Linux Notes

Claude Lu

October 3, 2024

1 Process

1. Run process in background by adding &:

   ```
   ./your_executable &
   ```

2. **UID**: user ID of the process owner, like *root*, *claude*
3. **PID**: Process ID
4. **PID**: Parent Process ID
5. **C**: CPU utilization
6. **STIME**: start time of the process
7. **TTY**: Terminal associated with the process
8. **TIME**: Total CPU time used
9. **CMD**: Command that started the process

Common usage

```
$ ps -ef
$ ps -fp <process ID>
$ ps -o pid,ppid,tty,time <process ID>
# Show call hierarchy
$ pstree -p <processID>
```

1. -p: required a process ID
2. -o: option, including:
   (a) **pid**
   (b) **ppid**
   (c) **tty**
   (d) **cputime**: CPU time used by the process
   (e) **etime**: Elapsed time in MM:SS
   (f) **stime**: start time of the process
   (g) **args**: command with all its arguments

2 chmod

Use in numeric mode

```
# Open all permission
# Owner(user) | Group | Others
chmod 777 <file>
# r (read)=4
# w (write)=2
# x (execute)=1
```

3 date and time

```
$ date +%[OPTION]
OPTION
%F: %+4Y-%m-%d
%r: 12 hour
```

4 Shell Script

4.1 Rule of Thumb

1. All bash script shall shall start with #!/bin/sh
2. #! Reads **sharp bang** or **Shebang**
3. If you need python to do the work, use #!/usr/bin/python instead

4.2 Special Variables

1. **Individual Arguments**: $1, $2, representing the n-th argument of the bash script, you can think of it as the combination of argc and argv. One can utilize shift command to increment the number of individual arguments by one.
2. **Number of Arguments**: $#
3. **All Arguments**: $@
4. **Script Name**: $0
5. **Process ID**: $$
6. **Exit Code**: $? The exit code holds the last command that shell executed

4.3 self-defined variables

1. No spaces before and after the *equal sign*
2. Variables are case-sensitive, and should be in uppercase

```
#!/bin/bash
# Define your variable
VARIABLE_NAME="VALUE
# Use your variable
echo "This is my variable ${VARIABLE_NAME}"
# assign the output of a command as variable
VARIABLE=$(<command>)
VARIABLE=$‘<command>‘
```

4.4 Conditionals

```
# establish a condition expression between brackets
[ condition-to-test-for ]
```

File operators

1. -d FILE if file is a directory
2. -e FILE if file exists
3. -f FILE if file exists and is a regular file
4. -r FILE if file is readable by you
5. -s FILE if file exists and is not empty
6. -w FILE if file is writable by you
7. -x FILE if file is executable by you

String operators:

1. -z STRING if string is empty
2. -n STRING is string is not empty
3. STRING1 = STRING2 if strings are equal
4. STRING1 != STRING2 if strings are not equal

Arithmeitc operators:

1. arg1 -eq arg2 : arg1 = arg2
2. arg1 -ne arg2 : arg1 != arg2
3. arg1 -lt arg2 : arg1 <arg2
4. arg1 -le arg2 : arg1 <= arg2
5. arg1 -gt arg2 : arg1 >arg2
6. arg1 -ge arg2 : arg1 >= arg2

4.5 if statement

```
# Must be space between conditional and if
# Must have space after left bracket and before right bracket
# Must have space before and after equal sign when used for conditionals
# use && for AND, || for or
if [ condition-true ]
then
  command 1
  command 2
elif [ condition-true ]
then
  command 3
  command 4
else
  command 5
  command 6
fi
```

4.6 for loop

```
# ITEM should be seperated by space
for VARIABLE_NAME in ITEM1 ITEM2 ITEM3
do
  command 1
  command 2
done
# One can store list of items in variable, then iterate
ITEMS="ITEM1 ITEM2 ITEM3"
for ITEM in ${ITMES}
do
  command 1
  command 2
done
```

4.7 read

```
read -p "ENTER THE INPUT: " INPUT
```

4.8 Logical Operator

1. the first second command will execute if and only if the first one exit with 0

   ```
   command1 && command2
   ```

2. the second command will execute if and only if the first one failed, in other word, if the first command succeed, the second one won't execute

   ```
   command1 || command2
   ```

3. two commands will execute no matter what

   ```
   command1 ; command2
   ```

4.9 exit

1. use **exit** command with a number from 0 to 255
2. If no exit code is specified, the previously executed command is used as the exit status

4.10 function

```
# create a function
# Method 1
function funcion-name(){
  # Code
}
# Method 2
function-name(){
  # Code
}
```