# C Development Environment

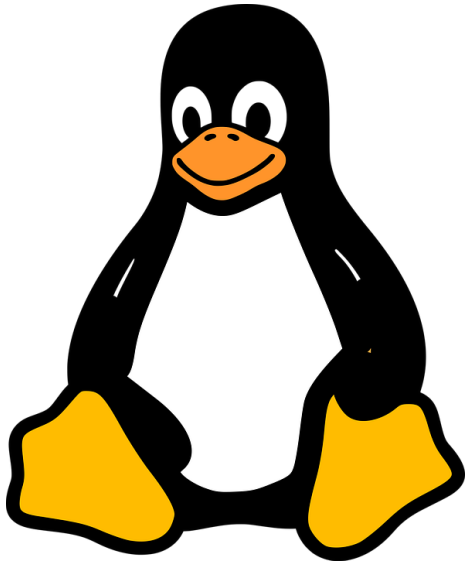
**Nicola Dall'Ora**
nicola.dallora@univr.it

**Nicola Drago**
nicola.drago@univr.it

Co-Author
Samuele Germiniani

# Outline

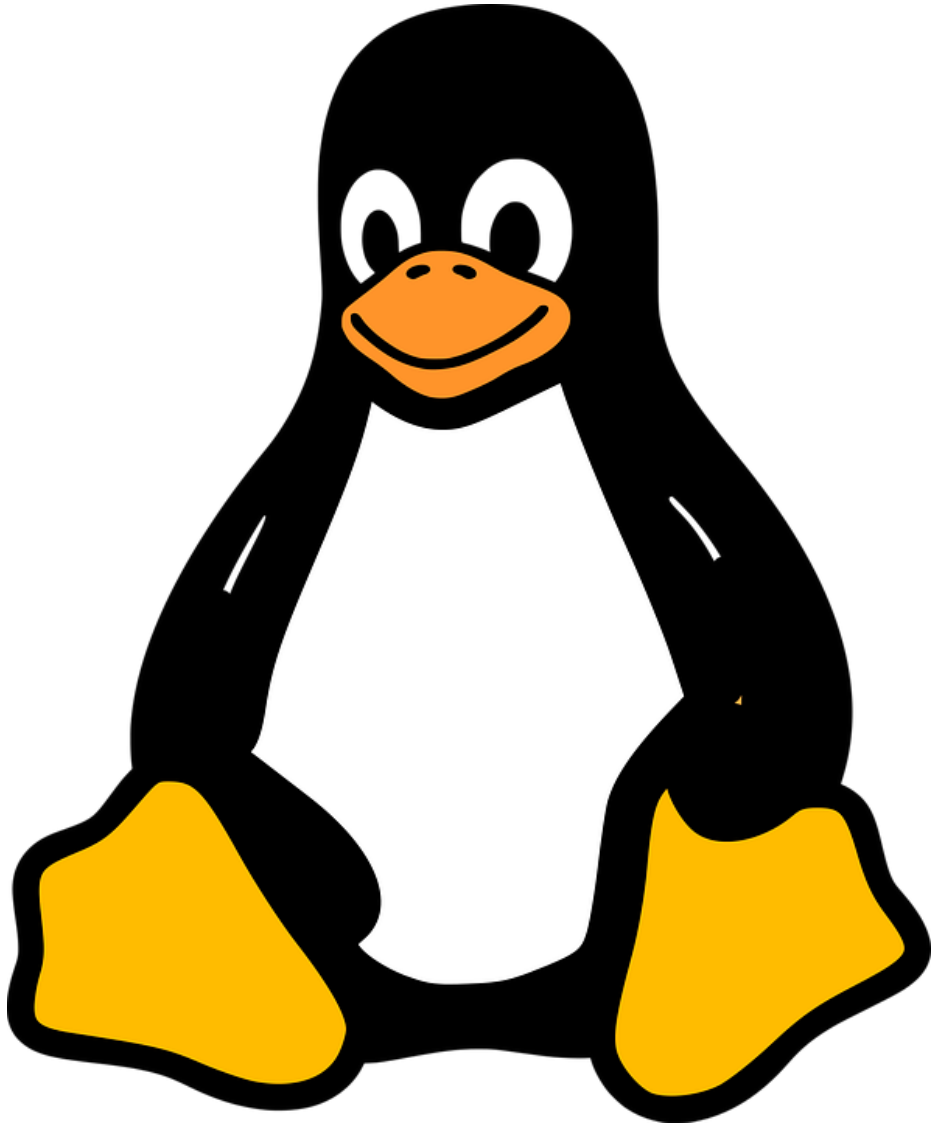
- Introduction
- Overview of tools
- git
- gcc
- makefile
- gdb
- doxygen



# Most Operating systems are written in C. Why ?

~~"Any application that can be written in JavaScript, will eventually be written in JavaScript" — Jeff Atwood, Co-Founder of Stack Overflow~~

- **Low-Level Control:** The C language provides a good balance between high-level abstractions and low-level control

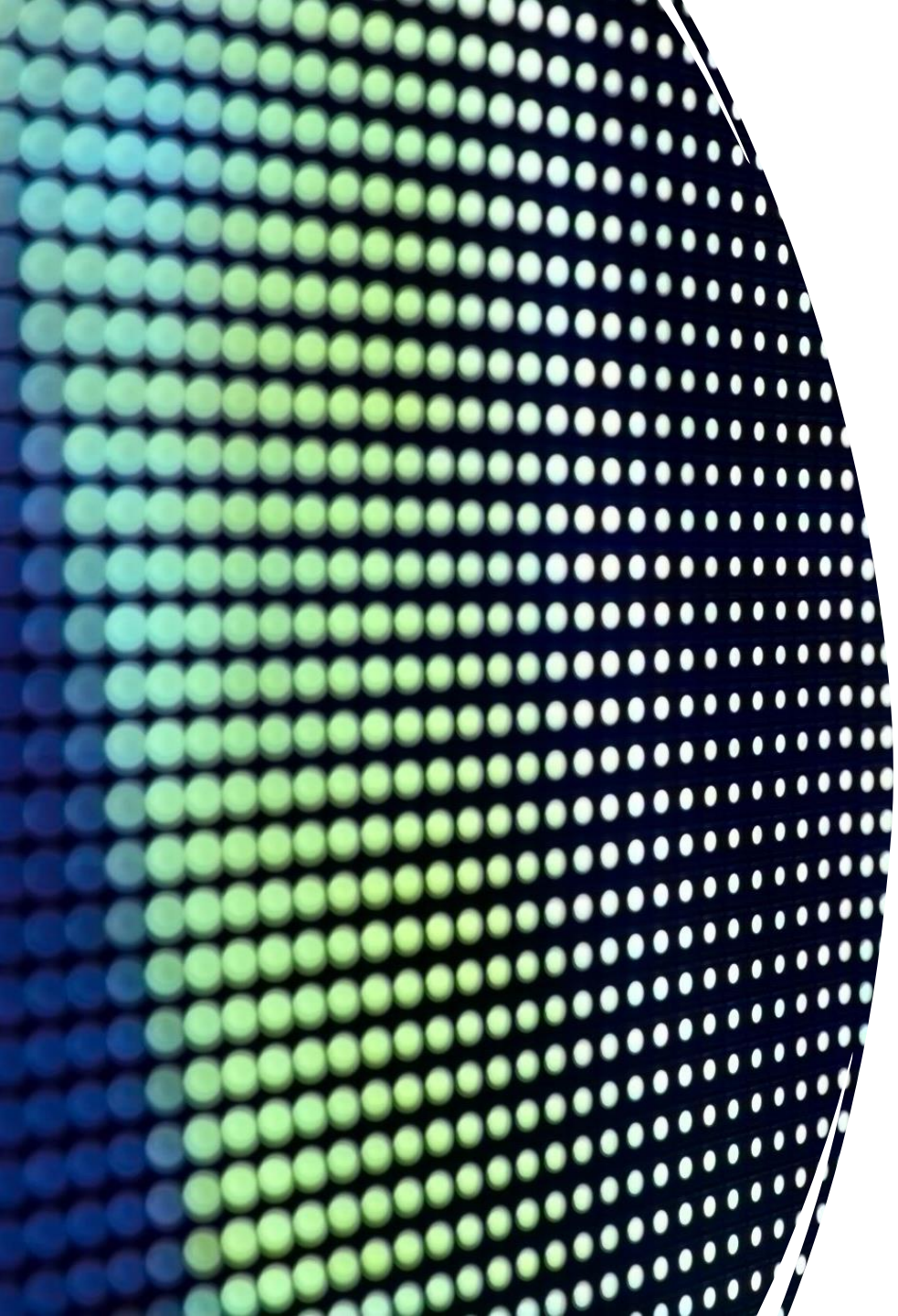


# Target OS: Linux

- Debian-based distro is suggested
  - Ubuntu20.04+
- OS on Virtual or Physical machine

# IDE minimum features

- Syntax highlighting
- Code completion
- Real-time error checking

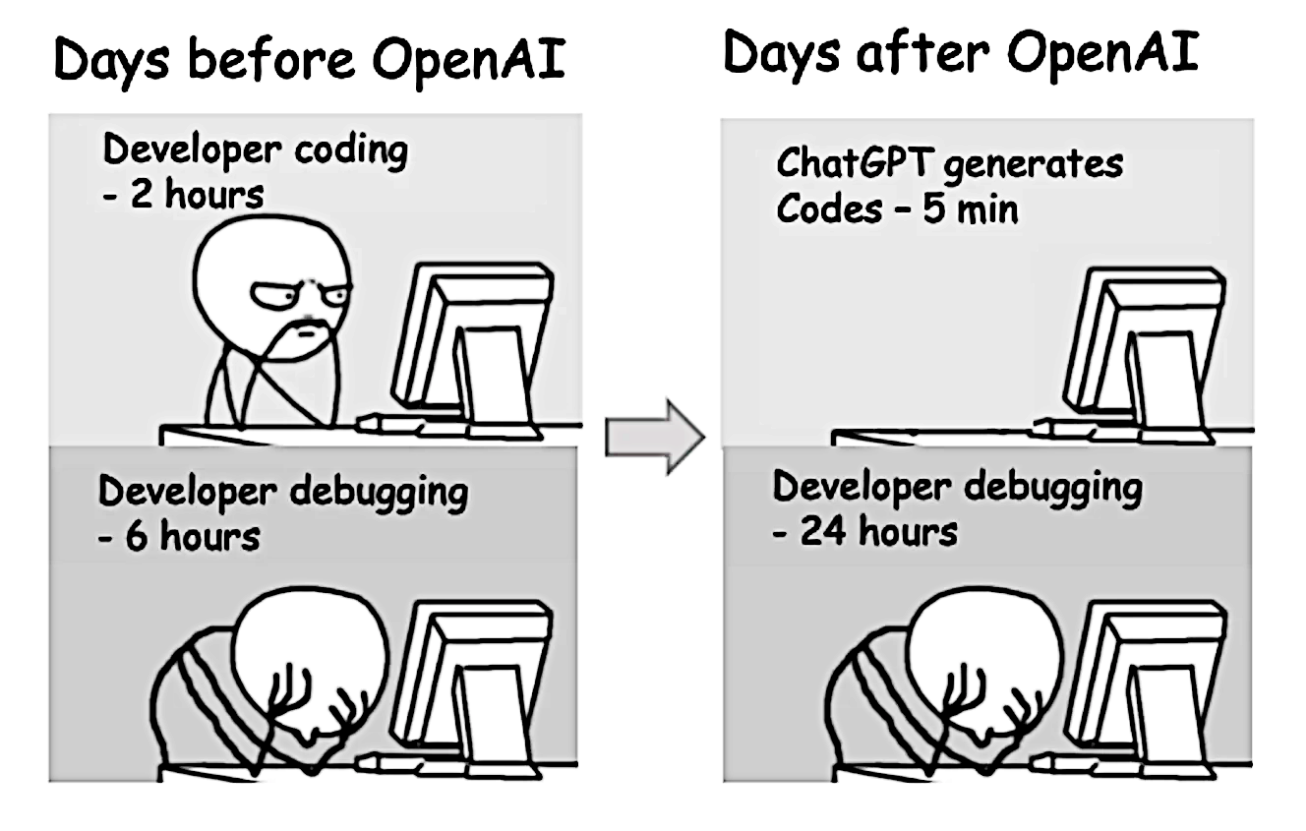


# Code indentation

- Clang format (suggested)
  https://clang.llvm.org/docs/ClangFormat.html

# AI code generators

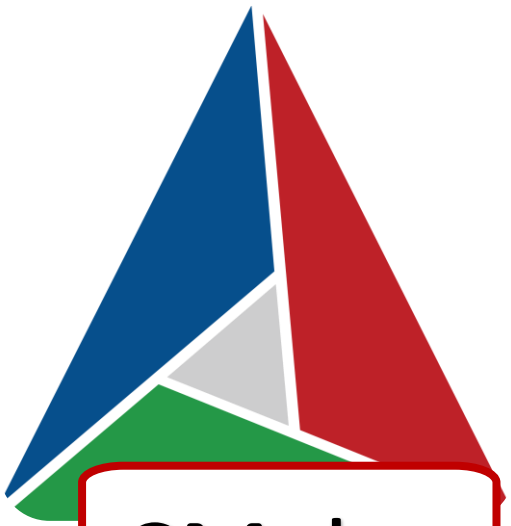

- ChatGPT
  - https://chat.openai.com/
- Copilot (strongly suggested)
  - https://github.com/features/copilot
  - Free if you are a student



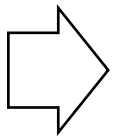

- Use AI at your own risk

# Online help

# C - Compilation chain



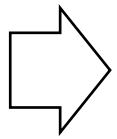
CMake



**Suggested option** ✔

https://www.gnu.org/software/make/manual/make.html

Makefile



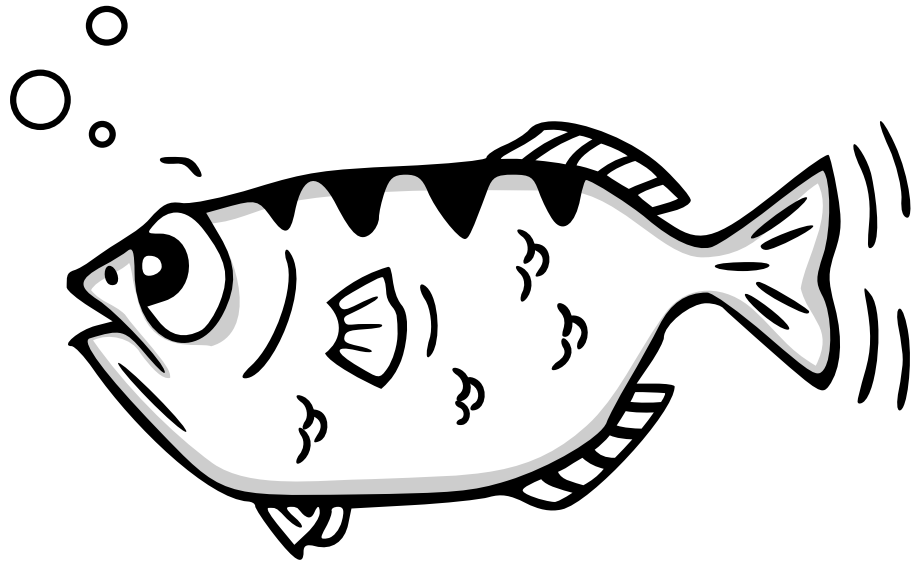

Compiler

High complexity project

Low complexity project

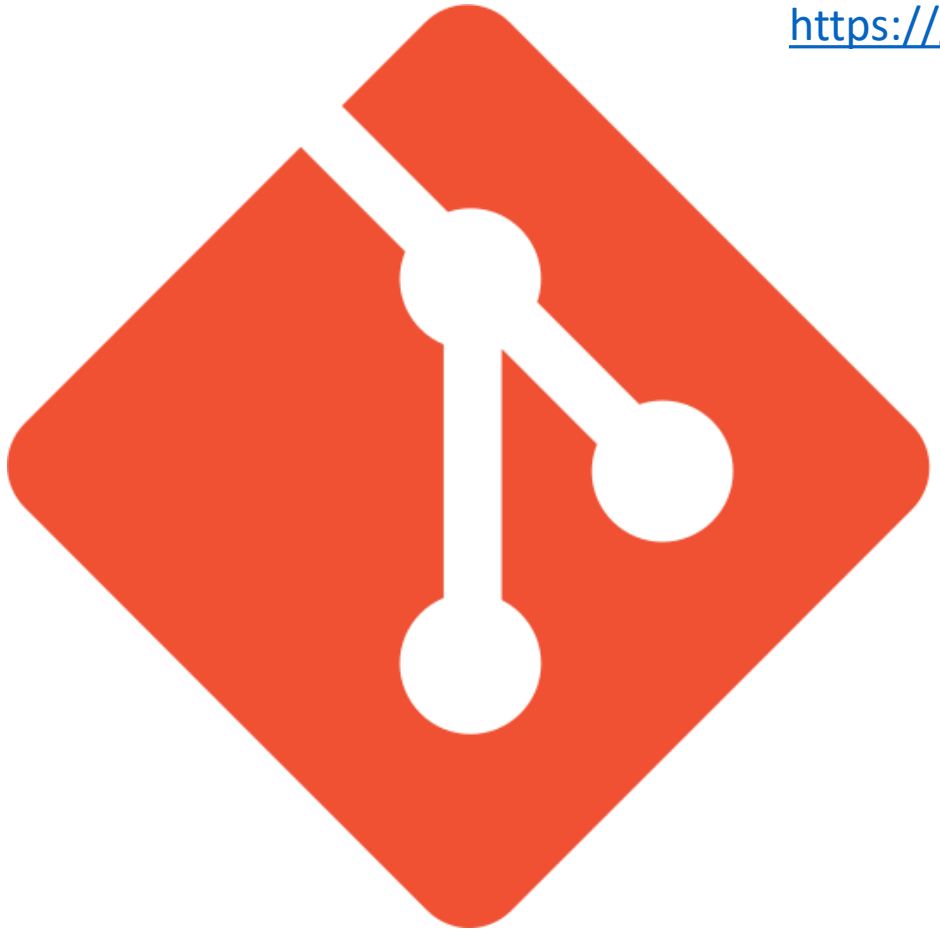
# Debuggers

- GDB
  - https://www.sourceware.org/gdb/



# Version control

https://git-scm.com/docs/gittutorial



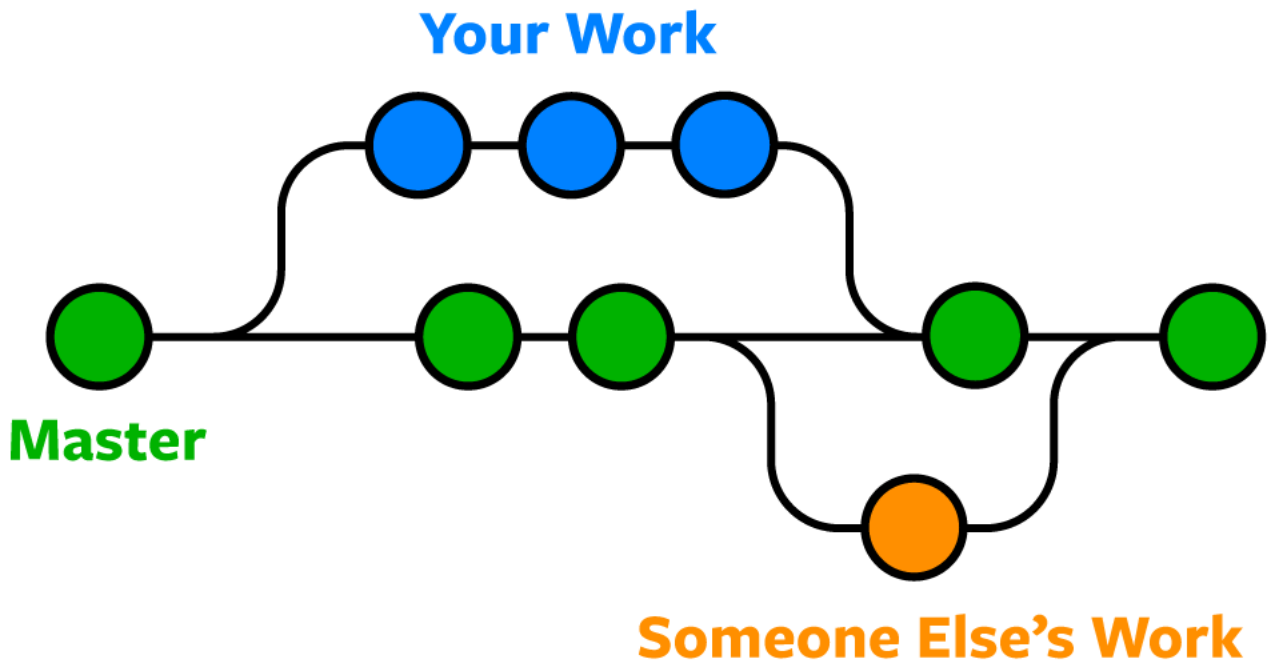
git


# Documentation

https://www.doxygen.nl/

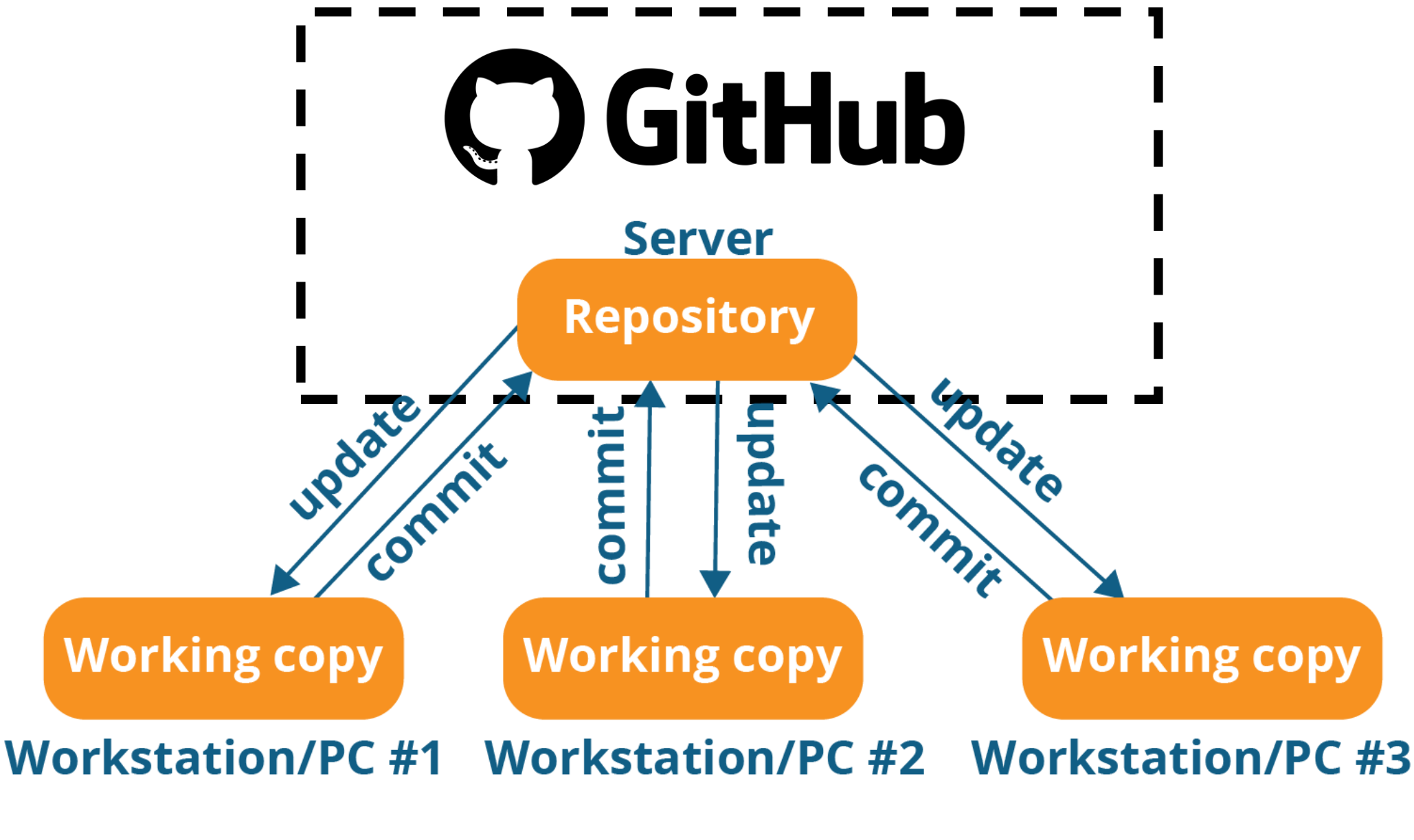
# Version control with git

- Track changes in source code and other files collaboratively




sudo apt-get install git

# Version control with git



# Basic commands

| Command | Description |
|---|---|
| git clone <url> | Clone a remote repository to your local machine |
| git init | Initialize a new Git repository (creates the .git directory) |
| git pull | Fetch and merge changes from a remote repository |
| git status | Show the status of your working directory |
| git diff | Show changes between working directory and staging area |
| git add <file> | Stage changes for commit. |
| git commit -m "message" | Commit staged changes with a message |
| git push -u origin main | Upload local commits to a remote repository called main |
| git branch -a | List branches in the repository (and remote repos) |
| git checkout <branch> | Switch to a different branch |
| git merge <branch> | Merge changes from one branch into another |


```
gcc <options> -I <header_dir> <sources> -o <binary>
```

**Useful options**

-Wall : warning all

-g : Compile with debugging symbols

-O3: Compile with optimizations

**Example**

```
gcc -Wall -O3 -I../include/ ../src/main.c -o executable.x
```

sudo apt-get install build-essential

Makefile

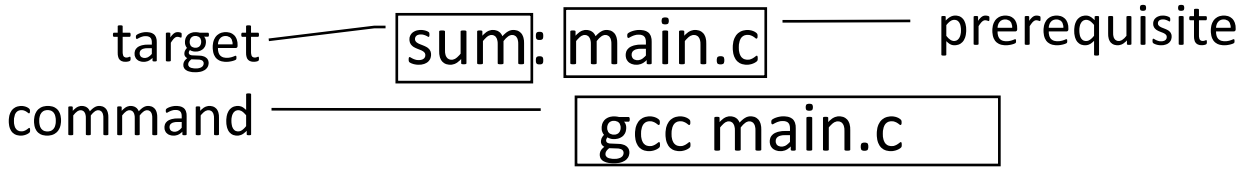
# Make compilation more efficient

## Why do Makefiles exist?

- Makefiles are used to help decide which parts of a large program need to be recompiled
- *gcc source1.c source2.c … sourceN.c -o executable.x*
  - This command recompiles all sources every time!
  - What if N is large and you only modified one source file? Plenty of time wasted recompiling everything.

Makefile

# The essence of a Makefile




- When we run the "make" command , the following set of steps happens:
  - The first target "sum" is selected, because the first target is the default target
  - This has a prerequisite of main.c
  - Make decides if it should run the "sum" target. It will only run if "sum" doesn't exist, or if main.c is newer than sum

Makefile

# Minimal example

Variables

```
NAME=  <exe_name>
CFLAGS=  <gcc_options>
INCLUDE=-I <path_to_header_directory>
SRCS=<source_file1.c> … <source_fileN.c>
```

Replaces the .c extension with .o

```
OBJS=$(SRCS:.c=.o)
```

Link .o object files into an executable

```
$(NAME): $(OBJS)
    @echo "Making executable: "$@
    @$(CC) $^ -o build/$@
```

Automatic variables

Compiles .c files into object .o files

```
%.o : %.c
    @echo "Compiling: "$<
    @$(CC) $(CFLAGS) $(INCLUDE) -c $< -o $@
```

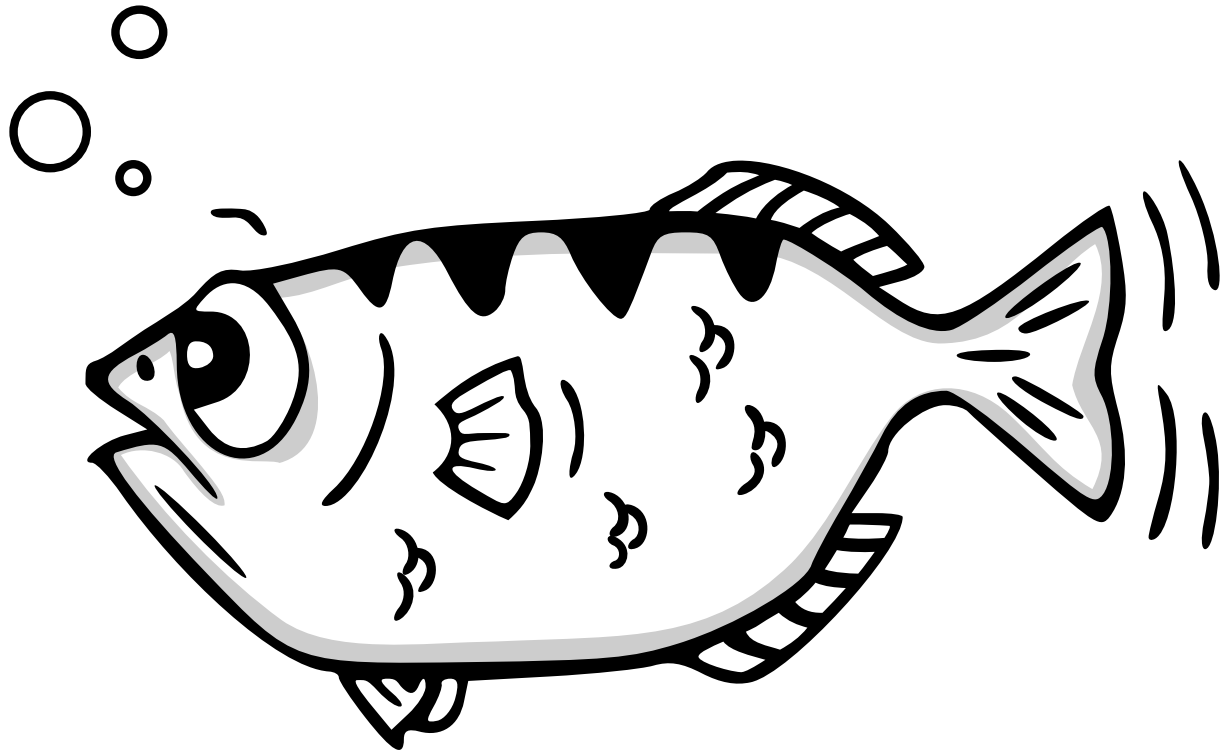
Custom rule to clean objects and executables

```
.PHONY: clean

clean:
    @rm -f src/*.o $(NAME)
    @echo "Removed object files and executable..."
```

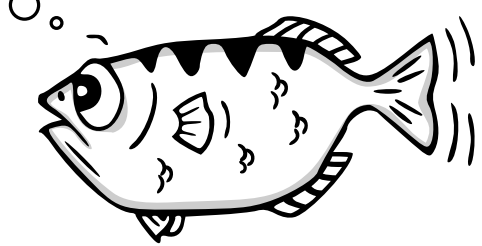
# GDB

- GDB allows you to see what is going on "inside" another program while it executes or at the moment it crashed



sudo apt-get install gdb

# Basic commands of GDB



| Command | Description |
|---|---|
| run or r | Start the program from the beginning |
| break <filepath> or b <filepath> | Set a breakpoint at a specific line or function |
| continue or c | Continue program execution until the next breakpoint |
| step or s | Execute the current line of code and stop at the next |
| next or n | Execute the current line, stepping over function calls |
| finish | Continue execution until the current function is done |
| info breakpoints | List all active breakpoints and watchpoints |
| print <var> or p <var> | Evaluate and print the value of an expression |
| backtrace or bt | Display the call stack of functions |
| quit or q | Exit GDB and terminate the debugging session |

Personal favorite

- Automatically run with args and print the backtrace (in case of failure)

```
gdb -ex r -ex bt --args <executable> <arg1> … <argN>
```

- Doxygen is the de facto standard tool for generating documentation from annotated C/C++ sources


sudo apt-get install doxygen

# Basic commands of doxygen

- Doxygen uses a configuration file to determine all of its settings

To simplify the creation of a configuration file, doxygen can create a template configuration file for you

```
doxygen -g <config-file>
```

Modify the configuration file according  to your needs and run Doxygen

```
doxygen <config-file>
```

# Suggested commenting style

doxygen

Long multi-line comment

```
/**
* text
*/
<Target of comment> (such as a function declaration)
```

Brief single-line comment

```
///text
<Target of comment>
```

More on this