Git Mantra

Agenda

1. Introdução
2. Instalação e configuração
3. Primeiros passos
4. Branches
5. Boas práticas
6. Próximos passos

Introdução

Contexto, motivação, objetivos

Contexto

- Desenvolvimento de código no Fluig não apresenta um padrão estabelecido
- Dificuldade de manutenção de código
- Ausência de versionamento
- Aumento da curva de aprendizagem de soluções

Motivações

- *“Qual a última versão da widget?”*
- *"Passa o novo script do dataset customizado no meu pen drive”*
- *"Espera eu exportar o processo e depois tu importa e edita”*
- *"Vou te mandar um link para baixar o projeto”*

Objetivos

- Centralizar soluções desenvolvidas
- Otimizar trabalho em equipe
- Histórico de versões
- Portfólio de excelência

Instalação

Download, configurações e setup inicial

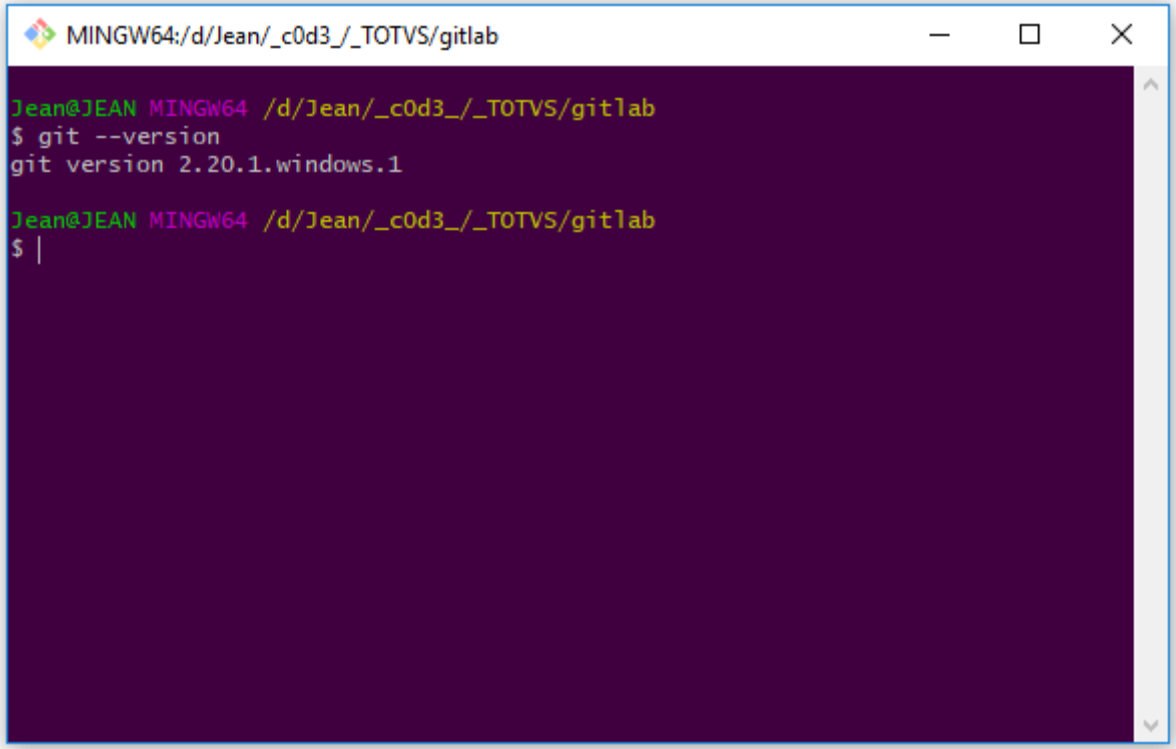
GitLab

- Acesse **gitlab.com**
- Crie uma conta na plataforma na opção *Register*
- Espere ser adicionado ao grupo da TOTVSRS no GitLab
- No grupo estarão todos os projetos já desenvolvidos

Download

- Acesse **git-scm.com**
- Baixe a versão de acordo com seu computador
- Em caso de dúvidas, consulte **git-scm.com/downloads**

Git Bash

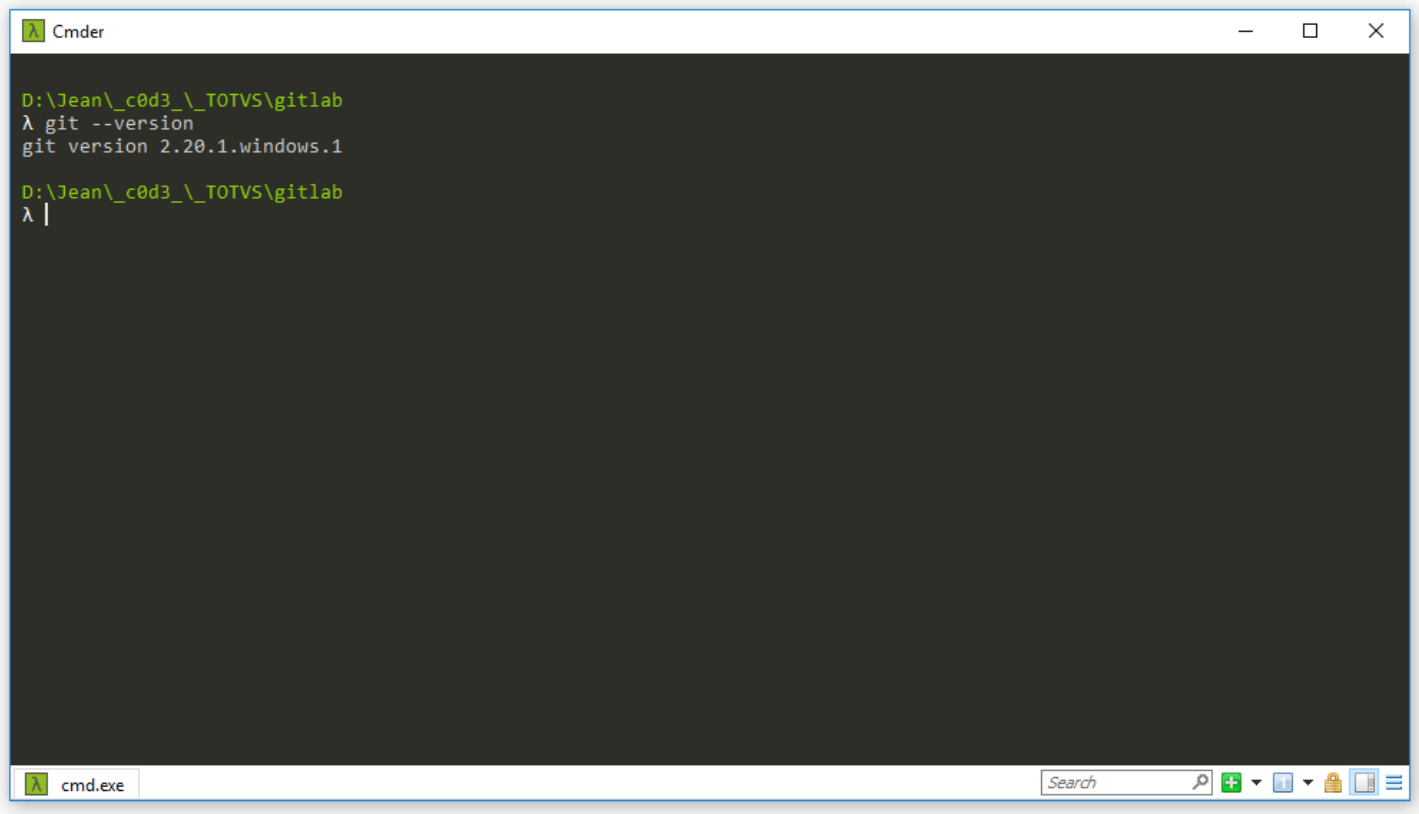


O **Git Bash** é o terminal que possibilita usar comandos do *git*

Cmder

- Cmder é um emulador de console com comandos de *Linux* para *Windows*
- É uma opção válida de utilização caso ache o Git Bash limitado
- É possível baixá-lo através do link **cmder.net**
- Escolha a opção **Download Full** para baixar a versão com comandos do Git

Cmder



Interface do **cmder**

Primeiros passos

Trabalhando com projetos no Git

Criar um novo projeto

- Acesse a página inicial do Git da TOTVSRS: **gitlab.com/company**
- Navegue até a pasta raiz
- Escolha **New subgroup** para criar uma pasta com o nome do cliente

- Entre na pasta e escolha **New project**

Configurar um projeto

- Escolha o nome do projeto
- Atente à palavras com acentuação no Project slug
- Escreva uma definição sucinta do projeto
- Deixe marcado como **Private** e **iniciar com o README**

Clonar um projeto

- Na página do projeto, clique em **Clone / Clone with HTTPS**
- Com o link copiado, escreva o comando no Git Bash / Cmder

```
$ git clone <path-copiado-do-projeto>
```

- Será criada uma pasta com o *slug* do projeto no diretório corrente

Sincronizar um projeto

- A pasta conterá um arquivo **.git** e um **README.md**
- Copie os dois arquivos para o *workspace* do Eclipse com a pasta do projeto Fluig
- Dentro da pasta, abra o Git Bash / Cmder e digite os comandos

```
$ git add .
$ git commit –m “<mensagem-do-commit>”
$ git push origin master
```

Atualizar um projeto

- Sempre que realizar alterações em um projeto você precisa enviar as informações para o servidor do GitLab (*push*)
- A sequência de comandos é parecida com o caso anterior
- Na pasta do projeto, abra o terminal e digite os comandos

```
$ git add .
$ git commit –m “<mensagem-do-commit>”
$ git push origin master
```

Comandos úteis

git <something>

Checar alterações

```
$ git status
```

- Sempre que realizar alterações em um projeto você precisa enviar as informações para o servidor do GitLab (*push*)
- A sequência de comandos é parecida com o caso anterior
- Na pasta do projeto, abra o terminal e digite os comandos
- O comando também informa em quantos commits o repositório local está a frente do servidor

Adicionar arquivos

```
$ git add “<arquivo>”
```

- Comando para vincular arquivos a um *commit*
- Deve ser informado todo o *path* do **arquivo**.
- Se informar uma **pasta**, adiciona todos os arquivos desta
- Se informar **.** adiciona todos os arquivos alterados

Salvar alterações no repositório

```
$ git commit -m "<mensagem-do-commit>"
```

- Ao criar um commit, a *branch atual* altera sua posição em relação às demais
- Todos os arquivos vinculados do commit integram esse novo estado
- **A mensagem do commit deve ser descritiva e sintetizar a ideia por trás da alteração**

Dicas de mensagem

- Escrever a sentença no formato **Verbo infinitivo + Objeto**
  - *Implementar validateForm()*
  - *Integrar formulário ao processo*
  - *Corrigir consulta de dataset*
  - *Estilizar header do layout*
- Evitar utilizar sentenças muito grandes
  - Implementar função para correção da consulta do dataset colleague no formulário auxiliar da widget
- Um commit é atribuído a uma

Dicas de mensagem

- Um commit é atribuído a uma tarefa macro e não a várias micros
  - Estruturar HTML, estilizar página e adicionar interação com JS

  - Estruturar HTML
  - Estilizar página
  - Adicionar interação com JavaScript

Receber arquivos do servidor

```
$ git pull origin <branch>
```

- Recebe todos os *commits* do servidor e envia para seu repositório local
- Se sua branch estiver em um nodo diferente da presente do servidor, podem ocorrer conflitos devido ao **merge**
  - *Alguns* arquivos são corrigidos automaticamente
  - Em outros, uma mensagem é listada dizendo que o merge automático falhou
  - Será necessário corrigir *manualmente* os arquivos, escolhendo e/ou ajustando os arquivos com a versão correta

Enviar arquivos ao servidor

```
$ git push origin <branch>
```

- Envia todos os *commits* locais para o servidor

- Necessário que a sua branch esteja sincronizada com o servidor
  - Caso contrário um *warning* será gerado e bloqueará o *push*
  - Utilize o comando **git pull (origin <branch>)**
  - Poderão ocorrer conflitos de arquivos modificados
  - Nesse caso, basta corrigir (escolhendo a versão certa) e realizar o *push* novamente

Desfazer alterações nos arquivos

```
$ git checkout "<nome-do-arquivo>"
```

- Desfaz as alterações no(s) arquivo(s) e pasta(s)
- Os arquivos modificados voltarão ao último nodo do repositório local

Desvincular do commit

```
$ git reset ”<nome-do-arquivo>”
```

- Remove o(s) arquivo(s) e pasta(s) vinculados ao *commit*

Outros comandos

- Foram listados aqui alguns dos comandos mais frequentes durante as diferentes etapas de desenvolvimento

- A documentação completa pode ser encontrada aqui:
  - **https://git-scm.com/docs/user-manual**

Branches

Trabalhando em paralelo

Branches

- Ao trabalhar em um projeto individualmente, é tolerável trabalhar na branch **master** diretamente
- Quando uma equipe está em um projeto, deve-se criar **branches**
- A **branch de integração** será a **dev** e cada usuário cria uma branch própria para desenvolvimento de suas atribuições

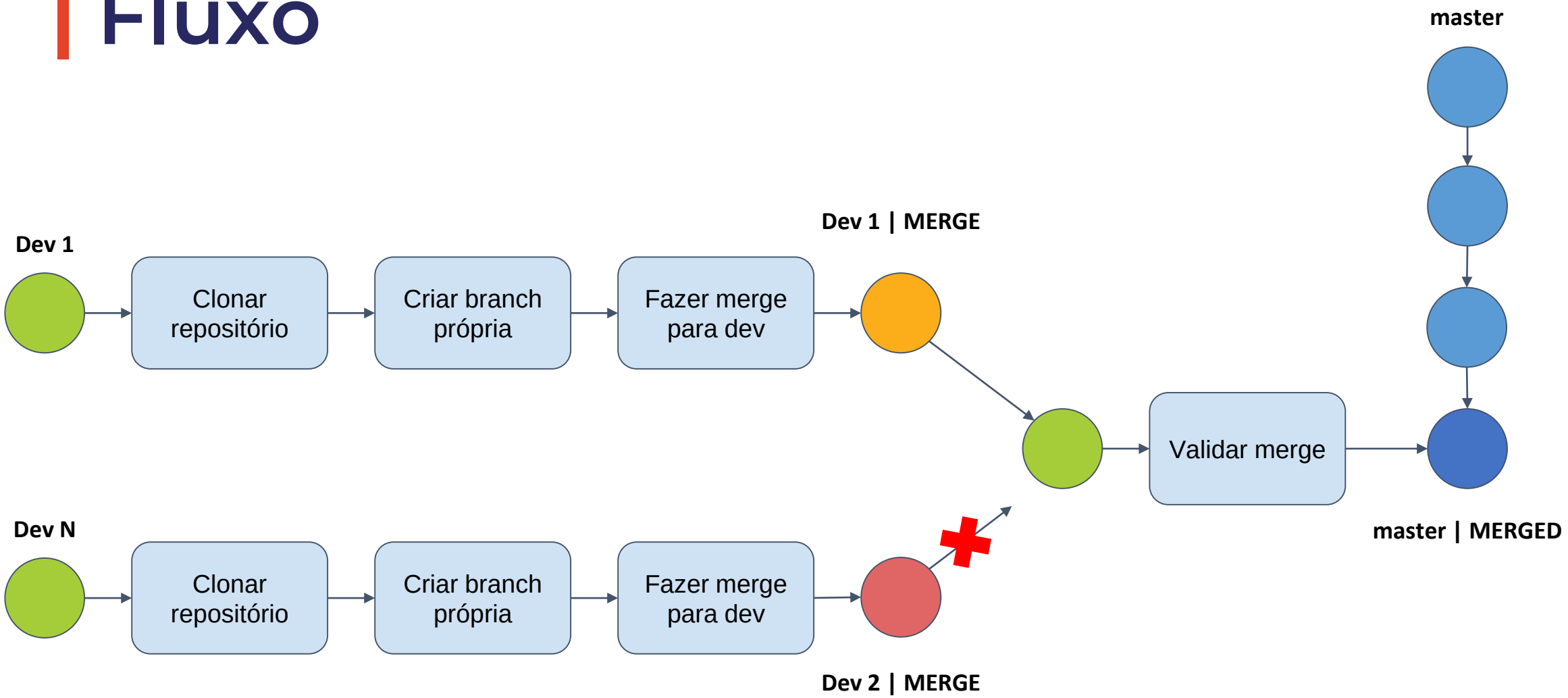
Fluxo

- Usuário **clona** repositório do projeto
- Repositório está apontando para branch master
- Criar e mudar para sua **branch própria** (ex.: *jean*)
- Desenvolver e testar funcionalidades
- **Commitar** alterações na branch pessoal

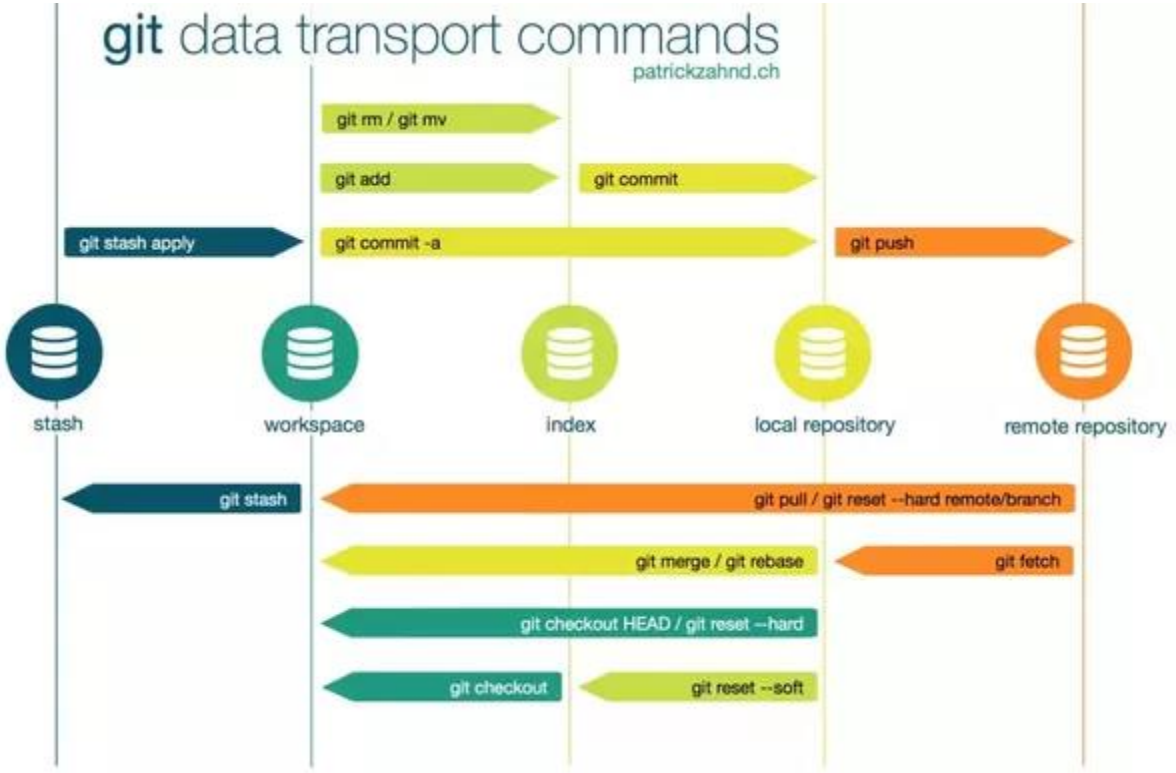
Fluxo

- Realizar um **merge** da branch pessoal com a **dev**
- Usuário responsável pela branch dev realiza **testes de integração**
- Após validação, fazer um push para a **master**
- Repositório está atualizado com versão pronta para produção
- Demais usuários devem sincronizar com última versão da master e caso ocorra conflitos com sua branch pessoal, corrigir antes de continuar o desenvolvimento

Fluxo


master
Dev 1
Dev 1 | MERGE
Clonar repositório
Criar branch própria
Fazer merge para dev
Validar merge
Dev N
Clonar repositório
Criar branch própria
Fazer merge para dev
Dev 2 | MERGE
master | MERGED

Fluxo completo + comandos



Merge request

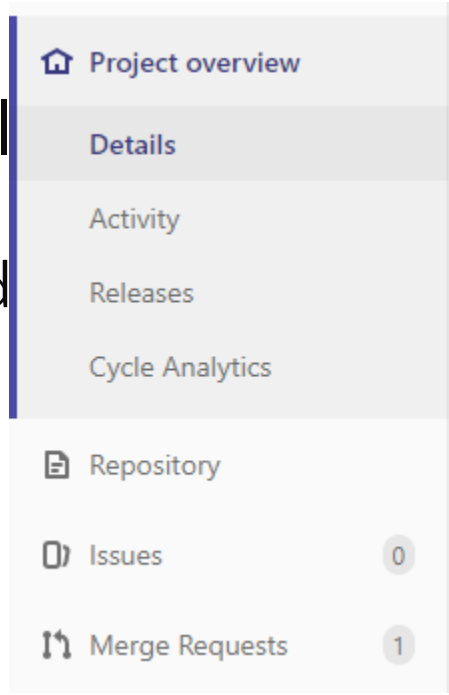
- Após finalizar uma funcionalidade, realizamos um *commit*

- Após um ou mais *commits* em branches auxiliares, precisamos fazer um **merge request** para uma branch principal
  - Branch *pessoal1* -> ... -> branch *pessoal2* ... -> *dev* -> **master**

- Unificar desenvolvimento paralelo no repositório do servidor!

Merge request

- Para realizar um pedido de merge, após o último commit na branch em que está trabalhando, vá ao **GitLab**
- No menu lateral do projeto, escol ção **Merge Requests**
  - O menu deixa listado quando ped ram realizados no repositório



Merge request

- Na nova página, serão listados todos os pedidos de merge existentes para o repositório
- No canto superior direito, clique no botão **New merge request**
- Em **source branch**, selecione a *branch* que você está trabalhando e deseja unificar à principal
- Em **target branch**, selecione a *branch* principal que será combinada com a sua

Merge request

- Na nova página, *ao menos*, defina um título e descrição para informar o responsável por merges do repositório

- Siga o **mesmo princípio** de um commit!
  - Mensagens informativas
  - Claridade e objetividade no texto

Permissões

- Por padrão, apenas usuários **Owner** e **Maintainer** podem commitar e realizar merges diretamente na branch master
- Usuários **Developer** precisam criar branches locais, trabalhar nelas e pedir um merge request para a branch master
- **Atenção:** plataforma GitLab não permite criar papéis de usuário por repositório
  - Um usuário owner será owner em todos os repositórios do grupo TOTVRS

Boas práticas

Organizar os repositórios

Convenções

- Cada **cliente terá um folder** no respectivo diretório
  - POA ou CAXIAS
- O repositório será um **projeto Fluig** com todas suas dependências
- Cada repositório deve possuir um **README.md** com dados gerais do projeto
- Cada componente do projeto (formulário, widget, processo) deve possuir um **README.md** com especificações detalhadas deste

Hierarquia de pastas

```
|- .git
|- datasets/
|- events/
|- forms/
|- mechanisms/
|- reports/
|- wcm/
|-- layout/
|-- widget/
|- workflow/
|- REAMDE.md
```

Hierarquia de pastas

```
|- .git
|- datasets/
|- events/
|- forms/
|-- MeuForm/
|--- MeuForm.html
|--- form.js
|--- REAMDE.md
|- mechanisms/
...
```

README.md

- O arquivo **README.md** é escrito sob a linguagem de marcação Markdown
- É comumente utilizado para guia de **documentações** e instruções de projetos em repositórios
- No nosso caso, usaremos-o para descrever **objetivos** e apresentar um apanhado geral dos projetos e componentes
- A sintaxe da linguagem Markdown pode ser conferida aqui

README.md

- Assim como um bom código, o arquivo README.md precisa ser bem escrito!
- Evite erros de português, abreviações e escrita coloquial
- Pense neste arquivo como o documento de comunicação entre *desenvolvedor-desenvolvedor*

- Assim como um bom código, o arquivo README.md precisa ser bem escrito!

Estrutura README.md

- Idealmente, o arquivo README.md da **raiz do projeto**, deve apresentar:
  - Lista de datasets
  - Lista de formulários
  - Lista de widgets
  - Lista de layouts
  - Lista de processos

Estrutura README.md

- Nos arquivos README.md de cada um dos **componentes** (formulários, processos, widgets, layouts):
  - Objetivos
  - Instruções de instalação e uso
  - Configurações existentes
  - Dependências
    - Bibliotecas externas
    - Outros componentes do projeto
  - Particularidades da aplicação
  - Autores
    - Contato para possíveis dúvidas

Estrutura README.md

- No caso de **datasets**, por serem um arquivo único em JavaScript, a recomendação é que se coloque a documentação como comentário no código

- Sugere-se, então:
  - Objetivo do dataset
  - Parâmetros de entrada (constraints)
    - Caso não possua nenhum, evidenciar com o tipo *NULL* ou *VOID*
  - Estrutura retornada (colunas das *rows*)

Simulação (novo projeto)

```
$ git clone ”https://gitlab.com/totvsrs/path-do-repositorio.git”
$ git checkout -b ”minhaBranch”
# Adiciona formulário e desenvolve alguns itens
$ git add ”forms/meu-formulario”
$ git commit -m ”Adicionar estrutura HTML do formulário”
# Desenvolve um pouco mais o formulário
$ git add ”forms/meu-formulario”
$ git commit -m ”Estilizar formulário”
```

Simulação (novo projeto)

```
# Desenvolve um pouco mais o formulário
$ git add "forms/meu-formulario"
$ git commit -m "Criar validação de formulário"
# Modela diagrama
$ git add "workflow/"
$ git commit -m "Adicionar diagrama do processo X"
```

Simulação (novo projeto)

```
# Desenvolve eventos de formulário / workflow
$ git add "forms/meu-formulario"
$ git commit -m "Desenvolver displayFields()"
$ git add "workflow/scripts/"
$ git commit -m "Adicionar validação de anexo enviado"
# Envia para o servidor (poderia ser feito a cada commit)
$ git push origin "minhaBranch"
```

Simulação (continuação de projeto)

```
$ git pull origin master
$ git pull origin ”branchQueTrabalharei”

# Fluxo semelhante ao exibido: add, commit, push...
$ git add ”forms/meu-formulario”
$ git commit -m ”Adicionar estrutura HTML do formulário”
$ git push origin add ”branchQueTrabalharei”
```

Próximos passos

Melhoria e desenvolvimento contínuo

Trabalhos futuros

- A medida que a cultura da utilização do Git crescer, é possível usufruir de mais ferramentas que plataforma oferece

- Organizar entregas de projetos em **sprints**

- Utilizar **issues** e **milestones** para métricas e atribuição de tarefas

Dúvidas, sugestões, críticas?

Estou à disposição para conversar e ajudar no que for preciso! :)

Igor.rodrigues@totvs.com.br

Obrigado!

Igor Rodrigues
Líder Desenvolvimento
Igor.rodrigues@totvs.com.br