
BLOX FEST
Infoblox

Customizing NetMRI Network Changes with Configuration Command Scripting (CCS)

Dave Signori

Senior Director, Product Management

Network Insight and NetMRI

Sif Baksh

Systems Engineer

John Belamaric

Software Architect

Cloud and Network Automation

Agenda

- Why script and what scripting options are available in NetMRI
- CCS Script Sections
- Variables
- Lists
- Triggers
- Filters
- Operators
- Other CCS Commands and Statements
- Viewing, Adding, and Running CCS Scripts
- Community Site and TAB

Why Use Scripting?

- Scripts:
  - Automate changes to infrastructure devices
  - Ensure consistent changes
  - Can be scheduled to run at specific times
  - Can be triggered to respond to specific events

- Scripting reduces the risk of errors being made during execution
  - No fat fingering
  - No instant mistakes

- Other features to consider before you script …
  - Config Templates
  - Config Search
  - Rules

Scripting Languages Supported

- Focus of this tutorial:
  - CCS (Change Control Scripting)
    - Is a proprietary, high-level scripting language designed for network admins
    - Primary goal is to convert device-specific commands into repeatable tasks

- Also supported:
  - Perl:
    - Is a general purpose, high-level scripting language
    - It has a large collection of 3rd party libraries and modules
    - It has powerful pattern matching and text processing features

CCS Sections

Section	Mandatory
Script-Filter	X
Action	X
Action-Filter	
Action-Commands	X
Script-Variables	
Output-Triggers	
Trigger	
Trigger-Filter	
Trigger-Variables	
Trigger-Template	
Trigger-Commands	

CCS Sections - Sample

Section	Mandatory
Script-Filter	X
Action	X
Action-Filter	
Action-Commands	X
Script-Variables	
Output-Triggers	
Trigger	
Trigger-Filter	
Trigger-Variables	
Trigger-Template	
Trigger-Commands	

Using these sections

Sample: Reset Password

- Reset Password commands can be written as a CCS script as follows:

```
Script-Filter:
        $Vendor eq "Cisco" and $sysDescr like /IOS/

Script-Variables:
        $username       word          "User Name"
        $password       password      "New Password"

# # # # # # # # # #
# This is a comment

Action:
        Set IOS User Password

Action-Commands:
        config terminal
        username $username password 0 $password
        exit
        write memory
```

Only run on Cisco IOS devices

Prompt user to type in an username and password

Execute commands

Sample: Reset Password

- The mandatory sections are highlighted in **Red**
- The optional sections are highlighted in **Blue**

```
Script-Filter:
        $Vendor eq "Cisco" and $sysDescr like /IOS/
```

Required

```
Script-Variables:
        $username       word          "User Name"
        $password       password      "New Password"

# # # # # # # # # #
# This is a comment
```

Optional

```
Action:
        Set IOS User Password
```

Required

```
Action-Commands:
        config terminal
        username $username password 0 $password
        exit
        write memory
```

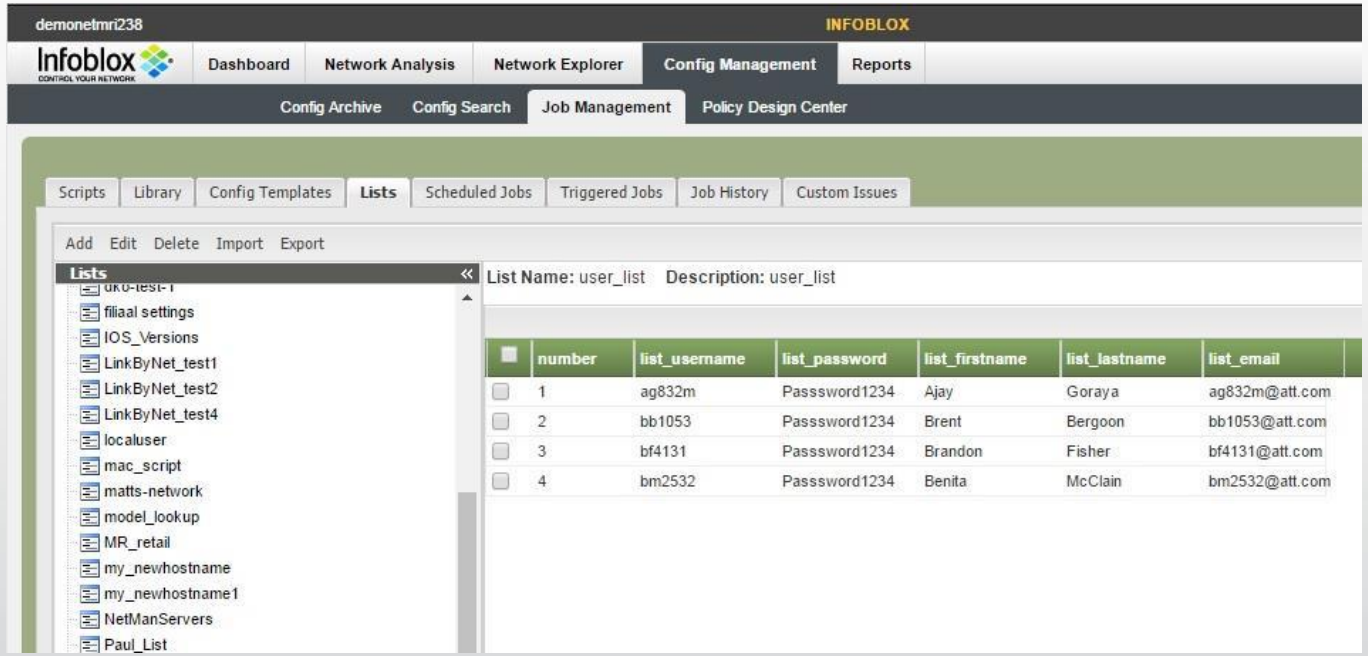
Required

Variables

- Temporary holding place
- Declared using a dollar sign ($) e.g. **$username**
- The value can change during the life of the script
- Are global in scope. Available in all sections.
- Are updated using the **SET:** directive
- Three types of variables:
  - Script-Variables: prompt the user at runtime
  - Trigger-Variables: populated from output of a previous section
  - Well-Known variables: bundled with NetMRI like **$Vendor, $Model**, and **$Version**
- When declaring a variable, you must specify a data type
  - i.e. number (integer), text (string), true/false (boolean)
  - Regex
    - **$one_digit /^[1-9]$/**
  - Predefined Regex
    - **word, ipaddress, url, phoneno, zipcode, email**

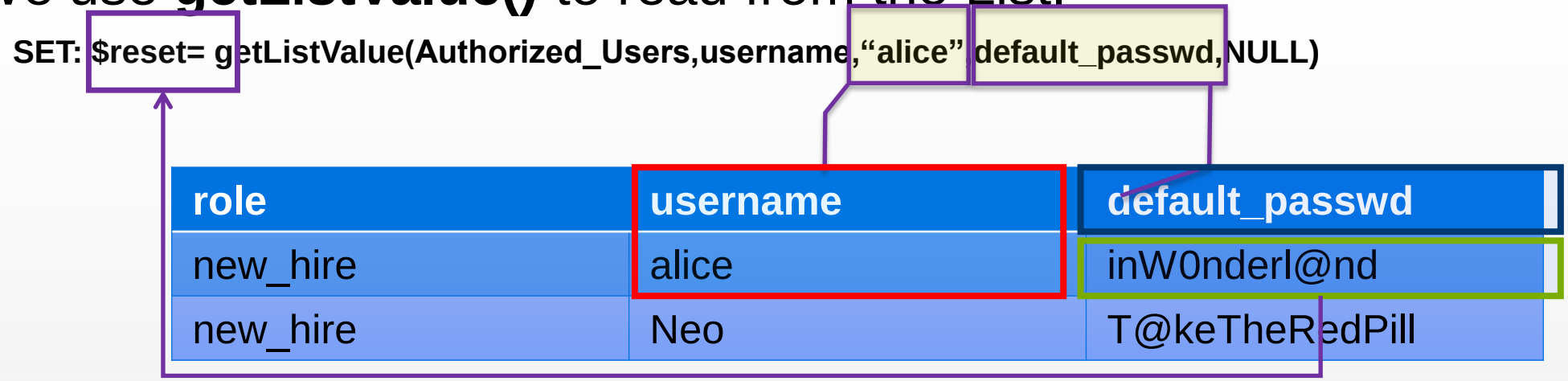
Lists

- CCS Scripts can reference a built-in file called a List. Think of it as a light-weight database.
- Use the command getListValue to get the content of the List.
- Lists can be created (or imported) under: Config Management -> Job Management -> Lists



Reading from a List

- We use **getListValue()** to read from the List.



- Look up from the List **Authorized_Users**, where username equals “**alice**”, and return the corresponding password.
- If the user “**alice**” cannot be found, return NULL as the password.
- Based on the List we defined, we expect to get the password “**inW0nderl@nd**” for the user “**alice**”.

Sample with List: Reset Password

Script-Filter:

$Vendor eq "Cisco" and $sysDescr like /IOS/

Script-Variables:

$user word “Enter username to reset password"

Action:

Get default password of a single user from List Authorized_Users

Action-Commands:

SET: $reset = getListValue(Authorized_Users,username,$user,default_passwd, “Nope”)

Action-Commands: {$reset ne “Nope”}

config terminal
username $user password 0 $reset
end
wr mem

Triggers

- Mini script that only handles a single iteration of a loop
- Action section runs, produces output, send it to the Trigger
- Trigger section runs, reading in information as input, uses Trigger-Template to extract what's necessary.
- Runs commands in Trigger-Commands using the information extracted

CCS Sections – Trigger Sample

Section	Mandatory
Script-Filter	X
Action	X
Action-Filter	
Action-Commands	X
Script-Variables	
Output-Triggers	
Trigger	
Trigger-Filter	
Trigger-Variables	
Trigger-Template	
Trigger-Commands	

Using these sections

Triggers

- Action-Commands generates a list of output, and calls the Trigger to process the entire list, one item at a time.
- Trigger-Commands defines what specific commands are run against each item of the list.
- Take the familiar CLI commands **sh run | inc username** for example, **“sh run”** is like the Action-Commands, and sends the output to **“inc username”** to filter out just the usernames.

```
NAM-HQ-4#sh run | inc username
username admin privilege 15 password 0 Goldil0cks
username alice password 0 Passw0rd123
username Neo password 0 Passw0rd321
NAM-HQ-4#
```

Triggers

```
Script-Filter:
        $Vendor eq "Cisco" and $sysDescr like /IOS/
Script-Variables:
        $user    word     “Enter username to reset password”
Action:
        Get existing usernames
Action-Commands:
        show run | inc username
Output-Triggers:
        Update Password
```

Action, sends output to “Update Password” Trigger

Trigger-Template extracts actual username from “show run “ inc username”

Execute commands on the username

```
##########

Trigger:
        Update Password
Trigger-Variables:
        $existing_user      word
Trigger-Template:
        username [[$existing_user]].+password 0.+

Trigger-Commands:
        SET $reset = getListValue(Authorized_Users,username,$existing_user,default_passwd,NULL)
        conf t
        username $existing_user password 0 $reset
        end
        wr mem
```

BLOX FEST

Triggers

```
Script-Filter:
        $Vendor eq "Cisco" and $sysDescr like /IOS/
Script-Variables:
        $user    word     “Enter username to reset password”
Action:
        Get existing usernames
Action-Commands:
        show run | inc username
Output-Triggers:
        Update Password

##########

Trigger:
        Update Password
Trigger-Variables:
        $existing_user      word
Trigger-Template:
        username [[$existing_user]].+password 0.+

Trigger-Commands:
        SET $reset = getListValue(Authorized_Users,username,$existing_user,default_passwd,NULL)

Trigger-Commands: {$reset ne NULL}
        conf t
        username $existing_user password 0 $reset
        end
        wr mem
```

Action, sends output to “Update Password” Trigger

Trigger-Template extracts actual username from “show run “ inc username”

Execute commands on the username

BLOX FEST

Filters

- The {} is used as a filter to restrict execution.
- Action-Commands and Trigger-Commands can be extended to specify optional filter criteria, restricting execution to cases where specific conditions exist.

Example 1:

Action-Commands: { $Vendor eq “Cisco” }

show interfaces

Example 2:

Trigger-Commands: { $existing_user eq “Neo” }

config t

username $existing_user password 0 $reset

Filter Sections

- Action-Filter and Trigger-Filter sections can also be used to apply the filter to the entire Action or Trigger section.

Example 1:

Action-Filter:

$Vendor eq “Cisco” show interfaces

Example 2:

Trigger-Filter:

$existing_user eq “Neo”

Filters

- Filters grant more flexibility and help optimize your code:
  - They work like an if-then, unless, or case statement
  - They can be written in two ways:

```
Action:
    Show Running Config
Action-Filter:
    $Vendor eq “Cisco”
Action-Commands:
    show run | inc username
```

```
Action:
    Update Logging Destination
Action-Commands {$sysDescr like /IOS/}
    logging 205.201.59.69
Action-Commands {$sysDescr like /NX-OS/}
    logging server 205.201.59.69
Action-Commands {$sysDescr like /SRX/}
    set system syslog host 205.201.59.69 any any
```

Operators

- There are several ways to compare two variables
- Here are four common approaches:

Expression	eq	ne	in	like
Definition	A equal to B	A not equal to B	A in list that follows	A is similar to B
Example	Trigger-Commands: {$user eq “Neo”}	Action-Commands: {$user ne “admin”}	Action-Commands: {$user in [“Neo”,”alice”,”Morpheus”]}	Trigger-Commands: {$user like /lic/}
Notes	Will execute if user is “Neo”	Will execute if user is not “admin”	Will execute if user is Neo, alice, or Morpheus	Will execute if the username contains “lic”: either “alice” or “click”

BLOX FEST

Assignment vs. Comparison

- The **=** symbol is NOT the same as the **eq** comparison operator.
- Assignment:
  - When we use =, we are taking whatever is B and copying it to A:

  Example: **SET: $A = “no”**

- Comparison:
  - When we use eq, we are making an evaluation that ends in true or false, by comparing A to B:

  Example: **{ $A eq “no” }**

Putting It All Together - State Variable Sample

```
Script-Filter:
        $Vendor eq "Cisco" and $sysDescr like /IOS/
Script-Variables:
        $role     word     “Enter the role, such as new_hire”
Action:
        Get existing usernames
Action-Commands:
        show run | inc username
        SET: $ChangesMade = “no”
Output-Triggers:
        Delete user
# # # # # # # # # # #
Trigger:
        Delete user
Trigger-Variables:
        $existing_user   word
Trigger-Filter:
        $existing_user ne “admin”
Trigger-Template:
        username [[$existing_user]].+password 0.+
Trigger-Commands:
        SET: $user_role = getListValue(Authorized_Users,username,$existing_user,role,NULL)
Trigger-Commands: { $ChangesMade eq “no” and $user_role eq $role }
        conf t
        SET: ChangesMade = “yes”
```

Initialize $ChangesMade to “no”

Set $ChangesMade to “yes”

State Variable Sample (continued)

```
Trigger-Commands: { $user_role eq $role }
        no username $existing_user

# # # # # # # # # # #

Action:
        Write to memory if changes were made
Action-Commands: { $ChangesMade eq “yes” }
        end
        wr mem
```

Iterations

Only execute if $ChangesMade is “yes”

Other CCS Commands

- Table below summarizes other CCS commands. For more details, refer to the *Network Automation CSS Scripting Guide*.

Command	Description
DEBUG	Provides a mechanism to check whether or not a loop is entered or a statement is executed. Place DEBUG statement in front of the operation to check, and a debug icon will appear next to it if it would be executed.
GET-CONFIGS	Provides a mechanism to ensure Network Automation has the most up-to-date configurations from the network devices.
LOG-INFO LOG-WARNING LOG-ERROR LOG-DEBUG	Generates log messages to be sent to the appropriate log facilities, and goes into the standard Network Automation logging.
PRINT	Allows the printing of simple text strings (similar to the C “printf” command) and the printing of values within variables in CCS scripts to output text files.
SKIPERROR	Turns off error handling for script attributes when an error may appear from the acted-upon device, potentially preventing further job execution.
SLEEP	Pauses script execution for a specified number of seconds.
EXPR	Evaluate expressions, discussed next.

Comments

- Anything after the # symbol is ignored by the script
- Use this to add human-readable comments so people reading the script (could be yourself later) understand the code.
- For example:

```
Action-Commands:
    # get current list of network interfaces to decide
    # which ones to disable
    # - Neo 04/25/2015
```

DEBUG: Statement

- Placing the DEBUG: directive in front of any statement will cause CCS to only print it out in the session log, but not execute the command.
- For example:

  Action-Commands:

  DEBUG: conf t

  DEBUG: no username $username

  DEBUG: end

  DEBUG: wr mem

Operators

Operator	Description
A | B	A if it is neither null nor 0, otherwise B
A & B	A if neither argument is null or 0, otherwise 0
A < B	A is less than B
A <= B	A is less than or equal to B
A = B	A is equal to B
A != B	A is not equal to B
A >= B	A is greater than or equal to B
A > B	A is greater than B
A + B	Arithmetic sum of A and B
A – B	Arithmetic difference of A and B
A * B	Arithmetic product of A and B
A / B	Arithmetic quotient of A divided by B
A % B	Arithmetic remainder of A divided by B

BLOX FEST

EXPR Command

- **EXPR** performs more advanced comparison and evaluations
- For example, increase $A by 1 (useful for iterating)

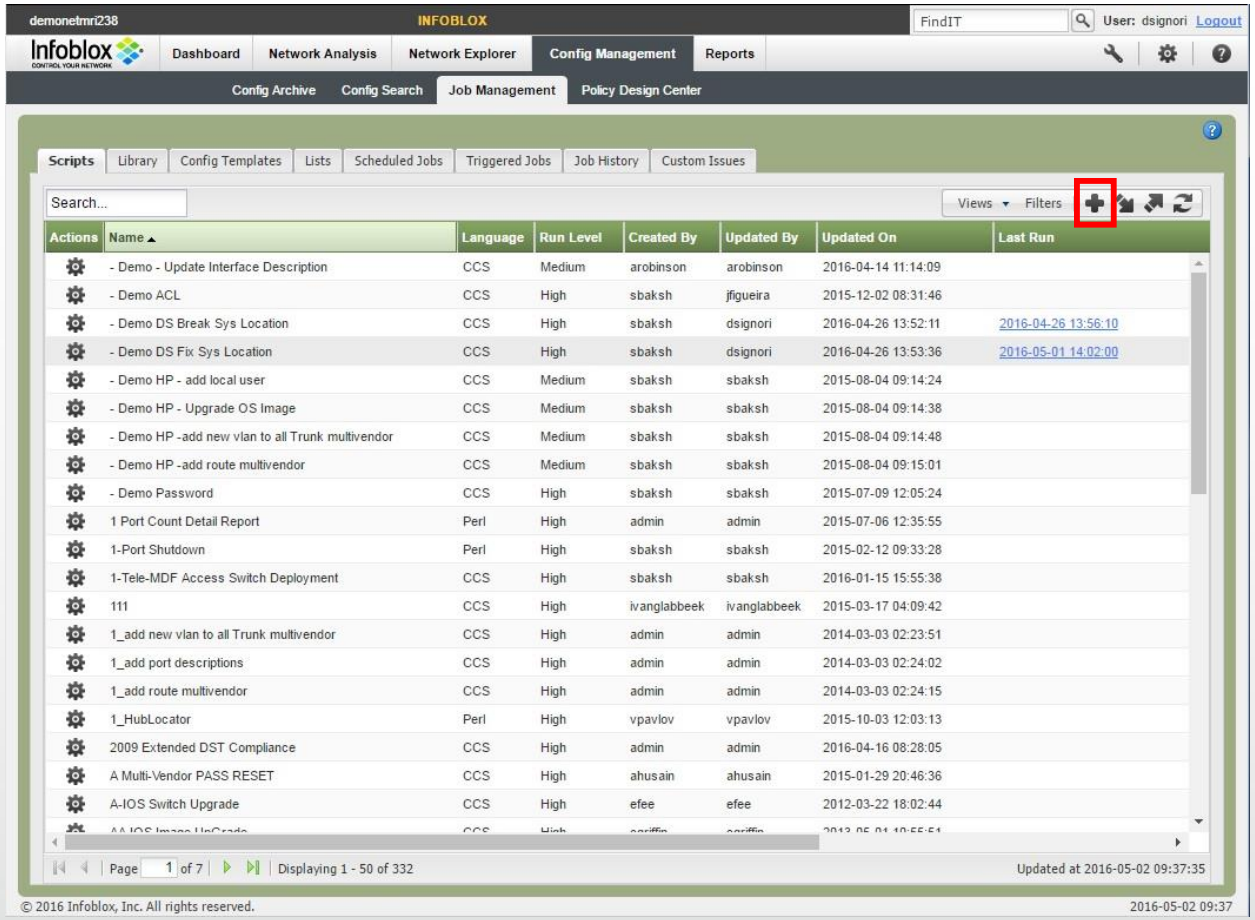
  EXPR: $A = $A + 1
- Or compute the product of $A times $B

  EXPR: $A = $A * $B

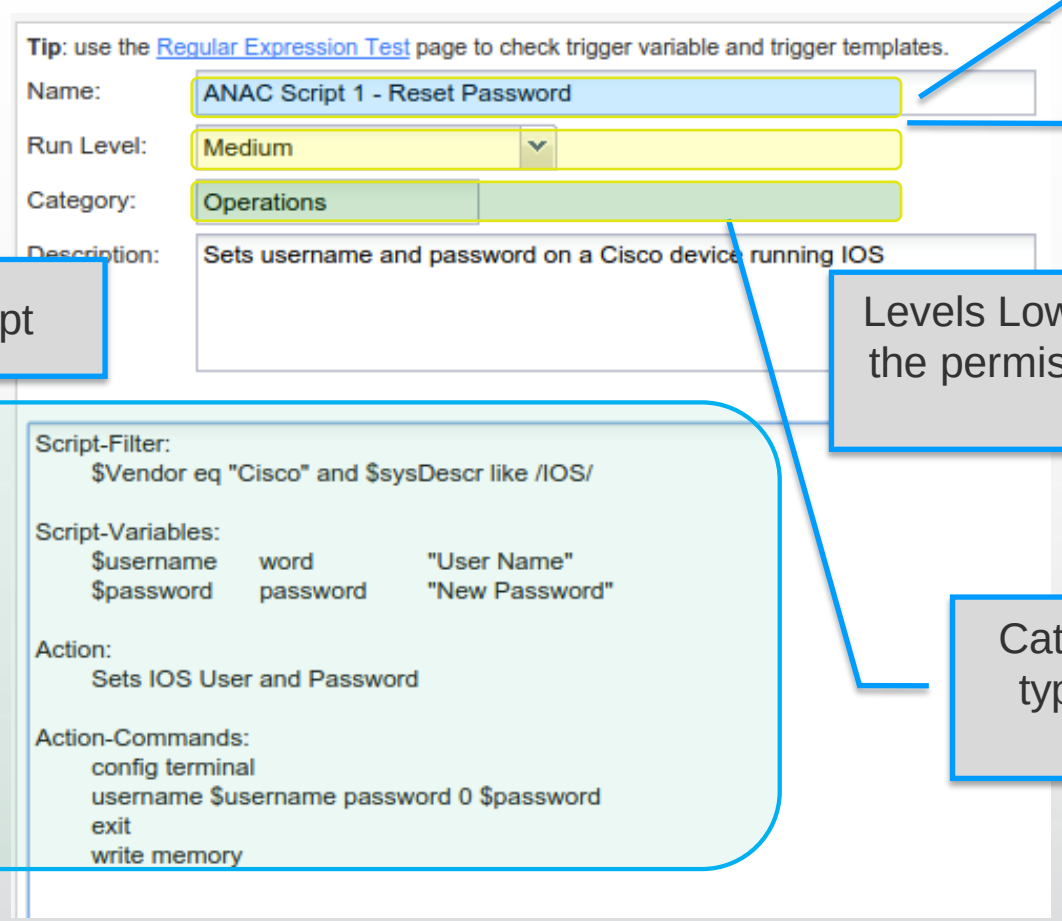
BLOX FEST

Viewing, Adding, and Running CCS Scripts

- Config Management -> Job Management -> Scripts
- Click Add to create



Adding New CCS Script



Name of script

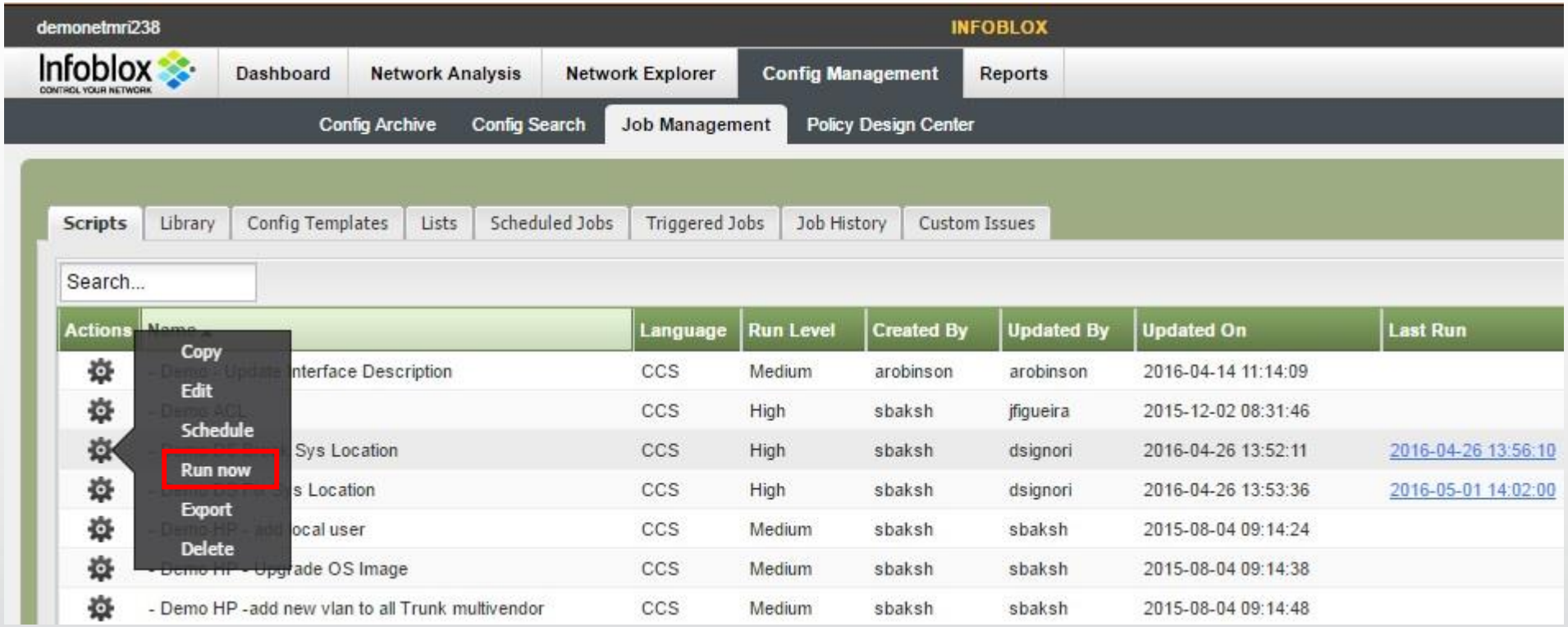
Body of CCS script

Levels Low, Medium, and High, indicate the permission level needed to execute the script

Category is anything you want to type in that helps you organize scripts

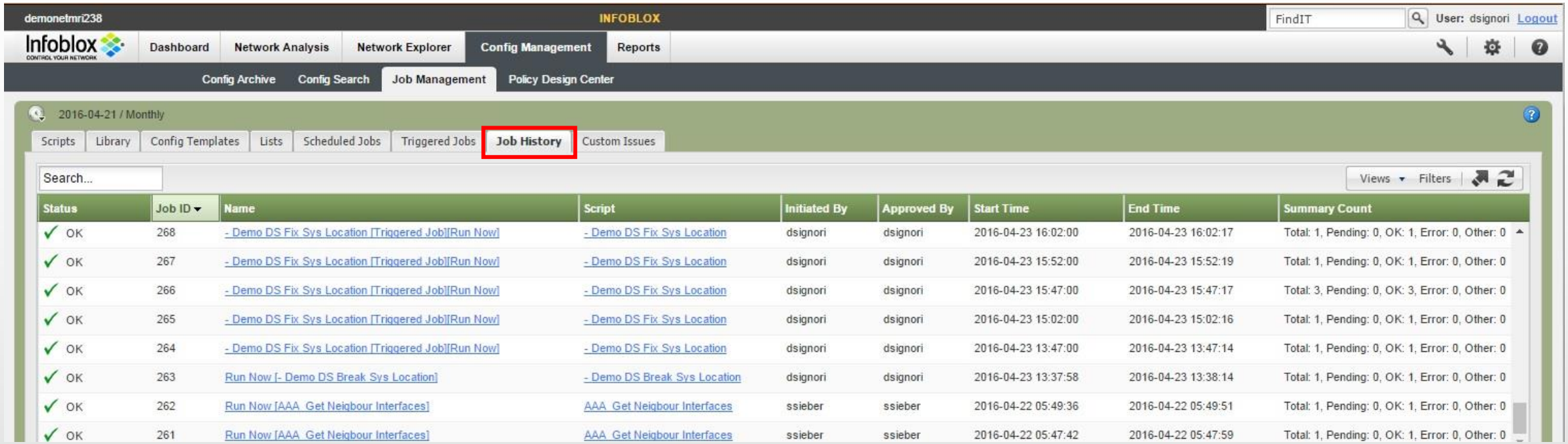
Running CCS Scripts

- Config Management -> Job Management -> Scripts -> Run now

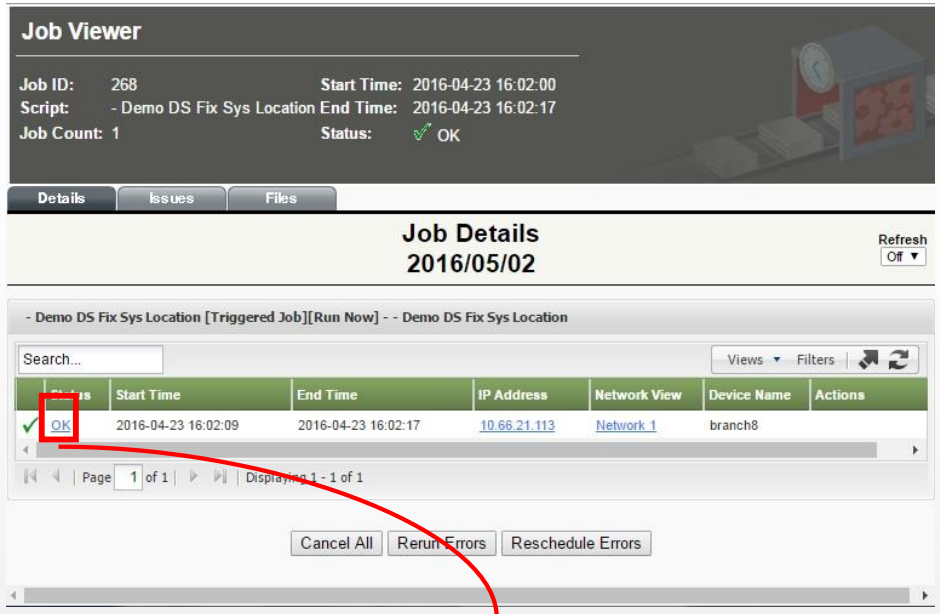


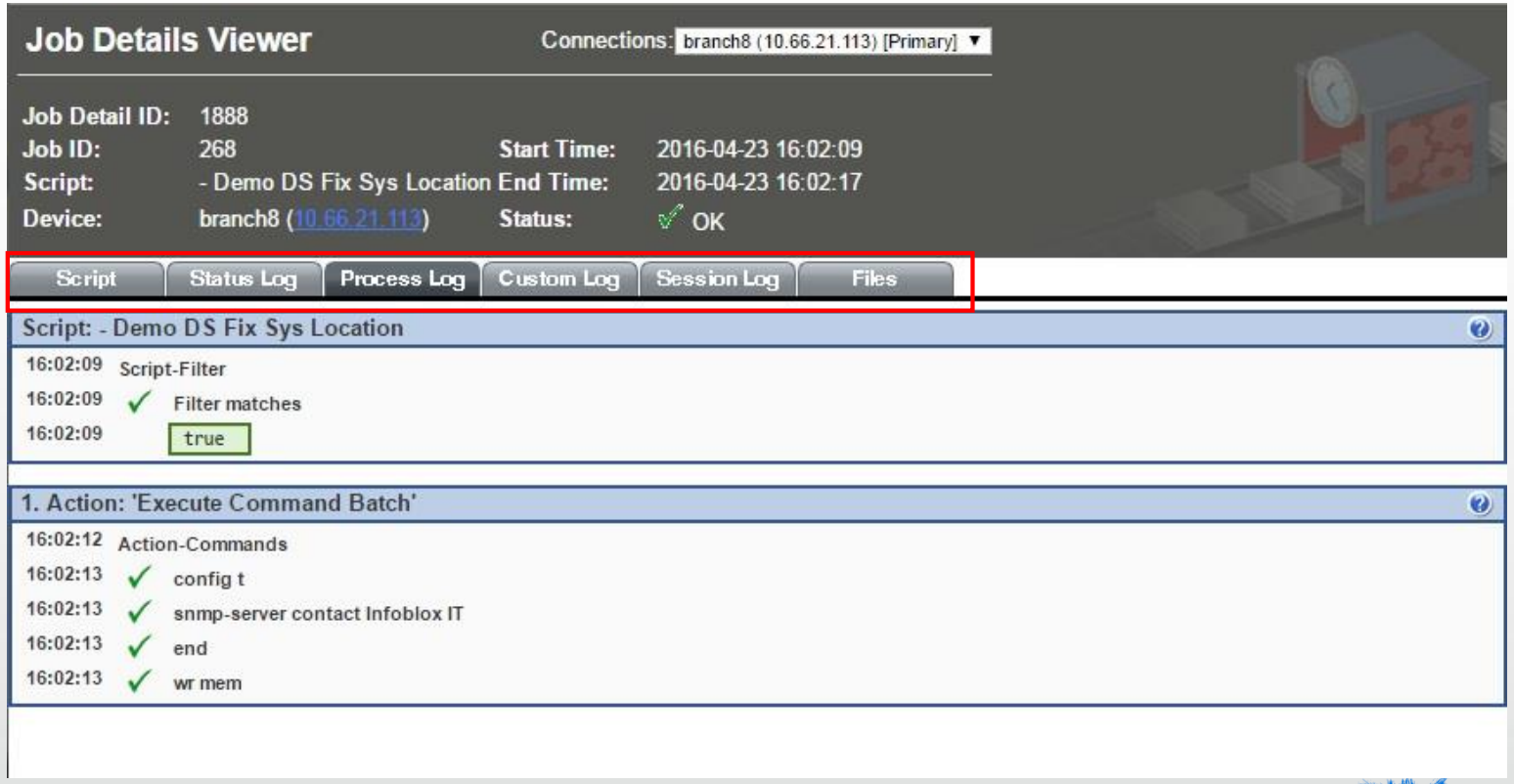
Viewing Status and Job Details

- Config Management -> Job Management -> Job History



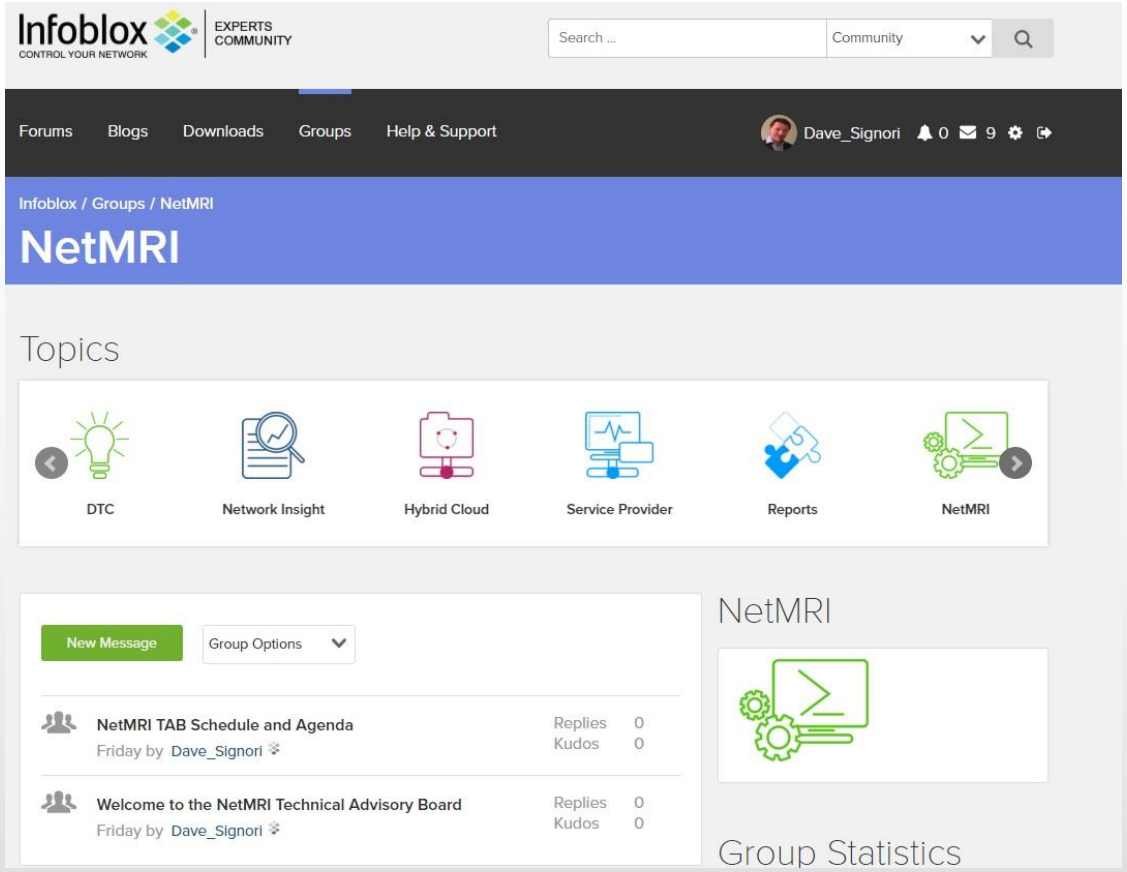
Viewing Status and Job Details





Customer Participation Opportunities

Technical Advisory Boards



- 10 TABs including NetMRI and Network Insight
- Roadmap and early look at pre-released features
- Input for future enhancements
- Best practices
- First NetMRI session held on May 12th
- Request membership at the Infoblox Community Site


BLOX FEST
Infoblox