Universidade Federal de Uberlândia
Faculdade de Computação

Introdução ao Controle de Versão (Git/GitHub) – Prof. Daniel A. Furtado

Esta atividade tem como objetivo introduzir o mecanismo de controle de versão de código utilizando o **Git** e **GitHub**. A atividade inclui a configuração de identidade, inicialização de repositórios, gerenciamento do ciclo de vida dos arquivos e sincronização com o **GitHub** online.

Instalação. Baixe o Git do website oficial https://git-scm.com e inicie a instalação. Na tela de escolha do editor padrão, altere de **Vim** para **Visual Studio Code**. Não é necessário alterar as demais opções.

No VS Code, abra o **Terminal** (Menu **Terminal** → **Novo Terminal**), digite **git --version** e confirme se a versão instalada aparece como resposta.

Configuração de usuário. Digite os comandos a seguir no Terminal do VS Code para configurar um nome de usuário no Git. Esse e-mail também será utilizado posteriormente para acesso à conta no GitHub.

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

Definição da pasta de trabalho. Crie uma pasta em seu computador com o nome **intro-git-ppi** para conter o projeto de exemplo. Abra a pasta no VS Code. Acesse o Terminal e confirme se a pasta corrente corresponde à pasta criada. Digite o comando a seguir para que o Git faça a inicialização da pasta de trabalho, criando um repositório Git local para o projeto. Será criada uma subpasta oculta para armazenar todo o histórico de versões do projeto, metadados, etc.

```
git init
```

Crie um arquivo HTML de exemplo (**index.html**) na pasta do projeto. Em seguida, digite o comando a seguir no Terminal:

```
git status
```

Observe as mensagens apresentadas. O nome do arquivo aparece em vermelho, indicando que o Git ainda não está rastreando-o (untracked).

Área de preparação. Para que o arquivo passe a ser monitorado pelo Git, para fins de versionamento, precisamos adicioná-lo à área de preparação (staging area). Isto é feito utilizando o comando **add**:

```
git add index.html
```

Após executar o comando acima, verifique novamente o status digitando **git status**. O arquivo **index.html** deve aparecer em verde. Todos os arquivos em verde apresentados com o **git status** serão gravados pelo Git durante a próxima operação de **commit**, e poderão ser acessados posteriormente no histórico de versões do projeto.

Crie um segundo arquivo HTML na pasta do projeto com o nome **home.html**. Em seguida, adicione-o também à área de preparação. Quando é necessário adicionar, de uma vez, todos os arquivos da pasta corrente, basta utilizar o comando:

```
git add .
```

Verifique novamente o status. Os nomes dos dois arquivos devem aparecer em verde.

Gravação (commit). Para gravar uma nova versão do projeto (arquivos colocados na área de preparação) no repositório do Git, basta utilizar o comando **git commit**. A opção **-m** permite adicionar uma mensagem explicativa da versão a ser gravada. Digite o comando a seguir:

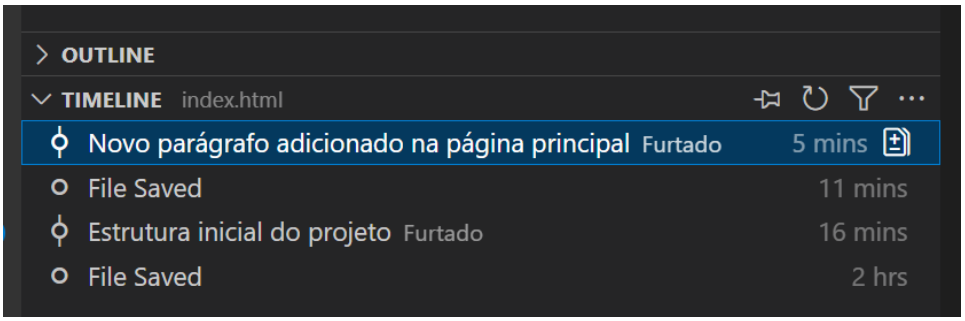
```
git commit -m "Estrutura inicial do projeto"
```

Neste momento, ao consultar o status com **git status** será apresentada a mensagem "*nothing to commit, working tree clean*", indicando que os arquivos modificados da área de preparação foram gravados no repositório e que não há mais nada a ser gravado.

Criando novas versões. Adicione um novo parágrafo de texto no arquivo **index.html**, salve o arquivo e repita o comando **git status**. Observe que o Git mostra o nome do arquivo em vermelho e informa que houve modificação. Quando há avanço relevante no projeto, uma nova versão pode ser gravada no repositório. Para isso, é necessário adicionar novamente os arquivos a serem gravados utilizando o comando **add** e prosseguir com a gravação utilizando **commit**. Execute:

```
git add .
git commit -m "Novo parágrafo adicionado na página principal"
```

Timeline de versões no VS Code. O histórico de versões do projeto pode ser acessado no VS Code por meio da aba **TIMELINE** do painel **Explorer** (extremidade inferior esquerda). Acesse essa aba e verifique que há duas versões já gravadas. Ao acessar as versões, repare que o VS Code apresenta as modificações realizadas. A aba **TIMELINE** aparece como na imagem a seguir:



Arquivo de exceções (.gitignore). Alguns arquivos do projeto podem não ser apropriados para fins de versionamento ou de disponibilização online. Por exemplo, um arquivo de configuração contendo nomes de usuários ou senhas de acesso. Para essas situações, o Git permite criar uma lista de exclusão. Crie um arquivo com o nome **config.local**, adicione o texto **senha: 1234** para simular uma informação restrita e salve-o. Em seguida, crie um arquivo na pasta raiz do projeto com o nome **.gitignore** (incluindo o ponto no início), adicione o texto a seguir e salve-o:

```
config.local
*.log
```

Esse arquivo diz ao Git o que deve ser ignorado ao usar o comando "**git add .**". Neste caso em particular, estamos dizendo que devem ser ignorados o arquivo **config.local** e todos os arquivos com a extensão **.log**.

Execute os comandos a seguir para criar uma nova versão do projeto. Repare que o arquivo **.gitignore** é adicionado à área de preparação, mas não o arquivo **config.local**.

```
git add .
git commit -m "Inclusão de arquivo de exceções"
```

Repositório remoto. Todas as versões do projeto criadas até o momento foram armazenadas no repositório local. Para publicação online, crie uma conta no GitHub utilizando o e-mail registrado no início deste roteiro. Em seguida, acesse a conta e crie um **novo repositório** público com o

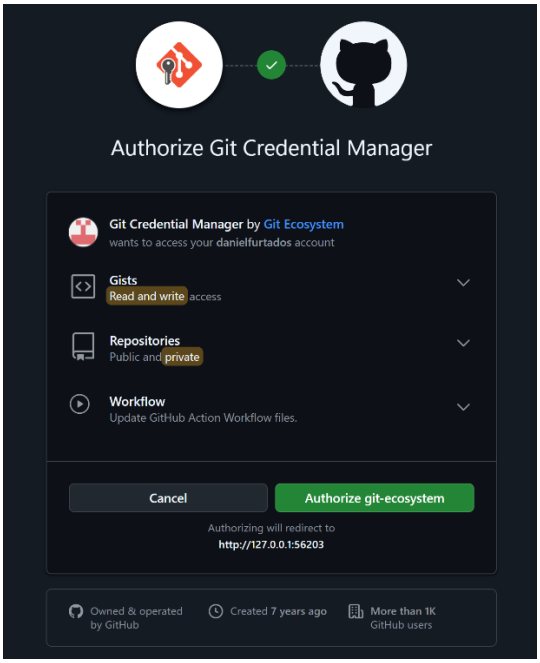
nome **intro-git-ppi**, sem marcar as opções de adicionar README, .gitignore e License, pois já criamos o .gitignore no repositório local.

Digite os comandos a seguir, substituindo a palavra **USUARIO** pelo seu nome de usuário no GitHub, para que as gravações locais sejam enviadas para o repositório remoto:

```
git remote add origin https://github.com/USUARIO/intro-git-ppi
git branch -M main
git push -u origin main
```

O primeiro comando acima adiciona o repositório remoto e o associa com o nome de identificação (apelido) **origin**. No Git, o desenvolvimento do código é estruturado em *timelines* independentes chamadas **branches** (ramificações), permitindo isolar o código principal de novas implementações. O segundo comando acima é executado para padronização. Ele renomeia a ramificação local atual do Git (denominada **master**) para **main**. Por fim, o comando **git push -u origin main** envia as gravações locais da ramificação **main** para o repositório remoto **origin**.

OBS: ao executar os comandos acima, o Git exigirá que o usuário faça a autenticação, que pode ser feita pelo próprio navegador utilizando o **Git Credential Manager** (imagem a seguir).



Acesso online no GitHub. Se os comandos anteriores foram executados com sucesso, o projeto já pode ser acessado pelo GitHub online. Recarregue a página do GitHub e acesse o repositório criado. Ao abrir o arquivo **index.html**, por exemplo, é possível visualizar os diversos "commits" clicando no botão **History** (canto superior direito).

Entrega

Adicione a URL do repositório criado no GitHub em um arquivo de texto (**repositório-online.txt**) e envie o arquivo pelo Sistema Acadêmico de Aplicação de Testes (SAAT) até a data limite indicada pelo professor em sala de aula.

Sobre Eventuais Plágios

Este é um trabalho individual. Os alunos envolvidos em qualquer tipo de plágio, total ou parcial, seja entre equipes ou de trabalhos de semestres anteriores ou de materiais disponíveis na Internet (exceto os materiais de aula disponibilizados pelo professor), serão duramente penalizados (art. 196 do Regimento Geral da UFU). Todos os alunos envolvidos terão seus **trabalhos anulados** e receberão **nota zero**.