The difference between "**oraenv**" and "**. oraenv**" on Unix.

"Hey!! I executed oraenv and my $ORACLE_HOME is still incorrect!?"
"oraenv is not working anymore!?"
...

The most common cause is that "oraenv" was executed instead of ". oraenv".

So what is the difference between "oraenv" and ". oraenv" ?

- When "oraenv" is executed, it will run as a child process of the current process, either your shell prompt or a batch script. A child process can impossibly change the environment of its parent process, so when oraenv script completes then all environment variables defined or modified in that script are lost.
- However, when ". oraenv" (mind the dot) is executed, it will be executed in the current shell environment. Because the script is not executed in a separate child process it will directly change the shell environment of the calling process.

Evidence :

```
# Small script to display current Process ID
$ cat abcd
echo "abcd PID= $$"

# chmod to read-only – display current PID – execute with dot command
$ chmod 400 abcd ; echo "PID =" $$  ;  . ./abcd
PID = 8036          (output from calling shell)
abcd PID= 8036      (output from script, note same PID as calling shell)

# chmod to read & execute – display current Process ID – execute the script
$ chmod 500 abcd ; echo "PID =" $$  ;  ./abcd
PID = 8036          (output from calling shell)
abcd PID= 8181      (output from script, different PID => child process)
```

The difference between "**oraenv**" and "**. oraenv**" on Unix.

Funny side effect :

```
# Again small script to display its PID, number of arguments and the arguments
itself.
$ cat abcd
echo "abcd PID= $$"
echo "Argument count= $#"
echo "Argument list = $@"

# Execute the script with one argument – behaves as expected.
$ ./abcd arg1
abcd PID= 10588
Argument count= 1
Argument list = arg1

# Execute again with no arguments – behaves as expected
$ ./abcd
abcd PID= 10591
Argument count= 0
Argument list =

# Execute with dot command with one argument – behaves as expected
$ . ./abcd arg1
abcd PID= 8036
Argument count= 1
Argument list = arg1

# Again – with 2 arguments – behaves as expected
$ . ./abcd arg1 arg2
abcd PID= 8036
Argument count= 2
Argument list = arg1 arg2

# Again – with no arguments – uh??
$ . ./abcd
abcd PID= 8036
Argument count= 2
Argument list = arg1 arg2

# Again without arguments or dot command – behaves as expected
$ ./abcd
abcd PID= 12037
Argument count= 0
Argument list =
```

So what happened here?

- when "./abcd" was executed it was executed as a child process, and therefore had no effect on the environment of the calling process. And the arguments counted and shown are the ones which were actually passed to that child process.
- when ". ./abcd" was executed it was executed in the current shell environment. So when arguments were supplied it (also) modified the argument list of the calling shell because it's one and the same.

The difference between "**oraenv**" and "**. oraenv**" on Unix.

References :

KSH man page :
. file [arg ...]
Read and execute commands from file and return. The commands are executed in the current shell environment.
The search path specified by PATH is used to find the directory containing file. If any arguments arg are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command
executed. It is not necessary that the execute permission bit be set for file

UNIX Power Tools (O'Reilly)
Part VIII : Shell Programming : "44.23 Reading Files with the . and source Commands"