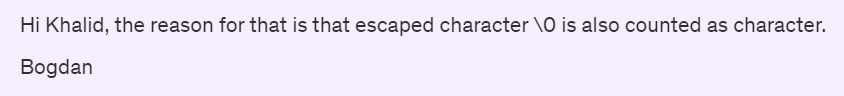
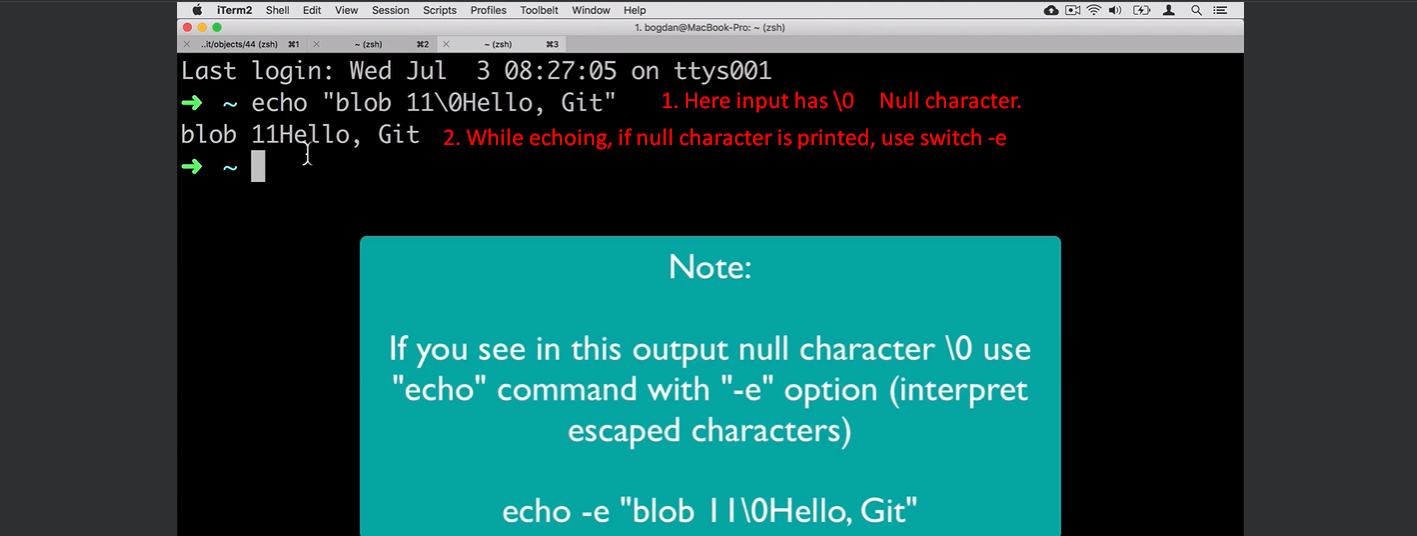
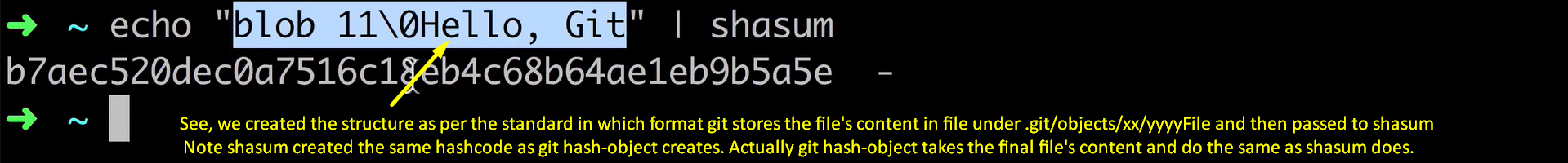
1. **Agenda**:

The structure of every **Git Object**.

1. Diagram

   Description automatically generated
   1. Why length is 11 for “Hello, Git”?
   2. **Answer**: 
   3. Every Git Object is consisting of 4 fields 🡺 **Object Type, Length, Delimiter & Content.  
      NOTE:** No matter, how an object is created using **git hash-object** or using high-level command **git add**, it has 4 fields and same structure.  
      **How we can prove this?**We can prove it by two ways:
      1. By reading the content of the Git Object (Blob) directly.
      2. By creating the SHA1 hash of the content (blob 11\0Hello, Git) as per structure as shown in the slide by using the **shasum** and comparing the hash with that generated by **git hash-object**.
   4. **Read Carefully, it will help**:   
      actually, it is about what is the final input that is given to **git hash-object** and **shasum** when hashcode is calculated.
2. Chart

   Description automatically generated with low confidence  
   Actually, “cat” doesn’t know how to uncompress the content of compressed file.
3. 
4. We will use the 2nd way of verifying the structure of Git Object  
   2nd Way 🡺 Generating the SHA1 HashCode by passing the assumed structure to both utilities 🡺 **git** **hash-object & shasum**.  
   Diagram

   Description automatically generated
5. **NOTE**: Before generating hashcode, please echo some characters with null character without -e, see if null character is being printed or not. If printed, then please include -e (escape null character) while generating hashcode.  
     
     
   If you don’t know how we calculated the content parts in structure, use the following commands  
   You must know the Hashcode for the blob. Say it is ae12  
   git cat-file **-t** ae12 🡺To get **type** 🡺 Like blob  
   git cat-file **-s** ae12 🡺To get **size** 🡺 Like 11  
   git cat-file **-p** ae12 🡺To get **Content** 🡺 Like Hello, Git  
   Now collect them as per structure 🡺 “blob 11\0Hello, Git” and pass it to shasum   
   Like echo -e “blob 11\0Hello, Git” | shasum 🡺 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  
   If we use git hash-object like   
   echo “Hello, Git” | git hash-object --stdin 🡺 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e  
   See, this hashcode and previous hashCode generated by shasum are same even though the input to both utilities are different this is because when passing “Hello, Git” to git hash-object, it will internally convert it to “blob 11\0Hello, Git” and this same will be stored into the blob object.