Minicurso Git

Autor: Fabrício G. M. de Carvalho

Última modificação: 2020-03-02

Versão 1.0.0

O que é o Git?

Trata-se de um sistema de gerenciamento de configuração de software (SCM - *Software Configuration Management*).

Obs: Versão diz respeito a um determinado estado de um artefato de software em um determinado instante de tempo. Configuração tipicamente se refere às várias versões de diferentes artefatos que fazem parte de um projeto de software. Fazer gerenciamento de configuração tipicamente é mais complexo do que fazer gerenciamento de versão.

Como o Git Funciona?

Trata-se de um sistema de controle de configuração que é distribuído.

Armazena pequenas "fotografias" (*snapshots*) das diferentes versões de um dado artefato ao longo do tempo. Quando um artefato não muda, o sistema simplesmente guarda uma referência à versão anterior.

Em quais estados um artefato pode estar?

Modificado
: Você criou ou alterou um artefato. O Git informa isso mas não se programa para enviá-lo à sua base de controle.

Preparado (Staged)
: O artefato modificado (*snapshot*) está pronto para ser enviado à base de controle do Git. Utiliza o comando **`git add`**.

Transferido/Enviado (Commited)
: O *snapshot* do artefato foi enviado à base de controle do Git. Utiliza o comando **`git commit`**.

Instalação, inicialização e configuração inicial

Para instalar o git localmente, fazer o download e seguir o procedimento simpes mostrado em https://git-scm.com/downloads.

Uma vez instalado, verificar se a ferramenta está disponível digitando o comando

```
git --version
```

Se tudo estiver ok, o terminal deverá exibir a seguinte mensagem (ou similar):

Para inicializar um repositório, criar uma pastar e digitar o comando

```
git init
```

Uma vez inicializado um repositório, deve-se configurar qual o nome e qual o email que vai ser registrado a cada vez que um arquivo for enviado para a base de dados de controle. Isso é feito através dos comandos

```
git config --local user.name "Seu nome"
```

```
git config --local user.email "user@email.com"
```

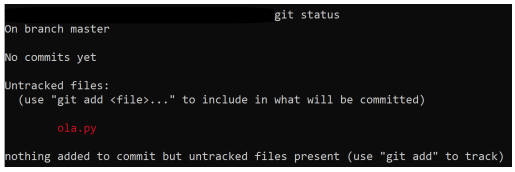
As configurações tais como mostradas são aplicadas ao projeto no local da pasta em uso. Caso se deseje realizar uma única configuração que será aplicada a todos os projetos, substituir `--local` por `--global`.

Criando um arquivo, deixando o git rastrear, adicionando ao repositório local

Na pasta onde o repositório foi criado, criar um arquivo chamado `ola.py`, por exemplo, o seguinte código-fonte em Python:

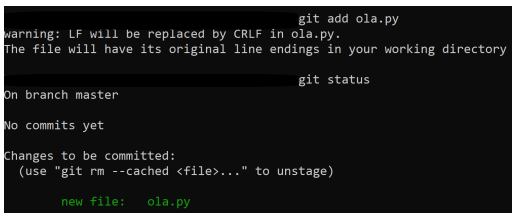
```
print("Olá mundo!")
```

Verificar que o arquivo não está sendo controlado pelo git através do comando `git status`



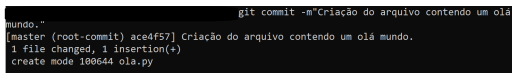
Feito isso, digitar o comando **`git add ola.py`**

Notar que agora o sistema de controle de versão possui um *snapshot* do arquivo na área de preparação. Além disso, a ferramenta informa como remover o arquivo da área de preparação, através do comando que, para esse exemplo, seria **`git remove --cached ola.py`**



Para enviar ao repositório local, basta utilizar o comando

`git commit -m"Criação do arquivo contendo olá mundo"`



Observação: Caso se deseje excluir um arquivo já rastreado pelo git, presente no repositório, utilizar o comando **`git rm nome_arquivo`** nesse caso, o arquivo será removido da área de trabalho local, do índice e não mais será rastreado pela ferramenta.

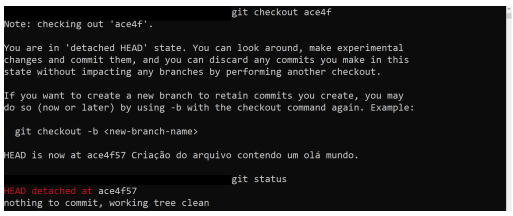
Enviando alterações, examinando o log e voltando para versões anteriores.

Para fazer com que a feramenta rastreie alterações e continue a "guardar" *snapshots* do arquivo rastreado, basta seguir o fluxo

1. **`git status`**
2. **`git add nome_arquivo`**
3. **`git commit -m"Mensagem informativa do commit"`**

Para examinar o histório de modificações, digitar **`git log`**.

Caso queira retornar para uma versão já submetida ao repositório, utilizar **`git checkout 'numero_commit'`**. Para o exemplo mostrado, caso se queira retornar à primeira versão submetida, faz-se:



Notar, que nesse caso, as alterações feitas na versão antiga não afetarão as vesões já submetidas ao repositório. Caso se queira fazer isso, deve-se criar um novo ramo (*branch*).

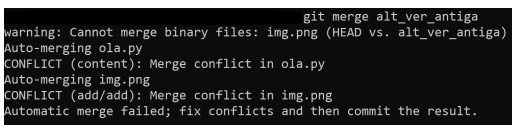
Criando um ramo e fazendo uma fusão (merge)

Um ramo é criado a partir de um ponto já controlado pelo repositório, esse ponto pode ser uma versão atual (HEAD) do ramo atual padrão (MASTER), ou a partir de qualquer ponto no passado já controlado pelo Git.

Para criar um ramo, utiiza-se o comando **`git branch nome_ramo ponto_de_referencia`**. Ponto de referência está relacionado ao *hash* do commit efetuado. A sequência seguinte, exemplifica a criação de um branch a partir do primeiro *commit*no repositório. Nesse caso, deve-se perceber que o número do *hash* corresponde ao gerado a partir de um *commit* de uma alteração a partir de um ponto antigo no repositório.

```
git  branch alt_ver_antiga 6edfd17
```

Após se efetuar alguma alteração no ramo, procede-se com o mesmo ciclo **`add --> commit`**. Caso se queira fundir essas alterações com as efetuadas em outro *branch*, troca-se para o ramo de destino (comando **`git checkout nome_ramo_destino`** e efetua-se o comando **`git merge nome_ramo_origem`**.



Outras dicas importantes

- Existem interfaces gráficas que dispensam a utilização da linha de comando para interagir com o Git (e.g.: Git GUI,

TortoiseGit, etc.). Entretanto, os conceitos e o fluxo de trabalho, que são os mais importantes, permanecem.
- Alterações locais podem ser enviadas para repositórios remotos, tais como os oferecidos pelo GitHub, Bitbucket, etc.
- Há ferramentas distintas, tais como Dropbox, que também possibilitam a gestão de configuração de software (incluindo documentos). Nesse caso, o fluxo de trabalho e as limitações também diferem do Git.
- Todas as ferramentas de gestão de configuração baseiam-se em estruturas de pastas e nomes de arquivo para o controle das versões. Deve-se adotar uma estrutura coerente e estável ao longo do desenvolvimento do projeto.

Fim do Tutorial Básico

Para mais informações, consultar o help do aplicativo ou então ler: https://git-scm.com/book/pt-br/v2