

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

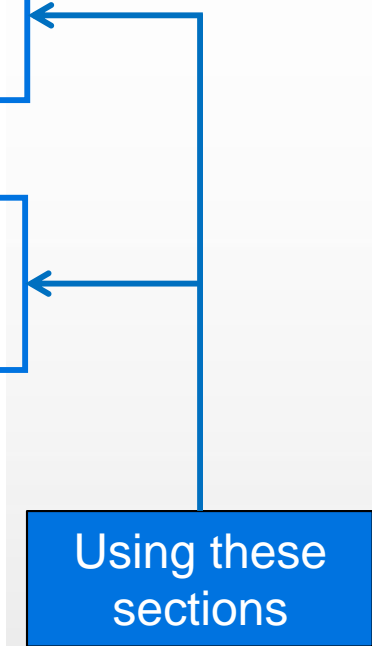
CCS Sections

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	



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

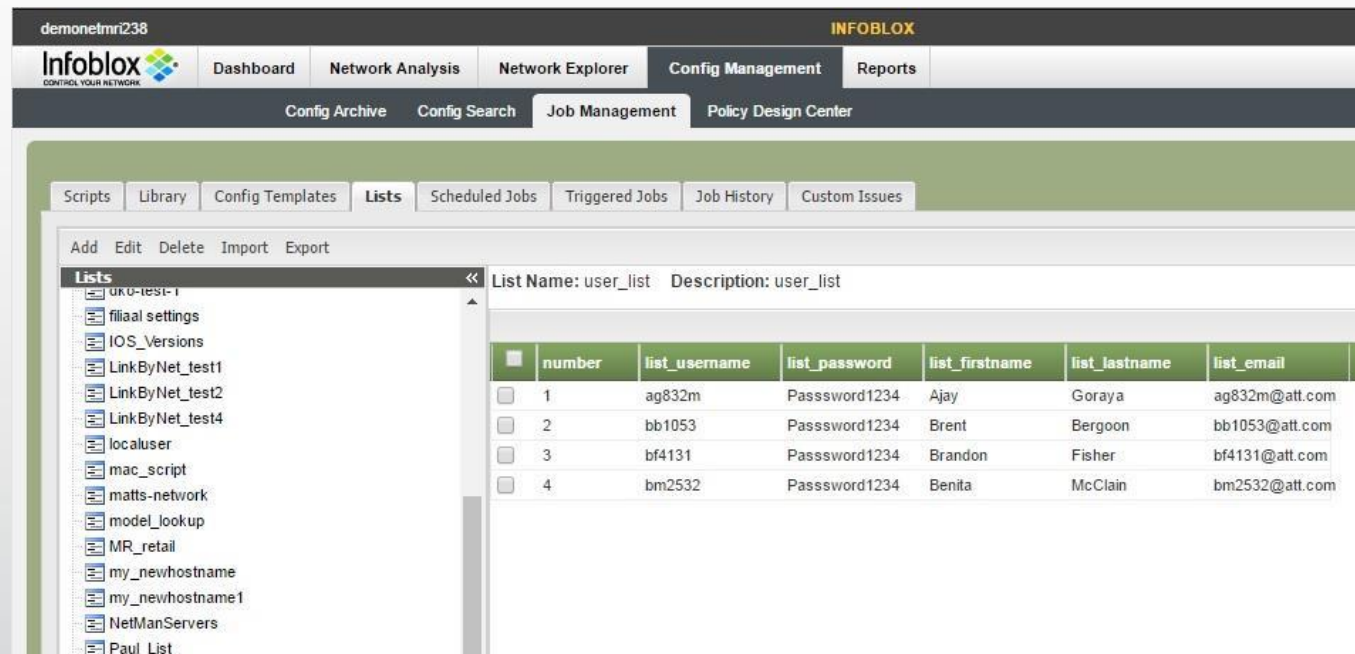
Variables

- 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 lightweight database.
- Use the command getListValue to get the content of the List.
- Lists can be created (or imported) under: Config Management -> Job Management -> Lists



The screenshot displays the Infoblox web interface. The top navigation bar includes 'Dashboard', 'Network Analysis', 'Network Explorer', 'Config Management', and 'Reports'. The 'Config Management' section is active, showing 'Config Archive', 'Config Search', 'Job Management', and 'Policy Design Center'. The 'Job Management' section is selected, showing 'Scripts', 'Library', 'Config Templates', 'Lists', 'Scheduled Jobs', 'Triggered Jobs', 'Job History', and 'Custom Issues'. The 'Lists' page is open, showing a list of existing lists on the left and a table of user list entries on the right.

number	list_username	list_password	list_firstname	list_lastname	list_email
1	ag832m	Passsword1234	Ajay	Goraya	ag832m@att.com
2	bb1053	Passsword1234	Brent	Bergoon	bb1053@att.com
3	bf4131	Passsword1234	Brandon	Fisher	bf4131@att.com
4	bm2532	Passsword1234	Benita	McClain	bm2532@att.com



Reading from a List

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

```
SET: $reset= getListValue(Authorized_Users,username,"alice" default_passwd,NULL)
```

role	username	default_passwd
new_hire	alice	inW0nderl@nd
new_hire	Neo	T@keTheRedPill

- 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
```

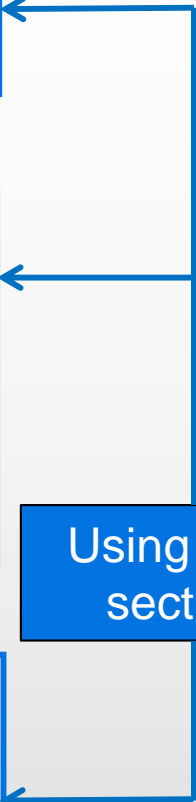


Triggers

- 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	



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 Goldilocks
username alice password 0 Password123
username Neo password 0 Password321
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
```

```
#####
```

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
```

Action, sends output to "Update Password" Trigger

Trigger-Template extracts actual username from "show run | inc username"

Execute commands on the username

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



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"

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"
```

Initialize \$ChangesMade to "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)
```

Set \$ChangesMade to "yes"

Trigger-Commands: { \$ChangesMade eq "no" and \$user_role eq \$role }

```
conf t
```

```
SET: ChangesMade = "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

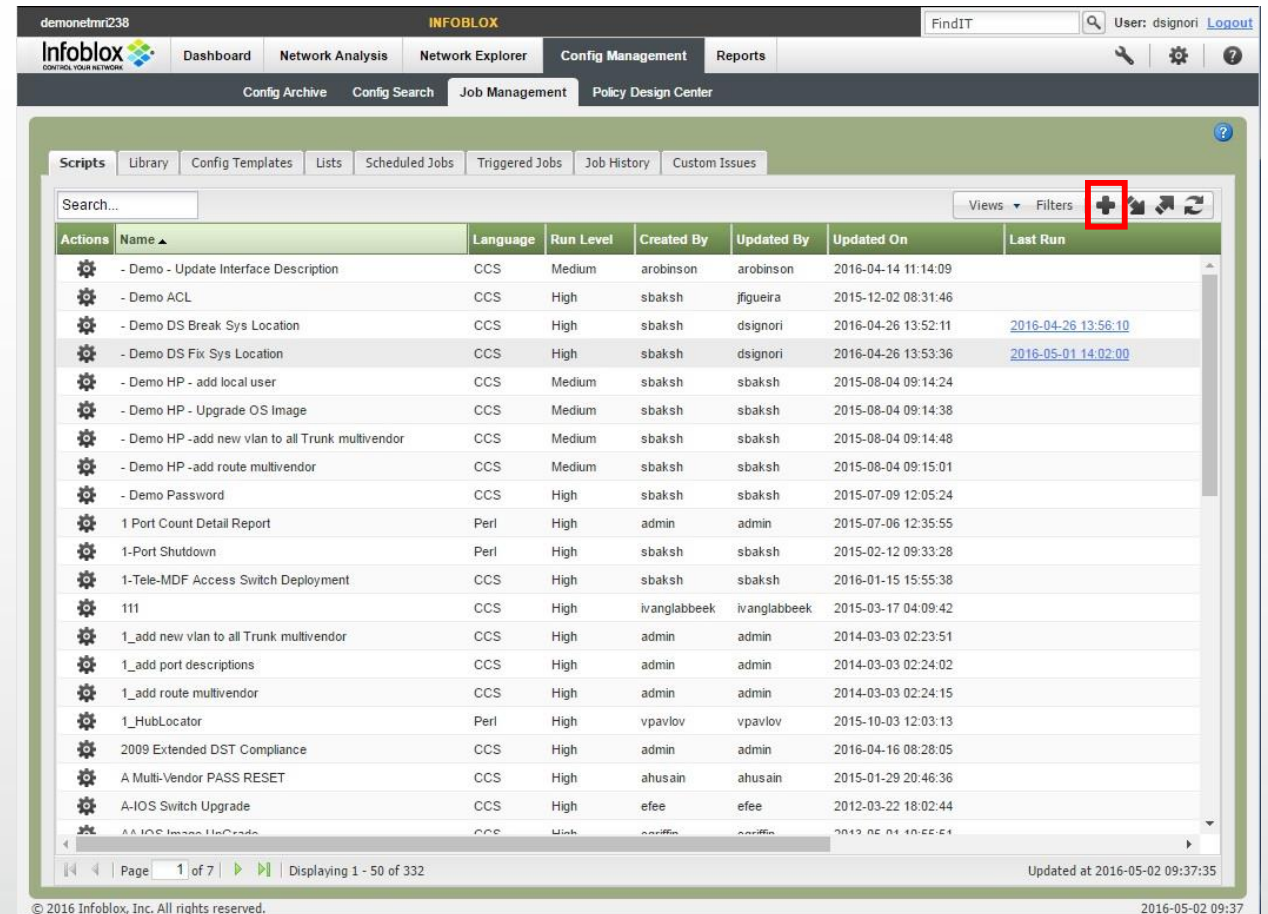
Operator	Description
$A B$	A if it is neither null nor 0, otherwise B
$A \& B$	A if neither argument is null or 0, otherwise 0
$A < B$	A is less than B
$A \leq B$	A is less than or equal to B
$A = B$	A is equal to B
$A \neq B$	A is not equal to B
$A \geq B$	A is greater than or equal to B
$A > B$	A is 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

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: $\$A = \$A * \$B$

Viewing, Adding, and Running CCS Scripts

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



The screenshot shows the Infoblox web interface for managing scripts. The navigation menu includes Dashboard, Network Analysis, Network Explorer, Config Management, and Reports. Under Config Management, there are sub-menus for Config Archive, Config Search, Job Management, and Policy Design Center. The main content area is titled 'Scripts' and contains a table of script actions. A red box highlights the 'Add' button in the top right corner of the table area.

Actions	Name	Language	Run Level	Created By	Updated By	Updated On	Last Run
⚙️	- Demo - Update Interface Description	CCS	Medium	arobinson	arobinson	2016-04-14 11:14:09	
⚙️	- Demo ACL	CCS	High	sbaksh	jfigueira	2015-12-02 08:31:46	
⚙️	- Demo DS Break Sys Location	CCS	High	sbaksh	dsignori	2016-04-26 13:52:11	2016-04-26 13:56:10
⚙️	- Demo DS Fix Sys Location	CCS	High	sbaksh	dsignori	2016-04-26 13:53:36	2016-05-01 14:02:00
⚙️	- Demo HP - add local user	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:24	
⚙️	- Demo HP - Upgrade OS Image	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:38	
⚙️	- Demo HP -add new vian to all Trunk multivendor	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:48	
⚙️	- Demo HP -add route multivendor	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:15:01	
⚙️	- Demo Password	CCS	High	sbaksh	sbaksh	2015-07-09 12:05:24	
⚙️	1 Port Count Detail Report	Perl	High	admin	admin	2015-07-06 12:35:55	
⚙️	1-Port Shutdown	Perl	High	sbaksh	sbaksh	2015-02-12 09:33:28	
⚙️	1-Tele-MDF Access Switch Deployment	CCS	High	sbaksh	sbaksh	2016-01-15 15:55:38	
⚙️	111	CCS	High	ivanglabbeek	ivanglabbeek	2015-03-17 04:09:42	
⚙️	1_add new vian to all Trunk multivendor	CCS	High	admin	admin	2014-03-03 02:23:51	
⚙️	1_add port descriptions	CCS	High	admin	admin	2014-03-03 02:24:02	
⚙️	1_add route multivendor	CCS	High	admin	admin	2014-03-03 02:24:15	
⚙️	1_HubLocator	Perl	High	vpavlov	vpavlov	2015-10-03 12:03:13	
⚙️	2009 Extended DST Compliance	CCS	High	admin	admin	2016-04-16 08:28:05	
⚙️	A Multi-Vendor PASS RESET	CCS	High	ahusain	ahusain	2015-01-29 20:46:36	
⚙️	A-IOS Switch Upgrade	CCS	High	efee	efee	2012-03-22 18:02:44	
⚙️	A-IOS Image Upgrade	CCS	High	cariffe	cariffe	2013-05-01 10:55:41	

Page 1 of 7 | Displaying 1 - 50 of 332 | Updated at 2016-05-02 09:37:35



Adding New CCS Script

The screenshot shows the configuration page for a new CCS script. The fields are as follows:

- Name:** ANAC Script 1 - Reset Password
- Run Level:** Medium
- Category:** Operations
- Description:** Sets username and password on a Cisco device running IOS

The script body is shown in a light blue box:

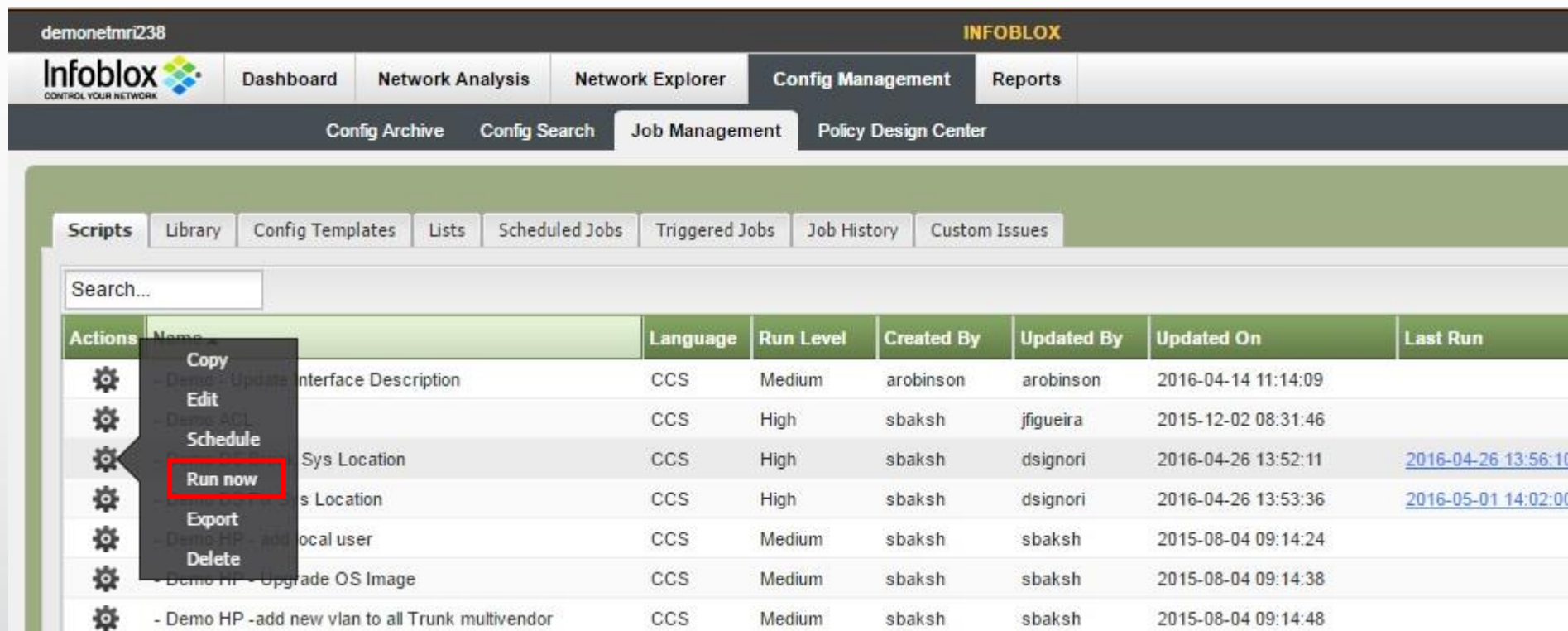
```
Script-Filter:  
$Vendor eq "Cisco" and $sysDescr like /IOS/  
  
Script-Variables:  
$username word "User Name"  
$password password "New Password"  
  
Action:  
Sets IOS User and Password  
  
Action-Commands:  
config terminal  
username $username password 0 $password  
exit  
write memory
```

Callout boxes provide the following explanations:

- Name of script:** Points to the Name field.
- Levels Low, Medium, and High, indicate the permission level needed to execute the script:** Points to the Run Level dropdown.
- Category is anything you want to type in that helps you organize scripts:** Points to the Category field.
- Body of CCS script:** Points to the script body text area.

Running CCS Scripts

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

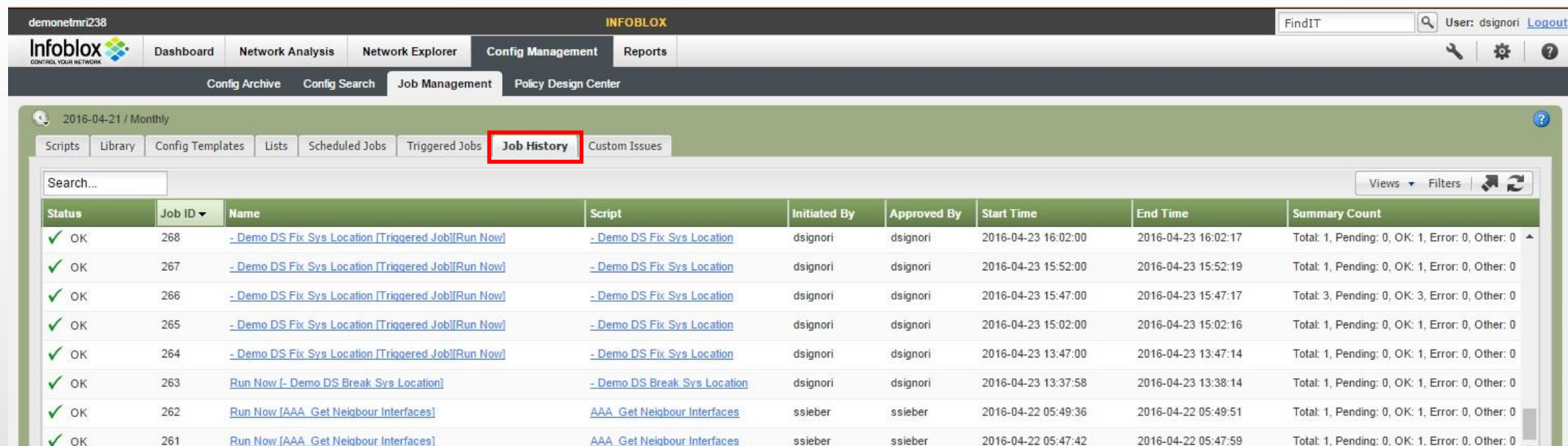


The screenshot shows the Infoblox web interface. The user is logged in as 'demonetmri238'. The navigation menu includes 'Dashboard', 'Network Analysis', 'Network Explorer', 'Config Management', and 'Reports'. Under 'Config Management', there are sub-menus for 'Config Archive', 'Config Search', 'Job Management', and 'Policy Design Center'. The 'Scripts' page is active, showing a search bar and a table of scripts. A context menu is open over the 'Run now' button of the 'Sys Location' script.

Actions	Name	Language	Run Level	Created By	Updated By	Updated On	Last Run
	Interface Description	CCS	Medium	arobinson	arobinson	2016-04-14 11:14:09	
		CCS	High	sbaksh	jfigueira	2015-12-02 08:31:46	
	Sys Location	CCS	High	sbaksh	dsignori	2016-04-26 13:52:11	2016-04-26 13:56:10
	s Location	CCS	High	sbaksh	dsignori	2016-04-26 13:53:36	2016-05-01 14:02:00
	- Demo HP - add local user	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:24	
	- Demo HP - Upgrade OS Image	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:38	
	- Demo HP - add new vlan to all Trunk multivendor	CCS	Medium	sbaksh	sbaksh	2015-08-04 09:14:48	

Viewing Status and Job Details

- Config Management -> Job Management -> Job History



The screenshot shows the Infoblox web interface. The user is logged in as 'demonetmri238' and 'User: dsignori'. The navigation menu includes 'Dashboard', 'Network Analysis', 'Network Explorer', 'Config Management', and 'Reports'. Under 'Config Management', there are sub-tabs for 'Config Archive', 'Config Search', 'Job Management', and 'Policy Design Center'. The 'Job History' tab is selected and highlighted with a red box. The main content area shows a table of job history for the date '2016-04-21 / Monthly'. The table has columns for Status, Job ID, Name, Script, Initiated By, Approved By, Start Time, End Time, and Summary Count. The 'Job History' tab is highlighted with a red box.

Status	Job ID	Name	Script	Initiated By	Approved By	Start Time	End Time	Summary Count
✓ OK	268	- Demo DS Fix Sys Location [Triggered Job][Run Now]	- Demo DS Fix Sys Location	dsignori	dsignori	2016-04-23 16:02:00	2016-04-23 16:02:17	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	267	- Demo DS Fix Sys Location [Triggered Job][Run Now]	- Demo DS Fix Sys Location	dsignori	dsignori	2016-04-23 15:52:00	2016-04-23 15:52:19	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	266	- Demo DS Fix Sys Location [Triggered Job][Run Now]	- Demo DS Fix Sys Location	dsignori	dsignori	2016-04-23 15:47:00	2016-04-23 15:47:17	Total: 3, Pending: 0, OK: 3, Error: 0, Other: 0
✓ OK	265	- Demo DS Fix Sys Location [Triggered Job][Run Now]	- Demo DS Fix Sys Location	dsignori	dsignori	2016-04-23 15:02:00	2016-04-23 15:02:16	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	264	- Demo DS Fix Sys Location [Triggered Job][Run Now]	- Demo DS Fix Sys Location	dsignori	dsignori	2016-04-23 13:47:00	2016-04-23 13:47:14	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	263	Run Now [- Demo DS Break Sys Location]	- Demo DS Break Sys Location	dsignori	dsignori	2016-04-23 13:37:58	2016-04-23 13:38:14	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	262	Run Now [AAA Get Neighbour Interfaces]	AAA Get Neighbour Interfaces	ssieber	ssieber	2016-04-22 05:49:36	2016-04-22 05:49:51	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0
✓ OK	261	Run Now [AAA Get Neighbour Interfaces]	AAA Get Neighbour Interfaces	ssieber	ssieber	2016-04-22 05:47:42	2016-04-22 05:47:59	Total: 1, Pending: 0, OK: 1, Error: 0, Other: 0

Viewing Status and Job Details

Job Viewer

Job ID: 268 Start Time: 2016-04-23 16:02:00
Script: - Demo DS Fix Sys Location End Time: 2016-04-23 16:02:17
Job Count: 1 Status: ✔ OK

Details Issues Files

Job Details
2016/05/02 Refresh
Of ▼

- Demo DS Fix Sys Location [Triggered Job][Run Now] - - Demo DS Fix Sys Location

Search... Views Filters

OK	Start Time	End Time	IP Address	Network View	Device Name	Actions
✔ OK	2016-04-23 16:02:09	2016-04-23 16:02:17	10.66.21.113	Network 1	branch8	

Page 1 of 1 | Displaying 1 - 1 of 1

Cancel All Rerun Errors Reschedule Errors

Job Details Viewer Connections: branch8 (10.66.21.113) [Primary] ▼

Job Detail ID: 1888
Job ID: 268 Start Time: 2016-04-23 16:02:09
Script: - Demo DS Fix Sys Location End Time: 2016-04-23 16:02:17
Device: [branch8 \(10.66.21.113\)](#) Status: ✔ OK

Script Status Log Process Log Custom Log Session Log Files

Script: - Demo DS Fix Sys Location

16:02:09 Script-Filter
16:02:09 ✔ Filter matches
16:02:09 true

1. Action: 'Execute Command Batch'

16:02:12 Action-Commands
16:02:13 ✔ config t
16:02:13 ✔ snmp-server contact Infoblox IT
16:02:13 ✔ end
16:02:13 ✔ wr mem

Customer Participation Opportunities

Technical Advisory Boards

The screenshot displays the Infoblox Experts Community website. At the top, the Infoblox logo is on the left, and a search bar and a 'Community' dropdown menu are on the right. Below this is a navigation bar with links for 'Forums', 'Blogs', 'Downloads', 'Groups', and 'Help & Support'. The user profile for 'Dave_Signori' is visible in the top right corner. The main content area is titled 'Infoblox / Groups / NetMRI' and features a large blue header with the text 'NetMRI'. Underneath, there is a 'Topics' section with six icons representing different categories: DTC, Network Insight, Hybrid Cloud, Service Provider, Reports, and NetMRI. Below the topics, there is a 'New Message' button and a 'Group Options' dropdown. A list of messages is shown, including 'NetMRI TAB Schedule and Agenda' and 'Welcome to the NetMRI Technical Advisory Board', both posted by Dave_Signori. To the right of the messages, there is a 'NetMRI' group card with a gear and code icon. At the bottom, there is a 'Group Statistics' section.

- 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 