

Módulo Git e Github

Ciclo Formativo Intermediário – PretaLab

Acordos importantes!!!



Aqui é um ambiente seguro;

Não existe pergunta boba;

Pausas são importantes;

Somos um coletivo;

Bebam água;

Feedbacks são tudo de bom!

Objetivo Principal

Fundamentar conhecimento teórico e prático sobre Git e Github

Semana 1:

- Explorar as habilidades fundamentais para lidar com sistemas de versionamento de código.
- Introduzir o conceito de controle de versão e os comandos essenciais do Git.

Aula 1 | GIT - Introdução

- O que é controle de versão?
- História do Git
- Instalando e configurando o Git
- Inicializando repositórios locais
- Ciclo de vida dos arquivos no Git (add, commit, status, log, diff).

O que é Git

Git é um serviço gratuito e open source(fonte aberta) sistema de controle de versão distribuído que lida com qualquer tipo de projeto – pequeno ou grande – com velocidade e eficiência.

O que é controle de versão?

Um projeto pode ser clonado e ter versões diferentes, feitas por pessoas diferentes, que são atualizadas através do controle proporcionado pelo Git.

Cada desenvolvedor pode fazer as alterações localmente na sua máquina e mesclar(juntar) essas versões diferentes em uma matriz(master/main) do projeto, feito remotamente.



Porquê usar controle de versão?

Proporcionar o trabalho conjunto e colaborativo ao mesmo tempo, evitando problemas nas mudanças de um projeto.

Isso evita perda de código e múltiplas versões.



História

- Criado em 2005 por Linus Torvalds (criador do Linux) depois de uma treta.
- Linus passou a criar sua própria ferramenta para gerenciar seus projetos
 - **Velocidade**
 - **Projeto simples**
 - **Forte suporte para desenvolvimento não-linear (milhares de ramos paralelos)**
 - **Completamente distribuído**
 - **Capaz de lidar com projetos grandes como o núcleo o Linux com eficiência (velocidade e tamanho dos dados)**

Mão na massa

Instalação do Git:

<https://git-scm.com/downloads>



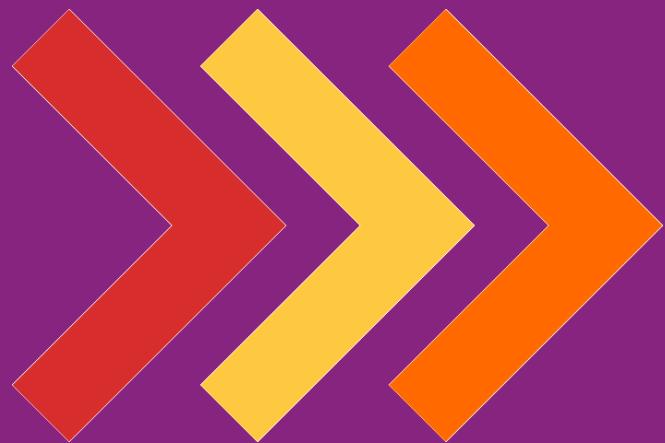
Sistemas operacionais:

- Linux:
 - <https://www.atlassian.com/br/git/tutorials/install-git#linux>
- Windows:
 - <https://www.atlassian.com/br/git/tutorials/install-git#windows>
- MacOs
 - <https://www.atlassian.com/br/git/tutorials/install-git#mac-os-x>

Instalando e configurando

- git --version
- git config --global user.name "Seu Nome"
- git config --global user.email "seuemail@email.com"

Importante!



- **Linux/MacOs**
 - Geralmente o Bash já vem instalado como padrão
- **Windows**
 - O terminal padrão é o Powershell (prompt de comando)
 - Após instalar o Git, busque o Git Bash na sua área de trabalho(caso tenha colocado) ou faça uma busca

Intervalo!



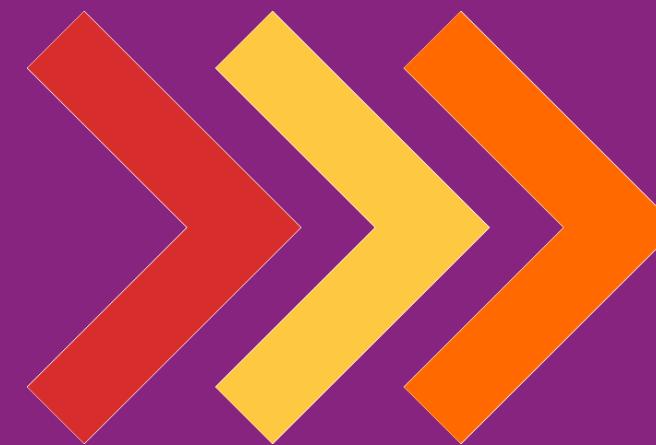
Criando repositórios

Estrutura de um projeto Git



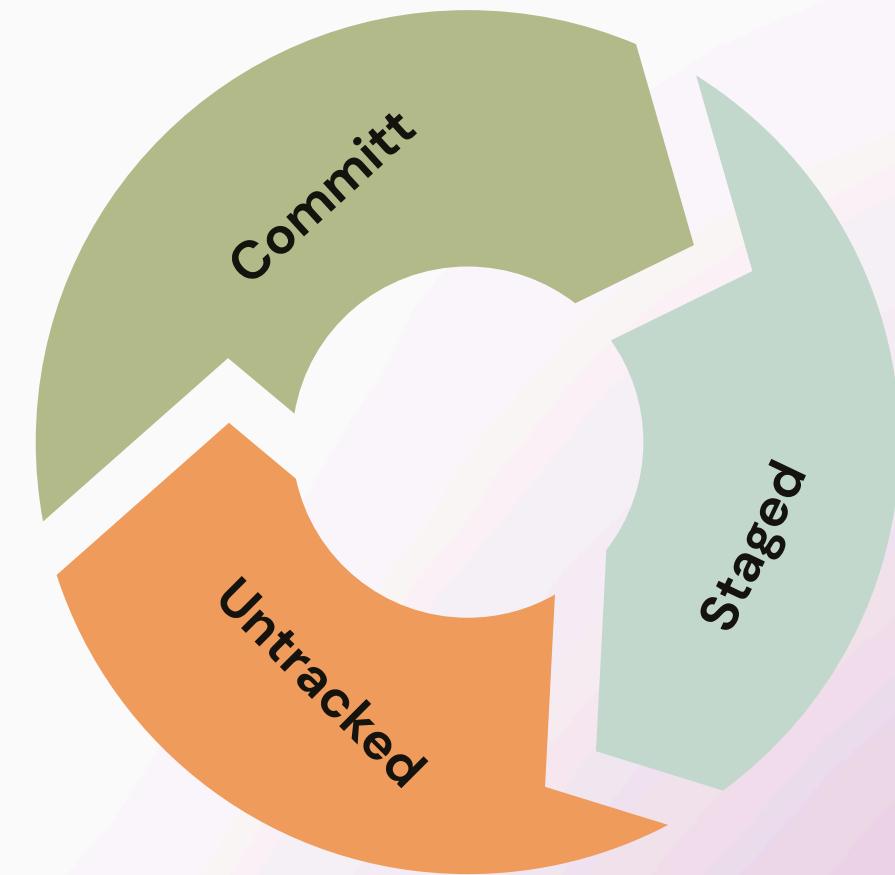
- **Inicialização:**
 - cd ~Desktop ou cd Desktop
 - cd ~Area de trabalho ou cd ~Area de trabalho
 - ls ~ (para ver a pasta do seu usuário)
 - mkdir pretalab (criar)
 - git init
- **Estrutura .git**
 - pasta oculta (coração do git)
 - guarda todas as informações de versionamento do repositório.
 - contém o histórico de commits, branches, tags, configurações, etc
 - HEAD → Aponta para o commit/branch atual.
 - config → Arquivo de configurações do repositório
 - objects/ → Onde o Git guarda os objetos do histórico (commits, blobs, árvores)
 - refs/ → Referências para branches e tags
 - logs/ → Registros de ações feitas no repositório

Importante!



- **Não alterar os arquivos manualmente**
- **Não apagar a pasta**
 - perde o histórico da branch
 - O repositório se torna uma pasta comum.

Ciclo de vida dos arquivos



Untracked → Staged → Committed

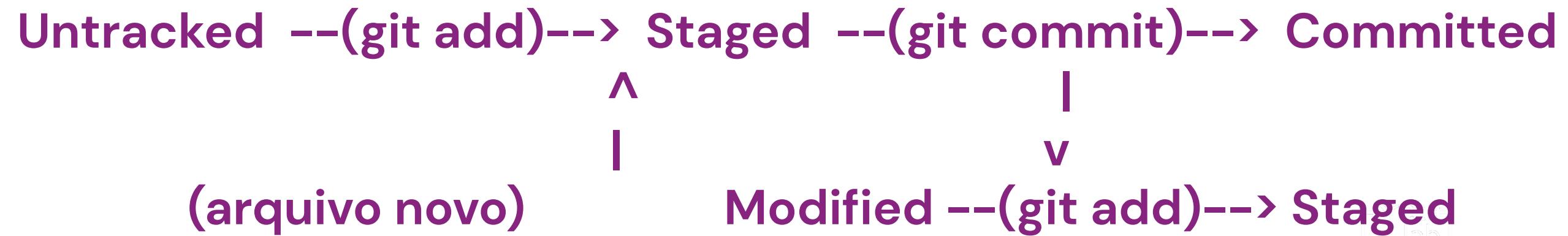
Comandos básicos:

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

- **git status:**
 - mostra o status dos arquivos, se foram adicionados, commitados
- **git add .**
 - adiciona todos os arquivos
- **git add nome-do-arquivo**
 - adiciona uma arquivo por vez
- **git commit -m "mensagem"**
 - inclui uma mensagem com a descrição do que foi realizado
- **git log**
 - histórico do commit da branch atual com id, data
- **git diff**
 - mostra as diferenças dos arquivos

Status dos arquivos

- **Untracked (não rastreado)**
 - quando o arquivo foi criado, mas não foi adicionado no versionamento.
- **Staged (indexado / pronto para commit)**
 - pronto para commit
- **Committed (comitado)**
- **Modified (indexado / pronto para commit)**
 - após a criação ou modificação de um arquivo, ele é adicionado com git add



Aula 2:

- Conexão com GitHub e desfazendo mudanças

O que é o Github



- O Github é uma plataforma remota de hospedagem de código e arquivos.
- Integrado ao Git, possibilita o versionamento de código de maneira colaborativa.

Mãos na massa!!!



Chave SSH

- Segurança
- Não precisa incluir usuário e senha ao clonar repositórios
- Chave privada fica no computador
- Chave Pública fica no Github
- Verificar se já existe a chave:
 - macOS/Linux
 - ls -al ~/.ssh
 - Windows
 - ls ~/.ssh
- Saida
 - id_ed25519 e id_ed25519.pub

Instalação

- ssh-keygen -t ed25519 -C "seu-email@example.com"
 - salva no caminho: ~/.ssh/id_ed25519
- Gera dois arquivos:
 - id_ed25519 → chave privada (fica só no seu PC, nunca compartilhe).
 - id_ed25519.pub → chave pública (essa sim vai para o GitHub).
- Ativar ssh e carregar a chave
 - Linux/MacOS/Windows
 - eval "\$(ssh-agent -s)"
 - ssh-add ~/.ssh/id_ed25519

Intervalo!



Qual são as minhas chaves?

Para verificar as chaves, use os comandos abaixo:

- **Para todos os sistemas operacionais, use os comandos:**
 - **Chave Pública:**
 - cat ~/.ssh/id_ed25519.pub
 - **Chave Privada:**
 - cat ~/.ssh/id_ed25519

Ativar no Github

- GitHub → Settings → SSH and GPG keys → New SSH key
- Dê um nome/Título
 - (ex: Meu PC)
- Cole o conteúdo da chave pública (id_ed25519.pub) que aparece após colocar o comando da página anterior
- Salve
- Testando
 - ssh -T git@github.com

Criando repositórios no Github

- **Criar conta**
- **Criar novo repositório**
 - Vá no GitHub → New Repository
 - Escolha um nome (ex: pretalab-git-github)
- **Vai gerar uma URL remota**
 - HTTPS → <https://github.com/seu-usuario/pretalab-git-github.git>
 - SSH → `git@github.com:seu-usuario/pretalab-git-github.git`
- **Adicionar repositório local (com chave ssh)**
 - `git remote add origin git@github.com:seu-usuario/pretalab-git-github.git`
- **Verifica se funcionou**
 - `git remote -v`

Revisando os comandos básicos

Criando o arquivo Readme.md

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

+ Comando necessário para envio do arquivo do Git para GitHub

- **git push -u origin Main**(primeira vez que vai usar o push em um novo repositório)
- **git push origin main ou git push** (depois do primeiro envio)

Desfazendo mudanças

| Comando | Atua em... | O que acontece com as mudanças locais | O que acontece com os commits existentes |
|-------------------------------------|--|--|--|
| git restore arquivo | Working Directory | As alterações do arquivo são descartadas | Nenhuma mudança — commits nunca são afetados |
| git restore --staged arquivo | Staging Area (Index) | As alterações saem do staged, mas continuam no working directory | Nenhuma mudança — commits nunca são afetados |
| git reset --soft HEAD~1 | Histórico de commits, staging area e working directory | Retira o arquivo do commit e volta a staged, sem alterações | Desfaz o último commit |



CheckList Semana 1

| | |
|---|--|
| Git instalado | Repositório remoto criado |
| Uso do terminal bash | Conexão repositório local e repositório remoto feita |
| Chave SSH Instalada | Arquivo Readme.md criado no repositório local |
| Chave Pública do SHH ativada no Github | Primeiras linhas escritas no Readme.md |
| Repositório local inicializado | Comandos básicos usados |
| Conta no GitHub | Enviadas alterações do arquivo local para o repositório remoto com git push |

Revisão Semana 1

Comandos Básicos

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

Comando para enviar alterações para o GitHub:

- git push -u origin Main
- git push origin main ou git push

Comandos para desfazer mudanças locais:

- git restore nome-do-arquivo
- git restore --staged nome-do-arquivo
- git reset --soft HEAD~1

Semana 2:

- **Git Avançado:**
 - Branches
 - Merge
 - Boas Práticas
 - Auto gestão
 - Projeto Prático

Aula 1

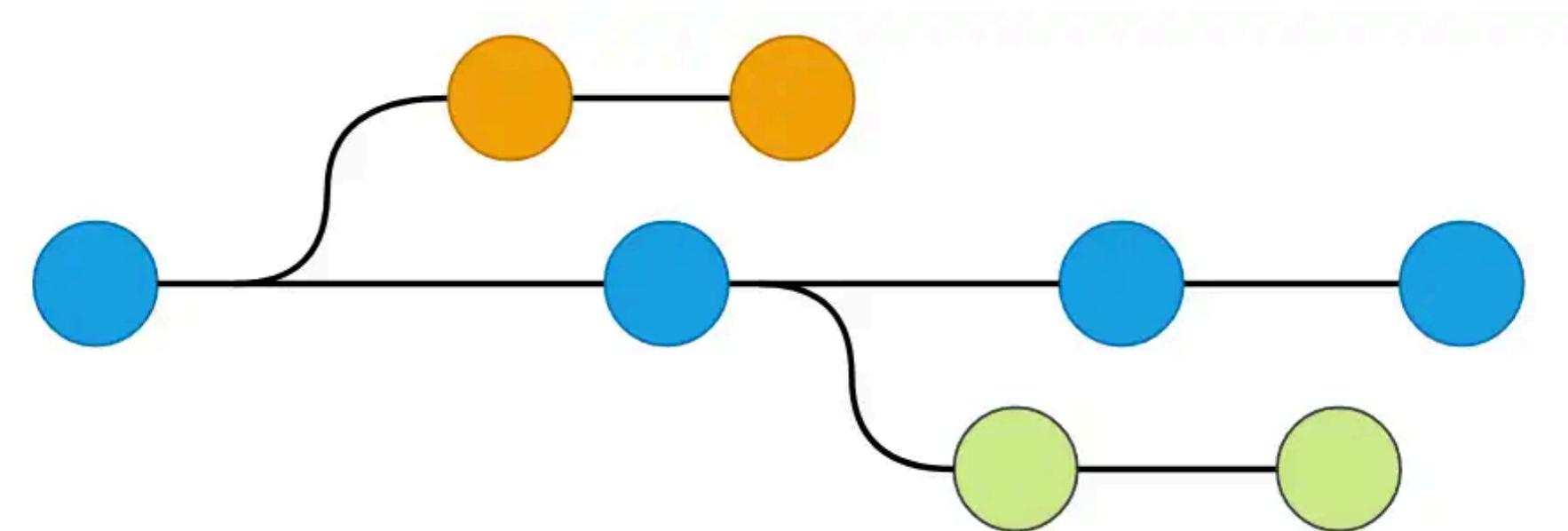
Fundamentos de Branches e Fluxo

O que são branches e porquê usar?

A branch é uma ramificação do nosso projeto.

A primeira branch do projeto é a **main** ou **master**(defasado).

Podemos criar *branches* para adicionar novas funcionalidades ou fazer correções de bug no projeto.



*Cada linha é um ramo(branch) do projeto.
Cada círculo indica um commit nas branches*

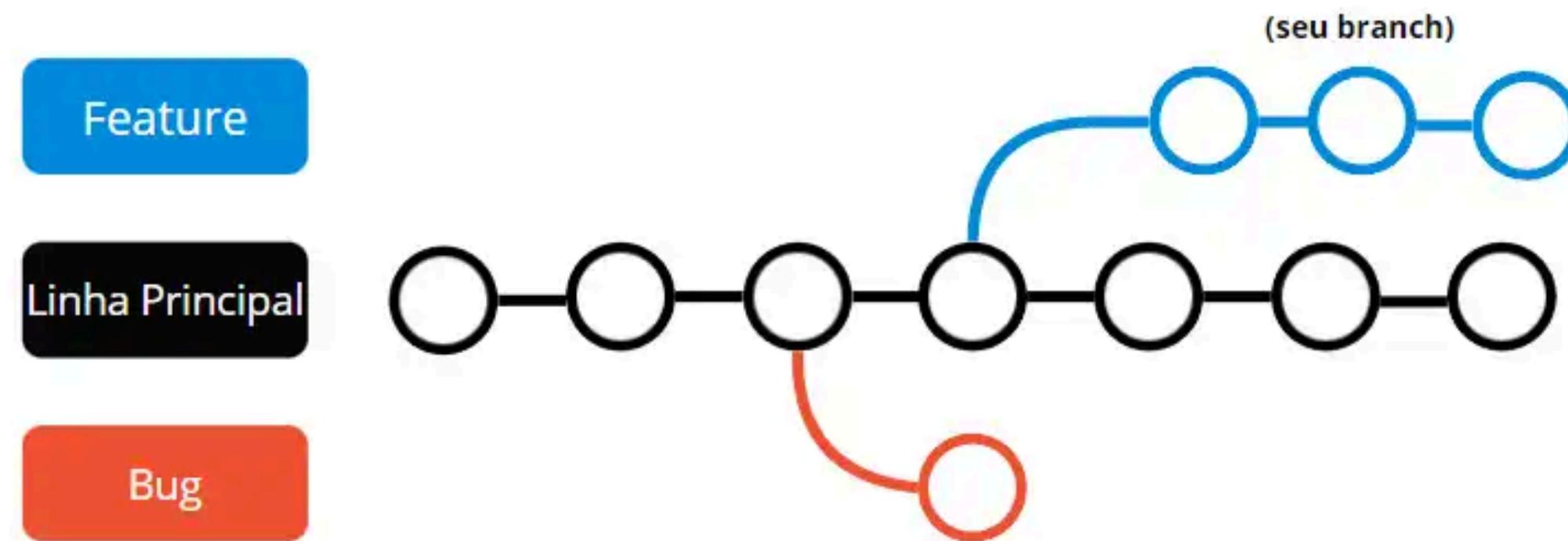


Imagen: Site Dio

Cada commit é um período no tempo que indica quando a alteração foi realizada. O propósito final é que as branches paralelas se encontrem com a branch main.

Comandos para branch local

| Comando | Funcão |
|---------------------------------------|---|
| git checkout -b nome-da-branch | Criar uma nova branch |
| git branch -D nome-da-branch | Força a deleção da branch |
| git branch -d nome-da-branch | Exclusão de forma segura, somente quando as alterações foram mescladas. |
| git checkout nome-da-branch | Trocar de branch |
| git branch | Verificar as branches do projeto |

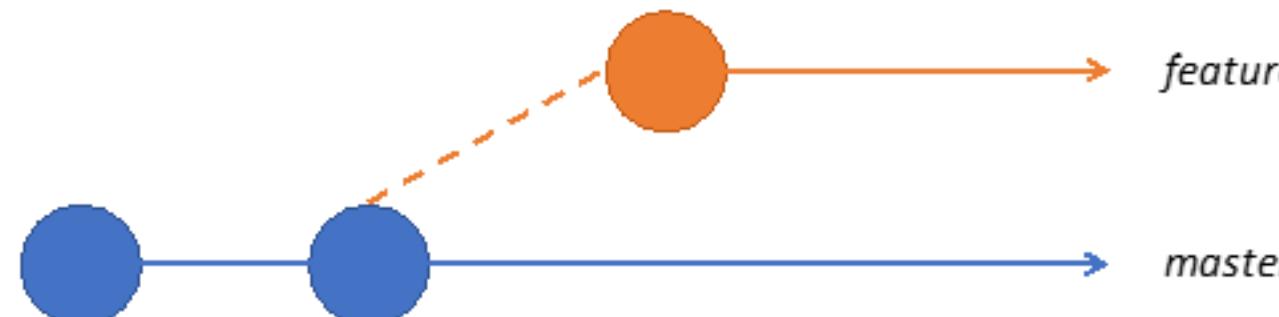
Novos comandos para branch

Atualizados na versão 2.23.0 do Git (em 2019)

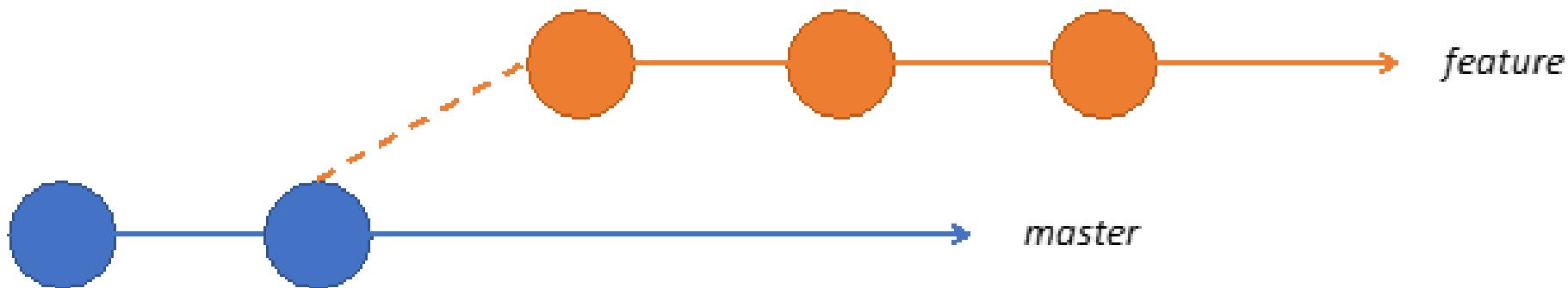
| Comando | Funcão |
|-------------------------------------|-----------------------|
| git switch -c nome-da-branch | Criar uma nova branch |
| git switch nome-da-branch | trocar de branch |

Trabalhando nas branches com merge

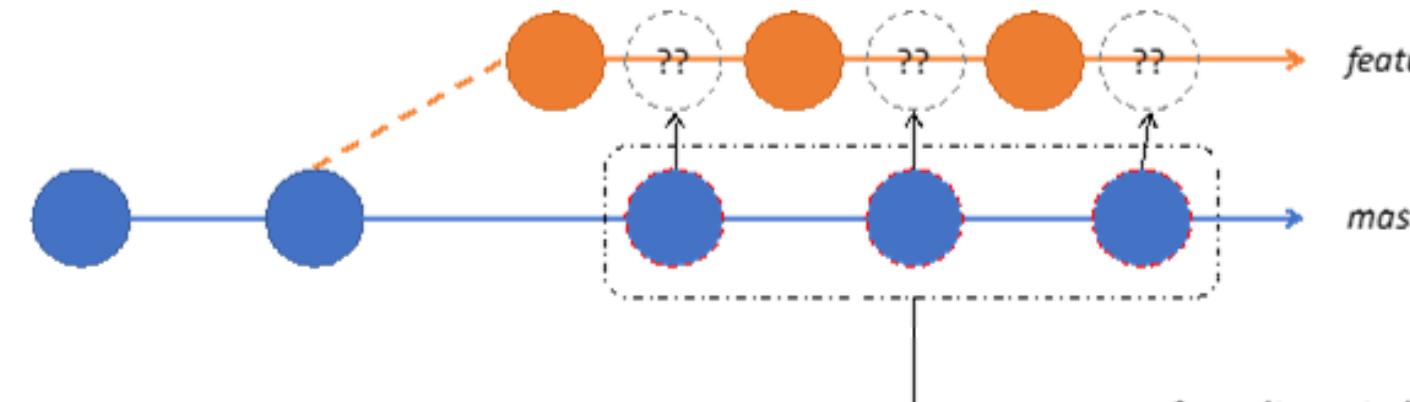
Iniciando commit em uma branch



Fazendo novas alterações



Em algum momento a branch main foi atualizada com outros commits de outras branches



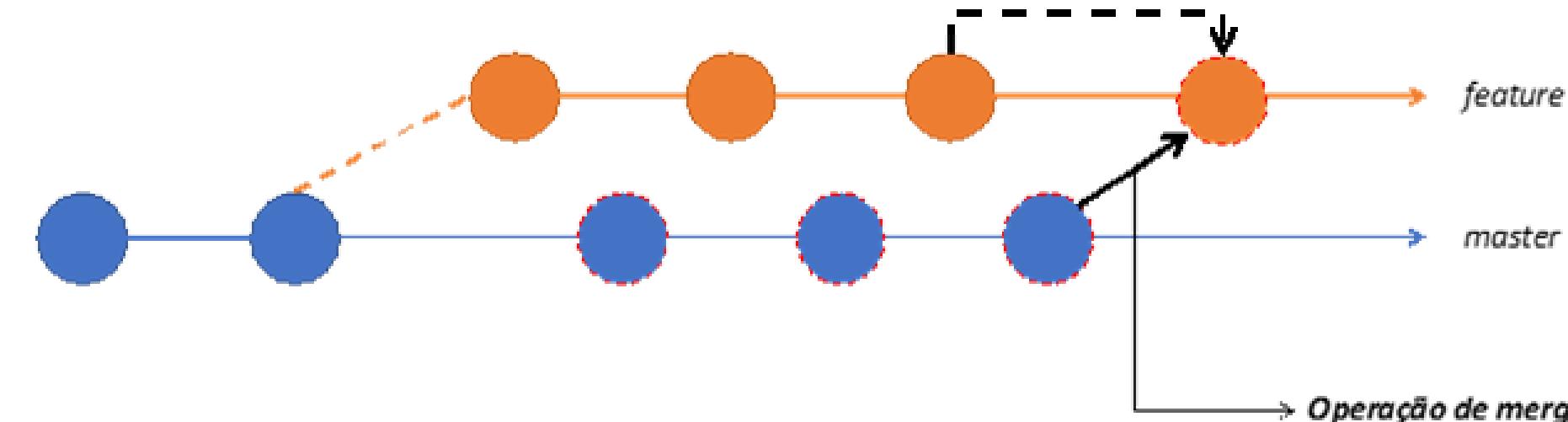
Commits posteriores que não existem no branch da nossa tarefa

Git Merge

+ simples

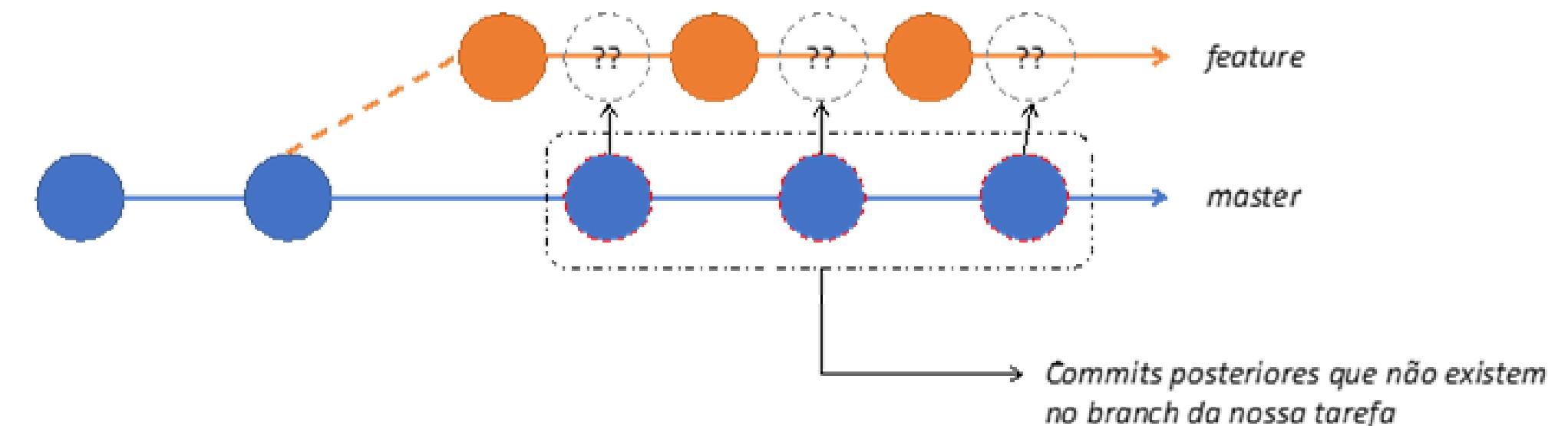
- mescla as branches
- o histórico é não linear
- cria um novo commit extra
- pode deixar o histórico poluído em grandes projetos

*git checkout feature
git merge main*



Trabalhando nas branches com rebase

Branches estão desincronizadas



Git Rebase

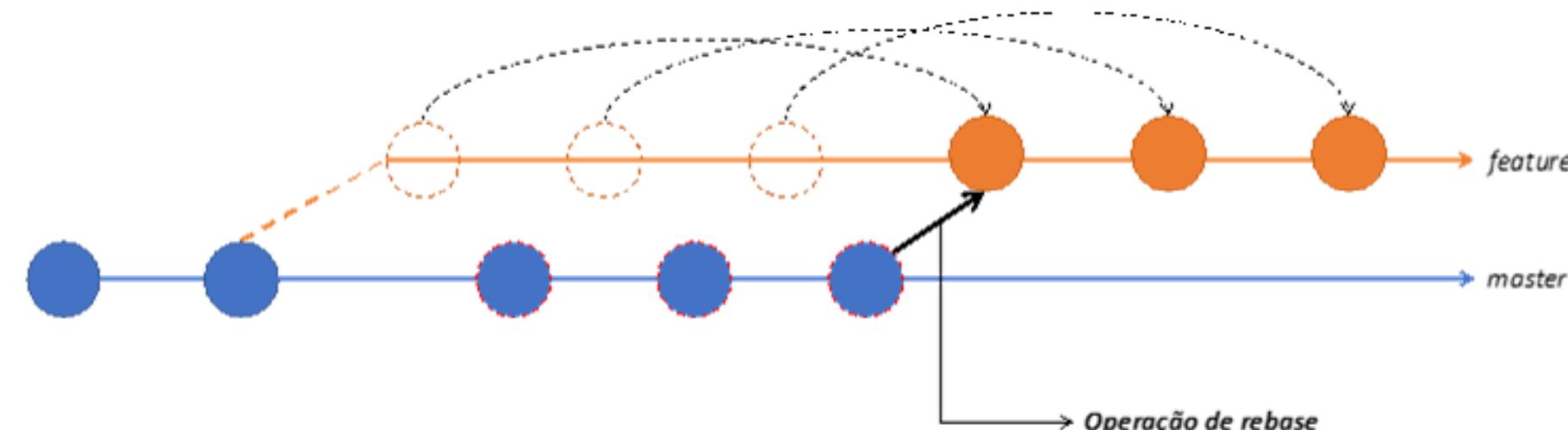
+

complexo

- deve ser feito antes do push
- aplica os commits da branch atual e os coloca “no topo” da branch base
- histórico fica linear, sem commits de merge.

git checkout feature
git rebase main

Os commits ficam à frente da branch master, criando um histórico linear e mais limpo. Há uma fusão e não é gerado commit extra.



| Ação | <code>git merge main</code> | <code>git rebase main</code> |
|----------------------|---|---|
| Histórico | Mantém todos os commits + cria um commit de merge extra | Reaplica seus commits em cima da main, histórico linear |
| Facilidade | Mais simples, seguro para times | Mais avançado, precisa atenção |
| Conflitos | Resolvidos no momento do merge | Resolvidos commit a commit durante o rebase |
| Colaboração | Ideal para trabalho em equipe (não reescreve histórico) | Melhor para uso individual (reescreve histórico) |
| Visual no log | Pode ficar “poluído” com commits de merge | Fica mais limpo, sem merges extras |
| Quando usar | Se já compartilhou a branch / projeto em equipe | Se está trabalhando sozinho ou antes de abrir PR |

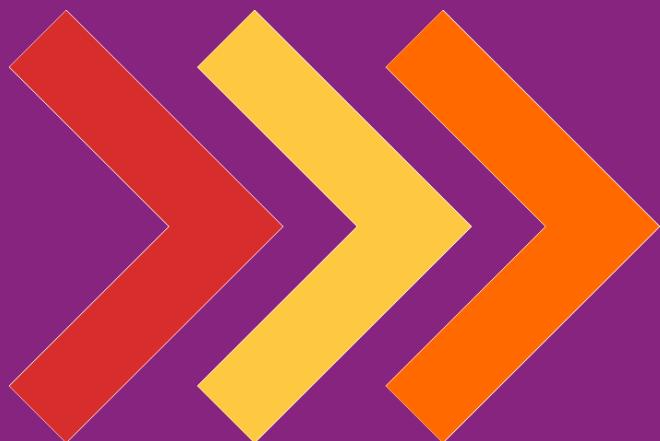
Importante!



A branch main é a base. Tem informações básicas e iniciais do projeto.

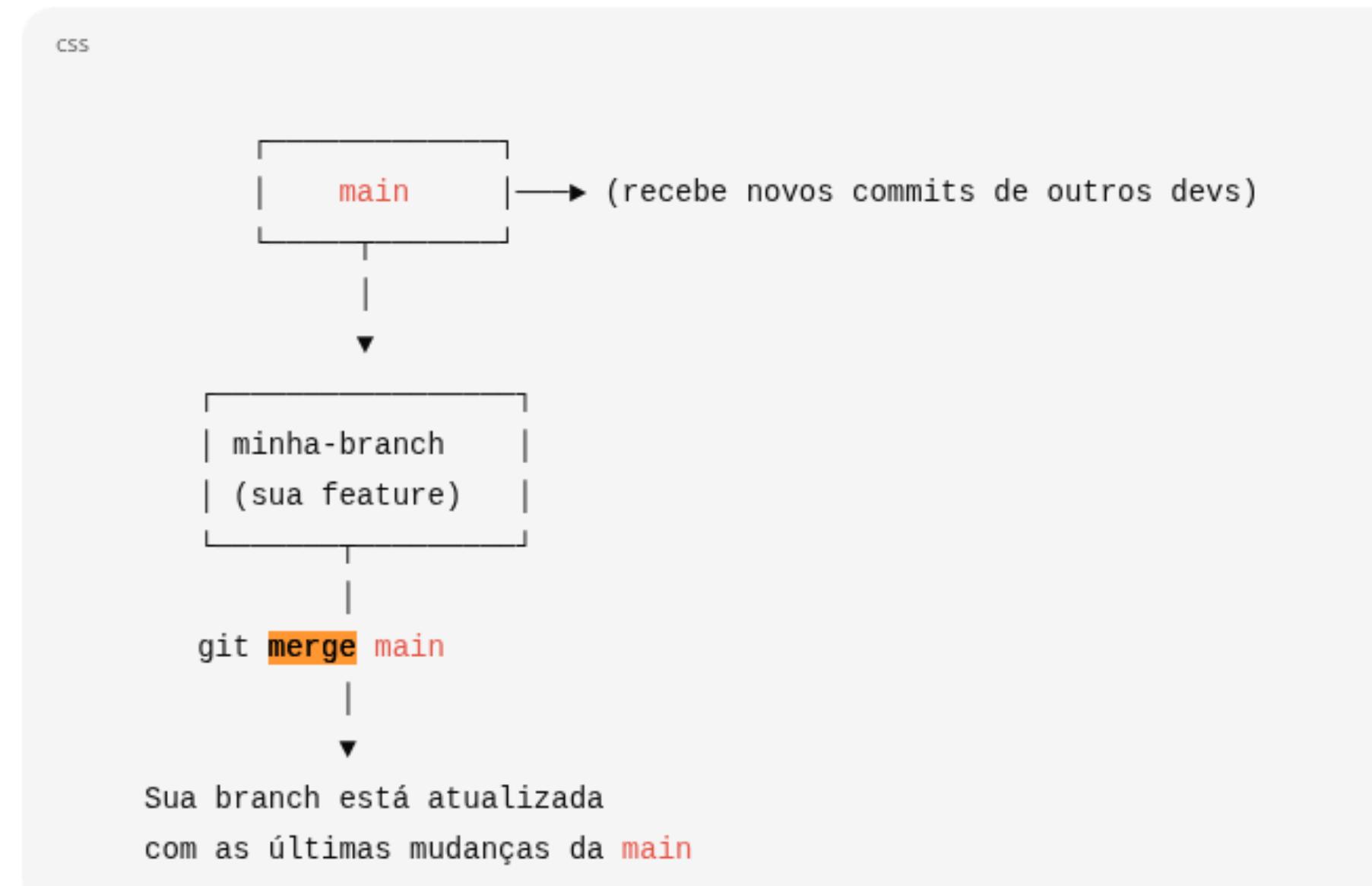
Os commits com as alterações devem acontecer nas branches.

Importante!



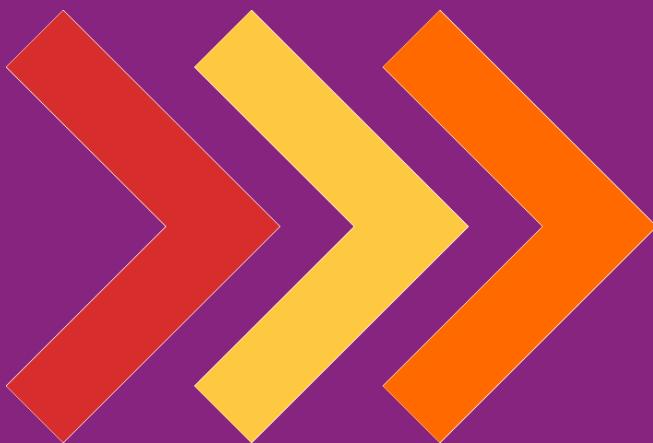
Quando quero trazer as alterações da main para a minha branch. Evita conflitos.

- **git checkout minha-branch**
- **git merge main**



Em trabalhos colaborativos, usamos este fluxo quando outros desenvolvedores atualizam main com as suas alterações.

Importante!



Quando terminamos as alterações da nossa branch e queremos levá-la para a main:

- ~~git checkout main~~
- ~~git merge minha branch~~

*Em trabalhos colaborativos,
usamos o Pull Request no GitHub para
mandar as alterações para a main!*

Fork:

Faz uma cópia do repositório remoto no próprio Github e você pode alterar esse repositório como se fosse seu, fazer o push, mesmo sem permissão.

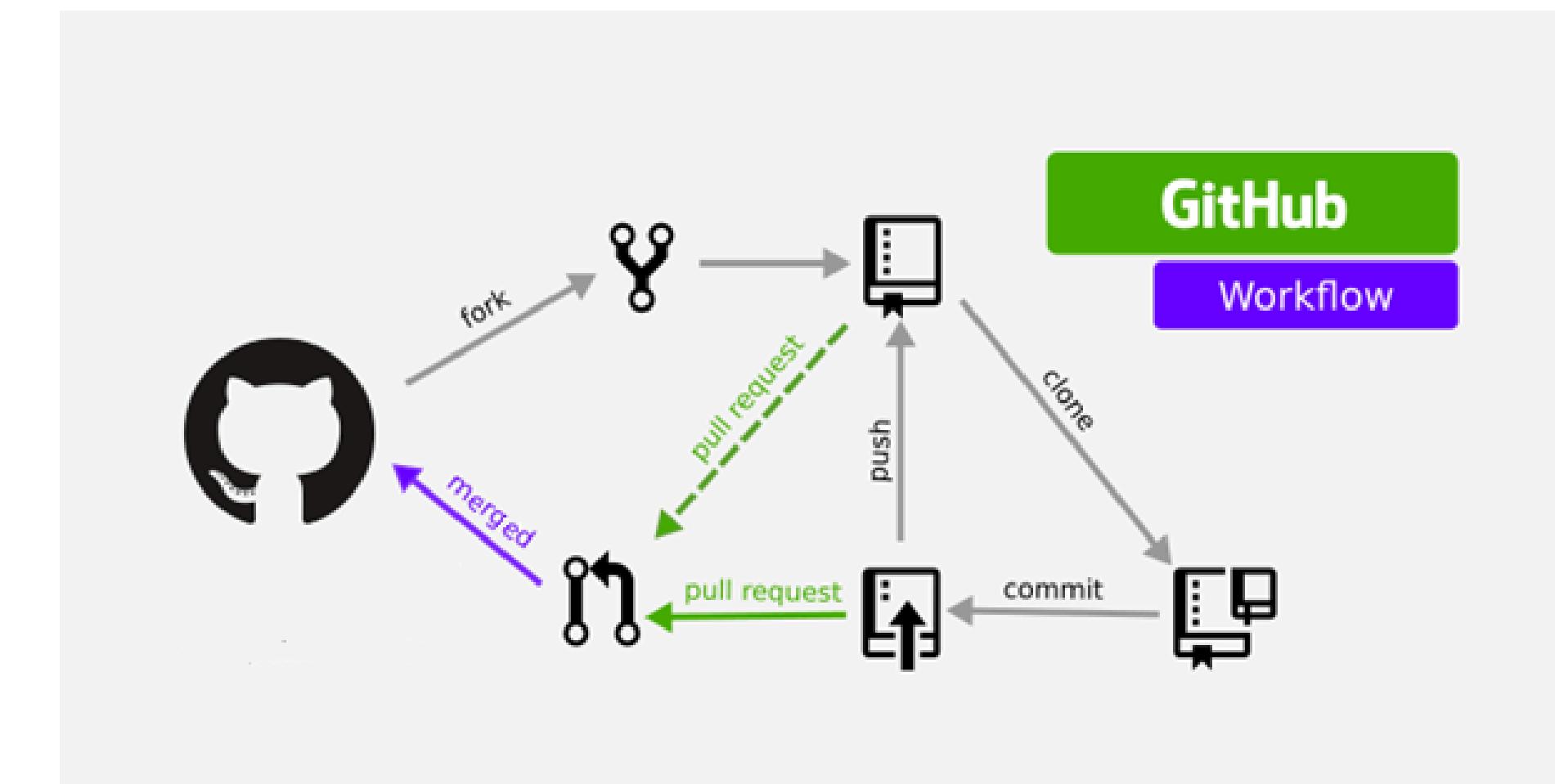
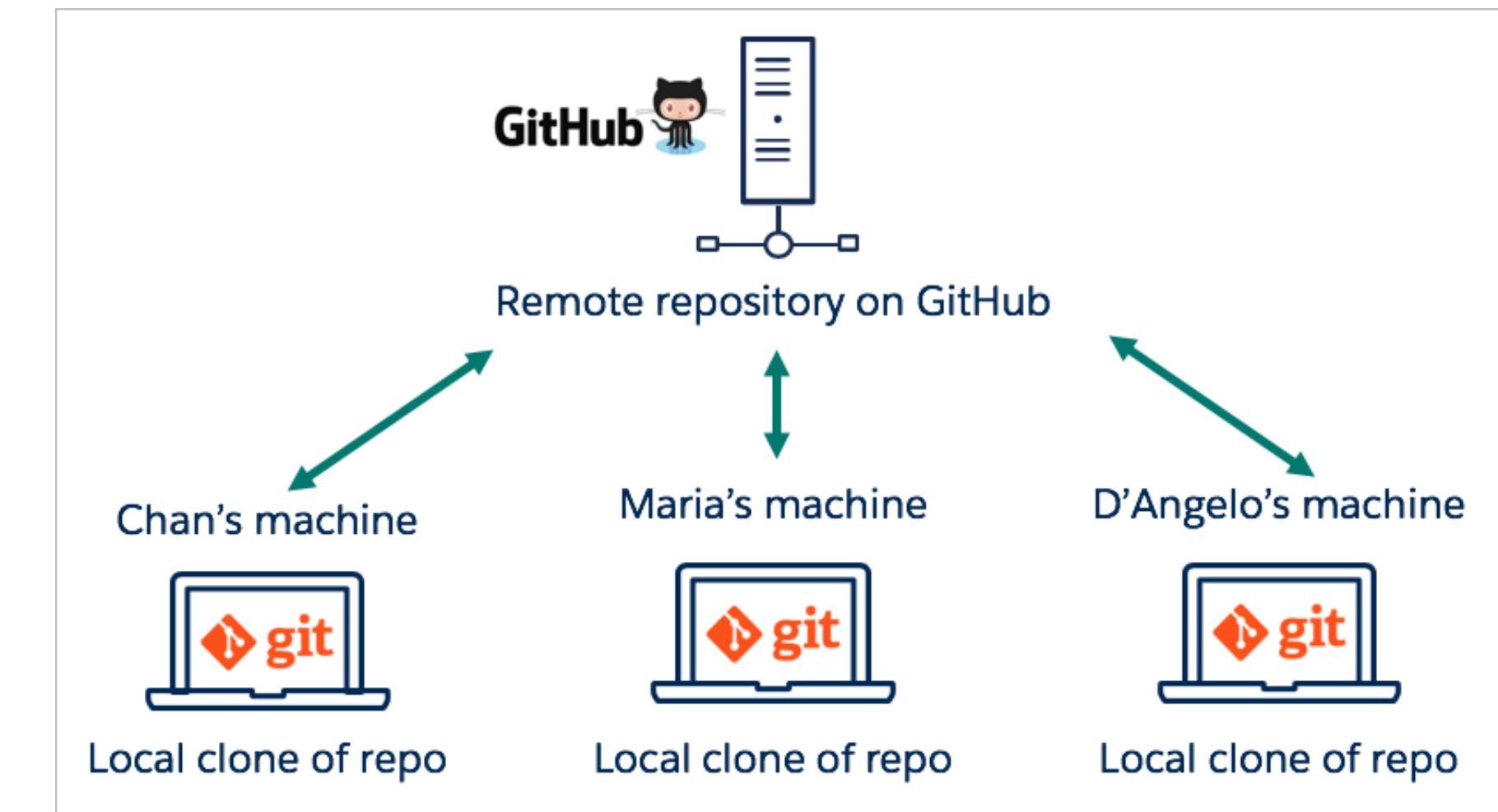


Imagen: Medium

Clone:

Faz uma cópia do repositório remoto na sua máquina.
Pode usar o repositório, mas para subir alterações,
precisa de permissão.

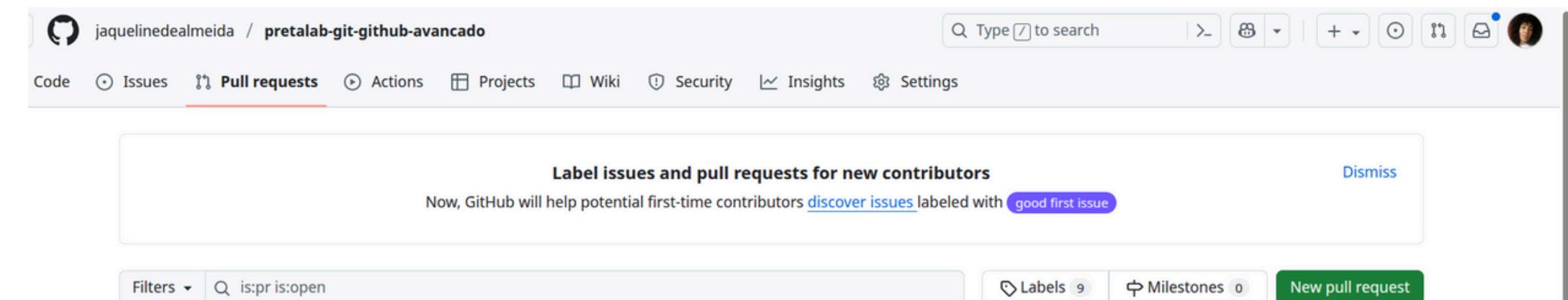


PULL REQUEST

Após **forkar ou clonar** um projeto, as alterações devem ser submetidas com Pull Request, feitas no GitHub.

Para nosso projeto:

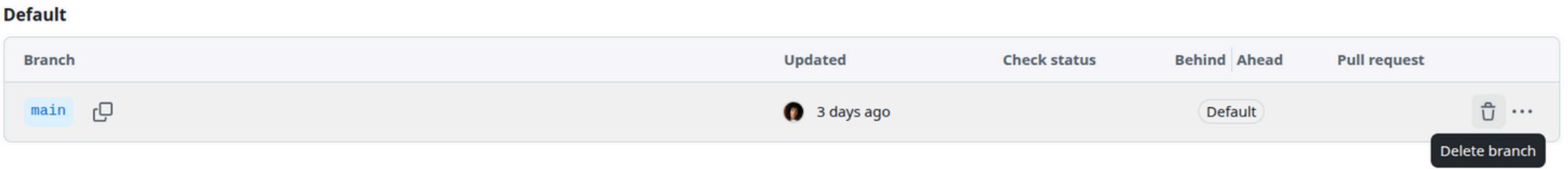
- Faço Fork
- Clono projeto na minha máquina com endereço do SSH
- Abro uma branch a partir da main
- Faço as alterações localmente
- Dou git add, git commit e git push
- Abro o PR, solicitando que as alterações sejam aceitas.
- Faço uma descrição do que foi alterado.



Apagando branch no GitHub

Remotamente:

Apagar via interface no Github:



Por comando:

`git push origin --delete minha-branch`

Aula 2

Ferramentas e Boas Práticas

Dúvidas?

Por enquanto é só(tudo) isso...

Obrigade

Aula 4 | CSS - Aprofundamento

- Espaçamento;
- Altura e largura;
- Bordas a elementos, tamanho do box, margem, inspecionando o css (box model do navegador);
- Flexfroggy.

Acordos importantes!!!



Aqui é um ambiente seguro;

Não existe pergunta boba;

Pausas são importantes;

Somos um coletivo;

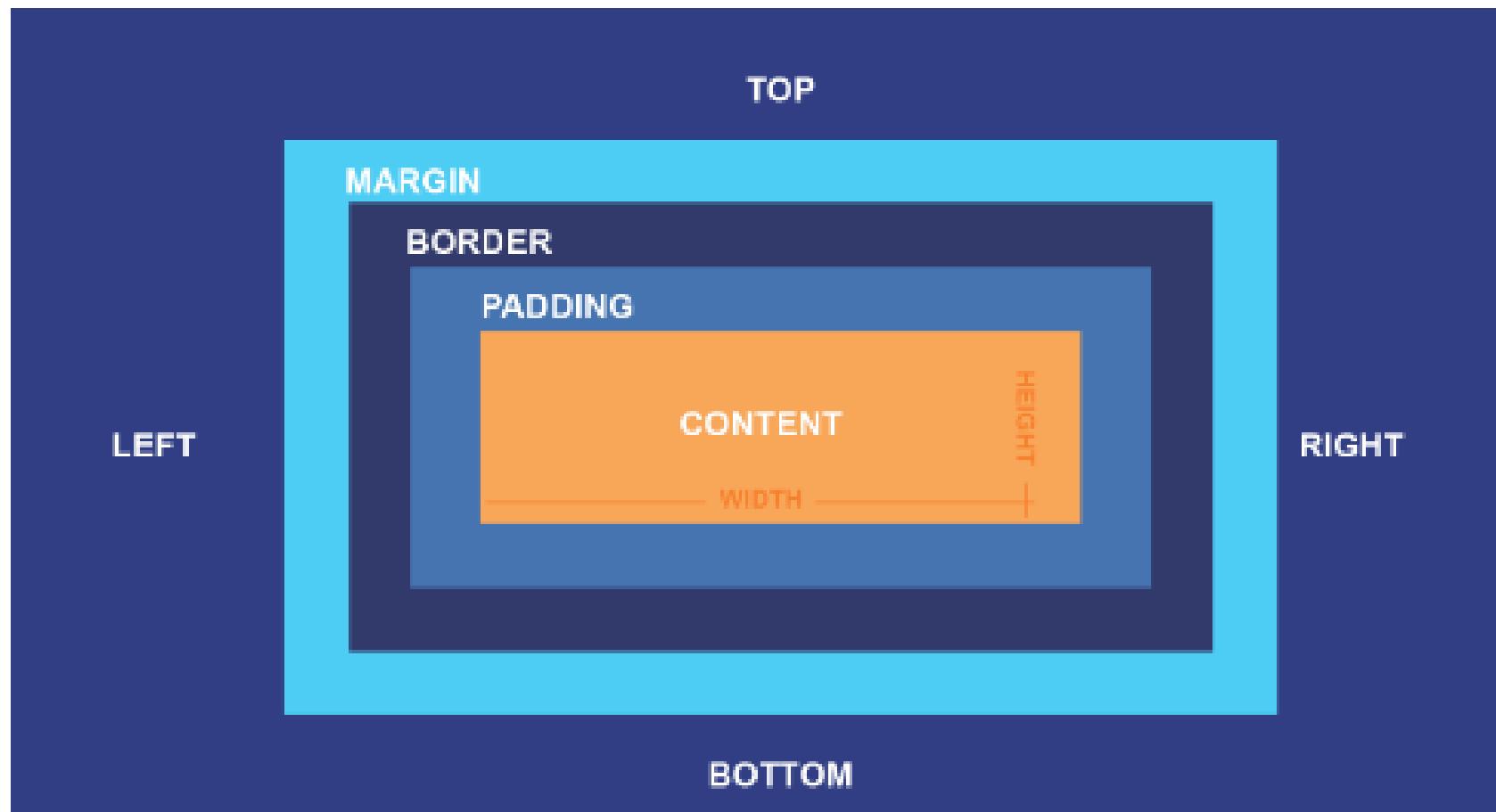
Bebam água;

Feedbacks são tudo de bom!

Box Model

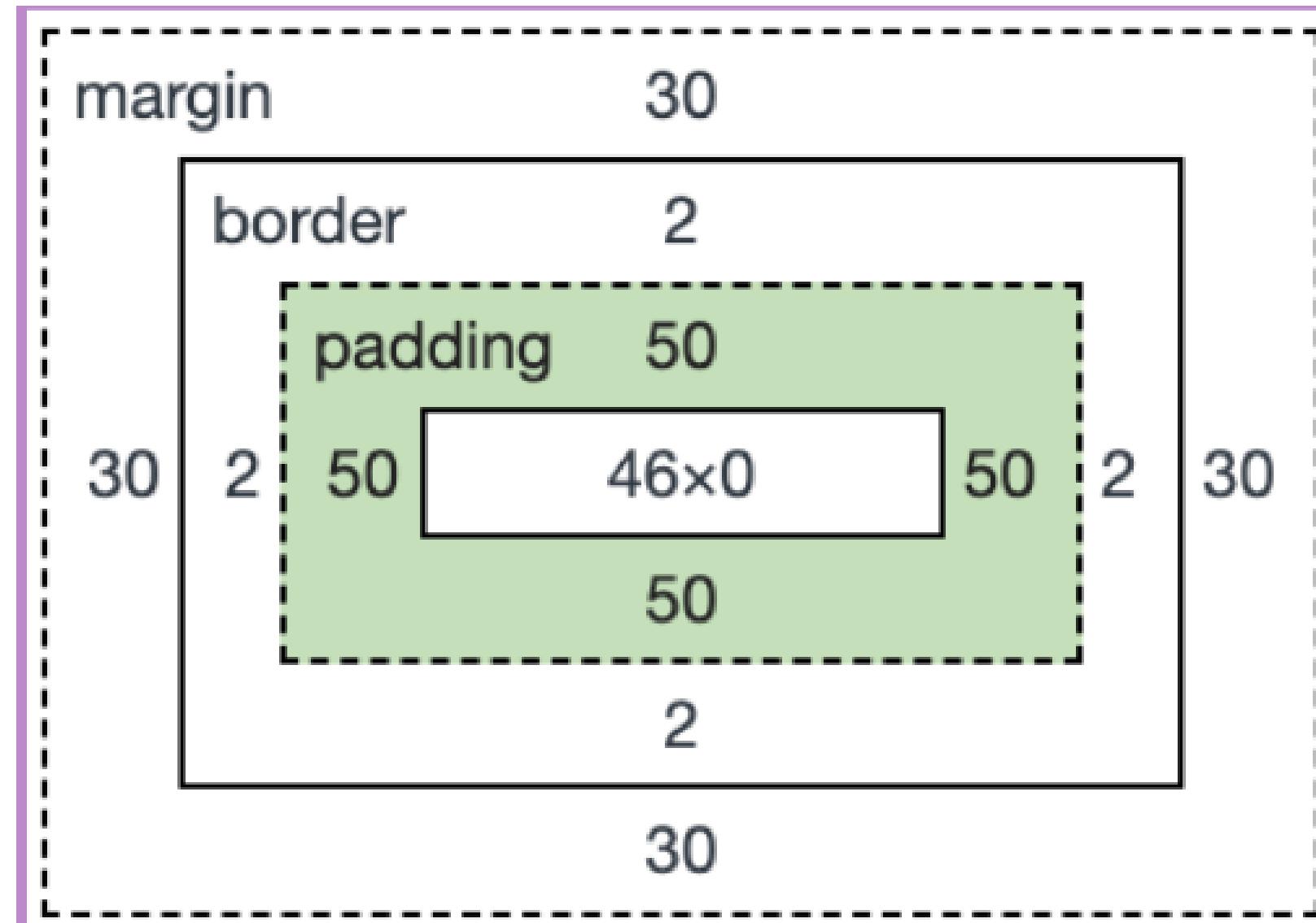
É o modelo de estilização em caixa com diversas propriedades que facilitam a vida do desenvolvedor.

A propriedade box-sizing é utilizada para alterar a propriedade padrão da box model, usada para calcular larguras (widths) e alturas (heights) dos elementos.



Espaçamento

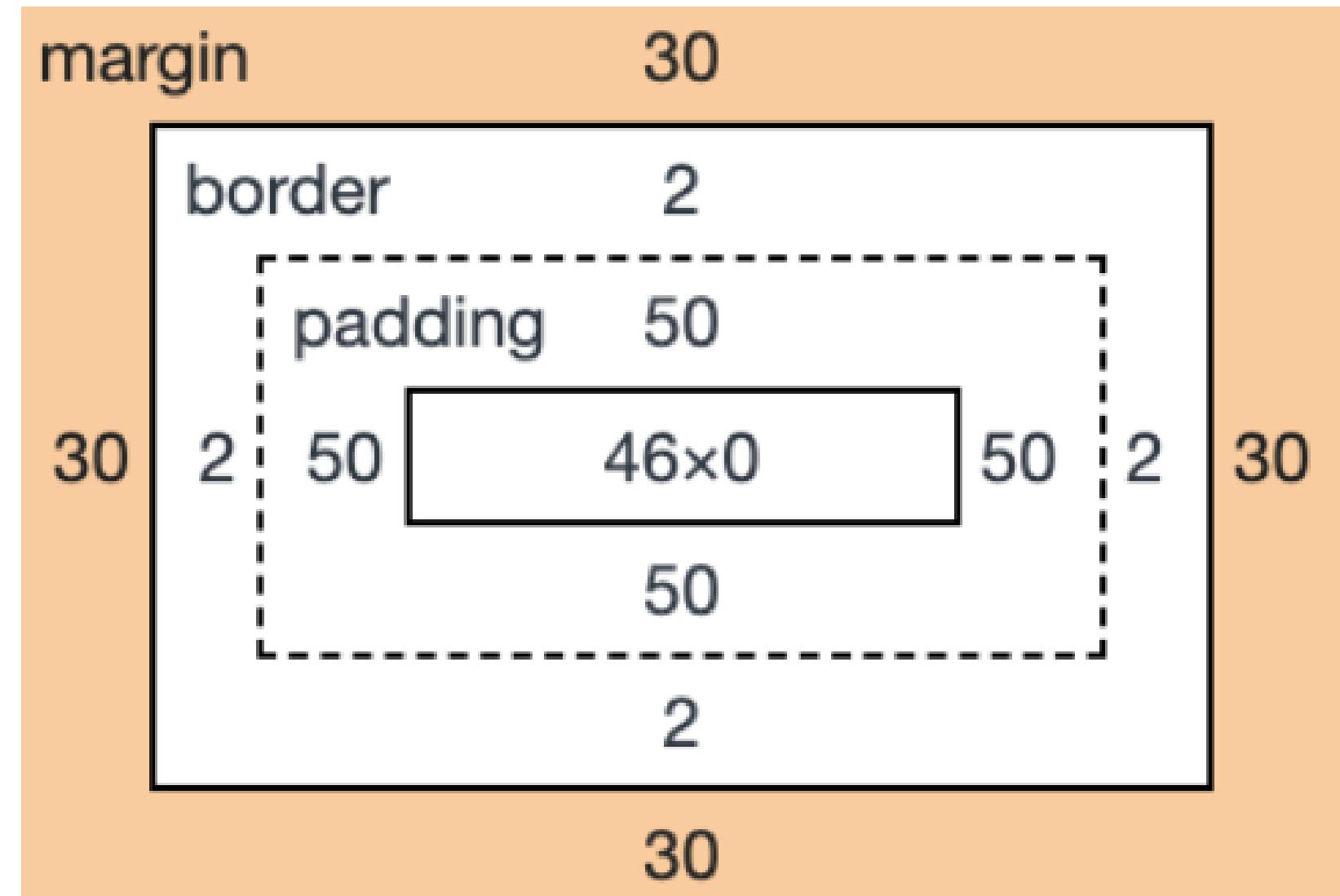
Padding: É o espaçamento interno do elemento.



padding: define espaçamento interno nos 4 lados;
padding-top: define espaçamento interno superior;
padding-right: define espaçamento interno à direita;
padding-bottom: define espaçamento interno inferior;
padding-left: define espaçamento interno à esquerda.

Espaçamento

Margin: É a parte do respiro externo, ou seja, é o espaçoamento que empurra o elemento do lado de fora.



margin: define espaçoamento externo nos 4 lados;
margin-top: define espaçoamento externo superior;
margin-right: define espaçoamento externo à direita;
margin-bottom: define espaçoamento externo inferior;
margin-left: define espaçoamento externo à esquerda.

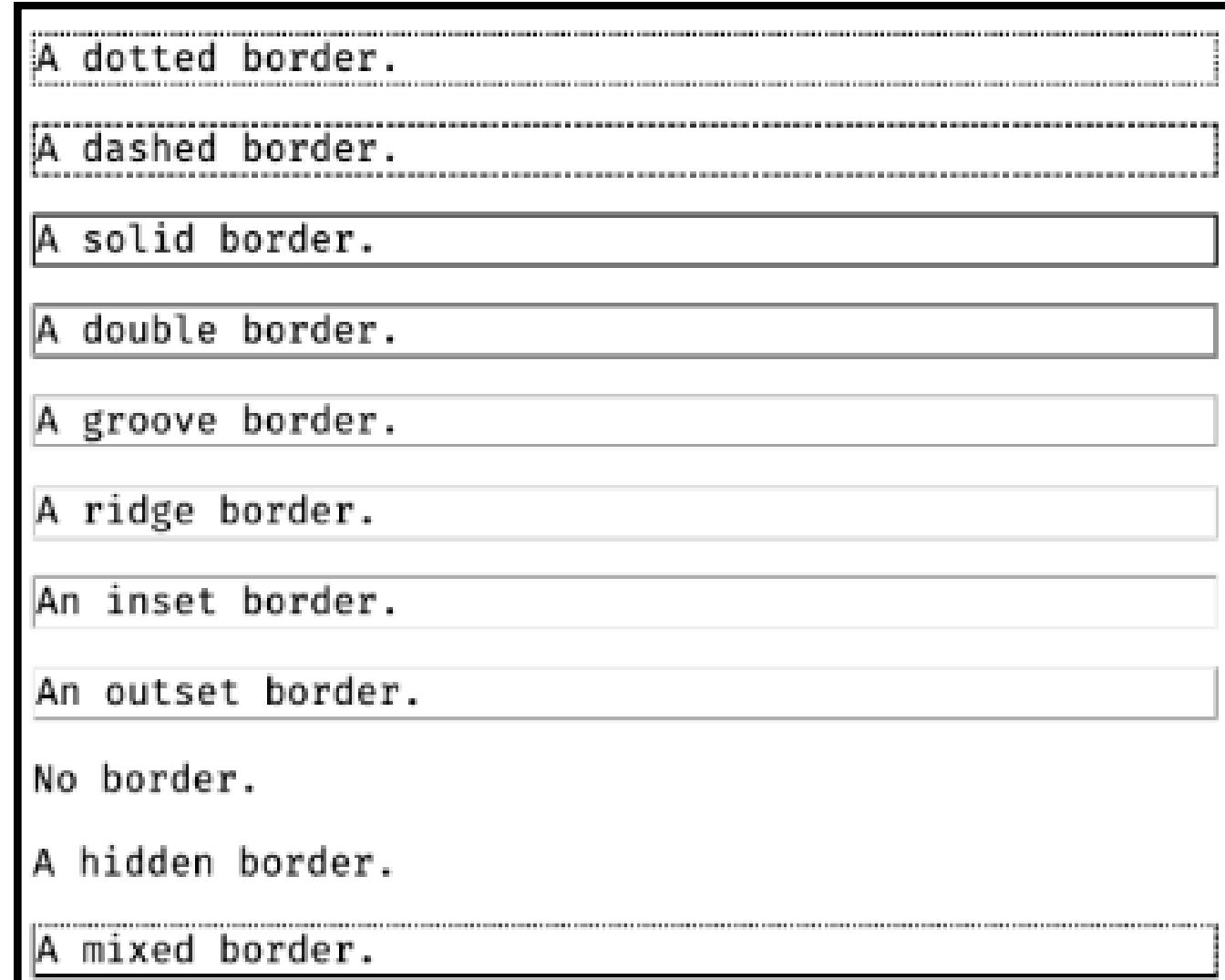
Bordas

A borda de um elemento HTML é a área ao redor do conteúdo desse elemento que pode ser estilizada visualmente com diferentes propriedades CSS, como a largura, cor, estilo e espessura.

[Saiba mais aqui!](#)

border: abreviação referente as quatro bordas;
border-top: propriedades da borda superior;
border-bottom: define propriedades da borda inferior.
border-right: propriedades da borda direita;
border-left: propriedades da borda esquerda;
border-style: define o estilo de borda dos elementos
border-width: define a largura do elemento
border-color: define cores visíveis.

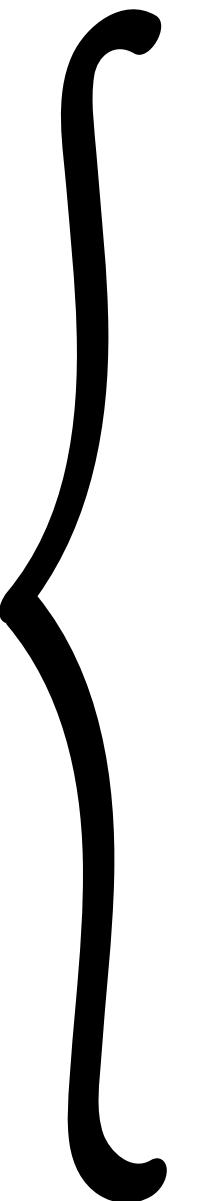
```
.caixa {  
    border: 4px solid yellowgreen;  
    border-radius: 10px;  
  
    padding: 20px;  
}
```



Altura (Height) e Largura (Width)

Height: É a altura do elemento

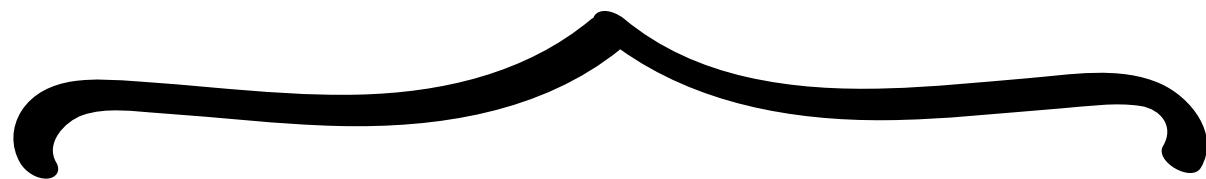
Height



Altura (Height) e Largura (Width)

Width: É a largura do elemento

Width



Posicionamento

Display: Define o comportamento de exibição de um elemento, como block (bloco), inline (linha), ou none (nada). Controla como os elementos ocupam espaço na página.

Float: Controla o posicionamento de um elemento, permitindo que ele flutue à esquerda ou à direita em relação aos elementos vizinhos. Usado principalmente para criar layouts de várias colunas.

Clear: Controla como os elementos se comportam em relação a elementos flutuantes. Pode ser usado para forçar um elemento a ser exibido abaixo de todos os elementos flutuantes anteriores.

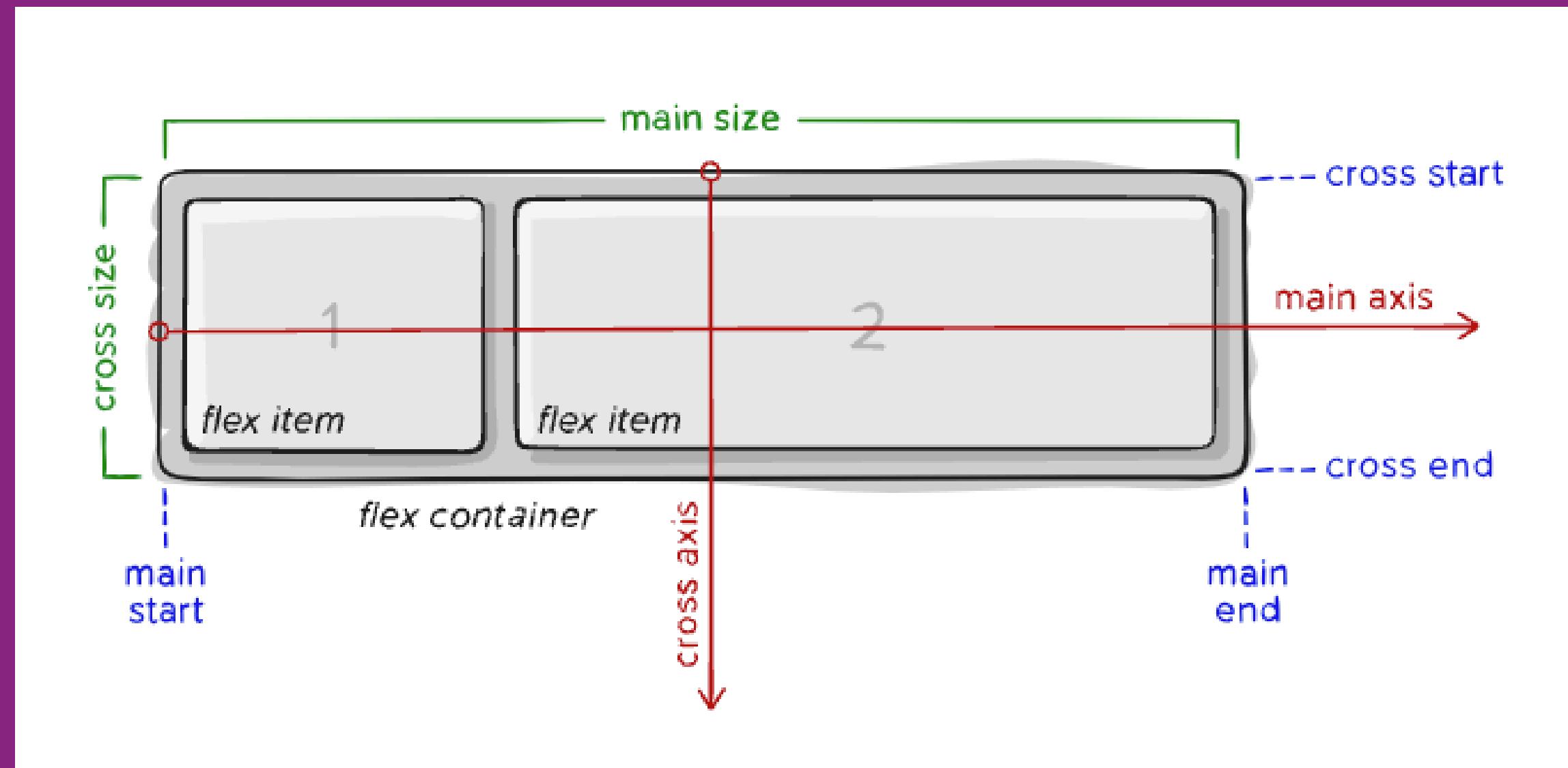
clearfix: Uma técnica usada para lidar com problemas de layout quando elementos filhos são flutuantes. Garante que o elemento pai tenha a altura correta, considerando os elementos filhos flutuantes.

Posicionamento

Display flex: é uma propriedade CSS que permite transformar um elemento pai em um container flexível. Isso faz com que os filhos desse elemento se comportem como itens flexíveis, permitindo a criação de layouts flexíveis e responsivos.



Posicionamento



Posicionamento

Propriedades para o elemento-pai:

- display
- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

Propriedades para elementos-filhos:

- order
- flex-grow
- flex-shrink
- flex
- flex-basis
- align-self

Flexfroggy



Dúvidas?

Por enquanto é só(tudo) isso...

Obrigade