



# Bash

---

- [About Bash, on page 1](#)
- [Guidelines and Limitations, on page 1](#)
- [Accessing Bash, on page 1](#)
- [Escalate Privileges to Root, on page 3](#)
- [Examples of Bash Commands, on page 4](#)
- [Managing RPMs, on page 6](#)
- [Persistently Daemonizing an SDK- or ISO-built Third Party Process, on page 8](#)
- [Persistently Starting Your Application from the Native Bash Shell, on page 9](#)
- [Copy Through Kstack, on page 9](#)
- [An Example Application in the Native Bash Shell, on page 10](#)

## About Bash

In addition to the Cisco NX-OS CLI, switches support access to the Bourne-Again Shell (Bash). Bash interprets commands that you enter or commands that are read from a shell script. Using Bash enables access to the underlying Linux system on the device and to manage the system.

## Guidelines and Limitations

The Bash shell has the following guidelines and limitations:

- The binaries located in the `/isan` folder are meant to be run in an environment which is setup differently from that of the shell entered from the **run bash** command. It is advisable not to use these binaries from the Bash shell as the behavior within this environment is not predictable.

## Accessing Bash

In Cisco NX-OS, Bash is accessible from user accounts that are associated with the Cisco NX-OS dev-ops role or the Cisco NX-OS network-admin role.

The following example shows the authority of the dev-ops role and the network-admin role:

```
switch# show role name dev-ops
```

```

Role: dev-ops
Description: Predefined system role for devops access. This role
cannot be modified.
Vlan policy: permit (default)
Interface policy: permit (default)
Vrf policy: permit (default)
-----
Rule      Perm      Type      Scope      Entity
-----
4         permit   command               conf t ; username *
3         permit   command               bcm module *
2         permit   command               run bash *
1         permit   command               python *

switch# show role name network-admin

Role: network-admin
Description: Predefined network admin role has access to all commands
on the switch
-----
Rule      Perm      Type      Scope      Entity
-----
1         permit   read-write

switch#

```

Bash is enabled by running the **feature bash-shell** command.

The **run bash** command loads Bash and begins at the home directory for the user.

The following examples show how to enable the Bash shell feature and how to run Bash.

```

switch# configure terminal
switch(config)# feature bash-shell

switch# run?
  run          Execute/run program
  run-script   Run shell scripts

switch# run bash?
  bash        Linux-bash

switch# run bash
bash-4.2$ whoami
admin
bash-4.2$ pwd
/bootflash/home/admin
bash-4.2$

```




---

**Note** You can also execute Bash commands with **run bash** *command*.

For instance, you can run **whoami** using **run bash** *command*:

```
run bash whoami
```

You can also run Bash by configuring the user **shelltype**:

```
username foo shelltype bash
```

This command puts you directly into the Bash shell.

---

# Escalate Privileges to Root

The privileges of an admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- Only an admin user can escalate privileges to root.
- Bash must be enabled before escalating privileges.
- Escalation to root is password protected.
- SSH to the switch using `root` username through a non-management interface will default to Linux Bash shell-type access for the root user. Type `vsh` to return to NX-OS shell access.

NX-OS network administrator users must escalate to root to pass configuration commands to the NX-OS VSH if:

- The NX-OS user has a shell-type Bash and logs into the switch with a shell-type Bash.
- The NX-OS user logged into the switch in Bash continues to use Bash on the switch.

Run `sudo su 'vsh -c "<configuration commands>"` or `sudo bash -c 'vsh -c "<configuration commands>"`.

The example below demonstrates with network administrator user MyUser with a default shelltype Bash using `sudo` to pass configuration commands to the NX-OS:

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ sudo vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface eth1/2 brief"
```

```
-----
Ethernet      VLAN      Type Mode      Status Reason                               Speed      Port
Interface                                           Ch #
-----
Eth1/2        --        eth  routed down  Administratively down                auto(D)  --
-----
```

The example below demonstrates with network administrator user MyUser with default shelltype Bash entering the NX-OS and then running Bash on the NX-OS:

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
Nexus 3600 software ("Nexus 3600 Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.
```

Any use or disclosure, in whole or in part, of the Nexus 3600 Software or Documentation to any third party for any purposes is expressly prohibited except as otherwise authorized by Cisco in writing. The copyrights to certain works contained herein are owned by other third parties and are used and distributed under license. Some parts of this software may be covered under the GNU Public License or the GNU Lesser General Public License. A copy of each such license is available at

```

http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
*****
* Nexus 3600 is strictly limited to use for evaluation, demonstration *
* and NX-OS education. Any use or disclosure, in whole or in part of *
* the Nexus 3600 Software or Documentation to any third party for any *
* purposes is expressly prohibited except as otherwise authorized by *
* Cisco in writing. *
*****
switch# run bash
bash-4.2$ vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface
eth1/2 brief"

```

```

-----
Ethernet      VLAN    Type Mode    Status Reason                               Speed    Port
Interface
-----
Eth1/2        --      eth  routed down  Administratively down                auto(D) --

```

The following example shows how to escalate privileges to root and how to verify the escalation:

```

switch# run bash
bash-4.2$ sudo su root
bash-4.2# whoami
root
bash-4.2# exit
exit

```

## Examples of Bash Commands

This section contains examples of Bash commands and output.

## Displaying System Statistics

The following example displays system statistics:

```

switch# run bash
bash-4.2$ cat /proc/meminfo
<snip>
MemTotal:      16402560 kB
MemFree:       14098136 kB
Buffers:        11492 kB
Cached:        1287880 kB
SwapCached:      0 kB
Active:         1109448 kB
Inactive:       717036 kB
Active(anon):   817856 kB
Inactive(anon): 702880 kB
Active(file):   291592 kB
Inactive(file): 14156 kB
Unevictable:    0 kB
Mlocked:        0 kB
SwapTotal:      0 kB
SwapFree:       0 kB
Dirty:          32 kB
Writeback:      0 kB
AnonPages:     527088 kB

```

```
Mapped:          97832 kB
<\snip>
```

## Running Bash from CLI

The following example runs **ps** from Bash using **run bash** command:

```
switch# run bash ps -el
F S  UID  PID  PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD
4 S   0    1    0  0  80   0  -   528 poll_s ?        00:00:03 init
1 S   0    2    0  0  80   0  -    0 kthrea ?        00:00:00 kthreadd
1 S   0    3    2  0  80   0  -    0 run_ks ?        00:00:56 ksoftirqd/0
1 S   0    6    2  0 -40  -  -    0 cpu_st ?        00:00:00 migration/0
1 S   0    7    2  0 -40  -  -    0 watchd ?        00:00:00 watchdog/0
1 S   0    8    2  0 -40  -  -    0 cpu_st ?        00:00:00 migration/1
1 S   0    9    2  0  80   0  -    0 worker ?        00:00:00 kworker/1:0
1 S   0   10    2  0  80   0  -    0 run_ks ?        00:00:00 ksoftirqd/1
```

## Running Python from Bash

The following example shows how to load Python and configure a switch using Python objects:

```
switch# run bash
bash-4.2$ python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

!Command: show running-config interface Ethernet1/3
!Time: Mon Nov 4 13:17:56 2013

version 6.1(2)I2(1)

interface Ethernet1/3
  vrf member myvrf

>>>
```

# Managing RPMs

## Installing RPMs from Bash

### Procedure

	Command or Action	Purpose
Step 1	<code>sudo dnf installed   grep platform</code>	Displays a list of the NX-OS feature RPMs installed on the switch.
Step 2	<code>dnf list available</code>	Displays a list of the available RPMs.
Step 3	<code>sudo dnf -y install rpm</code>	Installs an available RPM.

### Example

The following is an example of installing the **bfd** RPM:

```

bash-4.2$ dnf list installed | grep n9000
base-files.n9000                3.0.14-r74.2                installed
bfd.lib32_n9000                1.0.0-r0                    installed
core.lib32_n9000               1.0.0-r0                    installed
eigrp.lib32_n9000              1.0.0-r0                    installed
eth.lib32_n9000                1.0.0-r0                    installed
isis.lib32_n9000               1.0.0-r0                    installed
lACP.lib32_n9000               1.0.0-r0                    installed
linecard.lib32_n9000           1.0.0-r0                    installed
lldp.lib32_n9000               1.0.0-r0                    installed
ntp.lib32_n9000                1.0.0-r0                    installed
nxos-ssh.lib32_n9000           1.0.0-r0                    installed
ospf.lib32_n9000               1.0.0-r0                    installed
perf-cisco.n9000_gdb           3.12-r0                     installed
platform.lib32_n9000           1.0.0-r0                    installed
shadow-securetty.n9000_gdb     4.1.4.3-r1                  installed
snmp.lib32_n9000               1.0.0-r0                    installed
svi.lib32_n9000                1.0.0-r0                    installed
sysvinit-inittab.n9000_gdb     2.88dsf-r14                 installed
tacacs.lib32_n9000             1.0.0-r0                    installed
task-nxos-base.n9000_gdb       1.0-r0                      installed
tor.lib32_n9000                1.0.0-r0                    installed
vtp.lib32_n9000                1.0.0-r0                    installed
bash-4.2$ dnf list available
bgp.lib32_n9000                1.0.0-r0
bash-4.2$ sudo dnf -y install bfd

```



**Note** Upon switch reload during boot up, use the **rpm** command instead of **dnf** for persistent RPMs. Otherwise, RPMs initially installed using **dnf bash** or **install cli** shows `reponame` or `filename` instead of `installed`.

# Upgrading RPMs

## Before you begin

There must be a higher version of the RPM in the dnf repository.

## SUMMARY STEPS

1. `sudo dnf -y upgrade rpm`

## DETAILED STEPS

	Command or Action	Purpose
Step 1	<code>sudo dnf -y upgrade rpm</code>	Upgrades an installed RPM.

## Example

The following is an example of upgrading the **bfd** RPM:

```
bash-4.2$ sudo dnf -y upgrade bfd
```

# Downgrading an RPM

## SUMMARY STEPS

1. `sudo dnf -y downgrade rpm`

## DETAILED STEPS

	Command or Action	Purpose
Step 1	<code>sudo dnf -y downgrade rpm</code>	Downgrades the RPM if any of the dnf repositories has a lower version of the RPM.

## Example

The following example shows how to downgrade the **bfd** RPM:

```
bash-4.2$ sudo dnf -y downgrade bfd
```

## Erasing an RPM




---

**Note** The SNMP RPM and the NTP RPM are protected and cannot be erased.

You can upgrade or downgrade these RPMs. It requires a system reload for the upgrade or downgrade to take effect.

For the list of protected RPMs, see `/etc/dnf/protected.d/protected_pkgs.conf`.

---

### SUMMARY STEPS

1. `sudo dnf -y erase rpm`

### DETAILED STEPS

	Command or Action	Purpose
Step 1	<code>sudo dnf -y erase rpm</code>	Erases the RPM.

### Example

The following example shows how to erase the **bfd** RPM:

```
bash-4.2$ sudo dnf -y erase bfd
```

## Persistently Daemonizing an SDK- or ISO-built Third Party Process

Your application should have a startup bash script that gets installed in `/etc/init.d/application_name`. This startup bash script should have the following general format (for more information on this format, see <http://linux.die.net/man/8/chkconfig>).

```
#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
```



```
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
```

## Persistently Starting Your Application from the Native Bash Shell

- 
- Step 1** Install your application startup bash script that you created above into `/etc/init.d/application_name`
- Step 2** Start your application with `/etc/init.d/application_name start`
- Step 3** Enter `chkconfig --add application_name`
- Step 4** Enter `chkconfig --level 3 application_name on`
- Run level 3 is the standard multi-user run level, and the level at which the switch normally runs.
- Step 5** Verify that your application is scheduled to run on level 3 by running `chkconfig --list application_name` and confirm that level 3 is set to on
- Step 6** Verify that your application is listed in `/etc/rc3.d`. You should see something like this, where there is an 'S' followed by a number, followed by your application name (tcollector in this example), and a link to your bash startup script in `../init.d/application_name`

---

```
bash-4.2# ls -l /etc/rc3.d/tcollector
lrwxrwxrwx 1 root root 20 Sep 25 22:56 /etc/rc3.d/S15tcollector -> ../init.d/tcollector
bash-4.2#
```

## Copy Through Kstack

In Cisco NX-OS release 9.3(1) and later, file copy operations have the option of running through a different network stack by using the **use-kstack** option. Copying files through **use-kstack** enables faster copy times. This option can be beneficial when copying files from remote servers that are multiple hops from the switch. The **use-kstack** option work with copying files from, and to, the switch though standard file copy features, such as **scp** and **sftp**.



**Note** The `use-kstack` option does not work when the switch is running the FIPS mode feature. If the switch has FIPS mode that is enabled, the copy operation is still successful, but through the default copy method.

To copy through `use-kstack`, append the argument to the end of an NX-OS `copy` command. Some examples:

```
switch-1# copy scp://test@10.1.1.1/image.bin . vrf management use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf management
use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin . use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf default
use-kstack
switch-1#
```

The `use-kstack` option is supported for all NX-OS `copy` commands and file systems. The option is OpenSSL (Secure Copy) certified.

## An Example Application in the Native Bash Shell

The following example demonstrates an application in the Native Bash Shell:

```
bash-4.2# cat /etc/init.d/hello.sh
#!/bin/bash

PIDFILE=/tmp/hello.pid
OUTPUTFILE=/tmp/hello

echo $$ > $PIDFILE
rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
bash-4.2#
bash-4.2#
bash-4.2# cat /etc/init.d/hello
#!/bin/bash
#
# hello Trivial "hello world" example Third Party App
#
# chkconfig: 2345 15 85
# description: Trivial example Third Party App
#
### BEGIN INIT INFO
# Provides: hello
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Trivial example Third Party App
### END INIT INFO

PIDFILE=/tmp/hello.pid

# See how we were called.
```

```

case "$1" in
start)
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
stop)
    kill -9 `cat $PIDFILE`
    RETVAL=$?
;;
status)
    ps -p `cat $PIDFILE`
    RETVAL=$?
;;
restart|force-reload|reload)
    kill -9 `cat $PIDFILE`
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
bash-4.2#
bash-4.2# chkconfig --add hello
bash-4.2# chkconfig --level 3 hello on
bash-4.2# chkconfig --list hello
hello          0:off  1:off  2:on   3:on   4:on   5:on   6:off
bash-4.2# ls -al /etc/rc3.d/*hello*
lrwxrwxrwx 1 root root 15 Sep 27 18:00 /etc/rc3.d/S15hello -> ../init.d/hello
bash-4.2#
bash-4.2# reboot

```

### After reload

```

bash-4.2# ps -ef | grep hello
root      8790      1  0 18:03 ?          00:00:00 /bin/bash /etc/init.d/hello.sh
root      8973    8775  0 18:04 ttyS0    00:00:00 grep hello
bash-4.2#
bash-4.2# ls -al /tmp/hello*
-rw-rw-rw- 1 root root 205 Sep 27 18:04 /tmp/hello
-rw-rw-rw- 1 root root   5 Sep 27 18:03 /tmp/hello.pid
bash-4.2# cat /tmp/hello.pid
8790
bash-4.2# cat /tmp/hello
Sun Sep 27 18:03:49 UTC 2015
Hello World
Sun Sep 27 18:03:59 UTC 2015
Hello World
Sun Sep 27 18:04:09 UTC 2015
Hello World
Sun Sep 27 18:04:19 UTC 2015
Hello World
Sun Sep 27 18:04:29 UTC 2015
Hello World
Sun Sep 27 18:04:39 UTC 2015
Hello World
bash-4.2#

```

