Andrew Meyer
Report for 2019-6-19

What I have done:

- Created a more detailed plan for test images (next page). For each test I have created:
  - Description of test condition
  - Planned steps to get the file system to the desired state
  - Expected result of file recovery (if tool meets NIST standards)
  - Diagram of test condition
- Created a more detailed description of how I plan to run tests, with example commands for most steps (final two pages)
- Successfully created the first 4 test images

Goals for the next week:

- Create the remaining test images and upload them to the github repo, along with a file showing the sequence of commands that created each image
- Create and upload the expected recovered file for each test image (for most cases this will just be the original file or a portion of it)
- Run Autopsy on the test images and record results
- Study NTFS in preparation for making the second round of test images

Thoughts/Concerns:

- I couldn't think of any way to create 4ii and 5ii through typical file operations. Because the NIST standards say about the deleted file state "It is assumed that the files used to test the deleted file recovery process were created and deleted in a process similar to how an end-user would create and delete files," we might just want to remove these cases for being unrealistic.

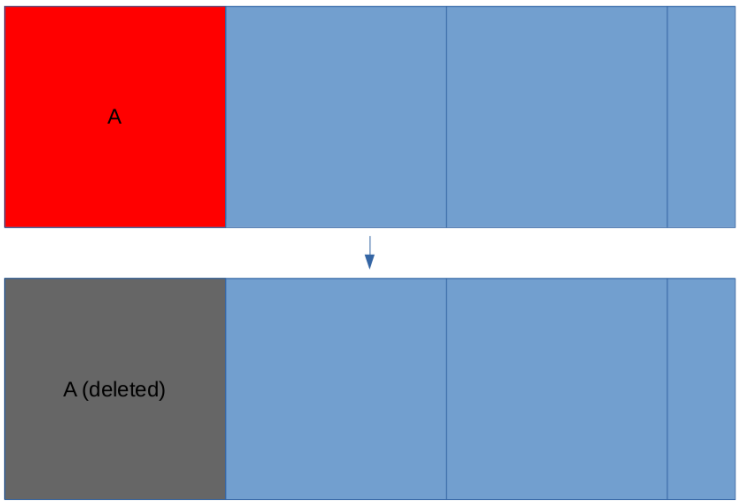
- I'd like feedback on the expected recovery results for each test case. Given that in the case of fragmentation there is more than one approach for recovering the fragments, I suggested multiple valid results for some (either the full file or just the first fragment would be valid). What are your thoughts on this?

- When creating test images, we need to remember the OS will buffer disk writes. I found in some cases when I would save a file and then immediately delete it, it was never on the disk to begin with. To keep behavior as predictable as possible, we should unmount and remount the file system before and after appending or deleting any files.

Possible FAT test cases:

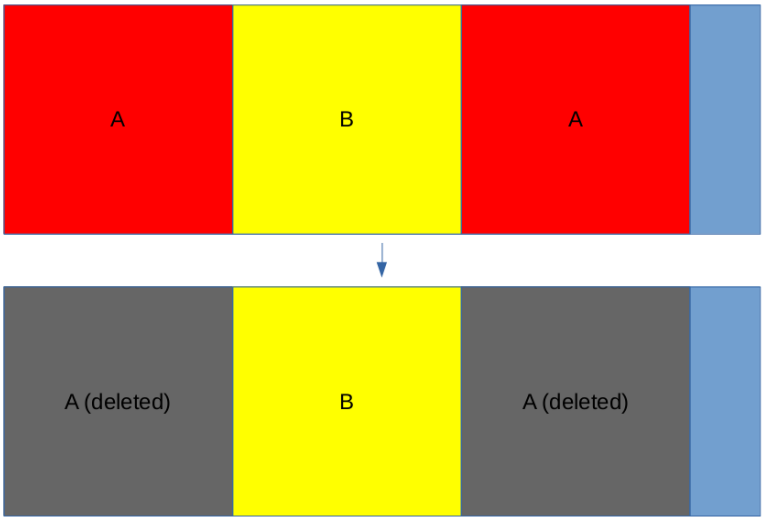
1. File A is written to contiguous clusters and deleted.
    1. Write file A
    2. Delete file A

    Should recover all of A.



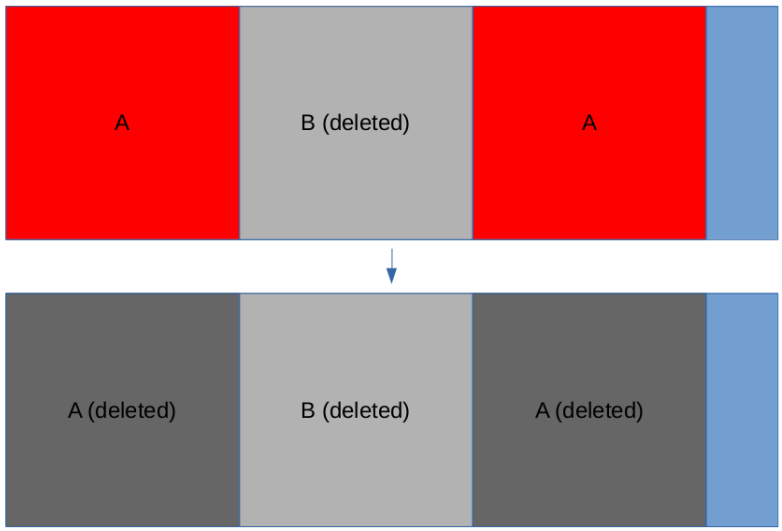
2. File A is written to non-contiguous clusters (A is fragmented) and deleted.
    1. Write file A
    2. Write file B
    3. Append to file A
    4. Delete file A

    Should recover either all of A or only the first fragment of A.



3. File A is written to non-contiguous clusters and deleted, and clusters between the fragments are de-allocated.
    1. Create drive 2
    2. Delete file B

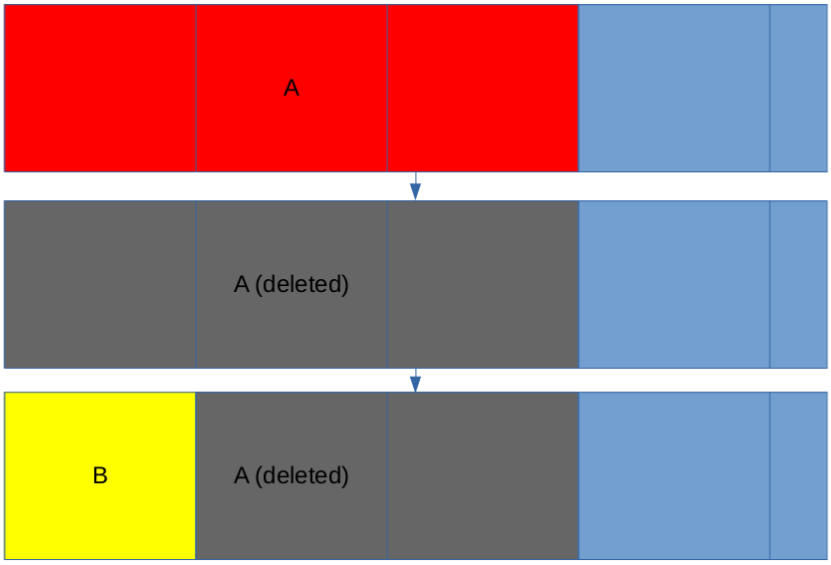
    Should recover either all of A or only the first fragment of A.



4. File A is written to contiguous clusters and deleted, and file B is written over one of those clusters.

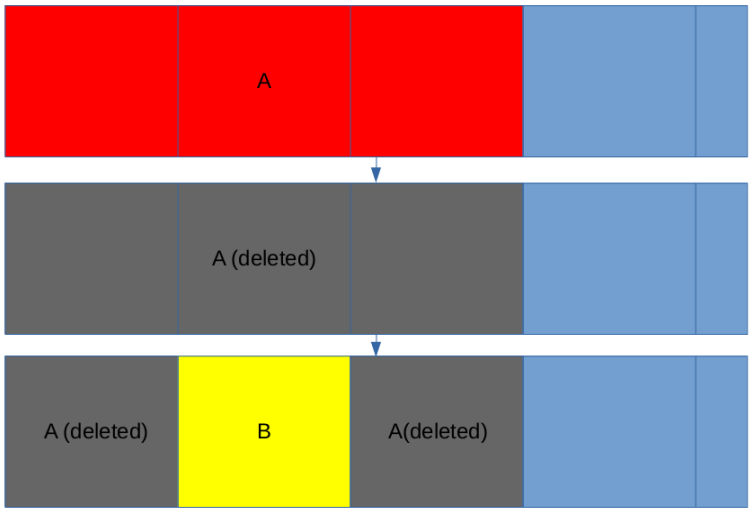
    i. B is written over A's first cluster.
        1. Create 4M filesystem
        2. Write file A (3M)
        3. Delete file A
        4. Write file B (1M)

        Should recover all of A that was not overwritten.



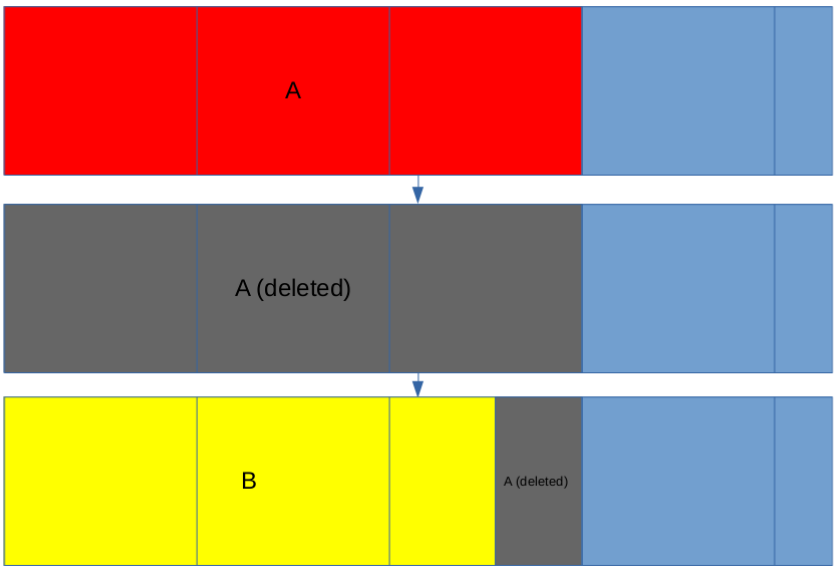
ii. B is written over the middle of A such that deleted data from A is present before and after B.
    1. ???

    Should recover all of A that was not overwritten.



iii. Data from B does not precisely fit the cluster size, so one cluster will contain the end of B followed by deleted data from A.
    1. Create filesystem
    2. Write file A (xM)
    3. Delete file A
    4. Write file B (xM – {y < cluster size})

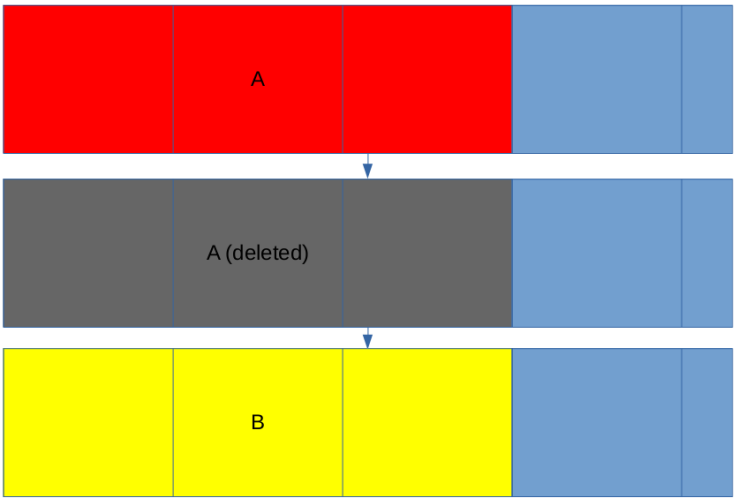
    Should recover all of A that was not overwritten.



iv. B completely overwrites A.

1. Create filesystem
2. Write file A (xM)
3. Delete file A
4. Write file A (xM)

Should recover nothing.

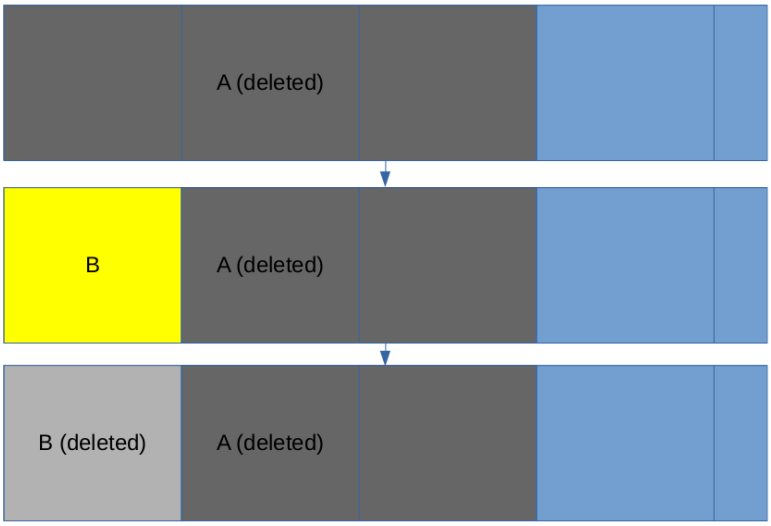


5. File A is written to contiguous clusters and deleted, and file B is written over one of those clusters and deleted.

i. B is written over A's first cluster.

1. Create drive 4i
2. Delete file B

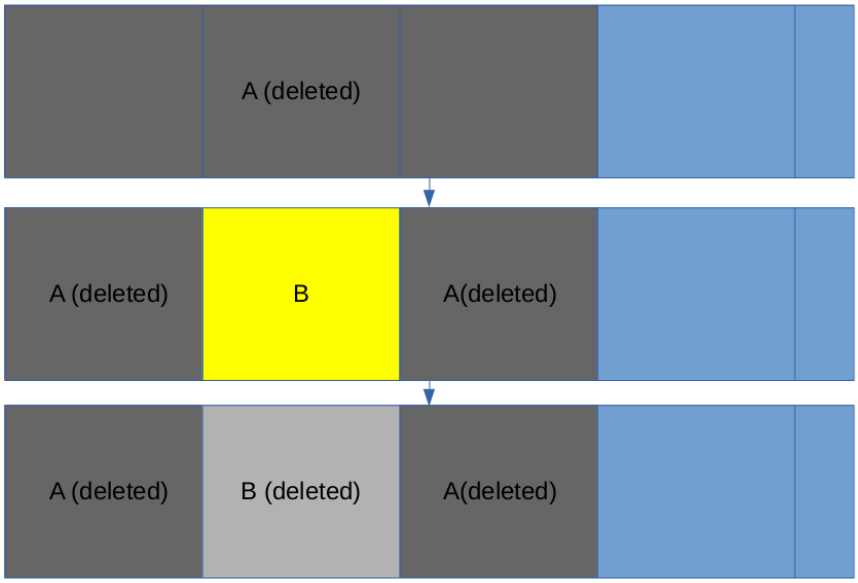
Should recover all of B, and all of A that was not overwritten.



ii. B is written over the middle of A such that deleted data from A is present before and after B.

1. Create drive 4ii
2. Delete file B

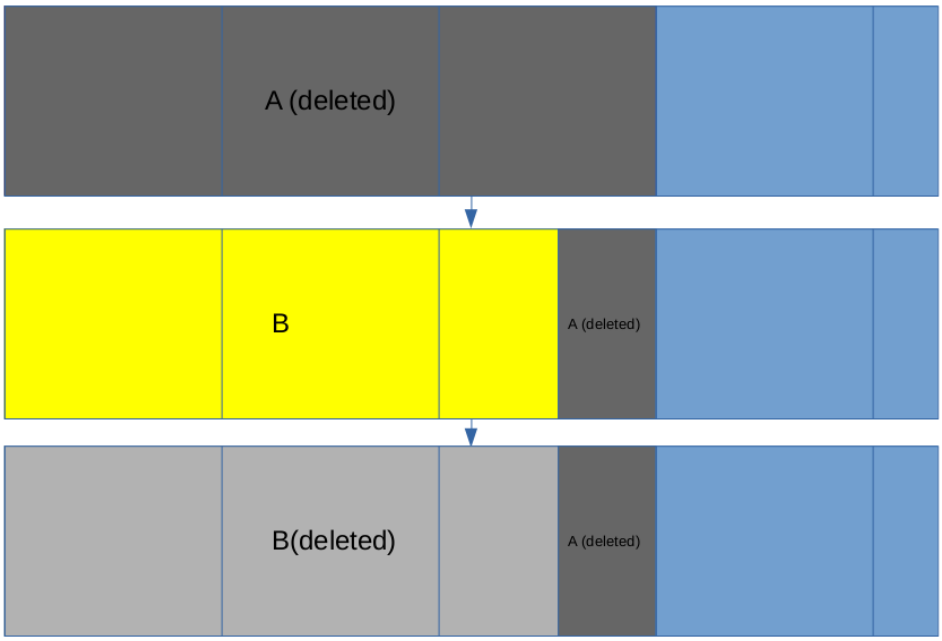
Should recover all of B, and all of A that was not overwritten.



iii. Data from B does not precisely fit the cluster size, so one cluster will contain the end of B followed by deleted data from A.

1. Create drive 4iii
2. Delete file B

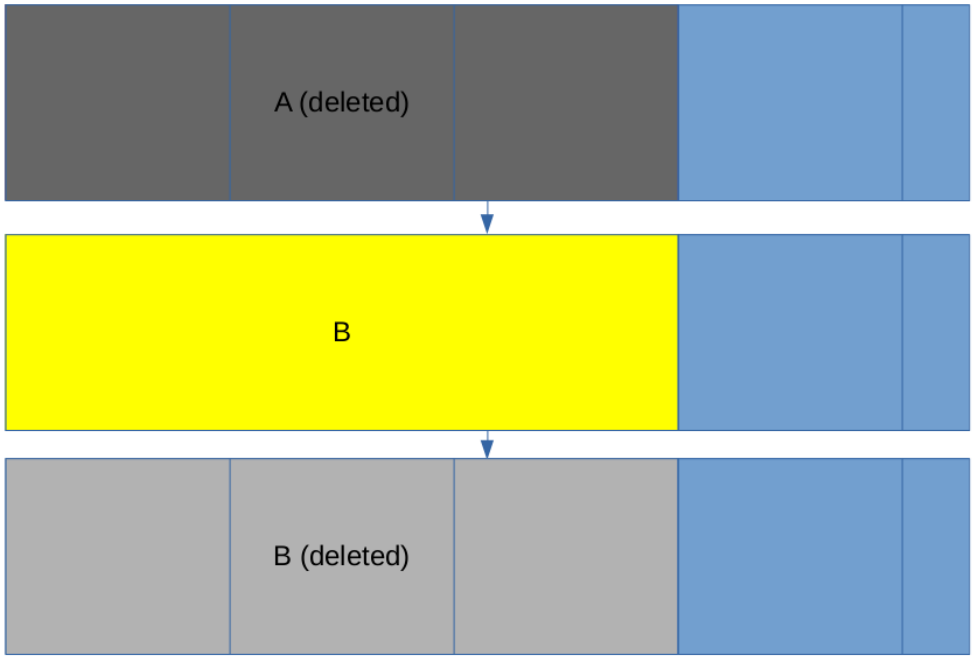
Should recover all of B, and all of A that was not overwritten.



iv. B completely overwrites A.

1. Create drive 4iv
2. Delete file B

Should recover B.

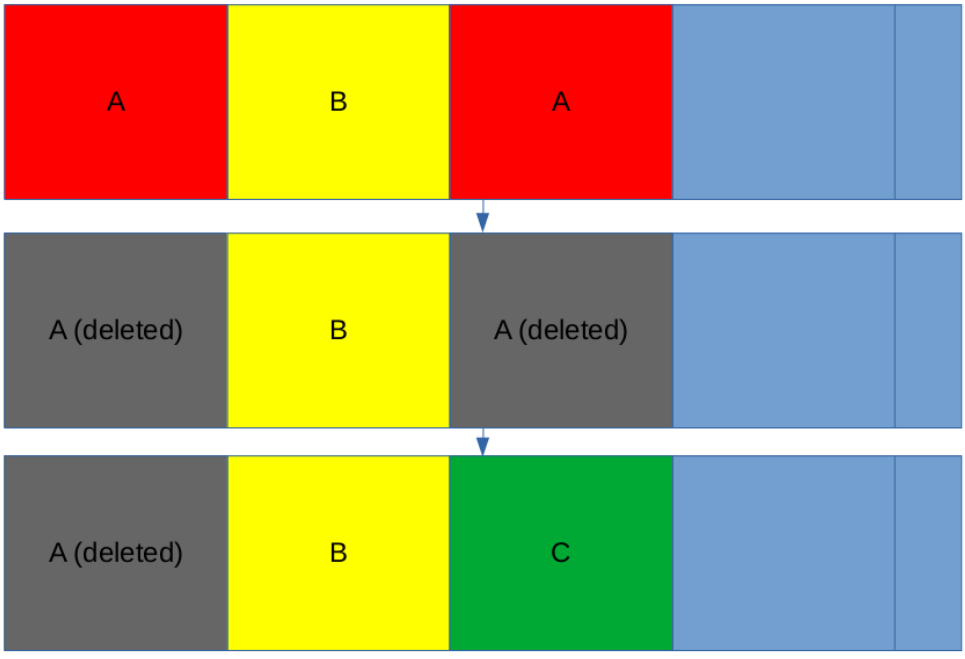


6. File A is written to discontiguous clusters and deleted, and file C is written over the second fragment.

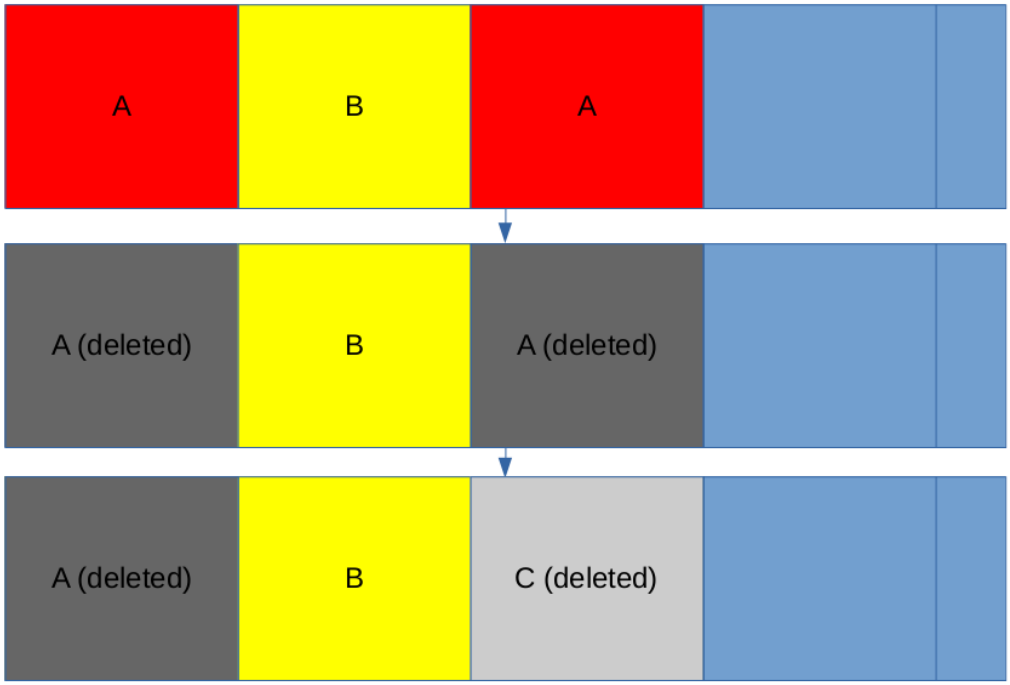
(If OS uses “best fit” allocation algorithm)

1. Create drive 2
2. Write file C (size of remaining space after file B)

Should recover the first fragment of A.



7. File A is written to discontiguous clusters and deleted, and file C is written over the second fragment and deleted.
    1. Create drive 6
    2. Delete file C



I haven't done anything with Quinton's test cases yet, however case 8 ends up being the way I was able to create case 2:

8. ~~File A is created, File B is created, File A is enlarged then deleted.~~ (covered by case 2)
9. File A is created, File B is created, File A is enlarged then deleted, File B is enlarged.
10. File A is created, File B is created, File A is enlarged then deleted, File B is enlarged then permanently deleted. File C is created.

Test procedure:

In these steps I use only command line tools for any step that modifies the file system, so the methods used to create the image can be easily documented and reproduced. All those steps are performed in the VM for the same reason.

Test files are each a sequence of the same ASCII character, of a set length. For example: aa1M is 1 MiB of ‘a’.

To prevent Ubuntu from automatically mounting filesystems:
https://help.ubuntu.com/community/Mount/USB#Configuring_Automounting

1. Partition USB drive **(host or guest OS)**
   I used gdisk for this, and made several 4MiB partitions

For each test image: (example commands used for creating test case 1)

2. Zero over partition (**host or guest OS)**
   dd if=/dev/zero of=/dev/sdx1

3. Write file system **(guest OS)**
   mkfs.fat -n CASE_1 /dev/sdx1

4. Mount file system **(guest OS)**
   mnt /dev/sdx1 /mnt

5. Perform and document file writes and deletions for given test case **(guest OS)**
   Unmount and remount file system before deletions to ensure buffered writes actually make it to the disk
   cp aa1M /mnt
   umount /mnt
   mnt /dev/sdx1 /mnt
   rm /mnt/aa1M

6. Unmount file system **(guest OS)**
   umount /mnt

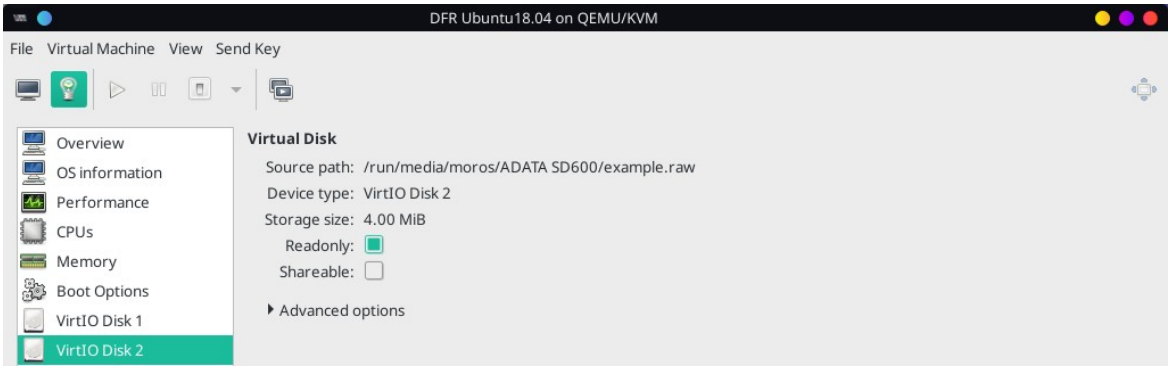
7. Make image of file system **(host OS)**
   dd if=/dev/sdx1 of=/.../case_1.raw

8. Manually inspect file system image to verify it fits the test case **(host or guest OS)**
   Using a hex editor or similar tool for viewing raw data

9. Create expected recovered file object(s) **(host or guest OS)**
   Either the original file or a subset of the original file

For each test (use the same images for each tool):

1. Add raw image to VM as a read-only disk



2. Run DFR tool **(guest OS)**

3. Compare recovered file with expected recovered file **(host or guest OS)**
   diff case_1_expected.raw recovered_1.raw

4. If they are different, manually check recovered file for a false negative and to document what the tool incorrectly recovered **(host or guest OS)**