Knowledge

Juan Pablo Prada

June 6, 2025

Contents

Containers **5**
- Important link . . . 5
- Notes . . . 5
- Networking . . . 5
- Bash Abstractions . . . 5
- djeada/Bash-Scripts . . . 5
- Text Processing . . . 5

Git **6**
- Notes . . . 6

Chapter_2 [Pointer Manipulation] **7**

Analysis of algorithms **9**

Data structure part **10**

Nand2tetris **11**
- Notes . . . 11

Git **12**
- Notes . . . 12
- git diff . . . 12
- basic-commit . . . 12
- basic-staging . . . 12
- basic-branching . . . 12
- Rebase branch . . . 12
- Basic revert . . . 12
- git reset . . . 13
- git clean . . . 13
- amend . . . 13
- advance rebase . . . 13
- git stash . . . 14
- git rm . . . 15
- git submodules . . . 15
  - Things to check . . . 15

Technical writting **16**
- Notes . . . 16
- Ideas . . . 16
- Why . . . 16
- gns3 . . . 16
  - Notes . . . 16

Virtualisation **17**
- Important link . . . 17
- What is . . . 17
- Notes . . . 17
- Usage . . . 17
  - Creating disk . . . 17
  - Creating vm . . . 17
  - Vm networking (Libvirt/Bridge) . . . 17
  - . . . 18
  - Give networking to vm without libvirt . . . 18

Makefiles **20**
- Notes . . . 20
- Link . . . 20
- ch_1 . . . 20

Containers

Important link

- docker network

Notes

- as vms but faster is above the kernel so if linux container then linux pc is basically using linux features such a cgroups and namespaces for isolating all processes

Networking

Bash Abstractions

Reading bash scripts from different sources and understand every-single line

- djeada/Bash-Scripts
- mertcangokgoz/UsefulBashScripts

djeada/Bash-Scripts

- disk_usage
- set -euo pipefail: ** bashDjeadaAbstraction **
  - I understand why the **-e**, so basically if there is a bashscript which have some pipes and one of those pipes return 1 it will exit the script inmediately if not set even if the pipes return 1 it will continues, if the 1 comes from an if then it will continue
  - For the -u flag I need to ask, not understood yet
  - For the -o flag, This will set the option corresponding to the option name in this case pipefail
  - The option pipeline needs to be used as "' #! /usr/bin/bash set -(.*)o pipefail

```
this will return 0 if all commands in an pipelined command ex: .*|.*|.......| return 0 or the right
- global variables defined as Mayus
- the cat <<EOF is useful when working with multilines text in bash ex: assigning multiline text t
- df, shows filesystem usage
- awk is used one line to understand is the following
```

local awk_script='NR>1' awk_script="$awk_script && $1 !~ /^tmpfs/ && $1 !~ /^udev/ && $1 !~ /^devtmpfs/"

```
- this script: is making use of awk pattern matching and logical operators
  so basically is doing for $awk_script="NR>1" then from the following input remove the first line
  then, once that done get the lines tha doesn't match the pattern /^tmpfs/ and /^udev/ and /^devtm
```

Text Processing

Learning text processing with bash, by understanding the coreutils.

- Selective editing -> https://learnbyexample.github.io/learn_gnused/selective-editing.html#regexp-filtering
- Check for ./sed.sh for see the actual examples of each concept

Git

Notes

Chapter_2 [Pointer Manipulation]

- Dangling pointers -> pointers that points to invalid addresses.
- When declaring a pointer variable only the pointer variable gets allocation not for the data it points.
- the data it points could be allocated in 2 ways
- Automatic alloc and dealloc by function or block
- Dynamically (Manual) with malloc or realloc
- ```
  //Dangling pointer
  int f(int **iptr) {
      int a = 10;
      *iptr = &a;
      return 0;
  }
  ```

is a dangling pointer because what is passing here is the following iptr will be a variable that will have a pointer variable of type int so when we will write the int pointer variable to iptr it will work but as soon as the functions finished the automatic allocation of a will be also automatically deallocated, letting *iptr pointing to an invalid address

- Figure 2-2 is a good photo and here is why The image show 3 boxes and is useful to understand the following iptr box has as value the address of jptr that will have as value an address in this case the heap of an int value. what the image let you understand if when calling g(&jptr) g function is locally creating a variable that will get the reference of jptr, that' the reason why iptr points to jptr. Then at assigning *iptr to (int)* malloc(sizeof(int)) what is happening is we are taking the local pointer variable that points to the "real" pointer variable that is pointing to the actual memory space. and as we do *iptr we are writting that gotten memory address returned by malloc and assigning it to jptr. So when the function finish iptr as was automatically generated by the function dissapear and the jptr get the address of the memory block he ca use.
  - Not clear how they assure them selves concerning to memory management issues. I will try to understand that when looking the implementaiton of the dsa done by themselves.
  - Paragraph (in page 16-17) is clear basically what is explaining is the following in C the array is sometimes seen as a pointer and 'a[i] == *(a+i)' here is something important in the right part of the equal (*(a+i)) what is happening is that for accessing the correct value, pointer arithmetic takes place so i wil increment the a(address) but not only itimes but it will increment it i*sizeof(a[0]) times a[0] will be of anytype, the important here is that (a+i) increments considering the type of the pointer if is an int it will be 4bytes so it will increment i*4.
  - Paragraph (in page 18) is important I understand, C only accepts passing by value, the reference in reality is a variable created locally in the scope of the called function that has as value the address of the variable passed as argument.
  - So the part of why we need to all but the first dimension is because the compiler, at the moment of getting an element from the array, he needs first to know the address of the memory itself and then in order to retrieve the correct element it needs to know the size of each element to do retrieve the corrects amount of bytes. Here an example

  if we have an array int T[3][2] this is transalted in row-major order so in reality is T[6] == T[3][2] but as the notion is we have instead of one element of type int we have two element of type int as one then the compiler for taking in consideration this needs to know how many bytes it needs to consider so when passing a an array as a function it needs the sizes of the inners arrays
  - Pointers to pointers as parameters I remember this was crucial when I did my implementations for different things
  - when I tried to move the head of the linked_list or trees that was needed because if not there was not possible to modify the actual head because as the reference passed is coppied by value the first element at dereferencing will be pointing to the next one and u need what is pointing to the first address itself.
  -
  ```

- This is good in the page 24 Basically is an idea that makes sense is the following I remember before sometimes I used to ask myself why did I need to do reallocs for char* and one of the reasons is because when I create a pointer the memory allocated is for the pointer itself not for the memory block it points. the memory block it points needs to be also allocated. So basically it works if the stack has done a previous allocation or if the user does it manually through malloc or realloc
- This of tail recursion is really clear what i get of this is the following, basically when u do a normal recursive function, if its depends to the following call, the compiler will understand that it can not remove the caller stack activation, so what woudl happend eventually is having a lot of stack frames until the unwiding face arrives, but instead if we make the compiler aware that the caller doesn't depend to the following code, (a modern compiler) will reuse the stack activation with the other call instead of creating a new stack frame. why this work because one the caller call the caller itself finished what it has to do, it doesn't need anything from the function it called.
- From the recursion chapter I need to understand something, I need to understand the mathematical notation such as `def T(n): if n==1: return 1 elif n>1: return 2*T(n/2)+n`

Analysis of algorithms

```
- Different way to measure performance, but also keep in mind which kind of performance
we are measuring, normally the most important in this kind (cs) is time, nevertheless the
space is less often demanded.
Whatever the case determining how and algorithms performs requires a deterministic and formal metho

- So far this chapter focus in 3 topics

- Worst-case analysis
- O-notation
- Computacional complexity

- Worst case analysis is the most used because
- The best case is normally the best case of the others
and is rarely that this happens.

- The average case is hard to get and honestly it depends the data,
so no really convenient.

- The worst case is more representative because it tells you exactly what
what would be the worst performance the algorithms could take and
you can not go further than that.

- O-notation

- There are some things to keep in mind at the moment
of getting the O(x). Basically takes the greater
f(x) = (x^0)? + (x^1)? + x(x^.....)?
in this expression takes the x with greater grade
also don't keep in mind the constants that are factor
of this x's because when x is to big this constants factor
doesn't represent a lot.
f(x) = (x^0)? * (x^1)? * x(x^.....)?
in this case that is multiplication
```

is different is not changed the so getting the O(x) will look similar but the expression normally get rewrite

```
- Computacional complexity
- basically is the growth rate of (time|space) depending of data.
```

- T(n) <= cn, where c is a constant factor that is not generated for the input(data) usually comes from (codes generated from the compiler, type of computer where the algo is running, and constants in the algot itself.

Data structure part

- This is the part I wanted to see, but all the previous information was good to have it, so my idea is read his implementation and trying to understand why in which things it differs with mine and do the implementations by myself.

Nand2tetris

Notes

Git

Notes

git diff

- git diff –> working tree and staging area
- git diff –staged –> staging area and repository
- git diff HEAD –> working tree and repository

basic-commit

- git works with 3 diff areas
  - working directory (where u makes your changes)
  - staging area (where all changes added through git add will stay)
  - the repository (where every commit ends up, staged changes are put in here by git commit)

basic-staging

- git works with 3 diff areas
  - working directory (where u makes your changes)
  - staging area (where all changes added through git add will stay)
  - the repository (where every commit ends up, staged changes are put in here by git commit)
- while doing the kata
- I realize some things
  - when you do git diff it will compare the working directory with the staged area if poblated if not it will compare with index(HEAD)
  - when you do git diff –cached|–staged it will compare the working directory with staged area
  - when you do git diff HEAD it will compare what is in the working directory against what is in the (HEAD) commit. It will not consider stagging changes
  - When you do git diff –cached HEAD it will compare what is in the staging area with he (HEAD) commit

basic-branching

- Something I realized is that git doesn't compare the whole working directory with HEAD if `git diff HEAD` it only compares the working directory files once staged or added previously
- Some useful things I saw -> I knew but worth it writting it again for comparing two branches branch_a branch_b then you can do `git diff branch_a branch_b` this will output the added changes (+) from branch_b taking in consideration branch_a (-)

Rebase branch

- Looking at the graph generated I can reach the following conclussion, when you do git rebase 'x' what happens is the following a temporay branch is generated that will be the result of checking out to 'x' and then the commits in the current brach are going to be picked and then the current branch will be the temporary branch

```
85ebb17 (HEAD -> uppercase) HEAD@{0}: rebase (finish): returning to refs/heads/uppercase
85ebb17 (HEAD -> uppercase) HEAD@{1}: rebase (pick): Change greeting to uppercase
777f34d (master) HEAD@{2}: rebase (start): checkout master
33f6feb HEAD@{3}: checkout: moving from master to uppercase
```

Basic revert

- revert command –> git revert <> this creates a new commit in wich it explains what changes were reverted, you can still modified the message but that is a way to clean some mistaken commit changes, and then make it clear.

- git revert may revert some commits and remove those chagnges or files creations from working directory but is still there in the git repository so if you go back to the reverted commit you will find the things there

git reset

- Cool so the first thing I could learn from this command is the following if you use the following command `git reset --soft HEAD~1` this will remove the commit but it will stay in the staging area
- Now I use this command `git reset --mixed HEAD~1` if I am not wrong is doing both removing commit and from staging area both
- Something interesting so basically in this kata I needed to do multiple resets with diff flavors (soft,mixed, hard) the first put the changes in the staging are and goes to the set commit, if is a previous one will be like only doing git restore –cached *files_modifieds when mixed this not only remoe the commit but also the staging are and will be only in the working directory hard remove everything and leave the working directory as the repository commit, something I saw was I don't lose any file with soft and mixed because they remain in the working directory.

git clean

- this allows you to clean the work tree (working directory) it removes untracked files `git clean -n` tells you what it can delete `git clean -f -d` remove untrcked files -f (–forced) is needed

amend

- git ammend, does what I knew if modifies last commit letting you chnage the message and other things

advance rebase

- Here is what I am thinking to do
  - doc's commits squashed -> and editing the commit message [x]
  - squash commits modifying hello.test [x]
  - Removing secret commit from history [x]
  - Edit commit message first commit after tag [x]

```
init status
6c7e04d I forgot a semicolon
22aac73 Test for feature hello world
1e6df11 Add doc - step 3
736a103 Add doc - step 2
c3efd54 Add doc - step 1
b4deca5 important secret
fe9bbf9 debugging
b1c16e0 Really made the thingy done
c7d2d4a Finished HW feature
a36a31f Helo Volrd feature
d5e9b93 Initial commit
```

```
git reflog
fe8c164 HEAD@{0}: rebase (finish): returning to refs/heads/master
fe8c164 HEAD@{1}: rebase (pick): Add doc
7cf4636 HEAD@{2}: rebase (pick): Test for feature hello world
986a018 HEAD@{3}: rebase (start): checkout refs/remotes/origin/master
c8779ab HEAD@{4}: rebase (finish): returning to refs/heads/master
c8779ab HEAD@{5}: rebase (pick): Test for feature hello world
51eb958 HEAD@{6}: rebase (pick): Add doc
986a018 HEAD@{7}: rebase (start): checkout refs/remotes/origin/master
c67882e HEAD@{8}: rebase (finish): returning to refs/heads/master
c67882e HEAD@{9}: rebase (pick): Test for feature hello world
812cacc HEAD@{10}: rebase (pick): Add doc
16575d3 HEAD@{11}: rebase (pick): debugging
```

```
986a018 HEAD@{12}: rebase (squash): Hello world feature
3daff85 HEAD@{13}: rebase (squash): # This is a combination of 2 commits.
332b16c HEAD@{14}: rebase (start): checkout refs/remotes/origin/master
b4a423f HEAD@{15}: rebase (finish): returning to refs/heads/master
b4a423f HEAD@{16}: rebase (pick): Test for feature hello world
a37c695 HEAD@{17}: rebase (pick): Add doc
c2447fd HEAD@{18}: rebase (pick): debugging
49b7f47 HEAD@{19}: rebase (pick): Really made the thingy done
9e3ae74 HEAD@{20}: rebase (continue): Finished HW feature
414d2e7 HEAD@{21}: rebase: fast-forward
332b16c HEAD@{22}: rebase (start): checkout refs/remotes/origin/master
f9e6476 HEAD@{23}: rebase (finish): returning to refs/heads/master
f9e6476 HEAD@{24}: rebase (pick): Test for feature hello world
edae9f7 HEAD@{25}: rebase (pick): Add doc
47cc327 HEAD@{26}: rebase (pick): debugging
0945b26 HEAD@{27}: rebase (pick): Really made the thingy done
414d2e7 HEAD@{28}: rebase (pick): Finished HW feature
332b16c HEAD@{29}: rebase (reword): Hello world feature
a36a31f HEAD@{30}: rebase: fast-forward
d5e9b93 HEAD@{31}: rebase (start): checkout v0.0
20177b9 HEAD@{32}: rebase (finish): returning to refs/heads/master
20177b9 HEAD@{33}: rebase (pick): Test for feature hello world
4f8cde1 HEAD@{34}: rebase (pick): Add doc
fe9bbf9 HEAD@{35}: rebase (start): checkout v0.0
6b61c82 HEAD@{36}: rebase (finish): returning to refs/heads/master
6b61c82 HEAD@{37}: rebase (squash): Test for feature hello world
e7410fd HEAD@{38}: rebase (start): checkout refs/remotes/origin/master
e6f7575 HEAD@{39}: rebase (finish): returning to refs/heads/master
e6f7575 HEAD@{40}: rebase (pick): I forgot a semicolon
e7410fd HEAD@{41}: rebase (pick): Test for feature hello world
9b9c466 HEAD@{42}: rebase (squash): Add doc
61b70fe HEAD@{43}: rebase (squash): # This is a combination of 2 commits.
c3efd54 HEAD@{44}: rebase (start): checkout v0.0
6c7e04d HEAD@{45}: commit: I forgot a semicolon
22aac73 HEAD@{46}: commit: Test for feature hello world
1e6df11 HEAD@{47}: commit: Add doc - step 3
736a103 HEAD@{48}: commit: Add doc - step 2
c3efd54 HEAD@{49}: commit: Add doc - step 1
b4deca5 HEAD@{50}: commit: important secret
fe9bbf9 HEAD@{51}: commit: debugging
b1c16e0 HEAD@{52}: commit: Really made the thingy done
c7d2d4a HEAD@{53}: commit: Finished HW feature
a36a31f HEAD@{54}: commit: Helo Volrd feature
d5e9b93 HEAD@{55}: commit (initial): Initial commit


result

* fe8c164 Add doc
* 7cf4636 Test for feature hello world
* 986a018 Hello world feature
* d5e9b93 Initial commit
```

git stash

- When using git-stash it puts the changes in a stack
- Few things to keep in mind
  - `git stash` command puts all those changes in a stack but it puts both changes in the working directory and changes stagged so once you try to do stash pop it will get them back but the files that where in the stagged area are removed from there
  - `git stash --index` can be kind of better this will literally left us as we where putting the stagging

files if they exist in its location

git rm

- Interesting if I want to remove something from the stagging area but keep it in my working directory the command that works is `git rm --cached`
- this is key to know .gitignore only ignores files that are not part of the repository yet. If you have already added (git add + git commit) their changes will be still tracked to remove them from your repo do (git rm –cached)
- **still doubt because –cached remove them from the index (stagging area) but not from the repo itself?**

git submodules

- I have seem some new things in this kata so basically I have taken a git repository and uses it as submodule of another git repository for doing that == in this case as we are simulating the host repository local, I needed to add `git -c protocol.file.allow=always submodule add ../remote include` if I execute that command it will simulate a clone of the component repository hosted in this case local in the remote folder So once that is done if pull from inside include folder, it will brings inside include all the latest changes and merge with the current version, once that is done if i go outside and do git status you will see that there are unstagged changes and it explains if modified or things, I even can modify the submodule making a global impact
- git repo wrapping the submodules, keep track of the commit ids
- need to check this part yet

Things to check

- git-filter-repo for rewriting Git history -Elijah Newren | GitMerge 2024

Technical writting

Notes

- Steps understand how pandoc works
- I will use this as the mechanism for technical writting

Ideas

- I think is a good idea using pandoc as a docker container for not having installation problems each time I switch computer
- Steps
- Read about pandoc in general
- Follow a technical writting guide
- Check makefiles for doing it recursively and just better [Putt what I learnt if makefiles in it's knowledge space]

Why

- I want to create a nice notebook in which I can showcase my knowledge
- This will allow me to review it, be proud of my learnings.
- We need to makes things better, things that looks beautiful and is also a learning process

gns3

Notes

Virtualisation

Important link

- redhat linux networking_1
- redhat linux networking_2
- software networking and interfaces
- crack doing networking

What is

Software that simulate hardware

Notes

Qemu popular vm in linux

Usage

Creating disk

- For creating disk
-

```
qemu-img create -f qcow2 HDA.img 10G
```

- -f –> stands for format
- qcow2 –> Is a format for file disk that qemu understand
- HDA.img –> image name
- 10G –> image capacity

Creating vm

```
qemu-system-x86_64 -cdrom ~/Downloads/isos/archlinux-2024.12.01-x86_64.iso -boot d -cpu host -enable-kv
```

This is telling qemu to create a virtual machine with the following hardware capacities

- -cdrom –> will have embedded the image
- -boot –> will have the way of booting in this case is a disk instead of the hardisk
- -cpu –> tells if taking or not host hardware capabilities is setted as host host architecture must be the same that the vm's
- -enable-kvm –> This allow to have accelerator so quicker performance
- -smp –> Assign number of cpus assigned to the vm
- -m –> Assign RAM capacity of the vm

Vm networking (Libvirt/Bridge)

Comparison at the network interfaces when activating default network by libvirt through virsh

```
diff ./beforeNetwork ./afterNetwork

16a17,24
> virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
>         inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
>         ether 52:54:00:86:b5:2b  txqueuelen 1000  (Ethernet)
>         RX packets 0  bytes 0 (0.0 B)
>         RX errors 0  dropped 0  overruns 0  frame 0
>         TX packets 0  bytes 0 (0.0 B)
>         TX errors 0  dropped 2 overruns 0  carrier 0  collisions 0
>
21c29
<         RX packets 169011  bytes 143424229 (136.7 MiB)
---
>         RX packets 169211  bytes 143443705 (136.7 MiB)
```

```
23c31
<         TX packets 52840  bytes 19056560 (18.1 MiB)
---
>         TX packets 52903  bytes 19064229 (18.1 MiB)
```

- First using libvirt through virsh for starting "default network" vm attach by this default network by default
- I am not getting that virbr0-nic (bridge between physical network card and virtual machine virtual network card)
- What is happenign is the following
  - that new interface created virbr0 acts like a virtual switch (switch connets multiple nodes)
  - Then when I start an vm another interface got created (tap interface)
  - vm is tight to vnet0
- dnsmasq –> like DHCP server

Libvirt default network

- by default

Bridge network

-

Attaching vms to the bridge

- tun0
- NAT
- ip forwarding
- tap0

Understood and doubts According of what I have understood, libvirt is creating a linux bridge (virtual switch) and each time I assign a vm. As it use NAT, the vm should not be visible to the outside world let's try it out.

- Ok I understood the following as libvirt created the virtual switch (linux bridge) my machine is able to comunicate with the virtual machines through networking because my machine is by default attached to that bridge I think so when the ping command see the ip it will switch automatically to the correct interfaces
- create a way to enable an outsider the vms
- DHCP sever is the one that have an iptable and assign ips dynamically
- DNS translates domain names into IP addresses

- https://www.youtube.com/watch?v=VCAqkyVd7dM&t=244s In this video the guy is doing the following for giving internet connection to the vm, so basically he is creating linux virtual switch by creating a linux bridge and to that bridge he is plugging the ethernet () to the bridge and for pluggin the vm he needs to create a kind of connector for that he uses a TAP interface that enables to be plugged to a switch networking diagram
- it was what I was thinking

Give networking to vm without libvirt

As the vm is given a private ip assigned by qemu this ip doesn't have a gateway that provide it with actual internet, so what we will do is change this settup

Steps

- Create a linux bridge (virtual switch)
- Once the virtual switch is created, we have to plug in it the different machines.
- Also as now we are redirecting the eth0 that is the interface connected to the NIC and is where we are really reciving the internet packets, we need to remove eth1 the ip-address done by our dhcp server and define an static ip address to the bridge, in such a way the bridge will be the one exposed IMPORTANT! this will messed up with the dns (This part I need to investigate a little bit more)
- For connecting a vm to the linux bridge, a TAP interface needs to be created this will be the connector between the virtual interface of the vm
  and the virtual switch (bridge) residing in the host.
- now the settup will be that our bridge will have attached the actual interface that is handling the actual NIC

  and also the vm/containers via tap that is a way to create a link between virtual machines and virtual switches
- Finally we will have a scheme like the following scheme

Makefiles

Notes

- using regex so (? means 0|1 ocurrence)
- Notes about makefile

Link

- https://www.youtube.com/watch?v=Y-m8hGZUX4E&list=PLSeOJEFuO7Y5Ven6i1-4XbrQw50TiT3QK

ch__1

- makefiles are composed of rules each rule follows the format

```
target(normally a? files?):prerequisites?
    recipe(shell commands?)
```

- Makefiles check the timestamp of modification and also is the files? exist or not for example in this makefile I will write the followings scenarios they will occurs

```
greeting.txt: hello.txt world.txt
    cat hello.txt world.txt > greeting.txt

hello.txt:
    cat hello.txt

world.txt:
    cat world.txt
```

- in this Makefile the first time I executed, the prerequisites will be checked as they don't exist it will go to the defined targets, as they don't have prerequisites and they don't exist they execute their recipe, and the recipe is a shell command that creates a file. once that is done then as greeting doesn't exist it will create the greeting.txt
- If i try to rerun make it won't do anything because nothings has to be modified and all the targets(files?) are already there
- If i delete a file it will go to the specific target and re-run the recipe
- If i modified a file it will modify the target that is requesting those prerequisites
- I have a question of the makefile I tweaked from Job Vranish (https://spin.atomicobject.com/2016/08/26/makefile-c-projects/) // inspired from How does it know how to change if the .h file is not listed
- Ok interesting but that question I need to answer it
- Ok, cool if I want to create a target in which I want to run the recipe in the same line, I have to add a semicolon

```
<target>:  ; <shell command>
```

- If I want to create a target which then I will redeclare, so there will be a chance of overwritting then Makefile by default overrides it but if instead we use double semicolon, it will run all instances

```
double_colon:: ; @echo "I was first declared"

double_colon:: ; @echo "I was second declared"
```

- For only caring if file exist and not timestamp if modified then we can use a pipe before prerequisite

```
new_file: | file.txt
        @echo "Not created yet"

file.txt:
        @echo "Creting file"
        touch file.txt
```

- Prefixes

  - '@' –> doesn't show the command
  - '-' –> ignores erros
  - '+' –> run always even with –dry-run command

- Pattern rules
  - exact matching has bigger importance than pattern matching
  - it can be used with '%' so basically the '%' acts like a joker

```
# curious things notice that the recipe even if i a shell command
# it doesn't consider $@ as a defined bash variable but instead it takes makefile
# default varible.
patter%matching:; @echo "it tooks $@"
%matching: ; @echo "it tookts $@"
# direct match takes precedence from pattern matching rules
hello-%:; echo "it tooks  $@"
hello-world:; echo "it tooks hello-world"
```

  - In this example any string like /(patter).*(matching)/ will match the first rule
  - In this example any string like /.*(matching)/ will match the second rule
  - if there is pattern matching matches but also a direct match the direct match takes relevance over the patter

- prerequisites handling
  - Really useful in work
  - Before I have the case in which for demostrating the 'double colon functionalitie' I create to rules with the same target name, but before putting the double colon I put only one and then I found out that gthe first rules got overriden by the following definition, now I know that is the case for the recipe it means the "code| shell part" but that's not the case for the prerequisites so the prerequisites gets appended to the list of prerequisites of the target that is overridign

```
multiple_prerequisites: dep2

multiple_prerequisites: dep1
    @echo "Building multiples -> prerequisistes"

# multiple_prerequisites: dep1 dep2
#   echo "Building multiples -> prerequisistes"
# so Dependences: dep1  -> Dependences: dep2 ->  "Building multiples -> prerequisistes"
dep%:; @echo "Dependencie: $@"
```

- static patter rules
  - Useful when you want to automatically generate prerequiisites based on the names targets
  - so what happens here is that the joker ('%') will replace get the value that we want to match and then it will executes the rules matching that pattern

- PHONY: prerequisites
  - This allows to have rules that whose targets doesn't have prerequisite
    - all, clean, test are common targets used