

---

---

# Aula 3

— Desenvolvimento Web Básico —

---

---

# Tags semânticas

# Tags semânticas

HTML adiciona significado ao texto dividindo-o logicamente e identificando o papel que ele desempenha na página

Ao escrever o HTML, utilize uma tag que represente o papel que o conteúdo desempenha na página, não à aparência

Por exemplo, links em um menu são como uma lista

- Deixar ela horizontal já é papel do CSS

# Tags semânticas

- Tags interessantes:

**<blockquote>** Textos citados de outra fonte

**<abbr>** Abreviaturas

**<code>** Códigos de computador

**<strong>** Um texto com impacto, algo que você iria querer destacar (bold)

**<em>** Texto com ênfase, resulta em um texto em itálico

**<sup>** Texto sobrescrito como este

**<sub>** Texto subscrito como este

# Tags semânticas

- Tags interessantes:

**<mark>** Destaca trechos relevantes

**<pre>** Texto pré formatado, mantém espaços em branco e quebras de linha do fonte

**<time>** Textos que indicam horários e datas

# Exercício

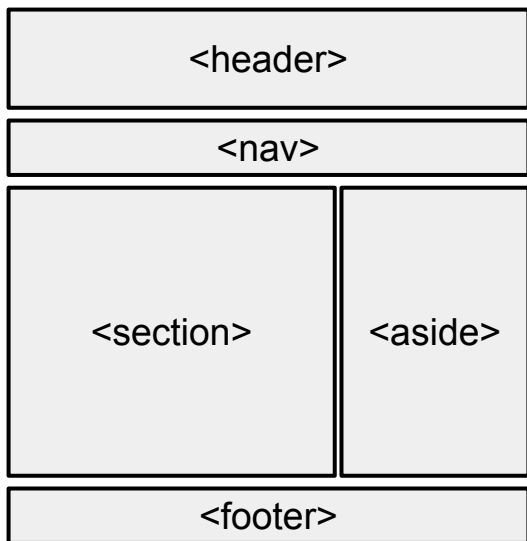
Use as tags apresentadas até agora para separar o conteúdo do arquivo **ex8.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Arquivo sem nenhuma tag</title>
</head>
<body>
  Computadores são máquinas incríveis.
  Como disse Alan Turing, se é que disse mesmo: As máquinas me surpreendem muito.
  Por exemplo, às 22:35hrs estava eu programando o seguinte código:
  // A utility function to swap two elements
  void swap(int* a, int* b)
  {
    int t = *a;
    *a = *b;
    *b = t;
  }

  Quando de repente um Australiano entra pela porta e me lança um:
  The book is on the table?
  Meio perdido, retribuí elevando ao cubo!
  (The book is on the table?)³
</body>
```

# Estrutura básica de uma página HTML

# Tags da estrutura de uma página



**<header>**: Cabeçalho da página ou seção

**<nav>**: Links de navegação

**<section>**: Seções da página

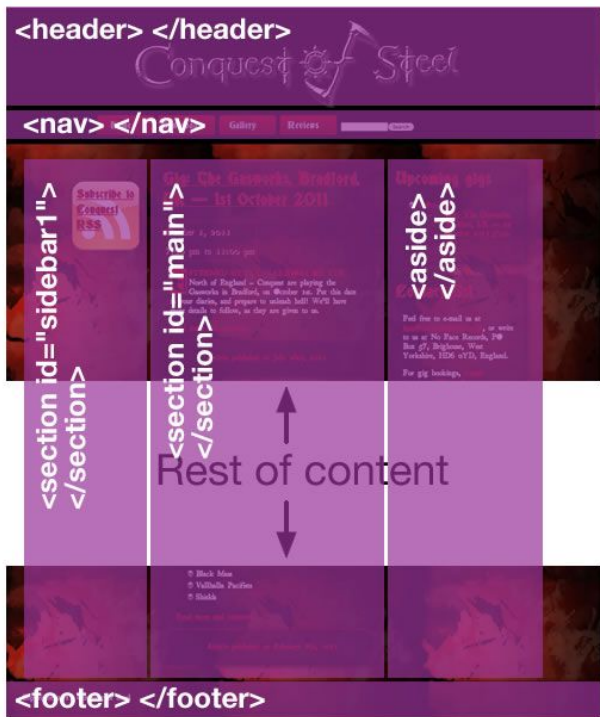
**<article>**: Conteúdo da página

**<aside>**: Conteúdo na barra lateral

**<footer>**: rodapé da página ou seção



# Tags da estrutura de uma página



**<header>**: Cabeçalho da página ou seção

**<nav>**: Links de navegação

**<section>**: Seções da página

**<article>**: Conteúdo da página

**<aside>**: Conteúdo na barra lateral

**<footer>**: rodapé da página ou seção

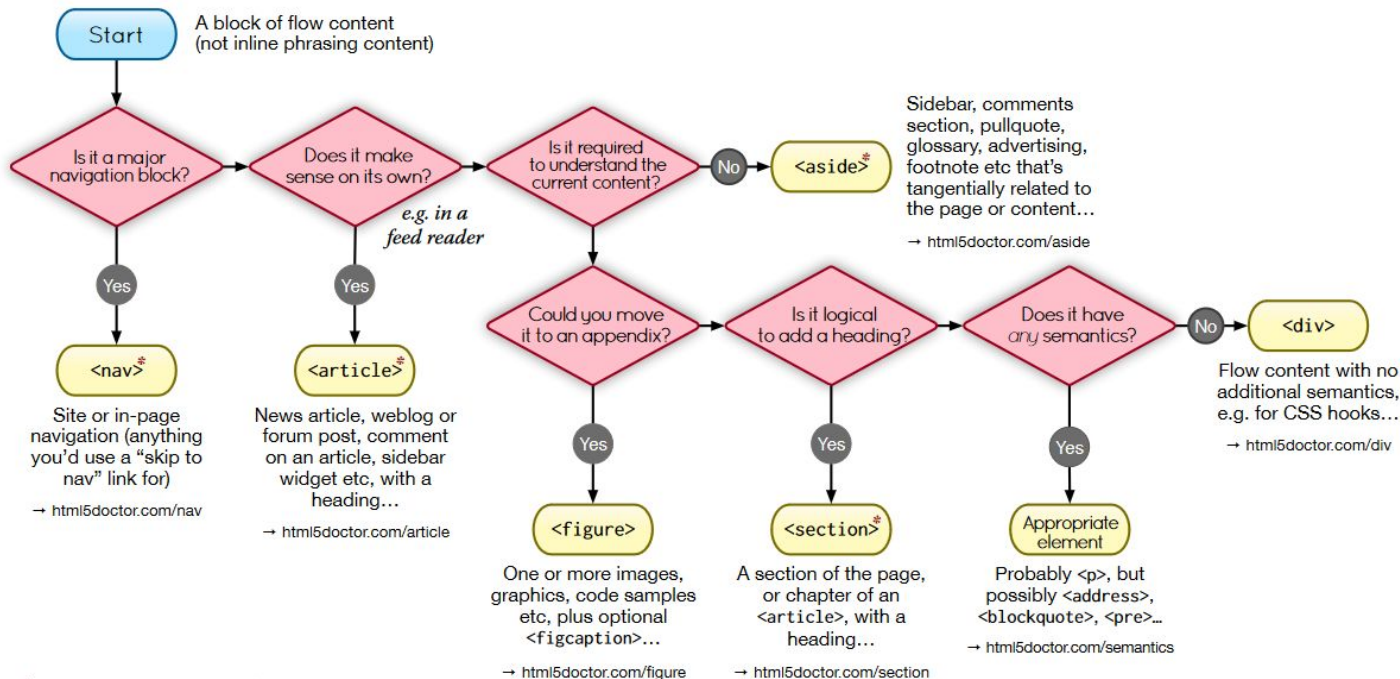


Doctor

# HTML5 Element Flowchart

Sectioning content elements and friends

By @riddle & @boblet  
[www.html5doctor.com](http://www.html5doctor.com)



\* Sectioning content element

These four elements (and their headings) are used by HTML5's outlining algorithm to make the document's outline  
→ [html5doctor.com/outline](http://html5doctor.com/outline)

2011-07-22 v1.5  
For more information:  
[www.html5doctor.com/semantics](http://www.html5doctor.com/semantics)

# Exemplo

Entre na página <https://pt.wikipedia.org> e vamos tentar encontrar algumas de suas tags:

- header
- nav
- section
- aside
- footer

# Símbolos

# Entidades de símbolo HTML

Existe uma infinidade de símbolos matemáticos, monetários e técnicos que podemos querer usar em nossas páginas.

Para adicionar esses símbolos, podemos usar seu nome ou número:

©	&#169;	&copy;
®	&#174;	&reg;
™	&#8482;	&trade;
≠	&#8800;	&ne;
£	&#163;	&pound;
∃	&#8707;	&exist;

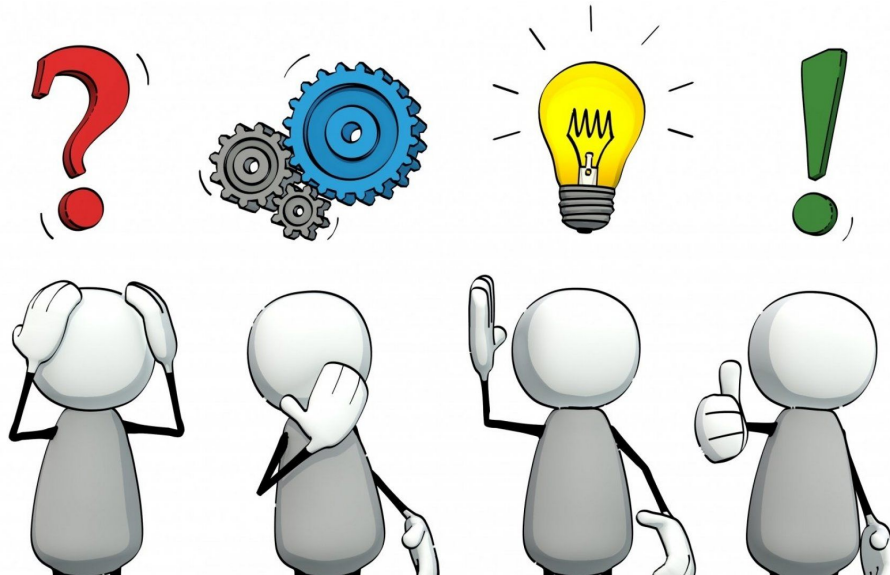
# Exemplo

Vamos colocar o seguinte texto em uma página Web:

While most countries recognize the ® designation, the **trademark symbol** "™" is mostly a product of the English common-law system.

# Exercício

Busque o símbolo HTML que achar mais curioso/interessante/útil e vamos fazer uma lista!



# Videos



# Vídeos

Podemos usar a tag **<video>** para inserir um vídeo em uma página

```
<video width="320" height="240" controls>  
  <source src="dogs.mp4" type="video/mp4">  
  <source src="dogs.webm" type="video/webm">  
  Seu navegador não suporta vídeos!  
</video>
```

# Vídeos

Podemos usar a tag **<video>** para inserir um vídeo em uma página

```
<video width="320" height="240" controls>  
  <source src="dogs.mp4" type="video/mp4">  
  <source src="dogs.webm" type="video/webm">  
  Seu navegador não suporta vídeos!  
</video>
```

- Os atributos **width** e **height** definem, respectivamente, a largura e a altura do vídeo.
- O atributo **controls** adiciona controles de vídeo, como reproduzir, pausar e volume.

# Vídeos

Podemos usar a tag **<video>** para inserir um vídeo em uma página

```
<video width="320" height="240" controls>  
  <source src="dogs.mp4" type="video/mp4">  
  <source src="dogs.webm" type="video/webm">  
  Seu navegador não suporta vídeos!  
</video>
```

- O elemento **source** permite especificar arquivos de vídeo alternativos que o navegador pode escolher. O navegador usará o primeiro formato reconhecido.

# Vídeos

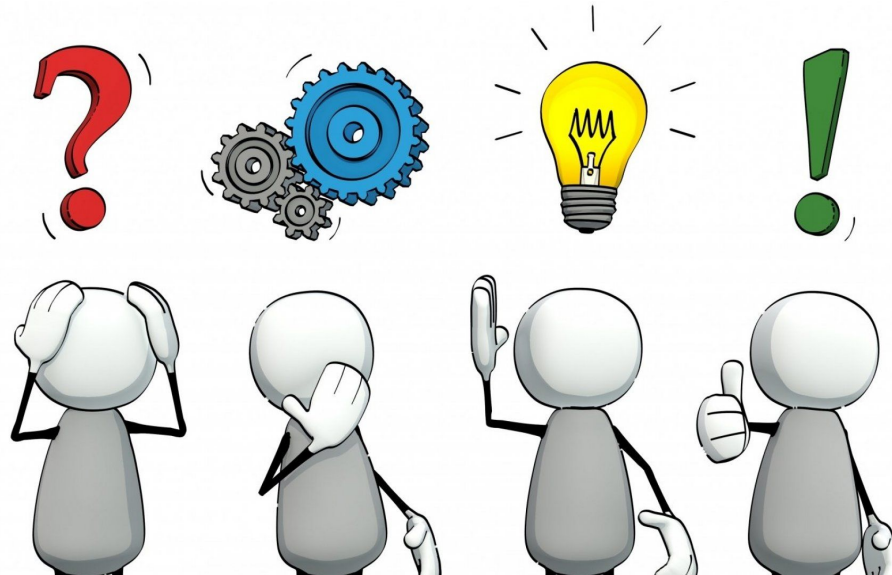
Podemos usar a tag **<video>** para inserir um vídeo em uma página

```
<video width="320" height="240" controls>  
  <source src="dogs.mp4" type="video/mp4">  
  <source src="dogs.webm" type="video/webm">  
  Seu navegador não suporta vídeos!  
</video>
```

- O texto entre as tags **<video>** e **</video>** só será exibido em navegadores que não suportem elementos de vídeo.

# Exemplo

Baixe a pasta **Vídeos** do BlackBoard



## Exemplo 2

Vamos adicionar um vídeo do Youtube a uma página Web!

Vídeo incorporado



```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/
pzzXm2Q0YWg" title="YouTube video
player" frameborder="0"
allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture; web-
share" allowfullscreen></iframe>
```



Começar em 0:00

### OPÇÕES DE INCORPORAÇÃO



Mostrar controles do player.



Ativar o modo de privacidade aprimorada

Copiar

# Boas práticas HTML

# Boas práticas

- **Sempre declare o tipo de documento:** sempre declare o tipo de documento como a primeira linha do seu documento. O tipo de documento correto para HTML5 é  
`<!DOCTYPE html>`
- **Use nomes de elementos em minúsculas:** o HTML permite misturar letras maiúsculas e minúsculas nos nomes dos elementos. No entanto, é recomendado o uso de nomes de elementos em letras minúsculas.

```
<!-- Use assim -->  
<body> <p> <img>  
  
<!-- Não assim -->  
<BODY> <P> <IMg>
```



# Boas práticas

- **Feche todos os elementos HTML:** em HTML, você não precisa fechar todos os elementos (por exemplo, o elemento `<p>` ). No entanto, é recomendado fechar todos os elementos HTML.

```
<!-- Use assim -->
<p>Texto</p>
<li>Item</li>

<!-- Não assim -->
<p>Texto
<li>Item
```

# Boas práticas

- **Use nomes de atributos em minúsculas:** o HTML permite misturar letras maiúsculas e minúsculas em nomes de atributos. No entanto, é recomendado o uso de nomes de atributos em letras minúsculas

```
<!-- Use assim -->  
<img href=      >  
  
<!-- Não assim -->  
<img HREF=
```

# Boas práticas

- **Sempre citar valores de atributo:** o HTML permite valores de atributo sem aspas. No entanto, é recomendado citar os valores dos atributos

```
<!-- Use assim -->  
<p id="valor"></p>
```

```
<!-- Não assim -->  
<p id=valor></p>
```

# Boas práticas

- **Sempre especifique alt, largura e altura para imagens:** sempre especifique o atributo alt para imagens.
  - Este atributo é importante se a imagem por algum motivo não puder ser exibida.
  - Além disso, sempre defina a largura e a altura das imagens. Isso reduz a oscilação, porque o navegador pode reservar espaço para a imagem antes de carregá-la.

```

```

# Boas práticas

- **Espaços e sinais de igualdade:** o HTML permite espaços ao redor dos sinais de igual. Mas sem espaço é mais fácil de ler e agrupa as entidades melhor.

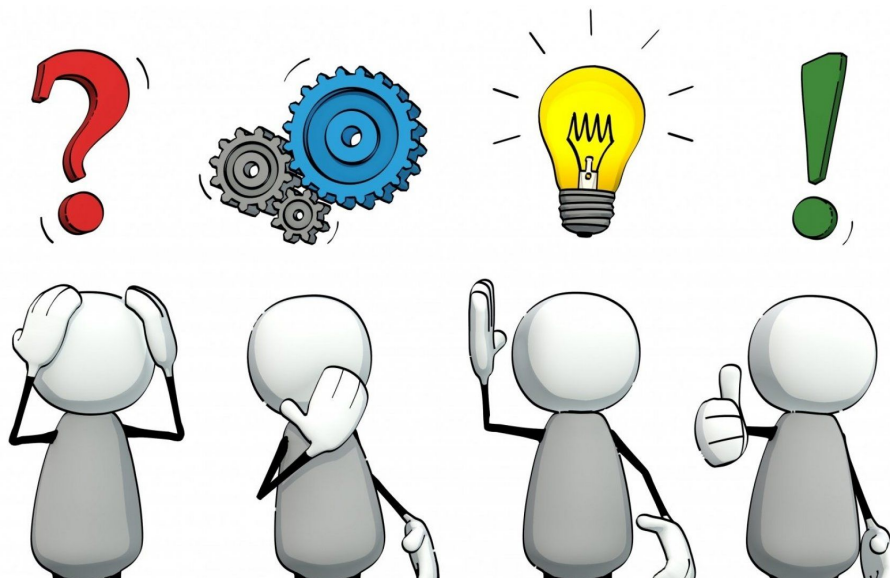
```
<!-- Use assim -->  
<a href="https://www.google.com/">Google</a>  
  
<!-- Não assim -->  
<a href = "https://www.up.edu.br/">Google</a>
```

# Boas práticas

- **Nunca pule o elemento <title>:**
  - O conteúdo do título de uma página é muito importante para a otimização de mecanismo de busca (SEO)!
    - O título da página é usado pelos algoritmos do mecanismo de pesquisa para decidir a ordem ao listar as páginas nos resultados da pesquisa.
  - Define um título na barra de ferramentas do navegador.
  - Define um título para a página quando ela é adicionada aos favoritos.
  - Define um título para a página nos resultados do mecanismo de pesquisa.

# Exercício

Use as boas práticas mencionadas para melhorar o código **boasPráticas.html**



# Validação de documentos HTML



# Validação

Como sabemos, o HTML é padronizado pelo Wide Web Consortium (W3C).

Assim, existe um conjunto de regras que devem ser seguidas na construção de páginas Web.

Portanto, o W3C disponibiliza uma validação das marcações presentes em documentos HTML !

- A validação do W3C é o processo de verificação do código de um site para determinar se ele segue os padrões adequados de formatação.

# Validação

A validação da marcação é um passo importante para garantir a qualidade técnica das páginas da web.

Se você não conseguir validar as páginas do seu site com base nos padrões W3C , seu site provavelmente terá **erros** ou apresentará um uso **ruim do tráfego** devido à **formatação**, **legibilidade** ou **velocidade de carregamento** da página inadequadas.

A ferramenta de validação HTML do W3C pode ser acessada em:

- <https://validator.w3.org/>

# Versionamento de Códigos

# Versionamento de códigos


Todo projeto de desenvolvimento é feito por etapas, sendo que as funcionalidades são incrementadas aos poucos.

Consiste em estratégias para gerenciar diferentes versões de um código, de um sistema ou de um modelo.

Forma de administrar mudanças feitas e garantir mais segurança na transição de uma versão para outra.


- Git é um sistema de controle de versão distribuído de código aberto e gratuito
- GitHub e GitLab são plataformas de hospedagem de código-fonte. Permitem a contribuição em projetos privados ou abertos


# Git - Instalação

 **git** --everything-is-local

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.


Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.






### About

The advantages of Git compared to other source control systems.




### Documentation

Command reference pages, Pro Git book content, videos and other material.




### Downloads

GUI clients and binary releases for all major platforms.





### Community


Get involved! Bug reporting, mailing list, chat, development and more.




**Pro Git** by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

 **Windows GUIs**

 **Tarballs**

 **Mac Build**


 **Source Code**

Latest source Release

## 2.40.0

[Release Notes \(2023-03-12\)](#)

[Download for Windows](#)



# Git - Configuração

Para conectarmos nosso Git a um repositório online, precisamos configurar nossas credenciais nesse repositório utilizando:

- `git config --global user.name "nome de usuario"`
- `git config --global user.email "seu_email@algo.com"`



# Git - Clonar repositório remoto

Git clone é usado para copiar um repositório Git existente para um novo diretório local.

- `git clone Endereço_do_repositorio`
- Exemplo: **git clone** <https://github.com/Aval7n/WebBasico.git>

# Git - Enviar modificações

Após criarmos um repositório remoto e o clonarmos, podemos realizar alterações em seus arquivos.

Para enviarmos nossas alterações, podemos usar os seguintes comandos:

- `git add .`
- `git commit -m "Descrição da alteração"`
- `git pull`
- `git push`



# Git - Enviar modificações

Após criarmos um repositório remoto e o clonarmos, podemos realizar alterações em seus arquivos.

Para enviarmos nossas alterações, podemos usar os seguintes comandos:

- `git add .`
  - Seleciona os arquivos modificados que devem ser enviados para o servidor. O ponto indica todos os arquivos alterados.
- `git commit -m "Descrição da alteração"`
- `git pull`
- `git push`

# Git - Enviar modificações

Após criarmos um repositório remoto e o clonarmos, podemos realizar alterações em seus arquivos.

Para enviarmos nossas alterações, podemos usar os seguintes comandos:

- `git add .`
- `git commit -m "Descrição da alteração"`
  - Define uma mensagem para identificar as alterações realizadas.
- `git pull`
- `git push`

# Git - Enviar modificações

Após criarmos um repositório remoto e o clonarmos, podemos realizar alterações em seus arquivos.

Para enviarmos nossas alterações, podemos usar os seguintes comandos:

- `git add .`
- `git commit -m "Descrição da alteração"`
- `git pull`
  - Obtém atualizações do servidor.
- `git push`

# Git - Enviar modificações

Após criarmos um repositório remoto e o clonarmos, podemos realizar alterações em seus arquivos.

Para enviarmos nossas alterações, podemos usar os seguintes comandos:

- `git add .`
- `git commit -m "Descrição da alteração"`
- `git pull`
- `git push`
  - Envia os arquivos alterados e selecionados para o servidor remoto.

# Exemplo - Criação de repositório

Vamos criar um repositório!

1. Crie uma conta no [github.com](https://github.com)!
2. Crie um repositório com um arquivo inicial README.md



☒  **Public**  
Anyone on the internet can see

☐  **Private**  
You choose who can see and c

**Initialize this repository with:**  
Skip this step if you're importing ar

☒ **Add a README file**  
This is where you can write a long de

**Add .gitignore**

Choose which files not to track from a lis

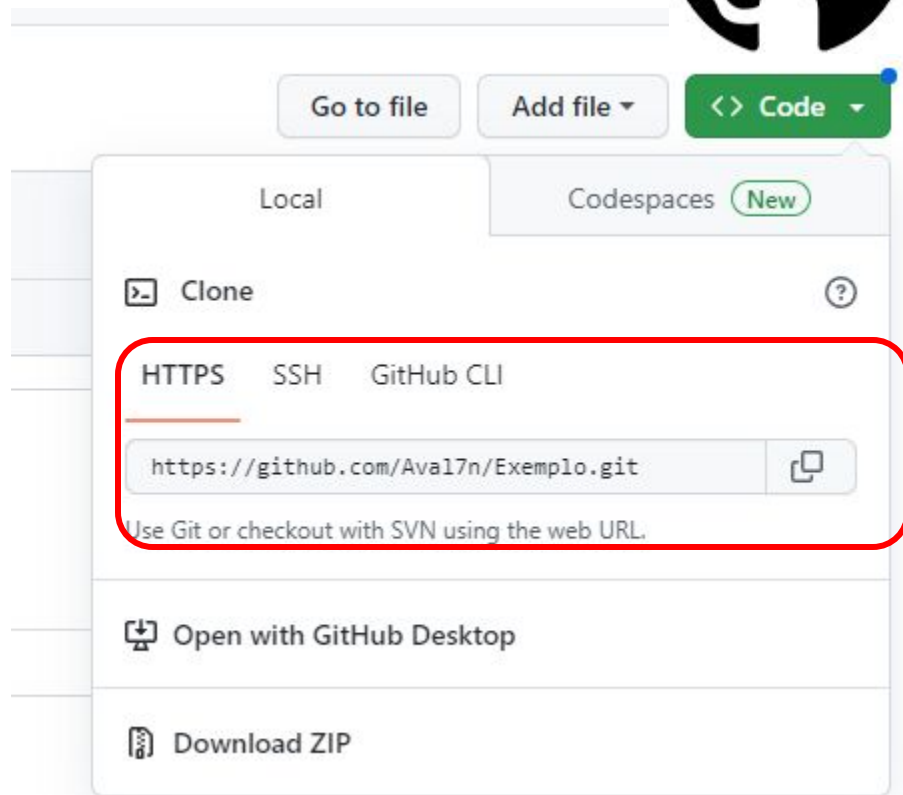
.gitignore template: None ▼

# Exemplo - Criação de repositório



Vamos criar um repositório!

1. Crie uma conta no [github.com](https://github.com)!
2. Crie um repositório com um arquivo inicial README.md
3. Clone o repositório:
  - a. `git clone seu_link_aqui`



# Exemplo - Edição de conteúdo

Agora vamos editar alguns arquivos!

1. Dentro da pasta clonada, crie um arquivo **index.html**
2. Crie a seguinte página nesse arquivo:

## Página de teste

Esta página serve apenas para fazermos testes com o GIT.

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6



## Exemplo - Envio dos resultados

Por fim, vamos salvar os arquivos alterados:

1. `git add .`
2. `git commit -m "Descrição da alteração"`
3. `git push`



# Passos

1. Criar um repositório no servidor (Github, Gitlab, ...)
2. Fazer um **clone** do repositório na máquina local
3. Realizar as alterações necessárias
4. Atualizar seus arquivos com as alterações que o servidor pode ter sofrido (**pull**)
5. Enviar as alterações para o servidor (**push**)

# Anotações

# Anotações

Podemos realizar um envio ao servidor (**commit**) a cada alteração realizada.

Em determinado momento, quando um módulo ou o projeto está estável, damos uma **tag** para essa versão.

A última versão com tag existente é chamada de versão **estável**

- **git tag** -a NomeDaTag -m "Descrição da tag"
- Exemplo: **git tag** -a v1.4 -m "my version 1.4"



## Exemplo - Criando tags

1. Crie mais duas alterações e commits.
2. Adicione uma tag utilizando: **git tag -a** NomeDaTag **-m** "Descrição da tag"
3. Liste seus commits usando: **git log**
4. Liste as tags utilizando: **git tag**
5. Mostre a descrição da tag criada usando: **git show** NomeDaTag
6. Envie a tag para o servidor usando: **git push --tags**

# Restauração de versão anterior

Caso algo dê errado durante o desenvolvimento, podemos **voltar a uma versão anterior** (similar a um rollback em um banco de dados)

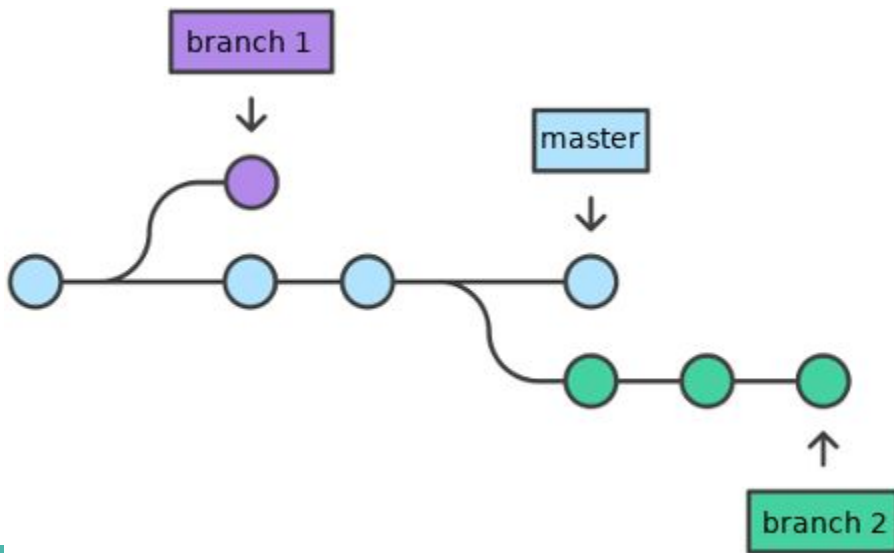


# Branches

# Branches

Branches são ramificações do seu projeto, que podem ser alteradas sem que a base dele seja alterada.

Como se existissem diversas “versões” do mesmo projeto simultaneamente



# Criando branches

- Podemos criar uma branch usando
  - **git branch** NomeDaBranch
- Podemos acessar uma branch usando
  - **git checkout** NomeDaBranch
- Para fazer as duas coisas simultaneamente:
  - **git checkout -b** NomeDaBranch
- Para verificar o branch atual
  - **git status**
  - **git branch**
- Para listar os branches
  - **git branch -a**



# Exemplo

Vamos criar um branch!

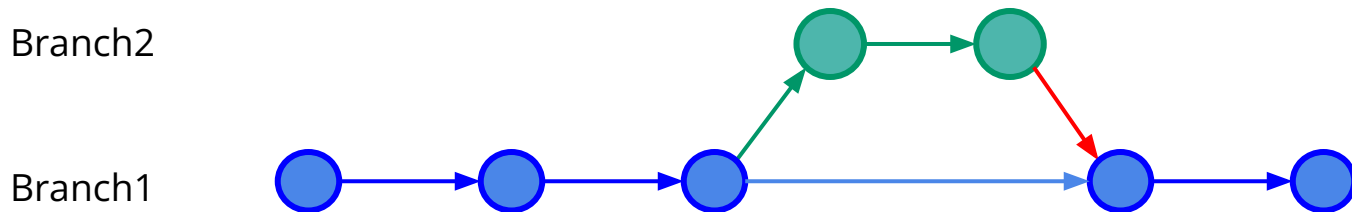
1. Crie a branch **NovoBranch**
  - a. **git branch** NovoBranch
2. Acesse essa branch
  - a. **git checkout** NovoBranch
3. Crie um arquivo chamado novo.html
4. Envie ele para o github
  - a. `git add .`
  - b. `git commit -m "Descrição..."`
  - c. `git push`
    - i. **git push --set-upstream** origin NovoBranch



# Unindo branches

Para unir dois branches, devemos ir até o branch de destino das alterações (**Git checkout** Branch1) e executar:

- **git merge** Branch2
- **git push**



# Branches - Projetos pessoais

Em projetos pessoais, é comum utilizarmos apenas um branch (master) para todos os commits.

O Master/Main é o branch principal de um projeto.

- Contém o código final da nossa aplicação.
- Contém a versão estável do projeto.

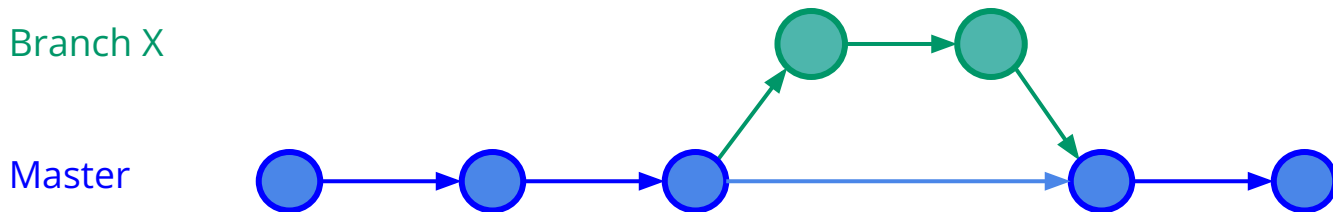
Master



# Branches - - Projetos reais

Em um projeto real, diversos desenvolvedores realizam alterações simultaneamente, que precisam ser controladas:

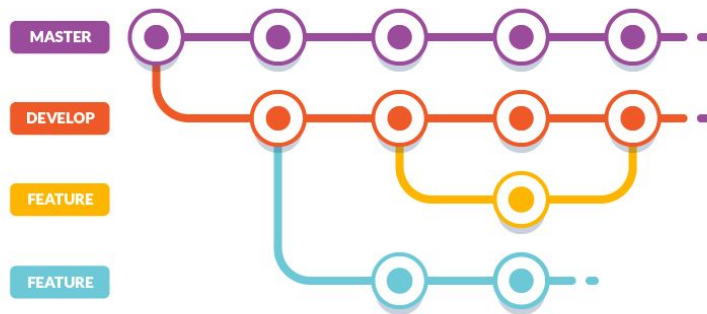
- Implementação de novas funcionalidades
- Correção de falhas
- Lançamento de versões
- ...



# Git Flow

Nesse contexto, o Git Flow é um modelo/estratégia utilizado na **organização** do versionamento de códigos.

## Objetivo: Melhorar a organização das branches



# Git Flow - Branches

O Git Flow é organizado em branches principais e de suporte

## **Branches principais**

- Develop
- Master

## **Branches de suporte**

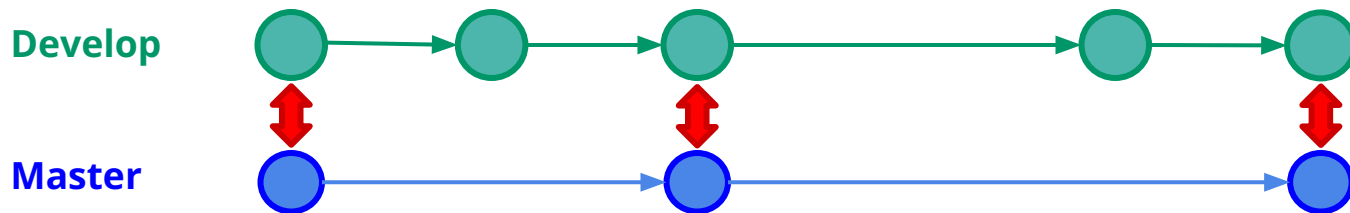
- Feature
- Release
- Hotfix

# Git Flow - Branches principais

Branches principais **duram para sempre**, registrando todo o histórico do projeto.

**Master:** armazena o histórico do lançamento oficial

**Develop:** serve como uma ramificação de integração para recursos



# Git Flow - Branches de suporte

Branches de suporte são **temporárias**, durando até realizar **merge** com as branches principais

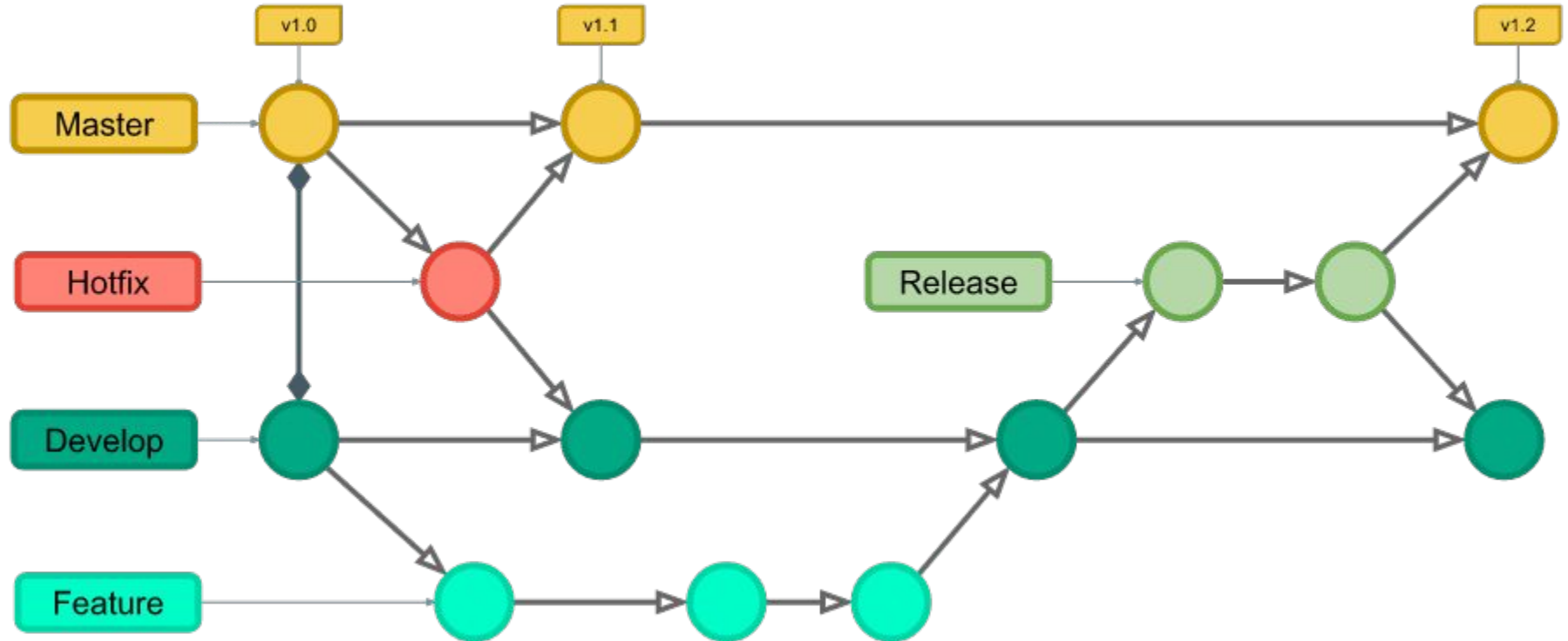
**Feature:** utilizada para o desenvolvimento de uma funcionalidade específica. São criadas sempre a partir da **Develop** e removidas após o merge com a **Develop**.

**Hotfix:** Criada a partir da **Master** para correções imediatas no sistema em produção. Quando é concluída, realiza merge com a **Master** e a **Develop** e é excluída.

**Release:** Serve como ponte e ambiente de homologação para o merge entre a **Develop** e a **Master**. É criada a partir da **Develop** e removida após o merge com a **Master**. Caso um bug seja encontrado e corrigido, também deve ser sincronizada com a **Develop**.

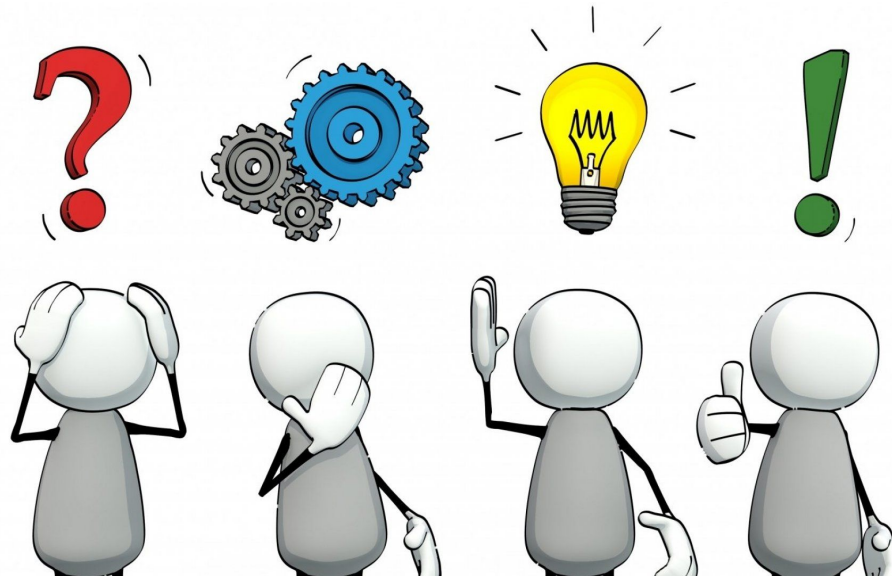


# Git Flow - Visão Geral



# Exercícios

Resolva a **lista de exercícios** presente no BlackBoard



# Conteúdo

HTML: semântica HTML. Versionamento de códigos.