Workshop 1 Intro to Git

July 13, 2026, 9:29AM

1h 37m 03s

Gianni Vesuviano started transcription

Simon Stanley 0:03

here at UKCEH.
And I'm going to run us through version control, and in particular version control with Git. Git is a...
piece of software that manages how you develop your code. Let me share my screen. It.
Uh, it's the wrong screen. It's always does this.
Sorry, 2 seconds.
This one.
Put this here.
Okay.
How's that? Can you see that?

Amulya Chevuturi 1:25

Yep.

Simon Stanley 1:28

Right, lovely. Let me just quickly move some other things around.
Okay.
So yeah, welcome.
So what we're going to go through.
Four things broadly, and this is...
Oh, actually, sorry, I should check before we start. Is everyone...
able to log into GitHub.
Um...
We gave a prompt earlier, but I'm aware some people joined after.
Let me put a link in the chat.
So this is the page we want to, oh yeah, this is the page we want to get to.
You may be prompted to log in. What I suggest, we don't need it just yet. I'm going

to do some slides and a demonstration and then there'll be an exercise where we'll use this. So I'll carry on. And then if you have problems logging in,
during that time, then just shout and we'll try and get to them when we get to that point.
Okay, so we're going to look at four things. First of all, the very basics of version control and what that actually means, what that is. And then we're going to look at branching and merging and remote repositories and what they are. And then just some further tools.
This is all.
Complete introduction, there's no experience required. It's just to give an overview of what Git can do and why it's used, when to use it, and even when not to use it. It's not a tool for everything.
So...
The first thing I want to talk about is just how we work normally without version control. And what we do, say we're working on a text document,
We do our first bit of writing, and then we do, we save it, and then we do some more, add some more, and we save, and we save, and each time.
we make that save, we only ever keep the latest version. So every time we update and save a file, we create a new version of that file, but it's only the latest one that is ever available.
So, all these previous versions are...
Gone.
So we have no access to them and we cannot see the changes that were made in each save. And for many examples, that's absolutely fine. You only ever need the latest version. So something like a text document or a presentation, as you improve it, it's okay. You don't need to know how it was before.
The.
When it comes to developing code.
This can be very useful, and so we'll see.
We'll see why.
So this is what version control does. Version control software like Git, there are others, but Git is certainly the most well used and popular. They give us access to see and interact with every change that's been made. So all these
or the history of the documents.
The other key difference is that version control, it works on directories instead of

individual files.
So, as I say, when you're working on a text document, you're just saving that file. When you're working with version control, you're saving the state of the entire directory, so that's all the files within it get saved together.
So I'm going to jump straight in and...
do a demonstration of this just to see how it works. Okay.
Please bear with me. Live demonstrations are always...
Interesting.
Can you all see the screen, this black screen?

Amulya Chevuturi 6:04
Yup.

Simon Stanley 6:05
Lovely. So this, thank you. Yeah, great. This is what you will be looking at when you do the exercise on GitHub. So hopefully this will be familiar. And we have 3 windows here. On the left, we're looking at the files that are in this directory.

AA

Ajadi Sodiq Abayomi 6:06
Yes.

Simon Stanley 6:25
Um...
The main window is we can look at each file, the contents of them, and then down here we have a command-based tool. And it's in here that we're going to type our Git commands.
So.
The first thing I'm going to say is that, well, the first thing I'm going to say is the contents of these files are Python, and...
If you have no Python experience, it doesn't matter. What is in these files is not important. What's important is that we're going to change them and save our changes and track the changes. So it's all very simple Python
Um...
Obviously.
As things get more complicated, as you use Git, then it becomes more useful. But for

this, we're going to keep things very simple. So what we have is we're going to make a zoo.
So the first file we have here is, it's a Python class for an animal, which is like the blueprint for an animal. And an animal has a species name. It has a type of species. We're going to give it a name. We're going to say how many legs it has, what noise it makes, and how old it is.
And then this animal comes with certain functions or certain actions, so it can make the noise, you can feed it, and all these things. We'll go through them. But at the moment, each of these functions, they don't do anything. They just pass.
Um...
There is also, then we're going to do a similar thing, but for a zoo, and we'll go through it. The zoo can do certain things. We can add animals to the zoo.
And then, uh...
this script kind of puts the animals in the zoo. But like I say, don't worry about if the code doesn't make sense, that's not important.
So.
At this point, this is just a normal file directory. This is not being tracked in any way. So if I make changes to it, if I start adding...
things to it. Here is some documentation. Documentation is very important.
And then I would save that.
And just like any other file, we're developing it. But what I've just done is not tracked in any way. So if I want to undo it, I have to manually undo it and save again.
File, Save.
So, the first thing I'm going to do is I'm going to turn this directory into a Git repository, and all that means is we're going to tell Git that we want to track the changes here, so...
I have, again, apologies if you're not too familiar with command line tools, but there's nothing to worry about if you don't understand. I am currently located in this command line tool. I am located inside the directory.
And I'm going to use the first command I'm going to show you is called git.
In it, this is big enough, by the way, to so I make it zoom in a bit.

Amulya Chevuturi 10:07
Yeah, a little bit zoom in might look nice.

Simon Stanley 10:11

How do I do that? This one?

Should know this.

Zoom in, no go.

Tom Keel 10:23

I wonder if you do, oh yeah, control plus.

Simon Stanley 10:25

Control plus equals. How's that?

Bit better.

Tom Keel 10:30

What does one more look like? Just one more zoom? Is that too much?

Simon Stanley 10:38

Simon.

Is that even more obvious?

Tom Keel 10:41

That's probably, yeah, that's for the best.

Simon Stanley 10:45

Yeah.

Okay, so the first command is every command starts with the term git, which is telling you which software to run, and then followed by some command telling you what to do. So git init, and this...

changes nothing in the directory. It just tells Git that we are now tracking this repository. Excuse me. The second command, which we will see a lot, is Git status. And this is, this tells us certain information. Oh.

OK, ignore this.

That was unexpected.

Okay, there we go. That's what we should see. This prints out information about the repository that we're in. So there's a few bits of information here. It's telling us that

we're on a branch called master.
Um...
I'm actually going to change that. Sorry, excuse me.
Okay, so ignore my, we're now on a branch called Main.
Do another get status.
It says there are no commits. We'll talk about commits in a minute. We'll talk about branches later. And it's saying that there are these untracked files. So it's listing these things which we see.
in the directory, which are currently untracked. So the first thing, this means that if we make changes to them, Git won't recognise those changes. It's ignoring these files. So the first thing we're going to do is going to add these files to the repository.
And that is done with git add.
And then we...
List each of the files.
And just so you know, when it comes to the exercises, you won't have to do this. This will be done for you. So don't worry too much.
I do another get status.
And now it's telling us that we have, again, no commits, and it's saying we have these changes ready to be committed. So.
We've put them in a staging area and then we're going to commit them and we'll talk more about this again later.
So this command is the git commit command. This is like...
file save. This is how we say to git, we want to save the directory in this state. And every time we do that, we'll be doing a lot of it.
We have to tell, we give Git a message to describe what the change is that we've done. So in this case, we've just done an initial commit. Let me spell that out. Just to start everything off.
And then I'll do another get status.
So I'm going to delete this because we don't need it.
Yes, lovely.
Status, so now it's telling us we're on this branch, there's nothing to commit, and everything is.
clean working directory. Okay, so this is the point from which we start. Oh, yes.

Amulya Chevuturi 14:51

Sorry, Simon. Robyn is asking how to get to the terminal.
But I don't think you're currently running the training now. They just have to follow you, right?

Simon Stanley 15:07
Yeah, exactly. So when we get to the exercise, I will instruct you how to get to this screen and all of this will appear. Yes, thanks, Amulya.

Amulya Chevuturi 15:19
Yeah, so Robyn, now just follow along visually what Simon is doing and then you get to do the hands on one yourself later. Thank you.

Simon Stanley 15:33
Yeah, and please...
I'm not expecting this. I know all this kind of code can be a bit much sometimes, so please don't worry about absorbing it all now. We're going to go through and repeat and I'll explain what I'm doing and what I've done. So what I'm going to do now is I'm going to start changing the
the code and start making some developments. So the first thing I'm going to do is this function here, make noise. So this is going to print out the noise that the animal makes. And that's.
Quite straightforward, so...
print
self dot noise.
Okay, I'm going to save this.
And then I'm going to do a git status, so we've now.
Oops.
We've changed the state of our directory. Something has changed in there after I did the save and Git has recognised this.
So the next command I'm going to show you is git diff.
And when we see that a change has been made, which has not been committed yet, so we haven't made the git, the commit is like the git save. We can use this git diff to see what the difference is between the state.
that Git knows and the changes that we've made. So all it is is just exactly what I've done. I changed the word pass to this. So this is very useful to actually check what

you're going to commit. So
It's just...
Bring this up again and I'm going to do that git add.
Name the file that I want to add.
And then, git commit with a message.
Add.
Noise
Function.
OK, and then I do I get status again, and I see everything is.
Up to date within Git.
Um...
Okay, let's make another change just to repeat exactly the same process. So this time we're going to do the feed function. And what we're going to do is we've got this attribute that says if the animal is hungry and the animal always starts out hungry. So we're going to say if
Animal is hungry.
Then, we will feed the animals, so we'll...
Print
Feeding.
I tried to keep this quick and easy, but...
self
dot
name.
And then let's say they...
They then make their noise, so self dot make noise.
If they're not hungry, you can print.
is not hungry.
Okay, again, it doesn't really matter if this code works or not. Just get some things in there.
And let's have a look.
Yes, there's been another change. We'll do a git diff to look at that change and check it. Let me see what we've just done.
Um, we then at...
and commit. If it's not clear by now, Git can be very repetitive.
But useful.

Add feed.
Function.
Okay.
So...
The next command I'm going to show you for this section is git log. And again, I will list all these out, all the things I've done, just trying to give a flavour of what you can do.
and git log lists all the commits that you've done. So there was the initial commit where we added all the files into the repository. We then added the noise function and we added the feed function.
Each of these commits comes with this big number, and this is a unique number for every single commit, or hash they're called.
And what that means is we can use that to actually look at each commit that we've made. So if I want to view this commit where we added the noise, I use the last function I'm going to show you for this section, and that is git show.
So we do git show and then we copy this commit.
And it prints out the change that we made, which is where we added.
the noise function.
Okay, I'm going to leave it there for now.
Let's move this away.
And what I'm going to do is just go through all of that again, but...
More visually.
So the first command was the Git init, which you won't need to worry about when it comes to the exercise. But we have this directory, and in this directory we have these files. And then we use Git init to turn this directory into a Git repository, which means Git is now tracking the changes that happen.
Put in there.
The next thing we did was we did a git add and we listed all the files and we committed them into the repository. So all these files are now being tracked in this repository. The reason we have this step is that there may be files in this directory that you don't want to track. So you have to explicitly say that you want them all in there.
And then, so we made this initial commit, which we saw, and it had some some number.
Um...

We made the...
We made changes to that, to the animal file, and we saved those changes.
And then we use two commands to just look at what we've done. So git status, which we, as I said, we use a lot, and it just shows the status of the repository, including any files that have changed since the last commit.
And then we use git diff to show the actual changes that have been made since the last commit. So we see what we did.
So then we want to commit those new changes. So we've made the changes, save the file. We use git add, and we specify the file name that we've changed. And this stages the file, which means it becomes ready to commit. And I'll talk more about what the staging area is for.
But once it's been staged, we then commit it, and then we add this new commit onto the list.
Um, and yes.
Just a reminder that every time you commit, you have to give a commit message.
You have to give a short description of what you've done, and this just means when you're looking through the history, you can more easily see what the commits refer to.
Um...

Amulya Chevuturi 24:21
Simon, there is a question in the chat. What is the difference between git init and git commit at the initial stage?

Simon Stanley 24:22
Yes.
Sorry, yes.
Oh yeah, that's a good question. So git init.
Is just telling is telling Git to.
Track this directory, turn it into, whereas it starts off as just a normal directory, it's git init tells git that this is now a special directory in which there will be changes that we will make, and when we make those changes, track them.
but we have to explicitly tell it what in that directory we want to track. So this, the git init sets it up, and then the first git commit is just like any other commit. It's as if instead of making a change to the file,

Let's imagine there were no files in it at all, and we create a new file and we add some stuff to that file. That's just like making a change to the file. We're sort of putting something new in the directory. So we do this initial commit to put it in there. It is just like any other commit.
But it's kind of the first one that sets it all up. Does that answer the question?

aninditadg2017 25:55
Yes, thank you so much.

Simon Stanley 25:57
Okay. And please, yeah, feel free to interrupt me at any point.

Amulya Chevuturi 26:02
Yeah, and apologies, I didn't know your name because I could only see your username. Sorry.

Simon Stanley 26:11
So when we keep doing this and we make changes and we commit changes, we end up with this list of commits and we saw that with git log.
And what we're doing here is we're creating a branch. That's the terminology that Git uses.
And.
So we, as you, if you remember when we did the first status, it told us that we were on the main branch. Sorry, it says master here, but.
Everything's we've changed it to main, but that's it's the branch can be called anything.
Um...
And one thing to remember is...
When, when we say we are on a branch.
What that means is the directory is in the state of the latest commit. So when we say we are on the main branch or the master branch, whatever it's called, it means the directory is in a state.
of the latest commit on that branch, which is maybe makes it more confusing than it is. It just means you're at, you know, if you're making saves, you're at the latest point.
But the reason I say that, it is possible to move the directory to any state on the

branch. You can jump around if you need to, but generally you will always be on the later state.
And then the last one I showed you was this, so git log shows you this big list of all the commits that have been made. And then git show is a way for you to inspect any of those commits to have a look at what change was made. Obviously, the example we've done so far,
Oh, Gianni, is that a question?
Are you on mute?

Gianni Vesuviano 28:16
Sorry, yes, there was a question in the chat.

Simon Stanley 28:19
Oh, sorry, yes. Hi, Simon. Thanks for the session. I'd like to know if the message after commit is optional and what is the penalty for omitting it?

Simon Stanley 28:31
It is, it's not optional. Although the command, the example you've given, git commit -m with a blank, is a good question because...
I mean, let's try it.
I'm going to make a change.
I'm going to just put a full stop there, so it's nice and simple.
Git.
Stick it status first, see the change, Git add. That's a really good question, actually.
OK, and then git commit -m, no message.
Ah, yes, so it says no. Aborting commit due to empty message. So you have to give a message. You have to supply some small description.
as to what the change is. Now, obviously...
It's easy when the changes are simple, but sometimes you can make huge changes and how you sum that up in a few words is difficult. And lots of times you see commit messages which are a bit like change made.
And Git's not going to complain about that. You're perfectly welcome to do it. But it is good practice.
for development to add more descriptive names. And also, like I say, you can make huge changes before, you can change lots of things before you commit it, which

makes it.
harder to describe. So another good practice and what is encouraged in software in version control.
is to kind of isolate your changes, to make it a change that you can describe in one sentence. And that's sometimes easier said than done, but it's the ideal.
Okay.
Yes, so that's the final command that we saw was git show, which allowed us to have a look at any of the commits that we've made.
Okay, so we're going to go for the first exercise and you'll hopefully be pleased to know it's basically repeating what I've done. Because these, what I've shown you is really the...
The basic.
Use that you need for.
Git so.
If you can, I think I'll post that. I'll post it in the chat again.
And we can follow along.
So if you can get to this page.
And...
Login.
Hopefully, you all have GitHub accounts.
And maybe...
by a show of thumbs up on the message I've just posted to let me know when you're ready and you have this page loaded.
And...
So I'm going to put a thumbs up for me because I've done it.
Ahh.
And if you have any problems.
Just say, and we'll try and try and fix.
That's good. Nine, lovely. How many are on the call?
Twenty-one.
We have a couple more minutes.
Well, doing well.
Anybody having problems?
So for those unfamiliar with what this page is, this is a repository on GitHub. And I will talk about GitHub a little bit later.

But it is a website where you can upload.
Your repositories, so...
Once you've made a repository on your computer, like I did with the git in it, to make some changes.
You can actually upload all your work.
into GitHub and that just means it's either publicly available or available to certain people that you want to help develop the code.
But yes, we'll talk about that a little bit later. For now, I'm just using it as a, it has a feature within it which allows us to basically do what I've just done. So we can kind of take a copy and start doing some development.
So I see 14, 14 out of 21.
I will carry on for now.
And as I say, just message or shout out if you're having trouble.
But what I'd like you all to do is this green button here that says code.
Click on that and you have this pop-up comes up and you have two options, local and code space. Click on code space and then click on plus.
And this should hopefully open up a new...
Window.
where you see something very similar.
To.
The screen I just shared.
This, it can take a little bit of time to load. So again, I'll...
Give a couple of minutes, but.
If you see on the left, we have, again, this list of files in there, and there's a few differences. We've got our animal file, which is the same, the same functions with nothing in them, and the zoo and the my zoo, you can ignore these for now.
but we're going to look at this exercise1.text file. This is here for your reference. I'll leave the same thing on the screen so you can see it. But there are five exercises to have a go up.
And they should pretty much be.
what I just did. So firstly, we're going to add a noise function. So this is all in the animal.py file.
Animal.py, add a noise function, use git status and git diff to check out the changes and see what they look like, add and git commit to commit those changes, and then do it again with the feed function.

And there's a little bit extra here. You can add to the information function. I would just again emphasise it really doesn't matter what you add. You can write whatever you want into these functions. You can just change them in any way.
The point is to make the commits so the code doesn't have to work.
On this third one, there is a slight difference in that we're going to actually change two functions from two files. So the list animals function, sorry, I should have written this down. This is in the zoo file. There is a
list animals.
And what this, what I want to show you here is that you can commit 2 files at the same time. I should have demonstrated that really.
And then again, list of the commits and revert one. So I will leave you.
to have a go. Let's say, how are we doing for time? Let's say 15 minutes.
When does this end, Amulya?

Amulya Chevuturi 38:04
Gianni.

Simon Stanley 38:05
Twelve.

Gianni Vesuviano 38:06
This will end at midday.

Simon Stanley 38:08
12, okay, so we've got time, yeah.

Amulya Chevuturi 38:09
No, but we need, we need like at least 15 minutes.

Simon Stanley 38:15
Okay, for the for the tech thing.

Amulya Chevuturi 38:15
to do the tech, maybe 10 minutes for the tech testing, 10 minutes.

Simon Stanley 38:18

Okay.

Gianni Vesuviano 38:19

The tech testing starts at 12.

Amulya Chevuturi 38:21

Okay, sorry, sorry.

Simon Stanley 38:23

Okay, lovely. So we've got time.

Simon Stanley 38:26

Yeah, so let's say 15 minutes.
Again, please ask questions if you get stuck. I'm going to leave.
This here, if this helps.
So, this is a...
description of all the functions. Oops.
Ignore this bottom one, get revert. Feel free to have a go, but we ignore that for now.
Yes, okay.
Let me know if you have any trouble.
And we'll, yeah, we'll say, let's say 20 past. Does that work? Yeah, 20 past.
And obviously all of this will be available if you want to finish off later.
Sorry, just to clarify, don't worry about step 5.
Um...
That's the git revert one, which I haven't covered, but feel free to experiment and have a go, but don't worry about that one.

Tom Keel 40:55

Ohh, did you see that message in the chat?

Simon Stanley 40:58

Yeah, just having a look.

Tom Keel 41:00
Oh, that's an unexpected one.

Simon Stanley 41:07
Is that?
Is that?
Is that an me thing?

Tom Keel 41:16
No, that's linked to their account, I think.

Simon Stanley 41:24
Um...
Yeah, that might be a tricky one to...
Uh...

Amulya Chevuturi 41:32
Um, she's asking if you can do, if you have Git installed, Avintha, then it might be okay to, is it it is okay right to do it on her local machine?

Simon Stanley 41:46
Yeah, if um...
If you have, do you have Git installed? Oh, is it? Oh, yes, I see. Yeah, yeah, yeah, absolutely. If you have it installed.
I would suggest...
As I say, it doesn't matter what the files are, so you can, if you can make a directory and...
Find that directory on your command line, and...
Just do it there if that works. Sorry about that, Avintha.

Senavirathna, Avintha (IWMI-consultant) 42:25
OK, sure, no worries, I will do like that, OK.

Simon Stanley 42:26

And then...
Okay, is there something you need to do before typing in git init and git diff?

Amulya Chevuturi 42:32
What?

SA

Senavirathna, Avintha (IWMI-consultant) 42:33
OK, thank you.

Simon Stanley 42:40
Typing in git init. You don't need to type in git init. That bit is done already.
Robyn and for git diff.
When you, after you've made a change and you've saved the change, you use git diff to...
to look at that change.
just to see what you've done. It's purely just to make sure when you commit, you're committing what you expect.
So...
Yeah, so don't worry about getting it and...
Yeah, git diff. You don't have to do anything before it, but it's like a useful one to view what you've done. Does that answer your question, Robyn?

JR

Jones, Robyn (MECP) 43:30
Yeah, I came up with an error at first, but I think I just misread the add noise bullet point. So we just type in git status, git diff, git add, and then git commit.

Simon Stanley 43:44
Yeah, so you need to save before. Maybe that's the answer, actually. You do have to save before git diff. So you make the change, save it.

JR

Jones, Robyn (MECP) 43:45
Yep.
And do we make the change in the like animal.py file right up at the top?

Simon Stanley 44:00
Yes.

Jones, Robyn (MECP) 44:01
Yeah.
Thank you.

Simon Stanley 44:03
Okay.
I say that, does it actually...
I think it actually saves automatically.
Um...
Yeah.
If I need that.
Yeah, thanks, Amulya.
I didn't practise with breakout rooms, so if it, if that comes to it, I might ask for your help.

Amulya Chevuturi 45:13
Yeah, we have, I think we have breakout rooms created. Don't worry about it. This is just in case.

Simon Stanley 45:20
Thanks.
Yeah, we can do, Robyn, if that, if that helps, yeah.

Jones, Robyn (MECP) 46:48
Yeah, I'm very lost, sorry.

Simon Stanley 46:51
Um...
Should I, should we do it? Should we do it together? I can, I'll do it on the screen now.
Um...

So.
First one, which at which point did you get lost, Robyn? Was it?

Jones, Robyn (MECP) 47:12
Like, right from the start.

Simon Stanley 47:14
Straight away. So that's all right. That's good.

Jones, Robyn (MECP) 47:15
Thanks.

Simon Stanley 47:19
So the first one is add the noise function.
So let's just, just for the sake of purposes, I'm going to do things that don't even make sense. Just going to make some changes.
And so, the once I've made a change to the code.
I'm going to use this.
git status, and this is just going to show us that something has changed in the animal file. It recognises that there's been a change. So then I use git diff to look at what that change is. None of this is necessary. It's just useful to know these commands so you can
See what's happened.
And.
Does that make sense so far?

Jones, Robyn (MECP) 48:12
I think I lost the terminal at the bottom, so I can't type in below there. And I don't know how to get it back.

Simon Stanley 48:16
Ahh.
Okay.
In that case, let's figure that out. Okay, I think, so if you, these, you see these three lines in the top left.

Jones, Robyn (MECP) 48:28
Yep.


Simon Stanley 48:29
and then terminal, new terminal.

Jones, Robyn (MECP) 48:34
Ah, okay.


Simon Stanley 48:35
Yeah, does that work?

Jones, Robyn (MECP) 48:37
Yes, thank you.


Simon Stanley 48:43
Do you want, should I carry on going through it? Is that useful?

Jones, Robyn (MECP) 48:43
Mm.
Yes, please.


Simon Stanley 48:46
Would you want to? Yeah, so we've...
We've made a change. We've had a look at that change with git diff, and then we're going to commit that change with git add.
And we list the file.
So I'm doing a copy and paste here. I mean, you can just type it out, animal.py.
And then I'm doing a git commit -m.
Change.
Noise function, some useful commit message.
Running that.
And then that's that change has been committed now, and we can check that by looking at git status.

And the fact that it says, um...
It doesn't it doesn't come up with this and shows that something to be modified means that it is it is up to date and we can double check that with a git log.
So, git log lists all of our commits.
And at the top we see this is the one that we've just done. It starts from its bottom up, if that makes sense. So the bottom one is the...
the first, the commit that was furthest away in the past.
And this is the most recent one at the top.
And then if we want to look at that again in more detail, we use the git show.
And then we take, we copy and paste this big hash number.
Oops.
Let's allow that.
And then that just shows us this commit that's happened. And again, this is not changing anything. This is just having a view on what's happened. And we can see that we've added this.
Nonsense commit there.
While I'm at it, the other thing I wanted to show is on one of the exercises. So if I change something here.
Feed equals true again.
And then, before I commit, I go to another one and I make a change here.
Print.
Animals.
When I do a git status now.
It recognises that two files have changed.
And I can do exactly the same process. I can do a git diff.
And it will list.
both files, so here's the animal file, the change that happened there, and then the zoo file and the change that happened there.
Um...
And then I can...
Add both of these.
When we do the git add command, I can list both of these files, just one after the other.
Like this.
And then git status, sorry, git commit -m.

add
What did I do? I listed the animals and I added add feed and list animals.
git log.
And you see, that's just one commit.
And when we view that commit.
git show.
We see that two again, 2 files have changed.
Okay, we'll go on to the next section.
We will be repeating this more, so it'll be more practice.
The key takeaway is this idea of...
committing our changes to the directory.
and the fact that we can list and track those changes.
Okay.
So.
One of the really useful and powerful features within version control and within Git.
is this idea of branching. Everything we've done so far, we've been...
making all our commits on one branch, the main branch. And every time we add a commit, we are building that branch.
So in Git we can create new branches that can chain their own chain of commits.
We will see this in more detail.
Um, um, the reason we have that is so when when we can...
Create these branches of commits, which are kind of commits that go off in isolation on their own. It means that...
we can maintain this main branch without breaking things. So with code, when we do lots of development, sometimes we want to try out changes before we actually commit them to the final repository, the final branch.
So yeah, this means we can test and experiment and develop and...
there is anything that you do on another branch is going to have no effect on the main branch.
So it's completely safe. You can break things and there's no problem.
So there are three key commands for branching. The first one is git branch. And what this does is it lists all the branches that you have. So in our repository, we should only have one called main.
I'm aware that I actually created another one because we had one called master, so we'll have a main and a master, but just ignore the master one.

At the moment, we just have one branch, which is called main. If we created a new one, then it would list them all.
So the second command is git switch, and this is where we switch the state of the directory, the state of the repository from one branch to another.
And so, this, as we switch between switch between branches, the content of our repository, what's in our what's in our repository will change.
Um...
And remembering, as I said earlier, when we say we are on a branch, it means the directory is in.
is changed to the state of the latest commit on that branch. So if we did git switch main, we would be on this commit, the latest one.
Once we have created a new branch and we see that and we do the git switch, then we move to this.
State.
And then the third command is creating a new branch, which is obviously the first one, and that is git switch with this -c, and then you give it a new branch name.
This creates a new branch from the commit that you are on.
And this new branch, it starts from...
where the commit, the latest commit that you're on, when you run this command, the new branch will start in the same place. So they will be identical to start with.
Again, we'll see that.
Oh, oh, I forgot it. Yes, so and then as we add commits, that branch will grow independently.
Okay.
Another demonstration of all of that.
So, we are on our...
Main branch, and the first thing I'm going to do is I'm going to create a new one.
And that's with git switch -c. So -c is that the “c” stands for create. So this is saying switch to a new branch, but the branch doesn't exist. So -c is create that branch. And we're going to call it dev.
Which... development branch
So now, if I do a git log, which is that...
Command that shows us all our commits.
We see this top commit here. It tells us that.
There are actually 2 branches which are at this commit, so dev and main.

are both identical at this point. They are both branches that contain the same history.
So, I'm going to make a change, and what I'm going to do is...
Add an animal.
All right.
So in this Python code is for a zoo. We're pretending we're building a zoo. And this zoo contains animals. And this function adds an animal to our zoo. So quite simply, Self dot animals, whoops.
which starts as a list, an empty list, so there are no animals in the zoo to start with.
And it appends a new animal which has been passed into the function.
Into that list.
Nice and simple.
Well, it's one line in that simple.
And okay, let's do the same thing. Let's...
First of all, have a look.
git status and git status tells us that we are on this dev branch. It's important information.
and it shows us that there's been a change. So git add, do the same things.
git commit -m add animal function.
OK, so let's do a git log again.
And.
And so...
This is a log of all the commits on the.
Development Branch.
Um, and it...
happened that it shows us that this point is where the main branch stops, but let's not worry about that. It's just the point is, like before, we're on a branch, we've added a commit, and the log shows us that new function that we added, the add animal function.
So what I'm going to do now is switch back.
To the main branch, and just keep an eye on this function here. This is the one I've just changed on the development branch, and I'm going to type in git switch.
And because we don't need to create it, I don't need the -c. The branch main exists, so git switch to main.
And you see when I did that, our change is now gone. Because that change was not made on the main branch, it was made on the development branch. We are now

back to the same state that was the latest state of the main branch.
So, I'm going to add, I'm now going to add, make a change on the main branch.
So let's add this, who's hungry?
function and it's going to be a list.
So.
Animals.
print
animal
is hungry.
So this will list through all the animals that exist in the zoo and just print the animal status as to whether they're hungry if we remember.
The animal has this is hungry attribute.
Okay, but that's fine. Don't need to worry about what it is. We're doing the same again.
Lots of lovely repetition.
Yes, there's the change. We'll just use a git diff to just have a look at that change, check there's nothing unexpected in there. That's just what we want. That's good. So we're going to git add.
The zoo file.
git commit.
add
Who's hungry?
Function.
Okay.
Now.
Bear with me because I'm aware this can get a little bit...
strange when we start dealing with branches. But so we have, we're on our main branch. We have the change we've just made on the main branch. And what I'm going to do is I'm going to switch back to the development branch. And we see we don't have this one. This is how it was passed. We do have this one.
Because we did this on the main branch, we're going to do a git switch to dev, our other branch.
And we see this change goes back.
disappears because that was only done on the main branch and then this one comes back.

Um...
OK
So, hopefully that...
Somewhat shows.
the kind of things we can do. And again, I'm just going to go through all of that, but more visually.
As to what's just happened there.
So...
The first thing I did was a git switch minus, oops, sorry, clicking away, a git switch -c and created a new branch. Sorry, I called it devzoo on the presentation, but we just called it dev in our example. So we created a new branch called devzoo.
And then it didn't change anything at that point. It just added a new branch. And our two branches, dev and main, were at the same point. And these, so these commits that are already there now belong to both those branches.
We then switched to, we were then on the dev branch and then we made a new commit. So we did this add animal.
function and so the dev branch kind of went off on its own. The main branch stayed where it was.
We then switched back to the main branch, so we came back to this point, and then we added a new commit onto that branch, the who's hungry. So our branches have forked, they've diverged from each other. There's one change was done on this branch, another change was done here.
Um...
To this.
That's what's happened.
So the next bit I want to talk about is merging.
So branching allows us to develop things in isolation. So we can keep a main branch, which we kind of know where things work. We know everything's okay. We want to make some changes. And we do them on a branch, so we don't
break any of the things that we know work.
Once we've made those changes and we're happy with them.
Um, we would want those changes to go on back, back onto the main branch, so... to move over and shift onto this branch and this is called merging and it's a very useful thing.
And this is done with a command called git merge.

and you specify the branch that you want to merge into.
So there are two types of merging that can happen.
You don't need to, this is, Git handles this on its own, so we don't have to do anything. We always just use the same command, but it's just worth knowing the two different ways that this can work. So the first way is called a fast forward merge, and this is where
Nothing has changed on the original branch; the commits can just be added on top, so...
Let's say, for example, we did not switch back to our main branch and made a change. We just made a new branch, did some changes, did some changes, and nothing changed here. Then when we do the merge of those new changes on the development branch back to the main branch, it's very simple.
It's just going to put those two commits on the branch, and it's as if we did just do them on the main branch originally. So there's nothing clever.
The other type of merging, which is called auto-merging, and this is where Git is... being very smart and working out how to...
combine things is like in our situation we made commits on both branches so
Where other commits have been made or merged into the original branch, Git must work out how to combine those changes.
So, to do this, well, Asher, so if that makes sense. So, where we had our main branch, we made a new branch, we made a commit, and then we went back onto our main branch and made another commit. What we want to do is kind of
get those commits all onto one branch so they all merge together.
Because Git has to do some working out.
as to how to combine all these changes together, it actually creates a new commit on its own, and it's called the merge commit.
And we'll talk, we'll get onto it a little bit later, but not I won't worry about it in this session, but if you can imagine.
If on one branch, if on two branches we made different changes to the same function, then there's going to be a conflict. So Git will recognise that two separate commits have been made making changes to the same thing. And in that situation, it will actually
Prompt you to.
resolve what should be done, whether it's, is it the change from that branch, is it the change from this branch, or is it some combination of the two and you have to

manually go in and resolve that. We're not going to look at it today, but for that reason, that's why you have this merge commit, because there is this extra step that needs to just say,
Here is how those things were combined. Most of the time, Git can do that combination itself.
Here we are. So in most circumstances you don't need to worry about this, but there are times when we make change to the same part of the file, we get a conflict.
So I'm going to demonstrate some merging.
This one.
So what I'm going to do is I'm going to go to the main branch. You can see here it tells you which branch I'm on, which is very useful. I'm going to do a git.
Switch.
Main.
So we should see our change appear again, this back to the state where we added this one, but we don't have this one.
So.
When we do a merge.
We go to the branch that we want to pull changes into. So we want to take the changes that were done on dev and merge them into main, which means we...
We go to the main branch and we do git merge and we name the branch that we want to merge in, which is dev.
And, in this case...
It should be straightforward, it should be able to work it out.
And what we have here is this is Git saying.
I've created a merge commit. I've given it a name, which is merge branch dev. Is there anything?
Is that okay?
This is one extra.
In doing this, it's asking you for a commit message, which is written out already, but it's opened up this editing software. In this case, it's called Vim. I don't want to get into the details of this, but all you need to know is if you press shift colon X and enter, then that will save that commit. When it comes to the exercise, I can help out with this. I can remind you of that.
And then it tells you, so it's done this auto merging of zoo, it's merged it in.
And so, now, if we look at the...

Log, look at all the commits.
Um...
Maybe you can guess what we're going to see.
So, we are on the main branch.
And this main, the top commit is this merge commit that we've just done. So this is not actually making changes that we've made. It's just saying I did, Git did a merge.
And now we have this who's hungry function, which is the commit that we made on the main branch.
anyway, but this extra one, this git animal is now, which it's saying is, you know, where the dev branch is at. This commit now exists in this main branch as well.
And then we can see that we have both this.
Where's my animal function?
Am I going mad?
Right there.
Oh, yes, there it is. A lag. Yes, we have the animal function and the listing hungry function.
Okay, so we're going to have another, oh yeah, here we go. So explaining to, sorry, visually what I just did. So I switched to the main branch.
to make sure we're on the branch that we want to merge into. And I did the command git merge dev. Apologies for that, it's named differently in the slides.
And what that did was a few things. First of all, it took all the commits from the branch. In this case, it is just one commit. And it put it into the main branch. The who's hungry was there already. That was part of that branch. And it added this extra merge commit branch.
on the end.
Again, I say, you know, not to worry too much about these details and thinking, oh, this is a bit confusing that it does this.
The key thing is the possibility of this idea that you can make changes separately and then when you're ready, push them into the branch.
As we go forward, why this is useful hopefully will become more obvious. I'm also conscious of time. We've got 15 minutes. So
This is the last exercise, and again, it is basically to repeat what I just did. The commands are here, so if you want to go back onto code space.
and have a go at doing that. Have a go at creating a new branch, switching onto that branch. Exercise 2 file has the instructions there. I'll leave them up on the screen.

So switch to the new branch, make some changes, doesn't matter what the changes are. Switch back to the, sorry, that should say main, switch back to the main branch. And.
Make a change.
Or you can use the git branch to see your branches.
Have a this is to practise switching between branches, make a change on the main branch, and then...
merge them in, then use git merge. So I'm going to edit that because...
It's too late, isn't it? Sorry.
Where it says master, that should say main. Hopefully that's clear.
And let's say, let's just say 5 minutes having a go at that.
Five or so minutes.
Don't worry if you don't get through it all. You'll have access to this afterwards so you can have a go after. And then.
For the last.
Five or 10 minutes, so just explain a bit about.
Remote repositories.
OK, and again.
Ask me any questions.

Tom Keel 1:19:22
Yeah, until half 12.
Um, this is fun for lunch.

Amulya Chevuturi 1:20:35
Simon.

Simon Stanley 1:20:38
Yeah, that's a good question and a good point to clarify. So Mayuran has asked, how do you add the add animal function? Is the code git add animal? So when you do git add, you add the name of the file, not
The function.
So, and a good way to see that.
Where's the add animal?
I'm just gonna type some things, add animal.

If you use the git status.
it shows you what's changed and it shows you it's the file that's changed. So it's a really good point to clarify is
You add the whole file. It's like saving the whole file, if that makes sense. So, and then you use git diff to look at what the changes within that file are, and then you see that, oh, it's the add animal thing that's been changed.
I hope that answers your question.
So git add zoo is the answer. git add zoo.py
And if, if anyone...
gets to the point of the merge.
Just a reminder.
That.
Let me just do this.
Ignore my change.
You get a screen.
Ohh.
Oh, there we go. Oh, sorry, it's slightly different here. When you get to the, if you get to the merge thing, it will bring a window up like this.
with a message already in there, I guess you just click the commit button.
So that should be a bit easier, actually.
Okay, I'm conscious of time, so I will leave it there, if that's okay. And as I say, you can complete these exercises.
Later on, if you want to go through them.
Um...
So just to the last few slides.
These are commands that we've just gone through. These slides will all be available for you to look through. So just to talk about remote repositories. So we've seen how Git allows us to develop files.
in isolated ways on separate branches and merge those changes together.
But we've just been doing this on our own, on our own computer. And in a lot of cases, it's not as necessary to work like that, to create branches and then merge them back in if you're just working on your own. So this branching and merging
It's most useful.
when you're working with multiple people and as a team you are developing code together. So it provides a way for multiple people to develop a single repository in

parallel.
So, in order for people to share the code they're working on, we have this centralised version of that repository hosted in the cloud, and we call these remote repositories. And so this is exactly what GitHub is. We create a centralised version of the repository that everyone can access. These are called remote repositories. And again, there are many out there. There are others, Codeberg and the likes. But the most famous
is GitHub and that's what we've been using.
So...
We're not going to practise this today. This is just to kind of so you understand how it works and what you can do with it. But when we're interacting with a remote repository, there are three main commands. First of all, git clone
This downloads a repository from GitHub. GitHub will give you a URL to use and it will download it onto your computer. And that is a lot like git init. If you git init is when you start on your computer from scratch, but often
You will start from an already existing repository in the remote in GitHub in a remote repository, and you will use Git clone to pull that down, and it will have all the commits and branches that are available straight away, so when you start actually doing your development.
You then make changes on your computer, you've pulled it down, you make some changes, create a branch, whatever you need to do. And then you want to...
Push those changes, so git push is the command that then...
pushes that the stuff that you've done back up to the remote repository. And then so other people can use the git pull to pull your changes back down onto their computer. So everyone can kind of keep up to date with what the latest developments are.
Um...
Okay.
And hopefully that's just clear that that's possible. And so this is where branching becomes very useful because you will have a remote repository which will contain the main branch.
you would pull that down. You would create a branch where you would do your development. You would then push that back up and that your new branch will go back up into the remote repository. And it's up there within the remote repository that you can do the merge. And when you do the merge up here, it can then get

reviewed by other people to check that everything looks okay. And so as you can imagine, if we've got bigger code repositories, people want to check that what is going into the main one is working and safe. There's lots of processes for reviewing branches and merging branches.
So that's where it makes more sense to.
work on your own branch. So where other people work on their own branch, they all get merged together. If there are conflicts, they get resolved and people can review what's going in. So hopefully that
Spells out.
how this Git and GitHub are used to develop code in teams. I will make these slides available, but just to quickly go through, there are many, many other commands with Git, and it's all about having full control over the changes that you make.
So just to kind of briefly run through them, just to give an idea. So when we do the git add, we can use this patch, which means we can just, if we make loads of changes to one file, but we actually only want to commit certain sections of that file.
Patch allows us to, and it opens up an interactive session where we can pick individual changes in a file to be staged. Git commit amend is where we've made a commit, but actually we want to change it. We want to add more so we can make some more changes, do git commit amend, and it will
add those changes to the previous commit rather than creating a new one.
There is a note that there are some things you should not do.
if you've already pushed things to a remote repository, but that's fine. There's this command called rebase, which is where you really get a lot of control over how you move your commits around. So you can take individual commits from another branch and put them onto other branches. So rather than merging the entire thing, you can
Shift things around.
You don't normally need to do that kind of thing, but there are cases.
Git stash is a useful one. So if you have to, so if you're working on some changes, but before you finish committing, you need to switch to another branch because you've got to quickly fix something or there's a major bug. You're not allowed to switch a branch unless
all your changes have been fully committed. So git stash allows you to kind of temporarily save those, commit those changes without, you don't commit them, you just kind of stash your changes, which means you can then switch branches, make a

change, come back, and then you can use git stash approach.
apply to get back to the state you're in.
Um, get blame is used for...
You look at any file and for every single line it will list what the latest commit that changed that.
line was. So it's a way if you're trying to debug, it's a way to find out what the commit was that made the change and why the change is there. You can use this git diff to compare the differences between branches. You can git log compares the different commits between branches. You can search
through the log for strings. So if you're looking for a particular commit message, within a commit message, you're looking, you're trying to find something, you know, at some point I added the noise function, but I can't remember where you can use this to find that commit. Git grep allows you to, oops, okay, search the entire repository.
for a particular string so you can just find things that's very useful. And I'll leave it there. I will say on these slides which will be available.
I then there's a number of other slides after this which go through installing. They go through the examples that I've done in a bit more detail. So you're welcome to look through and hopefully
It can answer and clear up any questions.
Happy to take any questions if we got time, Amulya.

Amulya Chevuturi 1:32:12
Yeah, yeah, of course.

Simon Stanley 1:32:21
But, if not...
I hope that gives a flavour of what it's all about.

Amulya Chevuturi 1:32:31
To those who have questions, yeah, Mayuran has this question.

Simon Stanley 1:32:37
What are some of the example cases you have mentioned where Git is utilised for hydrological studies?

Simon Stanley 1:32:47

Well, I guess I can use my own example where I have worked on a data system for drawing in hydrological data. So we have sites around the UK which monitor soil moisture.
and all that data gets sent into our systems. And then so I have written a, with the help of others,
Lots, excuse me.
lots of code which processes that data. So it finds the raw data, it applies quality control, it applies infilling, and all of this is written in Python. And it's, you know, it's about 20 or so files.
Python files working together.
And it was all done using, with the help of Git, over a period of two years. And it's very useful when, because it's quite a lot of code, things do go wrong.
and having it.
version controlled like that means it's much easier to find bugs and why bugs were made, because sometimes things are in there for a good reason. And so it helps just to figure out what's going on.
Thanks very much.
I'd like to know how to connect with other researchers on GitHub working on open projects since you said it's a cloud remote space. Yes. So the first thing is to have a GitHub account.
Um...
Lots of.
Lots of the repositories on GitHub are open and I believe you can just search on GitHub.
So, Tom, you might know more about this, but you can...
Let's go to the GitHub homepage.
To be honest, the best way is probably to Google things. If someone has written a GitHub repository with code out there, then it should show up in Google. I mean, I can point to the...
our GitHub space, there's lots of repositories in here.
We've got 301 repositories. I couldn't tell you what they all do, but lots of different models and hydrological things. But not all projects are public. Some are private, which means you need to be granted access.

Oh, thanks, Tom. Yeah. It's also used to prepare the materials for this very summer school. Yeah, that's right. So we didn't lose any code. Lots of us could all work on bits together without overwriting each other's work.
I'm going to stay on for the tech session this next half hour. I don't know if we, I mean, I'm aware we've been going an hour and a half, so do we want a little break or anything? Yes, okay.

Amulya Chevuturi 1:36:31
So everyone will be starting again in 4 minutes, let's call it 10 past 12, according to at least British time. So you have 4 to 5 minutes to take a break and come back. And then we'll have a 20-minute tech session, just testing.
So, feel free to just jump off.
Or you can stay and ask Simon a little few more questions, whatever works.

Gianni Vesuviano stopped transcription