Shell programming

More features in bash scripts

Debugging

```
set -x
./myscript.sh
```

Prints the command before executing it

Place the `set -x` inside the script

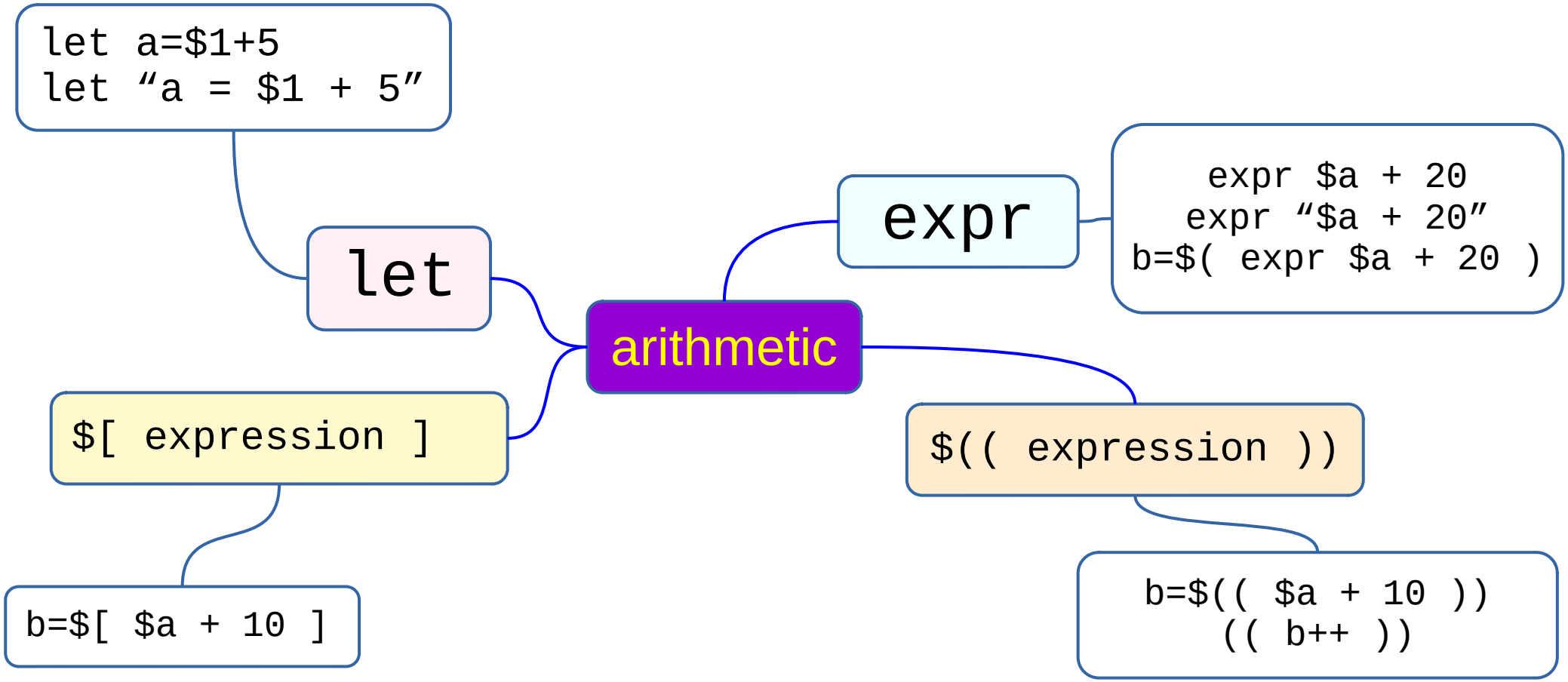
```
bash -x ./myscript.sh
```

Combining conditions

```
[ $a -gt 3 ] && [ $a -gt 7 ]
```

```
[ $a -lt 3 ] || [ $a -gt 7 ]
```

Shell arithmetic



expr command operators - 1

a + b	Return arithmetic sum of a and b
a - b	Return arithmetic difference of a and b
a * b	Return arithmetic product of a and b
a / b	Return arithmetic quotient of a divided by b
a % b	Return arithmetic remainder of a divided by b
a > b	Return 1 if a greater than b; else return 0
a >= b	Return 1 if a greater than or equal to b; else return 0
a < b	Return 1 if a less than b; else return 0
a <= b	Return 1 if a less than or equal to b; else return 0
a = b	Return 1 if a equals b; else return 0

expr command operators - 2

`a | b`	Return a if neither argument is null or 0; else return b
`a & b`	Return a if neither argument is null or 0; else return 0
`a != b`	Return 1 if a is not equal to b; else return 0
`str : reg`	Return the position upto anchored pattern match with BRE str
`match str reg`	Return the pattern match if reg matches pattern in str
`substr str n m`	Return the substring m chars in length starting at position n
`index str chars`	Return position in str where any one of chars is found; else return 0
`length str`	Return numeric length of string str
`+ token`	Interpret token as string even if its a keyword
`(exprn)`	Return the value of expression exprn

heredoc feature

```
a=2.5
b=3.2
c=4
d=$(bc -l << EOF
scale = 5
($a+$b)^$c
EOF
)
echo $d
```

```
1055.6001
```

Marker designation need not be EOF

```
a=2.5
b=3.2
c=4
d=$(bc -l <<- ABC
    scale = 5
    ($a+$b)^$c
    ABC
)
echo $d
```

A hyphen tells bash to ignore leading tabs

if-elif-else-fi loop

```
if condition1
then
    commandset1
else
    commandset2
fi
```

```
if condition1
then
    commandset1
elif condition2
then
    commandset2
elif condition3
then
    commandset3
else
    commandset4
fi
```

case statement options

```
case $var in
   op1)
      commandset1;;
   op2 | op3)
      commandset2;;
   op4 | op5 | op6)
      commandset3;;
   *)
      commandset4;;
esac
```

commandset4 is the default for values of `$var` not matching what are listed

c style for loop : one variable

```
begin=1
finish=10
for (( a = $begin; a < $finish; a++ ))
do
   echo $a
done
```

c style for loop : two variables

```
begin1=1
begin2=10
finish=10
for (( a=$begin1, b=$begin2; a < $finish; a++, b-- ))
do
   echo $a $b
done
```

Note: Only one condition to close the for loop

processing output of a loop

```
filename=tmp.$$
begin=1
finish=10
for (( a = $begin; a < $finish; a++ ))
do
   echo $a
done > $filename
```

Note: Output of the loop is redirected to the tmp file

break

```
n=10
i=0
while [ $i -lt $n ]
do
        echo $i
        (( i++ ))
        if [ $i -eq 5 ]
        then
                break
        fi
done
```

break out of inner loop

```
n=10
i=0
while [ $i -lt $n ]
do
        echo $i
        j=0
        while [ $j -le $i ]
        do
                printf "$j "
                (( j++ ))
                if [ $j -eq 7 ]
                then
                        break 2
                fi
        done
        (( i++ ))
done
```

break out of outer loop

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6
```

continue

```
n=9
i=0
while [ $i -lt $n ]
do
        printf "\n loop $i:"
        j=0
        (( i++ ))
        while [ $j -le $i ]
        do
                (( j++ ))
                if [ $j -gt 3 ] && [ $j -lt 6 ]
                then
                        continue
                fi
                printf "$j "
        done
done
```

```
loop 0:1 2
loop 1:1 2 3
loop 2:1 2 3
loop 3:1 2 3
loop 4:1 2 3 6
loop 5:1 2 3 6 7
loop 6:1 2 3 6 7 8
loop 7:1 2 3 6 7 8 9
loop 8:1 2 3 6 7 8 9 10
```

Continue will skip rest of the commands in the loop and goes to next iteration

shift

```
i=1
while [ -n "$1" ]
do
        echo argument $i is $1
        shift
        (( i++ ))
done
```

shift will shift the command line arguments by one to the left.

exec

```
exec ./my-executable --my-options --my-args
```

- To replace shell with a new program or to change i/o settings
- If new program is launched successfully, it will not return control to the shell
- If new program fails to launch, the shell continues

eval

```
eval my-arg
```

- Execute argument as a shell command
- Combines arguments into a single string
- Returns control to the shell with exit status

getopts

```
while getopts "ab:c:" options;
do
    case "${options}" in
        b)
            barg=${OPTARG}
            echo accepted: -b $barg
            ;;
        c)
            carg=${OPTARG}
            echo accepted: -c $carg
            ;;
        a)
            echo accepted: -a
            ;;
        *)
            echo Usage: -a -b barg -c carg
            ;;
    esac
done
```

This script can be invoked with only three options: a, b, c. The options b and c will take arguments.

select loop

```
echo select a middle one
select i in {1..10}
do
        case $i in
                1 | 2 | 3)
                        echo you picked a small one;;
                8 | 9 | 10)
                        echo you picked a big one;;
                4 | 5 | 6 | 7)
                        echo you picked the right one
                        break;;
        esac
done
echo selection completed with $i
```

Text menu !

You have seen most of the features needed to create a professional bash script

Explore by trying out !