Run commands sequentially across a cluster from a UNIX server, Part 1: Secure Shell (SSH)

Installing and configuring ssh

Level: Introductory

Harish Chauhan (hchauhan@in.ibm.com), Linux Architect, IBM

15 Aug 2006
Updated 21 Sep 2006

Configure Secure Shell (SSH) on IBM System p™ and System x™ computers so the UNIX® server can access a remote server without a password.

Note: This article is strictly for beginning UNIX users and administrators. Experts will likely already know several ways of accomplishing this task.

Introduction: Executing a remote command on multiple computers

When you hear someone referring to a shell, do you know what that really entails? It basically means that you can open a terminal session on any UNIX® machine, where you type your commands to perform an activity, such as `useradd username`, `passwd username`, `system-config-printers`, and so on. This shell is local to your machine, and whatever command you execute is performing activity on your local machine.

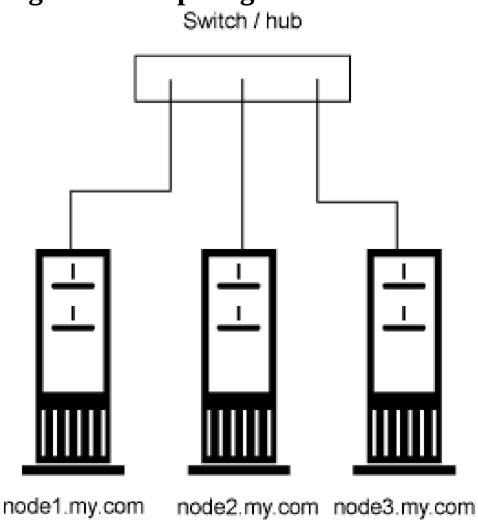
However, what if you want to execute a command on a set of machines, such as setting the date on all of the machines in the network. One way is to log in individually to each machine and execute the `date` command, one machine at a time. A better way would be to set it automatically, where you log in to a server and have that machine set the date on the rest of the machines serially. To set the date remotely, you need tools like Secure Shell (SSH), or remote shell (rsh), installed on your machines. Usually, when you try to access the remote machines, you will be prompted for a password to make sure that you are an authorized user. However, you can configure SSH and rsh in your environment to bypass password verification. In this article, you'll concentrate on how to configure SSH in your environment. In Part 2, you'll look at rsh.

Hardware, software, and setup

Use the following hardware and software, to perform the tasks described in this article:

- IBM System p™ and System x™ servers, such as System p520 or System x345
- Red Hat Enterprise Linux® Version 4.0 Update 3 (RHEL Version 4.0 Update 3)

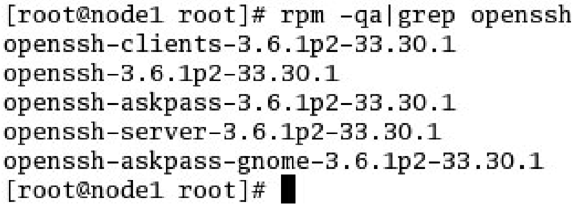
Figure 1. Setup diagram



Now follow these steps:

1. Install RHEL Version 4.0 Update 3 on all machines in the environment, such as node1.my.com, node2.my.com, and node3.my.com, as shown in Figure 1. Note that any of the nodes can be System p, or System x, servers.
2. Make sure `openssh` Red Hat Package Manager (RPM) is installed on all your machines, as shown in Figure 2.

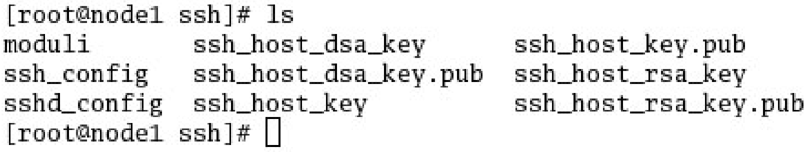
Figure 2. openssh RPMs



```
[root@node1 root]# rpm -qa|grep openssh
openssh-clients-3.6.1p2-33.30.1
openssh-3.6.1p2-33.30.1
openssh-askpass-3.6.1p2-33.30.1
openssh-server-3.6.1p2-33.30.1
openssh-askpass-gnome-3.6.1p2-33.30.1
[root@node1 root]# █
```

3. If you already have `openssh` installed, you will find the `/etc/ssh` directory on your machine, as shown in Figure 3. This directory holds all SSH-related configuration files. You can customize `sshd` by modifying the files here, but I'm not going to cover this in detail in this article.

Figure 3. Installed path, that is /etc/ssh



```
[root@node1 ssh]# ls
moduli       ssh_host_dsa_key       ssh_host_key.pub
ssh_config   ssh_host_dsa_key.pub   ssh_host_rsa_key
sshd_config  ssh_host_key           ssh_host_rsa_key.pub
[root@node1 ssh]# 
```

4. If you don't have `openssh` installed, then install it from the RHEL Version 4.0 Update 3 CDs using the following commands: `#rpm -ivh openssh-*` or `#system-config-packages`.

Configuring for root and standard users

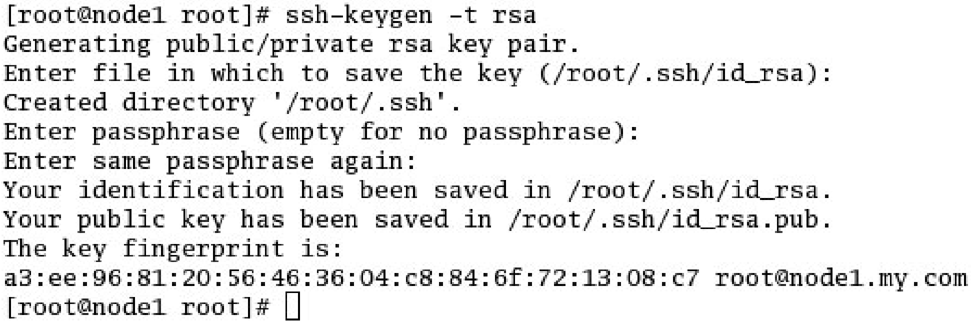
You have the following two different types of configurations to consider:

- `Root` user
- Standard user, in this case `myuser`

Let's first consider configuring SSH for the `Root`user. To configure the `Root`user, follow these steps:

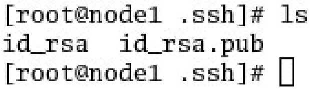
1. Generate the public and private key pairs. In order to generate the key pairs, you have to execute the `ssh-keygen` command, as shown in Figure 4. **Note:** `ssh-keygen` prompts you to set a passphrase, but you just continue by pressing the **Enter** key. As shown in Figure 5, the `.ssh` folder gets created in the `/root` folder, which holds the generated public (id_rsa.pub) and private (id_rsa) keys.

Figure 4. ssh-keygen



```
[root@node1 root]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
a3:ee:96:81:20:56:46:36:04:c8:84:6f:72:13:08:c7 root@node1.my.com
[root@node1 root]# 
```

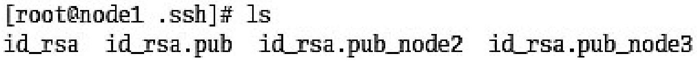
Figure 5. Generated private and public keys pair



```
[root@node1 .ssh]# ls
id_rsa  id_rsa.pub
[root@node1 .ssh]# 
```

2. Repeat the above step for every machine participating in your environment; public and private keys are different for each machine. **Note:** The generated public and private keys don't match even if you execute the `ssh-keygen` multiple times on the same machine.
3. Once you have executed `ssh-keygen` on all the nodes, you can collect the generated `id_rsa.pub` key from each machine. You can use any method for collecting the `id_rsa.pub` keys, including a floppy drive, USB device, FTP, and so forth.
4. In this step, I have assumed you have copied all the public keys in the `/root/.ssh` folder on `node1.my.com`, as shown in Figure 6, where `id_rsa.pub_node2` is the public key of node2.my.com and `id_rsa.pub_node3` is the public key of node3.my.com. Basically, you append the contents of all three files in one file using the `cat` command.

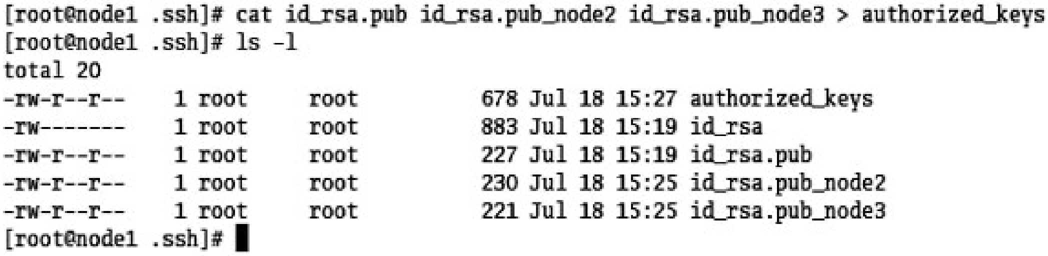
Figure 6. Collected id_rsa.pub keys



```
[root@node1 .ssh]# ls
id_rsa  id_rsa.pub  id_rsa.pub_node2  id_rsa.pub_node3
```

5. Now concatenate the contents of all the collected public keys in a file known as `authorized_keys`, as shown in Figure 7 below, and place the file in the `/root/.ssh` folder. **Note:** The file must be named as `authorized_keys`. Any other name will not work.

Figure 7. Create authorized_keys file

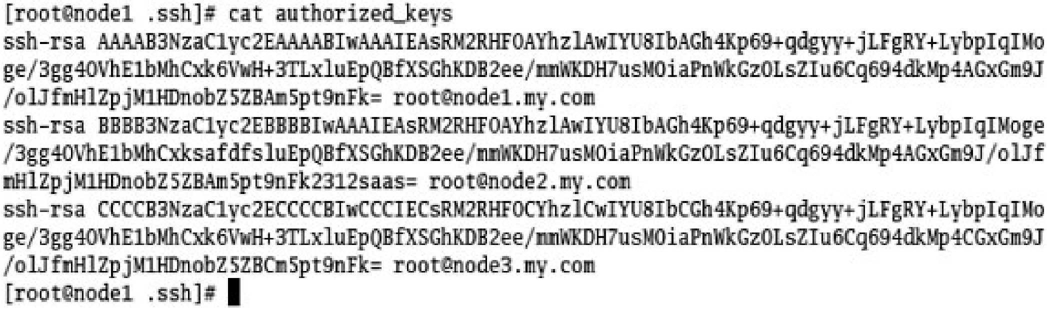


```
[root@node1 .ssh]# cat id_rsa.pub id_rsa.pub_node2 id_rsa.pub_node3 > authorized_keys
[root@node1 .ssh]# ls -l
total 20
-rw-r--r--    1 root     root          678 Jul 18 15:27 authorized_keys
-rw-------    1 root     root          883 Jul 18 15:19 id_rsa
-rw-r--r--    1 root     root          227 Jul 18 15:19 id_rsa.pub
-rw-r--r--    1 root     root          230 Jul 18 15:25 id_rsa.pub_node2
-rw-r--r--    1 root     root          221 Jul 18 15:25 id_rsa.pub_node3
[root@node1 .ssh]# 
```

Note: authorized_keys2 works as well, and only for SSH protocol Version 2.

6. The final contents of the `authorized_keys`file will look like the code shown in Figure 8. **Note:** Each stanza shown in Figure 8 corresponds to one machine in your environment.

Figure 8. Contents of authorized_keys file



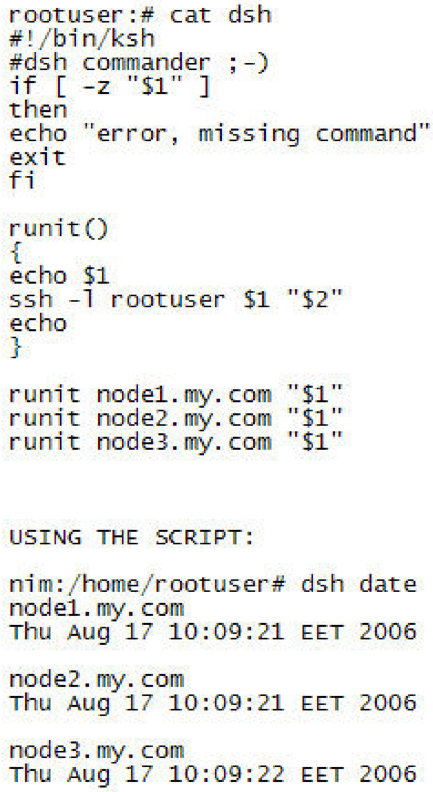
```
[root@node1 .ssh]# cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAsRM2RHFOAYhzlAwIYU8IbAGh4Kp69+qdgyy+jLFgRY+LybpIqIMo
ge/3gg4OVhE1bMhCxk6VwH+3TLxluEpQBfXSGhKDB2ee/mmWKDH7usMOiaPnWkGzOLsZIu6Cq694dkMp4AGxGm9J
/olJfmHlZpjM1HDnobZ5ZBAm5pt9nFk= root@node1.my.com
ssh-rsa BBBB3NzaC1yc2EBBBBIwAAAIEAsRM2RHFOAYhzlAwIYU8IbAGh4Kp69+qdgyy+jLFgRY+LybpIqIMoge
/3gg4OVhE1bMhCxksafdfsluEpQBfXSGhKDB2ee/mmWKDH7usMOiaPnWkGzOLsZIu6Cq694dkMp4AGxGm9J/olJf
mHlZpjM1HDnobZ5ZBAm5pt9nFk2312saas= root@node2.my.com
ssh-rsa CCCCB3NzaC1yc2ECCCCBIwCCCIECsRM2RHFOCYhzlCwIYU8IbCGh4Kp69+qdgyy+jLFgRY+LybpIqIMo
ge/3gg4OVhE1bMhCxk6VwH+3TLxluEpQBfXSGhKDB2ee/mmWKDH7usMOiaPnWkGzOLsZIu6Cq694dkMp4CGxGm9J
/olJfmHlZpjM1HDnobZ5ZBCm5pt9nFk= root@node3.my.com
[root@node1 .ssh]# 
```

7. Finally, you are ready to copy the `authorized_keys` file into the `/root/.ssh` folder on each machine where you would like to log in without receiving a password prompt. You can use any standard method to copy the file to each machine, including a floppy drive, USB device, FTP, and so on.
8. Once the `authorized_keys` file has been copied to all the machines, you can test your setup by executing the following command:

```
# ssh node2.my.com date
```

If everything has been done correctly, you should see the date output from `node2.my.com` without being prompted for a password.

Figure 9. Sample script



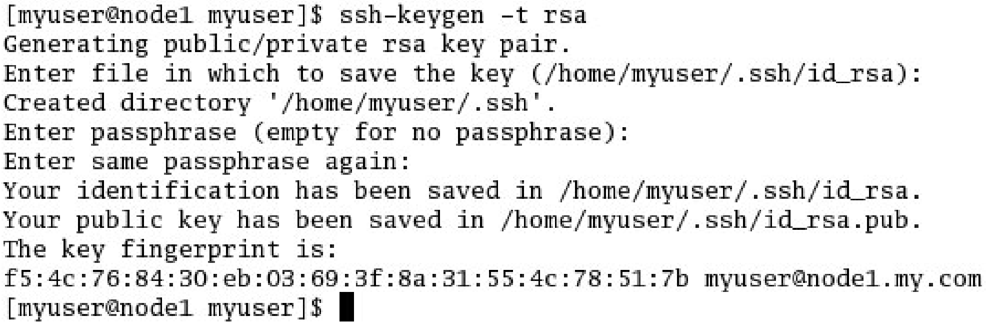
```
rootuser:# cat dsh
#!/bin/ksh
#dsh commander ;-)
if [ -z "$1" ]
then
echo "error, missing command"
exit
fi

runit()
{
echo $1
ssh -l rootuser $1 "$2"
echo
}

runit node1.my.com "$1"
runit node2.my.com "$1"
runit node3.my.com "$1"


USING THE SCRIPT:

nim:/home/rootuser# dsh date
node1.my.com
Thu Aug 17 10:09:21 EET 2006

node2.my.com
Thu Aug 17 10:09:21 EET 2006

node3.my.com
Thu Aug 17 10:09:22 EET 2006
```

Next, you should consider configuring SSH for any standard user, in this case, `myuser`.

To do this, make the assumption that user name, `myuser`, exists on all the nodes. You want to make sure `myuser` is able to execute the command without any password prompt. For example:

- Log in to any computer in the system as `myuser`.
- Execute the `#ssh-keygen -t rsa` command, which generates the public and private keys for `myuser` user in the `/home/myuser/.ssh` folder, as shown in Figure 10.

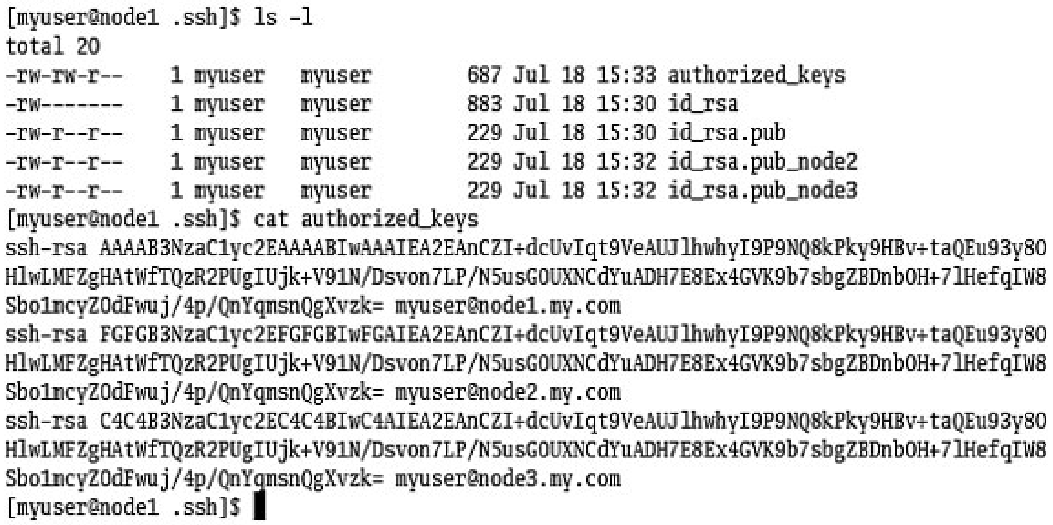
Figure 10. User public and private keys



```
[myuser@node1 myuser]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/myuser/.ssh/id_rsa):
Created directory '/home/myuser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/myuser/.ssh/id_rsa.
Your public key has been saved in /home/myuser/.ssh/id_rsa.pub.
The key fingerprint is:
f5:4c:76:84:30:eb:03:69:3f:8a:31:55:4c:78:51:7b myuser@node1.my.com
[myuser@node1 myuser]$ █
```

- Execute `ssh-keygen` on all the nodes as the `myuser` user.
- Collect all the public keys and create the `authorized_keys` file, as explained in Step 5 for the `root` user above. Refer to Figure 11 for the contents of `authorized_keys`.

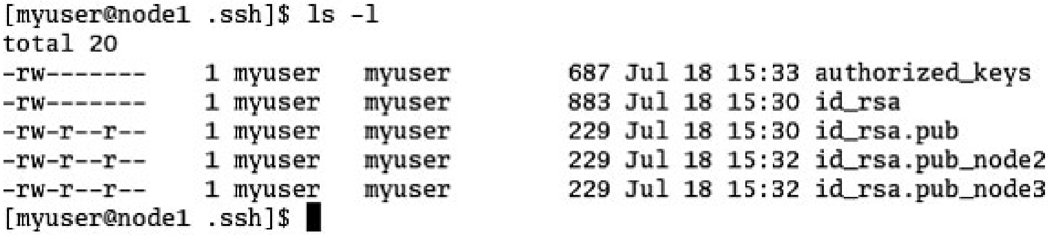
Figure 11. User authorized_keys file



```
[myuser@node1 .ssh]$ ls -l
total 20
-rw-rw-r--    1 myuser   myuser       687 Jul 18 15:33 authorized_keys
-rw-------    1 myuser   myuser       883 Jul 18 15:30 id_rsa
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:30 id_rsa.pub
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:32 id_rsa.pub_node2
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:32 id_rsa.pub_node3
[myuser@node1 .ssh]$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEA2EAnCZI+dcUvIqt9VeAUJlhwhyI9P9NQ8kPky9HBv+taQEu93y80
HlwLMFZgHAtWfTQzR2PUgIUjk+V91N/Dsvon7LP/N5usGOUXNCdYuADH7E8Ex4GVK9b7sbgZBDnbOH+7lHefqIW8
Sbo1mcyZOdFwuj/4p/QnYqmsnQgXvzk= myuser@node1.my.com
ssh-rsa FGFGB3NzaC1yc2EFGFGBIwFGAIEA2EAnCZI+dcUvIqt9VeAUJlhwhyI9P9NQ8kPky9HBv+taQEu93y80
HlwLMFZgHAtWfTQzR2PUgIUjk+V91N/Dsvon7LP/N5usGOUXNCdYuADH7E8Ex4GVK9b7sbgZBDnbOH+7lHefqIW8
Sbo1mcyZOdFwuj/4p/QnYqmsnQgXvzk= myuser@node2.my.com
ssh-rsa C4C4B3NzaC1yc2EC4C4BIwC4AIEA2EAnCZI+dcUvIqt9VeAUJlhwhyI9P9NQ8kPky9HBv+taQEu93y80
HlwLMFZgHAtWfTQzR2PUgIUjk+V91N/Dsvon7LP/N5usGOUXNCdYuADH7E8Ex4GVK9b7sbgZBDnbOH+7lHefqIW8
Sbo1mcyZOdFwuj/4p/QnYqmsnQgXvzk= myuser@node3.my.com
[myuser@node1 .ssh]$ █
```

- Finally, copy the `authorized_keys` file into the `/home/myuser/.ssh` folder on all the machines and set the permission as `600`, as shown in Figure 12, using the `$chmod 600 /home/myuser/.ssh/authorized_keys` command. This sets the access permission to 600, which means that only the owner or `myuser` has permission to read and write this file. No one else can modify it.

  Figure 12. Permission on authorized_keys file

```
[myuser@node1 .ssh]$ ls -l
total 20
-rw-------    1 myuser   myuser       687 Jul 18 15:33 authorized_keys
-rw-------    1 myuser   myuser       883 Jul 18 15:30 id_rsa
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:30 id_rsa.pub
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:32 id_rsa.pub_node2
-rw-r--r--    1 myuser   myuser       229 Jul 18 15:32 id_rsa.pub_node3
[myuser@node1 .ssh]$ █
```

- Test your environment by executing the `$ssh node3.my.com date` command. Doing so should return the date output of `node3.my.com`.

Conclusion: Saving time, providing flexibility

In this article, you learned how to configure SSH in your environment so that you can perform activities more easily and quickly. This not only helps in saving time, but it also gives you flexibility to perform activities serially on more machines automatically. Part 2 concentrates on configuring rsh, another way of executing serial commands in your environment when security is not of prime importance.

Resources

Learn

- "Run commands sequentially across a cluster from a UNIX server, Part 2" (developerWorks, August 2006): Learn how to configure remote shell (rsh) on IBM System p and System x computers.

- OpenSSH: Visit this site to learn more about OpenSSH.

- Configuring OpenSSH: Learn how to confirm OpenSSH for public key authentication.

- HOWTOs on Linux: Visit this site to learn more about Linux, or solve an issue.

- IBM Systems: Want more? The developerWorks IBM Systems zone hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials.

- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.

Get products and technologies

- IBM trial software: Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the IBM Systems forums, developerWorks blogs, and get involved in the developerWorks community.

About the author

Harish has been with IBM since 1998 and has 14 years of experience. During his last eight years with IBM, he has spent five years at the India Research Lab and one year at the IBM T.J.Watson Research Center. Harish has been leading the Linux Center of Competency in Bangalore, India for the past two years. You can contact him at hchauhan@in.ibm.com.