

# Commit, histórico e áreas do Git

## GERAR O PRIMEIRO COMMIT

Acessar o diretório do projeto via Git Bash

Verificar o estado atual do repositório

```
$ git status
```

Criar um novo arquivo README.md com um conteúdo "Repositório do curso Git Básico ao Avançado 2023"

```
$ echo "# Repositório do curso Git Básico ao Avançado 2023" > README.md
```

**Nota:** este arquivo encontra-se somente na área de trabalho e ainda não está preparado para entrar no próximo commit

Verificar o estado atual do repositório

```
$ git status
```

Adicionar o novo arquivo na área de preparo

```
$ git add README.md
```

**Nota:** o arquivo é adicionado a área de preparo e será considerado para entrar no próximo commit

Verificar o estado atual do repositório

```
$ git status
```



## SEÇÃO 2: INICIANDO COM O GIT

---

Realizar o commit com a mensagem “first commit”

```
$ git commit -m “first commit”
```

**Nota:** o arquivo é adicionado ao primeiro commit deste repositório

Verificar o histórico de commits

```
$ git hist
```

**Nota:** lembre-se que “hist” é um alias para o comando log do git, criado na seção anterior

Verificar o estado atual do repositório

```
$ git status
```



### Git show

#### ENTENDENDO O COMANDO GIT SHOW

Avaliar os detalhes do último (e primeiro) commit

```
$ git show
```



# Enviando modificações para o repositório remoto

## ENVIAR O PRIMEIRO COMMIT PARA O REPOSITÓRIO REMOTO

Tentar enviar o histórico com o primeiro commit para o repositório remoto

```
$ git push
```

**Nota:** uma mensagem de erro é exibida. Por hora, basta assumir a própria sugestão do Git. Em aulas futuras esse cenário será entendido melhor

Executar a sugestão do Git

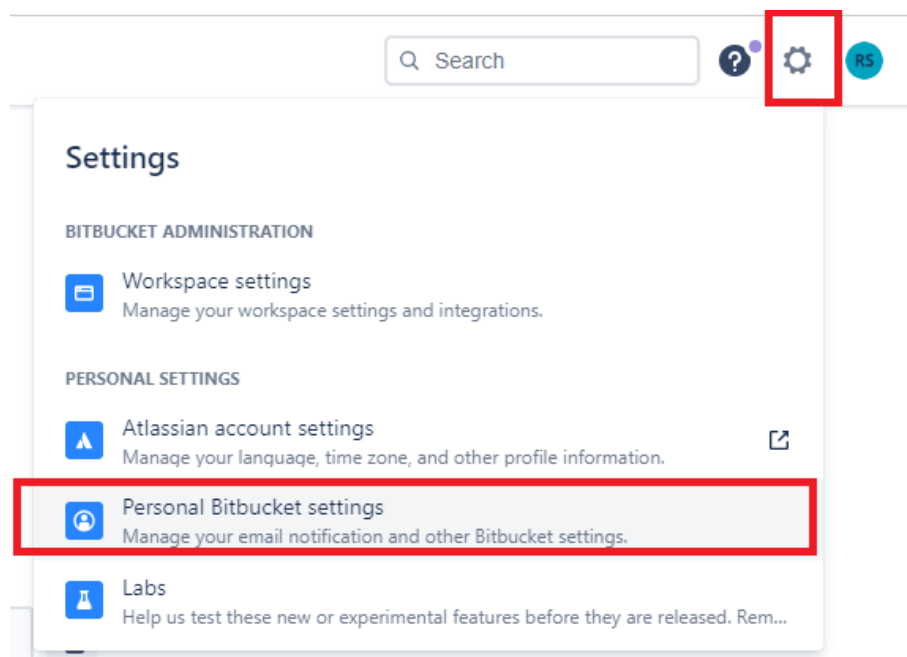
```
$ git push --set-upstream origin master
```

**Nota:** o envio acima pode solicitar um login, neste caso não feche a janela e siga para a próxima etapa

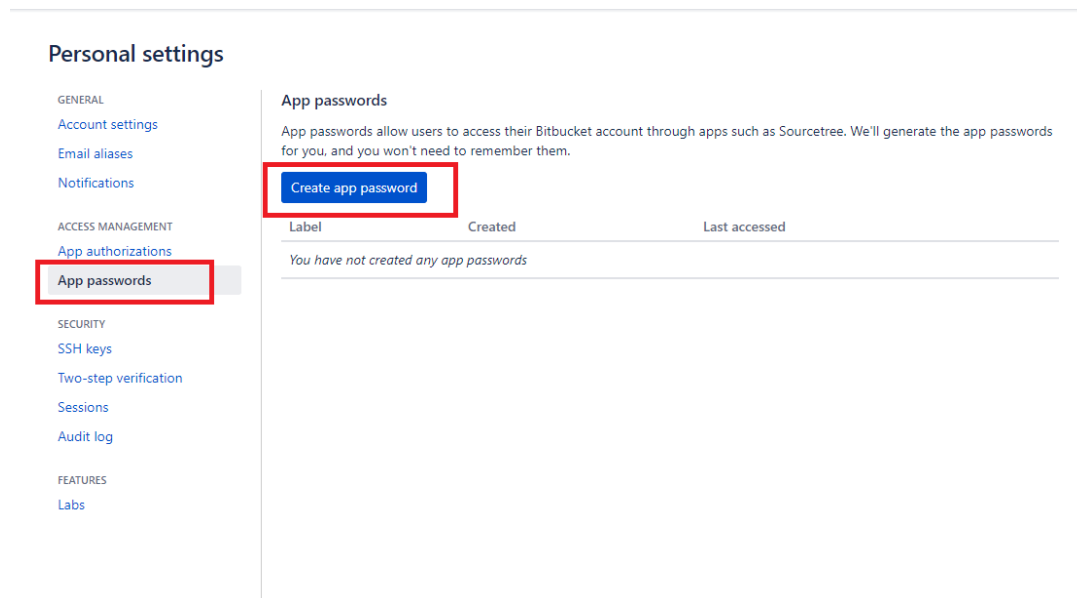
## CRIAR APP PASSWORD NO BITBUCKET

**Nota:** esta etapa é necessária somente se foi aberta uma janela de login no envio acima

Acessar o Bitbucket e clicar na engrenagem no canto direito superior e depois na opção “Personal Bitbucket Settings”



Na nova janela, clique em App Passwords e em seguida no botão “Create app password”



**Personal settings**

GENERAL

- Account settings
- Email aliases
- Notifications

ACCESS MANAGEMENT

- App authorizations
- App passwords**

SECURITY

- SSH keys
- Two-step verification
- Sessions
- Audit log

FEATURES

- Labs

**App passwords**

App passwords allow users to access their Bitbucket account through apps such as Sourcetree. We'll generate the app passwords for you, and you won't need to remember them.

**Create app password**

Label	Created	Last accessed
You have not created any app passwords		

Crie o app password definindo uma label e as permissões

### Add app password

#### Details

Label\*

#### Permissions

- |                             |  |                  |  |
|-----------------------------|--|------------------|--|
| <b>Account</b>              | <input checked="" type="checkbox"/> Email  | <b>Issues</b>    | <input checked="" type="checkbox"/> Read           |
|                             | <input checked="" type="checkbox"/> Read   |                  | <input checked="" type="checkbox"/> Write          |
|                             | <input checked="" type="checkbox"/> Write  | <b>Wikis</b>     | <input checked="" type="checkbox"/> Read and write |
| <b>Workspace membership</b> | <input checked="" type="checkbox"/> Read   | <b>Snippets</b>  | <input checked="" type="checkbox"/> Read           |
|                             | <input checked="" type="checkbox"/> Write  |                  | <input checked="" type="checkbox"/> Write          |
| <b>Projects</b>             | <input checked="" type="checkbox"/> Read   | <b>Webhooks</b>  | <input checked="" type="checkbox"/> Read and write |
|                             | <input checked="" type="checkbox"/> Write  | <b>Pipelines</b> | <input checked="" type="checkbox"/> Read           |
|                             | <input checked="" type="checkbox"/> Admin  |                  | <input checked="" type="checkbox"/> Write          |
| <b>Repositories</b>         | <input checked="" type="checkbox"/> Read   |                  | <input checked="" type="checkbox"/> Edit variables |
|                             | <input checked="" type="checkbox"/> Write  | <b>Runners</b>   | <input checked="" type="checkbox"/> Read           |
|                             | <input checked="" type="checkbox"/> Admin  |                  | <input checked="" type="checkbox"/> Write          |
|                             | <input checked="" type="checkbox"/> Delete |                  |  |
| <b>Pull requests</b>        | <input checked="" type="checkbox"/> Read   |                  |  |
|                             | <input checked="" type="checkbox"/> Write  |                  |  |

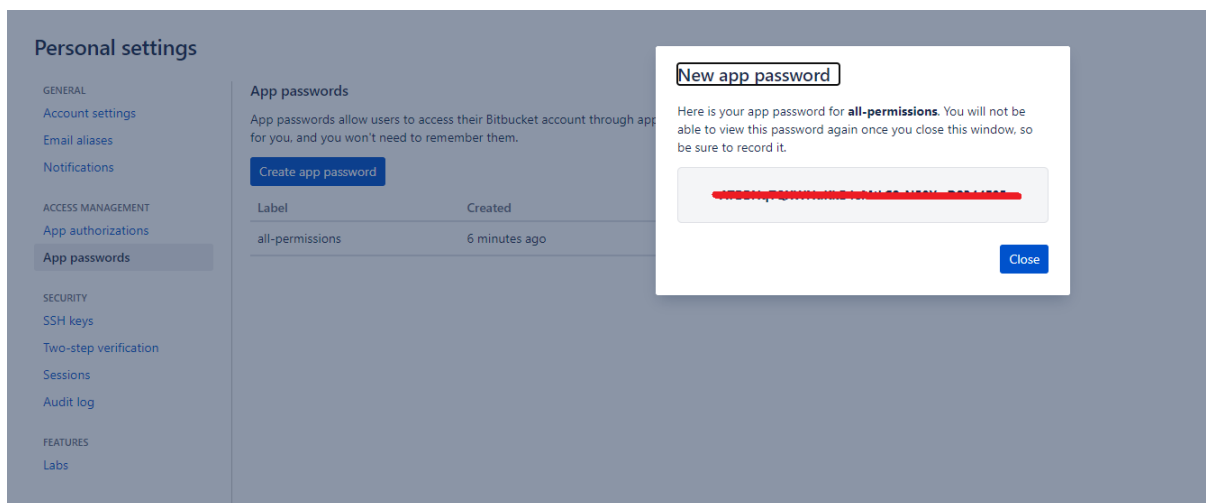
**Create** [Cancel](#)

**Nota:** as permissões representam níveis de acesso e as ações que um aplicativo ou script pode realizar em nome do seu perfil ou conta Bitbucket. As permissões concedidas a um app password controlam quais operações ele pode executar e até que ponto ele pode interagir com seus repositórios e recursos no Bitbucket



## SEÇÃO 2: INICIANDO COM O GIT

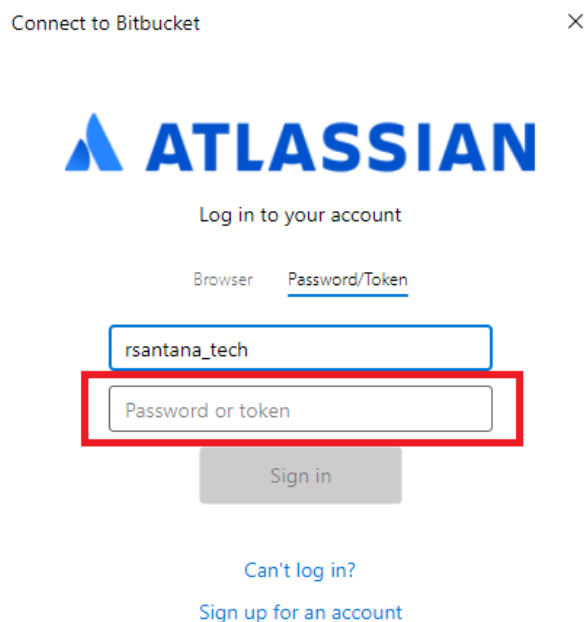
Copiar a senha gerada



**Nota:** essa senha não será exibida novamente. Portanto, certifique-se de salva-la. Será necessária na próxima etapa

Retornar para a janela de login aberta no início desta aula

Selecionar a aba "Password/Token" e preencher o campo "Password or token" com o valor copiado acima



### Git ids

#### ENTENDENDO GIT IDS

Verificar o histórico sem utilizar o alias hist

```
$ git log
```

**Nota:** o comando `git log` mostrará strings de 40 caracteres como o que vemos aqui. Essa string é o nome de um objeto de commit (contido em `.git\objects`)

Verificar o histórico utilizando o alias hist

```
$ git hist
```

**Nota:** o comando `git hist` é um atalho para o comando a seguir:

- `$ git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short`
- A opção `"%h"` torna a hash do commit abreviada com 7 caracteres
- Esses 7 caracteres são suficientes para identificar o commit



### Git objects

#### AVALIANDO ESTRUTURA ATUAL DO PRIMEIRO COMMIT

Verificar o histórico

```
$ git hist
```

Confirmar que o primeiro commit é armazenado como um tipo commit

```
$ git cat-file -t <hash do primeiro commit>
```

**Nota:** a opção `-t` é utilizado para imprimir o tipo de objeto

Avaliar o conteúdo do primeiro commit

```
$ git cat-file -p <hash do primeiro commit>
```

**Nota:** a opção `-p` é utilizado para imprimir o conteúdo com base em seu tipo

Copiar hash da tree, que neste caso é o diretório raiz

d60b3ad...

<u>Commit</u>		<u>Size</u>
<u>Tree</u>	68aba62	
<u>Parent</u>		
<u>Author</u>	Rodrigo Santana	
<u>Committer</u>	Rodrigo Santana	
<u>first commit</u>		

Confirmar que o diretório raiz é armazenado como um tipo tree

```
$ git cat-file -t <hash do diretório raiz>
```



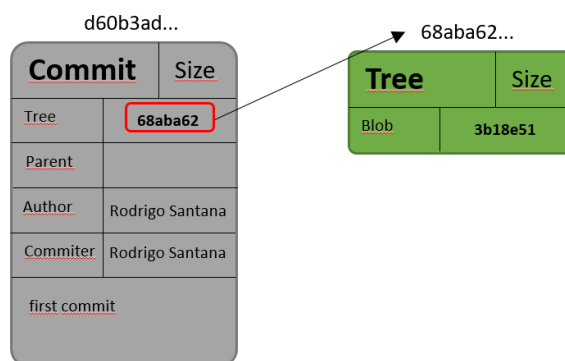


## SEÇÃO 2: INICIANDO COM O GIT

Avaliar o conteúdo do diretório raiz

```
$ git cat-file -p <hash do diretório raiz>
```

Copiar hash do blob README.md



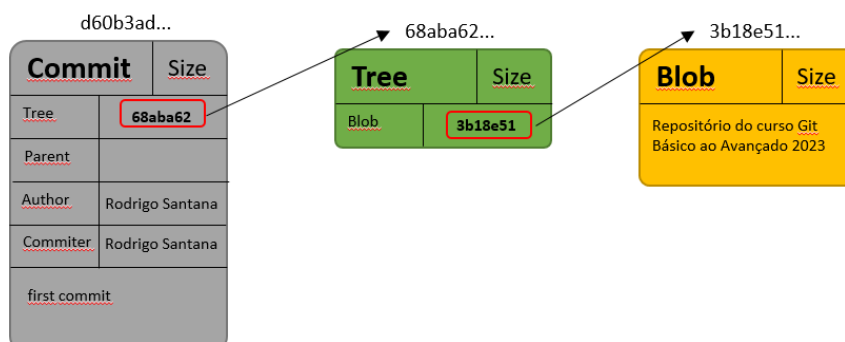
Confirmar que o arquivo README.md é armazenado como um tipo blob

```
$ git cat-file -t <hash do blob>
```

Avaliar o conteúdo do arquivo README.md

```
$ git cat-file -p <hash do blob>
```

**Nota:** o blob é um arquivo binário armazenado na estrutura interna do Git. O comando `git cat-file` consegue converter o binário em texto



## SEÇÃO 2: INICIANDO COM O GIT

---

Verificar que esse arquivo possui um conteúdo binário

```
$ cat .git/objects/<diretório correspondente a hash do blob>
```

### CRIANDO UM NOVO COMMIT

Criar um novo diretório chamado docs

```
$ mkdir docs
```

Acessar o diretório docs

```
$ cd docs
```

Criar dois arquivos

```
$ echo "Manual secops" > secops.md  
$ echo "Manual devops" > devops.md
```

Adicionar ambos a área de preparo

```
$ git add .
```

Realizar o commit

```
$ git commit -m "add docs"
```

### AVALIANDO A ESTRUTURA DO SEGUNDO COMMIT

Verificar o histórico

```
$ git hist
```



## SEÇÃO 2: INICIANDO COM O GIT

---

Avaliar o conteúdo do segundo commit

```
$ git cat-file -p <hash do segundo commit>
```

Copiar hash do diretório raiz

Avaliar o conteúdo do diretório raiz no segundo commit

```
$ git cat-file -p <hash do diretório raiz no segundo commit>
```

Copiar hash do diretório docs

Avaliar o conteúdo do diretório docs no segundo commit

```
$ git cat-file -p <hash do diretório docs no segundo commit>
```

Copiar hash do blob secops.md

Avaliar o conteúdo do arquivo secops.md no segundo commit

```
$ git cat-file -p <hash do blob secops.md no segundo commit>
```

No histórico do console, copiar hash do blob devops.md

Avaliar o conteúdo do arquivo devops.md no segundo commit

```
$ git cat-file -p <hash do blob devops.md no segundo commit>
```

Enviar o novo commit para o repositório remoto

```
$ git push
```

**Nota:** o envio ocorreu sem erro, diferentemente do primeiro envio que foi necessário executar `git push --set-upstream origin master`. Por enquanto, não se preocupe com isso

Acessar o Bitbucket e validar o conteúdo das páginas Source e Commits



### Clonando um repositório remoto

#### ENTENDENDO O COMANDO GIT CLONE

Criar um diretório “.../maria/cursogit”

Acessar esta pasta via Git Bash

Acessar novamente o Bitbucket, clicar em source e copiar a URL do git clone

Executar o git clone que foi copiado no terminal

Acessar o repositório que foi clonado

```
$ cd cursogit/
```

Listar arquivos e diretórios, confirmando que os arquivos enviados ao repositório remoto por meio do commit foram baixados para este diretório

```
$ ls
```



# Estados dos arquivos

## ENTENDENDO ESTADOS DE ARQUIVOS

Acessar o repositório de Maria via Git Bash

Verificar o estado do repositório

```
$ git status
```

Adicionar um novo arquivo temporário file.tmp

```
$ touch file.tmp
```

Verificar o estado do repositório

```
$ git status
```

**Nota:** o estado do arquivo file.tmp inicialmente é untracked. Esta arquivo ainda não está no controle de versão

**Nota:** se o comando 'git init' fosse executado em um projeto já existente com arquivos, todos os arquivos iniciariam como untracked

Adicionar file.tmp para a área de preparo (em inglês **staging area**)

```
$ git add file.tmp
```

Verificar o estado do repositório

```
$ git status
```

**Nota:** o estado atual do arquivo file.tmp é staged



## SEÇÃO 2: INICIANDO COM O GIT

---

Realizar o commit

```
$ git commit -m "add file temp"
```

Verificar o estado do repositório

```
$ git status
```

**Nota:** o estado atual do arquivo `file.tmp` é unmodified

Alterar o arquivo `file.tmp`

```
$ echo "Temporary data file" >> file.tmp
```

Verificar o estado do repositório

```
$ git status
```

**Nota:** o estado atual do arquivo `file.tmp` é modified

Adicionar `file.tmp` para a área de preparo

```
$ git add file.tmp
```

Realizar o commit

```
$ git commit -m "add description on file temp"
```



## SEÇÃO 2: INICIANDO COM O GIT

---

Verificar o estado do repositório

```
$ git status
```

Remover o arquivo

```
$ git rm file.tmp
```

**Nota:** diferentemente de um comando nativo do sistema operacional para remover, o comando “git rm” remove da área de trabalho e adiciona uma solitação de remoção na área de preparo

Validar que o arquivo foi excluído e verificar o estado do repositório

```
$ ls  
$ git status
```

Realizar o commit

```
$ git commit -m “remove file temp”
```

## ARQUIVO COM DUAS VERSÕES EM ESTADOS DIFERENTES

Adicionar um novo arquivo login.js inicialmente sem conteúdo

```
$ touch login.js
```

Adicionar login.js para a área de preparo

```
$ git add login.js
```



Adicionar um conteúdo para o arquivo login.js

```
$ echo "function login(){...}" >> login.js
```

Verificar o estado do repositório

```
$ git status
```

**Nota:** existem duas versões desse arquivo em estados diferentes. A primeira versão do arquivo vazio consta na área de preparo, com o estado *staged*. A segunda versão do arquivo com conteúdo consta na área de trabalho, com o estado *modified*. Caso fosse realizado um *commit* neste momento, seria considerado somente a versão que consta na área de preparo

Adicionar a inclusão de conteúdo na área de preparo

```
$ git add login.js
```

Verificar o estado do repositório

```
$ git status
```

Realizar o commit

```
$ git commit -m "add login"
```

Enviar o novo commit para o repositório remoto

```
$ git push
```





## Está gostando deste curso?

*Compartilhe sua experiência nas redes sociais com a tag  
**#rsantanatech** para que eu possa interagir com a sua postagem.*

**Acompanhe nas redes sociais  
e fique por dentro de todos os conteúdos.**

