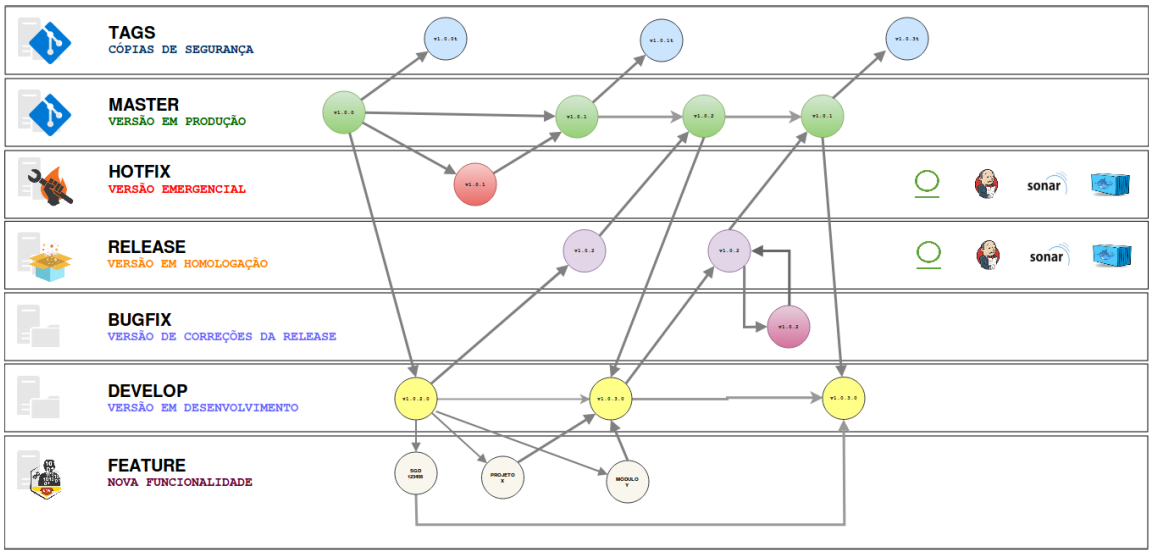
Fluxo de Trabalho - GITLAB/GETIC

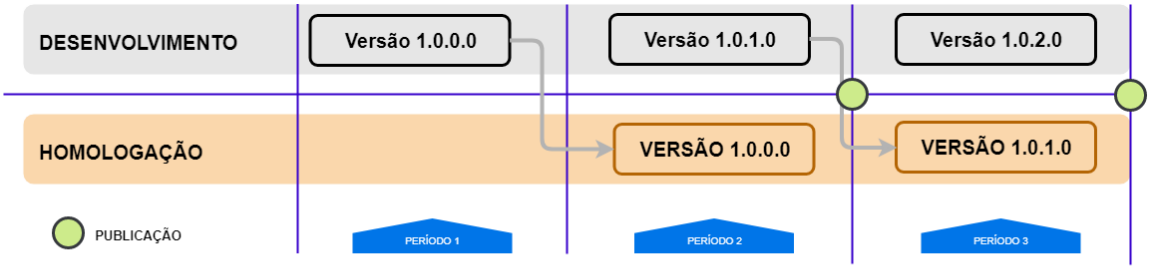
Este documento visa apresentar o processo definido para gestão de versões a partir da explanação do fluxo definido pela GETIC para o uso GITLAB como ferramenta de controle de versão.



Branches

branch	FUNÇÃO	PERMISSÕES
master	É o branch principal de fluxo de trabalho. Representa a versão inicial do projeto e ainda a versão em produção.	Master
release	É o branch do fluxo de trabalho que representa a versão em testes ou homologação, ou seja, a próxima versão a ser publicada em produção. Deste branch será criado o branch **bugfix**.	Master
develop	É o branch do fluxo de trabalho que representa a versão em desenvolvimento, ou seja, a próxima versão a ser homologada. Deste branch serão criados os branches de **features**. Ele deverá ser atualizado sempre que uma feature for finalizada.	Master/Developer
hotfix	É o branch do fluxo de trabalho que representa a **versão emergencial**, ou seja a versão a ser publicada com o proprósito de reestabelecer o funcionamento da última versão lançada. Nenhum outro branch deverá ser criado a partir de um branch hotfix.	Master/Developer
feature	É o branch do fluxo de trabalho destinado ao desenvolvimento. Cada funcionalidades deverá ter o seu próprio branch oriundo do branch **develop**. Estes branches somente deverão ser incorporados a origem quando da conclusão da demanda. Nenhum outro branch deverá ser criado a partir de um branch **feature**.	Master/Developer
bugfix	É o branch do fluxo de trabalho destinado às implementações de correções da versão em homologação. Nenhum branch deverá ser criado a partir de um branch **bugfix**.	Master/Developer

Planejamento das Versões

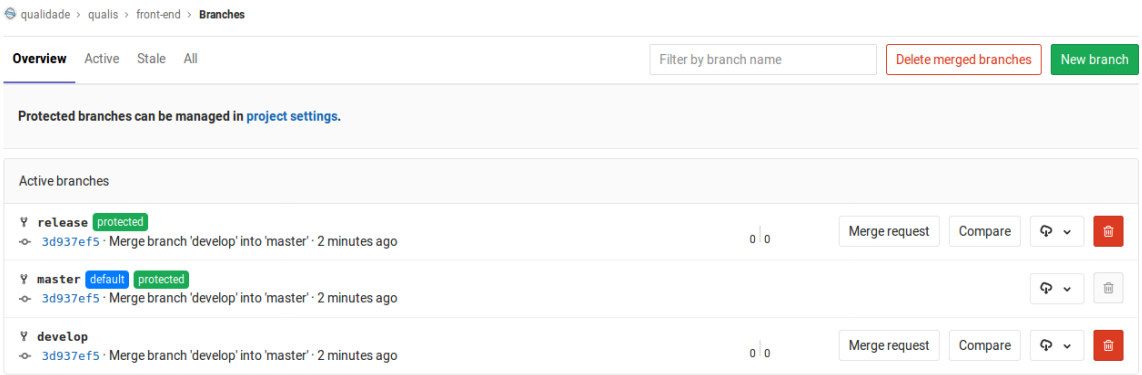


O planejamento de uma sprint deverá considerá períodos de 7 ou 15 dias para as fases de desenvolvimento e homologação. Ao final do período de desenvolvimento haverá uma movimentação de branches, incorporando-se o branch de desenvolvimento (**develop**) ao branch de homologação (**release**). Haverá sempre uma versão em desenvolvimento e outra em homologação.

Um sistema terá um dia específico da semana para a sua publicação. Desta forma na manhã do dia seguinte à publicação do sistema acontecerá a movimentação de branches.

A equipe de qualidade realizará a movimentação somente para os sistema sob sua responsabilidade. Os líderes técnicos dos demais sistemas ficarão com esta missão.

Estrutura inicial do projeto



Iniciando o desenvolvimento

O desenvolvedor deverá verificar junto à equipe de qualidade a sua permissão ao grupo de projetos da equipe no qual o projeto pertence. Deve ainda disponibilizar em seu perfil a sua chave **ssh** para só então realizar o clone do projeto em sua máquina.

Acesse GITLAB Profile [https://srvgit.int.cagece.com.br/profile/keys] para informar chave ssh.

Clonando o repositório do projeto

Um projeto poderá ser clonado utilizando os comandos do **git** ou ainda através da IDE de sua preferência. A Figura acima apresenta, no **Passo 1**, um projeto que ao ser clonado disponibiliza um repositório local do branch **master**.

> Exemplo:

```
git clone git@srvgit.int.cagece.com.br:qualidade/qualis/front-
end.git
```

branch local de trabalho

Todo o desenvolvimento deverá ser realizado em um branch do tipo **feature** criado a partir do branch **develop**. A criação do branch local poderá ser feito utilizando os comandos do **git** ou ainda através da IDE de sua preferência.

O nome de um branch de trabalho deverá respeitar o seguinte padrão:

> Sintaxe:

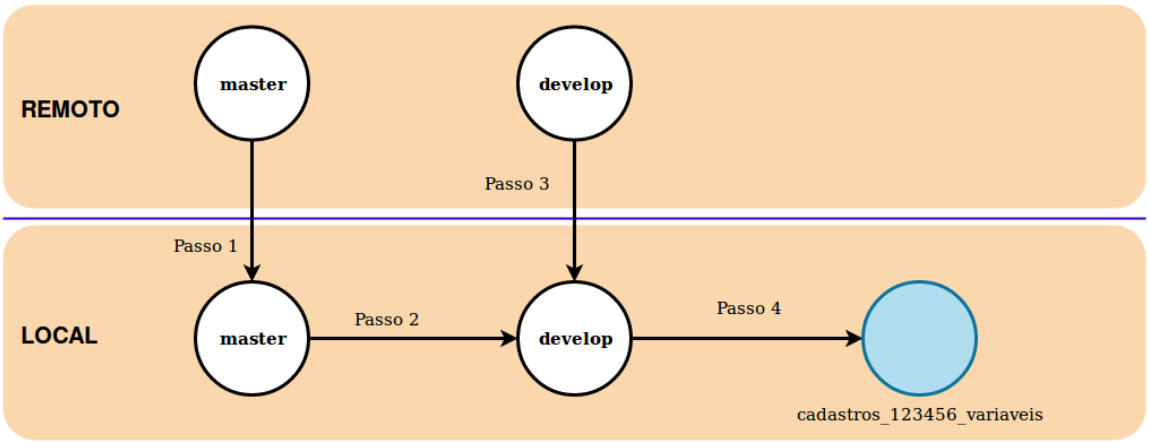
```
<MÓDULO>_<SGD>_<IDENTIFICAÇÃO>
```

> Exemplo:

```
- atendimento_123456_solicitacao_corte
- cobranca_755215_ajuste_prazos
- faturamento_652542_nova_tarifa
```

> Sintaxe:

```
git checkout develop -b <NovoBranche>
```



Para criar um branch local de trabalho:

1. Clone o projeto. (passo 1)
2. Mude para o branch **develop**. (passo 2)
3. Atualize-o branch **develop** local. (passo 3)
4. Crie um branch de trabalho local(**feature**) (passo 4)

> Exemplo:

```
git checkout develop
```

```
git pull
git checkout develop -b cadastros_123456_variaveis
```

Padrão de Commits

Durante a implementação de um funcionalidade realize commits parciais, não espere a finalização da atividade para enviar para o branch remoto.

Os commits devem respeitar o seguinte padrão:

> Sintaxe:

```
git add .
git commit -m "SGD 9999999 [%] <Descrição da implementação
realizada>"
                      |     |
                      |     |----------> Percentual (andamento da
tarefa)
                      |
                      |----------------> Nº do SGD
```

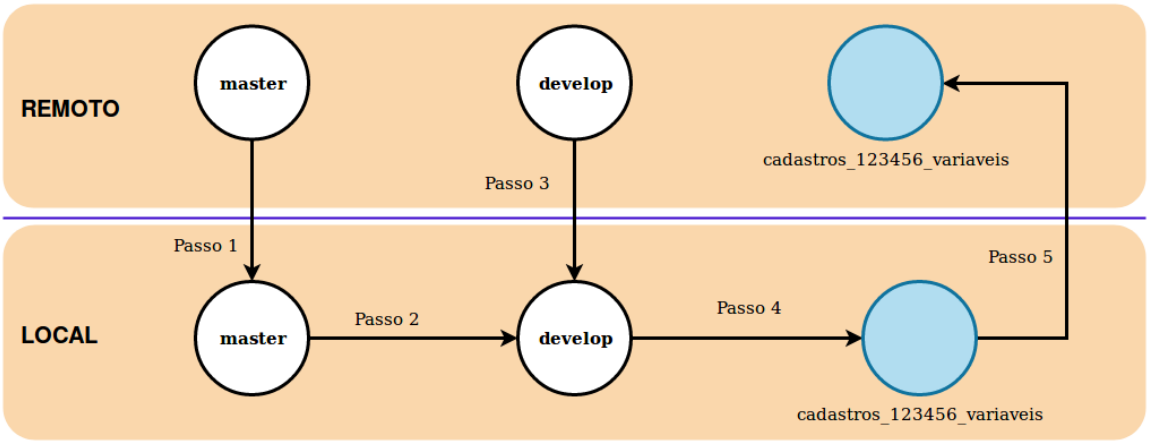
> Exemplo:

```
- git commit -m "SGD 123456 [50%] Implementado o cadastro básico de
sistemas"
- git commit -m "SGD 654321 [75%] Implementadas correções das
fórmulas X"
- git commit -m "SGD 123456 [50%] Finalizada a implementação do
cadastro"
```

branch Remoto de trabalho

Os branches de trabalho local precisam ser enviados para o servidor **git**.

A primeira vez que for realizar **push** do branch de trabalho local, para enviá-lo para criar o branch remoto no servidor git, é preciso adicionar o parâmetro **--set-upstream** informando a origem ao comando.



> Sintaxe:

```
git push --set-upstream origin <NovoBranche>
```

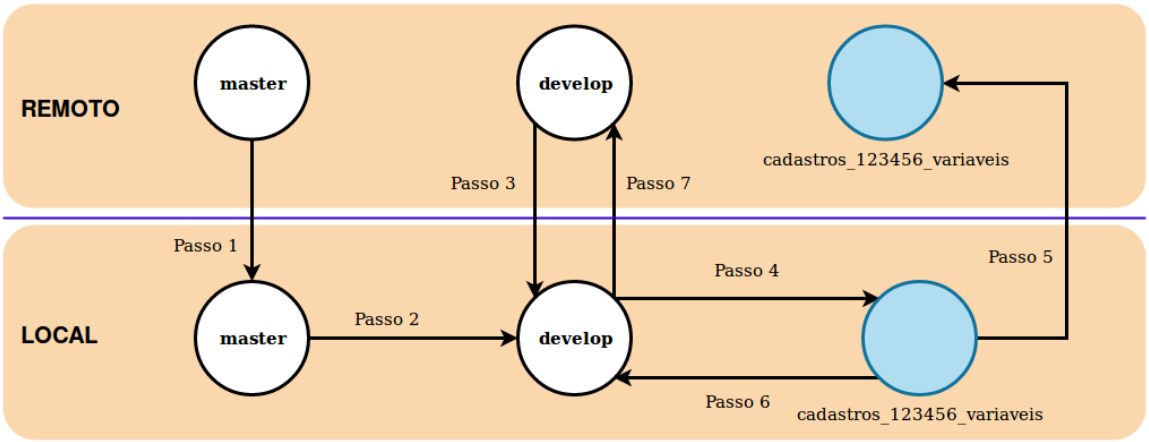
> Exemplo:

```
git push --set-upstream origin cadastros_123456_variaveis (passo 5)
```

> **Importante lembrar**:
>
> O parâmetro **--set-upstream** deverá ser utilizado apenas para a criação do branch remoto.

Realizando merge para o branch **develop**

Após a conclusão do trabalho em um branch **feature** é necessário disponibilizá-lo para a versão em desenvolvimento. Este processo é realizado a partir de execução um **merge** do branch desejado com o branch **develop**.



Para realizar um merge com o branch **develop**:

1. Mude para o branch **develop**. (passo 2)
2. Atualize-o branch **develop** local. (passo 3)
3. Realize merge do branch **feature** (passo 6)
4. Realize commit com o branch **develop** remoto. (passo 7)

> Sintaxe:

```
git checkout develop
git pull
git merge <NovoBrancheFeature>
git push
```

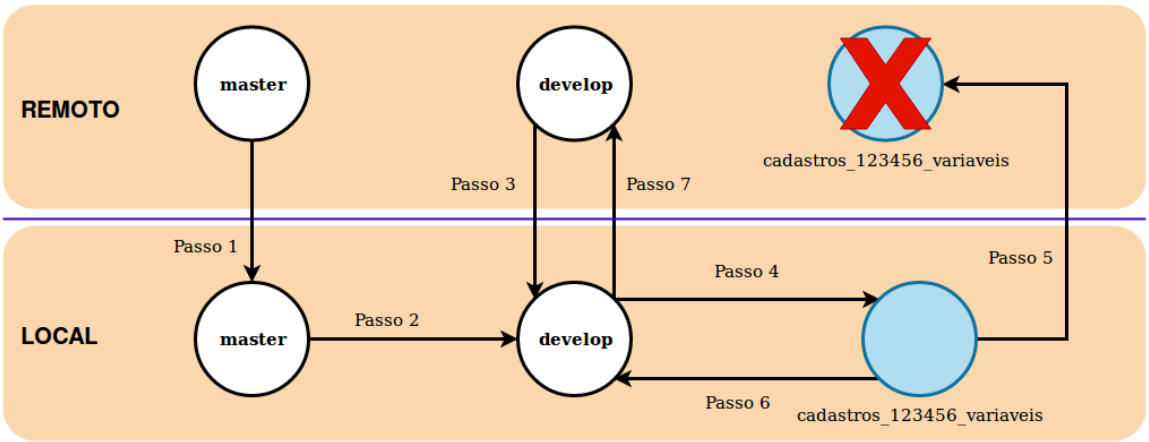
> Exemplo:

```
git checkout develop
git pull
git merge cadastros_123456_variaveis
git push
```

> **FIQUE ATENTO !!!**:
>
> Ao atualizar o branch **develop** local poderão surgir conflito. Estes conflitos precisarão ser resolvidos antes de enviar a atualização para o branch **develop** remoto.

Excluíndo branch de trabalho remoto

Os branches **features** precisarão ser eliminados após a realização do merge com o branch **develop**. A exclusão poderá ser feita no momento da realização do **push** ou não.



Para exclusão do branch no momento da realização do **push**, é preciso acrescentar o parâmetro **--delete** e a origem.

> Sintaxe:

```
git checkout develop
git pull
git merge <BrancheFeature>
git push --delete origin <BrancheFeature>
```

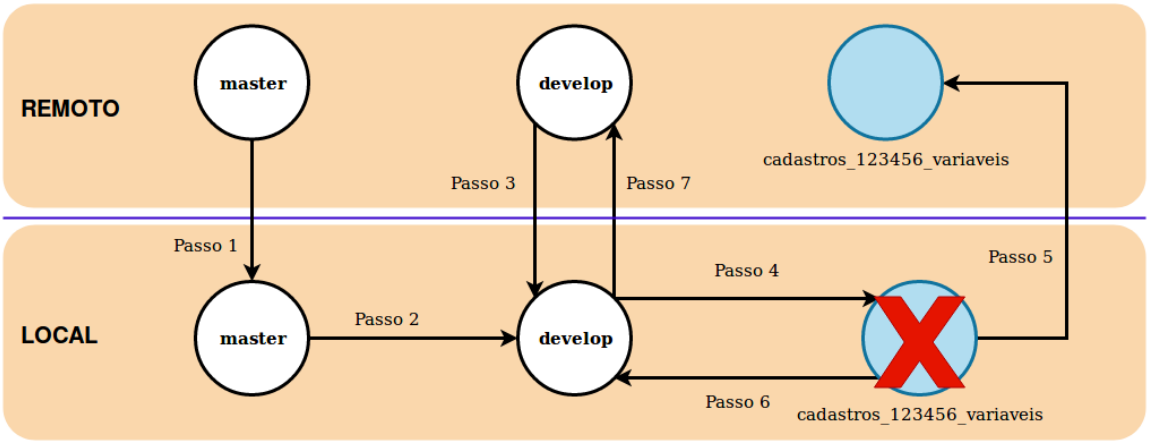
> Exemplo:

```
git checkout develop
git pull
git merge <NovoBrancheFeature>
git push --delete origin cadastros_123456_variaveis
```

Excluíndo branch de trabalho local

Os branches locais do tipo **features** precisarão ser eliminados após a realização

do merge com o branch **develop**. A exclusão deverá ser realizada de forma manual.



Para exclusão individual de branch você poderá excutar o comando utilizando o parâmentro "-D".

> Sintaxe:

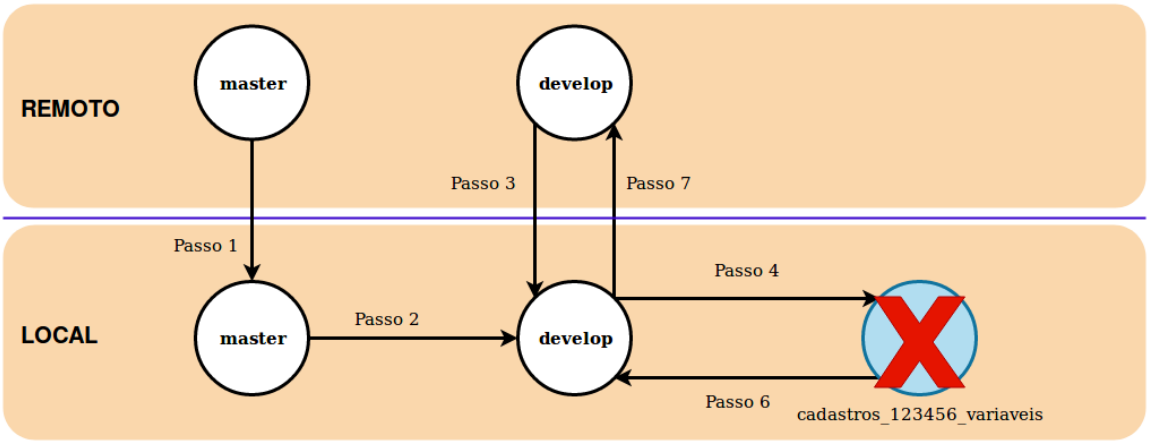
```
git branch -D <nome do branch>
```

> Exemplo:

```
git branch -D cadastros_123456_variaveis
```

Excluíndo branch de trabalho local órfãos

Os branches locais órfãos são aqueles que não mais possuem um branch remoto. O git permite a consulta e exclusão destes branches. O que diferencia a consulta da exclusão é a utilização do parâmetro **--dry-run** junto ao comando **prune**.



> Listando branch órfãos:

```
git remote prune origin --dry-run
```

> Excluíndo branch órfãos:

```
git remote prune origin
```

Uma vez excluído um branch remoto no momento da realização do push, o comando prune poderia ser utilizado para eliminar o branch local.

Gerando versões

Neste tópico serão detalhados os processos para geração das versões permitidas pelo Fluxo de Trabalho - GITLAB/GETIC.

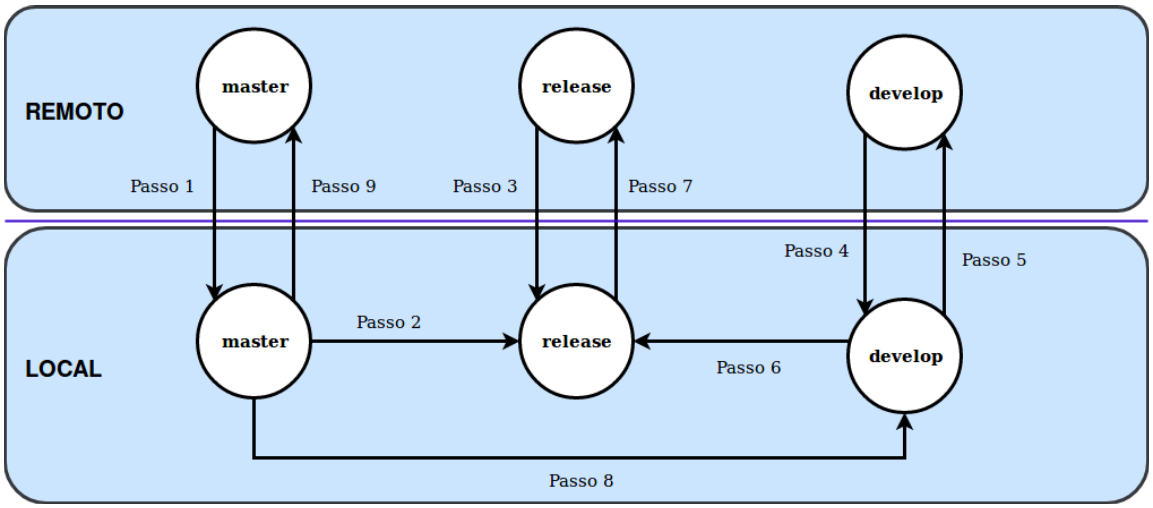
Os tipos de versões permitidos pelo processo GITLAB/GETIC são:

Tipo	Descrição
Normal	Versão gerada a partir do branch **release**
Emergencial	Versão gerada a partir do branch **hotfix**

Versão Normal (Release)

Ao fim de cada período de desenvolvimento, conforme detalhado no tópico Planejamento das Versões [#planejamento-das-versoes], o branch **develop** é incorporado ao branch **release**. Este processo é realizado pela equipe de qualidade para os projetos sob a o processo de controle de qualidade ou pelo líder do projeto para os demais projetos.

Podem ser utlizados os comando do git ou ainda o próprio Gitlab para a realização do merge do branch **develop** com o branch **release**.



Realizando merge através do git:

1. Alterne para o branch **develop**.
2. Atualize branch **develop**. (passo 4)
3. Alterne para o branch **release**.
4. Atualize o branch **release**. (passo 3)
5. Realize o merge. (passo 6)

> Exemplo:

```
git checkout develop
git pull (passo 4)
git checkout release
```

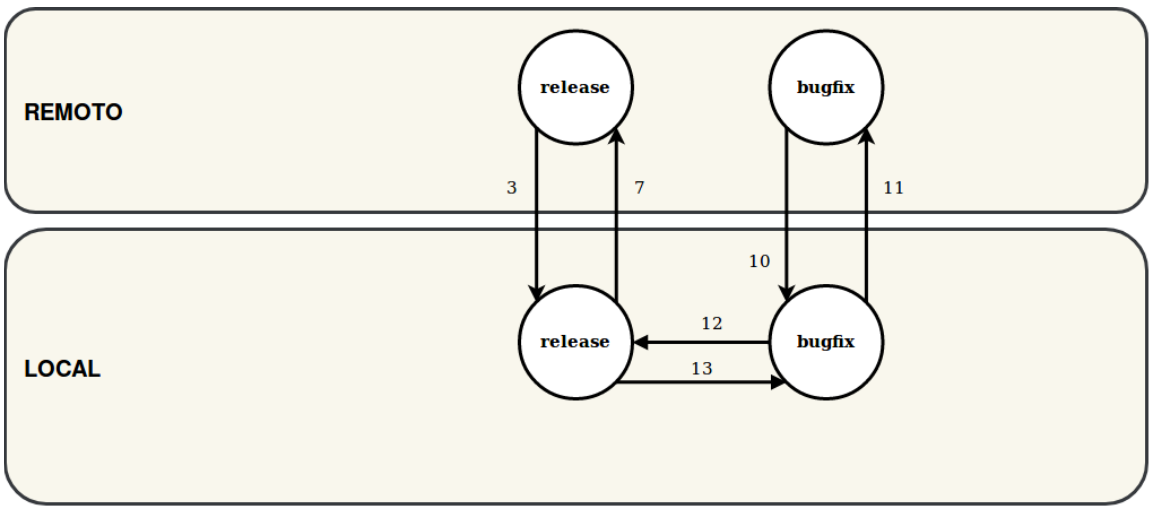
```
git pull (passo 3)
git merge develop (passo 6)
git push (passo 7)
```

> **FIQUE ATENTO !!!**:
>
> Ao atualizar o branch **release** local poderão surgir conflito. Estes conflitos precisarão ser resolvidos antes de enviar a atualização para o branch **release** remoto.

Versão de Correção (Bugfix)

Durante o processo de homologação de uma versão, se faz necessária, na maioria das vezes, a implementação de correções de bugs, falhas, erros e melhorias. Para não haver indisponibilidade da versão em testes, o branch **bugfix** deve ser criado com o objetivo específico para ajustes.

Um branch **bugfix** deverá ser criado sempre a partir do branch **release** e consequentemente retornado a ele após a implementação.



Utlizando branch **bugfix**:

1. Alterne para o branch **release**.
2. Atualize branch **release**. (passo 3)

3. Crie e alterne o branch **bugfix**. (passo 13)
4. Implemente as alterações da versão.
5. Crie e envie as alterações para o branch **bugfix** remoto, caso seja necessário realizar mais alterações.
6. Ao finalizar, realize o merge com o branch **release**. (passo 12)

> Exemplo:

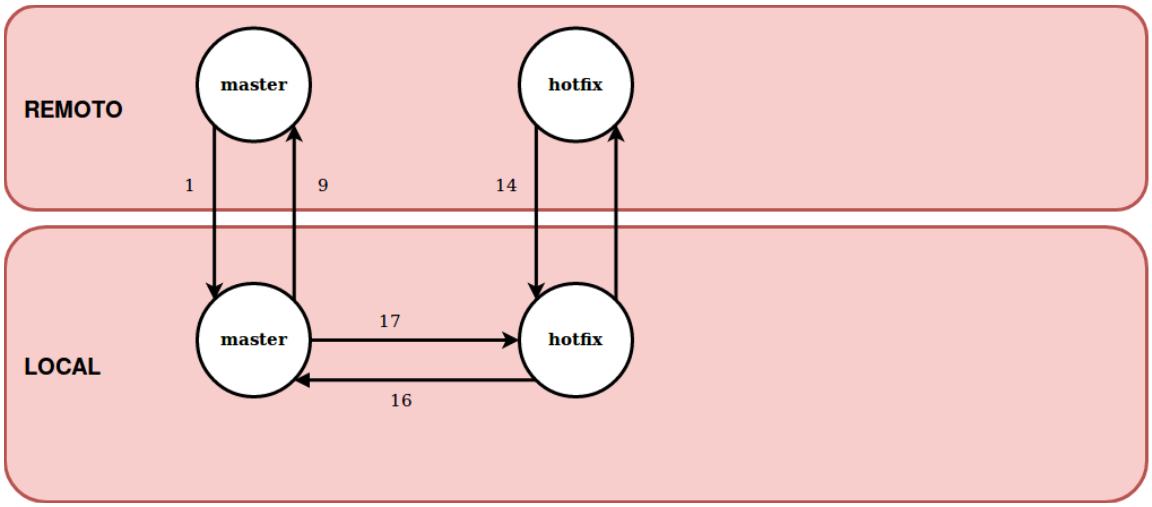
```
git checkout release
git pull (passo 3)
git checkout -b bugfix (passo 13)
git push (passo 11)
git checkout release
git merge bugfix (passo 12)
git push (passo 7)
```

> **FIQUE ATENTO !!!**:
>
> Ao atualizar o branch **bugfix** local poderão surgir conflito. Estes conflitos precisarão ser resolvidos antes de enviar a atualização para o branch **release** remoto.

Versão Emergencial (Hotfix)

Sempre que uma versão em produção apresentar problemas graves e necessitar de uma versão de correção emergencial para garantir a continuidade do serviço e não interromper o processo de homologação vigente, um branch **hotfix** deverá ser criado.

Um branch **hotfix** deverá ser criado sempre a partir de uma tag ou do branch **master** e consequentemente retornado a ele após a implementação.



Utlizando branch **hotfix**:

1. Alterne para o branch **master**.
2. Atualize branch **master**. (passo 1)
3. Crie e alterne o branch **hotfix**. (passo 17)
4. Implemente as alterações da versão.
5. Crie e envie as alterações para o branch **hotfix** remoto, caso seja necessário realizar mais alterações.
6. Ao finalizar, realize o merge com o branch **master**. (passo 16)

> Exemplo:

```
git checkout master
git pull (passo 3)
git checkout -b bugfix (passo 13)
git push (passo 11)
git checkout release
git merge bugfix (passo 12)
git push (passo 7)
```

> **FIQUE ATENTO !!!**:
>
> Ao atualizar o branch **hotfix** local poderão surgir conflito. Estes conflitos precisarão ser resolvidos antes de enviar a atualização para o branch **master** remoto.