

2024-04-11-2021219113-2021213595-沈尉林

1.

Q 1: 怎样将父进程里的变量定义 `temp=3`, 在随后的子进程 `child.sh` 中可以被使用, 举实例说明。

创建父进程脚本, 名为 `parent.sh`, 代码如下:

```
Bash
1  # 定义环境变量
2  export TEMP=3
3  # 启动子进程
4  ./child.sh
```

创建子进程脚本, 名为 `child.sh`, 代码如下:

```
Bash
1  # 使用父进程定义的环境变量
2  echo "The value of TEMP in child process is: $TEMP"
```

执行 `parent.sh` 脚本, 结果如下:

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ./parent.sh
The value of TEMP in child process is: 3
```

Q 2: 在子进程 `child.sh` 中, 将 `temp` 重新定义为4, 在子进程 `child.sh` 死掉后, 回到父进程, 这个重新定义的 `temp=4` 能被带回到父进程吗? 举实例说明。

不能

修改父进程脚本, 代码如下:

```
Bash
1  # 定义初始变量
2  TEMP=3
3
4  # 打印初始值
5  echo "Initial value of TEMP in parent process: $TEMP"
6
7
```

```
7 # 启动子进程
8 ./child.sh
9
10
11 # 打印子进程结束后父进程读取到的值
echo "Value of TEMP in parent process after child process
terminates: $TEMP"
```

子进程脚本修改为：

```
Bash
1 # 将temp重新定义为4
2 TEMP=4
3
4 # 打印修改后的值
5 echo "Value of TEMP in child process: $TEMP"
```

重新执行该父进程脚本，结果如下：

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ./parent.sh
Initial value of TEMP in parent process: 3
Value of TEMP in child process: 4
Value of TEMP in parent process after child process terminates: 3
```

可以看出：即使子进程在其环境中修改了 TEMP 的值为 4，父进程在子进程结束后依然会输出初始值 3，而不是子进程中修改后的值。这是因为子进程的环境和父进程的环境是相互独立的，子进程的修改不会影响到父进程。

Q 3: 如果我非要把子进程里的 `temp=4` 带回到父进程中，怎么办？举实例说明。

使用 `source` 命令在父进程中执行子进程的脚本，这样子进程中的变量定义就会影响到父进程。父进程脚本修改如下：

```
Bash
1 # 定义初始变量
2 TEMP=3
3
4 # 打印初始值
5 echo "Initial value of TEMP in parent process: $TEMP"
6
7 # 执行子进程脚本，并在父进程中运行，source命令会影响到父进程的环境
8 source ./child.sh
9
10 # 打印子进程结束后父进程读取到的值
11
```

```
echo "Value of TEMP in parent process after sourcing child process: $TEMP"
```

重新执行，结果如下：

```
Initial value of TEMP in parent process: 3
Value of TEMP in child process: 4
Value of TEMP in parent process after child process terminates: 4
```

可以看出已成功修改。

2. 看进程里的变量值，可以用 `set`，`env`，`export`，说明他们的使用区别。

1. `set` 命令：

- `set` 命令用于设置或显示shell的当前设置。它会显示当前shell中定义的所有变量（包括环境变量和局部变量）、函数和当前的shell设置。
- `set` 命令还可以用于设置shell选项和位置参数。

2. `env`命令：

- `env` 命令用于显示当前系统环境中的所有环境变量。
- 它通常用于在不同的环境中执行命令，例如在指定环境变量的情况下执行特定程序。

3. `export`命令：

- `export` 命令用于将变量导出为环境变量，使其在当前shell及其子进程中可见。
- 当你在shell中定义一个变量时，默认情况下它是一个局部变量，只在当前shell中可见。使用 `export` 命令可以将其提升为环境变量，使其对当前shell及其子进程可见。

假设有一个名为 `MY_VAR` 的变量：

- 如果你在当前shell中使用 `set` 命令，它将显示所有当前设置的变量，包括环境变量和局部变量。
- 如果你在当前shell中使用 `export MY_VAR=123` 命令，它会将 `MY_VAR` 导出为一个环境变量，使得它对当前shell及其子进程可见。
- 如果你在当前shell中使用 `env` 命令，它将显示当前系统环境中的所有环境变量，包括你已经导出的环境变量。

3. `cmd1;cmd2;cmd3` `cmd1&&cmd2&&cmd3` `cmd1||cmd2||cmd3` 以上3种使用的区别，举实例说明

`;` 分号：

- 使用分号；将多个命令串联在一起，这些命令将依次执行，无论前面的命令成功与否。
- 即使前面的命令失败，后续的命令仍然会执行。

&& 逻辑与：

- 使用逻辑与 && 将多个命令串联在一起，这些命令将依次执行，只有前一个命令成功执行（返回退出状态码0）时，才会执行后续的命令。
- 如果前一个命令失败，后续的命令将不会执行。

|| 逻辑或：

- 使用逻辑或 || 将多个命令串联在一起，这些命令将依次执行，只有前一个命令失败（返回非零退出状态码）时，才会执行后续的命令。
- 如果前一个命令成功，后续的命令将不会执行。

假设我们有三个命令：

1. `ls /some/directory` - 列出 `/some/directory` 中的文件和文件夹。
2. `grep "specific_pattern" file.txt` - 在 `file.txt` 文件中查找特定模式。
3. `echo "File not found"` - 打印消息 "File not found"。

1. ; 分号：

在终端中输入：

```
1  ls /some/directory ; grep "specific_pattern" file.txt ; echo "File not found"
```

结果如下：

```
root@iZ2ze0lgnf08nfb73yrokZ:/home/alanshen# ls /some/directory ; grep "specific_pattern" file.txt ; echo "File not found"
ls: cannot access '/some/directory': No such file or directory
grep: file.txt: No such file or directory
File not found
```

在这种情况下，无论 `ls /some/directory` 是否成功，后续的命令都会继续执行。如果 `ls /some/directory` 失败（因为目录不存在），后续的命令仍然会执行。

2. && 逻辑与：

在终端中输入：

```
1  ls /some/directory && grep "specific_pattern" file.txt && echo "File not found"
```

结果如下：

```
root@iZ2ze0lgnf08nfb73yrokZ:/home/alanshen# ls /some/directory && grep "specific_pattern" file.txt && echo "File not found"
ls: cannot access '/some/directory': No such file or directory
```

在这种情况下，只有当 `ls /some/directory` 成功时，才会继续执行后续的命令。如果 `ls /some/directory` 失败，后续的命令将不会执行，因为 `&&` 的前一个命令未成功。

3. `||` 逻辑或：

在终端中输入：

```
1  ls /some/directory || echo "File not found" || grep  
   "specific_pattern" file.txt
```

结果如下：

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ls /some/directory || echo "File not found" || grep "specific_pattern" file  
.txt  
ls: cannot access '/some/directory': No such file or directory  
File not found
```

在这种情况下，只有当 `ls /some/directory` 失败时，才会继续执行后续的命令。`echo "File not found"` 成功后，后续的命令将不会执行，因为 `||` 的前一个命令已经成功。

4.

现有2个无限循环脚本做实例例子：

1.sh :

```
1  let i=0  
2  while true  
3  do  
4      let i++  
5      echo hello $i  
6  done
```

2.sh:

```
1  for((i=0;;i++))  
2  do  
3      echo hello $i  
4  done
```

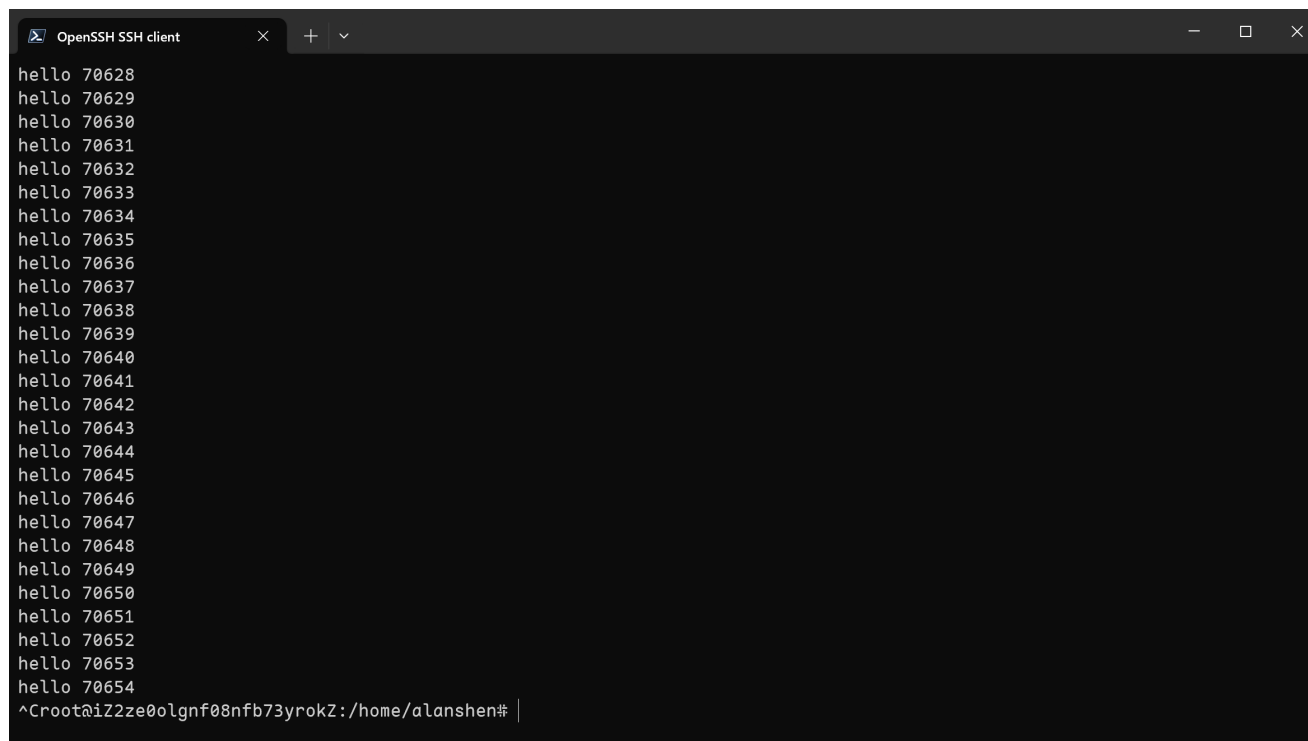
Step 1: 让 `1.sh` 在前景运行，然后终止它

在终端中输入：

```
1 ./1.sh
```

然后在脚本运行的终端中，按下 **Ctrl + C** 组合键，来终止脚本的执行。

结果如下：



```
hello 70628
hello 70629
hello 70630
hello 70631
hello 70632
hello 70633
hello 70634
hello 70635
hello 70636
hello 70637
hello 70638
hello 70639
hello 70640
hello 70641
hello 70642
hello 70643
hello 70644
hello 70645
hello 70646
hello 70647
hello 70648
hello 70649
hello 70650
hello 70651
hello 70652
hello 70653
hello 70654
^Croot@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen#
```

Step 2: 让 **1.sh** 在后台运行，然后终止它

在终端中输入，并将输出重定向到 **1.txt** 中：

```
1 ./1.sh > /1.txt &
```

结果如下：

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ./1.sh > /1.txt &
[1] 253086
```

接下来使用 **kill** 命令终止该 PID 对应的进程：

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# kill 253086
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# jobs
[1]+  Terminated                  ./1.sh > /1.txt
```

5. 分别调用上题的 **1.sh** 和 **2.sh** 运行在后台，然后用实例说明 **ps** **psxf** **top** **htop** 之间的区别

Step 1: 在后台运行这两个可执行脚本

在终端中输入：

Shell ▾

```
1  ./1.sh > 1.txt &
2  ./2.sh > 2.txt &
```

结果如下：

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ./1.sh > 1.txt &
[1] 253091
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ./2.sh > 2.txt &
[2] 253094
```

Step 2: 分别使用 `ps`、`psxf`、`top` 和 `htop` 命令来查看这两个后台进程，然后比较它们的区别。

使用 `ps` 命令：`ps aux | grep 1.sh` 和 `ps aux | grep 2.sh` 可以看到两个脚本进程的详细信息，包括 PID、CPU 使用率等。

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ps aux | grep 1.sh
root      255232  0.0  0.0   8900   656 pts/1    R+   17:50   0:00 grep 1.sh
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ps aux | grep 2.sh
root      255234  0.0  0.0   9032   656 pts/1    S+   17:50   0:00 grep 2.sh
```

使用 `psxf` 命令：`psxf` 命令可以以树状结构显示进程之间的关系，可以更清晰地看到脚本进程及其父进程之间的关系。

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ps xf | grep 1.sh
255236 pts/1      S+      0:00      \_ grep 1.sh
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# ps xf | grep 2.sh
255238 pts/1      S+      0:00      \_ grep 2.sh
```

使用 `top` 命令：`top` 命令会动态地显示系统中运行的进程的信息，包括两个脚本进程及其 CPU 和内存的使用情况。

```
OpenSSH SSH client
top - 17:51:58 up 41 days, 19 min, 1 user, load average: 2.70, 1.65, 1.25
Tasks: 90 total, 4 running, 86 sleeping, 0 stopped, 0 zombie
%Cpu(s): 88.5 us, 11.5 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1890.1 total, 354.9 free, 114.2 used, 1421.0 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 1603.8 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 250347 root        20   0  10628   3380  2028  R   99.7    0.2   3261:40 bash
 255228 root        20   0  10628   2672  1320  R   49.5    0.1    0:58.14 bash
 255229 root        20   0  10628   2672  1320  R   49.5    0.1    0:54.96 bash
 240903 root        10  -10 122204  21636 17572  S    1.0    1.1   66:55.47 AliYunDunMonito
 240892 root        10  -10  88276  14764 12520  S    0.3    0.8   41:47.15 AliYunDun
    1 root        20   0 168868   8568  4480  S    0.0    0.4    0:25.70 systemd
    2 root        20   0      0      0      0  S    0.0    0.0    0:00.30 kthreadd
    3 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 rcu_gp
    4 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 rcu_par_gp
    6 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 kworker/0:0H-kblockd
    8 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 mm_percpu_wq
    9 root        20   0      0      0      0  S    0.0    0.0    0:01.71 ksoftirqd/0
   10 root        20   0      0      0      0  I    0.0    0.0   13:54.22 rcu_sched
   11 root        rt    0      0      0      0  S    0.0    0.0    0:07.06 migration/0
   12 root       -51   0      0      0      0  S    0.0    0.0    0:00.00 idle_inject/0
   14 root        20   0      0      0      0  S    0.0    0.0    0:00.00 cpuhp/0
   15 root        20   0      0      0      0  S    0.0    0.0    0:00.00 cpuhp/1
   16 root       -51   0      0      0      0  S    0.0    0.0    0:00.00 idle_inject/1
   17 root        rt    0      0      0      0  S    0.0    0.0    0:07.12 migration/1
   18 root        20   0      0      0      0  S    0.0    0.0    0:01.76 ksoftirqd/1
   20 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 kworker/1:0H-kblockd
   21 root        20   0      0      0      0  S    0.0    0.0    0:00.00 kdevtmpfs
   22 root         0 -20      0      0      0  I    0.0    0.0    0:00.00 netns
```

使用 `htop` 命令：`htop` 命令提供了一个交互式界面，可以更直观地查看和管理进程，包括两个脚本进程的详细信息。

```
OpenSSH SSH client

1 [|||||||||||||||||||||||||||||||||||||100.0%] Tasks: 30, 61 thr; 2 running
2 [|||||||||||||||||||||||||||||||||||||100.0%] Load average: 2.74 1.70 1.27
Mem[|||||||||||||||||||||||||||||||||117M/1.85G] Uptime: 41 days, 00:19:47
Swp[|||||||||||||||||||||||||||||||||0K/0K]

  PID USER      PRI  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ Command
 250347 root        20   0  10628   3380  2028  R   99.7    0.2 54h21:54 -bash
 255228 root        20   0  10628   2672  1320  R   48.8    0.1  1:04.80 -bash
 255229 root        20   0  10628   2672  1320  R   48.2    0.1  1:01.60 -bash
 240905 root        10  -10  119M  21636 17572  S    0.7    1.1  1:08.30 /usr/local/aegis/aegis_client/aegis_11_91/AliYunDunMoni
 240923 root        10  -10  119M  21636 17572  S    0.7    1.1  1:44.12 /usr/local/aegis/aegis_client/aegis_11_91/AliYunDunMoni
 240903 root        10  -10  119M  21636 17572  S    0.7    1.1 1h06:55 /usr/local/aegis/aegis_client/aegis_11_91/AliYunDunMoni
 255240 root        20   0  10660   4068  3340  R    0.0    0.2  0:00.01 htop
    1 root        20   0   164M   8568  4480  S    0.0    0.4  0:25.70 /sbin/init noibrs
   234 root        19  -1  93116  27512 26452  S    0.0    1.4  0:17.34 /lib/systemd/systemd-journald
   267 root        20   0  21704   2840  1764  S    0.0    0.1  0:08.03 /lib/systemd/systemd-udev
   441 systemd-n  20   0  27404   1748    748  S    0.0    0.1  0:09.52 /lib/systemd/systemd-networkd
   457 systemd-r  20   0  24680   4588    372  S    0.0    0.2  0:35.74 /lib/systemd/systemd-resolved
   507 root        20   0   232M   3824  2884  S    0.0    0.2  0:49.27 /usr/lib/accountsservice/accounts-daemon
   570 root        20   0   232M   3824  2884  S    0.0    0.2  0:00.03 /usr/lib/accountsservice/accounts-daemon
   472 root        20   0   232M   3824  2884  S    0.0    0.2  0:49.81 /usr/lib/accountsservice/accounts-daemon
   478 root        20   0   9416   1304   1040  S    0.0    0.1  0:05.39 /usr/sbin/cron -f
   479 messagebu  20   0   7412   2356  1692  S    0.0    0.1  0:01.44 /usr/bin/dbus-daemon --system --address=systemd: --nofo
   501 root        20   0  32000   8040     4  S    0.0    0.4  0:00.05 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-sta
   564 syslog     20   0   219M  1552     0  S    0.0    0.1  0:01.48 /usr/sbin/rsyslogd -n -iNONE
   565 syslog     20   0   219M  1552     0  S    0.0    0.1  0:00.00 /usr/sbin/rsyslogd -n -iNONE
   566 syslog     20   0   219M  1552     0  S    0.0    0.1  0:01.30 /usr/sbin/rsyslogd -n -iNONE
   503 syslog     20   0   219M  1552     0  S    0.0    0.1  0:02.98 /usr/sbin/rsyslogd -n -iNONE
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit
```

6. 使用上题中的 `1.sh` 或者 `2.sh`，运行实例说明什么情况下使用 `nohup`

如果让程序始终在后台执行，即使关闭当前的终端也执行

在终端中输入：


```
1 nohup ./1.sh > 1.txt &
```

程序成功在后台运行，记录 PID

```
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# nohup ./1.sh > 1.txt &
[1] 255375
root@iZ2ze0olgnf08nfb73yrokZ:/home/alanshen# nohup: ignoring input and redirecting stderr to stdout
```

重启终端，输入 **top** 命令：

可以看到之前的程序一直在后台执行：

```
OpenSSH SSH client  ×  +  ▾
top - 18:00:40 up 41 days, 28 min,  1 user,  load average: 0.89, 1.20, 1.26
Tasks: 92 total,  3 running, 89 sleeping,  0 stopped,  0 zombie
%Cpu(s): 22.6 us, 28.1 sy,  0.0 ni, 49.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 1890.1 total,  448.2 free,  111.7 used,  1330.2 buff/cache
MiB Swap:  0.0 total,  0.0 free,  0.0 used. 1607.3 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 255375 root        20   0    2608    532    464 R   99.7   0.0    1:29.25 sh
 240903 root        10  -10   122204  21636  17572 R    0.7   1.1   66:58.63 AliYunDunMonito
 240858 root        10  -10   42552    7012   6032 S    0.3   0.4    6:59.34 AliYunDunUpdate
 240892 root        10  -10   88276   14764  12520 S    0.3   0.8   41:49.13 AliYunDun
 255085 root        20   0      0      0      0 I    0.3   0.0    0:03.58 kworker/0:0-events
   1 root        20   0   168868   8568   4480 S    0.0   0.4    0:25.71 systemd
   2 root        20   0      0      0      0 S    0.0   0.0    0:00.30 kthreadd
   3 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 rcu_gp
   4 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 rcu_par_gp
   6 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 kworker/0:0H-kblockd
   8 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 mm_percpu_wq
   9 root        20   0      0      0      0 S    0.0   0.0    0:01.71 ksoftirqd/0
  10 root        20   0      0      0      0 I    0.0   0.0   13:54.30 rcu_sched
  11 root         rt   0      0      0      0 S    0.0   0.0    0:07.06 migration/0
  12 root       -51   0      0      0      0 S    0.0   0.0    0:00.00 idle_inject/0
  14 root        20   0      0      0      0 S    0.0   0.0    0:00.00 cpuhp/0
  15 root        20   0      0      0      0 S    0.0   0.0    0:00.00 cpuhp/1
  16 root       -51   0      0      0      0 S    0.0   0.0    0:00.00 idle_inject/1
  17 root         rt   0      0      0      0 S    0.0   0.0    0:07.12 migration/1
  18 root        20   0      0      0      0 S    0.0   0.0    0:01.76 ksoftirqd/1
  20 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 kworker/1:0H-kblockd
  21 root        20   0      0      0      0 S    0.0   0.0    0:00.00 kdevtmpfs
  22 root         0  -20      0      0      0 I    0.0   0.0    0:00.00 netns
```