



Técnicas de Programação I

---

Git

## Requerimentos

- Todos aqui estão familiarizados com a ferramenta [Jupyter Notebook](#)?
- Você pode editar seu JN no browser, e usando anaconda. Usaremos o editor de texto [Visual Studio Code](#) (NNF).
- Os encontros com a língua inglesa serão inevitáveis daqui pra frente.

## Problema Gerador

Imagine que você trabalha em uma empresa com diversos cientistas de dados, você está envolvido no desenvolvimento de um pipeline de dados em que cada profissional é responsável por uma parte do código. Um dos profissionais tem a tarefa de compor um código final, a partir das contribuições de cada colega. Como ele pode gerenciar os diferentes trechos do código, sendo trabalhados e atualizados em diferentes momentos do projeto?

Sistema de controle de versão distribuído gratuito e de código aberto, pensado para projetos grandes e pequenos, com velocidade e eficiência.

## Branch e merge em Git



O Git permite e incentiva você a ter múltiplas filiais locais que podem ser totalmente independentes umas das outras.

A criação, fusão e exclusão dessas linhas de desenvolvimento levam segundos.





- **Troca de contexto sem atrito:** Crie um ramo (**branch**) para testar uma idéia, confirme (**commit**) algumas vezes, volte para onde você ramificou, aplique uma correção (**patch**), volte para onde você está experimentando e mescle-o (**merge**).

- **Linhas de código baseadas em funções (Role-Based Codelines):** Tenha um ramo (**branch**) que sempre contenha apenas o que vai para produção, outro no qual você mescla (**merge**) o trabalho para teste e vários outros menores para o trabalho do dia a dia.

- **Fluxo de trabalho baseado em recursos:** Crie novas ramificações (**branches**) para cada novo recurso em que você está trabalhando, para poder alternar facilmente entre elas e, em seguida, exclua cada ramificação quando esse recurso for mesclado em sua linha principal.

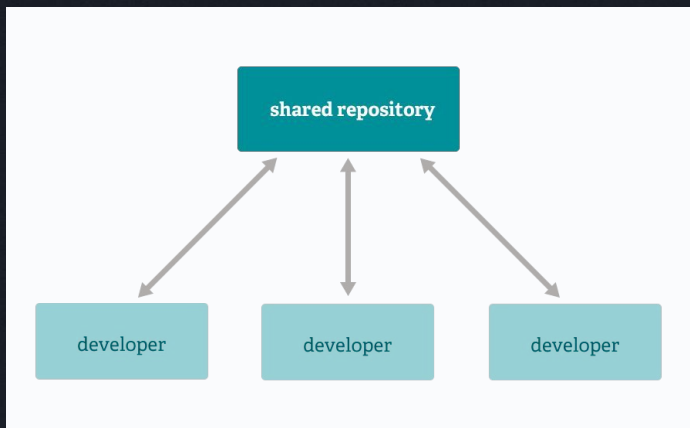


- **Experimentação Descartável:** Crie um ramo (**branch**) para experimentar, perceba que não vai funcionar e simplesmente exclua-o - abandonando o trabalho - sem que ninguém mais o veja.

Ao invés de checar apenas uma parte atualizada do atual do código fonte, você faz um “clone” de todo o repositório:

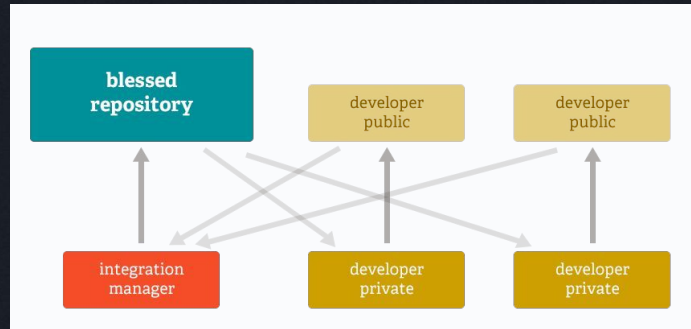
- Vários backups: Mesmo com um fluxo de trabalho centralizado, cada usuário terá essencialmente um backup completo do servidor principal. Cada uma dessas cópias pode ser enviada para substituir o servidor principal em caso de falha ou corrupção.
- Qualquer fluxo de trabalho: Devido à natureza distribuída e ao sistema de ramificação, um número quase ilimitado de fluxos de trabalho podem ser implementados com relativa facilidade

- **Fluxo de trabalho estilo Subversão (Subversion):** Um fluxo de trabalho centralizado é muito comum, especialmente entre pessoas que estão fazendo a transição de um sistema centralizado. O Git não permitirá que você faça **push** se alguém tiver feito **push** desde a última vez que você buscou, portanto, um modelo centralizado onde todos os desenvolvedores enviam **push** para o mesmo servidor funciona perfeitamente.

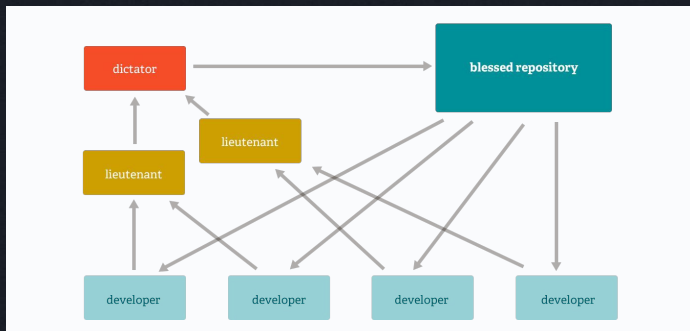


- **Fluxo de trabalho do gerenciador de integração (integration Manager):**

Uma única pessoa que se compromete com o repositório 'abençoado' (**blessed**). Vários desenvolvedores clonam esse repositório, enviam para seus próprios repositórios independentes e pedem ao integrador para extrair suas alterações. Este é o tipo de modelo de desenvolvimento frequentemente visto em repositórios de código aberto ou GitHub.



- **Fluxo de trabalho de ditadores e tenentes:** Algumas pessoas ("tenentes") ficam responsáveis por um subsistema específico do projeto e fundem (**merge**) em todas as mudanças relacionadas a esse subsistema. Outro integrador ('ditador') pode extrair alterações apenas de seus tenentes e depois enviá-las para o repositório 'abençoado' (**blessed**) de onde todos clonam novamente.



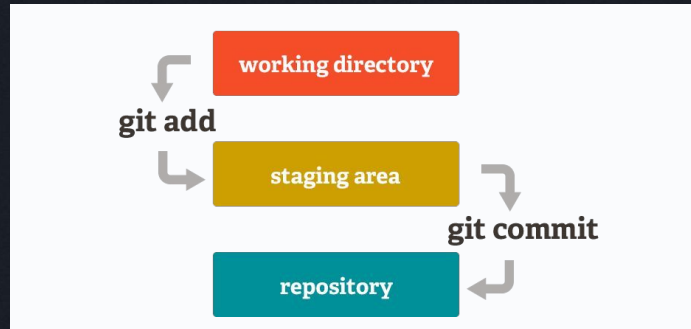


- O modelo de dados usado pelo Git garante a integridade criptográfica de cada parte do projeto. Cada arquivo e confirmação (**commit**) é verificado e recuperado por sua soma de verificação (**checksum**) quando verificado novamente.

## Área de preparação (staging area)



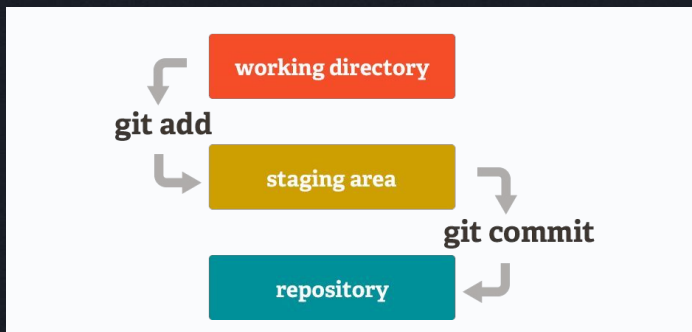
- O Git possui algo chamado "área de preparação", "área de teste" ou "índice". Esta é uma área intermediária onde as confirmações (**commits**) podem ser formatados e revisados antes de serem concluídos.



## Área de preparação (staging area)



- É possível preparar rapidamente alguns de seus arquivos e enviá-los sem submeter todos os outros arquivos modificados em seu diretório de trabalho ou ter que listá-los na linha de comando durante a confirmação (**commit**). Isso permite preparar apenas partes de um arquivo modificado.



## Área de preparação (staging area)

- Git também facilita ignorar esse recurso se você não quiser esse tipo de controle - basta adicionar um `'-a'` ao seu comando `commit` para adicionar todas as alterações em todos os arquivos à área de teste.

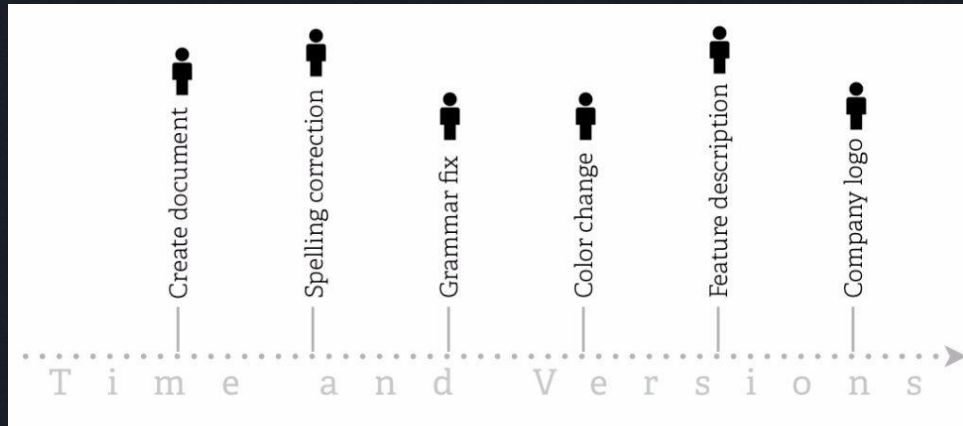


# Git - O que é controle de versão



Padrões e etapas para a criação e desenvolvimento de conteúdo:

- Criação;
- Salvamento e armazenamento;
- Edição, correção e modificação;
- Novo salvamento e armazenamento.





## Git - O que é controle de versão



Git é um software de versionalização de arquivos:

- Descrever edições no arquivo em cada ponto no tempo;



## Git - O que é controle de versão



Git é um software de versionalização de arquivos:

- Descrever edições no arquivo em cada ponto no tempo;
- Qual a razão da edição;



Git é um software de versionalização de arquivos:

- Descrever edições no arquivo em cada ponto no tempo;
- Qual a razão da edição;
- Gravação do conteúdo relacionado à edição do arquivo:
  - Troca de palavras,
  - Correções ortográficas,
  - Troca da cor ou do plano de fundo,
  - Adição do logo de uma companhia.

Git é um software de versionalização de arquivos:

- Descrever edições no arquivo em cada ponto no tempo;
- Qual a razão da edição;
- Gravação do conteúdo relacionado à edição do arquivo:
  - Troca de palavras,
  - Correções ortográficas,
  - Troca da cor ou do plano de fundo,
  - Adição do logo de uma companhia.
- Tarefa trivial para um único indivíduo e um único arquivo.

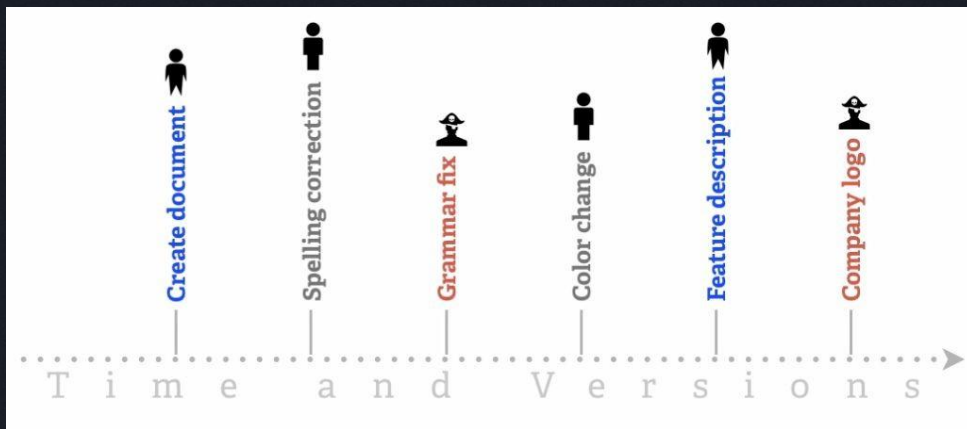
# Git - O que é controle de versão



A vantagem com Git é a versionalização de códigos editados em

## **colaboração:**

- Colaborativamente é necessário observar e rastrear cada alteração de cada indivíduo colaborador em cada ponto no tempo.





## Git - O que é controle de versão



- A vantagem com Git é a versionalização de códigos editados em

**colaboração:**



- A vantagem com Git é a versionalização de códigos editados em

### **colaboração:**

- É necessário:
  - Rastrear quem realizou a edição,
  - Quando a edição foi realizada,
  - Por que a edição foi realizada,

- A vantagem com Git é a versionalização de códigos editados em

### **colaboração:**

- É necessário:
  - Rastrear quem realizou a edição,
  - Quando a edição foi realizada,
  - Por que a edição foi realizada,
- Git unifica essas intercalações entre membros do time que editam simultaneamente o mesmo arquivo através da ação conhecida como **merge**.

- A vantagem com Git é a versionalização de códigos editados em

### **colaboração:**

- É necessário:
  - Rastrear quem realizou a edição,
  - Quando a edição foi realizada,
  - Por que a edição foi realizada,
- Git unifica essas intercalações entre membros do time que editam simultaneamente o mesmo arquivo através da ação conhecida como **merge**.
- A ação de merge, conduz o arquivo a um **estado final unificado** e **resolvido**.

- Em resumo:
  - Git é leve e rápido,
  - Fornece um histórico das edições de todos os arquivos que compõem o projeto:
  - Gráficos,
  - Designs,
  - Documentos,
  - Programming codes,
- Facilita a edição colaborativa de arquivos,
- User Friendly.



# O que é Git?



Projetos habilitados para  
Git

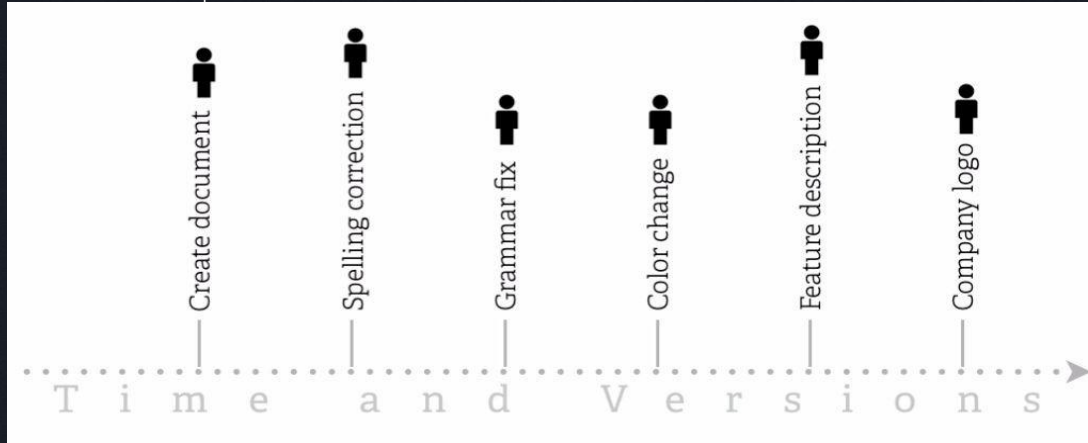


# O que é Git?



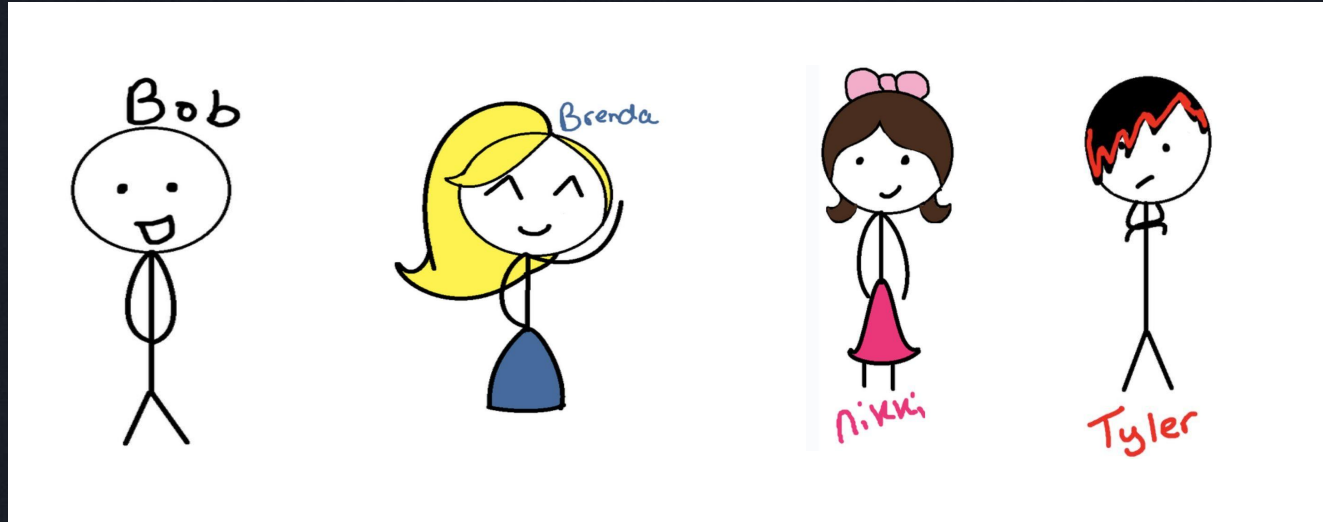
Git é uma implementação da ideia de controle de versão, ou versionamento:

- Git Local:
  - Distribuído, a conectividade não é um bloqueio.
  - É possível realizar versionamento de arquivos localmente
  - Fácil de aprender



## O que é Git?

Esses são Bob, Brenda, Nikki e Tyler, novos usuário do Git:



# O que é Git?



Rastreamento histórico:

- Para a inicialização de um projeto e configuração da estruturas de controle do Git:



```
$ git init meu_projeto
```

- Esse comando cria o diretório que conterá os arquivos do projeto, assim como os arquivos de controle, que registram as edições realizadas:



```
$ cd  
meu_projeto
```

# O que é Git?



- Rastreamento histórico:
  - Para reconhecer os arquivos e os colocar em uma zona de espera, prontos para serem confirmados ( **committed** ) permanentemente:

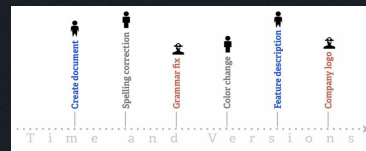


```
$ git add nome_arquivo
```

- Para registrar permanentemente uma versão histórica ou instantâneo dos arquivos conforme eles existem em um determinado momento:



```
$ git commit -m "Adicionar comentário"
```





# O que é Git?



- Rastreamento histórico:
- Para Redefinir o HEAD atual para o estado especificado:

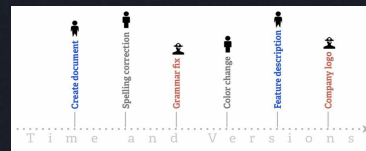


```
$ git reset HEAD^ -hard
```

- É possível forçar alterações no servidor com o repositório local, substituindo o código do repositório remoto pelo código do repositório local:



```
$ git push -f origin main
```



# O que é Git?



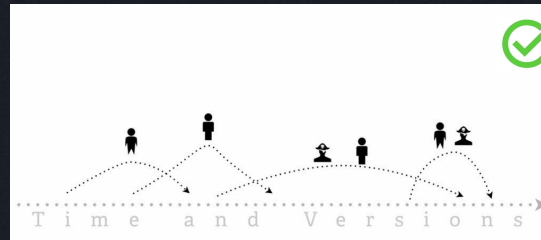
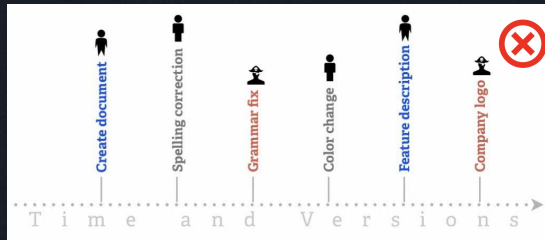
Git  
Colaborativo



# O que é Git?



- Git Distribuído:
  - **Centrado na Equipe**, para uma colaboração natural,
- Rastreamento histórico colaborativo:
  - Em um sistema colaborativo, as pessoas não trabalham de forma sequencial, mas normalmente em momentos diferentes, com sobreposições.



# O que é Git?



- Rastreamento histórico colaborativo:
  - Com o objetivo de unificar (**merge**) suas edições à cópia primária do projeto.
  - O gerenciamento da sobreposição entre tempos e trabalhos paralelos é implementado por Git:

- Um usuário pode trabalhar em um ramo (**branch**):



```
$ git checkout master
```

- Adicionar uma nova edição:



```
$ git commit -a -m "Trocando Logotipo"
```

- Enviar a nova edição a um servidor central:



```
$ git push
```

# O que é Git?



- Rastreamento histórico colaborativo:

- Outro usuário pode trabalhar em um ramo ( **branch** ):



```
$ git checkout -b brenda
```

- Adicionar uma nova edição:



```
$ git commit -a -m "Melhoramentos no código"
```

- Enviar a nova edição a um servidor central:



```
$ git push origin
```



# O que é Git?



- Rastreamento histórico
- Recuperar localmente uma cópia das versões centralizadas dos arquivos:



```
$ git pull
```

- Fundir o trabalho dos dois usuários:



```
$ git merge brenda
```

- Git provê ajuda em caso de arquivos conflitantes e na decisão de quais modificações manter:
- Caso os arquivos sejam similares, mas diferentes, eles serão fundidos automaticamente.



# O que é Git?



Git  
Avançado



- Git customizado:

- Flexível: Estrutura tipo unix, aglutinação de programas e controles de opção para se adaptar às necessidades de usuários avançados.

- Adicionar pequenos segmentos de um documento:



```
$ git add -p MeuRelatório.markdown
```

- Confirmando algumas das partes selecionadas:









```
$ git commit -m "Últimas Estatísticas"
```

- Representar graficamente o histórico de edições:



```
$ git log -- graph --decorate --abbrev  
-commit --all --pretty=oneline
```

- Instalando e configurando o Git:
  - [Instaladores](#) para Windows, Linux e Mac
  - [Configure](#) username e email (Not Credential):  
 \$ git config -global user.name "Tyler"
  - Configure email (Not Credential):  
 \$ git config -global user.mail "[Tyler@email.com](#)"

- Criando um repositório:
  - Na janela terminal inicie um novo projeto:  
 \$ git init projeto\_1
  - Entrando no diretório do projeto criado:  
 \$ cd projeto\_1
  - Criando um arquivo  
qualquer:        file.txt
  - Adicionando o arquivo ao versionamento:  
 \$ cd projeto\_1
  - Confirmando a gravação permanente:  
 \$ git commit -m "Meu primeiro Commit"



- **Content:** Git segue os movimentos do código fonte, de um arquivo para o outro, ao longo de um período de tempo.
- **Opt in:** O conceito de área de preparação (staging area) permite eleger quais arquivos participação na próxima confirmação (commitment)
- **Open:** A ideia de Code Open Source permite a oferta de contribuições, submetidas ao processo de revisão e publicação.



- **Distributed:** Por ser distribuído, Git é eficiente com as transferências de rede.



- **Conversations:** A troca de códigos se torna uma conversa.



- **People:** Git deve ser uma ferramenta invisível, foque no trabalho.



- **Journal:** Git funciona como um diário, não como um backup.



- **Anywhere:** Git funciona offline (Commits, branches, merges, re-bases, resetting history, search through historical events).



As trocas (push, pull) mantém o máximo possível localmente, puxando (pull) e empurrando (push), ocasionalmente, através de alguma conectividade.

## Ramificações (Branching) em Git.



- Ramificar (**Branching**) significa que você diverge da linha principal de desenvolvimento e continua trabalhando sem mexer nessa linha principal.
- A ramificação em Git é **leve**, com operações de ramificação quase **instantâneas**. O alternando entre **ramificações** também é **rápido**.
- O Git incentiva fluxos de trabalho que se ramificam e se mesclam (**merging**) com frequência, até mesmo várias vezes ao dia.



## Ramificações (Branching) em Git.



- Git não armazena dados como uma série de conjuntos de alterações ou diferenças, mas sim como uma série de **snapshots**.
- Quando você faz uma confirmação (**commit**), o Git armazena um objeto de commit que contém um ponteiro para o instantâneo (**snapshots**) do conteúdo que você preparou.



## Ramificações (Branching) em Git.



- Este objeto também contém o **nome** e endereço de **e-mail** do autor, a **mensagem** que você digitou e **ponteiros** para o **commit(s)** que vieram diretamente antes deste **commit** (seu pai ou pais).
- **0** (zero) pais para o **commit inicial**,
- **1** (um) pai para um **commit normal** e
- **múltiplos** pais para um **commit** que resulta de uma **fusão de dois ou mais ramos**.

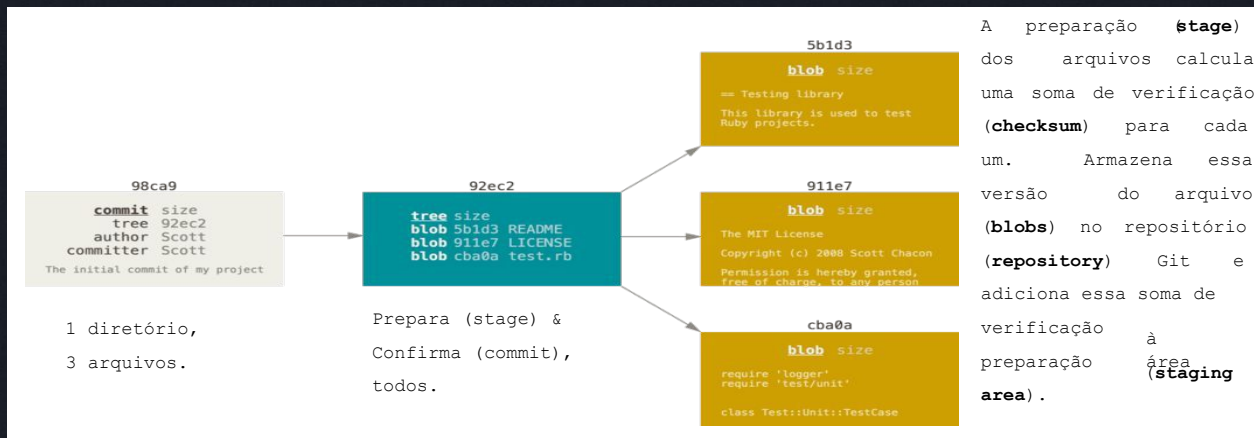




# Ramificações (Branching) em Git.



- Ao confirmar (**git commit**), Git realiza a soma a verificação de cada diretório raiz/subdiretório do projeto e os armazena como um objeto árvore (**tree object**) no repositório (**repository**) Git.



## Ramificações (Branching) em Git.



- O Git então cria um objeto de confirmação (**commit**) com os metadados e um ponteiro (**pointer**) para a árvore raiz do projeto (**root project tree**) para que possa recriar esse instantâneo quando necessário.
- Seu repositório Git agora contém cinco objetos:
  - Três **blobs**, cada um representando o conteúdo de um dos três arquivos,
  - Uma árvore (**tree**) que lista o conteúdo do diretório e especifica quais nomes de arquivos são armazenados em quais **blobs**,
  - Um **commit** com o ponteiro para aquela árvore raiz e todos os metadados do commit.
- Um ramo (**branch**) no Git é simplesmente um ponteiro móvel leve para um dessas confirmações (**commits**).

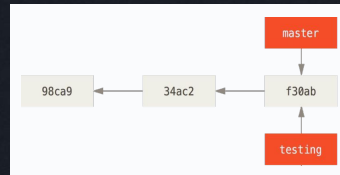


# Criando um ramo (branch) no Git

- Ao criar um novo ramo (**branch**), git cria um novo ponteiro (**pointer**) para você trabalhar.



```
$ git branch nome_do_ramo
```

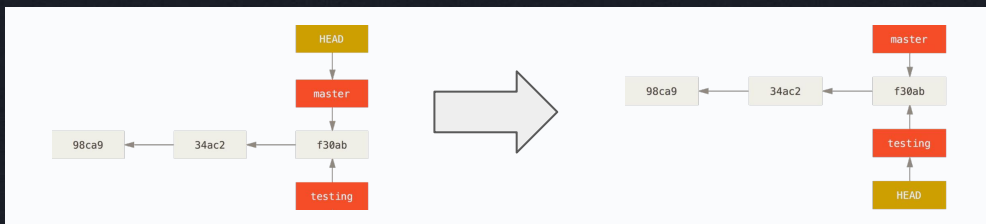


- É possível trocar entre ramos



```
$ git checkout nome_do_ramo
```

- O nome master/main costumam ser padrão git, mas podem ser alterados na criação do projeto. O termo **HEAD** indica em qual ramo (branch) você está



## Criando um ramo (branch) no Git



- Para checar em qual ramo (**branch**) está, use:



```
$ git log
```

- Realize suas edições e ao terminar, confirme-as:



```
$ git commit -a -m 'Comentário sobre alterações'
```

- Mude ao ramo printical:



```
$ git checkout master
```

## Mesclando um ramo (branch) no Git



- Uma vez que alterações foram realizadas em um ramo secundário, é necessário checar por conflitos entre as edições e então mesclar os ramo secundário ao principal:



```
$ git merge nome_do_ramo
```

- Você pode deletar o ramo (**branch**) secundário, caso deseje:



```
$ git branch -d nome_do_ramo
```



# Clonando (clone) um repositório (repository) no Git



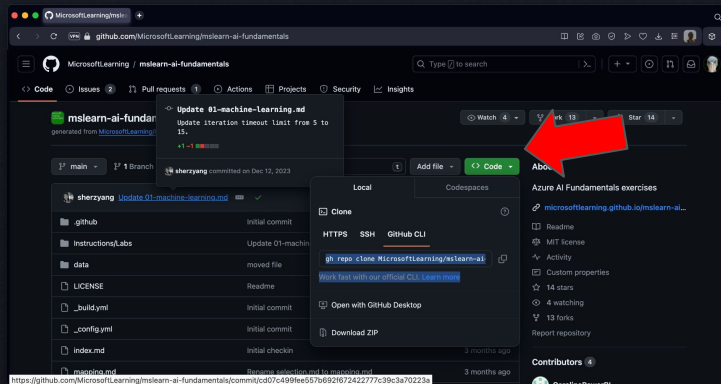
- Para clonar um repositório Git, mslearn-ai-fundamentals por exemplo, é necessário obter:

- HTTPS: Web URL,
- SSH: Password-protected SSH key,
- GitHub CLI: GitHub official Command-line Interface.



```
$ git clone web URL_repositório/SSH_repositório,
```

```
$ gh repo clone nome_repositório
```



## Puxando (pull) conteúdo no Git



- Para Incorporar alterações de um repositório remoto na ramificação atual.



```
$ git pull
```

- Se a ramificação atual estiver atrás do remoto, por padrão ele avançará rapidamente a ramificação atual para corresponder ao remoto.
- Se a filial atual e a remota divergirem, o usuário precisa especificar como reconciliar as filiais divergentes.

## Empurrando (push) conteúdo no Git

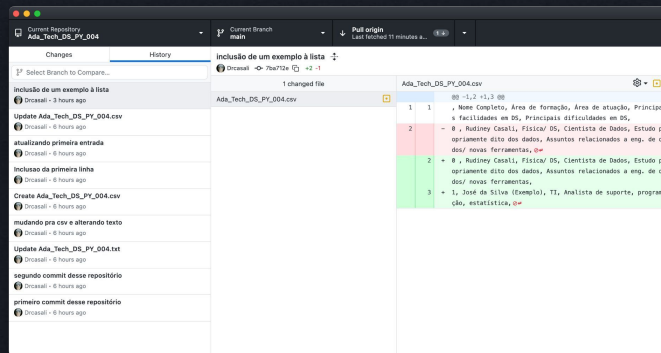
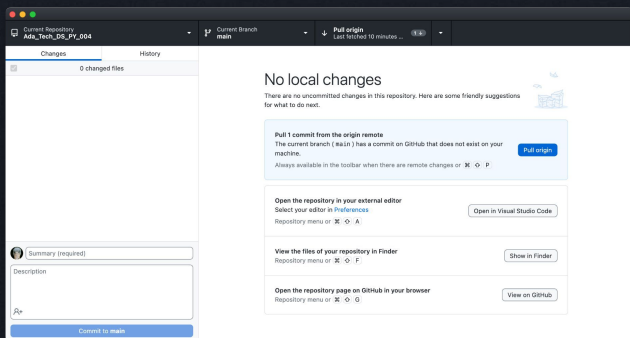


- Para atualizar referências remotas usando referências locais, enquanto envia objetos necessários para completar as referências fornecidas.



```
$ git push
```

# GitHub Desktop



## GitHub Desktop

Plataforma de desenvolvedor com tecnologia de IA que permite aos desenvolvedores criar, armazenar e gerenciar seu código. Ele usa software Git, fornecendo controle de versão distribuída do Git, além de controle de acesso, rastreamento de bugs, solicitações de recursos de software, gerenciamento de tarefas, integração contínua e wikis para cada projeto





## Diferença entre Git e GitHub Desktop



### Git

- Sistema de controle de versão distribuído para rastrear alterações em códigos-fonte durante o desenvolvimento de software.
- Projetado para coordenar o trabalho entre programadores, podendo ser usado para rastrear alterações em qualquer conjunto de arquivos.
- Objetivos incluem velocidade, integridade de dados e suporte para fluxos de trabalho distribuídos e não lineares.



# Diferença entre Git e GitHub Desktop



## GitHub

- Serviço de hospedagem de repositório Git baseado na web, que oferece todo o controle de revisão distribuída e funcionalidade de gerenciamento de código-fonte (Source Code Management - SCM) do Git, além de adicionar seus próprios recursos.



## Exercício proposto:

- Realize a clonagem do repositório [Ada\\_Tech\\_DS\\_PY\\_004](#),
- Crie um novo ramo com seu nome,
- Edite o arquivo 'Ada\_Tech\_DS\_PY\_004.csv', suas informações.
- Adicione o arquivo à área de preparação,
- Confirme suas alterações,
- E envie suas alterações para o Git, aguardando o merge de seu ramo ao principal.
- Para tal use a linha de comando do Git ou o aplicativo GitHub.



## Voltando ao Problema Gerador

Como ele pode gerenciar os diferentes trechos do código, sendo trabalhados e atualizados em diferentes momentos do projeto?

# Referências

- [GIT](#)
- [Difference Between Git and GitHub](#)
- [GIT-Download](#)
- [GIT-Documentation](#)
- [How To Fix "brew command not found" on Mac with zsh | How to fix Zsh:Command not found: Brew?](#)
- [Homebrew Documentation](#)
- [What is the PATH on the command line, and how do I change it?](#)
- [Aprenda a instalar e desinstalar o Homebrew no Mac](#)
- [GIT CHEAT SHEET](#)
- [Setting Up SSH Keys for GitHub](#)
- [Git Education](#)
- [Git pull: 7 passos para aprender a usar o comando](#)
- [Git HowTo: revert a commit already pushed to a remote repository](#)