Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
T1.txt
.git
Final file
T1.txt



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | Local disk/Word doc |  | Local repo |  | Git hub/URL address |
|  |  |  |  |  |

**Process of pushing the codes into local repository in git.**

* c:/ cd repo
* c:/CGBatch name> ls -la
  + >cd demo
  + >Mkdir demo
  + >Cd demo
  + >Touch t1.txt
  + >Ll
  + >git add t1.txt
  + It will push the reference to .git folder
  + Git commit is used to save it temporarily in local repository
  + >Git commit -m "Type anything"
  + If commit is not done we might lose the changes made. With commit every line of command is in local directory
  + >Git config --global user. Email "<email Id>"
  + >Git config user.name "<User name>"
  + This is used so that we don’t have to give username every time.
  + >Git pull origin main (If the code is not in the repository and push gets rejected.)
  + >Git push origin main
  + Pushing to original branch in the sub branch. When using this command it will push the file into git hub account.
* When touch <File Name> is used it will be created in the hard disk
* If git commit is used the references will be saved.
* If git push origin and main is used then only updated contents will be pushed. While pushing it will check URL for conflicts. To get the local repository updated in the global repository then we should pull the file first and then push it into global. If both the repositories are not in sync then they can't be updated.

**Make Utility**

* It is used to compile, build, execute and clean.
* Directory structure :

>>Mkdir bin obj src inc scripts libs

>>mv lib lib

>>ll

>>tree

>>gcc -c -I./inc ./src/calc.c -o ./obj/calc.o (To compile)

* "." Is current directory and ".." is parent directory.
* To access all the files we use "./".
* >>gcc -c -I../inc ./src/calc.c -o ./obj/calc.o (To compile the parent directory)

>>cd ..

>>ll

>>gcc -o app ./obj/calc.o ./obj/main.o (To compile main file)

>>ll

>>mv app bin

>>./bin/app (To execute bin)

>>vi ./src/main.c (To modify src code)

>> cd ..

* Even if the src code is modified we cannot execute it without the updation of src code.

>>gcc -c -I .inc/ ./src/main.c -o ./obj/main.o

>>gcc -o ./bin/app01 ./obj/calc.o ./obj/main.o (To update).

**Warning :** implicit declaration of function "null" (used the interface name but have not declared the interface).

(With standard ncc declaration it can be done).

Declaration :

>>vi ./inc/calc.h

>>cat ./inc/calc.h

>>gcc -o ./bin/app012./obj/calc.o ./obj/main.o (To update/prototype).

* As the source is modified, we can't compile and update every time we change so to overcome that we use make utility as it compiles right away.

>>rm ./bin/\* (To delete/cleaning).

>>rm ./obj/\*

>>vi ./scripts/Makefile

* cp -rf <src\_name> <dest\_folder> [To copy one repository from another]

To move

CC = gcc

Cflags=-c

Oflags = -o

OBJ=./obj

SRC=./src

INC=./inc

Echo $CC

Echo $cflags

Echo $oflags

Echo "Cleaning of the files"

Rm -rf ./obj/\*

Rm -rf ./bin/\*

Echo "Building the files"

$CC $cflags -I./inc/ ./src/main.c -o ./obj/main.o

$CC $cflags -I./inc/ ./src/calc.c -o ./obj/calc.o

$CC $oflags ./bin/app ./obj/calc.o ./obj/main.o

**>>sh ./scripts/myMakefile.sh**

**Code coverage Analysis using gcov**

* We know that we have tested out code well by one measure is that we have executed each code at least once.
* Code coverage tools, like gcov helps us know which lines of the code have been executed and how many times.
* Gcc -fprofile-arcs -ftest-coverage <file1.c> <file2.c> …
* Gcov notes files get created after the above command "file1.gcov". These note file contains structural info about the code which will help gcov understand the "gcov data" files created during execution of the executable.
* Also the executable has got added instrumentation code, to keep measuring code coverage.
* While we run the instrumental executable, "gcov data" files are created or updates e.g. file1.gcda.
* Gcno - g coverage note file.
* Gcov t1.c

Static Analysis using Splint

* Int 'ptr';

Printf("%d",\*ptr);

* A simple code review shows we are acessing an invalid ptr which has not been malloc.ed or pointed to a valid memory area.
* Static analysis tool analyze the code in similar but much more sophisticated ways, and warn us of potential bugs in the code.
* They are called "static" analyzers as they analyze the stationary(not runing) codes. This is in contrast with dynamic tools.

**To push into your git :**

* cd Batch17\_Oct/
* touch test01
* git add .
* git commit -m "updated with the git repo"
* git push origin main