Git Tutorial

Why C?

What is a Programming Language?

- Programming languages define and compile a set of instructions for the CPU (Central Processing Unit) for performing any specific task. Every programming language has a set of keywords along with syntax- that it uses for creating instructions.
- Till now, thousands of programming languages have come into form. All of them have their own specific purposes. All of these languages have a variation in terms of the level of abstraction that they all provide from the hardware. A few of these languages provide less or no abstraction at all, while the others provide a very high abstraction. On the basis of this level of abstraction, there are two types of programming languages:
  Low-level language
  High-level language
- The primary difference between low and high-level languages is that any programmer can understand, compile, and interpret a high-level language feasibly as compared to the machine. The machines, on the other hand, are capable of understanding the low-level language more feasibly compared to human beings.

What are High-Level Languages?

- One can easily interpret and combine these languages as compared to the low-level languages.
- They are very easy to understand.
- Such languages are programmer-friendly.
- Debugging is not very difficult.
- They come with easy maintenance and are thus simple and manageable.
- One can easily run them on different platforms.
- They require a compiler/interpreter for translation into a machine code.
- A user can port them from one location to another.
- Such languages have a low efficiency of memory. So, it consumes more memory than the low-level languages.
- They are very widely used and popular in today’s times.

- Java, C, C++, Python, etc., are a few examples of high-level languages.

What are Low-Level Languages?

- They are also called machine-level languages.
- Machines can easily understand it.
- High-level languages are very machine-friendly.
- Debugging them is very difficult.
- They are not very easy to understand.
- All the languages come with complex maintenance.
- They are not portable.
- These languages depend on machines. Thus, one can run it on various platforms.
- They always require assemblers for translating instructions.
- Low-level languages do not have a very wide application in today's times.

Low Level languages

- Assembly
- Machine language

High level languages

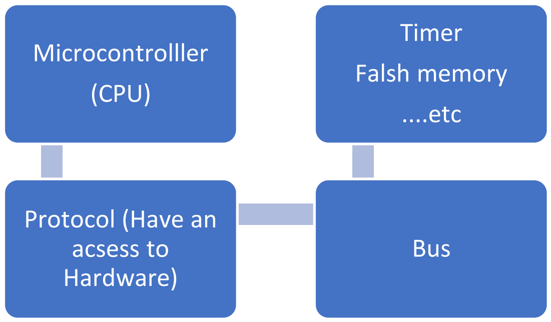
- Python
- C - Language
- C++
- Java..etc

- The primary design of C is to produce portable code while maintaining performance and minimizing footprint (CPU time, memory, disk I/O, etc.). This is useful for operating systems, embedded systems or other programs where performance matters a lot ("high-level" interface would affect performance)
- One powerful reason is memory allocation. Unlike most programming languages, C allows the programmer to write directly to memory.
- C gives control over the memory layout of data structures.
- Moreover, dynamic memory allocation is under the control of the programmer (which also means that memory deallocation has to be done by the programmer).

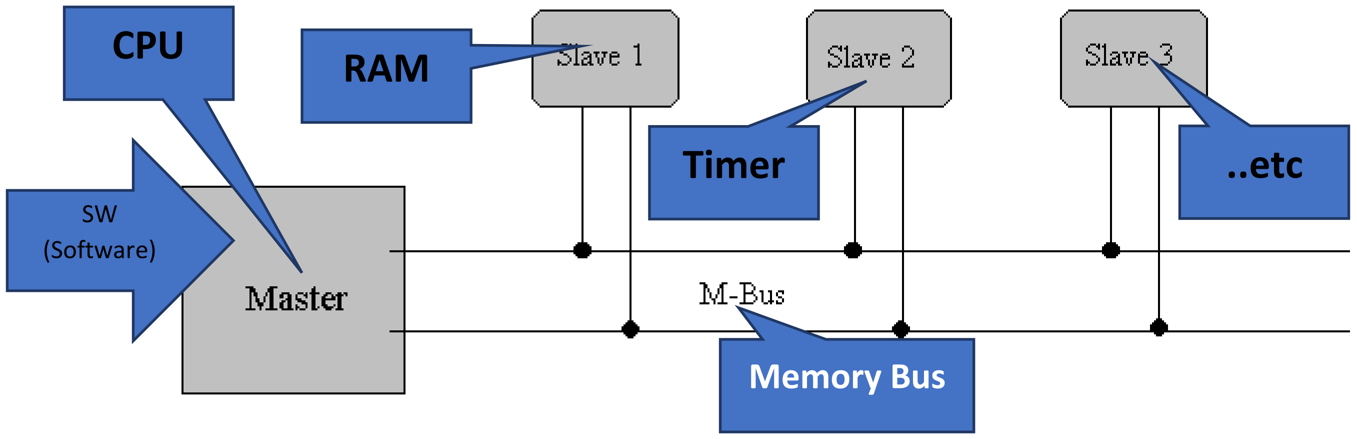
Differences between interpreters and compilers

- We generally write a computer program using a high-level language. A high-level language is one that is understandable by us, humans. This is called source code.
- However, a computer does not understand high-level language. It only understands the program written in 0's and 1's in binary, called the machine code.
- To convert source code into machine code, we use either a compiler or an interpreter.
- Both compilers and interpreters are used to convert a program written in a high-level language into machine code understood by computers. However, there are differences between how an interpreter and a compiler works.

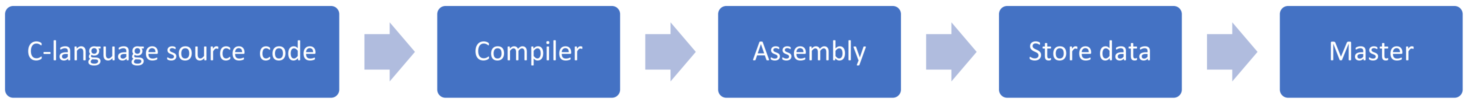
Interpreter	Compiler
• Translates program one statement at a time.	• Scans the entire program and translates it as a whole into machine code.
• Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers.	• Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters.
• No Object Code is generated, hence are memory efficient.	• Generates Object Code which further requires linking, hence requires more memory.
• Programming languages like JavaScript, Python, Ruby use interpreters.	• Programming languages like C, C++, Java use compilers.



-



- Only the CPU have the access (read or write) to (RAM – Timer - etc.).
- RAM or other components can't access to Timer or another component.
- SW (Software) has an access (read or write) to CPU So, if we need to access to RAM,



we will take to the CPU to get an access to take to RAM.

What Does Memory Bus Mean?

- The memory bus is a type of computer bus, usually in the form of a set of wires or conductors which connects electrical components and allow transfers of data and addresses from the main memory to the central processing unit (CPU) or a memory controller. It is part of a PC's collection of transport buses that are used for high-speed data channeling and transferring of information to and from certain parts of the system.
- To reduce time delays, newly-designed memory buses are made to directly connect to dynamic random-access memory (DRAM) chips instead of passing through different controllers.
- A memory bus is made up of two parts: the data bus and the address bus. The data bus is responsible for the transfer of information between the memory and the chipset. The wider a data bus is, the higher its performance since it can allow more data to pass through in the same amount of time; this is called data bandwidth.
- The address bus communicates with the system on where specific information can be located or stored when data either enters or leaves the memory. The speed and delays of an action made in a computer system depends greatly on the address bus

since it is the entity locating the information. Its width depicts the amount of system memory a processor can read or write into.

Memory address

- In computing, a memory address is a reference to a specific memory location used at various levels by software and hardware.
- Memory addresses are fixed-length sequences of digits conventionally displayed and manipulated as unsigned integers.
- Such numerical semantic bases itself upon features of CPU (such as the instruction pointer and incremental address registers), as well upon use of the memory like an array endorsed by various programming languages.

Physical addresses

- A digital computer's main memory consists of many memory locations. Each memory location has a physical address which is a code.
- The CPU (or other device) can use the code to access the corresponding memory location.
- Generally only system software, i.e., the BIOS, operating systems, and some specialized utility programs (e.g., memory testers), address physical memory using machine code operands or processor registers, instructing the CPU to direct a hardware device, called the memory controller, to use the memory bus or system bus, or separate control, address and data busses, to execute the program's commands.
- The memory controllers' bus consists of a number of parallel lines, each represented by a binary digit (bit). The width of the bus, and thus the number of addressable storage units, and the number of bits in each unit, varies among computers.

Logical addresses

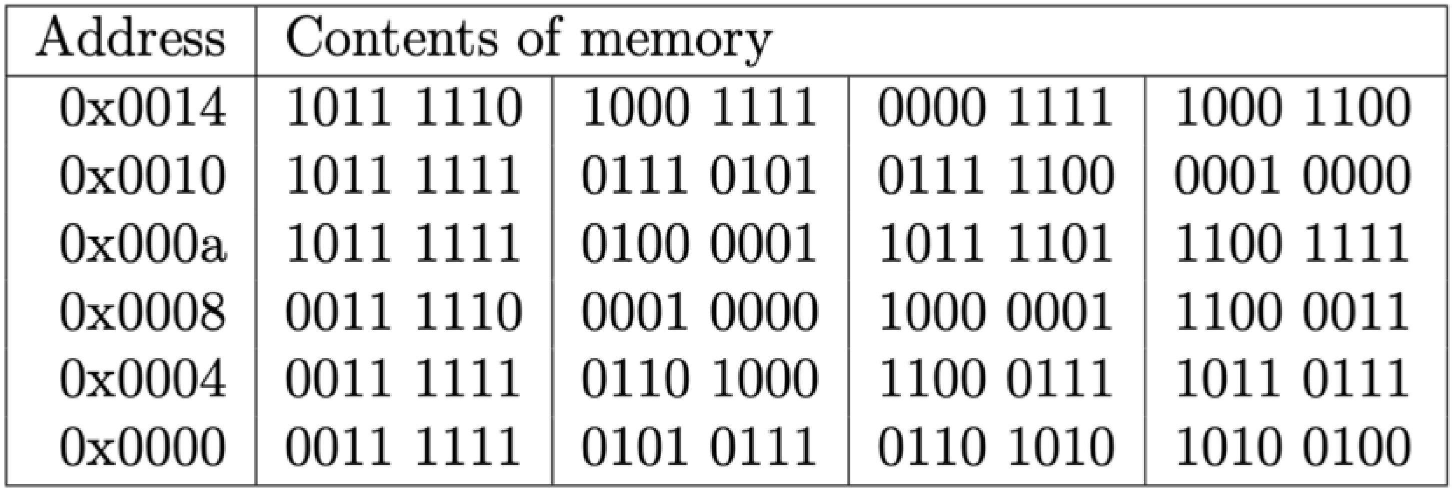
- A computer program uses memory addresses to execute machine code, and to store and retrieve data.
- In early computers logical and physical addresses corresponded, but since the introduction of virtual memory most application programs do not have a knowledge of physical addresses. Rather, they address logical addresses, or virtual addresses, using the computer's memory management unit and operating system memory mapping.

Unit of address resolution

- Most modern computers are byte-addressable.
- Each address identifies a single byte (eight bits) of storage.
- Data larger than a single byte may be stored in a sequence of consecutive addresses.
- There exist word-addressable computers, where the minimal addressable storage unit is exactly the processor's word. For example, the Data General Nova minicomputer, and the Texas Instruments TMS9900 and National Semiconductor IMP-16 microcomputers used 16-bit words, and there were many 36-bit mainframe computers (e.g., PDP-10) which used 18-bit word addressing, not byte addressing, giving an address space of 218 36-bit words, approximately 1 megabyte of storage.
- The efficiency of addressing of memory depends on the bit size of the bus used for addresses – the more bits used, the more addresses are available to the computer. For example, an 8-bit-byte-addressable machine with a 20-bit address bus (e.g., Intel 8086) can address 220 (1,048,576) memory locations, or one MiB of memory, while a 32-bit bus (e.g., Intel 80386) addresses 232 (4,294,967,296) locations, or a 4 GiB address space. In contrast, a 36-bit word-addressable machine with an 18-bit address bus addresses only 218 (262,144) 36-bit locations (9,437,184 bits), equivalent to 1,179,648 8-bit bytes, or 1152 KiB, or 1.125 MiB — slightly more than the 8086.
- Some older computers (decimal computers), were decimal digit-addressable.
- For example, each address in the IBM 1620's magnetic-core memory identified a single six bit binary-coded decimal digit, consisting of a parity bit, flag bit and four numerical bits. The 1620 used 5-digit decimal addresses, so in theory the highest possible address was 99,999. In practice, the CPU supported 20,000 memory locations, and up to two optional external memory units could be added, each supporting 20,000 addresses, for a total of 60,000 (00000–59999).

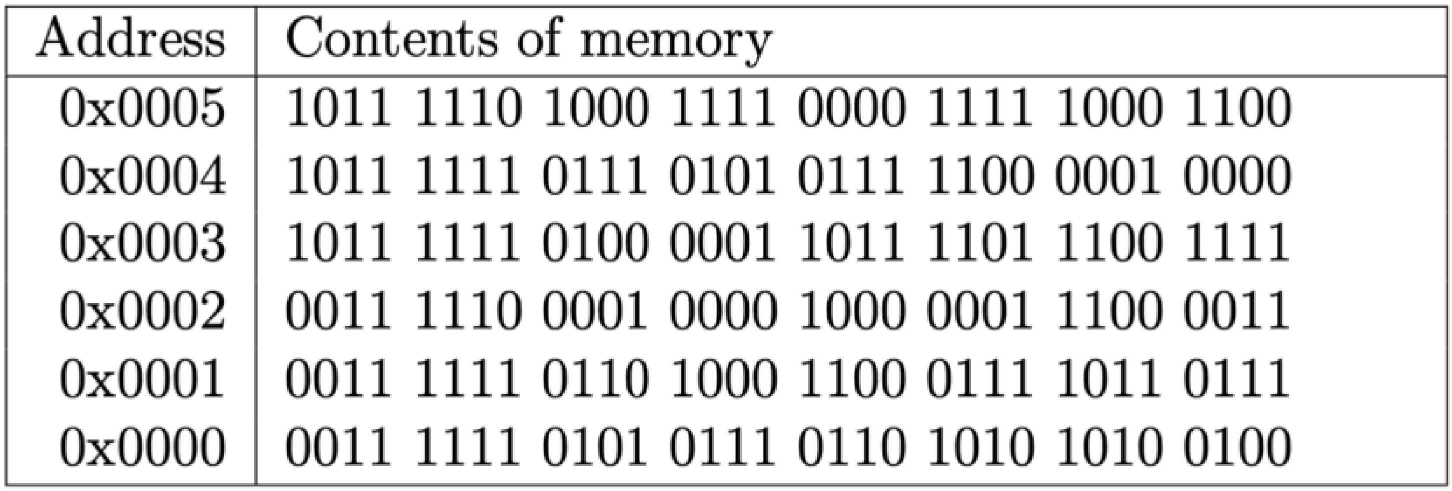
Word addressing

- In computer architecture, word addressing means that addresses of memory on a computer uniquely identify words of memory. It is usually used in contrast with byte addressing, where addresses uniquely identify bytes. Almost all modern computer architectures use byte addressing, and word addressing is largely only of historical interest. A computer that uses word addressing is sometimes called a word machine.



Address	Contents of memory			
0x0014	1011 1110	1000 1111	0000 1111	1000 1100
0x0010	1011 1111	0111 0101	0111 1100	0001 0000
0x000a	1011 1111	0100 0001	1011 1101	1100 1111
0x0008	0011 1110	0001 0000	1000 0001	1100 0011
0x0004	0011 1111	0110 1000	1100 0111	1011 0111
0x0000	0011 1111	0101 0111	0110 1010	1010 0100

24 bytes of memory using 16-bit addresses
and byte addressing



Address	Contents of memory
0x0005	1011 1110 1000 1111 0000 1111 1000 1100
0x0004	1011 1111 0111 0101 0111 1100 0001 0000
0x0003	1011 1111 0100 0001 1011 1101 1100 1111
0x0002	0011 1110 0001 0000 1000 0001 1100 0011
0x0001	0011 1111 0110 1000 1100 0111 1011 0111
0x0000	0011 1111 0101 0111 0110 1010 1010 0100

24 bytes of memory using 16-bit addresses
and 32-bit word addressing

تعريف المؤشرPointer

المؤشر أو الـ Pointer في لغة البرمجةC - هو عبارة عن متغير تحتوي قيمته على عنوان متغير آخر في الذاكرة، ومن الاسم فهو يشير إلى عنوان متغير آخر حجم المتغير من النوعPointe يكون ثابت أي كان حجم أو نوع البيانات التي يقوم بالإشارة إليها، ويتم تحديد الحجم الخاص بالمؤشر طبقا لمعمارية وحدة المعالجة المركزية وبالتبعية نظام التشغيل المستهدف، عادة ما يتم حجز مساحة 4 Byte-في المعمارية 32 bit-ويتم حجز مساحة 8 Byte-في المعمارية 64.bit-

- في لغة البرمجة C لا توجد الكلمة Pointer كنوع للبيانات كما هو الحال مع باقي الأنواع مثل int أو char، ولكن يتم إضافة رمز النجمة * مع نوع البيانات لتحديد أن هذا المتغير هو مؤشر.

مكونات الذاكرة العشوائية RAM

- تتكون الذاكرة من عدد كبير جدا من الوحدات القابلة لتخزين البيانات الثنائية (إما صفر أو واحد) تسمى هذه الوحدات بـ بت أو bit، ونظرا لصغر حجم هذه الوحدة يتم التعامل مع الذاكرة بوحدة أكبر قليلا تسمى بـ بايت أو Byte وهي مكونة من 8 بت، وتعد وحدة البايت هي أصغر وحدة يمكن حجزها لتخزين المتغيرات في أجهزة الحاسوب.
- حتى يتمكن جهاز الحاسوب من التعامل مع الذاكرة يتم تقسيم الذاكرة المتاحة في الجهاز إلى وحدات بايت، ويتم ترقيم كل وحدة برقم مسلسل يبدأ بـ صفر، ويسمى هذا الرقم بعنوان وحدة الذاكرة أو الـ Memory Address، فإذا كان جهاز الحاسوب يحتوي على ذاكرة بقدرة تخزينية واحد كيلو بايت KB1 (وهي قدرة صغيرة جدا فقط لتسهيل الفكرة) في هذه الحالة سوف تحتوي الذاكرة على 1024 بايت بما يعني وجود 1024 عنوان للذاكرة، بحيث يمثل العنوان صفر أول وحدة في الذاكرة والعنوان 1023 آخر عنوان في الذاكرة.
- تقيد معمارية الحاسوب عدد عناوين الذاكرة التي يمكن التعامل معها، على سبيل المثال المعمارية 32-bit يمكنها التعامل مع عناوين للذاكرة تصل إلى 4,294,967,296 عنوان وهي تساوي 4 جيجا بايت من الذاكرة، هذا العدد من العناوين يمكن تمثيله أو تخزينه في 32 بت أو 4 بايت وهو حجم المتغير من النوع Pointer في هذه المعمارية.
- يعتبر العنوان صفر في الذاكرة ذو معنى خاص، حيث يتم حجزه بواسطة نظام التشغيل وعادة ما يشار إليه بالقيمة NULL ويفضل أن يتم تخصيص هذه القيمة أثناء الإعلان عن المؤشرات.

أهمية استخدام المؤشرات في لغة البرمجة C

- تمتاز لغة البرمجة C بقدرتها على التعامل مع الذاكرة بشكل مباشر ما يتيح إنشاء برمجيات ذات كفاءة عالية في استخدام الذاكرة، ويرجع الفضل بشكل كبير إلى المؤشرات، حيث يمكن أن يتم التعامل مع أي متغير أي كان حجمه بمجرد معرفة عنوانه.
- وتستخدم المؤشرات بشكل كبير للتعامل مع المصفوفات، حيث تعتبر المصفوفة نوع من أنواع المؤشرات وبالتالي يتم استخدام العمليات الحسابية على المؤشرات (Pointer Arithmetic) للتنقل بين عناصر المصفوفة كما سنتعرف عليه لاحقا.
- يتم استخدام المؤشرات لتمرير المتغيرات ذات الحجم الكبير بين الدوال، فعند تمرير أي متغير لأي دالة عادة ما يتم إعادة نسخ المتغير في المساحة المخصصة من الذاكرة للدالة (باستثناء المصفوفات) وتسبب هذه العملية إهدار للوقت المستغرق في عملية النسخ وإهدار للذاكرة المتاحة حيث يتم نسخ المتغير والاحتفاظ بالمتغير الأصلي أي أن المتغير يوجد في الذاكرة مرتين، وتعرف عملية تمرير المؤشرات للدوال بالتمرير عن طريق المرجع أو العنوان Pass By Reference.
- تعتبر المؤشرات موضوع حيوي وأساسي عند التحدث عن هياكل البيانات والتي تعرف بـ Data Structures، حيث تستخدم المؤشرات للربط بين وحدات البيانات أو كما تعرف بـ Nodes.

- تستخدم المؤشرات لإنشاء متغيرات ديناميكية أو كما تعرف بـ Dynamic-Variables والتي تمكن المبرمج من إنشاء متغيرات أثناء تشغيل البرنامج عند الحاجة إليها، أو إنشاء مصفوفات بالحجم الذي يحدده مستخدم البرنامج، فكما تعلمنا أن المصفوفات في لغة البرمجة C يجب أن يتم تحديد عدد عناصرها بشكل ثابت أثناء إنشاؤها، ولكن باستخدام المؤشرات يمكنك إنشاء مصفوفات بأي حجم يحدده المستخدم أثناء التشغيل وهو ما يعرف بـ Dynamic Arrays.

Memory-mapped hardware

- On some computing architectures, pointers can be used to directly manipulate memory or memory-mapped devices.
- Assigning addresses to pointers is an invaluable tool when programming microcontrollers.

Standards Organizations

- International Electrical and Electronics Engineers (IEEE)
- International Organization for Standardization (ISO)
- American National Standards Institute (ANSI)

ANSI C

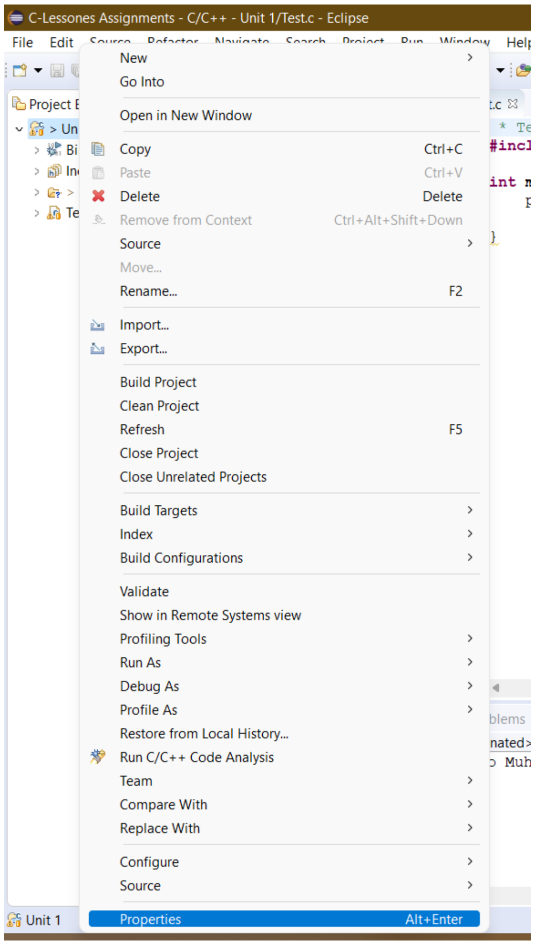
- ISO C, and Standard Care successive standards for the C programming language published by the American National Standards Institute (ANSI) and ISO/IEC JTC 1/SC 22/WG 14 of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). Historically, the names referred specifically to the original and best-supported version of the standard (known as C89 or C90). Software developers writing in C are encouraged to conform to the standards, as doing so helps portability between compilers.

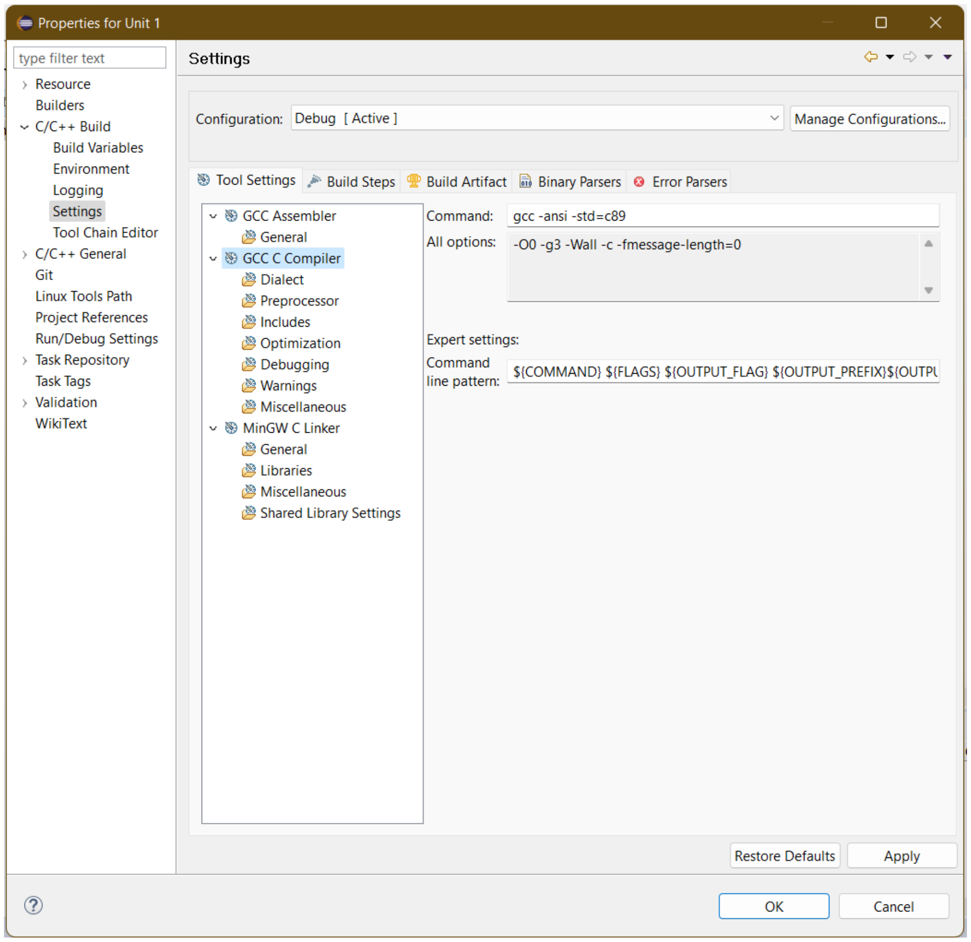
ANSI C Standards

- C89
- C95
- C99

Using ANSI C89 on Eclipse IDE

- Open Eclipse IDE
- Then open your project, (Right-click) Properties
- Then C/C++ Build, GCC Compiler
- At Command write (gcc -ansi -std=c89) to get the ANSI C89 compiler





Configuration: Debug [Active]

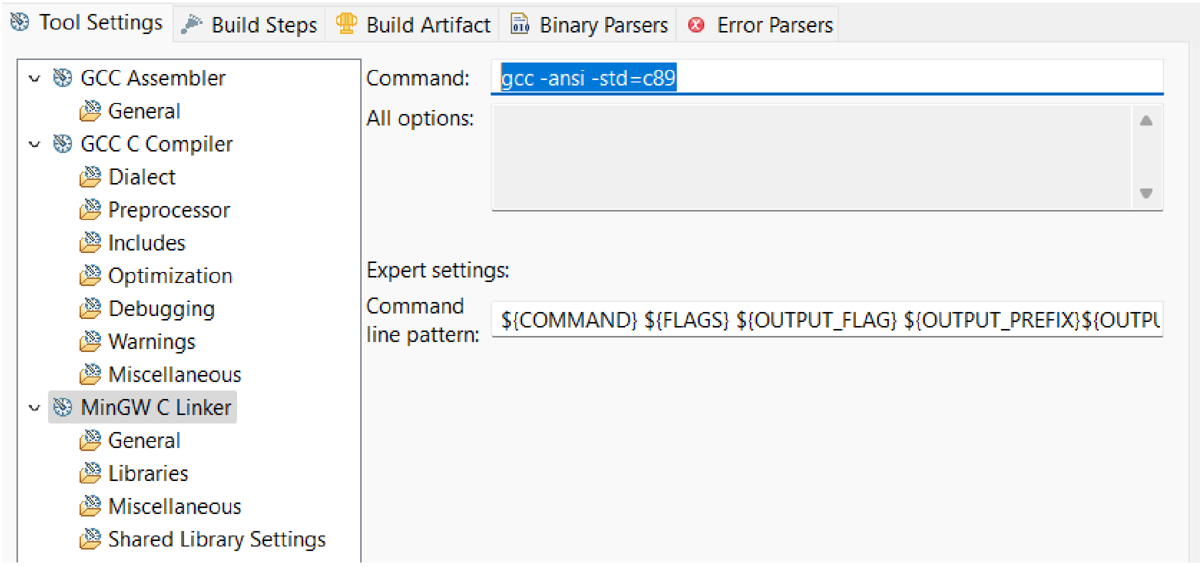
Command: gcc -ansi -std=c89

All options: -O0 -g3 -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPU

- copy that and past it at MinGW C Linker



Command: gcc -ansi -std=c89

All options:

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPU

- Now you are using ANSI C89 on your Eclipse IDE.

أي أنه يتحكم في الversions

Version control systems (VCS)

- Software package that allows users to track changes

Many different types

- Concurrent Versions System (CVS)
- Subversion (SVN)
- Git

VCS Features

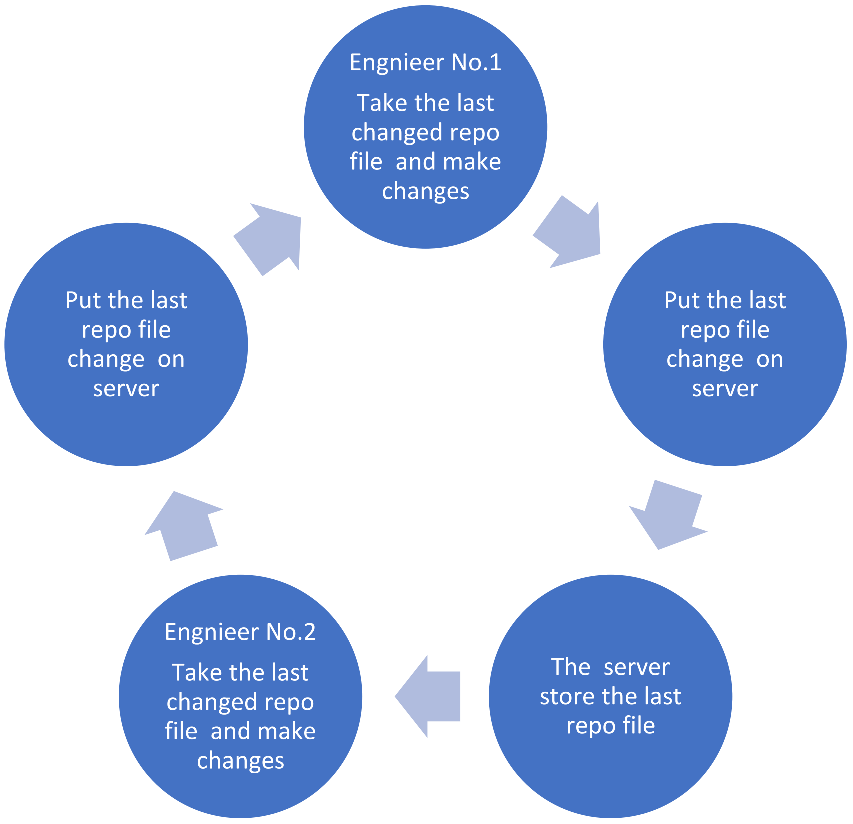
- Allows you to track:

1- Software
2- Documents
3- Build information
4- Software configuration information

- Repository: Collection of tracked files

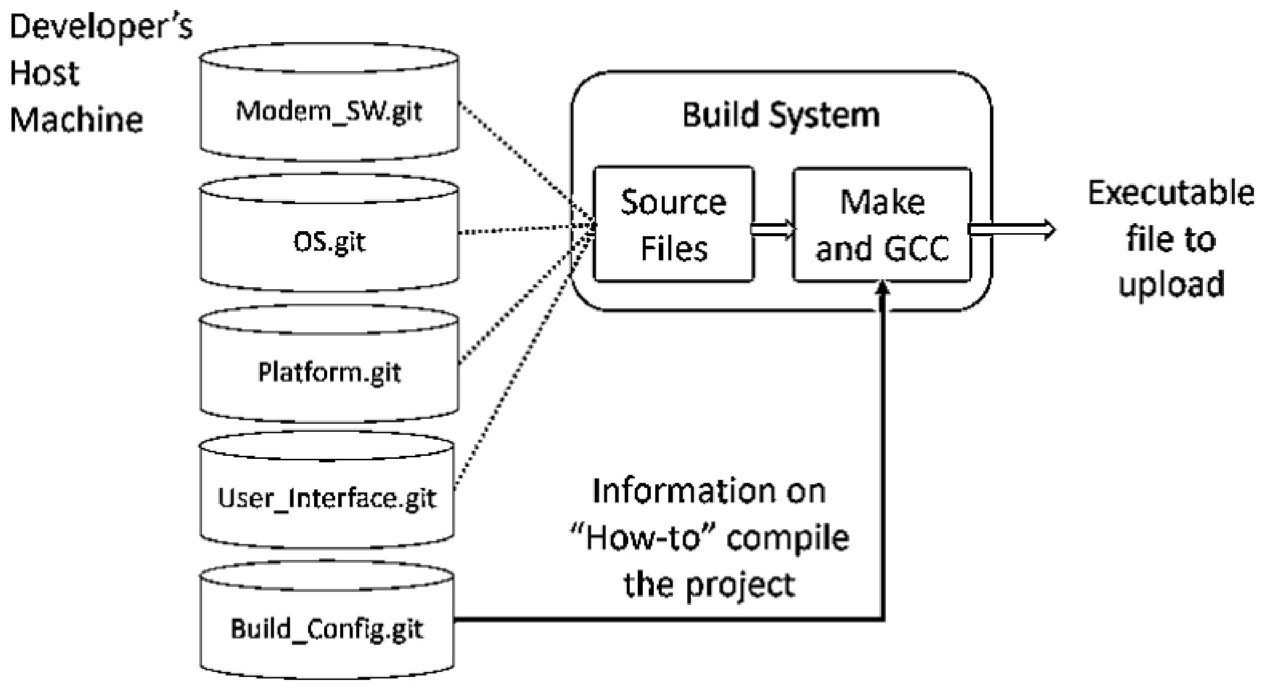
1- Acts like a normal file system

Example of Version control systems (VCS)


Engnieer No.1
Take the last changed repo file and make changes
Put the last repo file change on server
The server store the last repo file
Engnieer No.2
Take the last changed repo file and make changes
Put the last repo file change on server

Multiple Repositories (Repos)

- You can have many repositories contribute to a single software product



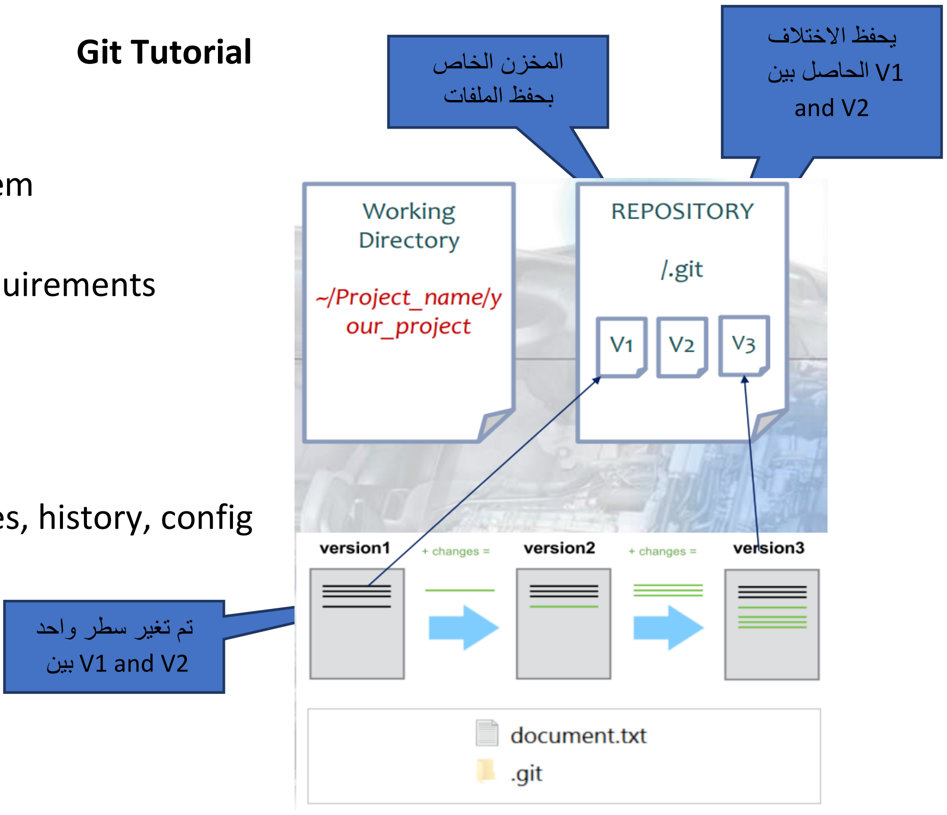
Git Tutorial

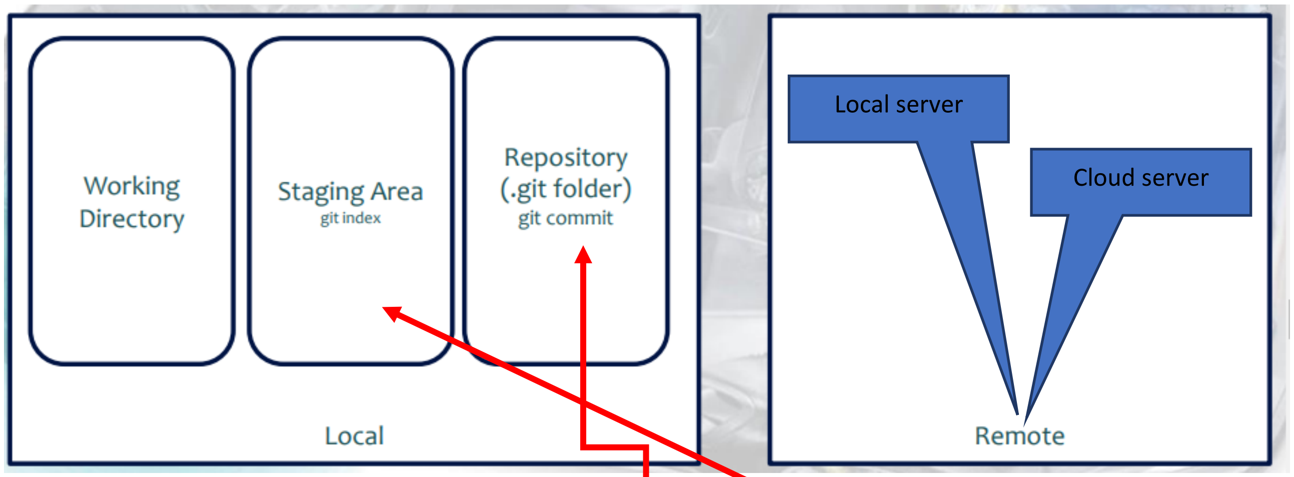
What is Git?

- Distributed source control system
- Open Source
- Developed for Linux project requirements
- Very Fast
- Active community

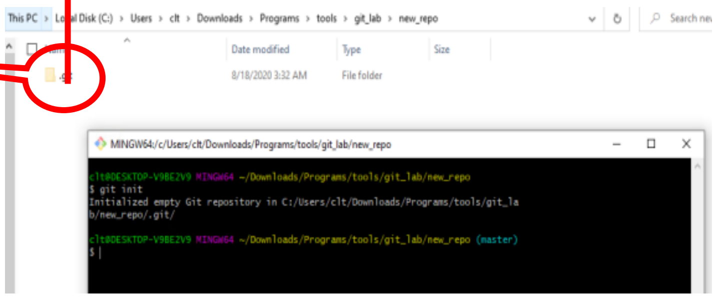
Git Concepts

- The Git Repository: contains files, history, config managed by git.


المخزن الخاص بحفظ الملفات
يحفظ الاختلاف الحاصل بين V1 and V2
Working Directory
~/Project_name/your_project
REPOSITORY
/.git
V1
V2
V3
version1
+ changes =
version2
+ changes =
version3
تم تغير سطر واحد بين V1 and V2
document.txt
.git



- By command (git add) you can add fill to staging area to git the git index.
- By command (git commit) to put it in (git. Folder).
- By command (git push) to git the all changes in file to push it on server.


This PC › Local Disk (C:) › Users › clt › Downloads › Programs › tools › git_lab › new_repo
Search new
Date modified
Type
Size
.git
8/18/2020 3:32 AM
File folder
MINGW64:/c/Users/clt/Downloads/Programs/tools/git_lab/new_repo

```
clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/new_repo
$ git init
Initialized empty Git repository in C:/Users/clt/Downloads/Programs/tools/git_la
b/new_repo/.git/

clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/new_repo (master)
$ |
```

Create local git repository

- Create New folder on your computer at your own git repo path.
- Open the folder then create 2 new folders
1- Repo_ProjectA (the main file of project)
2- Working_dr (local pc that work on main project)
- Open Repo_ProjectA folder then (left-click in your mouse) click on (Git Bash here)
- Command (git init) -- to make this folder as a repository folder
- There is a git folder will be created on this folder after command -- .git folder and this is the folder of repo, it will store any versions of main project on it
- To push every changes on git you will need to command (git config core.bare true) without this command you can't push your file you can just read it – its allow you to get a pull or push file
- Go back and open the Working_dr folder (we need here to get a copy (clone) form working main file)
- Copy the folder path
- Then (left-click in your mouse) click on (Git Bash here) then in command write (git clone/'the coped path')

You can drag your file in command screen directly

Note that:

when using a Local server, your command will be

- git clone /path/to/repository

when using a remote server, your command will be

- git clone username@host:/path/to/repository

when using a online server, your command will be

- git clone (URL of git file)/path/to/repository

Definition

No	Commands	To do
1	**Repository**	**المكان الذي يحفظ فيه ملفات الاكواد الخاصة بالمشروع**
2	**Branch**	**هو فرع من ال repository** **أي أنك تأخذ شيء من ال repositoryللتعديل عليه ثم ارجاعه**
3	**Local repository**	**هو المستودع الخاص الذي تضع فيه ملفاتك على الجهاز**
4	**Remote repository**	**هو المستودع الخاص الذي تضع فيه ملفاتك على server**
5	**Commit**	**Snapshot or checkpoint in your local repo** **أي نقطة معينة وصلت اليها ولكن لم يتم ارسالها بعد كتعديل داخل ال repo**
6	**Clone**	**استنساخ من ال local الى local or repo**
7	**Push**	**Upload local changes to remote** **وضع او ارسال التعديلات التي قمت بها على الملف الذي عندك على الجهاز الى ال server (remote repo)**
8	**Pull**	**You pull changes form remote repo to your local** **سحب التعديلات من ال remote repo to your local**
9	**Pull request**	**Tell other about your changes to pull it from local to remote** **سحب الملف من ال local لل remote repo لكن قبل وضعه على ال server يصل الى Coordinator منك طلب موافقة لإرسال الملف وذلك حتى يراجعه قبل وضعه على ال server**

Add file in repo

1- (left-click in your mouse) click on (Git Bash here) then in command write (git add (file name with extension).
2- Command (git commit -n “Your massage” (file with extension)).
3- Command (git push) to put it on remote repo. –to be appear to everyone

To git the last file on remote repo

1- (Left-click in your mouse) click on (Git Bash here) then in command write (git pull) to know if there is any changed happened on remote repo or not
   If there is any changes happened it will tell you, and if there is no thing happened it will tell you “You are up to date”.
2- Command (ls) to see every files there.
3- You can Command (cat (file name with extension)) to open file inside the command screen.
4- If you want to make an edit on file you can command (vi (file name with extension)) and it will open on command screen to change it.

- After save the changes you should push it on remote repo to make it appear with changes for everyone but your file already on the remote repo so,
  You will command (git commit -m “Your massage” (file with extension)).
- Then Command (git push) to put it on remote repo. –to be appear to everyone

Working flow

your local repository consists of three "trees" maintained by git.

- the first one is your Working Directory which holds the actual files.
- the second one is the Index which acts as a staging area and

  Git add
- finally, the HEAD which points to the last commit you've made.
- Your changes are now in the HEAD of your local working copy. To send those changes to your remote repository, execute
- git push origin master
  if you have not cloned an existing repository and want to connect your repository to a remote server, you need to add it with
- git remote add origin <server>
  Now you are able to push your changes to the selected remote server

بعض الأساسيات

1- لا تقوم بوضع جميع المشاريع داخل repoواحد كل مشروع له ال repoالخاص به
2- كل branch لكل تعديل خاص به
3- لست في حاجة دائمة بأن تكون دائما على اتصال بال remote repo
4- أي يشخص يمكنه عمل push or pull على حسب Permissions

Branching

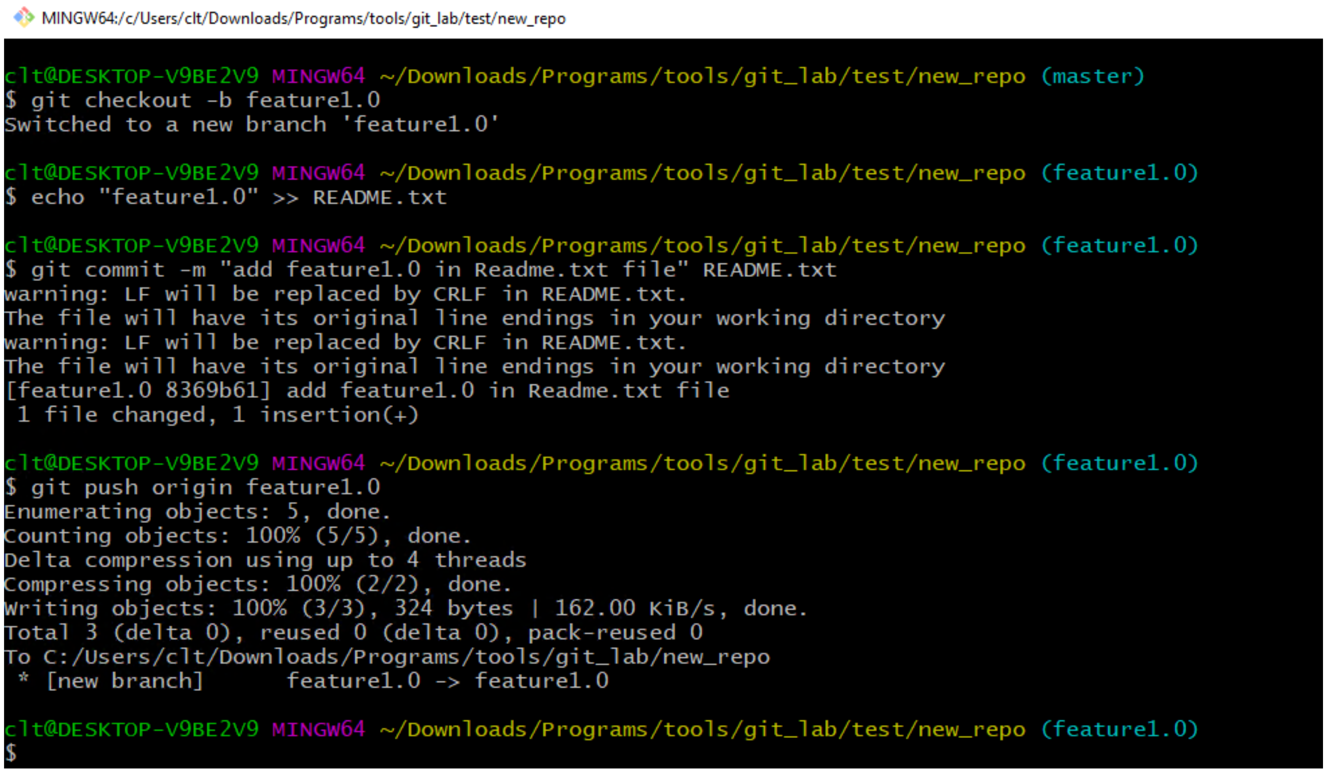
- Development phase

هي المراحل التي يمر بها المشروع من تعديلات واضافات

- ال branching هو عمل snapshoot (أي انه يأخذ نسخة عند وقت معين) من المشروع عند وقت معين (مثال نسخة نهائية للبرنامج لسنة 2020)
- في حال مرور الوقت ان كنا بحاجة الى الرجوع الى هذه النسخة يمكننا تحويل ال Git repo من ال master الى ال branching في تلك الحالة سيتم العمل على ال branching بدل من ال master
- عند الانتهاء من التعديلات والعمل على ال Branch file يمكن الرجوع مرة أخرى الى ال master file
- أيضا يمكن عمل merge لل branching with master

Create New branch file

- Command (Git checkout -b (the name of branch server)).



MINGW64:/c/Users/clt/Downloads/Programs/tools/git_lab/test/new_repo

```
clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/test/new_repo (master)
$ git checkout -b feature1.0
Switched to a new branch 'feature1.0'

clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/test/new_repo (feature1.0)
$ echo "feature1.0" >> README.txt

clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/test/new_repo (feature1.0)
$ git commit -m "add feature1.0 in Readme.txt file" README.txt
warning: LF will be replaced by CRLF in README.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in README.txt.
The file will have its original line endings in your working directory
[feature1.0 8369b61] add feature1.0 in Readme.txt file
 1 file changed, 1 insertion(+)

clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/test/new_repo (feature1.0)
$ git push origin feature1.0
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 324 bytes | 162.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To C:/Users/clt/Downloads/Programs/tools/git_lab/new_repo
 * [new branch]      feature1.0 -> feature1.0

clt@DESKTOP-V9BE2V9 MINGW64 ~/Downloads/Programs/tools/git_lab/test/new_repo (feature1.0)
$
```

Update and merge

to update your local repository to the newest commit, execute

git pull

in your working directory to fetch and merge remote changes.

- to merge another branch into your active branch (e.g., master), use

git merge <branch>

لدمج ال branch مع ال master (سيأخذ نسخة من ال branch يضعها داخل ال master للعمل بها)

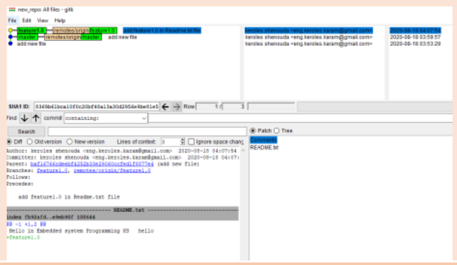
- in both cases git tries to auto-merge changes. Unfortunately, this is not always possible and results
  git add <filename>
- before merging changes, you can also preview them by using
  git diff <source_branch> <target_branch>

Branch (server name)

Master

لمشاهدة الاختلافات بين ال master file and branch file

Commands part 1

No	Commands	Main	To do
1	**Pwd**	**Print working directory**	**لمعرفة مكان العمل المفتوح حالياً**
2	**Git init**	**Initials git repository in this direction**	**قم بعمل مخزن لي في هذا المكان**
3	**git config core.bare true**		**يعطي لك السماحية بوضع الملفات على ال git repository**
4	**Git clone (Path)**		**لعمل استنساخ للملف الأصلي داخل المخزن repository**
5	**Git add**		**لعمل إضافة ملف** **الملف المضاف يجب استخدام هذا الامر له حتى يأخذ index number**
6	**Git commit -m "رسالة للفريق العمل توضح فيها ما تم عمله بد اخل الملف المرسل" (file name with extension)**	**أي ماذا اضفت n-**	**- عند عمل commitلأول مرة سيسألك عن الاسم والميل الذي تريده ان يظهر**
7	**Git push**		**لإضافة الملف الى remote repo**
8	**gitk**		**تقوم بفتح gitk gui window لإظهار كل التعديلات الحادثة وما هى**
9	**Git pull**		**الأخذ اخر نسخة محدثة من ال remote repo**
10	**ls**		**لمعرفة الملفات التي بداخل ال path الحالي**
11	**Cat (file name with extension)**		**لعرض محتويات الfile**

```
new_repo: All files - gitk
File  Edit  View  Help
feature1.0  remotes/origin/feature1.0  add feature1.0 in Readme.txt file   keroles shenouda <eng.keroles.karam@gmail.com>  2020-08-18 04:07:54
master  remotes/origin/master  add new file                             keroles shenouda <eng.keroles.karam@gmail.com>  2020-08-18 03:59:57
add new file                                                            keroles shenouda <eng.keroles.karam@gmail.com>  2020-08-18 03:53:29

SHA1 ID: 0369b61bca10f0c20bf40a13a30d2956e9be01e5   Row 1 / 3
Find  commit  containing:
Search                                              Patch  Tree
Diff  Old version  New version  Lines of context: 3  Ignore space change
Author: keroles shenouda <eng.keroles.karam@gmail.com>  2020-08-18 04:07:54     Comments
Committer: keroles shenouda <eng.keroles.karam@gmail.com>  2020-08-18 04:07:   README.txt
Parent: baf1676cdeebf4252b39e25060ccfed1f0077e4 (add new file)
Branches: feature1.0, remotes/origin/feature1.0
Follows:
Precedes:

    add feature1.0 in Readme.txt file

---------------------------------- README.txt ----------------------------------
index fb92afd..e9eb98f 100644
@@ -1 +1,2 @@
 Hello in Embedded system Programming KS   hello
+feature1.0
```

12	Vi (file name with extension)		للتعديل على الملف داخل command screen
13	Git stash		لمسح ال error
14	Git checkout -b (the name of branch server)		لعمل ملف Branch
15	Git checkout master		للتقل من ال branch الى ال master تستخدم أيضا للرجوع او الذهاب الى خطوة نعينة سابقة وذلك عن طريق فتح ال gitk أو عن طريق ال log ثم الذهاب الى نقطة الرجوع ثم نسخ ال ID الخاص بتلك النقطة ثم كتابه بدل من master
16	Git branch -d (the name of branch server)		
17	Git push origin <branch>		لعمل push للفيل على ال Branch
18	Git log		لعرض log لما حدث
19	Cp (file name with extension) (the copy path)		عمل copy

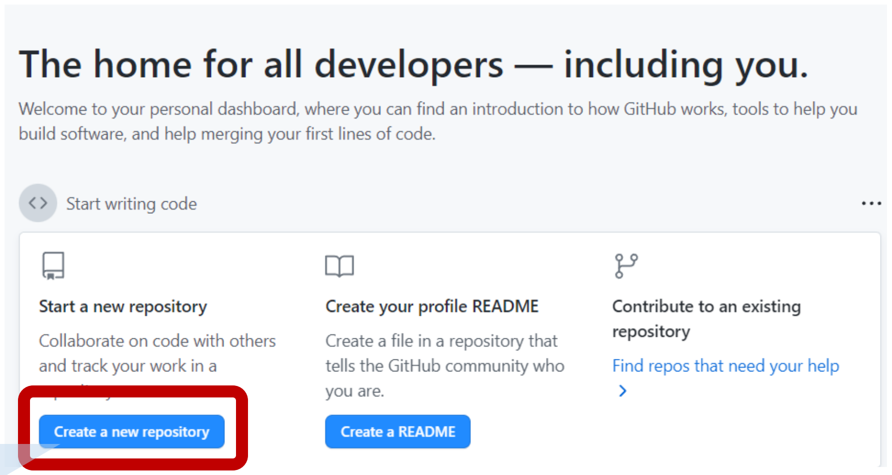
- عند الرجوع الى نقطة معينة عن طريق git checkout (ID number of check point) يتم عمل copy ال file عن طريق cp في مكان خارجي
- ثم التحويل الى ال master عن طريق git checkout master
- ثم عمل copy له مرة أخرى داخل ال master repo
- ثم git commit
- ثم git push

GitHub

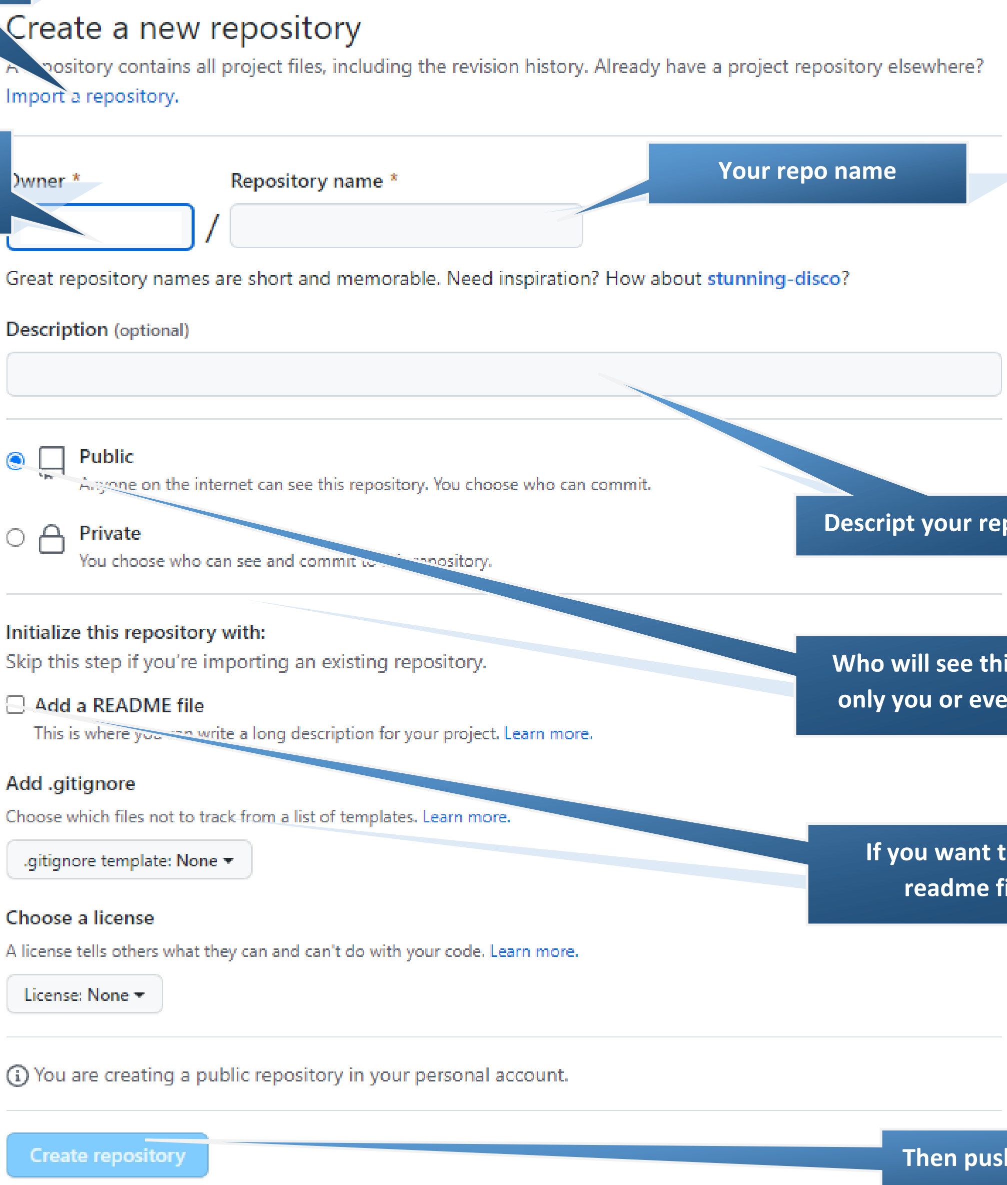
- Is a cloud server

Put your code on GitHub

- Create new repository


The home for all developers — including you.
Welcome to your personal dashboard, where you can find an introduction to how GitHub works, tools to help you build software, and help merging your first lines of code.
Start writing code
Start a new repository
Collaborate on code with others and track your work in a
Create a new repository
Create your profile README
Create a file in a repository that tells the GitHub community who you are.
Create a README
Contribute to an existing repository
Find repos that need your help

You can push your repo form you PC here


Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.
Owner *
Repository name *
Great repository names are short and memorable. Need inspiration? How about stunning-disco?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Skip this step if you're importing an existing repository.
Add a README file
This is where you can write a long description for your project. Learn more.
Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None
Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None
You are creating a public repository in your personal account.
Create repository

Your name and it will be auto selected

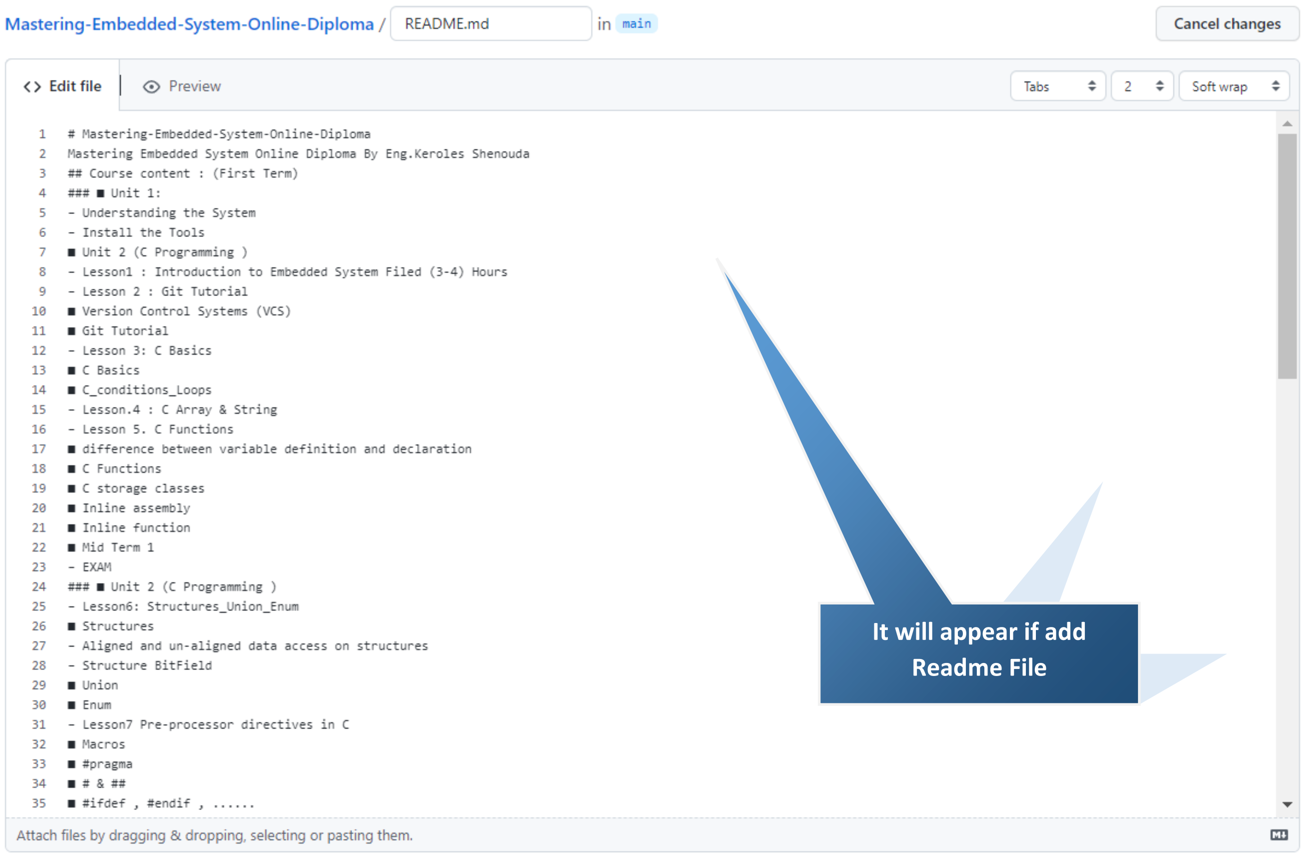
Your repo name

Descript your repo here

Who will see this repo only you or everyone

If you want to add readme file

Then push here to create your repo

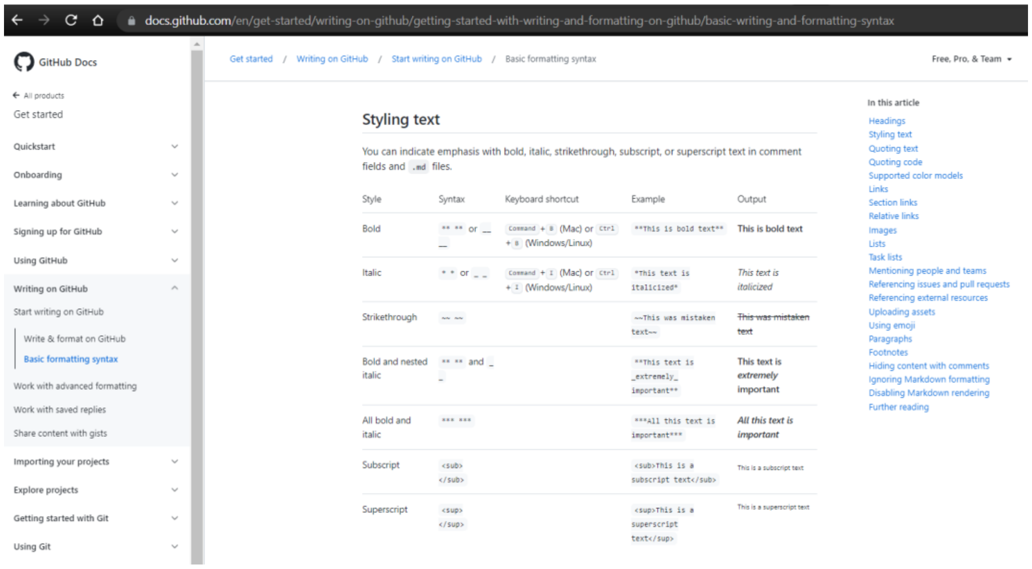

Mastering-Embedded-System-Online-Diploma / README.md in main
Cancel changes
Edit file
Preview
Tabs
2
Soft wrap
Attach files by dragging & dropping, selecting or pasting them.

```
# Mastering-Embedded-System-Online-Diploma
Mastering Embedded System Online Diploma By Eng.Keroles Shenouda
## Course content : (First Term)
### ■ Unit 1:
- Understanding the System
- Install the Tools
■ Unit 2 (C Programming )
- Lesson1 : Introduction to Embedded System Filed (3-4) Hours
- Lesson 2 : Git Tutorial
■ Version Control Systems (VCS)
■ Git Tutorial
- Lesson 3: C Basics
■ C Basics
■ C_conditions_Loops
- Lesson.4 : C Array & String
- Lesson 5. C Functions
■ difference between variable definition and declaration
■ C Functions
■ C storage classes
■ Inline assembly
■ Inline function
■ Mid Term 1
- EXAM
### ■ Unit 2 (C Programming )
- Lesson6: Structures_Union_Enum
■ Structures
- Aligned and un-aligned data access on structures
- Structure BitField
■ Union
■ Enum
- Lesson7 Pre-processor directives in C
■ Macros
■ #pragma
■ # & ##
■ #ifdef , #endif , ......
```

It will appear if add Readme File

You can change the style by this syntax at this page

https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax



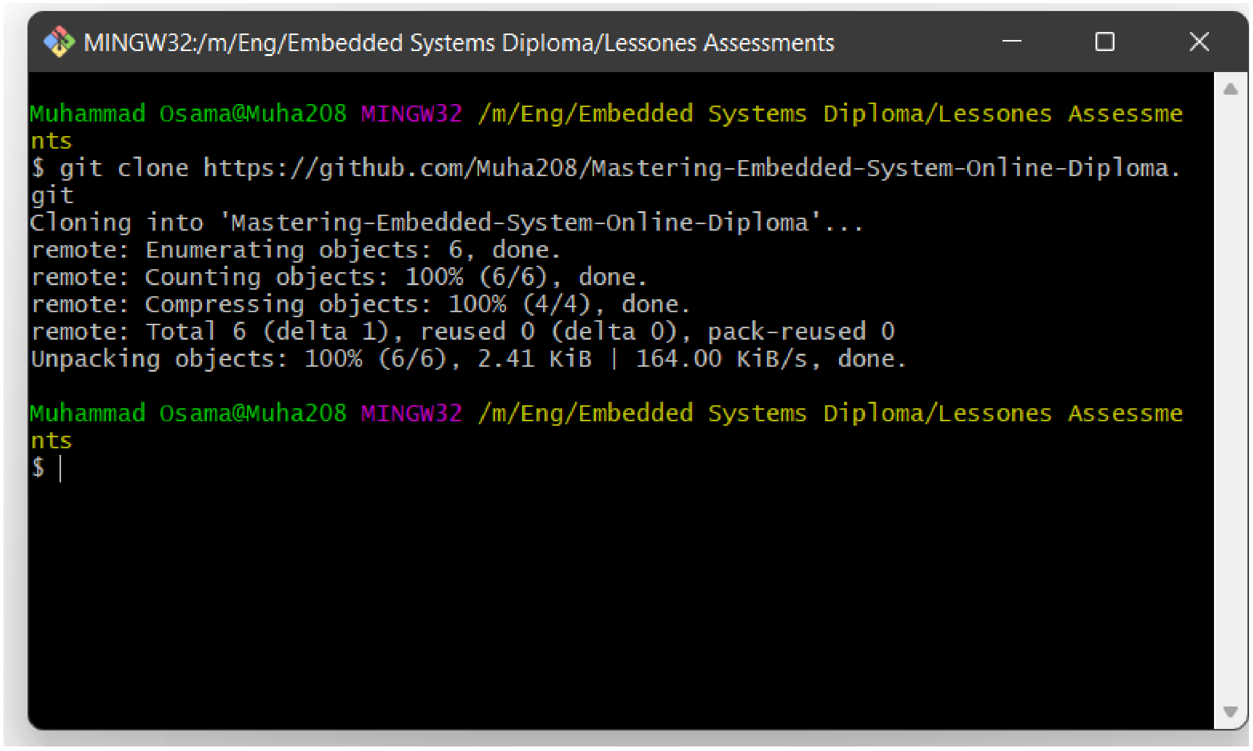
Styling text

You can indicate emphasis with bold, italic, strikethrough, subscript, or superscript text in comment fields and `.md` files.

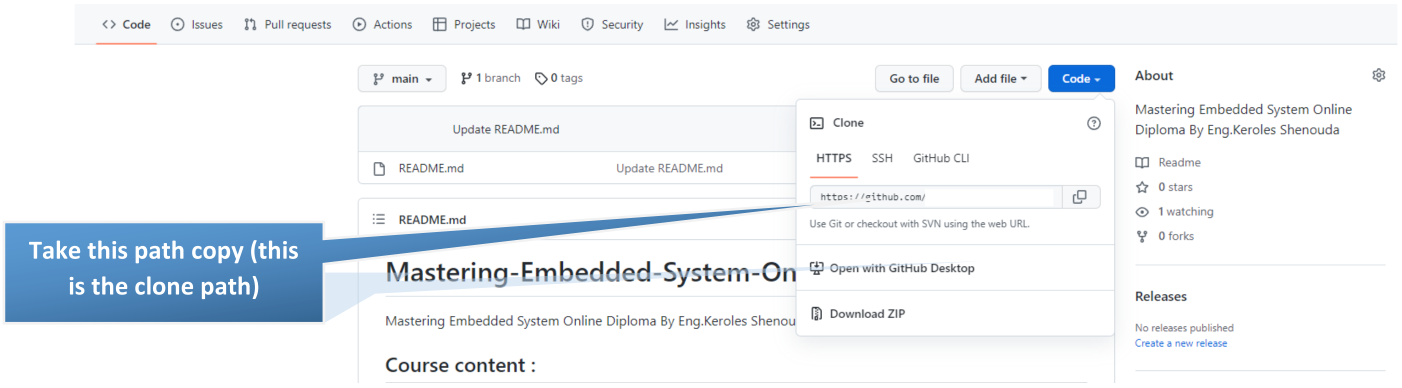
Style	Syntax	Keyboard shortcut	Example	Output
Bold	`** **` or `__ __`	`Command` + `B` (Mac) or `Ctrl` + `B` (Windows/Linux)	`**This is bold text**`	**This is bold text**
Italic	`* *` or `_ _`	`Command` + `I` (Mac) or `Ctrl` + `I` (Windows/Linux)	`*This text is italicized*`	*This text is italicized*
Strikethrough	`~~ ~~`		`~~This was mistaken text~~`	~~This was mistaken text~~
Bold and nested italic	`** **` and `_ _`		`**This text is _extremely_ important**`	**This text is _extremely_ important**
All bold and italic	`*** ***`		`***All this text is important***`	***All this text is important***
Subscript	`<sub> </sub>`		`<sub>This is a subscript text</sub>`	This is a subscript text
Superscript	`<sup> </sup>`		`<sup>This is a superscript text</sup>`	This is a superscript text

Make a clone from GitHub repo

- Go to the folder that you want to clone in it and open Git bash command screen.



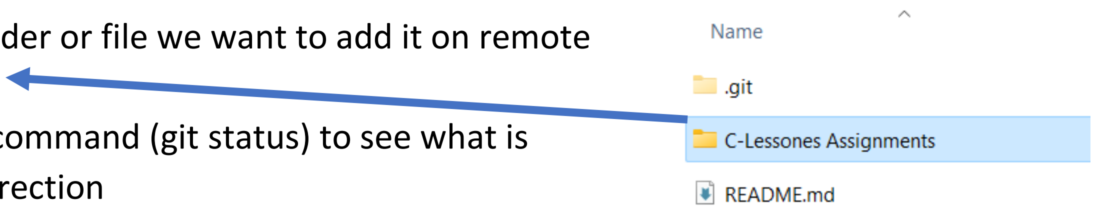
```
MINGW32:/m/Eng/Embedded Systems Diploma/Lessones Assessments

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts
$ git clone https://github.com/Muha208/Mastering-Embedded-System-Online-Diploma.
git
Cloning into 'Mastering-Embedded-System-Online-Diploma'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 2.41 KiB | 164.00 KiB/s, done.

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts
$ |
```



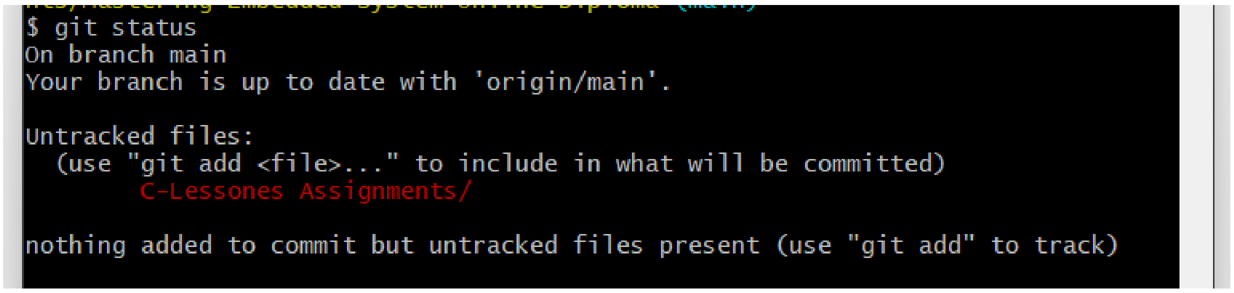
- Then command (git clone (paste the URL here)) to make a clone here
- You can command (dir) to show all file or folders in this path.

How to push your files and folders on the remote server

- First there is the folder or file we want to add it on remote repo
- Open git bash and command (git status) to see what is happened at this direction

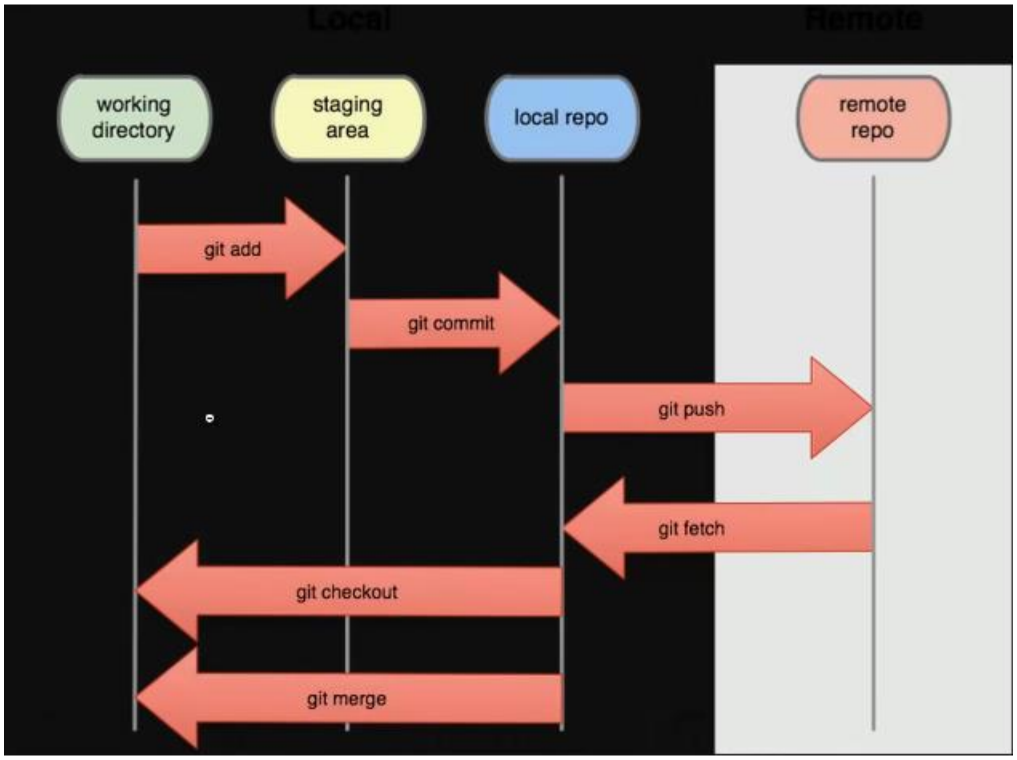


- I will make an C-language file to push it on my remote repo.

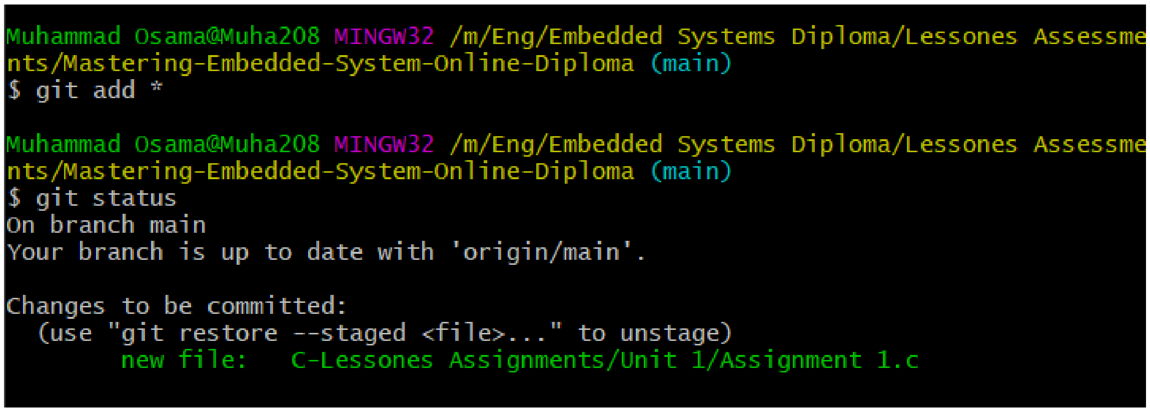


```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        C-Lessones Assignments/

nothing added to commit but untracked files present (use "git add" to track)
```

- That is tell you there is a new file created here and this file need to push in your remote repo directory.

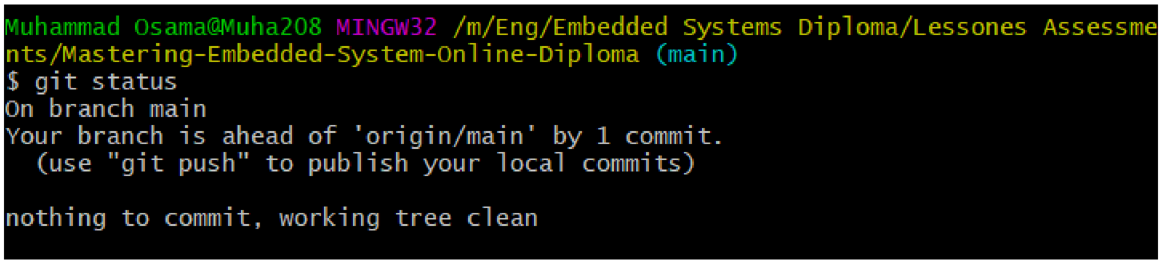


- Then we will command (git add) then * this is mean that we want add all files in this directory
- Then command (git status) to be sure that the files added or not



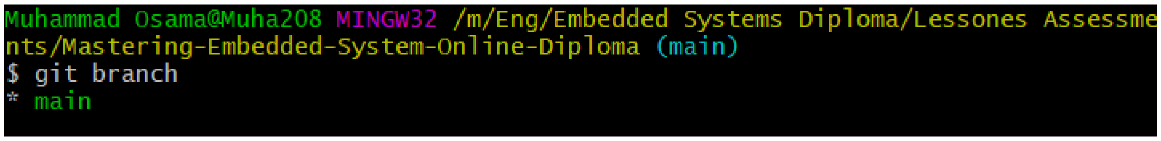
```
Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git add *

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   C-Lessones Assignments/Unit 1/Assignment 1.c
```

- We can see that it's add the file that we made it and it's wanted you to make a commit now.
- If we want to un-stage file from adding, Command (git reset (file name with extension).
- Command (git commit -m "Write your massage here") to commit your added files. You can commit a lot of time before you push your file if there is change happened.
- Then command (git status) to what happened tell now.



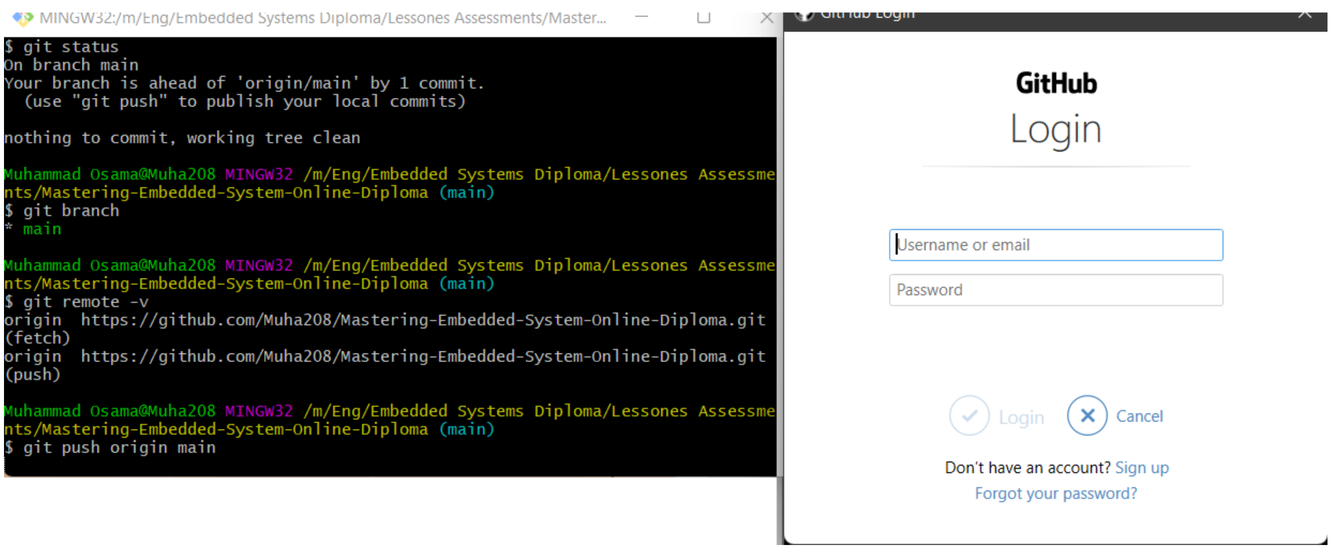
```
Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

- We can see here that comment tell us that we commit one file and it's need to push to our remote repo server.
- You can command (git branch) to see all branches under this master. We will take the name of the master from this order



```
Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git branch
* main
```

- To Command (git push (repo name) (branch name)) you should to get the repo name by command (git remote -v)
- The now we command (git push).



MINGW32:/m/Eng/Embedded Systems Diploma/Lessones Assessments/Master...

```
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git branch
* main

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git remote -v
origin  https://github.com/Muha208/Mastering-Embedded-System-Online-Diploma.git
(fetch)
origin  https://github.com/Muha208/Mastering-Embedded-System-Online-Diploma.git
(push)

Muhammad Osama@Muha208 MINGW32 /m/Eng/Embedded Systems Diploma/Lessones Assessme
nts/Mastering-Embedded-System-Online-Diploma (main)
$ git push origin main
```

- After we (git push) it's opened this window to login to this server to push your file on it. Enter your E-mail and your password.
- Then it will push files or folder on your remote repo server.

Command's part 2

No	Commands	Main	To do
1	**cd**	**change directory**	**تغير ال path الحالي**
2	**Mkdir**	**Make direction**	**لانشاء ملف جديد**
3	**dir**	**direction**	**لمشاهدة جميع الملفات او الفولدرات في ال path الحالي**
4	**Git status**		**لمعرفة التغيرات الحادثة داخل ال git path working directory**
5	**Git reset (file name with extension)**		**لإزالة ملف تم عمل add له**
6	**git branch**		**معرفة الفروع التي يحتويها ال master واسم ال master**
7	**git remote -v**		**لمعرفة اسم ال remote repo**