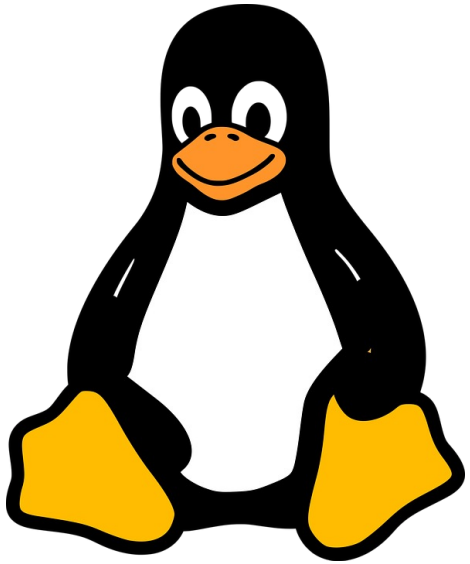
C Development Environment

Samuele Germiniani
samuele.germiniani@univr.it

Outline

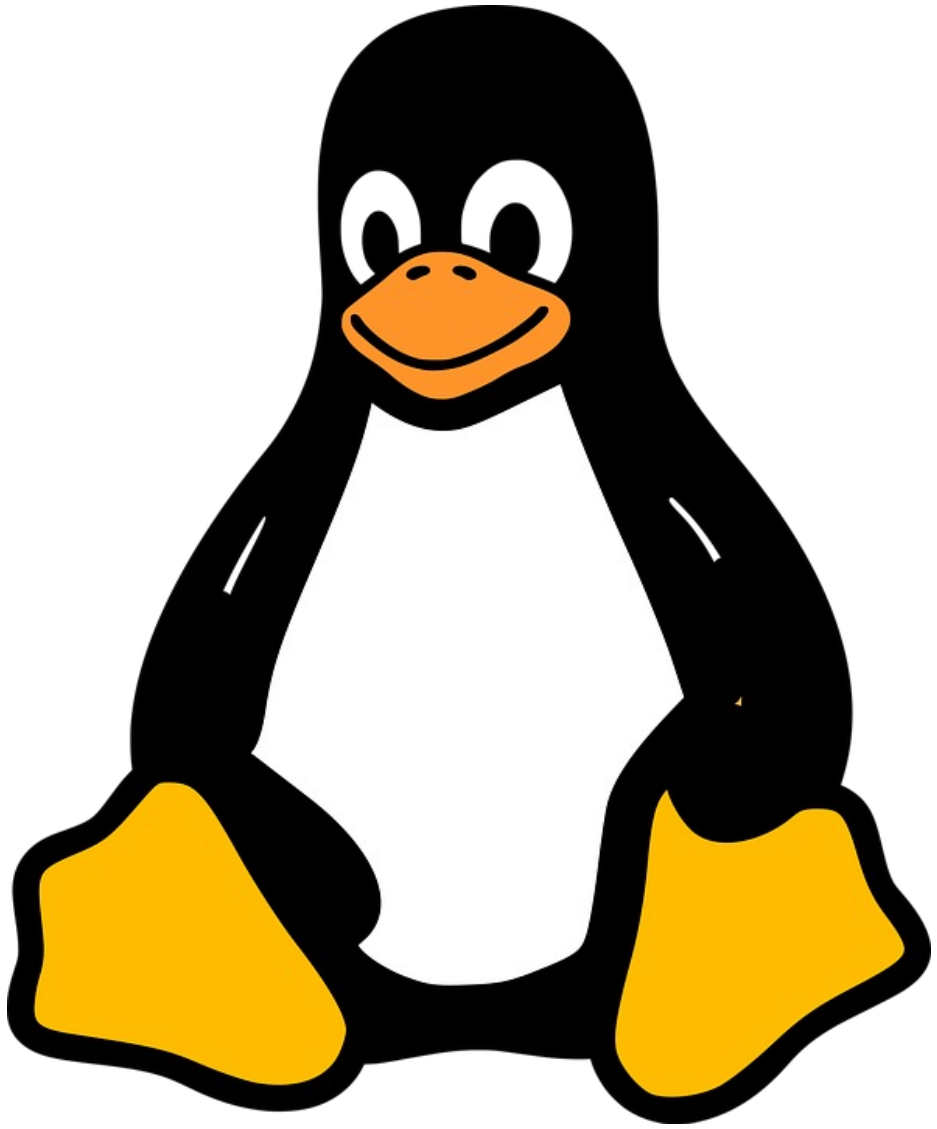
- Introduction
- Overview of tools
- git
- gcc
- Makefile
- gdb
- doxygen



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

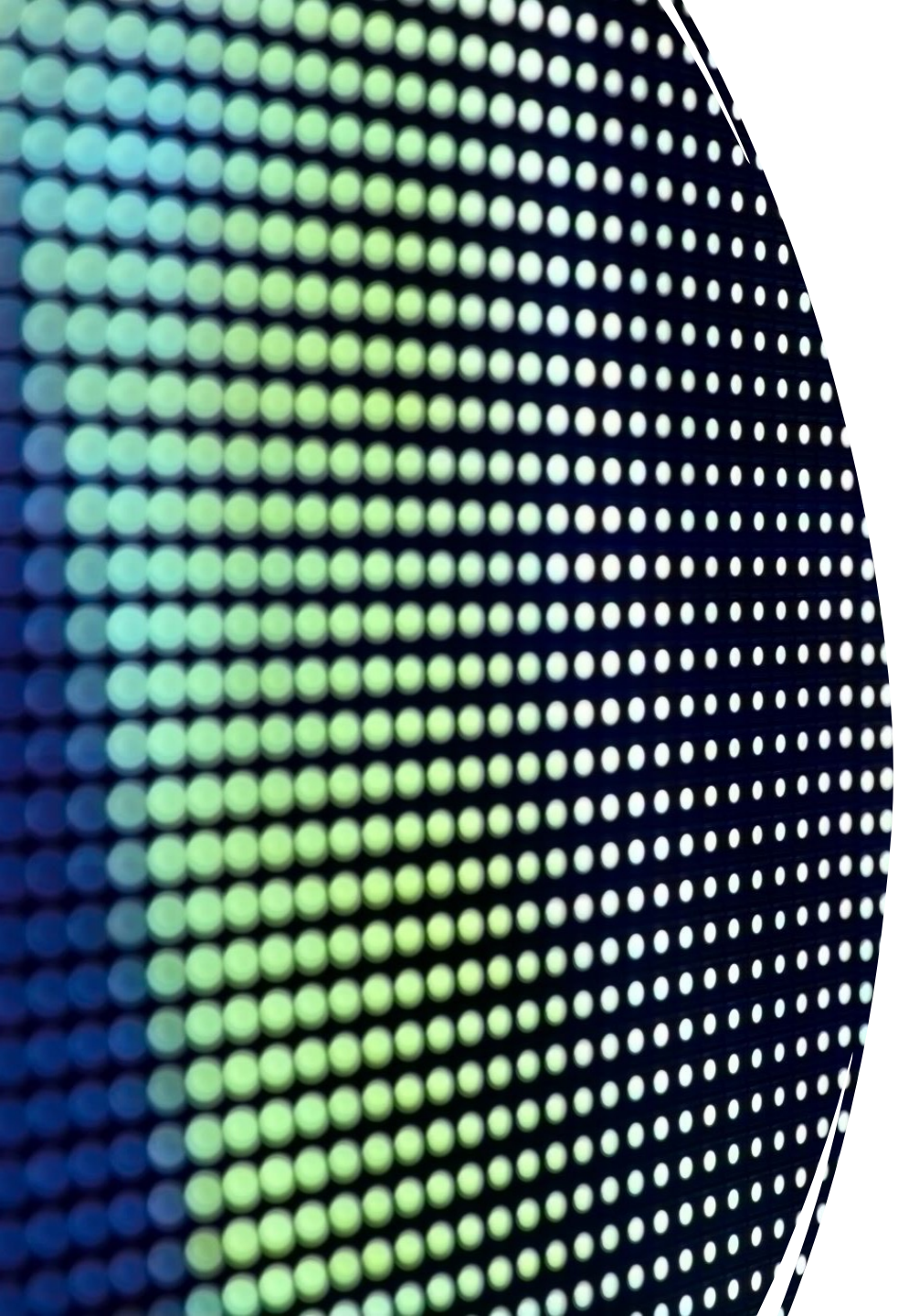
- Debian-based distro is suggested
  - Ubuntu20.04+
- OS on Virtual or Physical machine

IDE minimum features

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





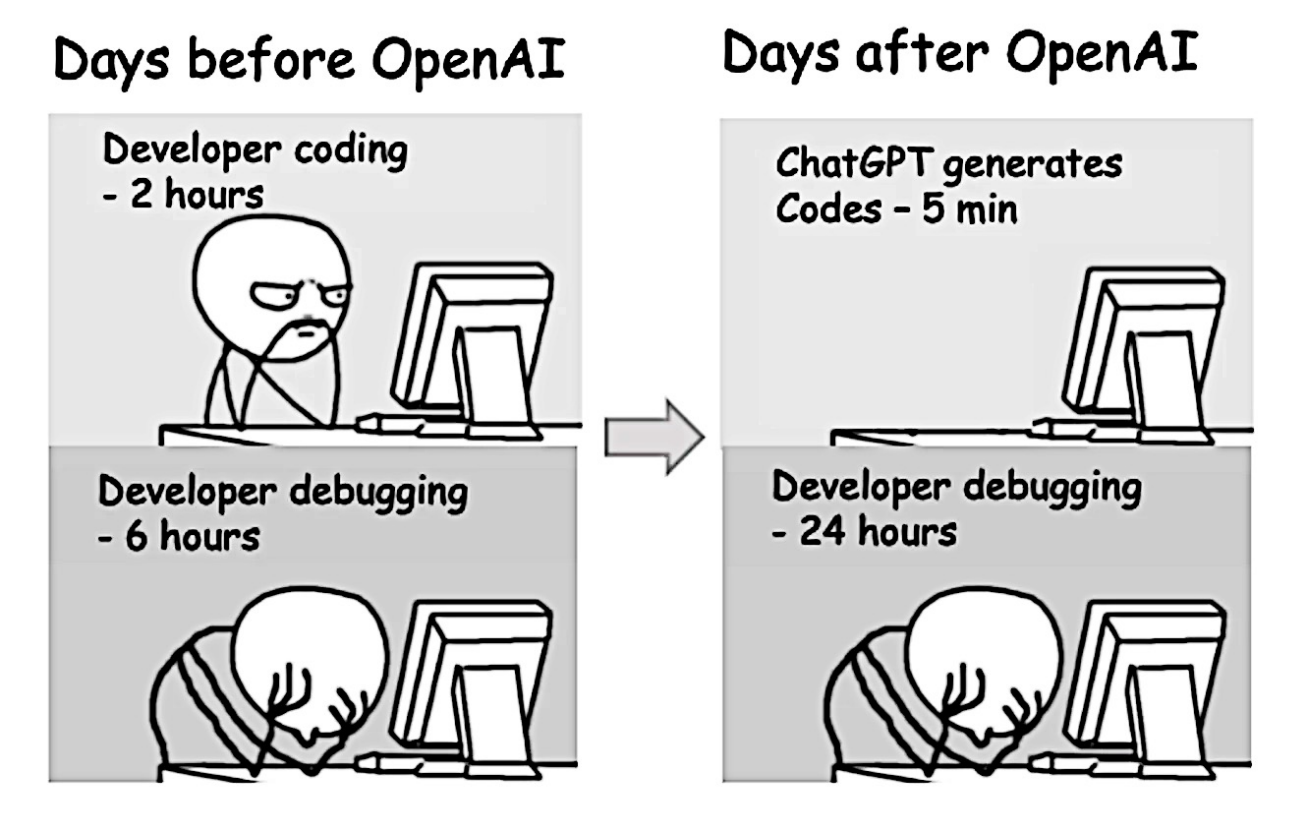



Code indentation

- 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

Suggested option ✓

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

Makefile

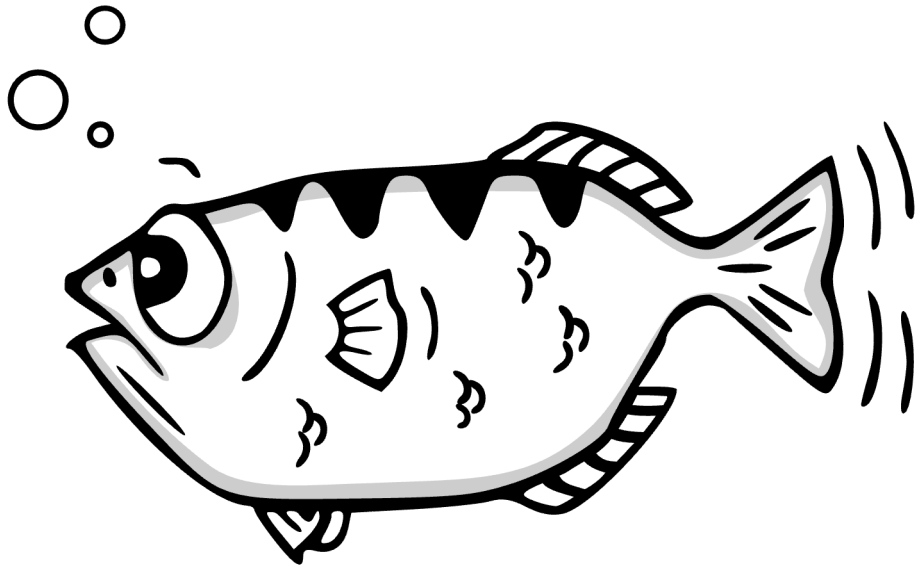
Compiler

High complexity project

Low complexity project

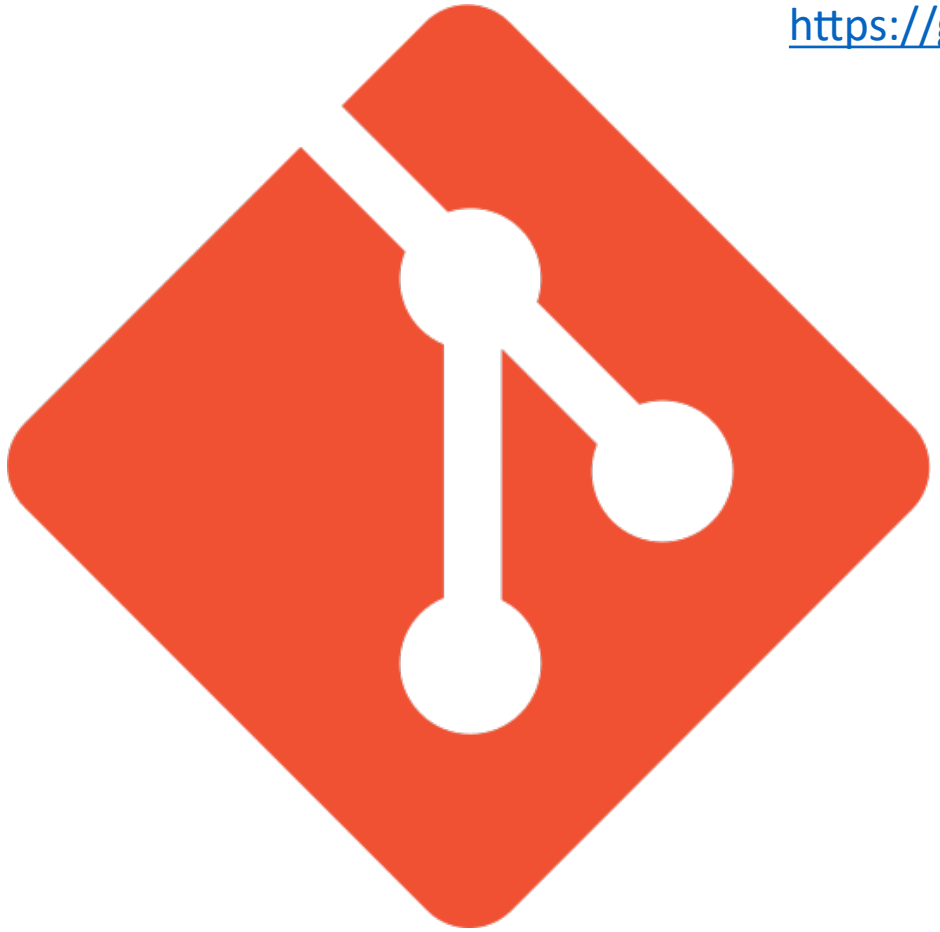
Debuggers

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



Version control

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



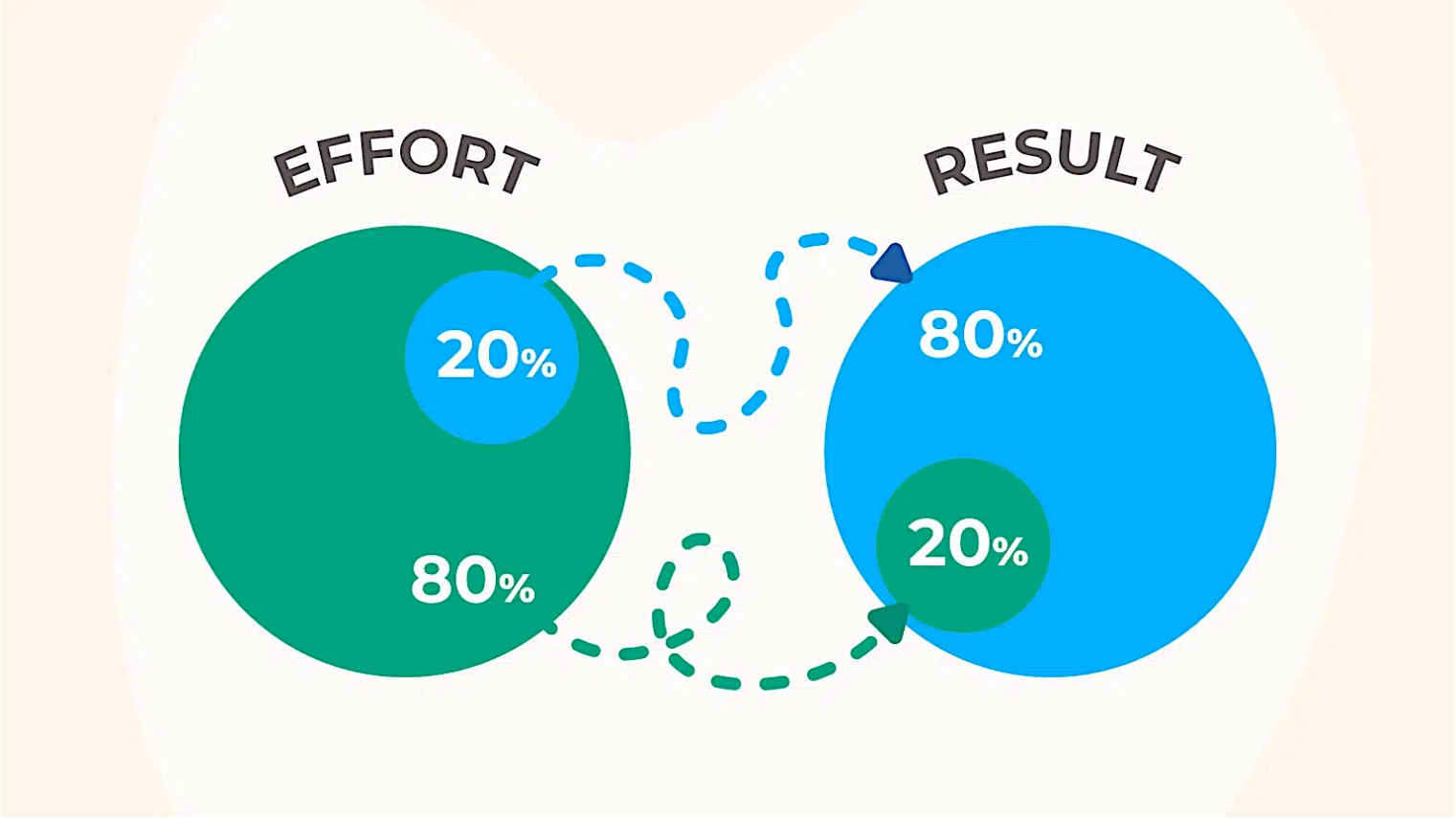
git

doxygen

Documentation

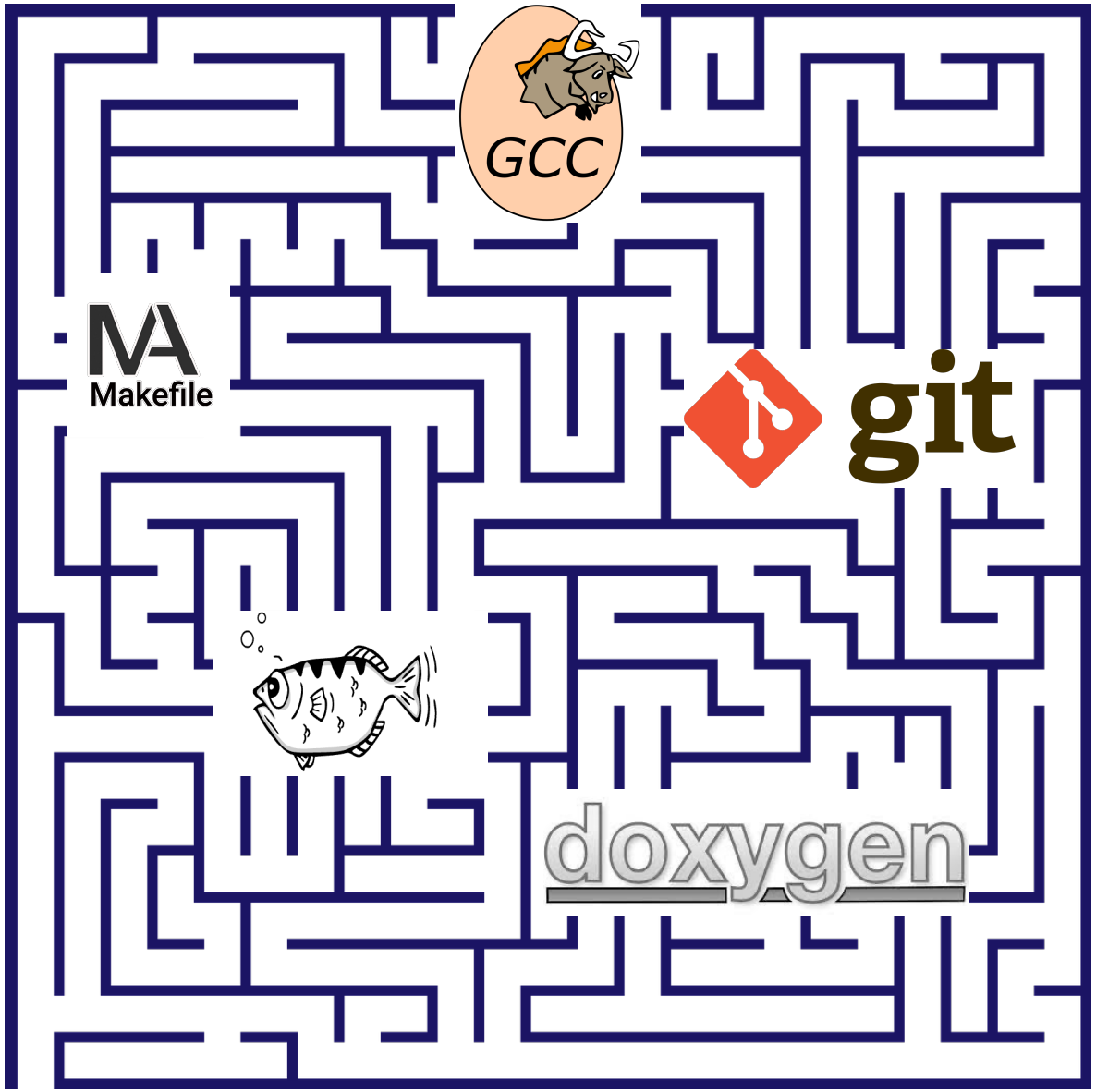
https://www.doxygen.nl/

Pareto principle & Just-in-time learning



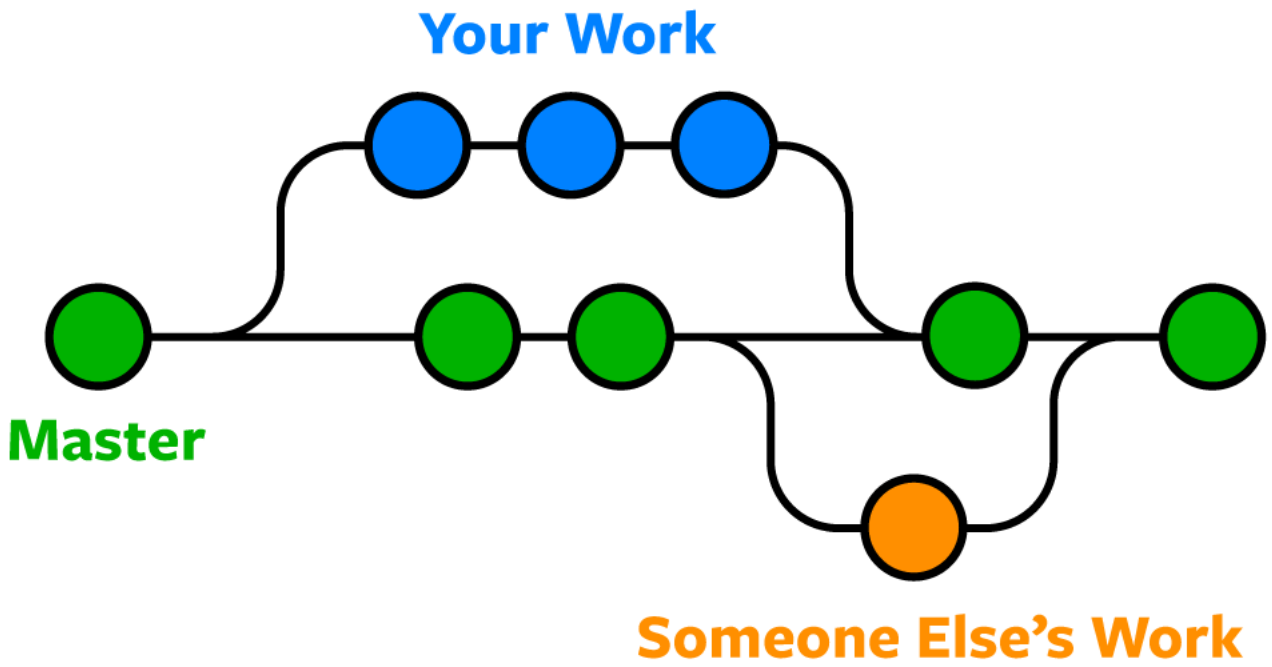
Learn just enough to get you going!

A quick walkthrough



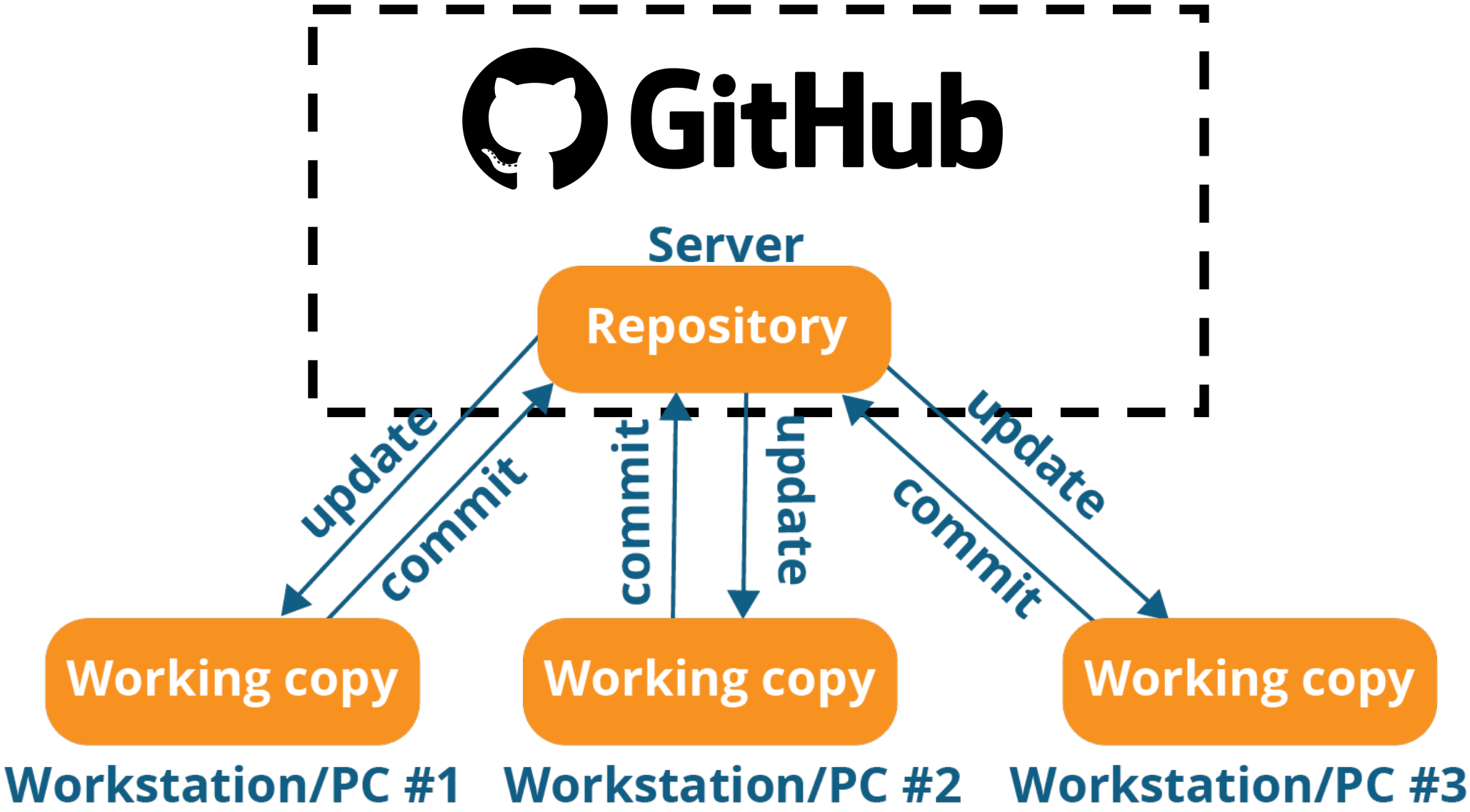
Version control with git

- 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

Exercise part 1 - Steal my repository

1. git clone	• Download the repository from this URL https://github.com/SamueleGerminiani/perm_generator
2. git branch	• List the available remote branches
3. git checkout	• Switch to "dev" branch
4. rm -rf .git	• Remove the git files (the .git directory)

At this stage, you should have a local repo (on your machine) without the references to the corresponding remote repository (on Github)

Exercise part 2 - Upload it to your Github

1. Github	• Create an account on https://github.com/ (unless you already have one)
2. Create a new public repo	• Go to https://github.com/new • Click on "Create Repository" (**Do not add a README or .gitignore**)
3. Follow the guide	• …or create a new repository on the command line

At this stage, you should have both a local repository (on your machine) and a remote repository (on github)

Exercise part 3 - Commit & Push

1. Edit a file	• Change the author in the main.c file to your name	Local
2. git status	• Use git status check which files where modified	
3. git diff	• Check the differences (is it what you expected?)	
4. git add	• Add the changes to the staging area	
5. git commit	• Commit the changes to your local repo	
6. git push	• Update the remote repo	Remote

At this stage, the remote repository should contain the changes of the local repository


```
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
```

Exercise - Simplest compilation

gcc

- Use gcc to compile the source code in "src" with the headers in "include" of the stolen directory

Makefile

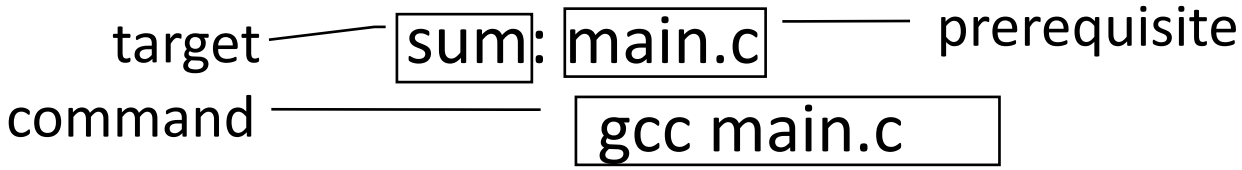
Make compilation more efficient

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

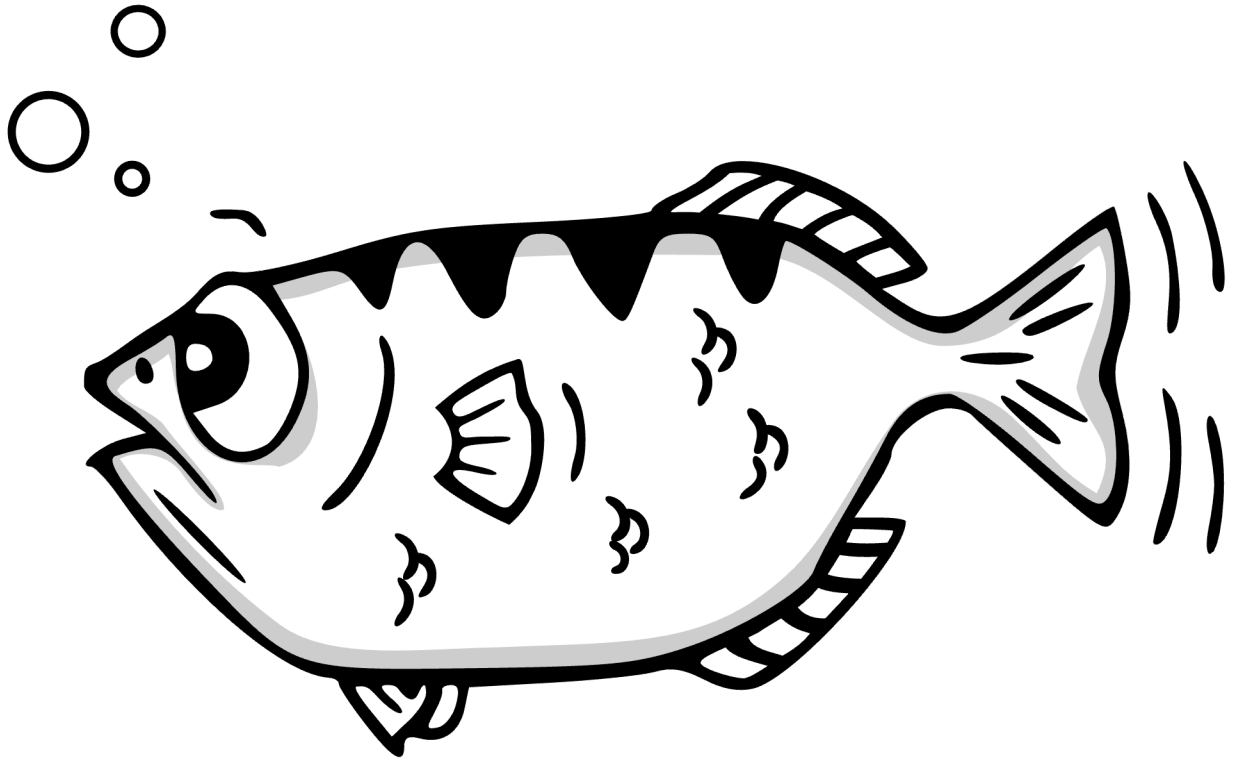
```
.PHONY: clean

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

Exercise - Compilation with Makefile

Makefile	• Fill the missing parts in the Makefile of the stolen directory
Build	• Compile using the "make" command
Run	• Execute the program

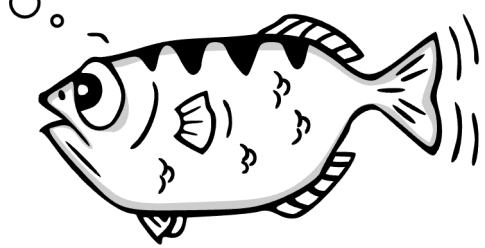
GDB

- 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



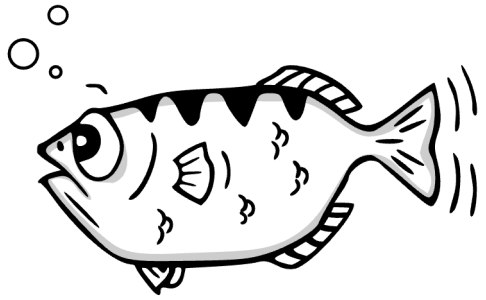
Command	Description
run or r	Start the program from the beginning
break <filepath> or b <filepath>	Set a breakpoint at a specific line or function
continue or c	Continue program execution until the next breakpoint
step or s	Execute the current line of code and stop at the next
next or n	Execute the current 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>
```

Exercise - Debugging with GDB



The source code in the stolen repository contains a bug:
running the program generates a "segmentation fault"

- Use gdb to find and correct the bug
- Hint: in most cases, inspecting the stack trace is enough to understand the bug

N.B. The project must be compiled with the -g flag of gcc

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

doxygen

```
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

Exercise - Generating documentation with doxygen

Config file	• Generate the default config file
Edit the config file	• Find the "OUTPUT_DIRECTORY" option in the config file and change it to *doc*; change the "EXTRACT_ALL" to *YES* and "RECURSIVE" to *YES.*
Add comments in doxygen format to the source code	• Comment the functions "generatePermutations" and "swap" in the perm.h and swap.h files of the stolen directory
Run doxygen	• Generate the documentation with the generated config file
Explore the generated documentation	• Open the doc/html/index.html file using a web browser