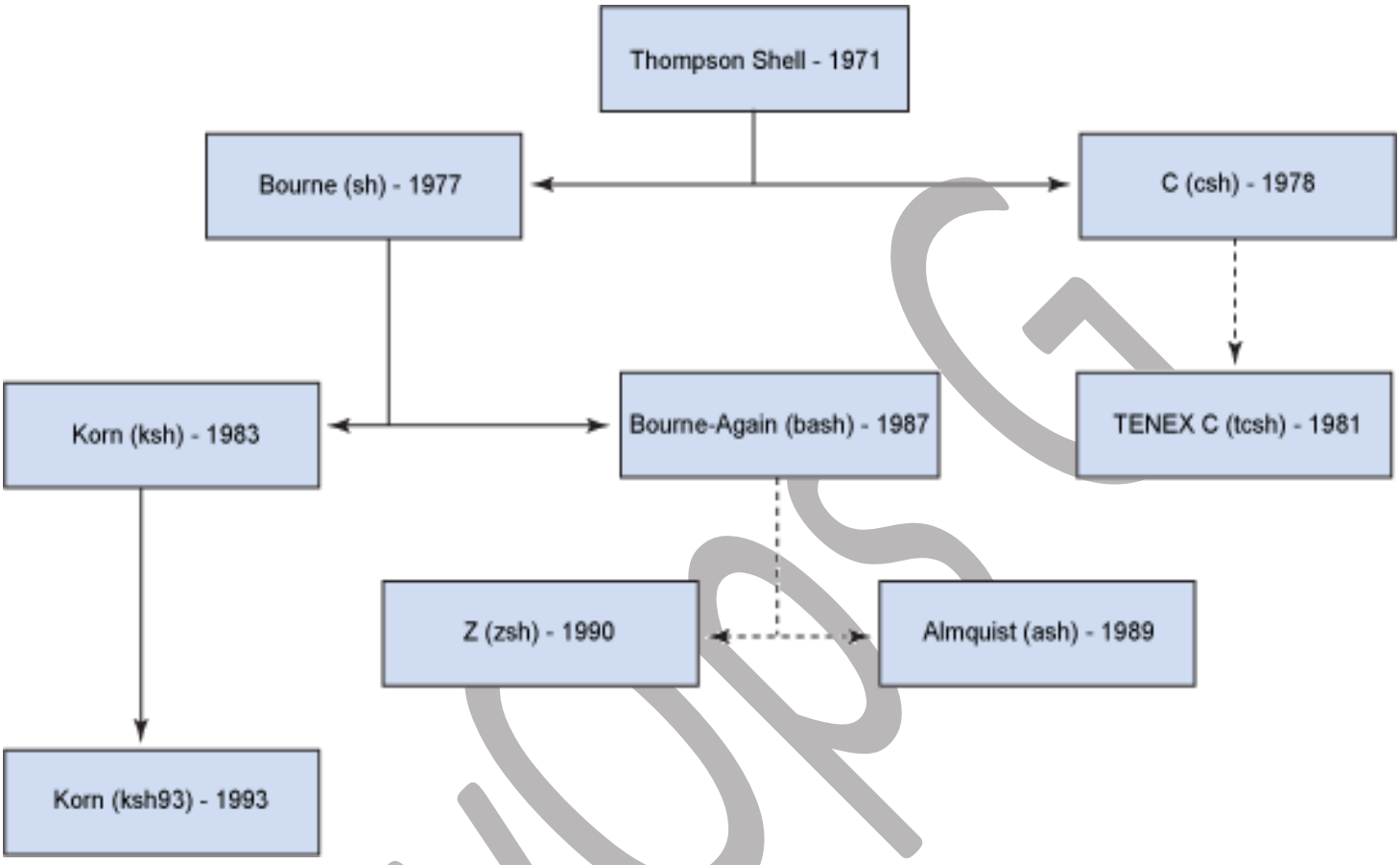
Shell Scripting by DevOps G

History of Shell: -



In below diagram we can see how to assign the variable and print its value by using $

```
[root@devopsg ~]# echo MYSTRING
MYSTRING
[root@devopsg ~]# echo $MYSTRING

[root@devopsg ~]# MYSTRING=devopsg
[root@devopsg ~]# echo $MYSTRING
devopsg
[root@devopsg ~]#
```

```
[root@devopsg ~]# MYVALUE=devops g
-bash: g: command not found
[root@devopsg ~]# #in above command we can see that space is considered
[root@devopsg ~]# #as seperator so whenever we use space we need to keep in quotes
[root@devopsg ~]# MYVALUE="devops g"
[root@devopsg ~]# echo $MYVALUE
devops g
[root@devopsg ~]# SUBVARIABLE="this is $MYVALUE to learn devops course"
[root@devopsg ~]# echo $SUBVARIABLE
this is devops g to learn devops course
[root@devopsg ~]# #now if we put the single quote instead of double quote then see the difference
[root@devopsg ~]# SUBVARIABLE='this is $MYVALUE to learn devops course'
[root@devopsg ~]# echo $SUBVARIABLE
this is $MYVALUE to learn devops course
[root@devopsg ~]# #we can see here $MYVALUE is printed as it is and not able to fetch its value
[root@devopsg ~]# echo $MYVALUE
devops g
[root@devopsg ~]# #however $MYVALUE values exists in shell
```

There exists few readonly variables whose value can't be changed by any user:-

```
[root@devopsg ~]# echo $PPID
1579
[root@devopsg ~]# PPID=myvalue
-bash: PPID: readonly variable
[root@devopsg ~]# #PPID is the parent process id and there exists come variable in shell which we can't change
[root@devopsg ~]# #even we can also create a readonly variable whose values can't be changed
[root@devopsg ~]# readonly MYREADONLYVAR=devopsg
[root@devopsg ~]# echo $MYREADONLYVAR
devopsg
[root@devopsg ~]# MYREADONLYVAR newvar
-bash: MYREADONLYVAR: command not found
[root@devopsg ~]# MYREADONLYVAR=newvar
-bash: MYREADONLYVAR: readonly variable
```

In below diagram, nested shell concept is shown: -

```
[root@devopsg ~]# MYVALUE=devopsg
[root@devopsg ~]# #now we are in parent shell, in which MYVALUE is devopsg
[root@devopsg ~]# echo $MYVALUE
devopsg
[root@devopsg ~]# #now getting into the subshell by typing bash, then we will see this value won't exist there
[root@devopsg ~]# bash
[root@devopsg ~]# echo $MYVALUE

[root@devopsg ~]# #we can see easily here $MYVALUE has not given any output here, now go to parent shell again
[root@devopsg ~]# exit
exit
[root@devopsg ~]# echo $MYVALUE
devopsg
[root@devopsg ~]# #this concept is nested shell
[root@devopsg ~]# #to remove the value assigned as variable in shell, use unset command
[root@devopsg ~]# unset MYVALUE
[root@devopsg ~]# echo $MYVALUE

[root@devopsg ~]# #but if we want to assign any variable to all bash including nested we should use export
[root@devopsg ~]# export MYVALUE=devopsg
[root@devopsg ~]# echo $MYVALUE
devopsg
[root@devopsg ~]# #going to nested bash
[root@devopsg ~]# bash
[root@devopsg ~]# echo $MYVALUE
devopsg
[root@devopsg ~]#
```

To check the bash version use command:-

bash –version

```
[root@devopsg ~]# bash --version
GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

```
[root@devopsg ~]# #we can check the available variable in os by using env or declare command
[root@devopsg ~]# #whatever variable exists over there we can print those using echo command
[root@devopsg ~]# echo $BASH_VERSINFO
4
[root@devopsg ~]# #if we see for BASH_VERSINFO using env command we will notice it is mentioned as array
[root@devopsg ~]# echo $BASH_VERSION
4.2.46(2)-release
[root@devopsg ~]# #to this BASH_VERSION BASH_VERSINFO will type as per array size defined, now we will see other conce
pt
[root@devopsg ~]# echo $BASH_VERSINFO[0]
4[0]
[root@devopsg ~]# #we see that [0] is printed insted of array value present at 0th position
[root@devopsg ~]# #this is because [0] is considered as string, we need to keep entire into {}
[root@devopsg ~]# echo ${BASH_VERSINFO[0]}
4
[root@devopsg ~]# echo ${BASH_VERSINFO[1]}
2
```

```
[root@devopsg ~]# #all bash variables are arrays
[root@devopsg ~]# A=1
[root@devopsg ~]# echo $A
1
[root@devopsg ~]# echo ${A[0]}
1
[root@devopsg ~]# echo ${A[1]}

[root@devopsg ~]#
```

```
[root@devopsg ~]# mkdir glob_test
[root@devopsg ~]# cd glob_test/
[root@devopsg glob_test]# touch file1 file2 file3
[root@devopsg glob_test]# ls
file1  file2  file3
[root@devopsg glob_test]# ls *
file1  file2  file3
[root@devopsg glob_test]# echo ls *
ls file1 file2 file3
[root@devopsg glob_test]# #in variable section we had seen '' prints plane value without derefrencing the values
[root@devopsg glob_test]# #whereas "" used to dereference the values and we gets the varibles values
[root@devopsg glob_test]# #but this is not the same in case of glob
[root@devopsg glob_test]# echo ls '*'
ls *
[root@devopsg glob_test]# #we see that '' behaves in same was it behaved in variables
[root@devopsg glob_test]# echo ls "*"
ls *
[root@devopsg glob_test]# #but we see that "" is also behaving here as ''
[root@devopsg glob_test]# ls *1
file1
[root@devopsg glob_test]# #this returns values which ends with 1 , also try ls *le1
[root@devopsg glob_test]# ls file[12]
file1  file2
[root@devopsg glob_test]# #it selects all files which has value 1 & 2, however if file12 will be present
[root@devopsg glob_test]# #it will also not be selected
[root@devopsg glob_test]# touch file12
[root@devopsg glob_test]# ls
file1  file12  file2  file3
[root@devopsg glob_test]# ls file[12]
file1  file2
[root@devopsg glob_test]#
```

```
[root@devopsg glob_test]# #we can see all files using * but can't see dot files .devopsg
[root@devopsg glob_test]# touch .devopsg
[root@devopsg glob_test]# ls *
file1  file12  file2  file3
[root@devopsg glob_test]# echo .*
. .. .devopsg
[root@devopsg glob_test]# #but here drawback is it will show all dot files or directories in same cetegory
[root@devopsg glob_test]# mkdir .mydotdir
[root@devopsg glob_test]# echo .*
. .. .devopsg .mydotdir
[root@devopsg glob_test]# #we can see .devopsg is files whereas .mydotdir is directory but here no difference shown
[root@devopsg glob_test]# ls .*
.devopsg

.:
file1  file12  file2  file3

..:
dockerfiles  glob_test  my_container.tar  my_nginx

.mydotdir:
[root@devopsg glob_test]# #in above o/p 1st .devopsg file is shown, then . content is shown(present dir)
[root@devopsg glob_test]# #then .. content is shown, means parent dir, then .mydotdir is shown
[root@devopsg glob_test]# #so when we uses .* it shows o/p of all current folder and parent folder
```

```
[root@devopsg pipes]# echo "this is shell scripting by devopsg" > file1
[root@devopsg pipes]# cat file1
this is shell scripting by devopsg
[root@devopsg pipes]# ls
file1
[root@devopsg pipes]# #pipe symobol is | and pipe is used to combine more then one command
[root@devopsg pipes]# echo 'this is shell scripting by devopsg' > file2
[root@devopsg pipes]# ls
file1  file2
[root@devopsg pipes]# cat file2
this is shell scripting by devopsg
[root@devopsg pipes]# #we can append any lines to a file using >>
[root@devopsg pipes]# echo 'this is new line appending in file2' >> file2
[root@devopsg pipes]# cat file2
this is shell scripting by devopsg
this is new line appending in file2
[root@devopsg pipes]#
```

```
[root@devopsg pipes]# hostname
devopsg
[root@devopsg pipes]# echo hostname
hostname
[root@devopsg pipes]# echo $hostname

[root@devopsg pipes]# #we see no hostname is printed with $
[root@devopsg pipes]# echo $(hostname)
devopsg
[root@devopsg pipes]# #or we can use `` also to print hostname instead of ()
[root@devopsg pipes]# echo `hostname`
devopsg
```

```
[root@devopsg pipes]# #now we will use nesting of $
[root@devopsg pipes]# ls
file1  file2
[root@devopsg pipes]# ls ..
dockerfiles  glob_test  pipes
[root@devopsg pipes]# #we had seen content of current file and parent file
[root@devopsg pipes]# echo $(touch $(ls ..))

[root@devopsg pipes]# ls
dockerfiles  file1  file2  glob_test  pipes
[root@devopsg pipes]# #we can see dockerfiles glob_test pipes which was part of parent dir created in current file
```

```
[root@devopsg pipes]# ls
[root@devopsg pipes]# ls ..
dockerfiles  glob_test  pipes
[root@devopsg pipes]# echo `touch \` ls .. \``

[root@devopsg pipes]# ls
dockerfiles  glob_test  pipes
[root@devopsg pipes]# #in above command we had seen `` is also behaving the same way of () for command substitution
```

```
[root@devopsg function]# ls
[root@devopsg function]# function myfunc {
> echo Welcome to DevOps G
> }
[root@devopsg function]# myfunc
Welcome to DevOps G
[root@devopsg function]# #we can pass the argument to function in runtime as well in cmd line argument
[root@devopsg function]# function myfunc {
> echo $1
> echo $2
> echo $3
> }
[root@devopsg function]# myfunc "Welcome to DevOps_G"
Welcome to DevOps_G


[root@devopsg function]# #above we can notice Welcome to DevOps_G is printed on one line and 2 blank line came
[root@devopsg function]# #this happened because of "" not lets remove ""
[root@devopsg function]# myfunc Welcome to DevOps_G
Welcome
to
DevOps_G
[root@devopsg function]# myfunc Welcome to DevOps_G to learn DevOps
Welcome
to
DevOps_G
[root@devopsg function]# #above we can see only first 3 argument printed as myfunc has 3 arg defined
[root@devopsg function]#
```

```
[root@devopsg function]# #we can define variable as well inside function
[root@devopsg function]# function myfunc {
> echo $myvar
> }
[root@devopsg function]# myfunc

[root@devopsg function]# #we see no o/p printed because variable myvar doesn't exists
[root@devopsg function]# myvar=DevOps_G
[root@devopsg function]# #here we defined variable myvar
[root@devopsg function]# myfunc
DevOps_G
[root@devopsg function]# #we can see now variable value is printed by running function we created myfunc
[root@devopsg function]# #we can define local variable inside the function as well that will have access inside those
function only but not be accessed outside of that function
[root@devopsg function]# function myfunc {
> local funvar="This variable is declared inside the function"
> echo $funvar
> }
[root@devopsg function]# myfunc
This variable is declared inside the function
[root@devopsg function]# echo $funvar

[root@devopsg function]# #we can see when we are running funvar variable, no o/p comes as it not acceible outside
[root@devopsg function]# local newvar="DevOps_G"
-bash: local: can only be used in a function
[root@devopsg function]#
```

Note: We can't create local variable outside of a function, it can only be created inside the function

To get the list of function defined by user use the command: - declare -f

```
[root@devopsg function]# declare -f
myfunc ()
{
    local funvar="This variable is declared inside the function";
    echo $funvar
}
[root@devopsg function]# alias devopsg=ls
[root@devopsg function]# #in above command we had aliased the devopsg with ls command means whenever we type the devop
sg we will get the same o/p of ls command
[root@devopsg function]# alias
alias cp='cp -i'
alias devopsg='ls'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[root@devopsg function]# devopsg
[root@devopsg function]# touch file1
[root@devopsg function]# devopsg
file1
[root@devopsg function]# #to remove the alias which we set use the command unalias
[root@devopsg function]# unalias devopsg
[root@devopsg function]# #now if we run devopsg we would get the error command not found
[root@devopsg function]# devopsg
-bash: devopsg: command not found
[root@devopsg function]#
```

```
[root@devopsg function]# #we uses the test command inside angular braces []
[root@devopsg function]# [1=0]
-bash: [1=0]: command not found
[root@devopsg function]# # remember [1=0] or [1 = 0] or [ 1=0] or [1 = 0 ] will give error command not found, remember
 always to give a space after opening and closing of braces, now we can see we haven't got any output either true or f
alse so we will use "echo $?" command to check o/p of last command as we know if output comes as 0 then its true else
false for other values
[root@devopsg function]# [ 1 = 0 ]
[root@devopsg function]# echo $?
1
[root@devopsg function]# [ 1 = 1 ]
[root@devopsg function]# echo $?
0
[root@devopsg function]# #similarly we can compare with variable as well
[root@devopsg function]# B=1
[root@devopsg function]# [ $B = 1 ]
[root@devopsg function]# echo $?
0
[root@devopsg function]# [ $B = 2 ]
[root@devopsg function]# echo $?
1
[root@devopsg function]# #even i can compare with srting value as well
[root@devopsg function]# [ $(hostname) = devopsg ]
[root@devopsg function]# echo $?
0
[root@devopsg function]# [ $(hostname) = unknown_host ]
[root@devopsg function]# echo $?
1
[root@devopsg function]#
```

```
[root@devopsg function]# #syntax for logical end is && and logical or is ||
[root@devopsg function]# nocommand && echo devopsg
-bash: nocommand: command not found
[root@devopsg function]# #in above command as nocommand failed so 2nd part not worked
[root@devopsg function]# nocommand || echo devopsg
-bash: nocommand: command not found
devopsg
[root@devopsg function]# #in above command as nocommand failed but still 2nd part worked because of logical or
[root@devopsg function]# #usecase of && in real life "mkdir devopsg && cd devopsg" what this command will do first
create a devopsg dir and only if the dir is created it will move inside the dir
[root@devopsg function]# # ! means no, contents inside round braces always evaluated first ()
[root@devopsg function]# ([ 1 = 1 ] || [ !0 = 0 ]) && [ 2 = 2 ]
-bash: !0: event not found
[root@devopsg function]# ([ 1 = 1 ] || [ ! 0 = 0 ]) && [ 2 = 2 ]
[root@devopsg function]# #in above command first () is evaluated then compared with [ 2 = 2 ]
[root@devopsg function]# echo $?
0
[root@devopsg function]# # to avoid this much of braces in || && we can use -o and -a , -o is || and -a is &&
[root@devopsg function]# [ 1 = 1 -o ! 0 = 0 -a 2 = 2 ]
[root@devopsg function]# echo $?
0
```

```
[root@devopsg function]# [[ 10 < 2 ]]
[root@devopsg function]# echo $?
0
[root@devopsg function]# [[ 10 -lt 2 ]] #lt is for less then
[root@devopsg function]# echo $?
1
[root@devopsg function]# [[ 10 -gt 2 ]] #gt is for greater then
[root@devopsg function]# echo $?
0
[root@devopsg function]# [[ 10 -eq 10 ]] #eq is for equalto
[root@devopsg function]# echo $?
0
[root@devopsg function]# [[ 10 -ne 10 ]] #ne is for not-equalto
[root@devopsg function]# echo $?
1
[root@devopsg function]# declare -i
declare -ir BASHPID
declare -ir EUID="0"
declare -i HISTCMD
declare -i LINENO
declare -i MAILCHECK="60"
declare -i OPTIND="1"
declare -ir PPID="3608"
declare -i RANDOM
declare -ir UID="0"
[root@devopsg function]# echo $RANDOM
7821
[root@devopsg function]#
```

```
[root@devopsg ~]# if [[ 10 -lt 2 ]]
> then
> echo "10 is less then 2"
> elif [[ 10 -gt 2 ]]
> then
> echo "10 is greater then 2"
> else
> echo "not available"
> fi
10 is greater then 2
[root@devopsg ~]#
```

```
[root@devopsg for_loop]# ls
[root@devopsg for_loop]# for (( i=0; i<5; i++ ))
> do
> echo $i
> echo $i > file${i}.txt
> done
0
1
2
3
4
[root@devopsg for_loop]# ls
file0.txt  file1.txt  file2.txt  file3.txt  file4.txt
[root@devopsg for_loop]# for f in $(ls *txt)
> do
> echo "File $f contains: $(cat $f)"
> done
File file0.txt contains: 0
File file1.txt contains: 1
File file2.txt contains: 2
File file3.txt contains: 3
File file4.txt contains: 4
[root@devopsg for_loop]#
```

```
[root@devopsg for_loop]# ls
file0.txt  file1.txt  file2.txt  file3.txt  file4.txt
[root@devopsg for_loop]# n=0
[root@devopsg for_loop]# while [[ ! -a newfile ]]
> do
> ((n++))
> echo doing iteration $n
> if [[ $(cat file${n}.txt) == "4" ]]
> then
> echo breaking out
> break
> fi
> sleep 1
> done
doing iteration 1
doing iteration 2
doing iteration 3
doing iteration 4
breaking out
```

Standard Exit code in shell: -

0 → OK

1 → General Error

2 → Misuse of shell builtin

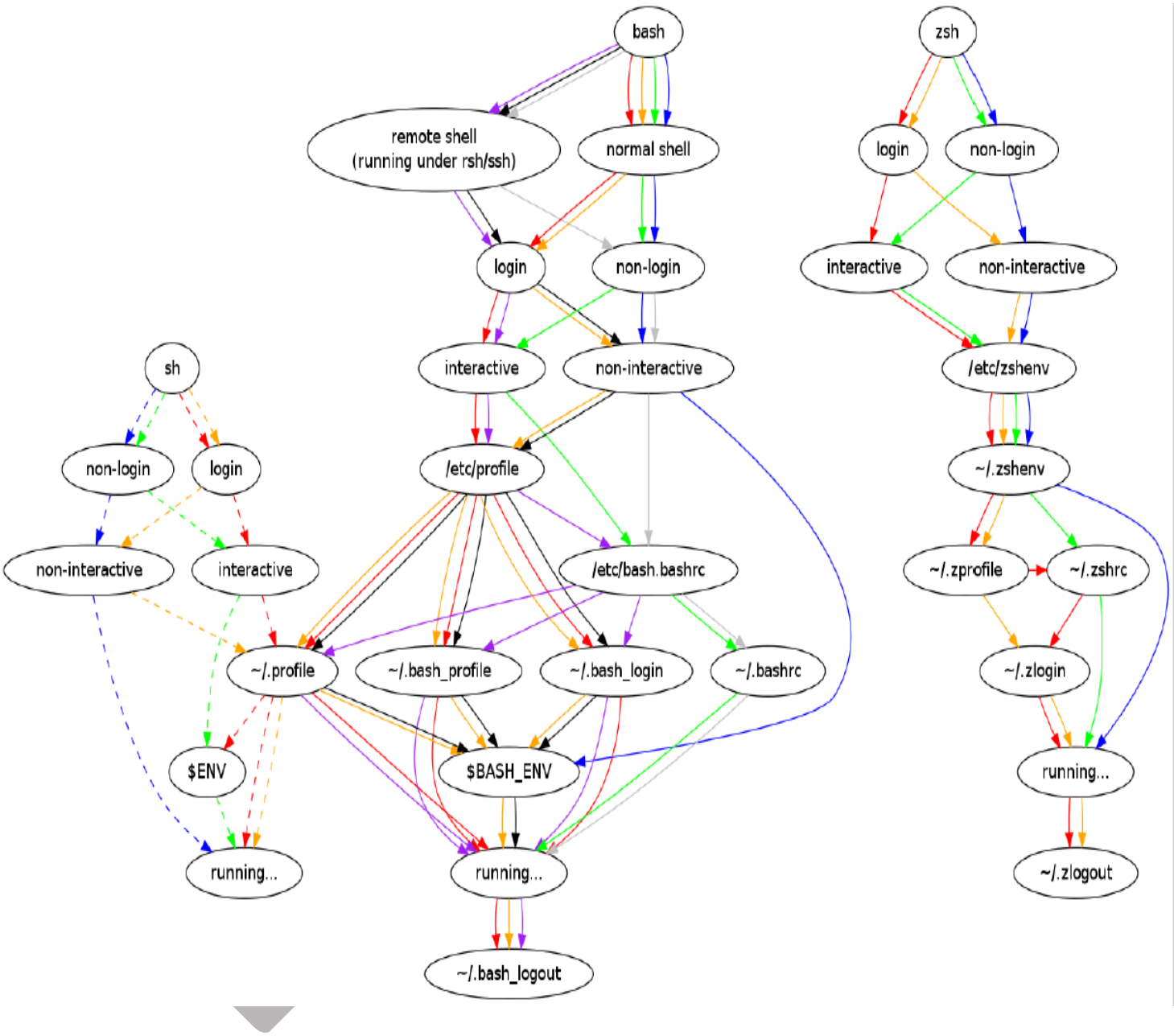
126 → Can't execute

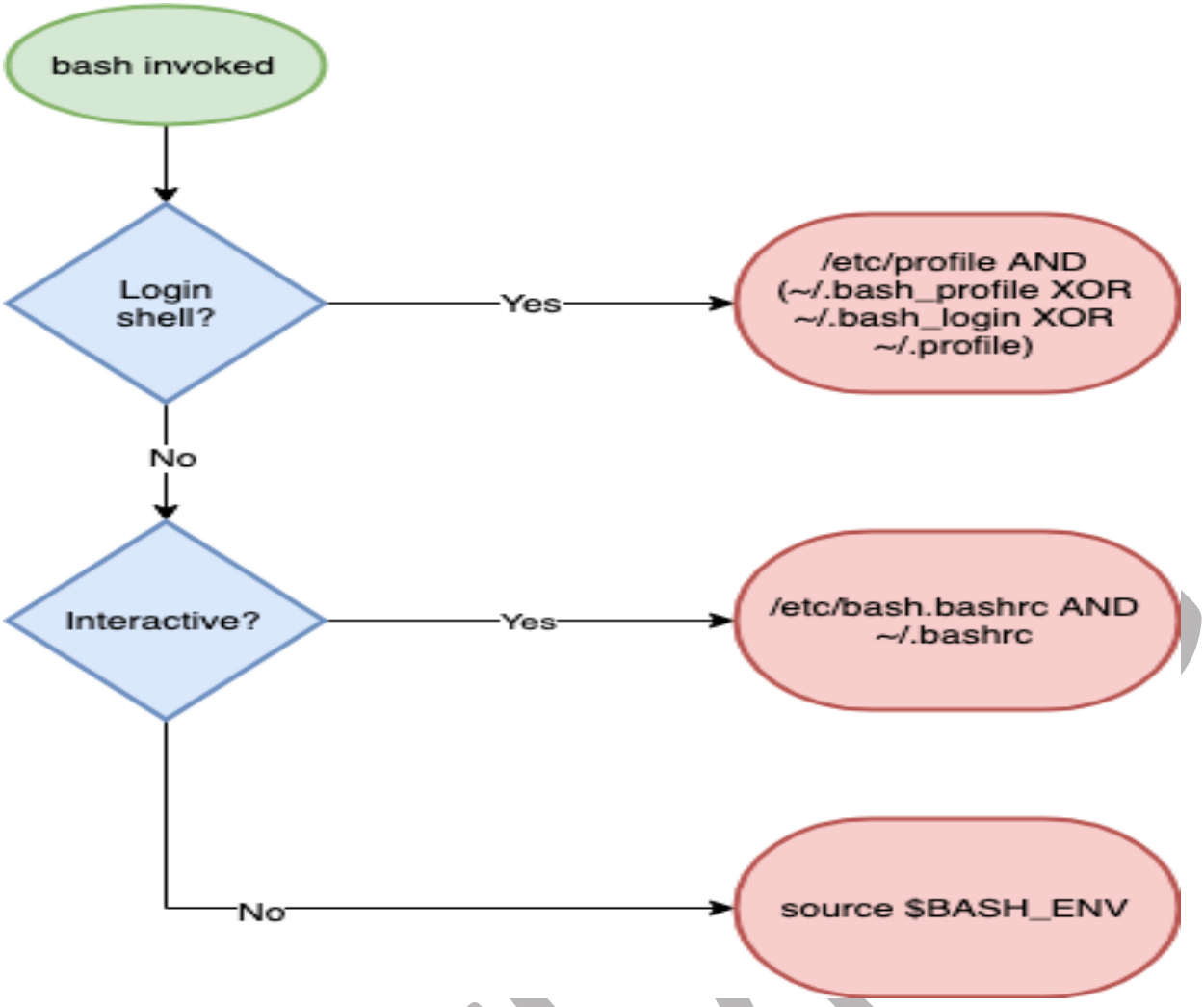
127 → No file found matching command

128 → Invalid exit value

(128+n) → process killed with signal n

Shell Start-up cycle: -





```
[root@devopsg script]# vi set_usage.sh
[root@devopsg script]# echo $HOME
/root
[root@devopsg script]# ll -a
total 8
drwxr-xr-x.  2 root root   26 Aug 24 10:33 .
dr-xr-x---. 17 root root 4096 Aug 24 10:28 ..
-rw-r--r--.  1 root root  369 Aug 24 10:33 set_usage.sh
[root@devopsg script]# chmod +x set_usage.sh
[root@devopsg script]# ll -a set_usage.sh
-rwxr-xr-x. 1 root root 369 Aug 24 10:33 set_usage.sh
[root@devopsg script]# ./set_usage.sh
+ set -o nounset
+ pwd
/root/script
+ cd /root
./set_usage.sh: line 8: DOESNOTEXIST: unbound variable
[root@devopsg script]# cat set_usage.sh
#!/bin/bash
set -o errexit #this exits the script on getting any error in the script
set -o xtrace #it will show status with + the steps appliec correctly or not
set -o nounset #it will throw error when any variable used is not defined in OS

pwd
cd $HOME
echo $DOESNOTEXIST
echo "The shell should not run this as thrown error in last step, DOESNOTEXIST not defined"

[root@devopsg script]#
```

```
[root@devopsg script]# set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
[root@devopsg script]# #we can see all available option here, and we can on or off any option using + or -, eg. to on
any option say posix , use "set -o posix" and again to off it use "set +o posix"
[root@devopsg script]#
```

```
[root@devopsg subshell]# ls
[root@devopsg subshell]# VAR1='Parent shell'
[root@devopsg subshell]# (
> echo "Running inside subshell"
> echo ${VAR1}
> VAR1='Updating var1 value'
> echo ${VAR1}
> VAR2='second variable'
> echo ${VAR2}
> )
Running inside subshell
Parent shell
Updating var1 value
second variable
[root@devopsg subshell]# echo $VAR1
Parent shell
[root@devopsg subshell]# #we can see inside subshell value of VAR1 was modified to 'Updating var1 value' but this
changes not implemented in parent shell, and VAR2 if we chack we won't get any o/p
[root@devopsg subshell]# echo $VAR2

[root@devopsg subshell]# pwd
/root/subshell
[root@devopsg subshell]# (
> cd /tmp
> pwd
> )
/tmp
[root@devopsg subshell]# pwd
/root/subshell
[root@devopsg subshell]# #we can see in subshell we was inside /tmp but in parent shell no changes observed
[root@devopsg subshell]#
```