1. Fundamentos do Dev Web:

Nessa jornada inicial, aprenderemos muito sobre o desenvolvimento web e este conhecimento virá em quatro partes:

1. Vamos conhecer mais sobre o sistema Bash e seus termos executados dentro do terminal;
2. Como funciona o Git, para armazenamento de códigos;
3. Toda a estrutura de HTML, CSS e JavaScript, criando e descobrindo o web dev;
4. Estilização em CSS, para sites responsivos e elegantes, além de recursos avançados de JS, como HOF, código assíncrono, testes unitários e muito mais!

## 2. Unix & Bash (part. 1):

O Unix é um sistema operacional datado de 1969 e, além de muito popular, ganhou diversas atualizações que se tornaram os principais sistemas do mercado. Aprenderemos como funciona o Unix e vamos interagir com ele utilizando o Bash. Serão abordados quais são seus principais componentes, suas variações, sua estrutura de diretórios, o que são seus arquivos e processos, o que é o terminal e como vamos interagir com ele através de comandos no shell. Após o módulo, seremos capazes de executar comandos no terminal para navegar entre os diretórios, criar e manipular arquivos através do diretório e utilizá-lo para realizar buscas.

Para desenvolver softwares cada segundo é precioso. Aprenderemos essa ferramenta com a intenção de sermos mais assertivos, ágeis e produtivos. Na grande maioria das vezes, o custo, performance e facilidade de interação em sistemas baseados em Unix são indicativos de otimização para a hospedagem de nossos projetos na Web.

O que é Unix, Shell e Kernel?

O Unix é o pai de quase todos os OS que conhecemos hoje. Surgido nos anos 60, e construído a partir da linguagem de programação C. Ele foi o primeiro a introduzir alguns conceitos muito importantes para os sistemas operacionais, como suporte a multiusuários (que permite o acesso de diversos usuários, sem que cada atividade interfira na atividade do outro), o conceito de multitasking (que permite o usuários a operar diversas tarefas simultaneamente) e tem portabilidade (o que permite a adaptação para diversos OS sem muito esforço). Dentre os mais famosos OS que são baseados no Unix, estão o Linux e o MacOsx. O Unix é, basicamente, montado em três níveis:

1. **Kernell** - Podemos dividir um PC em duas partes (softwares e hardwares) e o kernel é o responsável por fazer a interação entre os dois. É o núcleo do sistema operacional, que gerencia os recursos do sistema e permite que os programas façam uso deles. Ele começa a funcionar assim que o PC é ligado. Nesse momento ele verifica todo o hardware indispensável para o funcionamento (como placa de vídeo, processador e etc) da máquina. O Os é carregado em seguida e uma vez que o usuário faça seu login, ele é responsável por administrar as principais funções dentro do OS (incluindo o gerenciamento de memória, de processos e de arquivos).
2. **Shell** - É a interface entre o OS, o usuário e o núcleo do sistema, ou seja, o **Sistema Operacional, Você e o Kernel**. O primeiro processo executado ao entrar no sistema (ex: o login) é o shell. Geralmente, o shell usa uma interface de linha de comando (**Command Line Interface**) ou uma interface gráfica (**Graphic Use Interface**), dependendo da função e operação. A função do Shell é ler a linha de comando, interpretar seu significado, executar o comando e devolver o resultado para o usuário. Existem diversos tipos de Shell. Os principais são: **SH, BASH, CSH, TCSH, ZSH.**
3. **Aplicativos** - Nesta camada se encaixam os softwares que os usuários utilizam e a maioria dos comandos do Unix.

Processos e Arquivos:

**Processos -** Os processos é tudo que você faz no Unix. A cada vez que é executada uma tarefa ou um comando, ele é enviado ao Unix como um processo. Cada processo possui um identificador único, chamado de PID (ProcessIDentifier) que carrega consigo as informações como: **Proprietário, Sessão do Shell, Estado Atual (funcionando, suspenso ou aguardando), linha de comando utilizada e outras diversas**).

**Arquivos -** Tudo é um arquivo no Unix. Diretórios, arquivos e dispositivos são arquivos. Sistemas de arquivos do Unix são organizados em uma hierarquia de raiz. O diretório raiz (**root ou somente “/”**) é o nível mais alto do sistema de arquivos. Todos os outros arquivos e diretórios estão debaixo do diretório root. Todos os arquivos possuem informações como: **Tipo do Arquivo, Permissões, Hard links, dono, grupo, tamanho, Data e Hora da última modificação e Nome**).

Um exemplo de como funciona a hierarquia de arquivo no Unix:

**/**  - diretório raiz (acima de todos)

**boot** (arquivos de config para inicialização) **bin** (“binarie” - dispositivos binarios) **dev** (“devices” - arquivos do dispositivo) **etc** (“scripts de inicialização) **home** (diretório de direcionamento para diferentes usuários) **proc** (“process” - diretório dinâmico especial que mantém infos sobre o estado do sistema, incluindo processos recentemente executados) **root** (“adm sys” - diretorio home do usuario root) **sbin** (“sys binaries” - são os binários de sistema importantes) **tmp** (“temporarily” - arquivos temporarios) **usr** (“**U**ser **S**ystem **R**esources” - arquivos, na maioria das vezes, disponíveis para todos os usuários) **var** (“variable data” - arquivos variáveis, como logs e bancos de dados)



## 3. Unix & Bash (part. 2):

Nessa parte vamos aprimorar os conhecimentos no Unix,

sendo capazes de utilizar os comandos de **find** e os operadores **> (maior), >> (maior maior) e | (pipe)**, como alterar as permissões dos arquivos e diretórios, verificar status e enviar sinais a processos UNIX. Isso é extremamente importante pois traz flexibilidade e segurança para o código, sendo que o aprendizado em se comunicar com o próprio script garante que você usufrua de ciência sobre quem, como, porquê e onde estão alterando os códigos de manipulação (e se eles têm autorização para fazê-lo).

Os inputs e outputs são as entradas e saídas de dados que comandamos ao sistema. O comando **cat** vem de concatenação e ele é um dos comandos que consegue receber e atribuir valores como inputs no shell. Para colocarmos valores no comando cat, utilizamos o sinal de **> (significa atribuição de valores definitivos, sobrescritos e de gravação para determinado arquivo)**. Já o sinal de **>> significa que acrescentamos, sem sobrescrever, mais informações ao arquivo,** salvando as informações acrescidas após as informações já contidas nele. Ambos os sinais representam um **input** (**entrada**) de dados.

Para juntarmos o valor de dois arquivos diferentes em um terceiro arquivo, utilizando o sinal **> (maior que).** Para isso, sinalizamos os dois arquivos que queremos juntar, sob o comando **cat** e jogamos o nome do terceiro arquivo que queremos criar após o sinal de input.

Uma maneira de manipular as informações de arquivos é o comando **sort** e, ao utilizarmos ele, **catalogamos as informações em ordem alfabética.** Utilizando o sinal **<** os valores são mostrados para nós em ordem alfabética, **somente para visualização,** ou seja, o sinal de **< (menor que) é um operador de output (saída)**.

Podemos mesclar inputs e outputs, como por exemplo, usando o comando **sort,** seguido do output **<** e **>** de input, após o arquivo desejado, conseguimos criar um segundo arquivo em ordem alfabética. Nessa lógica, pedimos a ordem através de **sort,** pedimos uma saída do arquivo através de **<** e, logo após, uma entrada em outro arquivo através de **>**.

O conceito de **| (pipe)** junta comandos e valores, passando outputs de um comando para outro comando. Por exemplo **who | sort**  vai juntar as funcionalidades de ambos os comandos, mostrando uma lista dos usuários conectados ao terminal (**who**), em ordem alfabética (**sort**) concatenando as duas funções. O **| (pipe)** pode ser utilizado quantas vezes julgarmos necessário e com as funcionalidades **- (flag)** que precisarmos.

Permissões:

É importante sabermos quem tem acesso às nossas informações e para saber todas as permissões dadas dentro do nosso sistema, utilizamos o comando **who (“quem”)** e ele mostra a lista de usuários conectados ao terminal que estamos.

Dentro do Unix existem certos conceitos de acesso a arquivos. Podemos visualizar esses níveis de acesso pelo comando **ls -la (que mostra a lista de serviço (**ls - service list**), em uma lista aberta (**-la, que é uma flag + modo lista + all, selecionando todos os arquivos**).**

Dentro da lista de acesso, existem algumas nomenclaturas. A letra **r** significa **read** e que **leitura permitida do arquivo ou diretório.** Já o **w** vem de **write** e significa que você pode **editar um arquivo ou modificar o conteúdo do diretório.** O **x** significa **execute** e significa que **você pode executar um arquivo ou acessar um diretório pelo comando cd.**  Se no início da sequência existir um **-** ou **null** isso **indica um arquivo comum** e se existir um **d, significa que o arquivo é um diretório.**

Dentro dessa sequência de caracteres de permissão existem quatro grupos: a primeira é um **- ou d** e significa a modalidade do arquivo; a segunda parte indica a permissão de **usuário** e é dada em **três caracteres, sendo r, w ou - (null);** a terceira permissão é dada ao **grupo do usuário, também dada nos três caracteres.** E a quarta permissão diz respeito a **outros usuários que não fazem parte do grupo delimitado** de usuários do terminal, indicado pelos três caracteres.

Podemos sempre alterar as permissões de nossos arquivos pelo comando **chmod**  **(change mode),** que passa as informações que queremos atribuir às permissões. Seguido do caracter **u (user)** altera as apenas as permissões do usuário (**segundo bloco**), se seguido do caracter **g (group)** altera apenas as modalidades do grupo (**terceiro bloco**), seguido do caracter **o (others)** altera apenas a modalidade de outros usuários (**quarto bloco**) e com o caractere **a (all)** alteramos a permissão de todos os blocos. Dentro disso, temos a opção de retirar a permissão utilizando **- (flag)** aqui utilizado como sinal de subtração, com o caractere **+ (plus)** acrescentamos uma permissão e o **= (equal)** permanece as permissões como elas estão.

Podemos também utilizar as permissões de modo octal. Para compreendermos esta outra forma, precisamos entender que eles são administrados por meio de valores como descrito abaixo:

* Leitura r - 4
* Escrita w - 2
* Execução x - 1

Esses valores são permissões com base em bits de ligados = 1 e desligados = 0 ,

**rwx** = 111 ( 7 | **Acesso Total** )

**r--** = 100 ( 4 | **Somente Leitura** )

**-w-** = 010 ( 2 | **Somente Escrita** )

**--x** = 001 ( 1 | **Somente Execução** )

**rw-** = 110 ( 6 | **Somente Leitura e Escrita** )

**r-x** = 101 ( 5 | **Somente Leitura e Execução** )

-**wx** = 011 ( 3 | **Somente Escrita e Execução** )

**---** = 000 ( 0 | **Todos Acessos Negados** )

Portanto, cada vez que você liga a chave de leitura, atribui-se o valor para esta chave somando-se com as demais chaves de administração de escrita e execução caso você também deseje ligá-las. A sintaxe para realizar esta alteração continua a mesma que no modo literal, sendo **chmod 766 [arquivo ou diretório]** . Cada número corresponde aos grupos de pessoas usuárias, de grupo e de outras pessoas usuárias.

Processos e Trabalhos:

Para acessarmos a lista de processos que estão sendo executados no momento, utilizamos o comando **ps (process)** no terminal e ele dá o número de **PID (ProcessIDentifier)** único dos processos. Os processos podem estar em **execução, background ou suspensos.** O comando **sleep** é um exemplo de processo criado que para o terminal durante um tempo delimitado em **segundos**. O operador **& (‘e’ comercial)** junto com o comando, deixa ele em background, para que possamos fazer outras coisas enquanto eles acontecem. Para suspender um processo que não está em background, utilizamos o comando **ctrl + z** e isso dá o status de **suspenso** no processo. Para colocá-lo rodando em background, utilizamos o comando **bg (background)** e ele volta a rodar, retirando-o da suspensão mas rodando-o no plano de trás do que estamos fazendo. Para acessar **todos** os processos em execução, utilizamos o comando **jobs (trabalho).** Para voltar a executar em **primeiro plano** um processo que colocamos em **background**, utilizamos o comando **fg (for ground)**, seguido do caracter **% e seu número em lista.** Para finalizarmos um processo, utilizamos o **ctrl + c e isso indica que o processo em PRIMEIRO PLANO foi finalizado,** ou comando **kill** (**matar**), também seguido do caracter **% e seu número de lista** ou **seguido de seu número PID.** Quando um processo não aceita ser finalizado e precisamos forçar seu encerramento, utilizamos o comando **kill** seguido da flag **-9** e o número do processo que desejamos encerrar, de forma abrupta.

Outros exemplos de processos, são o **find (achar), history (histórico) e echo (eco):**

* O **find** é um comando para localizarmos arquivos ou diretórios dentro do Unix. Podemos utilizar os parametros de **name, date, size e type**. Caso nenhum atributo seja passado para o comando, ele pesquisará tudo que estiver dentro do **diretório atual**.

**Exemplos:**

**# Para listar todos os arquivos que terminam em .txt**

**find . -name "\*.txt"**

**# Para localizar todos os diretórios**

**find . -type d**

**# Para localizar todos os arquivos**

**find . -type f**

**# Localizar tanto arquivos quanto diretórios que comecem**

**# por algum trecho**

**find ./teste -name exemplo\***

**# Resultado**

**./teste/exemplo.txt**

**./teste/exemplo**

**# Localizar somente arquivos que comecem por algum trecho**

**find ./teste -type f -name "exemplo\*"**

**# Resultado**

**./teste/exemplo.txt**

**# Localizar somente diretórios que comecem por algum trecho**

**find ./teste -type d -name "exemplo\*"**

**# Resultado**

**./teste/exemplo**

* Já o **history** mostra o histórico geral de comandos que utilizamos no nosso terminal. A quantidade de comandos mostrados podem ser configurados a partir de um **- (flag)** e um número arbitrário que delimita a quantidade de comandos mostrados.

**Exemplo:**

**#** Mostra o histórico de comandos

history

**#** Pegar os últimos 10 comandos

history | tail

* O **echo** é utilizado em scripts ou terminais, e ele serve para exibir as mensagens de arquivos na tela, ou uma mensagem dentro de um arquivo, editando-o.

**Exemplo:**

echo "Este é um teste"

**#** Resultado

Este é um teste

**#** Pode ser usado para colocar textos dentro de arquivos.

echo "Este é mais um teste" > teste.txt

cat teste.txt

Este é mais um teste

