

# Linux Notes

Claude Lu

October 17, 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 Package

### 4.1 apt

```
sudo apt update
# upgrade all installed packages to their latest versions
sudo apt upgrade

# install a package
sudo apt install <package1> <package2> <package3>

# remove a package
sudo apt remove <package>

# removes both package and its configuration file
sudo apt purge <package>

# search for a package
apt search <package>

# show information about a package
apt show <package>

# list installed package
apt list --installed

# clean up unused packages and dependencies
sudo apt autoremove

# clean package cache
sudo apt clean

# fix broken dependencies
sudo apt --fix-broken install

# upgrade distribution (don't try easily)
sudo do-release-upgrade
```

```
# install local .deb package
sudo dpkg -i <package.deb>

# fix a package at certain package
sudo apt-mark hold <package>
sudo apt-mark unmode <package>
```

## 5 ssh

1. `ssh` is the *client program* of the OpenSSH
2. To set up an SSH connection, you need the remote host's *public key*
3. Public key cryptography have two keys:
  - (a) public key: can encrypt a message but not decrypt it
  - (b) private: can decrypt a message from the public key
4. the package name of ssh server on debian are `openssh-server`, and its systemd service name is `ssh`
5. Check configurations and hosts keys in `/etc/ssh`:
  - (a) `sshd_config`: server configuration filename
  - (b) `ssh_config`: client setup file
6. OpenSSH Key Files: (rsa and dsa are just different algorithms)
  - (a) `ssh_host_rsa_key`
  - (b) `ssh_host_rsa_key.pub`
  - (c) `ssh_host_dsa_key`
  - (d) `ssh_host_dsa_key.pub`
7. Use `fail2ban` to prevent malicious attacks from the internet

ssh client usage

```
# log in to a remote host
$ ssh remote_username@remote_host
```

scp usage

```
# transfer a file from local to remote
scp /path/to/local/file username@remote:/path/to/remote/directory
# transfer a directory from local to remote
scp -r /path/to/local/file username@remote:/path/to/remote/directory

# transfer a file from remote to local
scp username@remote:/path/to/remote/directory /path/to/local/file
# transfer a directory from remote to local
scp -r username@remote:/path/to/remote/directory /path/to/local/file
```

## 6 USB

### 6.1 99-com.rules

Add symbolic according to physical port. First cd `/etc/udev/rules.d/99-com.rules`

```
SUBSYSTEM=="tty", KERNELS="<your kernel>", SYMLINK+="<your name>"
```

```
# check attribute of a device
udevadm info --name=/dev/ttyACM* --attribute-walk
```

## 7 Shell Script

### 7.1 Rule of Thumb

1. All bash script 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

### 7.2 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

### 7.3 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>'
```

## 7.4 list and arrays

```
# create an array
declare -a ARRAY

# add element to array
ARRAY+=("element1")
```

## 7.5 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 if string is not empty
3. STRING1 = STRING2 if strings are equal
4. STRING1 != STRING2 if strings are not equal

Arithmetic 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

## 7.6 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
```

One can directly utilize the exit code of a command as the condition of if statement:

```
if <command>; then
// your code
fi
```

## 7.7 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 ${ITEMS}
do
    command 1
    command 2
done
```

An array-based for-loop

```
array=(item1 item2 item3)

for item in "${array[@]}"; do
// your code
done
```

A wildcard for loop

```
# Search all .pattern file in the directory same as the bashscript
for FILE in *.pattern
do
// code
done
```

Use pipe | to represent “or” in case

## 7.8 case statements

```
read -p "Enter y or n: " ANSWER
case "$ANSWER" in
  [yY] | [yY] [eE] [sS])
    echo "You enter yes"
    ;;
  [nN] | [nN] [oO])
    echo "You enter no"
    ;;
  *)
    echo "Invalid answer"
    ;;
esac
```

## 7.9 read

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

A common design pattern

```
read -p "Enter y or n: " ANSWER
case "$ANSWER" in
[yY] [])
```

## 7.10 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
```

## 7.11 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

## 7.12 function

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

#Passing arguments
function-name arg1 arg2 arg3
```

## 7.13 wildcard

1. **wildcard** is a character set used to represent one or more characters in file or directory names.
2. \*: asterisk, zero or more character
3. ?: question mark, one character
4. []: square brackets, any single character within the brackets.
5. [!...]: negated square brackets, matches any character **except** those in the character
6. **character class**: ca[nt]\* will match can, cat, canbus, or catch
7. **ranges**: [a-g] match all characters from "a" to "g", [3-6] match the numbers 3, 4, 5
8. **named character classes**:
  - (a) [:alpha:] Matches lowercase and uppercase letters
  - (b) [:alnum:] Matches alpha and digits.
  - (c) [:digit:] Represents numbers in decimal
  - (d) [:upper:] Matches uppercase letters
  - (e) [:lower:] Matches lowercase letters
  - (f) [:space:] Matches whitespace, such as spaces, tabs and newline characters
9. **Matching Wildcard Patterns**: to escape from wildcard patterns, use **backslash**

## 8 Embedded Linux

Four element of embedded Linux:

1. Toolchain: the compiler and other tools needed to create code for the target device
2. Bootloader: the program that initializes the board and loads the Linux kernel



3. Kernel: managing system resources and interfacing with hardware
4. Root filesystem: libraries and programs that are run once kernel has completed initialization

## 8.1 Toolchain

1. Toolchain comprising of the followings:
  - (a) compiler
  - (b) linker
  - (c) runtime libraries
2. [bootloader](#), [kernel](#) and [root filesystem](#) are compiled by toolchain
3. GNU tool chain is composed of three things:
  - (a) **Binutils**
  - (b) **GNU Compiler Collection (GCC)**
  - (c) **C library**: a standardized application program interface (API) based on POSIX specification.
4. **headers** should be from, or older than the kernel your using.
5. **GNU Debugger (GDB)** is usually considered a part of the tool chain.
6. toolchain can be categorized as below:
  - (a) Native: this toolchain runs on the same type of system as the program it generates.
  - (b) Cross: this toolchain runs on a different type of system than the target
7. to build toolchain, must consider the following things:
  - (a) CPU architecture
  - (b) big or little endian
  - (c) floating point support
8. application binary interface (ABI): how different pieces of compiled code (binaries) work together. For example, ARM use **Extended Application Binary Interface (EABI)**
9. The programming interface to Unix operating system is defined in the [C language](#), which is defined by **POSIX**
10. **C Library** is the *implementation* of Portable Operating System Interface (POSIX):
  - (a) **glibc**: use this !
  - (b) **musl libc**: use when storage less than 32 MiB
  - (c) **uClibc-ng**
  - (d) **eglibc**
11. All the applications need to communicate with Linux kernel through the [C library](#)