How to recover data from a corrupted .tar.bz2 file?

How to recover data from a corrupted tar.bz2 file ?

This page will show you how to recover data from a corrupted .tar.bz2 file,
and NOT from a .tar.gz file, since gzip cant do anything with corrupted archives,
it will just leave you despaired, sad, and lonely.
we'll assume you have an archive called "archive.tar.bz2", for which you want:

- To make sure it's not corrupted
- If it's corrupted, get most everything that is still valid out of it.

Is my archive corrupted ?

That is definitely a good question. To get the answer, bzip2 has a neat option: *-t*.

```
bzip2 -t archive.tar.bz2
```

This will tell you if your bzipped archive is fine, or not.
If it's fine, well, enjoy your day 😃 Otherwise, read on, we'll recover it.

Hey, my archive is corrupted. Should I go for the rope or the gun ?

General Information.

None. You can't fix archives with ropes, nor with guns.
Instead, we'll unleash the power of Bzip2 and its builtin Blockwise CRC checks.
Note: The next step will generate **A LOT** of files.
You might want to create a folder dedicated to that purpose, and copy your archive.tar.bz2 in it.
Let's assume you **did** that, and that you called that folder *recovery/*.

Getting the data blocks out of the bzip2 archive.

cd into recovery.
Here, we'll use the magic bzip2recover command. *Hey, but what's that bzip2recover command*. Hmmm.
Bzip2 compressed file are divided into blocks (each block being 100k, 200k, ..., 900k bytes big,
depending on what compression options you used - default is 900k).
What bzip2recover does, is splitting a bzip2 archive into many smaller
bzip2 archives (one per block, actually). That's why it's generating soooo many small files.
So, here we go:

```
bzip2recover archive.tar.bz2
```

You'll end up with all these rec00XXXarchive.tar.bz2 files. Lovely, huh ?

Hey, now I have 9k smaller archives, but it didnt fix my problem, you guy are so useless!

No, i'm *not*. I'm not the one with a corrupted archive <evil laugh>.
Seriously, now that we have divided the archive into smaller parts, we'll be able to "isolate" the corrupted parts.
To do so, we'll use *bzip2 -t*, as we did before, but this time on **every small archive** file.
Here we go:

```
bzip2 -tv rec*.bz2 > testoutput.log 2>&1
```

the 2>&1 stuff is to redirect the stderr output to stdout (otherwise we wouldn't see which file is corrupted. Mean, huh ?)

Ok, now, we will search for any corrupted small archive through the log file.

```
grep [^ok]$ testoutput.log
```

(this actually parses the output of bzip2 -t to extract only files which don't end up with a candid "ok"

- guess what, corrupted files don't generate this kind of candid output 😃)

Ouch, i've got corrupted blocks. What should I do with that ?

Note: We are here restoring from a file with only 1 corrupted block. The process would be the same
if you had 15 corrupted blocks, but you had to restore from begining of file to first corrupted block,
then from 1st corrupted block to 2nd, then from 2nd to 3d, ..., and finally
from 15th to the end of file. Did you get it ? I'm sure you did.
Let's focus on the recovery process, now.

Ok, let's say you have a corrupted block, which is block no. 10. ("rec00010archive.tar.bz2")
So, you have blocks 1-9 fine, then a dumb corrupted block, then everything is fine again.
We'll have to work on these 2 parts separately.
Lets create 2 directories, well call it "recovery1" and "recovery2".
Copy all blocks from block 1 to block 9 (our "clean" blocks until our corrupted block) into recovery1.
Copy all blocks from block 11 to the last block into recovery2.

Let's tackle everything up to the first corrupted block !

Ok, cd into recovery1.
Here, we have the beginning of a tar file, nothing's corrupted, but the tar file is not complete.
Right. That makes things easy.
We will just bunzip all the small archives into one recovery1.tar file:

```
bzip2 -dc rec*.bz2 > recovery1.tar
```

Let's have a look at the result .tar file :

```
tar tf recovery1.tar
```

Wow ! We're getting a list of file, and an error. Not perfect, but better than nothing!
We have here all the files which were into the original archive.tar.bz2 until the first corrupted block.
We're done for recovery1 !

And now, last but not least.....

recovery2 !! cd ../recovery2
Hmmmm trying the same method as above fails. Why that ? Because tar sux. Yes, it *does*.
It does not manage to find a correct header right at the start of the file, and so, fails.
Creepy, huh ? But we are smarter than Tar, and there's not much that a little of Perl Magic can't solve.
First, let's have our bzip2 small archives bunziped into a "failing" tar.

```
bzip2 -dc rec*.bz2 > recovery2_failing.tar
```

As I told you right before, a *tar tf recovery2_failing.tar* would.... fail 😃
What we would need to fix it, is having our recovery2_failing.tar
starting from the begining of a **clean** header block.
A simple but efficient perl script will help us to make our way out: find_tar_headers.pl.bz2

Yeah, bunzip2 . chmod +x on it.
Now, to find the first clean tar header on recovery2_failing.tar, do the following:

```
./findtarheader.pl recovery2_failing.tar
```

This will generate quite a bunch of output. The only one interesting here is the first result. You can then do :

```
./findtarheader.pl recovery2_failing.tar | head -n 1
```

to get only the first occurence of a tar header.
You get something like :
recovery2_failing.tar:17185:naked_girls/pamela.jpg:157106

Beside the file name "naked_girls/pamela.jpg", which obviously shows
that the tar file is the backup of Bharathy's Home directory, have a look
at the second field of the output:
17185 : this is the offset of the clean tar header, from the begining of the file.

Good ! Now we have the offset of a clean Tar Header !!
We will be able to recover everything in the tar file starting from that file !
Wooohhhooooooo .

To do so, do the following :

```
tail -c +17185 recovery2_failing.tar > recovery2_working.tar
```

This command copies everything from recovery2_failing.tar, **starting at offset +17185** into recovery2_working.tar.
Great, now we have a "recovery2_working.tar" tar file, which WORKS !

```
tar tf recovery2_working.tar
```

Yoodelihoo ! You did it !!

Food for thoughts.

Well, right, **you** did it.
We can get something out of it.
For instance, **thou shall not** use gzip compressed archives for relatively **critical** stuffs,
because if it **ever** gets corrupted, well, it's just lost. Sad story, huh ?
Second thing, tar archives are quite fine with data corrupting, at least,
they are better than gzipped files.
Here, we could restore **everything** from a .tar.bz2 file, EXCEPT what was
within the corrupted bzip block, and everything until the first clean header
after the corrupted block. To sum it up: we lost one block, and any file with
either its header or a part of the body in that block.
If you are saving critical stuff, you could tell BZip2 to use 100kb block-size.
If your archive gets corrupted, you loose a **multiple of 100kb**,
against a **multiple of 900kb** if you use 900kb block-size,
which could actually make a **BIG** difference!

Addendum : Expected Minimal Data Loss
Best case (minimal loss): No file has its header within a corrupted block and its data block in others.
Wors case (maximal loss): Each corrupted block contains the header of a **big** file. The whole block is lost, plus that file. (hypothetically, unlimited amount of data can be lost, it could be a 100GB file....)

Please note that statistically,with a size of block of 'B' kB on a high amount of corrupted blocks ('N'), if the average
filesize is 'M' kB, the expected data lost is around

```
Estimated average data lost over coruption: (B + (M+1)/2 ) x ( N ) kB
```

On a tar file within which the average file size is 200kB, bziped with 900 kB per block, 10 faulty blocks, data
loss is around (900 + 101) x 10 = 10.1 MB.
Same thing with 100kB per block, (100 + 101) x 10 = 2.02 MB.

This should be considered when deciding to build a bzip2 zipped archive, the smaller the block size is, the faster it will compress, the worse the compression will be, and the smaller data will be lost in case of corruption.