Enhance the IOS User Interface

By Ivan Pepelnjak

1. 5. 2007

Have you ever wanted to fine-tune the IOS **show** commands to provide you with the exact information you need instead of having to dig through long screens full of data you're not interested in to find what you need? In this article, you'll see how you can use the simple filters provided by Cisco IOS to pick only the information you need from the printouts, as well as how you can generate tailored printouts (even combining outputs from multiple **show** commands) with Tcl shell introduced in IOS release 12.3(2)T.

Introduction to Output Filters

The output filters of the **show** command were introduced in IOS release 12.0T. They give you the ability to filter the output of any **show** command supported by Cisco IOS by appending the pipe symbol (|) and a filter at the end of the **show** command. You can:

- Include or exclude the lines matching the specified regular expression with the **include** and **exclude** filter;
- Start the printout at the line matching a regular expression with the **begin** filter;

- Output only the matching sections of the printout with the **section** filter.

Note: A section starts with a line with no leading blank and includes all lines following it until the start of the next section. The ***section*** *filter is commonly used to filter router configuration printouts, but can also be used for any other* ***show*** *command.*

Similar filters (only **include**, **exclude** and **begin**) are provided by the **more** command. Other EXEC level commands do not support output filters, as they are not a generic extension of the command line interface (like pipes in Unix or Windows operating systems), but a special keyword in the **show** and **more** commands. That's also the reason you cannot cascade two (or more) filters like you can on Unix.

Note: The flexibility of the output filters relies heavily on the regular expressions. While you can use simple strings to filter the printouts (as we'll do in the next section), you'll achieve much more if you'll master the full scope of the regular expressions. You can get more information about regular expressions in the Appendix A of the Terminal Services Configuration Guide part of Cisco IOS documentation.

Simple Filters

To give you an idea what you can achieve with the output filters, consider a common Frame Relay-related problem: you would like to see which Frame Relay virtual circuits (DLCIs) are active. The only IOS command displaying DLCI status is the **show frame-relay pvc** command, which gives you way too much information (only a single DLCI is included in Listing 1)

Listing 1: Output of the show frame-relay pvc command

```
a1#show frame-relay pvc

PVC Statistics for interface Serial0/0/0 (Frame Relay DTE)

              Active     Inactive      Deleted       Static
  Local          1            0            0            0
  Switched       0            0            0            0
  Unused         3            0            0            0
```

```
DLCI = 100, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/0
.100

  input pkts 514           output pkts 410          in bytes 50186

  out bytes 43749          dropped pkts 0           in pkts dropped 0

  out pkts dropped 0                out bytes dropped 0

  in FECN pkts 0           in BECN pkts 0           out FECN pkts 0

  out BECN pkts 0          in DE pkts 0             out DE pkts 0

  out bcast pkts 334       out bcast bytes 39257

  5 minute input rate 0 bits/sec, 0 packets/sec

  5 minute output rate 0 bits/sec, 0 packets/sec

  pvc create time 00:44:30, last time pvc status changed 00:44:11

… remaining printout deleted …
```

If you use the output filter, displaying only those lines in the **show frame-relay pvc** printout that include the string *DLCI USAGE*, you'll get exactly what you need: a short printout of the DLCI status (Listing 2).

Listing 2: Short printout displaying just the DLCI status

```
a1#show frame pvc | include DLCI USAGE

DLCI = 100, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/0
.100

DLCI = 200, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0

DLCI = 301, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0

DLCI = 401, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0
```

If you'd like to include the WAN interface name in the printout (although it's specified in the DLCI status line), you have to modify the output filter a bit: it also has to include the lines with the string *for interface*. The *or* regular expression (pipe operator) provides that functionality (Listing 3).

Listing 3: Slightly enhanced printout of the DLCI status

```
a1#show frame pvc | include for interface|DLCI USAGE

PVC Statistics for interface Serial0/0/0 (Frame Relay DTE)

DLCI = 100, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/0
.100
```

```
DLCI = 200, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0

DLCI = 301, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0

DLCI = 401, DLCI USAGE = UNUSED, PVC STATUS = ACTIVE, INTERFACE = Serial0/0/
0
```

To make your enhancement to the IOS user interface really user-friendly, you can define an alias for it. For example, if you define a new command **dlci** as **alias exec dlci show frame pvc | include for interface|DLCI USAGE**, the DLCI status will be displayed every time the IOS user enters the **dlci** command. You can even use the new command through the IOS web interface: to display the DLCI status in your browser, use the http://*router-name*/level/1/exec/dlci/CR URL.

Note: The CR at the end of the URL indicates the end of the command line (as the router cannot determine whether you want to extend the regular expression that terminates the ***alias*** *definition or not).*

Displaying Router Configuration

The output filters are probably most useful when you're trying to find relevant parts of IOS configuration (more so when you're under stress in a network down situation and need to find something quickly on a high-end router with thousands of lines in its configuration). Here's also where the **section** filter is most useful. For example, to display all routing processes configured on a router, use the command in Listing 4.

Listing 4: Display all routing processes on a router

```
a1#show running-config | section ^router

router ospf 1

 log-adjacency-changes

 network 0.0.0.0 255.255.255.255 area 0

router bgp 1

 no synchronization

 bgp log-neighbor-changes

 network 172.16.0.1 mask 255.255.255.255
 neighbor 172.16.0.12 remote-as 1
 neighbor 172.16.0.12 update-source Loopback0
 no auto-summary
```

*Note: The regular expression ^router matches the string router only if it occurs at the beginning of the line (as requested by the ^ special character). The **section ^router** filter displays all sections of IOS configuration that start with the word **router, that is** all routing processes.*

Similarly, to display all access lists configured on a router, you can use a **begin** filter as shown in Listing 5. The regular expression specified in the **begin** filter matches either **access-list** or **ip access-list** strings occurring at the beginning of the line (to ensure that an **access-list** keyword in some other IOS command would not trigger the printout). While you could use the ^**ip access-list|^access-list** regular expression in the filter, the one used in the example is more concise and easier to understand once you master the subtleties of the regular expressions.

*Note: The **begin** filter displays all the routing configuration after the start of the access list definitions, not just the access lists, but I've decided to use it in this case to preserve the separations (exclamation marks) between the access lists.*

Listing 5: Display configured access lists

```
a1#show running-config | begin ^(ip )?access-list
ip access-list extended Test
 permit ip any any
!
ip access-list logging interval 1000
access-list 10 permit 192.168.0.10
access-list 200 permit 0x0801 0x0000
!
… rest of printout deleted …
```

Note: To enter question mark (?) in a configuration command, use the Ctrl-V,? sequence.

The next example illustrates why it's important to include the beginning-of-line checks in the regular expressions: if you define an alias for your filter, it might appear in the configuration output (Listing 6).

Listing 6: Command alias appearing in the configuration output

```
a2#show running-config | section event manager
alias exec events show running-config | section event manager
event manager applet after-restart
 event syslog occurs 1 pattern "%SYS-5-RESTART: System restarted"
 action 2.0 cli command "enable"
 action 2.1 cli command "configure terminal"
 action 2.2 cli command "interface loopback 101"
 action 2.3 cli command "no shutdown"
```

I would like to finish this section with a warning: before executing a **show running-config** command and filtering its output, you should check whether the filtered output is provided by the **show running-config** command. For example, in IOS release 12.4, you can display configuration of a single interface or all configured class-maps and policy-maps without generating the full router configuration, resulting in a significantly faster operation (more so on high-end devices with large configurations).

OSPF Example

The area where I use output filters most often is the routing protocol displays. For example, the **show ip ospf neighbor** command does not display the neighbors' areas, those can only be displayed with the **show ip ospf neighbor detail** command, which is way too verbose to be useful when you need a concise printout (the information displayed about a single OSPF neighbor is included in Listing 7).

Listing 7: Detailed printout of the OSPF neighbor data

```
a1#show ip ospf neighbor detail
 Neighbor 172.16.0.21, interface address 172.16.1.2
    In the area 0 via interface Serial0/0/0.100
    Neighbor priority is 0, State is FULL, 6 state changes
    DR is 0.0.0.0 BDR is 0.0.0.0

    Options is 0x52

    LLS Options is 0x1 (LR)

    Dead timer due in 00:00:37

    Neighbor is up for 00:39:02

    Index 1/1, retransmission queue length 0, number of retransmission 1

    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)

    Last retransmission scan length is 1, maximum is 1

    Last retransmission scan time is 0 msec, maximum is 0 msec

… rest deleted …
```

All the information I'm usually interested in (neighbor IP addresses, interface, adjacency state and area – highlighted in Listing 7) are displayed in lines that contain the string *Neighbor* (regular expressions are case-sensitive!) or *area*, so you can get close to what I would like to see with a simple filter matching these two words (Listing 8).

Listing 8: Concise OSPF neighbor printout

```
a1#show ip ospf neighbor detail | include Neighbor|area

 Neighbor 172.16.0.21, interface address 172.16.1.2

    In the area 0 via interface Serial0/0/0.100

    Neighbor priority is 0, State is FULL, 6 state changes

    Neighbor is up for 00:40:07

 Neighbor 172.16.0.12, interface address 10.0.0.6

    In the area 0 via interface FastEthernet0/0

    Neighbor priority is 1, State is FULL, 6 state changes

    Neighbor is up for 00:40:18
```

Getting rid of the neighbor uptime is harder. It would be ideal if we could just pipe the filtered output into another filter and exclude all lines with the string *Neighbor is*, but we cannot do it, as we only get one filter per command. Our **include** filter thus has to select:

- Lines that contain the string *Neighbor* and the string *priority;*
- Lines that contain string *Neighbor* and *interface;*
- Lines that contain the string *area*.

The first two requirements can be specified with the regular expression *StringA.*StringB* (meaning, literally, match *StringA*, then anything repeated as many times as needed, then *StringB*), resulting in a long output filter **include Neighbor.*interface|Neighbor.*priority|area**. If you want to make it shorter, you can group strings in a hierarchical structure, resulting in the filter displayed in Listing 9.

Listing 9: Perfect filter displaying only the relevant details about OSPF neighbors

```
a1#show ip ospf neighbor detail | include (Neighbor.*(interface|priority))|a
rea
 Neighbor 172.16.0.21, interface address 172.16.1.2
    In the area 0 via interface Serial0/0/0.100
    Neighbor priority is 0, State is FULL, 6 state changes
 Neighbor 172.16.0.12, interface address 10.0.0.6
    In the area 0 via interface FastEthernet0/0
    Neighbor priority is 1, State is FULL, 6 state changes
```

EIGRP Example

If you've deployed EIGRP stub neighbors in your network, you'll probably want to know which neighbors of a core EIGRP router are stub when doing the network troubleshooting. The **show ip eigrp neighbors** command does not give you that information, you have to use the **show ip eigrp neighbors detail** command (Listing 10), which is slightly too verbose (although extremely terse when compared to the equivalent OSPF command). Ideally, we would like to see only the highlighted lines from Listing 10 in the printout and include the EIGRP process information and the printout headers.

Listing 10: Detailed information about an EIGRP neighbor

```
a2#show ip eigrp neighbors detail
IP-EIGRP neighbors for process 1
H   Address         Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                    (sec)         (ms)       Cnt Num
1   172.16.1.6      Se0/0/0.101       12 00:00:12   36   216   0   9
```

```
   Version 12.4/1.2, Retrans: 1, Retries: 0, Prefixes: 2
   Stub Peer Advertising ( CONNECTED ) Routes
   Suppressing queries
```

The filtered printout should thus include:

- The EIGRP process information: any line containing the *process* string.
- Printout headers: lines containing *Interface* or *(sec)* strings.
- Neighbor addresses and interfaces: lines starting with a digit.
- Stub neighbor information: lines containing the *Stub* string.

The only regular expression we haven't encountered yet is the *lines starting with a digit* requirement. While you could code it as ^0|^1|^2…, it's simpler to write it as **^[0-9]** (meaning: beginning of the line and then any character from 0 to 9). The final filter is thus **include ^[0-9]|process|Interface|(sec)|Stub** (Listing 11).

Listing 11: Filtered printout of EIGRP neighbors

```
a2#show ip eigrp neighbors detail | include ^[0-
9]|process|Interface|(sec)|Stub
IP-EIGRP neighbors for process 1
H   Address         Interface      Hold Uptime    SRTT   RTO  Q   Seq
                                   (sec)          (ms)        Cnt  Num
1   172.16.1.6      Se0/0/0.101    14 00:02:01    36    216  0   9
   Stub Peer Advertising ( CONNECTED ) Routes
0   10.0.0.5        Fa0/0          12 00:03:33    27    200  0   11
```

Using the Tcl Shell

The output filters attached to the **show** commands can fine-tune the information displayed to the operator, but can never rearrange it or output it in a tabular format. All that is possible with Tcl, a general-purpose language that became available in the command-line interface with the introduction of Tcl shell in IOS release 12.3(2)T, now integrated in release 12.4 (Tcl was previously used in Cisco IOS only to filter syslog messages and in voice scripts).

The ability to generate user-defined printout in Cisco IOS relies on these IOS features:

- The **tclsh** command accepts a file name as its parameter, executing Tcl commands in that file. The file can be local (stored in on-board flash, USB flash or even NVRAM).
- You can use the **exec** function in Tcl to execute any EXEC mode IOS command and collect its printouts.
- You can configure an **alias** to give the Tcl procedure executed with the Tcl shell a simple command name.

For example, it would be nice to have the DLCI printout shown in the *Simple filters* section in a tabular format. An initial solution is shown in Listing 12. When you copy the Tcl code into *dlci.tcl* and save the file on router's flash memory, you can configure **alias exec dlci tclsh flash:dlci.tcl** to have it available as a simple exec-level command (the results are in Listing 13).

Listing 12: Simple Tcl procedure to display DLCI status in tabular format

```
set lineFormat "%4s %-10s %-10s %s"

puts [format $lineFormat "DLCI" "Status" "Usage" "Interface"]

puts "=============================================="

set text [exec "show frame-relay pvc"]

foreach line [split $text "\n"] {

  if {[regexp {DLCI = ([0-
9.]+).*USAGE = (\w+).*STATUS = (\w+).*INTERFACE = (.*)}
$line ignore dlci usage status ifname]} {

    puts [format $lineFormat $dlci $status $usage $ifname]

  }

}
```

Listing 13: Sample DLCI printout

```
a1#dlci

DLCI Status     Usage      Interface

=============================================

 100 ACTIVE     LOCAL      Serial0/0/0.100

 200 ACTIVE     SWITCHED   Serial0/0/0
```

```
301  ACTIVE      SWITCHED    Serial0/0/0

401  ACTIVE      UNUSED      Serial0/0/0
```

The Tcl procedure is quite simple (ignoring the obscure format that's even harder to understand for uninitiated than Perl or Lisp):

- The first line defines the output format (similar to C *sprintf* function).
- The next two lines print the legend (the *format* function is equivalent to *sprintf* in C and the *puts* function displays a line of text).
- The fourth line executes the **show** command and stores its results in variable *text*.
- The *foreach* loop iterates over each output line produced from the results of the **show** command with the *split* function, which breaks the text into individual lines.
- Each line is matched with a regular expression that also collects bits of text (the parts of regular expression enclosed in parentheses) and stores them in specified variables (one of the more confusing aspects of Tcl is that you have to write variables with or without the $ sign depending on the context).
- If the line matches the regular expression, the variables specified in the *regexp* function already contain the matched values, so they can be used immediately in the *format* function to generate a line of the printout.

If you find the procedure in Listing 12 way too confusing, don't worry – it's important that you know what can be done in Cisco IOS and how you can integrate Tcl procedures with the rest of the IOS, and everything else can be done by a decent Tcl programmer.

Similarly to the command aliases defined in previous sections, the Tcl procedures can be invoked via a web browser. The procedure defined in this section is executed with the URL http://*router-name*/level/1/exec/dlci (assuming the *dlci* alias has been configured).

Summary

All Cisco IOS **show** commands support output filters that can **include** or **exclude** lines matching a specified regular expression from the generated printout. A **section** filter also allows you to **include** or **exclude** whole sections of the output (a section is an output line without a leading blank and all subsequent lines having at least one leading blank). These output filters allow you to quickly get the information you need from IOS printouts that might be too verbose for you. With the **alias** configuration command, you can go a step further, defining new IOS commands that display only the information you're interested in.

The output filters only allow you to tailor the amount of information generated by the **show** commands; they cannot reorder data or display it in tabular format. All that is possible to implement with Tcl, a generic-purpose programming language included in Cisco IOS. You can execute any IOS command (or a set of commands) in Tcl, collect the output, extract the desired information and reformat it. The Tcl procedures can also be defined as aliases, giving network operators new commands that can be tailored to their environment or the tasks they have to perform. Even more, the same commands can be executed from a web browser, displaying the desired information without requiring the network operator to start a telnet session with the device.

NIL – More Than Just a Training Company

NIL Learning delivers the leading-edge Cisco training to IT professionals and companies around the globe. Through field-proven experts — each both active engineer and instructor — NIL Learning enhances the standard learning curriculum with real-life experience and helps clients to maximize their training investment.

NIL Learning is part of NIL, a leading global IT solutions provider. Since 1992, NIL has been at the forefront of advanced contributors to strategic partner Cisco's technologies, learning curriculum and value-added solutions deployed to clients around the globe. Today, NIL has earned the highest certifications offered by Cisco, VMware, EMC, HP, IBM, Microsoft, F5, Jive, MobileIron, RSA, VCE and others. Their portfolio of solutions consists of managed services, professional services and learning services.

NIL is headquartered in Slovenia, with regional offices in Croatia, Serbia, Saudi Arabia, the U.S., Turkey, South Africa, Morocco, Nigeria, Kenya and Botswana.

Why learn at NIL LEARNING?

- All NIL LEARNING instructors are **field-proven experts** - each both active engineer and instructor.
- **75% of NIL LEARNING engineers hold CCSI** certifica- tions, and **18 have already achieved the respected CCIE rank**.
- NIL LEARNING **enhances the standard learning curriculum** with real-life experience and helps clients to maximize their training investment.
- NIL has been a Cisco Training Partner for many years; it became a **Cisco Learning Partner in 1993**, and has been a **Cisco Gold Partner since 1995**.
- NIL was awarded the **Cisco Most Business Relevant Learning Partner in MEA in 2010 and the most innova- tive learning partner in MEA**.
- NIL received the **Innovation Award for its Technology Led Training and its extensive contribution to Cisco learning solutions** at the Cisco EMEAR Learning Partner Summit in 2012.
- NIL received the **Innovation Award for its Technology Led Training and Advanced Engineer Program** at the Cisco Global Learning Partner Summit in 2013.
- NIL LEARNING runs a **centralized training schedule across the whole EMEAR region**.

More Info

Slovenia

T: +386 1 4746 500

E: sales-support@nil.com

Nigeria

T: +27 (0)11 575 4637

E: mea_sales@nil.com

Botswana

T: +267 318 1684

E: training@it-iq.bw

Saudi Arabia

T: +966 1 465 4641

E: info.nilme@nil.com

Croatia

T: +385 (0)51 583 255

E: info-nilcroatia@nil.com

Serbia

T: +381 11 2282 818

E: info-nilserbia@nil.co.rs

Kenya

T: +27 (0)11 575 4637

E: mea_sales@nil.com

South Africa

T: +27 (0)11 575 4637

E: mea_sales@nil.com

Morocco

T: +212(0) 660 808 394

E: info-nilmorocco@nil.com

Turkey

T: +902 123 81 8639

E: info-nilturkey@nil.com

USA

T: +1 612 886 3900

www.learning.nil.com

E: info-nilusa@nil.com