INTRO-CS-6 - Source Control Using Git_vF_BPT

1. Controle de versão

O controle de versão, também conhecido como **controle de origem**, é a prática de rastrear e gerenciar alterações no código do software. Os sistemas de controle de versão são ferramentas de software que ajudam as equipes de software a gerenciar as alterações no código-fonte ao longo do tempo. À medida que os ambientes de desenvolvimento se aceleram, os sistemas de controle de versão ajudam as equipes de software a trabalhar de forma mais rápida e inteligente.

O software de controle de versão acompanha cada modificação do código em um tipo especial de banco de dados. Se for cometido um erro, os desenvolvedores podem voltar no tempo e comparar versões anteriores do código para ajudar a corrigir o erro e minimizar a interrupção para todos os membros da equipe.

Os desenvolvedores de software que trabalham em equipes estão continuamente escrevendo novo código-fonte e alterando o código-fonte existente. O código de um projeto, aplicativo ou componente de software normalmente é organizado em uma estrutura de pastas ou "árvore de arquivos". Um desenvolvedor da equipe pode estar trabalhando em um novo recurso enquanto outro desenvolvedor corrige um bug não relacionado alterando o código, cada desenvolvedor pode fazer suas alterações em várias partes da árvore de arquivos.

O software de controle de versão é uma parte essencial do dia a dia das práticas profissionais da equipe de software moderna. Desenvolvedores de software individuais que estão acostumados a trabalhar com um sistema de controle de versão capaz em suas equipes normalmente reconhecem o incrível valor que o controle de versão também lhes oferece, mesmo em pequenos projetos individuais. Uma vez acostumados aos poderosos benefícios dos sistemas de controle de versão, muitos desenvolvedores não considerariam trabalhar sem ele, mesmo para projetos que não sejam de software.

2. Git

De longe, o sistema de controle de versão moderno mais utilizado no mundo hoje é o Git. O Git é um projeto de código aberto maduro e mantido ativamente, originalmente desenvolvido em 2005 por Linus Torvalds, o famoso criador do kernel do sistema operacional Linux. Um número impressionante de projetos de software depende do Git para controle de versão, incluindo projetos comerciais e de código aberto. Os desenvolvedores que trabalharam com o Git estão bem representados no conjunto de talentos de desenvolvimento de software disponíveis e funcionam bem em uma ampla variedade de sistemas operacionais e IDEs (Ambientes de Desenvolvimento Integrado).

Para melhor compreensão em sentido de códigos segue uma breve definição dos comandos executados no terminal:

Configurar ferramentas

Configurar informações do usuário para todos os repositórios locais

- Define o nome que você deseja anexar às suas transações de confirmação

```
$ git config --global user.name "[name]"
```

- Define o e-mail que você deseja anexar às suas transações de confirmação

```
$ git config --global user.email "[email address]"
```

- Permite colorização útil da saída da linha de comando

```
$ git config --global color.ui auto
```

- Define credenciais de usuario do GitHub

```
$ git config --global credential.username "[user credentials]"
```

Branches

As ramificações são uma parte importante do trabalho com o **Git**. Quaisquer **commits** que você fizer serão feitos no **branch** para o qual você está atualmente **"check-out"**. Use **git status** para ver qual **branch** é esse.

- Criar uma nova ramificação

```
$ git branch [branch-name]
```

- Muda para a ramificação especificada e atualiza o diretório de trabalho

```
$ git switch -c [branch-name]
```

- Combina o histórico da ramificação especificada na ramificação atual. Isso geralmente é feito em solicitações pull, mas é uma operação importante do Git.

```
$ git merge [branch]
```

- Exclui o branch especificado

```
$ git branch -d [branch-name]
```

Criar repositórios

Um novo repositório pode ser criado localmente ou um repositório existente pode ser clonado. Quando um repositório foi inicializado localmente, você precisa envia-ló para o GitHub posteriormente.

- O comando git init transforma um diretório existente em um novo repositório Git dentro da pasta que você está executando este comando.

```
$ git init
```

- Depois de usar o comando **git init** , vincule o repositório local a um repositório vazio do GitHub usando o seguinte comando que especifica o repositório remoto para seu repositório local. A url aponta para um repositório no GitHub:

```
$ git remote add origin [url]
```

- Clone (baixe) um repositório que já existe no GitHub, incluindo todos os arquivos, branches e commits

```
$ git clone [url]
```

Sincronizar alterações

Sincronize seu repositório local com o repositório remoto no GitHub.com

- Baixa todo o histórico das ramificações de rastreamento remoto

```
$ git fetch
```

- Combina ramificações de rastreamento remoto na ramificação local atual

```
$ git merge
```

- Carrega todos os commits de branch local para o GitHub

```
$ git push
```

- Atualiza seu branch de trabalho local atual com todos os novos commits do branch remoto correspondente no GitHub. **git pull** é uma combinação de **git fetche** e **git merge**

```
$ git pull
```

Fazer mudanças

Navegue e inspecione a evolução dos arquivos do projeto.

- Lista o histórico de versões do branch atual

```
$ git log
```

- Lista o histórico de versões de um arquivo, além de renomeações (funciona apenas para um único arquivo)

```
$ git log --follow [file]
```

- Mostra as diferenças de conteúdo entre duas ramificações

```
$ git diff [first-branch]...[second-branch]
```

- Saídas de metadados e alterações de conteúdo do **commit** especificado

```
$ git show [commit]
```

- Preparação para controle de versão

```
$ git add [file]
```

- Grava alterações de arquivos permanentemente no histórico de versões

```
$ git commit -m "[descriptive message]"
```

Refazer confirmações

Apague os erros e o histórico de substituição

- Desfaz todos os commits após `[commit]`, preservando as alterações localmente

```
$ git reset [commit]
```

- Descarta todo o histórico e muda de volta para o **commit** especificado

```
$ git reset --hard [commit]
```

*CUIDADO! Mudar a história pode ter efeitos colaterais desagradáveis. Se você precisar alterar os commits que existem no GitHub (o remoto), prossiga com cuidado. Se precisar de ajuda, entre em contato em github.community ou entre em contato com o suporte.

.gitignore

Às vezes, pode ser uma boa ideia excluir arquivos do rastreamento com o Git. Isso geralmente é feito em um arquivo especial chamado `.gitignore`. Você pode encontrar modelos úteis para `.gitignore` arquivos em github.com/github/gitignore .

Glossário

- **git** : um sistema de controle de versão distribuído de código aberto
- **GitHub** : uma plataforma para hospedar e colaborar em repositórios Git
- **commit** : um objeto Git, um instantâneo de todo o seu repositório compactado em um SHA
- **branch** : um ponteiro móvel leve para um commit
- **clone** : uma versão local de um repositório, incluindo todos os commits e branches
- **remote** : um repositório comum no GitHub que todos os membros da equipe usam para trocar suas alterações
- **fork** : uma cópia de um repositório no GitHub pertencente a um usuário diferente
- **pull request** : um lugar para comparar e discutir as diferenças introduzidas em um branch com revisões, comentários, testes integrados e muito mais
- **HEAD** : representando seu diretório de trabalho atual, o ponteiro HEAD pode ser movido para diferentes branches, tags ou commits ao usar `git switch`

Para saber mais sobre o Git, acesse ao Link.

3. GitHub

GitHub é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e/ou Open Source de qualquer lugar do mundo. GitHub é amplamente utilizado por programadores para divulgação de seus trabalhos ou para que outros programadores contribuam com o projeto, além de promover fácil comunicação através de recursos que relatam problemas ou mesclam repositórios remotos

Fluxo de entrega quando não existe repositório criado:

1. Criar repositório no GitHub;
2. Clonar repositório em uma pasta na maquina: `$ git clone [url]`;
3. Fazer alterações necessárias;
4. Preparar alterações para controle de versão: `$ git add .` ou `$ git add [file]`;
5. Gravar alterações de arquivos permanentemente: `$ git commit -m "[descriptive message]"`;
6. Carrega todos os commits de branch local para o GitHub: `$ git push [nome-repo] [branch]`.

Fluxo de entrega quando já existe um repositório criado:

1. Clonar repositório em uma pasta na maquina: `$ git clone [url]`;
2. Fazer alterações necessárias;
3. Preparar alterações para controle de versão: `$ git add .` ou `$ git add [file]`;
4. Gravar alterações de arquivos permanentemente: `$ git commit -m "[descriptive message]"`;
5. Carrega todos os commits de branch local para o GitHub: `$ git push [nome-repo] [branch]`.

Fluxo de entrega quando o repositório já esta clonado na maquina:

1. Fazer alterações necessarias;
2. Preparar alterações para controle de versão: `$ git add .` ou `$ git add [file]`;
3. Gravar alterações de arquivos permanentemente: `$ git commit -m "[descriptive message]"`;
4. Carrega todos os commits de branch local para o GitHub: `$ git push [nome-repo] [branch]`.

Fluxo de entrega inicializando da maquina:

1. Garantir que o diretório na maquina tenha o repositório criado, caso não tenha executar comando: `$ git init`;
2. Git irá criar uma branch nomeada **master** agora basta começar a escrever códigos;
3. Preparar alterações para controle de versão: `$ git add .` ou `$ git add [file]`;
4. Gravar alterações de arquivos permanentemente: `$ git commit -m "[descriptive message]"`;
5. Incluir o **remote** do GitHub que deseja enviar as alterações: `$ git remote add [nome-repo] [url]`;
6. Carrega todos os commits de branch local para o GitHub: `$ git push [nome-repo] [branch]`.

Recomendações:

1. Sempre que estiver trabalhando em equipe, assim que criar o **commit** e antes de enviar com o comando **push**, é importante validar se é necessário puxar algo do repositório que estão trabalhando com o comando **pull**.
2. Conflitos, não são erros, quando o git precisa tomar decisões ele informa conflito e passa para o desenvolvedor escolher o qual decisão tomar.
3. O fluxo de trabalho pode ser modificado a qualquer instante caso seja necessário, não se acostume.
4. Fazer pequenas alterações de código, e criar o **commit**, desta maneira fica mais fácil fazer manutenção de uma pequena parte de código.