

# fLaborator 4

## Tooling

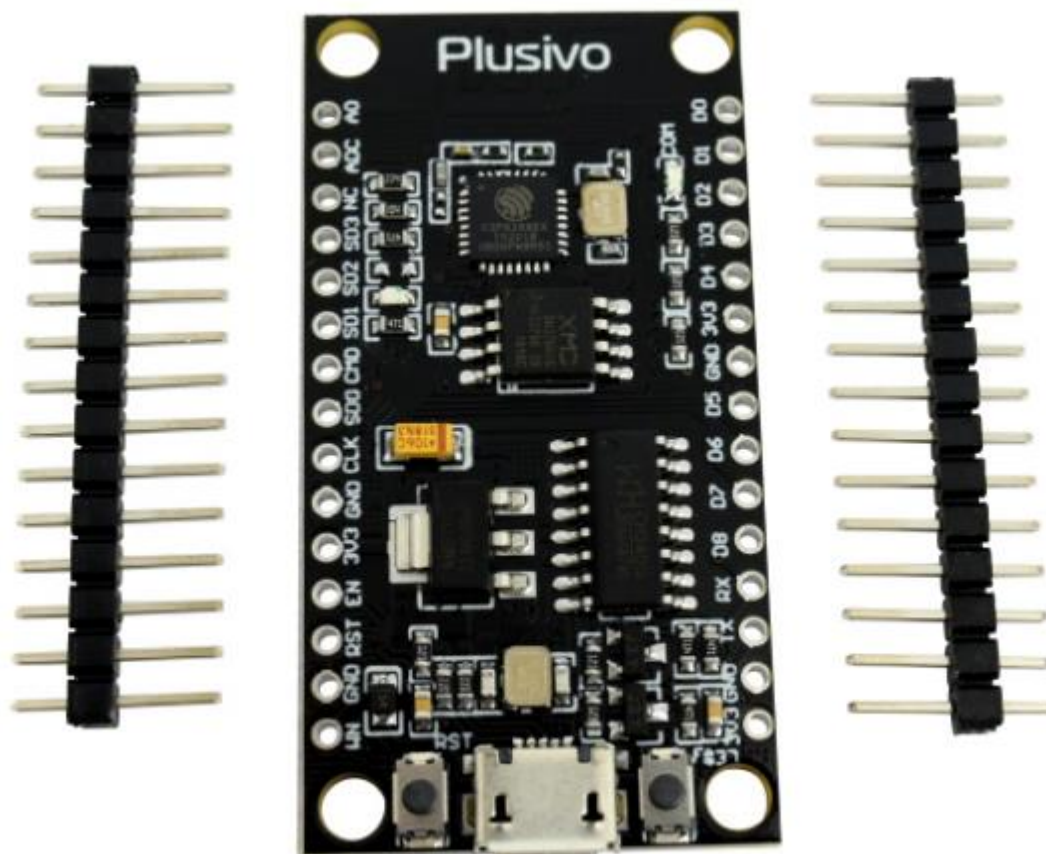
### Obiective

- Familiarizarea cu placa de dezvoltare
- Familiarizarea cu mediul de dezvoltare
- Înțelegerea conceptului de “version control”

### Cuprins

Obiective .....	1
Cuprins.....	1
Prezentarea plăcii de dezvoltare.....	2
Prezentarea mediului de dezvoltare.....	2
Crearea unui nou proiect .....	3

## Prezentarea plăcii de dezvoltare



Placa de dezvoltare folosită la acest laborator va fi: **Plusivo Micro WiFi Development Board with ESP8266 and CH340G**.

Această placă de dezvoltare este proiectată având la bază chipul **ESP8266EX**. Acesta funcționează la o frecvență de 80MHz, integrând procesorul RISC Tensilica L106 de 32 de biți.

## Prezentarea mediului de dezvoltare

Un mediu de dezvoltare (engl. software development environment, sau integrated development environment - "mediu integrat de dezvoltare") este un set de aplicații care ajută programatorul în scrierea programelor.

Un mediu de dezvoltare combină toți pașii necesari creării unui program:

- editarea codului sursă;

- compilarea;
- depanarea (debug);
- testarea;
- generarea de documentație.

Principalele componente ale unui mediu de dezvoltare sunt editorul de cod sursă și depanatorul. Mediile de dezvoltare apelează compilatoare sau interpretoare, care pot veni în același pachet cu mediul însuși sau pot fi instalate separat de către programator.

De obicei un mediu de dezvoltare este specific unui anumit limbaj de programare, însă există la ora actuală și medii de dezvoltare care pot lucra cu mai multe limbaje, de ex. Eclipse sau Microsoft Visual Studio.

Mediul de dezvoltare folosit în cadrul laboratorului este ECLIPSE IDE. Acesta permite editarea și compilarea codului sursă folosind un SDK produs de dezvoltatorul modulului ESP8266, Espressif (ESP8266 RTOS SDK - <https://docs.espressif.com/projects/esp8266-rtos-sdk/en/v3.2/api-guides/build-system.html>).

## Crearea unui nou proiect

Pentru a crea un nou proiect, vom folosi arhiva „ESP8266\_IDE.ZIP”. În această arhivă vom găsi mediul de dezvoltare Elipse IDE împreună cu toate componentele necesare rulării programului de dezvoltare, un proiect Demo deja configurat pentru utilizare și 2 fișiere cu extensia \*.bat utilizate pentru a deschide mediul de dezvoltare, respectiv configurarea parametrilor corespunzători plăcii de dezvoltare (COM, Baud Rate, etc.). Alături de arhivă vom găsi un fișier executabil „putty.exe” necesar depanării codului prin intermediul comunicării UART dintre placa de dezvoltare și calculator deoarece mediul de dezvoltare nu include această funcționalitate și un fișier word „readme – how to.docx” unde vom găsi câteva instrucțiuni pentru configurarea inițială a mediului de dezvoltare.

După cum se poate observa în figura următoare, fiecare proiect care utilizează acest SDK este împărțit în componente. Fiecare componentă are fișierele sursă și fișierele header (dacă este cazul) corespunzătoare. Astfel, fiecare proiect este organizat în felul urmator:

```
- myProject/  
  - Makefile  
  - sdkconfig  
  - components/ - component1/ - component.mk  
                        - Kconfig  
                        - src1.c  
                        - component2/ - component.mk  
                                      - Kconfig  
                                      - src1.c  
                                      - include/ - component2.h  
  - main/      - src1.c  
                - src2.c  
                - component.mk  
  - build/
```

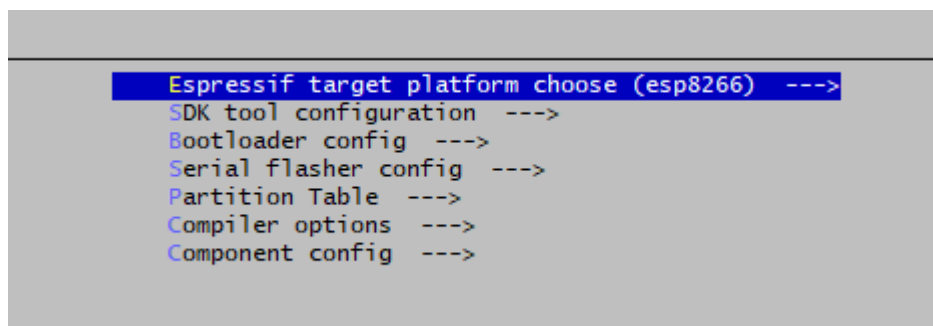
Fișierul „Makefile” conține detalii despre proiect pentru a putea fi compilat. Fișierul „sdkconfig” conține configurarea SDK-ului pentru a putea corespunde modelului plăcii noastre de dezvoltare. În folderul „components” o să găsim componentele create de noi după structura de mai sus și în folderul „main” o să găsim, de obicei, punctul de intrare principal al programului, adică funcția „app\_main”. În folderul „build” se găsesc fișierele rezultate în urma compilării.

```

9 COMPONENT_SRCDIRS +=   BSW/MCAL/ADC \
10                        BSW/MCAL/GPIO \
11                        BSW/MCAL/PWM \
12                        BSW/MCAL/WIFI \
13                        \
14                        BSW/HAL/Buzzer \
15                        BSW/HAL/Com \
16                        BSW/HAL/DC_Motor \
17                        BSW/HAL/Photo_Resistor \
18                        BSW/HAL/Proximity_Sensor \
19                        BSW/HAL/Servo_Motor \
20                        BSW/HAL/Shift_Register \
21                        BSW/HAL/Temp_Sensor \
22                        \
23                        RTE \
24                        \
25                        ASW/Ambient_Light \
26                        ASW/Climate_Control_System \
27                        ASW/Headlights \
28                        ASW/Horn \
29                        ASW/Locking_System \
30                        ASW/Security \
31                        ASW/Trunk \
32                        \
33                        SRVL/SCHEDULER \
34

```

Pentru a putea compila programul conform plăcii de dezvoltare alese, vom efectua câteva setări în fișierul „sdkconfig”. Primul pas va fi să deschidem fișierul „LaunchMSYS2.bat”. În acest moment vom avea o fereastră de comandă deschisă ce ne va permite să efectuăm setările necesare compilării corecte a programului. Introducem comanda “cd /d/ESP8266\_IDE/workspace/Demo/” pentru a naviga la adresa proiectului nostru. Următoarea comandă va fi “make menuconfig”, comandă care ne va deschide fereastra de configurare pentru proiect.



Din acest moment, navigarea în această fereastră se va face utilizând tastatura. Prima dată vom intra în meniul “Serial flasher config” și vom efectua următoarele setări:

```
(COM9) Default serial port
  Default baud rate (115200 baud) --->
[*] Use compressed upload
  Flash SPI mode (QIO) --->
  Flash SPI speed (40 MHz) --->
  Flash size (4 MB) --->
  Before flashing (Reset to bootloader) --->
  After flashing (Hard reset after flashing) --->
  'make monitor' baud rate (74880 bps) --->
[ ] Enable monitor time information
[ ] Enable console log save
```

În cazul primei setari, vom completa cu portul “COM” corespunzător plăcii noastre.

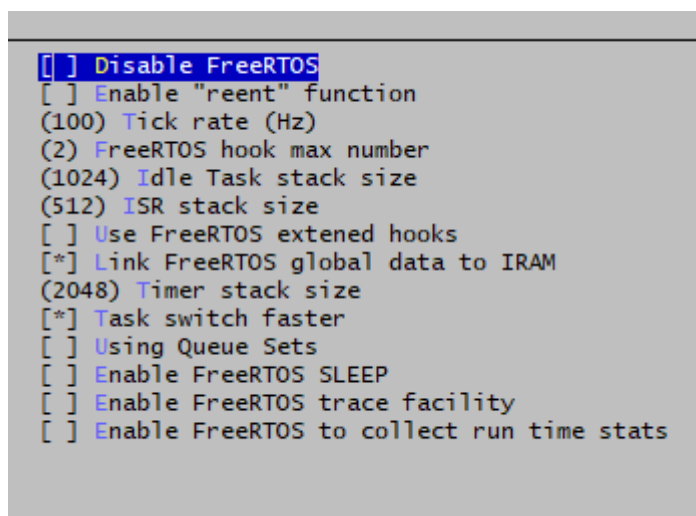
Pentru pasul urmator, vom naviga în meniul “Component config” -> “ESP8266-specific” și vom efectua urmatoarele setări:

```
CPU frequency (80 MHz) --->
  Line ending for UART output (CRLF) --->
  File name macro text (no PATH) --->
[*] Using new ets_vprintf instead of rom code
[ ] Link ets_printf to IRAM
[ ] Enable full cache mode
  UART for console output (Default: UART0) --->
(74880) UART console baud rate
[ ] Swap UART0 I/O pins
[ ] Disable ROM UART print
[ ] Output full stack data of task
  Panic handler behaviour (Print registers and reboot) --->
(3584) Main task stack size
[*] Initialize Task Watchdog Timer on startup
[*] Invoke panic handler on Task Watchdog timeout
  Task Watchdog timeout period (seconds) (26.2144s) --->
[*] Enable reset reason
(2048) ppT task stack size
(2048) Event loop stack size
[*] Link libcore.a internal global data to IRAM
  Crystal used in your module or board (26MHz) --->
[ ] ESP8266 update from old SDK by OTA
[ ] Boot copy app
[*] Enable lookup of error code strings
```

În continuare vom naviga în meniul “HTTP Server”:

```
(2048) Max HTTP Request Header Length
(1024) Max HTTP URI Length
```

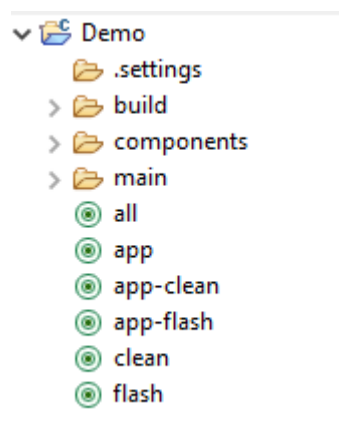
În ultimul pas vom naviga în meniul “FreeRTOS”:



Pentru a salva setările, apăsăm “Exit” până când ajungem în meniul principal unde vom apăsa “Save” pentru a genera fișierul “sdkconfig” corespunzător.

În cadrul proiectului nostru, vom utiliza o singură componentă, „VITAL” care conține folderele specifice fiecărui “strat” din cod și fișierele header corespunzătoare acestora. Alături de acestea, mai găsim un fișier „component.mk” în care se găsesc căile spre fișierele sursă pentru a putea fi recunoscute de compilator și o cale spre fișierul folosit ca pagină web, „index.html”.

Pentru a putea compila corect proiectul, în fereastra din dreapta a IDE-ului găsim o listă de butoane care conțin comenzile pentru aceste acțiuni:



Butonul numit „clean” va șterge fișierele compilate și butonul „all” va face o compilare a acestora. În cazul în care nu avem erori de compilare, putem apăsa butonul „flash” și codul va fi încărcat pe placa noastră de dezvoltare.

## Ce este version control?

**Version control** presupune organizarea fișierelor proiectului la care lucrăm astfel încât să monitorizăm mai ușor modificările ce apar pe parcurs.

Există astfel două metode: manuală și automată.

- *Metoda manuală* - se crează copii ale unui fișer când începem un bloc nou de modificări. Acest procedeu poate să scape ușor de sub control și nu aduce o bună oraganizare pe termen lung.
- *Metoda automată* - modificările se rescriu în fișier dar se păstrează versiunile vechi pentru motive de erori.

## Consola Git Bash

**Git** este un sistem de version control, open-source, care rulează pe majoritatea platformelor, inclusiv Linux, Windows și OS X.

## Care sunt noțiunile care stau la baza Git-ului?

[Git](#) funcționează pe bază de **commituri**. Adică un set de modificări semnificative sau mai puțin semnificative ale unuia sau mai multor fișiere, care de obicei au legătură sau funcționează în tandem. Deci fie că am modificat 3 linii sau 100 pe diferite fișiere, acestea se grupează în commituri. De regulă este recomandat să se păstreze o anumită frecvență și complexitate pentru a face commituri, spre exemplu când am implementat o nouă funcționalitate sau a trecut un număr de minute/ore.

O altă noțiune importantă – **branchurile** (ramuri). Ele sunt ramificații de la un proiect inițial cu scopul de a face o nouă copie, când avem de-a face cu o funcționalitate experimentală sau o secvență care nu suntem siguri dacă va funcționa. Astfel, prin a crea un branch, ne este mai ușor de a reveni la ceea ce funcționează în cazul unor erori masive. Dar nu numai, le putem folosi și pentru teste.

**Merge** – îmbinarea a două branch-uri într-unul. Ex: Funcționalitatea este reușită și trebuie adăugată la proiectul final. Prin merge putem concatena branch-ul principal (master) și cel al funcționalității.

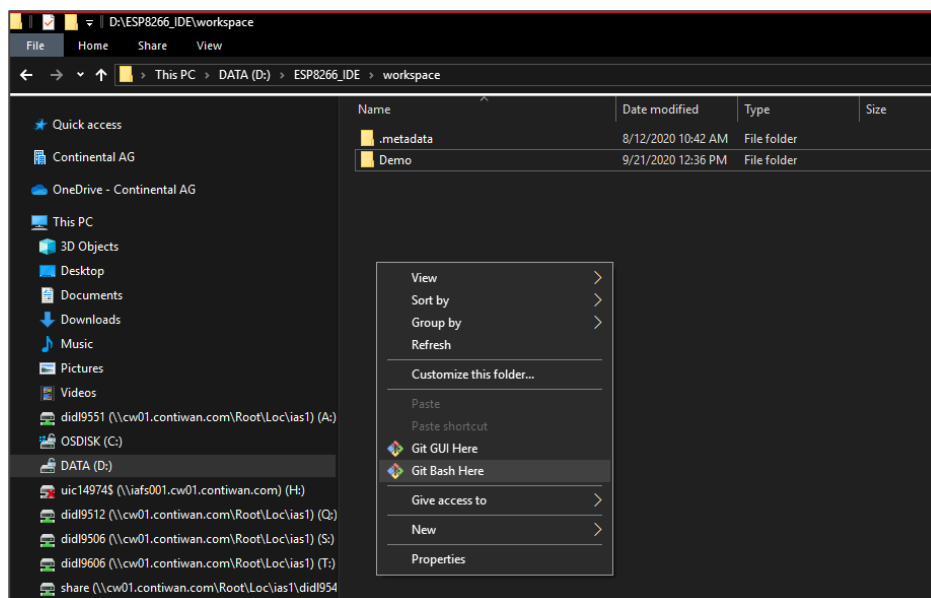
## Cum utilizăm Git Bash?

Git Bash permite utilizarea funcționalităților Git prin apelarea comenzilor din consolă.

### Pasul 1

Pentru inițializarea Git-ului într-un director, mai întâi trebuie să ajungem în directorul respectiv. Acest lucru se poate face în două moduri:

**Navigarea folosind Windows Explorer** până la directorul necesar și deschiderea acolo a consolei GitBash (click-dreapta, Git Bash Here).



### Navigarea folosind consola Git Bash

În primul rând, se deschide aplicația Git Bash.

Pentru navigație se utilizează comanda `cd` urmată de locație

**Ex:** `cd d:/ESP8266_IDE/workspace/Demo`

Când ne aflăm într-un director, dacă vrem să înaintăm în alt folder, apelăm comanda:

**cd <denumirea>**

Pentru a ne întoarce la folderul părinte, apelăm comanda

**cd ..**

Pentru a șterge comenzile anterioare (doar vizual) **Ctrl-L**

### Pasul 2

Pentru crearea unui repository Git se utilizează comanda:

**git init**

care va crea un repository Git gol în directorul în care ne aflăm

### Pasul 3

Pentru adăugarea unor fișiere noi, se utilizează comanda

**git add nume.format**

**ex:** `git add file.txt`

sau este posibilă utilizarea de *wildcard* (selectarea mai multor fișiere cu același format) prin comanda:

**git add \*.format**




**ex:** `git add *.c *.h` (va adăuga toate fișierele cu extensia .c și .h)

Pentru a adăuga toate fișierele din folder se apelează comanda: **git add.**

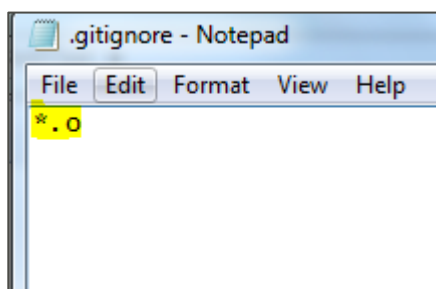
**Obs:** Uneori avem fișiere de o anumită extensie pe care nu dorim să le adăugăm deloc, spre exemplu fișierele obiect, cu extensia .o

Atunci, aflându-ne în folderul proiectului apelăm comanda: **touch.gitignore**

Această comandă crează un fișier cu extensia .gitignore

 .git	10/26/2018 4:44 PM	File folder
 Proiect	10/26/2018 4:09 PM	File folder
 .gitignore	10/29/2018 5:46 PM	Text Document

În acest fișier putem include extensia fișierelor pe care dorim să le ignorăm:



Acum, apelând comanda: **git status** (afișează starea curentă a proiectului) vom vedea că fișierele .o nu mai apar

#### Pasul 4

Pentru a da **commit** se apelează comanda:

**git commit -m "Mesaj asociat acestui commit"**

## Utilizarea Branch-urilor

Pentru crearea unui **branch** nou se apelează comanda:

**git branch <branch nou>**

Pentru a afișa toate branch-urile existente:

**git branch**

Pentru a trece pe alt branch:

**git checkout <numele>**

Pentru a șterge un branch:

**git branch -d <numele>**

### **Observație:**

Dacă ne aflăm pe un branch și adăugăm un fișier în „staging”(comanda: **git add**), iar apoi, fără a da commit, trecem pe alt branch, vom observa că acest fișier încă persistă și ar putea crea probleme mai apoi.

Respectiv folosim comandă **git stash** pentru a trece aceste schimbări într-o fază în care ele nu vor influența munca noastră de pe alt branch. Când suntem gata să continuăm munca pe branch-ul inițial, apelăm comanda: **git stash apply** , pentru a recupera acele modificări nefinisate.

Până acum am vorbit doar de manipularea locală a proiectului, dar pentru ca mai multe persoane să aibă acces la proiect, acesta trebuie urcat pe un server apelând comanda:

**git push** specificând:

1)Unde vrem să dăm upload (link-ul)

2)Care branch

Pentru a nu fi nevoiți să introducem de fiecare dată link-ul, putem să îl memorăm:

**git remote add <RepoName> <url>**

**ex: *git remote add myRepo https://github.com/ViTAL\_Laboratory/ViTAL\_lab.git***

Acum, când introducem **myRepo**, acesta va fi înlocuit cu link-ul de mai sus.

Pentru a da push branch-ului *master* apelăm comanda:

**git push myRepo master**

Pentru a descărca un proiect aflat pe server(integral sau parțial) folosim:

**a) git clone <RepoName>**

Obținem o copie integrală a proiectului (inclusiv folderul .git care conține toată istoria proiectului)

**b) git fetch <RepoName>**

Selectează și descarcă doar modificările făcute la proiect (diferențele dintre local și remote)

**c) git pull <RepoName>**

Este asemănător cu *fetch*, doar că în același timp realizează un *merge* între versiunea descarcată și cea locală

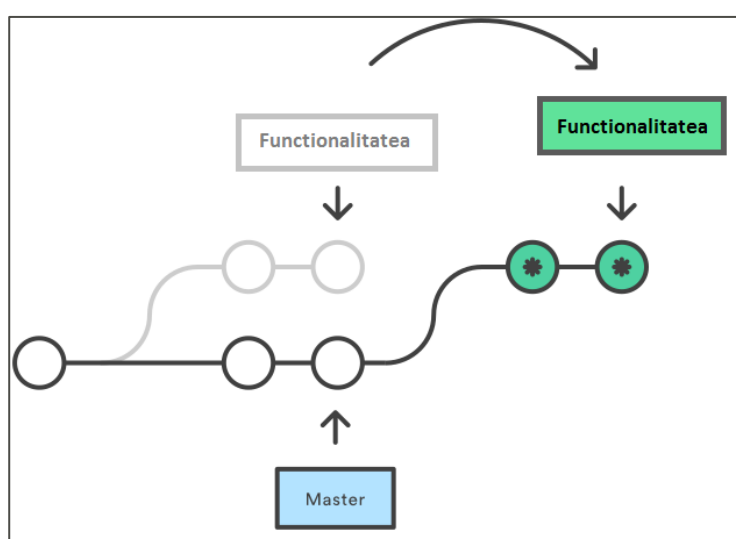
Uneori avem nevoie să anulăm anumite commit-uri (spre exemplu când nu am reușit să implementăm o funcționalitate).

Pentru aceasta se folosește comanda: **git reset**

Resetul poate fi de trei tipuri:

- 1) Soft reset – este analogic cu un **amend**. Modificările făcute sunt trecute în staged
- 2) Mixed reset – modificările făcute trec în unstaged
- 3) Hard reset – modificările făcute sunt înlăturate definitiv

O altă modalitate de a obține rezultatul unui **merge** este **rebase**.



Printr-o serie de commit-uri efectuate automat se obține o liniarizare a istoricului proiectului.

Pentru aceasta folosim comanda **git rebase** specificând:

- a) branch-ul pe care ne aflăm.
- b) branch-ul funcționalității

**Ex: git rebase Master Funcționalitatea**