IIC Systèmes Informatiques

Stéphane Genaud

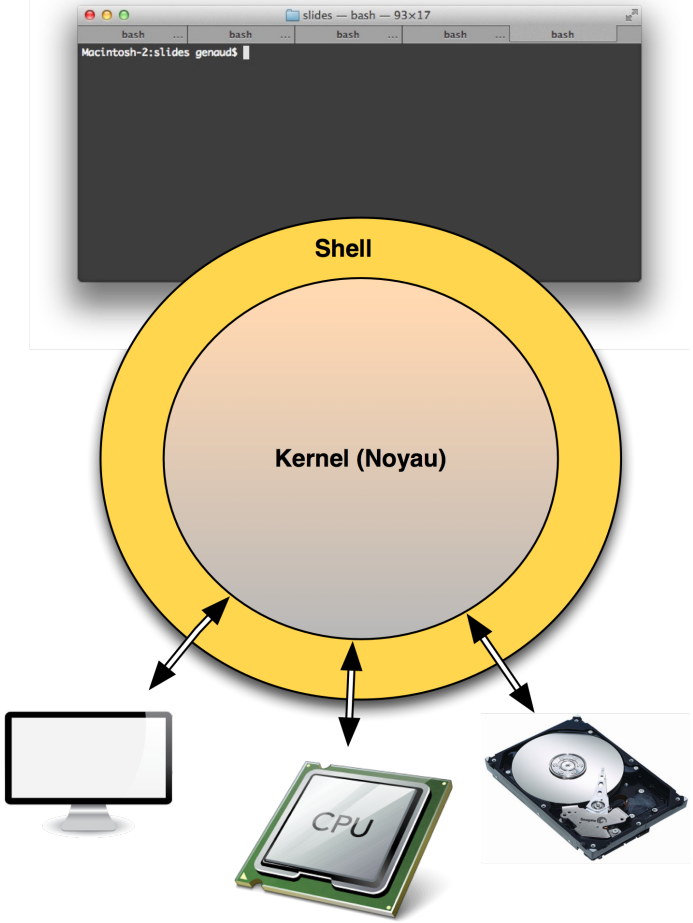
September 9, 2015

Outline

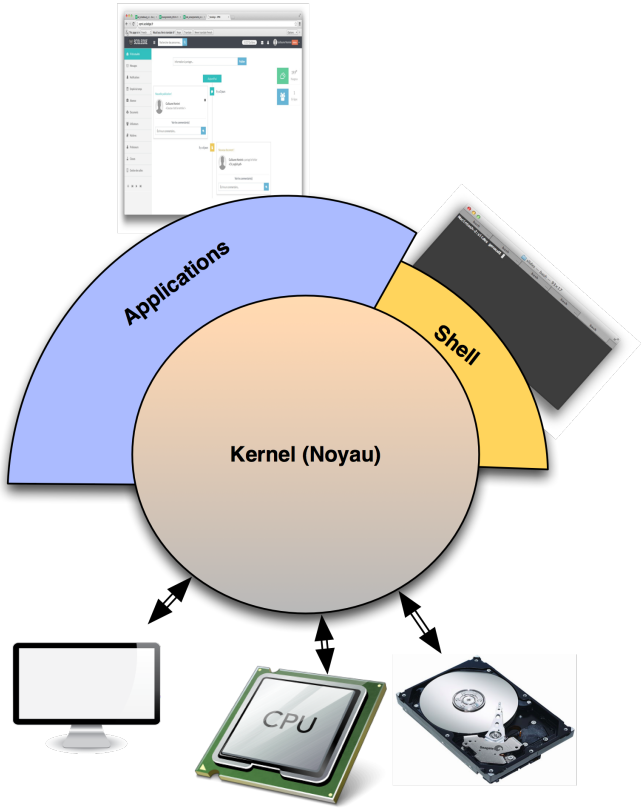
Table of Contents

Acces via a Terminal

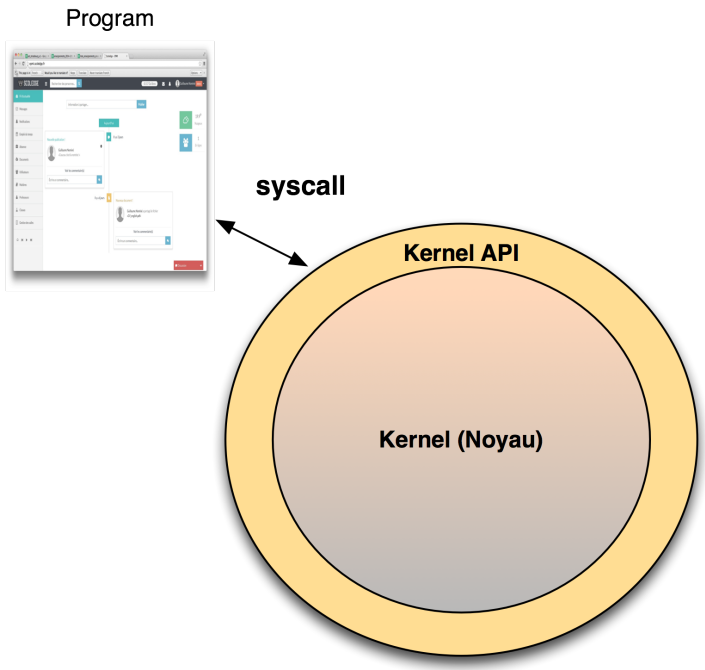
A Shell around the Kernel



Enriched Environment

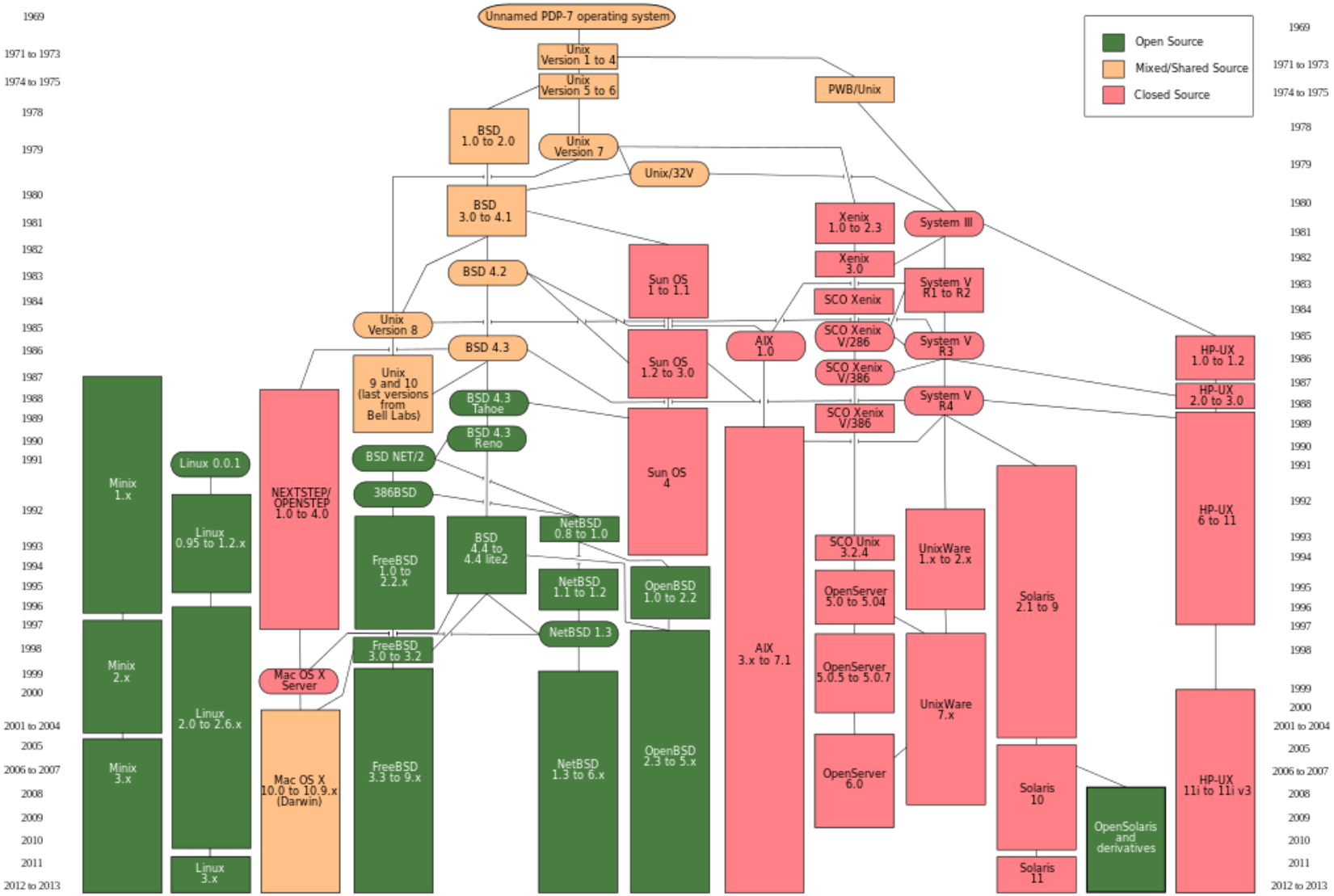


Kernel API (Syscalls)




```
% man syscalls(2)
```

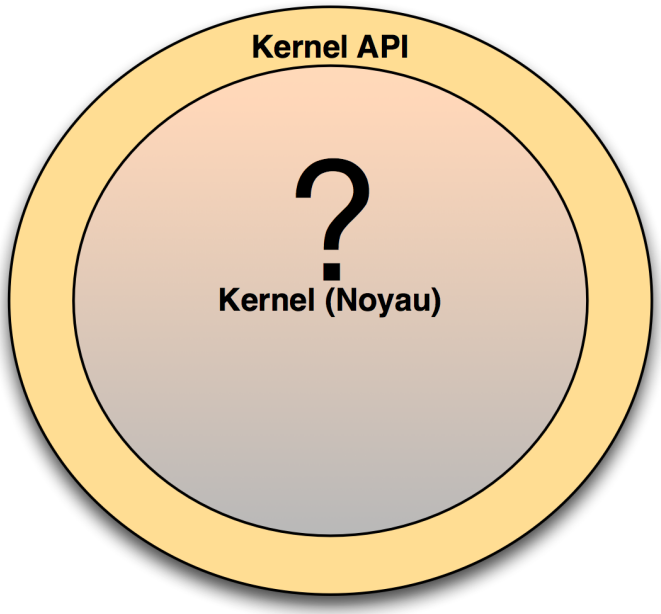
Unix


Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2013
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
PWB/Unix
BSD 1.0 to 2.0
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
Xenix 3.0
System III
BSD 4.2
Sun OS 1 to 1.1
SCO Xenix
System V R1 to R2
Unix Version 8
BSD 4.3
Sun OS 1.2 to 3.0
AIX 1.0
SCO Xenix V/286
System V R3
HP-UX 1.0 to 1.2
Unix 9 and 10 (last versions from Bell Labs)
SCO Xenix V/386
HP-UX 2.0 to 3.0
BSD 4.3 Tahoe
System V R4
BSD 4.3 Reno
SCO Xenix V/386
Minix 1.x
Linux 0.0.1
NEXTSTEP/ OPENSTEP 1.0 to 4.0
BSD NET/2
Sun OS 4
386BSD
HP-UX 6 to 11
NetBSD 0.8 to 1.0
SCO Unix 3.2.4
UnixWare 1.x to 2.x
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4 to 4.4 lite2
OpenBSD 1.0 to 2.2
NetBSD 1.1 to 1.2
OpenServer 5.0 to 5.04
Solaris 2.1 to 9
AIX 3.x to 7.1
Minix 2.x
FreeBSD 3.0 to 3.2
NetBSD 1.3
OpenServer 5.0.5 to 5.0.7
UnixWare 7.x
Mac OS X Server
Linux 2.0 to 2.6.x
Minix 3.x
Mac OS X 10.0 to 10.9.x (Darwin)
FreeBSD 3.3 to 9.x
NetBSD 1.3 to 6.x
OpenBSD 2.3 to 5.x
OpenServer 6.0
Solaris 10
OpenSolaris and derivatives
HP-UX 11i to 11i v3
Linux 3.x
Solaris 11

Kernel Roles

Roles

- Program Execution
- File Management
- I/O Management
- Communication
- Protection & Security
- Accounting
- Resource Allocation



Program Execution

fig/terminal-run-program.gif

- Spawn Processes
- Allocate Resources (RAM, CPU)
- Multi-tasking $\Rightarrow$ Scheduler

File Management

I/O Management

Communication

Table of Contents

Shell Families

- Unix has forked into a variety of implementations . . .
- . . . and so does its shell
- sh, ksh, zsh, bash, . . .
- csh, tcsh

Warning

`-- Programming in Bourne-Shell is a higher form of masochism.`
(Fortune)

File Management

`ls`	list files	*ls [options] [files]*
`cd`	change directory	*cd [file]*
`pwd`	current working directory	*pwd*
`mv`	move file	*mv [options] orig dest*
`cp`	copy file	*cp [options] orig dest*
`rm`	remove file	*rm [optons] file*
`mkdir`	create directory	*mkdir file*
`rmdir`	remove directory	*rmdir file*

Example File Management

```
% ls -l
total 3
drwxr-xr-x 2 genaud staff 544 16 jul 2013 CRs
drwxr-xr-x 2 genaud staff 136 23 nov 2012 conventions
-rw-r--r-- 1 genaud staff 3361 12 sep 16:30 effectifs.org
% cd CRs
% ls
CR-codir-20121119.org
% mkdir /home/alex/CRs
% cp CR-codir-20121119.org /home/alex/CRs
% mkdir old_conventions
% cp -r ../conventions old_conventions/
```

File Management: permissions

```
drwxr-xr-x 2 genaud staff 544 16 jul directory
-rw-r--r-- 1 genaud staff 361 12 sep non_exe_file
-rwxr-xr-x 1 genaud staff 987 12 sep exe_file
```

- Permissions: 3 categories : user | group | other
- ... for each category : read (r) | write (w) | execute (x)
- Ownership: user and group

chmod	change permissions
chown	change owner
chgrp	change group

Useful Filters (1)

`cat`	print file content
`more`	per page display
`head`	print first lines of file
`tail`	print last lines of a file
`cut`	extract columns of a file

Example cat, cut

```
$ cat a.txt
2013 TeamA 134423
2013 Rockx 120018
2014 Calyp 119741
2014 Vorta 99203

$ cut -c12- a.txt
134423
120018
119741
 99203
```

Useful Filters (2)

`wc`	count chars, words, lines of a file
`sort`	sort lines
`uniq`	filter out successive identical lines
`grep`	extract lines that match pattern
`paste`	merge files in a column-orientated way

Useful Filters (3)

`bc`	compute numeric expression
`rev`	reverse lines (reverse the order of characters)
`tr`	substitute one character by another
`nl`	number lines

Process Management

`ps`	list processes
`top`	interactive listing of processes
`kill`	send a signal to a process
`jobs`	display jobs in the current session
`fg`	make the job run in the foreground
`bg`	make the job run in the background

Environment

- Each process has its own environment defined through variables, e.g

`PWD`	current working directory
`HOME`	home directory for the user
`USER`	user login
`PATH`	list of directories where executables can be found
`PS1, PS2`	prompts

- Displayed with `env`
- New user-defined variables are added to the environment

Variables

Ordinary Variables

- Assign : variable=value (NO SPACE !)
- Get : $variable

```
$ nom=John
$ adresse="rue␣des␣Pommiers"
$ ville="Strasbourg"
$ echo "$nom␣habite␣$addresse,␣$ville"
John habite rue des Pommiers, Strasbourg
```

Variables: Scope

- Variables are only known in the process environment
- Variables can be exported to child processes

```
$ a=10 # not exported
$ /bin/bash
$ echo $a

$ export a=10 # exported
$ /bin/bash
$ echo $a
10
```

Variables: Special characters

- Used for filenames, the shell offers generic characters

*	expand to all filename in CWD
?	match any one character
[ ]	expand to a range
{ }	expand to an enumeration

Variable Expansion

- Variables (names starting with $) are automatically expanded
- The white-space is a separator $\Rightarrow$ need to pack values with quotes
- Automatically expanded when in double quotes, not with simple quotes

```
$ a="John"
$ echo "I␣am␣$a"
I am John
$ echo 'I am $a'
I am $a
```

Evaluation

- Request the result of an expression (generally assigned afterwards).
- Mark the expression to evaluate with back-quotes

Example: assume `date` is a command.

```
$ a=date # not evaluated
$ echo $a
date # not what we want ...
$ a=‘date‘ # evaluated
$ echo $a
Lun 15 sep 2014 16:14:00 CEST
```

Evaluation (2)

- Evaluation is often required to store intermediate results
- Examples:

```
$ number_of_files=‘ls -1 | wc -l‘
$ welcome_msg="Bienvenue.␣Nous␣sommes␣le␣‘date‘"
```

Evaluation of arithemic expressions

The old way

- `expr` : variable=value

```
$ a=10
$ b=2
$ expr $a + $b
12
$ c=‘expr $a + $b‘
$ echo $c
12
```

- Mind the spaces ! `expr` has 3 space-separated arguments.
- The external command `expr` is limited in many regards.

Evaluation of arithemic expressions (2)

The modern way

- `$(( ))` or `let` :

```
$ i=$(( $i+1 ))
$ let i=$i+1 # equivalent to previous line
```

- The syntax is less rigid with spaces : just do not leave space between `i` and =

Input/output

echo	display a string to screen
read	read from keyboard

```
$ read n
42
$ echo "Number␣read␣is␣$n"
Number read is 42
```

Standard input/output/error

Unix defines 3 standard destinations

- 0: standard input (stdin), default : keyboard
- 1: standard output (stdout), default : screen
- 2: standard error (stderr), default : screen

Input/output: redirections

Redirection operators

- > : output
- » : ouput (concatenation mode)
- < : input
- | : bidirectional (pipe)

```
$ ls > f1
$ ls /home/alex >> f1
$ cat < f1 | sort > f2
```

Filter : definition

Definition

A filter is a command that takes its input on standard input and outputs its results on standard output

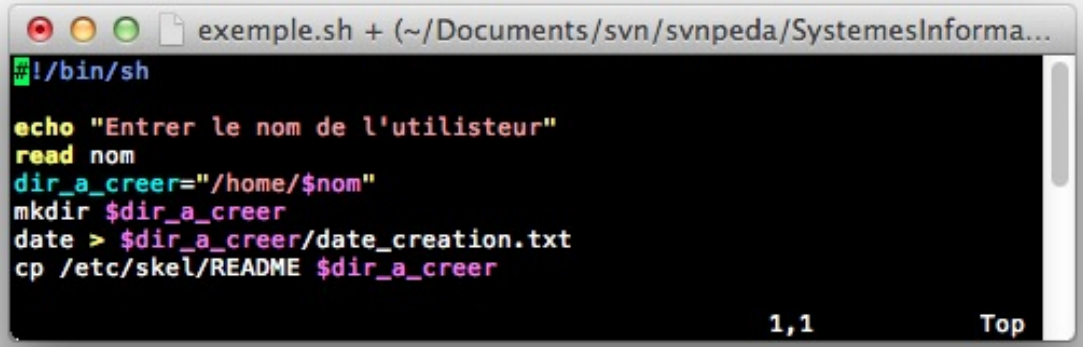
Conclusion

Filters generally have very focused roles but their combination is awesome.

Table of Contents

Batchs

- Objective: automate a chain of shell commands
- Put them in a file, execute it
- *Advice*: name the file `.sh` and make it `chmod +x`




```
$ ./exemple.sh
```

Positional Parameters

- Parameters may be passed when executing a script

```
$ ./exemple.sh hello 4
```

- hello is in variable $1, 4 in variable $2

```
#!/bin/bash
echo "Argument 1 is $1"
echo "Argument 2 is $2"
```

Positional Parameters (2)

`$#`	number of parameters
`$0`	script name
`$1-$9`	parameters by position entered
`$@`	list all parameters used (on separate lines)
`$*`	list all parameters used (on one line separated by spaces)
`$?`	return code of last command

Positional Parameters (3)

- Positional parameters can be identified from $0 to $9

```
#!/bin/bash
echo "Argument 1 is $1"
shift
echo "Argument 1 is $1"
shift 2
echo "Argument 1 is $1"
```

- `shift` shifts the numbering of arguments

```
$ ./myshift.sh arg1 arg2 arg3 arg4
Argument 1 is arg1
Argument 1 is arg2
Argument 1 is arg4
```

Control Structures: Boolean expressions

Boolean Expression

- `test` or `[` : unary or binary operator
- Example of unary operators

`[ -f file ]`	true if file exists and is a regular file
`[ -d file ]`	true if file exists and is a directory
`[ -n string ]`	true if length of string is non-zero
`[ -z string ]`	true if length of string is zero

- Negation is : `! expression`

Control Structures: Boolean expressions (2)

Boolean Expression

- `test` or `[` : unary or binary operator
- Example of binary operators

`[ s1 = s2 ]`	true if strings s1 and s2 are identical
`[ s1 != s2 ]`	true if strings s1 and s2 are not identical
`[ s1 < s2 ]`	true if string s1 is lexicographically less than s2
`[ n1 -eq n2 ]`	true if integers n1 and n2 are equal
`[ n1 -lt n2 ]`	true if integer n1 is lower than integer n2

Other operators on numerics : `-ne`, `-gt`, `-le`

Control Structures: If

If

```
if bool_expr
   then block_action1
   else block_action2
fi
```

- if, then, else, fi must be on distinct lines or separated by ;

```
if [ $var = "ok" ]
then
   rm datafile
   echo "Done."
else
   echo "Action␣canceled"
fi
```

Control Structures: Case

Case

```
case var in
   pattern_1) block_action1;;
   ...
   pattern_n) block_actionn;;
   *) other_actions;;
esac
```

- Mind the ;; after each block

```
case $var in
  ok) rm datafile
      echo "Done.";;
  *) echo "Action canceled";;
esac
```

Control Structures: For

For

```
for var in val1 val2 ... valn
do
   block_action
done
```

```
for i in 2013 2014 2015
do
    datfile="year-$i"
    grep "total" $datfile >> summary
done
```

Control Structures: While

While

```
while bool_expr
do
   block_action
done
```

```
while [ "$choice" != "quit" ]
do
    echo "Run␣once␣again␣..."
    read choice
done
```