FUNDAÇÃO
UNIVERSIDADE
FEDERAL DE
MATO GROSSO DO SUL

Gerência de Configuração de Software – T01

Aula 1 - Básico em Git

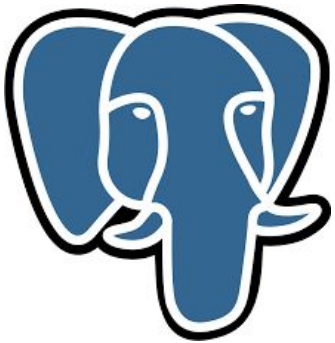
Prof. Ricardo M. Kondo

Introdução ao GIT

- Sistema de controle de versão distribuído
- Código Aberto e gratuito
- Projetado para lidar com tudo:
  - projetos pequenos e grandes
  - velocidade e eficiência

Usam Git

Linux



Servidores

Servidores/Suporte	Git	Mercurial	SVN
GitHub	V		
GitLab	**V**		
Bitbucket	V	V	
SourceForge	V	V	V

O que é um repositório Git?

- Um repositório Git é um armazenamento virtual do projeto que têm como objetivo armazenar, preservar, disseminar e possibilitar acesso a projetos.
- Permite salvar versões do seu código, arquivos, pastas, imagens, entre outros
- Pode acessar quando necessário, podendo ser repositórios abertos (*public*) ou privados (private)

Instalando Git

- Instalar o Git linux:
  - sudo apt-get install git

- Instalar o Git no Windows:
  - acesse o site oficial e faça o download do instalador do Git para Windows

- Instalar o Git no MacOS
  - acesse o site oficial e faça o download do instalador do Git para MacOS

Verificando se a instalação ocorreu com sucesso

1. Abra o terminal ou git bash
    a. digite o comando **git --version**

Configurando e-mail e nome de usuário

1. Abra o terminal ou git bash
   a. digite o comando **git --version**

Nota: Lembre-se de substituir **João Silva** e **exemplo@seuemail.com.br** com seus dados. Qualquer *commit* criado posteriormente será associado à esses dados.

2. Execute os seguintes comandos no terminal para configurar seu e-mail e nome de usuário que serão associados à sua conta GIT:

```
git config --global user.name "João Silva"

git config --global user.email "exemplo@seuemail.com.br"
```

GUI Clients

- Escolha sua interface gráfica:
  - https://git-scm.com/downloads/guis

- Fique a vontade de escolher a ferramenta de interface gráfica

Criar conta no GitLab

- Acessem https://about.gitlab.com/
- Crie sua conta
- Lembre-se, através dessa conta você poderá adicionar seus projetos e contribuir com milhões de projetos open source

Vamos brincar….

Sobre o SSH

- Secure Shell (SSH) é um protocolo de rede criptográfico para operação de serviços de rede de forma segura sobre uma rede insegura.
- Usando o protocolo SSH, você pode se conectar a servidores e serviços remotos e se autenticar neles.
- Com chaves SSH, é possível se conectar ao GitLab sem fornecer nome de usuário ou senha a cada visita.

Verificar se há chaves SSH

1. Abra o Git Bash (Windows) ou Terminal (Linux ou MacOS)
2. Digite ls -al ~/.ssh para verificar se as chaves SSH existentes estão presentes:

```
$ ls -al ~/.ssh
# Lista os arquivos no diretório .ssh, se existirem
```

3. Verifique a listagem do diretório para verificar se você já possui uma chave SSH pública. Por padrão, os nomes de arquivos das chaves públicas são um dos seguintes:
   - *id_rsa.pub*
   - *id_ecdsa.pub*
   - *id_ed25519.pub*

Gerando uma nova chave ssh

1. Abra o GitBash ou terminal
2. Cole o texto abaixo, substituindo no seu endereço de e-mail do GitLab.

   ```
   $ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
   ```

   Isso cria uma nova chave ssh, usando o e-mail fornecido como um rótulo

   ```
   > Generating public/private rsa key pair.
   ```
3. Quando você for solicitado a “Digite um arquivo para salvar a chave”, pressione Enter. Isso aceita o local padrão do arquivo

   ```
   > Enter a file in which to save the key (/c/Users/you/.ssh/id_rsa):[Press enter]
   ```
4. No *prompt*, digite uma senha segura.

   ```
   > Enter passphrase (empty for no passphrase): [Type a passphrase]
   > Enter same passphrase again: [Type passphrase again]
   ```

Adicionando sua chave SSH ao ssh-agent

Antes de adicionar uma chave SSH ao ssh-agent para gerenciar suas chaves, você deve ter verificado as chaves SSH existentes e gerado uma nova chave SSH.

1. Verifique se o ssh-agent está em execução:
   a. Se você estiver usando o *prompt* de terminal, como o Git for Windows, poderá usar as instruções “Iniciando automaticamente o ssh-agent” ou inicie-o manualmente:

```
# start the ssh-agent in the background
$ eval $(ssh-agent -s)
> Agent pid 59566
```

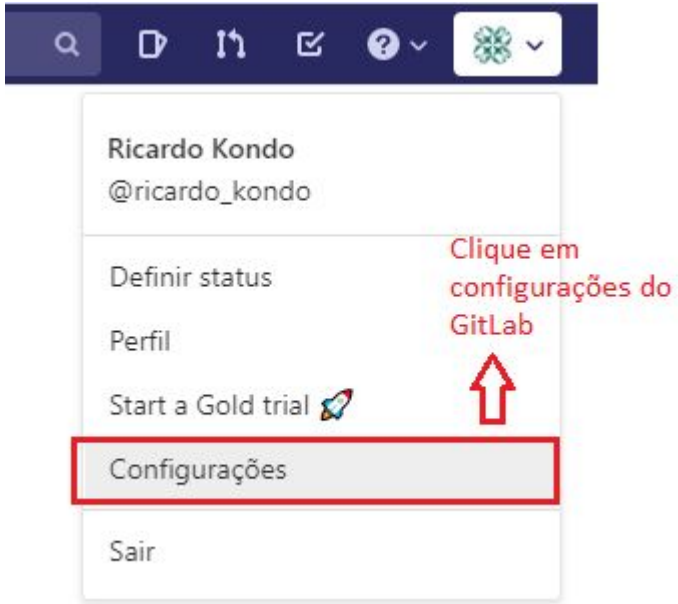
2. Adicione sua chave privada SSH ao ssh-agent. Se você criou sua chave com um nome diferente ou se está adicionando uma nova chave existente com um nome diferente, substitua *id_rsa* no comando pelo nome do seu arquivo de chave privada

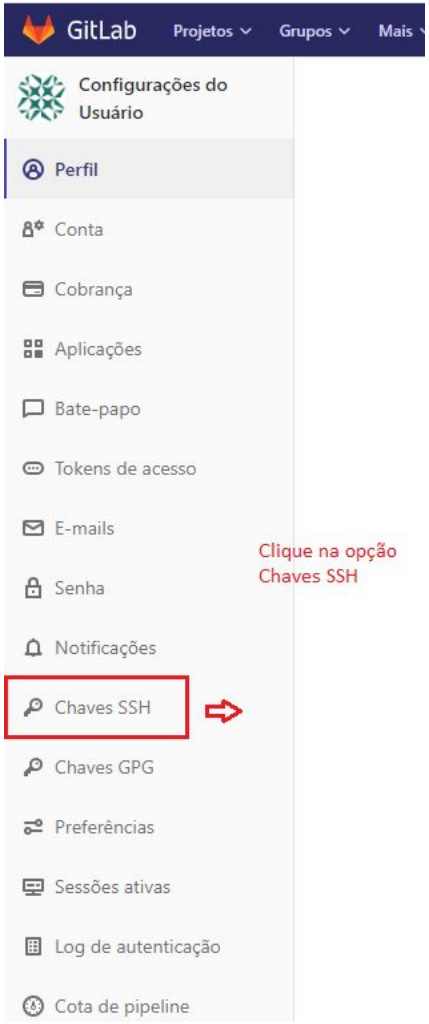
```
$ ssh-add ~/.ssh/id_rsa
```

3. Adicione a chave SSH à sua conta do GitLab

```
$ cat ~/.ssh/id_rsa (Copiar conteúdo para o local correto no GitLab)
```

GitLab - Configurações





Configurações do Usuário > **Chaves SSH**

Chaves SSH

SSH keys allow you to establish a secure connection between your computer and GitLab.

Adicionar chave SSH

Para adicionar uma chave SSH, você precisa gerar uma ou usar uma chave existente.

Chave

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Don't use your private SSH key.

Normalmente incia com "ssh-ed25519 ..." ou "ssh-rsa ..."

Adicione aqui a sua chave
ssh-rsa

Título

por exemplo, Chave do meu MacBook

Expires at

dd/mm/aaaa

Give your individual key a title

Adicionar chave

Repositório

- Instruções de linha de comando
  - Fazer *upload* de arquivos existentes do seu computador usando as instruções.

- Configuração global do Git:

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

- Três maneira:
  - Criar um novo repositório
  - Enviar uma pasta existente
  - Enviar um repositório Git existente

Criando um novo repositório

```
git clone git@gitlab.com:yourUser/repository.git

cd folderName //Acessar a pasta dos arquivos

touch README.md //Cria um arquivo chamado README.md

git add README.md //Adiciona esse arquivo

git commit -m "adiciona README" //Cria um commit

git push -u origin master //Envia os arquivos
```

Enviar uma pasta existente

```
cd existing_folder

git init

git remote add origin git@gitlab.com:yourUser/repository.git

git add .

git commit -m "Initial commit"

git push -u origin master
```

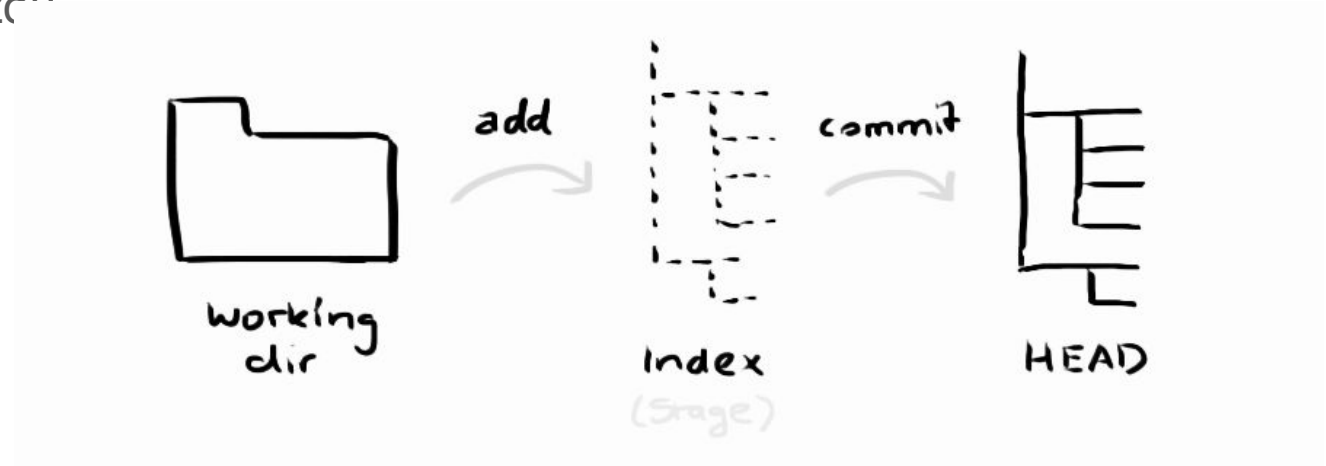
Envia um repositório Git existente

```
cd existing_repo

git remote rename origin old-origin

git remote add origin git@gitlab.com:yourUser/repository.git

git push -u origin --all

git push -u origin --tags
```

Fluxo de trabalho

Seus repositórios consistem em três “árvores” ou estados mantidas pelo *git*:

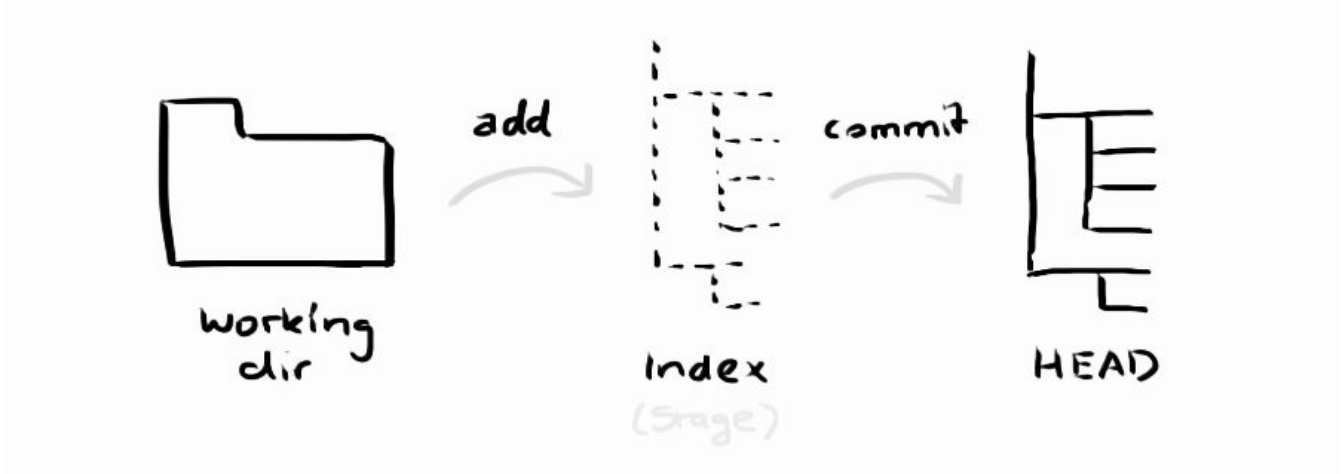
- *Working Directory* - contém os arquivos vigentes do projeto
- *Index (Staging Area)* - funciona como uma área temporária
- *Head (Repository)* - aponta para o último *commit* (confirmação) que o usuário realizo



Fluxo de trabalho

Seus repositórios consistem em três “árvores” ou estados mantidas pelo *git*:

- *Working Directory*
- *Index (Staging Area)*
- *Head (Repository)*



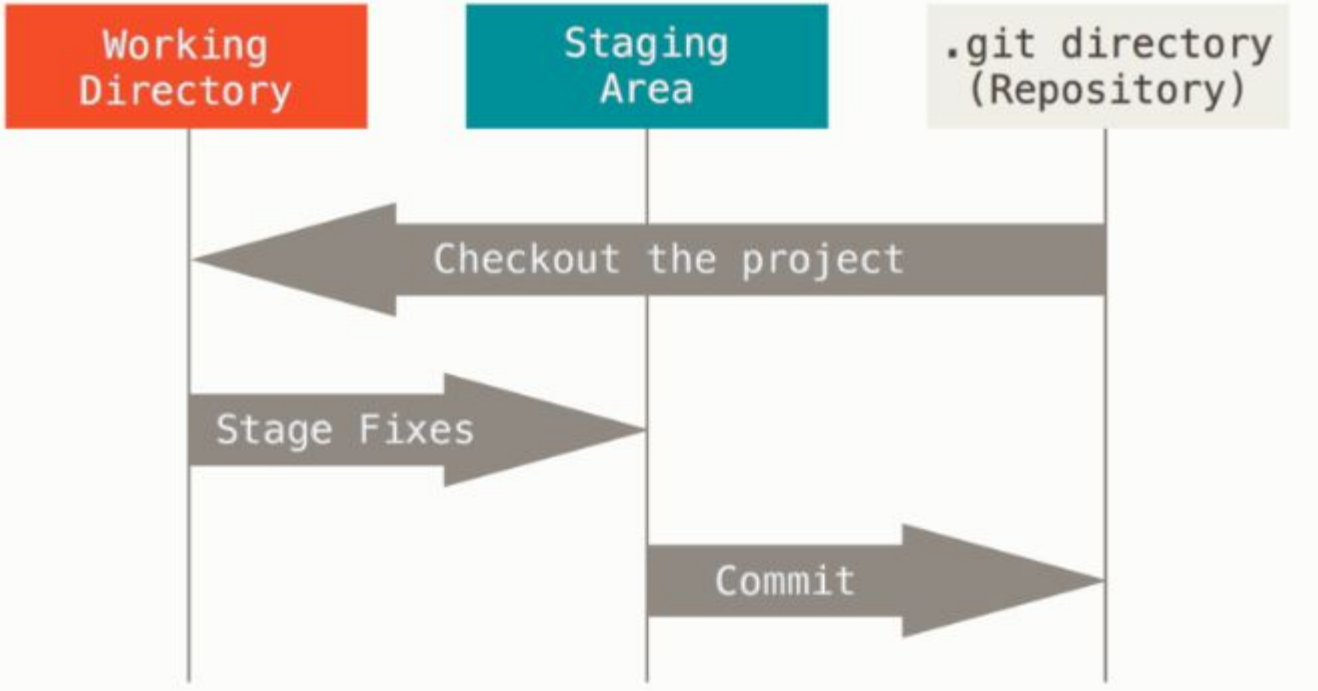


Figura 1 - Diretório de trabalho, área temporária e diretório Git

Staging Area (Área de Preparo): é um arquivo, geralmente contido em seu diretório Git, que armazena informações sobre o que vai entrar em seu próximo **commit**. É por vezes referido como o "índice" ou "*index*".

Head (Repository):
é onde o Git armazena os metadados e o banco de dados de objetos de seu projeto. Esta é a parte mais importante do Git, e é o que é copiado quando você clona um repositório de outro computador.

Working Directory (Diretório de Trabalho): é uma simples cópia de uma versão do projeto. Esses arquivos são pegos do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar.

O fluxo de trabalho básico é algo assim:

1. Você modifica arquivos no seu diretório de trabalho.
2. Você prepara os arquivos, adicionando imagens ou arquivos à sua área de preparo.
3. Você faz **commit**, o que leva os arquivos como eles estão na área de preparo e armazena essas imagens ou arquivos de forma permanente para o diretório do Git.

- Se uma versão específica de um arquivo está no diretório Git, é considerado **commited**.
- Se for modificado, mas foi adicionado à área de preparo, é considerado preparado.
- E se ele for alterado depois de ter sido carregado, mas não foi preparado, ele é considerado modificado.

Comando básico do Git - Ajuda

- Quando precisar de ajuda:
  - Git help <command>

- Site oficial:
  - https://git-scm.com/

Comando básico do Git - Clonar

- Para obter uma cópia de um repositório Git existente
  - Ex: um projeto

- Utilizar o comando:
  - `git clone <url>`

- O Git possui vários protocolos de transferência diferentes.
  - Pode utilizar:
    - protocolo:
      - `https://`
    - protocolo de transferência SSH:
      - `git://` ou `user@server:path/to/repo.git`

Comando básico do Git - Adicionar & confirmar

- Adicionar (*add*)
  - Propor mudanças (adicioná-las ao ***Index***) usando o seguinte comando:
    - `git add <arquivo> ou git add *`

- Confirmar (*commit*)
  - confirmar as mudanças realizadas no projeto, isto é, “fazer um *commit*”
    - `git commit -m “comentários das alterações”`
  - o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto

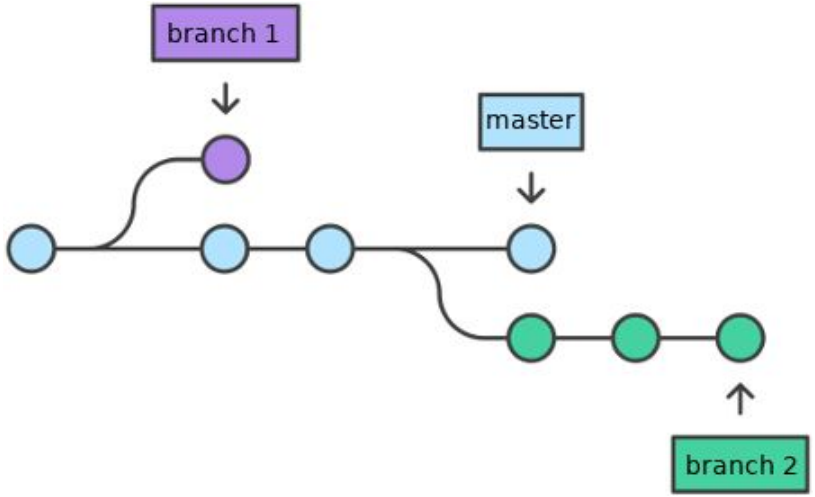
Comando básico do Git - Enviando alterações

- Enviar alterações (*push*)
  - Sua alterações agora estão no ***HEAD*** da sua cópia de trabalho local
  - Para enviar estas alterações ao seu repositório remoto, execute o comando:
    - `git push origin master`
- Altere master para qualquer ramo (*branch*) desejado, enviando suas alterações para ele

- Se você não clonou um repositório existente e quer conectar seu repositório a um servidor remoto, você deve adicioná-lo com
    - `git remote add origin <servidor>`
  - Agora você é capaz de enviar suas alterações para o servidor remoto selecionado

Ramo (*Branches*)

- São semelhantes a um ramo de uma árvore, onde o tronco seria a base do código. Desse modo é possível criar diversos ramos e fazer alterações, enquanto a base permanece intacta.
- São utilizados para desenvolver funcionalidades isoladas umas das outras.
- O ***branch master*** é o *branch* “padrão” quando você cria um repositório.
- Use outros *branches* para desenvolver e mescle-os (*merge*) ao *branch master* após a conclusão



Ramo (*Branches*)

- Criar um ramos (*branch*):
  - `git branch <nome do ramo>`
- Para acessar o ramo (*branch*):
  - `git checkout <nome do ramo>`
- Para usar um atalho para esses comandos acima:
  - `git checkout -b <nome do ramo>`
  - Desse modo o ramo será criado e em seguida irá transferi-lo para a *branch* criada
- Retornar para a *master*:
  - `git checkout master`

Ramo (*Branches*)

- Remover um ramos (*branch*):
  - `git branch -d <nome do ramo>`
- Remover um ramo (*branch*) de maneira forçada:
  - `git branch -D <nome do ramo>`
- Para acessar o ramo (*branch*) de outro usuário:
  - `git checkout origin <nome do ramo>`
- Um ramo (*branch)* não está disponível a outro a menos que você envie o ramo (*branch*) para o seu repositório remoto
  - `git push origin <nome do ramo>`

Comando básico do Git - Atualizar & mesclar

- Para atualizar seu repositório local com a mais nova versão, execute o comando ***git pull*** na sua pasta para obter e fazer *merge* (mesclar) alterações remotas
  - `git pull`
- Para fazer *merge* de um outro ramo (*branch*) ao seu ramo ativo (ex. *master*), use o comando:
  - `git merge <nome do ramo>`
- Em ambos os casos, o *git* tenta fazer o merge das alterações automaticamente. Infelizmente, isto nem sempre é possível e resulta em conflitos. Você é responsável por fazer o *merge* estes conflitos manualmente editando os arquivos exibidos pelo *git*. Depois de alterar, você precisa marcá-los como *merged* com o seguinte comando:
  - `git add <nome(s) do(s) arquivo(s)`
- Antes de fazer o *merge* das alterações, você pode também pré-visualizá-los usando
  - `git diff <ramo origem> <ramo destino>`

Comando básico do Git - Exibindo suas alterações

- Para exibir o status da árvore de trabalho, utilize o seguinte comando;
  - **git status** [<options>…] [--] [<pathspec>…]

- Exibe caminhos que têm diferenças entre arquivos de índice e o commit HEAD atual, caminhos que têm diferenças entre árvore de trabalho e o arquivo de índice e caminhos na árvore de trabalho que não são rastreados pelo Git

Comando básico do Git - Rotulando

- São etiquetas que demarcam um ponto (*commit*) que representa alguma
- mudança significativa no seu código, ou seja, uma versão (ou *release*) do seu projeto
- É recomendado criar rótulos para *releases* de software
- Usado também no SVN
- Criar um novo rótulo utilize o seguinte comando:
  - `git tag -a <nome da tag> -m <message>` //o comando -m é opcional usado para inserir uma mensagem (*tag annotated)*
- Obter informações sobre a tag, execute o seguinte comando:
  - `git show <nome da tag>`

Comando básico do Git - Rotulando

- Versionando uma *tag*, execute o seguinte comando:
  - `git push origin <nome da tag>`
- Criar um novo rótulo utilize o seguinte comando:
  - `git tag -a <nome da tag> -m <message>` //o comando -m é opcional usado para inserir uma mensagem (*tag annotated)*
- Obter informações sobre a tag, execute o seguinte comando:
  - `git show <nome da tag>`

- Excluir uma *tag*
  - Normalmente *tags* **não** são excluídas a menos que tenham sido geradas por engano! Se for o caso, primeiro realize a exclusão local:
    - `git tag -d <nome da *tag*>`
  - depois a exclusão no seu *remote*:
    - `git push --delete origin <nome da *tag*>`

Comando básico do Git - Ignorando arquivos

- Para que o Git não adicione automaticamente ou até mesmo mostre como não rastreado algum arquivo em específico.
  - Exemplos:
    - Você tem uma classe de arquivos que não deseja que o Git adicione automaticamente
    - Arquivos de log ou arquivos produzidos pelo sistema de construção
- Criar um padrão de listagem de arquivos para corresponder aos nomeados
  - `.gitignore`
  - `$ cat .gitignore`
  - `*.[oa]` //ignorar os arquivos que terminam em ".o" ou ".a" - objetos e arquivos que podem ser o produto da criação do seu código
  - `*~` //ignorar os arquivos que terminam em ".o" ou ".a" - objetos e arquivos que podem ser o produto da criação do seu código

Comando básico do Git - Ignorando arquivos

As regras para os padrões que você pode colocar no arquivo .gitignore são:

- Linhas em branco ou linhas iniciadas com # são ignoradas.
- Os padrões glob padrão funcionam e serão aplicados recursivamente em toda a árvore de trabalho.
- Você pode iniciar padrões com uma barra (/) para evitar recursividade.
- Você pode finalizar padrões com uma barra (/) para especificar um diretório.
- Você pode negar um padrão iniciando-o com um ponto de exclamação (!).

Comando básico do Git - Removendo arquivos

- Para remover um arquivo do Git, você deve removê-lo dos arquivos rastreados (com mais precisão, removê-lo da sua área de preparação - ***Staging Area***) e depois confirmar.
- Utilizar o seguinte comando:
  - `git rm <nome do arquivo>`