +d 退出，类似于exit

ctrl +k 敲命令时候，将光标之后的命令全删了

ctrl +u 敲命令时候，将光标之钱的命令全删了

ctrl +y 敲命令时候，将删除的命令撤销还原

13.sleep命令以及后台前台（&nohup）运行

sleep命令,非常实用的命令，表示休眠，在执行命令时候，可以让其休眠再执行

比如，延迟10秒后执行ls –al命令

sleep 10s;ls –al

系统会等待10秒然后执行

延迟5分钟后执行

sleep 5m;ls –al



但是有个问题，这个命令执行是在前台执行，会影响我们干其他活，我们想让他后台执行？什么是前台和后台

执行一个程序或者脚本，我们希望一直看着他执行，就是前台

比如手机电脑打开很多软件，其中我们只玩一个软件，其他游戏软件在挂机，就是放到后台，这个软件就是前台

执行一个软件，放在看不到的地方，但是依旧在运行，就是后台

linux 执行后台

命令 &

比如sleep 10000 &

这个程序就可以在后台执行了 ps –ef | grep sleep 可以查看

但是我们想等10分钟，执行ls –al呢

sleep 10m;ls –al &

这样是不行的，进不去后台

使用后台执行脚本就可以了

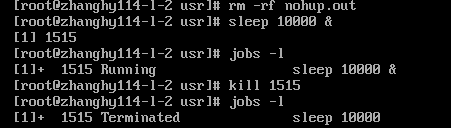
echo ‘sleep 10m;ls –al &’ >> a.sh

bash a.sh &

这样就可以在前台玩，等10秒自动执行

jobs –l可以让我们查看后台运行的程序

如果想关掉，直接kill -9 进程 （-9是强制关闭，不加也可以）

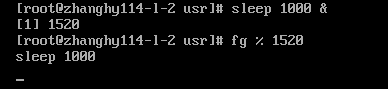


【1】是后台进程号，1515是进程号，一个是kill %1， 一个是kill 1515

.**fg %n 让后台运行的进程n到前台来  
bg %n 让进程n到后台去;（这个不知道怎么弄，试了不成功）  
PS：“n”为jobs查看到的进程编号。上面就是1**

**ctrl +c关闭前台进程**

kill %n 就是关闭进程号



将一个后台进程拉到前台了

&是将前台进程放到后台执行，但如果关掉终端或者关机，进程就会关闭

**nohup,一个更牛逼的命令**，不挂断执行命令，即使是关闭终端都可以（这个非常重要，因为远程执行的命令如果关闭了窗口也就意味着关闭，关闭shell也可以，关机就不可以）都还在执行

实验Nohup作用：

我登录putty，执行 sleep 10000 &

查看命令 ps –aux | grep sleep

存在

查看虚拟机本机进程，ps –aux | grep sleep

也是存在

但是关闭putty之后，再查看虚拟机本机进程，ps –aux | grep sleep

，进程不存在

这时候就需要nohup了

我在putty执行，nohup sleep 10000,关闭putty

查看本机进程：存在



1. 一旦执行这个命令，前台关闭，无法输入命令，所以一般配合&使用，放到后台
2. 一旦执行这个命令，会在执行**所在目录**（pwd的目录），下面生成一个nohup.out，会将这个命令执行的结输出到这里面，如果已经有，还无法执行，那么会生成到$HOME下面，但是如果一直这个执行，这个内容会一直增加，有时候需要我们定时清理

实验：

接下来在/usr下面生成一个s.sh脚本

并写入内容：

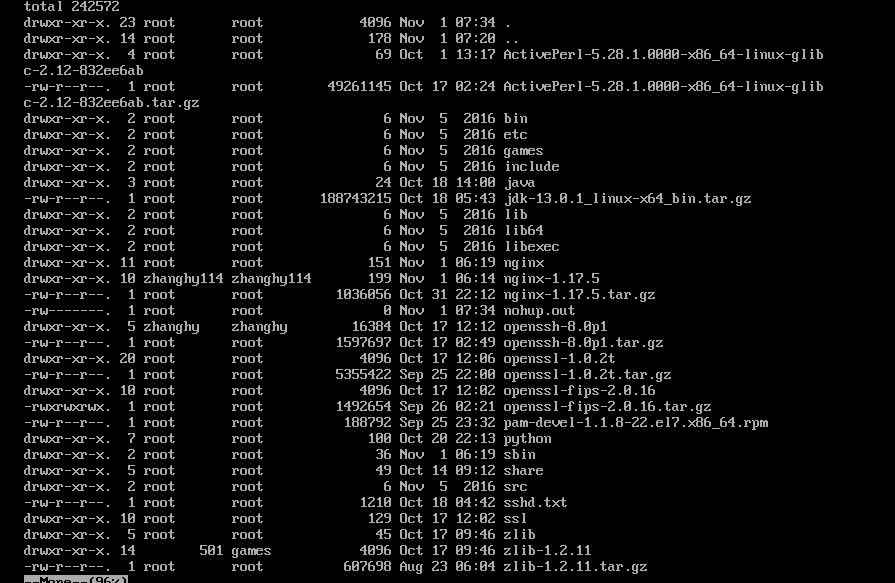
sleep 10s;ls –al

在/usr/local目录下执行 nohup bash /usr/s.sh



等待10秒，生成文件

打开 more nohup.out (more 分页查看文档)

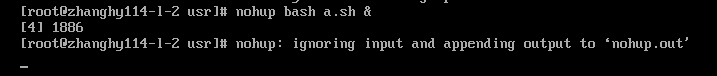


所以执行**nohup 命令 &**最好

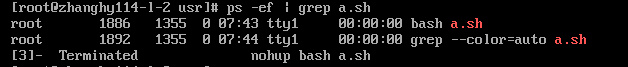
如何永久执行脚本并后台运行呢？（记住不是nohup bash a.sh &&，这样无效）

比如这个a.sh脚本

nohup bash a.sh &输入完之后



这时候记得退出，进入前台，ctrl +c



可以发现脚本已经后台运行了

重启

发现脚本并不在执行了



顺带说一下，查看目前终端命令：

tty

# 14.管道符|tee 以及输入输出重定向

管道符就不说了，说一下tee

将上一个命令结果打印输出到文档，并将内容输出到标准内容（与输入不一样，他不但输入到文档，还输出内容，前面类似于>输出），输出文档之前会格式化文档

输入重定向将键盘输入的结果作为之前命令

但是有<和<<

**<是将文件作为命令的标准输入**

**eg：wc –l < a.txt**

**统计a.txt的行数**

**cat < a.txt等等命令**

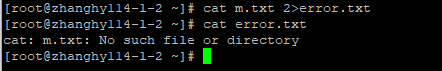
而<<则是分隔符，输入这个分隔符才停止

输出：

>将前面命令执行结果内容输入文档，但是会删除文档的内容再输入，和tee一样

>>不会删除文档内容，文档后续增加

2>将错误输出到一个文件内容



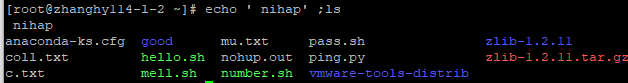
&>混合重定向，这个很重要，可以当作错误，也可以当作输出>，但不是>>

最最牛逼的， command &>/dev/null 执行一个命令不打印结果，内容直接给保存到一个空文件中

# 15.如何一行敲多个命令

用;（分号）之间有没有空格没关系

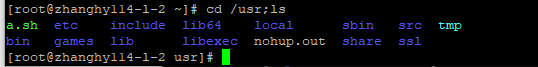
ls ; echo ‘ao’



但是命令有执行顺序，先执行第一个，后执行第二个

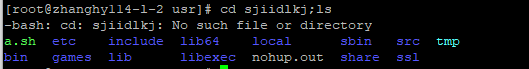
eg:

cd /usr;ls



ls不是root目录，而是/usr目录

但是如果第一个命令执行失败了，根本不影响第二个命令执行：



但如果是有顺序的命令，最好用&&，因为第一条命令失败就不会执行第二条命令

||是第一条命令成功就后续不执行

# 16．$?命令

一般打印上一个命令的的退出状态，或者上一个函数的返回值，就用到这个命令

eg:

1. #!/bin/bash
2. #得到两个数相加的和
3. function add(){
4. **return** `expr $1 + $2`
5. }
6. add 23 50 #调用函数
7. echo $? #获取函数返回值

echo $?

即为75

true一直为0

false一直为1



# 17.通配符

当我们搜索文件，不知道文件名时候，可以使用通配符模糊搜索

\*

查看以a开头的文件

ls a\*



查找文件时候非常牛逼

find / -name \*.txt

这样就查找系统所以txt文档了

？

代表一个任意字符



但是



因为只代表一个字符，二个就敲二个字符

[1-3] [a-z] [a,s,d]等[]：[a-zA-Z0-9]中间可以不加符号

当要模糊搜索一个文件时，发现我只记得这个好像是一个字母，或者一个数字，比如alice，我只记得alic，最后一个字母不记得，怎么办:

ls alic[a-z]

[a-z]代表其中一个字母

a2.txt只记得是一个数字，比如a数字.txt:

find / -name s[0-9].txt

只记得好像是2,又好像是a，怎么办？

ls [2a].txt

{}：

{1..3}代表1 2 3三个数

touch {1..3}.txt

会创建1.txt 2.txt 3.txt

{111，222，33}.txt

会创建三个文件

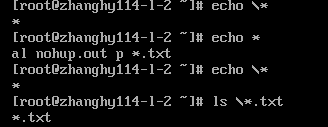
^：一般配合[使用]，比如除了数字的匹配，[^0-9]即可

\转义符(反斜线)，比如一个符号有意义，但是我们又只是要其字符怎么办？

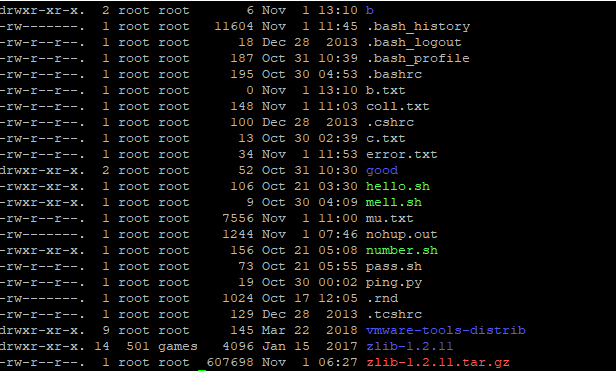
一个文件名字为\*.txt

怎么查询？

ls \\*.txt



实战：



我想删除除了隐藏文件之外的全部文件，怎么办？

ls /root [^.]\*（/root表示root目录下，[^.]是反，第一个字符不是.，后面全用\*表示）

会打印所有除了.隐藏文件

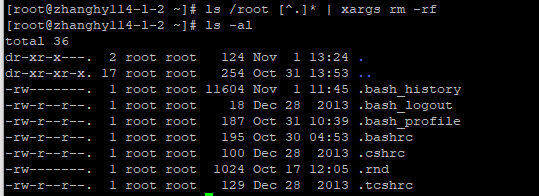
然后删除

ls /root [^.]\* |xargs rm –rf

即可

当然还有更简单的办法：

ls | xargs rm -rf

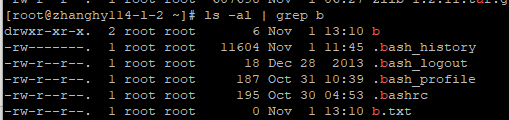


完美

# 18.umask创建文件目录的默认权限

当创建一个新文件时，权限肯定是644

创建目录，肯定是755



为什么，这就是umask设置

与chmod不同，chmod是直接设置权限（421的格式（其中代表可执行可写可读），第一个xwr是用户，第二个是用户组，第三个是其他用户，一般u+x既可以修改文件执行权限

）umask则是缺省权限，一般022，则表示目录777-022为755，文件666-022=644

所以默认创建一个新文件时，权限肯定是644，创建目录，肯定是755

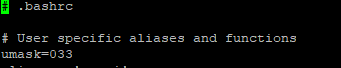
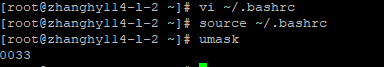
可以在/etc/bash\_profile中修改，完美

当然，在~/.bask\_profile中也可以，但是只针对这个用户

修改

umask 033 临时生效

永久生效：



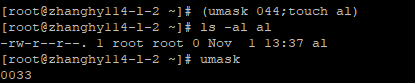
# 19.子shell

当在shell中设置一些bash，可以生效。

所谓子shell，即从当前shell环境新开一个shell环境，这个新开的shell环境就称为子shell(subshell)，而开启子shell的环境称为该子shell的父shell，子shell会从父shell中继承很多环境，如变量、命令全路径、文件描述符、当前工作目录、陷阱等等，但子shell有很多种类型，不同类型的子shell继承的环境不相同。

但是如果在子shell中设置一些bash，其实只能在子shell中生效，进入shell就无法生效

而（command）就是打开一个子shell



在子shell中执行，但是子shell设置时改变不了父shell的

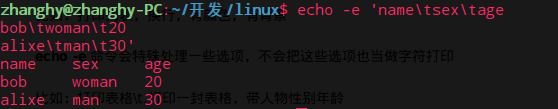
# 20.echo打印

当我们需要打出一些信息来确认时，有时候信息需要确认

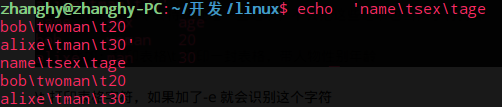
比如：打印列表，换行，有颜色，有背景

**echo -e**命令会特殊处理一些选项，不会把这些选项也当做字符打印

1.比如：打印表格\t,打印一封表格，带人物性别年龄



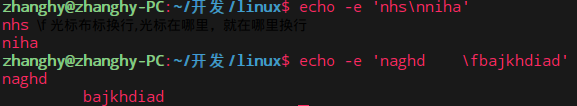
\t打印表格字符，如果加了-e就会识别这个字符，不加就如下图



1. 换行和光标不变换行

\n换行

\f光标布标换行,光标在哪里，就在哪里换行



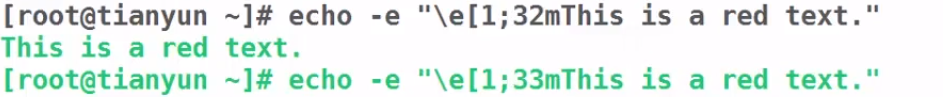
1. 颜色和背景

\e[1;序号m内容 （注意是;分号而不是：冒号）

序号：颜色码：重置=0，黑色=30，红色=31，绿色=32，黄色=33，蓝色=34，洋红=35，青色=36，白色=37

深度截图_选择区域_20191103175402

但是这个弄完之后，后面会一直都是这个颜色



要用重置/t[0m

echo -e '\e[1;33m'我将打印有颜色的\t[0m

即可只让这行内容显示有颜色

背景：

Greed Background颜色码：重置=0，黑色=40，红色=41，绿色=42，黄色=43，蓝色=44，洋红=45，青色=46，白色=47

格式和颜色一样，只是序号变了

深度截图_选择区域_20191103175926

后面一定要加\e[0m

这里就有疑问了：

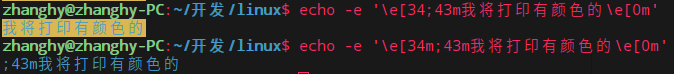
[1:31m这个1可以去掉吗？

当然可以

深度截图_选择区域_20191103180342

可以混合打印吗，比如带颜色的背景？

只要之间加;就可以了



最后一个带m，就行了

1. 字体特效

红色数字处还有其他数字参数：0 关闭所有属性、1 设置高亮度（加粗）、4 下划线、5 闪烁、7 反显、

打印闪烁的字体

echo -e "\033[37;31;5mMySQL Server Stop...\033[39;49;0m"

# 21.变量

变量名=内容/路径/函数/命令…..

例：a=10.145.192.94

注意，如果是字符串记得加’’’

echo $a

但这只是临时使用，要将这变量写到脚本中才可以

或者永久生效，在bash\_profile添加内容中也可以

eg:a=1082

export a=1092就可以生效

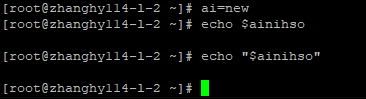
取消upset 变量名

如何查看所有环境变量：

env命令

‘’’ 和’’

当用””时，字符串内容如果有变量，那么变量内容可以打印出来

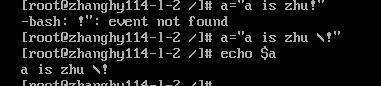
但是’’，无论内容是什么，都会将其打印成字符串

为什么上面没有打印变量内容，因为系统默认将ainihao默认为变量了

所以打印不出来内容

怎么办，给变量加{}

echo ${ai}nihao

但是双引号有时候需要转义字符

如上图所示，“”无法识别！，这时候可能需要单双引结合

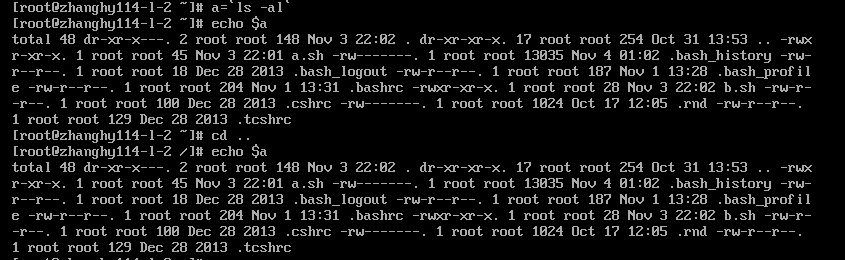
**``数字1左边的键**

命令替换符，可以将一串命令赋值给变量

a=`ls -al` (会先执行ls –al，然后将结果赋值给变量)

echo $a

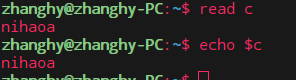
对于一些

所以命令执行结果已经赋值给变量，再其他目录再执行，还是这个值了

变量手动赋值:

read a

然后会手动给变量a赋值



但是没有提示内容：

这里我们给他提示

read -p ‘请输入变量a的值’ a

即可

$1，表示 脚本传入的第一个参数（位置变量）

在一个脚本中写入这个$1

可以在执行脚本时后面输入变量值来辅助给$1

eg:

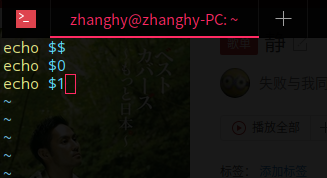
vi a.sh 内容如下：

echo $1

运行脚本，./a.sh 1028kai

打印内容就是1028kai

变量说明(预定义变量):   
$$ ——Shell本身的PID（ProcessID）   
$! ——Shell最后运行的后台Process的PID   
$? ——最后运行的命令的结束代码（返回值）   
$- ——使用Set命令设定的Flag一览   
$\* ——所有参数列表。如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。   
$@ ——所有参数列表。如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。   
$# ——添加到Shell的参数个数   
$0 ——Shell本身的文件名   
$1～$n ——添加到Shell的各参数值。$1是第1参数、$2是第2参数





可以看到$$是进程号，但是不要加’’，因为不是字符串

# 22.子shell与父shell

shell默认的是bash这个命令解释器

echo $SHELL

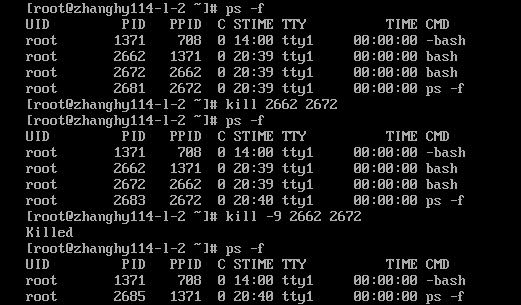
查看



但其实bash是有子shell

**如何创建子shell**

什么时候是子shell，运行bash，就会启动一个子shell

可以看见bash的进程是bash，这就是一子进程，用kill -9可以强制关停，当然输入exit也可以退出。

  子Shell从父Shell继承得来的属性如下：

               当前工作目录

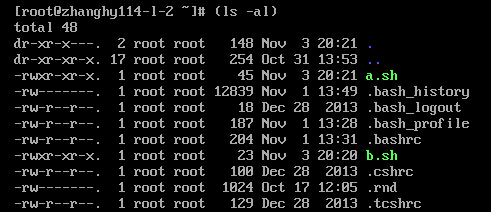
               环境变量

               标准输入、标准输出和标准错误输出

               所有已打开的文件标识符

       子Shell不能从父Shell继承的属性：

               除环境变量和.bashrc文件中定义变量之外的Shell变量

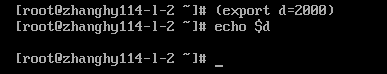


**其他子shell，括号**

用括号来执行命令，其实是开创了一个子shell来执行这个命令

（ls -al）

但是执行完命令就会关闭这个子shell

我们尝试在子shell加入环境变量：

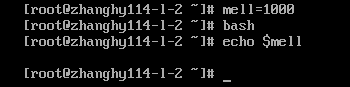
果然成功，这是正确的

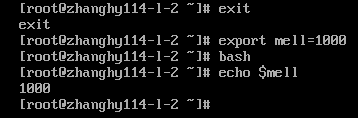
**后台子shell：**

当一个程序需要运行，可以通过加&放到后台让子shell来执行这个后台

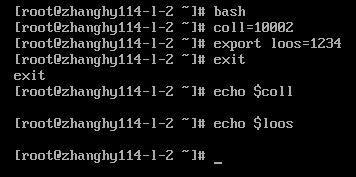
子shell里面还可以输入bash再创建一个子shell

**那么子shell和父shell有啥区别呢？**



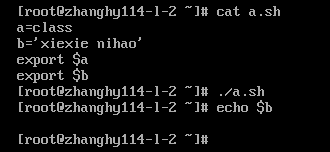


子shell无法继承父shell的临时变量，但可以继承环境变量



父shell无法继承子shell的临时和环境变量

**脚本子shell**



脚本子shell，这是因为执行脚本也开启了子shell的缘故

同理，使用bash也是一样

但是，有一种办法可以让我们引用这个变量

source 脚本名字

这样就可以



source和.执行文件不会生成子shell

     不创建子shell，在当前shell环境下读取并执行脚本中的命令，相当于顺序执行脚本里面的命令

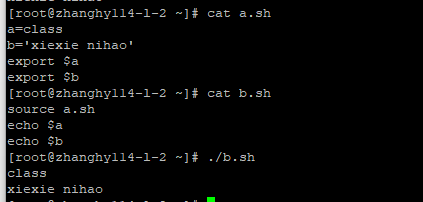
但是路径执行，和bash执行是会生成子shell

**ssh登录**

当我们用ssh登录系统时候，设置的一些临时变量，以及启动了相关进程，一旦关闭了远程，设置就会消失。

这是因为系统也给我们开启新的shell，所以我们即使设置了环境变量，设置了临时变量，都不会在系统生效。

实战：一个脚本添加环境变量，来让另外一个脚本来运行这个脚本



# 23.if判断

if 内容;then

‘’’

也可以写成

if 内容

then

‘’’

因为同一行多条必须用;隔开

if 内容

then

内容

else

内容

fi

接下来写一个ping条件，成功打印成功，不成功打印错误

if ping -c4 10.145.152.127 &>/dev/null

then

echo‘你成功了’

else

echo ‘你失败了’

fi

条件判断

if [#条件判断#]

#代表空格，记住一定要加

eg:

ping -c4 10.145.152.127 &>/dev/null

if [ $? -eq 0 ]

then

echo‘你成功了’s

else

echo ‘你失败了’

fi

4.逻辑判断符号

-eq equal等于

-gt great than 大于

-lt less than 小于

-ge great than and equal 大于等于

-le less than and equal 小于等于

linux中的命令判断：0为真，1为假

&& || ！就不说了

-f 文件 判断是否是文件

-d 目录 判断是否是目录

