Terminal & Shell Tutorial

Before Shell, what is Terminal

终端（Terminal） 是一种用来让用户输入数据至计算机，并显示其计算结果的**机器**，所以它是一个**硬件设备**，是**计算机的组成固件之一**。

而我们使用的其实是**终端模拟器（Terminal Emulator）**，它是一个用来模拟传统终端的行为的软件。

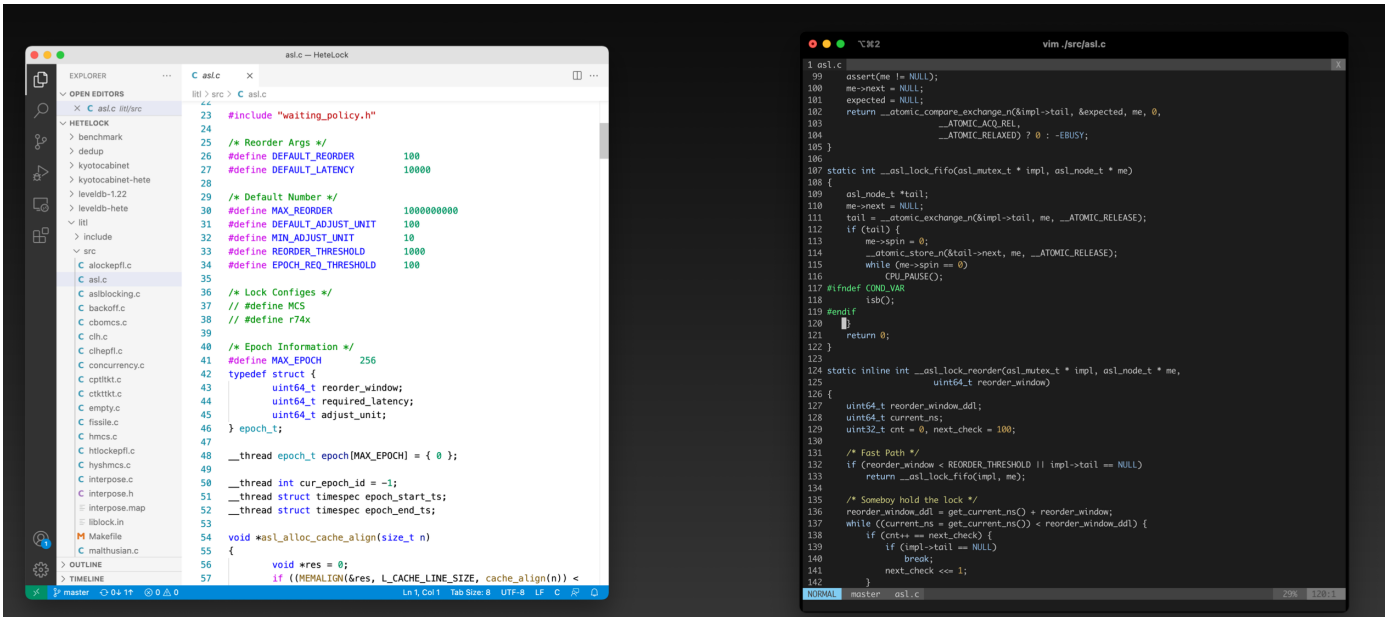
一个终端模拟器的标准工作流程（pipeline）

- 捕获你的键盘输入
- 将输入发送给命令行程序（Shell）
- 获得命令行程序的输出结果
- 调用图形接口（比如 X11），将输出结果渲染至显示器

常见的终端模拟器

- GNU/Linux：Gnome-terminal、Konsole
- Mac OS: Terminal.app、iTerm2
- Windows: Windows Terminal
- 跨平台：
  - Alacritty: https://alacritty.org/
  - Hyper: https://hyper.is/
  - Warp: https://www.warp.dev/
  - Kitty: https://sw.kovidgoyal.net/kitty/

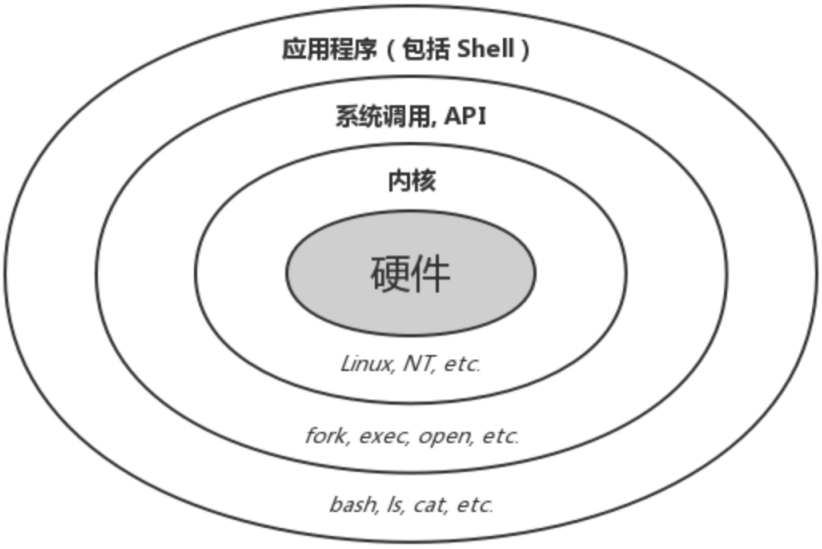
GUI vs CLI



What is Shell

Control your computer in a more efficient (also more geek) way.

> Shell是一个**命令行解释器**，是 Linux Kernel 的一个外壳，它是一个**软件**，负责外界与 Linux Kernel 的交互。Shell 接收用户或者其他应用程序的命令，然后将这些命令转化成 Kernel 能理解的语言并传给 Kernel，Kernel 执行命令完成后会将结果返回给用户或者应用程序。当你打开一个 Terminal 时，操作系统会将 Terminal 和 Shell 关联起来，当我们在 Terminal 中输入命令后，Shell 就负责解释命令，并将输出结果显示在 Terminal 中。



常见的 Shell

- **sh** (Bourne shell)，最经典的 Unix Shell
- **bash** (Bourne-Again shell)，目前绝大多数 Linux 发行版的默认 Shell
- **zsh** (Z shell)，一款可以高度自定义的 Shell

- **fish** (Friendly interactive shell)，专注于易用性与友好用户体验的 Shell
- **Cmd.exe / PowerShell**，Windows 下的图形化 Shell

Shell 与 Terminal

Terminal 干的活儿是从用户这里接收输入（键盘、鼠标等输入设备），传给 Shell，然后把 Shell 返回的结果传递给用户（比如通过显示器）。而 Shell 干的活儿是从 Terminal 那里拿到用户输入的命令，解析后交给操作 Kernel 去执行，并把执行结果返回给 Terminal。

一些例子

- Terminal 将用户的**键盘输入转换为控制序列**（除了字符以外的按键，比如 `左方向键` → `^[[D` ），Shell 则**解析并执行收到的控制序列**（比如 `^[[D` → `将光标向左移动` ）
- 终端在接收到 `Ctrl + C` 的组合键时，不会把这个按键转换为控制序列转发给当前的程序，而是会发送一个 `SIGINT` 信号给 Shell，这个信号会被 Shell 解析为 interrupt 指令从而终止当前运行的程序。

Shell Tutorial

命令提示符（Command Prompt）

当你首次打开 Terminal 时，会看到类似以下界面

```
Zicx:~$
```

这里的 `$` 是一个命令提示符，表明当前用户是一个普通用户，相对的，还有一个 root 用户，它对应的命令提示符为 `#` 。

这一行的意思是

> 当前的用户名是 `Zicx` 并且当前的工作目录（"current working directory"）或者说当前所在的位置是 `~` (表示 "home")。

你可以执行一些命令，这些命令本质上是**命令行程序**

```
Zicx:~$ date
Thu Nov 24 12:21:46 CST 2022
```

路径

Shell 中的路径是一组被分割的目录，在 Linux 和 macOS 上使用 `/` 分割，而在Windows上是 `\` 。路径 `/` 代表的是系统的根目录，所有的文件夹都包括在这个路径之下，在Windows上每个盘都有一个根目录（例如： `C:\` ）。

如果某个路径以 `/` 开头，那么它是一个 *绝对路径*，其他的都是 *相对路径* 。相对路径是指相对于当前工作目录的路径，当前工作目录可以使用 `pwd` 命令来获取。切换目录需要使用 `cd` 命令。在路径中，`.` 表示的是当前目录，而 `..` 表示上级目录

```
Zicx:~$ pwd
/home/Zicx
Zicx:~$ cd /home
Zicx:/home$ pwd
/home
Zicx:/home$ cd ..
Zicx:/$ pwd
/
Zicx:/$ cd ./home
Zicx:/home$ pwd
/home
Zicx:/home$ cd Zicx
Zicx:~$ pwd
/home/Zicx
Zicx:~$ ../../bin/echo hello
hello
```

有用的命令行工具

ls

```
Zicx:~$ ls
Zicx:~$ cd ..
Zicx:/home$ ls
Zicx
Zicx:/home$ cd ..
Zicx:/$ ls
bin
boot
dev
etc
home
...
```

optional parameters

- ls -a
- ls -l

```
Zicx:~$ ls -l /home
drwxr-xr-x 1 Zicx  users  4096 Nov 24  2022 Zicx
```

> -l 这个参数可以更加详细地列出目录下文件或文件夹的信息。首先，本行第一个字符 d 表示 Zicx 是一个目录。然后接下来的九个字符，每三个字符构成一组。（ rwx ）. 它们分别代表了文件所有者（ Zicx ），用户组（ users ） 以及其他所有人具有的权限。其中 - 表示该用户不具备相应的权限。从上面的信息来看，只有文件所有者可以修改（ w ）， Zicx 文件夹 （例如，添加或删除文件夹中的文件）。为了进入某个文件夹，用户需要具备该文件夹以及其父文件夹的“搜索”权限（以“可执行”： x ）权限表示。为了列出它的包含的内容，用户必须对该文件夹具备读权限（ r ）。对于文件来说，权限的意义也是类似的。注意， /bin 目录下的程序在最后一组，即表示所有人的用户组中，均包含 x 权限，也就是说任何人都可以执行这些程序。

mv

```
Zicx:~$ touch test.c
Zicx:~$ ls
test.c
Zicx:~$ mv test.c test1.c
Zicx:~$ ls
test1.c
```

optional parameters

- mv -i

cp

```
Zicx:~$ cp test.c test1.c
Zicx:~$ ls
test.c test1.c
```

optional parameters

- cp -r

mkdir

rmdir

rm

optional parameters

- rm -r
- rm -f

cat

touch

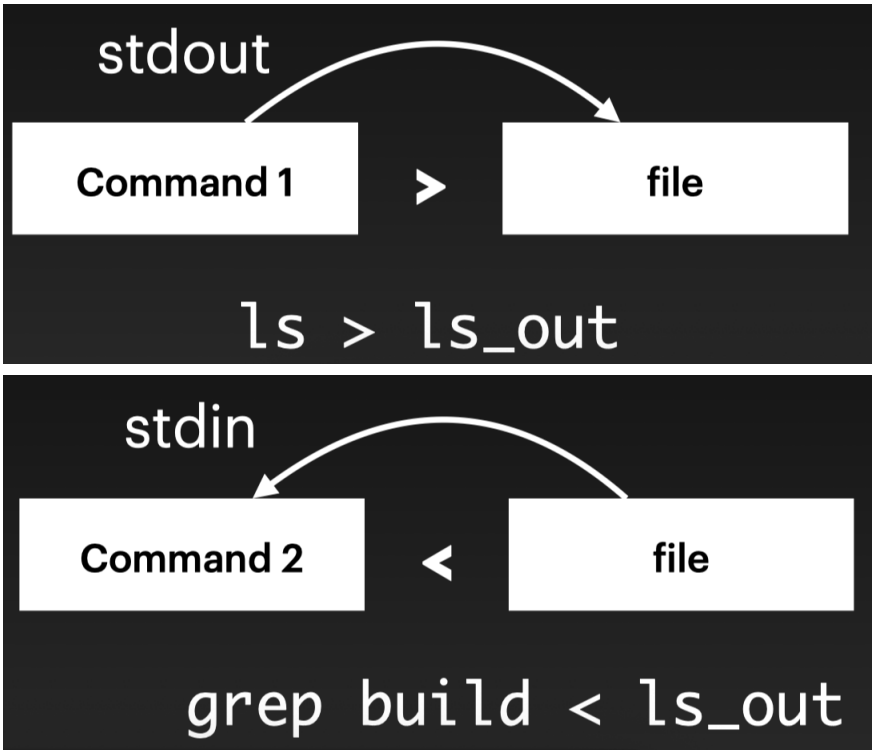
find

head & tail & less

man

重定向（Redirect）

在 Shell 中，程序有两个主要的“流”：**它们的输入流(stdin)和输出流(stdout)**。 当程序尝试读取信息时，它们会从输入流中进行读取，当程序打印信息时，它们会将信息输出到输出流中。 通常，一个程序的输入输出流都是您的终端。也就是，您的键盘作为输入，显示器作为输出。 但是，我们也可以重定向这些流。



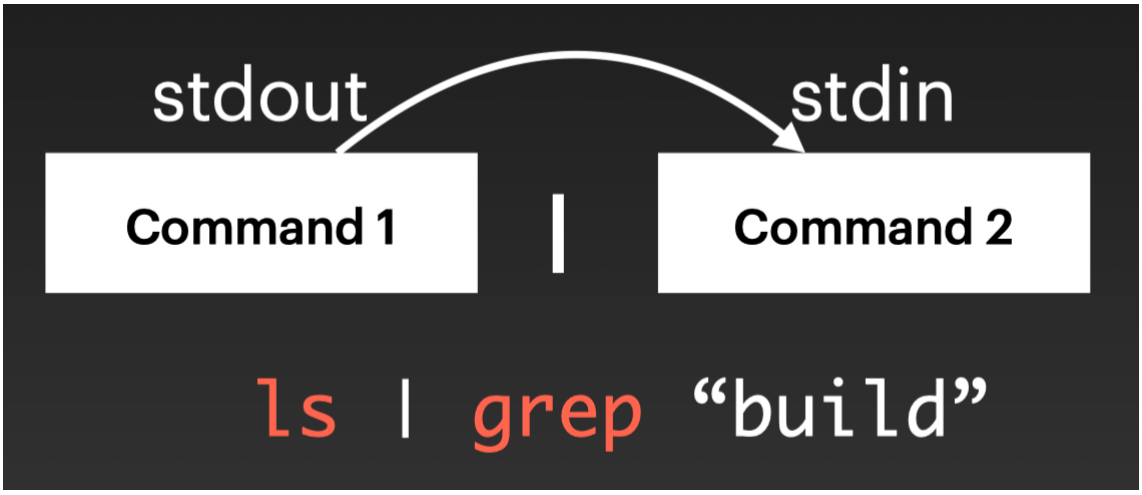
最简单的重定向是 `< file` 和 `> file` 。这两个命令可以将程序的输入输出流分别重定向到文件：

```
Zicx:~$ echo hello > hello.txt
Zicx:~$ cat hello.txt
hello
Zicx:~$ cat < hello.txt
hello
Zicx:~$ cat < hello.txt > hello2.txt
Zicx:~$ cat hello2.txt
hello
```

还可以使用 `>>` 来向一个文件追加内容而不覆盖

```
Zicx:~$ echo hello >> hello.txt
Zicx:~$ cat hello.txt
hellohello
```

管道（Pipe）



`|` 操作符允许我们将一个程序的输出和另外一个程序的输入连接起来：

```
Zicx:~$ ls -l / | tail -n1
drwxr-xr-x 1 root  root  4096 Nov 24  2022 var
Zicx:~$ curl --head --silent google.com | grep --ignore-case content-length | cut -
219
```

搭配 `grep` / `tail` / `less` 等一起使用

other tools

- ag: https://www.mankier.com/1/ag
- tldr: https://tldr.sh/
- curl: https://curl.se/docs/manpage.html
- exa: https://the.exa.website/
- ...

Shell Programming

到目前为止，我们已经学习来如何在 Shell 中执行命令，并使用管道将命令组合使用。但是，很多情况下我们需要执行一系列的操作并使用条件或循环这样的控制流，Shell 脚本是一种更加复杂度的工具，下面会简单介绍一下 Shell 脚本编程，以 Bash 为例。

Shell 脚本文件的后缀名为 `.sh`

注释

Bash 中用 `#` 来单行注释，`:' '` 来多行注释

```
# hello world

:'
echo "Hello World"
echo "Hello World"
echo "Hello World"
echo "Hello World"
echo "Hello World"
'
```

变量

Bash 中的字符串通过 ' 和 " 分隔符来定义，但是它们的含义并不相同。以 ' 定义的字符串为原义字符串，其中的变量不会被转义，而 " 定义的字符串会将变量值进行替换

```
foo="bar"
echo "$foo"
# 打印 bar
echo '$foo'
# 打印 $foo

res=`echo 'hello'` # res=hello
echo "$res"
# 打印 hello
```

函数

下面是一个函数 `mcd` 的例子

```
mcd () {
    mkdir -p "$1"
    cd "$1"
}
```

这里 `$1` 是脚本的第一个参数，其他还有很多参数

- `$0` - 脚本名
- `$@` - 所有参数
- `$#` - 所有参数的个数
- `$?` - 前一个命令的返回值

控制流

```
#!/bin/bash
```

```
echo "Starting program at $(date)" # date会被替换成日期和时间

echo "Running program $0 with $# arguments with pid $$"

for file in "$@"; do
    grep foobar "$file" > /dev/null 2> /dev/null
    # 如果模式没有找到，则grep退出状态为 1
    # 我们将标准输出流和标准错误流重定向到Null，因为我们并不关心这些信息
    if [[ $? -ne 0 ]]; then
        echo "File $file does not have any foobar, adding one"
        echo "# foobar" >> "$file"
    fi
done
```

注意第一行的 `#!` 叫做 shebang，用这个来指定解释器，例如，以指令 `#!/bin/sh` 开头的文件在执行时会实际调用 `/bin/sh` 程序。

if-else

```
a=10
b=20
if [ $a == $b ]
then
   echo "a 等于 b"
elif [ $a -gt $b ]
then
   echo "a 大于 b"
elif [ $a -lt $b ]
then
   echo "a 小于 b"
else
   echo "没有符合的条件"
fi
```

也使用 **((...))** 作为判断语句

for loop

```
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
```

while loop

```
int=1
while(( $int<=5 ))
do
```

```
    echo $int
    let "int++"
done
```

执行

写完一个 Shell 脚本之后，无法用对应的 Shell 解释器直接执行，需要添加 `x` 权限

`chmod +x test.sh` 可以添加可执行权限

```
Zicx:~$ ls -l
.rw-r--r--  120 vercent 24 Nov 21:56 test.sh
Zicx:~$ chmod +x test.sh
.rwxr-xr-x  120 vercent 24 Nov 21:57 test.sh
Zicx:~$ cat test.sh
foo="hello"
echo "$foo"
Zicx:~$ ./test.sh
hello
```

参考资料

- 菜鸟教程：https://www.runoob.com/linux/linux-shell.html
- mit 6.null：https://missing.csail.mit.edu/
- explain shell: https://explainshell.com/