BASH

THE BOURNE-AGAIN SHELL

Introduction

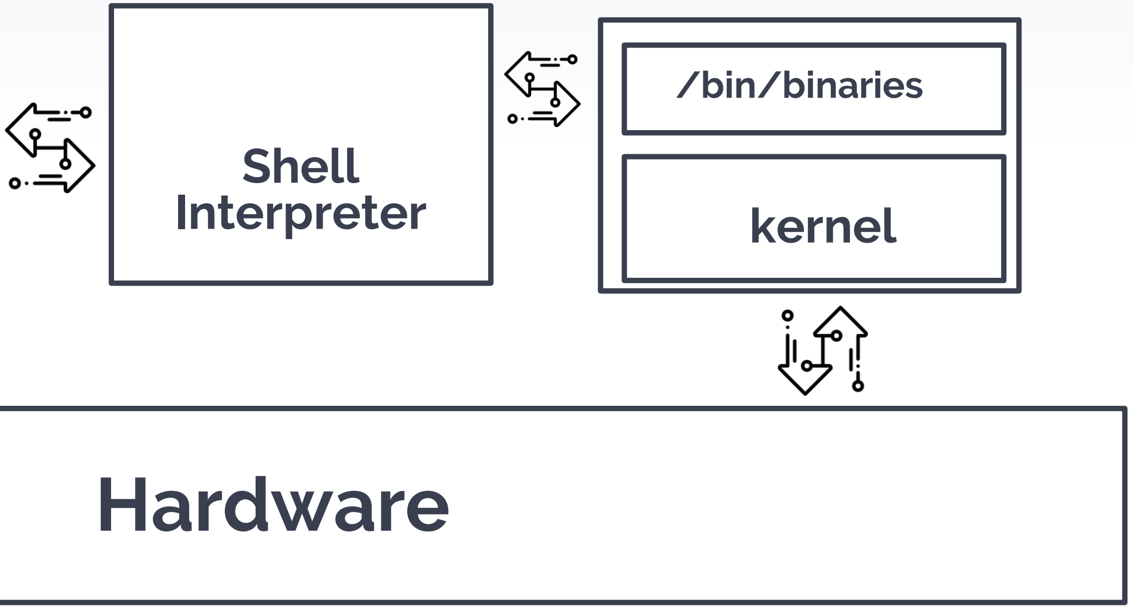
1

Basic commands

Basic command execution

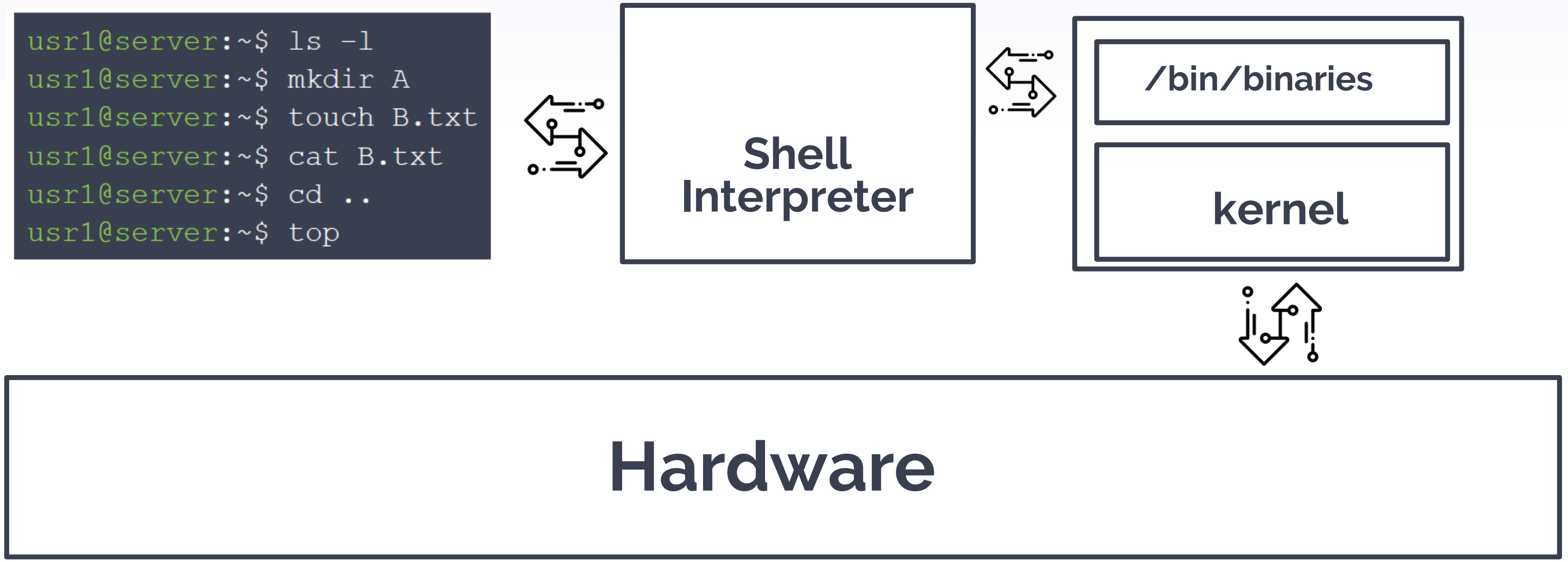
Shell interpreter

Kernel



Shell interpreter

Commands and args from terminal



```
usr1@server:~$ ls -l
usr1@server:~$ mkdir A
usr1@server:~$ touch B.txt
usr1@server:~$ cat B.txt
usr1@server:~$ cd ..
usr1@server:~$ top
```

Shell interpreter

Commands and args from terminal

```
username@hostname:~$ ls -l
username@hostname:~$ mkdir testDir
```

```
# terminal reads command and execute them when enter key is pressed
# commands are sent to shell interpreter who:

# 1) read 1st string and then
# 2) check if it is an alias and runs it in that case
# 3) if not looks for a binary exe file in /bin directory and runs it
# 4) parse all other strings as command arguments
```

Syntax
Comments

```
# this is a comment on one line (recommended)
# bash doesn’t really support multiline comment
# but there are some workaround: 1)

: '
this is a multiline comment
opens with colon, one space, and a singlequote
closes with single quote
'


<< 'A-MULTILINE-COMMENT'
    Everything inside the HereDoc body is
    a multiline comment
A-MULTILINE-COMMENT
```

Syntax
Variables

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Special variables

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Single and Double quote

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Immutable variables

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Shell expansion Arithmetics

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Shell expansion Command substitution

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Syntax

Shell expansion Process substitution

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

1

Input/Output

Read and Write data

Write on Standard Output

echo command

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Read from terminal

read command

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Script arguments

read script args

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Script Exit

Script execution exit code

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Redirection to file

write data to file

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Redirection to file

Read data from file

```
# read input from terminal into variable
#  else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

1

Conditional flow

Implement decisions

Conditional If

if [test-command] then

```
# if [ condition ] then <command>
#   else <command>
# fi




PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task..."
fi
```

Conditional If

if [test-command] then .. else

```
# if [ condition ]
#   then <command>
#   else <command>
# fi



PROCEED=YES
if [ "$PROCEED" = "YES" ]
then
    echo "Performing task... "
else
    echo "Task canceled ..."
fi
```

Conditional If

if [test-command] then .. elif.. else

```
# if [ condition ] then <command>
#   elif [ condition ] <command>
#   else <command>
# fi

VALUE=-10
if [ "$VALUE" -lt 0 ]; then
    echo "VALUE is less than 0"
elif [ "$VALUE" -eq 0 ]; then
    echo "VALUE is 0"
else
    echo "VALUE is greater than 0"
fi
```

Conditional If

nested conditions Syntax

```
# if [ test ]; then
#    <command>
# elif [ test ]; then
#        if [ test ]; then
#         <command>
#       else
#         <command>
#        fi
#else
#  <command>
#fi
```

Conditional If

nested conditions Example

```
if [[ $UID  -eq  0 ]] # If the user is root, its UID is zero.
then
  echo "You are root!"
elif [[ $UID  -eq 1002 ]]
then
  echo "You are user, welcome!"
else
  echo "You are not welcome here."
  exit 1;
fi
```

1

Loops

Repeated blocks of code

Loops

for each (enumeration of values)

```
# for <variable> in <list-of-elements>
# do
#   <command>
# done


for i in 1 2 3 # (enumerated values threated as string)
do
   echo $i
done
```

Loops

for each (enumeration of values)

```
# for <variable> in $(expression-returning-an-array-of-values)
# do
#   <command>
# done


for VARIABLE in $(ls) # all files in this dir
do
    echo $VARIABLE
done
```

Loops

for each (enumeration of values)

```
# ATTENTION!!: Legacy method (Prior to Bash 3.2)
# for <variable> in <list-of-elements>
# do
#   <command>
# done


for i in $(seq 1 2 20) # (start=1, step 2, end=20)
do
   echo $i
done
```

Loops

for each $(command-to-evaluate)

```
# ATTENTION!!: Legacy method (Prior to Bash 3.2)
# for <variable> in <list-of-elements>
# do
#   <command>
# done


for i in $(seq 1 2 20) # (start=1, step 2, end=20)
do
   echo $i
done
```

Loops

for (c-like syntax)

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Loops

for var in range

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

1

Arrays

Repeated blocks of code

Array

Indexed Array explicit declaration

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Associative Array explicit declaration

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Write data to Array

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Read data from Array

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Loops and Arrays

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Operations: Extract by offset

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Operations: Search and Replace

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#   <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Operations: Add new element

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Array

Operations: Remove element

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#   <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

1

Functions

Repeated blocks of code

Functions

Definition (name, input, output)

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Functions

Export functions to other scripts

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```

Functions

Import functions from other scripts

```
# for (( <init-variable>; <condition> ; <step> ))
# do
#    <command>
# done


for (( i=0; i<10; i=i+1 ))
do
    echo $i
done
```