Git - Relatório de estudo

INTRODUÇÃO

Este estudo aborda a integração de repositórios e versionamento por meio de Git ao nosso fluxo de trabalho, substituindo o SVN por Git e GitHub. Este documento cobre desde a instalação do Git nas máquinas, configurações iniciais, organização de repositórios e fluxo de commits e pull-requests.

A migração para Git oferece benefícios como o uso mais abrangente de branches nos repositórios de desenvolvimento, gerenciamento de equipes com níveis de acesso e permissão, e integração com ferramentas de CI/CD como o Azure. O SVN que utilizamos atualmente é a última versão gratuita disponível, com suporte limitado e funcionalidades de branching e merges restritas, o que torna a manutenção custosa.

PROGRESSO

Nesta seção irei listar as etapas tomadas ao longo deste estudo para chegar nas conclusões apresentadas neste documento:

- [x] ~~Criação de organização SISPRO que abrigará os repositórios~~
- [x] ~~Convidar alguns colegas para a organização para testar como fica o funcionamento~~
- [ ] Definição de equipes separadas
- [ ] Testes básicos de commits, pull-requests, push/pull, etc
- [ ] Simulação de fluxo de trabalho diário com tarefas e múltiplos devs
- [ ] *Integração de ferramentas de CI/CD

- [ ] Testar funcionalidades específicas
    - [ ] GitHub Flow
    - [ ] GitHub Actions (CI/CD)
    - [ ] GitHub Packages
    - [ ] GitHub Copilot

PLANOS E CUSTOS

GRATUITO

Repositórios

- **Repositórios Públicos e Privados:** O plano gratuito permite a criação de repositórios tanto públicos quanto privados.
- **Colaboradores:** Em repositórios privados, você pode adicionar um número ilimitado de colaboradores.

GitHub Actions (CI/CD)

- **Minutos de CI/CD:** No plano gratuito, você recebe 2000 minutos por mês para usar o GitHub Actions.
- **Armazenamento de Artefatos:** 500 MB de armazenamento de artefatos do GitHub Actions.

GitHub Packages

- **Armazenamento de Pacotes:** 500 MB de armazenamento para pacotes.
- **Transferência de Dados:** 1 GB de transferência de dados fora do GitHub Packages por mês.

Outros Recursos

- **Comunidades e Discussões:** Acesso a ferramentas de colaboração, como Issues e Discussions.
- **Segurança Básica:** Acesso a alertas de segurança e *dependabot* para atualizações automáticas de dependências.

PAGOS

PRO

- **Custo: $4** por usuário/mês.
- **Repositórios:** Repositórios públicos e privados ilimitados.
- **Minutos de CI/CD:** 3000 minutos por mês para GitHub Actions.
- **Armazenamento de Artefatos:** 1 GB de armazenamento de artefatos do GitHub Actions.
- **Armazenamento de Pacotes:** 2 GB de armazenamento para pacotes.
- **Transferência de Dados:** 10 GB de transferência de dados fora do GitHub Packages por mês.

- **Insights e Análises:** Acesso a estatísticas avançadas e insights do repositório.
- **Suporte:** Suporte prioritário.

TEAM

- **Custo: $4** por usuário/mês.
- **Recursos Adicionais:** Inclui todos os recursos do GitHub Pro mais permissões avançadas de repositório, e integrações com ferramentas de terceiros.
- **Ferramentas de Colaboração:** Acesso a recursos de gerenciamento de equipes, como revisões obrigatórias de pull requests e equipes com permissões específicas.

ENTERPRISE

- **Custo: $21** por usuário/mês.
- **Segurança Avançada:** Ferramentas de segurança avançadas, como SAML SSO, LDAP, e mais.
- **Compliance e Monitoramento:** Funcionalidades de compliance e monitoramento para grandes empresas.
- **Suporte e SLA:** Suporte prioritário com acordos de nível de serviço (SLA).
- **Armazenamento e Minutos de CI/CD:** Limites mais altos ou ilimitados, dependendo do contrato.

GITHUB ACTIONS (CI/CD)

- **Minutos de CI/CD:** Aumento significativo de minutos disponíveis para execução de ações.
- **Armazenamento de Artefatos:** Armazenamento maior para artefatos gerados durante a CI/CD.

GITHUB PACKAGES

- **Armazenamento de Pacotes:** Armazenamento maior para pacotes.
- **Transferência de Dados:** Mais transferência de dados fora do GitHub Packages por mês.

SUPORTE E SLA

- **Suporte Prioritário:** Planos pagos geralmente incluem suporte prioritário.
- **SLA:** Acordos de nível de serviço garantidos para tempos de resposta e resolução

GITHUB COPILOT

O Copilot é uma ferramenta de assistência de codificação que utiliza inteligência artificial para ajudar desenvolvedores a escrever código mais rapidamente e com mais eficiência. Desenvolvido em parceria com a OpenAI, o Copilot sugere linhas completas de código ou blocos de código enquanto você digita, com base no contexto do código atual. É ótimo também para ajudar os diversos desenvolvedores a manter o padrão de código pelos projetos, independente da linguagem.

Suas principais funcionalidades incluem mas não se limitam a:

- **Sugestões de Código:** Oferece sugestões em tempo real enquanto você digita, propondo linhas ou blocos de código completos.
- **Autocompletar:** Completa funções ou métodos, baseando-se em padrões de codificação comuns e no contexto do código que está sendo escrito.
- **Compatibilidade:** Suporta diversas linguagens de programação, incluindo Python, JavaScript, TypeScript, Ruby, Go, e muitas outras.
- **Ferramentas de Desenvolvimento:** Integra-se com editores de código populares como Visual Studio Code, Neovim, e JetBrains.

Alguns de seus benefícios são:

- **Produtividade:** Acelera o processo de escrita de código, ajudando desenvolvedores a se concentrarem mais na lógica e na arquitetura do que na sintaxe.
- **Qualidade do Código:** Sugestões de código podem ajudar a seguir padrões de codificação e a evitar erros comuns.
- **Apoio à Aprendizagem:** Desenvolvedores novos podem aprender melhores práticas e técnicas de codificação ao ver sugestões em tempo real.

CUSTOS

- **Individual:** Atualmente **$10** por usuário/mês ou **$100** por usuário/ano.
- **Business:** Atualmente **$19**.
- **Enterprise *(*plano recentemente adicionado)*:** Atualmente **$39**.

INSTALAÇÃO

Instalação do git bash

- Faça o download por meio do site oficial: https://git-scm.com/downloads
- Executar o instalador como Administrador
- Selecionar as opções padrão

Configuração inicial

- Configurar nome de usuário e email com os seguintes comandos

```
git config –global user.name “Seu Nome”
git config –global user.email “seu.email@empresa.com”
```

- Configurar o proxy

```
git config --global http.proxy
http://user:senha@siscanproxy2.sispro.com.br:3128

git config --global https.proxy
http://user:senha@siscanproxy2.sispro.com.br:3128

git config --global http.http://sispro.com.br
http://user:senha@siscanproxy2.sispro.com.br:3128

git config --global https.https://sispro.com.br
http://user:senha@siscanproxy2.sispro.com.br:3128
```

***Obs:** os dois primeiros comandos definem o proxy como um todo e os dois últimos definem o proxy para o domínio “sispro.com.br” que é muito utilizado por nós.

- Verificar as configurações

```
git config –list
```

REPOSITÓRIOS

Criar um novo Repositório Local

- Iniciar um repositório em um diretório existente

```
cd /caminho/do/projeto
git init
```

- Clonar um repositório remoto

```
git clone https://url.do.repositorio.git
```

Configurar o Repositório Remoto

- Adicionar um repositório remoto

```
git remote add origin https://url.do.repositorio.git
```

- Verificar os repositórios remotos

```
git remote -v
```

ACESSOS

SSH

O SSH (Secure Shell) é um protocolo de rede que fornece uma maneira segura de acessar remotamente computadores e transferir arquivos entre sistemas. No contexto de gerenciamento de repositórios Git, o SSH é frequentemente usado para autenticação segura e comunicação criptografada com servidores Git, como GitHub, GitLab, Bitbucket, entre outros.

Como o SSH Funciona

- **Autenticação Segura:** O SSH usa um par de chaves criptográficas para autenticação. Isso inclui uma chave privada (mantida em segredo no seu computador) e uma chave pública (que você carrega no servidor Git).
- **Comunicação Criptografada:** Todo o tráfego entre o seu computador e o servidor Git é criptografado, protegendo-o contra interceptações e ataques *man-in-the-middle*.

Como Utilizar o SSH em uma Organização Git com Muitos Colaboradores
Passos para Configurar SSH no Git:

1. **Gerar uma Chave SSH:** Cada colaborador deve gerar um par de chaves SSH em seu computador:

```
ssh-keygen -t rsa -b 4096 -C "seu.email@empresa.com"
```

Durante o processo, os usuários podem definir um nome de arquivo para a chave (por padrão é **id_rsa**) e uma senha (opcional, mas **recomendada para maior segurança**).

2. **Adicionar a Chave SSH ao Agente SSH:** Após gerar a chave, os colaboradores devem adicionar a chave privada ao agente SSH:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

3. **Adicionar a Chave Pública ao Servidor Git:** Cada colaborador deve copiar o conteúdo da chave pública (~/.ssh/id_rsa.pub) e adicioná-la à sua conta no servidor Git (GitHub, GitLab, Bitbucket, etc.):
    - **GitHub:** Navegar até Settings > SSH and GPG keys > New SSH key

- **GitLab:** Navegar até Profile Settings > SSH Keys > Add SSH Key
- **Bitbucket:** Navegar até Personal Settings > SSH keys > Add key

Gerenciamento de Colaboradores em uma Organização

- **Criar e Gerenciar Equipes/Grupos**

No servidor Git, você pode organizar colaboradores em equipes ou grupos com diferentes níveis de permissão (leitura, escrita, admin):

- **GitHub:** Organize os colaboradores em equipes dentro de uma organização.
- **GitLab:** Use grupos e subgrupos para organizar os projetos e colaboradores.
- **Bitbucket:** Utilize equipes e permissões de grupo.

- **Definir Políticas de Acesso**

Assegure-se de que apenas colaboradores autorizados tenham acesso aos repositórios, de acordo com suas funções e responsabilidades. Utilize políticas de acesso rigorosas para controlar quem pode ver, modificar ou administrar cada repositório.

- **Configurar Deploys Automáticos com SSH**

Para ambientes de deploy contínuo (CI/CD), você pode configurar chaves de deploy específicas para que os servidores de integração contínua (como Jenkins, GitLab CI, GitHub Actions) possam acessar os repositórios Git de forma segura.

- **Rotação e Revogação de Chaves**

Regularmente rotacione e revogue chaves SSH para manter a segurança. Quando um colaborador deixar a organização, revogue imediatamente suas chaves SSH para garantir que ele não tenha mais acesso.

Vantagens do Uso de SSH

- **Segurança:** O SSH oferece uma camada adicional de segurança, pois as chaves privadas nunca são transmitidas pela rede.
- **Praticidade:** Uma vez configurado, o SSH permite autenticação sem senha, simplificando o processo de push e pull.
- **Flexibilidade:** O SSH é amplamente suportado por serviços de hospedagem de repositórios Git e ferramentas de CI/CD, proporcionando uma integração suave no fluxo de trabalho.

Implementar o SSH para gerenciamento de repositórios Git em uma organização com muitos colaboradores melhora significativamente a segurança e a eficiência. Com a autenticação baseada em chaves SSH, a comunicação entre os desenvolvedores e o servidor Git é segura e confiável. Além disso, a organização

adequada de equipes e políticas de acesso garante que cada colaborador tenha os direitos necessários, sem comprometer a segurança do projeto.

Configurar SSH Keys

- Gerar uma nova chave SSH
- Adicionar a chave SSH ao agente SSH
- Adicionar a chave SSH à conta no GitHub
    - Copiar o conteúdo de ~/.ssh/id_rsa.pub
    - Adicionar nas configurações de SSH Keys do serviço utilizado

Gerenciamento de Acessos no Repositório

- Definir políticas de acesso
    - Acesso de admin para os gestores das equipes
    - Acesso de escrita para desenvolvedores daquela equipe
    - Acesso apenas de leitura para desenvolvedores de outras equipes
- Utilizar grupos e equipes para facilitar o gerenciamento de permissões

Azure

É possível integrar o Azure com o Git. O Azure oferece várias opções para essa integração, permitindo que você use Git para controle de versão e para automatizar processos de CI/CD. Aqui estão algumas maneiras de integrar o Azure com o Git:

AZURE REPOS

Azure Repos é um serviço dentro do Azure DevOps que oferece repositórios Git privados. Com ele, você pode hospedar seus repositórios Git, gerenciar branches, fazer pull requests e colaborar com outros desenvolvedores.

Passos para Usar Azure Repos:

1. Criar um Projeto no Azure DevOps:
   - Acesse Azure DevOps.
   - Crie um novo projeto ou use um existente.

2. Criar um Repositório Git:
   - Dentro do projeto, vá para **Repos > Files**.
   - Clique em "**New Repository**" e configure o repositório.

3. Clonar o Repositório para Seu Ambiente Local:
   - Copie a URL do repositório.
   - No terminal, use o comando: git clone

```
git clone <URL_DO_REPOSITÓRIO>
```

4. Adicionar, Commitar e Enviar Alterações:
   - Adicione arquivos ao repositório:

```
git add .
git commit -m "Mensagem do commit"
git push origin main
```

AZURE PIPELINES

Azure Pipelines é uma solução de CI/CD que pode ser integrada com repositórios Git. Ele permite automatizar a construção, teste e implantação do seu código.

Passos para Configurar Azure Pipelines:

1. Configurar um Pipeline:
    - Dentro do Azure DevOps, vá para **Pipelines > Pipelines**.
    - Clique em "**New Pipeline**".
    - Escolha o repositório Git onde seu código está hospedado (pode ser Azure Repos, GitHub, Bitbucket, etc.).
    - Siga as instruções para configurar o pipeline (pode ser usando YAML ou a interface visual).

2. **Definir o Arquivo YAML (Opcional):**
    - Crie um arquivo azure-pipelines.yml no repositório.

Exemplo de arquivo YAML:

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- script: echo Hello, world!
  displayName: 'Run a one-line script'

- script: |
    echo Add other commands here
    echo Upload artifacts, run tests, etc.
  displayName: 'Run a multi-line script'
```

3. **Executar e Monitorar Pipelines:**
    - Execute o pipeline e monitore os resultados dentro do Azure DevOps.

AZURE DEVOPS COM GITHUB

Se você está usando GitHub como seu repositório Git, pode integrar o Azure DevOps para CI/CD.

Passos para Integrar Azure DevOps com GitHub:

1. Conectar o Repositório GitHub ao Azure DevOps:
    - Dentro do Azure DevOps, vá para Pipelines > Pipelines.

- Clique em "New Pipeline".
- Selecione GitHub como a fonte do repositório.
- Autorize o acesso ao GitHub e selecione o repositório.

2. Configurar o Pipeline:
   - Configure o pipeline para o repositório GitHub usando o assistente ou arquivo YAML.

3. Automatizar Processos:
   - Configure triggers para build, test e deploy automáticos baseados em eventos no GitHub (como push para a branch principal ou abertura de pull requests).

USANDO GITHUB ACTIONS COM AZURE

Se você prefere usar GitHub Actions para CI/CD, pode integrar com Azure para implantação e gerenciamento de recursos.

Passos para Usar GitHub Actions com Azure:

1. Criar um Workflow no GitHub:
   - No repositório GitHub, crie um arquivo **.github/workflows/main.yml**.

2. Definir o Workflow:

Exemplo de workflow YAML para implantar uma aplicação no Azure:

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Node.js
      uses: actions/setup-node@v2
```

```
      with:
        node-version: '14'
    - run: npm install
    - run: npm run build
    - name: 'Deploy to Azure Web App'
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'meu-app'
        slot-name: 'production'
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
```

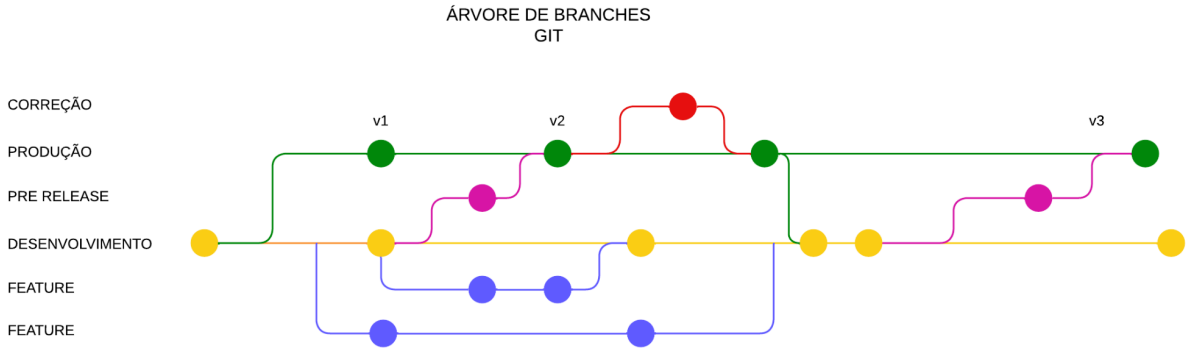
3. Configurar Segredos no GitHub:
    - Adicione segredos no repositório GitHub (por exemplo, *AZURE_WEBAPP_PUBLISH_PROFILE*).

Integrar Azure com Git proporciona um ambiente robusto para desenvolvimento colaborativo e automação de processos. Dependendo de suas necessidades específicas, você pode usar Azure Repos para controle de versão, Azure Pipelines para CI/CD, ou GitHub Actions para integração mais personalizada com Azure.

WORKFLOW

Estrutura de Branches

- **Branch Principal (main ou master):**
  - Contém o código pronto para produção.
- **Branches de Desenvolvimento (develop):**
  - Contém o código em desenvolvimento e testes.
- **Branches de Funcionalidades (feature/nomedafuncionalidade):**
  - Para desenvolvimento de novas funcionalidades.
- **Branches de Correção (hotfix/nomedahotfix):**
  - Para correções de bugs em produção.
- **Branches de Release (release/versao):**
  - Para preparar novas versões.



Fluxo de Commits e Pull Request

Commits:

- Realizar commits pequenos e frequentes.
- Mensagens de commit claras e descritivas.

Pull Requests:

- Abrir um pull request para cada nova funcionalidade ou correção.
- Revisão de código obrigatória por **pelo menos um desenvolvedor**.
- Realizar merges após aprovação.

Integração Contínua e Deploy Contínuo (CI/CD)

- **Configurar pipelines de CI/CD** para automação de testes e deploys.
- **Ferramentas sugeridas**:
    - GitHub Actions, GitLab CI, Jenkins.

BOAS PRÁTICAS

- **Documentação:** Manter a documentação do projeto atualizada.
- **Testes:** Escrever testes automatizados para novas funcionalidades e correções.
- **Code Reviews:** Práticas de revisão de código para melhorar a qualidade.

REFERÊNCIAS

- **Preços do GitHub:** Pricing · Plans for every developer (github.com)
- **GitHub Copilot:** GitHub Copilot · Your AI pair programmer
- **ChatGPT:** ChatGPT | OpenAI
- **GitHub Flow:** Fluxo do GitHub - GitHub Docs
- **Organização SISPROV6:** SISPRO (github.com)
- **GitHooks:** Git Hooks - A Guide for Programmers