

# Unix & Linux ☐



☐ ☐ Steve Parker☐ Unix & Linux Shell Scripting Tutorial☐ ☐☐☐☐ ☐☐ ☐☐☐☐ .

- [2. Philosophy](#)
- [3. A First Script](#)
- [4. Variables - Part I](#)
- [5. ☐☐☐☐ \(Wildcards\)](#)
- [6. ☐☐☐☐ ☐☐](#)
- [7. ☐☐](#)
- [8. Test](#)
- [9. Case](#)
- [10. Variables - Part II](#)
- [11. Variables - Part III](#)
- [12. External Programs](#)
- [13. Functions](#)
- [14. Hints and Tips](#)
- [15. Quick Reference](#)
- [16. Interactive Shell](#)



# 2. Philosophy

Unix 的哲学设计原则是：Unix 系统是由许多小的、简单的工具组成的。每个工具只做一件事，并且做得很好。这些工具通过管道（pipe）和重定向（redirect）来组合，以完成复杂的工作。

- 每个工具都应该只做一件事，并且做得很好。例如，C 语言编写的工具应该专注于处理数据，而 Perl 编写的工具应该专注于文本处理。
- 工具应该通过管道和重定向来组合，而不是通过复杂的脚本或宏。这样可以使程序更易于理解和维护。

Unix 的哲学设计原则使得 Unix 系统非常灵活和强大。通过组合简单的工具，用户可以完成几乎任何任务。例如，Perl 可以用于文本处理，C 可以用于系统编程，而 CGI 可以用于开发 Web 应用程序。这种设计哲学也使得 Unix 系统易于学习和使用。

- 工具应该通过管道和重定向来组合，而不是通过复杂的脚本或宏。这样可以使程序更易于理解和维护。
- 工具应该通过管道和重定向来组合，而不是通过复杂的脚本或宏。这样可以使程序更易于理解和维护。

Unix 的哲学设计原则使得 Unix 系统非常灵活和强大。通过组合简单的工具，用户可以完成几乎任何任务。例如，Perl 可以用于文本处理，C 可以用于系统编程，而 CGI 可以用于开发 Web 应用程序。这种设计哲学也使得 Unix 系统易于学习和使用。

1. 工具应该通过管道和重定向来组合，而不是通过复杂的脚本或宏。这样可以使程序更易于理解和维护。
2. 工具应该通过管道和重定向来组合，而不是通过复杂的脚本或宏。这样可以使程序更易于理解和维护。

Unix 的哲学设计原则使得 Unix 系统非常灵活和强大。通过组合简单的工具，用户可以完成几乎任何任务。例如，Perl 可以用于文本处理，C 可以用于系统编程，而 CGI 可以用于开发 Web 应用程序。这种设计哲学也使得 Unix 系统易于学习和使用。

Unix 的哲学设计原则使得 Unix 系统非常灵活和强大。通过组合简单的工具，用户可以完成几乎任何任务。例如，Perl 可以用于文本处理，C 可以用于系统编程，而 CGI 可以用于开发 Web 应用程序。这种设计哲学也使得 Unix 系统易于学习和使用。

```
cat /tmp/myfile | grep "mystring"
```

Unix 的哲学设计原则使得 Unix 系统非常灵活和强大。通过组合简单的工具，用户可以完成几乎任何任务。例如，Perl 可以用于文本处理，C 可以用于系统编程，而 CGI 可以用于开发 Web 应用程序。这种设计哲学也使得 Unix 系统易于学习和使用。

```
grep "mystring" /tmp/myfile
```

在 Linux 系统中，grep 命令用于在文件中搜索指定的字符串。例如，grep 命令可以在 /bin/ 目录下搜索包含 "mystring" 的文件。grep 命令的语法如下：



这个脚本使用 `OS` 变量来检测操作系统的类型。如果检测到 Linux，它会使用 `cat` 命令来读取文件。如果检测到 Windows，它会使用 `type` 命令来读取文件。

00000000, 00000000 '0000 0 00000000 cat0000 0000 00 000000 0000 0000  
 0000 00 (The Award For The Most Gratuitous Use Of The Word Cat In A Serious Shell Script)'0000  
 0000 Useless Use of Cat Award (UUoC)0000 0000 comp.unix.shell 0000 0000 0000 0000  
 0000 0 0 0000 . 0000 0000 0000 0000 0000 00 0000 00000000 0000 00  
 000000 .  
 000000

本文件係根據「[政府資訊公開法](#)」第 6 條第 1 項第 2 款規定，  
 本文件係根據「[政府資訊公開法](#)」第 6 條第 1 項第 2 款規定，  
 本文件係根據「[政府資訊公開法](#)」第 6 條第 1 項第 2 款規定，  
 本文件係根據「[政府資訊公開法](#)」第 6 條第 1 項第 2 款規定，



# 3. A First Script

我们使用 `chmod` 命令来给 `first.sh` 文件添加可执行权限，然后运行它。

```
$ chmod a+rx first.sh
```

```
$ ./first.sh
```

我们使用 `echo` 命令来输出 "Hello World"。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。

我们使用 `first.sh` 来指定脚本使用的解释器。

```
#!/bin/sh
# This is a comment!
echo Hello World # This is a comment, too!
```

我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。

我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。我们使用 `#!/bin/sh` 来指定脚本使用的解释器。

我们使用 `echo` 命令来输出 "Hello"。我们使用 `echo` 命令来输出 "Hello"。我们使用 `echo` 命令来输出 "Hello"。

我们使用 `#!/bin/sh` 来指定脚本使用的解释器。

我们使用 `chmod 755 first.sh` 来指定脚本使用的解释器。我们使用 `chmod 755 first.sh` 来指定脚本使用的解释器。我们使用 `chmod 755 first.sh` 来指定脚本使用的解释器。

```
$ chmod 755 first.sh $ ./first.sh
Hello World
$
```



我们 来写一个 简单的 程序 :

```
$ echo Hello World
Hello World
$
```

我们 在 终端 输入 命令 .  
我们 , echo 命令 在 终端 输入 命令 . "Hello" "World" 我们 看到 输出 结果 .  
我们 看到 输出 结果 ? 我们 在 终端 输入 命令 输出 结果 ?  
我们 看到 输出 结果 我们 在 终端 输入 命令 .  
我们 看到 输出 结果 ! 我们 在 终端 输入 命令 echo 输出 结果 我们 , 我们 看到 输出 结果  
我们 看到 输出 结果 cp 我们 看到 输出 结果 . 我们 看到 输出 结果 :

```
#!/bin/sh
# This is a comment!
echo "Hello World" # This is a comment, too!
```

我们 看到 输出 结果 . 我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 . 我们 看到 输出 结果  
我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 我们 看到 输出 结果 我们 看到 输出 结果  
我们 看到 输出 结果 : 我们 ?  
我们 **echo** 命令 在 终端 输入 命令 , 我们 "Hello World" 我们 看到 输出 结果 . 我们 看到 输出 结果  
我们 看到 输出 结果 .  
我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 我们 看到 输出 结果 我们 看到 输出 结果 我们 看到 输出 结果 .  
我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 .  
我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 . 我们 看到 输出 结果 :

```
#!/bin/sh
# This is a comment!
echo "Hello World" # This is a comment, too!
echo "Hello World"
echo "Hello * World"
echo Hello * World
echo Hello World
echo "Hello" World
echo Hello " " World
echo "Hello \"*" World"
echo `hello` world
echo 'hello' world
```

我们 看到 输出 结果 我们 在 终端 输入 命令 输出 结果 ! 我们 看到 输出 结果 我们 看到 输出 结果  
我们 看到 输出 结果 .... 我们 , echo 命令 在 终端 输入 命令 输出 结果 !







# 4. Variables - Part I

이제 우리는 변수(variable)에 대해 알아보겠습니다. 변수는 프로그램에서 데이터를 저장하고 참조하기 위한 공간입니다. Bourne shell에서 변수를 선언하고 사용하는 방법을 알아보겠습니다.

변수를 선언하는 방법은 다음과 같습니다. `VAR=값` 또는 `VAR = 값`의 형태로 작성합니다. 여기서 `VAR`은 변수의 이름, `값`은 저장할 데이터입니다. 변수를 선언한 후, `echo $VAR`과 같이 변수의 값을 출력할 수 있습니다.

예를 들어, `var1.sh`라는 파일을 만들고 다음 내용을 작성합니다:

```
#!/bin/sh
MY_MESSAGE="Hello World"
echo $MY_MESSAGE
```

이제 이 파일을 실행해 보겠습니다. `MY_MESSAGE` 변수에 "Hello World"라는 값을 저장하고, `echo` 명령을 사용하여 화면에 출력합니다.

Hello World라는 값을 `MY_MESSAGE` 변수에 저장하고, `echo` 명령을 사용하여 화면에 출력합니다. 이 때, `MY_MESSAGE=Hello World`와 같이 변수를 선언하고, `echo $MY_MESSAGE`와 같이 변수의 값을 출력합니다.

이제 우리는 변수의 다른 사용법에 대해 알아보겠습니다. 변수를 선언하고, 값을 출력하는 것 외에도, 변수를 다른 변수에 대입하거나, 변수를 산술 연산에 사용할 수 있습니다.

예를 들어, `Perl`과 `C` 언어를 비교해 보겠습니다. `Perl`은 변수를 선언하고, 값을 출력하는 데 매우 간단합니다. 반면, `C` 언어는 변수를 선언하고, 값을 출력하는 데 매우 복잡합니다.

이제 우리는 변수의 다른 사용법에 대해 알아보겠습니다. 변수를 선언하고, 값을 출력하는 것 외에도, 변수를 산술 연산에 사용할 수 있습니다.

```
$ x="hello"
$ expr $x + 1
expr: non-numeric argument
$
```

이제 우리는 변수의 다른 사용법에 대해 알아보겠습니다. 변수를 선언하고, 값을 출력하는 것 외에도, 변수를 산술 연산에 사용할 수 있습니다.

```
MY_MESSAGE="Hello World"
MY_SHORT_MESSAGE=hi
MY_NUMBER=1
MY_PI=3.142
```



```
MY_OTHER_PI="3.142"
```

```
MY_MIXED=123abc
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
#!/bin/sh
```

```
echo What is your name?
```

```
read MY_NAME
```

```
echo "Hello $MY_NAME - hope you're well."
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
if [ -d /dev/shm ]; then
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
MY_OBFUSCATED_VARIABLE=Hello
```

```
if [ -d /dev/shm ]; then
```

```
echo $MY_OSFUCATED_VARIABLE
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```

```
if [ -d /dev/shm ]; then
  if [ -d /dev/shm/6 ]; then
    echo "6 is a directory"
  else
    echo "6 is not a directory"
  fi
fi
```



```
#!/bin/sh
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo "MYVAR is: $MYVAR"
```

❏   ❏❏❏❏   ❏❏❏❏   :

```
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

MYVAR❏ ❏❏   ❏❏   ❏❏❏❏   ❏❏❏❏   ❏❏   ❏❏❏❏   . ❏❏   ❏❏   ❏❏   ❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏   .  
❏❏   ❏❏❏❏   :

```
$ MYVAR=hello
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

❏❏   ❏❏❏❏   ❏❏❏❏   !❏❏   ❏❏❏   ?  
❏❏❏   ❏❏❏   myvar2.sh❏ ❏❏❏❏   ❏❏❏❏   ❏❏❏❏   ❏❏   ❏   ❏❏❏❏   . ❏❏   ❏❏   ❏❏❏❏  
❏❏❏❏   ❏❏   ❏❏❏   #!/bin/sh❏ ❏❏❏❏❏   .  
❏❏   ❏❏❏❏   ❏❏❏   ❏❏   ❏❏❏❏❏❏   ❏❏❏   ❏❏❏❏❏   ❏❏❏   ❏❏❏❏   ❏❏❏   .❏❏❏   ❏❏❏❏❏   :

```
$ export MYVAR
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
```

❏❏   ❏❏❏❏   3❏❏   ❏❏❏   . MYVAR❏❏❏❏   ❏❏❏❏   ❏❏❏❏   . ❏❏❏   ❏❏❏   ❏❏❏   ❏❏   ❏❏   ❏❏❏  
❏❏❏   ❏❏❏❏   . MYVAR❏❏❏❏   ❏❏❏❏❏❏   :

```
$ echo $MYVAR
hello
$
```

❏❏❏❏❏❏   ❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏❏❏   . ❏❏❏   MYVAR❏❏❏❏   ❏❏❏❏   ❏❏❏❏   hello❏❏❏❏   ❏❏❏❏❏❏  
❏❏❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏❏❏   ❏❏❏❏❏❏   ❏❏❏❏   ❏❏❏❏   ,❏❏❏   ❏❏❏   ❏❏❏❏❏❏  
❏❏❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏   ❏❏❏❏❏❏   ❏❏❏❏❏❏   ❏❏❏❏   ❏❏❏❏❏❏  
"."(❏❏❏❏❏❏   ❏❏❏❏❏❏❏   ❏❏❏❏❏❏   ❏❏❏❏❏❏❏   :



```
$ MYVAR=hello
$ echo $MYVAR
hello
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
$ echo $MYVAR
hi there
```

[illegible]

```
#!/bin/sh

echo "What is your name?"

read USER_NAME

echo "Hello $USER_NAME"

echo "I will create you a file called $USER_NAME_file"

touch $USER_NAME_file
```

```

00000000  00 00 00 00 00 00 . 00 00 USER_NAME 00 "steve" 00 0000 00000000
00000004  steve_file 0000 0000 ?
00000008  00 , 0000 . USER_NAME_file 0000 0000 0000 000000 . 00 0000 0000 0000
0000000c  0000 000000 00 0000 . 00 0000 0000 00 0000 ? 00 00 0000 0000 ({}) 00 00
00000010  0000 :

```

```
#!/bin/sh
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I will create you a file called ${USER_NAME}_file"
touch "${USER_NAME}_file"
```

```

00000000  00 00 0000  USER_NAME 0000 0000 0000 0 0000  "_file"0000 0000 0000
00000008  0000 00 00 0000  . 0000 0000 0000 0000 0 00 0000 00 00 0 0000
00000010  000000 00 0000 00 0 0000  .

```

```
❯ echo "${USER_NAME}_file" | xargs touch -t 1970-01-01 00:00:00 "Steve Parker"
❯ ls -l $(xargs touch -t 1970-01-01 00:00:00 Steve Parker_file)
-rw-rw-r-- 1 root root 0 1970-01-01 00:00 Steve Parker_file
```





Chris



.



## 5. 通配符 (Wildcards)

通常、ファイル名やディレクトリ名に「\*」や「?」などの通配符を使用します。  
「\*」は任意の文字列を指定し、「?」は任意の1文字を指定します。  
例: `cp /tmp/a/* /tmp/b/` は、`/tmp/a/` ディレクトリ内のすべてのファイルを `/tmp/b/` にコピーします。  
例: `cp /tmp/a/*.txt /tmp/b/` は、`/tmp/a/` ディレクトリ内のすべての `.txt` 形式のファイルを `/tmp/b/` にコピーします。  
例: `cp /tmp/a/*.html /tmp/b/` は、`/tmp/a/` ディレクトリ内のすべての `.html` 形式のファイルを `/tmp/b/` にコピーします。

```
$ cp /tmp/a/* /tmp/b/
$ cp /tmp/a/*.txt /tmp/b/
$ cp /tmp/a/*.html /tmp/b/
```

例: `ls /tmp/a/` は、`/tmp/a/` ディレクトリ内のすべてのファイルとディレクトリを列挙します。  
例: `echo /tmp/a/*` は、`/tmp/a/` ディレクトリ内のすべてのファイルとディレクトリの名前を出力します。  
例: `ls *.txt` は、現在のディレクトリ内のすべての `.txt` 形式のファイルを列挙します。

```
$ mv *.txt *.bak
```

例: `mv *.txt *.bak` は、現在のディレクトリ内のすべての `.txt` 形式のファイルを `.bak` 形式に変換します。  
例: `mv $(echo *.txt) $(echo *.bak)` は、現在のディレクトリ内のすべての `.txt` 形式のファイルを `.bak` 形式に変換します。



## 6. 字符串和正则表达式

在 Linux 中，字符串（`"`）和正则表达式（`/`）是两种不同的数据类型。在 Linux 中，字符串和正则表达式的使用方式如下：

```
$ echo Hello World
Hello World
$ echo "Hello World"
Hello World
```

在 Linux 中，字符串和正则表达式的使用方式如下：

```
$ echo "Hello \World\"
```

在 Linux 中，字符串和正则表达式的使用方式如下：

```
$ echo "Hello " World ""
```

在 Linux 中，字符串和正则表达式的使用方式如下：

- "Hello "
- World
- ""

在 Linux 中，字符串和正则表达式的使用方式如下：

Hello World

在 Linux 中，字符串和正则表达式的使用方式如下：

在 Linux 中，字符串和正则表达式的使用方式如下：

```
$ echo "Hello "World""
```

在 Linux 中，字符串和正则表达式的使用方式如下：

在 Linux 中，字符串和正则表达式的使用方式如下：



```
$ echo *
case.shtml escape.shtml first.shtml
functions.shtml hints.shtml index.shtml
ip-primer.txt raid1+0.txt

$ echo *txt
ip-primer.txt raid1+0.txt

$ echo "*"
*

$ echo "*txt"
*txt
```

\* 是通配符，在 shell 中，\* 代表任意文件。
   
 在上面的例子中，\* 匹配了所有以 . 开头的文件，以及所有以 txt 结尾的文件。
   
 在上面的例子中，\*txt 匹配了所有以 txt 结尾的文件。

在 shell 中，双引号 " 和单引号 ' 用于引用字符串。
   
 在上面的例子中，\$ 代表美元符号，\ 代表反斜杠，` 代表反引号。
   
 在上面的例子中，( 和 ) 用于分组，: 用于分隔符。

A quote is ", backslash is \, backtick is `.  
 A few spaces are and dollar is \$. \$X is 5.

在 shell 中，\$ 代表美元符号，\ 代表反斜杠，` 代表反引号。

```
$ echo "A quote is \", backslash is \\, backtick is \`."
A quote is ", backslash is \, backtick is `.

$ echo "A few spaces are   ; dollar is \$. \X is ${X}."
A few spaces are   ; dollar is $. $X is 5.
```

在 shell 中，\$ 代表美元符号，\ 代表反斜杠，` 代表反引号。
   
 在上面的例子中，( 和 ) 用于分组，: 用于分隔符。
   
 在上面的例子中，\$X 代表美元符号和 X，\X 代表反斜杠和 X。

```
$ echo "This is \ a backslash"
This is \ a backslash

$ echo "This is \" a quote and this is \\ a backslash"
This is " a quote and this is \ a backslash
```

在 shell 中，\$ 代表美元符号，\ 代表反斜杠，` 代表反引号。
   
 在上面的例子中，( 和 ) 用于分组，: 用于分隔符。



# 7. 循环

循环是编程中非常基础且重要的概念。在 C 语言中，主要有两种循环结构：for 循环和 while 循环。for 循环适用于已知循环次数的情况，而 while 循环适用于未知循环次数的情况。

## For 循环

"for" 循环的语法如下：

```
#!/bin/sh
for i in 1 2 3 4 5
do
    echo "Looping ... number $i"
done
```

在上面的例子中，for 循环遍历了 1 到 5 的数字，并打印了每个数字。其中，in 后面的列表可以是数字、字符串或命令的输出。

```
#!/bin/sh
for i in hello 1 * 2 goodbye
do
    echo "Looping ... i is set to $i"
done
```

在上面的例子中，for 循环遍历了字符串 "hello 1 \* 2 goodbye"，并打印了每个单词。其中，\* 表示通配符，可以匹配任意数量的任意字符。

在上面的例子中，for 循环遍历了命令 "ls" 的输出，并打印了每个文件。其中，(ls) 表示在循环体内执行 ls 命令。

```
Looping .... number 1
Looping .... number 2
Looping .... number 3
Looping .... number 4
Looping .... number 5
```

以上就是 for 循环的基本用法。



```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (name of first file in current directory)
... etc ...
Looping ... i is set to (name of last file in current directory) Looping ... i is set to 2
Looping ... i is set to goodbye
```

, .

While ☐ ☐

```
"while" (
  ...
  ... )
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

INPUT_STRING=hello

while [ "$INPUT_STRING" != "bye" ]
do
    echo "Please type something in (bye to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```

0000  0000  00  0000  000  0 "bye" 000  000  00  0 00 00 000 0000
0000  0000  . 0000  00  '00  - 10 '(40 ) 0000  INPUT_STRING=hello 000  000
0000  0000  . 000  00  000  00  000  00  00  000  000  .

```

0000 (:) 0000 000000 . 0000 0000 00 0000 00 0000 , 00 00 0000 0000  
 00 0 0000 00 0000 . 00 0000 0000 00 0000 0000 0000 00 0000 00 00  
 0 0000 000000 . 00 0000 00 0000 0 0000 0 00 0 00 0000 0000 0000 :

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```
#!/bin/sh

while :
do
    echo "Please type something in (^C to quit)"
    read INPUT_STRING
    echo "You typed: $INPUT_STRING"
done
```

```

while read line; do
    # Process each line
done < file.txt

```



```
( : LF( ) )
. cat myfile.txt
. ).
"myfile.txt"
$input_text
. case
$input_text
"hello" "English"
"gd" "gday"
"Australian" echo
myfile.txt
"Unknown Language: $input_text"
$input_text
myfile.txt
```

```
#!/bin/sh

while read input_text
do
    case $input_text in
        hello)          echo English    ;;
        howdy)          echo American   ;;
        gday)           echo Australian ;;
        bonjour)        echo French     ;;
        "guten tag")    echo German
        *)              echo Unknown Language: $input_text ;;
    esac
done < myfile.txt
```

"myfile.txt"         :

```
this file is called myfile.txt. It is an example text file.  
hello  
gday  
bonjour  
hola
```













Unknown Language: this file is called myfile.txt. It is an example text file.

English

Australian

French

Unknown Language: hola

```

███  ██  ████  13 ██████  ██  ███  Bash(██  ██  ██ )██  ███  ████  :

```

```
mkdir rc{0,1,2,3,4,5,6,S}.d
```



```
for runlevel in 0 1 2 3 4 5 6 S
do
    mkdir rc${runlevel}.d
done
```

□ □□□ □□□□ □□ □ □□□ :

```
$ cd /
$ ls -ld {,usr,usr/local}/{bin,sbin,lib}
drwxr-xr-x  2 root  root  4096 Oct 26 01:00 /bin
drwxr-xr-x  6 root  root  4096 Jan 16 17:09 /lib
drwxr-xr-x  2 root  root  4096 Oct 27 00:02 /sbin
drwxr-xr-x  2 root  root 40960 Jan 16 19:35 usr/bin
drwxr-xr-x 83 root  root 49152 Jan 16 17:23 usr/lib
drwxr-xr-x  2 root  root  4096 Jan 16 22:22 usr/local/bin
drwxr-xr-x  3 root  root  4096 Jan 16 19:17 usr/local/lib
drwxr-xr-x  2 root  root  4096 Dec 28 00:44 usr/local/sbin
drwxr-xr-x  2 root  root  8192 Dec 27 02:10 usr/sbin
```

Test □ Case □□□ while □□□ □□ □□□ □□□□□□ .



# 8. Test

`test` 是一个测试命令，用于测试各种条件。它通常用于 `if` 语句中。在 Unix 系统中，`test` 命令通常位于 `/usr/bin/` 目录下。它的语法如下：

```
$ type [  
[ is a shell builtin  
$ which [  
/usr/bin/[  
$ ls -l /usr/bin/[  
lrwxrwxrwx 1 root root 4 Mar 27 2000 /usr/bin/[ -> test
```

例如，我们可以使用 `test` 来检查一个变量是否等于某个值：

```
if [ $foo = "bar" ]
```

在这个例子中，`test` 命令被用来检查变量 `$foo` 是否等于字符串 `"bar"`。如果条件为真，那么 `if` 语句后面的命令将会被执行。需要注意的是，在 `test` 命令中，字符串需要用单引号或双引号括起来，以避免空格和其他特殊字符引起的问题。

```
if SPACE [ SPACE "$foo" SPACE = SPACE "bar" SPACE ]
```

在这个例子中，我们使用 `SPACE` 来代表空格，以避免在 `test` 命令中产生歧义。我们可以看到，`test` 命令可以处理包含空格的字符串。

`test` 命令还可以用于测试文件的存在性。例如，我们可以使用 `test` 来检查一个文件是否存在：

```
test if while 等命令都可以使用 test 来测试条件。例如，我们可以使用 test 来检查一个文件是否存在，如果存在，则执行某些操作；否则，执行另一些操作。
```

```
if [ ... ] then  
  # if-code  
else  
  # else-code  
fi
```



```
fi
# do something
esac
.
if [ ... ]
then
# do something
fi
:
```

```
if [ ... ]; then
# do something
fi
```

```
elif
:
```

```
if [ something ]; then
echo "Something"
elif [ something_else ]; then
echo "Something else"
else
echo "None of the above"
fi
```

```
[something ]
echo "Something"
, [ something_else ]
.
[something_else ]
echo "Something else"
.
"None of the above"
```

```
X
(-1, 0, 1, hello, bye
).
( - 1
Dave
):
```

```
$ X=5
$ export X
$ ./test.sh
... output of test.sh ...
$ X=hello
$ ./test.sh
... output of test.sh ...
$ X=test.sh
$ ./test.sh
... output of test.sh ...
```

```
$X
( : /etc/hosts)
```

```
#!/bin/sh
if [ "$X" -lt "0" ]
```







```
[ -f $X ] && echo "X is a file" || echo "X is not a file"

[ -n $X ] && echo "X is of non-zero length" || \
    echo "X is of zero length"
```

0. 测试用例 [ 测试用例 (测试用例) ] 测试用例 , 测试用例 test 测试用例 . 测试用例  
 1. 测试用例 测试用例 测试用例 测试用例 测试用例 测试用例 . if...then...else... 测试用例  
 2. 测试用例 测试用例 . 测试用例 测试用例 测试用例 测试用例 测试用例 测试用例 [ ... ]  
 3. 测试用例 测试用例 .

X□ □□ □□ □□ □□ □□ □ □ □□ □□ □□□□ :

```
test.sh: [: integer expression expected before -lt
test.sh: [: integer expression expected before -gt
test.sh: [: integer expression expected before -le
test.sh: [: integer expression expected before -ge
```

00 -lt, -gt, -le 00 -ge 00 00 0000 0000 000000 0000 00 00000 . !=0 00  
 00 00 "5" 00 0000 0000 "Hello" 00 0000 0000 00 0000 00  
 00 00 0000 . 0 0000 0 0000 0000 00 0000 00 00 00 00  
 00 00 0000 00 :

```
echo -en "Please guess the magic number: "
read X
echo $X | grep "[^0-9]" > /dev/null 2>&1
if [ "$?" -eq "0" ]; then
    # If the grep found something other than 0-9
    # then it's not an integer.
    echo "Sorry, wanted a number"
else
    # The grep found only 0-9, so it's an integer.
    # We can safely do a test on it.
    if [ "$X" = "7" ]; then
        echo "You entered the magic number!"
    fi
fi
```

[illegible]



test2.sh while test2.sh :

```
#!/bin/sh
X=0
while [ -n "$X" ]
do
    echo "Enter some text (RETURN to quit)"
    read X
    echo "You said: $X"
done
```

RETURN 0 (X 0 ). Justin Heath . [ -n "\$X" ] \$X . \$X . :

```
$ ./test2.sh
Enter some text (RETURN to quit)
fred
You said: fred
Enter some text (RETURN to quit)
wilma
You said: wilma
Enter some text (RETURN to quit)
```

test2.sh :

\$

test2.sh :

```
#!/bin/sh
X=0
while [ -n "$X" ]
do
    echo "Enter some text (RETURN to quit)"
    read X
    if [ -n "$X" ]; then
        echo "You said: $X"
    fi
done
```



if [ "\$X" -lt "0" ]  
then  
echo "X is less than zero"  
fi

.....

```
if [ ! -n "$X" ]; then  
    echo "You said: $X"  
fi
```

if [ ! -n "\$X" ] then  
echo "You said: \$X"  
fi

```
if [ ! -n "$X" ]  
then  
    echo "You said: $X"  
fi
```

if [ ! -n "\$X" ] then  
echo "You said: \$X"  
fi



# 9. Case

case 语句 类似于 if .. then .. else 语句，但 case 语句 适用于 多个 分支 的情况。 :

```
#!/bin/sh
echo "Please talk to me ..."
while :
do
  read INPUT_STRING
  case $INPUT_STRING in
    hello)
      echo "Hello yourself!"
      ;;
    bye)
      echo "See you again!"
      break
      ;;
    *)
      echo "Sorry, I don't understand"
      ;;

  esac
done
echo
echo "That's all folks!"
```

运行该脚本，输入 hello，将看到 Hello yourself!，输入 bye，将看到 See you again!，输入其他内容，将看到 Sorry, I don't understand!

运行该脚本，输入 hello，将看到 Hello yourself!，输入 bye，将看到 See you again!，输入其他内容，将看到 Sorry, I don't understand!

```
$ ./talk.sh
Please talk to me ...
hello
Hello yourself!
What do you think of politics?
Sorry, I don't understand
bye
See you again!
```



That's all folks!

\$

```
case $1 in
    hello) echo "hello" ;;
    *) echo "unknown" ;;
esac
```

```
case $1 in
    hello) echo "hello" ;;
    goodbye) echo "goodbye" ;;
    *) echo "unknown" ;;
esac
```

```
case $1 in
    hello) echo "hello" ;;
    *) echo "unknown" ;;
esac
```

```
case $1 in
    hello) echo "hello" ;;
    *) echo "unknown" ;;
esac
```



# 10. Variables - Part II

Each time you run a script, the values of the variables are reset. This means that you can't rely on a variable's value from a previous run. You can, however, use the `set` command to display the current values of all variables.

For example, if you have a script named `var3.sh` with the following content:

```
#!/bin/sh
echo "I was called with $# parameters"
echo "My name is $0"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are $@"
```

and you run it with the following command:

```
$ ./var3.sh hello world earth
```

the output will be:

```
#!/bin/sh
echo "I was called with $# parameters"
echo "My name is $0"
echo "My first parameter is $1"
echo "My second parameter is $2"
echo "All parameters are $@"
```

Each time you run a script, the values of the variables are reset:

```
$ ./var3.sh
I was called with 0 parameters
My name is ./var3.sh
My first parameter is
My second parameter is
All parameters are
$
$ ./var3.sh hello world earth
I was called with 3 parameters
My name is ./var3.sh
My first parameter is hello
My second parameter is world
All parameters are hello world earth
```

For example, if you have a script named `var3.sh` with the following content:

```
#!/bin/sh
echo "My name is `basename $0`"
```

and you run it with the following command:

```
echo "My name is `basename $0`"
```



\$# \$1 ... \$2 个 个 个 个 . 个 个 个 个 shift 个 个 9 个 个 个 个 个 个 :

```
#!/bin/sh
while [ "$#" -gt "0" ]
do
    echo "\$1 is $1"
    shift
done
```

个 个 个 个 \$# 0 个 个 个 , 个 个 个 个 个 个 shift 个 个 .

个 个 个 个 \$? 个 个 个 个 个 个 个 个 . 个 个 :

```
#!/bin/sh
/usr/local/bin/my-command
if [ "$?" -ne "0" ]; then
    echo "Sorry, we had a problem there!"
fi
```

个 个 个 个 个 个 0 个 个 个 , 个 个 0 个 个 个 /usr/local/bin/my-command 个 个 个 . 个 个 个 个 \$? 个 个 个 个 个 个 个 个 个 个 个 个 . 个 个 个 个 个 个 个 个 个 个 个 个 . 个 个 个 个 个 个 个 个 0 个 个 个 . 个 个 个 个 个 个 :

“ "个 个 个 个 个 个 个 0 个 个 个 C 个 个 个 个 个 个 个 个 个 个 个 个 ."  
(Robert Firth)

个 个 个 个 个 个 个 个 \$ \$ \$! 个 个 个 个 . 个 \$ \$ 个 个 个 个 PID(个 个 个 )个 个 . 个 个 个 个 个 个 个 个 个 个 个 个 个 个 个 个 /tmp/my-script.\$ \$ 个 个 个 个 个 个 个 个 个 个 . ! 个 个 个 个 个 个 个 个 PID 个 个 . 个 个 个 个 个 个 个 个 个 个 .

个 个 个 个 个 个 IFS 个 个 . 个 个 个 个 个 个 . 个 个 SPACE TAB NEWLINE 个 个 个 个 个 个 个 个 个 个 :

```
#!/bin/sh
old_IFS="$IFS"
IFS=:
echo "Please input some data separated by colons ..."
```



```
read x y z
IFS=$old_IFS
echo "x is $x y is $y z is $z"
```

❏   ❏❏❏❏   ❏❏   ❏❏   ❏❏❏❏   :

```
$ ./ifs.sh
Please input some data separated by colons ...
hello:how are you:today
x is hello y is how are you z is today
```

❏❏   ❏❏   "[hello:how are you:today:my:friend]"❏❏   ❏❏❏❏   ❏❏❏   ❏❏   ❏❏❏❏   :

```
$ ./ifs.sh
Please input some data separated by colons ...
hello:how are you:today:my:friend
x is hello y is how are you z is today:my:friend
```

❏❏   IFS❏❏   ❏❏   ❏❏   ❏❏   , ❏❏   ❏❏   ❏❏   "❏❏❏   ❏❏   ❏❏   "❏❏❏   ❏❏❏   ❏❏   ❏❏❏   ❏❏   ❏❏❏❏   ❏❏  
❏❏❏❏   . ❏❏❏   ❏❏❏❏   ❏❏   ❏❏   ❏❏❏❏   (❏❏ : old\_IFS=\$IFS ❏❏   old\_IFS="\$IFS").



# 11. Variables - Part III

[illegible]

```
foo=sun
echo $fooshine # $fooshine is undefined
echo ${foo}shine # displays the word "sunshine"
```

`00`    `0` `00`    `0000`    `0` `000`    `00` `0` `000`    `000`    `0000`    . `000`    `0000`    `0000`    (undefined),  
**null** `0000`    `0000`    `0` `0000`    (`0000`    `0000`    `00` `00`    **null** `00` `0` `0000`    `0000` ).












 (snippet)
 
 :

```
#!/bin/sh
echo -en "What is your name [ `whoami` ] "
read myname
if [ -z "$myname" ]; then
    myname=`whoami`
fi
echo "Your name is : $myname"
```

echo "-en"                               (bash    csh       ). Dash, Bourne  
                  ,          "\c"      . Ksh                      "RETURN"  
                                       :

```
steve$ ./name.sh
What is your name [ steve ] RETURN
Your name is : steve
```

$$\dots \quad \begin{array}{|c|c|} \hline & \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \quad \vdots$$

```
steve$ ./name.sh
What is your name [ steve ] foo
Your name is : foo
```



```
echo -en "What is your name [ `whoami` ] "  
read myname  
echo "Your name is : ${myname:-`whoami`}"
```

이 코드는 echo 명령을 사용하여 "What is your name [ `whoami` ] "라는 메시지를 출력하고, read 명령을 사용하여 myname 변수에 값을 입력받습니다. echo 명령을 사용하여 "Your name is : \${myname:-`whoami`}"라는 메시지를 출력합니다.

```
echo "Your name is : ${myname:-John Doe}"
```

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:-John Doe}"라는 메시지를 출력합니다. 여기서 John Doe는 myname 변수가 비어 있을 때 echo 명령에 사용되는 기본값입니다.

## Using and Setting Default Values

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:=John Doe}"라는 메시지를 출력합니다. 여기서 John Doe는 myname 변수가 비어 있을 때 echo 명령에 사용되는 기본값입니다.

```
echo "Your name is : ${myname:=John Doe}"
```

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:=John Doe}"라는 메시지를 출력합니다. 여기서 John Doe는 myname 변수가 비어 있을 때 echo 명령에 사용되는 기본값입니다.



# 12. External Programs

在编写脚本时，我们经常会用到一些外部程序，如 `echo`、`which`、`test`、`tr`、`grep`、`expr`、`cut` 等。这些程序通常位于 `/usr/bin` 目录下。

例如，我们可以使用 `grep` 来查找 `/etc/passwd` 文件中包含特定字符串的行。以下是一个示例：

```
$ grep "^${USER}:" /etc/passwd | cut -d: -f5
Steve Parker
```

这里，`grep` 用于查找以 `^${USER}:` 开头的行，`cut` 用于提取第五列（即用户名）。

```
$ MYNAME=`grep "^${USER}:" /etc/passwd | cut -d: -f5`
$ echo $MYNAME
Steve Parker
```

在上面的示例中，我们使用了反引号（```）来执行命令并将结果赋值给变量 `MYNAME`。然后，我们使用 `echo` 来输出该变量的值。

```
#!/bin/sh
find / -name "*.html" -print | grep "/index.html$"
find / -name "*.html" -print | grep "/contents.html$"
```

这里，我们使用 `find` 来查找所有 `.html` 文件，并使用 `grep` 来筛选出包含特定字符串的文件。

```
#!/bin/sh
HTML_FILES=`find / -name "*.html" -print`
echo "$HTML_FILES" | grep "/index.html$"
echo "$HTML_FILES" | grep "/contents.html$"
```

在上面的示例中，我们使用 `find` 来查找所有 `.html` 文件，并将结果存储在变量 `HTML_FILES` 中。然后，我们使用 `echo` 来输出该变量的值，并使用 `grep` 来筛选出包含特定字符串的文件。

这里，我们使用 `grep` 来筛选出包含特定字符串的文件。注意，我们使用了 `grep` 而不是 `grep -l`，因为 `grep` 会输出所有匹配的行，而 `grep -l` 只会输出匹配的文件名。

在上面的示例中，我们使用 `grep` 来筛选出包含特定字符串的文件。注意，我们使用了 `grep` 而不是 `grep -l`，因为 `grep` 会输出所有匹配的行，而 `grep -l` 只会输出匹配的文件名。







# 13. Functions

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

```
./library.sh
```

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

- 函数名
- 函数参数
- **return** 返回退出状态码
- **echo** 输出到 **stdout**

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

在 shell 中，函数（function）是可以在脚本中调用的。函数可以接受参数，并返回结果。函数可以简化代码，提高可读性。函数可以重用代码，避免重复。函数可以封装逻辑，使代码更清晰。函数可以定义在脚本的任意位置，但必须在调用之前定义。函数可以调用其他函数，形成递归。函数可以调用系统命令，执行操作。函数可以返回退出状态码，表示执行结果。函数可以设置环境变量，影响后续操作。函数可以读取配置文件，获取配置信息。函数可以处理输入输出，与用户交互。函数可以调用库函数，扩展功能。函数可以调用外部程序，实现复杂操作。函数可以调用网络接口，获取数据。函数可以调用数据库，存储数据。函数可以调用硬件设备，控制设备。函数可以调用操作系统接口，管理资源。函数可以调用系统服务，执行任务。函数可以调用其他脚本，实现集成。函数可以调用其他语言编写的程序，实现混合编程。函数可以调用其他工具，提高效率。函数可以调用其他资源，丰富内容。函数可以调用其他方法，解决问题。函数可以调用其他途径，获取信息。函数可以调用其他手段，达成目的。函数可以调用其他方式，实现目标。

```
#!/bin/sh
# A simple script with a function...

add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
  echo "Adding user $USER ..."
```



```

echo useradd -c "$COMMENTS" $USER
echo passwd $USER $PASSWORD
echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

```

```
###
```

```
# Main body of script starts here
```

```
###
```

```
echo "Start of script..."
```

```
add_a_user bob letmein Bob Holness the presenter
```

```
add_a_user fred badpassword Fred Durst the singer
```

```
add_a_user bilko worsepassword Sgt. Bilko the role model
```

```
echo "End of script..."
```

4. () 脚本 在 目录 下 运行 . 脚本 { 参数 , 选项 } 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 .

脚本 在 目录 下 运行 脚本 在 目录 下 运行 echo 脚本 , 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 !

脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 add\_a\_user 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 "Start of script..." 脚本 echo 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 add\_a\_user bob letmein Bob Holness 脚本 在 目录 下 运行 add\_a\_user 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 :

```

$1=bob
$2=letmein
$3=Bob
$4=Holness
$5=the
$6=presenter

```

脚本 在 目录 下 运行 \$1 脚本 在 目录 下 运行 \$1 脚本 在 目录 下 运行 bob 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 '脚本 ' \$1 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 : A=\$1 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 \$A 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 脚本 在 目录 下 运行 \$3 脚本 在 目录 下 运行 \$@ 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 . 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 脚本 在 目录 下 运行 .



--	--	--

--	--

11 11 11 11111 1 11 1 11 1 1 111 . 1111 111 (\$1, \$2,  
 \$@ ) 1 111 1 11 111 . 11 11 1 111 1 1111 :

```
#!/bin/sh
```

myfunc()

{

```
echo "I was called as : $@"
```

 $x=2 \}$ 

```
### Main script starts here
```

```
echo "Script was called with $@"
```

 $x=1$ 

```
echo "x is $x"
```

```
myfunc 1 2 3
```

```
echo "x is $x"
```

scope.sh a b c

Script was called with a b c

x is 1

I was called as : 1 2 3

x is 2

```
$@          \      \      \      \      \      .   \    x\     \
(global)\ , myfunc\ \      \      \      \      \      \  \   \      \
\      \      \
```

```

out.log" 0 1 "x 1" 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038
```

[illegible]



```
#!/bin/sh
```

```
myfunc()
```

```
{  
    echo "\$1 is $1"  
    echo "\$2 is $2"  
    # cannot change $1 - we'd have to say:  
    # 1="Goodbye Cruel"  
    # which is not a valid syntax. However, we can # change $a:  
    a="Goodbye Cruel"  
}
```

```
### Main script starts here
```

```
a=Hello  
b=World  
myfunc $a $b  
echo "a is $a"  
echo "b is $b"
```

❏ ❏ ❏❏ ❏❏ \$a❏ ❏❏ "Hello World"❏ ❏❏ "Goodbye Cruel World"❏ ❏❏ .

## ❏❏ (Recursion)

❏❏ ❏❏❏ ❏ ❏❏❏ . ❏❏❏ ❏❏❏ ❏❏❏ ❏❏❏ ❏❏❏ :

```
#!/bin/sh
```

```
factorial()
```

```
{  
    if [ "$1" -gt "1" ]; then  
        i=`expr $1 - 1`  
        j=`factorial $i`  
        k=`expr $1 \* $j`  
        echo $k  
    else  
        echo 1  
    fi  
}
```



```
while :
do
    echo "Enter a number:"
    read x
    factorial $x
done
```

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

```

## common.lib

```
# common.lib
# Note no #!/bin/sh as this should not spawn
# an extra shell. It's not the end of the world # to have one, but clearer not to.
#
STD_MSG="About to rename some files..."

rename()
{
    # expects to be called as: rename .txt .bak
    FROM=$1
    TO=$2

    for i in *$FROM
    do
        j=`basename $i $FROM`
        mv $i ${j}$TO
    done
}
```

## function2.sh

```
#!/bin/sh
# function2.sh
. ./common.lib
echo $STD_MSG
rename txt bak
```

## function3.sh



```
#!/bin/sh
# function3.sh
. ./common.lib
echo $STD_MSG
rename html html-bak
```

```

common.lib  function2.sh  function3.sh
            .

```

## Exit Codes

[illegible]

```
#!/bin/sh

adduser()
{
    USER=$1
    PASSWORD=$2
    shift ; shift
    COMMENTS=$@
    useradd -c "${COMMENTS}" $USER
    if [ "$?" -ne "0" ]; then
        echo "Useradd failed"
        return 1
    fi
    passwd $USER $PASSWORD
    if [ "$?" -ne "0" ]; then
        echo "Setting password failed"
        return 2
    fi
    echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

## Main script starts here
```

```
#!/bin/sh

adduser()
{
    USER=$1
    PASSWORD=$2
    shift ; shift
    COMMENTS=$@
    useradd -c "${COMMENTS}" $USER
    if [ "$?" -ne "0" ]; then
        echo "Useradd failed"
        return 1
    fi
    passwd $USER $PASSWORD
    if [ "$?" -ne "0" ]; then
        echo "Setting password failed"
        return 2
    fi
    echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

## Main script starts here
```

```
#!/bin/sh

adduser()
{
    USER=$1
    PASSWORD=$2
    shift ; shift
    COMMENTS=$@
    useradd -c "${COMMENTS}" $USER
    if [ "$?" -ne "0" ]; then
        echo "Useradd failed"
        return 1
    fi
    passwd $USER $PASSWORD
    if [ "$?" -ne "0" ]; then
        echo "Setting password failed"
        return 2
    fi
    echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}

## Main script starts here
```



```
if [ "$?" -eq "1" ]; then
    echo "Something went wrong with useradd"
elif [ "$?" -eq "2" ]; then
    echo "Something went wrong with passwd"
else
    echo "Bob Holness added to the system."
fi
```

0 00000      0 00 00 00 (useradd 0 passwd)0 0000      000      000000      00000 . 00  
 00 0 000    000    000    000 00 00 00 00 10 , 00000      000    00 00 00 00 20  
 00000      . 000    00 00 00000      000    000    000    0 0 00000 .



# 14. Hints and Tips

提示: 你可以在 <https://www.shellscript.sh/tips> 找到很多提示。 有些提示是关于 CGI 脚本的，有些是关于 shell 脚本的。

在编写脚本时，请记住，脚本是为了解决问题而编写的。如果你遇到一个问题，不要试图编写一个完美的脚本，而是编写一个能解决问题的脚本。如果你需要，可以使用一些工具，如 (CLI) 或 GUI 工具。请记住，脚本是为了解决问题而编写的，而不是为了展示你的编程技巧。

\* 在编写脚本时，请记住，脚本是为了解决问题而编写的。如果你遇到一个问题，不要试图编写一个完美的脚本，而是编写一个能解决问题的脚本。

在编写脚本时，请记住，脚本是为了解决问题而编写的。如果你遇到一个问题，不要试图编写一个完美的脚本，而是编写一个能解决问题的脚本。

## CGI Scripting

CGI 脚本是一种可以在 Web 服务器上运行的脚本。它们通常用于处理 Web 表单、生成动态内容、与数据库交互等。CGI 脚本通常使用 Perl、Python、Ruby 或 C 编写。在编写 CGI 脚本时，请记住，脚本是为了解决问题而编写的，而不是为了展示你的编程技巧。

## Exit Codes

在 Unix 系统中，每个进程都有一个退出代码。这个代码通常是一个 0 到 255 之间的整数。0 表示成功，非 0 表示失败。在编写 CGI 脚本时，你应该使用适当的退出代码来指示脚本是否成功运行。

在编写 CGI 脚本时，你应该使用适当的退出代码来指示脚本是否成功运行。如果你遇到一个问题，不要试图编写一个完美的脚本，而是编写一个能解决问题的脚本。

在编写 CGI 脚本时，你应该使用适当的退出代码来指示脚本是否成功运行。如果你遇到一个问题，不要试图编写一个完美的脚本，而是编写一个能解决问题的脚本。



❏   ❏❏   ❏❏❏   ❏   ❏❏   ❏❏   ❏❏❏   ❏   ❏❏   ❏❏❏   ❏❏   ❏❏   ❏❏❏❏   .

❏❏ , ❏❏❏   ❏❏   ❏❏❏❏   :

```
#!/bin/sh
# First attempt at checking return codes
USERNAME=`grep "^${1}:" /etc/passwd|cut -d":" -f1`
if [ "$?" -ne "0" ]; then
    echo "Sorry, cannot find user ${1} in /etc/passwd"
    exit 1
fi
NAME=`grep "^${1}:" /etc/passwd|cut -d":" -f5`
HOMEDIR=`grep "^${1}:" /etc/passwd|cut -d":" -f6`

echo "USERNAME: $USERNAME"
echo "NAME: $NAME"
echo "HOMEDIR: $HOMEDIR"
```

❏   ❏❏❏❏   /etc/passwd❏   ❏❏❏   ❏❏❏   ❏❏❏   ❏❏❏❏   ❏❏❏❏   . ❏❏❏   ❏❏❏  
❏❏❏   ❏❏❏❏   ❏❏❏   ❏❏❏❏   ❏❏   ❏❏❏❏   ❏❏   ❏❏   ❏❏❏❏   ❏❏❏   ❏❏❏❏   :

```
USERNAME:
NAME:
HOMEDIR:
```

❏   ❏❏❏❏   ? ❏❏   ❏❏❏❏❏   \$? ❏❏❏   ❏❏❏❏❏   ❏❏❏   ❏❏❏   ❏❏   ❏❏❏   ❏❏❏❏❏   . ❏❏   ❏❏❏ , ❏❏❏  
cut❏❏❏❏ . ❏❏   ❏❏❏❏❏   ❏❏❏❏   ❏❏❏❏   ❏❏   ❏❏   ❏❏❏ , cut❏❏❏❏❏   ❏❏❏❏   ❏❏❏❏   ❏❏❏❏   ❏❏❏  
❏❏❏❏   ❏❏❏❏❏ . ❏❏   ❏❏❏❏❏   ❏❏❏❏❏   ❏❏   ❏❏❏❏❏   ❏❏❏❏   ❏❏   ❏❏❏❏❏   ❏❏❏❏❏❏❏ .  
❏❏❏   ❏❏❏❏   ❏❏   ❏❏❏❏❏ ? ❏❏❏❏   ❏❏❏❏   ❏❏❏❏   grep❏❏❏❏❏❏❏   ❏❏❏❏   ❏❏❏❏❏❏❏❏ . ❏❏❏❏   cut❏❏  
❏❏   grep❏❏❏   ❏❏❏   ❏❏❏❏❏❏❏   ❏❏❏❏❏❏❏ .

```
#!/bin/sh
# Second attempt at checking return codes
grep "^${1}:" /etc/passwd > /dev/null 2>&1
if [ "$?" -ne "0" ]; then
    echo "Sorry, cannot find user ${1} in /etc/passwd"
    exit 1
fi
USERNAME=`grep "^${1}:" /etc/passwd|cut -d":" -f1`
NAME=`grep "^${1}:" /etc/passwd|cut -d":" -f5`
HOMEDIR=`grep "^${1}:" /etc/passwd|cut -d":" -f6`
```



```
echo "USERNAME: $USERNAME"
echo "NAME: $NAME"
echo "HOMEDIR: $HOMEDIR"
```

Das ist ein Skript, das die Umgebungsvariablen USERNAME, NAME und HOMEDIR ausliest und sie auf dem Bildschirm ausgibt. Es ist ein einfaches Shell-Skript, das in einem Texteditor erstellt werden kann.

Das Skript ist in einem Texteditor erstellt worden. Es ist ein einfaches Shell-Skript, das in einem Texteditor erstellt werden kann.

```
#!/bin/sh
# A Tidier approach

check_errs()
{
    # Function. Parameter 1 is the return code
    # Para. 2 is text to display on failure.
    if [ "${1}" -ne "0" ]; then
        echo "ERROR # ${1} : ${2}"
        # as a bonus, make our script exit with the right error code. exit ${1}
    fi
}

### main script starts here ###

grep "^${1}:" /etc/passwd > /dev/null 2>&1
check_errs $? "User ${1} not found in /etc/passwd"
USERNAME=`grep "^${1}:" /etc/passwd|cut -d":" -f1`
check_errs $? "Cut returned an error"
echo "USERNAME: $USERNAME"
check_errs $? "echo returned an error - very strange!"
```

Das Skript ist ein Shell-Skript, das die Umgebungsvariablen USERNAME, NAME und HOMEDIR ausliest und sie auf dem Bildschirm ausgibt. Es ist ein einfaches Shell-Skript, das in einem Texteditor erstellt werden kann.

Das Skript ist ein Shell-Skript, das die Umgebungsvariablen USERNAME, NAME und HOMEDIR ausliest und sie auf dem Bildschirm ausgibt. Es ist ein einfaches Shell-Skript, das in einem Texteditor erstellt werden kann.



```
#!/bin/sh
cd /usr/src/linux && \
    make dep && make bzImage && make modules && \
    make modules_install && \
    cp arch/i386/boot/bzImage /boot/my-new-kernel && \ cp System.map /boot && \
    echo "Your new kernel awaits, m'lord."
```

`if` `(` `int` `a` `=` `0` `)` `{` `int` `b` `=` `1` `;}`

```
#!/bin/sh
cd /usr/src/linux
if [ "$?" -eq "0" ]; then
    make dep
    if [ "$?" -eq "0" ]; then
        make bzImage
        if [ "$?" -eq "0" ]; then
            make modules
            if [ "$?" -eq "0" ]; then
                make modules_install
                if [ "$?" -eq "0" ]; then
                    cp arch/i386/boot/bzImage /boot/my-new-kernel
                    if [ "$?" -eq "0" ]; then
                        cp System.map /boot/
                        if [ "$?" -eq "0" ]; then
                            echo "Your new kernel awaits, m'lord."
                        fi
                    fi
                fi
            fi
        fi
    fi
fi
```

...      .

☐☐ && ☐ || ☐☐ AND ☐ OR ☐☐☐ ☐☐☐ ☐☐☐☐☐☐ . ☐ ☐ ☐ ☐☐ ☐ ☐ ,  
☐ :



```
#!/bin/sh
cp /foo /bar && echo Success || echo Failed
```

```
#!/bin/sh
cp /foo /bar && echo Success || echo Failed
```

```

$ echo "hello" | cat -n
 1 hello

```

Success

Failed

□□ cp □□ □□□□ □□□□ □□ □□□□ . □□ □□ □□ □□ :

```
command && command-to-execute-on-success \  
|| command-to-execute-on-failure
```

□ □□□ □□ □□ □□ □ □□□ . □ □□ □□ □ /□ □□□□ □□□□ , □  
□ □□ □□ □□□□ □ □□ &&□ ||□ □ □□ □□□ □ □□ □□□ . □  
□ □□ □ □□□□ . □□ □ □□ □□ □ □□□ □□ □ □□□ .

[illegible]

```
cp /foo /bar && \
( echo Success ; echo Success part II; ) || \
( echo Failed ; echo Failed part II )
```

0000 0000 Marcel 0000 0000 0000 0000 00 00000 . 0000 000 000  
 0000 :

```
( command1 ; command2; command3 )
```

```

#####   ##   #####   ##   ##   (# ##### command3)##   #####   .   ##   #####   ##   #####
#####   .#####   ##   #####   #####   :

```

```
cp /foo /bar && \
( echo Success ; echo Success part II; /bin/false ) ||\
( echo Failed ; echo Failed part II )
```

```
cp /bin/false
```

:



Success  
Success part II  
Failed  
Failed part II

Success  
Success part II  
Failed  
Failed part II

Success  
Success part II  
Failed  
Failed part II

Success  
Success part II  
Failed  
Failed part II

$\square \square \square \quad \square \square \quad \square \square \square \quad \square \square \square \quad \square \square \quad \square \square \quad \square \square \square \quad \square \square \square \square \quad \square \square \quad \square \square \quad \square \square \quad \text{if, then, else} \quad \square \square \square \quad \square \square \square \square \quad \square \square \quad \square$   
 $\square \square \quad \square \square \quad \square \square \square \square \square \quad .$

## Simple Expect Replacement

expect, Sun Microsystems Explorer, .  
 , Sun Microsystems Explorer, .  
 .

expect.txt    ☐    ☐    ☐    ☐    :

S command E[delay] expected\_text

```

00000000 0000 "S"(Send 0000 ) 0000 0000 , 00 0000 "E" 00000000 . 00 0000 0000
00000000 00 0000 0000 0000 0000 00 00 0000 0000 0000 : "E10 $" 10 00
00000000 0 00 000000 000000 . 00 0000 00 0000 000000 1 00 0000 0 00
00000000 , 2 00 , 3 00 0000 00 00 0000 000000 MAX_WAITS 0000 0000 0000 0000
00000000 . 0000 00 000000 "E $" 00 000000 0000 0000 000000 .

```

**MAX\_WAITS=5**    5    1+2+3+4+5=15    .

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```

```
#!/bin/sh

# expect.sh | telnet > file1

host=127.0.0.1

port=23

file=file1

MAX_WAITS=5


echo open ${host} ${port}


while read l
do
c=`echo ${l}|cut -c1`
if [ "${c}" = "E" ]; then
    expected=`echo ${l}|cut -d" " -f2-`
```



```

delay=`echo ${l}|cut -d" " -f1|cut -c2-`
if [ -z "${delay}" ]; then
    sleep ${delay}
fi
res=1
i=0
while [ "${res}" -ne "0" ]
do
    tail -1 "${file}" 2>/dev/null | grep "${expected}" > /dev/null
    res=$?
    sleep $i
    i=`expr $i + 1`
    if [ "${i}" -gt "${MAX_WAITS}" ]; then
        echo "ERROR : Waiting for ${expected}" >> ${file}
        exit 1
    fi
done
else
    echo ${l} |cut -d" " -f2-
fi
done < expect.txt

```

### ##### :

```
$ expect.sh | telnet > file1
```

### ## ##### file1### ## . ## ## , /tmp ## ls, cal ##  
 ### ## . ## ## :

```

telnet> Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.

declan login: steve
Password:
Last login: Thu May 30 23:52:50 +0100 2002 on pts/3 from localhost.
No mail.
steve:~$ ls /tmp
API.txt          cgihtml-1.69.tar.gz      orbit-root
cal
a.txt            cmd.txt                  orbit-steve

```



```
apache_1.3.23.tar.gz  defaults.cgi          parser.c
b.txt                diary.c                patchdiag.xref
background.jpg       drops.jpg              sh-thd-1013541438
blocks.jpg           fortune-mod-9708.tar.gz  stone-dark.jpg
blue3.jpg            grey2.jpg              water.jpg
c.txt                jpsock.131.1249
steve:~$ cal
      May 2002
Su Mo Tu We Th Fr Sa
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
steve:~$ exit
logout
```

# Trap

```
Trap[] [][] [][] [] . [] [][] [] [] FOO[] BAR[] [] []
[] [] [] [] [] [] [] [] [] [] , [][] [] [] /tmp[] [][] []
[] [] [] [] [] /tmp[] [] [] [] [] :
```

```
#!/bin/sh
```

```
trap cleanup 1 2 3 6
```

```
cleanup()
```

```
{
  echo "Caught Signal ... cleaning up."
  rm -rf /tmp/temp_*. $$
  echo "Done cleanup ... quitting."
  exit 1
}
```

```
### main script
```

```
for i in *
```

```
do
```



```
sed s/FOO/BAR/g $i > /tmp/temp_${i}.$$ && mv /tmp/temp_${i}.$$ $i
done
```

```
trap cleanup signal 1, 2, 3, 6 cleanup()
(CTRL-C) signal 2(SIGINT) .
```

```
#!/bin/sh

trap 'increment' 2

increment()
{
    echo "Caught SIGINT ..."
    X=`expr ${X} + 500`
    if [ "${X}" -gt "2000" ]
    then
        echo "Okay, I'll quit ..."
        exit 1
    fi
}

### main script
X=0
while :
do
    echo "X=$X"
    X=`expr ${X} + 1`
    sleep 1
done
```

```

. CTRL-C
.
4
( 2000 )
kill -9 <PID>
```

:

Number	SIG	
0	0	
1	SIGHUP	
2	SIGINT	



3	SIGQUIT	⏏ (Quit)
6	SIGABRT	⏏ (Abort)
9	SIGKILL	⏏ Kill⏏ (⏏ ⏏ ⏏ )
14	SIGALRM	⏏ ⏏
15	SIGTERM	⏏ (Terminate)

⏏⏏⏏⏏ ⏏⏏⏏⏏ ⏏⏏ ⏏⏏⏏ ⏏⏏ (⏏ : **nohup** ⏏⏏ ⏏⏏ )⏏⏏ ⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏ ⏏⏏  
⏏⏏⏏ ⏏⏏⏏⏏ .

# echo: -n vs \c

⏏⏏ ⏏⏏⏏⏏⏏⏏ , echo ⏏⏏ ⏏⏏⏏ ⏏⏏⏏ ⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏ . ⏏⏏ ⏏⏏ ⏏⏏ ⏏⏏⏏  
⏏⏏⏏⏏ ... ⏏⏏ , ⏏⏏ ⏏⏏⏏⏏⏏ ⏏⏏ ⏏⏏ ⏏⏏⏏ ⏏⏏⏏⏏⏏⏏ .

⏏⏏ Unix ⏏⏏⏏⏏⏏⏏ **echo -n message**⏏⏏ ⏏⏏⏏⏏ echo⏏⏏ ⏏⏏ ⏏⏏⏏⏏ ⏏⏏⏏ ⏏⏏⏏⏏⏏⏏ , ⏏⏏  
⏏⏏⏏⏏⏏⏏ **echo message \c**⏏⏏ ⏏⏏⏏⏏ ⏏⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ :

```
echo -n "Enter your name:"
read name
echo "Hello, $name"
```

⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏ ⏏⏏⏏⏏ ⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏ :

```
Enter your name: Steve
Hello, Steve
```

⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ ⏏⏏⏏⏏ ⏏⏏ ⏏⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ :

```
echo "Enter your name: \c"
read name
echo "Hello, $name"
```

⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ ⏏⏏⏏⏏ ⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ .

⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ . ⏏⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ ⏏⏏⏏⏏⏏⏏ ⏏⏏ ⏏⏏⏏⏏⏏⏏⏏ :

```
if [ "`echo -n`" = "-n" ]; then
  n=""
  c="\c"
```



```
else
    n="-n"
    c=""
fi
```

```
echo $n Enter your name: $c
read name
echo "Hello, $name"
```

echo -n 空格 空格 -n 空格 echo 空格 , 空格 \$n 空格 空格 空格  
\$c \c 空格 . 空格 空格 空格 空格 空格 \$n -n 空格 \$c 空格 空格  
 空格 .

空格 空格 空格 空格 cut 空格 空格 空格 . 空格 空格 空格 空格  
 空格 空格 空格 空格 空格 空格 空格 .

grep 空格 空格 空格 空格 空格 . grep 空格 空格 空格 :

```
#!/bin/sh
steves=`grep -i steve /etc/passwd | cut -d: -f1`
echo "All users with the word \"steve\" in their passwd"
echo "Entries are: $steves"
```

空格 空格 空格 空格 空格 空格 空格 空格 . 空格 /etc/passwd "steve" 空格  
 空格 空格 空格 空格 空格 空格 空格 :

```
$> grep -i steve /etc/passwd
steve:x:5062:509:Steve Parker:/home/steve:/bin/bash
fred:x:5068:512:Fred Stevens:/home/fred:/bin/bash
$> grep -i steve /etc/passwd |cut -d: -f1
steve
fred
```

空格 空格 空格 :

Entries are: steve fred

空格 空格 空格 NEWLINE 空格 空格 . sh 空格 空格 \$IFS 空格 空格  
 空格 空格 空格 空格 . IFS 空格 <space><tab><cr> 空格 . 空格 空格  
NEWLINE 空格 空格 空格 : 空格 NEWLINEs.... 空格 空格 空格 空格 空格 . 空格 tr  
 空格 空格 空格 :



```
#!/bin/sh
steves=`grep -i steve /etc/passwd | cut -d: -f1`
echo "All users with the word \"steve\" in their passwd"
echo "Entries are: "
echo "$steves" | tr ' ' '\012'
```

tr 把 8 个 \012(NEWLINE) 插入到 steves 中。tr 把 空格 替换为 \012。结果如下：

```
#!/bin/sh
steves=`grep -i steve /etc/passwd | cut -d: -f1`
echo "All users with the word \"steve\" in their passwd"
echo "Entries are: "
echo "$steves" | tr ' ' '\012' | tr '[a-z]' '[A-Z]'
```

tr [a-z] [A-Z] 把 a-z 替换为 A-Z。a-z 和 A-Z 的 ASCII 码相差 32。所以，把 a-z 替换为 A-Z 就是把 ASCII 码加上 32。tr [:lower:] [:upper:] 把小写字母替换为大写字母。tr [:upper:] [:lower:] 把大写字母替换为小写字母。tr [:lower:] [:upper:] 把小写字母替换为大写字母。tr [:upper:] [:lower:] 把大写字母替换为小写字母。

## Cheating

在 Linux 中，我们可以使用 sed 和 awk 来对文本进行批量操作。

sed 和 awk 都是流编辑器。sed 主要用于对文本进行批量替换、删除、插入等操作。awk 主要用于对文本进行批量计算、统计等操作。sed 和 awk 的语法都非常简洁，使用起来非常方便。

sed 和 awk 都是流编辑器，这意味着它们可以逐行处理文本。sed 的语法非常简单，只需要一行命令就可以完成复杂的操作。awk 的语法稍微复杂一些，但也非常灵活，可以满足各种需求。

## Cheating with awk

awk 是一个非常强大的文本处理工具，它可以对文本进行批量操作。awk 的语法非常简单，只需要一行命令就可以完成复杂的操作。

```
$ wc hex2env.c
102 189 2306 hex2env.c
```

awk 是一个非常强大的文本处理工具，它可以对文本进行批量操作。awk 的语法非常简单，只需要一行命令就可以完成复杂的操作。



NO\_LINES=`wc -l file`

[illegible]

```
NO_LINES=`wc -l file | awk '{ print $1 }`'
```

```

00 NO_LINES 00 10200 .

```

## Cheating with sed

[illegible]

```
sed s/eth0/eth1/g file1 > file2
```

```
1 # eth0 2 eth1 . 3 tr 4 .tr 5 :
```

```
echo ${SOMETHING} | sed s/"bad word"/g
```

[illegible]

This line is okay.

This line contains a bad word. Treat with care.

This line is fine, too.

```
grep -l -E '[0-9]{1,3} [0-9]{1,3} [0-9]{1,3}' sed : 
```

This line is okay.

This line contains a . Treat with care.

This line is fine, too.

## Telnet hint

Sun Explorer .  
 .  
 :



```
$ ./telnet1.sh | telnet
```

telnet 127.0.0.1 23  
Trying 127.0.0.1: 23...  
Connected to 127.0.0.1.  
Escape character is '^['.  
#!/bin/sh  
host=127.0.0.1  
port=23  
login=steve  
passwd=hellothere  
cmd="ls /tmp"

```
#!/bin/sh
```

```
host=127.0.0.1
```

```
port=23
```

```
login=steve
```

```
passwd=hellothere
```

```
cmd="ls /tmp"
```

```
echo open ${host} ${port}
```

```
sleep 1
```

```
echo ${login}
```

```
sleep 1
```

```
echo ${passwd}
```

```
sleep 1
```

```
echo ${cmd}
```

```
sleep 1
```

```
echo exit
```

Sun Jul 1 12:00:00 2012 (telnet 127.0.0.1 23)  
telnet 127.0.0.1 23: Connected to 127.0.0.1.

```
$ ./telnet2.sh | telnet > file1
```

```
#!/bin/sh
```

```
# telnet2.sh | telnet > FILE1
```

```
host=127.0.0.1
```

```
port=23
```

```
login=steve
```

```
passwd=hellothere
```

```
cmd="ls /tmp"
```

```
timeout=3
```

```
file=file1
```

```
prompt="$"
```

```
echo open ${host} ${port}
```



```
sleep 1
tout=${timeout}
while [ "${tout}" -ge 0 ]
do
    if tail -1 "${file}" 2>/dev/null | \
        egrep -e "login:" > /dev/null
    then
        echo "${login}"
        sleep 1
        tout=-5
        continue
    else
        sleep 1
        tout=`expr ${tout} - 1`
    fi
done

if [ "${tout}" -ne "-5" ]; then
    exit 1
fi

tout=${timeout}
while [ "${tout}" -ge 0 ]
do
    if tail -1 "${file}" 2>/dev/null | \
        egrep -e "Password:" > /dev/null
    then
        echo "${passwd}"
        sleep 1
        tout=-5
        continue
    else
        if tail -1 "${file}" 2>/dev/null | \
            egrep -e "${prompt}" > /dev/null
        then
            tout=-5
        else
            sleep 1
            tout=`expr ${tout} - 1`
        fi
    fi
done
```



```
    fi
done

if [ "${tout}" -ne "-5" ]; then
    exit 1
fi

> ${file}

echo ${cmd}
sleep 1
echo exit
```

□ □□□□ □□ file1□ □□□ , □ □□ □□ □□□□□ □□ □□ □□□ □ □□□□ .  
"> \${file}"□ □□□ □□ □□□ □□ □□ □□□□ □□ □□ □□ □□ □□□□ .



# 15. Quick Reference

This table lists the most commonly used shell metacharacters and their functions.

Metacharacter	Function	Example
&	Background process	ls &
&&	AND	if [ "\$foo" -ge "0" ] && [ "\$foo" -le "9" ]
	OR	if [ "\$foo" -lt "0" ]    [ "\$foo" -gt "9" ] (not in Bourne shell)
^	Start of line	grep "^foo"
\$	End of line	grep "foo\$"
=	String comparison (cf. -eq)	if [ "\$foo" = "bar" ]
!	NOT	if [ "\$foo" != "bar" ]
\$\$	Current PID	echo "my PID = \$\$"
\$_	Previous command's exit status	ls & echo "PID of ls = \$_"
\$?	Current command's exit status	ls ;
\$0	Current script's name	echo "I am \$0"
\$1	First argument	echo "My first argument is \$1"
\$9	Ninth argument	echo "My ninth argument is \$9"
\$@	All arguments	echo "My arguments are \$@"
\$*	All arguments	echo "My arguments are \$*"



-eq	if [ "\$foo" = "9" ]	if [ "\$foo" -eq "9" ]
-ne	if [ "\$foo" != "9" ]	if [ "\$foo" -ne "9" ]
-lt	if [ "\$foo" < "9" ]	if [ "\$foo" -lt "9" ]
-le	if [ "\$foo" <= "9" ]	if [ "\$foo" -le "9" ]
-gt	if [ "\$foo" > "9" ]	if [ "\$foo" -gt "9" ]
-ge	if [ "\$foo" >= "9" ]	if [ "\$foo" -ge "9" ]
-z	if [ -z "\$foo" ]	if [ -z "\$foo" ]
-n	if [ -n "\$foo" ]	if [ -n "\$foo" ]
-nt	if [ "\$filea" -nt "\$fileb" ]	if [ "\$filea" -nt "\$fileb" ]
-d	if [ -d /bin ]	if [ -d /bin ]
-f	if [ -f /bin/ls ]	if [ -f /bin/ls ]
-r	if [ -r /bin/ls ]	if [ -r /bin/ls ]
-w	if [ -w /bin/ls ]	if [ -w /bin/ls ]
-x	if [ -x /bin/ls ]	if [ -x /bin/ls ]
function myfunc() { echo hello }		



# 16. Interactive Shell

Both UNIX and Linux have interactive shells. The most common interactive shell is `bash`. It is the default shell for most users. To run `bash`, type `bash` or `/bin/sh` at the prompt.

## bash

`bash` has many features. It is a command-line interpreter. It can execute commands and scripts. It can also be used to edit files. Press `Ctrl+R` to search for a command. Press `ESC` to enter vi mode. Press `ESC` to exit vi mode.

Press `Ctrl+U` to delete the current line. Press `Ctrl+K` to delete the current line. Press `Ctrl+D` to exit the shell.

```
bash$ ls /tmp
(list of files in /tmp)
bash$ touch /tmp/foo
bash$ !
ls /tmp
(list of files in /tmp, now including /tmp/foo)
```

Press `PageUp` or `PageDn` to scroll through the command history. Press `Ctrl+N` to search for a command.

## ksh

`vi` and `emacs` are text editors. `ksh` is a command-line interpreter. It can execute commands and scripts. It can also be used to edit files. Press `set -o vi` to enter vi mode. Press `set -o vi` to exit vi mode. Press `exec ksh -o vi` to enter vi mode. Press `exec ksh -o vi` to exit vi mode.

Press `Ctrl+U` to delete the current line. Press `Ctrl+K` to delete the current line. Press `Ctrl+D` to exit the shell.

```
csh% # oh no, it's csh!
csh% ksh
ksh$ # phew, that's better ksh$ # do some stuff under ksh
ksh$ # then leave it back at the csh prompt: ksh$ exit
csh%
```



Now we'll use `ksh` to run `csch`, to see how it works. We'll use `csch` to run `ksh` to see how it works:

```
csch% # oh no, it's csch!  
csch% exec ksh  
ksh$ # do some stuff under ksh ksh$ exit
```

login:

Now we'll use `csch` to run `ksh` to see how it works.

We'll use `csch`:

```
csch% ksh  
ksh$ set -o vi  
ksh$ # You can now edit the history with vi-like commands,  
# and use ESC-k to access the history.
```

`ESC-k` will take you to the previous command. We'll use `vi` to edit the command history:

```
ksh$ touch foo  
ESC-k (enter vi mode, brings up the previous command)  
w (skip to the next word, to go from "touch" to "foo")  
cw (change word) bar (change "foo" to "bar")  
ksh$ touch bar
```