

# KANBAN BOARD + VERSIONAMENTO



DEVinHouse

Parcerias para desenvolver a sua carreira

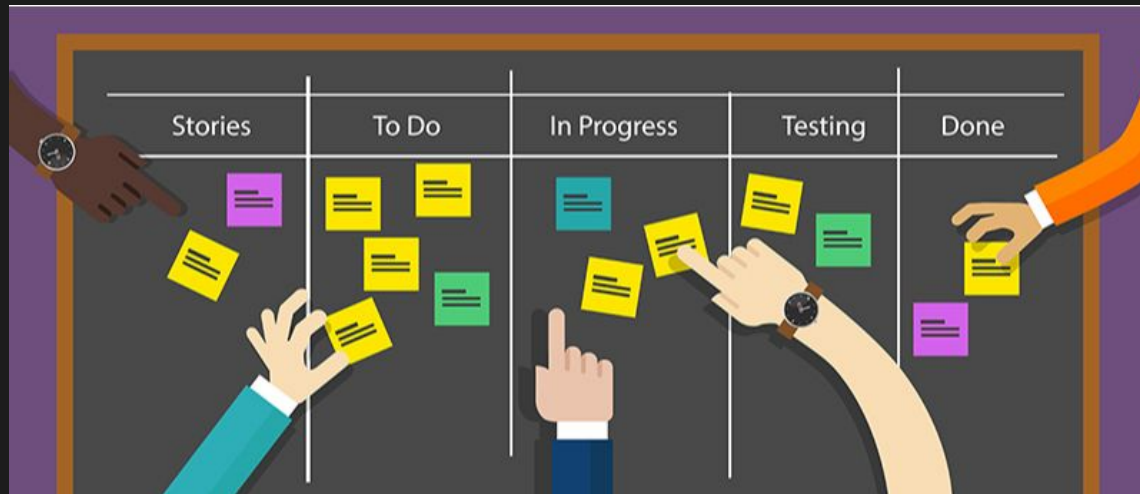
**SENAI**

<LAB365>

# AGENDA

- Kanban;
- Trello;
- Versionamento de código;
- Git.

O que você entende por Kanban?



- Kanban é um sistema de controle de estoque e fluxo de peças desenvolvido pela Toyota por volta de 1950/60;
- Foi adaptado para diversas áreas, inclusive desenvolvimento de software, onde é muito utilizado para Gerenciamento de Projetos;
- É um sistema visual de gestão de trabalho, que busca conduzir cada tarefa por um fluxo pré definido de trabalho.

# KANBAN



Fonte: [O que é Kanban: Guia completo \(atualizado 2021\)](#)

# KANBAN

- Kanban é um **fluxo de trabalho** que busca **indicar (e limitar)** o **trabalho em andamento**;
- O foco do Kanban é **priorizar** a produtividade e a **organização** das entregas;
- O objetivo é proporcionar um **trabalho** mais **transparente** e **direcionado**.

- O kanban tem três principais funções:
  - Gerenciar o fluxo de trabalho e facilitar a visualização da dimensão do que está sendo produzido e em que ritmo está sendo produzido;
  - Equilibrar os processos que vêm antes e depois, para que uma atividade não seja interrompida pela falta de uma outra que deveria ter sido entregue anteriormente;
  - Limitar a quantidade de trabalho que deve ser realizada pela equipe, respeitando a capacidade produtiva.

- As três partes principais do Kanban são:
  - **Cartão:**
    - É a menor parte do kanban;
    - Trata-se de uma tarefa ou ação que precisa ser tomada para que o resultado final seja entregue;
    - Geralmente são diferenciados por um sistema de cores;
    - Cada vez que uma atividade muda de status, basta mover o cartão para a outra coluna que indica o estado atual.



- As três partes principais do Kanban são:
  - **Colunas:**
    - Representam os status dos cartões;
    - Geralmente possui três colunas: A Fazer, Em Execução e Feito, mas essas colunas podem mudar de acordo com a necessidade da equipe de trabalho;
    - Os cartões devem ser movidos entre as colunas conforme seu status for mudando, dando um panorama do que está pendente e do que já foi concluído.

- As três partes principais do Kanban são:
  - **Quadro:**
    - O quadro nada mais é do que o kanban como um todo, organizado em colunas e cartões.
    - Cada quadro é um kanban e uma única equipe pode trabalhar com vários quadros simultaneamente.

# VANTAGENS DO KANBAN

- **Priorização de tarefas:**
  - Com a adequação de cores, ordem ou colunas, é possível destacar uma tarefa para demonstrar a importância da mesma.
- **Aumento da produtividade:**
  - Considerando que o Kanban permite ver futuras tarefas, não há tempo ocioso. Também, se especifica o que deve ser feito, não sendo necessário o retrabalho.

# VANTAGENS DO KANBAN

- **Redução de custos:**

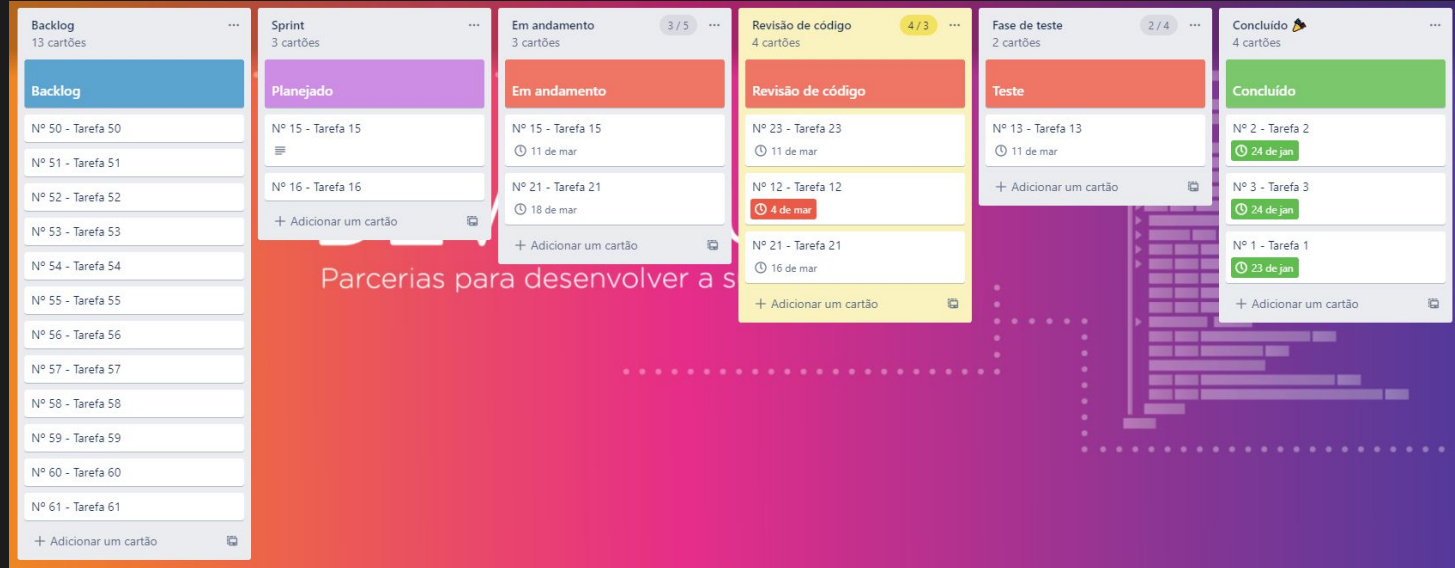
- O aumento da produtividade já traz uma redução de custos. Mas, também é possível direcionar melhor a equipe com a eficiência e a facilidade que a metodologia Kanban mostra.

- **Autonomia:**

- É fácil olhar para o quadro e entender o status das entregas e também o que precisa ser feito, isso estimula a autonomia da equipe de trabalho já que eles podem verificar sozinhos o andamento das entregas.

- Existem vários softwares que possibilitam a implementação do sistema Kanban;
- Um dos mais famosos é o Trello, é bastante completo e gratuito;
- Possibilita criação de quadros, colunas e cartões;
- Disponibiliza diversas formas de edição dos cartões e customização das listas (colunas);
- Na forma de Power-Ups, permite integração com diversos outros serviços de software.

## Modelo de Kanban para facilita a visão do andamento do projeto



# LISTA DE EXERCÍCIOS

Vamos ver como vai funcionar o processo de apresentação e resolução das listas de exercícios que devem ser realizadas semanalmente, visando a fixação dos conteúdos apresentados na semana!



# LISTA DE EXERCÍCIOS

Às sextas-feiras ocorre o encontro de tira-dúvidas, nesse momento vocês devem tirar todas as suas dúvidas referente aos conteúdos apresentados.

Também é realizada a correção dos exercícios, reforçando a fixação dos conteúdos apresentados na semana.



# LISTA DE EXERCÍCIOS

O objetivo após esse encontro é que vocês não fiquem com nenhuma dúvida e dominem o conteúdo proposto.

Caso tenha alguma dúvida sobre a tarefa a realizar (exercício), pergunte no canal tira-dúvidas do Slack, ou tragam na sexta-feira para discutirmos.

# INTERVALO DE AULA

## **DEV!**

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

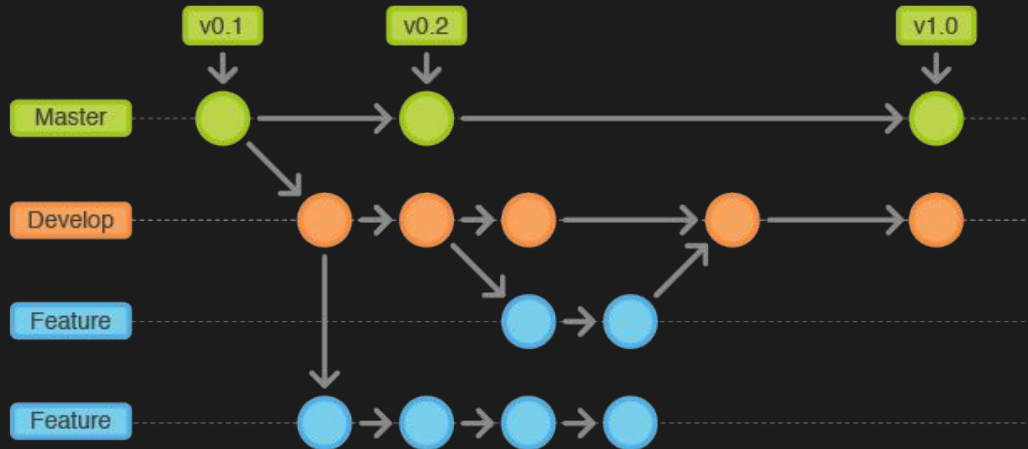
**Início:** 20:20

**Retorno:** 20:40



# VERSIONAMENTO DE CÓDIGO

O que vocês entendem por versionamento?



# VERSIONAMENTO DE CÓDIGO

- O versionamento é o gerenciamento de versões diferentes de um documento de texto qualquer;
- Não precisa ser apenas para código;
- É utilizado no desenvolvimento de software para controlar as diferentes versões e histórico de desenvolvimento do código.

# VERSIONAMENTO DE CÓDIGO

Imagine que você está escrevendo um programa e precisa editar um arquivo, porém, você quer manter uma cópia da versão anterior desse arquivo como segurança.

Desse modo, você pode retomar seu arquivo da versão anterior, caso algo em sua alteração dê errado, ou por algum motivo você queira voltar para a versão anterior.

# VERSIONAMENTO DE CÓDIGO

Vamos supor que o seu arquivo se chama `index.html`. Você já construiu bastante coisa, e não gostaria de perder seu progresso caso algo dê errado. Por isso, você cria uma cópia desse arquivo e a chama de `index_estavel.html`, e continua editando o arquivo `index.html`.

Desta forma você versionou seu código, pois agora existem 2 versões do arquivo `index.html`.

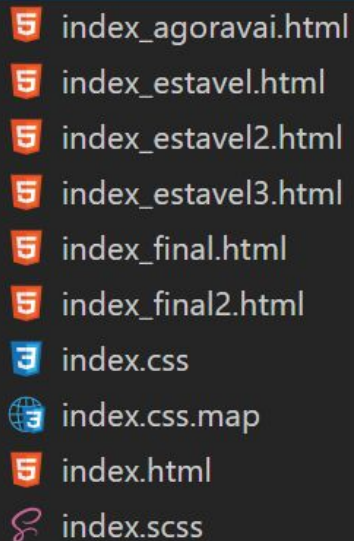
# VERSIONAMENTO DE CÓDIGO











E qual o problema de versionar código criando cópias físicas para cada alteração?

Imagine que você continuou trabalhando nesse mesmo programa por mais alguns dias e foi criando várias cópias desse arquivo.

# VERSIONAMENTO DE CÓDIGO

Quando você olha para sua pasta, ela está assim:



-  index\_agoravai.html
-  index\_estavel.html
-  index\_estavel2.html
-  index\_estavel3.html
-  index\_final.html
-  index\_final2.html
-  index.css
-  index.css.map
-  index.html
-  index.scss



Alguns dias depois, você volta a mexer nesse mesmo projeto e se depara com alguns problemas:

- Você não sabe mais qual é a versão mais recente do arquivo;
- Você não sabe qual a diferença entre cada versão do arquivo;
- Você não sabe em qual delas está aquela alteração específica que você fez.

- Criado por Linus Benedict Torvalds e lançado em 2005, se tornou a principal ferramenta de controle de versão;
- Através dela podemos desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas.

- Git é um sistema de controle de versão de arquivos;
- Com ele, podemos manter um histórico de todas as alterações que foram realizadas em nossos arquivos de código fonte;
- Em equipe, podemos visualizar todas as alterações realizadas por todos os membros.

- Cada nova funcionalidade (ou correção) que você adicionada na sua aplicação, você salva nesse sistema de controle;
- Muitas vezes, uma nova funcionalidade/correção significa vários arquivos alterados de uma só vez;
- Ao final do desenvolvimento de cada nova funcionalidade ou correção, salvamos o estado dos arquivos naquele momento.

- Cada “salvar” desses, fica registrado em um “pacotinho de alterações”. É o que chamamos de “commit”;
- Quando você quiser voltar o estado da sua aplicação para um ponto específico do passado, ou apenas visualizar como era esse código em algum momento passado, você faz isso facilmente.

- Seja para comparar uma versão do código que funcionava com outra que não está mais funcionando, resolvendo um bug(erro);
- Ou apenas descobrir o que foi alterado de uma versão para outra, por diversas outras razões.

- Vamos realizar a instalação do Git.
  - Windows: <https://gitforwindows.org/>
  - Linux/macOS: <https://git-scm.com/downloads>
- Para verificar se a instalação foi executada com sucesso:
  - `git --version`

- Vamos criar uma conta no github.
  - <https://github.com/>



- Vamos configurar o Git.
  - Configurar o nome e e-mail que irão aparecer no commit. Para isso utilizamos os seguintes comandos:
    - `git config --global user.name "Seu nome"`
    - `git config --global user.email "email@example.com"`
    - `git config --list`

- O SSH é um protocolo de rede que permite a conexão com determinados servidores por meio de uma comunicação criptografada, trazendo mais segurança para as transações de dados;
- O Github permite que você crie chaves SSH para que você gerencie tudo de maneira remota, com segurança e sem precisar fornecer seu nome de usuário e token de acesso pessoal toda vez que quiser acessar.

- Configurar a chave SSH:
  - Abrir o terminal;
  - Executar o comando para criação da chave:
    - `ssh-keygen -t ed25519 -C "email@example.com"`
  - Adicionar chave privada no ssh-agent, o ssh-agent é um gerenciador de chaves ssh:
    - `eval $(ssh-agent -s)`
    - `ssh-add ~/.ssh/id_ed25519`

- Copiar chave pública:
  - No Windows:
    - `clip < ~/.ssh/id_ed25519.pub`
  - No Linux:
    - `cat ~/.ssh/id_ed25519.pub`
  - MacOS:
    - `pbcopy < ~/.ssh/id_ed25519.pub`

- Adicionar chave no Github
  - Abra o Github e vá no ícone de perfil > Settings (configurações), no canto superior direito;
  - Na barra lateral de configurações do usuário, clique em "SSH and GPG keys";
  - Clique no botão "New SSH key";
  - No campo "Título", adicione um rótulo descritivo para a nova chave;
  - Cole a chave pública que está na área de transferência no campo "Chave";
  - Clique em "Add SSH key".

- Testando a conexão SSH
  - Executar o seguinte comando: `ssh -T git@github.com`
  - Aguarde as mensagens e digite "yes" para continuar;
  - Verifique se a mensagem resultante contém seu nome de usuário e o sucesso da sua autenticação.

- Fluxo de trabalho:
  - Modificar arquivos no seu diretório de trabalho;
  - Preparar os arquivos, adicionando eles à sua área de preparo;
  - Fazer o commit, que leva os arquivos como estão na área de preparo e armazena de forma permanente no diretório do Git.

- Antes de começar, precisamos aprender os principais comandos Git:
  - **git init**: inicia um novo repositório;
  - **git add .**: Adiciona todos os arquivos que foram modificados ou podemos remover o ponto e adicionar um arquivo específico, ex: `git add index.html`;
  - **git status**: Verificar o status dos arquivos adicionados;
  - **git commit -m "Mensagem do commit"**: comando utilizado para salvar suas informações no repositório.

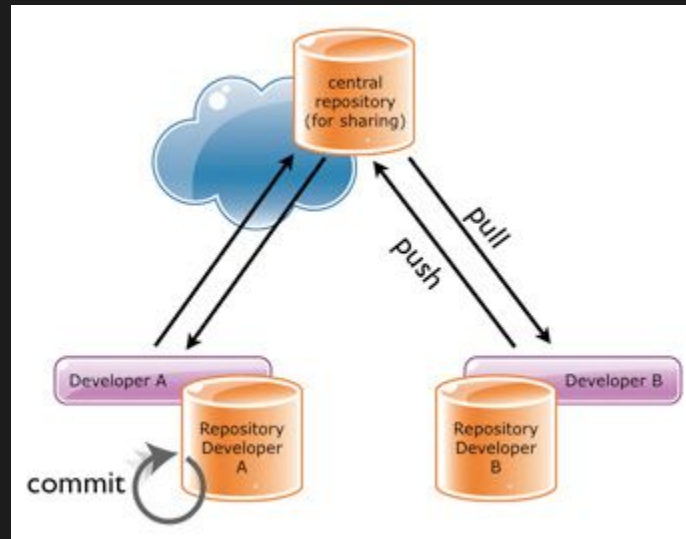


- Em uma pasta com os arquivos, iniciaremos um repositório git com **git init**
- Em seguida, se deve preparar os arquivos para adicioná-los de fato ao repositório. Fazemos isso com **git add** .
- Agora vamos executar o **git status** para ver o que foi adicionado;
- Por fim criamos um commit integrando o código com **git commit -m "mensagem do commit"**

- Além do gerenciamento local do repositório, existe o gerenciamento remoto, ou simplesmente repositório remoto;
- Esse repositório é o original do qual os locais são uma cópia, e é para onde as alterações dos repositórios locais vão;
- Os repositórios podem ser públicos ou privados. Em ambos os casos para alterarmos o repositório principal precisamos de uma permissão correspondente;

- Fluxo de trabalho remoto:
  - Obter um repositório remoto com “**git clone url-do-repositório**”;
  - Enviar uma alteração de código(commit) com “**git push**”;
  - Obter as alterações de um repositório posteriores a ser clonado com “**git pull**”.

Fluxo de trabalho remoto:



# GIT REMOTE - PRÁTICA

- Vamos **criar um repositório** no github;
- Em seguida vamos clonar esse repositório utilizando o comando **git clone**;
- Vamos adicionar alguns arquivos na pasta e realizar o **commit**;
- Atualizar as alterações no git remoto com o comando **git push**.

# LISTA DE COMANDOS GIT

- **git init** - inicia um novo repositório;
- **git add .** - Adiciona todos os arquivos que foram modificados;
- **git status** - Mostra informações dos arquivos que foram modificados;
- **git commit -m "Mensagem do commit"** - comando utilizado para salvar suas informações no repositório;

# LISTA DE COMANDOS GIT

- **git push** - Comando utilizado para enviar as informações para o repositório criado;
- **git pull** - Comando utilizado para pegar/atualizar as informações do repositório direto do servidor(github por exemplo);
- **git log** - Mostra um log de todos os commits feitos;

# LISTA DE COMANDOS GIT

- **git clone urlDoRepositorio** - Utilizado para baixar um repositório de fora para o seu computador.



# MATERIAL COMPLEMENTAR

- Kanban - [kanbanize](#)
- Trello - [blog.trello](#)
- Trello - [trello](#)
- Kanban - [rockcontent](#)

# MATERIAL COMPLEMENTAR

- Git - [woliveiras](#)
- Git - [medium](#)
- Commit - [conventionalcommits](#)
- Commit - [karma-runner](#)
- Playlist Git e GitHub para iniciantes: [Willian Justen](#);
- Configuração SSH: [GitHub](#);

## AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

[Clique aqui](#) ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.





# DEVinHouse

Parcerias para desenvolver a sua carreira

**OBRIGADO!**



<LAB365>