Princípios do git e Github

Conhecimentos	
Data da aula	@5 de setembro de 2024
Tipo	Atividade em Classe Aula Expositiva

Objetivos de aprendizado

1. O que é git?
2. O que é github?
3. Exemplos práticos

🤖 O que é o git e o Github

Git é um sistema de controle de versão, enquanto GitHub é uma plataforma que utiliza o Git para fornecer serviços adicionais, facilitando a colaboração e o gerenciamento de projetos.

Introdução ao Git e GitHub

Git: É um sistema de controle de versão distribuído (DVCS), projetado para lidar eficientemente com projetos de qualquer tamanho. Ele foi desenvolvido por

Linus Torvalds em 2005 para gerenciar o desenvolvimento do kernel do Linux. O Git permite que várias pessoas trabalhem em um projeto simultaneamente, rastreando as mudanças em cada parte do código ao longo do tempo.

Em termos simples, o Git permite que você mantenha um histórico de alterações em seu código, facilitando a colaboração em equipe e o gerenciamento de projetos. Cada contribuição é registrada como um "commit", que contém uma descrição da alteração e uma referência única (hash).

Github: É uma plataforma de hospedagem de código que utiliza o Git. Fundado em 2008, o GitHub fornece um ambiente colaborativo para desenvolvedores trabalharem em projetos, facilitando o compartilhamento, colaboração e controle de versão. É especialmente popular devido à sua interface amigável e a uma série de recursos que vão além do controle de versão.

Como eles se relacionam

1. Hospedagem de Repositórios:
    a. Git é o sistema que rastreia as alterações e gerencia o controle de versão localmente em sua máquina.
    b. GitHub é um serviço que hospeda repositórios Git remotamente, permitindo que você armazene e compartilhe seu código na nuvem.
2. Colaboração em Equipe:
    a. Git possibilita a colaboração em equipe, permitindo que vários desenvolvedores trabalhem no mesmo projeto e combinem suas alterações.
    b. GitHub facilita a colaboração ao fornecer recursos como "pull requests", que permitem revisar, discutir e mesclar alterações propostas por outros membros da equipe.
3. Rastreamento de Problemas e Projetos:
    a. GitHub oferece ferramentas para rastreamento de problemas, gerenciamento de projetos e integração contínua, indo além do controle de versão.
    b. Git é focado principalmente no controle de versão, enquanto o GitHub adiciona funcionalidades extras para melhorar a colaboração e o gerenciamento de projetos.
4. Forks e Branches:

a. Git permite criar "branches" para isolar o desenvolvimento de novos recursos ou correções de bugs.

b. GitHub estende essa funcionalidade permitindo "forks", que são cópias independentes de um repositório, permitindo que desenvolvedores contribuam para um projeto sem modificar diretamente o repositório original.

5. Versionamento de Código:

a. Git controla versões localmente e realiza operações como commit, branch e merge.

b. GitHub torna esse processo mais acessível e visível, fornecendo uma interface gráfica para navegar pelas versões, comparar alterações e colaborar de maneira eficiente.

⚙️ Como instalar o git

Instalação e Configuração do Git no Windows:

- **Download:**
  - Acesse o site oficial do Git para Windows.
  - Clique no botão "Download" para baixar o instalador.
- **Instalação:**
  - Execute o instalador baixado.
  - Siga as instruções na tela, aceitando as configurações padrão, a menos que você tenha uma razão específica para alterá-las.
- **Configuração Inicial:**
  - Após a instalação, abra o "Git Bash" (um terminal Git para Windows).
  - Configure seu nome de usuário e endereço de e-mail:

```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
```

 Criando uma conta no Github

1. **Acesse o site do GitHub:**
    - Abra seu navegador da web e vá para https://github.com/.
2. **Preencha as informações básicas:**
    - Clique no botão "Sign up" (Inscrever-se), geralmente localizado no canto superior direito da página inicial.
    - Insira seu endereço de e-mail válido.
3. **Escolha um nome de usuário:**
    - Escolha um nome de usuário único para sua conta. Se o nome que você escolher já estiver em uso, o GitHub sugerirá variações ou você poderá escolher outro.
4. **Digite uma senha segura:**
    - Crie uma senha forte e única para proteger sua conta.
5. **Complete o desafio do CAPTCHA:**
    - Para garantir que você não é um robô, você precisará completar um desafio CAPTCHA. Siga as instruções na tela para provar que você é humano.
6. **Escolha um plano (opcional):**
    - O GitHub oferece planos gratuitos para usuários individuais e empresas. Selecione o plano que melhor atende às suas necessidades. Se você está apenas começando, o plano gratuito é uma excelente opção.
7. **Complete o processo de verificação de e-mail:**
    - Após preencher as informações necessárias, o GitHub enviará um e-mail de verificação para o endereço que você forneceu. Abra seu e-mail e clique no link de verificação fornecido pelo GitHub.
8. **Configure seu perfil (opcional):**
    - Após verificar seu e-mail, você será redirecionado para o GitHub. Preencha seu perfil adicionando uma foto, uma breve descrição e outras informações relevantes, se desejar.
9. **Explore o GitHub:**

- Agora, você tem sua conta no GitHub! Explore os repositórios, siga outros usuários, contribua para projetos ou crie seus próprios repositórios.

🧩Principais Comandos do Git

Alguns dos principais comandos do Git. A baixo deixo uma lista com os detalhes de cada comando. Espero que ajude em sua jornada de estudos práticos.

Comandos Git

Git Comandos - Detalhes em Português

1. **git init**
    - Inicia um novo repositório Git no diretório atual.
2. **git clone [URL]**
    - Clona um repositório Git existente para o diretório local.
3. **git add .**
    - Adiciona alterações ao índice (staging area) para prepará-las para o commit.
4. **git commit -m "mensagem"**
    - Realiza um commit com as alterações adicionadas, incluindo uma mensagem que descreve as mudanças feitas.
5. **git status**
    - Exibe o estado atual do repositório, indicando quais arquivos foram modificados, adicionados ou removidos.
6. **git log**
    - Mostra o histórico de commits do repositório.
7. **git branch**
    - Lista todas as branches locais e destaca a branch atual.
8. **git branch [nome-da-branch]**
    - Cria uma nova branch.
9. **git checkout [nome-da-branch]**
    - Altera para uma branch específica.
10. **git merge [branch]**
    - Combina as alterações de uma branch para a branch atual.
11. **git pull**
    - Atualiza o repositório local com as alterações do repositório remoto.
12. **git push [remote] [branch]**
    - Envia os commits locais para o repositório remoto.

13. **git remote -v**
    - Lista os repositórios remotos configurados.
14. **git fetch**
    - Recupera as últimas alterações do repositório remoto, mas não faz merge automaticamente.
15. **git reset [arquivo]**
    - Desfaz as alterações no arquivo especificado, removendo-o do índice.
16. **git rm [arquivo]**
    - Remove um arquivo do repositório e o inclui no próximo commit.
17. **git diff**
    - Mostra as diferenças entre as alterações que ainda não foram adicionadas ao índice.
18. **git remote add [nome-remoto] [URL]**
    - Adiciona um repositório remoto com um nome específico.
19. **git push add origin main**
    - Executado para efetuar push das alterações locais para o repositório online.

🔐 Autenticações

Autenticação do Github

>> NOME DE USUÁRIO E SENHA

Existem diferentes maneiras de se autenticar no GitHub. Uma delas é usando nome de usuário e senha, mas essa opção é considerada arriscada para informações sensíveis. Recomenda-se explorar outras opções mais seguras disponíveis.

>> TOKENS DE ACESSO PESSOAL

Os PATs (Tokens de Acesso Pessoal) são como senhas especiais que substituem o uso da senha normal ao acessar o GitHub pela API ou pela linha de comando. Você cria esse token nas

configurações do GitHub e decide quais ações ele pode realizar em um repositório ou organização. Quando você usa a linha de comando do Git para trabalhar no GitHub, em vez de digitar seu nome de usuário e senha, você insere esse token para se autenticar. Isso torna a interação mais segura e prática.

>> CHAVES SSH

Chaves SSH são como chaves especiais que ajudam as pessoas a se conectarem a computadores remotos de forma segura, sem precisar sempre digitar senha ou token.

Ao configurar o SSH, as pessoas criam uma chave especial e a adicionam ao seu perfil no GitHub. Essa chave é protegida por uma "frase secreta" para garantir ainda mais segurança. Elas podem configurar seu computador para usar essa chave automaticamente, ou digitar a "frase secreta" quando necessário.

É possível até usar essas chaves em organizações que usam uma forma avançada de login. Se a organização fornece certificados especiais, as pessoas podem usá-los para acessar os repositórios sem precisar adicionar nada à sua conta no GitHub. Resumindo, as chaves SSH tornam as interações com o GitHub mais seguras e convenientes.

>> CHAVES DE IMPLEMENTAÇÃO

Chaves de implantação são como chaves especiais que permitem acesso a apenas um lugar específico no GitHub, como um cofre digital. No GitHub, a parte da chave que todos podem ver é conectada diretamente ao local desejado (um repositório), enquanto a parte secreta fica guardada no seu próprio computador.

Essas chaves são configuradas para permitir apenas leitura por padrão, o que significa que você pode ver o que está dentro, mas não modificar nada. No entanto, se quiser também fazer alterações, você pode configurar essas chaves para ter permissão de escrita, adicionando-as ao local específico (repositório). Resumindo, são como chaves digitais que abrem a porta para um lugar específico no GitHub, e você decide se só quer olhar ou também mexer nas coisas.

Opções de seguranças adicionais

>> Autenticação de dois Fatores - 2FA

A autenticação de dois fatores (2FA) é como adicionar uma camada extra de segurança quando você entra em sites ou aplicativos. Além de digitar seu nome de usuário e senha, você precisa fornecer mais uma prova de que é realmente você.

No caso do GitHub, essa prova extra geralmente é um código gerado por um aplicativo no seu celular ou enviado por mensagem de texto. Depois de ativar a 2FA, sempre que alguém tenta entrar na sua conta, o GitHub pede esse código adicional. Ou seja, para entrar, a pessoa precisa não só da senha, mas também do código enviado para o celular.

Os donos de organizações no GitHub podem pedir que todos, membros ou colaboradores, ativem a 2FA em suas contas pessoais. Isso dificulta para pessoas mal-intencionadas acessarem informações importantes.

Além disso, em empresas, os donos podem impor regras de segurança para todas as organizações vinculadas a uma conta corporativa, garantindo uma proteção adicional para todos os envolvidos. Resumindo, é uma maneira de deixar as coisas mais seguras na internet.

>> SSO do SAML

SSO do SAML é uma forma de segurança no GitHub que permite controlar o acesso aos recursos da organização de maneira centralizada. Em vez de usar senhas, os usuários são redirecionados para um sistema central de login (IdP), como o Microsoft Entra ID ou Okta. Após autenticados lá, eles retornam ao GitHub com acesso aos recursos da organização.

Essa abordagem facilita o gerenciamento, pois os proprietários da organização controlam quem pode acessar o quê. O GitHub suporta vários provedores populares, como Active Directory, Microsoft Entra ID e Okta. Em resumo, é uma maneira mais segura e eficiente de gerenciar o acesso aos dados no GitHub.

>> LDPA

O LDAP é um protocolo usado para acessar e organizar informações em diretórios, especialmente em grandes empresas.

No contexto do GitHub Enterprise Server, ele permite integrar e gerenciar centralmente o acesso aos repositórios usando contas existentes.

O GitHub Enterprise Server é compatível com vários serviços LDAP conhecidos, como Active Directory, Oracle Directory Server Enterprise Edition, OpenLDAP e outros. Em resumo, o LDAP é uma ferramenta que ajuda na organização e controle de acesso em ambientes corporativos no GitHub.

👨‍👧‍👧 Colaboração com o Github

Breve passo a passo de como podemos criar nosso primeiro repositório e clona-lo localmente em nosso computador.

Antes de tudo, o que é um repositório?

Um repositório é o local onde contém todos os arquivos do nosso projeto. Seria como uma pastinha que guarda todos esses arquivos de nossos projeto, como os nossos projetos de software. É com eles que podemos colaborar, gerenciar nosso trabalho, acompanhar as alterações, armazenar o histórico de alterações, etc...

O que são Branches?

É uma linha de desenvolvimento separada no controle de versão. Ela permite que a gente trabalhe em modificações no código sem afetar diretamente o código principal (geralmente chamado de branch principal, como "main" ou "master"). Isso facilita o desenvolvimento simultâneo de recursos ou correções de bugs sem interferir no código estável da versão principal. Depois de concluir as alterações em uma branch, você pode mesclar essas alterações de volta ao branch principal.

E o Pull Resquest e Merge?

Um Pull Request (PR), em português, significa "Solicitação de Pull" ou "Pedido de Mesclagem". É um recurso comum em plataformas de hospedagem de código-fonte colaborativo, como o GitHub. O objetivo principal de um Pull Request é propor alterações em um

repositório e solicitar que essas alterações sejam revisadas e mescladas (merged) no código principal.

Já o "merge" é uma operação no controle de versão que combina as alterações de duas branches diferentes. Quando a gente conclui o desenvolvimento em uma branch e deseja incorporar essas alterações de volta à branch principal (ou a outra branch desejada), realizamos um merge.

Já o Fork?

É basicamente uma cópia de um repositório (um projeto de software) de outra pessoa para o seu próprio espaço no GitHub. Isso permite que você faça alterações no código sem afetar o projeto original. Se você quiser contribuir de volta, pode enviar um "pull request" para que o dono do projeto original considere suas mudanças e as incorpore.

E as Issues?

As issues são usadas para rastrear tarefas, bugs, melhorias ou qualquer discussão relacionada ao código-fonte do projeto. Elas nos fornecem um meio de comunicação e colaboração entre os membros da equipe e da comunidade. As issues podem ser abertas por qualquer pessoa, incluindo desenvolvedores do projeto e usuários externos. Servem para discussões, planejamento, atribuição de tarefas e acompanhamento do progresso.

Wikis no GitHub:

Propósito: As Wikis no GitHub têm como propósito fornecer uma plataforma colaborativa para documentação de projetos. Elas são espaços onde membros da comunidade podem contribuir com informações, tutoriais e detalhes sobre o projeto, facilitando a compreensão e colaboração.

Para que serve:

1. **Documentação Colaborativa:** Permite que membros da comunidade contribuam para a criação e atualização da documentação do projeto.

2. **Transparência:** Torna a informação acessível a todos, promovendo transparência sobre o funcionamento do projeto.
3. **Aprimoramento Contínuo:** Facilita a melhoria contínua da documentação à medida que o projeto evolui.
4. **Acesso Rápido:** Oferece um local centralizado para informações importantes relacionadas ao projeto.

Passo a Passo para Utilizar Wikis no GitHub:

1. **Criar uma Wiki:**
    - Vá para o repositório no GitHub.
    - Clique na aba "Wiki".
    - Se não existir uma Wiki, você será solicitado a criar uma.
2. **Editar Conteúdo:**
    - Cada página na Wiki tem um botão "Editar".
    - Clique em "Editar" para modificar o conteúdo.
    - Utilize a linguagem de marcação Markdown para formatar a página.
3. **Histórico de Revisão:**
    - A Wiki mantém um histórico de revisões.
    - É possível visualizar e reverter para versões anteriores.
4. **Controle de Acesso:**
    - Gerencie quem pode editar a Wiki através das configurações de permissões do repositório.

Gists no GitHub:

Propósito: Os Gists no GitHub são destinados a serem repositórios Git pequenos e independentes, geralmente contendo um único arquivo. Eles são úteis para compartilhar pequenos trechos de código, notas ou até mesmo scripts.

Para que serve:

1. **Compartilhamento Rápido:** Permite compartilhar rapidamente pequenos trechos de código ou informações.

2. **Colaboração Simples:** Facilita a colaboração em pequenos projetos ou soluções específicas.
3. **Visualização Direta:** Os Gists podem ser visualizados diretamente no navegador, sem a necessidade de clonar o repositório.

Passo a Passo para Utilizar Gists no GitHub:

1. **Criar um Gist:**
   - Vá para a página inicial do GitHub.
   - Clique em "Gist" no canto superior direito.
   - Adicione seu código ou texto e forneça uma descrição.
2. **Personalização:**
   - Escolha as opções de visibilidade (público, secreto, privado).
   - Adicione um nome de arquivo e uma descrição significativa.
3. **Salvar e Compartilhar:**
   - Clique em "Create Gist" para salvar.
   - O Gist terá uma URL única para compartilhar.
4. **Revisões e Forks:**
   - Assim como em repositórios, os Gists mantêm um histórico de revisões e podem ser bifurcados (forked).

Atividade de Aprendizagem

1. Criar um repositório local e adiciona-lo ao GitHub - remoto
2. Clonar um repositório remoto para o computador local
3. Fazer alterações >> Adicionar | Commit | Enviar arquivos
4. Criar uma nova Branch
5. Realizar um Pull Request e Merge
6. Criar um Fork

📄 Formatação com o Markdown

O que é Markdown?

Markdown é uma linguagem de marcação que oferece uma abordagem simplificada para edição de conteúdo, protegendo os criadores de conteúdo das complexidades do HTML. Enquanto o HTML é excelente para renderizar o conteúdo exatamente como foi pretendido, ele ocupa muito espaço e pode ser difícil de trabalhar, mesmo em pequenas doses. A invenção do Markdown ofereceu um ótimo equilíbrio entre o poder do HTML para descrição de conteúdo e a facilidade do texto simples para edição.

Sintaxe

A parte mais importante de qualquer comunicação no GitHub geralmente é o texto em si, mas como mostrar que algumas partes do texto são mais importantes do que outras?

Usar itálico no texto é tão fácil quanto cercar o texto-alvo com um único asterisco (*) ou um único sublinhado (_). Certifique-se apenas de fechar uma ênfase com o mesmo caractere com o qual a abriu. Esteja atento à combinação de asteriscos e sublinhados. Aqui estão alguns exemplos:

Marddown:

```
Isso é texto *itálico*.
Isso também é texto _itálico_.

    Isso é texto em itálico. Isso também é texto e
m itálico.
```

Crie texto em negrito usando dois asteriscos (**) ou dois sublinhados (__).

Markdown:

```
Isso é texto **negrito**.
Isso também é texto __negrito__.
```

```
    Isso é texto em negrito. Isso também é texto e
m negrito.
```

Você também pode misturar diferentes ênfases.

```
_Isso é texto **itálico e negrito**_ usando um úni
co sublinhado para itálico e dois asteriscos para
negrito.
__Isso é texto negrito e *itálico*__ usando dois s
ublinhados para negrito e um único asterisco para
itálico.

    Isso é texto itálico e negrito usando um único
sublinhado para itálico e dois asteriscos para neg
rito.
    Isso é texto negrito e itálico usando dois sub
linhados para negrito e um único asterisco para it
álico.
```

Para usar um asterisco literal, anteceda-o com um caractere de escape; no GFM, isso é uma barra invertida (). Este exemplo resulta em sublinhados e asteriscos sendo mostrados na saída.

Markdown:

```
\_Isso é todo \*\*texto\*\* simples\_.

    _Isso é todo **texto** simples_.
```

Declarar Cabeçalhos

HTML fornece cabeçalhos de conteúdo, como a tag `<h1>`. No Markdown, isso é suportado via símbolo #. Basta usar um # para cada nível de cabeçalho de 1 a 6.

```
###### Isso é texto H6

    Isso é texto H6
```

Link para imagens e sites

Links de imagem e site usam uma sintaxe semelhante.

Markdown:

```
![Link de uma imagem.](/learn/azure-devops/shared/media/mara.png)

    Link de uma imagem.
```

Markdown:

```
[Link para o Treinamento da Microsoft](/training)

    Link para o Treinamento da Microsoft
```

Criar Listas

Você pode definir listas ordenadas ou não ordenadas. Também é possível definir itens aninhados por meio de indentação.

Listas ordenadas começam com números. Listas não ordenadas podem usar asteriscos ou traços (-).

Aqui está o Markdown para uma lista ordenada:

Markdown:

```
1. Primeiro
1. Segundo
1. Terceiro
```

Resultado:

```
    Primeiro
    Segundo
    Terceiro
```

Markdown:

```
- Primeiro
  - Aninhado
- Segundo
- Terceiro
```

Aqui está o Markdown para uma lista não ordenada:

```
    Primeiro
        Aninhado
    Segundo
    Terceiro
```

Construir Tabelas

Você pode construir tabelas usando uma combinação de barras verticais (|) para quebras de coluna e traços (-) para designar a linha anterior como cabeçalho.

Markdown:

```
Primeiro | Segundo
-|-
1 | 2
3 | 4
```

Resultado:

```
    Primeiro    Segundo
    1   2
    3   4
```

Citar Texto

Você pode criar blocos de citação usando o caractere maior que (>).

Markdown:

```
> Este é um texto citado.

        Este é um texto citado.
```

Trabalhar com Código

Markdown fornece um comportamento padrão para trabalhar com blocos de código inline delimitados pelo caractere de crase (`). Ao decorar o texto com este caractere, ele é renderizado como código.

Markdown:

```
Isso é `código`.

    Isso é código.
```

Se você tiver um segmento de código abrangendo várias linhas, pode usar três crases (```) antes e depois para criar um bloco de código cercado.

```
var primeiro = 1;
var segundo = 2;
var soma = primeiro + segundo;
```