

# 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

Perl 的哲学思想源自 Unix 的哲学思想。Unix 的哲学思想是：每个程序都应该只做一件事，并且把它做好。

- 每个程序都应该只做一件事，并且把它做好。C 语言是编写 Unix 系统工具的首选语言，Perl 则是编写系统工具的首选语言。
- 每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

- 每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。
- 每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

1. 每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。
2. 每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

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

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

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

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。

Perl 的哲学思想是：每个程序都应该只做一件事，并且把它做好。Perl 语言是编写系统工具的首选语言，C 语言则是编写系统工具的首选语言。



# 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` 来指定脚本使用的解释器。我们使用 `echo` 来指定脚本使用的解释器。我们使用 `echo` 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 `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
$
```

이 단락을 읽고 !이 단락을 읽고 이 단락을 :

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

이 이 단락을 읽고 이 단락을 .  
이 , echo 이 이 이 이 이 이 . "Hello" "World" 이 이 이 이 이 .  
이 이 이 ? 이 이 이 이 이 ?  
이 이 이 이 이 이 이 .  
이 이 ! 이 이 이 echo 이 이 이 , 이 이 이  
이 이 이 cp 이 이 이 . 이 이 이 :

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

이 이 . 이 이 이 이 이 이 이 . 이 이  
이 이 이 이 이 이 이 이 이 이 이 이  
이 이 : ?  
이 **echo** 이 이 이 , 이 "Hello World" 이 이 . 이 이 이  
이 .  
이 이 이 이 이 이 이 이 이 이 이 이 .  
이 이 이 이 이 이 이 .  
이 이 이 이 이 이 . 이 이 이 :

```
#!/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
```

이 이 이 ? 이 이 이 ! 이 이 이  
이 .... , echo 이 이 이 이 이 !



## 4. Variables - Part I

```

#!/bin/sh
# 脚本名称: var1.sh
# 作者: 张三
# 版本: 1.0
# 描述: 打印 Hello World 并设置环境变量。

# 打印 Hello World
echo "Hello World"

# 设置环境变量
VAR="Hello"

# 打印环境变量
echo $VAR

# 打印脚本名称
echo $0

```

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

```
00000000 00000000 "Hello World" 00000000 MY_MESSAGE 00000000 00000000 00000000 00000000 00000000 .
```

Hello World 0000 0000 0000 0000 0000 0000 0000 0000 . echo00 00 0000 0000 0000  
 00 0000 Hello World00 0000 00 0000 , 0000 0000 00 0000 00 0000 0000 0000 00  
 0000 00 00 0000 0000 0000 0000 0000 . 0000 0000 00 **MY\_MESSAGE=Hello**  
 00 0000 00 World 0000 0000 0000 .

[illegible]


















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


















□□ □□ □□□ expr□ □□ □□□ □□□□ . □□ □ □□□ □□□ □□ □□□ :

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

```
MY_OTHER_PI="3.142"
MY_MIXED=123abc
```

```
MY_OTHER_PI="3.142"
MY_MIXED=123abc
```















 :

```
#!/bin/sh

echo What is your name?

read MY_NAME

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

```
#!/bin/sh

echo What is your name?

read MY_NAME

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

```
#!/bin/sh

echo What is your name?

read MY_NAME

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

```
#!/bin/sh

echo What is your name?

read MY_NAME

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

0000 00000 00000 00 3000 00000 000 "you're"00 000 00000 0000  
 000 000 00000 000 00000 . 0 00000 000 000 00 000 000  
 00000 !

```

read
echo
MY_MESSAGE

```

???

□ □ □ □ C □ □ □ □ □ □ □ □ □ □ . □ □ □ □ □ □ □ □ □ □  
 □ □ □ □ □ □ . □ □ □  
 □ - □ □ □ □ □

```
MY_OBFUSCATED_VARIABLE=Hello
```


 11

```
echo $MY_OSFUCATED_VARIABLE
```

  
  
  
  
(       OBFUSCATED             ).

`export`

```
myvar2.sh : 
```



```
#!/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
```

```
# #   #   #   ##### ! #   # .profile #   .bash_profile #   #  
#####  
# # MYVAR# #   #   #   #####  
echo MYVAR# # , $MYVAR echo### #   #   # sway##  
##### . # #####    ##   #   #   #   #   #  
###   ###   #   #   ####      : 
```

```
#!/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
```

```

// 1. Create a file named 'steve_file' in the 'USER_NAME' directory.
// 2. Write the string 'steve' to the file.
// 3. Close the file.
// 4. Print the file path.
// 5. Return 0.

```

```
#!/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"
```

```
-- USER_NAME VARCHAR(60) NOT NULL;
-- "_file" VARCHAR(100);
-- . VARCHAR(10);
-- .
```

```
$ echo "${USER_NAME}_file" touch Steve Parker_file . , Steve Parker_file .
```

□□□□ Chris□□ □□□□□□ .

# 5. ??????(Wildcards)

????? ???? ???? ???? ? ???? ???? ???? ???? ???? .  
???? ? ???? ???? ???? ???? ???? ???? . ???? ? ????  
???? ???? ???? ? ???? ???? ???? ???? , ???? ???? ???? ????  
????? ???? ???? ???? . ???? ???? ???? ???? ???? ???? .  
??? /tmp/a/ ???? ???? /tmp/b/ ???? ???? ???? ???? . ???? .txt ???? ???? .html ???? ?  
???? ???? ???? :

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

??? ls /tmp/a/ ???? ???? /tmp/a/ ???? ???? ???? ???? ???? ?  
echo /tmp/a/\* ???? ???? ls ???? ???? ???? ???? ???? ???? ?  
???? ???? ???? ?  
??? .txt ???? ???? .bak ???? ???? ???? ???? ???? .

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

? ???? ???? ???? ???? ? ???? , ???? ???? ???? mv ???? ???? ?  
???? ???? mv ???? echo ???? ???? ???? .  
??? ???? ???? ???? ???? ???? ???? ???? ???? .

## 6. ?????? ??

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

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

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

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

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

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

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

- "Hello "
- World
- ""

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

Hello World

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

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

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。我们使用 `echo` 命令来测试一下。

```
$ 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
```

□ □ □□ \*□ □□□ □ □ □□ □□□□ . □ □ □□ \*txt□ txt□  
 □□ □ □□ □□□□ . □ □ □□ \*□ □□□ □ □ □ □ □□□□ . □ □  
 □□ □□ □□□ □□□ txt□ □□□□ .

□□ " , \$ , ` □ \ □ □□ □ □ □□ □□ □□□□ . □□□ (\) □□ □□ □ □ □□  
 □□ □ □□□ □□ □□ □ □ □ □ □ □ (□ : echo)□ □□□□ . □□ □□□  
 □□ □ □ □□□ : (\$X□ □ □ 5□ □ □ )

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

□□ □ □ □□ □□ :

```
$ 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.
```

" □ □□ □□□ □ □□ □□ □□□□□□ . □□ (\$)□ □□ □□□ □□ □□□ , \$X□  
 □ □ 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
```

□□ □□□ □□ □□□□ □□□ □ □ □□ □□□□ □ □ □□ □□ . □ □ □ □ □ ,  
 □□ □□□ 12□ "□ □□□ "□ □□□□ □□□□□ .

7. ??

1. 在 20 以内，找出所有能被 3 整除的数，并计算它们的和。

For ??

"for"         :

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

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

[illegible]







 (






 ),










 :

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

```
Looping ... i is set to hello
Looping ... i is set to 1
```

Looping ... i is set to goodbye

## While ??

```

while read line
do
    if [ -z "$line" ]
    then
        continue
    fi
    echo "$line"
done < myfile.txt

cat myfile.txt | while read line
do
    echo "$line"
done

while read line
do
    echo "$line"
done < myfile.txt

while read line
do
    echo "$line"
done < myfile.txt

```



```
$input_text=$(cat /dev/urandom | tr -dc 'a-z' | fold -w 10 | head -n 1 | sed 's/ / /g')
echo "Australian" | echo $(cat /dev/urandom | tr -dc 'a-z' | fold -w 10 | head -n 1 | sed 's/ / /g') | myfile.txt
cat /dev/urandom | tr -dc 'a-z' | fold -w 10 | head -n 1 | sed 's/ / /g' | echo "Unknown Language: $input_text"
echo "$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
```

A sequence of five rectangles. The first rectangle has a width of 2 and a height of 1. The second rectangle has a width of 1 and a height of 2. The third rectangle has a width of 3 and a height of 2. The fourth rectangle has a width of 2 and a height of 3. The fifth rectangle has a width of 4 and a height of 3. This is followed by an ellipsis.

```
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。它通常用于脚本中，以根据条件的真假来执行不同的操作。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` 命令来检查变量 `foo` 是否等于 `bar`：

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

在上面的例子中，`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，则执行后面的命令。需要注意的是，在 `test` 命令中，字符串需要用双引号括起来，以避免空格等字符引起的问题。

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

在上面的例子中，我们使用 `SPACE` 来代表空格。可以看到，`test` 命令在比较字符串时，对空格的敏感性。通常，我们使用 `test` 命令来比较字符串是否相等，可以使用 `test` 命令的 `=` 操作符。

`test` 命令还支持其他操作符，如 `-eq` 用于比较数字是否相等。要了解更多关于 `test` 命令的信息，可以使用 `man test` 命令查看手册页。

`test` 命令通常与 `if`、`while` 等控制语句一起使用。例如，我们可以使用 `test` 命令来编写一个简单的 `if...then...else...fi` 结构：

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

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

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

```
elif
then
:
```

```
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"
```

```

if (test) {
    // if test is true, do this
} else {
    // if test is false, do this
}

// if test is true, do this
// if test is false, do this

```

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
```

[illegible]

```
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
```

```

[[{"id": 1, "name": "John", "age": 30, "gender": "Male", "email": "john.doe@example.com", "phone": "123-456-7890"}, {"id": 2, "name": "Jane", "age": 25, "gender": "Female", "email": "jane.smith@example.com", "phone": "987-654-3210"}, {"id": 3, "name": "Bob", "age": 35, "gender": "Male", "email": "bob.jones@example.com", "phone": "555-111-2222"}, {"id": 4, "name": "Alice", "age": 28, "gender": "Female", "email": "alice.brown@example.com", "phone": "444-333-2222"}, {"id": 5, "name": "Charlie", "age": 32, "gender": "Male", "email": "charlie.white@example.com", "phone": "777-888-9999"}]]

```

while test [ -n "\$X" ] :

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

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

:  
 :

\$

:  
 :

```
#!/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
```





# 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，输入其他内容，将看到 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，输入其他内容，将看到 That's all folks!。

```
$ ./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 [[ [ [[[ [[ [[ [[[[ , [[ [[ INPUT\_STRING[[ [[  
[[[[ [[ [[[[[[ .

[[ [[ [[ [[[[ [[[[ [[[[ [[ [[ [[[[ hello) [[ bye) [[ [[[[ . [[ ,  
INPUT\_STRING[[ hello[[ [[[[ [[ [[ [[[[ [[ [[[[[[ [[[[ , INPUT\_STRING[[ "bye"  
[[ [[[[ "goodbye"[[[[ [[[[ [[[[ [[[[ . [[[[ [[ [[[[ break [[  
exit [[ [[[[ [[[[ . [[[[ [[ [[ [[ [[[[ \*) [[ [[ catch-all [[[[ , [[[[ [[[[ test  
[[ [[ [[ [[ [[[[ [[[[ [[[[ [[[[ [[[[ [[[[ [[ [[ [[[[ .

[[ case [[ esac([[[[[[ [[[[ !)[[[ [[[[ , done[[ while [[[[ [[[[[[ .

Case [[[[ [[[[[[ [[[[ [[ [[[[ [[[[ [[[[ [[ [[ [[[[ . case [[[[ [[ [[[[[[  
[[[[ [[[[[[ [[ [[[[ [[ [[ [[[[ [[[[ , [[ [[[[ [[[[[[ .

# 10. Variables - Part II

이 단락에서는 변수, 특히, 셸 변수에 대해 알아보겠습니다. 이 단락에서는 변수의 선언, 사용, 그리고 변수의 범위(scope)에 대해 알아보겠습니다.

이 단락에서는 변수의 선언, 사용, 그리고 변수의 범위(scope)에 대해 알아보겠습니다. 변수의 선언은 변수의 이름을 만들고, 변수의 사용은 변수의 값을 출력하거나, 변수의 범위는 변수가 어디서 어디서 사용되는지를 나타냅니다. 변수의 선언은 변수의 이름을 만들고, 변수의 사용은 변수의 값을 출력하거나, 변수의 범위는 변수가 어디서 어디서 사용되는지를 나타냅니다. 변수의 선언은 변수의 이름을 만들고, 변수의 사용은 변수의 값을 출력하거나, 변수의 범위는 변수가 어디서 어디서 사용되는지를 나타냅니다.

```
#!/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 $@"
```

이 단락에서는 변수의 선언, 사용, 그리고 변수의 범위(scope)에 대해 알아보겠습니다.

```
$ /home/steve/var3.sh
I was called with 0 parameters
My name is /home/steve/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
```

\$0은 변수의 이름을 나타내며, \$1은 변수의 값을 나타냅니다. 이 단락에서는 변수의 선언, 사용, 그리고 변수의 범위(scope)에 대해 알아보겠습니다.

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

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

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

参数 参数 \$# 0 参数 参数 , 参数 参数 参数 参数 shift 参数 参数 .

参数 参数 参数 \$? 参数 参数 参数 参数 参数 参数 . 参数 参数 :

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

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

“参数 参数 参数 参数 参数 参数 0 参数 参数 C 参数 参数 参数 参数 参数 参数 .”  
(Robert Firth)

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

参数 参数 参数 IFS 参数 . 参数 参数 参数 参数 . 参数 SPACE TAB NEWLINE 参数 参数 参数 参数 参数 参数 参数 参数 :

```
#!/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"
```

❏ ❶❷❸❹ ❺❻ ❼ ❶❷❸❹ :

```
$ ./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
```

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

```
$ ./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❏❏ ❶❷ ❶❷❸❹ ,❏❏ ❶❷❸❹ "❶❷❸❹ ❶❷❸❹ "❶❷❸❹ ❶❷❸❹ ❶❷❸❹ ❶❷❸❹ ❶❷❸❹  
❶❷❸❹ .❶❷❸❹ ❶❷❸❹ ❶❷❸❹ (❶❷ : old\_IFS=\$IFS ❶❷❸❹ old\_IFS="\$IFS").

# 11. Variables - Part III

$$4 \times (10^6 - 1) \times 10^6 = 999600000$$

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

[illegible]

???

|||||    ||    |||||    |||||    |||||    |||||    ||    ||    ||    (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" # prints -en (bash & csh do ). Dash, Bourne  
# prints , \c . Ksh does . "RETURN"  
# prints :  

```

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

...    :

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

[illegible]

```
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 사용자 입력을 myname 변수에 저장합니다. 마지막으로 echo 명령을 사용하여 "Your name is : \${myname:-`whoami`}"라는 메시지를 출력합니다. 여기서 \${myname:-`whoami`}는 myname 변수가 설정된 경우 그 값을 사용하고, 그렇지 않으면 `whoami` 명령의 출력을 사용합니다.

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

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:-John Doe}"라는 메시지를 출력합니다. 여기서 \${myname:-John Doe}는 myname 변수가 설정된 경우 그 값을 사용하고, 그렇지 않으면 John Doe라는 값을 사용합니다. 이 코드는 `whoami` 명령을 사용하여 현재 사용자의 이름을 확인하고, 그 값을 myname 변수에 저장합니다. cd 명령을 사용하여 현재 디렉토리를 변경합니다.

## Using and Setting Default Values

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:=John Doe}"라는 메시지를 출력합니다. 여기서 \${myname:=John Doe}는 myname 변수가 설정된 경우 그 값을 사용하고, 그렇지 않으면 John Doe라는 값을 설정합니다.

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

이 코드는 echo 명령을 사용하여 "Your name is : \${myname:=John Doe}"라는 메시지를 출력합니다. 여기서 \${myname:=John Doe}는 myname 변수가 설정된 경우 그 값을 사용하고, 그렇지 않으면 John Doe라는 값을 설정합니다. 이 코드는 `whoami` 명령을 사용하여 현재 사용자의 이름을 확인하고, 그 값을 myname 변수에 저장합니다. cd 명령을 사용하여 현재 디렉토리를 변경합니다.

# 12. External Programs

□ □□□ □□ □ □□□ □ □□□ , □ □ □ □ □ □ (echo, which, test □ □□□ □ □ □ □) □ □□ □□ □□ □□ **tr, grep, expr, cut** □ □□ □□□ □□□ .

□□ ( ` ) □ □ □ □ □□ □□□ □□□ . □□ □□ □ □ □ □□□□□□□ . □□ □ □ □□ □□ □□ □□ □□ □ □□□ □ □□□ . □□ □□ □ □ □□□□ . □ □ □□ □ □ □□ /etc/passwd □ □ □ □ □□ :

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

□□ □ □□ □ □ □ □□ □ □ □ □ □ □ □□ :

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

□□ □□ □□ □□□ □□ □ □ □ □ □□ □ □ □ □ □□ □ □ □ □ □□ . □ □ □ □ □ □ □ □ □□ □□ □□ □□ □□ □□ □□ □ □ □ □ □□ □ □□ :

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

□ □□ □□□ □ □□ □ □ □ □ □□ , □ □ □□□ □□□ ! □ □ □ □□ □ :

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

□□ : □□ □ □ □ □ □ □ □□ □ □ **\$HTML\_FILES** □□ □□ □□□□ . □□ □□ **grep** □□ □ □ □ □ □ □ □ □ □□□ .

□□ □ □ □ □ □ □ □□ □□□ □ □ □ □ □ □□ □ □ □ □ □□ . □□□□ □ □ □ □ □ □□ □ □ □ □□□ .

□□□ □□ □ □□□□ 14 □ , □ □ □ □ □□ □□ □□□□ .





# 13. Functions

在 shell 中，函数（function）是可以在脚本或交互式 shell 中调用的可重用代码块。函数可以接受参数，执行一系列操作，并将结果返回给调用者。函数可以简化代码，提高可读性，并避免重复代码。在 shell 中，函数通常通过以下语法定义：

```
./library.sh
```

在 shell 中，函数通常通过以下语法定义：

在 shell 中，函数通常通过以下语法定义：

在 shell 中，函数通常通过以下语法定义：

- 函数名 函数体
- 函数名 函数体 函数名 函数体
- **return** 函数名 函数体 函数名 函数体
- **echo** 函数名 **stdout** 函数名 函数体 , 函数 c=`expr \$a + \$b` 函数 函数 函数

在 shell 中，函数通常通过以下语法定义：

在 shell 中，函数通常通过以下语法定义：

```
#!/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..."
```

???

\$@ )\$ \$\$\$\$\$\$ \$ \$\$\$ \$\$\$\$ \$\$\$\$ \$\$\$\$ . \$\$\$\$\$\$ \$\$\$\$ (\$1, \$2,  
\$@\$ )\$ \$\$\$\$\$\$ \$\$\$\$ \$\$\$\$ \$\$\$\$ . \$\$\$\$ \$\$\$\$ \$\$\$\$ \$\$\$\$\$\$ \$\$\$\$ \$\$\$\$\$\$ :

```
#!/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
```

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

```

out.log" 0 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 1039
```

```
#!/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"

```

\$ myfunc Hello World
 Hello World
 Goodbye Cruel World

## ??(Recursion)

A function that calls itself.
 .
 .
 .
 :

```

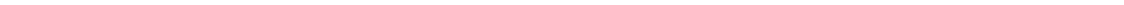
#!/bin/sh

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
```



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
```

```
function2.sh function3.sh
common.lib
.
.
```

## Exit Codes

```
(14)
.
```

```
#!/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
```

```
adduser bob letmein Bob Holness from Blockbusters
```

```
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
```

```
if [ $(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 32 | head -n 1 | tr -d '\n') = "root" ]; then
    echo "Root user added to the system."
else
    echo "Root user not added to the system."
fi
```



## 14. Hints and Tips

00 : 00 00 00 00 <https://www.shellscript.sh/tips> 00 00 0000 . 00 0000  
 00 00 00 00 00 00 . CGI 0000 00 00 0000 00 0000 .

在 Linux 中，我们通常使用 `ls` 命令来列出目录内容。但是，如果我们想要更详细的信息，比如文件的权限、所有者、大小等，我们可以使用 `ls -l`。此外，我们还可以使用 `du` 命令来查看目录的大小。

\*   "   "    , -  .

# CGI Scripting

```
CGI [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] . [REDACTED] CGI  
[REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] , [REDACTED] [REDACTED] [REDACTED] [REDACTED]  
CGI [REDACTED] [REDACTED] [REDACTED] [REDACTED] [REDACTED] , [REDACTED] [REDACTED] [REDACTED] CGI [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] . fortune.cgi [REDACTED] cookie.cgi
```

## Exit Codes

0 到 255 的整数，在 Unix 系统中，端口号的范围是 0 到 255，以及 1024 到 65535。

□ □□□ □□ □□ □□ □□□ □□ □□ □□ □□ □□ □□ □□ □  
 □□□ . □□ □□□ 10□ , "□□ - 2□ "□□ □□□ □□□□□ . □□□ □□ □□ □□ □□  
 □ □ □□ □□□□□ .

00 (Success) 00000 00 00 0000 , 00 (Failure) 00000 00 00 00 00  
 00000 . 00 00 000 000 000 00 0000 . 00 00 GNU grep 0000 00 ,  
 0000 000 000 10 , 00 00 (00 00 , 0000 00 00 00 00 ) 0000 20 00000 .

```
#!/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"
```

USERNAME :  
NAME :  
HOMEDIR :

```
#!/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 `NAME` und `HOMEDIR` auswertet und deren Werte ausgibt. Es ist ein einfaches Shell-Skript, das in einem Texteditor erstellt werden kann.

Das Skript wird mit dem Befehl `./script.sh` ausgeführt. Die Ausgabe zeigt die Werte der Variablen `NAME` und `HOMEDIR`.

```
#!/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 Umgebungsvariable `NAME` auswertet und deren Wert ausgibt. Es ist ein einfaches Skript, das in einem Texteditor erstellt werden kann.

Das Skript wird mit dem Befehl `./script.sh` ausgeführt. Die Ausgabe zeigt den Wert der Variable `NAME`.

```
#!/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."
```

[illegible]

```
#!/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
```

❏   ❏   ❏   echo❏   .

Success

❏

Failed

❏   cp❏   ❏❏❏   ❏❏❏❏   ❏   ❏❏❏   .❏   ❏❏   ❏❏   ❏❏   :

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

❏   ❏❏❏   ❏❏   ❏❏   ❏❏   ❏   ❏❏❏   .❏   ❏❏❏   ❏❏   ❏   /❏   ❏❏❏❏❏   ❏❏❏❏   ,❏  
❏   ❏❏❏   ❏❏   ❏❏❏❏   ❏   ❏   &&❏   ||❏   ❏   ❏❏   ❏❏❏❏   ❏   ❏❏❏   ❏❏❏   .❏  
❏   ❏❏❏   ❏   ❏❏❏❏   .❏❏❏   ❏   ❏❏❏   ❏❏   ❏   ❏❏❏❏   ❏❏❏❏   ❏   ❏❏❏   .

❏   ❏❏❏❏   cp❏❏❏   ❏   ❏   ❏   ❏❏   ❏   ❏❏❏❏   ❏❏❏❏   ❏   ❏❏❏   ❏❏❏   ❏   ❏❏❏  
❏❏❏   ❏   ❏❏❏❏   :

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

❏❏❏   ❏❏❏   Marcel❏   ❏❏❏   ❏❏❏❏   ❏❏❏❏   ❏   ❏❏❏❏❏❏   .❏❏❏❏   ❏❏❏   ❏❏❏  
❏❏❏❏   :

```
( 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
```

if, then, else  
.

## Simple Expect Replacement

expect  
Sun Microsystems Explorer  
.

expect.txt

```
S command E[delay] expected_text
```

"S"(Send  
"E"  
: "E10 \$"  
1  
MAX\_WAITS  
"E \$"

**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-`
```

```

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

```

$\square\square\square$        $\square\square\square\square$       :

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

```

root@kali:~# cat file1.txt
root@kali:~# ls -la /tmp/
total 12
drwxrwxrwt 1 root root 4096 Nov 11 12:00
-rw-r--r-- 1 root root    0 Nov 11 12:00
-rw-r--r-- 1 root root    0 Nov 11 12:00
-rw-r--r-- 1 root root    0 Nov 11 12:00

```

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

```
Connected to 127.0.0.1.  
Escape character is '^]'.
```

```
Escape character is '^['.
```

```
declan login: steve
Password:
```

```

Password:
Last login: Thu May 30 23:52:50 +0100 2002 on pts/3 from localhost.

```

```
Last login: Thu May 30 23:52:50 +0100 2002 on pts/3 from localhost.  
No mail.  
steve:~$ ls /tmp
```

```
No mail.  
steve:~$ ls /tmp
```

```
steve:~$ ls /tmp
API.txt          cgihtml-1.69.tar.gz  orbit-root
```

API.txt	cghtml-1.69.tar.gz	orbit-root
cal		

cal  
a.txt cmd.txt orbit-steve

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  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
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
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
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

```

```
#!/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/F00/BAR/g $i > /tmp/temp_${i}.$$ && mv /tmp/temp_${i}.$$ $i
```



done

trap '' 0 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)



fi

```
read name
```

```
echo -n   |   |   -n|   |   echo    ,  $n  |   |  
$c \c |   .|   |  |   |   |   $n -n|   |   $c  |   |  
      .
```

```
grep  1  1111  1111111  11  111  1111111  . grep  1  11  111  1111 :
```

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

```

root@kali:~# cat /etc/passwd | grep steve
steve:x:1000:1000::/home/steve:/bin/bash

```

```
$> 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
```

```

[[ [[ [[ [[ NEWLINE [[ [[ [[ [[ . sh [[ [[ [[ [[ $IFS [[ [[ [[
[[ [[ [[ [[ [[ [[ [[ [[ [[ . IFS [[ [[ [[ [[ <space><tab><cr>[[ [[ [[ [[
NEWLINE [[ [[ [[ [[ [[ [[ [[ [[ [[ : [[ [[ NEWLINEs.... [[ [[ [[ [[ [[ [[ [[ [[
[[ [[ [[ [[ [[ :

```

```
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) 的字符串 的 字符串 . tr 的 的 的 的 的 的 的 的 . 的 的 的 的 的 的 的 的 的 的 :

```
#!/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]'
```

把 [a-z] 的 [A-Z] 的 的 的 . a-z 的 A-Z 的 的 的 的 的 的 的 的 .  
的 ASCII 的 a-z 的 的 的 A-Z 的 的 的 . 的 , 的 的 的  
的 . tr 的 的 的 的 . tr [:lower:] [:upper:] 的 的 的 的 的 的  
的 . 的 的 的 的 tr 的 的 的 的 的 的 .

## Cheating

的 的 的 的

的 的 的 的 ! 的 的 的 的 的 . 的 的 的 的 sed 的 awk 的 .  
的 的 的 的 的 的 的 的 的 的 的 的 的 的 的 的 , 的  
的 的 的 的 的 的 的 的 的 的 .

的 的 的 的 (sed 的 52k, awk 的 110k) 的 的 的 的 ,  
的 的 的 , 的 的 的 的 的 的 的 的 的 的 的 的  
的 的 . 的 的 的 的 的 的 的 .

## Cheating with awk

的 的 的 的 , 的 , 的 的 的 wc 的 的 的 . 的 的 的 :

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

的 的 的 的 的 的 的 :

```
NO_LINES=`wc -l file`
```

的 的 的 的 的 的 . 的 的 的 的 102 的  
的 的 的 . 的 , awk 的 C 的 scanf 的 的 的 的 的

이제 `NO_LINES` 변수를 사용하여 `$1 $2 $3`의 값을 출력하는 프로그램을 작성해 보겠습니다.

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

이제 `NO_LINES` 변수의 값을 102로 설정해 보겠습니다.

## Cheating with sed

이제 `sed` (stream editor)을 사용하여 `Perl`의 코드를 `sed`로 대체하는 프로그램을 작성해 보겠습니다. `sed`의 기본 사용법은 `s/from/to/g`입니다.

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

이제 `1`번 줄의 `eth0`을 `eth1`로 대체하는 프로그램을 작성해 보겠습니다. `tr` (translate) 명령어를 사용하여 문자열을 대체하는 프로그램도 작성해 보겠습니다.

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

이제 `grep`를 사용하여 `"bad word"`를 찾는 프로그램을 작성해 보겠습니다. `grep`의 기본 사용법은 `grep pattern file`입니다. `grep`의 옵션으로는 `-v` (invert match)가 있습니다.

```
This line is okay.  
This line contains a bad word. Treat with care.  
This line is fine, too.
```

`grep`를 사용하여 `bad word`를 찾는 프로그램을 작성해 보겠습니다. `sed`를 사용하여 `bad word`를 대체하는 프로그램도 작성해 보겠습니다.

```
This line is okay.  
This line contains a . Treat with care.  
This line is fine, too.
```

## Telnet hint

이제 `Sun Explorer`를 사용하여 `telnet`을 실행하는 프로그램을 작성해 보겠습니다. `telnet`의 기본 사용법은 `telnet host port`입니다.

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

이제 `grep`를 사용하여 `bad word`를 찾는 프로그램을 작성해 보겠습니다. `grep`의 옵션으로는 `-v` (invert match)가 있습니다.

编译选项 -q 编译选项 GNU grep 编译选项 grep 编译选项 /dev/null 编译选项  
编译选项 . 编译选项 编译选项 编译选项 .

```
#!/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
```

编译选项 Sun 编译选项 编译选项 编译选项 编译选项 (编译选项 编译选项 编译选项 编译选项 编译选项 编译选项  
编译选项 编译选项 编译选项 编译选项 编译选项 编译选项 编译选项 编译选项):

```
$ ./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

if [ "${tout}" -ne "-5" ]; then
```

```
    exit 1
fi

> ${file}

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

0 00000 000 file1 0000 , 0 000 000 000000 00 000 0000 0 00000 .  
"> \${file}" 0 0000 000 0000 000 000 00000 00 000 000 000 00000 .



# 15. Quick Reference

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

Metacharacter	Function	Example
&	Execute the following command in the background.	ls &
&&	Execute the following command only if the previous command succeeded (AND).	if [ "\$foo" -ge "0" ] && [ "\$foo" -le "9" ]
	Execute the following command only if the previous command failed (OR).	if [ "\$foo" -lt "0" ]    [ "\$foo" -gt "9" ] (not in Bourne shell)
^	Match the beginning of a line.	grep "^foo"
\$	Match the end of a line.	grep "foo\$"
=	Test if two strings are equal (cf. -eq).	if [ "\$foo" = "bar" ]
!	Test if a string is not equal (NOT).	if [ "\$foo" != "bar" ]
\$\$	Expand to the shell's PID.	echo "my PID = \$\$"
\$_	Expand to the PID of the last command executed.	ls & echo "PID of ls = \$_"
\$?	Expand to the exit status of the last command executed.	ls ;
	Expand to the exit status of the last command executed.	echo "ls returned code \$?"
\$0	Expand to the name of the shell script being executed.	echo "I am \$0"
\$1	Expand to the first argument of the shell script.	echo "My first argument is \$1"
\$9	Expand to the ninth argument of the shell script.	echo "My ninth argument is \$9"
\$@	Expand to all arguments of the shell script.	echo "My arguments are \$@"
\$*	Expand to all arguments of the shell script.	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/lis ]	if [ -f /bin/lis ]
-r	if [ -r /bin/lis ]	if [ -r /bin/lis ]
-w	if [ -w /bin/lis ]	if [ -w /bin/lis ]
-x	if [ -x /bin/lis ]	if [ -x /bin/lis ]
function myfunc() { echo hello }	function myfunc() { echo hello }	function myfunc() { echo hello }

## 16. Interactive Shell

UNIX Linux `uname -a` . `cat /etc/passwd`  
 \*nix `cat /etc/passwd` bash  
 . `cat /bin/sh` .

**bash**

```
bash$ cd /usr/bin; ls -l | grep mv
-rwxr-xr-x 1 root root 108672 Aug 19 2014 mv
mv: cannot move './mv' to './mv': Device or resource busy
```









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

```

0000 0000 0000 PageUp 0 PageDn 0000 0000 0000 0000 0
0000.

```

**ksh**

```
vi emacs 1000 1000 1000 ksh 1 1000 1000 1 1000. 1
1000 1000 1000 1000 1000 1000 1000 1000. set -o vi, ksh -o vi 1000
exec ksh -o vi(1000 1000 1000 1000 "vi" "emacs" 1000 1000 1000).
```

□□ □□□ □□ ksh □□□ □□□□ □□□ □□ ksh□ □□□□ □□□:

```
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%
```

但是 ksh 的别名, 在 shell 中 是 不 能 用 的。 所以  
我们 使用 csh(或 sh) 的 ksh 的 别名:

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

我们 使用 csh 的 别名 是 不 能 用 的。

我们 使用:

```
csh% 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 是 在 k 的 前面 加上 一个 字母。 我们 使用 vi 的  
我们 使用 的 别名 是 不 能 用 的:

```
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
```