Introduction

Git object store

```
mkdir workshop_repo
cd workshop_repo
git init
ls .git
```

This is where we talk about the contents of the .git directory. As a reminder:

- HEAD - Our location in the repo
- config - Repo specific information
- description - GitWeb specific file — ignore
- hooks - Scripts that run on events (e.g. committing, pushing, etc.)
- info - Holds the exclude file, which is nice
- objects - Has two empty directories, but we'll work with it in a minute
- refs - Has two more empty directories, but we'll see it again

```
echo 'echo "Hello World!"' > hello.sh
git hash-object -w hello.sh
find .git/objects -type f
```

This is where we discuss the concept of a content-addressable filesystem.

```
git hash-object -w hello.sh
find .git/objects -type f
```

Now we will get the contents of those files

```
git cat-file -p <hash>
```

And we can show how to restore files

```
rm hello.sh
git cat-file -p <hash> > hello.sh
cat hello.sh
```

Now let's make something a bit more complicated

```
mkdir src
echo '#!/usr/bin/env python' > src/hello.py
echo 'print("Hello World")' » src/hello.py
chmod +x src/hello.py
git hash-object src/hello.py
find .git/objects -type f
```

Blobs and trees

```
git update-index -add hello.sh
ls .git
git ls-files
git status
```

Talk about permanent snapshots

```
git write-tree
find .git/objects -type f
git cat-file -p <hash>
git cat-file -t <hash>
```

Talk about how trees can point to blobs

```
git update-index -add src/hello.py
git ls-files
git write-tree
git cat-file -p <hash>
git cat-file -p <subdir hash>
```

Show tree structure

Commits

```
echo "Initial commit" | git commit-tree <hash>
git cat-file -t <hash>
find .git/objects -type f
git cat-file -p <hash>
```

Talk about the contents of a commit. Show commit structure slide.

```
mv src src_moved
git update-index -add src_moved/hello.py
git update-index -remove src/hello.py
git write-tree
echo "Moved src" | git commit-tree <tree hash> -p <other commit hash>
git cat-file -p <hash>
```

Talk about commit parents. Show new commit structure slide.

```
git log <commit hash>
find .git/objects -type f
```

Talk about blob persistence and show total repo structure slide.
Now let's make a merge commit.

```
mv src_moved src
git update-index -add src/hello.py
git update-index -remove src_moved/hello.py
mv hello.sh hi.sh
git update-index -add hi.sh
git update-index -remove hello.sh
git write-tree echo "Renamed hello.sh" | git commit-tree <tree hash> -p <root hash>
mv src src_moved
git update-index -add src_moved/hello.py
git update-index -remove src/hello.py
git write-tree
echo "Merged histories" | git commit-tree <tree hash> -p <hash p1> -p <hash p2>
git log <hash>
git log <hash> -decorate -oneline -graph
```

Show the merge commit slide.

References

```
ls .git/refs
find .git/refs -type f
git update-ref refs/heads/master <newest hash>
find .git/refs -type f
cat .git/refs/heads/master
```

We can use this reference instead of a hash anywhere

```
git cat-file -p master
git log master
git rev-parse master
```

Now let's add another branch to show the branch we already merged

```
git update-ref refs/heads/dev <branch commit>
find .git/refs -type f
git branch
```

Talk about the special reference, HEAD

```
cat .git/HEAD
```

Now let's switch branches.

```
git symbolic-ref HEAD refs/heads/dev
git branch
```

Show the branches slide. What about the tags file?

```
git update-ref refs/tags/v0.0.1_rc1 <moved src hash>
find .git/refs -type f
cat .git/refs/tags/v0.0.1_rc1
git cat-file v0.0.1_rc1
```

Talk about the lack of context for these tags

```
git tag -a v0.0.1 <merge hash> -m 'Released at the workshop'
git cat-file -p v0.0.1
find .git/objects -type f
```

Show the tags and branches slide.

Example: Rebase

1. Talk to the audience about the steps necessary to do a manual rebase
2. Show them the source that can do it

Audience choice

Whatever they want to ask about - This is a possibility instead of the manual rebase, especially if there is no time.