




Linux Shell Tutorial

2024 3 9

李云广




Linux Shell

- ✓ Brief Intro: all you need to know about starting using a CLI
 - ✓ Basic but useful command line tools
 - ✓ How to write a bash scripts and what can those scripts do?
- 



Target

1. Using shell to get the work done (doing experiment, coding, etc.) efficiently
 2. Using bash script
- 

GUI vs CLI: which is better?

```
1 /*
2  * @Author: lyg 1286285985@qq.com
3  * @Date: 2023-02-22 14:05:39
4  * @LastEditors: lyg 1286285985@qq.com
5  * @LastEditTime: 2023-02-23 11:33:09
6  * @FilePath: \hi3861project\hi3861\src\applications\sample\wi
7  * @Description: 这是默认设置,请设置`customMade`, 打开koroFileHe
8  */
9 #include <stdio.h> // c语言的标准库文件
10 #include <hi_io.h>
11 #include <hi_pwm.h>
12 #include <hi_gpio.h>
13
14 #include "ohos_init.h" // 提供用于openharmony初始化和启动服务
15
16 #define PWM_FREQ_FREQUENCY (60000)
17
18 hi_u16 car_speed;
19
20 hi_void pwm_control(hi_io_name id, hi_u8 val, hi_pwm_port port
21 {
22     hi_io_set_func(id, val);
23     hi_pwm_init(port);
24     hi_pwm_set_clock(PWM_CLK_160M);
25     hi_pwm_start(port, duty, PWM_FREQ_FREQUENCY);
26 }
27
28 hi_void gpio_control(hi_io_name gpio, hi_gpio_idx id, hi_gpio_
29 {
```

```
12 #####
11 # CS:APP Attack Lab
10 # Main makefile
9 #####
8
7 all:
6     @echo "Please specify a rule"
5
4 # Start the Lab by running the main attacklab daemon, which nannies the
3 # request and result servers and the report daemon.
2 start:
1     @touch log.txt
13     @./attacklab.pl -q &
1     @ sleep 1
1
2 # Stops the attack Lab by killing ALL attacklab daemons
4 stop:
5     @killall -q -9 attacklab.pl attacklab-requestd.pl attacklab-reportd.pl \
6     attacklab-resultd.pl ; true
7
8 # Cleans soft state from the directory. You can do this at any time
9 # without hurting anything.
10 clean:
11     rm -f *~
12     rm -f attacklab-scoreboard.html scores.csv
13     (cd src; make clean)
14     (cd writeup; make clean)
15
16 #
17 # Cleans the entire directory tree of ALL soft state, as well as the
18 # hard state releated to a specific instance of the course, such as
19 # targets and log files.
20 #
21 # Do this whenever you need a fresh directory, for example while you're
22 # getting the Lab set up and just testing things out for yourself, or
23 # at the beginning of the term when you need to reset the Lab.
24 #
25 # DON'T DO THIS UNLESS YOU'RE REALLY SURE!
26 #
27 cleanallfiles:
28     rm -rf *~ scores.csv reports/* targets/target* log.txt log-status.txt *.html
Makefile
"Makefile" 46L, 1334B
```



GUI

- Graph (e.g., code analysis tools)
- More intuitive user-interface, especially in complex software

CLI

- Data analysis
- Home-brew app
- When connecting to a server

Shell: The system user-interface in CLI

Just like the Desktop in GUI world (from user's view)

Capability:

- Launch app
- execute command
- manage foreground/background tasks

Basic Setup

Terminal (emulator): emulate a (texted-based) terminal inside the GUI environment.

SSH to server `ssh 2183311128-ics@igw.dfshan.net -p2291`

Running sshd: daemon of SSH server

- **Strong password** or use **ssh key** to login
- • Keyboard shortcuts
 - ctrl + r (to find history), tab (to autofill)
 - ctrl + c (to kill SIGINT)

Install Software in CLI

Package manager: `apt` (ubuntu, Debian), `brew` (macOS), `dnf` (fedora)

- Search (e.g. `apt search`)
- <https://command-not-found.com/>

Build from source (no suitable version, or need to modify their code)

- README/INSTALL doc
- `configure` and `make install`

Basic Tools (Commands)

Directories: `pwd`, `cd`

File: `touch`, `cp`, `mv`, `rm`, `cat`, `less`, `mkdir`

Simple functions: `sort`, `wc`, `echo`

Others: `grep`, `chmod`

Code Editor: `vim`

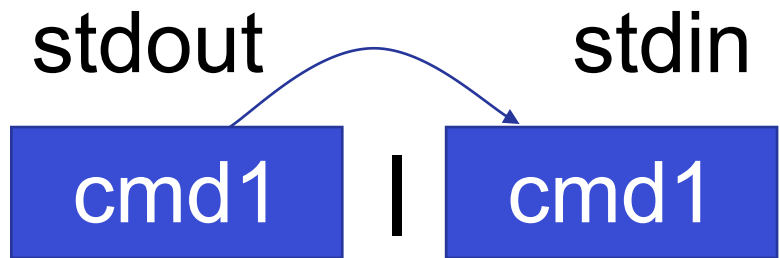
Keep the connection: `tmux`, `screen`, etc.

How to use?

- `-help`, `--help` • `man [command]`
- <https://command-not-found.com/>
- TLDR <https://tldr.sh/>

Communication: Pipe & Redirect

- A lot of CLI tools, communication is required to do complex jobs
- Pipe: | use the stdout of previous command as the stdin of the next

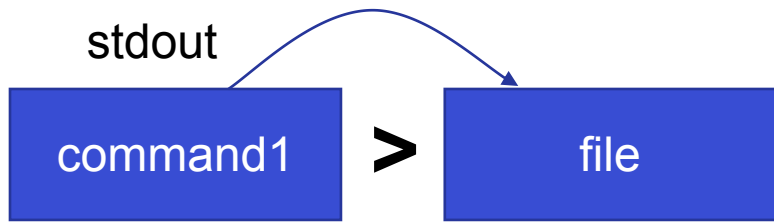


ls | grep "sh"

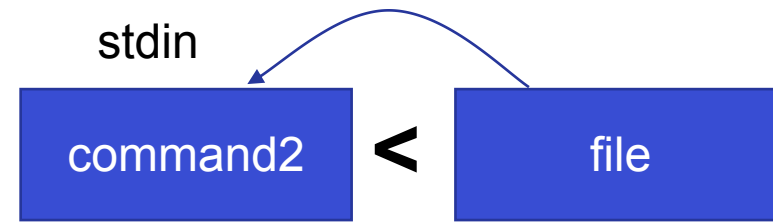
```
ygli@dc2:~/lab1$ ls | grep "sh"  
lab1.sh  
lab2.sh  
rum_dummy.sh  
run_exp.sh  
test.sh  
ygli@dc2:~/lab1$
```

Communication: Pipe & Redirect

- A lot of CLI tools, communication is required to do complex jobs
- **Redirect: > & <, stdout to file or file to stdin (normally)**



ls > ls_out



grep sh < ls_out

```
ygli@dc2:~/lab1$ ls > ls_out
ygli@dc2:~/lab1$ grep sh < ls_out
lab1.sh
lab2.sh
rum_dummy.sh
run_exp.sh
test.sh
ygli@dc2:~/lab1$ █
```

Communication: Pipe & Redirect

- A lot of CLI tools, communication is required to do complex jobs
- Redirect: > & <, stdout to file or file to stdin

- 0 - stdin, the standard input stream.
- 1 - stdout, the standard output stream.
- 2 - stderr, the standard error stream.

```
1 #include <iostream>
2 #include <stdio.h>
3 using namespace std;
4
5 int main(){
6     printf("stdout\n");
7     fprintf(stdout, "stdout\n");
8     fprintf(stderr, "stderr\n");
9     return 0;
10 }
11
```

test1.cpp	11,0-1	All
-----------	--------	-----

```
ygli@dc2:~/lab1$ g++ test1.cpp -o test1
ygli@dc2:~/lab1$ ./test1
stdout
stdout
stderr
ygli@dc2:~/lab1$ ./test1 > test_out
stderr
ygli@dc2:~/lab1$ ./test1 > test_out 2>test_err_out
ygli@dc2:~/lab1$ cat test_err_out
stderr
ygli@dc2:~/lab1$
```

Tar

Usage Scenario: archive files in 1 bundle (and compress them)

Flags

- c: create a tarball
- x: open a tarball
- z: zipped using gzip
- v: verbose mode [displays progress]
- f: specify file name

```
tar -cf name-of-archive.tar /path/to/dir/ compress directory
```

```
tar -cf name-of-archive.tar /path/to/filename compress file
```

```
tar -cf name-of-archive.tar dir1 dir2 dir3 compress multiple directories
```

```
tar -xf name-of-archive.tar open a tar file in current directory
```

Some useful tools

ag

awk

sed

shell script

```
touch lab1.sh
echo hello1
echo hello2
echo hello3
chmod +x lab1.sh
./lab1.sh
```

```
ygli@dc2:~/lab1$ cat lab1.sh
echo hello1
echo hello2
echo hello3

ygli@dc2:~/lab1$ ./lab1.sh
hello1
hello2
hello3
ygli@dc2:~/lab1$ █
```

shell script

- With local variables
- Passing in as an arguments

lab2.sh

```
hello1=$1  
hello2=$2  
hello3=$3
```

```
echo "$hello1"  
echo "$hello2"  
echo "$hello3"
```

\$0 \$1 \$2 \$3



```
ygli@dc2:~/lab1$ ./lab2.sh 1 2 3  
1  
2  
3  
ygli@dc2:~/lab1$ ./lab2.sh 4 5 6  
4  
5  
6
```


Example

Running Experiments

```
ygli@xjtu-ics: ~/shell
#!/bin/bash
# 1-10 helloi.sh
mkdir hello
cd hello
for i in {1..10}; do
    touch hello$i.sh
    chmod +x hello$i.sh
    echo "echo hello$i" > hello$i.sh
done
4,8 All
```

```
ygli@xjtu-ics: ~/shell
#!/bin/bash
for i in {1..10}; do
    ./hello/hello$i.sh
done
~
~
~
~
```



**The best way to learn it,
is to use it.**



Happy Shell-ing!

Learn more

<https://www.bilibili.com/video/BV1y44y1v7c3/>

<https://www.bilibili.com/video/BV14E411J7n2/>