General terminal commands:

Disable ASLR on your system until next reboot: **`echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`**
Enable ASLR on your system again: **`echo 2 | sudo tee /proc/sys/kernel/randomize_va_space`**
gives information about the file, e.g. 32 bit vs. 64 bit **`file <binary>`**
Shows dynamic libraries which are loaded by the binary and their file-path **`ldd <binary>`**

gdb / peda

`disas <function>`	Disassembles code	`disas main`

`break`
`b`
	Sets a breakpoint
	- when debugging your exploit, set the breakpoint on return!
		`break *main+117`
		`b     *main+117`

`run`
`run < <input-file>`
	runs the binary	`run`
		`run < payload.bin`

`ctrl+c`	Stops the execution	
`c`	continue execution until next stop	
`ni`	"next instruction", next instruction line (steps over function calls)	
`si`	"step into", next instruction, but steps into function calls	
`checksec`	Shows which security features are turned on/turned off	

`vmmap`	Shows memory mapping (during execution)	`run`
		`break with ctrl+c`
		`vmmap`

`aslr on`	Turns aslr in gdb on	
`pattern create <number>`		`pattern create 70`
`pattern offset <pattern>`	Take the pattern you find in EIP (64 bit: RIP does not load the overflown pattern, take RSP)	`pattern offset AA(A`

Important addresses and offsets inside a binary or the libc:

	Libc base	Offset `system`	Offset `"/bin/sh"`
Command line	`ldd ./binary`	**readelf -s** /path/to/libc **| grep system**	**strings -tx** /path/to/libc **| grep /bin/sh**
gdb-peda	⇒ `run` ⇒ **vmmap**	⇒ `run` absolute address (if ASLR is disabled): ⇒ **`p system`**	⇒ `run` absolute address (if ASLR is disabled): ⇒ **searchmem /bin/sh**
Hopper/ IDA		**search in labels** for `system`	**search in Strs** for `"/bin/sh"`

Command line tricks:

store a payload that spawns a shell into a file, and provide it as input to the vulnerable binary and keep stdin open so the shell does not exit:

```
./exploitscript.py > payload.bin
cat payload.bin -| ./01_exercise
```

pwntools:

`from pwn import *`	to use pwntools in python	
`p32(<integer>)`	convert 32 bit integer to little endian bytestring	`p32(0x7fabc)`
`p64(<integer>)`	convert 64 bit integer to little endian bytestring	`p64(0x7fabc)`

ROPgadget:

```
ROPgadget --binary <binary>
```

Hopper:

`File → Read executable to disassemble`	Open binary to disassemble
`Searchfield for labels`	Search for functions (or sections, like GOT)
`Navigate → show section list → search for section`	Search for sections (like e.g. PLT)