IOS Tips and Tricks

Al Friebe, Cisco Certified Systems Instructor, CCDA, CCDP, CCNA, CCNP, A+

Introduction

There are some helpful things we can do with Cisco’s IOS to make our lives easier. This white paper will present some useful commands, followed (in parentheses) by their most concise shortcuts (at the time of this writing).

But we begin, please note that nothing requires you to use the briefest shortcuts. For example, many people shortcut the command configure terminal as conf term, config t, or config term. You can find the shortcuts you like, and use them. Remember that when shortcutting commands, you can only shortcut the keywords, not the variables (such as names or IP addresses). Finally, don’t forget that you can use the “Tab” key to complete keywords (but not variables, of course).

Also note that although this white paper is illustrating these commands on a router, they also work on the IOS-based switches. With these basics in mind, let’s move on to some ways to streamline our work environment.

User versus Privileged Mode

As you may know, when interacting with IOS from the command line interface (CLI), there are two main EXEC modes, user and privileged (the latter is also referred to as enabled mode). In user mode, you can do limited examination of the device (via show commands), and the prompt appears as the device’s hostname (Router in this example) followed by the greater-than (>) symbol:

- Router>

In contrast with user mode, in privileged mode you can see everything the device is capable of displaying (via show commands), access the various configuration modes, and execute the copy and debug commands, among others. To access privileged mode, use the command enable:

- Router>enable (“en”)
- Router#

As you can see, when in privileged mode, the hostname is followed by the pound sign (#). If necessary, you can move from privileged mode back to user mode with the disable command:

- Router#disable (“disa”)
- Router>
You can enter **global configuration mode** from privileged mode:

- Router#configure terminal ("conf t")
- Router(config)#

**Hostname Lookup**

By default, if you mistype a command, the router will attempt to resolve it as a hostname via DNS. This will ultimately fail if there is no DNS server available, but it will take time (behind the scenes it makes twelve attempts). To speed things up, you can tell the device not to bother, like this:

- Router(config)#no ip domain-lookup ("no ip domain-lo")

Speaking of name-to-address resolution, you can manually build a host table that allows you to use the IP utilities (ping, trace, telnet, etc) by host name:

- Router(config)#ip host Big_Switch 1.2.3.4
- Router(config)#ip host SmallSwitch 10.20.30.40
- Router(config)#ip host TFTP-Server 100.200.50.150

Once you’ve created it, you can display the host table:

- Router#show host ("s ho")

**Console Messages & Terminal Commands**

When various events occur, the device will display informational messages on the console. If you’re annoyed by these console messages, you can shut them off:

- Router(config)#no logging console ("no logg con")

Unfortunately, if you disable console logging, you won’t receive any more of those very informative console messages! Nor will you see any debug output, even if debugs are running. I suggest that a better way is to leave the console logging enabled (**logg con**), and synchronize the console output with your typing, like this:

- Router(config-line)#logging synchronous ("logg s")

Now if a console message appears while you are typing, it will display the message, and then re-display your input right where you left off, so that you can keep typing. It’s the best of both worlds.

In a lab environment, it’s sometimes handy to disable the inactivity timeout for the console line (the default setting is ten minutes):

- Router(config)#line console 0 ("lin c 0")
- Router(config-line)#exec-timeout 0 ("exec-t 0")
It also works for the aux and vty lines. You can also use `no exec-timeout`, but be careful not to shortcut it to `no exec`, which shuts off the EXEC process, preventing future logins via the line. I made this mistake once, so I don’t recommend it.

Aside from saving a router or switch configuration to NVRAM, it’s always a good idea to have a backup copy of your current configuration in a separate location in case the device bursts into flame and needs to be replaced. You can do this with `copy run tftp` (or similar), but this requires a file server.

Another way is to do a `show run`, and capture the output to a file. The problem is that as the config is displayed, it will give the “more” prompt every 24 lines (by default). You can disable the “more” function like this:

- `Router#terminal length 0 ("ter l 0")`

Now you can do the `show run` (or whatever) and obtain a continuous output stream. When you’re done with the capture, don’t forget to reset the terminal length to enable the “more” function:

- `Router#terminal length 24 ("ter l 24")`

And speaking of `terminal` commands, remember that to see console messages and debug output in a vty session (telnet or SSH) or the aux port, you have to specifically request it from within the session:

- `Router#terminal monitor ("ter mon")`

**Debugs**

While we’re on the subject of debugs, you can display a list of all of the debugs that are currently running:

- `Router#show debugging ("s deb")`

And you can disable all of the running debugs with:

- `Router#no debug all ("no deb all")`

To save a few keystrokes, you can also do it like this:

- `Router#undebug all ("u all")`

**Config Commands**

To see the startup config (stored in NVRAM):

- `New#show startup-config ("s start")`
- `Old#show config ("s conf")`
Note that `show config` does not display the running config, it displays the startup config. To erase the startup config:

- New#erase startup-config ("erase start")
- Old#write erase ("wr er")

To display the running config (also referred to as the **active** or **current** config):

- New#show running-config ("s run")
- Old#write terminal ("wr t")

To save the running config to NVRAM:

- New#copy running-config startup-config ("copy run start")
- Old#write memory ("wr", you don’t need the “mem”)

Although Cisco no longer officially endorses the old commands (due to confusion over what “show config” does), you might see “old-timers” using them, especially “wr”. Why type fourteen or so keystrokes (“copy run start”) when two (“wr”) will do?

Speaking of configurations, to display only the lines of the running config that contain a specific alphanumeric string (such as “rip”), “pipe” it into “include”:

- Router#show run | include rip ("s run | in rip")

The pipe (redirect) symbol (which looks like a vertical bar) is often found as a shifted backslash, above the Enter key. To display the section of the running config that begins with a specific alphanumeric string (for example, “ospf”), pipe it into **begin**:

- Router#show run | begin ospf ("s run | beg ospf")

Among other options, you can also do an **exclude** (show all lines that do not include the string). Use the question mark to see the other options:

- Router#s run | ?
- Router#s start | ?

Note that the **include**, **begin** and **exclude** options are case-sensitive. To display the section of the running config for a specific interface or subinterface, you could pipe it into **begin**, but you have to identify the interface exactly as it appears in the config:

- Router#s run | beg FastEthernet0/0

The problem with this is that you can’t shortcut the interface type (for example, “fa”), and the search string is case-sensitive. A slicker way to display the interface-specific config info is:

- Router#s run int fa0/0
When you use this option, it's not case-sensitive, and you don’t have to bother with the pipe. Although you can shortcut it, you do have to be precise when specifying the interface identifier. For example, if you want to display the Serial 1/2.345 subinterface config, this will work:

- Router#s run int s1/2.345

But this will not:

- Router#s run int s1/2.3

As always, **no shortcutting of variables**! These options also work when viewing the startup config. And speaking of **show start**, you can also display the startup config with line numbers, which can sometimes be handy when reviewing or discussing it:

- Router#s start linenum ("s star li")

As with **show run**, you can use the question mark with **show start** to see any other available options:

- Router#s run ?
- Router#s star ?

Some IOS features require PKI certificate data, which appears in the running config as a large block of nonsense text. To bypass the PKI certificate data when displaying the running config, use the **brief** option:

- Router#s run brief

Remember, when the output of any **show** command displays **-More-** at the bottom, you have several options:

- To see the next line, hit the **Enter** key
- To see the next screen, hit the **spacebar**
- To return to the CLI prompt, hit any other alphanumeric key

**Keyword “Do” Commands**

With recent versions of IOS you can execute privileged mode commands from the various **config** modes by prefacing the command with the keyword **do**. Here are some examples:

- Router(config)#do show running-config (displays the running config)
- Router(config-if)#do sh start (displays the startup config)
- Router(config-line)#do copy run start (saves the running config to NVRAM)
- Router(config-ext-nacl)#do wr (also saves the running config to NVRAM)
- Router(config-subif)#do ping 1.2.3.4
- Router(config-router)#do trace 10.20.30.40
Unfortunately, once you’ve gotten into the habit of typing `do` to the point where you can’t type a command without it, you’ll find that if you try to use it from user or privileged mode, it doesn’t work:

- Router>do show interface (this doesn’t work)
- Router#do sh run (neither does this)
- Router#do wr (nor this)

### Changing Interfaces

On a related topic, you can move from a subordinate config mode, such as `config-if` or `config-router`, to global config mode (GCM) with the `exit` command:

- Router(config-if)#exit
- Router(config)#

In other words, `exit` took us one level up. You can also move from a subordinate mode to GCM by executing any GCM command. Note how this moves us to GCM, executes the command, and leaves us in GCM:

- RouterA(config-router)#hostname RouterB
- RouterB(config)#

You can also jump from one subordinate config mode to another:

- Router(config-if)#router rip
- Router(config-router)#

Note that if you jump from one interface to another, the prompt doesn’t change, but you’re wherever you last told the router to be:

- Router(config)#int fa0/0 (places us into interface config mode for fa0/0)
- Router(config-if)#shutdown (shuts down fa0/0)
- Router(config-if)#int s1/1 (moves us to serial 1/1)
- Router(config-if)#shut (shuts down serial 1/1)

Since the prompt does not indicate which interface you’re configuring, if at any time you’re not sure where you are, execute the commands necessary to put yourself back where you need to be. This applies to routing protocols, as well, for which the prompts all appear as `config-router`:

- Router(config)#router rip (places us into router config mode for RIP)
- Router(config-router)#version 2 (selects RIP v2)
- Router(config-router)#router ospf 1 (moves to OSPF process 1)
- Router(config-router)#router-id 1.2.3.4 (configures router ID for OSPF 1)
Tab Key Completion

Online help and tab-key completion for a command are only available at the native prompt for that command. For example, this will display the available IP routing protocols:

- Router(config)#router ?

But this will not:

- Router(config-if)#router ?

Likewise, this will display the interface types:

- Router(config)#int ?

But you can’t display the interface types from within interface config mode:

- Router(config-if)#int ?

The tab key will work here, displaying interface as the complete keyword:

- Router(config)#int

But not here:

- Router(config-router)#int

Reloads & Restarts

To get a router to begin using the new IOS after an upgrade, you either have to power-cycle it or execute the privileged mode reload (software restart) command:

- Router#reload

Let’s imagine that we’ve established a Telnet (or SSH) session to a router for some remote-control configuration. What if we make a mistake that not only terminates our session, but also prevents us from reconnecting, such as a misconfigured access list? The result could be a CLE (Career-Limiting Event).

To prevent this, we connect to the router, instruct it to perform a reload in five or ten minutes, then make the changes to the config. Assuming that all goes well, we save the config, and cancel the reload. If, on the other hand, all does not go well (and we cut ourselves off), the scheduled reload will occur. After the router reboots, it will come up with the old config, allowing us to reconnect and try again.

You can schedule reloads for the future by using the in option. For example, to reload five minutes from now:

- Router#reload in 5
You can also reload at a certain time and date with the at option. For example, to reload on August 31 at 1:00 AM:

- Router#reload at 1:00 31 august

To display a reload scheduled via the in or at options:

- Router>show reload ("s rel")

When there is one minute remaining before the scheduled shutdown, the system will display messages to all active lines (console, aux, and vty). The system will also display a message just prior to the reload, but at that point it’s too late to stop the reload from occurring.

To cancel a scheduled reload:

- Router#reload cancel ("rel can")

You should see a message confirming that the shutdown was aborted. Make sure that you see this message, because if you mistyped the “cancel” command, the reload clock is still running. Note that you can view a scheduled reload from user mode, but you must be in privileged mode to schedule or cancel a reload.

**Traceroute**

Every programmer thinks that his or her way of doing things is the best way, but I often wish that they would make a little more effort to be consistent. A case in point is the **Traceroute** command, which exploits the TTL field in the IP header to determine the routers traversed on the way to a specified destination. Like UNIX, the Cisco IOS implementation of Traceroute uses UDP with high port numbers, whereas Microsoft’s implementation uses ICMP Echo Requests ("Pings"). The result of this is that a trace from a Cisco machine may make it through firewalls and router access lists, while a trace from a Microsoft machine may not, or vice-versa.

Another difference is that Cisco’s command is traceroute (which can be shortcut as trace or even tr) and Microsoft’s command is tracert, which can’t be shortcut at all. What makes this really annoying is that Cisco’s traceroute (or trace or tr) and the like don’t work on a Microsoft machine, and Microsoft’s **tracert** doesn’t work on Cisco. This means that if you work in a mixed Cisco/Microsoft environment (as lots of us do), you have to think about which machine you’re on every time you do a trace.

Cisco has given us a way around this, though … the **alias**. What we can do is set up an alias on the Cisco, so that typing the Microsoft tracert command on a Cisco machine will invoke the Cisco traceroute. First, create the alias:

- Router(config)#alias exec tracert traceroute

Now, whenever the router (or switch) sees the string tracert from an **Exec** prompt (that is, user or privileged mode), it substitutes the string **traceroute** in its place. You can now execute the tracert from user or privileged mode:

- Router>tracert 1.2.3.4
From privileged mode you can also invoke the extended tracert, which like extended ping, will prompt you for additional information. Granted, we’ve just “dumbed-down” Cisco IOS to the Microsoft level with regard to trace, but at least now \texttt{tracert} will work on both platforms. The other option, as mentioned before, is to just use \texttt{tra} on a Cisco and \texttt{tracert} with Windows.

By the way, if you work in a Microsoft environment, don’t forget about the Windows \texttt{pathping} command which is similar to Cisco’s \texttt{extended trace}, but using ICMP echoes, of course. Try this on a Windows machine:

- \texttt{C:\WinXP>pathping /?}

\section*{Alias}

The alias feature of IOS can be used for other things. For example, if you make frequent use of the \texttt{show ip ospf neighbor detail} command, you might have discovered that you can shortcut it, like this:

- \texttt{Router#s ip o n de}

Or, you could set up an alias, such as \texttt{siond}, from global config mode:

- \texttt{Router(config)#alias exec siond show ip ospf neighbor detail}

Now you can use \texttt{siond} (or whatever you set up) in place of the full-blown command, including any options, such as:

- \texttt{Router#siond fa0/0}

To display what a particular alias represents:

- \texttt{Router#siond?} (with no space between the alias and the question mark)

To display all existing aliases:

- \texttt{Router#alias}

And, of course, to delete an alias, precede it with “no” in global config mode:

- \texttt{Router(config)#no alias exec siond show ip ospf neighbor detail}

\section*{Summary}

These are just a few ways that IOS commands can help streamline your work, and give you more insight to your system. Note that the shortcuts shown here are not necessarily the most concise possible. Use the question mark option to find shortcuts that you like, and use them.
Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge. Check out the following Global Knowledge courses:

ICND1 – Interconnecting Cisco Network Devices 1
ICND2 – Interconnecting Cisco Network Devices 2
CCNA Boot Camp v2.0

For more information or to register, visit www.globalknowledge.com or call 1-800-COURSES to speak with a sales representative.

Our courses and enhanced, hands-on labs offer practical skills and tips that you can immediately put to use. Our expert instructors draw upon their experiences to help you understand key concepts and how to apply them to your specific work situation. Choose from our more than 700 courses, delivered through Classrooms, e-Learning, and On-site sessions, to meet your IT and management training needs.

About the Author

Al Friebe is a Cisco Certified Systems Instructor (CCSI) who has been teaching networking classes since 1995. In the past, he has served as Global Knowledge’s Course Director for BGP and BSCI, and he is the author of our current ICND2 labs. His previous experience includes instructor duty in the U.S. Navy's Nuclear Power School, radiochemistry, software engineering, and network management. His certifications include CCDA, CCDP, CCNA, CCNP, A+ and others.