

Módulo Controle de Versão

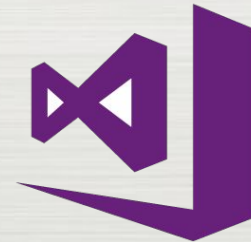


Ao criarmos códigos e scripts, seja para uma aplicação, banco de dados, automação de testes, é imprescindível termos um controle das informações para que estejam guardadas com segurança e que permitam que uma equipe trabalhe em paralelo. Para isso utilizamos o recurso de controle de versão.

Benefícios:

- Rapidez - controle seu trabalho na sua máquina na medida que novos recursos forem sendo desenvolvidos com histórico (commit).
- Autonomia - não é necessário internet para controlar localmente.
- Segurança - controle dos códigos com fácil backup.
- Ramos individuais - vários desenvolvedores podem trabalhar no mesmo projeto.
- Facilidade de fusão - unificar códigos após finalizar o desenvolvimento.

Exemplos de versionadores:



Team Foundation Server



Uso do controle de versão Git

O padrão atual utilizado na *Base2* é o controle de versão através do Git. Por isso recomendamos que seja estudado e aprofundado o entendimento nos recursos que ele nos dispõe.

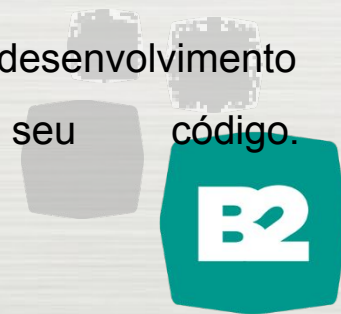
O Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência

Software e recursos recomendados para versionamento:

- [Git bash](#) - terminal linux para controle de versão Git através da linha de comando.
- [GitKraken](#) - ferramenta visual para controle de versão de maneira simplificada.

Ferramentas de desenvolvimento também disponibilizam recursos para controle de versões:

- [Controle de Versão no VisualStudio](#) - recurso nativo na ferramenta de desenvolvimento para CSharp permitindo gerenciar e versionar seu código.
- [Controle de Versão no IntelliJIDEA](#) - recurso nativo na ferramenta de desenvolvimento para Java permitindo gerenciar e versionar seu código.



Uso do controle de versão Git

Aprendendo comandos mais utilizados no GIT:

- Comandos básicos de GIT:

git config

Um dos comandos git mais usados é o **git config** que pode ser usado para definir valores de configuração específicos do usuário como e-mail, algoritmo preferido para diff, nome de usuário e formato de arquivo etc. Por exemplo, o seguinte comando pode ser usado para definir o email:

```
git config --global user.email sam@google.com
```

git init

Este comando é usado para criar um novo repositório GIT. Uso:

```
git init
```

git help

Este comando git lista todos os comandos que podem ser usados no Git

```
git --help
```



Uso do controle de versão Git

git add

O comando **git add** pode ser usado para adicionar arquivos ao índice. Por exemplo, o seguinte comando irá adicionar um arquivo chamado temp.txt presente no diretório local para o índice:

```
git add temp.txt
```

Outro exemplo quando a intenção for adicionar TODOS arquivos presentes no diretório local para o índice

```
git add . ou git add -all
```

git clone

O comando **git clone** é usado para fins de verificação de repositório. Se o repositório estiver em um servidor remoto, use:

```
git clone alex@93.188.160.58:/path/to/repository
```

Por outro lado, se uma cópia de trabalho de um repositório local for criada, use:

```
git clone /path/to/repository
```



Uso do controle de versão Git

git commit

O comando **git commit** é usado para confirmar as alterações na cabeça. Tenha em atenção que quaisquer alterações efetuadas não irão para o repositório remoto. Uso:

```
git commit -m "coloque sua mensagem aqui"
```

git status

O comando **git status** exibe a lista de arquivos alterados juntamente com os arquivos que ainda não foram adicionados ou confirmados. Uso:

```
git status
```

git push é outro dos comandos git básicos mais usados. Um simples envio envia as alterações feitas para o ramo mestre do repositório remoto associado ao diretório de trabalho. Por exemplo:

```
git push origin master
```



Uso do controle de versão Git

git checkout

O comando **git checkout** pode ser usado para criar ramos ou alternar entre eles. Por exemplo, o seguinte cria um novo ramo e muda para ele:

```
command git checkout -b <branch-name>
```

Para simplesmente mudar de um ramo para outro, use:

```
git checkout <branch-name>
```

git remote

O comando **git remote** permite que um usuário se conecte a um repositório remoto. O comando a seguir lista os repositórios remotos atualmente configurados:

```
git remote -v
```

Esse comando permite que o usuário se conecte a um servidor remoto:

```
git remote add origin <93.188.160.58>
```



Uso do controle de versão Git

git branch

O comando **git branch** pode ser usado para listar, criar ou excluir ramos. Para listar todos os ramos presentes no repositório, use:

```
git branch
```

Para incluir um ramo:

```
git branch branch-name
```

Para excluir um ramo:

```
git branch -d <branch-name>
```

git pull

Para mesclar todas as alterações presentes no repositório remoto para o diretório de trabalho local, o comando pull é usado. Uso:

```
git pull
```



Uso do controle de versão Git

git merge

O comando **git merge** é usado para mesclar uma ramificação no ramo ativo. Uso:

```
git merge <branch-name>
```

git diff

O comando **git diff** é usado para listar os conflitos. Para visualizar conflitos com o arquivo base, use

```
git diff --base <file-name>
```

O seguinte comando é usado para exibir os conflitos entre ramos about-to-be-merged antes de mesclá-los:

```
git diff <source-branch> <target-branch>
```

Para simplesmente listar todos os conflitos atuais, use:

```
git diff
```



Repositórios remotos

O uso de um repositório remoto para controle de versão é essencial para uma maior segurança das informações estando hospedados na internet. Você pode ter vários deles, geralmente cada um é somente leitura ou leitura/escrita pra você. Colaborar com outros envolve gerenciar esses repositórios remotos e fazer o push e pull de dados neles quando você precisa compartilhar trabalho.

Exemplos de repositórios remotos:



GitHub




GitLab



Crie e gerencie seus projetos de automação de testes e desafios de testes automatizados da Base2 com controle de versão e utilize um repositório remoto sempre para ter segurança e não perder nenhum código por problemas técnicos, forças da natureza ou erros humanos.

E lembre-se sempre:

In case of fire 

 1. `git commit`

 2. `git push`

 3. `leave building`

