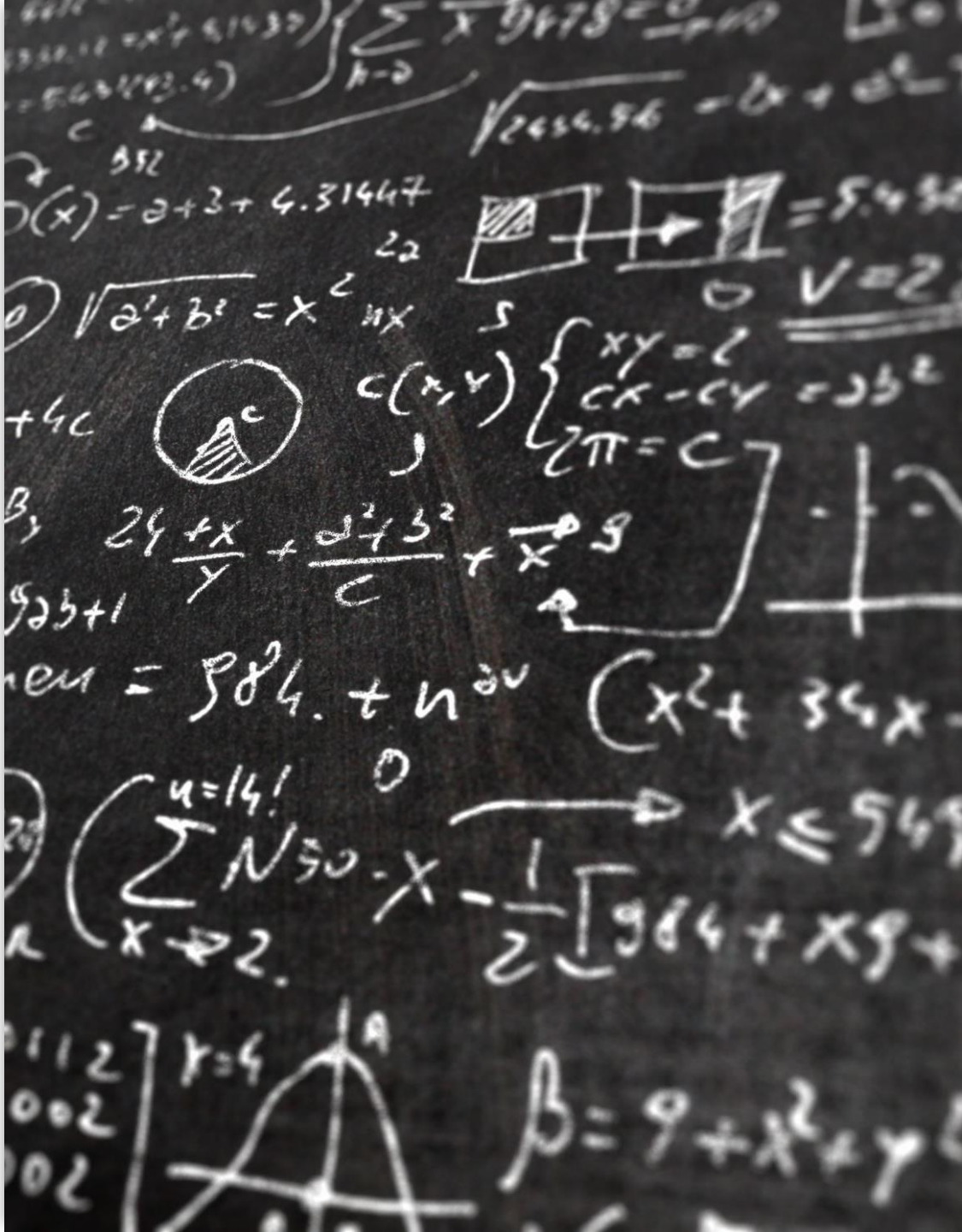
School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 17: Version Control

Adrian Friday, Nigel Davies, Hansi Hettiarachchi, Saad Ezzini

This lecture

- How to keep track of code as it changes/ evolves
- Why this is important for you and for team work
- An example: the **git** version control system



What’s the problem?

- Code changes all the time
  - We start with a nothing and we write code piece by piece
  - We fix logical errors
  - The requirements evolve so we extend our codebase
  - We work with others in teams

Scenario 1: Imagine this scenario

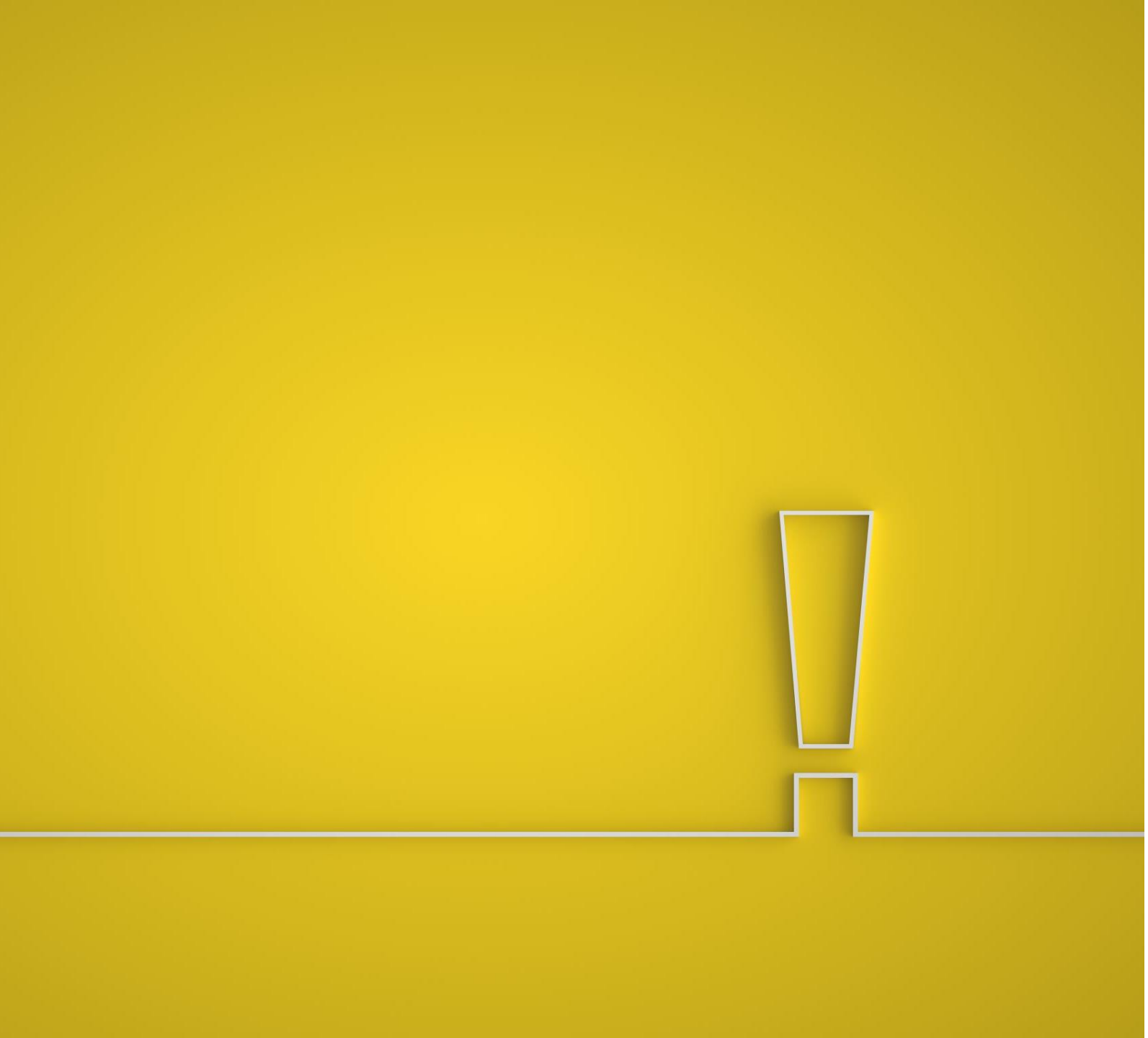
- You write your code for your boss/customer
- Your code meets the requirements as you understand them
- It passes the tests/ regression tests (i.e. it meets the behavioural spec, and you didn’t break anything that depends on it, ideally ☺)
- You ‘release’ the code into the production pipeline to the customer

Imagine this scenario… contd.

- You continue to evolve the code to meet new demands/ fix issues
- In the meantime your previous customer finds an issue
- You need to ‘go back’ and fix it
- How on earth do you know what code (including versions of libraries and dependencies) they have?
- Can you fix their issue, without losing what you’re currently working on?

Scenario 2

- You're working on a group coursework e.g. *next year*
- Several colleagues are working on the same codebase
- You've got it working, and it's close to the deadline
- But someone changed something, and now the demo doesn't work
- Who changed what and when? How can you identify where the problem is? *You haven't got long…*



What to avoid

- Files called things like:
  - Week10cwv4-final-FINALFINAL+AF-RELEASED

You need a more principled approach to tracking your code base

And specifically, how it changes, when and by whom…. you need, version control

Firstly, what is a ‘version’?

Put simply, you create a new version

- Every time you (or someone else) changes the code (additions, substitutions, removals)
- *Including* adding and removing files and dependencies to the project
- The version is the sum of the differences between the source files
- It’s only a version when you chose to ‘mark’ the set of changes at some point in time

What is version control?

- Software under ‘version control’ is software where…
- We explicitly choose to track or mark certain changes
- Revisions are created by marking or ‘committing’ the changes
- We label and/or tag each revision
- The differences between the source files are stored
- This forms *a revision history* over the timeline of the project

The cool thing

- The revision history allows us to see…
- The cumulative differences, i.e. how the code has evolved over time (when)
- Where the changes are
- What the changes were (…and who made them)
- And even better, we can *go back in time*… E.g. reverting (undoing) changes, rolling back to previous versions (e.g. that release to that customer)

A linear revision history…

Revision 1: initial commit

```
int main()
{
}
```

Changes made

Revision 2: Added headers etc.

```
#include <stdio.h>

int main()
{
  printf("Hello, world\n");
}
```

t_1 <timestamp + message + code snapshot>

t_2 <timestamp + message + differences>

Revision 1: initial commit

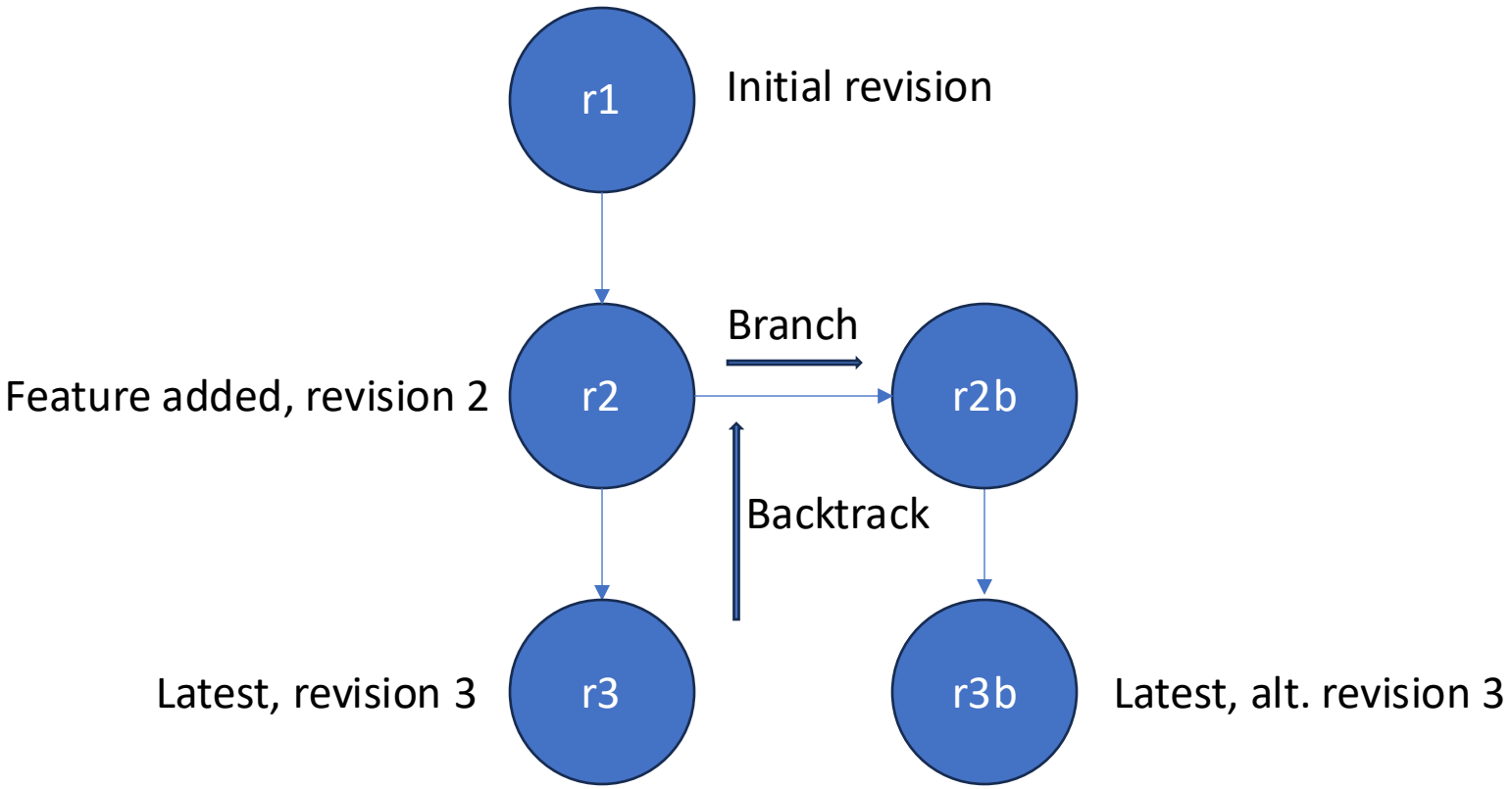
```
int main()
{
}
```

Revision 2: Added headers etc.

```
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,4 +1,6 @@
+#include <stdio.h>
+
 int main()
 {
-
-}
\ No newline at end of file
+    printf("Hello, world\n");
+}
```

Revision 2 is just Revision 1 plus this 'patch' or sequence of changes

More interesting example



Basic version control principles

You want to create a repository (‘repo’) for your project files

Add the files you want to track versions of to the repo (don’t forget new files when you create them in multifile projects!)

Periodically ‘commit’ versions to the repo (e.g. at sensible milestones)

As with comments, the commit message is the history of what you’ve done, so make them count/ add value (*get in good habits now!*)

Let’s look at an example using the tool ‘git’

- Recipe:
  1. create a folder for the project
  2. create a file in the folder
  3. create a **repo** for the project (**init**ialise repo)
  4. **add** files to repo (put under version control)
  5. **commit** initial version to repo
  6. make changes
  7. **commit** and create a new version

```
git init .
code helloworld.c
git add helloworld.c
git commit .
code helloworld.c
git commit .
```

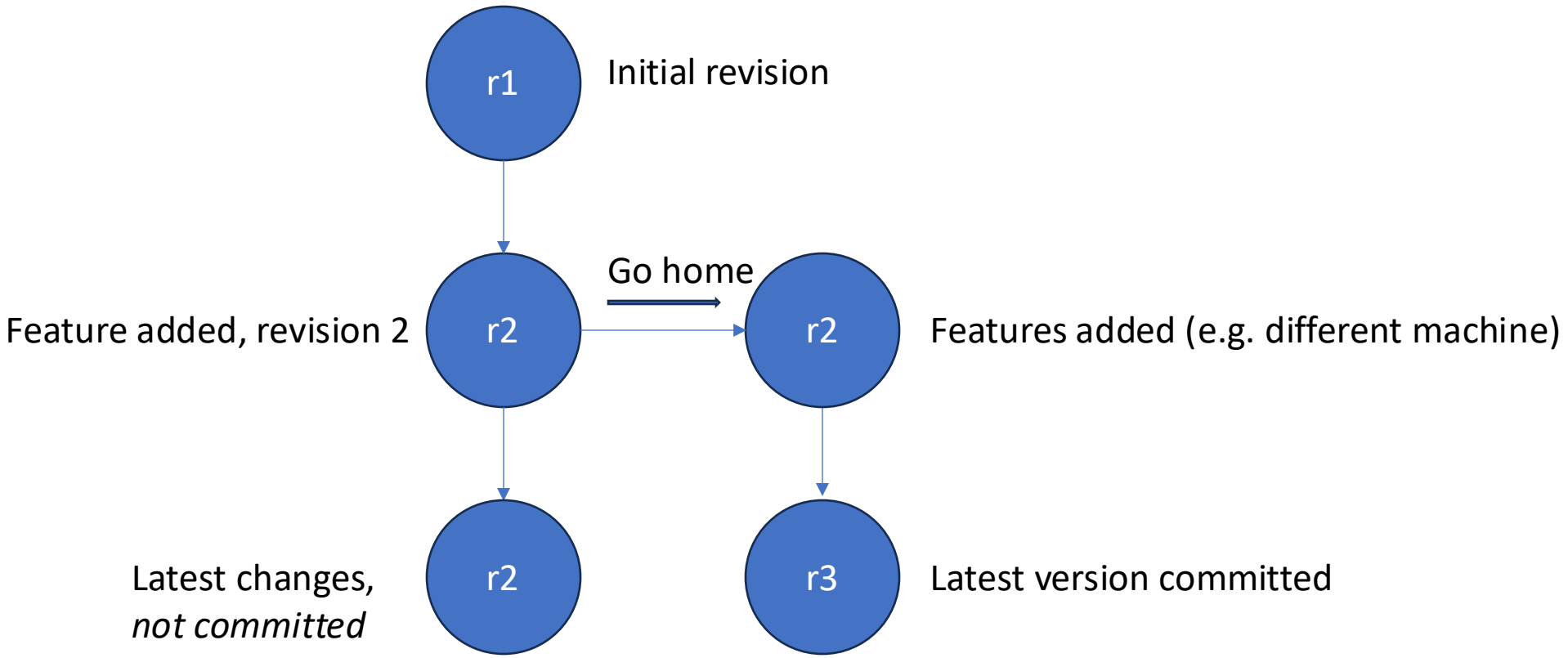
t_1

t_2

Some care required…!

- Version control systems only track the files you remember to add!
- They only create a version when you explicitly do so… (*e.g. remember to commit your latest changes before leaving the lab…*)
- git status - is useful for seeing what the state of your local repo is

More interesting example 2

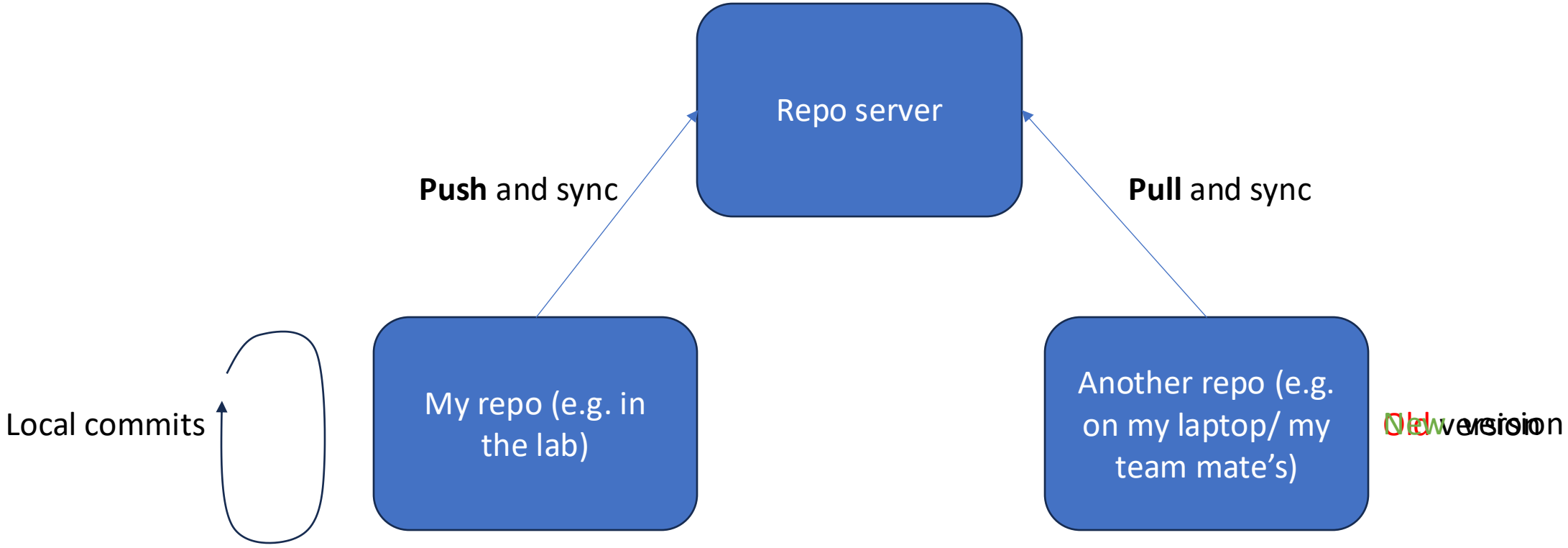


What just happened?

Synchronising versions across machines

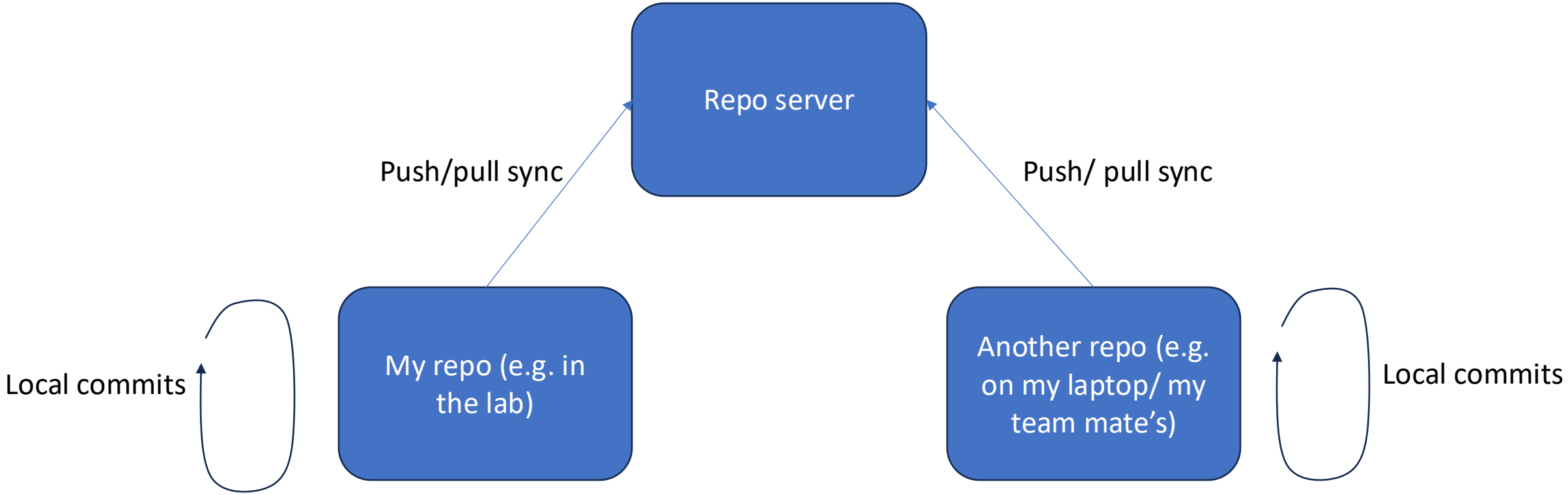
- Git is a ‘peer to peer’ distributed version control system
- It tracks changes locally (in a hidden folder called **.git**)
- You can ‘push’ and ‘pull’ changes (sync) between replicas of your repo (e.g. team members, or via some server)
- Servers include public repo servers such as github.com or (what we recommend, gitlab on scc-source.lancs.ac.uk)

Rendezvous *via* the server copy



Server copy is only as fresh as last push

What could possibly go wrong? ☺



!!! Unexpected version conflict !!!

Eek!

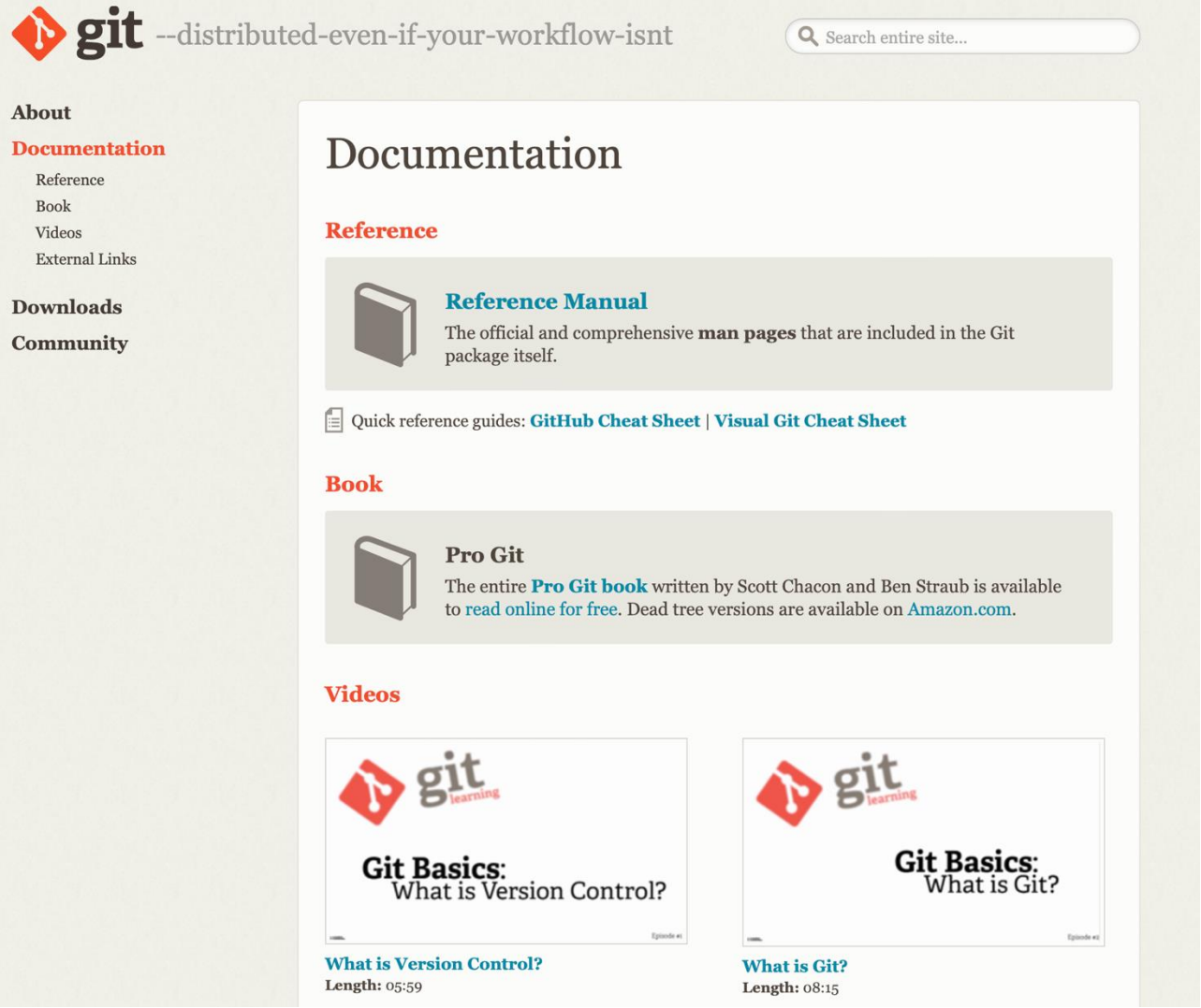
Enabling Teamwork

- You will need to create **an account** on a repo server, and a **repo** for your project
- Either ‘clone’ the server repo to create a local copy, or add a remote repo to your local project to create the remote (e.g. server) version
- Remember to **pull** before you **push** your changes to avoid version conflicts (keep repos up to date) – *a side chat channel can help if collaboration is fast paced*
- Version control systems will expect you to resolve and commit a reconciled version (from the merge branch) in git’s parlance

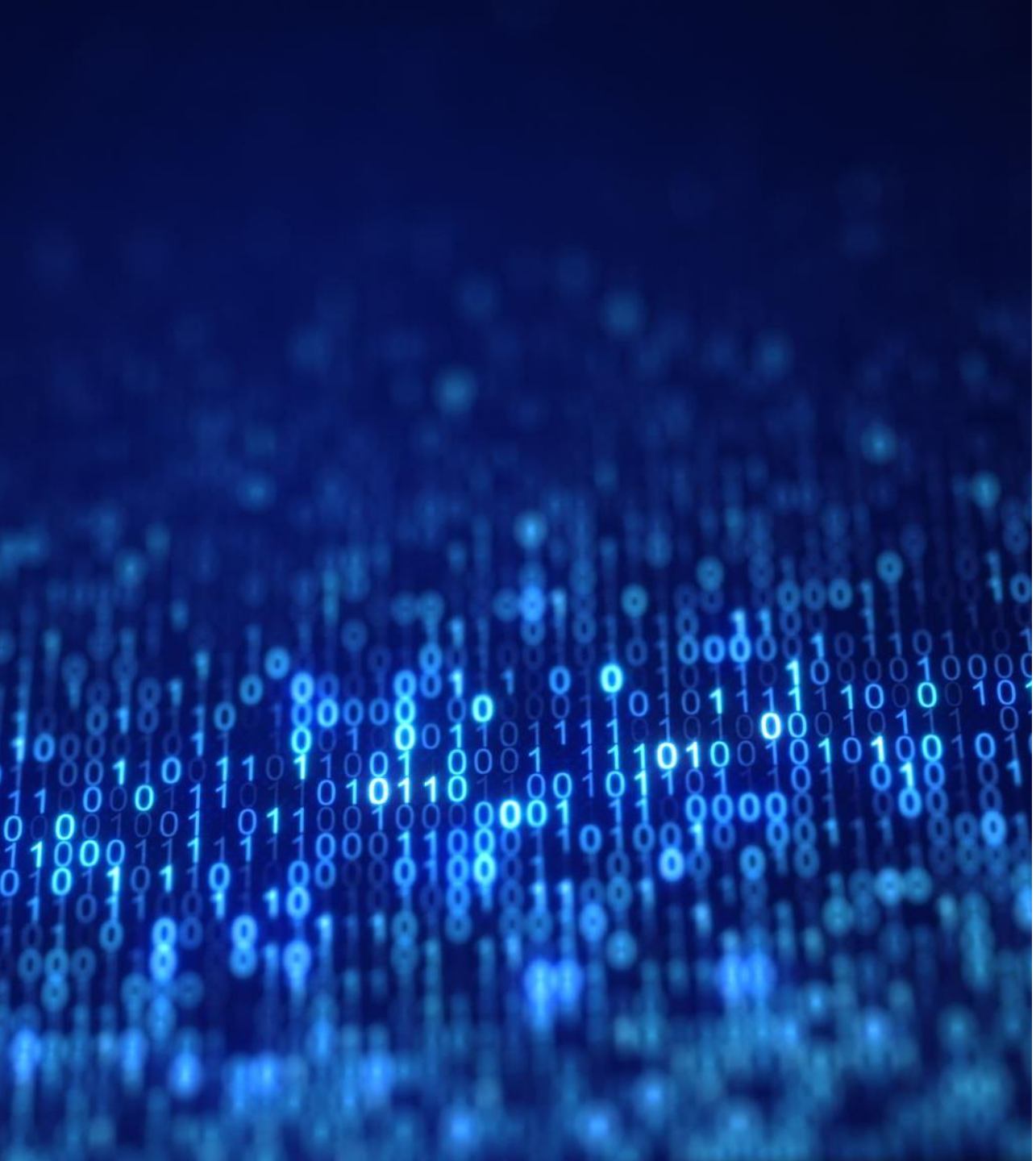
Cautionary note

- Use version control, use gitlab on scc-source, it can also help you evidence work is yours… and a host of other benefits
- You can choose to make repos on github.com etc., especially for your own projects
- Please create private **not public** visible repos of your coursework, or someone will find your work and you are helping them cheat… *you will also be under suspicion!*

More about version control with git



- https://git-scm.com/doc



Summary

- For non-trivial projects use version control
- Maybe even for trivial projects too ☺
- Git is one of many version control systems, but is recommended for SCC
- Remember: ’private’ repos!!!