

# Workflow Git

- Conceitos Básicos do Git
- Workflow e Comandos Git
- Branches, Commits e Melhores Práticas

- Conceitos básicos de Git
- Workflow e Comandos Git
- Branches, Commit e Melhores Práticas

# Conceitos básicos de Git

## O que é um repositório?

Um repositório (ou "repo") é uma pasta cujo histórico de alterações é rastreado pelo Git. Toda mudança feita nos arquivos pode ser registrada, revertida e auditada. Os repositórios ficam hospedados no GitLab da UFMG e cada pessoa trabalha com uma cópia local na própria máquina.

## O que é um commit?

Um commit é um registro permanente de um conjunto de alterações. Cada commit tem:

- Um identificador único
- Uma mensagem descrevendo o que foi alterado
- A data, hora e autoria da mudança

Commits são a unidade básica de histórico do Git. Toda alteração que você quer preservar precisa virar um commit.

## O que é uma branch?

Uma branch (ramo) é uma linha independente de desenvolvimento. Imagine o histórico do repositório como uma linha do tempo: uma branch é uma bifurcação dessa linha onde você pode fazer experimentos e alterações sem afetar o restante.

A branch principal chama-se `main` e representa o estado atual da infraestrutura em produção. **Nunca se trabalha diretamente nela.** Para qualquer alteração, cria-se uma branch nova, faz-se as mudanças lá, e depois integra-se de volta à `main` por meio de um Merge Request.

## O que é push e pull?

- **pull:** baixar as últimas alterações do GitLab para a sua máquina
- **push:** enviar os commits da sua máquina para o GitLab

## O que é um Merge Request (MR)?

Um Merge Request é uma solicitação para integrar as alterações de uma branch de trabalho na `main`. Ele serve como ponto de revisão: outra pessoa analisa o que vai ser feito antes que qualquer coisa seja aplicada em produção.

No GitLab, o MR também dispara o pipeline de CI automaticamente, gerando um preview (plan/diff) das mudanças na infraestrutura.

## O que é um pipeline de CI?

CI significa Continuous Integration (Integração Contínua). O objetivo é que a cada vez que alguma alteração é feita no código, uma automação é executada para sincronizar a infraestrutura com o novo código adicionado no repositório.

O pipeline é um conjunto de scripts que rodam automaticamente no GitLab quando há um push ou um MR. Nos repositórios DICOM, o pipeline verifica e pré-visualiza o que seria alterado na infraestrutura — mas **nunca aplica nada automaticamente**. A aplicação final sempre exige aprovação manual.

# Workflow e Comandos Git

Todo trabalho segue este fluxo, independentemente do repositório:

1. Atualizar repositório local (main)
2. Criar branch de trabalho
3. Fazer alterações e commits
4. Enviar branch para o GitLab (push)
5. Abrir Merge Request
6. Revisar output do CI
7. Solicitar aprovação
8. Revisor aprova e faz o merge
9. Revisor confere CI pós-merge e aciona o apply

## 1. Atualizar o repositório local antes de começar

Sempre comece atualizando sua cópia local para não trabalhar em cima de uma versão desatualizada:

```
git checkout main  
git pull
```

## 2. Criar uma branch de trabalho

Crie uma branch com um nome descritivo a partir da `main` atualizada:

```
git checkout -b feat/nova-vm-portal
```

O nome da branch deve seguir os padrões definidos na seção abaixo.

## 3. Fazer alterações

Edite os arquivos necessários com seu editor de preferência.

## 4. Ver o que foi alterado

Antes de commitar, confira o que mudou:

```
git status      # lista arquivos modificados
git diff        # mostra as diferenças linha a linha
```

## 5. Registrar as alterações (commit)

Adicione os arquivos que fazem parte da mudança e crie o commit:

```
git add caminho/para/arquivo.tf
git commit -m "feat(clusters): adiciona VM para cluster portal"
```

Evite usar `git add .` sem antes revisar o `git status` — você pode incluir arquivos indesejados por acidente.

## 6. Enviar a branch para o GitLab (push)

```
git push -u origin feat/nova-vm-portal
```

O `-u origin feat/nome-da-branch` só é necessário no primeiro push. Nos seguintes, basta:

```
git push
```

## 7. Abrir o Merge Request

Após o push, o GitLab exibirá um link direto para abrir o MR. Clique nele ou acesse o repositório no GitLab e clique em "Create merge request".

Preencha:

- **Título:** descreva a mudança de forma clara (ex: `feat(clusters): adiciona VM para cluster portal`)
- **Descrição:** explique o que muda e por quê
- **Assignee:** atribua a você mesmo
- **Reviewer:** escolha a pessoa que irá revisar

## 8. Acompanhar o pipeline e revisar

Após abrir o MR, o pipeline de CI roda automaticamente. Aguarde a conclusão e confira o output do job de pré-visualização (`plan` no infra, `diff` no stacks) para verificar se o que será alterado corresponde ao esperado.

## 9. Revisão e aprovação

O revisor deve:

1. Ler a descrição do MR
2. Verificar o output do CI (plan/diff)
3. Fazer perguntas ou solicitar ajustes se necessário
4. Aprovar o MR se estiver tudo correto

**Quem aprovou é quem faz o merge** — não quem abriu o MR. Isso garante uma segunda verificação ativa antes da aplicação.

# Branches, Commit e Melhores Práticas

## Nomenclatura de branches

Use sempre letras minúsculas e hífens como separador. O prefixo descreve a intenção da mudança:

Prefixo	Quando usar
feat/	Nova funcionalidade ou novo recurso (novo cluster, nova VM, novo serviço)
upgrade/	Atualização de versão de ferramenta, imagem ou dependência
fix/	Correção de configuração com defeito ou comportamento incorreto
resize/	Ajuste de recursos de hardware (CPU, memória, disco)
docs/	Alterações exclusivamente em documentação

### Exemplos:

```
feat/cluster-portal  
upgrade/k3s-1.32  
fix/rede-portal-ingress  
resize/admin-node1-memoria  
docs/workflow-infra
```

## Mensagens de commit

Use uma linha curta no imperativo descrevendo **o que** foi feito, com um prefixo de tipo e escopo opcional:

```
<tipo>(<escopo>): <descrição curta>
```

Tipo	Uso
feat	Nova funcionalidade
fix	Correção de bug ou configuração errada
upgrade	Atualização de versão
refactor	Reorganização sem mudança de comportamento
docs	Documentação
chore	Manutenção (ajustes de CI, lock files)

### Exemplos:

feat(clusters): adiciona cluster portal com 3 nós k3s  
fix(admin): corrige IP do gateway do nó admin-node1  
upgrade(cert-manager): atualiza para v1.21.0  
chore(ci): adiciona tag vsphere nos jobs de apply

## Regras gerais

- **Nunca** commite diretamente na `main`
- **Nunca** use `git push --force` na `main`
- **Nunca** aplique mudanças manualmente no ambiente sem antes registrar no repositório. TODAS as alterações de infraestrutura devem ser feitas via GitLab.
- Secrets e senhas **nunca** vão no código — use as variáveis de CI configuradas no grupo `dicom` do GitLab ou no projeto específico que precise de acesso à variável.