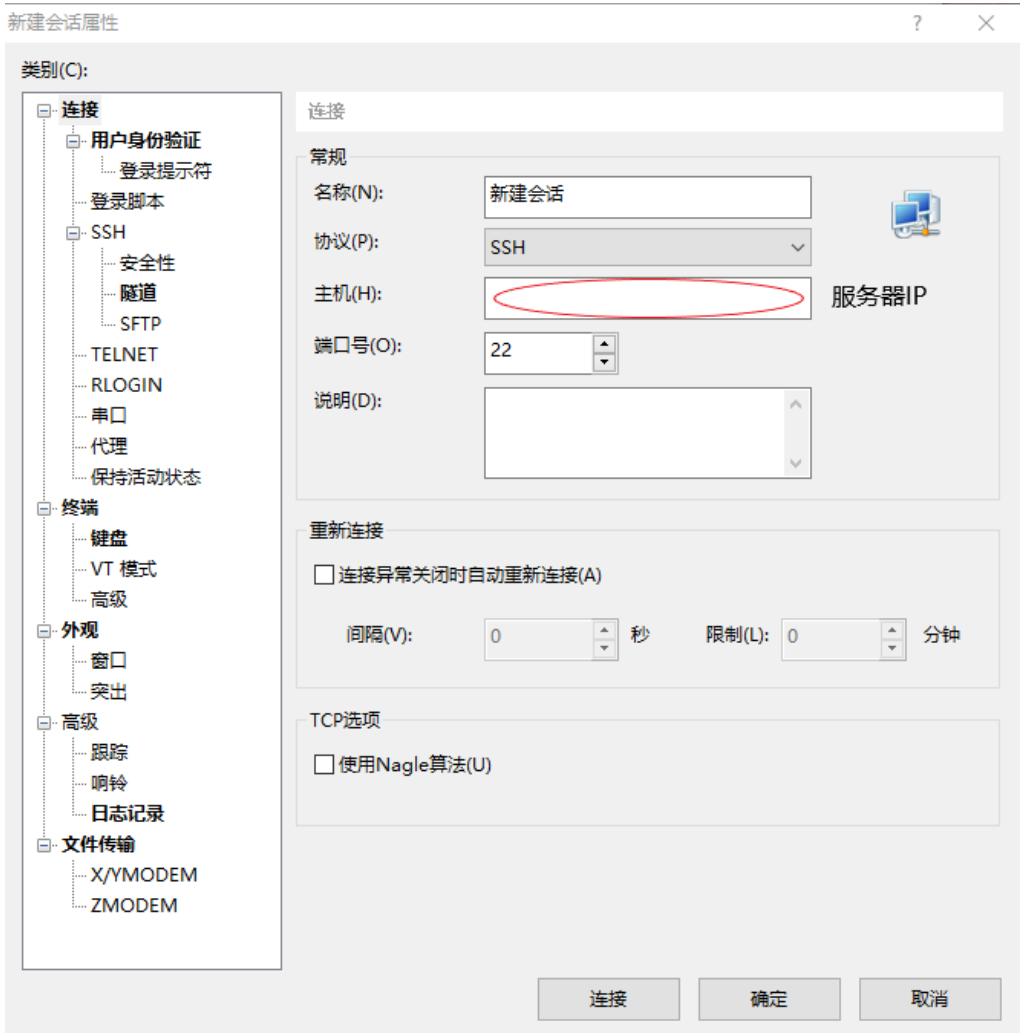
实验环境简介

November 18, 2020

如何登录 linux 服务器

首先，使用在 windows 机器上启动 VMware 程序，开启 ubuntu 镜像机器。当 ubuntu 操作系统启动起来时，其 sshd 服务会随着开机自动起来。由于，安装的 ubuntu 服务器是无图形界面的，因此我们使用 ssh 客户端登录 ubuntu 服务器上。为了简化，我们使用 root 账号登录。ssh 登录成功后，我们在机器上启动 hadoop 服务。

如何登录 linux 服务器



如何登录 linux 服务器

```
Last login: Thu Nov  5 21:26:32 2020 from 192.168.254.1
root@ubuntu:~# ls
hadoop-2.7.7.tar.gz  test.txt
root@ubuntu:~# whereis hadoop
hadoop: /opt/hadoop-2.7.7/bin/hadoop.cmd /opt/hadoop-2.7.7/bin/hadoop
root@ubuntu:~# cd /opt/hadoop-2.7.7/
root@ubuntu:/opt/hadoop-2.7.7# l
bin/  etc/  include/  input/  lib/  libexec/  LICENSE.txt  logs/  NOTICE.txt  README.txt  sbin/  share/
root@ubuntu:/opt/hadoop-2.7.7# clear
root@ubuntu:/opt/hadoop-2.7.7# l
bin/  etc/  include/  input/  lib/  libexec/  LICENSE.txt  logs/  NOTICE.txt  README.txt  sbin/  share/
root@ubuntu:/opt/hadoop-2.7.7# cd sbin/
root@ubuntu:/opt/hadoop-2.7.7/sbin# ls
distribute-exclude.sh  hdfs-config.cmd  kms.sh                    slaves.sh      start-balancer.sh  start-secure-
hadoop-daemon.sh       hdfs-config.sh   mr-jobhistory-daemon.sh  start-all.cmd  start-dfs.cmd      start-yarn.cm
hadoop-daemons.sh      httpfs.sh        refresh-namenodes.sh     start-all.sh   start-dfs.sh       start-yarn.sh
root@ubuntu:/opt/hadoop-2.7.7/sbin# ./start-dfs.sh
Starting namenodes on [localhost]
root@localhost's password:
localhost: starting namenode, logging to /opt/hadoop-2.7.7/logs/hadoop-root-namenode-ubuntu.out
root@localhost's password:
localhost: starting datanode, logging to /opt/hadoop-2.7.7/logs/hadoop-root-datanode-ubuntu.out
Starting secondary namenodes [0.0.0.0]
root@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop-2.7.7/logs/hadoop-root-secondarynamenode-ubuntu.out
```

本实验的 hadoop 配置

伪分布式配置 (Pseudo-Distributed Operation)

本次实验我们使用伪分布 hadoop，整个运行在单一物理机器上。NameNode 运行在一个 java 进程中，DataNode 也运行在一个 java 进程中。
使用如下配置：

编辑 $HADOOP_PREFIX/etc/hadoop/core-site.xml文件:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

编辑 $HADOOP_PREFIX/etc/hadoop/hdfs-site.xml文件:

```
<configuration>
  <property>
   <name>dfs.replication</name>
   <value>1</value>
  </property>
</configuration>
```

实验开启步骤总结

实验开启步骤可以总结如下：

1. 从 VMware 虚拟机软件上，启动 ubuntu 镜像服务器。
2. 从 windows 本机上，使用 Xshell 软件通过 ssh 协议，使用 root 账号登录服务器。
3. 登录成功后，切换到/opt/hadoop-2.7.7/目录下，通过 bin/hdfs namenode -format 和 sbin/start-dfs.sh 开启 nameNode 和 dataNode 服务。

本实验的 hadoop 配置

如果启动成功后，如图所示：

```
bash-4.1# ./start-dfs.sh
Starting namenodes on [208c73071e94]
208c73071e94: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-208c73071e94.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-208c73071e94.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-208c73071e94.out
bash-4.1# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Nov03 ?        00:00:00 /bin/bash /etc/bootstrap.sh -d
root        25     1  0 Nov03 ?        00:00:00 /usr/sbin/sshd
root      8516     1  0 07:30 ?        00:00:00 sleep 1000
root      8517     0  0 07:30 pts/0    00:00:00 bash
root      9536     1 21 07:33 ?        00:00:03 /usr/java/default/bin/java -Dproc_namenode -Xmx1000m -Djava.net.preferIPv4Stack=true -Dhadoop.log.dir=/usr/l
root      9673     1 24 07:33 ?        00:00:03 /usr/java/default/bin/java -Dproc_datanode -Xmx1000m -Djava.net.preferIPv4Stack=true -Dhadoop.log.dir=/usr/l
root      9862     1 34 07:33 ?        00:00:02 /usr/java/default/bin/java -Dproc_secondarynamenode -Xmx1000m -Djava.net.preferIPv4Stack=true -Dhadoop.log.d
root     10013  8517  0 07:33 pts/0    00:00:00 ps -ef
bash-4.1#
```

会出现三个 java 进程，

- namenode 节点的进程。
- datanode 节点的进程。
- secondarynamenode 节点的进程。

bash 简介

Bash 是一个 sh 兼容的命令执行引擎，能够从标准输入或者文件中执行命令。

Shell 语法

主要包含 5 块：

- Simple Commands: 就是一个简单的命令行，包括命令名称，传入的参数列表，和控制符。
- Pipelines: 是由一个或多个控制符 (| 或者 |&) 分隔的一个或者多个命令的序列。
- Lists: 由一个或多个管道组成的序列，由 (;,&,&&,||) 中的一个分隔开。
- Compound Commands: 是 Shell 的编程语言结构，实现编程的基本功能。

bash 简介

Pipelines 管道

基本用法：

$$command \quad | \quad comand2$$

将 $command$ 的标准输出通过管道连接到 $command2$ 的标准输入。

bash 简介

Lists

如果一个命令是以 $\&$ 控制符结尾的，Bash 会在启动一个后台子 Shell，在这个后台子 Shell 中执行这个命令。Shell 不等待这个命令执行完，直接返回 0。

$$command1\ \&$$

如果一个命令是以 ; 控制符结尾的，这个命令是被顺序执行的。

$$command1\ ;\ command2$$

AND list 的形式如下：

$$command1\ \&\&\ command2$$

只有当 command1 的返回状态为 0 成功时，command2 才开始执行。

bash 简介

Lists

OR list 的形式如下：

$$command1 \mid\mid command2$$

只有当 command1 的返回状态非 0 失败时，command2 才开始执行。

bash 简介

Compound Commands

$$((expression))$$

在数学规则下，对 expression 中的内容进行求值。如果，在 expression 中的值是非零的，返回值为 0；否则，返回值为 1。

$$[expression]$$

根据条件表示式的评估，返回值为 0 或者 1。

$$for((expr1; expr2; expr3)); do\ list; done$$

先对算术表达式 expr1 评估，然后，再对算术表达式 expr2 评估，直到其值为 0。每次，expr2 的评估值为非零时，list 命令进行执行，同时算术表达式 expr3 进行评估。

bash 简介

Compound Commands

$$if\ list;\ then\ list;\ else\ list;\ fi$$

if 的 list 先执行，如果 list 命令的结束状态值是零，then 的 list 被执行；否则执行 else 的 list。

$$while\ list1;\ do\ list2;\ done$$

只要 list1 命令的结束状态值是零，那么就不断的执行 list2 命令。

bash 简介

Compound Commands 简单例子

```
for((i = 0; i < 3; i++)) do echo $i; done
```
输出结果:
```
0
1
2
```

```
i=0; if ((i < 2)); then echo "hello 1"; else echo "hello 0";fi
```
输出结果:
```
hello1
```

```
i=0; while ((i++ < 3)); do echo $i; done
```
输出结果:
```
0
1
2
```

```
i=0; while ((i++ < 3)); do sleep 1 && echo $1; done
```
输出结果为每隔一秒打印i的值:
```
0
1
2
```

```
if [ -e Intro.tex ]; then echo "Intro.tex 文件存在"; else echo "Intro.tex 文件不存在"; fi
```
如果当前目录下存在文件Intro.tex，输出结果为:
```
Intro.tex 文件存在
```
; 否则输出结果为:
```
Intro.tex 文件不存在
```

bash 简介

EXPANSION(展开)

当命令行被拆分为单词后，展开操作开始执行。主要包括 7 中展开操作：

- branch expansion
- tilde expansion
- parameter and variable expansion
- command substitution
- arithmetic expansion
- word splitting
- pathname expansion

展开操作执行的顺序为：branch expansion; tilde expansion; parameter and variable expansion; command substitution; arithmetic expansion; word splitting;pathname expansion。

bash 简介

branch expansion

Branch expansion 是一种能够任意生成字符串的展开操作。
branch expanded 中的模式 (Patterns) 可以有一种可选的前言 (preamble)，或者一种可选的后言 (post script)。在展开操作后，前言会变成在花括号中的字符的前缀，后言会变成在花括号中的字符的后缀。
Branch expansion 也可以生成序列，形式为 $\{x..y[..incr]\}$。
一个例子：

```
echo a{1,2,3,4}b;
输出结果为：a1b a2b a3b a4b

echo a{1..4}b;
输出结果为：a1b a2b a3b a4b

将文件ab1,ab2,ab5,ab4移动到文件夹tmp/中：
mv ab{1,2,5,4} tmp/ ;
```

bash 简介

parameter expansion

$$\${parameter}$$

parameter 的值将会替换。parameter 是一个存储值的实体，一个 parameter 被赋值了，那么它就是 (set) 设置了的。如果想要 unset 变量，使用 unset 命令。一个变量可以通过如下方式赋值：

$$\text{name=[value]}$$

注意，等号间没有空格。
一个例子：

```
sen='hello bash'; echo ${sen};
输出结果：
hello bash
```

bash 简介

command substitution

$$\$(command)$$

Bash 执行这种展开操作，通过在子 Shell 环境中执行 command，然后使用 command 的标准输出替换这个 command substitution。如果这个 command substitution 是被双引号包括的，不会在结果中，执行 word splitting 和 pathname expansion。
一个例子：

```
echo "当前的时间为：$(date +%Y/%m/%d\ %H:%M:%S)";
输出结果为：
当前的时间为：2020/11/04 17:13:06
```

bash 简介

arithmetic expansion

算术展开操作可以允许执行算术表达式，并且替换结果。形式如下：

$$\$((expression))$$

这个 expression 就好像是被双引号包括着的，但是在括号内的双引号不被特殊对待。
一个例子：

```
i=6; echo $(( 1 + 2 + 3 + 4 + ${i}));
输出结果为：
16
```

bash 简介

pathname expression

当 word split 过程执行后，bash 会在每个单词中扫描字符 *,? 和 [。如果有这些字符出现，这个单词被认为是一个模式。然后，被匹配的文件名称所代替。

`*`: 匹配任意的字符串，包括空字符。
`?`: 匹配任意一个字符串。
`[...]`: 匹配在`[...]`中任意字符。

一个例子：

打印当前目录下的所有`pdf`文件的名称：

```
echo *.pdf;
```

bash 简介

Shell 参数 (parameter)

Shell 参数主要分成三类：

- 普通参数，由name=[value]形式赋值，通过${parameter}进行访问。
- 位置参数，当一个命令被调用时，其传入的参数列表会被赋予为位置参数列表。通常通过${1}访问第一个传入的参数，${2}访问第二个传入的参数，等等。
- 特殊参数，这些参数只能读取，不能赋值。通常有 $*, @, \#, \$$ 等。

bash 简介

位置参数

当一个命令被调用时，其传入的参数列表会被赋予为位置参数列表。使用${1}访问第一个传入的参数，${2}访问第二个传入的参数。
一个例子：

```
function sayHello() {
  echo "first parameter $2, second parameter $1";
};sayHello a b;
输出结果：
first parameter b, second parameter a
```

bash 简介

特殊参数

- *：扩展成位置参数列表，从 1 开始。如果，是被双引号包括的，"$*" 等价为"$1c$2c..."，其中 c 是 IFS 变量的第一个字符。
- @：扩展成位置参数列表，从 1 开始。如果，是被双引号包括的，"$@" 等价为"$1" "$2" ...。
- #：扩展为位置参数列表的数目。
- $：扩展为当前进程的 id，在 () 子 shell，扩展为当前进程的 id，而不是子 shell 的进程 id。

一个例子：

```
function sayHell() {
    IFS=:; echo "$*"; echo "$@"; echo $#; echo $$;
}; sayHello a b;
输出结果:
a:b
a b
2
12382
```

bash 简介

单引号和双引号

- ' ': 被单引号包括的内容，保持其文本含义，不做 expansion。
- " ": 被双引号包括的内容，除了 $,‘,\ 以外，其他字符保持其文本含义。$ 的作用不发生变化。

一个例子：

```
name=abc; echo '${name}'; echo "${name}";
输出结果:
${name}
abc
```

bash 简介

Shell 函数

定义如下:

function name [( )] compound-command [redirection]

一个简单的例子:

```
function hello() { i=$1; while ((i > 0)); do echo "$i" && i=$((i-1));  done }; hello 5;
```

输出结果:

```
5
4
3
2
1
```

bash 简介

Shell 重定向

在命令执行前，它的输入和输出可以使用 shell 理解的符号进行重定向。重定向允许命令的文件句柄被复制，打开，关闭，可以引用不同的文件，改变读入的文件或者写入的文件。

bash 简介

重定位输入 (Redicting Input)

重定位输入将把 word 单词展开表示的文件打开，以供在文件描述符 n 上读取。重定位输入的格式为：

[n]<word

如果，n 不指定，默认为标准输入 (文件描述符 0)。
一个例子：

```
cat < hello.txt
打印hello.txt的内容。
```

bash 简介

重定位输出 (Redicting Output)

重定向输出将把 word 单词展开表示的文件打开，以供文件描述符 n 上写入内容。重定向输出的格式为：

$$[n]>word$$

如果，n 不指定，默认为标准输出 (文件描述符 1)。
一个例子：

```
echo {a..z} > hello.txt
```

bash 简介

追加输出

追加输出将把 word 单词展开表示的文件打开，以供文件描述符 n 上追加内容。追加输出的格式为：

$$[n] >> word$$

如果，n 不指定，默认为标准输出 (文件描述符 1)。
一个例子：

```
echo {a..z} >> hello.txt
```

bash 简介

文件句柄复制

复制句柄的格式如下：

[n]<&word

用来复制输入的文件描述符。如果，word 单词能被扩展为一个或多个数字，那么，由 n 表示的文件描述符会成为 word 单词表示的描述符号的一份副本。

[n]>&word

用来复制输出的文件描述符。如果，word 单词能被扩展为一个或多个数字，那么，由 n 表示的文件描述符会成为 word 单词表示的描述符号的一份副本。

一个例子：

```
cat > log 2>&1
```

将cat的标准输出和错误输出重定向到文件log中。

linux 下常用命令

文件操作

- mv SOURCE DEST: 将源文件名称修改为目标文件名称，或者将源文件移动到目标目录中。
- ls FILE: 列举当前文件的信息。
- mkdir DIRECTORY: 如果目录不存在，则创建目录。
- cp SOURCE DEST: 拷贝源文件到目标文件，或者拷贝多个源文件到目录中。
- cd dir: 把当前的目录切换到 dir。
- cat FILE: 将文件的内容输出到标准输出。

linux 下常见命令

杂项

- find [starting-point...] [expression]：在 start-point 点下，根据 expression 表达式，查找文件。expression 主要包括 Tests 和 Actions。
  常见 Tests 表达：
  - -name pattern: 基本文件名称匹配 Shell Pattern。
  - -path pattern: 所有文件名称 (包括路径) 匹配 Shell Pattern。
  - -ctime n: 文件的状态在 $n * 24$ 小时前改变。$+n$ 大于 n；$-n$ 小于 n；n 等于 n。

  常见 Actions 表达：
  - -print format : 将匹配的文件列表打印到标准输出。
  - -exec command; : 执行 command 命令。

一个例子：

```
查看当前目录下所有的java文件，打印路径以'\n'分割。
find . -name '*.java' -print
```

linux 下常见命令

杂项

- tar 一个压缩命令。用于，压缩文件；解压文件；查看压缩文件内容等。
  - tar -c [-f ARCHIVE][OPTIONS] [FILES...]: 将文件集合，压缩为一个压缩文件。
  - tar -x [-f ARCHIVE][OPTIONS] [MEMBER...]: 将压缩文件中的成员，解压到本地。
  - tar -t [-f ARCHIVE][OPTIONS] [MEMBER...]: 将压缩文件的成员列举处理。

linux 下常用命令

辅助操作

- man page: 每个 page 的参数通常是程序名或函数名等。用于查看当前当前程序或函数的使用方法。

HDFS 文件系统的 Shell 操作

put

使用方法：bin/hdfs dfs -put <localsrc> ... <dst>
从本地文件系统中复制单个或多个源路径到 HDFS 文件系统。
也支持从标准输入中读取输入写入目标文件系统。实例：

```
bin/hdfs dfs -put tmp.txt  /
#将本地的tmp.txt文件拷贝到HDFS文件系统的根目录下。
```

HDFS 文件系统的 Shell 操作

get

使用方法：bin/hdfs dfs -get [-ignorecrc] [-crc] <src> <localdst>
从 HDFS 文件系统复制文件到本地文件系统。可用-ignorecrc 选项复制 CRC 校验失败的文件。使用-crc 选项复制文件以及 CRC 信息。实例：

```
bin/hdfs dfs -get /tmp.txt .
#将HDFS文件系统上根目录下的tmp.txt拷贝到本地文件系统上。
```

HDFS 文件系统的 Shell 操作

mkdir

使用方法：bin/hdfs dfs -mkdir -p <paths> 接受路径指定的 uri 作为参数，创建这些目录。其行为类似于 Unix 的 mkdir -p，它会创建路径中的各级父目录。实例：

```
bin/hdfs dfs -mkdir /data1/note1 /data1/note2
#创建两个目录，note1和note2
```

HDFS 文件系统的 Shell 操作

ls

使用方法：bin/hdfs dfs -ls <paths> 列举指定的 HDFS 路径的信息。实例：

```
bin/hdfs dfs -ls /
#列举根目录的文件的信息。
```

HDFS 文件系统的 Shell 操作

cat

使用方法：bin/hdfs dfs -cat URI 将路径指定文件的内容输出到 stdout。实例：

```
bin/hdfs dfs -cat /tmp.txt
#输出HDFS文件系统上的文件的内容。
```

HDFS 文件系统的 Shell 操作

rm

使用方法：bin/hdfs dfs -rm -r URI 将路径指定文件删除。实例：

```
bin/hdfs dfs -rm -r data1/note1/tmp.txt
```

HDFS 文件系统的 Shell 操作

一个完整的例子：

```
bash-4.1# bin/hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - root supergroup          0 2015-05-16 05:42 /user
bash-4.1# bin/hdfs dfs -mkdir -p /data1/note1 data2/note2
bash-4.1# bin/hdfs dfs -put tmp.txt /data1/note1
bash-4.1# bin/hdfs dfs -ls /data1/note1
Found 1 items
-rw-r--r--   1 root supergroup         11 2020-10-22 23:42 /data1/note1/tmp.txt
bash-4.1# bin/hdfs dfs -cat /data1/note1/tmp.txt
hello hdfs
bash-4.1# bin/hdfs dfs -get /data1/note1/tmp.txt tmp2.txt
20/10/22 23:43:24 WARN hdfs.DFSClient: DFSInputStream has been closed already
bash-4.1# ls
LICENSE.txt  README.txt  etc      input  libexec  sbin   tmp.txt
NOTICE.txt   bin         include  lib    logs     share  tmp2.txt
bash-4.1# cat tmp2.txt
hello hdfs
bash-4.1#
```