Rsync How-to Guide

January 13, 2021
David Nichols

What is rsync and why should I care?

Rsync stands for “remote synchronization”. It is used to transfer files with minimum actual transfer of data by copying only the sections of a file that have updated. Some of the useful features include:

- Copying links, devices, owners, groups, and permissions.
- It does not require elevated privileges (though it does require file permissions).
- Files can be copied to a remote machine, remote to local, or local to local.
- Rsync is standard in OSX and MobaXterm.

No, really, why should I care?

Rsync allows recursive[1], differential[2] copies of file trees. With a single command, a folder can be copied without risking overlooking files. If the connection fails during the transfer, reentering the command simply continues the process. This makes it ideal for backing up working files and archiving completed work, which is the use case explored in the rest of this guide.

What it isn’t.

Rsync does not keep differential backups like Time Machine and similar programs such as Subversion. That is, there is no version history, no daily differential backups. Rsync’s ‘differential’ file copy leaves only one copy of a given file, rather than a list of different versions, any of which can be recovered. In that way, data can be lost.

For example, if you have a file, file_1, which is already backed up to your archive, the archive has a copy of file_1 from the last backup. If you then go in and edit file_1 and modify it, another rysnc backup will replace the copy of file_1

---

1 Recursive here refers to rsync’s capability of working its way through sub-folders, applying the same copy method to each folder that it has the first.

2 Differential backups look for changes in a file or file structure and copy only the changes made to the files and file structures.

Any backup is a snapshot of the existing files and folders at the time the backup operation is run. Time Machine takes numerous snapshots. The first Time Machine backup takes lots of time because it is backup up everything. Subsequent snapshots contain the differentials – the changes made to individual files and overall file structure, which takes up considerably less space than subsequent full backups.

Rsync, on the other hand, is a single snapshot. The differentials are written to the existing files and file structure. (It doesn’t remove anything, unless the -**-delete** flag is applied.)

with the updated file. If, while modifying it, you accidentally remove an important paragraph, the copy of file_1 will reflect the changes – that paragraph will still be missing, and there will be no way to recover the lost data.

Rsync works of full path entries. That is, it reads each file according to its full path: /home/user/file1.txt rather than file1.txt. This means that it if you move file1.txt from /home/user/ to /home/user/docs/, it will not remove the old copy of file1.txt from /home/user/file1.txt and add /home/user/docs/file1.txt to the backup. This can lead to unwanted replication of files. There is a flag: **--delete**, which will remove files from such a backup, which will be discussed below. As always, caution is required when deleting files in Linux – once a file is deleted, there is **no** way to recover it.

Finally, rsync cannot copy files and directories for which you lack privileges. If you don't own files in a file tree, using the **-P** flag (detailed below) will show an error indicating your lack of file permissions.

How to use rsync

Here is the basic syntax of rsync:

```
rsync [options] source [destination]
```

The options are one or more of the numerous flags, the source is the file tree you wish to back up, and the destination is the location of the backup. Rsync can be used in many different ways, but in our use case, destination is not optional, and we will need flags. Rsync has two operations; uploading data is called a push 'operation', while a download is called a 'pull' operation. Push is used to back up your data. The basic syntax for a push is:

```
rsync /local/file/path/ username@<remote.host>:/remote/file/path
```

where **<remote.host>** is the remote location to which you are backing up. If you are using an attached drive, omit the '**username@<remote.host>:**' from the command:

```
rsync /local/file/path/ /path/to/backup
```

In our use case, pull operations are used to recover data. A pull reverses the syntax:

```
rsync username@<remote.host>:/remote/file/path /local/file/path
```

In practice, however, you'll need to use several flags ([options] in the basic syntax) in order to copy file trees.

Flags in rsync:

Before going into detail about the flags, it is useful to note that there are many more flags which you are unlikely to find useful. To find out about those flags, enter `man rysnc` to get the manual page for rsync. In the explanations of the various flags below, the man page entry is included in an edited form.

-r, --recursive recurse into directories

Recursive mode copies entire file trees. Without this flag, rsync will only copy the files in the folder specified by `/local/file/path`. This flag is assumed in archive mode (below) and can be omitted.

-a, --archive mode

Archive mode copies links, preserves permissions, modification times, group & owner, and, with superuser access, preserves device files (drivers, etc.). This flag is essential.

--exclude exclude files matching the pattern

Exclude skips files matching the folder entry. It could be used to exclude types of files with a wildcard (such as `*.wav`). It is also useful to exclude certain branches in your file tree.

-P shows progress during transfer including the progress of individual files

The **-P** flag is the verbose output shown as rysnc runs. This is not strictly necessary, but if you have large and/or many files, this can be very reassuring. Without it, there is no output – all that is visible is the command itself, until it has completed, at which time the command prompt returns.

--delete deletes 'extraneous' files

This deletes files from the remote server which no longer exist in the local directory. --delete can be used to clean up backups and archives in order to save space. This comes with a huge caution:
once files are deleted from a backup or archive in this manner, they cannot be recovered. Think carefully before using this option.

Importance of the trailing ‘/’:

The command line interface is unforgiving of errors. Mistakes in either the source or destination paths, the flags used, and syntactic errors can and do cause headaches. Fortunately a lot of mistakes cause errors prior to the rysnc command running. One of the pitfalls is keeping track of the trailing ‘/’ in `/local/source/path/`. Excluding the ‘/’ from the end changes how the file tree is copied to to the backup.

Excluding the ‘/’ from the `/local/source` directory as below appends the copied directory to the destination folder: `/remote/destination/source`.

```
rsync -aP /local/source username@<remote.host>/remote/destination
```

Using the trailing ‘/’ from the `/local/source/` directory appends the copied data directly to `/remote/destination/`.

```
rsync -aP /local/source/ username@<remote.host>/remote/destination
```

Neither of these is wrong, but consistency is key. For our use case, and the examples below, we will use the trailing ‘/’.

Examples:

Even with the best of guides, CLI[3] commands can be difficult to understand. Concrete examples are extremely helpful, but ultimately experience is the best teacher. Many of the following examples build on one another.

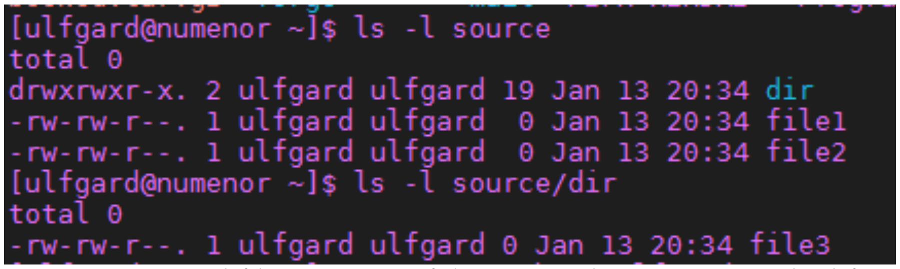
The setup:

Using username ‘ulfgard’ with the source path `/home/ulfgard/source/` on the local system, and the following directory structure:

```
/home/ulfgard/source/
/home/ulfgard/source/file1
/home/ulfgard/source/file2
/home/ulfgard/source/dir/
/home/ulfgard/source/dir/file3
```

3 Command Line Interface.

This is illustrated by using the command l**s -l**, which lists the contents of a folder (Fig. 1). Viewing first the and the empty directory /home/ulfgard/destination on the remote server zeos.ling.washington.edu. (Future **ls -l** commands will omit /home/ulfgard/, as it is the working directory on both the local and remote systems.)



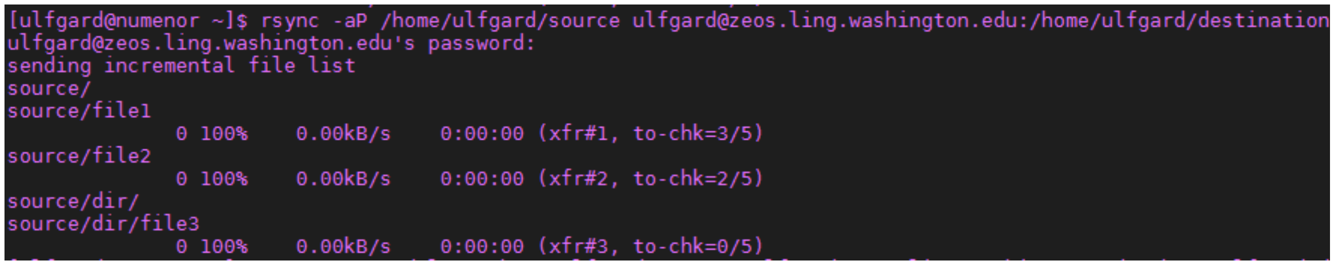
```
[ulfgard@numenor ~]$ ls -l source
total 0
drwxrwxr-x. 2 ulfgard ulfgard 19 Jan 13 20:34 dir
-rw-rw-r--. 1 ulfgard ulfgard  0 Jan 13 20:34 file1
-rw-rw-r--. 1 ulfgard ulfgard  0 Jan 13 20:34 file2
[ulfgard@numenor ~]$ ls -l source/dir
total 0
-rw-rw-r--. 1 ulfgard ulfgard 0 Jan 13 20:34 file3
```

Figure 1: Initial file structure of the source directory, using ls -l for source/ and source/dir/

Standard use case:
Using the rsync command to transfer the contents of the source directory (on a single line):

rsync -aP /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination

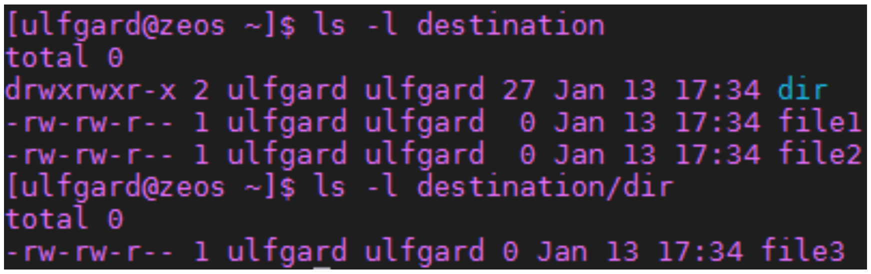
The rsync push command and output from the **-P** flag:



```
[ulfgard@numenor ~]$ rsync -aP /home/ulfgard/source ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
source/
source/file1
              0 100%    0.00kB/s    0:00:00 (xfr#1, to-chk=3/5)
source/file2
              0 100%    0.00kB/s    0:00:00 (xfr#2, to-chk=2/5)
source/dir/
source/dir/file3
              0 100%    0.00kB/s    0:00:00 (xfr#3, to-chk=0/5)
```

Figure 2: The complete rsync command, followed by its output.

All of the data has been appended to /home/ulfgard/destination/:



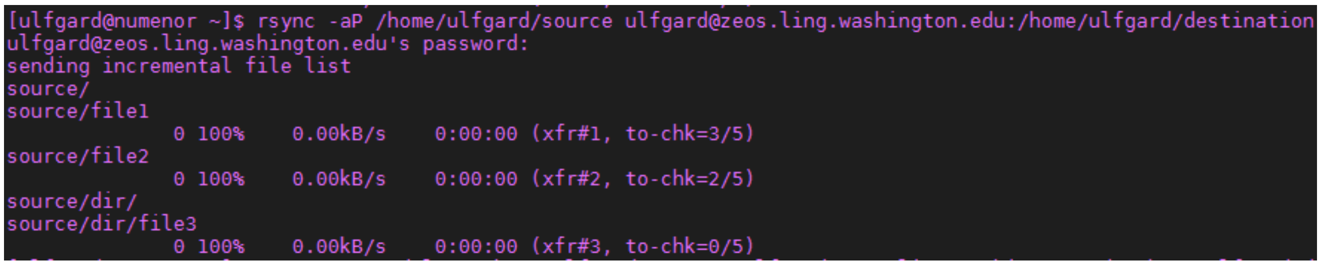
```
[ulfgard@zeos ~]$ ls -l destination
total 0
drwxrwxr-x 2 ulfgard ulfgard 27 Jan 13 17:34 dir
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 file1
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 file2
[ulfgard@zeos ~]$ ls -l destination/dir
total 0
-rw-rw-r-- 1 ulfgard ulfgard 0 Jan 13 17:34 file3
```

Figure 3: destination/ folder after the push operation, using ls -l.

That trailing '/':
Doing nothing else, we now run the same command, excluding the trailing '/' (on a single line):

rsync -aP /home/ulfgard/source ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination

The rsync push command and output from the **-P** flag:



```
[ulfgard@numenor ~]$ rsync -aP /home/ulfgard/source ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
source/
source/file1
              0 100%    0.00kB/s    0:00:00 (xfr#1, to-chk=3/5)
source/file2
              0 100%    0.00kB/s    0:00:00 (xfr#2, to-chk=2/5)
source/dir/
source/dir/file3
              0 100%    0.00kB/s    0:00:00 (xfr#3, to-chk=0/5)
```

Figure 4: Push operation without the trailing '/' and the resulting output.

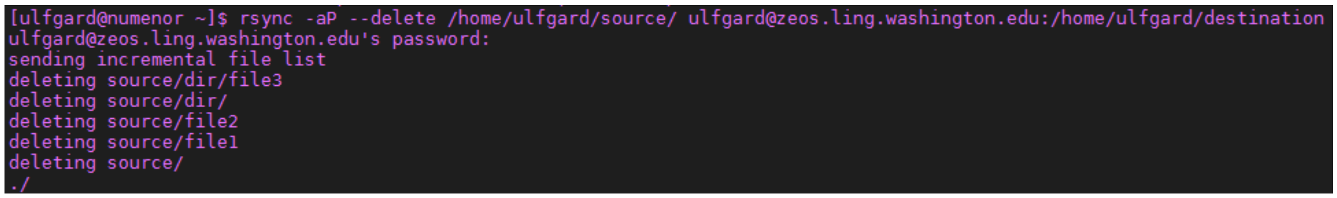
The result is to add a copy of the entire source directory as /home/ulfgard/destination/source:

Figure 5: Destination folder after the second push operation -- without the trailing '/'.

Using --delete:
Realizing that a mistake was made, we can rerun the original command with the additon of the --delete flag. Once again, be cautioned that **files deleted files cannot be recovered.** The command (on a single line):

rsync -aP--delete /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination

This removes the extraneous second source directory. Here is the command and the output:



```
[ulfgard@numenor ~]$ rsync -aP --delete /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
deleting source/dir/file3
deleting source/dir/
deleting source/file2
deleting source/file1
deleting source/
./
```

Figure 6: Push command with the --delete flag and the output.

The extraneous folder has been removed from the remote server.

(Unwanted) file replication:

Suppose that we move file1 to the dir directory, so that the file structure now looks like this:

source/
source/file2
source/dir/
source/dir/file1
source/dir/file3

We run the rsync command again with the trailing '/' and excluding the **--delete** flag. This is exactly like the standard use case, detailed in the standard use case, above.

Figure 7: New file structure, followed by the rsync push command and its output.

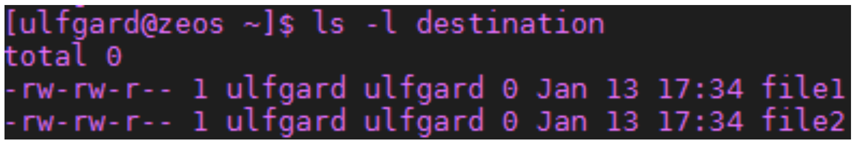
Looking at our destination folder, we find that we now have 2 copies of file1, one in the destination folder, the other in the destination/dir folder (highlighted in blue):

```
destination/
destination/file1
destination/file2
destination/dir
destination/dir/file1
destination/dir/file3
```

Figure 8: File structure on remote server, using ls -l to reveal folder contents.

This may not be an undesirable effect, but it generally bloats the size of a backup or archive.

Excluding a directory:
For this example, we reset our source directory (from the setup, above), and empty destination folder. Suppose we want to suppress the sub-directory dir (i.e., source/dir). We run the following push command, using the --exclude flag



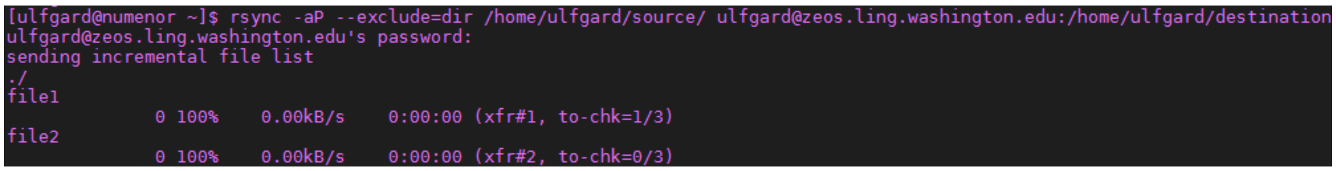
```
[ulfgard@zeos ~]$ ls -l destination
total 0
-rw-rw-r-- 1 ulfgard ulfgard 0 Jan 13 17:34 file1
-rw-rw-r-- 1 ulfgard ulfgard 0 Jan 13 17:34 file2
```

Figure 9: File structure in the destination on the remote server.

command (on a single line):

rsync -aP –exclude=dir /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination

The command and the output from it:



```
[ulfgard@numenor ~]$ rsync -aP --exclude=dir /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
./
file1
              0 100%    0.00kB/s    0:00:00 (xfr#1, to-chk=1/3)
file2
              0 100%    0.00kB/s    0:00:00 (xfr#2, to-chk=0/3)
```

Figure 10: Rsync push command using the --exclude flag, and its output.

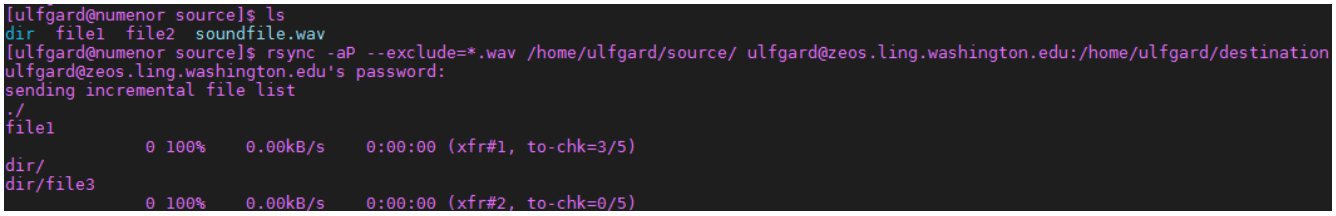
Only file1 and file2 are copied to the remote server:

Excluding .wav files:

Continuing the above example, we add the file soundfile.wav to the source directory. We now exclude all .wav files. Instead of a directory, we use the file extension preceded by a wildcard (*):

rsync -aP –exclude=*.wav /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination

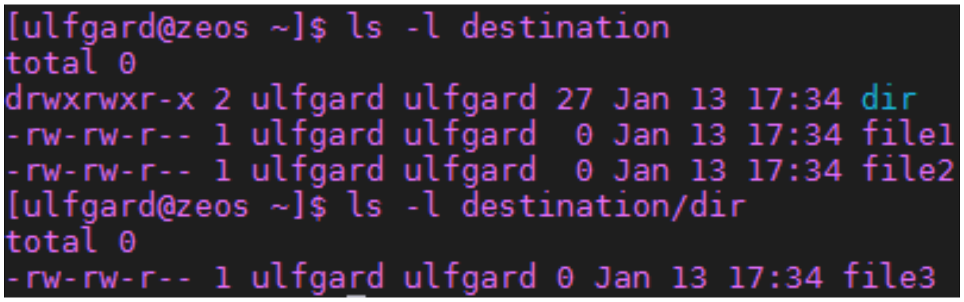
Because we did not suppress source/dir, it is copied to the destination directory.



```
[ulfgard@numenor source]$ ls
dir  file1  file2  soundfile.wav
[ulfgard@numenor source]$ rsync -aP --exclude=*.wav /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
./
file1
              0 100%    0.00kB/s    0:00:00 (xfr#1, to-chk=3/5)
dir/
dir/file3
              0 100%    0.00kB/s    0:00:00 (xfr#2, to-chk=0/5)
```

*Figure 11: Push command excluding *.wav and it's -P output.*

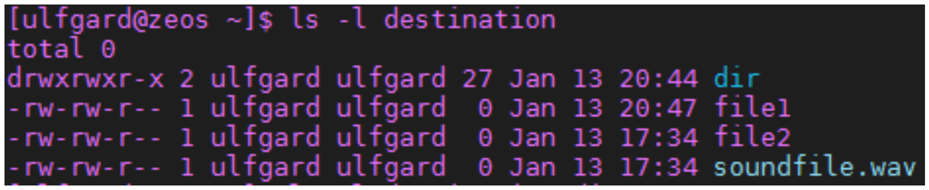
This looks exactly like our initial rsync backup:



```
[ulfgard@zeos ~]$ ls -l destination
total 0
drwxrwxr-x 2 ulfgard ulfgard 27 Jan 13 17:34 dir
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 file1
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 file2
[ulfgard@zeos ~]$ ls -l destination/dir
total 0
-rw-rw-r-- 1 ulfgard ulfgard 0 Jan 13 17:34 file3
```

Figure 12: destination/ file structure after the push command, using ls -l.

Running the basic command without the **--exclude=*.wav** flag uploads soundfile.wav, so that all of the files have now been backed up to the remote destination folder:

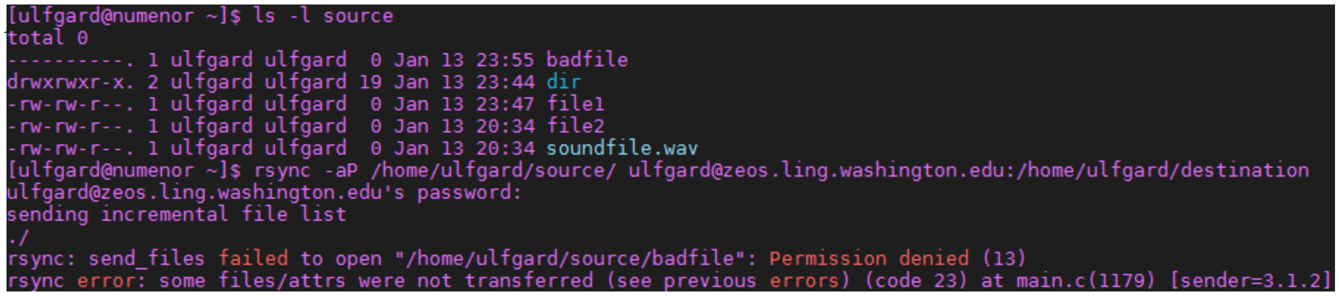


```
[ulfgard@zeos ~]$ ls -l destination
total 0
drwxrwxr-x 2 ulfgard ulfgard 27 Jan 13 20:44 dir
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 20:47 file1
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 file2
-rw-rw-r-- 1 ulfgard ulfgard  0 Jan 13 17:34 soundfile.wav
```

Figure 13: Rsync push with no exclusions, pushing soundfile.wav.

Push / pull of files without permissions:

Adding the file 'badfile' for which we have no permissions, we try rsync using the standard use case push command from source/, which would simply transfer the only new file – badfile – but it lacks permissions. It results in errors, but those errors do not stop rsync from continuing with the push operation.



```
[ulfgard@numenor ~]$ ls -l source
total 0
----------. 1 ulfgard ulfgard  0 Jan 13 23:55 badfile
drwxrwxr-x. 2 ulfgard ulfgard 19 Jan 13 23:44 dir
-rw-rw-r--. 1 ulfgard ulfgard  0 Jan 13 23:47 file1
-rw-rw-r--. 1 ulfgard ulfgard  0 Jan 13 20:34 file2
-rw-rw-r--. 1 ulfgard ulfgard  0 Jan 13 20:34 soundfile.wav
[ulfgard@numenor ~]$ rsync -aP /home/ulfgard/source/ ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination
ulfgard@zeos.ling.washington.edu's password:
sending incremental file list
./
rsync: send_files failed to open "/home/ulfgard/source/badfile": Permission denied (13)
rsync error: some files/attrs were not transferred (see previous errors) (code 23) at main.c(1179) [sender=3.1.2]
```

Figure 15: Output via the -P flag when rsync attempts to operate on a file without permissions.

Pull operations:

Restoring files from backup back is as simple as reversing the order of folders:

rsync -aP ulfgard@zeos.ling.washington.edu:/home/ulfgard/destination/ /home/ulfgard/source/

Doing so with the file tree we just transferred has little effect. However, reversing the order and using a pull operation is how you restore your data if it has been lost. To restore individual files, it is best to use SFTP; SFTP put (equivalent to the rsync push) overwrites destination files. Any rsync operation in which no files are transferred will have an empty (or mostly empty, depending on the version) output from the **-P** flag.

That’s great, but I’m not using a remote server.

To copy files from one folder to another on a local machine only changes the source/destination by removing the necessity for the “**username@<remote.host>:**” portion of the command, so that it becomes:

rsync -aP /path/to/source/ /path/to/destination

Okay, but how do I find my USB device?

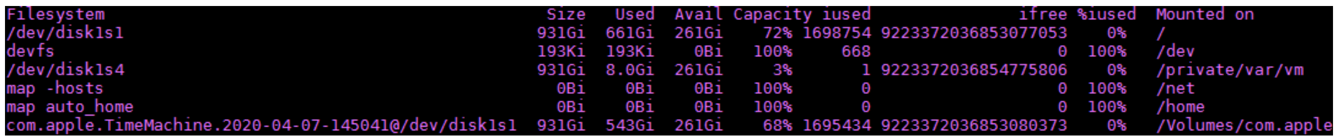
The most useful local-only backup is from your computer to an external drive. This is fairly easy on a Windows system. Using the GUI[4], go into file explorer and select “This PC” to show the Devices and Drives. External drives are assigned a letter once they have been recognized and mounted by the system. Suppose the drive letter for your USB device is D. Using MobaXterm, the destination folder should be preceded by /drives/d/, thus the full path is /drives/d/destination. Assuming a source directory in your account, the standard rsync push command is:

rsync -aP /path/to/source/ /drives/d/destination

In OSX, all of this is done via the CLI. Enter the command:

df -h

This will produce a list of drives showing: file systems, size, space used, space available, remaining capacity as a percentage, information about inodes[5] (iused, ifree, %iused), and the mount point – the path used to access it. We’re interested in the “Mounted on” column. Find your USB drive in that column. It will generally be preceded by /Volumes/. This is the path which precedes every file or folder on your USB device.



```
Filesystem                                    Size   Used  Avail Capacity iused               ifree %iused  Mounted on
/dev/disk1s1                                 931Gi  661Gi  261Gi    72% 1698754 9223372036853077053    0%   /
devfs                                        193Ki  193Ki    0Bi   100%     668                   0  100%   /dev
/dev/disk1s4                                 931Gi  8.0Gi  261Gi     3%       1 9223372036854775806    0%   /private/var/vm
map -hosts                                     0Bi    0Bi    0Bi   100%       0                   0  100%   /net
map auto_home                                  0Bi    0Bi    0Bi   100%       0                   0  100%   /home
com.apple.TimeMachine.2020-04-07-145041@/dev/disk1s1  931Gi  543Gi  261Gi    68% 1695434 9223372036853080373    0%   /Volumes/com.apple
```

Figure 16: Results of df -h on chesterton.

For example, if the device is named MY USB STICK, you’ll likely find it mounted under /Volumes/MY USB STICK. The destination file would then be accessed at

4 Graphical User Interface.
5 Inodes are lower level metadata about files and directories in unix-like file systems. That data is generally of little use for most users.

/Volumes/MY USB STICK/destination.

A word of caution about whitespace in a unix-like CLI: each space delimits a different “word” or portion of a command. Using /Volumes/MY USB STICK/destination will result in rsync seeing /Volumes/MY as the destination, and USB, STICK/destination as extraneous information. If /Volumes/MY exists, the data will go directly to it. Otherwise, it will fail to find the specified destination directory and will fail to run. To solve this problem, enclose the path in single quotes: ‘/Volumes/MY USB STICK/destination’. OSX then sees the entire string as a single entity. Assuming a source directory as above, the command is:

```
rsync -aP /path/to/source/ ‘/Volumes/MY USB STICK/destination’
```