Intro to Git Transcript

10 July 2025, 09:21am

Samantha Rees started transcription

Simon Stanley 0:17

Profiles shortly in, in and what we're going to do, we're potentially going to build a programmatic zoo where we have this concept of an animal and we can. The animal can do some things and then we have a concept of a zoo and then we put the animals in the zoo and and that kind of business.

So what I'm going to do to start with is this is so git is not involved at this point. This is these are just normal files and so like in a traditional sense, if I was to make some changes. So here's a function that is the animal's going to make a noise.

You'll see it has this noise attribute, so this is simply going to be print.

Noise.

So when this function runs, it prints whatever that is again. Don't don't worry about the actual coding, so at this point I'm just going to save it and carry on doing my development. So on the next function I'm this. This feed function is going to set the hunger attribute which.

Is hungry starts as true. It's going to set it to false. Well, I'm going to start doing that and I'm going to say, oh, is legs is true. And then as I'm sure you spotted, I've just made a mistake. So I will save it.

And obviously this is an extremely simple example. All I need to do is just realise I've made a mistake and change it. But if this function was a bit more complicated then there's just there's more chance of me having to.

Making mistakes in terms of undoing it or I've made other changes afterwards and I don't spot this so.

It's in situations like this. This is kind of a very simple example where having those each chains tracked and having to being able to see where where each change is made. You can have more control over undoing mistakes or seeing where mistakes are made.

So in this example, so for now what I'm actually just going to undo it all and I'm just going to save each time all the changes. It can't see kind of do it more visually save so back to the back to scratch. So now what I'm going to do is I'm going to set this

directory, this zoo exercise directory as a.
Git repository and this is the very first command that we're going to learn is let me make this a bit bigger as well.
Oops.
How do I do it?
Oh yeah.
That's right. Sorry, I should have done this before.
They're out there.
So also I'm not sure how familiar you are with command line.
Tools, but so this is a command line tool and it's basically it's a. It's a directory explorer just like any kind of Windows Explorer you move around, but you do it through commands and this is where.
We do all our get commands, so at the moment this shows me the directory that I'm in and I'm in this zoo exercise directory which is this one. If I do this command LS, it shows me the files that are in here. So the first command we're going to learn is git. In it and what this does, it converts the current directory into a git repository, and as you'll see, nothing really changes. But under the hood, what's now happened is that Git is tracking any changes.
This repository.
Umm.
The next.
Status.

Samantha Rees 4:32
Simon, you throw. Sorry. You froze a little bit there, sorry.

Simon Stanley 4:34
Oh.
Am I back?
OK. Thanks Sammy. So Git status, once we have initialised the git repository, git status shows us the status of that repository. So if I press enter and what this is telling me is that.

Samantha Rees 4:41
Yeah.

Simon Stanley 4:54

I'm on a branch called Master. We'll we'll talk about branches in a bit. There are no commits and we'll talk about commits shortly, and I have these untracked files. So what it's saying is we have this repository, but at the moment the files that it can see in there are not being tracked, which means the changes that we make.
Won't affect git. Git won't recognise them, so the very first thing we do is we add these files.
To into the repository.
Now these all this is you do this once at the beginning, so.
Don't worry too much, but you'll see a lot of git adding and git committing, so I'm going to do a git status again. You I do get status almost every other command just to see what's happening, and again still on the master branch, and it's saying there's these changes ready to be committed, which is.
I've staged these files and by adding them so I now do a git commit and then minus M is to add a message.
And I'm going to call it the initial commit.
I do have. I'm going to go through kind of more visual.
A visual example of of everything I'm doing in a second, so hopefully it'll make more sense, so I do another git status.
And now it's telling me that I'm on this branch and there's nothing to commit. So I basically now set up this directory to be any changes to be tracked. So what I'm going to do is make the same changes that I've just done.
But this time I'm going to commit each change. Now a commit is basically git's version of a save.
And we'll see how it handles them. So this first function make noise, it's going to print. Oops.
Left dot noise. I'm going to save that file.
And now when I do a get status.
Git is telling me that recognise that there are changes which are ready for which which have not been committed, and that's this. So saying this file has been modified this animal file.
Another very useful command is get diff and get. Diff tells us these differences. It's saying what has been changed but not committed. And when I do that it shows me that where it used to say pass, it now says.

Print noise.
So what I'm gonna do is I'm gonna commit these changes.
So it's the same as before. I'll do another kit status.
I'm going to get add this file copy paste.
And that adds. It adds these changes into this staging area. I will talk about the staging area a bit more, but for now it's it's an intermediate step that's necessary before committing. So when I do a get status it's saying.
These changes are ready now to be committed, so I do a git.
Commit minus M to add a message. So every time we commit something we want to add a description of what that change is doing. So we're adding.
Make noise function.
OK, So what we've effectively done is we've now saved this change into the Git repository and this is now tracked forever. So if I do a git status again, it's saying.
On this branch, there's nothing to commit. It's everything's good. So.
When we want to look at the changes we've made, the next command and I've written all these commands out. I'm sorry. I'm. I'm giving you a lot of commands straight away, but hopefully we'll. I've laid them out clearly on the slide. Get log.
Shows us a list of all the commits that we've made, so so far we've made two commits. This initial commit, where we added the files and then this one we've just done make noise function and this chain of commits.
This is part of our branch.
Umm.
Now what? I'm just going to add another commit just quickly. Well, what I'm going to do is I'm going to add this feed function now.
So dot legs equals true and as you've probably spotted, I've done it wrong.
But hopefully it demonstrates.
Yeah.
Yeah.
So I'm going to add this change. I'll do a get dif and we'll see the change that I've made. It's always good to check what you're about to change so you know you haven't done it wrong. In this case, I've been lazy going to add.
The file.
And please do ask any questions at any point. Interrupt me if if something's not making sense, I'm then going to get committed.
Add feed function.

That's now committed. I'm going to do get log.
And we'll see. This has been added on to the end of it.
OK, well, I'm going to keep going because I think, you know, my development's going very well, so I'm going to do the next function, add a year. So this one is self dot age, which is this attribute here.
Plus equals one, so I'm just adding a year to the age of the animal.
Yeah.
And once again do the same things get add.
It commit.
At age function.
OK, so now at this point.
I'll show you the get log.
What Git has the power to do is we've got this list of commits, and you can interact with these.
In lots of different ways without ever losing work. So if for example, I want to see what the state of the directory was before I'd done any of this, this each commit comes with this huge number, which is a.
A unique reference number. It's the git commit ID or the commit ID. So Git has this command called Checkout, which allows you to.
Go to the state of any given commit. So if I do Git check out and then copy and paste this big number, if you watch the code here, you should see that it all goes back to its initial state.
So.
None.
I've lost all the work that I've done it seems, but this is the magic of git is that I have not lost it. So if I want to move to another commit in this list.
Oopsie. Has it gone?
Let's go to the one where I added this. I can copy this git check out.
This commit.
And it should take me to the state.
Oops.
Yeah, it takes me to the state where I had done this commit, but not this one. So the point being is that you never lose any work and you can move around. Now if you remember, I was on a branch called the Master Branch.
The other COM the other way you can use check out.

Is if I say get check out master it will take me to the latest commit on that branch, so that just takes me back to where I was basically. So now you see all my work is is back here. So I'll do a git log again.
Now let's say at this point I realise I've my feed function was wrong. I'll quickly show you the the other command that's very useful for investigating commits is get show. And again you can use any of the commit references and it will show you the work that was done in that commit. So if I have a look at this one where I added the feed function and I investigate it and I see, oh OK, I did that wrong.
Oh, no, obviously I can correct it and add another commit, which is probably what I would do here because it's so simple. But if this was quite a big complicated change, what I might want to do is just revert the changes in these in this commit.
So Git has this command, Git revert and again don't worry about necessarily remembering this all at the same point. At this point. Hopefully it's just a demonstration of what it can do, and then we'll get a chance to practise it later.
So if I do Git revert this commit reference, then Git automatically creates a new commit, which is the opposite of what this is.
This is it asking me to if I want to change it's it's come up with this default message saying revert this one. I'm happy with that.
So it should revert it for me hopefully.
Still there.
Oh yeah, there we go. Just need to update and again if I look at the Git log.
You'll see it's added.
Added this extra commit on top of the branch, that sort of getting a bit longer and longer to see, but added this extra 1 which reverts this one down here. So now I could then add the correct function which is to.
Self dot hungry.
Oops. Spell that wrong.
It was false.
OK. And then one more time, I'm going to add this and then we can stop looking at code.
It commit add a message.
Correct the feed.
Function.
OK, so hopefully that gives you a flavour of just the general day-to-day working of Git and how we use it.

And so just to kind of re go through what we've just seen, but more visually. So hopefully it kind of makes a bit more sense because sometimes just command lines is quite hard to visualise. So we started off with this.
This directory, this zoo exercise directory, and it had these files within it. We did this Git init command which turned this directory into a track repository, but without these files in it.
We then used Git add to stage these files and git commit to get these files into the repository. So Git was now tracking them.
So then we by doing that we created this initial commit and these are not the same numbers, but each one has this kind of reference number. We then we made some changes to the file and we saved the file and we use a get status to show the status of the repository.
And that showed us that there were some changes that had been made and then we used Git diff to actually look at what those changes were.
So the.
The most common set of commands you use as you're doing this is you have your, your commit. You make some changes, you add those changes, get add, and then the files that the changes they're in.
And then Git commit and you use Git commit minus M to add a commit message. You every commit has to come with a message. If you use Git commit without the minus N, it will open up a default text editor in which you put a message in. So if. You have to you have to specify what the what the commit is doing.
And then by doing that, you've added this new commit to the branch.
So this is what a branch looks like and this was when we did get log. This was what we were looking at. So each one of these is a commit as we go along. Every commit has a commit message, sending it what it did and a commit reference number, which is how get knows which how to kind of.
Look at or move to or interact with each commit and this whole thing together is the is the master branch, and then the latest commit is the head of the branch. So when we, when we are when we say we're on a branch.
What that means is that we the commit that we're on is whatever the latest one on that branch is. So we use get log to see a list of all the commits on the branch that we're on, and then we use get show to inspect.
Any individual one and look at the changes that were made.
OK, so time for exercise one to try and put that into practise. Now I'm hoping this

works. I have tried it out with other people and I hope you're all OK to.
Have a go at doing some exercises. I will just put this in the chat.
There is a link here. Where's the chat? Here's the chat. If you all go to this page.
You should.
Have you should see a GitHub repository called Intro to get exercises. Let me share again. Sorry and I'll show you what you should see.
So you should see a page like this. Just shout if you don't. If you have any. If anyone's not seeing that.
So what we're going to do, should we move that there is we're going to click on this green button here where it says code on this drop down there is this tab called code Spaces.
If you click on the code spaces and then if you click on this plus button.
Hopefully this should open a new window.
And it takes a minute or so, but it should open up something very similar to what you've just seen. It might not be black, it might be white for you, but something very similar to what I was just working on where you have these files in the left.
And this command line at the bottom.
Yeah, just shout or leave a message in the chat if anyone's having any trouble viewing this.

Subhajit Ghosh 22:40
I.
Can I show it once more? How to get to the code special?

Simon Stanley 22:46
Yeah, no problem. So on this link there's this code green code button.

Subhajit Ghosh 22:47
Yeah.
OK.

Simon Stanley 22:55
If you click on the the down arrow and then on the code spaces tab, there is this plus and you should just click that plus.

And it should open a new window.
Has that worked?

Subhajit Ghosh 23:22
If I if I click it shows clone or something.
Clone HTTP.

Simon Stanley 23:28
Oh yeah, so this one. So if you click on the, there's two tabs, there's the local and the code spaces.
Well, there should be. Hopefully if you click on the code spaces.
Yeah.

Subhajit Ghosh 23:44
Problem is, I cannot find the code spaces, it's only showing me the local 1.

Simon Stanley 23:49
Ah.

Ashling Laffey 23:49
Sorry, I'll just hop in quickly when I clicked the link it sent me to Microsoft Edge, whereas I had signed up for GitHub on Google, so I just had to paste that link in the chat into Google and then the code space came up so that might work for you.

Simon Stanley 24:07
Ah, OK is it a browser?

Ashling Laffey 24:08
Just depending on what where you signed up on or whatever it was.

Simon Stanley 24:13
Oh, that's good to know which browser are you using?

Subhajit Ghosh 24:17
Using Google Chrome.

Simon Stanley 24:19
OK.
And do you have a GitHub account?

Subhajit Ghosh 24:26
Yes, I have created one.

Simon Stanley 24:30
And the are you and you've logged in.

Subhajit Ghosh 24:34
No, I haven't locked in, but I can.

Simon Stanley 24:37
Yeah, maybe it it may be that you need to log in.

Subhajit Ghosh 24:41
OK.

Simon Stanley 24:42
See if that.

Samantha Rees 24:46
Simon, there's a that sorry, there's a question in the chat. I think this my institute's IT policy has blocked my access. At this point. We had that, didn't we when we were testing.

Simon Stanley 24:50
Oh.

Samantha Rees 24:57
Did we?

Simon Stanley 24:57
Oh yeah, this was something Matt had.

Lloyd, Hywel 25:00
Yeah. So that's me. Yeah. Sorry. I work for local government based or. Yeah, Northern Irish Government ID system. So they tend to block everything as default.
I can I can speak to them, but it might take me a little bit to get this sorted.

Simon Stanley 25:22
Is that um, which browser are you using by chance?

Lloyd, Hywel 25:26
I'm on Chrome.

Simon Stanley 25:29
Yeah. OK. So this is. Is this something that you've had before? Is it?

Lloyd, Hywel 25:31
They.
Oh, sorry. No, I'm not. Yeah, I I tell I I'm. I'm not. I'm on Microsoft on Edge, I think. Yeah. But I expect it'll be on Chrome as well.

Simon Stanley 25:47
It might not be the same thing, but we did have there was an issue with Firefox browser blocking something, but this sounds like it could be something else.

Lloyd, Hywel 25:47
See.
And.
I think yeah, the message I guess is it's where's the message again? Yeah. When Web page blocked and it's blocked by the company policy. So I will probably need to speak to my IT department to get that sorted.

Simon Stanley 26:12

OK.

LH **Lloyd, Hywel** 26:16

Yeah.

Simon Stanley 26:17

OK, there is an alternative. Do you happen to have Git installed?

LH **Lloyd, Hywel** 26:27

No, no, I don't. I can check if that's again, I won't be able to install it unless it's on their list, which I can check now.

Simon Stanley 26:28

OK.

None.

OK, in which case?

LH **Lloyd, Hywel** 26:42

Boy otherwise.

Permission. Sorry, go.

Simon Stanley 26:48

OK. Yeah. If it is, everyone else is. Everyone else managed to get there OK? Or is anyone else stuck? I think I think if if the majority I'll just set everyone off if that's OK.

LH **Lloyd, Hywel** 27:02

Yeah.

Simon Stanley 27:04

So in in this directory there is a file called exercise one, and it's basically a list of the things that I've just done. So you have the same files the animal.

Lloyd, Hywel 27:05
Yeah.


Simon Stanley 27:21
Zoo and my zoo. But just just we're just looking at the animal file. You don't need to worry about doing the initial commit because that's already set up. So if you do a git log, you should see that there are already some commits in there.
But if you'd like to have a go, the instructions are are in exercise one, just have a go at doing using those commands we've seen get status get, diff. I'll put them up on the screen so you can see them and just see how you get on.
Um, let me leave these up.
I don't know.
So here's the list of the commands that might help. I'm not sure what the can I check also, so savvy Jits. I'm not sure if I'm saying your name correctly. Did you manage to get it working after logging in?

Subhajit Ghosh 28:20
Yes, yes it it's working.


Simon Stanley 28:21
OK, great. I'll bring it in.
Yeah, Lloyd, I'm not sure sorry. Is that your surname or first name?
Oh, sorry you're on mute.

Lloyd, Hywel 28:41
Thank you. Yeah, see me. Don't worry about it. I've I have got a vision that I'm trying to install at the moment and it seems to be going very slowly and that would be to install git.
Itself.


Simon Stanley 28:59
OK.

Lloyd, Hywel 29:02
Otherwise I will start something with my it.

Simon Stanley 29:02
Umm.
OK, that might be. That might be, yeah. Quick 'cause then it's a case of downloading things and.

Lloyd, Hywel 29:08
But.

Simon Stanley 29:15
Is that OK to sit this out for this one and?

Lloyd, Hywel 29:18
Yep, Yep.

Simon Stanley 29:19
Thank you. Thanks.

Lloyd, Hywel 29:23
I'm.

Simon Stanley 29:23
None.
So I will give everyone let's say.
5-10 minutes, 10 minutes to have a go. Don't worry about getting through everything. The most important thing is to familiarise yourself with making commits. If you can do a git ad and the Git commit, that's brilliant. If you get around to doing a revert, great. Having a look at the differences you've made git status and just have a play around and yeah, and just any questions you have.
So I haven't done any breakout rooms or anything. I think that probably make it more complicated. But yeah, leave questions in the chat or shout out.

Weeraphat Duangkhwan 30:55
I can exit on the git lock.


Simon Stanley 30:59
Oh, good question, you press Q.

Weeraphat Duangkhwan 31:05
Thank you. OK, I see. Thank you very much.


Simon Stanley 31:06
Yeah, sorry.
Yep, that's a very good question.
Umm.
So Shango's just asked, so we don't have don't have rights, access to the intricate repository. So you cannot push your changes. Yes, that's right. So at the moment we're not doing any pushing or pulling at the moment. We're just making changes locally.
You are welcome if you if you know how to.
Fork it, then you can, but I would say no. No, don't fork, because then you'd be creating. It'd be this. It's just a new remote repository to which you would then be pushing. So just for now, just work on your local.
Cool.
Branch in the code space if that makes sense.
Yeah, we're gonna touch on pushing and pulling a bit later.
I just give it a couple more minutes and then we'll move on to the next section.
Doing well for time I think thanks to Sammy.
Yeah.

Fatima, Khunsa 34:40
Yeah.


Simon Stanley 34:47
Welcome, Fatima.

OK, let's leave it there. I hope you've all managed to have a good go and that code space will remain there, so you can always come back and.

Fatima, Khunsa 34:59
Yeah.
Yeah.
Yes.
To.

Simon Stanley 35:12
Carry on later if you please.
So what we're going to talk about next is branching. So, so far everything we've done is being on one branch and in this case it's called the master branch, which is this kind of.
Main trunk of the thing. So in Git we can create new branches which contain their own chain of commits and what this allows us to do is. It means that we can make. Changes to the code in isolation and in safety. So if we have this mainstream of development and we want to add some changes or experiment with some changes or develop something without.
You know, kind of breaking what's being what's being done on the on the main. Branch. Then we can do these things in isolation. And as we've seen, we have the power to kind of move around with commit. So everything is stored.
So yes, this means that we can test an experiment to our hearts content.
So there are three key commands for branching, which I'll go through and then demonstrate. So Git branch this.

Fatima, Khunsa 36:41
Yeah.

Simon Stanley 36:44
On its own, this just lists all the branches that are in the repository. So for the example of of this image here, there are there are two branches, so it would list master and it would list Dev.
Git check out which we we saw earlier Git check out with a branch name. This moves us to the chosen branch. And remember when we move on to a branch, we move

the. It means we're taking the directory that we're working in into the state.
Of the latest commit on that branch. So Git check out Master takes us to this commit.
Git check out Dev takes us to this commit and we can switch between them.
And then the last one, git check out minus B and a branch name. This creates a new branch. I just found out yesterday. There is actually a new command which is.
Much better named, which is get switch and get switch switches between branches and creates new branches and I've only just learned this so it's just an FYI for me as well.
But.
Yes. So these are the, the, the three main ones, there's list the branches there are move to a branch and create a new branch.
And then so just to note, when you create a new branch, you actually you don't create a new commit, you just it. It starts in the same place of of where you were, where you branch from. So it's. It's only when you start making commits that the branch will grow independently.
From where it is.
So I think it's, oh, there we go. Yes. So you start making commits and they and they grow. So I'm going to do another demonstration.
Let's pop this one up.
So.
We're in the same spot as where we were. We'll have a quick look at the Git log.
This is our we're on this branch, master. It tells us and the the head of the branch.
That's our current position and it's on this commit. So what I'm going to do is create a new branch with Git.
Check out minus B and I'm going to call this dev zoo so I'm naming this is the name of the branch and this again we wanted to kind of make sense, it's because I'm going to develop.
This the this zoo file.
And that's done. So they switched to a new branch. So if I do a get log again, we see that we're in the same position. Nothing's changed, but now this head, this current position is that on it's got this 2 branch names, this dev, zoo and master. So these two branches are.
Currently contain the same commits.
The same list of commits.
So now I'm going to start doing some development and I'm going to develop this

zoo class and we have this add animal function which is basically to add an animal into the zoo and it's quite simple. We have this list ready to go.

Fatima, Khunsa 40:09
OK.

Simon Stanley 40:20
OK.
And we're going to append the animal that has been passed into the function.
And yes, as you can see, I'm keeping this all very simple. It's not about the code, but hopefully it makes sense. So I'm going to do my status.
I see that something has changed. That's good to know. I get dif to the check. What that change is? Yes, that's the one I did. That makes sense. I'm going to get add.
The sue.
And then get commit minus M with a message to say at.

Fatima, Khunsa 41:07
Yeah.
Yeah.

Simon Stanley 41:09
Animal.
Function.
OK, so now let's have a look at Git log again.

Fatima, Khunsa 41:17
Yeah.

Simon Stanley 41:22
And this time it's showing us that we have this extra commit on this branch that we're on and it's Dev Zoo branch and it's also showing that the master branch is back here. It's behind. So this is it's diverged from the master branch.

Fatima, Khunsa 41:22
Yeah.

Simon Stanley 41:39

And then so just to demonstrate, if I check out get check out which is to move to the other branch, I'm going to move to master. Oops.

Again, watching the the code, you'll see that that that work I've just done should disappear, so it's gone back to what it was and I can of course switch back if I want to get check out Dev Zoo.

Fatima, Khunsa 41:52

Yeah.

Simon Stanley 42:10

And it's back again.

So while I'm well, actually I'm going to, I'm going to go back to the master branch.

And I'm now going to make a change on this.

Branch on the master branch. I'm going to add a commit. So who's hungry? So this is going to.

For animal and.

Yeah.

It doesn't really matter if this works or not. Print and then more.

Is.

Hungry. So that's going to loop through the animals and it's going to print their is hungry status, which is honest. So it is hungry. Oh, it is hungry, obviously. I've got that wrong again.

Umm.

So yes, so added something to this function and again we're going to do the same thing.

I also I tell you what, it's a good time to demonstrate. I've noticed I've done this wrong and I'm going to correct it here.

Is hungry, so I've made a change to two different files, so I do a git status and it recognises they both had a change. I can do git diff and it lists.

Both those changes.

From the zoo file here and the animal file here. And again that's I use Q to get out of that to quit out. So now.

I'm going to add both of these changes to the same commit, so I'm going to get add

this change.
And this change.
Yes status get commit. I will say that this isn't best practise because normally you want a commit to involve a kind of single unit of change. But I've done two things in this commit I've corrected something.
Mistake and I've added this function, so obviously it's perfectly possible to do that and lots of people do. But what you really want as best practises every commit to be a single isolated change, which of course when developments more complicated can involve.
Um, lots of changes to different things, but they're all kind of working together to to achieve one thing. So.
Correct is agree and oops.
At.
What was it again?
Who's hungry?
OK, so I'll do a git log and you can see that the master branch has has extended and we've with because we've we've now diverged from the.
Both the Dev Zoo and the master branch were branched off this commit and they've gone off in different directions. Once the master has added this commit and the Dev Zoo has added another one, so we can't see it here.
So I'm going to leave it there at this point. So and then I'm going to show you what I've just done, but visually so you can see.
OK, so we have we first thing we did was we created a new branch with Git checkout minus B Dev Zoo and at that point the commits that we had this list of commits were part of both Master and Dev. They were they were the same thing.
We then added a new commit onto the dev branch, so we we were checked out onto the dev branch, we added a new commit and we added this animal function and the master branch remained unchanged. That was still back here.
So we then switched back to the master branch, Git check out master. We went back to here and then we added a new commit. This, who's hungry plus the fix. And so the branches have now diverged. They both have this common ancestor, but they've gone in different directions.
Yeah.
So at this point, and this is very common, so when you have multiple people working together on a project, they will be working on their own branch and they'll be

making these independent changes. So one of the most powerful things that get. Does is merging these changes together and so there is this command Git merge which we will look at and this allows us to develop code in isolation and safety. So once we're happy.
With our developments and we want to merge them back into the master branch, so we want them all onto this kind of single branch.
We do it with this with this command Git merge.
So there are two types of merging. The first one is called a yes.

Samantha Rees 48:00
I'm in. Sorry. There's a hand up I didn't know. Is there a question? That's OK.

Simon Stanley 48:03
Oh, sorry. Thank you.

SG

Subhajit Ghosh 48:07
Yeah, just a bit confused. So when we are branching, right, if I write a new function and merge it so the new piece of the code will go to the master branch.

Simon Stanley 48:25
Yes, exactly.

SG

Subhajit Ghosh 48:25
Or that function always has to be in the master branch. Then I can work on the development branch.

Simon Stanley 48:34
So if you.
Umm.
So the chain. So if you if you if you add that function on a branch and then merge it back into the the you you need to merge it back into master before it's on the master branch. You can create a new branch.
Off any point in a branch. Sorry, does that answer your question? Maybe. Maybe you need to ask it again.

Subhajit Ghosh 49:06
OK.
No, but basically maybe I have created a branch from master so now this is the duplicate copy right in the development branch. Then I wrote something new piece of code which is not in the master branch.


Simon Stanley 49:17
Yeah.
Yep.

Subhajit Ghosh 49:29
So if I commit, it will directly go to the master branch or that function previously has to be in the master branch. Then I it will only go.


Simon Stanley 49:29
Yeah.
It.
It will the the key thing which perhaps I didn't make clear actually, is when you do your commit. So if I go.
Check out Dev zoo.
And I I add I add a function, let's just say print.
Hello.
Kilo.
Save it when I do a get status. So just before I oops.
Oh.
Something funny now when I do a git status before I make my commit, it tells me which branch I'm on. So when I make that commit, it will the changes that I've added will only be committed onto this branch, so the master branch will.
Remain the same.

Subhajit Ghosh 50:46
OK. Yes.


Simon Stanley 50:47

Does that make sense?
Yeah. And then yes, so it's basically.
So that's where we're in this situation. We, we've we've added a commit on to this branch and this commit does not exist on the master branch. So that changed. The animal function doesn't exist here, and then you have this, who's hungry change, which doesn't exist on here. So there's you kind of got this.
Complex situation where there's two changes being made in isolation separate from each other, so when it comes to merging these changes together so they all are on one branch.
There's there's two situations. There's one where if, like this one on the left, the grey ones that say that's the master branch, we've created a new branch and we added some commits, but nothing else. No other changes were made to the master branch. So this is a Fast forward.
Merge and all we're doing is moving those commits onto the master branch, and it's as if they were made in the master branch originally, so there's no. It's very easy for Git to do that, it just adds those changes straight onto the master branch.
This branch will still exist and it will have a duplet, so these commits will be on both those branches, just like how the branch shares all these bottom commits, it now will share all these commits as well. They will be the same thing.
The other situation which we have found ourselves in with the diverging branches is we've made a commit on our new branch and also a commit on the master branch. So when we merge these changes together, this is where Git is clever about how it does this. If these changes are in complete isolation to each other. So, for example, what we did, we made a change to.
Where is it? We made a change to this function and then we made a change to this function so get can see that there's no problem, it just adds these changes and it adds these changes and.
It can work that out and I will demonstrate this in a second. And there's no problem. But if on one if on our development branch I made a change to this function.
Well, let's say this change and then on the master branch I made another change to the same function that was slightly different. Then you come into a situation where you have a conflict. You've have two people who have tried to change the same function and get will recognise this and present you with a an interface in order. Decide how to resolve those conflicts. I'm not actually going to go into that in this because of time.

There is I. I've got some examples later on that you can look through, but in terms of for the sake of time, I'm not going to get into that.
But yes, that's just to say that this is what this is, the main power of git. This kind of merging capability. So I will demonstrate that now quickly. So when we're doing merging you, you have to be on the.
On the branch that you want the changes to be pulled into. So I'm going to switch to the master branch.
Oh.
Yeah.
So this is a important thing. You can't switch branches if there are changes that haven't been committed, because if you change branch with uncommitted work then you lose work and the whole point of git is it stops this from happening. So this was a message saying your local changes.
Will be overwritten by the Chuck app, so you have to commit them or stash them before doing it. So I'm going to remove those changes by doing that.
So I do a get status.
I'm on nothing to commit clean branch. Now do a kit check out.
Master.
OK. And hopefully you'll see. So here's the function that I added, the who's hungry? In the master branch and I'm going to merge in Git merge Dev Zoo so Dev Zoo is the one that contains the change I made to the feed animals. So hopefully when I do this. It creates it's again. It brings up this editor to say I'm going to add a new commit. A merge commit. Are you happy with this message? Yes, I'm happy with that message. So it merges it in and.
He's a bit of a kick.
Oh, was it the ad? Oh, wait, it was this one, wasn't it? Was it? Yes. This. No. Sorry. This was the function. I did. Sorry. Yeah, the ad animal. So when I do a git log.
It shows me that this I'm on the master branch and what it's done is is added the add animal commit from the Dev Zoo and it's also still got that this is the one I did on the branch where I corrected on the master branch and it's also added this.
This extra merge commit. Now, normally this merge commit is. It doesn't actually change anything, but when there's a conflict, then the changes you make in that conflict go into this. But again, at this stage I'm not going to go into this for the sake of time.
Like I said, this git is quite a lot to it, so to this is a real crash course. To do it all in an

hour. I'm hoping this kind of gives you a flavour of what it can do, but I don't expect you to be experts at it by the end of this.
So again to to show you visually what I've just done there. So we had these two branches with the different functions. I did a git check out onto the master branch, so then I was here.
And I wanted to merge that all the changes from the dev branch in this instance it was just one commitment. But remember this branch could have as many commits as you want and get merged. Would merge them all in, so you'd get all the commits from it.
Umm.
And then it created that. Merge it on the master branch. It added this commit, which we did into the master branch. We had this one that was there already, and then it added this additional merge commit.
Which doesn't make any changes, but it's just tells you that a merge has happened.
OK, so we've got about 10 minutes left, so if you'd like to go back onto your code space.
And.
In the exercise 2 file, there should be a list of. Again what I've just done with the commands that should are there to help you and just have a go.
Have a create a new branch. You can call the branch whatever you like. Make some changes on that branch and then go back to the master branch and merge them in. You can, if you want, you can make changes on the master branch first.
Um, let's see.
See how you get on and again, just just shout out any questions you have.
And and so I haven't been really keeping an eye on the chat, but thank you, Amelia here and Tom and Sammy.
Yes. So just let me know any questions and I will leave the commands up for you.
And then yeah, we'll just do. We'll do 5 minutes or so. This will be available afterwards for you to carry on with.
One thing that might come up actually is when you when you get to doing a merge, it opens up this text editor.
Umm.
And I should demonstrate, Ashley.
So when you get when you get the text editor come up that asks you to write a message.

If you to to save and close it.
You do shift colon X, but I will. I'll write that down.

Amulya Chevuturi 1:00:44
Simon, did you want to share your code space screen because currently we just are seeing your PowerPoint screen.

Simon Stanley 1:00:46
Yes.
Yeah. So these that hopefully everyone's got their code space up and these are the commands that I should be useful.

Amulya Chevuturi 1:01:05
I think there was a question about, I think they want to know about the shift command that you were just talking about.

Simon Stanley 1:01:11
Oh yeah. Sorry I'm. I'm OK. I'm typing it in and then I will.
And then I will demonstrate.
Sorry, umm.
Sorry, ignore the comma there.
So is it?
So I can I can get to the same editor by making a change print.
Let's.
So if I do do get commit without the message, it opens up this same. Oops.
At first.
It commit.
Opens up this editor you might see. So what I'm pressing is shift colon.
Oh, I need to type a message. Sorry, you won't have this problem.
Yeah.
Test it.
I'm forgetting my commands now myself, so you'll get to this editor. You press shift, then colon and you see down at the bottom of the screen that one and then X and then press enter and that will save and close it.
This this is. I'm sorry, that's a that's a neatly little complication.

Thanks.
You can, when you're when you're working, you can change your default editor, so it's a bit easier to negotiate these things.
But yeah, let me know if you get stuck on that.
Yeah.
Given there is 5 minutes left, what I am going to do is just run through the last few slides. If that's OK. Yeah, you're very welcome to come back to it afterwards.
Yeah.

Samantha Rees 1:04:09
I mean, there is a sorry, there's a question in the chat. Do you have any thoughts slash experience on integrating copilot in the GitHub slash vs code environment?

Simon Stanley 1:04:09
And just talk about.
Oh.
Yeah, I've recently started using copilot, so for those that don't know, copilot is an AI code developer built into VS code that helps you. It helps predict.
What it thinks you're trying to do with the code, so it comes up with lots of suggestions. Sometimes it can be a bit annoying because it's suggesting things you don't have, but I it is very good understanding the code that you're writing.
Umm.
So I I do quite like it. I have to say, Oh yeah, and thanks.
Thanks, Matt. Yeah, as he says, if he uses the minus M for the commit.
It will do it.
OK. So yeah, just to run through the the final section on remote repositories and I'm going going to sort of briefly go over this. So we've seen how Git allows us to develop files in isolated ways using branches and merging the changes together.
But so far we've just done this on our own. So it sort of seems a bit pointless for us to create a branch, then merge it back into the same branch that we're the only ones working on it. So the whole reason this exists is so development can really jump to the next level when we have.
Lots of people working on the same repository, so we can they can work in parallel and this is only possible when you have a remote repository. So.
This is where you you you have a centralised version of the code you're working on

and everyone takes copies of that code. They create their branches, they do their work and then they.
Merge those those changes back into the remote copy, so the most the most famous version of remote repository a host for remote repositories is is GitHub, and that's what we've been looking at. The code there is.
The central copy of what I have on my computer, so I could push and pull changes to and from it in the code space. You're actually just working within the remote repository, so it's kind of a special feature just for.
This kind of practise.
So the main commands that we use for interacting with remote repositories are a Git clone, which is to download a repository onto your machine. So you can I think we actually saw it.
If we go back to here in this code, one in this local tab, you have these three options and these are basically URLs or keys that are pointing you to this remote repository. So you can use a git clip Git clone command with.
These URLs to download this code and as I say I don't have quite enough time to actually demonstrate this, but the point is it's quite easy. It's very possible. So then once you start, you have your copy and you start making changes and doing development.
You can push those changes to the remote repository and also as other people, push their changes. You can pull them down so you always have the latest, so there's this pushing and pulling interaction and then these commands. They just allow everyone to stay up to date and for things to get developed.
Synchronicity synchronically. I have a whole list of other commands which I don't want to overcomplicate things, but just to kind of briefly go through to demonstrate the more kind of detail and power that Git has. So.
You can when when we made changes. If you remember we were adding the whole file so if we had made lots of changes within a file but we only want to commit individual parts of them. There is this patch command where you can go through and select individual changes within a file.
There's this get commit amend which is where you you've made your commit, but actually you you've missed something out and you want to add more to that commit. So instead of a separate one, you want to extend the previous commit. You can do that.
These kind of things are not normally recommended because once you've pushed to

commit then you shouldn't change it or you you you can't. Really we can, but you shouldn't change. It causes it causes issues, so it's easier just to make a new commit and there's nothing wrong with that.
Get rebase is quite a powerful one. This is where again, you can kind of squash commits into individual commits, but not necessarily recommended, but you can also where we merged an entire branch into another branch, get.
Dbase allows you to take individual commits and move them around to different branches, so it's kind of Special Situations when you need to do this, but the point being it has the power to completely control your your branch, kind of.
Architecture Git stash is very useful. I don't know if you saw, but when I tried to switch branches, switch branches without committing my changes, Git didn't let me. So if there's a situation where you you're in the middle, you haven't finished the work for the commit, but you need to switch.
And another branch to fix something. Then you can stash your changes so it just creates this kind of local commit where everything you've done disappears. So then you can switch branches, but then you have this git stash apply which brings those changes back, so you can carry on working from where you were git blame.
Where you can look at a file and it will show you every single commit that's been made on every single line. So you can find out it's not a particularly nice word for it, but you can basically find out who's done what in the file, so you can. It helps with debugging and you can see what changes were made when.
You can look at the differences between branches. So exactly what what?
Where the differences are you can look at the different commits, so the Git diff branch dot dot branch shows you the actual changes, the actual differences Git log branch dot dot dot branch shows you the the different commits on the different. Branches.
There is this functionality for searching for a string within your entire history, so if you want, there's a particular function that you want to know what commits have changed. This function then get log minus S and then a search string will show you all the commits where.
Your your search string has been changed or found and then also get grep is a very good one where it will you're just searching for where something is in your directory. You want to find out where a.
Function name is get. Grep will very quickly show you in the current state. Where will that exists.

I ended there. I will say so. These slides I assume are going to be shared. I probably on me to actually share them. There's a whole after this. I've gone into loads more detail about each of the commands with examples.
Lots to look at so you can go through. I talk a bit more about staging, which is this intermediate bit. I talk a bit more about conflicts, so there's lots more resource here for you to use. More about branching.
Umm.
But unfortunately don't have time to go through it here, but I hope that's a kind of useful resource for you, so I will end it there and say thank you very much.

Samantha Rees 1:12:35
Thank you, Simon. That was really brilliant. I now know what GitHub is, so I feel like I've learned something here. Was that a question? A hand up there or?

Simon Stanley 1:12:41
Yeah.
It.

Samantha Rees 1:12:47
Brilliant. OK.

Lloyd, Hywel 1:12:48
Yeah.

Samantha Rees 1:12:49
Thank you so.

Lloyd, Hywel 1:12:49
Yeah. No, it wasn't a question. So it was me.

Samantha Rees 1:12:52
No, that's OK. Yeah, I do that all the time where you're trying to clap and you. Yeah, I will get these slides off of Simon, and I will send them around. And I will also send the recording around. I'll try and get it done by the end of today.
And mulia, I see you have taken your camera off. So now we're just going to have a

look at any technical issues and sort of go through the requirements. So I will stop talking now and and hand over.

Amulya Chevuturi 1:13:21
Hi everyone. First of all just answer Simon's question about who's hungry. I'm hungry. Very hungry. So let's get this done very quickly so that all of us can go back and have some lunch break or to whoever is in a different time zone.

Matt Dalle Piagge 1:13:29
I'm hungry.
Search.

Amulya Chevuturi 1:13:37
Just a break, Simon, would you be OK to stop sharing? Thank you. So we'll start off with a few introductions. I am a senior hydroclimate scientist at UK Centre for Ecology and Hydrology.
And I'll be running some one of the sessions in the next week. Today we are just going to do some testing so that we can know that you guys are able to work on the workshops with us because these are mostly hands on sessions.
I mostly work on hydroclimate variability in current and future climate and also look at forecasting of such hydroclimate variabilities. Looking at especially extremes like floods and routes. I will pass on to Matt and then Tom to introduce themselves and then we'll get on to the session.
Itself.

Matt Dalle Piagge 1:14:31
Cool. Yeah, I'm Matt. Yeah. Always hungry. Particularly particularly right now. Yeah. I'm also working at the UK Centre for Ecology and Hydrology. My official job title is research software engineer, which.
Is sort of helping helping develop good practises and good software practises, but I also do a fair share of data wrangling on the side as well so I feel like that's that's often as an official job title. I tend to take on. Yeah, I'll be going through.
One of the sessions with you next week, there's some different types of data you're likely to encounter in the hydrological and and wider, wider science fields.
Yeah. And I think that probably do for now. So that we can all practically have lunch.

Amulya Chevuturi 1:15:19
Yes, Tom.

Tom Keel 1:15:19
Yes.
Yes. Hi everyone. I'm Tom. So I'm also at the UK Centre for Ecology and Hydrology. My title is hydrological data analysts and what I do day-to-day is sort of very much. Well, I've only just joined, but I'm looking mostly at sort of rainfall data and how we sort of quality control that my expertise is a bit more with sort of data science for Python. And my background is a bit more environmental, data science broadly. I've recently finished PhD.
Before joining and yeah, I'm looking forward to helping you guys out.

Amulya Chevuturi 1:16:03
Great. So thank you, Tom. Thank you, Matt. So the next sessions, the workshops next week will mostly all be running on Google Collabs. We have a GitHub repository set up which to be fair is currently under development and will be developed by the time.
And we run this workshop sessions, but because we are running the workshop sessions live with you and hands on, we will continue to develop the GitHub repository because we'll take feedback from you and maybe add a new thing. So by the end of the next week, we will give you.
Like it will be finalised. I mean you'll get the link today itself, but please do remember that this is currently under development so don't worry if certain links are not working when the workshops will be work when you are working on that workshop on the day of that link will work.
But obviously it will be continuously under development and we will give like the final version will be ready by the end of the week. OK. So everyone, if you could, I have put a link of Google, Google Research collab ones.
In the link, if you guys could open it all U and could, could you tell me if anyone is having problems? Do you all have a Google account? Can you log in? Are there any issues?
If anyone has any issues, please speak up.

Lloyd, Hywel 1:17:27
Yeah.

Amulya Chevuturi 1:17:33
So I'll share my screen, hopefully all of you have something like this showing up on your screen. If someone doesn't have like this, please do let me know. You should have logged in like this.
A login should have been there for your Google account, so something like this.
OK.
I will give it another minute just for anyone who doesn't have it to speak up.
Otherwise, then we'll move on to the next step.
OK.
I am assuming all of you guys can do this, so now if we go to file and we open.
Open notebook the second option. So if you all do the open notebook you get something. You get like a new page that opens up and you go to the GitHub option GitHub tab there.
And there you can post the GitHub repository link that I've posted in the chat like so and then hit the search button. The magnifying glass button.
Is everyone able to see that?
Yes. OK, so here I would suggest please don't open the workshop one. I am going to run the workshop two. So let's just open the Workshop 2 for now. Because if something breaks, I'm happy to fix it. So if you guys click on workshop two I so it is the title is.

Lloyd, Hywel 1:19:03
Yeah.

Amulya Chevuturi 1:19:23
Worksho 2 underscore remote underscore data underscore access Dot I nyb if you click on that you will open up.
The notebook that will be used for the workshop two training. But I would say please don't make changes as this one because this is not your copy because we want to run these. We want to make changes on our own. What you should do is save this onto.

To your Google Drive. So every session that you run, whichever notebook you open, you have to follow the next couple of commands to be able to store that repository. Sorry, the notebook onto your drive. So what you have to do is again once the notebook is open, you go into file.
You go into save a copy in drive. If you see that if you do that, it will open up a copy for you.
In a new tab, as you see, called and then it will be called copy of Workshop two and you can rename it whatever you want, but this is what you should be working on. I would suggest you should close the old copy.
I'm sorry. Close the original so that you don't make changes in that because it will not get saved. It's not working on it okay it's because of my window. Yeah, I've closed it. So I've closed the old version.
And now I have the copy saved up. It will be saved on my drive it.
If you see, you can open it up on your drive, locate in drive. If you go to file, you can do locate in drive and then it will just take you to your drive and locate where it is stored on your.
Drive.
So you can go back to it whenever next you want to open, so it's saved on your drive. Any changes you make in dint any notes meet you make in it, that'll be fine. Thank you, Matt. As you said, yes, we will go through it briefly.
At the start of our sessions to make sure everyone has copies, so I will stop now. Does anyone have issues? Did anyone have problems opening this? Copying this anything or if I have missed anything Matt and Tom please.

LH **Lloyd, Hywel** 1:21:54
But.

 **Amulya Chevuturi** 1:21:57
Pop.
By by bin yes.

LH **Lloyd, Hywel** 1:22:04
I.
Just to say I I had an issue. It's all right again. Yeah, I think again they might be working on a work computer. We're not supposed to use Google Drive. It was

working to open things and check to start with, but actually creating a copy of my own is not working.

Amulya Chevuturi 1:22:09
Mm hmm.
Mm hmm.
Right.
Yeah.

Lloyd, Hywel 1:22:26
So you have to.

Amulya Chevuturi 1:22:26
OK.
Can we perhaps create like a?
Local do you think Matt or Tom?

Matt Dalle Piagge 1:22:42
I'm not sure I need to play around with it.

Amulya Chevuturi 1:22:44
Yeah, maybe what we'll do is we we'll stay back with you and then ask you to share the screen once. Let's let us resolve all the other people's issues. Are there any other issues?

Tom Keel 1:22:56
Hmm.

Lloyd, Hywel 1:22:58
Yeah, yeah.

Tom Keel 1:22:58
I mean, yeah, I mean, I was just just to say if you if you don't have access to Google Drive, it would still work. It's just you won't be able to save anything. So if you close that tab again that's lost.

Amulya Chevuturi 1:23:10
Hmm.

Lloyd, Hywel 1:23:12
This.

Tom Keel 1:23:14
So so you can sort of follow along and do things, but you won't have your local copy. And then that could lead to a problem if you yeah, laptop goes off or something like that.

Lloyd, Hywel 1:23:14
Yeah.
Play song.
Yeah, I I yeah. I also don't want to destroy the the original version either, so it'd be nice to have.

Tom Keel 1:23:31
Oh, it's yeah, it shouldn't be the original version. Sorry. It is a copy already. It's just a copy that's loaded to Google servers and not your Google Drive. Yeah.

Amulya Chevuturi 1:23:31
No, no, no, I it doesn't.

Lloyd, Hywel 1:23:34
OK, OK, that's fine.
Yeah. OK.
Yeah, OK. Yeah.

Amulya Chevuturi 1:23:47
Yeah.
I'm.
Can I? I'll give another minute for anyone to say anything, but otherwise Sammy from our side, I think we're done. Right, Tom and Matt.

Lloyd, Hywel 1:23:56
Yeah.


Amulya Chevuturi 1:24:06
OK. So Sammy, did you have anything else and?


Samantha Rees 1:24:09
My last thing was just a reminder, if you wanted to submit a poster for the poster session on Monday afternoon, please send it to me by the end of today. I'm going to be uploading them onto the virtual platform tomorrow, so if you could get them to me today, that would be.
Brilliant. I need them in a PDF or a slide, like a slide format please.

Lloyd, Hywel 1:24:34
Yeah.


Amulya Chevuturi 1:24:35
OK. Just one last thing. Again if anyone has any issues please to pop please to pipe in now or post to the chat but if you don't want to speak up now or forget how to do this in the GitHub repository read me. There is a full.
Step by step instructions on how to do this yourself. So even if you have forgotten what I have said, please do go ahead and follow this and you will be able to do it yourself too. But yeah for now I would say don't delve into the GitHub repository too much because we, as I said, it's currently under development so.
There is no need to read ahead as they would say in the class and we will go through all the steps 1 by 1.
I'm OK with people logging off if they have to. I think, Sammy, you're also OK with that. And I will if you want to stay back. Happy to chat with you.


Samantha Rees 1:25:38
I'll stop the recording now.

Lloyd, Hywel 1:25:38
Great. Thank you.

Amulya Chevuturi 1:25:42

Have a great lunch, guys.

Samantha Rees stopped transcription