[ENVIRONEMENT SETTING UP](#env_setup)

[GCC](#gcc) =

compile results

compile options

compile staitc library

compile dynamic library

[GDB](#gdb) =

examine

backtrace

list, disassemble

break

print

[VSCODE](#vscode) =

installing on linux

useful features

[GIT and GITHUB](#git_github)  =

schemes legend

installing

git workspace

repo basic

branch

diff

link local to remote

remote

push

[CLANG-FORMAT](#clang_format) =

[BASH](#bash) =

basics

MAKE = [link](https://habr.com/ru/post/477656/) , [link](https://habr.com/ru/post/155201/) , [link](https://yandex.ru/search/?text=make+tutorial&clid=2270455&banerid=5000004765%3A60be83d194172c001e64abb3&win=493&lr=2)

[CMAKE](#cmake) =

installing on linux

[GTEST/GMOCK](#gtest) =

installing on linux

[XTERM and SIXEL](#sixel) =

preparing

control sequences

colors using

one unit description

**ENVIRONEMENT SETTING UP**

List of necessary tools

GCC, G++

GDB

VSCODE

GIT

CLANG\_FORMAT

CMAKE

GTEST/GMOCK

Optional tools (for single projects)

XTERM (for sixel)

SIXEL

**GCC**

*more about: official site -* [*link*](https://gcc.gnu.org)

*official manual -* [*link*](https://gcc.gnu.org/onlinedocs/gcc/)

*optimization options -* [*link*](https://wiki.gentoo.org/wiki/GCC_optimization/ru)

**\*\*\*\*\* COMPILE RESULTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

- preprocess only \*.c gcc -E

- compile only without assemble \*.s gcc -S

- compile to object file without link \*.o gcc -c

- compile to binary file \*.out

-fsyntax-only ; only syntax without compiling at all

- compile project

g++ –c main.cpp lib.cpp ; = main.o lib.o

g++ main.o lib.o –o filename.out ; = executable file

**\*\*\*\*\* COMPILE OPTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

- dinamically linked positions

-pie ; default

-no-pie ; appropriate for debugging

- debugging inforamtion

-g0 ; default (no debug info)

-g1 ; minimum info

-g ; standart info

-g2 ; maximum info

- optimizations

-O0 ; default (debugging optimization)

-O1 ; min = without comlile time increasing

-O2 ; recommend (but –ftree-vectorize is turned off)

-O3 ; not recommend

-Os ; optimize by size (for min memory usage)

-Og ; is O2 but without flags conflict with debugging

- other

-H ; print recursively all of libs directories

**\*\*\*\*\* COMPILE STATIC LIBRARY \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Project consists of: main.cpp and some \*.hpp ⬄ \*.cpp

; main.cpp ; project directory

1: #include "\_1.hpp" a.cpp , b.cpp , c.cpp

2: #include "\_2.hpp" d.cpp , e.cpp

When build project compiler:

compile \*.cpp to \*.o files main.o , a.o , b.o ... etc.

link these files to find definitions of what declared in \*.hpp

So linking step look like:

-$ g++ main.o a.o b.o c.o …

For simplicity it is possible to join all object files into one lib

-$ ar rc libname.a a.o b.o c.o …

-r ; replace existing or insert new file into lib

-c ; do not warn if lib is created

lib\*.a = required format for library \*\*

\*\* it might have another format if using 'gcc –soname'

Then compile might looks like this:

-$ g++ main.cpp –lname ; if lib in /usr/local/lib

-$ g++ main.cpp –lname –L<path> ; if lib in other place <path>

**\*\*\*\*\* COMPILE STATIC LIBRARY \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<https://www.techytalk.info/c-cplusplus-library-programming-on-linux-part-two-dynamic-libraries/>

**GDB**

*more about: gdb helper -* [*gdb-dashboard*](https://github.com/cyrus-and/gdb-dashboard)

**\*\*\*\*\* EXAMINE (исследовать) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

(gdb) x / \_ \_ \_ \_

^ ^ ^ ^

| | | |

| | | адрес 'откуда'

| | размер отображаемых данных 'чего'

| формат, в котором отображать данные 'как'

сколько раз считать по указанному размеру 'сколько'

(gdb) x/2xw $rip+4 = вывести откуда : из адреса [rip + 4]

сколько : 2

чего : слова (4 байт)

как : в hex формате

'откуда' = 0x... (непосредственно адрес), $base+offset

'сколько' = любое целое число, по умолчанию = 1

'чего' = байты : b (1), h (2), w (4), g (8)

'как' = числа : t (2), o (8), u (10), x (16)

другое : с (char), s (str), i (instr)

**\*\*\*\*\* BT (backtrace) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

В любой точке, где находится программа (точка останова или просто очередная инструкция), команда bt выведет цепочку вызовов функций. Применяется для исследования стека, на котором хранятся адреса возвратов в вызывающую функцию.

(gdb) bt

* #0 \_f (arg=1) at file.c:6
* #1 0x0000000000401181 in main () at file.c:17

Расшифровка:

порядок то большего к меньшему (#0 = текущая функция)

\_f (int arg) превратилось в \_f (arg=1) (в функию передали 1)

\_f( ) , main( ) = названия функций

0xADRESS = адрес возврата из функции

file.c:\_\_ = номер строки в исходном файле

(gdb) bt full // полная информация о фреймах стека

**\*\*\*\*\* LIST and DISASSEMBLE (дизассемблировать) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Команда list выводит список строк исходного кода на

N строк выше FROM

N строк ниже FROM

указанного по уvолчанию места.

(gdb) show listsize ; show default n

(gdb) set listsize <n> ; number or "unlimited"

(gdb) list <from> ; line number / function name

(gdb) list - ; lines before last printed

(gdb) list + ; lines after last printed

(gdb) list <from>,<to> ; lines exactly

; ',' is necessary, from/to is optional

(gdb) disassemble \*function\* ; короткое = disass

В качестве аргумента команды используется имя функции

**\*\*\*\*\* BREAK (точка останова) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

(gdb) break \*argument\*

В качестве аргумента может использоваться:

- имя\_функции

- имя\_исходного\_файла:номер\_строки

(gdb) break main

(gdb) break file.c:10

**\*\*\*\*\* PRINT (печать) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

(gdb) print \*argument\*

При использовании команды print отладчик сохраняет аргумент в переменную с имененем $N , где N – ее номер (начинается с 1). Допустим, есть некоторый указатель:

1. int i = 1;
2. int\* ptr = &i;

(gdb) print ptr

* $1 = (int \*) 0x7fffffffdeac ; тип и содержимое ptr

(gdb) print &ptr ; взятие адреса работает аналогично языку С

* $2 = (int \*\*) 0x7fffffffdeb0 ; адрес самого ptr

(gdb) print \*ptr ; разыменование работает алогично языку C

* $3 = 1 ; значние переменной, на которую указывает ptr

Поведение print будет немного отличаться, если типом данных, на

которые указывает указатель, будет строка:

1. char s[20] = "Hey!";
2. prt\* str = s;

(gdb) print str

* $4 = 0x7fffffffdec0 "Hey!"

; адрес внутри str сразу интерпретируется

(gdb) print &str

* $5 = (char \*\*) 0x7fffffffdeb8

; адрес самого указателя str

(gdb) print \*str

* $6 = 72 'H' ; значение непосредств. байта по адресу из str

**VSCODE**

*more about: vscode main -* [*link*](https://code.visualstudio.com)

*build* *with cmake* -[*link*](https://github.com/microsoft/vscode-cmake-tools/blob/main/docs/README.md)

**\*\*\*\*\*** **INSTALLING** **ON LINUX** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

instruction = <https://code.visualstudio.com/docs/setup/linux>

-$ code --help

-$ code [options] <file>

getstarted = <https://code.visualstudio.com/docs/getstarted/>

settings\_global = $HOME/.config/Code/User/settings.json

settings\_project = ./project/.vscode/\*setting\*.json

C++ = <https://code.visualstudio.com/docs/cpp/introvideos-cpp>

**\*\*\*\*\*** **USEFUL FEATURES** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

SH = shift

CC = cursor

firstly 🡺 secondly

; hotkeys

выделить весь текст между { } SH + ALT + CC

выделить только столбцы CC 🡺 SH + ALT

; manual region

#pragma region

код регион

#pragma endregion

SH + F12 ; show all of inclusions in all of files

F2 ; rename inclusion in current file

**\*\*\*\*\*** **BUILD and DEBUG OPENED UNIT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

добавить task = Terminal 🡺 Configure Default Build Task

появится .vscode/task.json

построить проект = Terminal 🡺 Run Build Task

для отладки настроить launch = .vdcode/launch.json

любой запуск программы происходит в соответствие с этим файлом

**GIT and GITHUB**

more about: trainer - [link](https://learngitbranching.js.org/)

step by step manual - [link](https://githowto.com/ru)

bool - [link](https://git-scm.com/book/ru/v2)

**\*\*\*\*\* SCHEMES LEGEND \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

= loc. master pointer

= rem. master pointer

= commits

\* = pointer (HEAD)

loc = local

rem = remote

<alias> = name of remote repository

cur = current

<com> = command

<current> = branch\* or HEAD

<num> = commit number

<br> = loc. rundom branch

<obr> = rem. rundom branch

<file> = particular file

<what> and <where> = branch, hash or HEAD - if <where> is in command template, but absents in really command, it means that <current> is used impicitly

**\*\*\*\*\* INSTALLING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

-$ sudo apt install git

-$ git config --list ; info

-$ vim ~/.gitconfig ; file with configs

; set user name for every commit

-$ git config --global user.name Daniil

; set user e-mail for every commit

-$ git config --global user.email tr1ang1e.goog1e@gmail.com

**\*\*\*\*\* GIT WORKSPACE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**WORKSPACE**

**HEAD**

**INDEX**

a

b

c

git add a b

a

b

c

**a**

**b**

git commit

\*now is empty\*

**a**

**b**

**с**

\*previous head\*

Обратить внимание : GIT = система контроля версий (а не файлов). Наша задача = каждый раз (!) указывать ему, версию какого файла нужно обновить.

Если мы уже положили файл в репозиторий, он будет находиться там даже в том случае, если мы больше ниразу не обновляли его и даже если мы его удалили из рабочего пространства.

В примере выше, таким файлом может быть "c". Он тянется в ветке потому, что был когда-то добавлен. Даже если мы удалим его из workspace, он все равно останется в системе контроля. Чтобы удалить его еще и из системы контроля, нужно добавить его в индекс и потом сделать коммит.

**\*\*\*\*\* REPO BASIC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

-$ git <com> --no-color ; don't color console output

-$ git log ; commit history

-$ git status ; info about current status

; changes in files

; added and not added files

; loc. master vs rem. master

; process of adding file to next commit is **indexing (staging)**

-$ git add <obj> ; add obj to index

--all ; add all objects to index

; remove object from not yet committed index

-$ git rm –-cached <obj> ; obj is still in the work space

; remove object from workspace and repository = variant 1

-$ rm <obj> ; remove object manually

-$ git add <obj> ; notify git about changes

-$ git commit –m "..." ; make commit

; remove object from workspace and repository = variant 2

-$ git rm <obj> ; remove from everywhere

-$ git commit –m "..." ; make commit

; remove object from repository but keep safe in workspace

-$ echo "<obj>" >> .gitignore ; add to .gitignore

-$ git rm --cached --ignore-unmatch <obj> ; remove from repo

-$ git commit –m "..." ; make commit

-$ git reset ; ...

**\*\*\*\*\* BRANCH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Каждая ветка представляет из себя "копию" того состояния репозитория, из которого она ответвляется. Соответственно, и весь набор объектов попадает в эту ветку. Ветка = параллельный проект целиком!

**A**

**K**

**B**

**C**

**x**

**x**

**y**

**L**

**x**

**y**

**i**

**x**

**y**

**j**

**x**

**y**

**master**

**feature**

Имя ветки = всегда является указателем на последний коммит в ней

-$ git branch ; info about branches and \*

-$ git checkout <br> ; move \* to the <br>

! don't checkout if non-commited objects

-$ git branch <br> ; create <br> at current \*

-$ git branch <br> <where> ; create <br> at <where> place

-$ git checkout -b <br> <where> ; both at the same time

; detach \* from <br> + attach \* to commit

-$ git checkout <current>

**master**

**X**

**master**

**X**

**\*\*\*\*\* DIFF \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Command to figure out defferences between two git entities

-$ git diff ; curr\_file vs file\_in\_nearest\_index

--staged ; curr\_index vs last\_commit

--cached ; curr\_file, curr\_index vs last\_commit

-$ git diff <what1> <what2> <file> ; hashes or branches

-$ git diff –U0 ; exclude context lines

**\*\*\*\*\* LINK LOCAL TO REMOTE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

At first create github perository named "example" …

; create local repo

-$ git init

; link local repo with remote / default remote alias = 'origin'

-$ git remote add <alias> git@github.com:tr1ang1e/example.git

; add first commit to remote repo

-$ git add <obj>

-$ git commit –m "readme"

-$ git push –u <alias> master

LOCAL REPO

REMOTE REPO

alias = ORIGIN

**A**

**A**

**\*\*\*\*\* REMOTE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

; see remote repos

-$ git remote

-v; see remote url

; rename remote repository alias

-$ git remote rename <rem.alias> <new.alias>

; set or change remote url

-$ git remote set-url <rem.url>

--push ; only for push

; remove link with remote

-$ git remote rm <rem.alias>

**\*\*\*\*\* PUSH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

-$ git push <alias> <br\_loc>:<br\_rem>

–u ; link branches and just -$ get push in future

**A**

**A**

**B**

**C**

**A**

**B**

**C**

**A**

**B**

**C**

>> will create <br\_rem> if doesn't exists

>> -$ git push <alias> <br\_loc> ; if <br\_loc> == <br\_rem>

>> -$ git push <rem> :<br\_rem> ; push nothing = remove <br\_rem>

**CLANG-FORMAT**

*more about: git + clang-format combination -* [*link*](https://habr.com/ru/post/433832/)

INSTALL -$ sudo apt install clang-format

RUN

; by default read from stdin write to stdout

; before and after cannot be the same file !

-$ clang-format [options] <before> > <after>

CLANG-FORMAT CONFIGURATIONS FILE

; create with pre-config (gnu format here)

-$ clang-format -style=llvm -dump-config > .clang-format

name .clang-format

dir same as a formattimg file

syntax YAML ([link](https://tproger.ru/translations/yaml-za-5-minut-sintaksis-i-osnovnye-vozmozhnosti/))

comment #

empty lines allowed

configs key : value

non-config epsent at all or false value

CONFIG DESCRIPTION = [link](https://github.com/llvm-mirror/clang/blob/master/docs/ClangFormatStyleOptions.rst)

OPTIONS DESCRIPTION = [link](https://clang.llvm.org/docs/ClangFormat.html)

VSCODE EXTENSION = [link](https://marketplace.visualstudio.com/items?itemName=xaver.clang-format)

PROTECT CODE FROM FORMATTING

1. \*formatted\_code\*
2. // clang-format off
3. \*unformatted\_code\*
4. // clang-format on
5. \*formatted\_code\*

/\* clang-format off \*/ to /\* clang-format on \*/ is also possible

**BASH**

*more about:*

**\*\*\*\*\*** **BASICS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

СКРИПТ = интерпретируемый

ОБЯЗАТЕЛЬНАЯ СТРУКТУРА СКРИПТА

- расширение скрипта : .sh

- первая строка скрипта : #!/bin/bash (или -$ whereis bash)

- пробелы являются разделителями!

ПЕРЕМЕННЫЕ

- var=1 ; без типов

- echo $var ; для использования нужен знак $

ВЕТВЛЕНИЕ

1. if [ $var –lt 1 ]
2. then
3. echo "less"
4. elif [ $var –eq 1 ]
5. then
6. echo "equal"
7. else
8. echo "greater"
9. fi

WHILE\_ЦИКЛ

1. while [ $var –lt 10 ]
2. do
3. echo "current value = &var"
4. (( var++ ))
5. done

FOR\_ЦИКЛ

1. for (( i = 0; i < 10; ++i ))
2. do
3. echo "$i"
4. done

СИНТАКСИС = обратить внимание, что типичный bash\_синтаксис может быть заменен привычным Си. Однако типы данных все равно не указываются. Также неверно, что всегда Си синтаксис, может быть заменен синтаксисом bash.

**CMAKE**

*more about: official site –* [*link*](https://cmake.org/download/)

**\*\*\*\*\* INSTALLING** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

; install using apt

-$ sudo apt update

-$ sudo apt install cmake

; install using downloaded archive

1. unpack archive

2. read README.rst

3. if error with openssl:

-$ sudo apt install openssl

-$ sudo apt install libssl-dev

**\*\*\*\*\* xxx** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**GTEST / GMOCK**

*more about: official git repo –* [*link*](https://github.com/google/googletest)

*getting started, primer –* [*link*](https://google.github.io/googletest/primer.html)

*gtest and cmake –* [*link*](https://www.kaizou.org/2014/11/gtest-cmake.html) *,* [*link*](https://raymii.org/s/tutorials/Cpp_project_setup_with_cmake_and_unit_tests.html)

**\*\*\*\*\* INSTALLING** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Release version here - [link](https://github.com/google/googletest/releases)

Installing manual - [link](https://github.com/google/googletest/blob/main/googletest/README.md)

1. install as 'standalone\_cmake\_project'

2. \*.a libs in /usr/local by default

3. to use as static library:

#include <gtest/gtest.h>

compile with c++ link options –l<library>

**\*\*\*\*\* GTEST STRUCTURE** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<https://google.github.io/googletest/primer.html>

TEST PROGRAM

ASSERTIONS

TEST

TEST SUIT

TEST FIXTURE CLASS

**whether the condition is true**

**must be only inside tests**

- **success** = is true

- **nonfatal failure** = continues normally

- **fatal failure** = abort current function

**verify the tested code behavior**

- **failure** = test crashed or assertion failed

- **success** = otherwise

**tests reflected tested code structure**

might be used if multiple tests in a

test suit share objects or subroutines

contains multiple test suites

**SIXEL**

*more about: documentation -* [*link*](https://vt100.net/docs/vt3xx-gp/chapter14.html)

*libsixel -* [*link*](https://github.com/saitoha/libsixel#video-streaming)

**\*\*\*\*\* PREPARING** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Should have XTERM terminal emulator

-$ sudo apt install xterm

Run xterm with both of options

-$ xterm

-xrm "XTerm\*decTerminalID: vt340"

-xrm "XTerm\*numColorRegisters: 256"

**\*\*\*\*\* CONTROL SEQUENCES** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

chars descri-гption

\x1bPq enter sixel mode (<esc> + P + [options] + q)

\x1b\\ leave sixel mode (<esc> + \)

& x

~ x

**\*\*\*\*\* COLORS USING** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\* ONE UNIT DESCRIPTION** **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***