University of Nottingham
UK | CHINA | MALAYSIA

Computer Engineering and Mechatronics MMME3085

Dr Louise Brown

Week 8

- Information about the final project submission
- Using arrays of structures (may be useful for project!)
- Software engineering best practise – the tools we use
    - More on using version control

Project final submission

Software Project – Final Submission (30%)

- Project submission – **3pm Thursday 12th December**

- Learning outcomes:

- To write robust code for a mechatronics problem using a specification (as developed in the project planning submission)
- To develop the documentation required to allow the project to be developed by a team
- To be able to use version control for software development

Submission Requirements

Learning outcomes will be demonstrated in the submission documents by:

- Developing the **code** to fulfil the project brief, which is well structured and follows software engineering good practice (e.g. well named variables, error trapping).
    - Submit your VSCode project
- Providing a **system manual** which could be used by a software development team to maintain or continue to develop the code including the following:
    - A clear explanation of what the program does, including sample emulator output.
    - Description of files used in the program
    - Description of functions used in the program
        - The function declarations (prototypes) for each function identifying whether parameters are input or output and the return value (if any). You are encouraged to give a return value which indicates successful execution or failure.
    - Specification of the main data items used in the program.
    - Test data which will validate the program, confirming conformance of the program/function to its specification.
    - Flowcharts which show the structure of the program
- Using the git repository set up at the planning stage of the project

Robot Testing (5% of project mark)

ESLC B05:

5th December: 4-6pm

6th December: 9-11am and 2pm – 6pm

Coates C19:

9th December: 10am – 12pm (Computer lab session

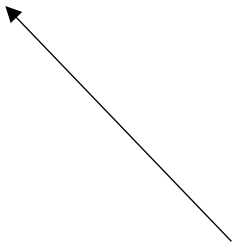
A spreadsheet will be made available closer to the dates to sign up for a 15 minute testing slot.

Note – you can sign up for any of the dates (you don't need to stick to the date/time on your timetable)

fscanf

Nearly the same as scanf **but** must pass the file handle created when the file was opened

```
fInput = fopen("fred.txt", “r");
fscanf( fInput, “%s %d”, charArray, &num );
```



Can read in more than one item at once

calloc

Allocates space for n items of size bytes each and **initialises each item to zero**

```
void *calloc(size_t nitems, size_t size);
```

Prototypes in stdlib.h and alloc.h

Inputs:

- nitems: number of items to allocate memory for
- size: size, in bytes, of each item

Returns

- a pointer to the newly allocated block or
- **NULL** if not enough space exists.

free

Frees blocks allocated with

- `malloc` or
- `calloc`

Prototype is

```
void free (void *block);
```

Found in stdlib.h & alloc.h

Note:

- We must use this to return memory, it is NOT automatically done when a function exits (only the pointer is released)

The `strcmp` function compares two strings. It returns a value of 0 if the strings are identical. e.g:

```
if ( strcmp( string1, string2) == 0 )
{
    // Code to execute if strings are the same
}
```

Storing an array of structures within a structure - static

```
# define SIZE 10

struct POINT
{
    float x,y;
};

struct MULTI_POINTS
{
    char name[20];
    struct POINT point[SIZE];
};

int main()
{
   struct MULTI_POINTS Points;
   int i;

   for (i = 0; i < SIZE; i++)
   {
        Points.point[i].x = i;
        Points.point[i].y = i;
   }

   for (i = 0; i < SIZE; i++)
         printf( “Point %d: x = %f, y = %f\n”, i, Points.point[i].x, Points.point[i].y);
}
```

Structure MULTI_POINTS contains an array of POINT structures. In this case we know the size of the array.

StructureArrays.c

Storing an array of structures within a structure – functions (1)

```
# define SIZE 10

struct POINT
{
    float x,y;
};

struct MULTI_POINTS
{
    char name[20];
    struct POINT point[SIZE];
};

int main()
{
   struct MULTI_POINTS Points;
   int i;

   for (i = 0; i < SIZE; i++)
   {
        Points.point[i].x = i;
        Points.point[i].y = i;
   }

   for (i = 0; i < SIZE; i++)
         printf( “Point %d: x = %f, y = %f\n”, i, Points.point[i].x, Points.point[i].y);
}
```

How can we assign values to these points within a function?

Storing an array of structures within a structure – functions (2)

```c
int main()
{
    struct MULTI_POINTS Points;
    int i;

    PopulatePointArray( &Points );

    for (i = 0; i < SIZE; i++)
        printf( "Point %d: x = %f, y = %f\n", i, Points.point[i].x, Points.point[i].y);
}

void PopulatePointArray( struct MULTI_POINTS *points)
{
    int i;
    for (i = 0; i < SIZE; i++)
    {
            points->point[i].x = i;
            points->point[i].y = i;
    }
}
```

Pass a pointer to the structure to the function

Declare the function parameter as a pointer to a MULTI_POINTS structure

`points` is a pointer to the structure so use the arrow notation to access its members

StructureArraysFunc.c

Storing an array of structures within a structure – dynamic (1)

```
# define SIZE 10

struct POINT
{
    float x,y;
};

struct MULTI_POINTS
{
    char name[20];
    struct POINT point[SIZE];
};

int main()
{
   struct MULTI_POINTS Points;
   int i;

   for (i = 0; i < SIZE; i++)
   {
        Points.point[i].x = i;
        Points.point[i].y = i;
   }

   for (i = 0; i < SIZE; i++)
         printf( “Point %d: x = %f, y = %f\n”, i, Points.point[i].x, Points.point[i].y);
}
```

What do we do if we don’t know the size of the array at compile time?

Storing an array of structures within a structure – dynamic (2)

```
struct POINT
{
    float x,y;
};

struct MULTI_POINTS
{
    char name[20];
    struct POINT *point;
};

int main()
{
   struct MULTI_POINTS Points;
   int i;
   int numPoints = 0;

   printf( "Please input number of points: ");
   scanf("%d", &numPoints);
   Points.point = calloc( numPoints, sizeof(struct POINT) );

   // Use the array

   free( Points.point );
}
```

Declare a pointer to the structure type

Allocate the memory to the pointer using malloc or calloc

Number of items in the array

Size of each item in the array (ie size of the POINT structure)

Finally, free up the memory

Storing an array of structures within a structure – dynamic (3)

```
struct POINT
{
    float x,y;
};

struct MULTI_POINTS
{
    char name[20];
    struct POINT *point;
};

int main()
{
   struct MULTI_POINTS Points;
   int i;
   int numPoints = 0;

   printf( "Please input number of points: ");
   scanf("%d", &numPoints);
   Points.point = calloc( numPoints, sizeof(struct POINT) );

   // Use the array

   free( Points.point );
}
```

To view elements of a dynamically allocated array in the debugger use the form `*pointer@numelements` in the watch window.
Here it will take the form:
`*(Points.point)@numPoints`

StructureArraysDynamic.c

Storing an array of structures within a structure – functions (3)

The structure containing the dynamically allocated array can be used in the function in exactly the same way but it will be necessary to pass the size of the array as a parameter.

(An alternative could be to store the size of the array in the multi-point structure)

To view elements of the dynamically allocated array in the debugger use the form `*(points->point)@numPoints` when viewing the code within the function

StructureArraysDynamicFunc.c

Software Engineering Best Practice

Part 2

Development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

- The tools that we use to make the coding process more straightforward and robust
- The practise of writing good code
  - There may be some overlap between the two (e.g. using a debug environment combined with coding techniques to identify bugs)
- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

The tools used for code development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

- The tools that we use to make the coding process more straightforward and robust
  - What editor or development environment will be used?
  - How do we create build files (important once a project has more files)?
  - How do we find bugs?

- The practise of writing good code
  - There may be some overlap between the two (e.g. using a debug environment combined with coding techniques to identify bugs)

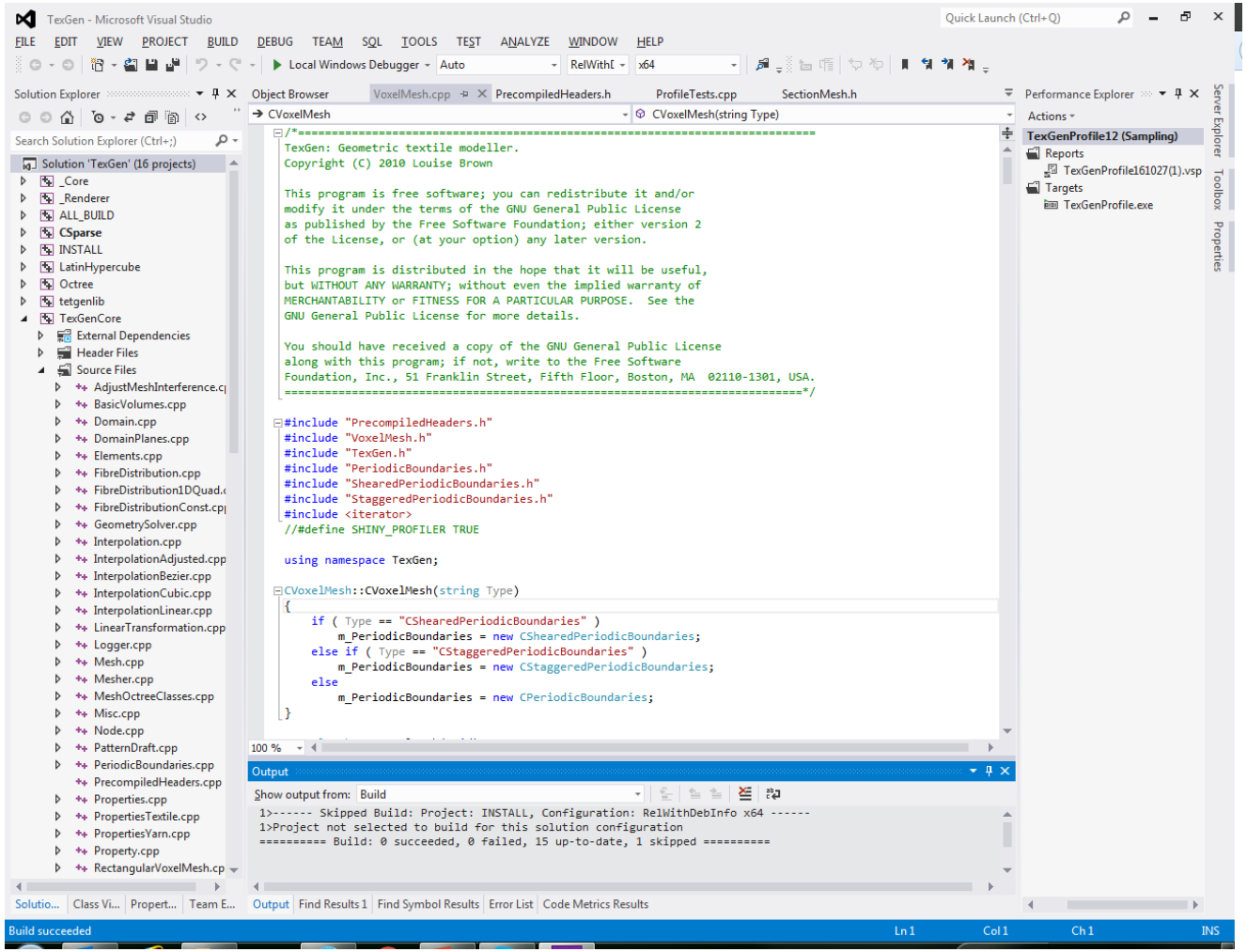
- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

Use an IDE (Integrated Development Environment)

Include features such as:

- Code editor
- Debugging facility
- Compiler
- Profiler
- Auto code-completion
- Version control integration
- …and many more

Many IDEs will support multiple languages



IDE	C/C++	Fortran	Python	MATLAB	Windows	Linux/Unix	MacOS
Eclipse	Eclipse CDT	Eclipse Photran	Using PyDev		✓	✓	✓
Code::Blocks	✓	Code::Blocks IDE for Fortran			✓	✓	
Visual Studio Code	✓	✓	✓		✓	✓	✓
PyCharm			✓		✓	✓	✓
Spyder			✓		✓	✓	✓
MATLAB				✓	✓	✓	✓
Microsoft Visual Studio. Community Edition is free, Professional and Enterprise are paid.	✓		✓		✓	✓	✓

And many more…

Python: https://realpython.com/python-ides-code-editors-guide/

C: https://www.geeksforgeeks.org/10-best-ides-for-c-or-cpp-developers-in-2021/

Fortran: https://cyber.dabamos.de/programming/modernfortran/editors-and-ides.html

The tools used for code development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

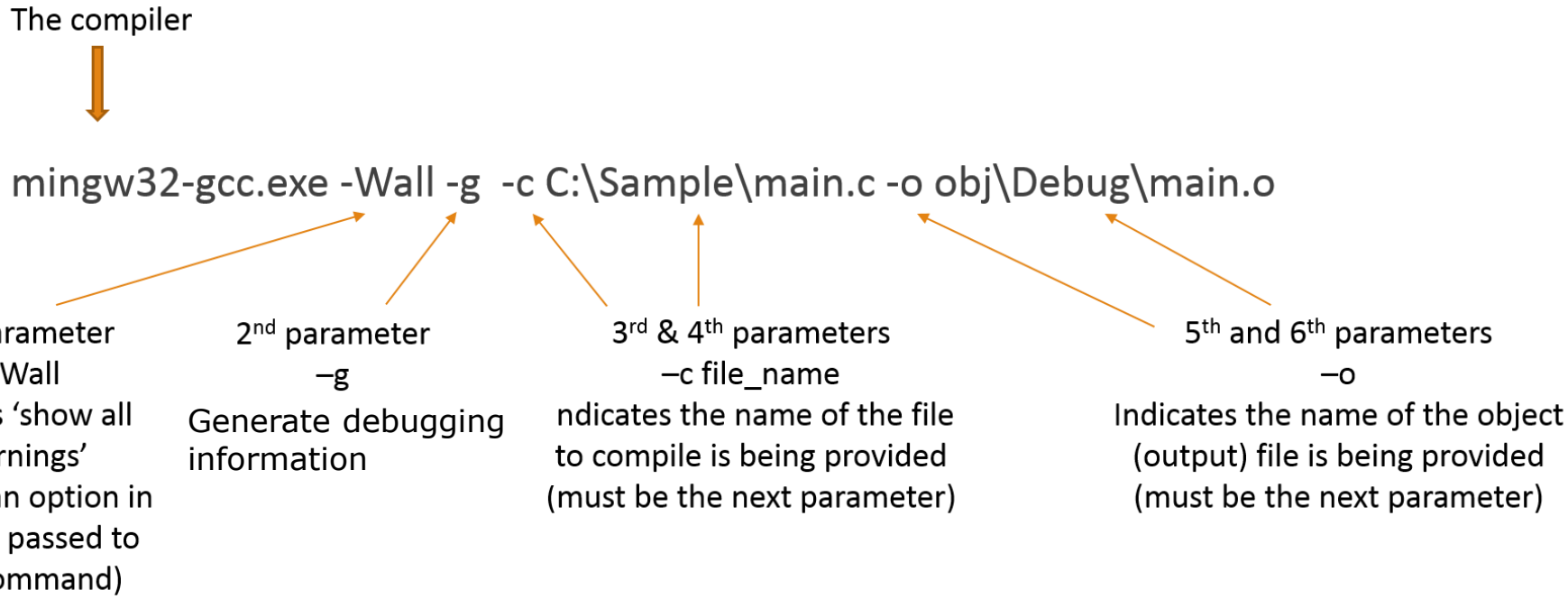
- The tools that we use to make the coding process more straightforward and robust
  - What editor or development environment will be used?
  - How do we create build files (important once a project has more files)?
  - How do we find bugs?

- The practise of writing good code
  - There may be some overlap between the two (e.g. using a debug environment combined with coding techniques to identify bugs)

- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

Creating build files

With a small number of files it is possible to generate the command to build and link code by hand



Tools for creating build files

There are various tools which can be used to automate the generation of build files.
Which one is chosen may depend on the programming language and operating system. A comprehensive list can be found here:
https://en.wikipedia.org/wiki/List_of_build_automation_software

Tools for creating build files

- In VSCode we have been using the C/C++ Runner extension to generate the build file automatically.

- TexGen uses CMake as this allows C++ to be built on multiple platforms.
  - A CMakeLists.txt file is produced for each project which tells CMake where to look for required files, executables and libraries
  - When CMake runs it processes the CMakeLists files and generates the build file automatically

The tools used for code development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

- The tools that we use to make the coding process more straightforward and robust
  - What editor or development environment will be used?
  - How do we create build files (important once a project has more files)?
  - How do we find bugs?

- The practise of writing good code

- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

The practise of writing good code

Superior coding techniques and programming practices are hallmarks of a professional programmer.

The bulk of programming consists of making a large number of small choices while attempting to solve a larger set of problems.

How wisely those choices are made depends largely upon the programmer's skill and expertise. [1]

[1] https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx

The Practise of Writing Good Code: Functions

Try to keep a function for one purpose only. For example don't write a function to calculate some variables and then plot them. Create two functions – you may also want to do the calculations without plotting and a general purpose plot function is more likely to be reused.

If you find yourself repeating a very similar piece of code around your program it should probably be a function

Where possible use a title which describes what both what the function **does** and an object

- PrintDocument()
- CalcPenPosition()
- CalcStartPosition()

Where possible, limit the number of parameters passed. Make sure all parameters are used.

The Practise of Writing Good Code

Even though we've selected an IDE and designed code to the level of function definitions there is another step before actually writing the code:

Pseudocode - a plain language description of how an algorithm, function, class or program will work.

- Describe specific operations using English-like statements
- Do not use syntax from the final programming language
- Write at the level of intent. Describe the meaning rather than how it will be done
  - If the pseudocode is written in the IDE as comments these will stay in your code
- Write at a low enough level that generating the code will be almost automatic. It may be an iterative process.

Example Pseudocode

```
ReadShape( fileHandle, ShapeData )
{
   read shape name from file
   read number of strokes from file

   allocate memory for number of pen strokes
   if failed to allocate memory
        return false
   endif
   for each stroke in file
       read x coord into ShapeData penStroke array
       read y coord into ShapeData penStroke array
       read pen up/down into ShapeData penStroke array
   end loop

   return true
}
```

Best Practice – a guide

- As you start to develop code that will be both shared and that will ‘grow’ over time it is important that you start to adopt some best practices
- Often companies will have their own ‘house’ style
  - For example, how to align the brackets when ‘blocking’ code for an loop/condition etc.
- This best practice guide produced by Microsoft is a few years old but still provides examples of very good practice:
  - https://msdn.microsoft.com/en-us/library/aa260844(v=vs.60).aspx
- Another, more general guide is given here:
  - https://mitcommlab.mit.edu/broad/commkit/coding-and-comment-style/
  - Links at the end of the page give style guides for specific languages.

The tools used for code development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

- The tools that we use to make the coding process more straightforward and robust
  - What editor or development environment will be used?
  - How do we create build files (important once a project has more files)?
  - How do we find bugs?

- The practise of writing good code
  - There may be some overlap between the two (e.g. using a debug environment combined with coding techniques to identify bugs)

- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

Debugging

“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”

- *Brian Kernighan*

Debugging

“Software quality must be built in from the start. The best way to build a quality product is to develop requirements carefully, design well, and use high-quality coding practices. **Debugging is a last resort**”

McConnell, S. (2004). Code Complete, Microsoft Press.

How not to debug!

- By guessing:
  - Scatter code with print statements
  - Randomly change things until it works
  - Don’t back up original version
  - Don’t keep notes
- By not spending time understanding the problem
- Fixing the problem with a workaround
  - Make a special case to deal with the error

Debugging tools

Source-code comparators

- Diff, WinDiff or git diff to see what has changed from the last working version

Compiler warning messages

- Set the compiler to the highest warning level
  - Uninitialised variables, pointers etc often cause problems
- Some compilers allow warnings to be treated as errors

Lint utility and static code analysis tools

- Check for code issues

Symbolic debugger

- Part of the IDE
- Use to step through code to see exactly what the code is doing
  - It won’t solve the problem for you – it will help you to find it
- Great for understanding someone else’s code
- Set breakpoints to home in on a particular part of the code

The tools used for code development

Having decided on the low level design we are in a position to start writing some code.

We need to think about:

- The tools that we use to make the coding process more straightforward and robust
  - What editor or development environment will be used?
  - How do we create build files (important once a project has more files)?
  - How do we find bugs?

- The practise of writing good code
  - There may be some overlap between the two (e.g. using a debug environment combined with coding techniques to identify bugs)

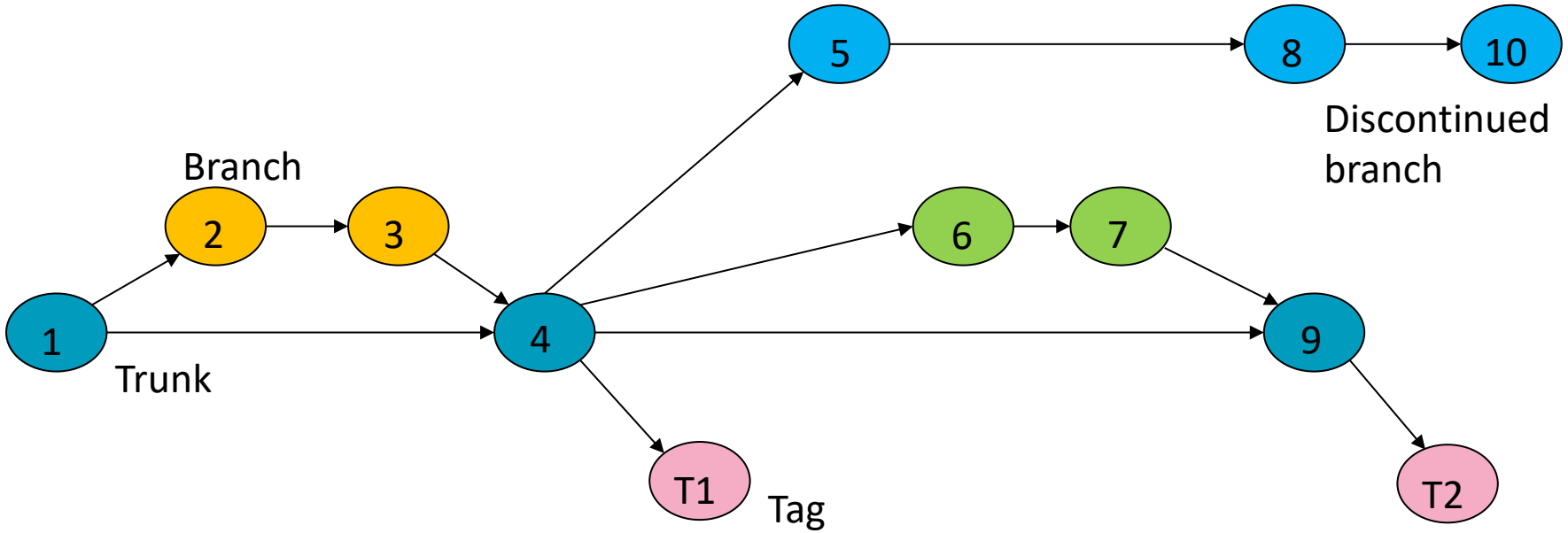
- How will we keep a track of changes to the code, particularly if being developed by a group
- How do we keep the code safe in the event of a system crash?
- How do we share code with our fellow developers?

Git Revisited

- git: To keep our code ‘safe’
  - A free version control system
  - It allows us to keep version of the code so we can ‘go back’
  - We can ‘branch’ code to try things
  - Share code with others who can then ‘check in’ code when they have finished with it
  - https://git-scm.com/downloads

Local git workflow

- Distributed system – have own version of the repository on local computer
- Using a remote repository gives backup and easier sharing between developers
- Integrated into some IDEs eg VSCode, Visual Studio and Matlab
- Easy use of branches for experimental code development

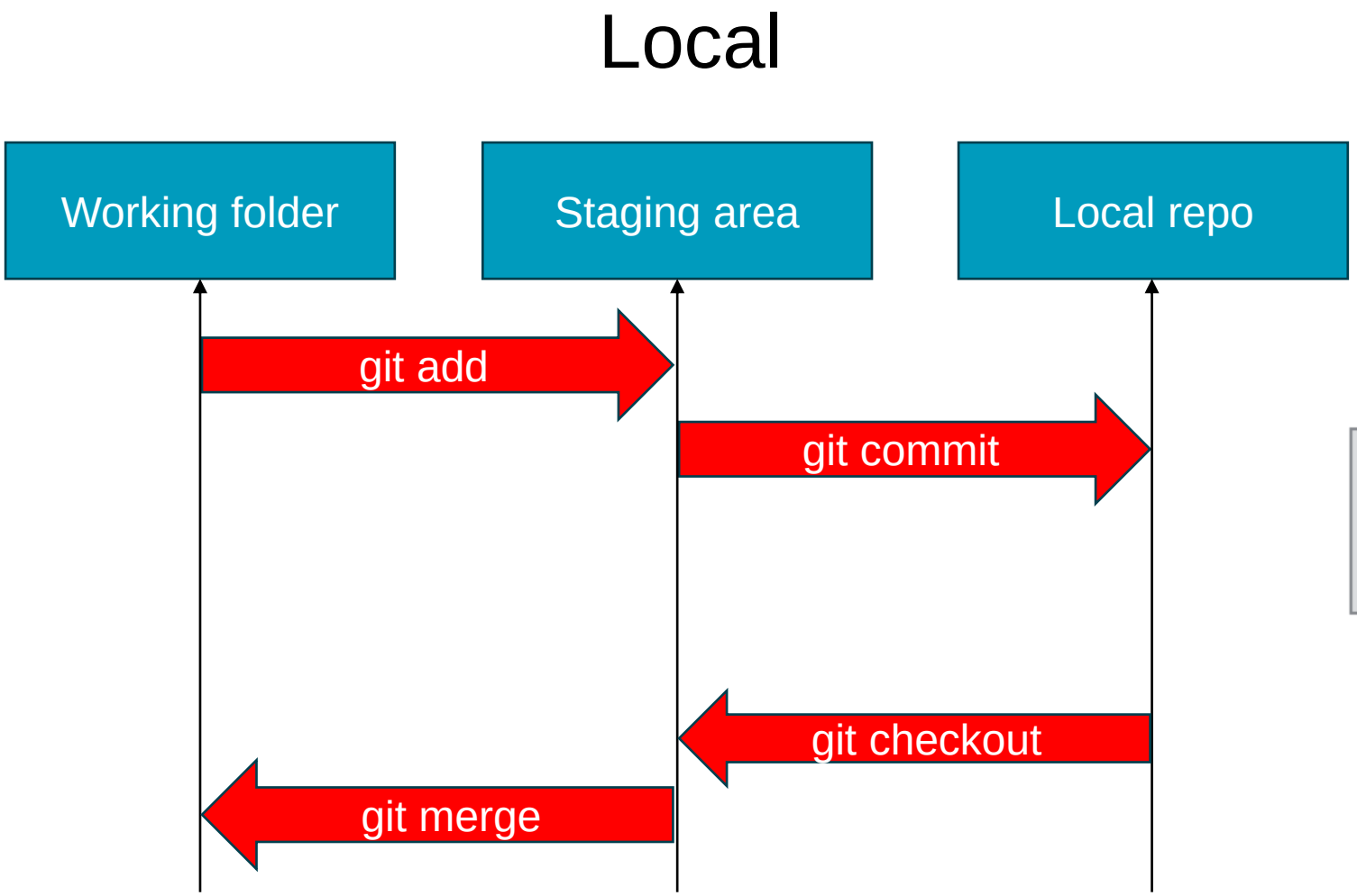


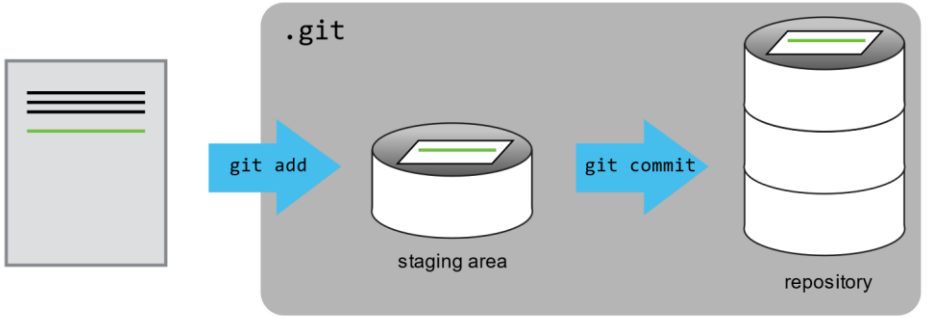
Where does Git store information?

- The command `git init` will create a repository in the folder where the command was executed
  - This creates a hidden subfolder called .git
  - This contains all the information about the project, including files and sub-folders.
  - **Do not manually change anything in this folder – the Git commands executed will do this for you**

- We will be using the VSCode – GitHub – Git integration to create repositories

Tracking Changes - Git Workflow

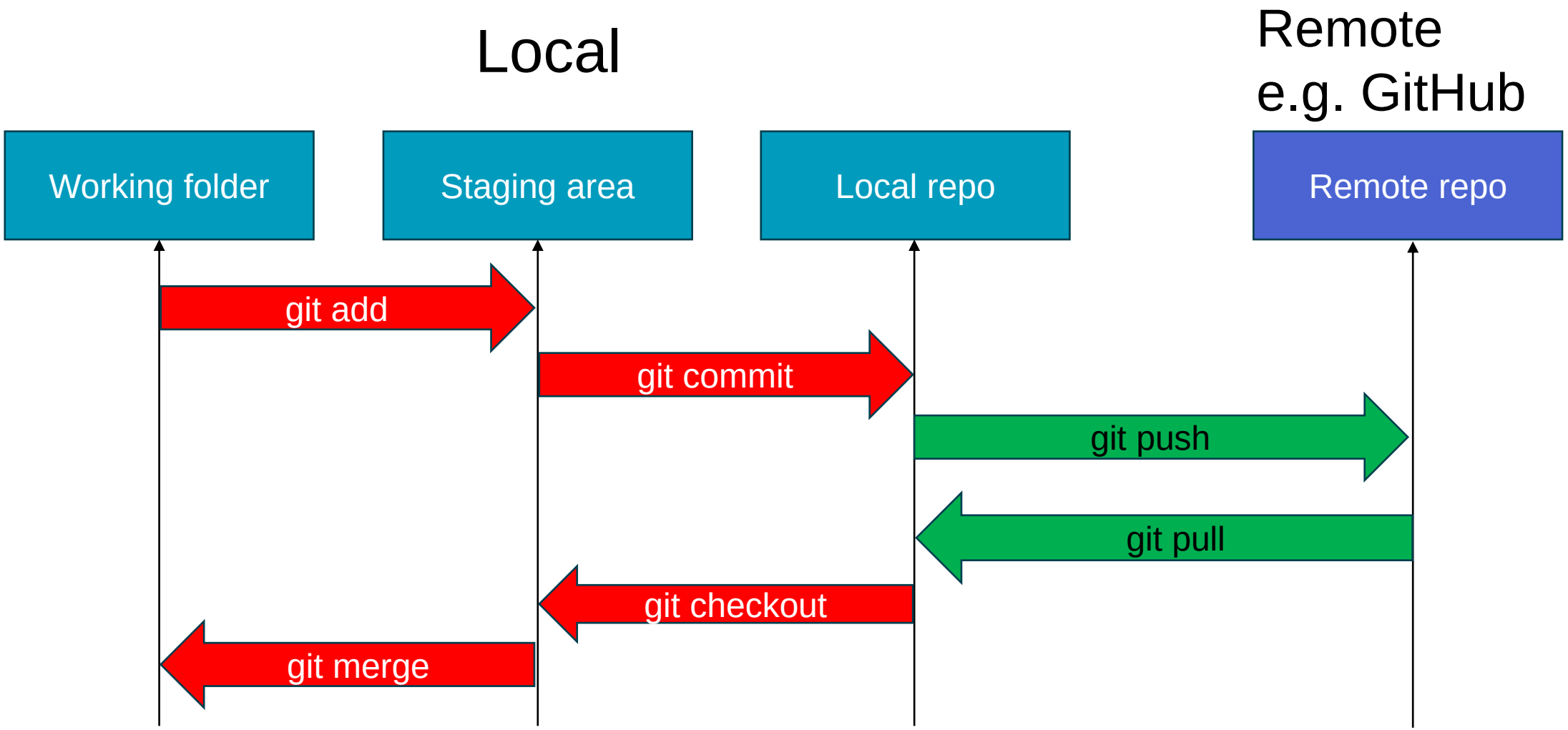




Save one or more changes to go into the 'snapshot'

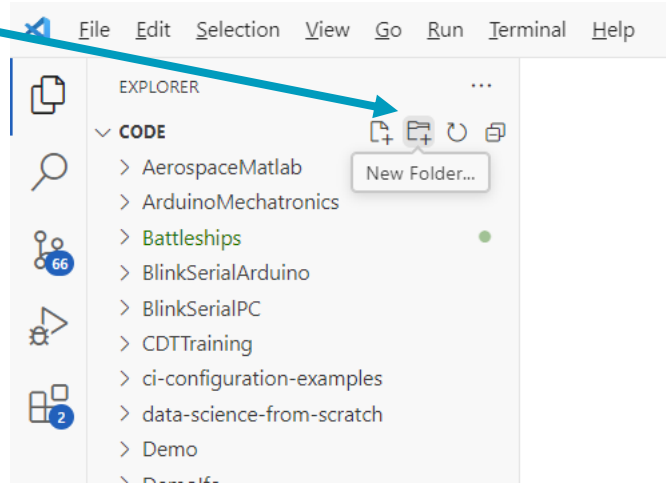
Save the snapshot to the repository

Git Workflow – with remote

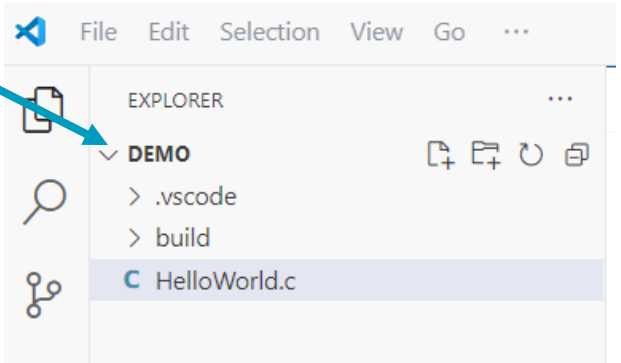


Integrate VSCode and Git – Create a new repo (1)

- First, from VSCode create a new folder for your repository :

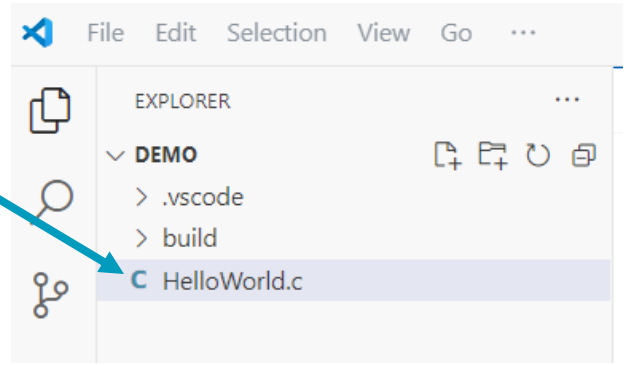


- Open the folder in VSCode and add a file (in this case HelloWorld.c)
- VSCode automatically creates a .vscode folder
- When the code is built the ‘build’ folder is created

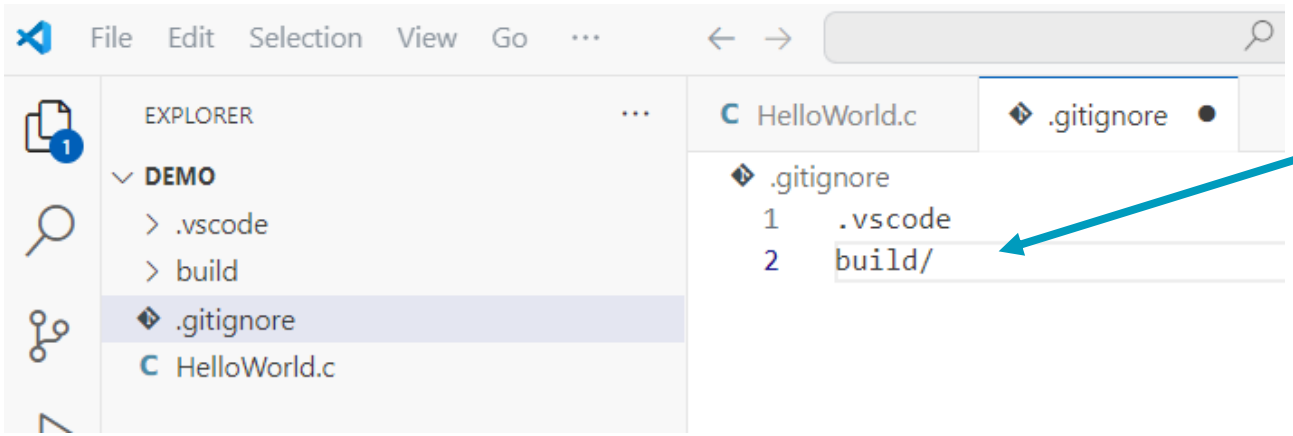


Integrate VSCode and Git – Create a new repo (2)

- We only want to track our code in the git repository
- We don't need to track the .vscode folder which contains project information or the build folder which contains the object files and binary executables



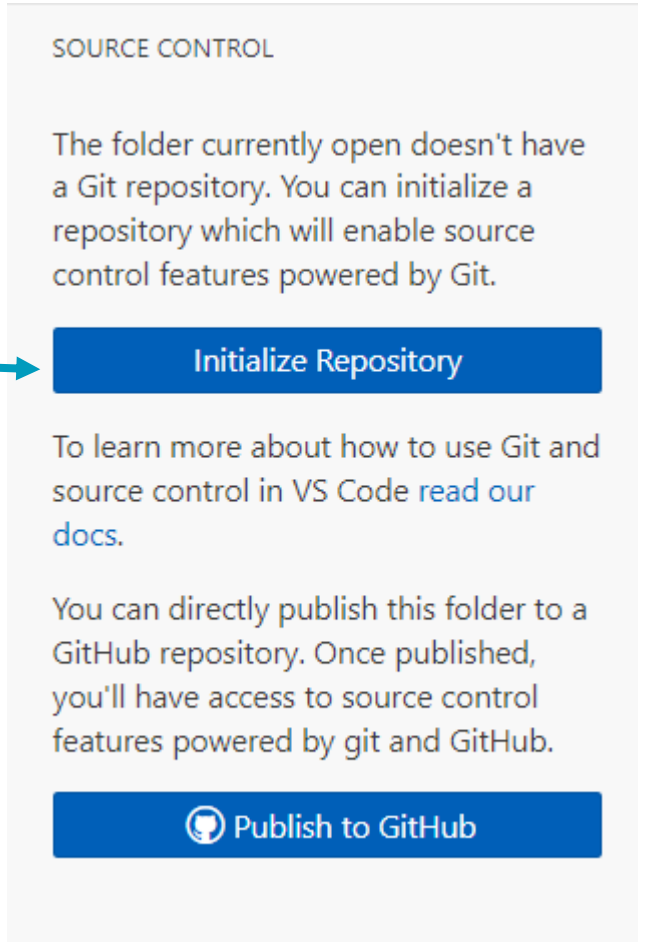
- Create a .gitignore file to contain information about files which are not tracked in the repo



- Add the two lines of code to the .gitignore file and save the file
- The contents of the .vscode folder will be ignored
- The contents of the build folder and all subfolders will be ignored

Integrate VSCode and Git – Create a new repo (3)

- Click on the Source Control icon in the lefthand toolbar
- The Source Control dialog will open
- Select Initialize Repository to create a git repo in the folder
- A hidden folder .git will be created in the folder (this may not be visible from VSCode)
- This contains all the information about the repository and the commit history
- **Do not edit anything in this folder!**



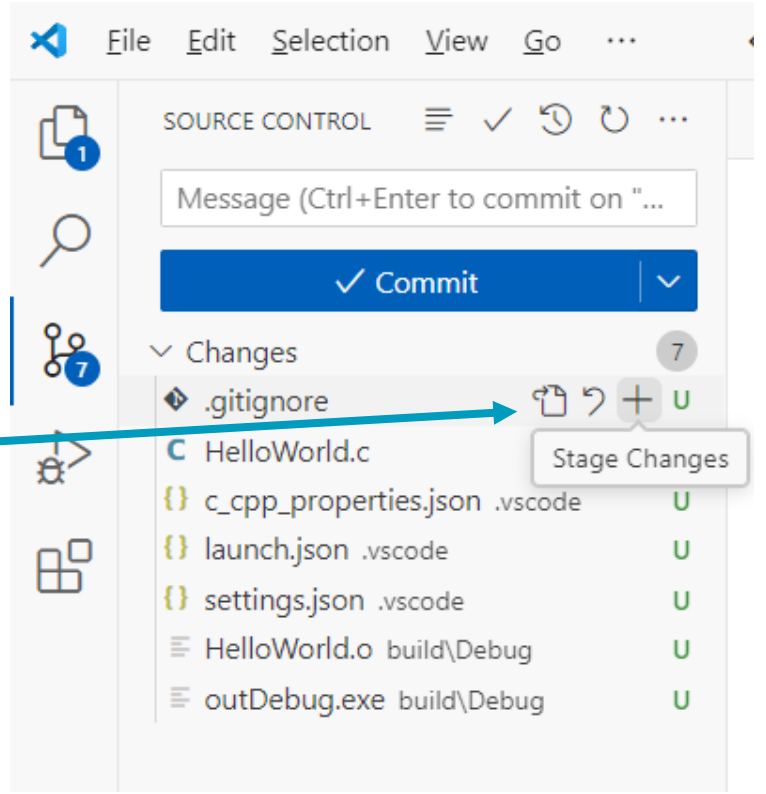
Integrate VSCode and Git – Stage initial changes

- The Source Control dialog now shows the changes that have been made to files and folders in the repo
- The U shows files which have not yet been added to the repo and are untracked

- Hovering the mouse over one of the files will show a submenu
- Select the + to Stage Changes
- An A will show that the file will be added to the repo at the next commit

- Add the HelloWorld.c and .gitignore files

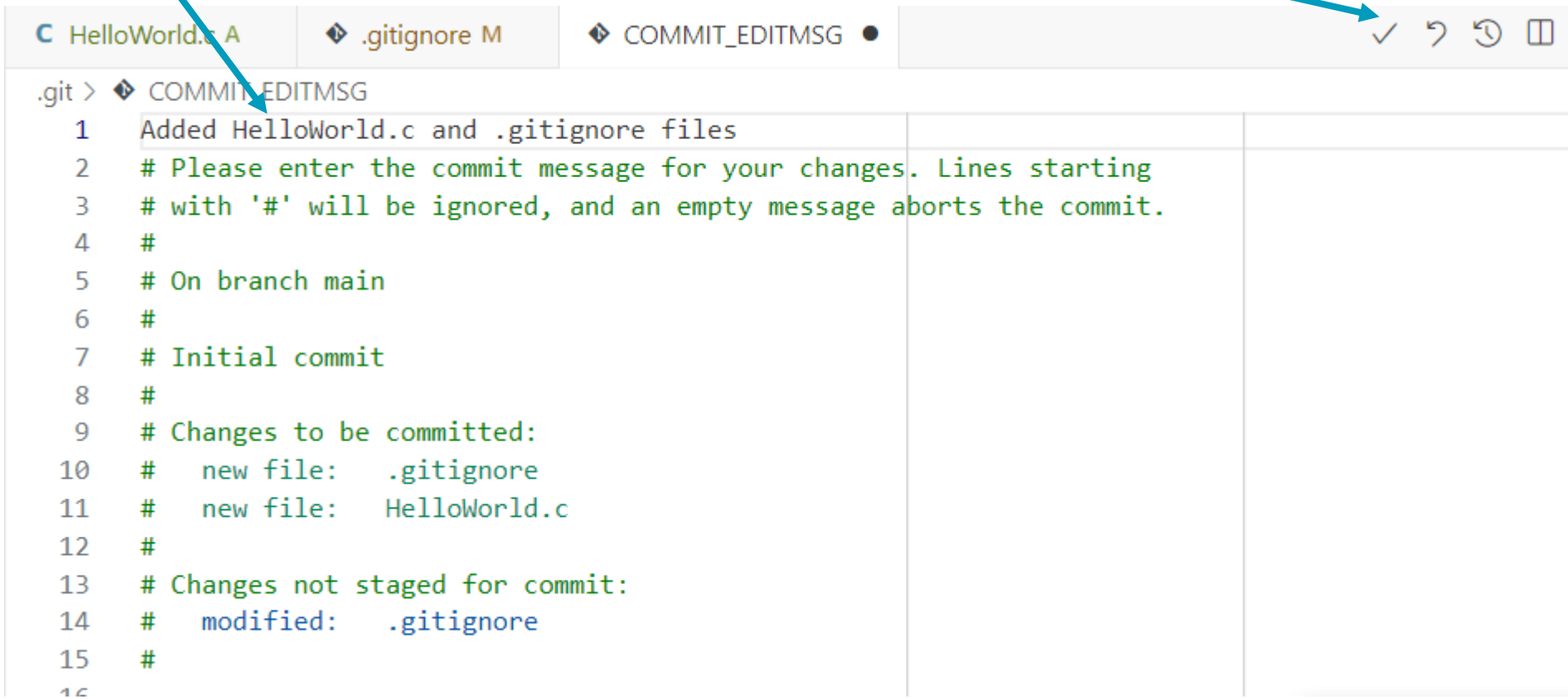
- Click the Commit button



Add a Commit message

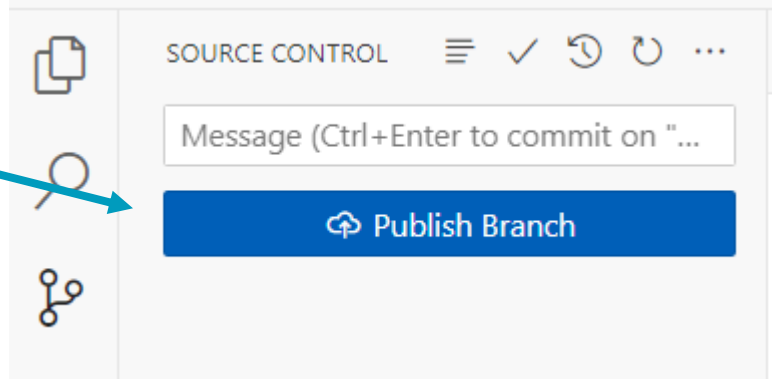
Add a meaningful message to say what was included in the commit

Then accept the changes to the message

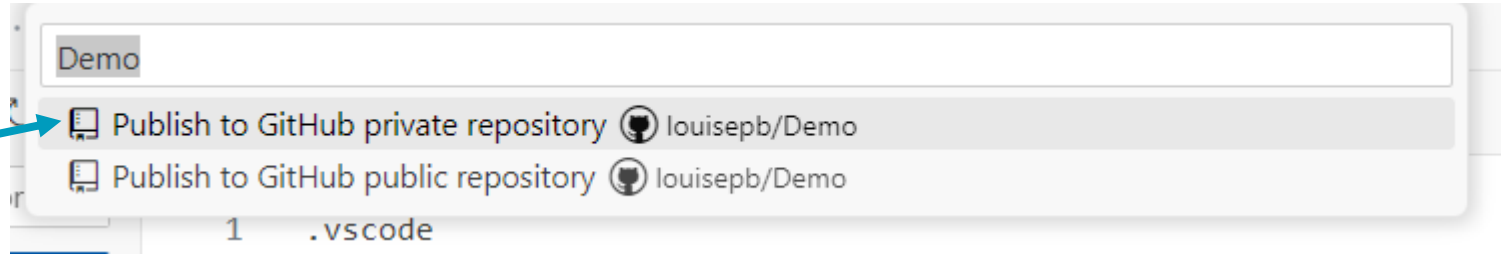


Publish to GitHub

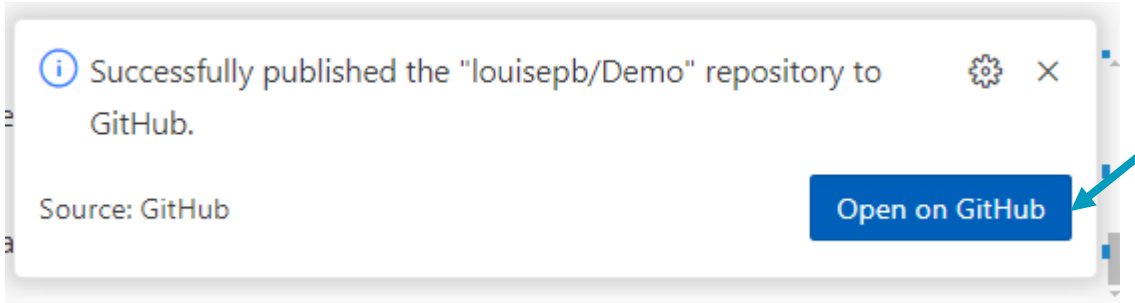
- Select Publish Branch



- Select whether to create a public or private repository on GitHub

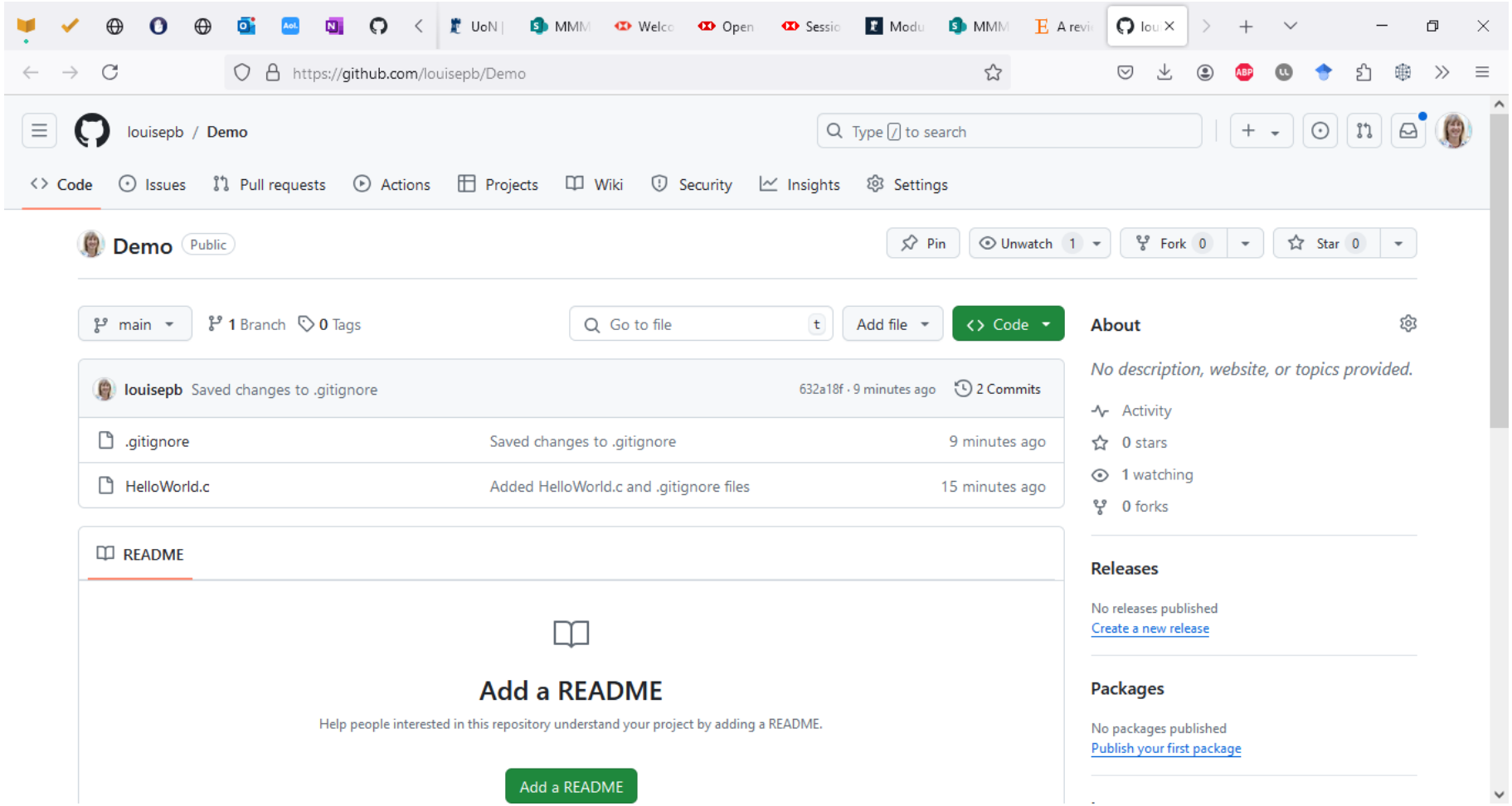


- You will probably want a private repo for your coursework!



Finally, click the button to see your repo on GitHub!

View your new repo on GitHub!



Useful git blogs

- Using VSCode and Git – useful blog
- https://kbroman.org/github_tutorial/pages/init.html - introduction to starting a new repository using the command line