rsync

Making backups with rsync

Davis Claiborne

LUG @ NC State

November 10, 2020

Linux Users Group
at NC State University

What I want from my backups

- Fast and easy to do
- Easy to access
- Full control

What I want from my backups

- Fast and easy to do
- Easy to access
- Full control

- Before I get into how I make my backups, I want to talk a bit about what I want to get out of my backups
- I have three main desires: the process of making a backup should be fast and easy to do, I should be able to easily access the files once they're backed up, and I want to have full control over what is and isn't backed up

What I want from my backups

- Fast and easy to do
  - $<$ 15 minutes to finish
  - Run in background
- Easy to access
  - Plug and play
- Full control
  - No wasted space

What I want from my backups

- Fast and easy to do
  - < 15 minutes to finish
  - Run in background
- Easy to access
  - Plug and play
- Full control
  - No wasted space

- Let me be a bit more specific: when I say I want my backups to be fast and easy to do, I want to be able to start a backup and have it done in less than 15 minutes.
- Ideally, it should be able to just run in the background without me noticing and be over pretty quickly.
- When I say it should be easy to access, I should be able to plug in my backup medium and access files immediately
- Finally, by having full control, I mean that I don't want to waste time or space backing up files I don't care about

Some caveats

- Just *my* setup - might not work for you
- Probably can adapt it
- Some issues

Some caveats

- Just *my* setup - might not work for you
- Probably can adapt it
- Some issues

- Obviously this should go without saying, but this is just the way that I do this
- There's sure to be some kind of issue with my setup that might not work with the way you do things
- That being said, with a bit of effort you can probably adapt my script to meet your needs
- There are some issues with it, though, that I'm not sure how to address - I'll get to these later

What is rsync?

- Copies files
- Originally for syncing with remote systems
- Delta-based changes

What is rsync?

- Copies files
- Originally for syncing with remote systems
- Delta-based changes

- rsync is tool for copying files
- Originally intended to be used for remote connections (like scp)
- Because it was originally intended for use with remote systems, one of its goals was to only change files that it needed to
- It does this by a) only updating files that are newer on the source system and b) doing ”delta-based” changes - only changing what is needed
- This makes it much faster than just blindly re-copying every file every time

The script

```
# Gets all native packages.
echo Storing native packages
native_packages="$(pacman --query --explicit --native --quiet --unrequired)"
format_package_list "$native_packages" > $backup_dir/pacman-native-packages.txt

# Gets a list of all foreign packages.
echo Storing foreign packages...
foreign_packages="$(pacman --query --explicit --foreign --quiet --unrequired)"
echo "$foreign_packages" > $backup_dir/pacman-foreign-packages.txt

# Performs backup
echo Performing backup
rsync --archive --verbose --human-readable --progress --cvs-exclude \
      --exclude='/VirtualBox VMs' \
      --exclude='/temp' \
      --exclude='/dotfiles' \
      --exclude='.*' \
      --exclude='_minted*' \
      --exclude='CMakeFiles' \
      --exclude='__pycache__' \
      $directory $backup_dir
```

See full script here [2]

```
# Gets all native packages.
echo Storing native packages
native_packages="$(pacman --query --explicit --native --quiet --unrequired)"
format_package_list "$native_packages" > $backup_dir/pacman-native-packages.txt

# Gets a list of all foreign packages.
echo Storing foreign packages...
foreign_packages="$(pacman --query --explicit --foreign --quiet --unrequired)"
echo "$foreign_packages" > $backup_dir/pacman-foreign-packages.txt

# Performs backup
echo Performing backup
rsync --archive --verbose --human-readable --progress --cvs-exclude \
      --exclude='/VirtualBox VMs' \
      --exclude='/temp' \
      --exclude='/dotfiles' \
      --exclude='.*' \
      --exclude='_minted*' \
      --exclude='CMakeFiles' \
      --exclude='__pycache__' \
      $directory $backup_dir
```

See full script here [2]

- Here's the main portion of my backup script
- You can read the rest of the script online, but it's mostly just simple flag parsing, setting variables, and other misc functions
- It's basically doing two things here: storing all the packages I have installed and starting the actual backup
- Let's take a look at a few of these flags:
  - `archive`: Basically, keep all files, permissions, etc.
  - `verbose`: Output lots of information
  - `human-readable`: Make numbers easier to read
  - `progress`: Show progress on file updates
  - `cvs-exclude`: Ignore most temp programming files
  - `excludes`: Keep out stuff I don't want backed up
- `archive` is the most useful flag: it tells rsync to copy all the files (and the structure) of the source to the dest

The script

```
# Gets all native packages.
echo Storing native packages
native_packages="$(pacman --query --explicit --native --quiet --unrequired)"
format_package_list "$native_packages" > $backup_dir/pacman-native-packages.txt

# Gets a list of all foreign packages.
echo Storing foreign packages...
foreign_packages="$(pacman --query --explicit --foreign --quiet --unrequired)"
echo "$foreign_packages" > $backup_dir/pacman-foreign-packages.txt

# Performs backup
echo Performing backup
rsync --archive --verbose --human-readable --progress --cvs-exclude \
      --exclude='/VirtualBox VMs' \
      --exclude='/temp' \
      --exclude='/dotfiles' \
      --exclude='.*' \
      --exclude='_minted*' \
      --exclude='CMakeFiles' \
      --exclude='__pycache__' \
      $directory $backup_dir
```

See full script here [2]

The script

```
# Gets all native packages.
echo Storing native packages
native_packages="$(pacman --query --explicit --native --quiet --unrequired)"
format_package_list "$native_packages" > $backup_dir/pacman-native-packages.txt

# Gets a list of all foreign packages.
echo Storing foreign packages...
foreign_packages="$(pacman --query --explicit --foreign --quiet --unrequired)"
echo "$foreign_packages" > $backup_dir/pacman-foreign-packages.txt

# Performs backup
echo Performing backup
rsync --archive --verbose --human-readable --progress --cvs-exclude \
      --exclude='/VirtualBox VMs' \
      --exclude='/temp' \
      --exclude='/dotfiles' \
      --exclude='.*' \
      --exclude='_minted*' \
      --exclude='CMakeFiles' \
      --exclude='__pycache__' \
      $directory $backup_dir
```

See full script here [2]

- Let's explain some of the excludes a bit
- The syntax is pretty similar to, say, a `.gitignore` file
- The leading slash on the first 3 means to only ignore those files at the root level
- The rest of the file are ignored everywhere and follow standard shell expansion options (`.*` for all files/directories that start with a ., for example)
- Most of them are pretty obvious: don't include temp files, output from compilation processes, etc.
- But why not dotfiles? The answer is pretty simple: if I care about them, they're already backed up in my dotfiles repo

Pros

- Fast
- Easy
- Painless
- Flexible

Pros
- Fast
- Easy
- Painless
- Flexible

- With that being said, I'll close with what I like and dislike about this setup
- The main things I like about it are how quick, easy, and painless it is to use: I just plug in my flash drive, run the script, and don't think about it
- One other advantage is that it's super easy to swap where/how you store your backups without changing anything else. Rsync even has a flag specifically for syncing with remote servers that compresses the data before transmitting it

Cons

- Duplicated files/buildup
- Not two-way
- Issues scaling

Cons

- Duplicated files/buildup
- Not two-way
- Issues scaling

- There are some downsides, however
- One big problem with it is that files can easily get duplicated: because of the way rsync works, it doesn't attempt to detect if a file has moved versus it just being a new file
- This means that if you move a file between syncs, you can get multiple copies of the same file
- This also means that files you've deleted stay on your backup. Most of the time this is useful, but it can lead to certain scenarios where deleted files (you no longer need) take up most of your backup
- It's also not a two-way backup. This means that you can only modify one of the sources. This isn't a big deal for most use cases, but it would be nice to have for some scenarios (for instance, syncing music/pictures between phone, where I may edit tags, etc.)

Cons

- Duplicated files/buildup
- Not two-way
- Issues scaling

Cons

- Duplicated files/buildup
- Not two-way
- Issues scaling

- Finally, there's the issue of size: what happens if your backup size outgrows your storage device? Obviously, you can always buy a larger drive, but it'd be nice to be able to have it split itself across multiple drives automatically or something
- This solution, unfortunately, doesn't handle things like this very well natively.

Possible solutions

- Duplicate files: `fdupes` [3]
- File buildup: Manual moving
- Two-way: `Unison` [4]
- Scaling:
  - Filesystem
  - Virtual filesystem
  - Manual partitioning

Possible solutions

- Duplicate files: `fdupes` [3]
- File buildup: Manual moving
- Two-way: `Unison` [4]
- Scaling:
  - Filesystem
  - Virtual filesystem
  - Manual partitioning

- While these are issues, most of them can be remedied, potentially by using some other program
- For instance, the problem of duplicated files can be fixed by listing duplicate files with fdupes. I prefer to have the output sorted by date so I don't have to sift through the same (intentional) duplicates every time
- To fix file buildup, my main solution is just manually going through and deleting files that I know I won't need any more. But you could do something more sophisticated, though, like deleting any files on the backup older than a certain date, knowing that they'll be synced if they're still on your current system and just hoping you won't need them anymore
- If you want two-way syncs, you could use a program calle unison. I've never used it, but it supposedly is pretty good, albeit a little slow

Possible solutions

- Duplicate files: `fdupes` [3]
- File buildup: Manual moving
- Two-way: `Unison` [4]
- Scaling:
  - Filesystem
  - Virtual filesystem
  - Manual partitioning

Possible solutions

- Duplicate files: `fdupes` [3]
- File buildup: Manual moving
- Two-way: `Unison` [4]
- Scaling:
  - Filesystem
  - Virtual filesystem
  - Manual partitioning

- Finally, scaling is probably the hardest issue to address, and honestly the one I know the least about
- My backup hasn't outgrown my flashdrive yet, but it almost certainly will at some point, since my harddrive is 1 TB and my flashdrive is 64 gigs.
- One potential solution to this is to use a different file system that would allow you to mount multiple flash drives as a single filesystem.
- Apparently btrfs can do this, though I'm not sure how it would work (what happens if your system spans more than drives than you have ports? Do they all need to be plugged in? etc.)
- Another potential solution is to use a virtual filesystem, then mount drives as one point. Supposedly LVM can do this, but I've never tried it; also suffers same problems as btrfs
- You could also manually break up the syncs across multiple drives if you get to that point and just exclude them from others

References

[1] rsync man page
https://man7.org/linux/man-pages/man1/rsync.1.html

[2] Backup script
https://github.com/davisdude/dotfiles/blob/master/misc-scripts/backup

[3] fdupes
https://github.com/adrianlopezroche/fdupes

[4] Unison
https://www.cis.upenn.edu/~bcpierce/unison/