

COMEÇANDO-DEVOPS

Todo material de estudo desse documento foi obtido a partir do curso de formação da Alura “começando em devops”.

Curso de formação em Começando em DevOps | Alura

Passo a passo O que é DevOps? É cultura? É uma carreira? Comece entendendo um pouquinho mais sobre. Depois para entender o que acontece dentro do seu computador, vamos te ensinar sobre arquitetura de computadores.

 <https://www.alura.com.br/formacao-primeiros-passos-devops>



▼ Entendendo DevOps

▼ O que é devops

Login | Alura - Cursos online de tecnologia

Faça seu login e boa aula! Esqueceu sua senha? Digite seu e-mail que enviaremos um link para definir uma nova senha. São mais de mil cursos nas seguintes áreas:

<https://cursos.alura.com.br/extra/alura-mais/o-que-e-devops--c82>

- Dev - cria da aplicação, atualização, resolução de problemas, faz mudanças, o ambiente precisa ser melhorado
 - Muda o tempo todo
- Ops - infra, monitoramento, saúde da operação, não curte muito mudanças, o ambiente precisa estar funcional
 - Presa estabilidade
- DevOps - Junta os dois lados (operações+desenvolvimento), é um movimento cultural empresarial → Busca melhorar colaboração, qualidade, diminuir o tempo de entregas, e fazer entregas mais seguras.
 - Tools Devops

- Virtualização e containerização
- IaC
- CI & CD
- Ferramentas de monitoramento e analise
- <https://gomex.me/blog/carreira-devops/>
- <https://aws.amazon.com/pt/devops/what-is-devops/>
- <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops/>

https://www.youtube.com/watch?v=lQ8-_khQATQ

- Um Profissional em DevOps é responsável pela codificação e pela infraestrutura de uma organização.
- A cultura devops também se amplia a outros membros da organização
 - Dev → Entender bem do que se trabalha, mas entender da infraestrutura em que se está responsável
 - Ops → Entender o que os códigos dos desenvolvedores influenciam na infraestrutura
- Os profissionais Devops irão realizar a integração entre as áreas, fazendo com que elas se comuniquem de maneira correta e eficaz.
- IaC → Área em devops voltada para codificação de um infraestrutura, ex: O código irá criar n vms, n redes, etc.
 - Antes utilizava-se arquivos sh em bash para automatização da infraestrutura
 - Idenpotencia → Roda o arquivo qnts vezes quiser, e tem o mesmo resultado, ex: Um arquivo sh vai criar x máquinas quando rodar, e caso rode o mesmo arquivo vai criar novamente x máquinas
 - Terraform → Ferramenta mais atual de IaC para gerenciamento e implantação de infraestrutura.

▼ Arquitetura de computadores: por trás de como seu programa funciona

▼ Como o computador lê o seu código?

▼ O Código de Maquina

- Linguagem de alto nível → Entendível pelos humanos (ex: java, python, c#, etc.), está é uma linguagem na qual o computador não entende.

```
let a = 7;  
let b = 2;  
let c = a + b;  
if (c !== 0) {  
    c++;  
}
```

JavaScript

- Linguagem de baixo nível → Ou linguagem de máquina na qual o computador soma, guarda, compara.

```
guarda 7 em a  
guarda 2 em b  
pega a  
soma b  
guarda em c  
compara c e 0  
se_igual_pula 1 linha  
incrementa c
```



- Na realidade o computador se comunica por 0 e 1 → Código Binário
 - Se o fio tá passando energia = 1 senão = 0

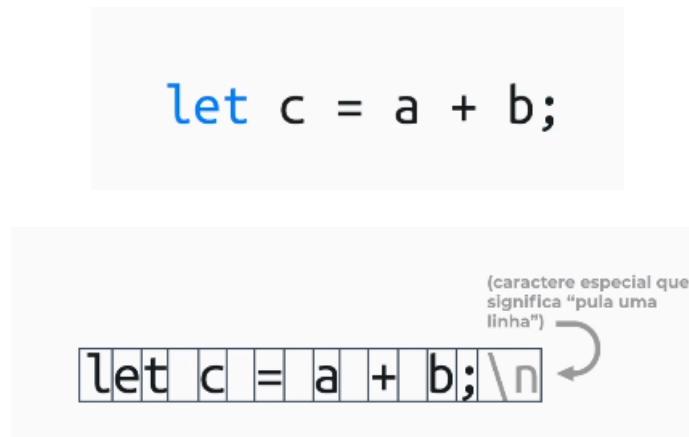
▼ bits e Bytes

- 0 e 1 é bit -> menor unidade possível para o armazenamento de informação
8 bits = 1 byte -> Medição de armazenamento de um computador
b -> bit & B -> A byte
1B = 8b
- k -> mil, quilo
M -> milhão, mega
G -> bilhão, giga
e assim por diante

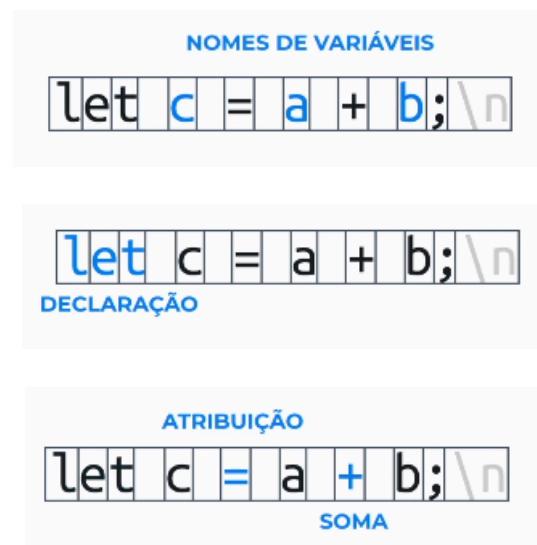
- 200MB é igual a 200 milhões de bytes
- 20Mbps é 20 milhões de bits por segundo ou 2,5 milhões de bytes por segundo

▼ Tradutor do computador

- O tradutor interpreta o texto puro, ou seja a sequencia de caracteres



- Após isso ele valida os conjuntos e as relações dos caracteres
- validas as variaveis, em seguida a declaração, depois as atribuições, e ação a ser realizada (ex: soma)



- Nesse processo ele identifica erros, e verifica os significados, após esses processos e segue com a tradução para a linguagem de maquina, em seguida ele cria o codigo de maquina, que o computador executa.

pega a
soma b
guarda c

11011001 10110011
11001101 1011000
11000001 1001001

CÓDIGO DE MÁQUINA

110110011011001111001
1011011000110000011
001001

- O processo de tradução de pilhas de códigos ocorre da seguinte forma:

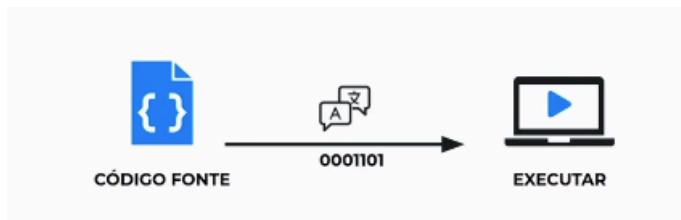


- A tradução realizada dessa forma, no caso de um arquivo só, é através de um compilador, que irá pegar todo o código fonte e traduzir para o código máquina.



- Traduções de trechos por trechos no código, comando por comando, ou seja uma tradução em tempo real é realizada por um Interpretador que traduz um comando por

vez



▼ Executando diferentes linguagens

- Neste caso vamos demonstrar a diferença de um tradutor interpretado pra um compilado com duas linguagens diferentes, programamos dois algoritmos em duas linguagens diferentes, C e python
 - Compilado - C

```
c programac.c
1 //COMPILADO
2
3 #include <stdio.h>
4
5 int main() {
6     int a = 7;
7     int b = 2;
8     int c = a + b;
9
10    if(c!=0){
11        c++;
12    }
13
14    // for (int i = 0; i < 10000000; i++) {
15    //     c++;
16    // }
17
18    printf("Olá mundo, c = %d\n", c);
19    return 0;
20 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
lsantos@pop-os ~/Documents/compsinterp gcc programac.c -o programac
lsantos@pop-os ~/Documents/compsinterp ./programac
Olá mundo, c = 10
lsantos@pop-os ~/Documents/compsinterp
```

Compila o arquivo
Exécuta o arquivo
Resultado

- Programas compilados normalmente é necessário executar sempre um passo a mais que é o passo da compilação, antes de executar o programa em si
- Ou seja para cada modificação no código você precisa compilar todo o arquivo

```

C programac
1 //COMPILADO
2
3 #include <stdio.h>
4
5 int main() {
6     int a = 7;
7     int b = 2;
8     int c = a + b + 10;
9
10    if(c!=0){
11        c++;
12    }
13
14    // for (int i = 0; i < 10000000; i++) {
15    //     c++;
16    // }
17
18    printf("Olá mundo, c = %d\n", c*2 + [1]);
19    return 0;
20}
21
22

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

- lsantos@pop-os ~/Documents/compsinterp gcc programac.c -o programac

Olá mundo, c = 20
- lsantos@pop-os ~/Documents/compsinterp gcc programac.c -o programac

Olá mundo, c = 40
- lsantos@pop-os ~/Documents/compsinterp gcc programac.c -o programac

Olá mundo, c = 41
- lsantos@pop-os ~/Documents/compsinterp

- A princípio a tradução interpretada parece ser mais eficiente porém quando se trata de arquivos com códigos grandes, a compilação tende a ser melhor, ou seja a velocidade de execução.

```

C programac
1 //COMPILADO
2
3 #include <stdio.h>
4
5 int main() {
6     int a = 7;
7     int b = 2;
8     int c = a + b + 10;
9
10    if(c!=0){
11        c++;
12    }
13
14    for (int i = 0; i < 10000000; i++) {
15        c++;
16    }
17
18    printf("Olá mundo, c = %d\n", c*2 + 1);
19    return 0;
20}
21
22

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

- lsantos@pop-os ~/Documents/compsinterp gcc programac.c -o programac

Olá mundo, c = 20000041
- lsantos@pop-os ~/Documents/compsinterp time ./programac

./programac 0.02s user 0.00s system 98% cpu 0.022 total
- lsantos@pop-os ~/Documents/compsinterp

Add esse for que irá adicionar 10.000.000 na variável C.

Compilando e executando o programa, podemos obter o resultado de forma quase instantânea e isso é um diferencial se comparado com interpretadores. Add o "time" antes da execução do programa, podemos ver em quanto tempo a execução ocorreu no caso 0.02s

- Interpretado - Python

```

programap.py > ...
1 # INTERPRETADO
2
3 a = 7
4 b = 2
5 c = a + b
6
7 if c != 0:
8 | c+=1
9
10 # for i in range(10000000):
11 # c += 1
12
13 print('Olá mundo, c =', c)
14

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

- lsantos@pop-os ~/Documents/compvsinterp python3 programap.py Olá mundo, c = 10 Já interpreta e executa o programa mostrando o resultado
- lsantos@pop-os ~/Documents/compvsinterp

- Programas interpretados já temos um acionamento de primeira quando executado

```

programap.py > ...
1 # INTERPRETADO
2
3 a = 7
4 b = 2
5 c = a + b + 10
6
7 if c != 0:
8 | c+=1
9
10 # for i in range(10000000):
11 # c += 1
12
13 print('Olá mundo, c =', c*2 + 1)
14

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

- lsantos@pop-os ~/Documents/compvsinterp python3 programap.py Olá mundo, c = 10
- lsantos@pop-os ~/Documents/compvsinterp python3 programap.py Olá mundo, c = 20
- lsantos@pop-os ~/Documents/compvsinterp python3 programap.py Olá mundo, c = 40
- lsantos@pop-os ~/Documents/compvsinterp python3 programap.py Olá mundo, c = 41
- lsantos@pop-os ~/Documents/compvsinterp

- A tradução interpretada é eficiente pois demonstra o resultado logo de cara, então caso haja algum erro no código, podemos visualizar de primeira qual é o motivo e qual é o código. Mas em questão de tempo de execução do código, ela pode deixar a desejar.
- Seguindo o mesmo exemplo do código em C, vamos incrementar a variável c 10.000.000x, segue abaixo:

The screenshot shows a terminal window with the following content:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
lsantos@pop-os ~/Documents/compsinterp python3 programap.py
Olá mundo, c = 20000041
lsantos@pop-os ~/Documents/compsinterp time python3 programap.py
Olá mundo, c = 20000041
python3 programap.py 0.61s user 0.00s system 99% cpu 0.612 total
lsantos@pop-os ~/Documents/compsinterp []

```

Annotations in green text and arrows highlight specific parts of the code and output:

- A box highlights the line `for i in range(10000000):` with the annotation: "Adicionamos a mesma incrementação do código em C".
- An arrow points from the terminal output to the line `time python3 programap.py` with the annotation: "Efetuamos a execução e percebemos um delay logo de cara. No caso podemos ver que o delay da execução é de 0.61s, ou seja demorou mais a execução do que em um tradutor compilado."

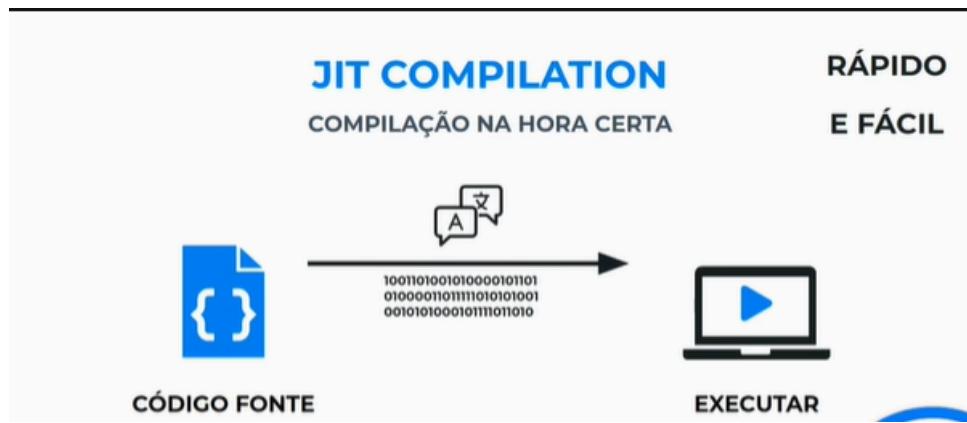
- Programas com mais performance utilizam mais linguagem com tradutores compilados do que interpretados devido à velocidade de execução de cada.

▼ Compiladores vs Interpretadores

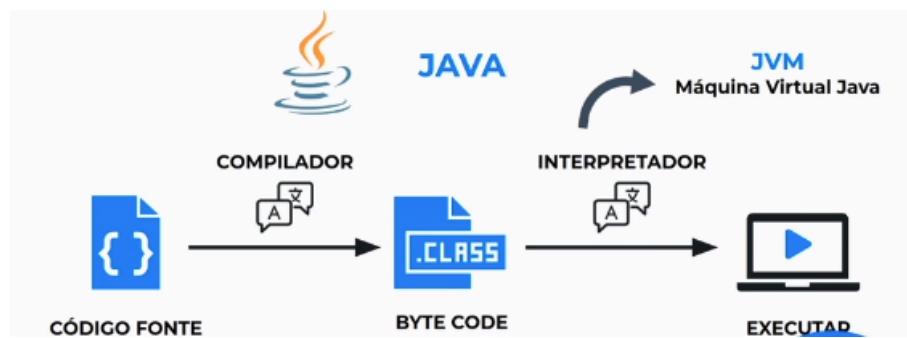
- **Compiladores → Linguagem: C, Rust, Go**
 - Execução Rápida
 - Verifica Erros antes de executar
 - Tempo extra para executar
 - Executa em uma máquina apenas
- **Interpretadores → Linguagem: Js, Python, php, Ruby**
 - Execução lenta
 - Só verifica erros executando
 - Começa a executar na hora
 - Executa em diferentes máquinas
- Uma mesma linguagem pode ter os dois tradutores para implementação
- Pode haver também implementação híbrida

▼ Implementações modernas

- **Jit Compilation → Compilação na hora certa**
 - Ao invés de compilar o código inteiro, copila apenas a função



- V8 Engine utiliza está implementação no chrome para Js
- Java
 - Compilar o código para rodar em vários computadores
 - Compilador gera um byte code, arquivo também conhecido como .class, antes de traduzir o código para binário o arquivo passa por um segundo tradutor no caso um interpretador no caso, o JVM (Máquina virtual java)



- William terminou de programar o código fonte de um jogo e, para executá-lo, precisou passar esse código por um tradutor. Ele observou que após esse processo foi gerado um novo arquivo que pode ser executado diretamente pelo computador. Sobre esse processo de tradução, marque as alternativas corretas:
 - O novo arquivo gerado é o jogo em **código de máquina**
 - Para o programa ser executado diretamente pelo computador, ele precisa ser escrito como código de máquina.
 - Como o código foi traduzido de uma vez só para um executável, o tradutor usado foi um **compilador**
 - Como o próprio nome indica, um compilador junta todo o código traduzido de uma vez. Assim, ele pode ser executado diretamente pelo computador.

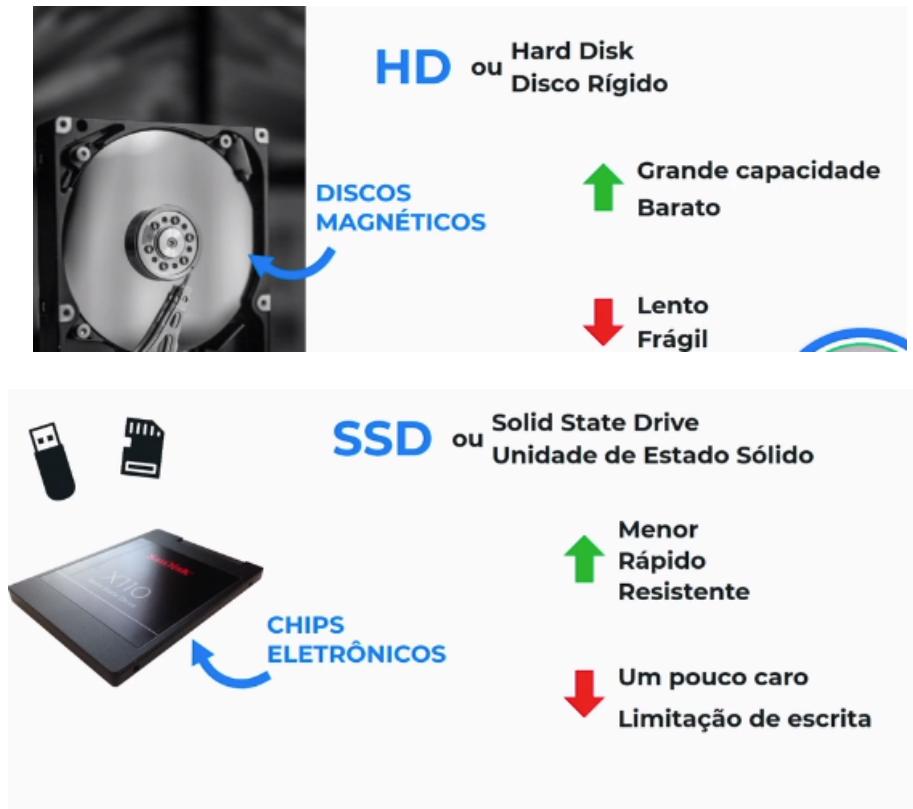
▼ Como o computador executa um programa?

▼ Armazenando o código

- Cada código criado fica armazenado em um sistema de pastas no qual é gerenciado pelo SO



- O código é armazenado em um memória não volátil, ou seja quando reiniciar o computador os arquivos e as pastas ainda vão estar lá, exemplos:

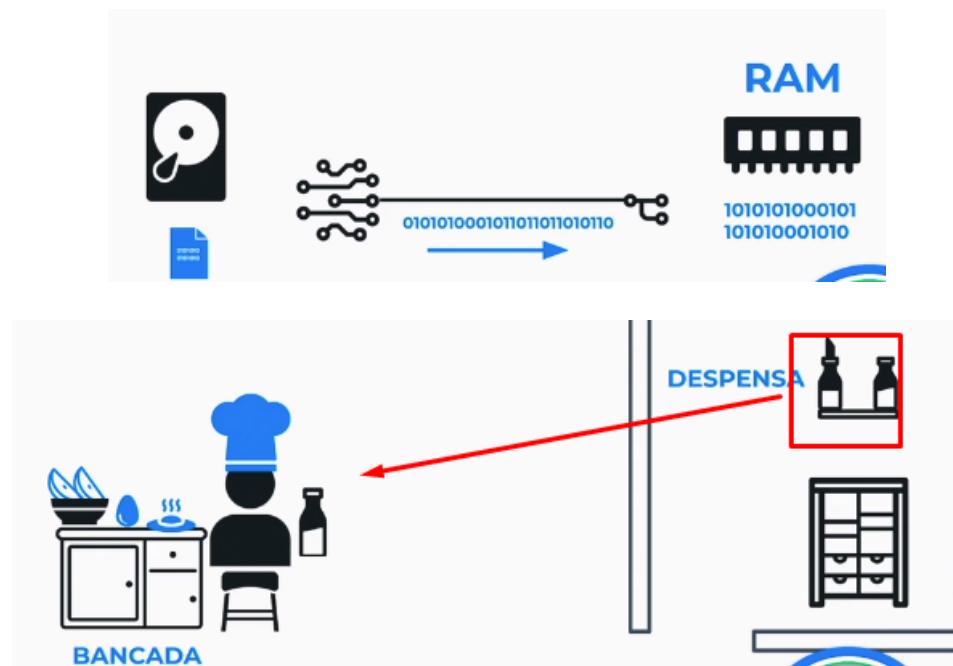


- O armazenamento não volátil, é o onde as informações são guardadas, por isso é chamada de memória secundária, enquanto que a primária é um sistema de armazenamento volátil, no qual onde os programas são armazenados em sua execução, a mesma é denominada memória de trabalho

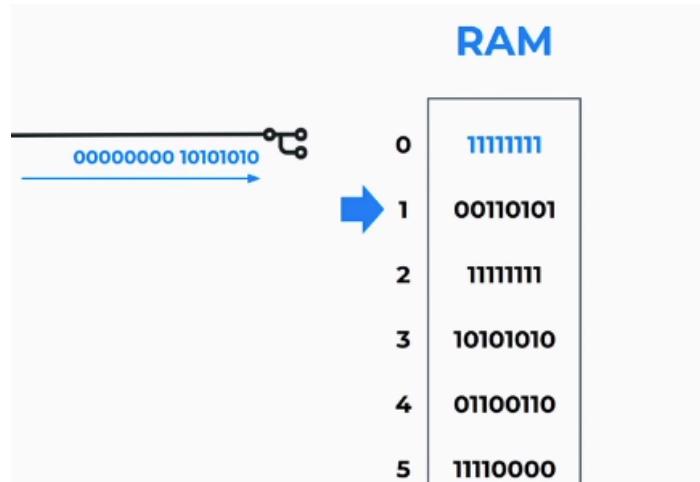


▼ Memória RAM

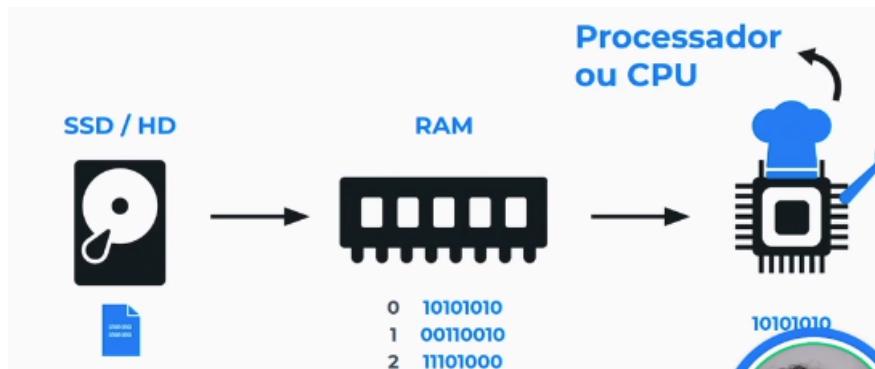
- O armazenamento na memoria ram se assemelha a bancada onde se faz a comida, no caso precisamos pegar a comida na dispensa/armario (memoria secundaria)



- A memoria Ram guarda os programas que estão em execução
- memoria ram é como um indice onde cada armazenamento é um byte
- O programa qnd escreve na memoria ele indica a posição



- Memoria de acesso aleatorio → Porque é se tempo de acesso por exemplo no indice 0 será o mesmo no 1000



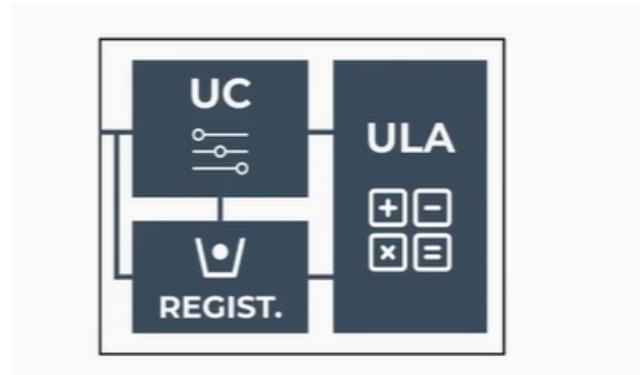
- Processador irá ser o responsável pelo processamento do programa
 - Letícia quer investir em um novo computador. Ela usualmente possui poucos arquivos e programas instalados, mas usa muitos deles todos de uma vez. Pensando nessa situação, quais são conclusões corretas para a escolha do novo computador?
 - Como Letícia usa vários programas ao mesmo tempo, investir em uma memória RAM com mais capacidade é importante.
 - Da mesma forma que uma bancada maior permite fazer várias receitas ao mesmo tempo, uma RAM com grande capacidade permite o uso de vários programas juntos. Com isso, o computador não vai ficar tão lento quando abrir várias abas do navegador.
 - Mesmo com poucos programas, ainda é possível abrir eles mais de uma vez ao mesmo tempo. Por exemplo, podemos abrir várias janelas de um mesmo navegador ou o mesmo editor de texto para documentos diferentes. Dessa forma, o consumo de RAM será bem alto.
 - A velocidade de um SSD, em comparação ao HD, pode acelerar a inicialização do computador e de programas. Como não há a necessidade de uma grande

capacidade de armazenamento, o SSD possui um bom custo-benefício.

- o momento que o computador liga, como ele sabe quais dispositivos estão conectados nele e onde ele precisa ler as informações?
 - Esse tipo de dado inicial é crucial para o funcionamento do computador é localizado na memória ROM, do inglês **Read-Only Memory** ou “memória de apenas leitura”. Ela é um tipo de memória não-volátil com baixo armazenamento, bem diferente da RAM ou HD/SSD. Uma ROM possui vários modelos, mas todos são projetados para serem apenas lidos, ou seja, não é esperado que o usuário escreva informações nessa memória
 - Dessa forma, quando o computador é inicializado, as primeiras informações que ele irá buscar estarão na ROM, por exemplo, a **BIOS** (Basic Input/Output System ou sistema básico de entrada/saída). Elas vão ajudá-lo a identificar os dispositivos conectados, como o HD, e carregar o sistema operacional.

▼ Funcionamento de um processador

- CPU → Unidade central de processamento
 - Pega o que ta na memoria RAM e processa
- Partes da CPU
 - UC → Unidade de controle → Valida a instrução e o que significa, coordena
 - ULA → Unidade Lógica Aritmética → Calculos e validações (Soma, subtrai, guarda)
 - Registradores → Guardam uma informação para calculo ou validação posterior



▼ Executando o Código

- Processo
 - Buscar → Guarda no registrador
 - Decodificar → Unidade de controle
 - Executar → ULA
- Esse processo é coordenado pelo CLOCK

- Velocidade do processo
 - Dá o ritmo
- A velocidade de um Clock se dá por GHz, quanto maior a velocidade, maior será a velocidade de processamento
- As instruções são processadas num ciclo de três passos chamados **buscar, decodificar e executar**
 - Cada instrução passa por esse ciclo e o tempo entre cada uma é sincronizado pelo clock do processador.
 - O clock define a velocidade com que as instruções são executadas. Ou seja, quanto maior a velocidade do clock, mais rápido um programa será executado.
 - A velocidade do clock é um dos principais fatores para medir a velocidade de execução de um certo programa. Muitos entusiastas utilizam a técnica de overclocking, que aumenta o desempenho em programas mas pode causar superaquecimento do processador, um maior gasto energético e uma redução no seu tempo de vida.
 - Antes de serem executadas, as instruções precisam ser decodificadas pela Unidade de Controle (UC).
 - É a UC mesmo que é responsável por decodificar uma instrução lida na RAM. Cada bit decodificado liga um fio ou circuito específico que modifica o estado do processador.

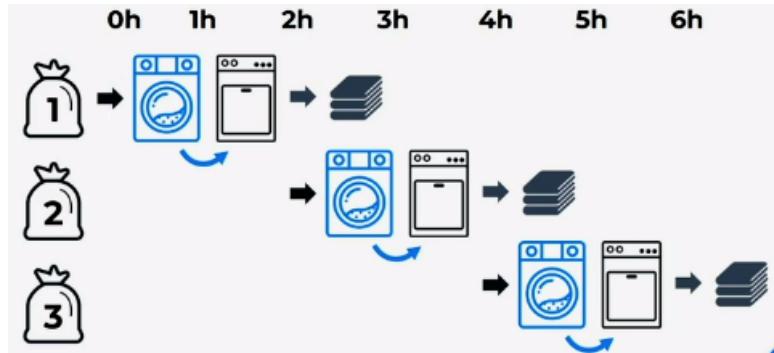
▼ Como o computador executa vários programas?

▼ Processadores Modernos

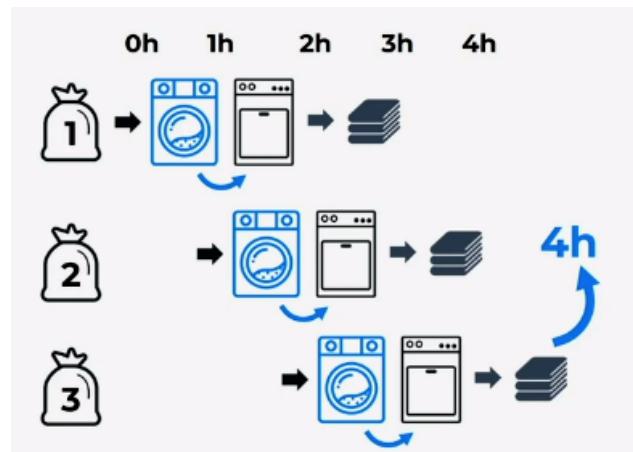
- Modelo padrão de um processador



- A única coisa que diferenciava entre os processadores era a velocidade do clock ou seja quanto maior o ritmo do clock mais rápido o processador era, porém o processador ficava muito ocioso
 - Exemplo:



- Imagine uma maquina de lavar e uma secadora que demora 1h para realizar esse processo para uma muda de roupas, no caso temos 3 mudas de roupas para lavar
- O processo de lavagem das mudas irá demora cerca de 6h para 3 mudas de roupas, ou seja haverá momentos em que a maquina de lavar ficara ociosa e que a secadora ficara ociosa
- Esse exemplo se aplica aos processadores e as tarefas que ficam em execução, o ritmo pode até ser rapido mas ainda assim haverá ociosidade, nesse caso o melhor método a se fazer é conforme o próximo exemplo
 - Exemplo:



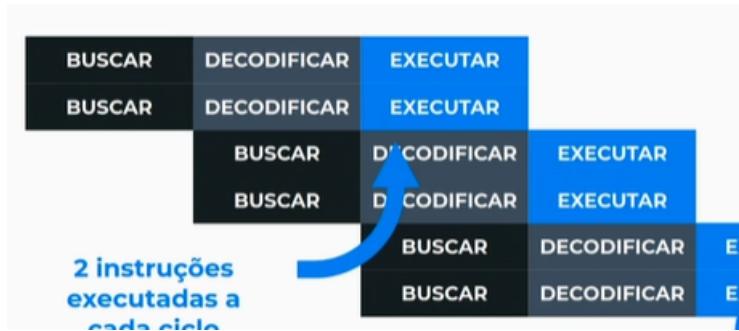
- Nesse processo, enquanto uma muda de roupa está sendo lavada outra está sendo secada, seguindo este fluxo para 3 mudas de roupas, o tempo se reduz quase que pela metade e não há ociosidade
- No caso este exemplo de otimizar o tempo de lavagem de roupa se aplica também para os processadores modernos que não buscam apenas a velocidade do clock, mas também a otimização das tarefas
 - Ao invés de seguir esses passos para um processamento



- Os processadores modernos otimizam o tempo de processamento evitando ociosidade

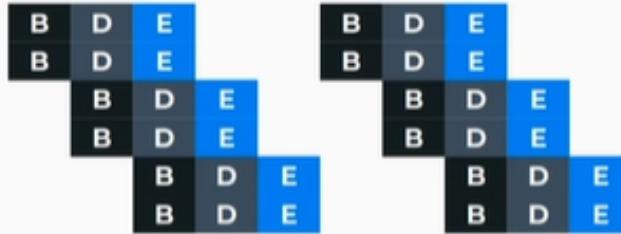


- Esse processo é denominado de 'pipeline de instruções'
- E esse processo pode ser duplicado



- Sendo assim pode aumentar gradativamente as taxas de processamento de um processador

DUAL-CORE



CORE

NÚCLEO

QUAD-CORE



- E por ai vai

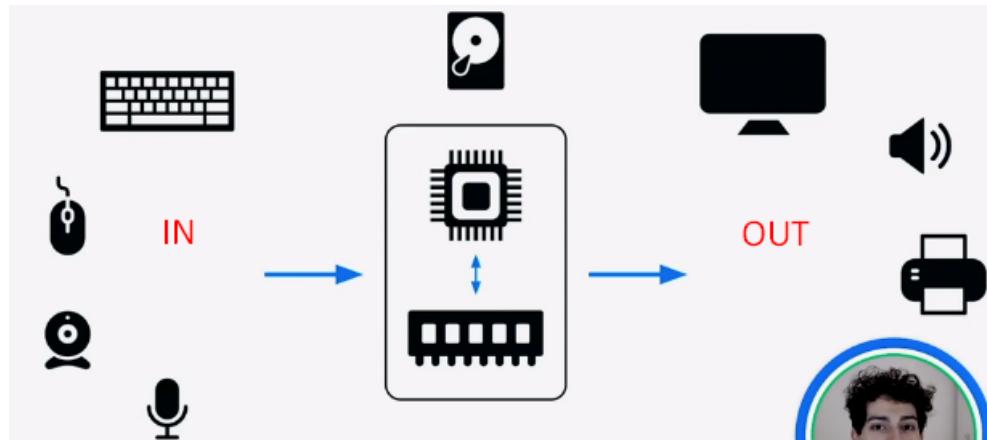
▼ Lei de Moore

- Em 1965, Gordon Moore previu que o número de componentes de um circuito integrado (transistores) dobraria a cada período de 18 meses, resultando num processador duas vezes mais rápido.
- Entretanto, nos últimos anos, os fabricantes de processadores estão encontrando limites físicos para manter esse padrão e o crescimento está cada vez diminuindo mais.
- Por isso, pesquisadores procuram outras soluções para manter o crescimento da performance de um processador, mas sem precisar aumentar o número de transistores ou a velocidade de clock.

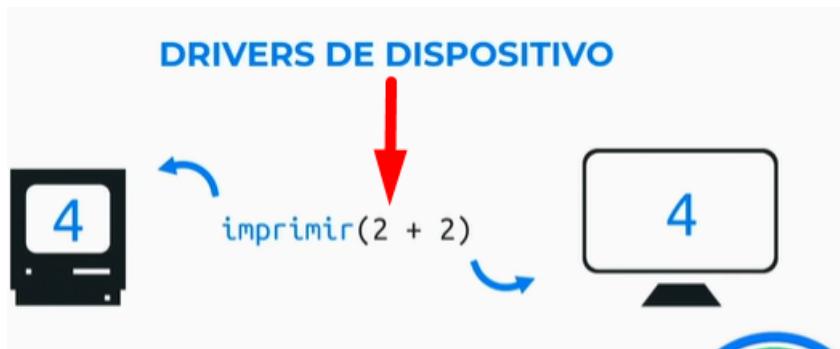
O que é a Lei de Moore e por que você deve se preocupar com o fim dela Jensen Huang, executivo-chefe da fabricante de chips Nvidia, fez um宣言 na CES (Consumer Electronics Show) deste ano, a maior feira de tecnologia do mundo, que aconteceu em janeiro em Las Vegas (EUA):A Lei de  <https://www.uol.com.br/tilt/noticias/redacao/2019/01/29/o-que-e-a-lei-de-moore-e-porque-voce-deve-se-preocupar-com-o-fim-dela.htm>



▼ Entrada e Saída



- Os principais que realizam as atividade computacionais é o processador e a memoria ram
- Os demais são considerados perifericos que integram ao computador
- Driver → Programa responsavel que vai fazer a comunicação dos processos com saída ou entrada de um periferico



- Monitor
 - Composto por pixels
 - Cores de um pixel → vermelho, verde, e azul
 - Os pixels mudam sempre da esquerda pra direita
 - Uma tela full HD possui cerca de 1920 pixels, por 1080 pixels, oq gera por area cerca de 2 milhões de pixels, e as alterações são feitas por segundo
 - Placa de Vídeo
 - GPU → Responsavel pelo o processamento de imagem
 - Auxilia o processador
 - Pode ser integrada ao mesmo

<https://www.youtube.com/watch?v=3BJU2drirtCM>

- Teclado
 - ativa uma linha energizada com uma coluna energizada



- O processador processa a linha e o caractere que pressionado

<https://www.youtube.com/watch?v=zfMvxTmY2rQ>

- Os *drivers de dispositivo* permitem escrevermos o mesmo código para diversos dispositivos diferentes.
 - São os drivers de dispositivo que abstraem as particularidades de cada dispositivo. Dessa forma, a gente pode escrever um código que desenha algo na tela e isso funcionará para vários monitores diferentes.
- Como telas atuais possuem milhões de pixels, computadores pessoais precisam de placas de vídeo (GPUs) para renderizar eles a tempo para quem está usando.
 - Placas de vídeo são especialistas em realizar tarefas repetitivas, simples e em grande quantidade ao mesmo tempo, como o cálculo de vários pixels da tela. Muitas delas são instaladas com o processador e são chamadas de *placa de vídeo integrada*.

▼ Placa de video

- Uma **GPU** (*Graphics Processing Unit*) ou **placa de vídeo** é a unidade responsável pela renderização dos pixels na tela do computador e está em todos os computadores pessoais

O que é placa de vídeo e qual sua importância?

A placa de vídeo é um dos principais componentes de qualquer PC, especialmente para computadores voltados à jogos, e geralmente é um dos componentes mais caros do seu PC. Neste artigo, explicaremos o

 <https://www.oficinadanet.com.br/hardware/27791-o-que-e-a-placa-de-video>

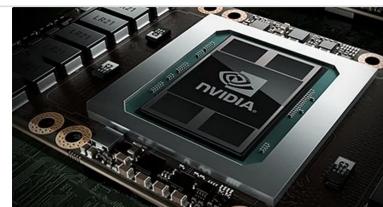


- Além disso, esse tipo de unidade de processamento é muito usado hoje em dia para diversas aplicações fora do escopo gráfico. Algoritmos de *machine learning* e ciência de dados utilizam a alta capacidade de processamento numérico da GPU para acelerar seus cálculos. Isso é chamada de **computação acelerada via GPU**

GPU e Deep Learning - Ciência e Dados

Cada vez mais as GPU's tem sido usadas para análises avançadas utilizando algoritmos de Machine Learning, especialmente Deep Learning. As GPU's estão disponíveis em desktops, notebooks,

 <https://www.cienciaedados.com/gpu-e-deep-learning/>



- Por fim, para o ramo de *machine learning*, foi criada uma nova unidade para acelerar a computação: a **TPU** (*Tensor Processing Unit* ou Unidade de Processamento de Tensores). Elas aumentam muito a performance nessas aplicações, em relação à GPU, e é usada em grandes empresas de tecnologia.

<https://www.youtube.com/watch?v=MXxN4fv01c8>

Nvidia, GPUs, Games e Deep Learning - Hipsters #204 - Hipsters Ponto Tech

Podcast: Play in new window | Download | Embed RSS | More Compartilhar Assinar
Dos dias 22 a 26 de Junho vai rolar a Imersão Gamedev JavaScript: sua oportunidade de aprender a criar um game do zero! Matricule-se gratuitamente!

 <https://www.hipsters.tech/nvidia-gpus-games-e-deep-learning-hipsters-204/>

Nvidia, GPUs, Games e Deep Learning



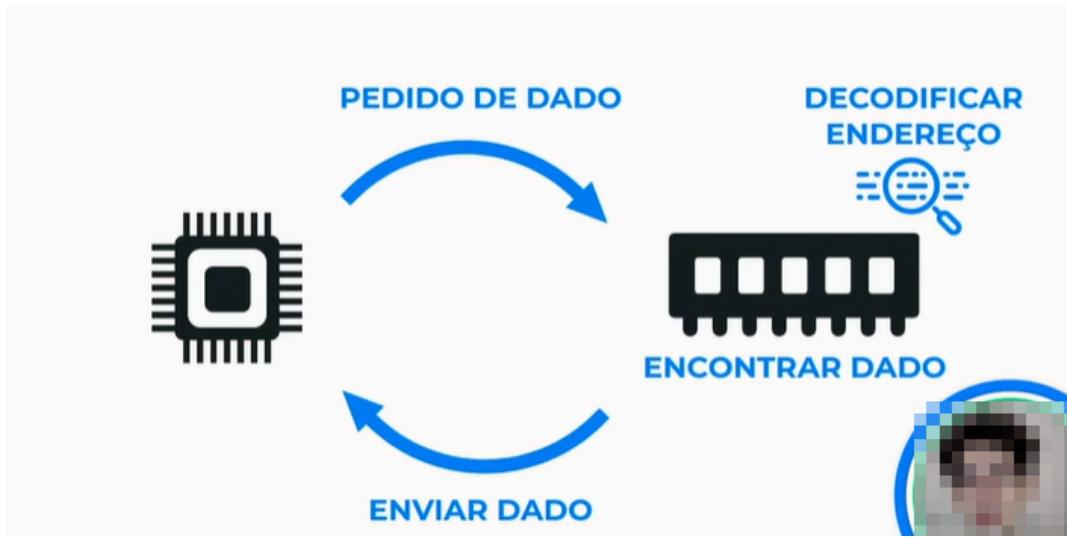
▼ Tarefas Simultâneas

-

▼ Como a memoria funciona ?

▼ Cache - Trazer para perto

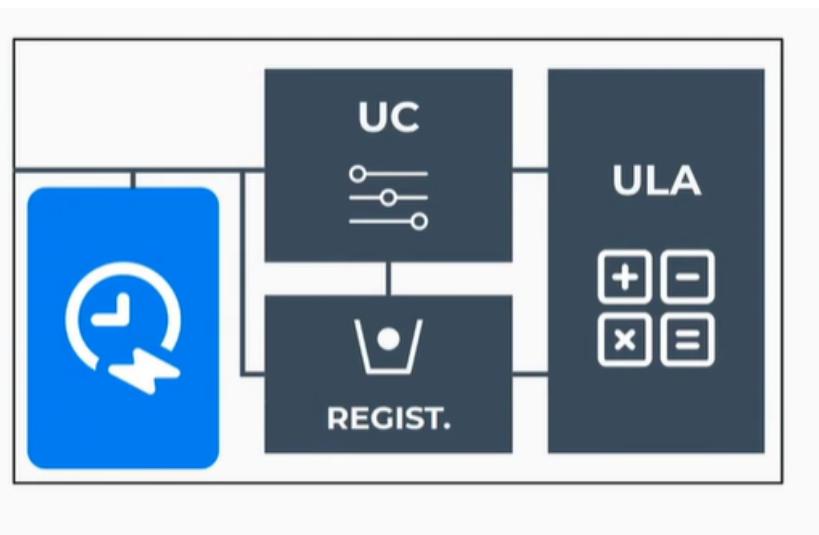
- O ciclo de uma solicitação de um dado do processador para a memoria ram segue da seguinte forma



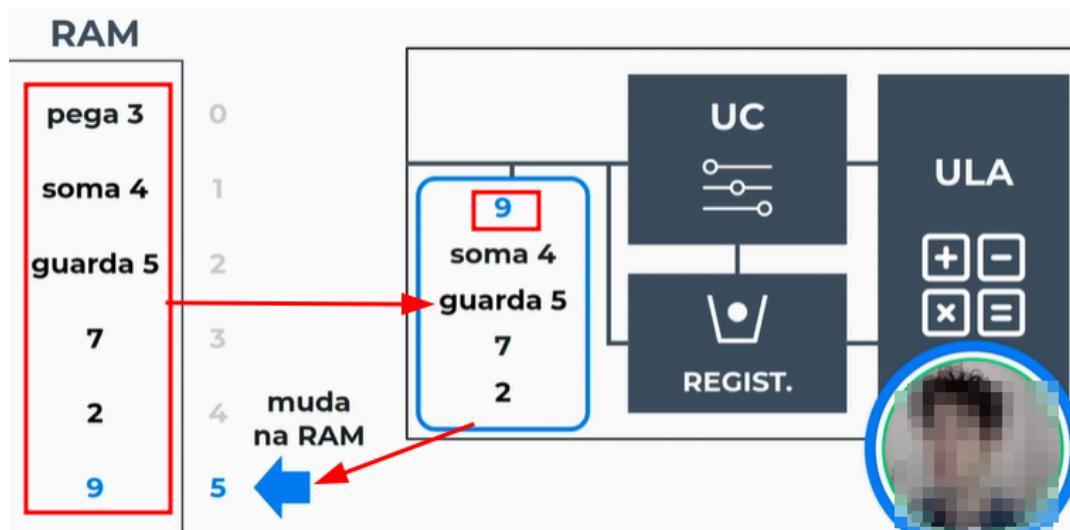
- Para melhorar a performance de comunicação com a memoria trazemos os dados para mais perto do processador
- No caso em um computador moderno temos dois tipos de memoria



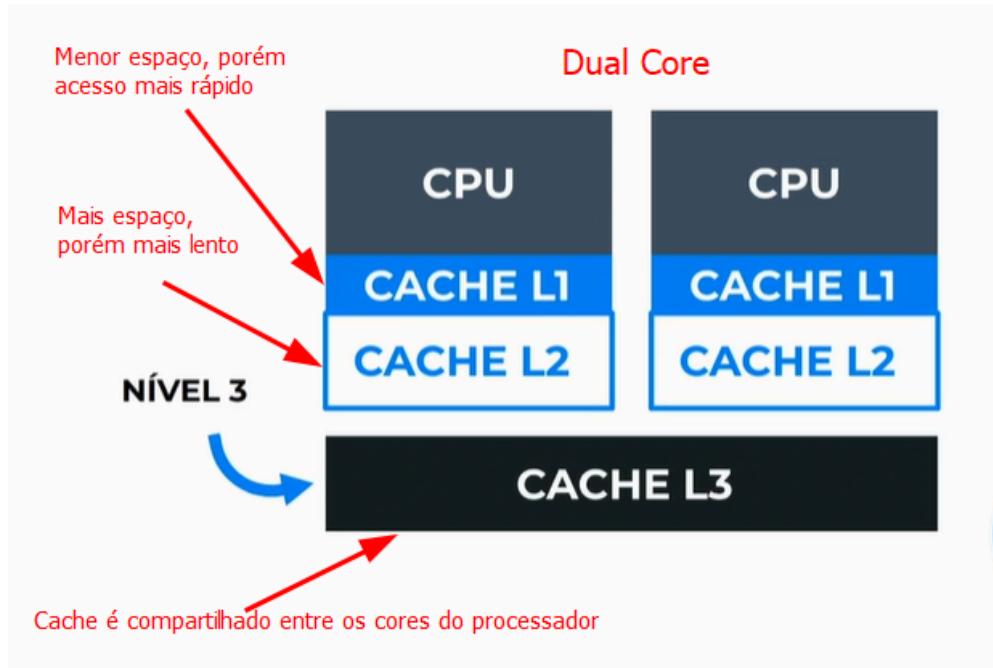
- A SRAM vai alocada dentro dos processadores atuais facilitando a comunicação



- Essa memoria é denominada memoria cache
- Ela pega temporariamente os dados da RAM, modifica quando necessário e manda de volta pra ram,



- Existem varios 3 tipos de cache



▼ Outros tipos de cache

- A memória cache é uma técnica muito importante para otimizar o funcionamento do processador e é cada vez mais explorada atualmente.
- Mesmo assim, essa não é uma ideia limitada aos componentes do processador mas é usada em quase todas as comunicações de dados atualmente.
- **Cache em navegadores**
 - Quando você acessa páginas em um navegador, como Chrome ou Firefox, você precisa se comunicar com servidores pela internet e esperar todo o conteúdo da página ser transferido para seu computador.
 - Como as páginas não costumam ser alteradas com frequência, os navegadores costumam guardar esses dados dentro do próprio computador.
 - Dessa forma, a próxima vez que você acessar ela, será muito mais rápido.
 - Também é usado em outros aplicativos, além de navegadores, como o Facebook.
- **Cache em bancos de dados**
 - Muitos bancos de dados utilizam uma estrutura de dados rápida (muitas vezes em memória RAM) para guardar essas respostas, como o Memcached e o Redis
- **Cache em servidores**
 - Muitas vezes, alguns serviços de um servidor podem ser bem mais requisitados que outros. Por exemplo, uma página inicial de um site de computadores que mostra todos os produtos em desconto naquele dia.

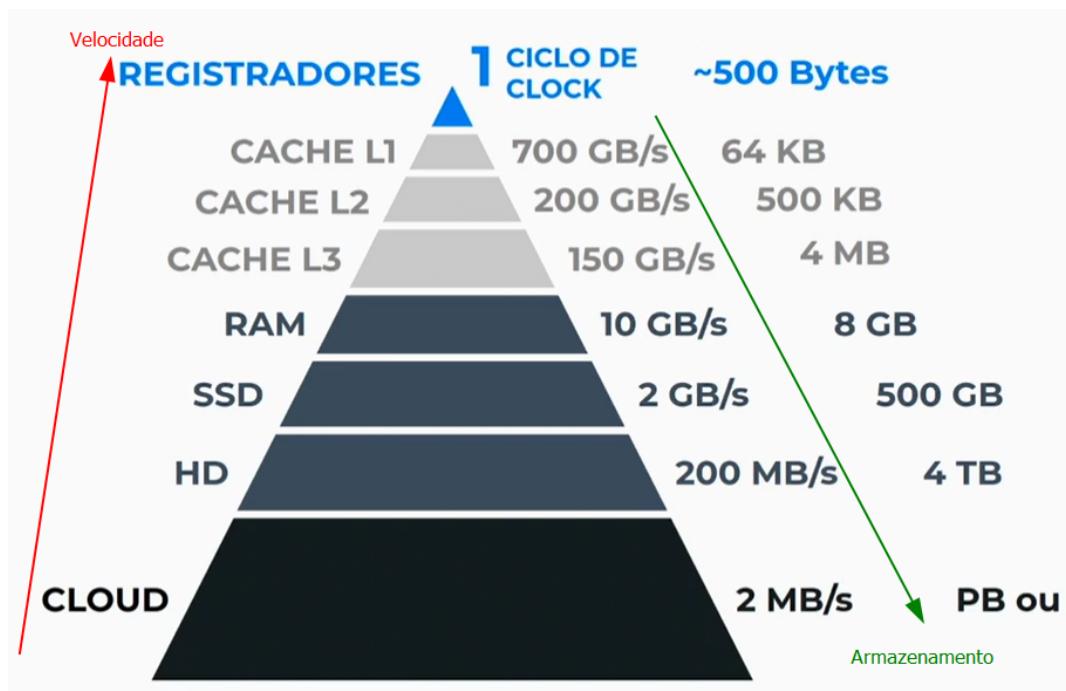
- Para acelerar a resposta e diminuir a carga nas máquinas, esses equipamentos podem guardar versões prontas dessas páginas durante um período de tempo.
- Assim, sempre que uma pessoa pedir a página para o servidor, ele não precisará buscar no banco de dados nem fazer um processamento complexo para devolver a resposta.
- Esse processo é normalmente feito em conjunto com uma **Content Delivery Network** (CDN) ou Rede de Distribuição de conteúdo.
- Elas basicamente distribuem cópias das suas mídias para diversos servidores para que o dado esteja o mais próximo possível de você.

▼ Programas cache - Friendly

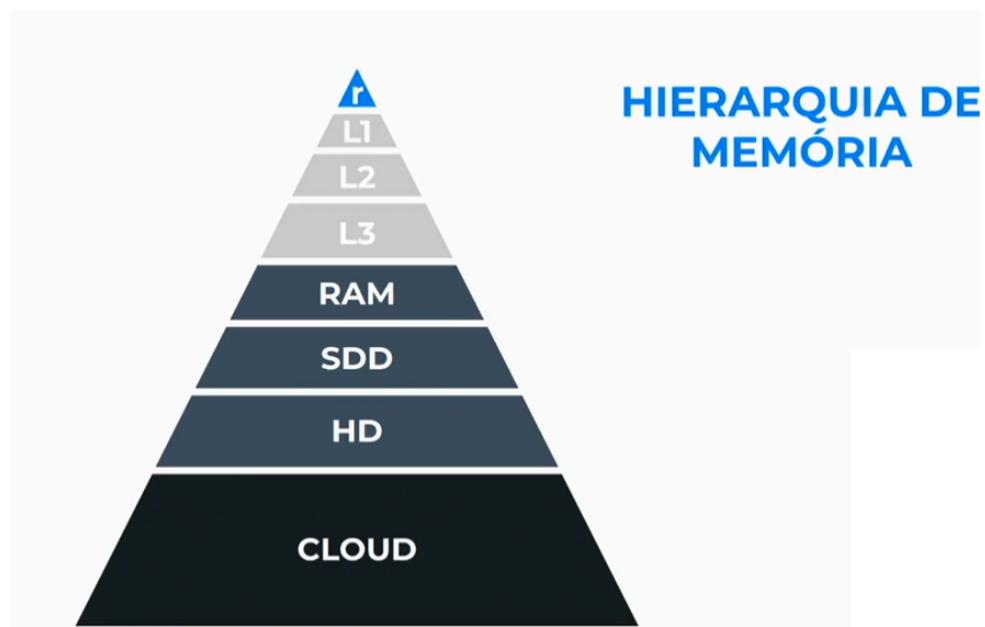
- Algumas linguagens que oferecem um controle maior na alocação de memória, como C e C++, podem ter o tempo de execução de seus programas influenciados pela memória cache.
- Ou seja, só mudando a ordem de execução de alguns trechos de código, ele pode ficar bem rápido ou lento.
- Programas que conseguem otimizar o máximo isso são chamados de *cache-friendly* ou bons para o cache.
- Um exemplo ruim é quando as estrutura do algoritmo faz execução não de forma linear. Assim, quando o computador for copiar um bloco de dados para o cache, o próximo elemento da nossa lista nunca vai estar lá.
- Não são todas as implementações de linguagens que precisam desse cuidado com o cache.
- O Node.js, uma implementação de JavaScript, faz várias otimizações que quase mascaram esse tipo de coisa.

▼ Hierarquia de memoria

- As características principais da memoria se divide em 3 pontos
 - Custo
 - Capacidade de armazenamento
 - Velocidade de acesso
- Se quisermos uma memoria com velocidade ela vai ser mais cara e a capacidade de armazenamento será menor
- Se quisermos uma memoria com capacidade de armazenamento, ela vai ser mais barata, porém não vai ser tão rápida
- Hirarquia de memorias



- Princípio da localidade
 - Localidade temporal → Acessar de novo em breve
 - Princípio do cache
 - Localidade Espacial → Acessar o vizinho em breve
 - Princípio de armazenamento um bloco inteiro para acesso posterior



- Nesta aula, vimos que o computador acessa dados a partir de diferentes tipos de memória, cada uma com sua particularidade. Por que precisamos desses vários tipos de memória?
 - Como o princípio da localidade diz que nós usamos a mesma porção de dados durante um período de tempo, é importante ter memórias rápidas para poucos dados e lentas para muitos dados.
 - Nós costumamos acessar sempre uma mesma porção de dados durante um período de tempo. Por isso, temos várias camadas de memória com capacidades e velocidades diferentes. Dessa forma, a região pequena que vai ser mais acessada vai estar numa memória mais rápida que outros dados numa região mais abrangente.
 - Memórias rápidas são muito caras, então não é viável usá-las para armazenar uma grande quantidade de dados.
 - Memórias rápidas são muito caras para armazenar grandes quantidades de dados e elas gastam mais energia que outras. Por exemplo, uma RAM de 16GB é quase o mesmo preço de um HD de 2TB (100 vezes mais capacidade).

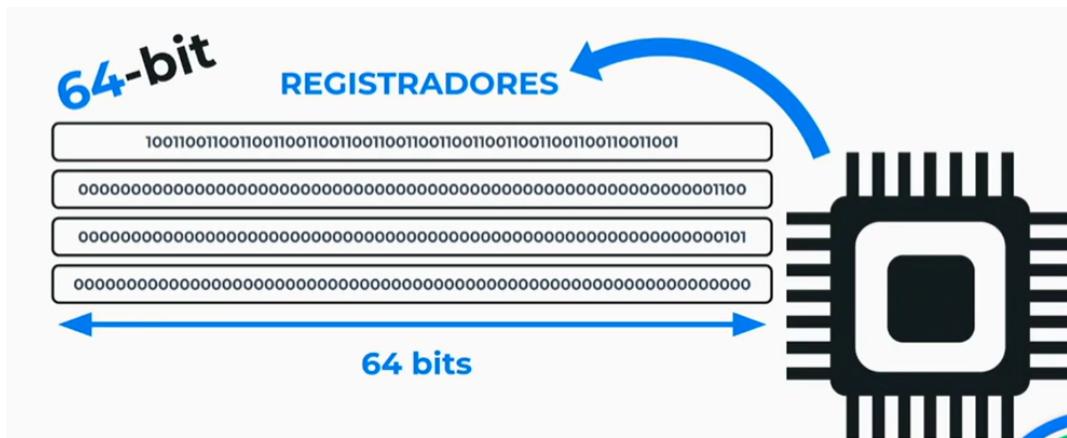
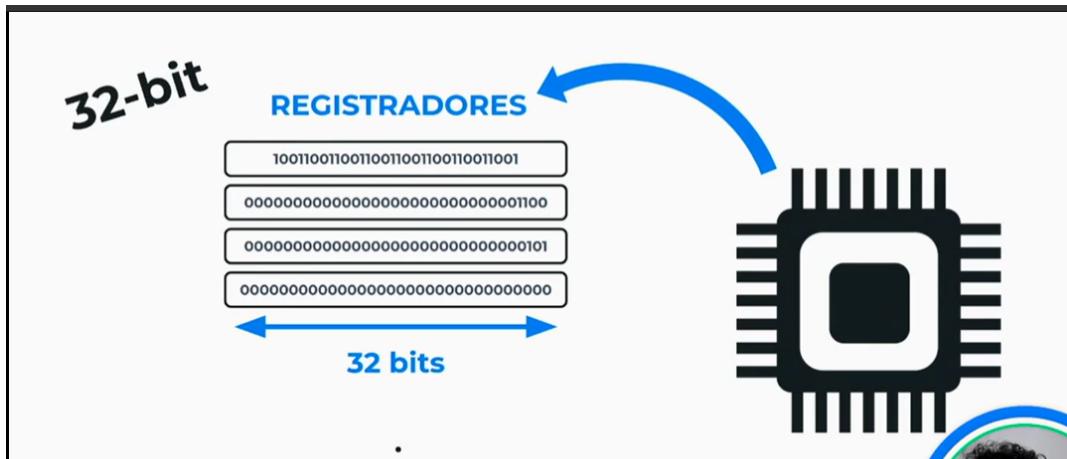
▼ Fitas magnéticas

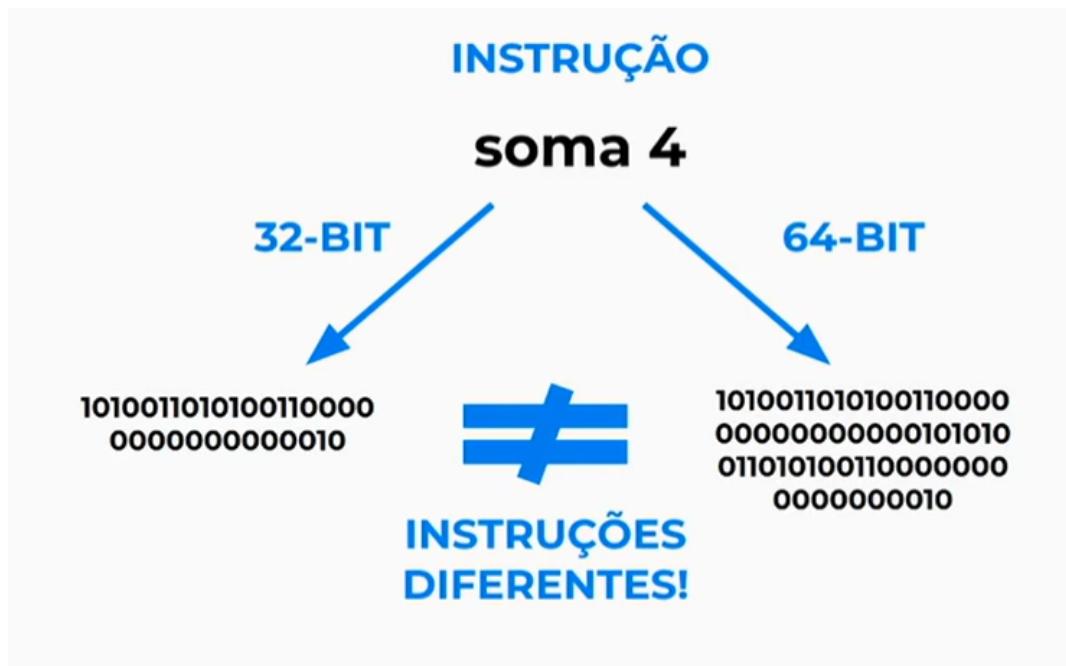
- Esse tipo de tecnologia ficou muito popular entre os anos de 1970 e 1990 para o armazenamento de áudio, com as **fitas cassetes**
- Uma das grandes desvantagens das fitas magnéticas é que elas são muito lentas para ler e escrever! A coisa complica mais se você precisa fazer o acesso aleatório de dados, porque a máquina precisaria rebobinar toda a fita para conseguir ir de um ponto da memória até outro.
- Por outro lado, fitas magnéticas podem armazenar enormes quantidades de dados por um preço baixo e possuem um tempo de vida bem alto.
- Estimativas de 2016 da Forbes mostraram que HDs são 65% mais caros que fitas.
- Além disso, você pode encontrar fitas com 12 TB de capacidade (e, a partir de 2030, podem aparecer modelos com 400 TB).
- Ainda, fitas tem um tempo de vida de até 30 anos, 6 vezes maior que um HD.
- Esse é o cenário perfeito para momentos em que precisamos armazenar grandes quantidades de dados que vamos mexer muito raramente: os backups.
- Muitos dados no mundo são armazenados em fitas, como física de partículas, arquivos nacionais, grandes filmes e até bancos.

▼ Processador de 32 ou 64 bits

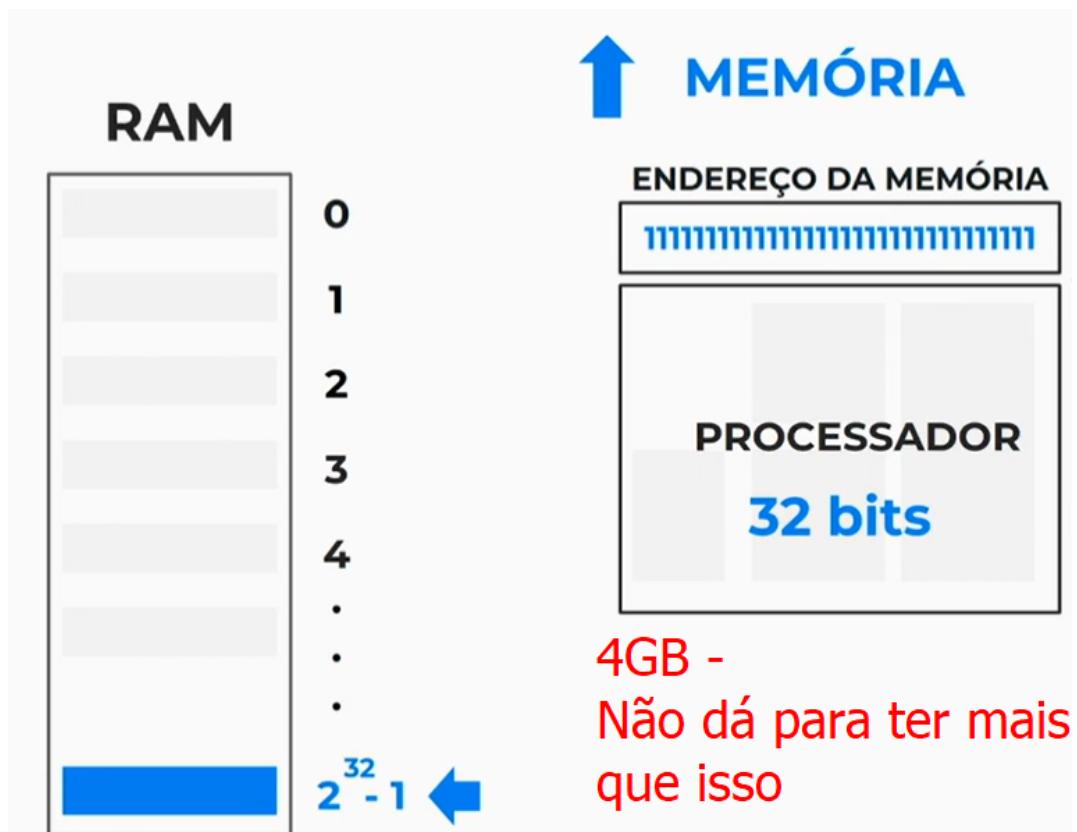
tamanho de informação
que pode ser processada
na CPU em um ciclo de
clock

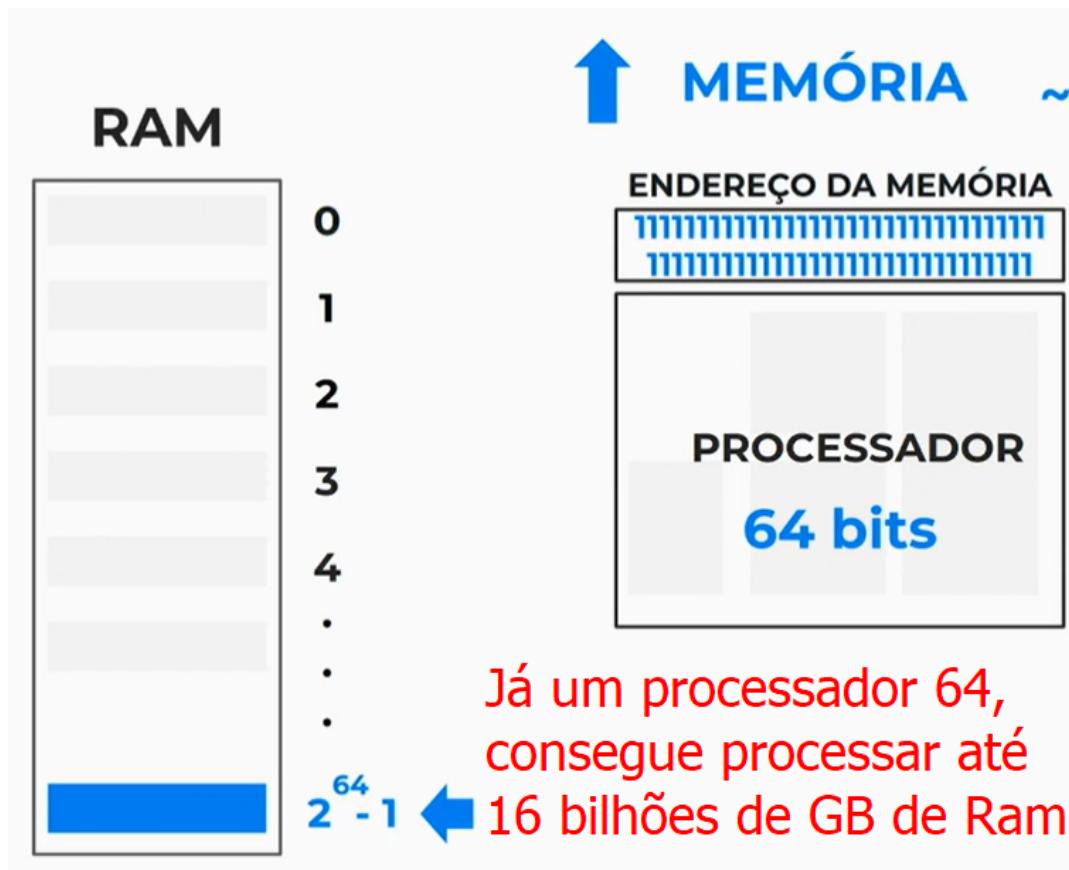
32-bit ou 64-bit?





- Processador 64 bits
 - + Processador + Otimização e + memoria!
- Em um processador 32 bits, não há espaço suficiente para processos acima de 4GB



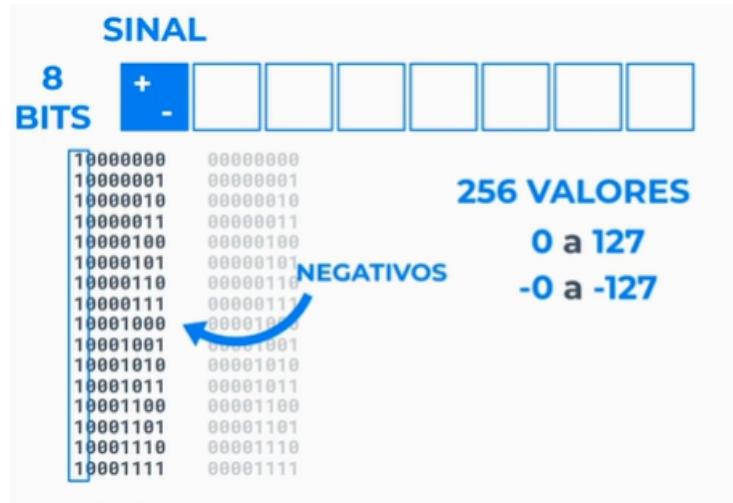


- Um processador 64 consegue alocar 32 bits normalmente.

▼ Como os dados são armazenados ?

▼ Números inteiros

- Os valores que colocamos nas variáveis ficam armazenados na memoria, sejam eles números ou letras
- Para valores positivos é acrescentado um bit na memoria para cada numero
- Para os números negativos é acrescentado um bit extra que representa o sinal de “-”



- Na linguagem C → 4 Bytes está para um numero inteiro

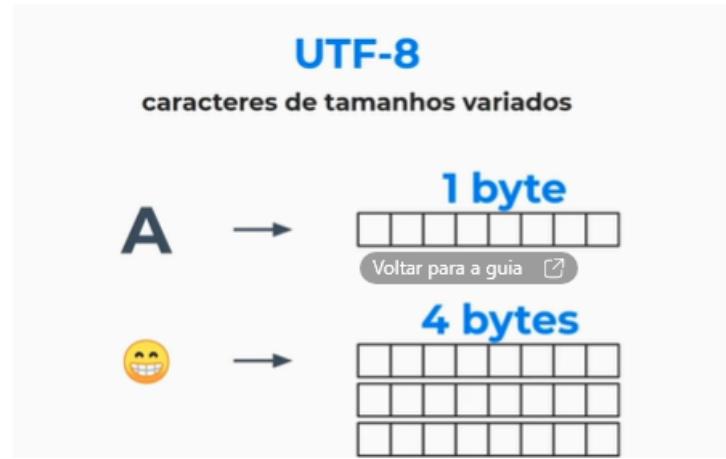
▼ Caracteres

- Para as Letras a representação é feita da mesma forma que para numeros sendo que se 0 é A, 1 é B, etc. Para o simbolo temos um numero que será alocado na memoria.
- Para a ordem se 0=A ou 0=a → Criado ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[\NUL]	32	20	[\SPACE]	64	40	[@]	96	60	~
1	1	[\START OF HEADING]	33	21	!	65	41	[@]	97	61	a
2	2	[\START OF TEXT]	34	22	!	66	42	b	98	62	b
3	3	[\END OF TEXT]	35	23	#	67	43	c	99	63	c
4	4	[\END OF TRANSMISSION]	36	24	\$	68	44	d	100	64	d
5	5	[\ENQUIRY]	37	25	%	69	45	e	101	65	e
6	6	[\ACKNOWLEDGE]	38	26	&	70	46	f	102	66	f
7	7	[\BELLI]	39	27	'	71	47	g	103	67	g
8	8	[\BACKSPACE]	40	28	(72	48	h	104	68	h
9	9	[\HORIZONTAL TAB]	41	29)	73	49	i	105	69	i
10	A	[\LINE FEED]	42	2A	*	74	4A	j	106	6A	j
11	B	[\VERTICAL TAB]	43	2B	+	75	4B	k	107	6B	k
12	C	[\FORM FEED]	44	2C	,	76	4C	l	108	6C	l
13	D	[\CARRIAGE RETURN]	45	2D	-	77	4D	m	109	6D	m
14	E	[\SHIFT OUT]	46	2E	.	78	4E	n	110	6E	n
15	F	[\SHIFT IN]	47	2F	/	79	4F	o	111	6F	o
16	10	[\DATA LINK ESCAPE]	48	30	0	80	50	p	112	70	p
17	11	[\DEVICE CONTROL 1]	49	31	1	81	51	q	113	71	q
18	12	[\DEVICE CONTROL 2]	50	32	2	82	52	r	114	72	r
19	13	[\DEVICE CONTROL 3]	51	33	3	83	53	s	115	73	s
20	14	[\DEVICE CONTROL 4]	52	34	4	84	54	t	116	74	t
21	15	[\NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	u	117	75	u
22	16	[\SYNCHRONOUS IDLE]	54	36	6	86	56	v	118	76	v
23	17	[\ENG OF TRANS. BLOCK]	55	37	7	87	57	w	119	77	w
24	18	[\CANCEL]	56	38	8	88	58	x	120	78	x
25	19	[\END OF MEDIUM]	57	39	9	89	59	y	121	79	y
26	1A	[\SUBSTITUTE]	58	3A	:	90	5A	z	122	7A	z
27	1B	[\ESCAPE]	59	3B	:	91	5B	{	123	7B	{
28	1C	[\FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	\
29	1D	[\GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[\RECORD SEPARATOR]	62	3E	>	94	5E	-	126	7E	-
31	1F	[\UNIT SEPARATOR]	63	3F	?	95	5F	-	127	7F	[DEL]

- Como o ASCII era feito em 7 bits, após a modernização para 8 bits de armazenamento foi lançado o latin1 que aloca os demais sinais.
- UNICODE → Fazer letras e caracteres serão numeros ex : A → 65, emoji → 128513, etc. 3 bytes será necessário para cada caracter
- Por fim para resolver o problema do unicode com relação aos outros protocolos escritos, surgiu o UTF-8



▼ O Contexto da Codificação

- 65 → A
- O computador não sabe se está falando de numero ou caracter, isso é indicado no código.
- Imprimindo em C

```
home > andrew > c chars.c > main()
1 #include <stdio.h>
2
3 int main() {
4     printf("%d\n", 65);
5     return 0;
6 }
```

gcc chars.c -o chars; ./chars
65

D = Decimal

```
home > andrew > c chars.c > main()
1 #include <stdio.h>
2
3 int main() {
4     printf("%c\n", 65);
5     return 0;
6 }
```

gcc chars.c -o chars;
A

C = Caracter

▼ UTF-8,UTF-16,UTF-32

- Atenção: todos os três sistemas conseguem codificar todos os caracteres do Unicode. Apenas a quantidade de bits que eles usarão para fazer isso que muda.

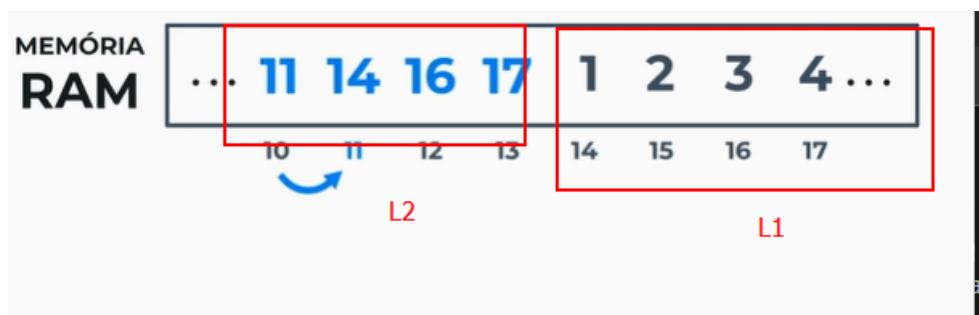
- UTF-8 → Essa codificação representa os caracteres em pedaços de **8 bits** e sua grande vantagem é manter os textos codificados apenas em ASCII (a grande maioria da Web no passado) intactos.
- UTF-16 → Essa codificação representa os caracteres em pedaços de **16 bits**. Isso significa que a codificação não é mais compatível com ASCII e ocupa o dobro de memória em textos que possuem apenas caracteres da língua inglesa. Sua grande vantagem é ocupar menos espaço quando o texto possui muitos caracteres asiáticos (UTF-8 usaria 3 bytes por caractere e UTF-16 apenas 2).
- UTF-32 → Essa codificação representa os caracteres em pedaços de **32 bits**. Ela não é compatível com ASCII e ocupa **4 vezes** mais espaço em textos que só utilizavam ASCII. Mesmo assim, diferentemente do UTF-8 e UTF-16, essa codificação tem um **tamanho fixo** e essa é sua grande vantagem.

▼ Listas

- As listas são representadas na memória de forma contínua, e isso é eficiente com cache, sendo fácil de encontrar os valores
- O índice segue o mesmo padrão dos números
- A lista segue da seguinte forma



- Na memória é marcado o começo de cada lista



▼ Valor e referência

- Para cada valor dentro da memoria, é gerando os ponteiros, que significa que para cada ponteiro na memoria, irá ocorrer um ligação entre outros valores , exemplo:

```
> const lista1 = [1, 1, 1, 1, 1];
undefined
> lista1
[ 1, 1, 1, 1, 1 ]
> █
```

```
> const lista2 = lista1;
undefined
> lista2
[ 1, 1, 1, 1, 1 ]
> █
```

- Caso modificamos um numero na lista1 ou 2 ocorrerá alteração nas duas listas

```
> lista2[0] = 3;
3
> lista2
[ 3, 1, 1, 1, 1 ]
> lista1
[ 3, 1, 1, 1, 1 ]
> █
```

Indice 0 = 1º
Valor

- Existe uma diferença entre passar a referencia de uma variavel, passar o valor da variavel.
- Referencia da variavel é a forma como foi demonstrado os ponteiros de uma lista

▼ Passagem por referência

Stefany vai fazer aniversário e, para isso, ela decidiu criar um programa em **JavaScript** que mexe com idades.

Primeiro, ela fez uma função `calculaProximaIdade()`, que recebe a idade que ela tem agora e imprime quantos anos ela terá depois do aniversário. Em seguida, ela criou a função `calculaProximasIdades()`, que recebe a lista de idades dela e de seus amigos e devolve quantos anos todos terão ao final do ano. Por fim, ela fez uma função `calculaIdadesDaqui5Anos`, que recebe a mesma lista de antes mas devolve as idades que todos terão daqui cinco anos.

O programa ficou da seguinte forma:

```
function calculaProximaIdade(idade) {
    idade += 1;
    console.log(idade);
}

function calculaProximasIdades(idades) {
    for (let i = 0; i < idades.length; i += 1) {
        idades[i] += 1;
    }
}
```

```

        console.log(idades);
    }

function calculaIdadesDaqui5Anos(idades) {
    for (let i = 0; i < idades.length; i += 1) {
        idades[i] += 5;
    }
    console.log(idades);
}

const idadeStefany = 21;
calculaProximaIdade(idadeStefany);

const idadesAmigos = [idadeStefany, 20, 23, 18, 7];
calculaProximasIdades(idadesAmigos);

calculaIdadesDaqui5Anos(idadesAmigos);

```

O programa de Stefany se comporta como o esperado?

- Não. A segunda função modifica o valor da variável `idadesAmigos`. Assim, a última função vai mostrar `[27, 26, 29, 24, 13]`, todos com exatamente um ano a mais que o esperado.
 - Listas são passadas por referência para funções. Dessa forma, se modificamos seu conteúdo dentro da função, a lista é modificada. Por outro lado, isso não acontece com número, que são passados por valor. Sabendo disso, tente arrumar o programa de Stefany e teste o resultado!

▼ Imagem e áudio

- Imagens podem ser representadas como uma lista de três componentes RGB (vermelho, verde e azul), onde cada componente vai representar um pixel.
- O [vídeo do code.org](#) explica de forma bem ilustrada como elas são armazenadas e como aplicamos filtros nelas. Além disso, alguns formatos não armazenam imagens dessa forma. Formatos como JPEG realizam a compressão de uma imagem e você pode ver como isso funciona no [vídeo do Leo Isikdogan](#)
- Áudios são propagados como ondas, então podemos capturar a altura que a onda está em certos períodos de tempo, isso dá um número, e guardamos em uma lista de alturas. Você pode ver uma explicação mais detalhada disso no [vídeo do Computerphile](#).

▼ Números de ponto flutuante

- É a forma na qual o computador entende os números grandes
- Semelhante a notação científica



- Porém o coeficiente não consegue representar tudo, e os valores são arredondados
- A base que o expoente usa é 2! portanto 0,5 é representado como ponto flutuante 1×2^{-1}
- Portanto, devido à base 2 não é possível representar o número exato

NÚMERO DE PONTO FLUTUANTE

0,3 + 2 * ERRO



▼ Inteiros de precisão arbitrária

- Essa técnica, também chamada de *BigInteger* ou *Bignum*, armazena números sem restrição de posição. A ideia é, em vez de fixar a mesma quantidade de bits para representar todos os números, utilizar uma quantidade variável.
- Isso pode ser feito guardando cada dígito do número numa lista e as contas são realizadas dígito a dígito, como aprendemos na escola. Para otimizar, as implementações não usam a nossa base decimal (dígitos de 0 a 9) mas uma base de 2^{30} ("dígitos" de 0 a $2^{30} - 1$). Com isso, as operações dígito a dígito são feitas como operações de inteiros convencionais.
- Dessa forma, não há uma restrição para o tamanho que esses números podem ter (além do limite de memória que o seu computador deve ter para armazenar tudo isso). Por isso, é dito que esses números têm precisão arbitrária
- De qualquer forma, se um *Bignum* tem precisão infinita por que não usamos ele como o padrão para tudo? Isso acontece porque fazer contas com ele é muito lento.

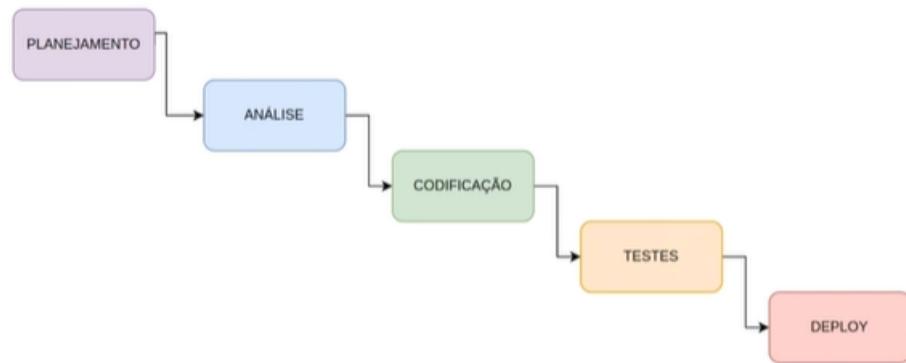
▼ Tipos em outras linguagens

- Vimos nessa aula como implementar diversos valores no computador. Mesmo assim, cada linguagem de programação usa essas ideias de forma diferente. Por exemplo, em JavaScript, todos os números são representados pelo tipo Number, que é um número de ponto flutuante de 64 bits.

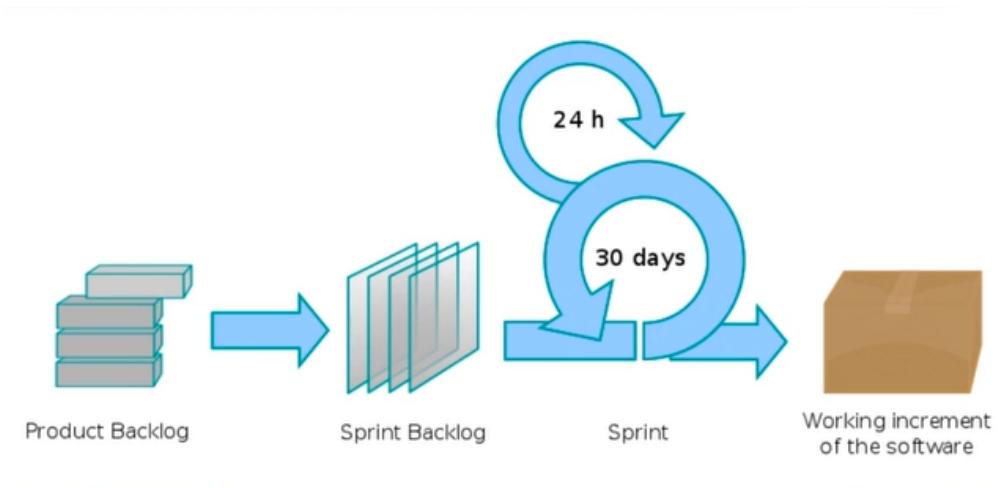
▼ O que há em DevOps

▼ Shift Left em DevOps

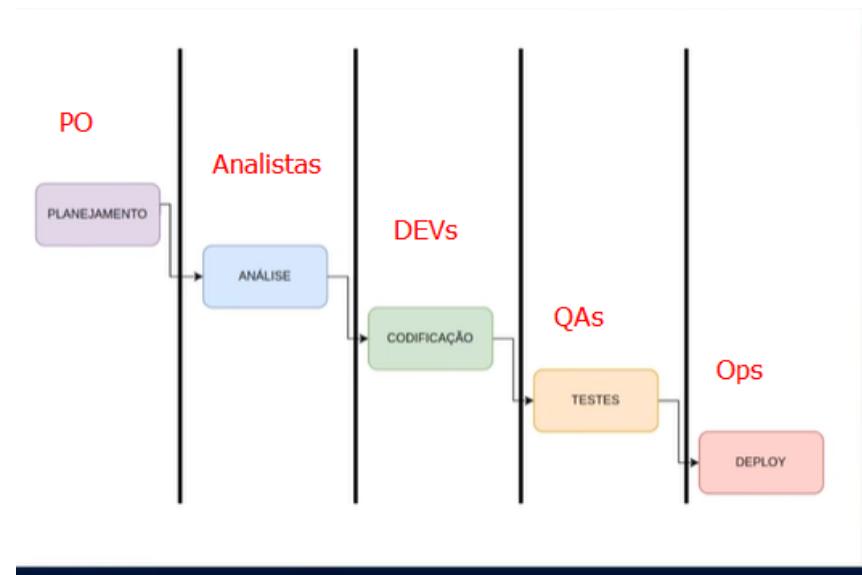
- Processos de desenvolvimento de software
 - Modelo Cascata



- Este processo possuia muitos problemas na entrega
 - Metodologia Ágil



- Ainda haverá as etapas do modelo cascata, porém na forma em entrega continua

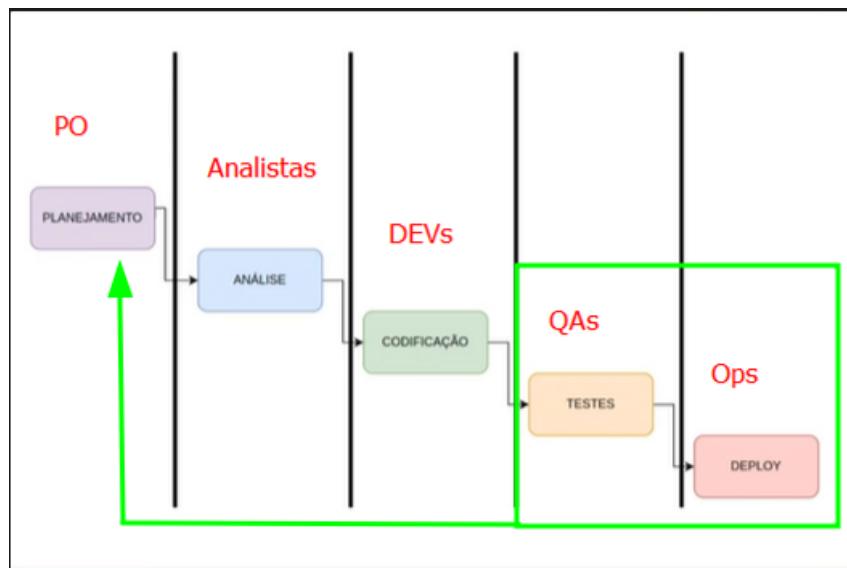
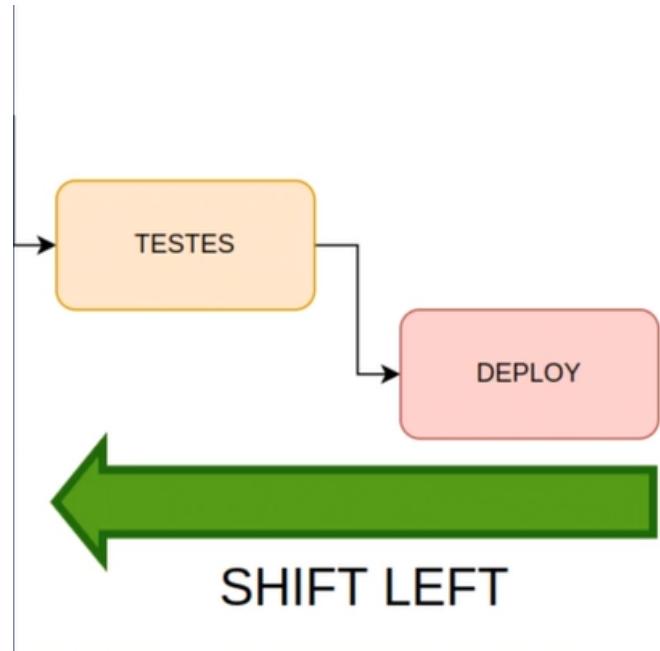


- Porém ainda ocorre problemas de divisão :

- Espera - osiosidade das esquipes
- Gargalo
- Sobrecarga do trabalho
- Atrito entre times



- Shift Left → Deslocar para esquerda



- Ao invés de testar e implantar no final, estas etapas também serão trabalhadas no inicio, na fase de planejamento, no inicio do ciclo
 - Colaboração entre times → o alinhamento será entre todos os times
 - Planejamento prévio das etapas
 - Evita gargalo e sobrecarga
 - Agiliza entrega e evita retrabalho

▼ Cloud

- Uma nuvem é composta de vários recursos de computação, que abrangem desde os próprios computadores (ou instâncias, na terminologia de nuvem) até redes, armazenamento, bancos de

dados e o que estiver em torno deles. Ou seja, tudo o que normalmente é necessário para montar o equivalente a uma sala de servidores, ou mesmo um *data center* completo, estará pronto para ser utilizado, configurado e executado.

- A entidade que fornece esses recursos de computação é chamada de **provedor de nuvem**. Os provedores de nuvem mais famosos são empresas como Amazon, Microsoft e Google.
- A entidade que fornece esses recursos de computação é chamada de **provedor de nuvem**. Os provedores de nuvem mais famosos são empresas como Amazon, Microsoft e Google.
- **Tipos de Cloud**

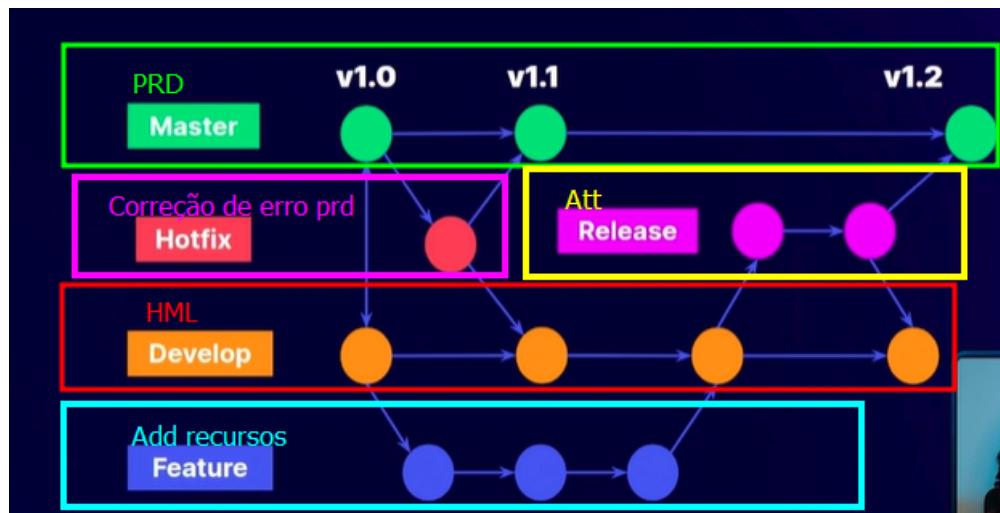
- IAAS
 - IaaS (que significa infraestrutura como serviço) é o tipo mais comum de serviço de nuvem. Sob IaaS, o provedor de nuvem fornece a infraestrutura de TI, como servidores, armazenamento e redes, e paga-se com base no uso.
 - A maioria dos recursos de TI oferecidos no modelo IaaS não são pré-configurados, o que significa que o consumidor tem um alto grau de controle sobre o ambiente da nuvem. Ele é quem deve configurar e manter qualquer software que deseje rodar em cima da infraestrutura fornecida.

- PAAS
 - PaaS (que significa plataforma como serviço) é um modelo de computação no qual o provedor de nuvem aloca, configura e gerencia toda a infraestrutura de computação, como servidores, redes como no IaaS, além de sistemas operacionais, bancos de dados e ferramentas de desenvolvimento e gerenciamento do negócio.
 - Outras palavras, no PaaS toda a configuração do banco de dados, segurança e replicações são realizadas pelo provedor com poucas margens de configuração, diferente do IaaS, onde as principais configurações são feitas pelo desenvolvedor, tornando o PaaS mais caro que o IaaS.

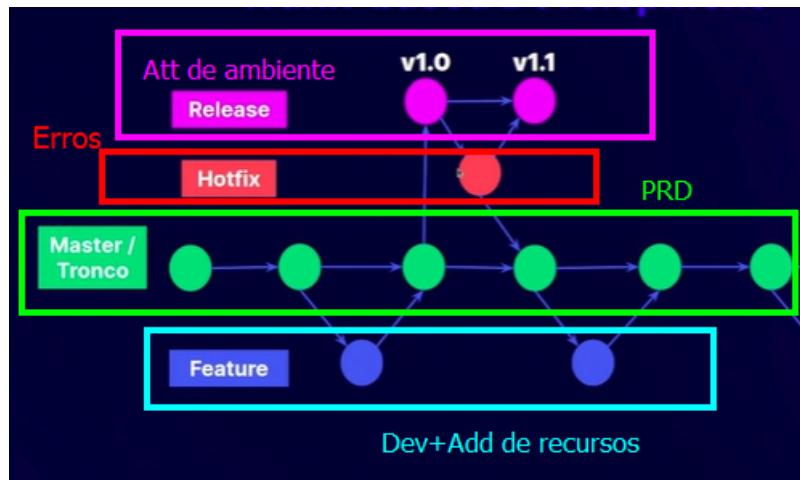
- SAAS
 - SaaS (que é software como serviço) é como um provedor de nuvem que entrega aplicativos de software sob demanda. Neste modelo o provedor gerencia não apenas a infraestrutura, mas também os aplicativos de software e os usuários que se conectam ao aplicativo pela internet.
 - O software é modelado como um serviço de nuvem compartilhado e disponibilizado aos usuários como um produto. Os consumidores de nuvem têm controle administrativo e de gerenciamento limitado.

▼ Git Flow versus Trunk-based development

- Sistemas de controle de versão
 - Git Flow - Fluxo de código



- Vantagem
 - Otimo para execução de projetos de codigo aberto
 - Supervisão de desenvolvedores juniores
 - Tempo não é uma restrição
- Desvantagem
 - Qnd está apenas começando
 - Qnd a entrega precisa ser rapida
- Trunk-based Development



- Vantagem
 - Quando se esta começando
 - Rapidez de entrega
 - Qnd trabalha principalmente com desenvolvedores seniores

- Desvantagem
 - Exige Experiência
- Trunk-based Development → CI/CD (mais defendido)
- IAAC - Qualquer aplicação depende de uma infraestrutura como computador, sistema operacional, rede, espaço no HD e muito mais. A **Infraestrutura como Código** permite descrever essas características em um **arquivo de texto** e executar como se fosse um código da aplicação

▼ DevSecOps



- Prioridade sempre será segurança
- Entregar com mais segurança
- Em CI/CD será acrescentar a Pipeline ferramentas de segurança. Caso seja encontrado algo dentro da integração que ocasiona um possível erro, a mesma será quebrada pela auditoria
- Priorização da segurança é o principal pilar da cultura DevSecOps, automação é fundamental, velocidade é importante, mas segurança com quase a mesma velocidade é mais!
- Etapas de uma pipeline com devsecops
 - Compilação, Auditoria, teste, package, deploy

▼ Observabilidade

- Validação e acompanhamento de uma aplicação
- Problema → Monitorar servidores e aplicações em sistemas distribuídos



- Monolito → Aloca toda a aplicação
- Microservices → serviços separados em sistemas separados, que representão uma aplicação
- Desafio → Agir proativamente ou reativamente em metrica e logs
- Pilares da observabilidade
 - Metricas → dados, graficos, validações
 - Prometheus - metricas customizadas, padrão - não plota as informações em graficos - bd de metricas
 - Grafana - visualizavel as metricas que foram criadas no prometheus - visualização
 - Traços distribuídos → investigar uma requisição por todos os sistemas que ela passou
 - Jaeger → intercepta a chamada e traça o rastro dela, coleta e gera os traços distribuidos
 - Service Mesh → gera os traços | Proxy → envoy, traefik, kong
 - Possui interface grafica para visualização
 - Logs → Traço que indica por onde a requisição passou
 - Gerenciador de logs - Irá gerar um arquivo pré configurado
 - Sidecar → Faz o deploy da aplicação
 - Logging agen instalado no host
 - Shell script
 - Graylog → trás todos os dados de log das aplicações

- Em sistemas distribuídos, observabilidade é crucial
- Muitas ferramentas no mercado (OSS e Proprietárias)
 - ELK
 - New Relic
 - Dynatrace
 - Prometheus
 - Grafana
 - Jaeger
 - Graylog

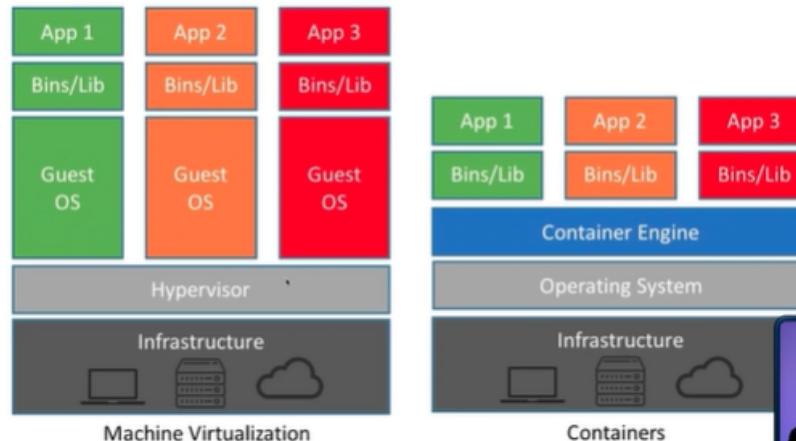
▼ Feature Flags ou Feature Toggles

- Feature Flag ou em tradução livre, alternância de recursos, é uma técnica que permite, dentre outras coisas:
 - Que seja possível ativar ou desativar uma funcionalidade em produção em tempo real sem fazer deploy
 - Disponibilizar uma funcionalidade para uma parte dos seus clientes para fazer experimentação;
 - Desativar uma funcionalidade, sem precisar fazer rollback do código (o que exigiria outro deploy da aplicação)
 - Ativar uma feature somente para testes, definindo quem vai ver a feature (que podemos escolher somente um dev, por exemplo)
 - Permite que a pessoa QA (Quality Assurance ou analista de qualidade) faça testes na aplicação com a funcionalidade habilitada e desabilitada
 - Facilita a remoção de funcionalidades que ninguém mais usa no seu sistema.
- A *ideia do Feature Flag* é economizar tempo e esforço de desenvolvimento do time, além de também diminuir riscos lançando funcionalidades de uma maneira mais incremental e evitando alterações manuais no sistema - que podem gerar erros.

- Uma forma simples de construir essas Feature Flags é usar uma API (ou micro serviço), que vai consultar em um banco de dados quais Feature Flags seu sistema possui.
 - Então o front-end, por exemplo, vai chamar essa API que vai ver no banco de dados correspondente, se existe essa flag e se ela está ativa ou inativa. Também existem plataformas pagas que oferecem esse serviço, com foco em teste a/b.

▼ Containers

- Antes era necessário um computador, com SO, para cada tipo de servidor, apache, nginx, tomcat, mysql, mongodb etc. E ainda havia a configuração de rede para cada máquina.
- Também havia o problema, na minha máquina funciona
- Em seguida surgiu as VMs, que eram criadas dentro de um servidor
 - Vantagem → Ambiente compartilhável, passível de centralização, rápida restauração, rápida configuração de rede
 - Desvantagem → Um SO por VM, desperdício de recursos, necessidade de hardware mais potente, dificuldade na criação
- ERA DOS CONTAINERS



- Um container não precisa de um SO completo. Possui uma container engine
- É uma camada fina, que isola determinada aplicação, e os containers compartilham as bibliotecas entre si
- Instalando o programa de container, será garantido que tudo será compartilhado, e a aplicação ficará nessa camada que irá isolar o processo, permitindo e acabando com o problema “na minha máquina funciona”
 - Vantagem → Mais leve, sem custo de manutenção de vários SOs, mais rápido de provisionar

▼ Serverless

- Executar o código sem se preocupar com o servidor, porém o servidor ainda existe a diferença é que o desenvolvedor irá se preocupar somente com a parte da aplicação e não com a infraestrutura
- No caso o provedor da nuvem será o responsável pela camada de infraestrutura
 - AWS Lambda, Google Functions, Azure Functions
- Vantagens
 - Pega pelo uso
 - Cada aplicação pode ser criada em uma linguagem diferente
 - Muitos eventos pré-configurados na cloud ajudam a criar arquiteturas orientadas a eventos
 - Autoscaling, altamente disponível por natureza
- Desvantagens
 - Duração de execução - o tempo é curto e pode variar para cada provedor
 - Vendor lock-in - Migrar pode levar tempo
 - Difícil de debugar
 - É necessária configuração extra para controlar (parcialmente) o ambiente de execução
- Componentes serverless AWS
 - Lambda, API Gateway, SQS, DynamoDB, SNS, S3

▼ SRE

- Desenvolvimento X Infra
 - Como confiar que o software irá funcionar corretamente em produção sem que nenhuma anormalidade aconteça?

- Confiabilidade → SRE (Engenharia de confiabilidade de sites)
 - Observabilidade
 - Através de monitoramento, automações para reduzir e liquidar tarefas manuais
 - Utilização de métrica, logs, e rastreabilidade operacional
 - Analise preditivas, mitigação de ambiente, antecipação de validações
 - Golden signal → latência, tráfego, erros, saturação → Base de observabilidade
 - Acompanhamento através de logs, métricas, e rastreabilidades
 - Chaos Engineering → Consiste no processo de falhar o ambiente para validação → Dia do caos
- SRE entrega confiabilidade
 - SRE é uma disciplina DevOps/DevSecOps

▼ Terminal & Prompt

▼ Windows prompt

▼ Primeiros comandos

▼ Primeiros Comandos

- CMD - prompt de comando
 - echo → escreve na linha do comando
 - dir → mostra os diretórios e arquivos na pasta atual
 - cd → troca de pasta
 - mkdir → cria pasta
 - mv ou move → move as pastas e arquivos
 - type → mostra o conteúdo do arquivo
 - echo + texto + ">" → Envia para um arquivo de texto
 - cd .. → Volta uma pasta
 - copy → copia o arquivo
 - rename → muda o nome do arquivo ou pasta
 - del → remove arquivo e pastas
 - cls → limpa o prompt

▼ Separador de diretórios

O caractere correto é o **contra-barra**: `\`

O caractere `>` tem o significado de *prompt* (o convite de escrever um comando no terminal).

O caractere `:` faz parte da partição, por exemplo: `C:`

A barra `/` é utilizada como separador no mundo Unix, mas também funciona no mundo Windows

```
cd C:  
cd Users/caelum
```

▼ Help

Cada comando possui uma pequena ajuda no terminal, basta digitar `help <nome do comando>`, por exemplo:

```
help dir
```

A opção `/O` existe para ordenar a saída do comando `dir` e possui mais parâmetros para definir qual ordem queremos, por exemplo:

Ordenando pelo tamanho (S-Size):

```
dir /O:S
```

Ou ordenando pela data:

```
dir /O:D
```

Pelo comando `dir` podemos também mostrar o conteúdo de um diretório que não é o atual.

Por exemplo, para **ordenar pela data**

e mostrar o conteúdo do diretório `C:`

digite: `dir /O:D C:\`

Você pode inverter a ordem dos parâmetros do comando `dir` para obter o mesmo resultado:

```
dir C:\ /O:D
```

▼ Trocar diretório

O comando para trocar de pasta é `cd` (*change directory*), por exemplo

```
cd C:\
```

Também podemos definir o caminho:

```
cd C:\Documentos\workspaces
```

▼ Diferença entre '.' e '..'

`.` se refere ao diretório atual, `..` se refere ao diretório logo acima na hierarquia.

Você pode testar isso e mostrar o conteúdo do diretório atual usando o `.`:

```
dir .
```

Nesse caso podemos omitir o `.` já que é o padrão do comando `dir`.

Análogo podemos usar `..`:

```
dir ..
```

Que mostra o conteúdo da pasta logo acima na hierarquia de diretórios.

O `.` e `..` também funcionam com outros comando como o `cd`

▼ Redirecionar a saída

O caractere `>` redireciona a saída de um comando.

Por exemplo, podemos gravar a saída do comando `set` em um arquivo:

```
set > variaveis.txt
```

E para mostrar o conteúdo basta digitar:

```
type variaveis.txt
```

▼ Add conteúdo a um arquivo

Quando colocamos o sinal de maior, duas vezes, o Prompt entende que só deve criar um arquivo novo quando o arquivo que pedimos não existir! Caso ele já exista, ele adiciona o novo conteúdo ao final do arquivo, sem sobrescrevê-lo!

Veja só como funciona:

Digamos que primeiro criamos um novo arquivo de texto com o nosso conhecido comando `echo`:

```
echo Ola mundo > arquivo.txt
```

Ao abrirmos nosso `arquivo.txt`, vemos o texto que esperávamos:

```
type arquivo.txt
```

```
//arquivo.txt  
Ola Mundo
```

Porém se agora desejarmos adicionar um novo texto abaixo de "Ola Mundo"? Utilizamos o '`>>`'!

Desta forma:

```
echo Novos dados! >> arquivo.txt
```

Quando abrimos nosso `arquivo.txt`, vemos que o texto foi adicionado corretamente:

```
type arquivo.txt
```

```
//arquivo.txt  
Ola Mundo  
Novos dados!
```

▼ Pasta home do user

A pasta do seu usuário é conhecida como pasta **Home**, e essa nomenclatura é adotada por muitos desenvolvedores quando querem se referir a pasta do usuário, independentemente do sistema operacional.

▼ Comando tree

- O comando `tree` mostra as pastas e subpastas organizadas em uma árvore.

```
PS C:\Users\LucasSantos> tree
Listagem de caminhos de pasta para o volume OS
O número de série do volume é 4E78-A676
C:.
└── azure
    ├── commands
    ├── ErrorRecords
    ├── logs
    └── telemetry
└── eclipse
    └── org.jkiss.dbeaver.product_22.3.0_1535670467_win32_win32_x86_64
        ├── configuration
        │   ├── .settings
        │   ├── org.eclipse.core.runtime
        │   │   └── .manager
        │   ├── org.eclipse.e4.ui.css.swt.theme
        │   ├── org.eclipse.equinox.app
        │   │   └── .manager
        │   ├── org.eclipse.equinox.launcher
        │   │   └── org.jkiss.dbeaver.ui.app.standalone_22.3.0.202212041619
        │   └── org.eclipse.osgi
        │       └── .manager
        ├── 124
        │   └── 0
        │       └── .cp
        │           └── icons
        │               └── obj
        ├── 134
        │   └── 0
        │       └── .cp
        │           ├── icons
        │           │   └── full
        │           └── org
        │               └── eclipse
        │                   └── jface
        │                       ├── action
        │                       │   └── images
        │                       ├── fieldassist
        │                       │   └── images
        │                       └── wizard
        │                           └── images
        ├── 146
        │   └── 0
        │       └── .cp
        ├── 151
        │   └── 0
        │       └── .cp
        └── icons
            └── full
                ├── dlcl16
                ├── elcl16
                └── etool16
```

▼ Comando more

O comando `more` funciona de um jeito semelhante ao comando `type`, com a diferença de exibir página por página do arquivo no terminal, em vez de mostrar todo de uma vez.

O seu uso é análogo ao comando `type`, podendo ser chamado assim:

```
more arquivo.txt
```

Ele exibirá uma página de cada vez do arquivo, sendo muito útil quando queremos exibir arquivos de texto com várias linhas para ir lendo lentamente, em vez de abri-lo todo no terminal de uma vez. Um exemplo disso é quando queremos ler os logs de uma aplicação que está em um servidor na nuvem, neste caso é preciso ler grandes arquivos de texto linha a linha, para identificar um bug ou realizar algum teste.

▼ Um novo prompt e executando scripts

▼ Um novo prompt e executando scripts

- As versões mais antigas do prompt, copiar e colar linhas de texto, não era possível, no caso as alternativas era utilizar o cmder
- Hj em dia para utilizar alternativas do CMD, utilizar-se o app terminal
- Os scripts são automações dos comandos do sistema, no caso do prompt/cmd utilização a extenção .bat no final do arquivo
- xcopy → Comando para copiar todos os diretórios e subs
- Script de bkp

```

cls
echo Deseja fazer o bkp?
echo bkp em andamento.....

cd c:\user\documents
mkdir backup
cd codigos
xcopy /e /y c:\user\user\codigos\ c:\users\user\backup

cd c:\user\documents\bkp
dir

```

▼ Script em lote

O *batch script* é um arquivo simples que agrupa uma serie de comandos, normalmente utilizado para automatizar tarefas cotidianas. Um arquivo *batch* (`.bat`) é um script que é executado sequencialmente pelo prompt (um comando depois do outro).

No desenvolvimento usamos scripts de diversas formas. Por exemplo, para limpar uma máquina antes de instalar arquivos novos, guiar o usuário pela instalação, preparar arquivos de configuração e importar dados dentro da aplicação ou de um banco de dados.

▼ Script em lote pt2

Os scripts podem ser muito mais poderosos. Até é possível programar dentro deles, criar menus, acessar a rede e muito mais. Aqui estamos focados para guiar você melhor na plataforma Alura. Normalmente, os recursos mais avançados são utilizados pelos administradores de infraestrutura.

E para não esquecer o mundo Linux: esses pequenos scripts de automação são muito mais frequentes no Linux e Unix. Lá se chamam de *shell script* mas tem o mesmo propósito, automatizar tarefas!

▼ Copiando subdiretórios

O argumento que deve ser passado para o `xcopy` é o `/E`. Este argumento indica ao `xcopy` que ele deve copiar todas as pastas e sub-pastas, incluindo as que estão vazias.

Então para copiar uma pasta e seus subdiretórios devemos fazer assim:

```
xcopy /E C:/Users/caelum/origem C:/Users/caelum/destino
```

▼ Ocultando os comandos

Se você reparar na execução do simples script abaixo:

```
cls  
echo Dia de hoje:  
echo %date%  
echo Hora atual:  
echo %time%
```

```
echo Dia de hoje:  
Dia de hoje:  
  
echo 01/02/2016  
01/02/2016  
  
echo Hora atual:  
Hora atual:  
  
echo 17:49:20,96  
17:49:20,96
```

É exibido duas vezes cada mensagem, uma vez exibindo o comando e outra na saída dele, pois quando executamos um script no prompt, ele exibe o nome do comando que está dentro do script e o seu resultado. Para desabilitar essa exibição dos comandos na hora de executar um script, devemos começar o script com o comando `@echo off`, ficando desta maneira:

```
@echo off  
cls  
echo Dia de hoje:  
echo %date%  
echo Hora atual:  
echo %time%
```

```
Dia de hoje:  
01/02/2016  
Hora atual:  
17:48:20,36
```

▼ Travando a execução

O comando `pause` espera por uma interação do usuário com o terminal, pausando o script em que ele foi chamado. Ele serve para o usuário ter tempo de confirmar se quer mesmo continuar executando o script atual ou se deseja abortar a sua execução.

Para dar continuidade a execução do script, basta apertar qualquer tecla, mas para abortá-lo é preciso utilizar o atalho `CTRL + C` ou fechar o terminal que chamou o script.

▼ Facilidades do cmder

- **Copiar e Colar**

- **Múltiplas abas**

- **Configurações extras**

- O cmder é um terminal extremamente personalizável, se você usar o atalho **Windows + ALT + P**, você abrirá as configurações dele, indo na parte de **Features**, você verá que é possível alterar quase tudo do terminal. Lá tem configuração das cores que ele utiliza, do nível de transparência, do tipo de cursor e muitas outras coisas que podem deixar o terminal do seu jeito.
- Ele também tem como exportar e importar as configurações, então caso um dia você precise formatar ou trocar de computador, você pode levar as suas configurações com você.

▼ Variáveis de ambiente e instalação do JDK

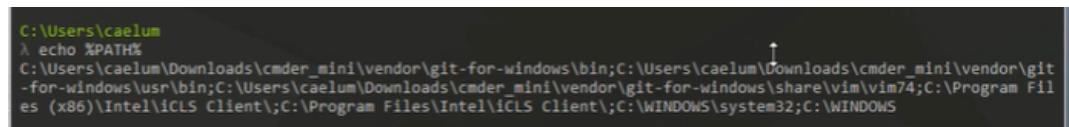
▼ Criando bkp do seu path

Então **antes de executar qualquer alteração do seu PATH**, execute o comando abaixo para salvar um backup:

```
echo %PATH% > backupPATH.txt
```

▼ Variáveis de ambiente e instalação do JDK

- O windows possui a variável de ambiente path que possui todos os caminhos que busca executáveis e scripts do windows



```
C:\Users\caelum
λ echo %PATH%
C:\Users\caelum\Downloads\cmder_mini\vendor\git-for-windows\bin;C:\Users\caelum\Downloads\cmder_mini\vendor\git-for-windows\usr\bin;C:\Users\caelum\Downloads\cmder_mini\vendor\git-for-windows\share\vim\vim74;C:\Program Files (x86)\Intel\iCLS Client;C:\Program Files\Intel\iCLS Client;C:\WINDOWS\system32;C:\WINDOWS
```

- A pasta bin possui os executaveis e scripts do sistema
- Através do comando **set PATH=%PATH%;<caminho do executavel>** irá add no path do sistema, e será executado de qualquer lugar em que esteja no terminal.
- **set** → Aponta todas as variáveis de ambiente do sistema
 - Ele acrescenta o caminho na variável temporariamente
 - **setx** → Acrescenta definitivamente o caminho, porém a syntax é diferente:
 - **setx PATH "%PATH%;<caminho do executavel/script>" /M** → No CMD do windows mesmo utilizamos este comando para acrescentar o executavel, é necessário abrir o prompt em modo administrador
- Seguindo com a instalação do JAVA JDK
 - Validar a arquitetura do processador, usar o instalador conforme a arquitetura do sistema



```
C:\Users\caelum
λ echo %processor_architecture%
AMD64
```

- Seguir com a instalação normalmente
- Após a instalação será necessário adicionar o executável javac.exe ao caminho do PATH do sistema

```
C:\Windows\system32>setx PATH "%PATH%;C:\Program Files\Common Files\Oracle\Java\javapath" /M
C:\Windows\system32>echo %PATH%
C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin;C:\Windows\system32;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH;C:\Program Files\dotnet\;C:\Program Files\PowerShell\7\;C:\Users\LucasSantos\AppData\Local\Microsoft\WindowsApps;C:\Users\LucasSantos\AppData\Local\Programs\Microsoft VS Code\bin

C:\Windows\system32>java --version
java 19.0.1 2022-10-18
Java(TM) SE Runtime Environment (build 19.0.1+10-21)
Java HotSpot(TM) 64-Bit Server VM (build 19.0.1+10-21, mixed mode, sharing)

C:\Windows\system32>javac --version
javac 19.0.1

C:\Windows\system32>```

```

- É recomendado criar o caminho JAVA_HOME no PATH
- setx JAVA_HOME "C:\Program Files\Java\jdk-19" /M

```
C:\Users\LucasSantos>echo %JAVA_HOME%
C:\Program Files\Java\jdk-19

C:\Users\LucasSantos>
```

▼ Variáveis de ambiente

O comando `set` mostra todas as variáveis na linha de comando

Como vimos o set também serve para alterar uma variável existente ou criar um nova

Lembrando que esta alteração só dura enquanto o terminal estiver aberto! (A mesmo que seja através do `setx`)

▼ Variáveis de ambiente pt2

Para acessar a variável devemos usar `%NOME_DA_VAR%`, no nosso caso: `echo %EAD%`

É muito comum construir um caminho a partir de variáveis já existentes. Por exemplo, já existe uma variável com o nome `USERPROFILE`

que guarda o caminho para pasta *home*
do usuário. No meu computador é:

```
echo %USERPROFILE%
C:\Users\caelum
```

O *home* do usuário é `C:\Users\caelum`. Sabendo dessa variável podemos navegar para nossa pasta de código usando: `cd %USERPROFILE%/codigo`

▼ Diferença entre set e setx

A diferença é que o `setx` altera as variáveis permanentemente, enquanto o `set` serve apenas enquanto o terminal está aberto. Lembrando que devemos sempre o usar o `setx` com a flag `/M` se quisermos que a alteração seja permanente.

Outra diferença é que com o comando `set` podemos imprimir todas as variáveis de ambiente.

▼ Uma variável para sempre

Para configurar uma variável de ambiente permanente, devemos usar o comando `setx`.

O comando `setx` deve ser usado sempre no prompt de comando **original** do Windows, e sempre em modo de **administrador**.

Seu uso correto fica assim:

```
setx PASTA_CODIGO "C:\Users\caelum\codigos" /M
```

O `setx` não possui o sinal de igual, e o conteúdo da variável deve ficar entre aspas. Não podemos esquecer também da importante flag `/M`, que indica que estamos criando uma variável do sistema.

▼ Testando seu conhecimento

O comando `setx` necessita da flag `/M` para alterar as variáveis do sistema, então a primeira afirmativa é **verdadeira**.

A variável de ambiente que contém a arquitetura do sistema é a `PROCESSOR_ARCHITECTURE` e não a `SYSTEM_ARCHITECTURE`, então a segunda afirmativa é **falsa**.

O comando que exibe todas as variáveis de ambiente quando executado sozinho é o `set` e não o `setx`, então a terceira afirmativa é **falsa**.

E por último, se você testou no seu Cmder, um outro jeito de exibir uma variável de ambiente é utilizando o comando `set` e o nome da variável em seguida, como no caso do `set PATH`, logo a quarta afirmativa é **verdadeira**.

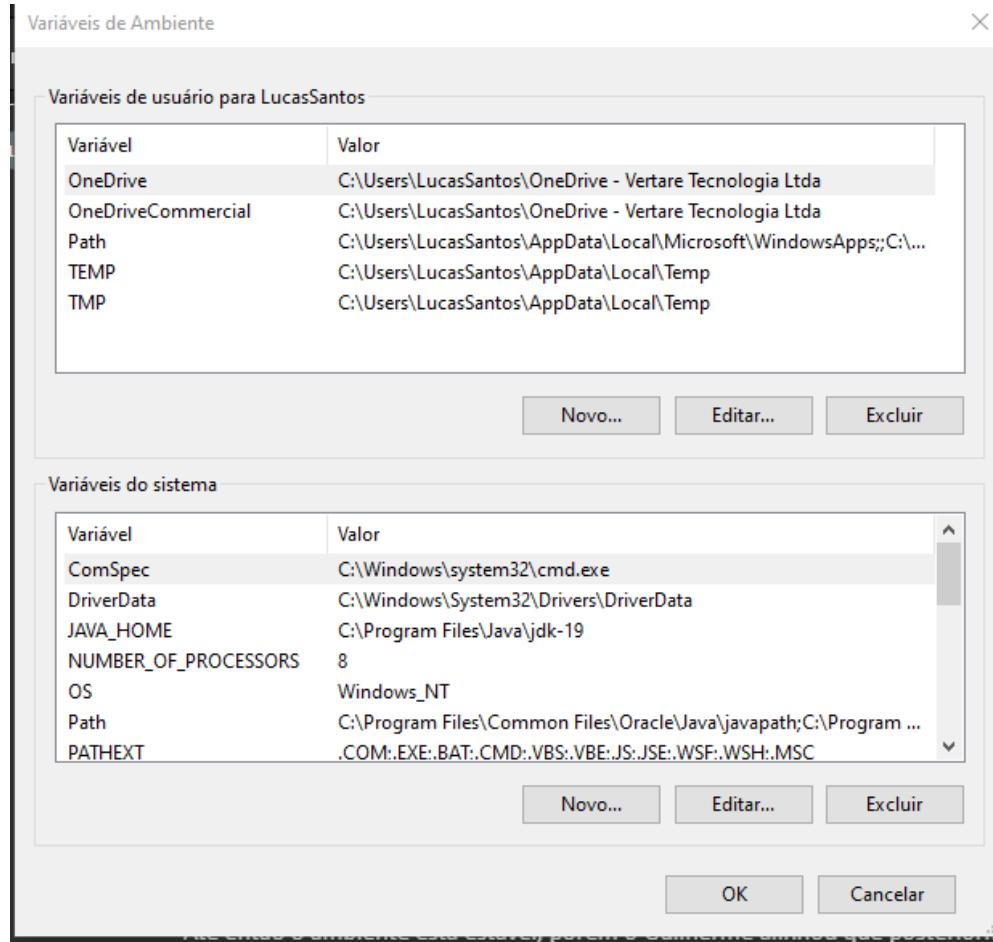
▼ Alterando o Path pela interface

Podemos alterar as variáveis de ambiente (como o PATH) pela interface gráfica do Windows.

Vou mostrar como:

- Clique no ícone Pesquisar e digite *Painel de Controle*
- Clique em -> Painel de Controle -> Sistema de Segurança -> Sistema-> Configurações Avançadas
- Na aba *Avançado das Propriedades do sistema* clique em *Variáveis de Ambiente*,
- Em *Variáveis do Sistema* localize `PATH` e clique nele.
- Na janela *Editar* você pode modificar o PATH adicionando a localização da classe para o valor de PATH.

Para ver as alterações feitas no PATH é preciso reabrir o mesmo (não precisa reiniciar).



▼ Dev Java

Você pretende desenvolver usando a plataforma/linguagem Java?

Se você pretende desenvolver com Java esta aula é para você, e mesmo não querendo usar Java ainda assim faz sentido assistir e praticar os exercícios desse capítulo pois configuraremos uma vez todo ambiente na raça (sem nenhuma ferramenta auxiliar) e este conhecimento também ajuda para outras plataformas de desenvolvimento, como Node, Python, C e outras diversas ferramentas!

▼ Arquitetura do processador

Vimos no video que há duas versões do JDK (*Java Development Kit*), para 32bit (`x86`) e 64bit.

Usamos a variável de ambiente `PROCESSOR_ARCHITECTURE` para descobrir a arquitetura do sistema:

```
echo %PROCESSOR_ARCHITECTURE%
```

```
C:\Users\caelum
λ echo %PROCESSOR_ARCHITECTURE%
x86

C:\Users\caelum
λ |
```

No entanto, existe uma alternativa na linha de comando para descobrir a arquitetura do processador. Tente executar no prompt do cmder:

```
wmic OS get OSArchitecture
```

E a saída é até um pouco mais bonita:

```
C:\Users\caelum
λ echo %PROCESSOR_ARCHITECTURE%
x86

C:\Users\caelum
λ wmic OS get OSArchitecture
OSArchitecture
32-bit

C:\Users\caelum
λ |
```

wmic é um comando que dá acesso ao *Windows Management Instrumentation*. O **wmic** é mais importante para administradores de infraestrutura e possui muito mais funções. Para gente basta saber se é 32bit ou 64bit!

▼ Arquitetura pela interface gráfica

Acesse o *Painel de Controle* (ou *Configurações* no Windows 10) e clique no *Sistema* para *Ver o nome do computador*. Depois clique no *Sobre*. Nessa tela também encontramos a informação sobre x86 (32bit) ou x64 (64bit).

Dica: Se seu teclado tiver a tecla Pause, pode usar o atalho Windows + Pause.

▼ Dir JAVA

É o **JAVA_HOME**!

Algumas ferramentas precisam dessa variável de ambiente para encontrar as bibliotecas do Java. Alguns exemplos de ferramentas são: ANT, Maven, Gradle ou Tomcat. Não se preocupe se você não conhece nenhuma dessas ferramentas. Para isso há outros treinamentos nas trilhas Java.

Você pode verificar a existência da variável pelo comando: `echo %JAVA_HOME%`

Ou simplesmente imprimir todas as variáveis pelo comando `set`.

▼ Verificando instalação JAVA

Para verificarmos a versão do Java e de seu compilador, devemos utilizar os comandos `java -version` e `javac -version`, respectivamente.

Vale lembrar também que devemos verificar se a variável de ambiente **JAVA_HOME** está configurada, já que muitas ferramentas do Java dependem dela. Fazemos isso através do comando:

```
echo %JAVA_HOME%
```

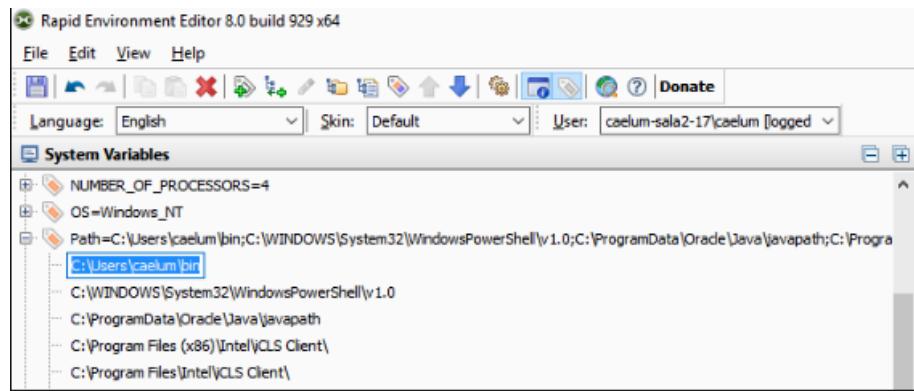
▼ Configurando a JAVA_HOME

Como queremos que a variável `JAVA_HOME` seja permanente, devemos usar o comando `setx`. E lembrando que o comando `setx` não utiliza o sinal de `=`, então a resposta correta seria:

```
setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_65" /M
```

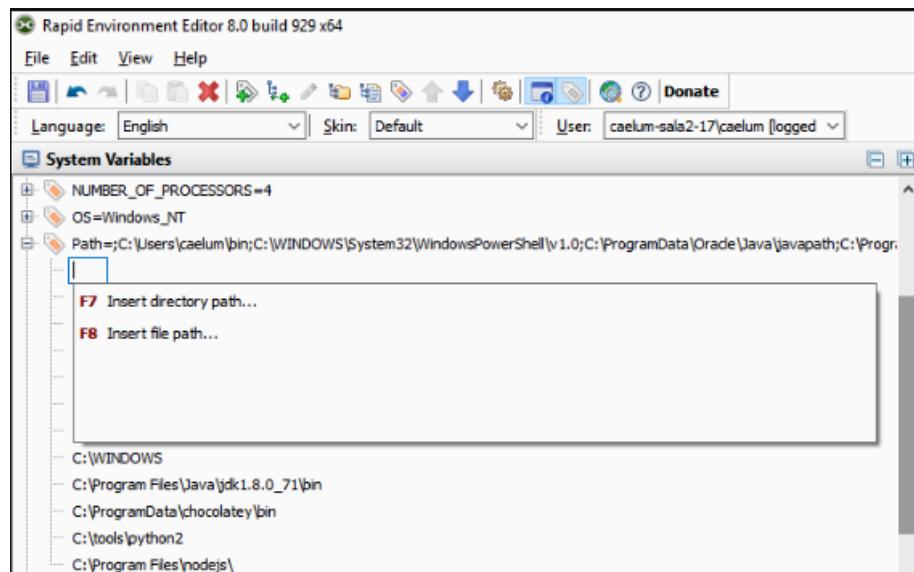
▼ Ferramenta para variáveis de ambiente

- No Windows nós temos diversas variáveis de ambiente, que nos fornecem várias informações diferentes sobre o nosso sistema. Vimos que para alterá-las, podemos utilizar o prompt de comando, mas agora quero lhe mostrar um pequeno programa que nos permite alterá-las de um modo um pouco mais visual. Este programa é RapidEE
- Ele é um editor de variáveis de ambiente, que usa a interface gráfica para exibir tanto as **váriaveis ambientes do sistema** (lado esquerdo), como as **váriaveis ambientes do usuário** (lado direito).
- Para fazer sua instalação, basta acessar o site <http://www.rapidee.com/en/download> e escolher a opção `Download RapidEE_setup.exe`.
- A instalação é bastante simples, bastando apenas seguir o instalador.
- Com ele instalado, podemos fazer as operações como editar, modificar e criar novas variáveis através de sua interface.
- **Criando novas variáveis**
 - Para adicionar uma nova variável, basta clicar com o botão direito e selecionar *Add Variable*. Aparecerá uma caixinha com o nome da variável que você deseja criar, e a opção de selecionar o conteúdo como uma *String* normal ou *Expanded String*. A diferença é que devemos usar *Expanded String* quando queremos usar uma variável que refencie outra, por exemplo, caso queiramos criar uma variável chamada **PASTA_TEMP**, que tenha o valor **%TEMP%**, referenciado o valor da variável TEMP.
- **Editando variáveis**
 - Para editar o valor de alguma variável, basta expandir seu conteúdo clicando no simbolo de `+` ao lado do nome dela, e dar um duplo clique no valor que desejámos alterar.



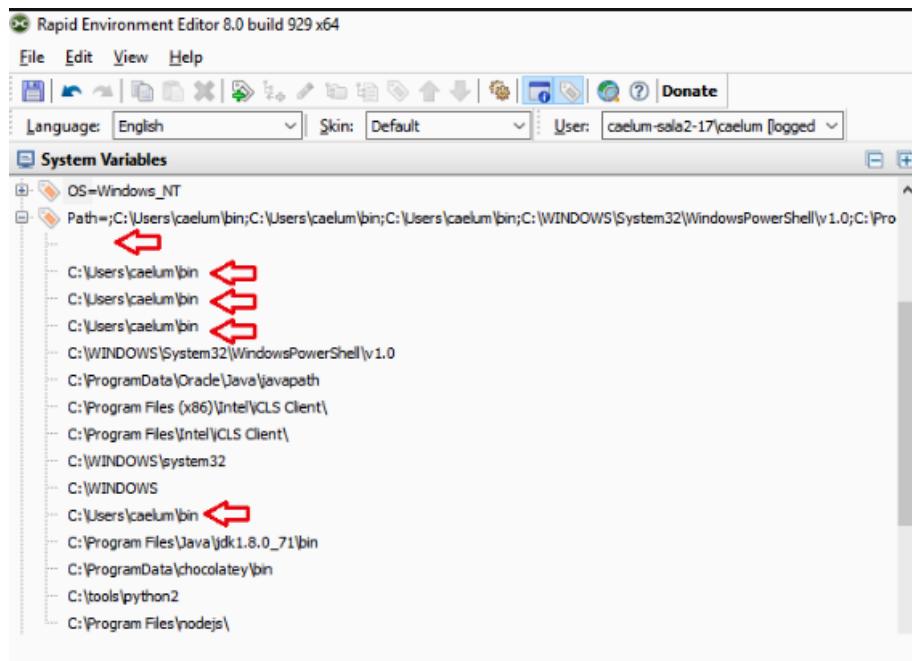
- **Adicionando novos valores**

- Para adicionar novos valores, clicamos com o botão direito na variável que queremos alterar e selecionamos a opção *Add value....*. Podemos escrever o caminho na mão, como fazímos no prompt, ou selecionar entre as opções *Insert directory path...* e *Insert file path...*, para inserir uma pasta ou um arquivo, respectivamente, através do sistema de janelas do Windows.

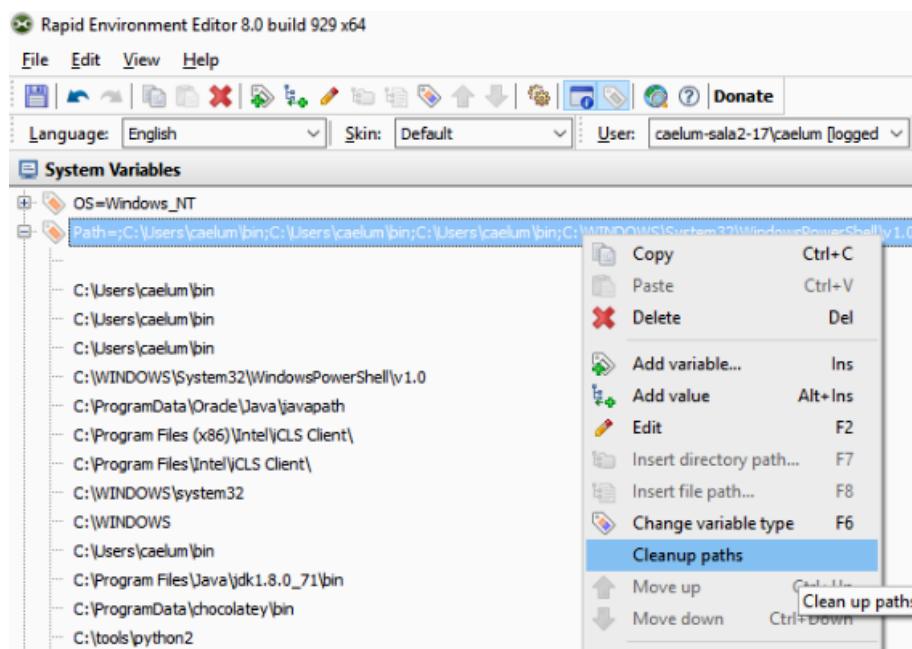


- **Removendo duplicatas e variáveis em branco**

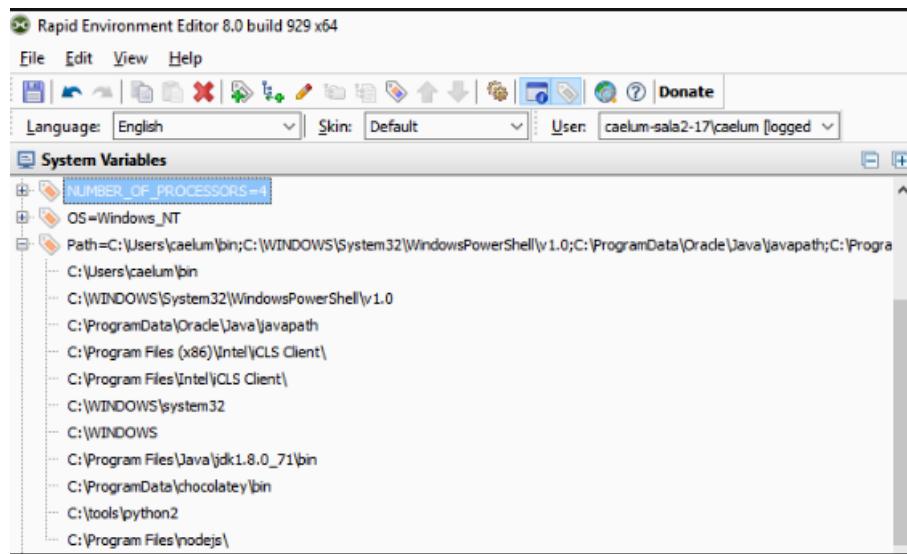
- Quando estamos alterando o PATH, podemos cometer algum erro , como inserir duas vezes a mesma variável ou inserir uma variável em branco, e isto é um pouco difícil de detectar quando estamos na linha de comando, pois se o PATH for muito grande, pode ficar difícil de notar algum descuido nosso.



- O RapidEE tem uma função que corrige isto automaticamente para gente, é a *Cleanup Paths*, que remove valores repetidos e valores em branco. Para usá-la basta clicar com o botão direito em cima de uma variável e selecioná-la, veja só:

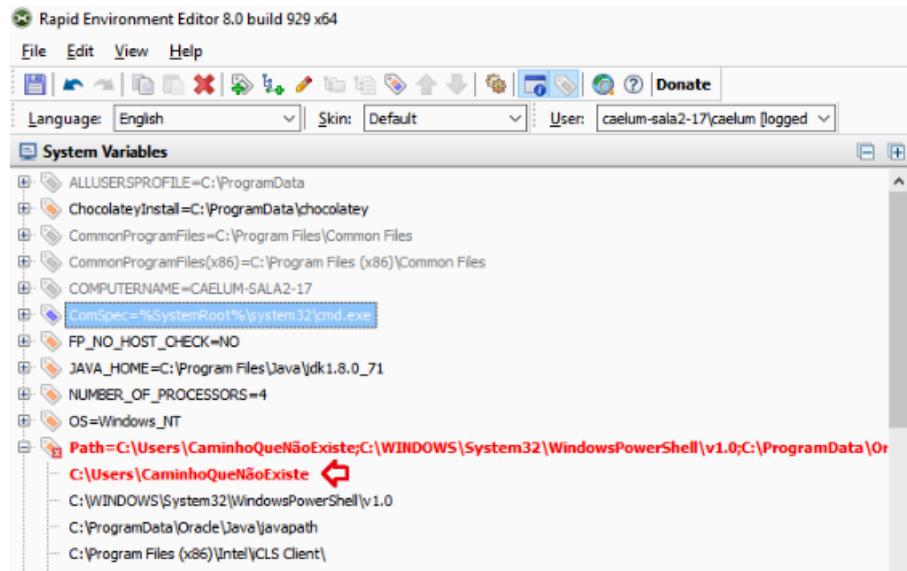


- Repare que na imagem acima, temos a pasta `C:\Users\caelum\bin` adicionada diversas vezes, e inclusive temos um valor em branco. Vamos corrigir isto com o *Cleanup Paths*:



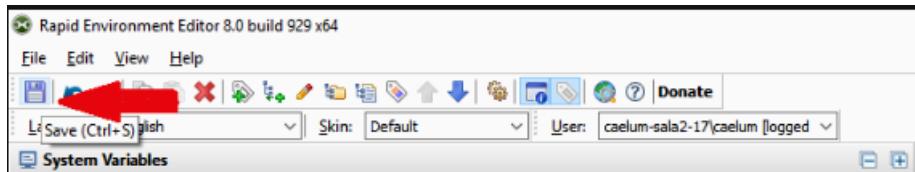
- **Detectando caminhos inválidos.**

- Um outro erro bastante comum quando estamos instalando programas ou alterando a variável **PATH** é que acabamos criando certos caminhos inválidos. O rapidEE detecta isto para a gente e coloca esses caminhos em vermelho:



- **Salve suas modificações!**

- Uma coisa importante ao utilizar a ferramente rapidEE é lembrar de salvar as alterações antes de fechá-la, clicando no pequeno ícone de salvar da barra de ferramentas dele.



▼ Gerenciando pacotes com chocolatey

Instalar via cmd →

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

▼ Gerenciando pacotes com chocolatey

- Assim como o linux possui o apt-get, no windows temos o chocolatey

Chocolatey - The package manager for Windows

Chocolatey is software management automation for Windows that wraps installers, executables, zips, and scripts into compiled packages.

Chocolatey integrates w/SCCM, Puppet, Chef, etc. Chocolatey is trusted

<https://chocolatey.org/>



- Para instalar através do CMD :

Chocolatey Software Docs | Setup / Install

Windows 7+ / Windows Server 2003+ PowerShell v2+ (Not PowerShell Core yet though)(minimum is v3 for install from this website due to TLS 1.2 requirement) .NET Framework 4+ (the installation will attempt to

<https://docs.chocolatey.org/en-us/choco/setup>



- Para validar a instalação utilize `choco`
- Para instalar algum pacote utilize `choco install 'nome do programa'`
- O chocolatey instala desde programas de desenvolvimento, até programas comuns como google chrome

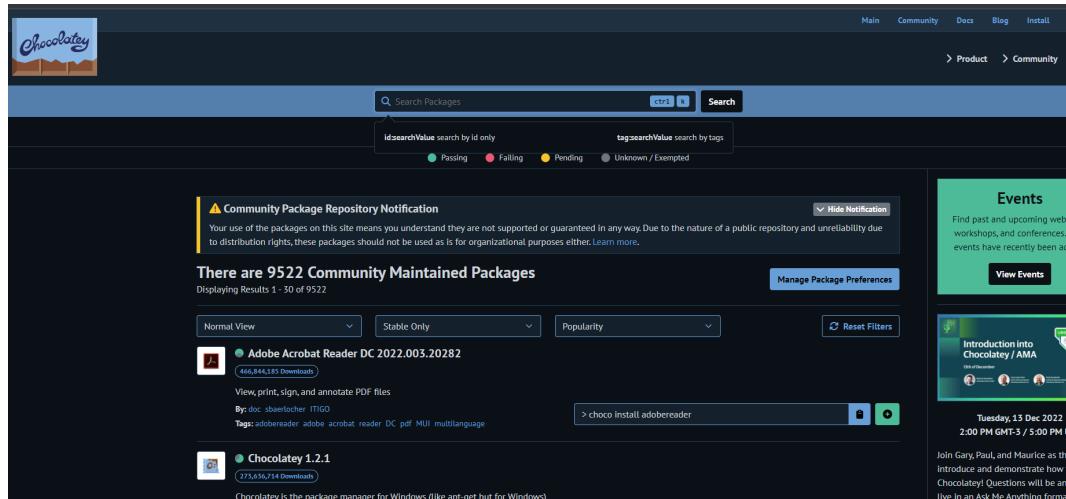
Packages

Chocolatey is software management automation for Windows that wraps installers, executables, zips, and scripts into compiled packages.

Chocolatey integrates w/SCCM, Puppet, Chef, etc. Chocolatey is trusted

<https://community.chocolatey.org/packages>





▼ Instalando um pacote com chocolatey

Apesar da ferramenta se chamar chocolatey, para usá-la na linha de comando devemos usar o comando **choco**, que é até bem mais fácil de digitar. Então, para fazer uma instalação usando o chocolatey, devemos utilizar o:

```
choco install <nome_do_pacote>
```

▼ Flag para agilizar

Quando instalamos um pacote com o `choco install`, ele pede algumas confirmações, que devemos aceitar para poder confirmar que queremos realizar a instalação. A flag `-y` faz com que essas confirmações sejam aceitas automaticamente, fazendo com que não tenhamos que nos preocupar em verificar a instalação e aceitar repetidas vezes as confirmações.

▼ Requisitos das instalações

Quando queremos instalar algo com o Chocolatey o único cuidado que devemos ter é **executar o prompt em modo de administrador**, pois como estamos instalando pacotes são necessárias certas permissões que só o prompt em modo de administrador possui.

Lembrando que os comandos do Chocolatey funcionam tanto no Cmder quanto no prompt de comando normal do Windows, é questão de gosto! E a flag `-y` é opcional no comando `choco install nome-do-pacote`, só que sem ela vamos ter que aceitar as perguntas que o Chocolatey faz manualmente.

▼ Como remover pacotes?

Para desinstalar um pacote podemos usar o comando `choco uninstall nome-do-pacote`, no nosso caso ficando:

```
choco uninstall python2
```

A remoção do pacote é tão rápida quanto a instalação :)

E caso você não se lembre todos os pacotes que tem instalados em seu computador, o comando `choco list -l` exibe-os para você!

▼ Choco list

- Vimos no treinamento que podemos verificar os pacotes disponíveis baixados através do site do chocolatey, porém também é possível fazer isto pela linha de comando! O comando `choco list` lista todos os pacotes que podem ser instalados pelo chocolatey mas antes de executá-lo tome cuidado por quê o chocolatey possui milhares de pacotes e seu computador vai passar um bom tempo listando-os! Para executar uma busca por pacotes mais específica, podemos utilizar o comando `choco list` com um nome do pacote que queremos buscar, por exemplo:

- `choco list git`

- **Pacotes locais**

- Uma outra opção de uso do `choco list` é para listar os pacotes locais que foram instalados pelo chocolatey, para isso basta usarmos a flag `-l` deste modo:
 - `choco list -l` Ele exibe uma lista de todos os pacotes do seu computador! Muito útil quando queremos saber quais pacotes e suas versões que possuímos instalados.

- **choco search ?**

- Você pode acabar encontrando na internet pessoas utilizando o `choco search` para fazer buscas, de um jeito muito semelhante ao que foi ensinado aqui, e se perguntando qual é a diferença. A resposta é: Nenhuma! O `choco search` é um comando análogo ao `choco list` e ambos fazem a mesma coisa debaixo dos panos. Eles são intercambiáveis entre si e tudo que você pode passar como argumento em um, pode usar no outro também! Então não se preocupe, os dois fazem a mesma coisa :)

Podemos usar o comando `choco list` para exibir tanto os pacotes que possuímos instalados localmente, quanto os disponíveis para serem instalados. Lembrando sempre de colocar o nome do pacote que você quer buscar, quando for buscar por um pacote que quer instalar:

```
choco list <nome_do_pacote>
```

Para listar os pacotes localmente usamos a flag `-l`, por exemplo executando `choco list -l` no meu computador trouxe o resultado:

```
chocolatey 0.9.9.11
git 2.6.4
git.install 2.6.4
jdk7 7.0.79.1
jdk8 8.0.66
nodejs 5.3.0
nodejs.install 4.2.2
ruby 2.1.6
```

▼ Gerenciador OneGet

O mundo Windows demorou muito para usar um gerenciador de pacote mas hoje em dia até existe um gerenciador nativo. Nativo significa que vem configurado com o Windows, mas apenas com Windows 10. O Gerenciador se chama **OneGet** e deve ser utilizado no PowerShell do Windows.

Mais informações sobre o OneGet na página do projeto:

<https://github.com/OneGet/oneget/wiki/cmdlets>

▼ Outros gerenciadores

- Assim como existe o chocolatey para Windows, no mundo Linux nós temos o apt-get. O apt-get também é um gerenciador de pacotes, e inclusive foi ele que inspirou a criação do chocolatey. Ele já vem por padrão nas distribuições mais comuns do Linux, como o Ubuntu e o Mint e do mesmo modo do chocolatey, facilita muito nossa vida quando queremos instalar programas no Linux.
- Já quando desenvolvemos aplicações, por exemplo usando Java, é muito comum usarmos código (dependências) de outros desenvolvedores no nosso código para ajudar em alguma tarefa. No mundo Java possuímos algumas ferramentas que nos ajudam a baixar e a lidar com essas dependências como o *maven*, *ivy* e o *gradle*. Elas agilizam o processo de desenvolvimento, já que, com um pequeno código, qualquer desenvolvedor que baixar nossa aplicação pode baixar as dependências do mesmo e ter o projeto funcionando rapidamente.
- Como desenvolvedor, é bom ter conhecimento das diversas ferramentas que existem por aí para facilitar nossa vida. Os gerenciadores de dependência são apenas um tipo delas, e mesmo não utilizando todas elas no dia a dia, é uma boa prática estar sempre atualizado em relação as tecnologias dos sistemas operacionais e linguagens mais comuns do mercado.

▼ Comandos do Linux bash no windows

▼ Comandos Linux bash no windows

- linux bash é o padrão de desenvolvedores
- cmder full já possui os comandos do linux para o windows
- Porém há tbm o wsl, CLI do linux no windows
- ~ → pasta home
- dir → Lista windows | ls → lista linux
- cls → limpa terminal windows | clear → limpa terminal Linux
- copy → Copia no windows | cp → copia no linux
- move → move no windows | mv → move no linux
- del → remove no windows | rm → remove no linux
- type → mostra conteudo do arquivo windows | cat → mostra conteudo do arquivo Linux
- grep → somente linux → busca informações → | - pipe → entre comandos ajuda a filtrar informações com o grep
- find → lista todos os diretorios abaixo da atual → com o pipe + grep, encontra palavras e informações e etc.

▼ Win e Unix

O comando `ls` não funciona no prompt padrão do Windows! O `ls` é a abreviação para *listar* e mostra tudo o conteúdo de uma pasta.

Por exemplo, podemos listar e ordenar pela data de modificação:

```
ls -lt
```

Podemos mostrar o tamanho dos arquivos e kB e MB:

```
ls -lth
```

E ordenar pelo tamanho:

```
ls -lsh
```

São muitas as opções, e você pode ver todas elas usando:

```
ls --help
```

▼ Win e Unix pt2

O comando `type`

é um dos comandos clássicos do prompt do Windows, ou seja funciona sem problemas nenhum e mostra o conteúdo de um arquivo:

```
type arquivo.txt
```

Todos os outros comandos só vão funcionar no *cmd*.

▼ Pesquisa dentro de um texto

O comando `grep` procura na entrada de dados, mostrando as linhas onde foi encontrado a palavra em questão, por exemplo:

```
ls -al | grep Documentos
```

Nesse exemplo combinamos `ls -al` com `grep`. Usamos o `|` para juntar os comando. O sinal `|` também é chamado de *pipe*.

`ls -al` lista todo o conteúdo da pasta atual e passa isso como entrada para o comando `grep`. Em cada linha dessa entrada o `grep` procura a palavra `Documentos`.

Podemos combinar o `grep` com outros comando como o `cat`:

```
ls -al > arquivo.txt  
cat arquivo.txt | grep Documentos
```

▼ Exibindo o conteúdo de um arquivo

O comando `cat` serve para exibir um arquivo no terminal. Ele se parece com o comando `type` do Windows, então ao fazer:

```
cat arquivo.txt
```

O conteúdo do `arquivo.txt` é exibido na tela do terminal.

▼ Arquivos ocultos

Para listar todos os arquivos do diretório atual, devemos utilizar a flag `-al` ("a" de all e "l" de list) , assim todos os arquivos serão exibidos em formato de lista.

Por curiosidade:

A flag `-hl` transforma as unidades de tamanho em um formato mais amigável, por exemplo 4K em vez de 4096.

A flag `-rl` lista os arquivos em ordem reversa a normal.

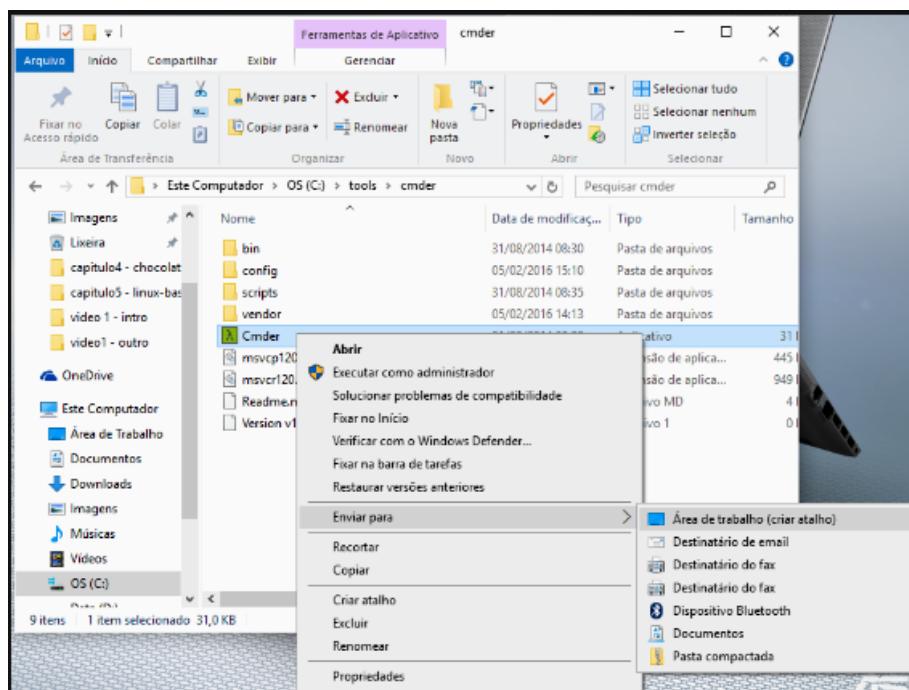
A flag `-sl` lista os arquivos por ordem de tamanho.

Usando a flag `--help` do comando `ls` você pode ver várias outras opções de como listar seus arquivos!

▼ Atalho para o seu cmder

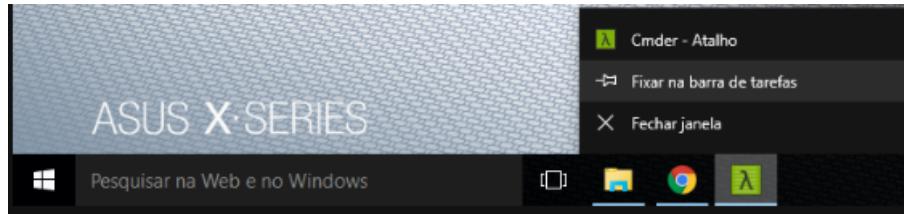
Como o cmder é um prompt muito versátil, nada mais justo do que ter um jeito fácil de acessá-lo, não é mesmo? Para isso, vamos criar um atalho para a versão full do cmder que instalamos neste capítulo. Assim evitamos aquele processo de abrir o prompt de comando tradicional para chamar o cmder.

Primeiro, vá na pasta `C:\tools\cmder`. Lá você deve encontrar a versão full do cmder, que foi instalado pelo chocolatey. Para criar um atalho dela, basta clicar com o botão direito do mouse no ícone do cmder, e depois ir em *Enviar para*, selecionando *Área de Trabalho (criar atalho)* em seguida.



Outra opção é deixá-lo fixado na barra de tarefas. Quando você estiver executando o cmder, clique com o botão direito no ícone dele que fica na barra do menu iniciar e clique em *Fixar na barra de tarefas*

, assim, mesmo que você feche-o , ele estará logo ali embaixo, sendo bem rápido de acessar.



▼ WSL

Install WSL

Developers can access the power of both Windows and Linux at the same time on a Windows machine. The Windows Subsystem for Linux (WSL) lets developers install a Linux distribution (such as Ubuntu, OpenSUSE, Kali, Debian, Arch Linux,

 <https://learn.microsoft.com/en-us/windows/wsl/install>



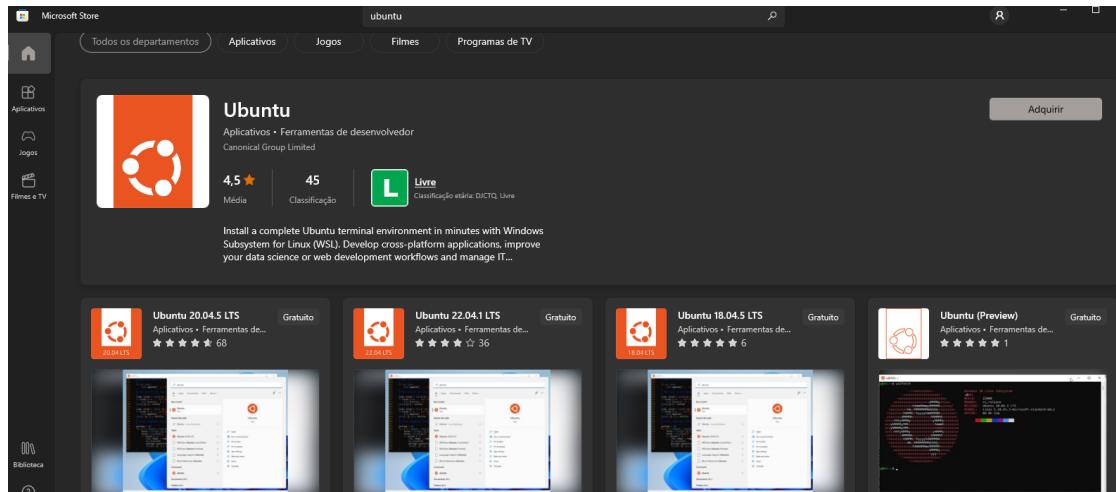
Etapas de instalação manual para versões mais antigas do WSL

Para simplificar, geralmente recomendamos usar o wsl --install para instalar o Subsistema do Windows para Linux, mas se você estiver executando um build mais antigo do Windows, talvez não tenha suporte para esse comando. Incluímos

 <https://learn.microsoft.com/pt-br/windows/wsl/install-manual>



- Roda a CLI do linux, no caso o kernel de uma determinada distro, dentro do proprio linux
- Etapas da instalação
 - Habilitar o subsistema do windows para linux - dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
 - Habilitar recurso VM → dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
 - Reiniciar
 - Instalar via powershell → wsl --install
 - Ou via store



▼ Linux Onboarding - Usando a CLI de uma forma rápida e prática

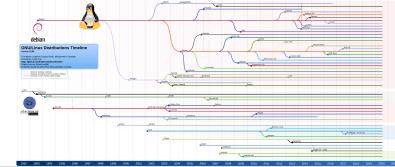
▼ Por que Linux?

▼ Pq Linux, e as principais distribuições

- O backend de linux é muito grande, e é necessário para desenvolvimento ou qualquer outra área em computação.
- Debian → Base muitos servidores originaram dele → Ubuntu
- Slackware → Muito utilizada e muito personalizável → Red Hat

<https://distrowatch.com/>

<https://i.redd.it/aygzaivcbmd51.png>



▼ Preparando o setup

- 1- Usar um virtualizador permite instalar diversos sistemas dentro do próprio SO
- 2 - Utilizar uma plataforma cloud também é viável
- 3 - Utilizar um computador com o Linux mesmo
- Virtualizador
 - Virtualbox

Oracle VM VirtualBox

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for

 <https://www.virtualbox.org/>



- Baixar a ISO do ubuntu

Get Ubuntu | Download | Ubuntu

Download Ubuntu desktop and replace your current operating system whether it's Windows or macOS or run Ubuntu alongside it. Do you want to upgrade? Follow our simple guide Whether

 <https://ubuntu.com/download#download>



- Versões mais recomendadas são as LTS → long-term support → tempo de manutenção muito maior
- LTS é a principal para servidores
- No ubuntu é possível baixar a versão desktop e a server(Full CLI)

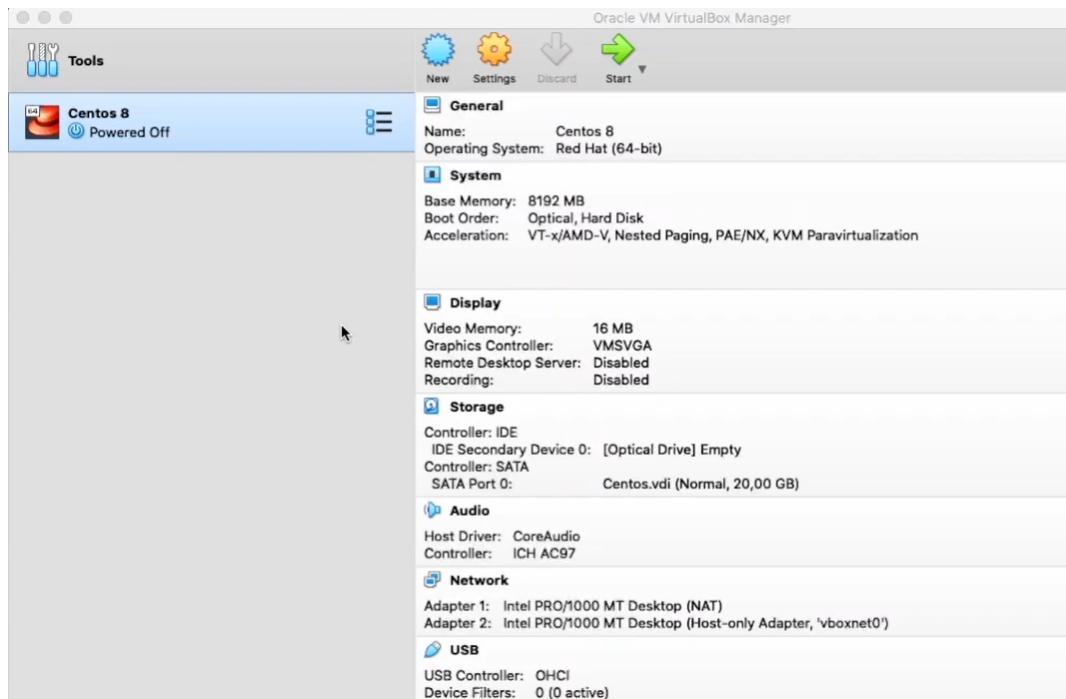
▼ Tipos de instalação

Para executarmos o SO Linux é necessário fazer uma instalação do tipo:

Diretamente no servidor, como a maior parte dos SOs temos a opção de fazer a instalação diretamente no servidor ou mesmo no desktop.

▼ Instalando o Linux

- Não tem segredo na hora da instalação



- Interfaces de rede

- NAT
 - Envia o pacote para a internet
 - Não é possível gerenciar via SSH
- Host only adapter
 - Cria uma rede interna que associa um IP na rede para o host e para a VM
 - Assim com a criação e associação de IPs, é possível realizar SSH
- Bridge adapter
 - Pega a interface física e cria uma interface logica na vm, sem precisar a rede
 - Na interface logica, ele pega o IP da rede, assim fica possível realizar o SSH

▼ Instalação e acesso remoto

▼ Acessando remotamente

- SSH
 - Primeiro passo é pegar o IP da VM que foi criada no virtual box
 - Como a VM está na rede podemos usar o SSH + nome do usuario@IP da VM
 - Será criado as keys automaticamente, e o acesso será possível
- APT → Gerenciador de pacotes do debian
- Update → Atualizar a base local

- SUDO → Tarefas administrativas, ou passa pra root e executa a atividade administrativa

▼ Acesso remoto

Por default utilizamos qual protocolo para acessar remotamente os sistemas Linux:

SSH

Por padrão utilizamos o ssh que é um protocolo seguro para a gerência dos ambientes.

▼ Utilizando uma estância na nuvem

- Após criar a instância dentro do provedor
- Habilitamos o SSH no firewall
- Baixamos as chaves SSH, salvamos e rodamos o comando de acesso a vm
- Ou podemos colocar nossas próprias chaves na vm

▼ Linux na cloud

Para utilizarmos o Linux em um ambiente de nuvem é necessário?

Ambas as opções, lembrando que as imagens customizadas já vem com algumas ferramentas e configs ajustadas para o provedor de nuvem. Por exemplo, na Amazon existe a imagem: Amazon Linux 2 AMI (HVM).

▼ Navegando no sistema

▼ Navegando no sistema PWD e LS

- Por padrão o ubuntu utiliza o bash que é o sistema de gerenciamento do SO, no caso CLI
- Por padrão a identificação é apresenta na tela do linux

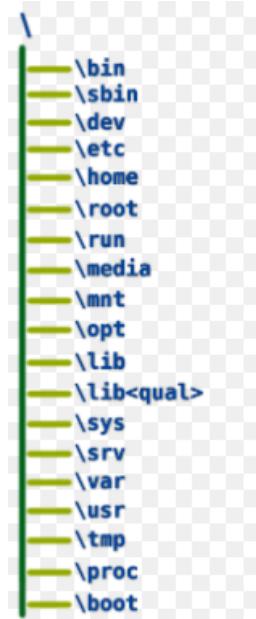


- pwd → Pasta atual
- / → essa é a barra do bash enquando que essa “\” é do linux
- ls → mostra oq tem no diretorio

- ls -a → todos os conteúdos e ocultos
- ls -al → mostra conteúdos e permissionamentos
- ll → atalho
- clear → limpa a tela
- <comando> -- help → Mostra a ajuda de determinado comando
- man + comando → Mostra o manual

▼ Hierarquia no filesystem - FHS

- ctrl + L → limpa a tela tbm
- cd → faz a navegação (só vai pro home)
- O sistema de diretórios no linux funciona hierarquicamente
 - Filesystem Hierarchy Standard → FHS



- Indo para \, temos acesso a todas essas pastas

▼ Atalhos para navegação

- Tab → autocompleta
- Setas → histórico de comandos
- cd - → faz um switch entre os diretórios, volta e vai pra dois diretórios diferentes
- ls -l → lista longa
- ls -la → Mostra todo o conteúdo detalhadamente
- cd . → volta um

- cd .. → volta pra raiz da pasta
- cd ~ → home

▼ Atalhos do CD

Estamos no diretório `/etc` e na sequência fomos para o `/var/log`. Qual comando devemos utilizar para voltarmos ao diretório `/etc`?

CD -, com o `cd -` retornamos ao diretório anterior.

▼ Arquivos e diretórios

▼ Criando diretórios com o MKDIR

- mkdir → Cria diretórios
- d → diretório
- mkdir -p dir/dir1/etc → Permite criar uma arvore de diretórios
 - -p → Não vai ter erros, cria a estrutura caso ela não exista!
- cd ../../.. → Vai voltando das pastas
- touch cria arquivos
- .arquivooupasta → sempre que houver um . antes significa q ele é oculto

▼ Criando diretórios

- O diretório e seu respectivo subdiretório `/projetos/eng` não existem. Para criá-los de uma forma mais eficiente devemos utilizar:

- `mkdir -p /projetos/eng`
 - com o `-p`
 - o mkdir cria a estrutura caso ela não exista!

▼ Removendo diretórios e arquivos - RMDIR e RM

- rmdir → remove diretórios
- rm → remove arquivos porém também serve para remover diretórios
- rm -r → está flag utiliza recursividade, então independente doq seja, é apagado
 - -rf → remove definitivamente
- Quando usamos “\” é possível criar arquivos e pastas com espaço

```
ricardo@ubuntu-server:~/labs/arqs_dirs$ mkdir diretorio\ 1
ricardo@ubuntu-server:~/labs/arqs dirs$ ls
1 2  Dir1  diretorio  'diretorio 1'
```

▼ Copiando arquivos com o CP

- cp → copia os arquivos
- cp -r * → copia diretórios com arquivos dentro

- * → é o coringa, sempre tudo e em diante
- cp -r dir/* + caminho que quer mandar → Copia de um e joga pra outro

▼ Utilizando o cp

- A partir do diretório `~/labs/` qual comando deves utilizar para copiar todo o conteúdo do `/etc` ?
 - `sudo cp -r /etc .` neste comando utilizamos o `.` para indicar o diretório correto e o `-r` para indicar a recursividade.

▼ Movendo e renomeando com o Mv

- mv → Move o arquivo
- mv + dir ou arq + local q quer enviar
- mv → serve também para renomear também
- para mover a pasta toda usamos `*`
- mv dir/* dir4 → Vai jogar a pasta dir em dir4
- history → Historico de comandos utilizados

▼ Manipulando arquivos e diretórios

▼ Globbing

- * → nada ou qualquer coisa
- ls <arquivo ou pasta>* → Lista a sequencia
- ls arq

```
ricardo@ubuntu-server:~/labs/glob$ ls
arq1 arq10 arq100 arq2 arq3
ricardo@ubuntu-server:~/labs/glob$ ls *
arq1 arq10 arq100 arq2 arq3
ricardo@ubuntu-server:~/labs/glob$ ls -a
. ..
arq1 arq10 arq100 arq2 arq3
ricardo@ubuntu-server:~/labs/glob$ touch tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls
arq1 arq10 arq2 arq3 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls arq*
arq1 arq10 arq2 arq3
ricardo@ubuntu-server:~/labs/glob$ touch arq
ricardo@ubuntu-server:~/labs/glob$ ls
arq arq1 arq10 arq100 arq2 arq3 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls arq*
arq arq1 arq10 arq100 arq2 arq3 ← Lista tudo continuando
ricardo@ubuntu-server:~/labs/glob$ ls arq1*
arq1 arq10 arq100 ← Lista as sequencias
ricardo@ubuntu-server:~/labs/glob$ ls arq1?
arq10 ← Lista arq com uma casa
```

- Usar ?? → vai aparecendo os caracteres
- ls *1* → Aparece tudo que tem 1

▼ Utilizando “coringas”

No cenário abaixo o usuário precisa copiar somente os arquivos de 01 á 05. Qual comando deverá ser utilizado?

Conteúdo do diretório:

arq01, arq02, arq03, arq04, arq05, arq10, arq20, arq30, arq40, arq50

`cp arq0? ~/projetos`, utilizamos o ? para substituir o caracter na posição específica

▼ Globbing pt2

- [1-5] → mostra os 5 primeiros itens do diretório

```
ricardo@ubuntu-server:~/labs/glob$ ls ****
arq1 arq2 arq3 arq5 arq9 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls ???[1-5]
arq1 arq2 arq3 arq5 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ touch arq50
ricardo@ubuntu-server:~/labs/glob$ ls ???[1-5]*
arq1 arq10 arq100 arq2 arq3 arq5 arq50 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls *[1-5]
arq1 arq2 arq3 arq5 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ touch arquivo1 arquivo2
ricardo@ubuntu-server:~/labs/glob$ ls *[1-5]
arq1 arq2 arq3 arq5 arquivo1 arquivo2 tmp1 tmp2
ricardo@ubuntu-server:~/labs/glob$ ls arq[1-5]
arq1 arq2 arq3 arq5
ricardo@ubuntu-server:~/labs/glob$
```

- Colocar a , fica somente os items com 1 e 5

```
ricardo@ubuntu-server:~/labs/glob$ ls arq[1-5]
arq1 arq2 arq3 arq5
ricardo@ubuntu-server:~/labs/glob$ ls arq[1,5]
arq1 arq5
```

- É possível usar range de letras

```
ricardo@ubuntu-server:~/labs/glob$ ls [A,a]rq[1-5]
Arq1 Arq2 arq1 arq2 arq3 arq5
ricardo@ubuntu-server:~/labs/glob$
```

▼ Utilizando “coringas” pt2

No cenário abaixo o usuário precisa listar somente os arquivos que contenham o prefixo arq e o numero 3.

Conteúdo do diretório:

arq01, arq02, arq03, arq04, arq05, arq10, arq20, arq30, arq40, arq50

```
ls arq*3*
```

O `*` pode ser utilizado para substituir qualquer caractere inclusive “nada”. Por isso o arq30 também é mostrado nesta saída.

▼ Linux Onboarding - Localizando arquivos e conteúdos

▼ Localizando conteúdos e trabalhando com as saídas

▼ Filtrando o conteúdo

- `.` → irá referir sempre ao diretório atual

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ ls
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ ls /etc/services
/etc/services
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ cp /etc/services .
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ ls
services
```

Estamos copiando o arquivo services para o dir atual.

- `cat` → mostra tudo oq está dentro do arquivo
- `grep` → faz buscas no sistema, no arquivo, na pasta
 - No exemplo ele está buscando uma informação dentro do arquivo services, no caso a informação que ele está buscando é http

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep http services
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml .
http      80/tcp      www          # WorldWideWeb HTTP
https     443/tcp     www          # http protocol over TLS/SSL
http-alt   8080/tcp    webcache    # WWW caching service
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ 
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep https services
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml .
https     443/tcp     www          # http protocol over TLS/SSL
ricardo@ubuntu-server:~/labs/filtrando_conteudo$
```

- `grep -i` → ignora maiúsculo e minúsculo
 - No ex ele está fazendo as duas buscas

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -i http services
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml .
http      80/tcp      www          # WorldWideWeb HTTP
https     443/tcp     www          # http protocol over TLS/SSL
http-alt   8080/tcp    webcache    # WWW caching service
hkp      11371/tcp    www          # OpenPGP HTTP Keyserver
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep http services
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml .
http      80/tcp      www          # WorldWideWeb HTTP
https     443/tcp     www          # http protocol over TLS/SSL
http-alt   8080/tcp    webcache    # WWW caching service
ricardo@ubuntu-server:~/labs/filtrando_conteudo$
```

- grep <informação> * → Busca em todos os arquivos

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep ricardo *
passwd:ricardo:x:1000:1000:ricardo merces:/home/ricardo:/bin/bash
nome do arquivo
```

- grep -l <informação> * → busca os arquivos que contem determinada informação

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -l ricardo *
passwd
```

- grep -L <informação> * → busca os arquivos que não contem determinada informação

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -L ricardo *
services
```

▼ Buscando conteúdo dentro dos arquivos

Para localizar a string `SSH` e todas as suas variações dentro do diretório `~/` devemos utilizar o seguinte comando/syntaxe: `grep -i ssh *`, a opção `-i` representa o “ignore-case”

▼ Utilizando a recursividade no grep

- Recursividade → Seguir de um ponto em diante

```
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ mkdir teste
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ cd teste
ricardo@ubuntu-server:~/labs/filtrando_conteudo/teste$ cp ../services services2
ricardo@ubuntu-server:~/labs/filtrando_conteudo/teste$ ls
services
ricardo@ubuntu-server:~/labs/filtrando_conteudo/teste$ cd ..
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep HTTP services
http      80/tcp      www          # WorldWideWeb HTTP
hkp      11371/tcp    www          # OpenPGP HTTP Keyserver
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep HTTP *
services:http   80/tcp      www          # WorldWideWeb HTTP
services:hkp   11371/tcp    www          # OpenPGP HTTP Keyserver
grep: teste: Is a directory
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -r HTTP services
http      80/tcp      www          # WorldWideWeb HTTP
hkp      11371/tcp    www          # OpenPGP HTTP Keyserver
ricardo@ubuntu-server:~/labs/filtrando_conteudo$
```

- Neste exemplo criamos uma pasta chamada teste na qual copiamos o arquivo service para dentro da mesma
- Se rodarmos grep http service, será feito uma busca dentro do arquivo atual referente ao conteúdo, no exemplo buscamos http dentro de service
- Ao utilizarmos http *, irá fazer uma busca tanto no arquivo quanto na pasta, porém na pasta irá apontar como se ocorresse um erro “grep: teste: is a directory”
- portanto se usarmos -r será feita uma busca recursiva, nesse caso no ultimo comando ele buscou dentro do arquivo

```

ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -rl HTTP services
services
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -rl HTTP *
services
teste/services2
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ grep -r HTTP *
services: http 80/tcp www # WorldwideWeb HTTP
services: hkp 11371/tcp # OpenPGP HTTP Keyserver
teste/services2: http 80/tcp www # WorldwideWeb HTTP
teste/services2: hkp 11371/tcp # OpenPGP HTTP Keyserver
ricardo@ubuntu-server:~/labs/filtrando_conteudo$ 

```

- Porém neste segundo exemplo como podemos ver no segundo comando, ele utiliza `-rl` para validar quais arquivos possuem determinado conteúdo
- E quando rodamos `-r http *`, ele mostra o conteúdo referente aos dois arquivos, ou seja uma busca recursiva, de um ponto em diante
- `grep -r` → recursividade faz buscas dentro de arquivos e mostram os conteúdos
- `grep -rl` → mostra os arquivos que possuem determinado conteúdo
- `*` → irá apontar para toda a pasta

▼ Buscando conteúdos de forma recursiva

- Para localizar a string `http` e todas as suas variações a partir do diretório `/etc` devemos utilizar o seguinte comando/syntaxe:
 - `grep -r http *`
 - Com o `-r` ou `-R` incluímos os subdiretórios na busca.

▼ Formatando a saída da tela

- Podemos controlar o output da tela, ou seja as saídas
- `more` → irá paginar a tela com todas as informações do arquivo

```

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
[More--(75%)]

```

- Enter → vai paginando
- b → volta a pagina
- q → sai
- cat → mostra o conteudo no prompt

```

ricardo@ubuntu-server:~/labs/filtrando_conteudo$ cat passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin

```

- less → vai paginando porém mostrando de forma mais enxutada as informações

```

# UNIX specific services
#
exec      512/tcp
biff      512/udp      comsat
login     513/tcp
who       513/udp      whod
shell     514/tcp      cmd syslog    # no passwords used
syslog    514/udp
printer   515/tcp      spooler      # line printer spooler
talk      517/udp
ntalk     518/udp
route    520/udp      router routed # RIP
gdomap   538/tcp
gdomap   538/udp
uucp     540/tcp      uucpd       # uucp daemon
klogin    543/tcp
kshell   544/tcp      krcmd       # Kerberized 'rlogin' (v5)
dhcpv6-client 546/udp
dhcpv6-server 547/udp
afpovertcp 548/tcp
nntp     563/tcp      snntp       # NNTP over SSL
submission 587/tcp
ldaps    636/tcp
ldaps    636/udp

```

- tail → mostra as 10 ultimas linhas de um arquivo

```

ricardo@ubuntu-server:~/labs/filtrando_conteudo$ tail /etc/passwd
tss:x:106:111:TPM software stack,,,:/var/lib/tpm/:/bin/false
uidd:x:107:112:/:/run/uidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
landscape:x:109:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
sshd:x:112:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
ricardo:x:1000:1000:ricardo merces:/home/ricardo:/bin/bash

```

- Porém podemos fleguear para mostrar as quantidades de linhas específicas com -n
- head → mostra as 10 primeiras linhas de um arquivo

```

ricardo@ubuntu-server:~/labs/filtrando_conteudo$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin

```

▼ Procurando arquivos no sistema

▼ Procurando arquivos no sistema

- Localizadores de conteúdo → grep, cat, more, tail head
- ls → mostra conteúdo em um diretório
- find → procura arquivos a partir de um determinado diretório
 - find faz a recursão com base no parâmetros
 - -name *.<tipo de arquivo> → Flag para especificar o tipo de arquivo

- -maxdepth x → auxilia para redefinir as buscas, ou seja ele vai para uma quantidade especifica de pastas, para realizar a busca

▼ Onde estão os meus arquivos

- Para localizar o arquivo *teste.arq* em todo o FHS devemos utilizar:

- `sudo find / -name teste.arq`

- É importante especificar o `/` para que a procura seja feita a partir daí.

▼ Mais recursos do find - amin, atime, iname

- Realizar auditoria com o find

- `find /<dir>`

- -amin -5 → alterações na pasta dos ultimos 5 minutos

- -atime -2 → alterações na pasta dos ultimos 2 dias

- -size +3g -3g 3g → mostra os arquivos maiores que 3g, menores que 3g, e iguais a 3g

- -iname → ignora parte da letra

- ls -lh → mostra leitura númerica de forma humana 1024 → 1h

▼ Fazendo buscas pelo tamanho do arquivo

- Para localizar arquivos com 2GB ou maiores utilizamos as seguintes opções do find:

- `sudo find / -size +2G`

- Não esqueça de adicionar o `+` caso deseje procurar por arquivos de 2GB ou maiores.

▼ Redirecionamentos e pipe

▼ Redirecionando a saída padrão para um arquivo

- Redirecionamento

- `>` → é possível fazer redirecionamentos

```

ricardo@ubuntu-server:~/labs/redirecionamento$ cp /etc/services .
ricardo@ubuntu-server:~/labs/redirecionamento$ ls
services
ricardo@ubuntu-server:~/labs/redirecionamento$ grep ssh services
ssh          22/tcp      # SSH Remote Login Protocol
ricardo@ubuntu-server:~/labs/redirecionamento$ grep ssh services > listagem.txt
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt
ssh          22/tcp      # SSH Remote Login Protocol
ricardo@ubuntu-server:~/labs/redirecionamento$ 
  
```

Copiamos o arquivo para a pasta do projeto

Pesquisamos este conteúdo no arquivo

Jogamos o conteúdo em outro arquivo

- `>>` → Acrescenta a informação em arquivo já existente

```

ricardo@ubuntu-server:~/labs/redirecionamento$ grep ssh services > listagem.txt
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt
ssh          22/tcp                  # SSH Remote Login Protocol
ricardo@ubuntu-server:~/labs/redirecionamento$ grep 3389 services >> listagem.txt
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt
ssh          22/tcp                  # SSH Remote Login Protocol
ms-wbt-server 3389/tcp
ricardo@ubuntu-server:~/labs/redirecionamento$ 

```

- PIPE

- | → com o pipe é possível fazer redirecionamento de saídas com outros comandos
 - EX: Juntar o cat + grep

```

ricardo@ubuntu-server:~/labs/redirecionamento$ cat /etc/passwd | grep ricardo
ricardo:x:1000:1000:ricardo merces:/home/ricardo:/bin/bash

```

▼ Redirecionando a saída

- Para acrescentarmos um conteúdo a uma arquivo utilizamos:

- ls -l >> listagem.txt
 - Exatamente, o >> acrescenta o conteúdo.

▼ Combinando comando com o pipe

- sort → Comando para reordenar saídas, otimo de ser usado com o cat ou grep

```

cat: listagem: No such file or directory
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt
ssh          22/tcp                  # SSH Remote Login Protocol
ms-wbt-server 3389/tcp
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt | sort ↗
ms-wbt-server 3389/tcp ←
ssh          22/tcp                  # SSH Remote Login Protocol
ricardo@ubuntu-server:~/labs/redirecionamento$ 

```

- É possível usar o sort + > para alterar arquivos

```

ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt
ssh          22/tcp                  # SSH Remote Login Protocol
ms-wbt-server 3389/tcp
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt | sort
ms-wbt-server 3389/tcp
ssh          22/tcp                  # SSH Remote Login Protocol
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem.txt | sort > listagem2.txt
ricardo@ubuntu-server:~/labs/redirecionamento$ cat listagem2.txt

```

- Exemplo de combinação de comandos :

```

ricardo@ubuntu-server:/var/log$ tail -n 5 syslog | grep systemd | sort > ~/labs/redirecionamento/log5.txt

```

- o user está na pasta /var/log no sistema → dir responsavel pelos logs
- tail -n 5 syslog → Vai mostrar as 5 ultimas linhas do arquivo syslog
- grep systemd → Vai filtrar o conteúdo com a string systemd

- sort > ~/labs/redirecionamento/log5.txt → Vai ordenar o conteúdo, e vai jogar no arquivo log5.txt dentro da pasta home do usuário

▼ Utilizando os pipes

- A função dos pipes é:
 - A saída do primeiro comando sirva de entrada para o segundo comando.
 - Exato no exemplo `cat teste.txt | more`, a saída do `cat` vira o `input` para o segundo comando.

▼ Filtrando as informações com o cut

- wc → mostra o número de linhas, palavras, e a informação em byte
 - -l → imprime o nº de linhas
- cut → serve para filtrar a saída de conteúdo de algum arquivo
 - -d → delimitador o que diz que está separando um campo de outro campo (cut -d "")
→ o espaço está separando)
 - -f1 → Indicador do campo escolhido
 - Podemos usar -f1,2,3,6 ou -f11,6- (este vai até o final)

▼ Utilizando as Regex

▼ Regex com o grep

- No Ubuntu utiliza o apt, com base na configuração do debian para gerenciamento de pacotes
- grep -E "^(pesquisa)" → vai filtrar palavras que começam com uma determinada palavra

```

computerization
computerization's
computerize
computerized
computerizes
computerizing
computers
microcomputer
microcomputer's
microcomputers
minicomputer
minicomputer's
minicomputers
supercomputer
supercomputer's
supercomputers
ricardo@ubuntu-server:~/labs/expresoes_regulares$ cat american-english | grep -E "^(computer"
computer
computer's
computerization
computerization's
computerize
computerized
computerizes
computerizing
computers

```

- grep -E "computer\$" → vai trazer o final

```
computer
microcomputer
minicomputer
supercomputer
```

- se for feito grep -E “^computer\$” → vai trazer somente o conteúdo com essa palavra

▼ Regex com o grep pt2

- grep -i → ignora a caixa da letra (ou seja A ou a)
- echo → joga uma determinada informação na tela
- grep -iE → ignora maisculo e minusculo, e puxa determinada informação especifica usando “^” e “\$”
- egrep → atalho para grep -E → Vai filtrar palavras que começam com uma determinada palavra

▼ Trabalhando com expressões regulares

- Para localizar a string `http` no início de cada linha do arquivo `/etc/services`, devemos utilizar:
 - `cat services | grep -E "^http"`
 - adicionamos o caracter ^ para indicar que a string deve estar localizada no início da linha.

▼ Regex com o grep pt3

- É possível pesquisar por variações nas palavras

```
ricardo@ubuntu-server:~/labs/expressoes_regulares$ egrep "^.oot$" american-english
Root
boot
coot
foot
hoot
loot
moot
root
```

- Nesta exemplo só vai buscar o conteúdo que contenha oot
 - Colocar `^.oot..$` → irá delimitar a quantidade de caracteres com os dois pontos
 - Colocar `[aeio]oot..$` → irá indicar as variações do começo
 - Colocar `[1-7]oot..$` → range numerico
 - Colocar `[l-n]oot..$` → range alfabetico

▼ Trabalhando com expressões regulares pt2

- Para localizar todas as linhas em um determinado arquivo que iniciem com os números de 00 a 07 devemos utilizar a seguinte comando:

- `cat teste.txt | grep -E "^0[0-7]"`

- Nesta pattern indicamos que o segundo caracter pode variar entre 0 e 7.

▼ Editores de texto

▼ Editores de texto nano e vi

- nano → Editor mais novo
 - nano + arquivo que você quer editar
 - todos os comandos para editar tem no help do nano
- Vi → mais antigo, e mais comum entre as distribuições
 - vi + arquivo que você quer editar
 - :q → sair do editor
 - :wq → sair e salvar a edição
 - i → inserir/editar
 - esc → sair
 - w + novo nome → altera o nome do arquivo

▼ Editores de texto

- Por padrão as novas distribuições Ubuntu estão vindo com quais editores pré instalados?
 - nano
 - o nano é o editor que vem por padrão no Ubuntu.

▼ Avançando no vi

- d → apaga linhas (recorta)
- 11 + d d → recorta 11 linhos
- :w → mantém o nome do arquivo
- Navegação
 - :40 → linha 40 vai para está linha
 - :1 & gg → volta para o início do arquivo
 - G → final do arquivo
 - / → busca a informação dentro do conteúdo arquivo
 - p → cola dps e o P → cola antes, do cursor
 - s/antigo/novo → substituir as strings dentro do arquivo
 - /g → todas as ocorrências nessa linha serão substituidas
 - %s → aplica para todo arquivo

▼ Copy e paste no vi

- Para fazermos o copy paste dentro do vi, utilizamos:

- `yy` e `p`.

- O `yy` copia as linhas e o `p` faz a colagem.

▼ Shell, GIT, Redes

▼ Lógica de programação com Shell Scripting

Gerando uma nova chave SSH e adicionando-a ao agente SSH - GitHub Docs

É possível acessar e gravar dados em repositórios no GitHub.com usando o SSH (protocolo Secure Shell). Ao se conectar por meio do SSH, você se autentica usando um arquivo de chave privada no computador local. Para obter mais informações, confira "<https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>"



- Criar um script shell que puxe todos os repositórios do github
 - Necessário ter a chave configurada.
- Primeiro criamos um arquivo .sh
- Utilizando a APIRest do github do nosso perfil conseguimos listar todas informações, e pesquisar pelas ssh_urls para poder integrar no scrip
- curl → imprime tudo oq é puxado da API
 - -s → silencia o comando curl
- awk → utilizando junto com o curl +| → faz manipulação de todo o texto, '/padrão que se repete/{ação}'
- No caso o comando ficará da seguinte forma `curl -s url-da-api | awk '/ssh_url/{print}` → Irá trazer todos os sshs dos repositórios
- `curl -s url-da-api | awk '/ssh_url/{print $2}'` → usando o \$2 puxamos somente o segundo campo da url ssh
- `curl -s url-da-api | awk '/ssh_url/{print $2}' | sed 's/^"/g'` → Com está extensão do comando irá remover as aspas do começo de todo o texto
- `curl -s url-da-api | awk '/ssh_url/{print $2}' | sed 's/^"/g' | sed 's/",$/g'` → irá remover as aspas e a vírgula do final
- No caso o comando todo ficará → `curl -s url-da-api | awk '/ssh_url/{print $2}' | sed 's/^"/g' | sed 's/",$/g'` → Utilizaremos em uma variável este comando
- Para utilizarmos o "git clone repositórios" teremos que fazer um loop se não ele não conseguira puxar
 - Para cada repositorio na lista de repositorios faça git clone repositorios
 - `for $repositorio in $repositorio do git clone $repositorio done`
- Por fim o script ficará da seguinte forma :

- Para rodar usar sh ou bash + nome do arquivo.

```
#!/bin/bash
//Necessario para chamar o bash

mkdir meus-repositorios-git
cd meus-repositorios-git

repositorios=$(curl -s url-da-api | awk '/ssh_url/{print $2}'|sed 's/^"//g'|sed 's/",$/g')

for $repositorio in $repositorio
do
  git clone $repositorio
done
```



[Git & GitHub](#)



[Redes](#)