BPT3 Live Session

Agenda

- Explain BPT3 questions briefly
- Not discuss BPT3 solutions
- Go through relevant W4,W5,W6 topics
- Discuss tips and tricks

Prerequisites

- How to navigate a linux VM
- Know how to solve questions in VM
- Know, Understand, Solve W1, W2, W3, W4, W5*

Variables

```
4 #!/usr/bin/env bash
3 a=1
2 echo $a
1 ((a++))
5 echo $a
```

Declaration and increment

Variables

```
((a*=5))
echo $a
```

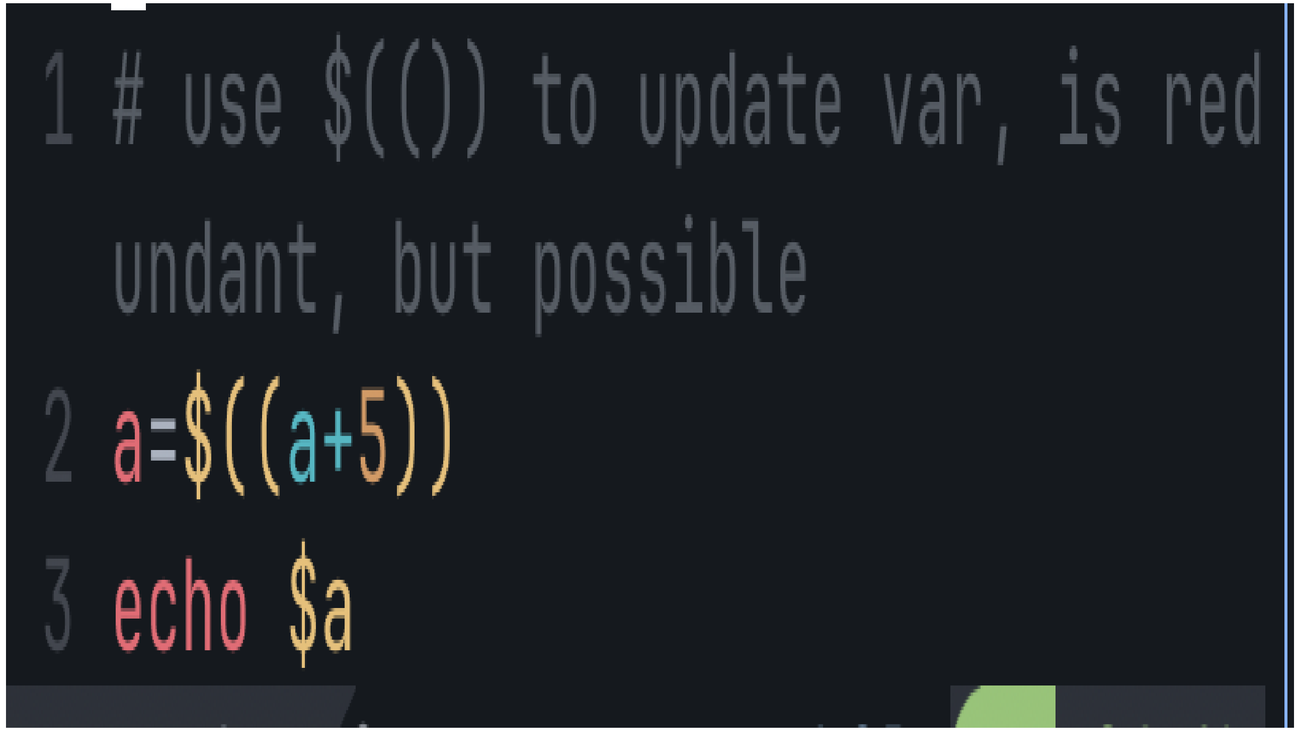
All normal mathematical operators work as expected

Variables

```
# get back eval result of math
echo $((a*2))
echo $a
```

We can get back the result of the eval using ‘$‘

Variables



Redundant usecases

read

- Take ONE line of input from user and store it in a variable.
- Can split the line into multiple tokens usign a delimiter.
- Delimiter is set using IFS variable.

read

```
4 #!/usr/bin/env bash
3
2 echo -n "Enter line: "
1 read -r line
5 echo "Line: $line" # always quote
  unknown valued variables, else spa
  ces will cause bugs
```

test

- Used to evaluate boolean expressions as exit code
- Used inside ‘if‘, ‘while‘, etc, but also standalone
- can test unary conditions (file, string)
- and binary conditions (file, string, numbers)

test

- ‘[‘ is same as ‘test‘
- ‘[[‘ is a bash keyword, it is more powerful
- ‘[[‘ can match regex (ERE) and glob patterns
- dont quote ERE inside ‘[[‘

Stop and Think

Why is ‘test 5 -gt 15‘ ‘false‘ but ‘test 5 ’>’ 15‘ ‘true‘?

[[glob matching

[[(and not test or [) can match globs

[[regex matching

[[can also match ERE

Stop and Think

- What is difference between ‘*‘ and ‘+‘ in ERE?
- Why does ‘hello world‘ match ‘x*‘?

negating test

- We can use ‘!‘ to negate the expression

Why use [[over [

- ‘[[‘ features are superset of ‘[‘
- It supports regex and glob matching
- It supports multiple conditions inside the same ‘[[‘
- It is a bash keyword, its faster than invoking a command
- It handles empty variables better

if conditions

- It is a conditional statement
- if runs any command, and checks its exit code
- based on it, we split the code into two branches
- if we want to check boolean expressions, use ‘test‘

if conditions

```
if ls hello &> /dev/null; then
    echo "hello exists"
else
    echo "hello does not exist"
fi
```

if conditions

```
if [[ -e hello ]]; then
  echo "hello exists"
else
  echo "hello does not exist"
fi
```

if conditions

```
word1="hello"
word2="hello123"

if [[ "$word1" =~ ^[a-z]+$ ]]; then
  echo "word1 is valid word"
fi

if [[ "$word2" =~ ^[a-z]+$ ]]; then
  echo "word2 is valid word"
fi
```

if conditions

```
❯ bash if.sh
hello does not exist
hello does not exist
word1 is valid word
```

while loop

- We can loop a block of code until a condition is met
- Like ‘if‘, ‘while‘ also executes a command and sees exit code
- To use expressions, use test

while and read

- we can pair while with read to read multiple lines
- read gives error code if no more lines to read
- read can be augmented with IFS

while and redirections

- like any other shell command, we can redirect input and output
- we can read a file instead of stdin using ‘<‘

while

```
i=5
while ((i < 10)); do
  echo -n "$i "
  ((i++))
done
```

while

```
i=5
while [[ $i -lt 10 ]]; do
   echo -n "$i "
   ((i++))
done
```

while

```
3
2 while [[ ! "$password" =~ ^[a-zA-
  Z0-9]{8,}$ ]]; do
1   read -rp 'Enter password: ' pas
  sword
3 done
```

while

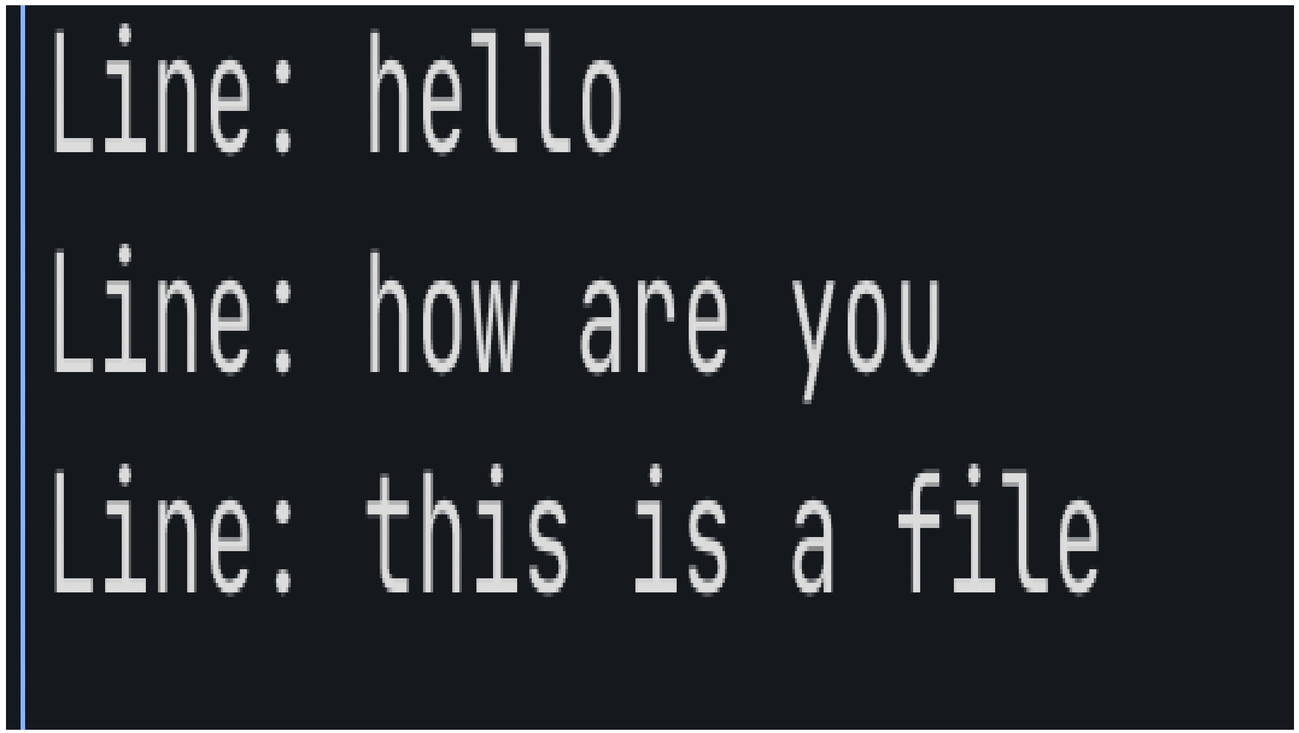
```
❯ bash while.sh
5 6 7 8 9
---
5 6 7 8 9
---
Enter password: hello
Enter password: helloworld123$
Enter password: hello12
Enter password: hello123
```

while

```
cat > data <<EOF
hello
how are you
this is a file
EOF

while read -r line; do
  echo "Line: $line"
done < data
```

while



multiples of 5, 2, and 4

- can you think of a string pattern of these multiples
- that let us regex match them?

factorial using paste and bc

- paste can be used as ‘tr ’\n’ "$delim"‘
- it takes care of last line
- bc is a calculator that evaluates expressions in stdin

find

- find lets you recurse filesystem hierarchy
- it can filter by name, type, size, etc
- it can perform actions on the found files
- learn more about find using ‘man find‘

Additional Resources

Navigating Linux Book

System Commands Channel with Additional Videos