One day you might be overseeing a big team of developers.

Can you imagine how complex it gets to keep

track of everyone to changes and updates to files?

Fortunately, git has a very helpful command

for keeping track of who did what and when.

It's called git blame.

In this video, you will learn about git blame and I

will demonstrate how it is used with a few examples.

One of the core functions of git

is its ability to track and

record the full history of changes

for every file in the repository.

In order to view and verify those changes,

git provides a set of tools to allow users to

step through the history and view

the edits made to each file.

The git blame command is used to look

at changes of a specific file and show the dates,

times, and users who made the changes.

By now, you should know how to use

commands like git log to see the changes made.

I will now use the feature.js

file to demonstrate how git blame works.

Let's get started with the git blame command.

To run the git blame command,

type git blame and the name of the file.

In this case, feature.js.

After pressing Enter,

git returns a list of all changes on the file.

To understand what is happening,

let's break down the blame messages

and go through it line by line.

Firstly, let me guide you

through the format of each line.

Every line will start with the ID and then the author,

the date and time when the change was made,

and the exact line number where the change occurred.

Then the actual change or content is also returned.

The ID is a reference ID of the commit.

The same ID might appear in several lines.

This happens when a single commit

has been made by the same developer.

The author is the person who created the commit.

The timestamp is the date

and time when the changes were committed.

Line number represents the location in

the file or the exact line where the changes were made.

The content is the code that was added to the file.

Now that you know the meaning of

each line in the blame output,

let's explore a real example.

In this example, you will check who made changes,

when the changes were made,

and also what changes were added.

For the purposes of demonstration,

I will be using a public repository called MK docs.

MK docs has various different contributions

from many different developers,

so it's a good way to see a log file

from all the changes of specific files.

To begin, I will check inside

the directory by using the ls command and

passing in -l to get a list

of all the available files and I'll just pick one.

The file I will use is called setup.py,

which is a Python file.

In order to examine the different changes to that file,

I create a command called

git blame and then pass in the name of the file,

setup.py and press Enter.

The output will list

all the changes made by all the different developers.

It will also indicate the timestamps and line numbers,

as well as the actual changes that were made.

Now, I will talk you through the output.

Starting from the left of the list is

what's called a hash dash ID.

It just represents the commit of when a change occurred,

then the name of the developer

who worked on the file is listed,

and then you have the timestamp when the change went in.

Next is the line number in sequential order.

Finally, the actual change that was implemented.

I can scroll through the list of

changes all the way to the end of the file,

depending on the size of the file

or whatever number of lines it has.

If I want to exit out, I click on Q.

This will clear the screen to make it easier.

Take note that git blame on its own and

bypassing the filename will list the entire file.

In a lot of cases,

you will work with very large files and then you need

a way to abbreviate the output or chop it down based on,

say, line numbers.

Fortunately, git blame offers a flag for that.

To do this, I type git blame and pass in the flag

of -l and specify

the starting line number and the end line number.

I will type 5, 15.

Then pass in the file name again,

setup.py, and press Enter.

This time, a smaller subset is returned that

only starts at line 5 and ends on line 15.

The output indicates that there are

four different changes made by

five different developers across these lines.

Let me give you a few more tips around git blame now.

Firstly, you can change

the format of how the list is displayed.

This is similar to what you can do with

the ls command on the Unix commands.

You can also pass in

a -l flag for changing the output itself.

Again, let's run git blame -l,

followed by the filename, and press Enter.

This time, there are a few changes to the output.

For instance, the hash dash ID

is in its full length form.

It's not in the shortened version.

The output is now a bit more detailed.

You can also control if you want to show

email addresses or change the date format.

These are the examples of

the various things that you can do.

Secondly, another aspect

of using git blame is that you can see

detail changes or the actual commit changes

of a specific hash dash ID.

To do that, I will run a git blame command on that file

again in order to copy a hash dash ID from the output.

Now, I will use that with

a git log -p and pass in

the hash dash ID and press Enter.

This gives you the actual change that occurred.

Just to clarify, git blame

will display to you the point where it was changed,

git log will give you the detail of the change.

I always use the two in conjunction to

get more details about what changes occurred.

You are a team leader at a software development company. You run the git blame command to see the changes on all the files for the week. What is the order in which the change information will display in each line?



<ID><Author><Line number> <Date><Time><Content>



<ID><Author><Date><Time><Line number><Content>



<Author><ID><Line number> <Date><Time><Content>



<ID><Author><Line number> <Date><Time><Content>



<ID><Author><Date><Time><Line number><Content>



<Author><ID><Line number> <Date><Time><Content>

Correct

That's correct! The order in which the change information will display in each line as follows: < ID><Author><Date><Time><Line number><Content>