Duplicating your System

Using duplicity to back up your FreeBSD desktop

BY JASON TUBNOR

You've just installed your new FreeBSD desktop (or laptop), got it just the way you like it, and are about to start work on that new novel or porting a new piece of software to FreeBSD that has just been released by your favourite software vendor. Then you realise that you're only one defective NVMe controller away from total data loss, taking all your work and time with it.

Backing up in modern times, we've had ZFS snapshots and replication to make this task extremely easy. However, you may not have access to another ZFS endpoint for replication, need to diversify risk by using a non-ZFS tool for backup, or are simply using UFS2, living the old skool life.

Then you realise that you're only one defective NVMe controller away from total data loss.

For these situations, my first recommendation is to lean on Tarsnap for its ease of use and simplicity, making restoration just as easy as backing up. But some situations call for a different approach. Maybe you have a strict firewall at your company that doesn't allow Tarsnap data streams to egress from your corporate network, or you have internal/easy access to storage endpoints, such as S3-compatible object storage or a large-file storage location with SFTP access.

When you are faced with the latter, the duplicity (sysutils/duplicity in ports) utility is available as an easily installable package onto your FreeBSD system:

```
# pkg install duplicity par2cmdline
```

In the above case, the par2cmdline package is also installed, as it isn't a dependency of duplicity but is used in this article (and by duplicity) to avoid bit rot in backup archives.

Duplicity (https://duplicity.gitlab.io/) will back up your various files and folders to encrypted tar-format volumes, while using librsync to improve compression and perform differential actions on data. While it doesn't have some of the really good features that make Tarsnap appealing, it does provide an alternative that works well for a lot of use cases.

While duplicity has many options for target storage, we are going to focus on using S3-compatible object storage here, as it is readily available from many cloud providers with varying cost vs access speed vs durability. The way duplicity works is more of an interactive method (especially when preening old data sets). While you might have success committing backups to AWS Glacier or similar object storage, restoration and management of data will be extremely slow, complex, and expensive; you have been warned. Don't aim for cold, cheap storage; go a little higher/warmer for a better experience. In saying it is interactive,

it will work perfectly fine when called from cron, executed within a script, or simply used on the interactive command line.

Before starting, we need to define the parameters and best practices for backing up data. Key considerations are:

- New full backup monthly — A long chain of incremental backups, while fast to back up, can blow out your Recovery Time Objective (RTO). Introducing frequent full backups will ensure the RTO window remains low.
- Encrypt your backup — You are backing up to someone else's computer. Unless you control the remote computer, data should be encrypted on your computer before it hits the network.
- Don't trust endpoint durability — Use the par2 function to incorporate parity blocks into your backup in the event of bit rot. The default is 10%, which can increase the cost for large archives. Assess your risk and adjust accordingly. We will use 5% in this example.

Before starting, we need to define the parameters and best practices for backing up data.

Prepare the AWS environment variables. These can go into a .duplicity.env file in your home directory. If you are on a multi-user system, ensure the .duplicity.env file is chmod 0600 to prevent other system users from gaining unauthorised access to private API keys:

```
~/.duplicity.env

export AWS_ACCESS_KEY_ID="<key_id>"
export AWS_SECRET_ACCESS_KEY="<access_key>"
export AWS_ENDPOINT_URL="https://sg-s3.storage.bunnycdn.com"

Then load our environment file for use:

$ . .duplicity.env
```

The final requirement before we begin using duplicity is to define a PASSPHRASE key for symmetrical encryption of your data when GPG is used (encryption is used by default). GPG Public key encryption can be used but is beyond the scope of this article (see the duplicity man page for further information). The PASSPHRASE variable can also be defined in the environment file we used above and then called from your backup script. For ease of documentation, it will be just exported here:

```
$ export PASSPHRASE="4640f58235c4ff7ad359fdcb5f2d8ac48f71c26bcff4a40f6d85503d0288a45e"
```

We are now ready to perform the first backup of our home directory from our freshly configured system:

```
$ duplicity backup --full-if-older-than 1M --exclude ./.cache \
        --par2-redundancy 5 ./ par2+s3:///mybucket-s3backup/computer
```

The above command engages the backup component of duplicity, creates a full backup if the previous full backup is older than one month, excludes the .cache directory, sets the par2 redundancy to 5%, taking all the files and folders from the current directory (the root

of the users home directory) and writes the backup to the mybucket-s3backup bucket in the /computer folder, using the par2 wrapper for block redundancy.

Once duplicity has completed its run, you'll be presented with the statistics for the backup:

```
Last full backup date: none
Last full backup is too old, forcing full backup
--------------[ Backup Statistics ]--------------
StartTime 1773460673.41 (Sat Mar 14 14:57:53 2026)
EndTime 1773460673.42 (Sat Mar 14 14:57:53 2026)
ElapsedTime 0.01 (0.01 seconds)
SourceFiles 15
SourceFileSize 6302 (6.15 KB)
NewFiles 15
NewFileSize 6302 (6.15 KB)
DeletedFiles 0
ChangedFiles 0
ChangedFileSize 0 (0 bytes)
ChangedDeltaSize 0 (0 bytes)
DeltaEntries 15
RawDeltaSize 6277 (6.13 KB)
TotalDestinationSizeChange 3884 (3.79 KB)
Errors 0
-------------------------------------------------
```

As this is a clean system, our home directory hasn't seen much activity yet and is quite small, so the above backup didn't take long to execute. The initial total that we have backed up:

```
$ du -sh .
62K     .
```

As you can see, even on a small dataset, 62KB of files (allocated disk space), 6.15KB is the actual file total with a destination size of 3.79KB after compression and encryption is taken into account.

Testing has shown that duplicity works as quickly as your hardware allows; the throttle will be set by how fast your object storage is and by the size of your network egress throughput. To observe how the backup is performing, use the **`--progress`** switch when executing duplicity on the command line.

At this point, your data is encrypted on remote storage and is unavailable, even to you, if you lose your symmetrical encryption key (PASSPHRASE). Take the time to put a copy of the key in a safe place in case you need to restore data to a fresh system.

Let's view what is in our first backup:

```
$ duplicity ls par2+s3:///mybucket-s3backup/computer/

Last full backup date: Sat Mar 14 14:57:52 2026
Sat Mar 14 14:00:35 2026 .
Sat Mar 14 13:33:22 2026 .boto
Sat Mar  7 15:55:14 2026 .cshrc
Thu Mar 12 22:15:04 2026 .gnupg
```

```
Thu Mar 12 22:15:04 2026 .gnupg/common.conf
Thu Mar 12 22:15:04 2026 .gnupg/private-keys-v1.d
Sat Mar 14 14:56:10 2026 .gnupg/random_seed
Sat Mar 14 14:00:35 2026 .lesshst
Sat Mar  7 15:55:14 2026 .login
Sat Mar  7 15:55:14 2026 .login_conf
Sat Mar  7 15:55:14 2026 .mail_aliases
Sat Mar  7 15:55:14 2026 .mailrc
Thu Mar 12 22:13:12 2026 .profile
Sat Mar 14 13:47:32 2026 .sh_history
Sat Mar  7 15:55:14 2026 .shrc
```

The list (ls) command in duplicity displays archives similar to **`tar -t`**. Again, here, the S3 connection has been made to mybucket-s3backup with the par2 wrapper to validate the consistency of each block and repair on the fly.

Time to start being productive in our home directory. Let's clone the ports branch to start with:

```
$ git clone --depth 1 https://git.freebsd.org/ports.git
Cloning into 'ports'…

$ du -sh .
1.0G    .
```

That has grown our home directory considerably. Now it is time to back up our work across multiple ports:

```
$ duplicity backup --full-if-older-than 1M --exclude ./.cache \
      --par2-redundancy 5 ./ par2+s3:///mybucket-s3backup/computer

Local and Remote metadata are synchronized, no sync needed.
Last full backup date: Sat Mar 14 14:57:52 2026
--------------[ Backup Statistics ]--------------
ElapsedTime 102.56 (1 minute 42.56 seconds)
SourceFiles 214568
SourceFileSize 682126695 (651 MB)
NewFiles 214554
NewFileSize 682120407 (651 MB)
RawDeltaSize 681815400 (650 MB)
TotalDestinationSizeChange 224163465 (214 MB)
-----------------------------------------------
```

As you can see, duplicity is extremely efficient. Our backup has only grown by 214MB, even though when the ports tree was checked out, our system reported 1GB.

Restoring a backup is just as simple as backing up:

```
$ duplicity restore --path-to-restore ports \
         par2+s3:///mybucket-s3backup/computer /tmp/temp/ports
Local and Remote metadata are synchronized, no sync needed.
Last full backup date: Sat Mar 14 14:57:52 2026
```

This command restores only the ports directory from the home directory that was previously backed up to the S3 bucket using the par2 wrapper. The destination of the ports directory will be /tmp/temp/

```
jtubnor@disk:~ $ du -sh /tmp/temp && ls -lsa /tmp/temp

1.0G    /tmp/temp
total 18
1 drwxr-xr-x   3 jtubnor wheel    3 Mar 14 16:56 .
9 drwxrwxrwt  16 root    wheel   16 Mar 14 16:57 ..
9 drwxr-xr-x  70 jtubnor jtubnor 82 Mar 14 15:18 ports
```

The output above shows that our 1GB ports directory has now been restored to the /tmp/temp directory. If you have had a complete machine failure, restoring your home directory is as simple as:

```
$ cd / && duplicity restore par2+s3:///mybucket-s3backup/computer /home/jtubnor/
```

You need to ensure that you are currently not in the directory being restored to; this is why we have traversed to the root directory before issuing the restore.

This article barely scratches the surface of the many features contained within duplicity. It is even suitable for whole-system configuration backups, if needed, by adjusting as necessary. For more information on features and use case examples, check out the duplicity man page (https://duplicity.gitlab.io/stable/duplicity.1.html)

Duplicity is a small, mature program with only a few dependencies, but it works well in most situations and is easily installable on any FreeBSD system.

JASON TUBNOR has over 30 years of IT industry experience in a vast range of disciplines and is currently the Manager of Technology Operations at Latrobe Community Health Service (Victoria, Australia). Discovering Linux and Open Source in the mid 1990s, then being introduced to OpenBSD in 2000, Jason has used various BSDs to solve problems in organisations that cover different industries. Jason is also a co-host on the weekly BSDNow Podcast (https://bsdnow.tv).