KOÇ
UNIVERSITY

KOÇ UNIVERSITY

COMP 201/Fall 2020
C Bootcamp
5th October 2020

Mandana Bagheri Marzijarani

mmarzijarani20@ku.edu.tr

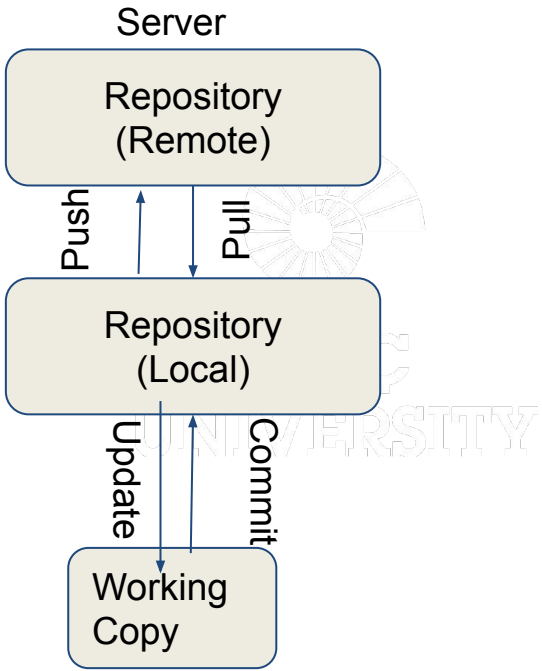
What we are going to cover today

- What is a *Version Control System* (VCS)?
- Git/Github introduction
  - Basics of Git
- Github Classroom & REPL.it introduction
  - How to accept assignments from Github Classroom
  - How to work on assignments and submit a solution through Github Classroom
- Simple C programs
  - How to compile and run a C program
  - What is a makefile
- Demo

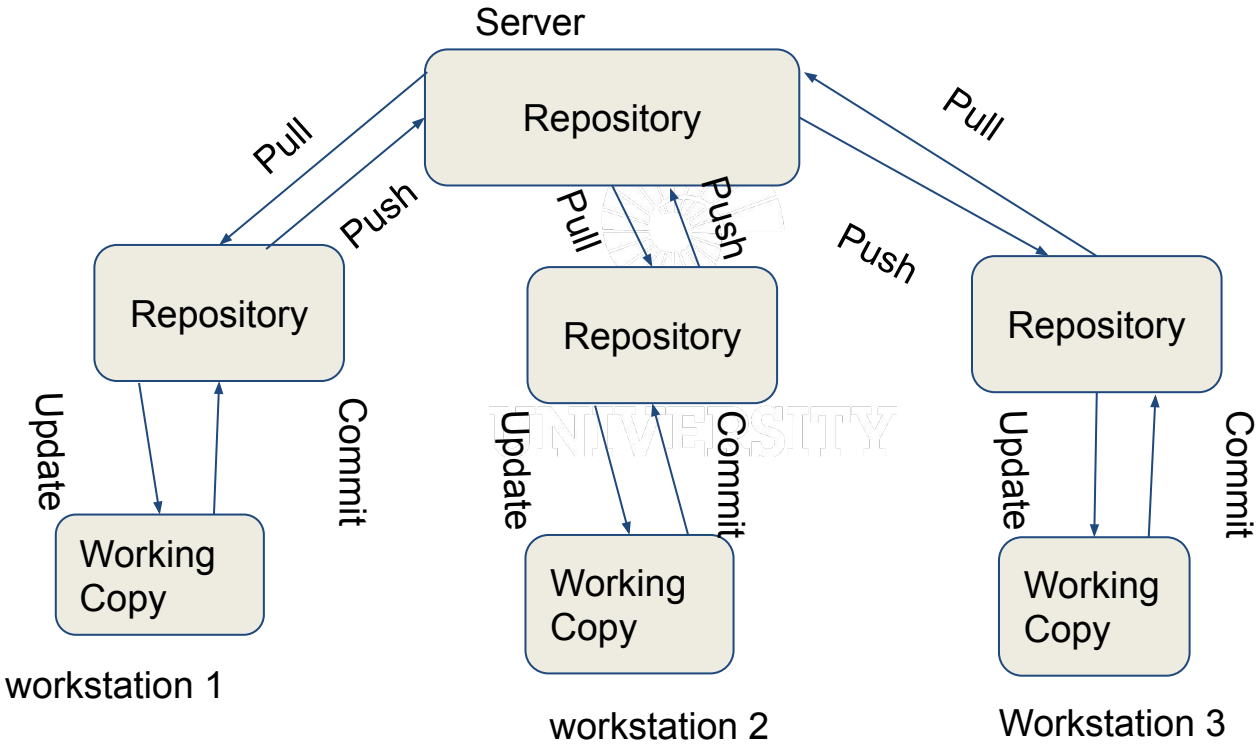
Version Control System

- Version controls systems are a class of software tools that keep track of every modifications to source code over time.
- Using VCS developers can recover earlier versions in case they needed it.
- VCSs support multiple team members working on the same project while minimizing conflict.
- They can be centralized or distributed

Git workflow (single user)

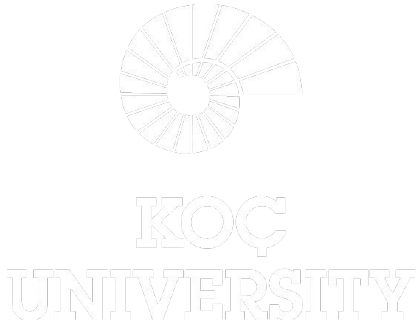


Git workflow (multi user)



Git basic commands

1. Clone
2. Commit
3. Push
4. Pull



Github Classroom

- It’s an app integrated with Github
- Assignments are based on a template repository
- Instructor can see your work and progress
  - But only anything that’s pushed
- Github Classroom can be utilized to automatically tests and grades your work

Github Classroom+REPL.it

- **Repl.it** is a cloud IDE
  - Lets you work on your repository on your browser
  - No need to have git or compiler installed
- Since *May 2020* Github Classroom and RELP.it are integrated

Accept and work on assignments

- During this semester, you will get an invitation link to accept assignments.
- To work on the assignment, you have two options:
  - Clone it on your local machine and push changes to your Github Classroom repository.
  - Work on it in your browser using REPL.it
    - Which automatically clones but you still need to push your work when done

C Programming

- C is the most widely used programming language
- Almost all new hardware come with C compilers
- Is very tightly coupled with the platform (OS+HW)
- Provides direct access to the memory
  - Which is the source of it being hard and prone to errors if attention is not paid
- Is otherwise very similar to other languages

Sample C Program

- All C programs start with main() function
- All variables have types
- Library *definitions* are added via #include
- Address of variables are sent to many functions via & operator
- Pointers are a big deal!

```c
#include <stdio.h> // standard IO functions

int main() // main is the program entry function
{
    int a, b; // two int variables
    scanf("%d %d", &a, &b); // address of a, b
    printf("Hello world %d\n", a + b);
    return 0;
}
```

How to compile and run a C program

- C programs are compiled
- There are generally two means of compiling C programs:
  - **Partially** (turns into object files and libraries)
  - **Fully** (all the source code into one binary)
- Partially is still needed if the project is too large, or if uses external libraries that are not source code

How to compile and run a C program (2)

- C programs are compiled with C compilers
- The most common compilers are **GCC** (GNU Compiler Collection) and **Clang** (Apple's LLVM compiler)
- gcc -o target_binary -Wall -g -O3 file1.c file2.c file3.c
  - -o defines output file
  - -W all means emit all warnings
  - -g includes debug symbols (more info on errors)
  - -O means optimization (0 to 3)
  - The rest are C source code files

What is a Makefile

- Compiling large C programs is very slow
- Instead of compiling all of the program at any change, we compile files separately into object files
  - That way only the changed files are recompiled
- Then we *link* all the object files into the binary
- But how can we tell which file is changed to compile easily?

What is a Makefile (2)

- Make is a UNIX utility that given a list of source code files
  - Can detect which ones have changed
  - Can run commands on those files
  - Can clear out extra files
  - Can determine simple dependencies
- Make reads the list and configurations from a file called *Makefile*

C Makefile example

- Items in the file:
  - List of source code files
  - Flags to compiler/linker
  - Cleanup operations
  - Name of binary
- **Make** automatically resolves dependencies, and only rebuilds files that have changed.

```
1  TARGET = my_program
2  SOURCES := $(find . -name *.c)
3  OBJECTS = $(SOURCES:.c=.o)
4  CC=gcc
5  CFLAGS= -O1 -g -Wall
6  LD=ld
7  LDFLAGS=
8
9  all : $(TARGET)
10
11 $(TARGET) : $(OBJECTS)
12     $(LD) $(LDFLAGS) -o $@ $^
13
14 %.o : %.c
15     $(CC) $(CFLAGS) -o $@ -c $<
16
17 .PHONY : clean all
18 clean :
19     rm -f $(OBJECTS)
20     rm -f $(TARGET)
```

THANK YOU!

Questions?