

# I SIMPÓSIO DAS ENGENHARIAS ACET

OFICINA:

Versionamento de software com GIT

Eduardo Comin



# Introdução

- Onde armazenamos os arquivos do projeto?
- Principalmente quando trabalhamos em uma equipe com outros desenvolvedores, é necessário que o projeto esteja armazenado em um local acessível a todos os membros da equipe, com as respectivas permissões para que os mesmos sejam alterados, excluídos ou mesmo para que novos arquivos sejam criados.
- Hoje em dia já não podemos mais nos limitar a backups em computadores ou mesmo servidores, mantendo muitas cópias dos mesmo arquivos e sem um histórico do que foi modificado.

# Introdução (2)

- Para resolver este tipo de problema foram criadas ferramentas de controle de versão.
- Uma das mais utilizadas hoje em dia é o Git.
- O Git é utilizado em muitos projetos de código aberto no mundo todo. Um dos maiores e mais importantes é o kernel do Linux.
- É possível, porém, utilizá-lo para projetos privados com todas as restrições e permissões de acesso necessárias.

# Git e GitHub

- Uma grande vantagem do Git é a existência de ferramentas comerciais de hospedagem de código na Web.
- A maior provedora de hospedagem de repositórios Git, abertos ou privados, hoje é o Github (<http://www.github.com/>).
- Ele permite criar gratuitamente repositórios abertos.
- Os repositórios privados são pagos.

# Download e instalação do Git

- Para que seja possível utilizar o Git teremos que fazer o download da ferramenta para instalação. (<http://git-scm.com/>)
- O Git é uma ferramenta baseada em linha de comando, ou seja, realizamos as operações de controle dos arquivos pelo *prompt* de comando.
- Porém se você é adepto de ferramentas gráficas não se preocupe, existem algumas opções para facilitar seu trabalho. (<http://www.syntevo.com/smartgithg/>)
- Iniciaremos o curso utilizando o Git no prompt de comando, para que seja possível demonstrar suas funcionalidades que, de certa maneira, são mascaradas em botões e opções de menus nas ferramentas gráficas.

# Inicialização de um repositório

- O Git é uma ferramenta de controle de versão baseada no sistema de arquivos, ou seja, podemos fazer a associação de uma pasta diretamente a um repositório. Podemos criar um novo diretório que conterá os arquivos do nosso projeto:

***mkdir mini\_curso\_git***

***cd mini\_curso\_git***

- Como indicamos ao Git que ele deve tratar como um repositório uma pasta do computador? Isso é feito a partir do seguinte comando:

***git init***

- Exibida uma mensagem de criação do repositório e pronto, já temos um repositório.

# Adicionando arquivos ao repositório

- Podemos incluir um projeto java do NetBeans ou qualquer outro dentro da pasta e teremos nossos primeiros arquivos no repositório, porém ainda não adicionados ao Git.

## ***git status***

- Percebemos uma mensagem onde os arquivos não estão monitorados pelo git. Para que o git passe a monitorá-los apenas precisamos dizer a ele:

***git add [nomeDoArquivo]*** ou ***git add .***

# Commits

- Se novamente executarmos o git status, os arquivos passaram para condição de ***Changes to be committed***, isto é, na lista de arquivos que estão prontos para o commit.
- Commit: toda vez que terminamos de realizar as alterações nos arquivos de um projeto, precisamos "entregar" essas alterações, ou seja, realizar um commit.
- Somente as alterações que estiverem sob a condição de "Changes to be committed" é que serão entregues.



# Usuário responsável pelo commit

- Como o git sabe que usuário executou o commit, ou seja a alteração?
- Precisamos definir isso nas configurações antes do primeiro commit.

*git config user.name "Eduardo Comin"*

*git config user.email "eduardo.comin@unoesc.edu.br"*

- Podemos definir globalmente para todos os projetos git.

*git config --global user.name "Eduardo Comin"*

*git config --global user.email "[eduardo.comin@unoesc.edu.br](mailto:eduardo.comin@unoesc.edu.br)"*

- Caso não tenhamos a definição o git assume o usuário do sistema.

# Identificação das alterações

## ***git commit -m "Início do projeto"***

- A opção "-m", indica que o conteúdo a seguir é a mensagem que será utilizada para descrever o que está sendo feito no commit.
- Depois do commit, se verificarmos os estados dos arquivos do nosso sistema com o comando git status, verificamos que não há nenhuma alteração aberta em nosso projeto.

***# On branch master***

***nothing to commit (working directory clean)***

# Alterações em arquivos

- Ao fazermos novas alterações em nossos arquivos e executando o comando `git status`, percebemos que o Git já reconhece que temos arquivos que foram modificados no nosso projeto desde o nosso último commit:

***# On branch master***

***# Changes not staged for commit:***

***# (use "git add <file>..." to update what will be committed)***

***# (use "git checkout -- <file>..." to discard changes in working directory)***

***# modified: nomeDoArquivo.txt***

- no changes added to commit (use "git add" and/or "git commit -a")
- **Agora precisamos repetir novamente o ciclo, executamos o add e depois o commit;**

# Comentários importantes

- Uma ferramenta como o Git nos permite fazer alterações em nosso projeto com muita segurança pois sabemos que podemos controlar cada alteração, às vezes de maneira bem detalhada, de cada um dos nossos arquivos e usuários.
- Apesar de poderosa, o Git não é uma ferramenta totalmente automatizada (automatizar esse processo seria impossível, pois o gerenciamento das versões varia muito de acordo com o ambiente do projeto e da equipe). Precisamos interagir com o Git constantemente para podermos extrair o melhor dos benefícios que ele oferece.
- É preciso moldar a utilização do Git.

# Criação de repositórios no GitHub

- Até agora tudo que fizemos está somente na nossa máquina local, mas como garantimos o trabalho em equipe?
- Precisamos de um repositório remoto, compartilhado por todos os membros.
- GitHub é a solução mais popular e grátis para projetos OpenSource, projetos privados podem existir para planos pagos.

# Cadastro GitHub

---

- Criar cadastro;
- Criar repositório;

# Configuração do repositório remoto

- Para conseguirmos compartilhar nosso projeto entre os membros, precisamos indicar que o diretório do nosso projeto apontará para um repositório remoto, no caso, o que acabamos de criar no Github.
- Para realizarmos esse processo, o Git possui o comando `git remote add`, com o qual podemos indicar a localização do repositório remoto e o nome que queremos dar para ele (um apelido ou alias).
- Precisamos de duas informações para completar o comando, que são o alias do repositório e a url de onde ele estará disponível.
- Existe uma convenção adotada para utilização do nome do repositório remoto como "origin". Porém, qualquer outro nome pode ser utilizado. Em seguida, devemos saber também a URI do repositório, que é um caminho único que indica o local onde ele ficará armazenado. O próprio Github, segue uma convenção com relação à URI de seus repositórios e disponibiliza as instruções quando criamos um repositório:

`git remote add origin https://github.com/seuUsuario/seuRepositorio.git`

`git remote add origin https://github.com/ducomin/oficinaGit.git`

# Envio dos commits locais para remoto

- Agora queremos enviar ao servidor nossas alterações para que outros membros da equipe consigam visualizar o trabalho feito.
- Para isso, uma vez que o repositório já tenha sido inicializado com o comando `git init`, os commits locais já tenham sido realizados e o repositório remoto configurado, basta executarmos o comando `git push`, indicando qual é o repositório remoto para onde os commits serão enviados e a branch na qual o commit será aplicado.
- O repositório remoto será o "origin", que acabamos de configurar, enquanto a branch será a "master", criada por padrão sempre que um repositório é criado. O Conceito de branches serão comentamos posteriormente.



# Envio dos commits locais para remoto (2)

## ***git push origin master***

- Executando o push, teremos uma saída similar com a seguinte, no prompt:

***Counting objects: 3, done.***

***Delta compression using up to 2 threads.***

***Compressing objects: 100% (2/2), done.***

***Writing objects: 100% (3/3), 272 bytes, done.***

***Total 3 (delta 0), reused 0 (delta 0)***

***To git@github.com:[seu\_usuario\_do\_github]/mini\_curso\_git.git***

***\* [new branch] master -> master***

- Verifique a página no GitHub, onde já devem aparecer os seus commits...

# Sincronização com o repositório

- A partir das atualizações no repositório, os outros desenvolvedores que já o possuem em seus computadores não estarão sincronizados com estas alterações, ou seja, eles ainda não possuem em seus computadores as novas versões dos arquivos.
- Para que a sincronização seja realizada e o desenvolvedor tenha em seu computador as novas versões dos arquivos, basta que ele execute o comando:

***git pull origin master***

# Sincronização com o repositório (2)

- A saída no prompt será similar à abaixo, indicando os arquivos que tiveram alterações e quantas linhas do arquivo foram afetadas.

*remote: Counting objects: 5, done.*

*remote: Compressing objects: 100% (2/2), done.*

*remote: Total 3 (delta 0), reused 3 (delta 0)*

*Unpacking objects: 100% (3/3), done.*

*From github.com:[usuario\_no\_github]/mini\_curso\_git*

*c886b6a..b9aa50c master -> origin/master*

*Updating c886b6a..b9aa50c*

*Fast-forward*

*arquivoAlterado.txt | 1 +*

*1 files changed, 1 insertions(+), 0 deletions(-)*

# Clone de repositórios

- Nem sempre somos nós que criamos o repositório, caso estejamos começando em um projeto em andamento precisamos copiar o projeto a partir do ponto onde está, com todos seus arquivos e alterações.
- O processo de copiar o repositório remoto para um computador, a fim de realizar alterações nos arquivos ou até mesmo para ter os arquivos no computador, é chamado de "clone" e pode ser realizado através do comando:

***git clone [uri\_do\_repositorio]***

# Clone de repositórios (2)

- Pronto, agora é possível que outros desenvolvedores trabalhem no projeto. Basta executar o clone com o comando:

```
git clone git@github.com:[usuario_do_dono_do_repositorio]/  
mini_curso_git.git
```

- Após a execução do comando, um diretório chamado "curso-git" será criado e a seguinte saída será exibida:

***Cloning into curso-git...***

***remote: Counting objects: 3, done.***

***remote: Compressing objects: 100% (2/2), done.***

***remote: Total 3 (delta 0), reused 3 (delta 0)***

***Unpacking objects: 100% (3/3), done.***

- Agora temos a pasta do projeto dentro de onde executamos o clone.

# Alterações no mesmo arquivo

- É muito comum em equipes de desenvolvimento existirem alterações em um mesmo arquivo fonte por mais de um desenvolvedor.
- Existem algumas estratégias para que os desenvolvedores possam mexer nos arquivos.
- Uma das estratégias é realizar um "Lock" em um arquivo antes de editá-lo. Dessa maneira, nenhum outro desenvolvedor consegue manipulá-lo até que o "Lock" seja liberado. Essa abordagem possui alguns pontos falhos como, por exemplo, o desenvolvedor passa o dia inteiro trabalhando naquele arquivo e, ao ir embora para casa no final do dia, esquece de liberar o "Lock". Outro desenvolvedor conseguirá editá-lo quando o "dono do Lock" chegar ao trabalho e liberar.
- Uma outra abordagem possível é permitir que vários desenvolvedores manipulem o mesmo arquivo ao mesmo tempo. No entanto, o envio do arquivo para o repositório só pode ser feito se o desenvolvedor que deseja enviá-lo estiver devidamente sincronizado (atualizado) com o repositório. Essa é justamente a maneira seguida pelo Git.
- **Portanto sempre é necessário sincronizar antes de enviar informações.**

# Tratamentos de conflitos

- Quando sincronizamos as alterações com o repositório remoto o git tentará mesclar nossas alterações com o que vier.
- Caso as alterações não seja nas mesmas linhas, geralmente ele consegue mesclar e juntar as alterações.
- Porém quando não é possível mesclar o git gera um conflito que precisa ser corrigido manualmente.

# O tratamento manual de conflitos

- Quando não é possível mesclar uma alteração o git deixa as duas opções para que o usuário trate manualmente como deixará o arquivo.
- Assim isso gera uma nova alteração que deve ser enviada (add e commit) para garantir que tudo seja compartilhado por toda equipe, juntando as duas alterações e criando uma terceira que junta tudo.



# Visualização de um conflito

```
<html>
<head>
  <title>Proposta 1 para homepage da empresa</title>
</head>
<body>
  <h1>Cabeçalho do site</h1>
  <div>
<<<<<< HEAD
  Isso aqui é o corpo da página
=====
  Aqui está o conteúdo da página
>>>>>> e5600a4fd30fa7df61e6c1f156f55222f3041de4
  </div>
  <h1>Rodapé do site</h1>
</body>
</html>
```

## Descartando alterações antes de adicionar

- Basta copiarmos o nome do arquivo que aparece como alterado no git status.

**git checkout [nomeDoarquivo]**

# Descartando arquivos antes do commit

- Depois que adicionamos o arquivo ao índice, ainda podemos reverter.

***git reset HEAD [nomeDoArquivo]***

# Existe muito mais...

- Sugestão de estudos:
  - Git ignore
  - Branches;
  - Tags;
  - Stash;
  - Rebase;
  - Blame;
  - Ammend

# Ferramentas gráficas

- <http://www.syntevo.com/smartgit/index.html>
- <http://git-scm.com/downloads/guis>
- Cliente nativo NetBeans
- Eclipse plugin EGit

# Tags

- Criar Tag conforme quiser:

- Criar Tag:

***git tag TAG.x.x.x***

- Enviar para repositório remoto:

***git push --tags***

# GitHub & CIA

- Existem outros servidores de repositórios git muito bons além do GitHub:
- Quer um provedor de serviço git que permite grátis repositórios privados?
- <https://bitbucket.org/>

# Referências para estudo

- <http://git-scm.com/book/pt-br>
- [http://rogerdudler.github.io/git-guide/index.pt BR.html](http://rogerdudler.github.io/git-guide/index.pt_BR.html)
- <http://www.akitaonrails.com/2012/04/09/screencasts-liberados-gratuitamente>
- <http://www.akitaonrails.com/2010/08/17/screencast-comecando-com-git>



# Obrigado!

Contato

[eduardo.comin@unoesc.edu.br](mailto:eduardo.comin@unoesc.edu.br)

@ducomin

<https://www.linkedin.com/in/ducomin/>



UNOESC