



**实训总结**

|  |  |
| --- | --- |
| **学习课程：** | shell脚本编程实训 |
| **学 院：** | 数学与信息工程学院 |
| **班 级：** | 2020级网络2班 |
| **学 号：** | 2020531214 |
| **姓 名：** | 肖文杰 |
| **实习企业：** | 达内集团股份有限公司 |
| **学习地点：** | 线上 |
| **学习方式：** | 校内 |
| **学习时间：** | 2023年3月4日至2023年3月10日 |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第1天） | | 2022年3月 4日 |
| 主题 | 案例1:逢3或3的倍数，就输出“过！”，否者就输出当前数字，并让用户重新输入！ | |
| 内容 | #!/bin/bash  while true; do  read -p "请输入一个数字：" num  if [ $(($num%3)) -eq 0 ] || [[ $num =~ "3" ]]; then  echo "过！"  else  echo $num  continue  fi  done  这个脚本会不断地让用户输入数字，并判断这个数字是否为3的倍数或者包含数字3。如果是，就输出“过！”；否则，输出这个数字，并让用户重新输入。  请注意，在判断是否为3的倍数时，我们使用了取余运算符“%”。如果一个数能被3整除，那么它对3取余的结果一定为0。因此，我们只需要检查余数是否为0即可。另外，我们还使用了正则表达式来判断一个数字是否包含数字3。如果一个数字包含数字3，那么它一定会出现在这个数字的某一位上。因此，我们可以使用正则表达式“=~”来判断数字是否包含数字3 | |
| 实习总结 | 本案例主要练习了while循环的用法在判断是否为3的倍数时，我们使用了取余运算符“%”。如果一个数能被3整除，那么它对3取余的结果一定为0。因此，我们只需要检查余数是否为0即可。另外，我们还使用了正则表达式来判断一个数字是否包含数字3。如果一个数字包含数字3，那么它一定会出现在这个数字的某一位上。因此，我们可以使用正则表达式“=~”来判断数字是否包含数字3 | |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第2天） | | 2023年3月 5 日 |
| 目的 | 案例2:利用shell编写一个爬虫软件，获取https://www.ldxy.edu.cn/ 官网首页所有图片内容 | |
| 内容 | #!/bin/bash  # 设置爬虫参数  url="https://www.ldxy.edu.cn/"  user\_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"  referer="https://www.baidu.com/"  # 下载网页内容  curl -A "$user\_agent" -e "$referer" -s "$url" > page.html  # 提取所有图片链接  grep -o '<img[^>]\*src="[^"]\*"' page.html | sed 's/<img[^>]\*src="//g' | sed 's/"$//' > images.txt  # 下载所有图片  while read image\_url; do  # 生成随机休眠时间（1~3秒）  sleep\_time=$(( (RANDOM % 3) + 1 ))  echo "下载图片: $image\_url，等待 $sleep\_time 秒后继续..."  sleep "$sleep\_time"  wget -p /images -q "$image\_url"  done < images.txt  # 清理临时文件  rm page.html images.txt  这段代码是一个 Bash 脚本，主要用于爬取指定网站的所有图片，并将它们下载到本地的 /images 目录中。  下面是每个步骤的解释：  设置爬虫参数：设置要爬取的网站 URL，用户代理（User-Agent）以及来源（Referer）。  url="https://www.ldxy.edu.cn/"  user\_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"  referer="https://www.baidu.com/"  下载网页内容：使用 curl 命令下载网站的 HTML 内容，并将结果保存到本地的 page.html 文件中。-A 参数用于设置用户代理，-e 参数用于设置来源，-s 参数用于静默模式，即不输出下载过程中的信息。  curl -A "$user\_agent" -e "$referer" -s "$url" > page.html  提取所有图片链接：使用 grep 命令和正则表达式从 page.html 文件中提取所有的图片链接，并将结果保存到 images.txt 文件中。-o 参数用于仅输出匹配到的内容，不输出整行，这样可以方便后续的字符串处理。正则表达式 <img[^>]\*src="[^"]\*" 用于匹配所有以 <img 开头、包含 src 属性且以双引号结束的 HTML 标签。接着，使用 sed 命令去除所有匹配行中的 <img 和 " 符号，仅保留图片链接。  grep -o '<img[^>]\*src="[^"]\*"' page.html | sed 's/<img[^>]\*src="//g' | sed 's/"$//' > images.txt  下载所有图片：使用 while 循环遍历 images.txt 文件中的每个图片链接，并使用 wget 命令将图片下载到本地的 /images 目录中。-p 参数用于创建递归目录结构，-q 参数用于安静模式，即不输出下载过程中的信息。此外，为了防止下载过于频繁被网站封禁，每次下载前还会生成 1~3 秒的随机休眠时间。  while read image\_url; do  # 生成随机休眠时间（1~3秒）  sleep\_time=$(( (RANDOM % 3) + 1 ))  echo "下载图片: $image\_url，等待 $sleep\_time 秒后继续..."  sleep "$sleep\_time"  wget -p /images -q "$image\_url"  done < images.txt  清理临时文件：删除下载的网页内容文件 page.html 和提取出的图片链接文件 images.txt | |
| 实  习  总 结 | 这段代码是一个 Bash 脚本，主要用于爬取指定网站的所有图片，并将它们下载到本地的 /images 目录中。 | |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第3天） | | 2023年3月 6 日 |
| 目的 | 案例3:写一个shell脚本获取本机MAC地址/ip地址 | |
| 内容 | MAC地址对应于OSI参考模型的第二层数据链路层，交换机维护着计算机MAC地址和自身端口的数据库。同时MAC地址也称为物理地址，是硬件设备如计算机、手机等唯一标识。  IP地址缩写为IP Adress，是一种在Internet上的给主机统一编址的地址格式，也称为网络协议（IP协议）地址。它为互联网上的每一个网络和每一台主机分配一个逻辑地址，常见的IP地址，分为IPv4与IPv6两大类 当前广泛应用的是IPv4，目前IPv4几乎耗尽，下一阶段必然会进行版本升级到IPv6  MAC地址： ifconfig | grep "ether" | awk '{print $2}'  IP地址： ifconfig | grep "inet " | awk '{print $2}'  相关命令解释：  ifconfig 命令用于获取网卡配置与网络状态等信息  grep 文本搜索工具，根据用户指定的“模式（过滤条件)”对目标文本逐行进行匹配检查，打印匹配到的行  print $2：指获取第二列  Awk 行处理器 | |
| 实  习  总 结 | MAC地址对应于OSI参考模型的第二层数据链路层，交换机维护着计算机MAC地址和自身端口的数据库。同时MAC地址也称为物理地址，是硬件设备如计算机、手机等唯一标识。  IP地址缩写为IP Adress，是一种在Internet上的给主机统一编址的地址格式，也称为网络协议（IP协议）地址。它为互联网上的每一个网络和每一台主机分配一个逻辑地址，常见的IP地址，分为IPv4与IPv6两大类 当前广泛应用的是IPv4，目前IPv4几乎耗尽，下一阶段必然会进行版本升级到IPv6 | |
| **学习报告** | | |
| 学习日期（第4天） | | 2023年3月 7 日 |
| 目的 | 案例4:用shell 写一个双色球选好工具.  双色球规则：红色球1-33号 随机选取6个，蓝色1-16随机选取1个 | |
| 内容 | #!/bin/bash  # 定义红球和蓝球的范围  red\_range=({1..33})  blue\_range=({1..16})  # 定义随机选取红球的函数  function pick\_red\_balls {  red\_balls=()  for ((i=0; i<6; i++)); do  ball=${red\_range[$(($RANDOM % ${#red\_range[@]}))]}  while [[ "${red\_balls[@]}" =~ "$ball" ]]; do  ball=${red\_range[$(($RANDOM % ${#red\_range[@]}))]}  done  red\_balls+=($ball)  done  }  # 定义随机选取蓝球的函数  function pick\_blue\_ball {  blue\_ball=${blue\_range[$(($RANDOM % ${#blue\_range[@]}))]}  }  # 输出选号结果  pick\_red\_balls  pick\_blue\_ball  echo "红球：${red\_balls[@]}，蓝球：$blue\_ball"  在脚本中，首先定义了红球和蓝球的范围，即1-33和1-16。然后，定义了两个函数：  pick\_red\_balls函数：用于随机选取6个不重复的红球。在函数内部，使用for循环和$RANDOM来生成随机数字，并使用while循环来确保生成的数字不会与已选的红球重复。最后，将选取的6个红球存储到一个数组中。  pick\_blue\_ball函数：用于随机选取一个蓝球。在函数内部，同样使用$RANDOM来生成随机数字，然后将选取的蓝球存储到一个变量中。  最后，将选号结果输出到屏幕上，包括选取的6个红球和1个蓝球 | |
| 实  习  总 结 | 在脚本中，首先定义了红球和蓝球的范围，即1-33和1-16。然后，定义了两个函数：  pick\_red\_balls函数：用于随机选取6个不重复的红球。在函数内部，使用for循环和$RANDOM来生成随机数字，并使用while循环来确保生成的数字不会与已选的红球重复。最后，将选取的6个红球存储到一个数组中。  pick\_blue\_ball函数：用于随机选取一个蓝球。在函数内部，同样使用$RANDOM来生成随机数字，然后将选取的蓝球存储到一个变量中。  最后，将选号结果输出到屏幕上，包括选取的6个红球和1个蓝球 | |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第5天） | | 2023年3月 8日 |
| 目的 | 案例5:获取当前系统打开的端口号，选取其中单数的端口号，并为其添加防火墙禁用规则 | |
| 内容 | #!/bin/bash  # 获取当前系统打开的端口号  open\_ports=$(sudo netstat -tlnp | awk '{print $4}' | grep -oE "[0-9]+$")  # 选取单数的端口号  odd\_ports=$(echo $open\_ports | tr ' ' '\n' | grep -oE "^[0-9]+$" | grep -oE "[0-9]+" | awk '$1%2==1')  # 为单数端口添加防火墙规则  for port in $odd\_ports; do  iptables -A INPUT -p tcp --dport $port -j DROP  echo "禁用端口 $port"  done  当需要获取系统上打开的端口号并禁用其中一些端口时，可以使用一个脚本来实现。上面的脚本用于获取当前系统打开的端口号，选取其中单数的端口号，并为其添加防火墙禁用规则。  下面是脚本的详细解释：  #!/bin/bash  这一行指定了脚本要使用的解释器，这里是Bash shell。  open\_ports=$(sudo netstat -tlnp | awk '{print $4}' | grep -oE "[0-9]+$")  这行命令使用netstat命令获取当前系统打开的端口号，然后使用awk和grep命令对输出进行格式化和筛选，最终得到一个包含所有打开端口的列表。具体来说：  。  netstat -tlnp选项用于列出所有正在监听的TCP端口，并显示相关进程的PID和名称。  awk '{print $4}'用于从netstat输出的每一行中提取第四个字段，即端口号和IP地址。  grep -oE "[0-9]+$"用于从awk输出的每一行中提取一个由数字组成的字符串，即端口号。其中，-o选项用于只显示匹配的文本，-E选项用于启用正则表达式匹配。  最终，将所有端口号存储在变量open\_ports中。  odd\_ports=$(echo $open\_ports | tr ' ' '\n' | grep -oE "^[0-9]+$" | grep -oE "[0-9]+" | awk '$1%2==1')  这行命令用于选取单数的端口号。具体来说：  echo $open\_ports用于将open\_ports变量的值输出到标准输出。  tr ' ' '\n'用于将空格分隔的端口号列表转换为每行一个端口号的列表，以便进行后续的处理。  grep -oE "^[0-9]+$"用于筛选只包含数字的行。  grep -oE "[0-9]+"用于从每行中提取出数字。  awk '$1%2==1'用于筛选只包含奇数的数字。其中，$1表示每行的第一个字段，即端口号，%2==1表示只选取余数为1的端口号，即奇数。  最终，将所有奇数端口号存储在变量odd\_ports中。  for port in $odd\_ports; do  iptables -A INPUT -p tcp --dport $port -j DROP  echo "禁用端口 $port"  done  这个for循环用于为每个奇数端口  iptables -L用这个查看 | |
| 实  习  总 结 | 需要获取系统上打开的端口号并禁用其中一些端口时，可以使用一个脚本来实现。上面的脚本用于获取当前系统打开的端口号，选取其中单数的端口号，并为其添加防火墙禁用规则。 | |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第6天） | | 2023年3月 9 日 |
| 目的 | 案例6:在虚拟机添加一个磁盘并自动对磁盘分区、格式化、挂载 | |
| 内容 | #!/bin/bash  # 定义磁盘设备名称和挂载点  disk\_name="sdb"  mount\_point="/mnt/data"  # 创建分区  echo "n  p  1  w" | fdisk /dev/$disk\_name  # 格式化分区为 ext4 文件系统  mkfs.ext4 /dev/${disk\_name}1  # 挂载分区到指定挂载点  mkdir -p $mount\_point  mount /dev/${disk\_name}1 $mount\_point  # 将分区挂载信息写入 /etc/fstab 文件，使其开机自动挂载  echo "/dev/${disk\_name}1 $mount\_point ext4 defaults 0 0" >> /etc/fstab  echo "磁盘已添加、分区、格式化并挂载到 $mount\_point"  脚本的作用是在虚拟机中添加一个磁盘（假设为 /dev/sdb），然后将其分区、格式化为 ext4 文件系统，并挂载到指定的挂载点（例如 /mnt/data）。最后，将挂载信息写入 /etc/fstab 文件，以便在下次开机时自动挂载该磁盘。  注意，此脚本假设你已经知道要添加的磁盘设备名称和挂载点，并且已经掌握了 fdisk 和 mkfs 等命令的使用方法。在实际使用中，请根据自己的需求进行修改。  这个脚本具有一定的风险，如果设备上原有数据会被格式化，如果挂载点目录已存在会被删除，因此在运行之前请确保备份重要数据并确认挂载点目录不存在或者不会影响其他程序的正常运行。  这里是脚本的详细解释：  #!/bin/bash  # 定义磁盘设备名称和挂载点  disk\_name="sdb"  mount\_point="/mnt/data"  这里定义了磁盘设备名称和挂载点。disk\_name 表示磁盘设备的名称，mount\_point 表示将要挂载到的目录。  # 创建分区  echo "n  p  1  w" | fdisk /dev/$disk\_name  这里使用了 echo 命令和管道 | 将分区的命令传递给 fdisk 命令。fdisk 命令用于对磁盘进行分区。n 表示创建新分区，p 表示创建主分区，1 表示新分区的编号为 1，后面的两个空行表示使用默认值。最后的 w 表示将分区表写入磁盘。  # 格式化分区为 ext4 文件系统  mkfs.ext4 /dev/${disk\_name}1  这里使用了 mkfs.ext4 命令将分区格式化为 ext4 文件系统。  # 挂载分区到指定挂载点  mkdir -p $mount\_point  mount /dev/${disk\_name}1 $mount\_point  这里使用了 mkdir -p 命令创建挂载点目录（如果不存在），然后使用 mount 命令将分区挂载到指定的挂载点。  # 将分区挂载信息写入 /etc/fstab 文件，使其开机自动挂载  echo "/dev/${disk\_name}1 $mount\_point ext4 defaults 0 0" >> /etc/fstab  这里使用了 echo 命令将分区挂载信息写入 /etc/fstab 文件，以使其在开机时自动挂载。/dev/${disk\_name}1 表示要挂载的分区设备名，$mount\_point 表示挂载点目录，ext4 表示文件系统类型，defaults 表示挂载选项，0 0 表示备份选项。  echo "磁盘已添加、分区、格式化并挂载到 $mount | |
| 实  习  总 结 | 脚本的作用是在虚拟机中添加一个磁盘（假设为 /dev/sdb），然后将其分区、格式化为 ext4 文件系统，并挂载到指定的挂载点（例如 /mnt/data）。最后，将挂载信息写入 /etc/fstab 文件，以便在下次开机时自动挂载该磁盘。  注意，此脚本假设你已经知道要添加的磁盘设备名称和挂载点，并且已经掌握了 fdisk 和 mkfs 等命令的使用方法。在实际使用中，请根据自己的需求进行修改。  这个脚本具有一定的风险，如果设备上原有数据会被格式化，如果挂载点目录已存在会被删除，因此在运行之前请确保备份重要数据并确认挂载点目录不存在或者不会影响其他程序的正常运行。 | |

|  |  |  |
| --- | --- | --- |
| **学习报告** | | |
| 学习日期（第7天） | | 2023年3月 10 日 |
| 目的 | 案例7:切割 Nginx 日志文件(防止单个文件过大,后期处理很困难) | |
| 内容 | #!/bin/bash  LOG\_PATH=/usr/local/nginx/logs/  YESTERDAY=$(date -d "yesterday" +%Y-%m-%d)  PID=/usr/local/nginx/logs/nginx.pid  mv ${LOG\_PATH}access.log ${LOG\_PATH}access-${YESTERDAY}.log  mv ${LOG\_PATH}error.log ${LOG\_PATH}error-${YESTERDAY}.log  kill -USR1 `cat ${PID}`  此脚本用于自动分割Nginx的日志，包括access.log和error.log  每天00:00执行此脚本 将前一天的access.log重命名为access-xxxx-xx-xx.log格式，并重新打开日志文件  #!/bin/bash  #Nginx日志文件所在目录  LOG\_PATH=/usr/local/nginx/logs/  #获取昨天的日期  YESTERDAY=$(date -d "yesterday" +%Y-%m-%d)  #获取pid文件路径  PID=/usr/local/nginx/logs/nginx.pid  #分割日志  mv ${LOG\_PATH}access.log ${LOG\_PATH}access-${YESTERDAY}.log  mv ${LOG\_PATH}error.log ${LOG\_PATH}error-${YESTERDAY}.log  #向Nginx主进程发送USR1信号，重新打开日志文件  kill -USR1 `cat ${PID}`  然后配置crontab定时任务即可  配置文件切割日志：  在http块种如下配置：  log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '  '$status $body\_bytes\_sent "$http\_referer" '  '"$http\_user\_agent" "$http\_x\_forwarded\_for"';    map $time\_iso8601 $logdate {  '~^(?<ymd>\d{4}-\d{2}-\d{2})' $ymd;  default 'date-not-found';  }    access\_log logs/access-$logdate.log main;  open\_log\_file\_cache max=10; | |
| 实  习  总 结 | 此脚本用于自动分割Nginx的日志，包括access.log和error.log  每天00:00执行此脚本 将前一天的access.log重命名为access-xxxx-xx-xx.log格式，并重新打开日志文件 | |