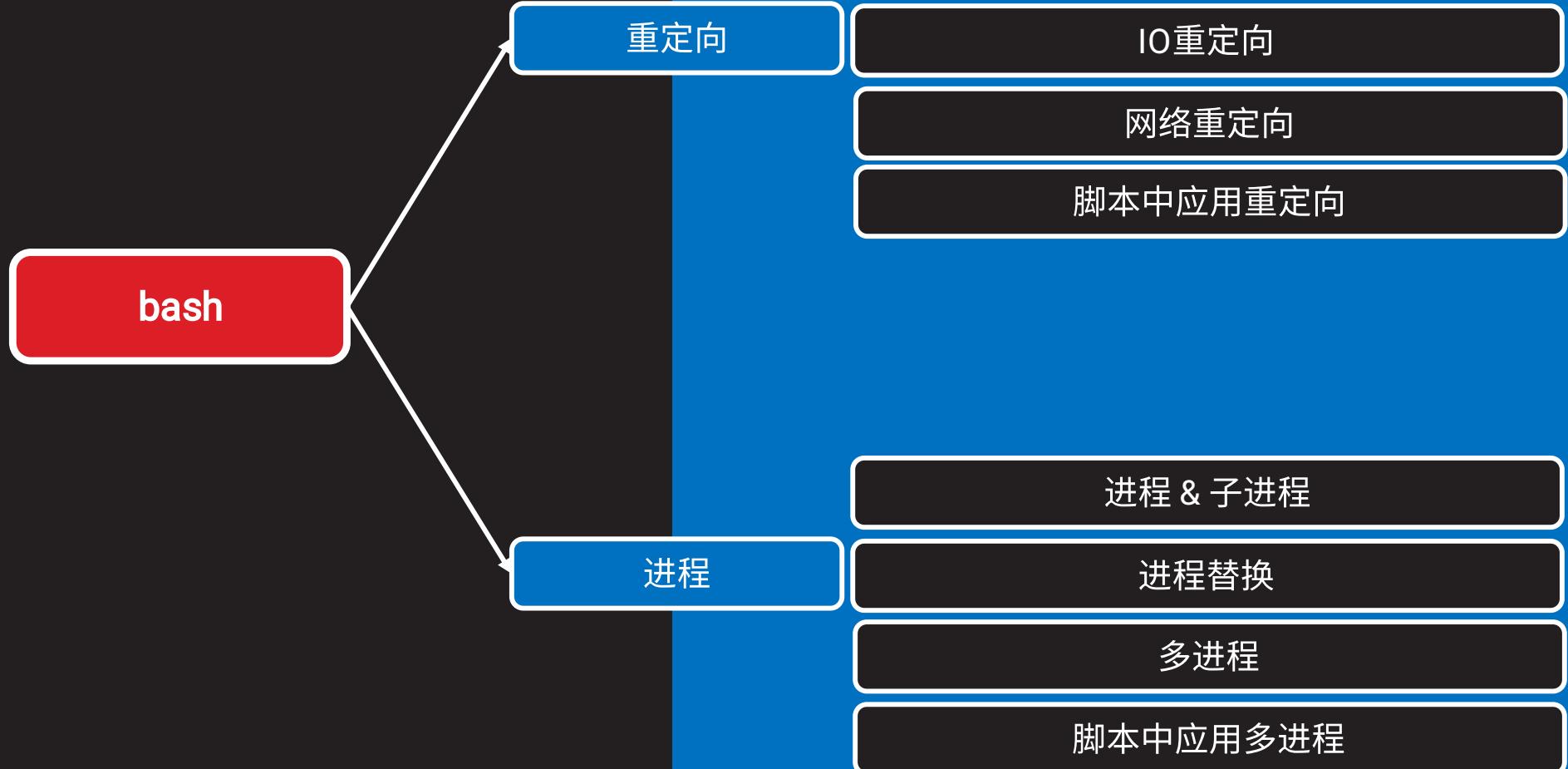


运维工作中脚本使用

SHELL

SHELL



SHELL 脚本

介绍

知识讲解

- 为什么选择 shell?

- UNIX/Linux 最重要的软件之一就是 shell，目前最流行的 shell 被称为 Bash 几乎所有的 Linux 和绝大部分的 UNIX 都可以使用 Bash。作为系统与用户之间的交互接口，因此学好 shell，是学习 Linux/UNIX 的开始，并且它会始终伴随你的工作学习
- 无论你是新手还是老手，都希望你能从今天的分享中获得你们需要的东西



重定向

重定向

- 在默认的情况下，我们登录使用系统，总有三个默认打开的文件描述符 0、1、2 分别对应 `stdin`、`stdout`、`stderr`
- 这三个文件描述符也我们经常用到的
- 多个用户同时登录到系统，每个用户都有自己的 `stdin`、`stdout`、`stderr`，对应的都是 0、1、2
- 每个用户有自己的终端，对应不同的进程



重定向

- 单行重定向(每次执行完后重定向都会 reset)
 - command > urfile
 - command 2>urfile
 - command 1>urfile 2<&1
 - command &>urfile
 - :>urfile
 - command < urfile >output
 - command >>urfile
 - (noclobber, 允许追加, 防止覆盖)



重定向

知识讲解

- > 与 <
 - > 标准输出，标准错误重定向
 - < 标准输入重定向
 - <(read) >(write)
- 如何关闭文件描述符
 - n<&- 关闭输入文件描述符 n.
 - 0<&-, <&- 关闭 stdin.
 - n>&- 关闭输出文件描述符 n.
 - 1>&-, >&- 关闭 stdout



重定向

- 这两个命令有区别吗?
 - `ls /tmp >urlog 2<&1`
 - `ls /tmp 2<&1 >urlog`
- 重定向与管道
 - 在两个命令之间使用管道 | 操作符将的一个命令的 stdout 指向第二个命令的 stdin。



重定向

- 管道后端产生了子进程，子进程继承了打开的文件描述符。这就是为什么管道可以工作。下面展示了对 stdin、 stderr 的一些操作

- ls /tmp /abc |grep .
- ls /tmp /abc 2>&1 |grep .
- ls /tmp /abc 3>&1 1>&2 2>&3 3>&- |grep .



重定向

- exec 重定向
 - exec fd < > fd
 - exec 1>./urfile

```
#!/bin/bash
ls
id
```



重定向

- exec 重定向应用之防止子shell
 - #!/bin/bash
 - #exec 4<&0
 - #exec 0</etc/passwd
 - L=0
 - cat /etc/passwd |while read;do
 - ((L++))
 - done
 - #exec 0<&4 4<&-
 - echo \${L}



网络重定向

- exec 重定向应用之网络重定向

- tcp 重定向 /dev/tcp/host/port
- exec 8</>/dev/tcp/ip.xx.xx.xx/port
- udp 重定向 /dev/udp/host/port
- exec 9</>/dev/udp/ip.xx.xx.xx/port



网络重定向

- exec 重定向应用之网络重定向
 - udp 重定向写 rsyslog 日志

```
function sendmsg(){  
    msg=<134>$(date '+%b %d %T') ${HOSTNAME} ${USER}:$1  
    exec 9<>/dev/udp/192.168.4.11/514  
    echo "${msg}" >&9  
    exec 9<&-  
}
```



网络重定向

- exec 重定向应用之网络重定向
 - tcp 重定向访问 web server

```
function gethtml(){  
    [ ${1:0:7} == "http://" ] && url=${1:7} || url=${1}  
    read host url <<<${url}/\//  
    exec 9<>/dev/tcp/${host}/80 8>&1 >&9  
    echo -ne "GET /${url} HTTP/1.1\r\n"  
    echo -ne "Host: ${host}\r\n"  
    echo -ne "Connection: close\r\n\r\n"  
    exec >&8  
    cat <&9  
    exec 9>&- 8>&-  
}
```



进程

进程

- 都是子 shell 的锅

```
X=0
cat urfile |while read;do
    ((X++))
done
echo $X
```

- 一般来讲 () | 都会产生子 shell



进程

- 进程替换
 - 进程替换与命令替换很相似. 命令替换把一个命令的结果赋给一个变量,例如A=\$(grep word urfile).
 - 进程替换则是把一个进程的输出回馈给另一个进程 (换句话说,它把一个命令的结果发送给另一个命令)
 - echo <(:)
 - diff <(cmd1) <(cmd2)
 - tar cf >(gzip -c > file.tgz) urfile



进程

- 进程替换
 - 利用进程替换解决讨厌的子 shell 问题

```
X=0
while read;do
    ((X++))
done <<(cat urfile)
echo ${X}
```



进程

- 子进程不是背锅侠
 - 虽然子进程有很多小坑，但有很多时候我们还非常需要子进程
 - 检测一个网段能 ping 通那些主机

```
for i in 192.168.1.{1..254};do
    { ping -c 2 ${i} &>/dev/null && echo ${i} >>a.log } &
done
wait
```



进程

- 进程太多怎么办?
 - 控制进程总数

```
Number="1000";
Proc="10";
Timeout="1";
mknod ${PID} p ;exec 4<>${PID}
for((i=0;i<Proc;i++));do echo "OK" >&4;done
for((i=0;i<Number;i++));do
    read;
    ( RunCmd; echo "OK" >&4 ) &
done <&4
wait
kill $(jobs -p)
```



进程

- shell 还能干什么？

- .(){ .|. & };.

感谢大家观看