Migrating a Linux server from the command line - running the sync

With our preparations done it's time to start copying files between your Linux servers.

Let's move

In the **previous article** we looked at the conditions we want to match on the origin and destination servers and ways to prepare the origin server for a more efficient migration. Now we'll start the actual process of migrating your Linux server.

Using rsync

With the prep work done let's look at the tool we'll use to copy the files: rsync.

The rsync program lets you copy a directory from one server to another over an SSH connection. It can optionally delete any files in the destination folder that aren't in the original folder and lets you skip over subdirectories you don't want to copy. Best of all, rsync only copies files that are different in the origin and destination folders. You can run an rsync more than once without worrying about unnecessary overhead from repeat copies.

We'll leverage all those features of rsync. The trick lies in knowing what files and directories need to be left alone on the destination server. Then we can let rsync copy the rest.

Rsync excludes

Luckily rsync can work with an "exclude file" - a list of directories and files that rsync should overlook when performing a sync.

You'll use the exclude file to list files and directories specific to the original server that wouldn't be meaningful if copied or that might change during the copy if your server is live during the migration. You just need to create a text file with the list of those items, one on each line.

Do consider your excludes carefully. You don't want to sync any files that reference or could affect the IP address of the destination server because they could make it difficult to connect to the server or prevent services from starting.

Exclude example

A simple exclude file that covers a variety of Linux distributions can look like this:

```
/boot
/proc
/sys
/tmp
/dev
/var/lock
/etc/fstab
/etc/mdadm.conf
/etc/mtab
/etc/resolv.conf
```

```
/etc/resolv.conf
/etc/conf.d/net
/etc/network/interfaces
/etc/networks
/etc/sysconfig/network*
/etc/sysconfig/hwconf
/etc/sysconfig/ip6tables-config
/etc/sysconfig/kernel
/etc/hostname
/etc/HOSTNAME
/etc/hosts
/etc/modprobe*
/etc/modules
/etc/udev
/net
/lib/modules
/etc/rc.conf
```

Note that it's okay if you list something in the exclude file that doesn't exist on your system. Rsync won't care because rsync was going to ignore that item anyway.

System files

The most important excludes are the ones that reference hardware and system files that keep track of the server's status while it's running. That's why directories like /boot, /proc, and /dev are essential excludes. The destination server should already have those directories and config files in place and they shouldn't be overwritten.

Databases

If you have a database that's being regularly accessed and changed you'll want to leave it out of the sync until the very end. Otherwise your rsync runs will need to recopy the database each time. Get the rest of the system out of the way then copy the database at the end (preferably after you've stopped the database service).

Application directories

The same holds true for directories that hold application state files or cache files. If they change often they introduce extra work for the first sync. Best to put them in the exclude file at first then remove them when you're ready to complete the migration.

Do a dry run

Now we test rsync to make sure everything is in order. Make note of the port sshd is listening to on the destination server as well as the name of the exclude file.

If ssh is running on port 30000 and you named your exclude file "exclude.txt" in the home directory of user "demo", you would kick off the sync test by running:

```
sudo rsync -e 'ssh -p 30000' --dry-run -azPx --delete-after --exclude-from="/home/demo/exclude.txt" / root@1.2.3.4:/
```

Real short summary of the above command: It pretends to copy everything from "/" on the original server to the same place on the destination server so you can make sure it doesn't have a problem making the connection.

Change the "ssh -p 30000' part to match the destination's ssh port.

The "--dry-run" flag is the bit that tells rsync to go through the motions but not actually copy anything yet.

The "-azPx" stuff tells rsync to try and keep everything the same as it was on the original server, compress it in transit to save bandwidth, show its progress during the copy, keep partially-transferred files if interrupted, and don't cross filesystems (e.g. don't copy files from remotely mounted filesystems).

The "--delete-after" flag tells rsync not to delete anything from the destination until all the copying is done.

The "--exclude-from" value tells rsync to use your exclude file, and where to find it.

The last two items are the actual arguments to rsync. First is the directory we're copying from and the following argument is the destination. In our case the origin is "/". The destination is in the form "username@ip-address:destination-directory". So the above command would connect to the server at 1.2.3.4 as root then copy the files using "/" as the destination for the files. Don't forget the colon.

Rsync will ask you for the password to connect just as it would if you were trying to ssh to the destination server (which, technically, you are).

Once it's done look over the list of directories to make sure it doesn't plan on copying anything you should exclude.

Start the syncing

If all went well (as in, there were no errors you need to puzzle out) we're ready to start copying. Again using ssh on port 30000 for the example, we would run:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after --exclude-from="/home/demo/exclude.txt" / root@1.2.3.4:/
```

Note that it's the same command as before without the "--dry-run" flag.

Now...wait. How long it will take depends on factors like how much disk space you're using and how many files are being transferred. You'll see a list of what's being copied scroll down the screen so you can get an idea of the sync's progress.

Should the sync be interrupted you can just run the same command again and it will pick up where it left off. Make sure you don't reboot the destination server until you're completely done with the copy. A partially-migrated server will probably get very confused if it tries to reboot.

Finalizing the migration

Doing one pass got the more static files over to the new server. That's good, but that may not be everything you needed to copy. An active database or a directory where a lot of files get created and deleted may have been copied but changed already. Or, more practically, they may have been in your exclude file.

To get those last files you'll want to bring the services that use them down. That way you know the entire file will be migrated with no changes made after your last sync.

Edit your exclude file to remove references to the busy files you were holding off on then run the rsync command again.

Once you're satisfied that you've got a full sync done take a last look around the destination server. Make any necessary tweaks to the new server. For example, if you use our sample exclude file the hostname of the destination server won't change. If you need to set that to match the original now would be the time to make that manual edit.

If you're running Arch Linux check the /etc/rc.conf file. Since that file contains networking configuration we put it in our sample exclude file so it wouldn't be migrated. You'll need to manually edit rc.conf to set the services that will start at boot time (the "DAEMONS" value).

Boot the new server

It's time to reboot the destination server and see how it looks. Ideally it will look just like the original server and will be running the same services when it comes up.

Note that the new server will be using the same sshd configuration as the old server as well as the same SSH host signature. You probably need to delete any saved key for the new server from the "~/.ssh/known_hosts" file or equivalent on your desktop's operating system. Otherwise you'll get a warning about a changed server key or you might be prevented from connecting at all.

The users and passwords on the new server will also match those of the old server. That includes changing the root password.

Thorough testing is definitely advised if you're migrating a production server. Once your testing is done you can change your DNS to point to the new server and you'll be finished. Nice work!

Summary

In case it wasn't emphasized enough, test the new server before putting it into production. We've tried to cover all the possibilities but it's always possible a network config file slipped through or that one of the files you moved referred to the old IP address and needs to be updated. It's good to catch those issues before letting the rest of the world access your services.

The **next article** examines alternative approaches to server migrations in case a live rsync approach isn't the best approach for your environment.

-- Jered

Article Comments:

Kevin Keane commented Sat May 07 07:12:35 UTC 2011:

> I found that a few more excludes and SSH parameters are useful here.
> The -e 'none' argument turns of SSH escape keys. Without it, some byte sequences rsync sends can be interpreted as escape characters.
> Also make sure that the SSH server allows root connections.
> rsync \ -e "ssh -i $dirname/sync-key -e 'none'" \ -azPx \ -v \ --exclude-from="$dirname/exclude.txt" \ --include-from="$dirname/include.txt" \ root@server:/ /
> I use both an exclude and include text file, and install the correct packages in advance. This way, you can sometimes get away with migrating to a different OS version (for instance, from CentOS 5.5 to 5.6) without clobbering packages too badly.
> exclude.txt:
> /boot /bin /sbin /lib /lib64 /proc /sys /tmp /dev /usr /var/lock /var/log /var/tmp /var/named /etc/fstab /etc/mtab /etc/ld.so* /etc/lvm /etc/resolv.conf /etc/conf.d/net /etc/network/interfaces /etc/networks /etc/rc.d/rc?.d /etc/yum.repos.d /etc/sysconfig/network* /etc/sysconfig/hwconf /etc/sysconfig/ip6tables-config /etc/sysconfig/kernel /etc/hostname /etc/HOSTNAME /etc/hosts /etc/modprobe* /etc/modules /etc/makedev* /etc/iproute2 /net /lib/modules /etc/rc.conf
> The include.txt file is necessary to copy /usr/local to the new machine.
> / /usr/local

Raif Atef commented Sun Dec 11 10:46:04 UTC 2011:

> Don't forget to add /etc/mdadm.conf to the exclude list or you will mess up your software RAID and get an unbootable system !

Jered commented Fri Dec 16 03:00:14 UTC 2011:

> Ooh, good point. This article was admittedly written from a VPS-centric point of view, so it can definitely use some updates to cover more environments. I'll try to revisit this in more detail soon, but for now I'll definitely add mdmadm.conf to the article proper.

Dave commented Thu Jan 05 00:45:28 UTC 2012:

> You'll probably want to add: /etc/udev/rules.d/70-persistent-net.rules To your excludes list. Otherwise your ethernet devices might change names so eth0 because eth1 and upon reboot you might loose ssh access because eth1 is not configured.

Jered commented Thu Jan 05 16:13:57 UTC 2012:

> Thanks Dave. Added /etc/udev to the list, since it's probably best to avoid syncing that directory entirely.

Benjamin commented Thu Mar 01 21:28:36 UTC 2012:

> Great great article, this is exactly what I needed, and it works great. However, I have a problem when trying to copy the /home directory, which is on a separate partition. I've removed the -x from rsync (with the -x it syncs correctly and completes, excluding the /home directory) but without it just sits there indefinitely once it has echo'd 'xxxxxx files to consider'. Any tips on what this could be?

Benjamin commented Fri Mar 02 17:26:49 UTC 2012:

> Okay, scratch that previous comment! For anyone wondering, if the command pauses for a while (>~60m) after counting up the 'files to consider', don't panic, it's still working. What it's doing, I don't know, but it's working so no worries :)

Pierre commented Wed Oct 24 06:44:53 UTC 2012:

> You forgot something VERY IMPORTANT: --numeric-ids

Jered commented Tue Nov 27 20:34:01 UTC 2012:

> Since this article is aimed primarily at migrating to a fresh new server, numeric-ids wouldn't be an important option to use. Most of the time users either wouldn't have been created yet or would have been created with the same names.
> It's good to be aware of the option though, especially if the target is a server that's seen some use. The --numeric-ids option tells rsync to only map ownership by the numeric values of users and groups. Without that option, rsync looks for a user or group of the same name on the target server and uses that as a priority for mapping. If the user or group doesn't exist on the target then rsync uses the numeric ID anyway.

Want to comment?

Name:

Email Address: (not made public)

Website: (optional)

Comment: (use plain text or Markdown syntax)

Post comment

Tags: admin apache api arch at awstats backup capistrano centos cloud courier cron dapper debian dig django dns dstat ebook email fail2ban failover fedora feisty forensics ftp gentoo git gutsy ha hardy heartbeat ibex intrepid iotop iptables jaunty karmic kernel lenny logrotate logs lucid mail maverick migration mod_rails mongrel monitoring munin mysql nagios network nginx ntp open relay passenger permissions php postfix postgresql pv-grub rails rescue rescue mode resize rhel rootkit rsync security setup sftp shorewall slice slice administration slice manager slicemanager spam spamhaus spf ssh ssl subversion tar tcpdump telnet thin tomcat ubuntu untar upgrade vhosts windows winscp