Get Git.

An introduction to Git.

Firstly.

Thank you for coming.

Secondly.

The plan.

1. This preamble.
2. On with some learning by doing.

Thirdly.

Some quick rules.

1. This is about learning.
2. This is a safe space: It's OK to say you do not understand.
3. We will learn in this session by doing.
4. We will learn for future sessions from your feedback.
5. If at any point you feel you know this, you do not have to stay.

Fourthly.

An admission.

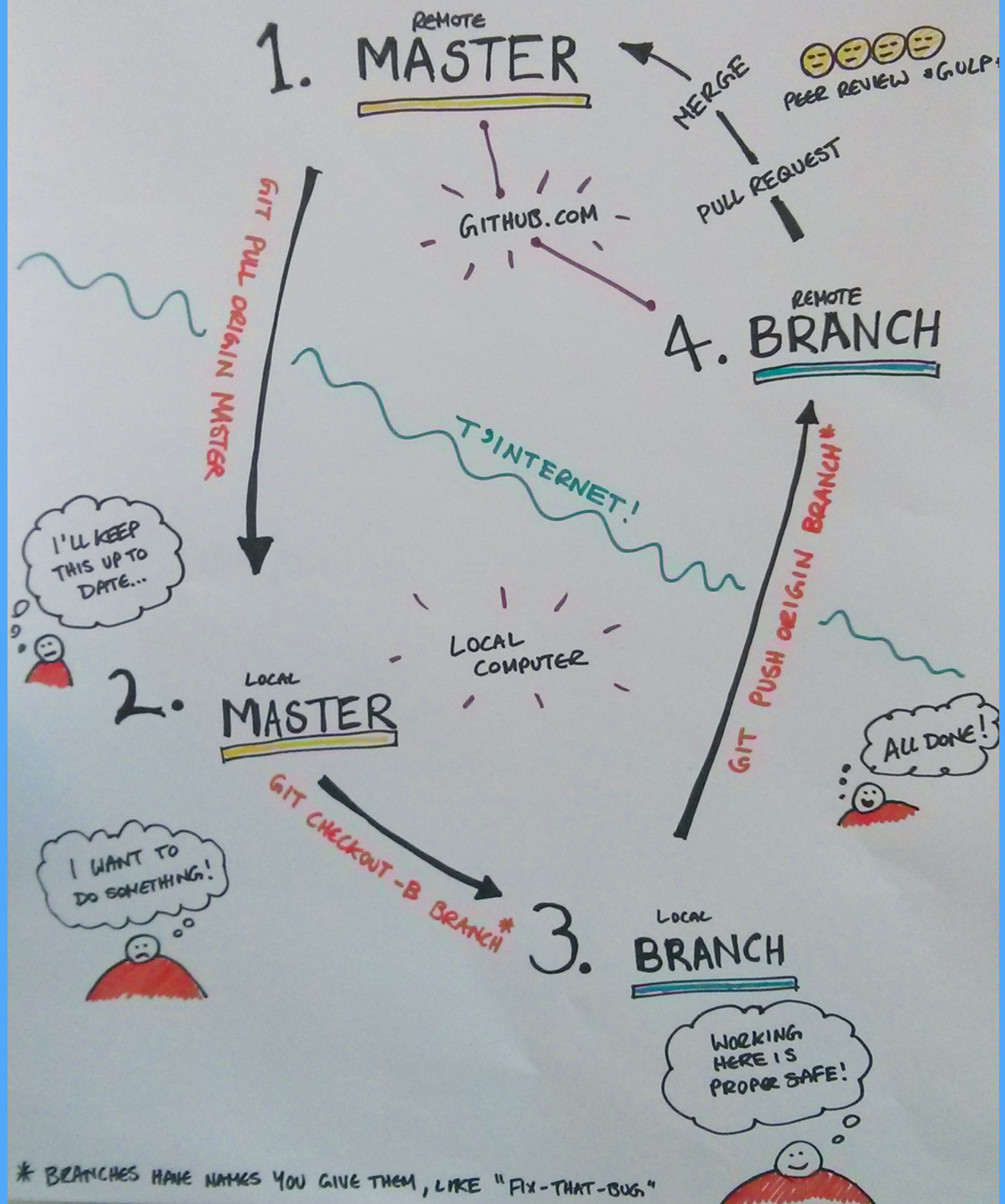
I am nervous about doing this.

Sometimes I feel I do not get Git.

I never had lessons. Or read a manual.

Git is a thing I learnt as I went along.

THE GIT CIRCLE OF SAFETY!



1. What is Git?

Git is a **source control system**.

It allows to **store versions of our work, safely, securely**.

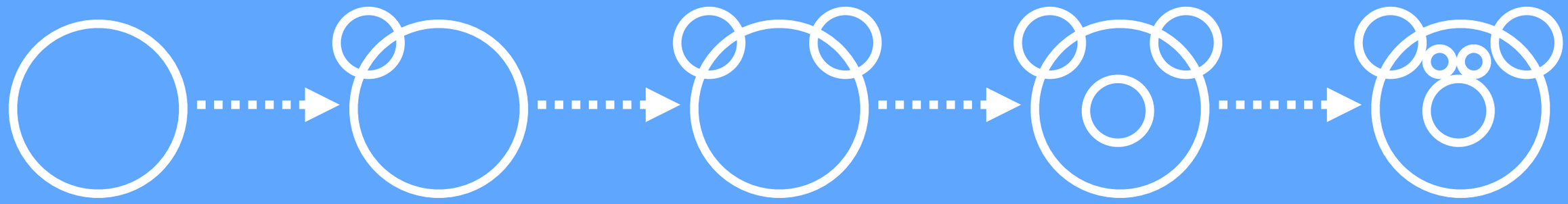
It allows to **save work locally – and remotely**.

2. What is source control?

Also can be known as **version control**.

> The management of changes to documents, computer programs, large websites, and other collections of information.
>
> Changes are usually identified by a number or letter code (the revision number).
>
> *Wikipedia*

3. What are versions?



Version 1 Version 2 Version 3 Version 4 Version 5

4. What are ‘local’ and ‘remote’?

Local
The machine you work on. Usually your laptop.

Remote
Another computer. A server. On another network. In the cloud. Somewhere else.

5. How does Git work?

- Majority of work is done locally.
- When you feel the need you *push* to the server.
- Git creates “versions” of your stuff.
- Git keeps a history of changes – which you can comment on.
- Git allows you to go back to a previous version.
- Git allows you to branch work – yours’, others’.
- Git allows you to merge work together.

6. Why use Git?

Why not?

- It provides us with *security* across many, if not all definitions of the word.
- Git lets us return code to previous state: Git allows you to *roll back* – revert to a previous commit.
- This allows us to be playful.
- Git allows us to save work in more than one place.
- Git allows us to share.
- Git allows more than one person to contribute to work.

Get busy!

Form pairs.

Zero. We are typing commands

We are going to use a **shell** or **terminal** window for this session.

One, we're using a mixture of Windows, Ubuntu and OX X machines.

Two, it should help us concentrate on the commands.

One. Create a folder.

To start with we need a folder.

This is where **we store files**, just like another day.

So, create a folder. (And create the folder somewhere you know where it is.)

Two. Create a file.

In the folder, let's create a file.

Files can be anything that is digital information.

Let's keep it simple and create a basic HTML file.

And call it ***your_name*.html**

(Replace *your_name* with your name.)

```
<html>
<body>
<p>This is a test file.</p>
</body>
</html>
```

Three. Let's get ready to Git!

Open a shell window.

Navigate to the directory we created earlier.

Four. Set up Git.

We have to “log in” or “sign in” to Git so it knows who you are.

Type:

```
git config --global user.name 'ermlikeyeah'
```

```
git config --global user.email ‘si@studiooofthings.com’
```

Five. Create a repository.

A repository is a directory Git uses to version control your files. This isn't for us. It is for Git.

We are in our work folder so type:

```
git init
```

This creates a folder called **.git**

Six. Check where Git is at.

At any point we can get Git to tell us how it's doing.

Type:

```
git status
```

Seven. Add file to Git.

We need to tell Git we need it to track a file.

Type:

```
git add your_name.html
```

Eight. Check where Git is at.

Let’s check where Git is at.

Type:

```
git status
```

Nine. Create another file.

In the folder, let's create another HTML file.

Get the other person in the pair to do this.

Call the file ***another_name*.html**

Again, replace *another_name* with an actual name.

And make the content different to the previous file.

Ten. Add the second file to Git.

Do you remember how to do that?

Ten. Add the second file to Git.

Do you remember how to do that?

Type:

```
git add another_name.html
```

Eleven. Check where Git is at.

Let’s check where Git is at.

Eleven. Check where Git is at.

Let’s check where Git is at.

Type:

```
git status
```

Twelve. Committing to Git.

Let’s commit our files to Git.

This is like taking a snapshot of where the files are.

Type:

```
git commit –m “Your commit message”
```

Pro tip: Make your commit message useful!

Thirteen. Ch-ch-changes.

Make some changes to your two files.

Do not add the files.

Do not commit the files.

Fourteen. C’est la difference.

Git can tell us what the differences between commits are.

To get Git to tell us all the differences type:

```
git diff
```

```
git add filename.html
```

```
git status
```

```
git commit –m “Your commit message”
```

Fifteen. MORE files.

Create another HTML file.

DIFF.

Add it to Git.

DIFF.

Commit it to Git.

DIFF.

```
git add something.html
```

```
git diff
```

```
git commit –m “Your commit message”
```

```
git status
```

Sixteen. Adding in more files.

If you want to add lots of files there is a shortcut.

We have lots of files now.

So use:

```
git add .
```

Seventeen. Viewing the log.

Everything you commit into Git is logged.
(Remember: The your *useful messages*.)

Type:

```
git log
```

Next time.

We will look at pushing, pulling, fetching, branches and merges.

Any questions, feedback, cake?

Email
simon.wilson@digital.hmrc.gov.uk

Reference and sources.

1. Yusuf's original inter-HMRC Digital Git presentation.
2. That picture early on is from Mat Johnson, elsewhere in government.
3. A really good step-by-step demo
4. Another really good step-by-step demo
5. A look at the flow of Git