Algoritmos Naturais

Desenvolver Algoritmos

Msc. Lucas G. F. Alves
e-mail: lgfalves@senacrs.com.br

Senac

Planejamento de Aula

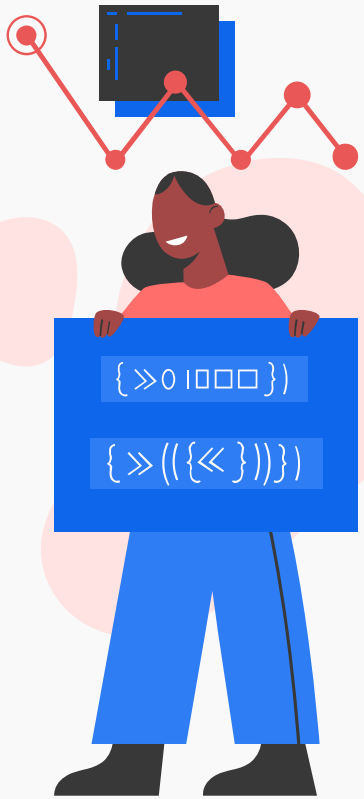
Revisão Ambiente Dev

JavaScript

Sintaxes JavaScript

Tipos de Variáveis

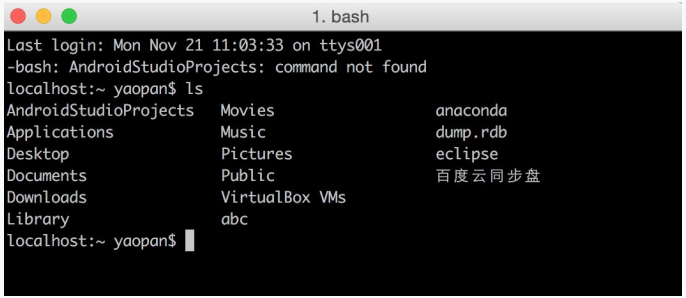
Conversão entre Tipos



Introdução ao Computador

Terminal

Os terminais dependem do Sistema Operacional.

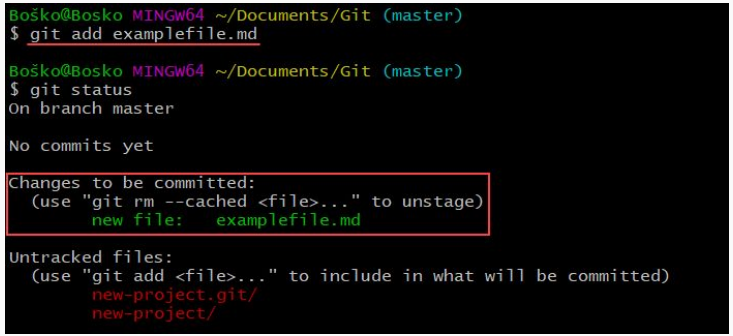


Sistemas Unix (Linux e MacOS)

- Herdaram o terminal da época em que não existiam interfaces gráficas.
- **Bash/Zsh**: um terminal muito poderoso.

Windows

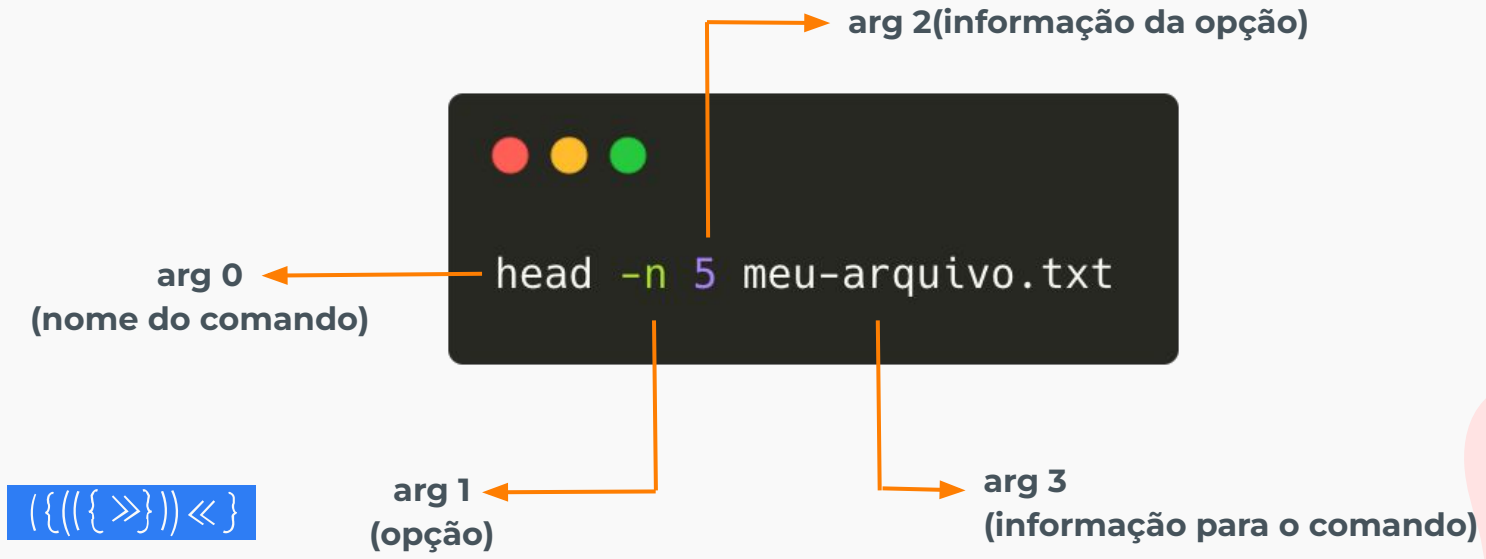
- Foi desenvolvido com foco na interface gráfica.
- Usaremos, então, o **GitBash** que simula o bash no Windows.



Comandos

O que são comandos?

Exemplo: Visualizar as 5 primeiras linhas de um arquivo chamado arquivo.txt.



Comandos

echo

Imprime algo no terminal.

```
echo "Hello World" # imprime Hello World no terminal
```

whoami

Imprime o nome do usuário na tela

```
whoami # retorna o nome do usuário atual ex: 'seuUsuario'
```

Comandos

clear

Limpa tudo que está aparecendo no terminal, serve para leitura e organização.

```
clear #limpa tudo que está no terminal
```

pwd

Sigla para print working directory. Mostra o endereço completo do diretório/pasta em que estamos trabalhando.

```
pwd # retorna a pasta que o terminal está atualmente ex: '/c/Users/seuUsuario'
```

Comandos

ls

O nome vem de list. Lista os arquivos e pastas do diretório em que estamos.

```
ls # retorna nome de arquivos e pastas presentes na pasta atual
ls -a # retorna nome de arquivos e pastas, incluindo os ocultos (cujo nome começa com `.`)
ls -l # retorna informações de arquivos e pastas, incluindo tamanho, proprietário e outras informações
ls -la # soma dos dois modificadores anteriores
```

cd

Sigla para change directory. Muda o diretório/pasta em que estamos.

```
cd ./minha-pasta # troca o diretório atual para a subpasta "minha-pasta"
cd # vai para a pasta "home" do usuário atual
cd ../ # vai para a pasta acima da atual
```

Comandos

mkdir

Sigla para make directory. Cria um novo diretório.

```
mkdir minha-pasta # cria uma pasta chamada 'minha-pasta' no diretório atual
```

touch

Cria um novo arquivo.

```
touch index.html # criará um arquivo chamado index.html na pasta atual
```

Comandos

rm

Vem da palavra remove. Apaga arquivos de maneira IRREVERSÍVEL e SEM PEDIR CONFIRMAÇÃO.

```
rm ./meu-arquivo-gigante.txt # remove imediatamente o arquivo 'meu-arquivo-gigante.txt'
rm -r ./minha-pasta # remove Recursivamente todos os arquivos e sub-pastas da 'minha-pasta'
```

mv

Vem da palavra move. Permite mover ou renomear arquivos de um diretório.

```
mv ./meu-arquivo-gigante.txt ./minha-sub-pasta # move 'meu-arquivo-gigante' para 'minha-sub-pasta'
mv ./meu-arquivo-gigante.txt ./meu-gigante.txt # renomeia 'meu-arquivo-gigante.txt' para 'meu-gigante.txt'
```

Comandos

cp

Vem da palavra copy. Copia arquivos de um diretório para outro.

```
cp ./meu-arquivo-gigante.txt ./minha-sub-pasta # copia 'meu-arquivo-gigante' para 'minha-sub-pasta'
```

cat

Vem da palavra concat. Ele concatena tudo que está no arquivo e imprime no terminal.

```
cat meu-arquivo-gigante.txt # imprime o conteúdo do arquivo 'meu-arquivo-gigante.txt'
```

Comandos

head

Imprime as 10 primeiras linhas de um arquivo. A opção **-n** permite indicar quantas linhas são.

```
head meu-arquivo-gigante.txt # imprime as 10 primeiras linhas do arquivo 'meu-arquivo-gigante.txt'
head -n 20 meu-arquivo-gigante.txt # imprime as 20 primeiras linhas do arquivo 'meu-arquivo-gigante.txt'
```

tail

Imprime as 10 últimas linhas de um arquivo. A opção -n permite indicar quantas linhas são.

```
tail meu-arquivo-gigante.txt # imprime as 10 últimas linhas do arquivo 'meu-arquivo-gigante.txt'
tail -n 20 meu-arquivo-gigante.txt # imprime as 20 últimas linhas do arquivo 'meu-arquivo-gigante.txt'
```

Comandos

grep

Permite buscar um determinado texto no conteúdo de um arquivo

- -A x imprime x linhas após o texto
- -B y imprime y linhas antes do texto

```
grep Future4 ./lista-de-empresas.txt # Busca pela palavra Future4 no arquivo lista-de-empresas.txt e imprime toda a linha encontrada
```

Exercícios

1) Abrir o terminal dentro da pasta do template. Dica: botão direito do mouse em qualquer parte dentro da pasta ou no vs code.
2) Ler o conteúdo do arquivo de texto pokemons.txt. Dica: comando ‘cat’
3) Descobrir qual é o número do Pikachu. Dica: comando ‘grep’.
4) Descobrir os dois pokémons que vêm antes do Pikachu. Dica: comando ‘grep’ com opção -B.
5) Descobrir os três pokémons que vêm depois do Pikachu. Dica: comando ‘grep’ com opção -A.
6) Mostrar apenas os pokémons da primeira geração (do 1 ao 151). Dica: comando ‘head’ com opção -n.
7) Mostrar apenas os 100 últimos pokémons da lista. Dica: comando ‘tail’ com opção -n.

Git

O que é Git?

O git é uma ferramenta que permite fazer o gerenciamento de versão de nossos projetos (de programação ou de outros arquivos).

Facilita o trabalho colaborativo.

Fácil de manter o rastreamento de arquivos que são alterados por duas pessoas ao mesmo tempo.

Git

Git vs. Github

O **Git** é a ferramenta que gerencia as versões e colaborações em projetos.

O **Github** é um serviço cloud que permite armazenar os projetos.

Existem outros, como **Bitbucket** e **Gitlab**. Todos usam a mesma ferramenta, o **Git**.

Github

Repositórios

O projeto que está na nossa máquina é chamado de:

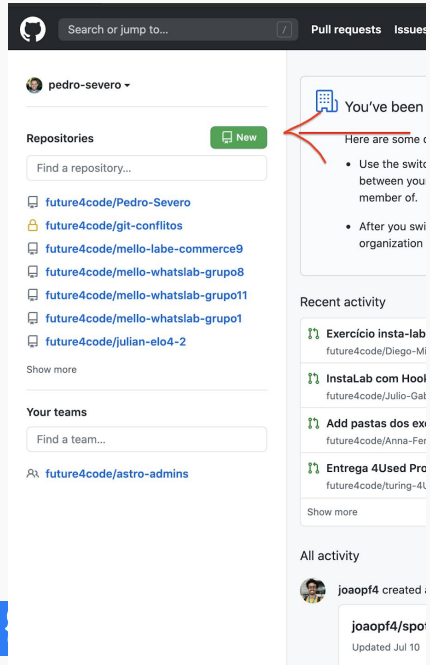
- **repositório (ou repo) do git local.**

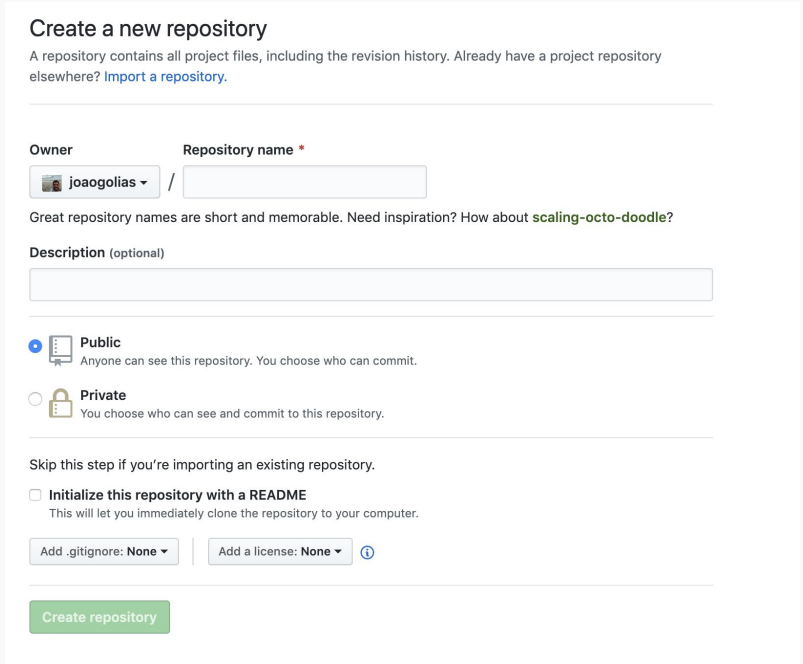
O projeto que está no github, chamados de:

- **repositório (ou repo) do git remoto.**

Github

Criando um repositório.

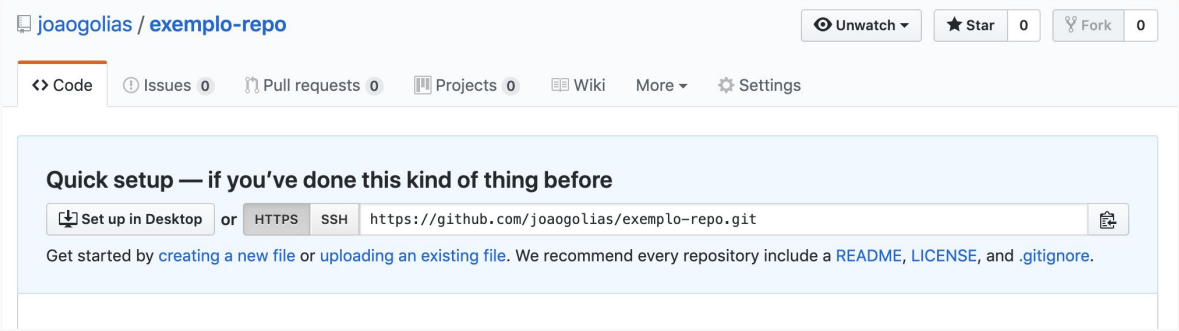




Github

git clone link-do-repo

É o comando que clona as informações do repositório remoto em uma pasta (repositório) na nossa máquina.



Github

Comandos do Github para salvar localmente

git status

Indica o status do repositório.

- Arquivos/pastas criados;
- Arquivos/pastas modificados;
- Arquivos/pastas removidos;

Github

Comandos do Github para salvar localmente

git add nome-do-arquivo

Envia os arquivos modificados, removidos e criados para a Staging Area (que é local).

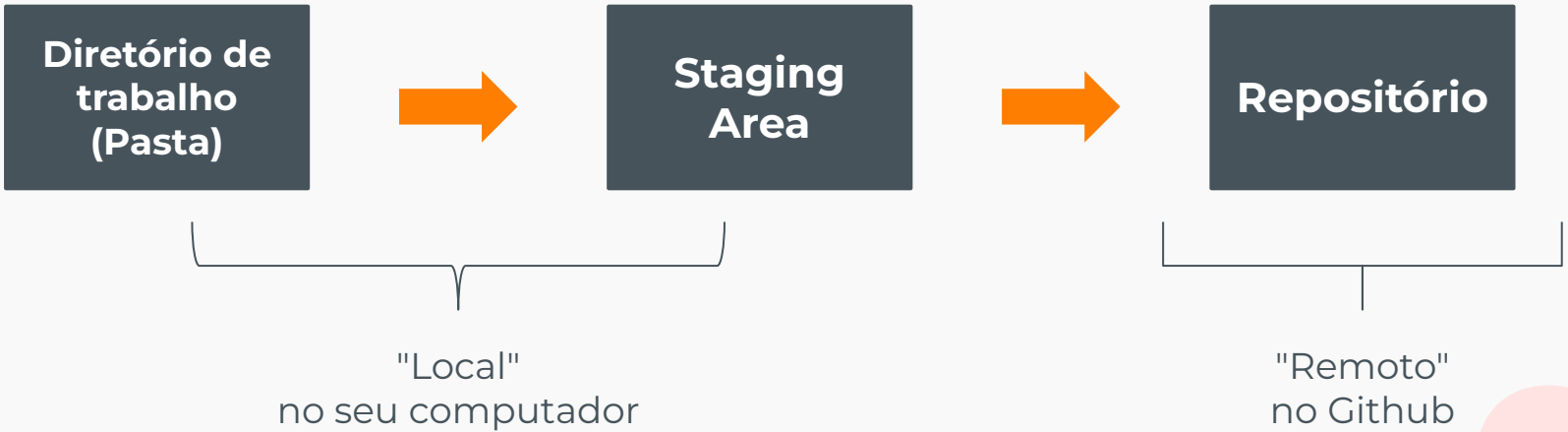
Também podemos utilizar a opção **git add --all** para adicionar todos os arquivos do repositório.

A opção **git add .** para adicionar todos os arquivos da pasta onde você se encontra.

Github

Comandos do Github para salvar localmente

git add .



Github

Comandos do Github para salvar localmente

git commit -m "mensagem"

Demarca uma versão do seu projeto com os arquivos que estiverem na Staging Area.

A mensagem deve explicar as modificações, criações e deleções feitas.

Github

Comandos do Github para salvar localmente

git commit -m "mensagem"

Demarca uma versão do seu projeto com os arquivos que estiverem na Staging Area.

A mensagem deve explicar as modificações, criações e deleções feitas.

Não esquecer do **-m**

Caso esqueça, você vai entrar em uma parte do terminal, que, para sair, você deve digitar: **esc esc :q**

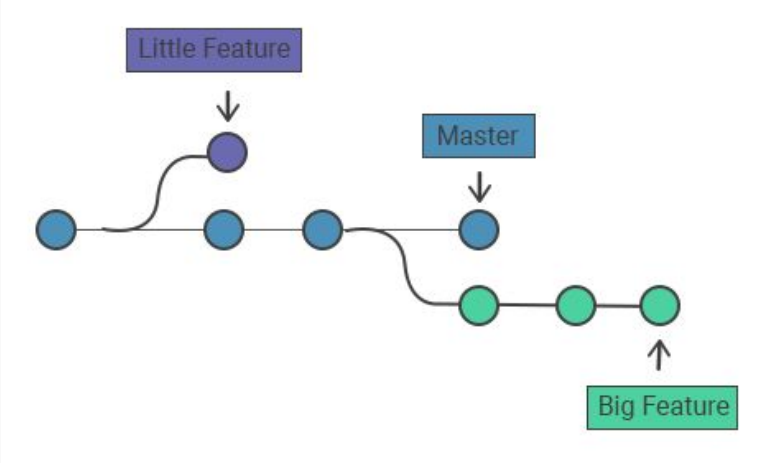
Não esquecer das aspas (").

Github

Dividindo o Trabalho

git branch

Branch (ramo/galho) é uma ramificação do projeto principal



Github

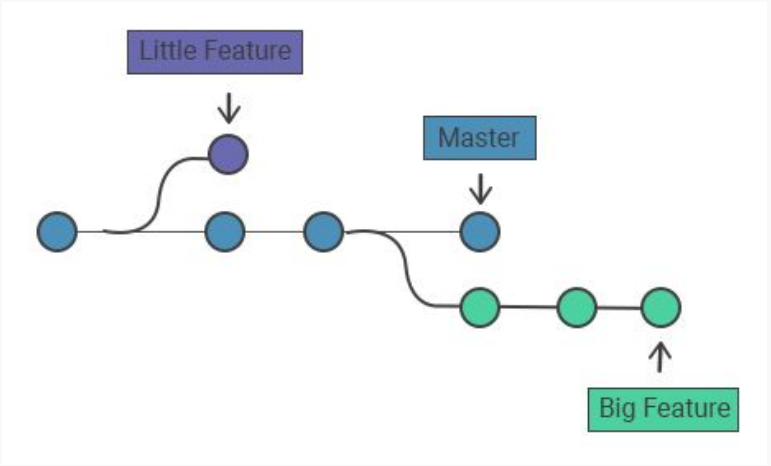
Dividindo o Trabalho

git branch

Branch (ramo/galho) é uma ramificação do projeto principal

Este comando em si mostra a lista de branches que estão no seu repositório local.

A branch padrão se chama main* e, a princípio, apenas ela vai existir no seu repositório.



Github

Dividindo o Trabalho

git branch nome-da-branch

Permite criar uma nova branch, com o nome que você escolheu.

git checkout nome-da-branch

Permite acessar uma branch que já foi criada (localmente ou remota).

git checkout -b nome-da-branch

É uma junção dos comandos anteriores.Ele cria uma nova branch e já acessa diretamente.

Github

Salvando no Remoto

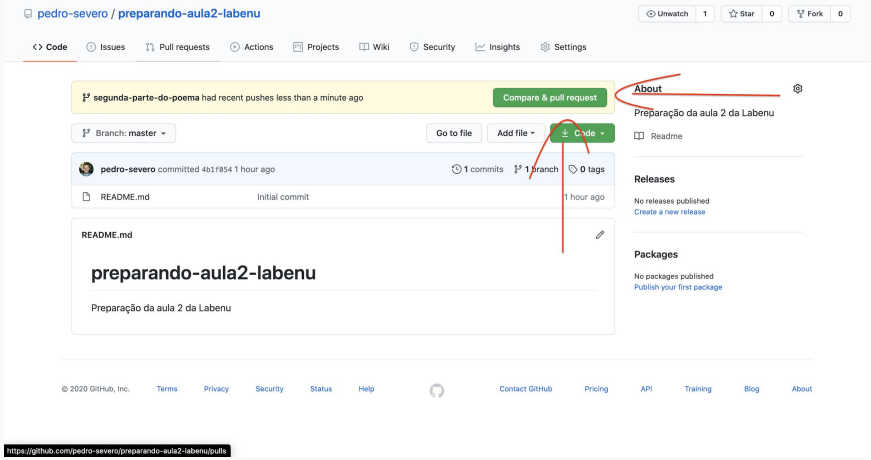
git push origin nome-da-branch

Envia as suas alterações feitas para a branch no repositório remoto Ele só envia as alterações que foram colocadas no commit

Github

Pull Request (PR)

Depois de fazer todas as alterações na sua branch, você deve querer que elas sejam mescladas com a branch principal (a master).
A esta **mesclagem**, damos o nome de **merge.**
Para fazer um merge no GitHub, nós devemos criar um Pull Request (ou PR) antes



Github

Pull Request (PR)

Quando trabalhamos em equipe, os membros dela avaliam os nossos PRs.

- Pedindo correções no código
- Sugerindo alterações

Após o processo de **Code Review** (CR) e o seu código estiver aprovado, ele pode ser **mergeado** na main.

Github

Atualizando o local

git pull origin nome-da-branch

Atualiza a branch em questão no seu repositório local com as alterações commitadas na branch remota.

Se você já estiver acessando a branch que deseja atualizar, o comando pode ser reduzido a **git pull**.

Exercícios

Exercícios

1) Criar um repositório chamado **DA-24-T.**
2) Clonar o repositório no computador local em C:Git.
3) Entrar na pasta do repositório utilizando o comando CD.
4) Criar a pasta aula 2 e colocar o arquivo pokemons.txt dentro da pasta.
5) Utilizar comando Git Status para verificar se apareceu o arquivo pokemons.txt.
6) Adicionar com o comando git add .
7) Commitar com o comando git commit -m “adicionado o arquivo pokemons.txt”
8) Salvar no repositório com o git push.

Github

Resumo

Começando o repositório

git clone link-do-repo

Salvando localmente

git status
git add nome-do-arquivo
git add .
git commit -m "mensagem"
git log

Github

Resumo

Dividindo o Trabalho

git branch
git branch nome-da-branch
git checkout nome-da-branch
git checkout -b nome-da-branch

Salvando no Remoto

git push origin nome-da-branch
git pull origin nome-da-branch

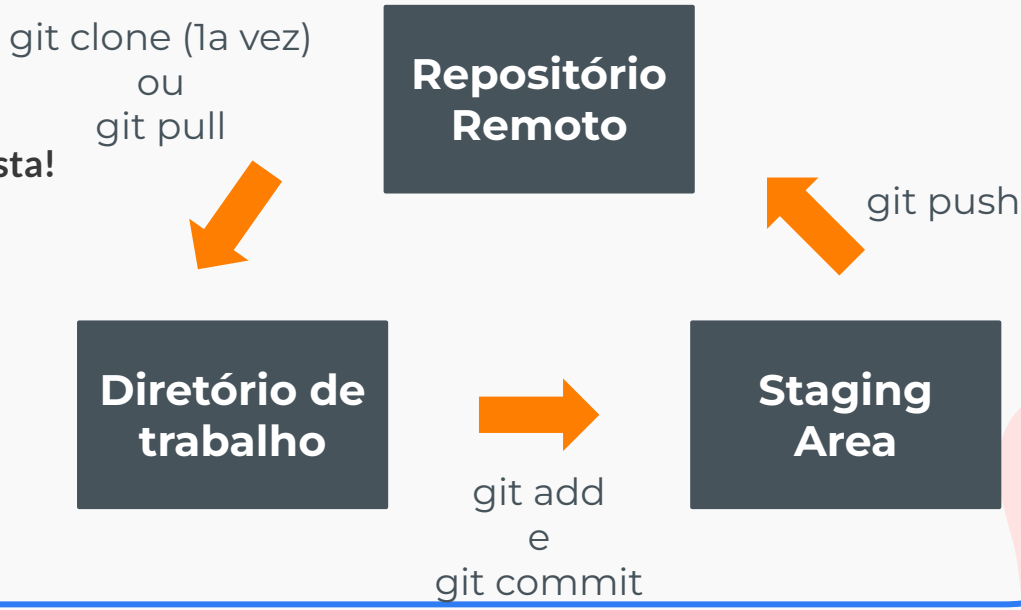
Github

Resumo

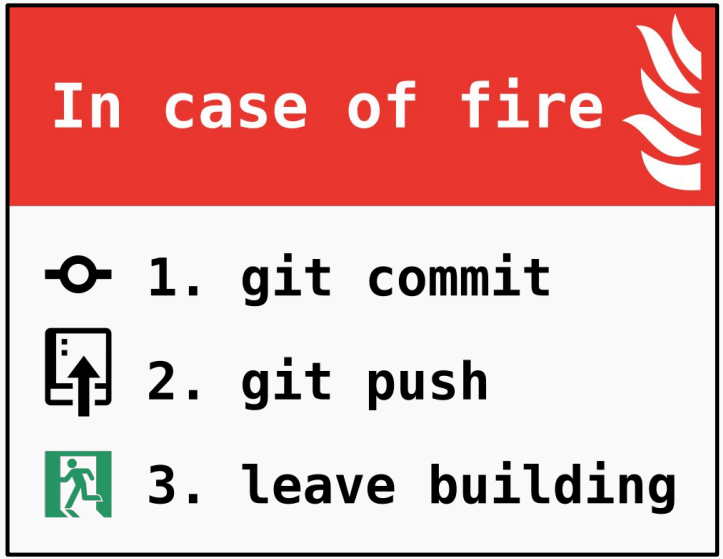
Importante: comandos de git **não são** o mesmo que comandos do terminal!

- Ex: git mkdir

Importante 2: **branch não é pasta!**



Github



JavaScript

JavaScript

Uma linguagem de programação é um conjunto de normas (sintaxe) que permite criar comandos para o computador

Baixo **Alto**

Nível de Abstração

Binário

Assembly

C, C++

Java, Python, **Javascript**

JavaScript

JavaScript é uma linguagem moderna e aceita por todos os navegadores mais usados hoje em dia.

Usaremos, agora no início do curso, o nosso navegador (indicamos Chrome ou Firefox) para ver os códigos que criamos

Um navegador precisa de uma página HTML para rodar o seu código, então veremos agora de maneira muito simples como criar uma página

JavaScript

Começando um Projeto

Precisamos ter a extensão **live server** instalada no **VSCode** e criar uma pasta com dois arquivos:

index.html ⇒ Primeiro arquivo que o navegador olha.

index.js ⇒ Arquivo onde escreveremos nosso código JS.

Obs: os dois arquivos devem estar na mesma pasta!

JavaScript

Começando um Projeto

Começamos colocando o código padrão do HTML (se você apertar os botões **!** + **enter** o VSCode faz esse código pra você!)

Adicionamos uma linha de código, dentro da tag head, que vai ligar nosso arquivo **index.js** ao HTML

```
<script src="index.js" defer></script>
```

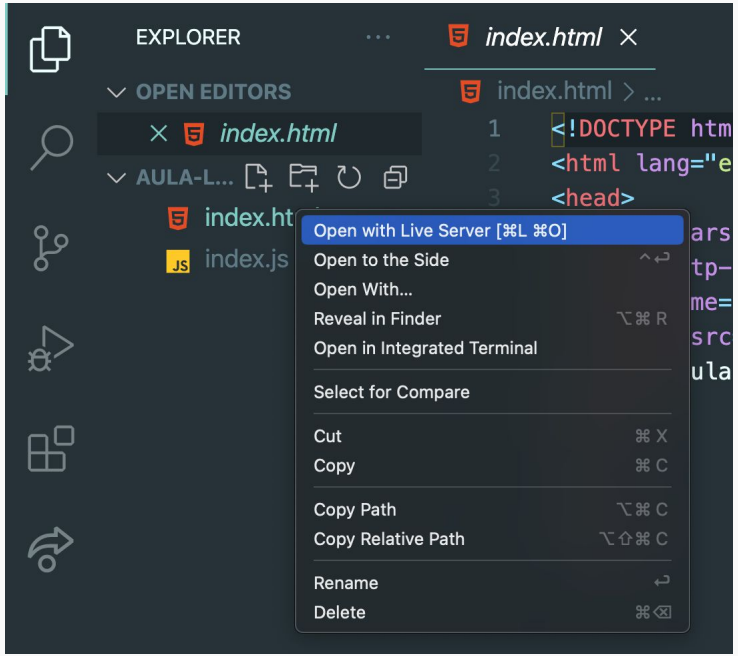
JavaScript

Começando um Projeto

Para abrir o html com o Live Server:

Clique com o botão direito em cima do arquivo index.html;

Depois clique em Open with Live Server ou Abrir com Live Server;



Sintaxes Basicas

Sintaxes Basicas

Comentários

São estruturas que permitem escrevermos textos que serão ignorados para executar o programa.

Eles devem começar com **// ISSO É UM COMENTÁRIO** ou estarem entre

/*

Um comentário é ignorado no momento em que o programa é rodado

*/

Sintaxes Basicas

Imprimindo no console

O JS possui uma sintaxe específica para imprimir informações no console do navegador.

```
console.log("Olá Mundo!")
```

Sintaxes Basicas

Imprimindo no console

O JS possui uma sintaxe específica para imprimir informações no console do navegador.

```
console.log("Olá Mundo!")
```

Pedindo informações para o usuário

Em aplicações Web, conseguimos pedir que o usuário nos passe alguma informação, assim:

```
prompt("Qual é o seu nome?")
```

Sintaxes Basicas

Resumo

Para trabalhar com JS, vamos usar um arquivo index.html e um index.js.

Os dois arquivos devem estar na mesma pasta.

Para linkar o arquivo JS ao HTML, usamos a tag <script src="./index.js" defer></script>.

Comentários: de linha // e de bloco /* */.

Imprimir uma info: console.log().

Solicitar uma info do usuário: prompt().

Variáveis

Variáveis

O que sao?

Variáveis são estruturas que permitem guardar e acessar quaisquer informações no nosso código

Funcionam como **gavetas** (guardar e acessar)

Gaveta B recebe o arquivo A, mas o Arquivo A não é a gaveta B

Variáveis

Sintaxes

Antes de usarmos estas variáveis, nós precisamos declará-las (criá-las).

```
    const   novaVariavel     =          100
//      ^           ^           ^           ^
// declaração   nome    atribuição     valor
```

Devemos escolher nomes significativos.
Nomes não podem começar com números ou caracteres especiais.
Utilizamos o padrão **camelCase**.

- primeira letra minúscula;
- primeira letra entre uma palavra e outra é maiúscula;

Variáveis

Sintaxes

const: quando uma variável é declarada usando const, nós dizemos que ela é constante.

O seu valor **NÃO** pode mudar ao longo do programa.

```
const idade = 22
idade = 24
```

Errado

let: quando uma variável é declarada usando let, ela **PODE** ter seu valor alterado.

```
let idade = 22
idade = 24
```

Variáveis

Sintaxes

Podemos imprimir mais de uma coisa no console separando elas por vírgula.

Será adicionado um espaço entre as palavras.

```
const nome = "Fulana"
const idade = 21

console.log("Olá!", "Meu nome é ", nome, "e eu tenho", idade, "anos")
// Olá! Meu nome é Fulana e eu tenho 21 anos
```

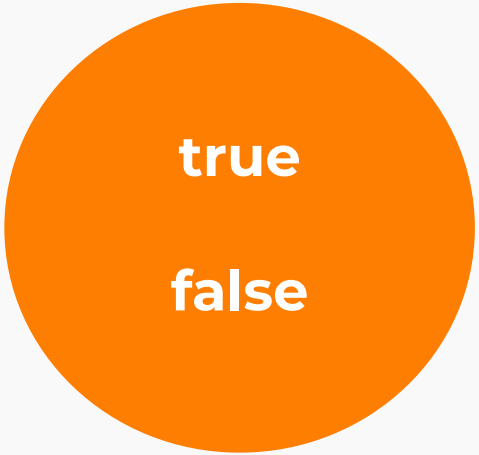
Tipos de Variáveis

Tipos de Variáveis

Os valores que as variáveis do JS assumem possuem tipos. Hoje apresentaremos três deles:

Numbers

Strings



Booleans

Tipos de Variáveis

Numbers: são os tipos que representam números

```
const idade = 23
const altura = 1.79
const temperatura = -20
```

Strings: são os tipos que representam conjunto de caracteres (texto)

```
const nome = "Pedro"
let idade = "23"
```

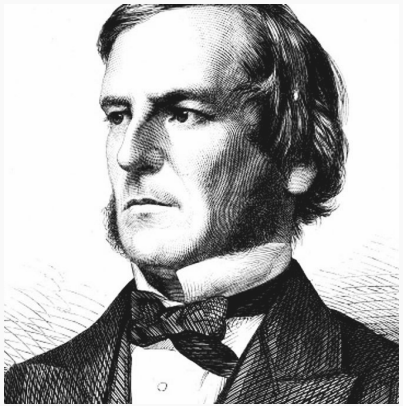
Tipos de Variáveis

Booleans

George Boole foi o inventor do que chamamos de álgebra booleana.

Esta álgebra leva em consideração que os valores assumidos são somente:

- TRUE ou 1;
- FALSE ou 0;



```
let souUmBoolean = true
souUmBoolean = false
```

Exercícios

Exercícios

Faça os seguintes itens:

1) Crie uma variável e atribua seu primeiro nome.

2) Crie uma variável e atribua seu sobrenome.

3) Crie uma variável e atribua sua idade.

4) Crie uma variável que diga se você é ou não estudante.

5) Imprima o seu nome, sobrenome, idade e status de estudante no console.

Resumo

Usamos **let** para declarar variáveis que podem ter seu valor alterado no decorrer do programa.

const para variáveis que terão valores constantes.

Strings representam textos.

Numbers representam números.

Booleanos são tipos que recebem apenas dois valores: verdadeiro (true) ou falso (false).

Tipos de Variáveis

Como descobrir o tipo da variável.

typeof: comando que permite ver o tipo do valor da variável.

```
const got = "Game Of Thrones"
const temporadasDeGot = 8

typeof got //string
typeof temporadasDeGot //number
```

Tipos de Variáveis

undefined

Tipo que representa a falta de valor de uma variável.

```
let novaVariavel
typeof novaVariavel //undefined

novaVariavel = 2
typeof novaVariavel //number

novaVariavel = undefined
typeof novaVariavel //undefined
```

Tipos de Variáveis

null

null: também representa a falta de valor da variável.

Existem algumas diferenças entre undefined e null, e uma delas é que o null precisa ser atribuído diretamente a uma variável.

```
let minhaVariavel
console.log(minhaVariavel) //undefined
minhaVariavel = null
console.log(minhaVariavel) //null
```

Exercícios

Exercícios

1) Peça o nome do usuário através do prompt e guarde em uma variável.

2) Peça a idade do usuário através do prompt e guarde em uma variável.

3) Veja qual é o tipo das variáveis de nome e idade.

Conversão entre Tipos

Conversão entre Tipos

Como vimos no exercício anterior, tudo o que o usuário insere em um prompt é uma string!

Podemos fazer a conversão entre esses dois tipos usando métodos fornecidos pelo Javascript!

Número ⇒ String: toString()

```
const idadeNumero = 23
const idadeTexto = idadeNumero.toString()
console.log(typeof idadeNumero)
console.log(typeof idadeTexto)
```

String ⇒ Número: Number()

```
const idadeTexto = "23"
const idadeNumero = Number(idadeTexto)
console.log(typeof idadeTexto)
console.log(typeof idadeNumero)
```

Resumo

Resumo

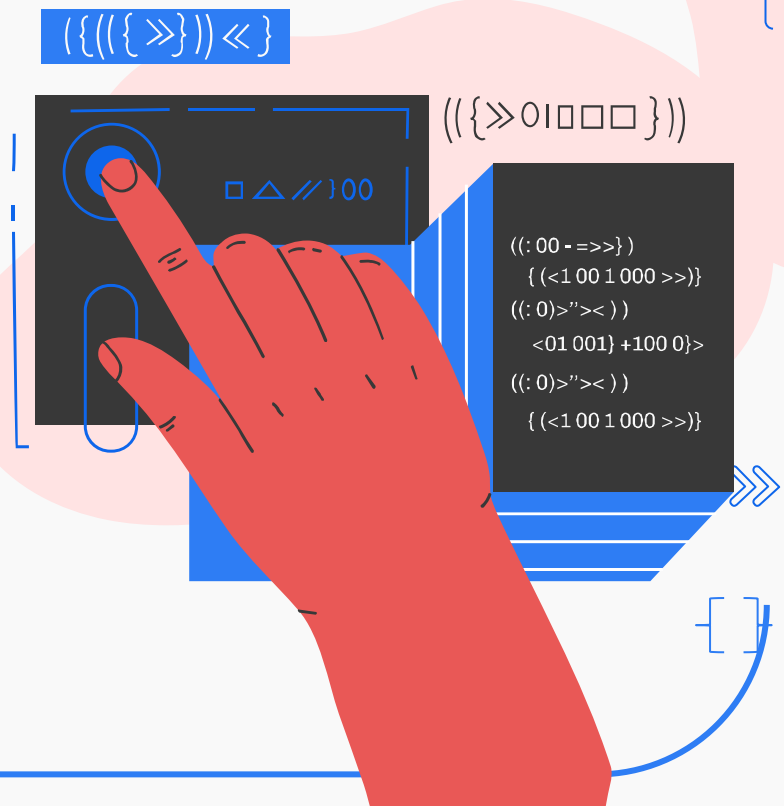
Conseguimos criar comentários usando // ou /**/

console.log(mensagem) gera uma mensagem no console. **prompt()** solicita ao usuário que insira uma informação.
Variáveis declaradas com **const** não mudam enquanto as criadas com **let** podem mudar.
Numbers: representam números.
Strings: representam texto.
Boolean: são tipos que recebem apenas dois valores: verdadeiro (true) ou falso (false).
typeof: permite ver o tipo do valor de uma variável.
Conversões entre tipos:

Número ⇒ String: **toString()**
String ⇒ Número: **Number()**

Obrigado!

E-mail: lgfalves@senacrs.com.br



Professor

Lucas G. F. Alves

Senac