Artigo
Eduard Lebedyuk · Nov. 22, 2021 8min de leitura

Entrega contínua de sua solução InterSystems usando GitLab – Parte I: Git

Todo mundo tem um ambiente de teste.

Algumas pessoas têm a sorte de ter um ambiente totalmente separado para executar a produção.

-- Desconhecido

Nesta série de artigos, gostaria de apresentar e discutir várias abordagens possíveis para o desenvolvimento de software com as tecnologias InterSystems e GitLab. Vou cobrir tópicos como:

- Git Básico
- Fluxo Git (processo de desenvolvimento)
- Instalação do GitLab
- Fluxo de Trabalho do GitLab
- GitLab CI/CD
- CI/CD com contêineres

Esta primeira parte trata do pilar do desenvolvimento de software moderno - sistema de controle de versão Git e vários fluxos Git.

Git Básico

Embora o tópico principal que iremos discutir seja o desenvolvimento de software em geral e como o GitLab pode nos capacitar nesse esforço, o Git, ou melhor, os váriosconceitos de alto nível subjacentes no design do Git, são importantes para o melhor entendimento de conceitos posteriores.

Dito isso, o **Git** é um sistema de controle de versão, baseado nessas ideias (existem muitas outras, essas são as mais importantes):

- **Desenvolvimento não linear** significa que enquanto nosso software é lançado consequentemente da versão 1 para a 2 para a 3, sob a mesa a mudança da versão 1 para a 2 é feita em paralelo - vários desenvolvedores desenvolvem uma série de recursos/correções de bugs simultaneamente.
- **Desenvolvimento distribuído** significa que o desenvolvedor é independente de um servidor central ou de outros desenvolvedores e pode desenvolver facilmente em seu próprio ambiente.
- **Fusão** - as duas ideias anteriores nos levam à situação em que muitas versões diferentes da verdade existem simultaneamente e precisamos uni-las de volta em um estado completo.

Agora, não estou dizendo que Git inventou esses conceitos. Não. Em vez disso, o Git os tornou fáceis e populares e isso, juntamente com várias inovações relacionadas, ou seja, infraestrutura como código/conteinerização mudou o desenvolvimento de software.

Termos básicos do Git

Repositório é um projeto que armazena dados e metainformações sobre os dados.

- O repositório "físico" é um diretório em um disco.
- O repositório armazena arquivos e diretórios.
- O repositório também armazena um histórico completo de alterações para cada arquivo.

O repositório pode ser armazenado:

- Localmente, em seu próprio computador
- Remotamente em um servidor remoto

Mas não há nenhuma diferença particular entre repositórios locais e remotos do ponto de vista do git.

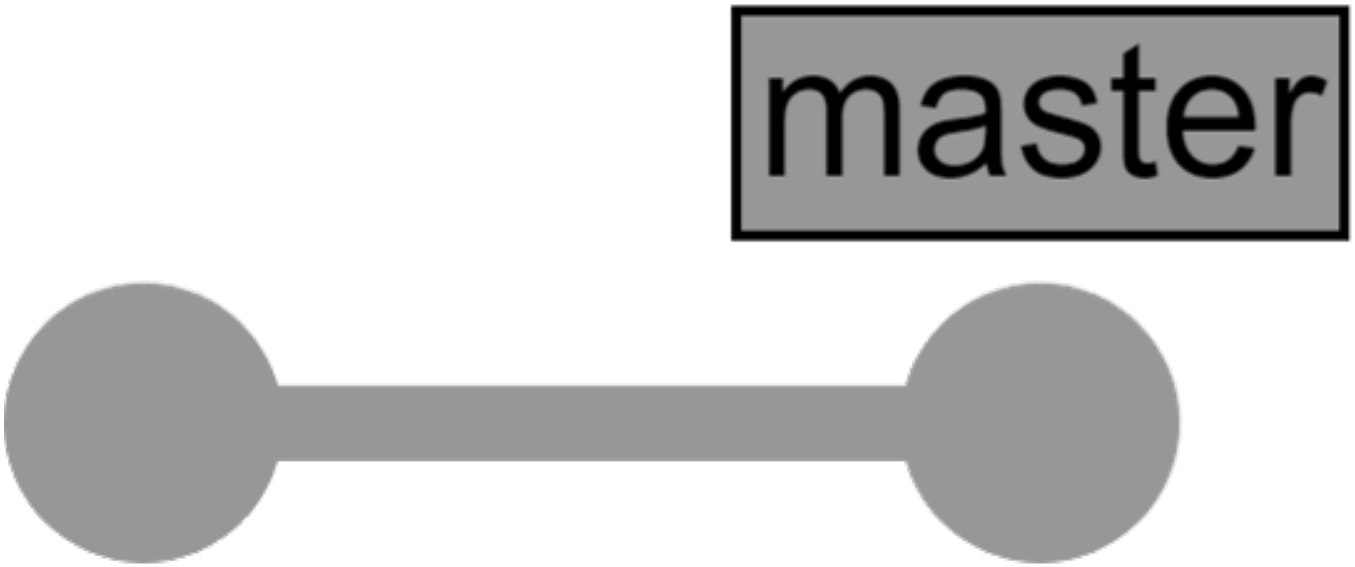
Commit é um estado fixo do repositório. Obviamente, se cada commit armazenasse o estado completo do repositório, nosso repositório cresceria muito rapidamente. É por isso que um commit armazena um **diff** que é uma diferença entre o commit atual e seu **commit pai**.

Commits diferentes podem ter um número diferente de pais:

- 0 - o primeiro commit no repositório não tem pais.
- 1 - conforme o habitual - nosso commit mudou algo no repositório como era durante o commit pai
- 2 - quando temos dois estados diferentes do repositório, podemos uni-los em um novo estado. E esse estado e esse commit teriam 2 pais.
- >2 - pode acontecer quando unimos mais de 2 estados diferentes do repositório em um novo estado. Não seria particularmente relevante para nossa discussão, mas existe.

Agora, para um pai, cada commit diferente é chamado de **commit filho**. Cada commit pai pode ter qualquer número de commits filhos.

Branch é uma referência (ou ponteiro) para um commit. Veja como funciona:

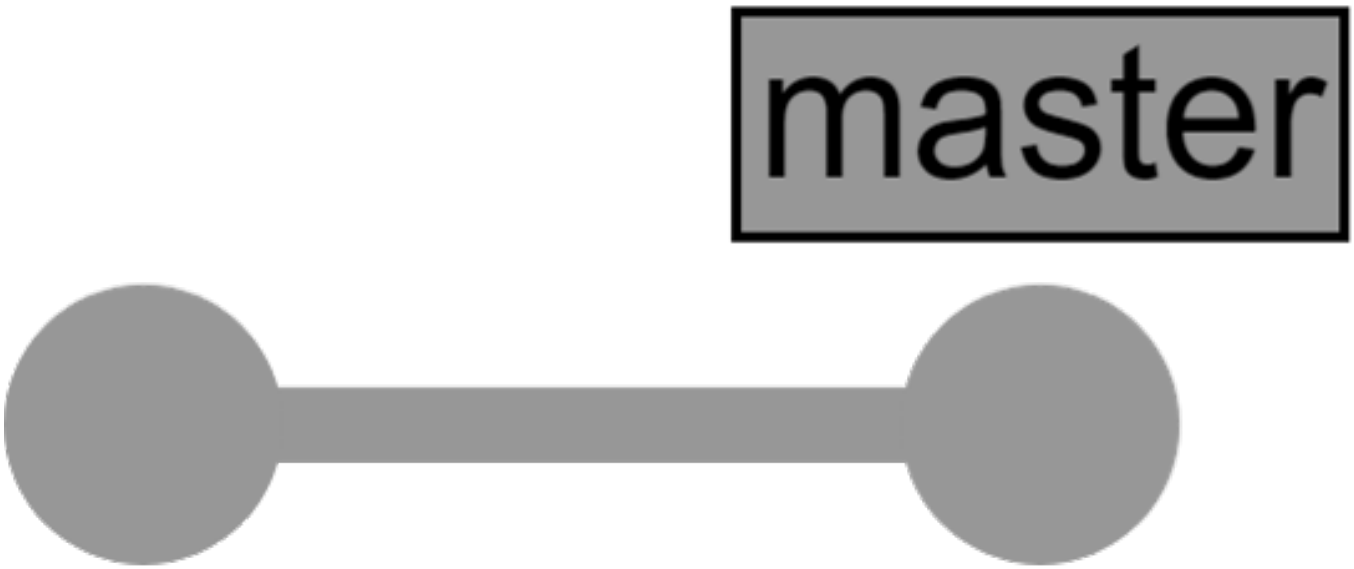


Nesta imagem, podemos ver o repositório com dois commits (círculos cinza), o segundo é o head do branch master. Depois de adicionar mais commits, nosso repositório começa a ficar assim:

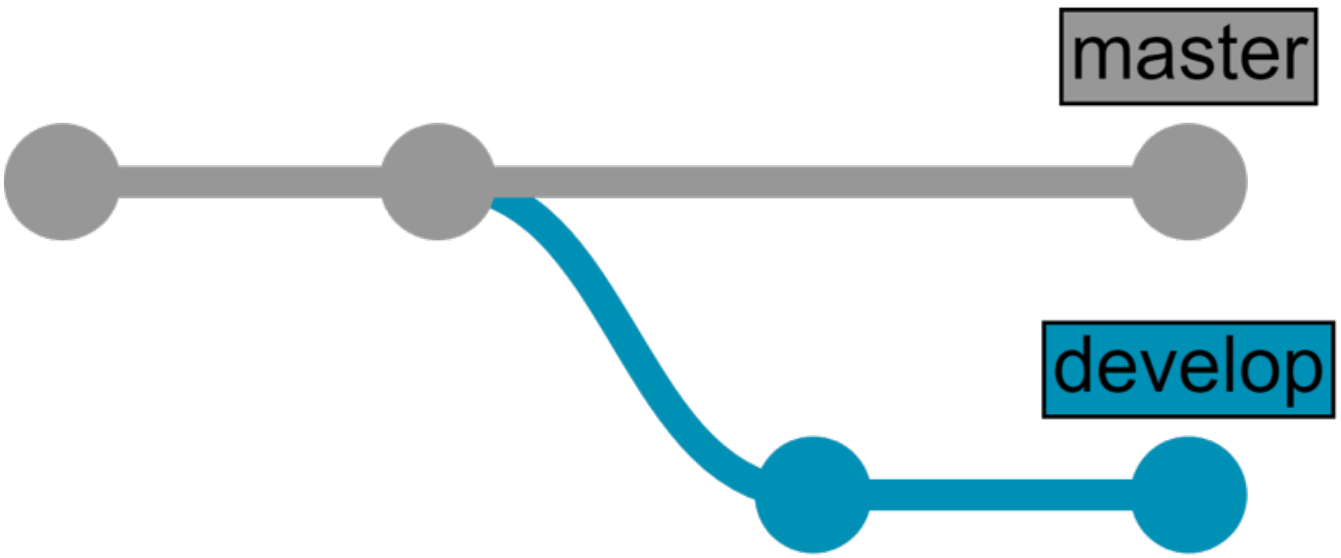
Esse é o caso mais simples. Um desenvolvedor trabalha em uma mudan ç a de cada vez. No entanto, normalmente, existem muitos desenvolvedores trabalhando simultaneamente em diferentes recursos e precisamos de uma **árvore de commit** para mostrar o que está acontecendo em nosso repositório.

Árvore de commit

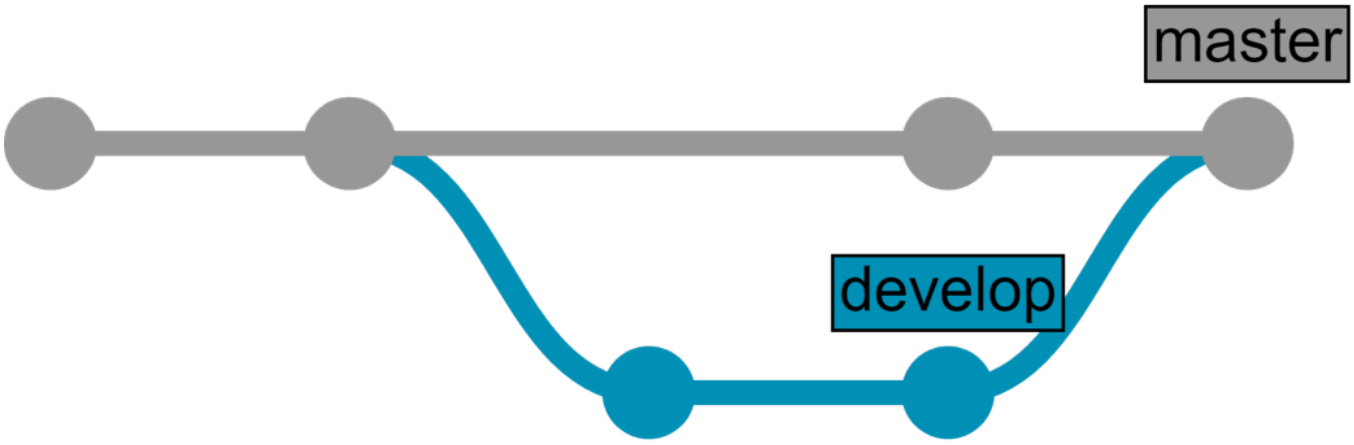
Vamos come ç ar do mesmo ponto de partida. Aqui está o repositório com dois commits:



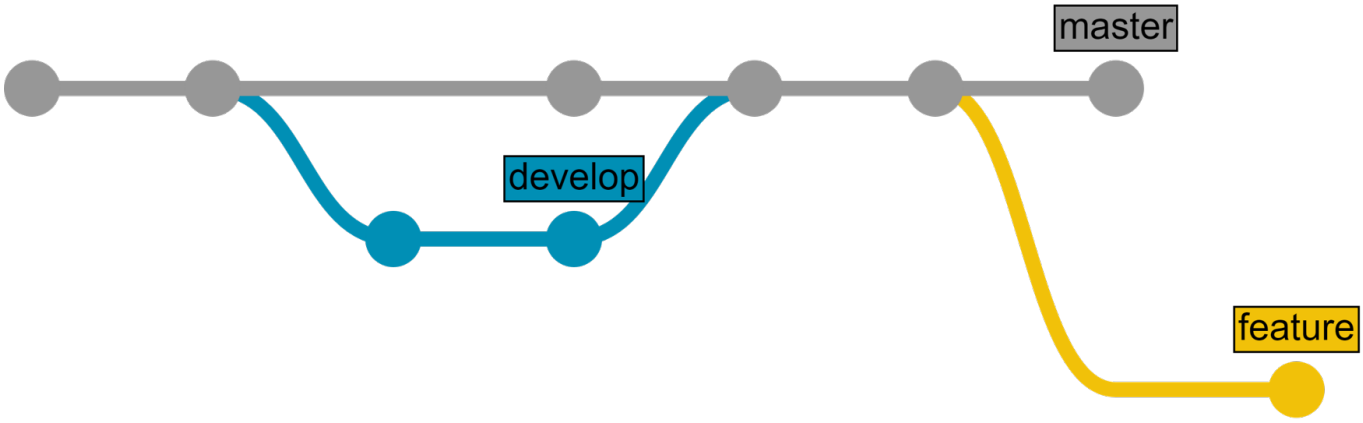
Mas agora, dois desenvolvedores estão trabalhando ao mesmo tempo e para não interferir um no outro, eles trabalham em branches separados:



Depois de um tempo, eles precisam unir as altera ç ões feitas e para isso eles criam uma**solicita ç ão de mesclagem (merge)** (também chamada de **pull request**) - que é exatamente o que parece - é uma solicita ç ão para unir dois estados diferentes do repositório (no nosso caso, queremos mesclar o develop branch no master branch) em um novo estado. Depois de ser devidamente revisado e aprovado, nosso repositório fica assim:



E o desenvolvimento continua:



Resumo - Git Básico

Conceitos principais:

- **Git** é um sistema de controle de versão distribuído não linear.
- **Repositório** armazena dados e metainforma ç ões sobre os dados.
- **Commit** é um estado fixo do repositório.
- **Branch** é uma referência para um commit.
- **Solicita ç ão de mesclagem**(também chamada de **pull request**) - é uma solicita ç ão para unir dois estados diferentes do repositório em um novo estado.

Se você quiser ler mais sobre o Git, existem livros disponíveis.

Fluxos Git

Agora que o leitor está familiarizado com os termos e conceitos básicos do Git, vamos falar sobre como a parte do desenvolvimento do ciclo de vida do software pode ser gerenciada usando o Git. Existem várias práticas (chamadas de fluxos) que descrevem o processo de desenvolvimento usando Git, mas vamos falar sobre duas delas:

- Fluxo do GitHub
- Fluxo do GitLab

Fluxo do GitHub

O fluxo do GitHub é tão fácil quanto parece. Aqui está:

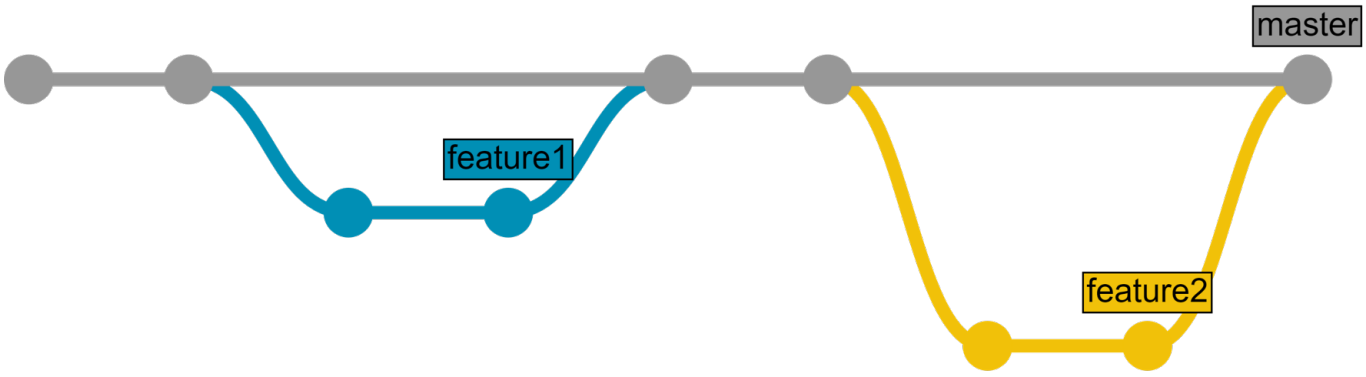
1. Crie um branch (ramificação) do repositório.
2. Commit suas alterações para seu novo branch
3. Envie um pull request do seu branch com as alterações propostas para iniciar uma discussão.
4. Commit mais alterações em seu branch conforme necessário. Seu pull request será atualizado automaticamente.
5. Mescle o pull request assim que o branch estiver pronto para ser mesclado.

E existem várias regras que devemos seguir:

- master branch é sempre implantável (e funcionando!)
- Não há desenvolvimento indo diretamente para o master branch
- O desenvolvimento está acontecendo nos branches de recursos
- master == ambiente** de produção*
- Você precisa implantar na produção o mais rápido possível
    - Não confunda com "Produções Ensemble", aqui "Produção" significa SISTEMA EM PRODUÇÃO.

** Ambiente é um local configurado onde seu código é executado - pode ser um servidor, uma VM, até mesmo um contêiner.

Veja como funciona:



Você pode ler mais sobre o fluxo do GitHub aqui. Também há um guia ilustrado.

O fluxo do GitHub é bom para pequenos projetos e para testes se você está começando com os fluxos do Git. No entanto, o GitHub o usa, portanto, também pode ser viável em projetos grandes.

Fluxo do GitLab

Se você não estiver pronto para implantar na produção imediatamente, o fluxo do GitLab oferece um fluxo do GitHub + ambientes. É assim que funciona - você desenvolve em branches de recursos, como acima, mescla

(merge) no master, como acima, mas aqui está uma diferen ç a: o master é igual apenas no ambiente de teste. Além disso, você tem "Branches de ambiente" que estão vinculados a vários outros ambientes que você possa ter.

Normalmente, existem três ambientes (você pode criar mais se precisar):

- Ambiente de teste == master branch
- Ambiente de pré-produ ç ão == preprod branch
- Ambiente de produ ç ão == prod branch

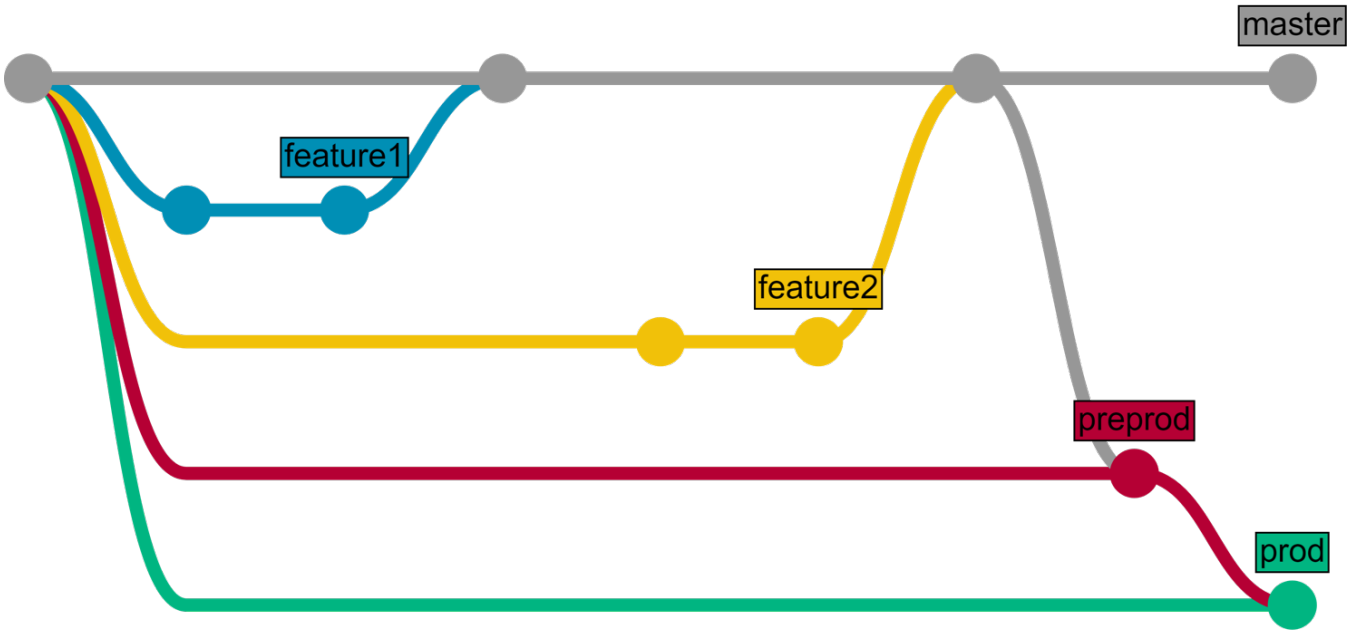
O código que chega em um dos branches do ambiente deve ser movido para o ambiente correspondente imediatamente, isso pode ser feito:

- Automaticamente (cobriremos isso nas partes 2 e 3)
- Parcialmente automático (igual ao automaticamente, exceto que um botão que autoriza a implanta ç ão deve ser pressionado)
- Manualmente

Todo o processo é assim:

1. O recurso é desenvolvido no branch de recursos.
2. O branch de recurso é revisado e mesclado no master branch.
3. Depois de um tempo (vários recursos mesclados), o master é mesclado com o preprod
4. Depois de um tempo (teste do usuário, etc.), o preprod é mesclado com o prod
5. Enquanto estávamos mesclando e testando, vários novos recursos foram desenvolvidos e mesclados no master, então vá para parte 3.

Veja como funciona:



Você pode ler mais sobre o fluxo do GitLab aqui.

Conclusão

- Git ****é um sistema de controle de versão distribuído não linear.
- O fluxo Git pode ser usado como uma diretriz para o ciclo de desenvolvimento de software; existem vários

que você pode escolher.

Links

- Livro Git
- Fluxo do GitHub
- Fluxo do GitLab
- Fluxo Driessen (fluxo mais abrangente, para compara ç ão)
- Código para este artigo

Questões para discussão

- Você usa um fluxo git? Qual?
- Quantos ambientes você tem para um projeto padrão?

O que vem a seguir

Na próxima parte, iremos:

- Instalar o GitLab.
- Falar sobre alguns ajustes recomendados.
- Discutir o fluxo de trabalho do GitLab (não deve ser confundido com o fluxo do GitLab).

Fique ligado.

#Administra ç ão do Sistema #Conteineriza ç ão #Docker #Gestão da Mudan ç a #Git #Implanta ç ão #Iniciante #Integra ç ão Contínua #Caché