HrOUG – Spring 2022

Scripting in SQLcl

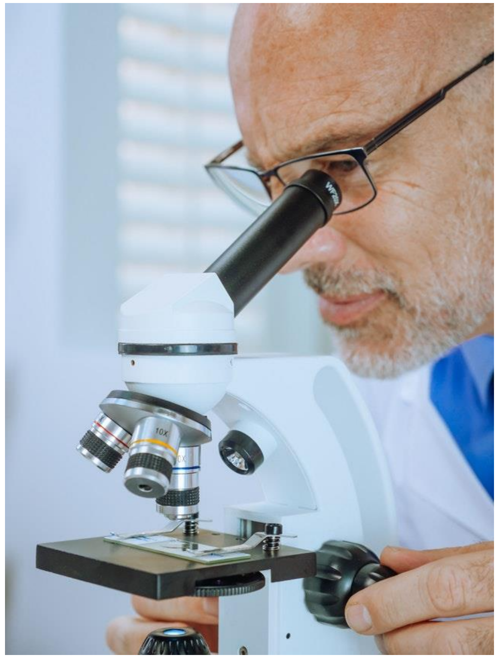
You can never have enough of a good thing

Erik van Roon

EvROCS
COMPLETING THE PUZZLE

Who Am I?

Erik van Roon



1995 → ORACLE DATABASE >=Oracle5 → 2009 → EvROCS COMPLETING THE PUZZLE

SYM42

https://sym42.org/

MASH

Core team
MASH Program

ORACLE ACE Pro

: erik.van.roon@evrocs.nl
: www.evrocs.nl
: @evrocs_nl

MASH PROGRAM

Mentor and Speaker Hub

Our goal is to *connect* speakers with mentors to assist in *preparing* technical sessions and *improving* presentation skills

Interested? Read more and get in touch

https://mashprogram.wordpress.com

ORACLE
ACE Program

500+ technical experts helping peers globally

The **Oracle ACE Program** recognizes and rewards community members for their technical contributions in the Oracle community

3 membership tiers

ORACLE ACE Director | ORACLE ACE | ORACLE ACE Associate

For more details on Oracle ACE Program:
bit.ly/OracleACEProgram

Oracle Groundbreakers

Nominate
yourself or someone you know:
acenomination.oracle.com

Connect: **oracle-ace_ww@oracle.com** Facebook.com/oracleaces @oracleace

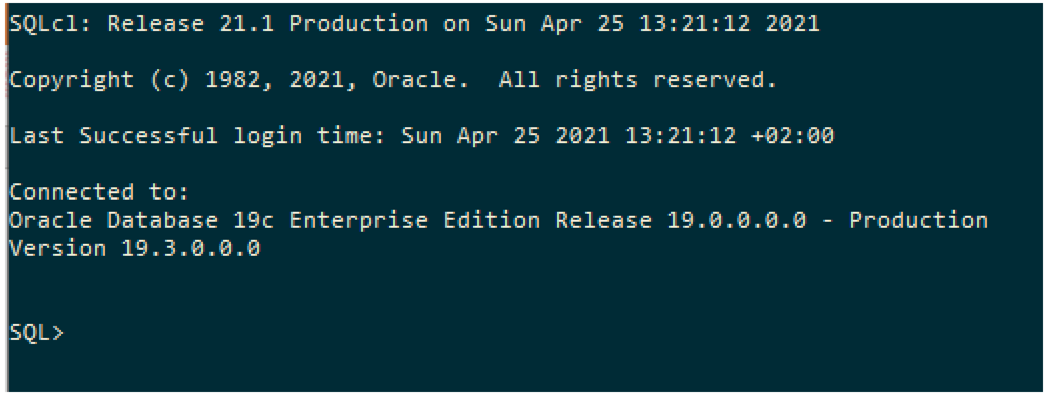
What's that ?
SQLcl ???
Not a clue!?!

What is SQLcl?

Command line interface for the database

Like SQL*Plus, but with soooooo much more

This .....



```
SQLcl: Release 21.1 Production on Sun Apr 25 13:21:12 2021

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Last Successful login time: Sun Apr 25 2021 13:21:12 +02:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0


SQL>
```

Command history

Command completion

Aliases

Apex export

CD
PWD

Liquibase integration

In line editing

Syntax Highlighting

Codescan
(SQL Injection & performance warnings)

Code formatting

Load & Unload commands

Startup.sql

Datapump command

Query output formatting
(csv, json, insert statements, smart column widths, color highlighting)

DDL Command

Status Bar

Spool to zip file

Bridge command

Info / Info+
(Desc but a lot more information)

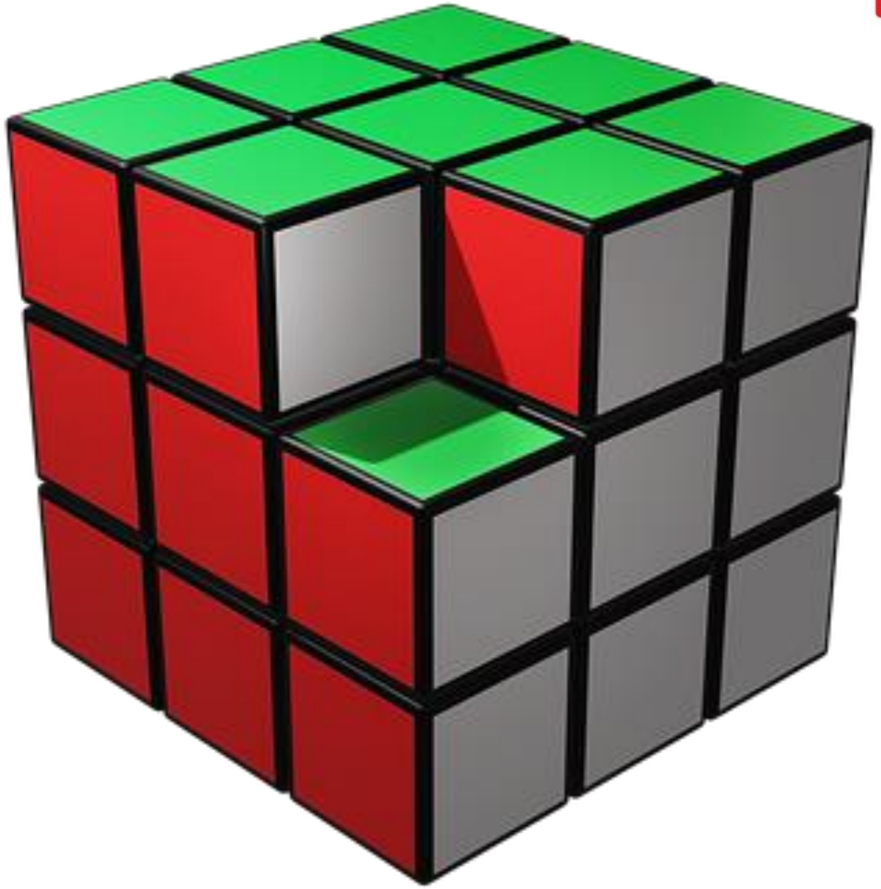
Dataguard command

... And much more...

Is it perfect?

No!

No tool ever is……

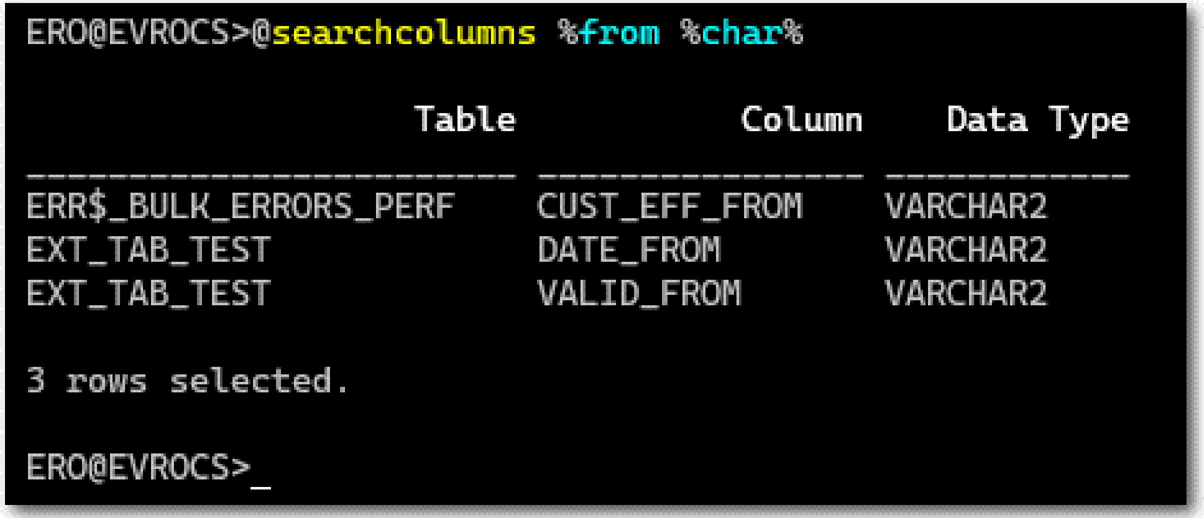


Example:

A SQL-script can have parameters.

Script SearchColumns.sql

```
select atc.table_name    "Table"
,      atc.column_name   "Column"
,      atc.data_type     "Data Type"
from   user_tab_columns    atc
where  atc.column_name like upper('&1')
  and  atc.data_type   like upper('&2')
order by 1, 2, 3
;

undef 1
undef 2
```

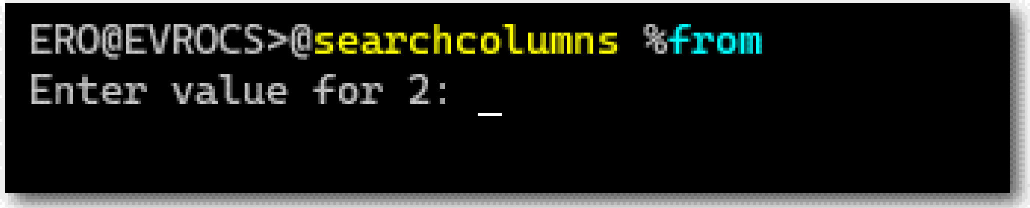


```
ERO@EVROCS>@searchcolumns %from %char%

                    Table           Column     Data Type
------------------------- ---------------- ------------
ERR$_BULK_ERRORS_PERF     CUST_EFF_FROM    VARCHAR2
EXT_TAB_TEST              DATE_FROM        VARCHAR2
EXT_TAB_TEST              VALID_FROM       VARCHAR2

3 rows selected.

ERO@EVROCS>_
```

But the parameters can not be optional

I want it to use "%" if I don't provide a second parameter value

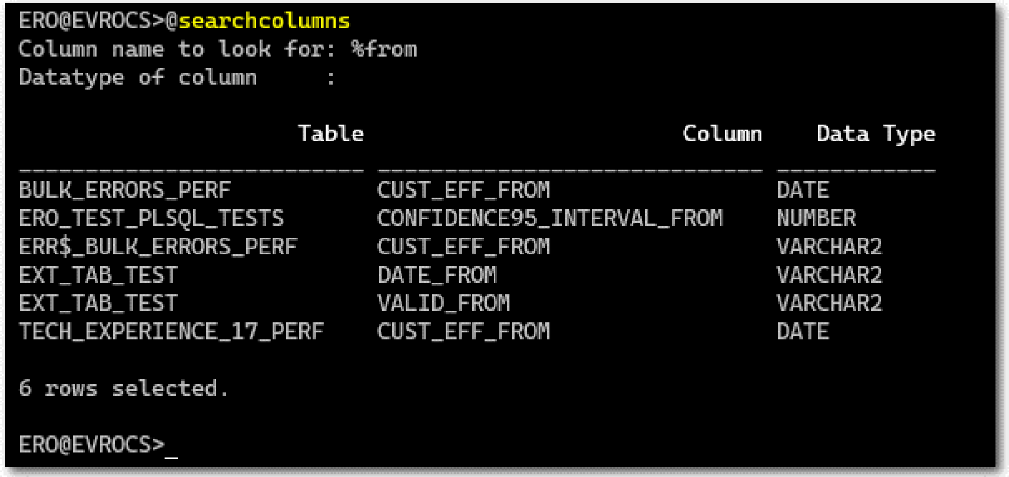
But obviously.....



```
ERO@EVROCS>@searchcolumns %from
Enter value for 2: _
```

Solution 1: Use "Accept"

```
accept col_name prompt "Column name to look for: "
accept datatype prompt "Datatype of column      : "

select atc.table_name    "Table"
,      atc.column_name   "Column"
,      atc.data_type     "Data Type"
from   user_tab_columns   atc
where  atc.column_name like coalesce
                            (upper('&col_name'),'%')
  and  atc.data_type   like coalesce
                            (upper('&datatype'),'%')
order by 1, 2, 3
;

undef col_name
undef datatype
```



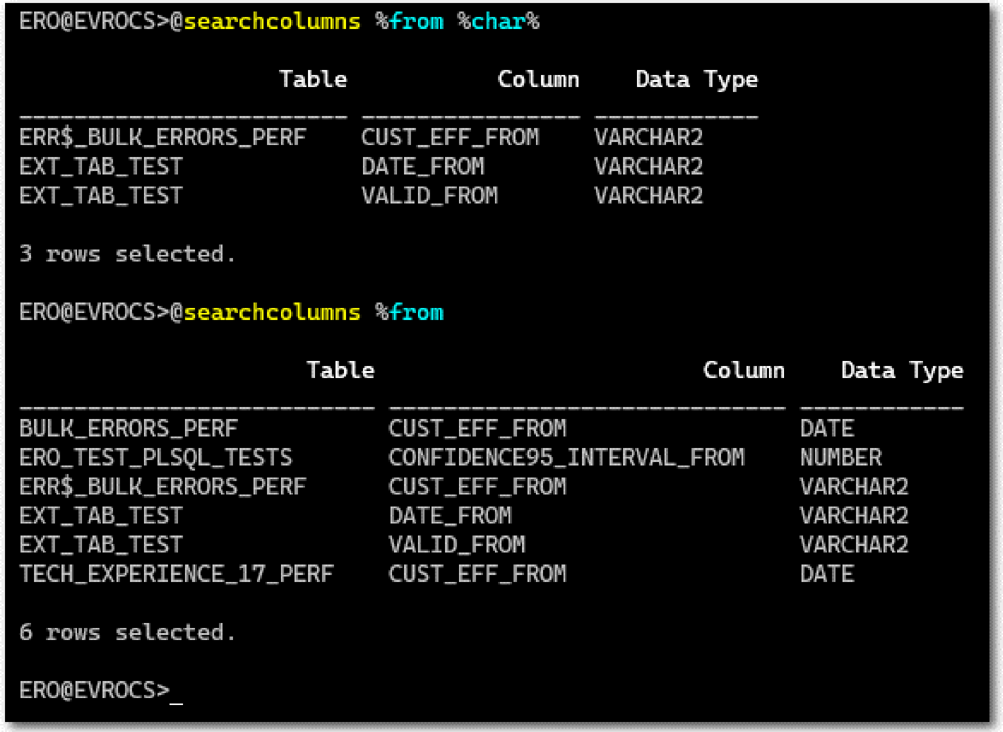
```
ERO@EVROCS>@searchcolumns
Column name to look for: %from
Datatype of column      :

                    Table                           Column   Data Type
------------------------- ----------------------------- ------------
BULK_ERRORS_PERF          CUST_EFF_FROM                 DATE
ERO_TEST_PLSQL_TESTS      CONFIDENCE95_INTERVAL_FROM    NUMBER
ERR$_BULK_ERRORS_PERF     CUST_EFF_FROM                 VARCHAR2
EXT_TAB_TEST              DATE_FROM                     VARCHAR2
EXT_TAB_TEST              VALID_FROM                    VARCHAR2
TECH_EXPERIENCE_17_PERF   CUST_EFF_FROM                 DATE

6 rows selected.

ERO@EVROCS>_
```

Awesome, but……

The script is now interactive

Solution 2: Use "column new_value"

```
column 1 new_value 1
column 2 new_value 2

set termout off
select null  "1"
,      null  "2"
from   dual
where  rownum = 0
;
set termout on

select atc.table_name    "Table"
,      atc.column_name   "Column"
,      atc.data_type     "Data Type"
from   user_tab_columns    atc
where  atc.column_name like coalesce(upper('&1'),'%')
  and  atc.data_type   like coalesce(upper('&2'),'%')
order by 1, 2, 3
;

undef 1
undef 2
```



```
ERO@EVROCS>@searchcolumns %from %char%

                   Table           Column    Data Type
------------------------ ---------------- ------------
ERR$_BULK_ERRORS_PERF    CUST_EFF_FROM    VARCHAR2
EXT_TAB_TEST             DATE_FROM        VARCHAR2
EXT_TAB_TEST             VALID_FROM       VARCHAR2

3 rows selected.

ERO@EVROCS>@searchcolumns %from

                     Table                        Column    Data Type
-------------------------- ----------------------------- ------------
BULK_ERRORS_PERF           CUST_EFF_FROM                 DATE
ERO_TEST_PLSQL_TESTS       CONFIDENCE95_INTERVAL_FROM    NUMBER
ERR$_BULK_ERRORS_PERF      CUST_EFF_FROM                 VARCHAR2
EXT_TAB_TEST               DATE_FROM                     VARCHAR2
EXT_TAB_TEST               VALID_FROM                    VARCHAR2
TECH_EXPERIENCE_17_PERF    CUST_EFF_FROM                 DATE

6 rows selected.

ERO@EVROCS>_
```

Works as intended, but.....

- Is a bit of a hack
- Too many lines to get 1 thing done
- Needs termout off (in SQLcl) because "noprint" is not supported

What I would want.....

A command like this:

```
VariableDefault <VariableName> <DefaultValue> [<TargetVariable>]
```

Example 1:

```
VariableDefault 2 %
```

Checks if variable 2 exists (if second parameter value is supplied)
If so : Leaves value of "2" as it is
If not : Assigns value "%" to variable "2"

Example 2:

```
VariableDefault 2 % my_param_2
```

Checks if variable 2 exists (if second parameter value is supplied)
If so : Assigns value of "2" to variable "my_param_2"
If not : Assigns value "%" to variable "my_param_2"

But that doesn't exist

What can we do?

1. We can wait until Oracle spontaneously implements it

2. We can keep asking Oracle for it

3. We can ……………….
……………….. implement it ourselves

We can run JavaScript scripts from SQLcl

(And use java classes as well)

Version >= 22.1

Requires JDK >= 11

But Nashorn is removed in Java 15

So for JavaScript scripting: JDK **=** 11 required

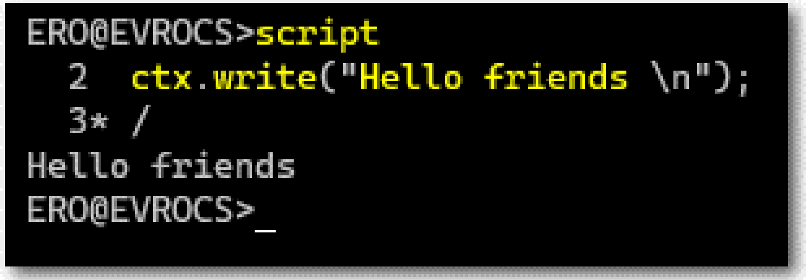
Or: use GraalVM for Java >= 11

I recommend this because:

- JavaScript engine remains included
- Some JavaScript things not supported by Nashorn in JDK that *are* supported in JavaScript engine in GraalVM (Example: const)

So we can execute JavaScript, you say?

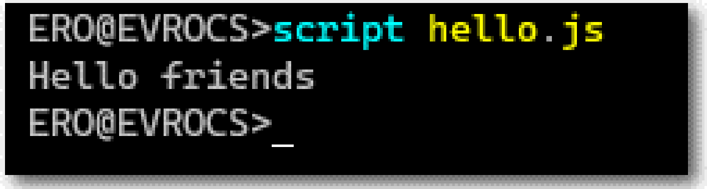
From the command line



```
ERO@EVROCS>script
  2  ctx.write("Hello friends \n");
  3* /
Hello friends
ERO@EVROCS>_
```

(type "script" followed by JavaScript and terminated with slash)

From a script file



```
ERO@EVROCS>script hello.js
Hello friends
ERO@EVROCS>_
```

(type "script" followed by filename, possibly including path)

What would our script need to do?

Remember?

```
VariableDefault <VariableName> <DefaultValue> [<TargetVariable>]
```

It has to

- Accept 2 or 3 parameters
    1. <VariableName> - Name of the variable to check
    2. <DefaultValue> - Default value to use if variable doesn't exist
    3. <TargetVariable> - (Optional) Name of variable to set
- Check if a variable exists with name <VariableName>
- Set the value for a variable
    - If a variable name is given in <TargetVariable>, then <TargetVariable>
    - If not, then <VariableName>

Simple......

The Script…..

```
"use strict";
```

Use the "use strict" directive to force declaration of variables before using them

The Script.....

```
"use strict";

var paramCount        = args.length - 1;
var substVarSource    = null;
var substVarTarget    = null;
var substVarValue     = null;


// Param 1: name of the substVar that needs to be defaulted if it does not exist
substVarSource     = args[1].toUpperCase();







// Param 2: value to be used as default for the substVar if it does not exist



  substVarValue = args[2];


// Param 3 (optional): name of a substVar to which the value should be written
if (paramCount === 3) {
  substVarTarget    = args[3].toUpperCase();
} else {
  substVarTarget    = substVarSource;
}
```

Accept 2 or 3 parameters:

All parameters are in an array "args"
First element (index 0) is the scriptname
The other elements (1-n) are parameters.

Get value of parameters into variables

The Script.....

```
"use strict";

var paramCount       = args.length - 1;
var substVarSource   = null;
var substVarTarget   = null;
var substVarValue    = null;
var substVarFound    = false;

// Param 1: name of the substVar that needs to be defaulted if it does not exist
substVarSource     = args[1].toUpperCase();

// Check if the source variable exists
for each (var definedVarName in ctx.getMap().keySet()) {
  if (definedVarName === substVarSource) {
    substVarFound = true;
  }
}

// Param 2: value to be used as default for the substVar if it does not exist



  substVarValue = args[2];


// Param 3 (optional): name of a substVar to which the value should be written
if (paramCount === 3) {
  substVarTarget   = args[3].toUpperCase();
} else {
  substVarTarget   = substVarSource;
}
```

Check if substitution variable exists

A "map" is available that is 'linked' to the substitution variables:

ctx.getMap()

Unfortunately the method "has" of maps does not seem to be available

So we just have to loop through the entries

The Script.....

```
"use strict";

var paramCount        = args.length - 1;
var substVarSource    = null;
var substVarTarget    = null;
var substVarValue     = null;
var substVarFound     = false;

// Param 1: name of the substVar that needs to be defaulted if it does not exist
substVarSource     = args[1].toUpperCase();

// Check if the source variable exists
for each (var definedVarName in ctx.getMap().keySet()) {
  if (definedVarName === substVarSource) {
    substVarFound = true;
  }
}

// Param 2: value to be used as default for the substVar if it does not exist
if (substVarFound) {
  substVarValue = ctx.getMap().get(substVarSource);
} else {
  substVarValue = args[2];
}

// Param 3 (optional): name of a substVar to which the value should be written
if (paramCount === 3) {
  substVarTarget    = args[3].toUpperCase();
} else {
  substVarTarget    = substVarSource;
}
```

If the substitution variable was found, then put its value in the substVarValue variable instead of the default value.

Now substVarValue contains the value that we want to end up with

The Script…..

```
"use strict";

var paramCount       = args.length - 1;
var substVarSource   = null;
var substVarTarget   = null;
var substVarValue    = null;
var substVarFound    = false;

// Param 1: name of the substVar that needs to be defaulted if it does not exist
substVarSource     = args[1].toUpperCase();

// Check if the source variable exists
for each (var definedVarName in ctx.getMap().keySet()) {
  if (definedVarName === substVarSource) {
    substVarFound = true;
  }
}

// Param 2: value to be used as default for the substVar if it does not exist
if (substVarFound) {
  substVarValue = ctx.getMap().get(substVarSource);
} else {
  substVarValue = args[2];
}

// Param 3 (optional): name of a substVar to which the value should be written
if (paramCount === 3) {
  substVarTarget   = args[3].toUpperCase();
} else {
  substVarTarget   = substVarSource;
}

// Set the target variable to be the value we determined.
ctx.getMap().put(substVarTarget, substVarValue);
```

Finally put the desired value into the desired substitution variable

Just "put" the entry in the map.

The name of the variable to be set is in the substVarTarget
The value it should be set to is in substVarValue

The Script.....

```
"use strict";

var paramCount       = args.length - 1;
var substVarSource   = null;
var substVarTarget   = null;
var substVarValue    = null;
var substVarFound    = false;

// Param 1: name of the substVar that needs to be defaulted if it does not exist
substVarSource     = args[1].toUpperCase();

// Check if the source variable exists
for each (var definedVarName in ctx.getMap().keySet()) {
  if (definedVarName === substVarSource) {
    substVarFound = true;
  }
}

// Param 2: value to be used as default for the substVar if it does not exist
if (substVarFound) {
  substVarValue = ctx.getMap().get(substVarSource);
} else {
  substVarValue = args[2];
}

// Param 3 (optional): name of a substVar to which the value should be written
if (paramCount === 3) {
  substVarTarget   = args[3].toUpperCase();
} else {
  substVarTarget   = substVarSource;
}

// Set the target variable to be the value we determined.
ctx.getMap().put(substVarTarget, substVarValue);
```

The working script……

Just 34 lines.
Including empty lines
Including comments.

Nope!
It could be!!
But there's more........

First: I made it a bit more robust/fancy

- Introduction of functions:
  - Modularization and smaller code blocks to test
- Main code also goes in a function
- So main code becomes single function call
  - `mainCode();`

  We'll see later ("Custom Commands") why this is useful
- Parameter checks are built in:
  - Error message when less than 2 or more than 3 parameters supplied
  - Display syntax help when no parameters or parameter 'help' is supplied

So the script becomes

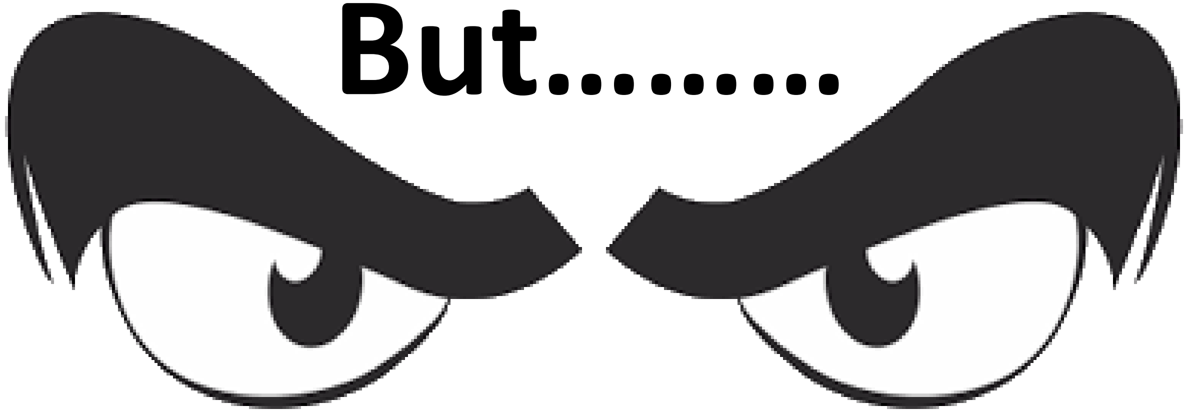
(Don't worry script files will be joined with the slides)

```
"use strict";

// Function writes text and an end-of-line to screen
function writeLine (line) {
  ctx.write (line + "\n");
}

// function displays an error message on screen
function errorMsg (message) {
  writeLine ("");
  writeLine ("##################################################");
  writeLine ("== ERROR == ");
  writeLine (message);
  writeLine ("");
  writeLine ("Call this script with parameter 'help' for usage");
  writeLine ("#################################################");
  writeLine ("");
}

// Prints the Help Text on screen
function displayHelp (scriptName) {
  writeLine (''                                                                           );
  writeLine ('==========================================================================='); 
  writeLine ('Usage of script ' + scriptName                                              );
  writeLine ('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~');
  writeLine (scriptName + ' help'                                                         );
  writeLine ('  => Display this help text'                                                );
  writeLine (''                                                                           );
  writeLine ('>>>>> Version with 2 parameters <<<<<'                                      );
  writeLine (scriptName + ' SubstVar DefaultValue'                                        );
  writeLine (''                                                                           );
  writeLine ('  => If the substitution variable named in the first parameter exists it is');
  writeLine ('     left unchanged.'                                                       );
  writeLine ('     If the substitution variable named in the first parameter does NOT'    );
  writeLine ('     exist it is created and given the value of the second parameter'       );
  writeLine (''                                                                           );
```

```
  writeLine ('>>>>> Version with 3 parameters <<<<<'                                      );
  writeLine (scriptName + ' SubstVar1 DefaultValue SubstVar2'                             );
  writeLine (''                                                                           );
  writeLine ('  => The substitution variable named in the third parameter is created if'  );
  writeLine ('     it does not yet exist, and given either the value of the substitution' );
  writeLine ('     variable named in the first parameter if it exists, or the value of'   );
  writeLine ('     the second parameter if the substitution variable named in the first'  );
  writeLine ('     parameter does not exist.'                                             );
  writeLine (''                                                                           );
  writeLine ('First and Third parameter (variable names) are case-INsensistive'           );
  writeLine (''                                                                           );
  writeLine ('Examples:'                                                                  );
  writeLine (scriptName + ' my_var my_default'                                            );
  writeLine ('   if variable &MY_VAR exists, nothing will happen'                         );
  writeLine ('   if variable &MY_VAR does not exist it will be created and given'         );
  writeLine ('   the vale "my_default"'                                                   );
  writeLine (''                                                                           );
  writeLine (scriptName + ' my_var my_default target_var'                                 );
  writeLine ('   if variable &MY_VAR exists, its value will be assigned to variable'      );
  writeLine ('   &TARGET_VAR, which will be created if it does not yet exist'             );
  writeLine ('   if variable &MY_VAR does not exist "my_default" will be assigned to'     );
  writeLine ('   variable &TARGET_VAR, which will be created if it does not yet exist'    );
  writeLine ('==========================================================================='); 
  writeLine (''                                                                           );
}

// Function returns true or false indicating if a substitution variable with the indicated name exists
function substVarExists (substVarName) {
  var substVarFound    = false;

  for each (var definedVarName in ctx.getMap().keySet()) {
    if (definedVarName === substVarName) {
      substVarFound = true;
    }
  }

  return substVarFound;
}
```

```
// Function contains the main functionality of the script
function mainCode () {
  // Main code

  var paramCount       = args.length - 1;
  var substVarSource   = null;
  var substVarTarget   = null;
  var substVarValue    = null;

  // Check and handle parameters
  if (paramCount > 3) {

    // More than 3 parameters supplied: syntax error
    errorMsg (args[0] + " - Too many parameters (" + paramCount + ")");

  } else if ((paramCount === 0) ||
             ((paramCount === 1) && (args[1].toLowerCase() === "help"))
            ) {


    // No parameters supplied, or 1 parameter which is "help": show help text
    displayHelp (args[0]);

  } else if (paramCount < 2) {

    // Fewer than 2 parameters supplied: syntax error
    errorMsg (args[0] + " - Not enough parameters (" + paramCount + ")");

  } else {

    // Either 2 or 3 parameters supplied
    // Param 1 containts the name of the substVar that needs to be defaulted if it does not exist
    substVarSource     =  args[1].toUpperCase();

    // Param 2 contains the value to be used as default for the substVar if it does not exist
    // If the source variable exists, use its value, else use the default value from param 2
    if (substVarExists (substVarSource)) {
      // The source variable exists, so instead of the provided default value, use its value for the target variable
      substVarValue = ctx.getMap().get(substVarSource);
    } else {

      substVarValue =  args[2];

    }

    // Param 3 (optional) contains the name of a substVar in which the value should be placed
    //         if not supplied, it should be placed in the variable that is named in the first parameter
    if (paramCount === 3) {

      // 3 parameters supplied, so the target variable is named in the third parameter
      substVarTarget   =  args[3].toUpperCase();

    } else {

      // 2 parameters supplied, so the target variable is named in the first parameter
      substVarTarget   = substVarSource;

    }

    // Set the target variable to be the value we determined.
    ctx.getMap().put(substVarTarget, substVarValue);

  }

}

// Execute the Main Code
mainCode();
```

Awesome!!!!!

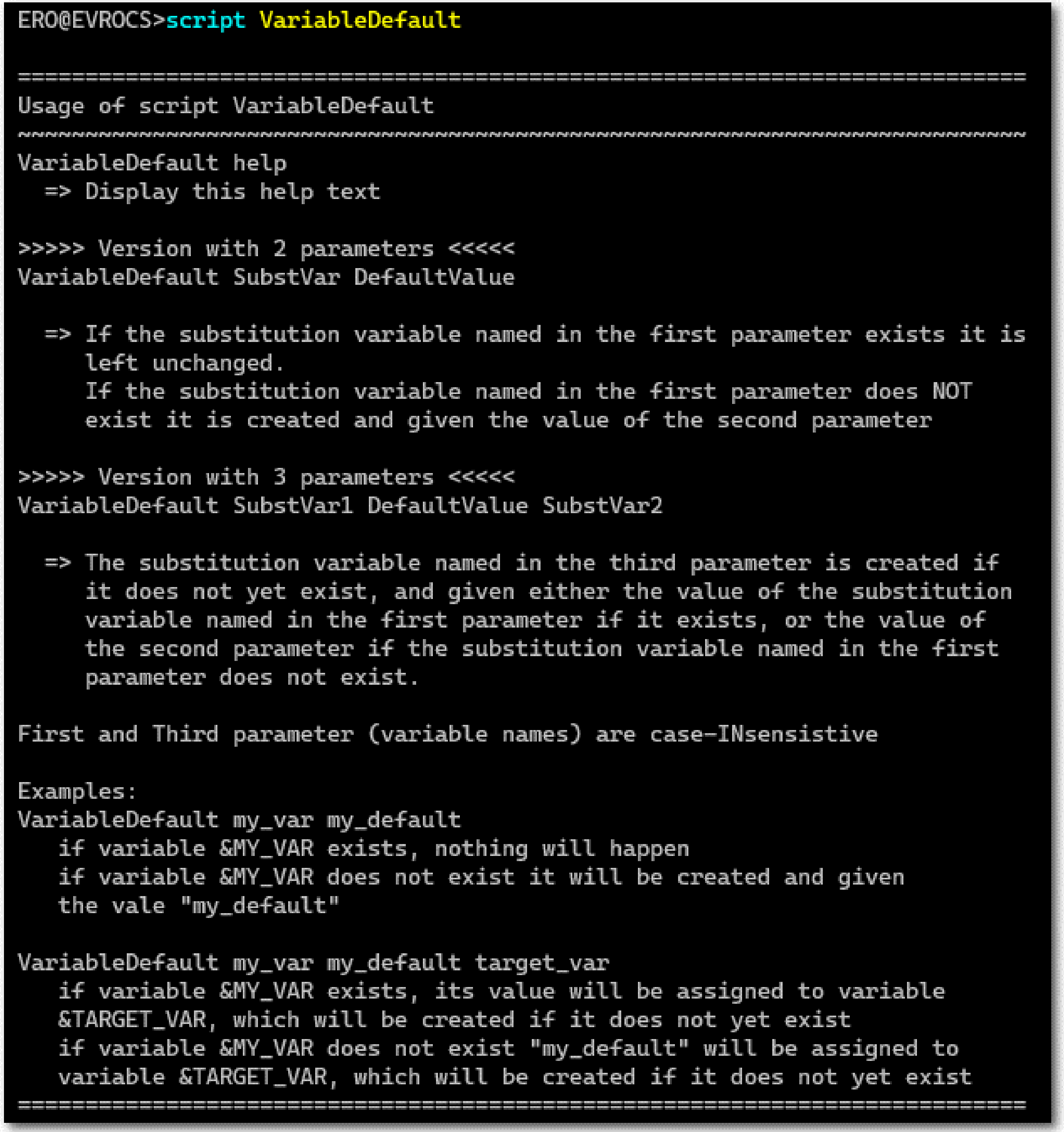


But.........

Does it work???

Run with 0 parameters:

script VariableDefault

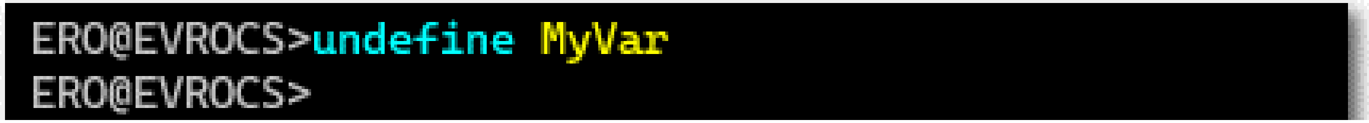


```
ERO@EVROCS>script VariableDefault

===========================================================================
Usage of script VariableDefault
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VariableDefault help
  => Display this help text

>>>>> Version with 2 parameters <<<<<
VariableDefault SubstVar DefaultValue

  => If the substitution variable named in the first parameter exists it is
     left unchanged.
     If the substitution variable named in the first parameter does NOT
     exist it is created and given the value of the second parameter

>>>>> Version with 3 parameters <<<<<
VariableDefault SubstVar1 DefaultValue SubstVar2

  => The substitution variable named in the third parameter is created if
     it does not yet exist, and given either the value of the substitution
     variable named in the first parameter if it exists, or the value of
     the second parameter if the substitution variable named in the first
     parameter does not exist.

First and Third parameter (variable names) are case-INsensistive

Examples:
VariableDefault my_var my_default
   if variable &MY_VAR exists, nothing will happen
   if variable &MY_VAR does not exist it will be created and given
   the vale "my_default"

VariableDefault my_var my_default target_var
   if variable &MY_VAR exists, its value will be assigned to variable
   &TARGET_VAR, which will be created if it does not yet exist
   if variable &MY_VAR does not exist "my_default" will be assigned to
   variable &TARGET_VAR, which will be created if it does not yet exist
===========================================================================
```

Run with 2 parameters:

script VariableDefault MyVar MyDefault

(If MyVar does not exist, create it with value 'MyDefault', otherwise leave it unchanged)

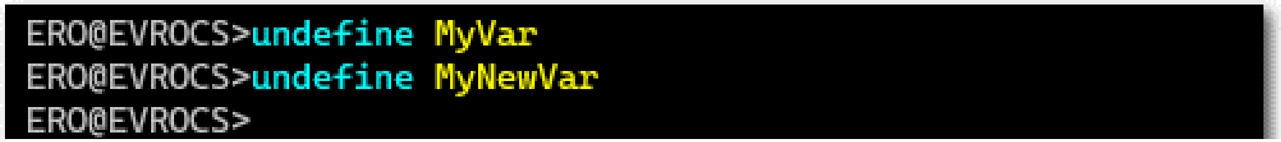


```
ERO@EVROCS>undefine MyVar
ERO@EVROCS>
```

Run with 3 parameters:

script VariableDefault MyVar MyDefault MyNewVar

(Create variable MyNewVar and assign it the value
of MyVar if it exists,
or the value 'MyDefault' if MyVar doesn't exist)

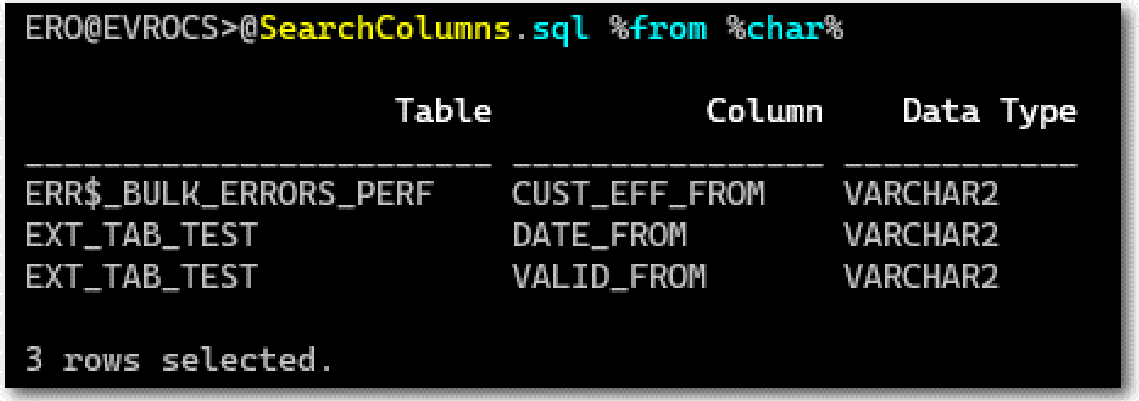


```
ERO@EVROCS>undefine MyVar
ERO@EVROCS>undefine MyNewVar
ERO@EVROCS>
```

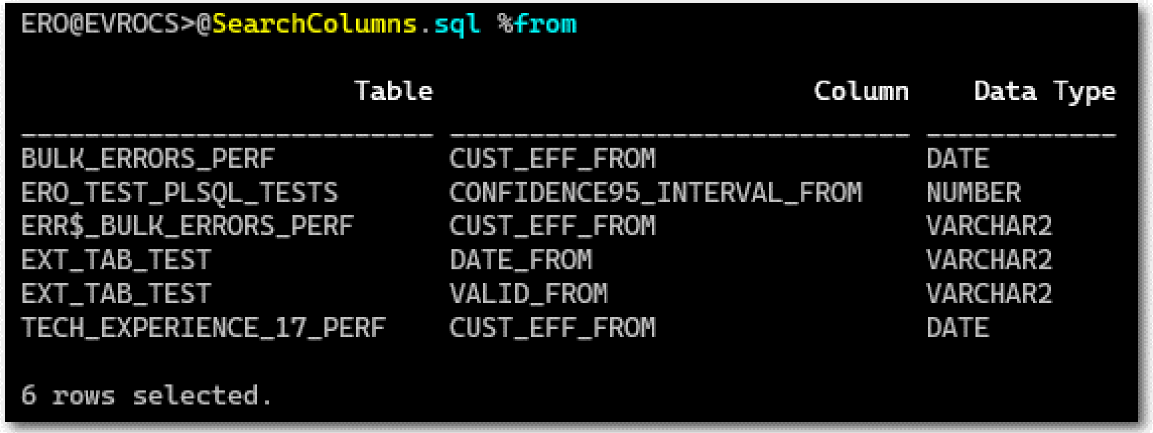
But the goal was to make sql-script **parameters** optional……

Change the sql script into:

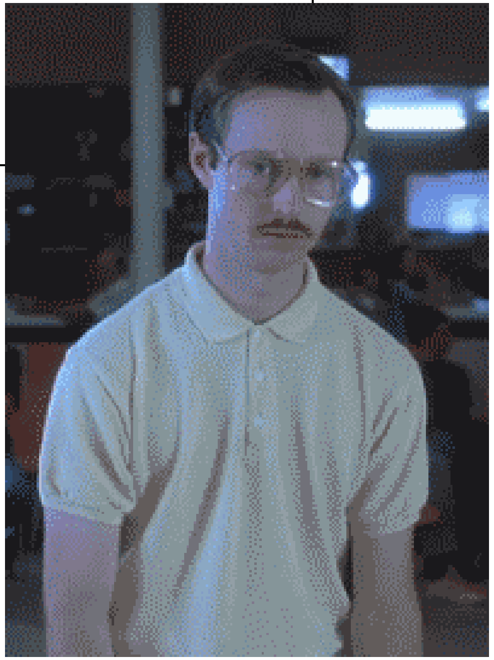
```
script VariableDefault 1 %
script VariableDefault 2 %

select atc.table_name    "Table"
,      atc.column_name   "Column"
,      atc.data_type     "Data Type"
from   user_tab_columns    atc
where  atc.column_name like upper('&1')
  and  atc.data_type   like upper('&2')
order by 1, 2, 3
;

undef 1
undef 2
```



```
ERO@EVROCS>@SearchColumns.sql %from %char%

                     Table             Column    Data Type
------------------------ ----------------- ------------
ERR$_BULK_ERRORS_PERF    CUST_EFF_FROM     VARCHAR2
EXT_TAB_TEST             DATE_FROM         VARCHAR2
EXT_TAB_TEST             VALID_FROM        VARCHAR2

3 rows selected.
```



```
ERO@EVROCS>@SearchColumns.sql %from

                     Table                         Column    Data Type
------------------------ ------------------------------ ------------
BULK_ERRORS_PERF         CUST_EFF_FROM                  DATE
ERO_TEST_PLSQL_TESTS     CONFIDENCE95_INTERVAL_FROM     NUMBER
ERR$_BULK_ERRORS_PERF    CUST_EFF_FROM                  VARCHAR2
EXT_TAB_TEST             DATE_FROM                      VARCHAR2
EXT_TAB_TEST             VALID_FROM                     VARCHAR2
TECH_EXPERIENCE_17_PERF  CUST_EFF_FROM                  DATE

6 rows selected.
```



After 27 years of
waiting for this……

Happy?

Well, not completely ….

I wanted a command

```
VariableDefault 1 %
```

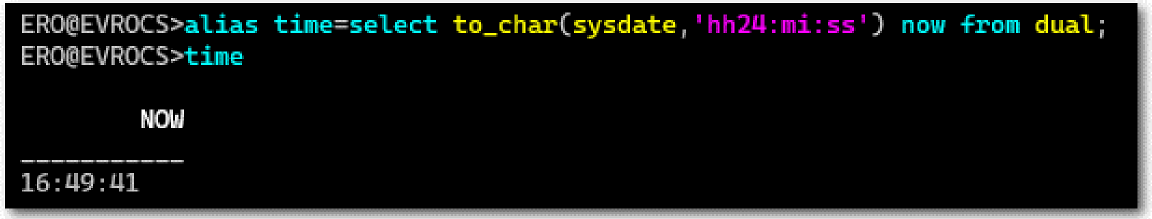
Not a sentence

```
script VariableDefault 1 %
```

Alias ???

For example

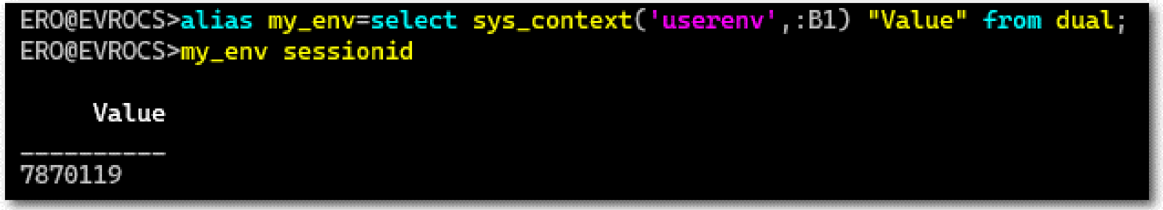
```
alias time=select to_char(sysdate,'hh24:mi:ss') now from dual;
```



```
ERO@EVROCS>alias time=select to_char(sysdate,'hh24:mi:ss') now from dual;
ERO@EVROCS>time

        NOW
-----------
16:49:41
```

Also with binds as parameters

```
alias my_env=select sys_context('userenv',:B1) "Value" from dual;
```

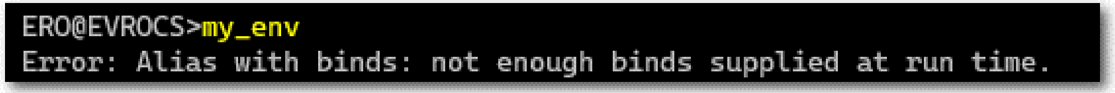


```
ERO@EVROCS>alias my_env=select sys_context('userenv',:B1) "Value" from dual;
ERO@EVROCS>my_env sessionid

     Value
----------
7870119
```

So we could try….

```
alias variabledefault=script d:\Scripts\VariableDefault.js :1 :2 :3;
```

But Alias variables are **not optional** ☹



```
ERO@EVROCS>my_env
Error: Alias with binds: not enough binds supplied at run time.
```

Solution: Custom Commands!!!

We can register our JavaScript as a new command in SQLcl

Custom Commands!!!

The steps for this are:

- Create a function expression for what the command has to do

- Create a SQLcl Command Listener with that functionality

  Java Class:

  oracle.dbtools.raptor.newscriptrunner.CommandRegistry

- Register this listener with the SQLcl Command Registry

  Java Class:

  oracle.dbtools.raptor.newscriptrunner.CommandListener

```
// *** ^^^ The code of our script is above this ^^^ ***
mainCode();

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");
```

First we have to get a handle to Java types

- CommandRegistry
- CommandListener

Just use "Java.Type" and assign
the result to a variable

```
// *** ^^^ The code of our script is above this ^^^ ***
mainCode();

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");
```

```
var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

Near the end we will extend the
Command Listener by adding the
- handleEvent (The command handling)
- beginEvent (a kind of 'pre statement')
- endEvent (a kind of 'post statement')

And at the end we register the listener

```
// *** ^^^ The code of our script is above this ^^^ ***
mainCode();

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
```

handleEvent, beginEvent and endEvent are methods.
We declare them as function expressions.

beginEvent and endEvent are not needed for this functionality, so these will be 'empty' functions

Example of the use of beginEvent:
Autocorrect script by Kris Rice
https://github.com/oracle/oracle-db-tools/blob/master/sqlcl/examples/autocorrect.js

```
var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

```
// *** ^^^ The code of our script is above this ^^^ ***
mainCode();

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
var custom_handle = function (conn,ctx,cmd) {







                    }

var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

Now the real command handler:
We need to move the call of the main function to this handler (without the brackets)

```
// *** ^^^ The code of our script is above this ^^^ ***
// mainCode();  moved to custom_handle

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
var custom_handle = function (conn,ctx,cmd) {
                      var stmntArgs = splitCommandLine (cmd.getSql());


                        mainCode(stmntArgs);




                    }

var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

Array args shows the arguments of the call of the script, not the command line arguments

So we need to get the command line with
cmd.getSql()

And write a function to split this into command and parameters

And pass the array to the mainCode function

```
// *** ^^^ The code of our script is above this ^^^ ***
// mainCode();  moved to custom_handle

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
var custom_handle = function (conn,ctx,cmd) {
                      var stmntArgs = splitCommandLine (cmd.getSql());

                      if ( stmntArgs[0].toLowerCase() === "variabledefault") {
                        mainCode(stmntArgs);

                      }

                    }

var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

Currently this handler will execute for ***anything*** we type on the command line.

So we need to check if the command that is typed is the one this handler should act upon

```
// *** ^^^ The code of our script is above this ^^^ ***
// mainCode();  moved to custom_handle

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
var custom_handle = function (conn,ctx,cmd) {
                      var stmntArgs = splitCommandLine (cmd.getSql());

                      if ( stmntArgs[0].toLowerCase() === "variabledefault") {
                        mainCode(stmntArgs);
                        return true;
                      }
                      return false;
                    }

var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

By returning either true or false the function can indicate whether the command has been handled

```
// *** ^^^ The code of our script is above this ^^^ ***
// mainCode();  moved to custom_handle

var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");
var CommandListener = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener");

var custom_begin  = function (conn,ctx,cmd) {}
var custom_end    = function (conn,ctx,cmd) {}
var custom_handle = function (conn,ctx,cmd) {
                      var stmntArgs = splitCommandLine (cmd.getSql());

                      if ( stmntArgs[0].toLowerCase() === "variabledefault") {
                        mainCode(stmntArgs);
                        return true;
                      }
                      return false;
                    }

var myCustomCommand = Java.extend(CommandListener, {
                                    handleEvent: custom_handle ,
                                    beginEvent:  custom_begin  ,
                                    endEvent:    custom_end
                                  }
                                 );

CommandRegistry.addForAllStmtsListener(myCustomCommand.class);
```

So this, plus the splitCommandLine function, is what we have to add to change the script in a command registration script.

Put all this in a file "`VariableDefault_Cmd.js`"

And put

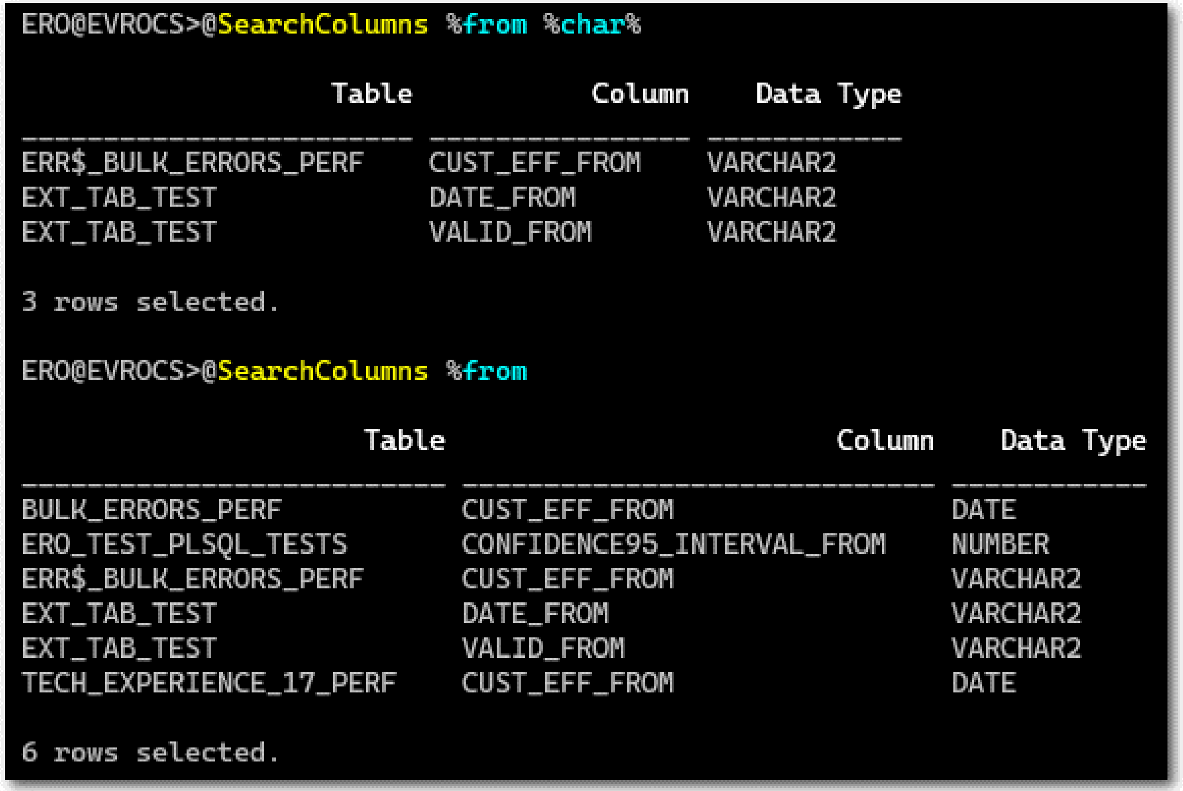
```
script d:\Scripts\VariableDefault_Cmd.js
```

In your login.sql script

And you will always have command `VariableDefault` at your disposal

Change the sql script into:

```
VariableDefault 1 %
VariableDefault 2 %

select atc.table_name     "Table"
,      atc.column_name    "Column"
,      atc.data_type      "Data Type"
from   user_tab_columns    atc
where  atc.column_name like upper('&1')
  and  atc.data_type   like upper('&2')
order by 1, 2, 3
;

undef 1
undef 2
```



```
ERO@EVROCS>@SearchColumns %from %char%

                   Table           Column    Data Type
------------------------ ---------------- ------------
ERR$_BULK_ERRORS_PERF    CUST_EFF_FROM    VARCHAR2
EXT_TAB_TEST             DATE_FROM        VARCHAR2
EXT_TAB_TEST             VALID_FROM       VARCHAR2

3 rows selected.

ERO@EVROCS>@SearchColumns %from

                     Table                        Column    Data Type
-------------------------- ----------------------------- ------------
BULK_ERRORS_PERF           CUST_EFF_FROM                 DATE
ERO_TEST_PLSQL_TESTS       CONFIDENCE95_INTERVAL_FROM    NUMBER
ERR$_BULK_ERRORS_PERF      CUST_EFF_FROM                 VARCHAR2
EXT_TAB_TEST               DATE_FROM                     VARCHAR2
EXT_TAB_TEST               VALID_FROM                    VARCHAR2
TECH_EXPERIENCE_17_PERF    CUST_EFF_FROM                 DATE

6 rows selected.
```

Unregistering a custom command

Currently the command is simply added

So what if we run the script again?
- New login in same SQLcl session
- Run the script with changed code

It gets added again
And again
And again
And again

You can't run changed code (without restarting SQLcl)
Because the old version will run and
report the command handled

So we need to make SQLcl 'forget' the old version

Please welcome……

Philipp Salvisberg

Came up with a trick to unregister a command

See:
https://github.com/Trivadis/plsql-formatter-settings/blob/main/sqlcl/format.js

Steps for unregistering

1. Make sure we can identify a listener by the name of the command
   On registration override toString method of listener: make it return the name

2. Create an unregister function that will
   1. Retrieve a list of current listeners
   2. Run a command that removes all listeners
   3. Retrieve a list of listeners that could not be removed (if any)
   4. Re-register all the listeners, except
      - The one we want to unregister
      - The ones that could not be removed

3. Execute the unregister function just before registering a custom command

The unregister function

```
function unregisterCommand () {
  var Collectors      = Java.type("java.util.stream.Collectors");
  var SQLCommand      = Java.type("oracle.dbtools.raptor.newscriptrunner.SQLCommand");
  var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");

  var listeners = CommandRegistry.getListeners (ctx.getBaseConnection(), ctx)
                    .get(SQLCommand.StmtSubType.G_S_FORALLSTMTS_STMTSUBTYPE);

  CommandRegistry.removeListener(SQLCommand.StmtSubType.G_S_FORALLSTMTS_STMTSUBTYPE);
  CommandRegistry.clearCaches(null, ctx);
  CommandRegistry.clearCaches(ctx.getBaseConnection(), ctx);

  var remainingListeners = CommandRegistry.getListeners(ctx.getBaseConnection(), ctx)
                           .get(SQLCommand.StmtSubType.G_S_FORALLSTMTS_STMTSUBTYPE)
                           .stream().map(function(l) l.getClass())
                           .collect(Collectors.toSet());

  for (var i in listeners) {
      if (!listeners.get(i).toString().equals("VariableDefault") &&
          !remainingListeners.contains(listeners.get(i).getClass())) {
          CommandRegistry.addForAllStmtsListener(listeners.get(i).getClass());
      }
  }
}
```

Thank you,
Philipp Salvisberg

But ?!?

a 1-line "hello world" script

Takes about 50 lines of extra code
for registering/unregistering as custom command

Library to the rescue

See: https://www.evrocs.nl/your-library-is-your-paradise/

```
"use strict";


// Load Library
var libraryPath = java.lang.System.getenv("SQLCL_JS_LIB")
          .replace(/\\/g, "/").replace(/\/?$/, "/");
load (libraryPath + "ELib_registration.js");


[... THE PART OF THE SCRIPT WHERE THE REAL WORK IS DONE ...]


// Execute or register the main code
registration.runCommand (mainCode);
```

Get the location of your libraries from an environment variable

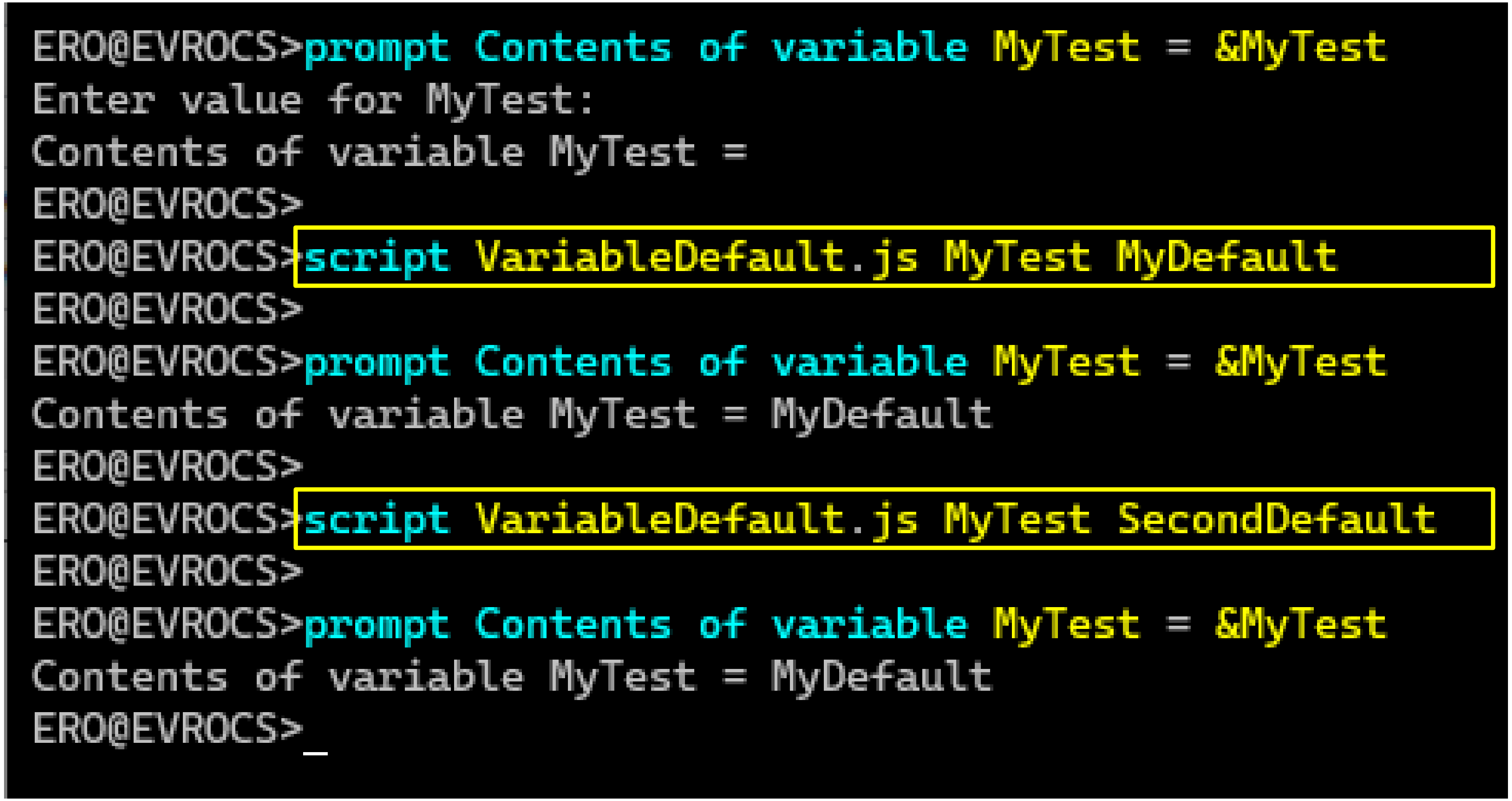
Load the library with hardcoded path or use an environment variable

Instead of executing "mainCode"
Pass the function expression (so without brackets) to the runCommand function

All it needs to take is 2 or 3 extra lines of code

Result - 1

You can still use the script as a regular script, without registering

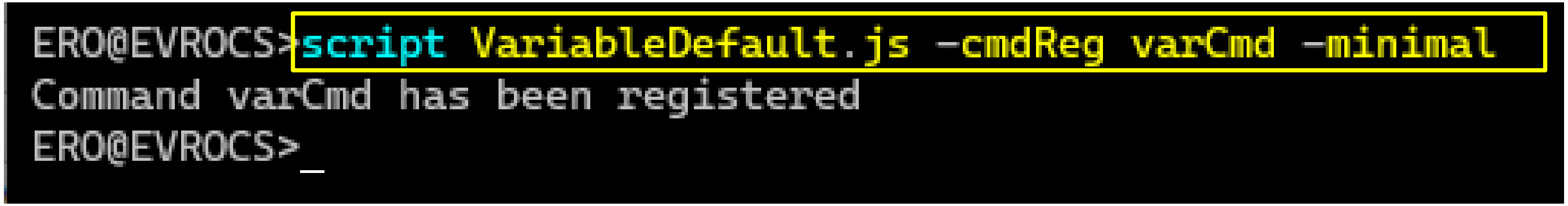


```
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Enter value for MyTest:
Contents of variable MyTest =
ERO@EVROCS>
ERO@EVROCS>script VariableDefault.js MyTest MyDefault
ERO@EVROCS>
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Contents of variable MyTest = MyDefault
ERO@EVROCS>
ERO@EVROCS>script VariableDefault.js MyTest SecondDefault
ERO@EVROCS>
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Contents of variable MyTest = MyDefault
ERO@EVROCS>_
```

But also……….

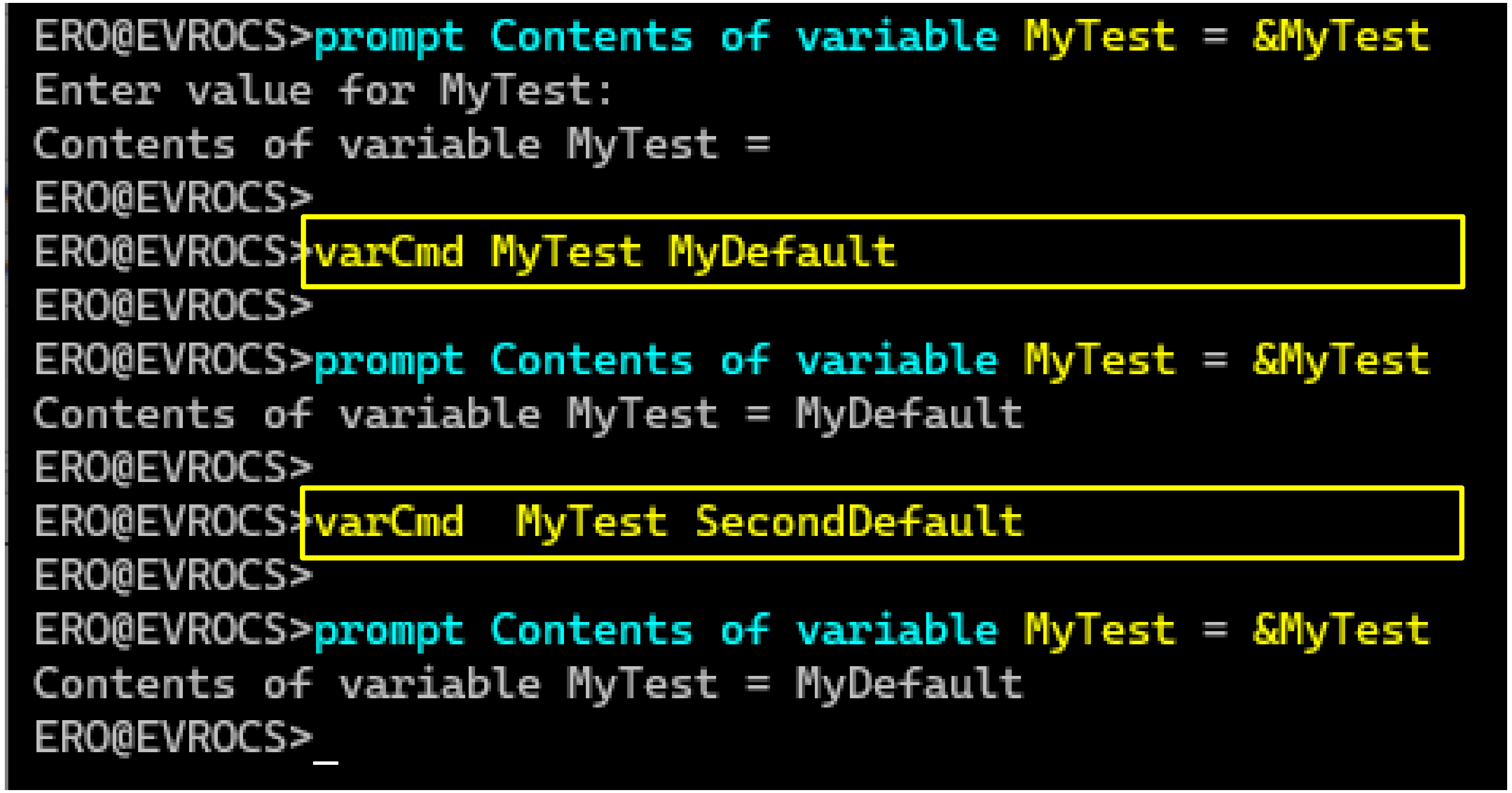
Result - 2

You can register it as a custom command using special parameters



```
ERO@EVROCS>script VariableDefault.js -cmdReg varCmd -minimal
Command varCmd has been registered
ERO@EVROCS>_
```

And then the command can be used



```
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Enter value for MyTest:
Contents of variable MyTest =
ERO@EVROCS>
ERO@EVROCS>varCmd MyTest MyDefault
ERO@EVROCS>
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Contents of variable MyTest = MyDefault
ERO@EVROCS>
ERO@EVROCS>varCmd  MyTest SecondDefault
ERO@EVROCS>
ERO@EVROCS>prompt Contents of variable MyTest = &MyTest
Contents of variable MyTest = MyDefault
ERO@EVROCS>_
```

"Stupid questions do exist.
But it takes a lot more time and energy to correct a stupid mistake than it takes to answer a stupid question, so please ask your stupid questions."

a wise teacher who taught me more than just physics

Thanks !!