**Universidade Federal de Campina Grande – UFCG**

**Centro de Engenharia Elétrica e Informática – CEEI**

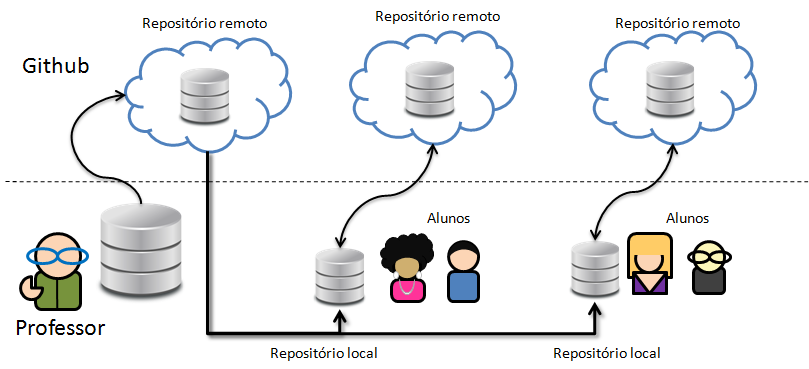
**Departamento de Sistemas e Computação – DSC**

Disciplina: Laboratório de Programação 2

# **Laboratório 05 - Parte 2 - Controle de Versão com o Git**

O objetivo desse lab é ensinar algumas operações básicas utilizando o git. Também será necessário criar conta no github, para compartilhar o código. Utilize materiais auxiliares para procurar detalhes sobre os comandos do git. Nesse roteiro, a(o) aluna(o) irá utilizar os comandos de git: init, add, status, commit, push, pull. Além disso, devem aprender os diferentes estados do seu código no git (staged, pushed, etc.) e os diferentes “locais” em que seu código fica armazenado.

**Passo 1: A união faz a força**



**Figura 1 - Modelo de repositórios usado no roteiro do lab.**

Uma das principais vantagens do controle de versão é o seu uso no **desenvolvimento em times** e em diferentes máquinas. Para isso, escolha uma colega e formem uma **dupla** para trabalharem juntos. O papel será dividido da seguinte forma:

* **Pessoa 1 (P1):** Irá configurar o repositório remoto e dar o primeiro commit;
* **Pessoa 2 (P2):** Irá iniciar o código, baixar o repositório feito por P1, e dar o commit do primeiro código;

Nesse passo vocês vão usar um **repositório pronto** que o professor disponibilizou (Figura 1). Porém, **cada dupla terá seu próprio repositório remoto**. Portanto, antes de iniciar, as duas pessoas devem possuir uma **conta e perfil no github**. Se não tem, crie a sua agora no <https://github.com/>;

**Importante:** Garanta que o seu sistema operacional possui uma versão do git instalada. Para Windows, recomendo praticar esses passos no terminal (por exemplo, no git bash).

*Obs: Todas as máquinas do lcc (1 e 2) possuem o git, porém é necessário realizar as configurações pessoais para utilizar.*

**Passo 1 - P1: Criar um repositório:**

1. P1 deve pegar a estrutura base do projeto feito pelos professores e disponível no github.
2. Navegue até a pasta onde você fará o desenvolvimento (não precisa ser no workspace), e faça:
   1. **git clone https://github.com/fgoneto/Lab5\_Git.git**
3. Abra o Eclipse e importe a pasta que você baixou.
   1. Vá em File>Import>Existing Project into workspace;
   2. Selecione a pasta Lab5\_Git e clique em Finish

Isso vai criar um **repositório local** na sua máquina copiando o projeto do github. Note que Neto não lhe deu acesso ao repositório dele, então você não pode compartilhar o código com ele. Daí, você precisa **iniciar o git e enviar esse repositório local para o seu próprio repositório remoto**.

1. Vá no github e crie seu próprio repositório. Use o nome: NomeP1\_NomeP2\_L5
2. Vá até a pasta onde você clonou o projeto (passo 2), entre na pasta Lab5\_Git e inicie um repositório local de git:
   1. **cd Lab5\_Git**
   2. Comando: **git init**
3. Conecte seu repositório local ao repositório remoto criado no seu perfil do github;
   1. Pegue a URL do seu repositório, no site do gitHub há um botão cinza na direita escrito “Clone or Download”, ele te mostra a URL como git espera
   2. Comando: **git remote add origin <URL do seu repositorio>**

**OBS:** Se der erro “**remote origin already exists**”, significa que o seu git setou o remote como o remote do prof. Neto. Você pode mudar isso **usando uma** de duas formas:

**Opção 1:** Force um novo remote usando o set-url

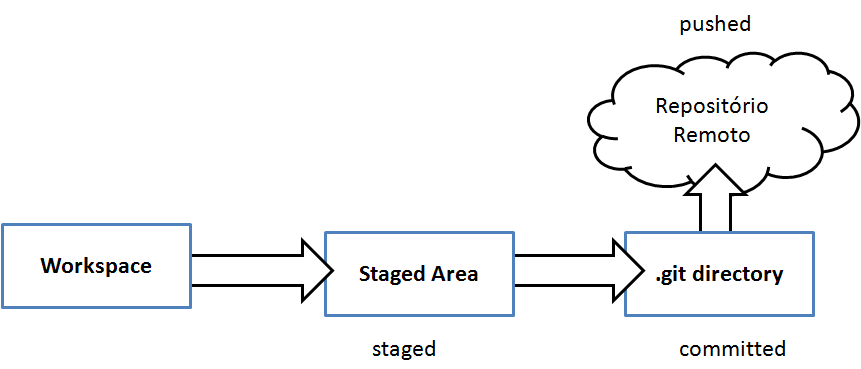
* 1. **git remote set-url origin <URL do seu repositorio>**

**Opção 2:** Delete a pasta **.git** e reinicie seu diretório git local usando:

* 1. **git init**
  2. **git remote add origin <URL do seu repositorio>**

1. Lembre de **adicionar sua colega P2** ao repositório. No home do repositório no github (em seu browser), vá em:
   1. Settings>Collaborators;
   2. Coloque o **username** de P2 como colaborador,
   3. Confirme com P2 se ela(e) recebeu o convite. (**Ela deve aceitar a invitation recebida por email**)

Pronto, agora você possui um repositório local e outro repositório remoto. Lembre que as modificações nos arquivos devem seguir o fluxo de estados explicado em sala de aula (Figura 2);



**Figura 2 - Locais e estados dos arquivos nos repositórios git**

1. No root do diretório de seu repositório local, **edite o arquivo** chamado README.md inserindo apenas o seu **nome completo** e sua **matrícula**;
2. Verifique que o arquivo está na sua máquina, mas não faz parte do git (estado staged);
   1. Comando: **git status**
   2. Esse comando mostrará todos os arquivos untracked (caso exista outros arquivos no diretório) em vermelho, entre eles veja o seu README.md;
3. Adicione o README.md a Staged Area com o comando **git add README.md**;
4. Execute **git status** para verificar que o README.md agora faz parte dos seus arquivos do repositório git local;
   1. Ao executar o **git status** os arquivos recém adicionados serão mostrados na cor verde.
5. Registre um commit indicando a sua colega as modificações que você realizou no projeto.
   1. Nesse caso, apenas a adição de um arquivo README.md;
   2. **git commit -m “Criei um arquivo README.md” README.md**

Nesse momento o estado do seu arquivo README.md passa de staged para commited. Note que ao executar o comando git status ele não mostrará mais o arquivo README.md. Chegou o momento de **enviar seu arquivo para o repositório remoto**.

1. Para enviar seus arquivos no estado committed para pushed execute o comando:
   1. **git push origin master**

Esse comando manda o git empurrar (push) os arquivos committed para o seu local remoto de nome origin (veja passo 3). O master é a principal branch de desenvolvimento… não se preocupe com isso por enquanto. :) Hora de adicionar ao “Staged” o restante do conteúdo do projeto que você clonou. Para isso, faça:

* 1. **git add --all**
  2. **git commit -m “Adicionei o restante do projeto” -a**
  3. **git push origin master**

1. Pronto, agora P2 deve baixar o repositório remoto para que vocês possam codificar em paralelo.

**Passo 1 - P2: Baixar o repositório remoto**

1. Vá até uma pasta em seu computador em que trabalhará no código (não precisa ser o workspace do eclipse).
2. Inicie um repositório local com o **git init**
3. Conecte seu repositório local com o repositório remoto fazendo:
   1. Pegue a URL do repositório da dupla, criado por P1, no site do gitHub há um botão cinza na direita escrito “Clone or Download”, ele te mostra a URL como git espera
   2. **git remote add origin <URL do seu repositorio>**
4. Agora que você está com um git conectado com um repositório remoto, baixe o repositório remoto completo para o seu git local fazendo:
   1. **git pull origin master**

Esse comando faz o git puxar (pull) os arquivos no diretório remoto (origin) para a sua workspace. O master é o branch de desenvolvimento, de novo, não se preocupe com isso por hora (spoiler dos próximos episódios)

Pronto agora você está com o repositório local. Vamos fazer sua contribuição para o repositório atualizando o arquivo README.md

1. **Importe o projeto** que foi baixado do github para o Eclipse.
   1. Vá em File>Import… > Existing project into workspace.
2. Abra o arquivo README.md com qualquer editor de texto (vi ou geany, por exemplo) e adicione na segunda linha o seu **nome completo e matrícula**; Salve o arquivo, feche e volte para o terminal.
3. Agora **no terminal** veja que um arquivo que faz parte do repositório (o README.md que P1 colocou no primeiro commit) foi alterado.
   1. Use o comando: **git status**
   2. Esse comando mostrará todos os arquivos untracked (no caso, o README.md que foi modificado) em vermelho
4. Agora envie suas modificações para seu colega fazendo:
   1. **git commit -m “Adicionei meu nome no README.md” -a**
   2. **git push origin master**

**Finalmente vocês estão prontos para codificar em paralelo.** Para essa etapa e para evitar conflitos vocês vão codificar classes distintas. Utilizem o projeto de classes que disponibilizamos para nomear seus métodos facilitando que sua colega possa saber que métodos chamar na sua classe.

**Passo 2: Codificando como um time;**

Vocês devem implementar um sistema para um hotel para animais de estimação. Cada **hóspede** do seu hotel terá um **nome**, um **tipo** (cachorro, gato, cobra, camaleão, etc.) e uma **idade** (um valor inteiro de meses). Ao fazer checkin no hotel, é criada uma **Estadia** associada aquele animal (cada estadia só está associada a um animal). Além do respectivo animal, cada **estadia** tem uma **quantidade de dias** na qual o animal estará hospedado e o **valor do quarto** que ocupará. O sistema conta também com uma **recepção**, onde serão organizadas as estadias, note que é possível existirem estadias iguais. A recepção também realiza **check-out** a partir do **nome de um animal** e **check-in** com as **informações necessárias para iniciar uma estadia**. Por último, o hotel faz a gerência (delegação/forwarding) para a recepção em qualquer operação.

**O objetivo é que P1 e P2 criem classes distintas**. Como, inevitavelmente, cada classe necessita de otura, disponibilizamos o **projeto de classes no final do roteiro** para verificar qual a assinatura dos métodos (retornos e entrada) que serão chamados.

**Passo 2 - P2: Codificar (Classe *Animal*)**

Crie a classe **Animal** que possui um nome, tipo e idade. Todo animal deve ser criado com esses parâmetros. Dois animais são iguais se possuem o mesmo nome e tipo. Crie métodos gets e sets para sua classe animal. **Não crie um projeto no eclipse,** pois você usará o projeto importando anteriormente.

*Obs: Utilize o diagrama abaixo para não entrar em conflito com a sua colega de implementação. Nele há todas as assinatura de métodos, bem como nomes de atributos, etc.*

*Obs 2: Ao terminar a implementação, envie o código para o github, assim sua dupla poderá baixar sua implementação e acoplar a atual. Lembre dos passos para o envio de alterações no projeto do github: add>commit>push; bem para receber as alterações feitas: pull.*

**Passo 2 - P1: Codificar (Classe *Recepção*)**

Crie a classe **Recepcao** que possui uma coleção de de Estadias (*Obs: Utilize o diagrama abaixo para não entrar em conflito com a sua colega de implementação. Nele há todas as assinatura de métodos, bem como nomes de atributos, etc.*). A Recepção pode fazer check-in de um Animal (recebendo os dados necessários para a criação de uma Estadia e de um Animal). Check-out a partir do nome de um animal. Deve informar o número de hóspedes no pet shop bem como o lucro total que tais hóspedes darão.

A recepção também deve informar, em forma de String, quem são os hóspedes atuais, da seguinte forma:

*Estadias:*

*Vagabundo (Cachorro): 5 dias com o preço de R$ 35.00*

*Paola Bracho (Cobra): 3 dias com o preço de R$ 48.00*

*Frederico Mercury (Chinchila): 2 dias com o preço de R$ 25.00*

*Dama (Cachorro): 5 dias com o preço de R$ 38.00*

*Obs: Ao terminar a implementação,* ***envie o código para o github****, assim sua dupla poderá baixar sua implementação e acoplar a atual. Lembre dos passos para o envio de alterações no projeto do github:* ***add>commit>push****; bem para receber as alterações feitas:* ***pull****.*

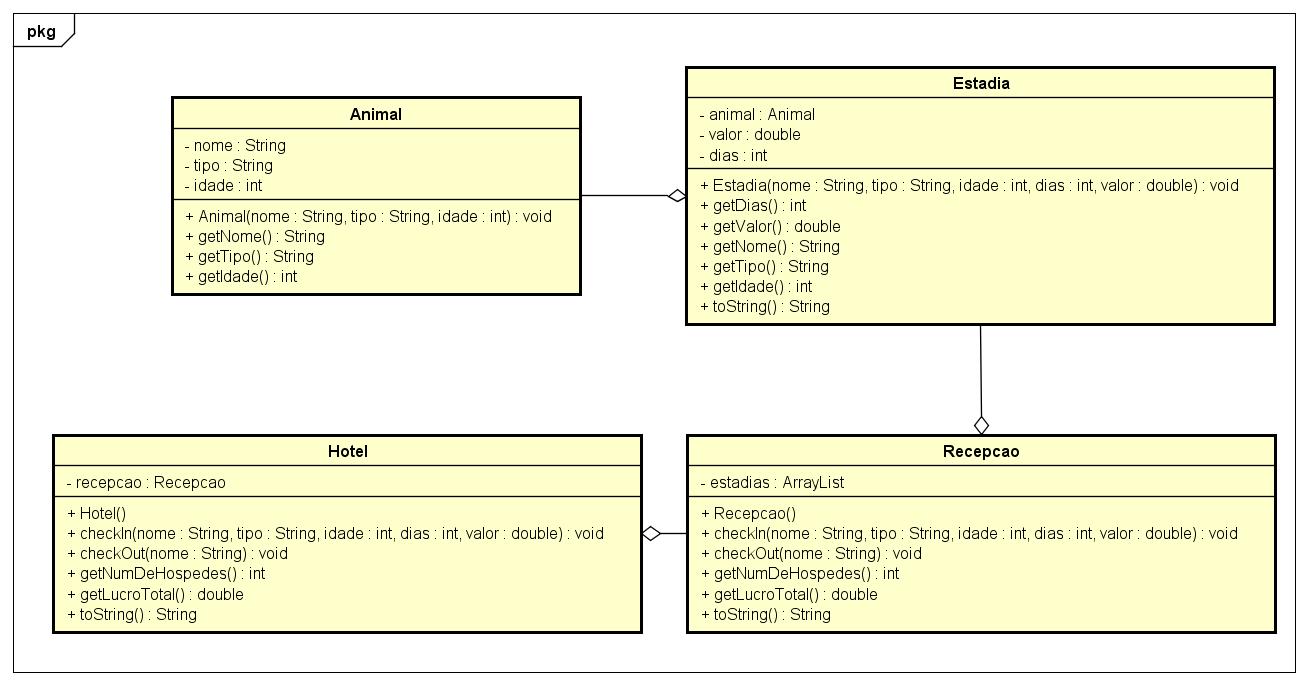
**Passo 3 - P2: Codificar (Classe *Estadia*)**

Implemente a classe **Estadia** que é uma representação da Estadia de um Animal no Pet Hotel. Cada Estadia possui um animal correspondente (que estará hospedado), o valor do quarto que ficará hospedado e a quantidade de dias. Note que, como uma Estadia está relacionada a um Animal, ela deverá ser capaz de informar ao sistema os dados do dito cujo (**use delegação!**), bem como os valores do quarto e dos dias de hospedagem.

Cada Estadia também deve ser capaz de informar ao sistema o Animal relacionado em forma de String, da seguinte forma:

*Garfield, Gato, 6 dias com o preço de R$ 80.00*

*<Nome, Tipo, X dias com o preço de R$ valor>*



*Obs: Ao terminar a implementação, envie o código para o github, assim sua dupla poderá baixar sua implementação e acoplar a atual. Lembre dos passos para o envio de alterações no projeto do github: add>commit>push; bem para receber as alterações feitas: pull.*

**Passo 3 - P1: Codificar (Classe *Hotel*)**

Implemente a classe Hotel do sistema que possui uma Recepção. Hotel possui todas as funcionalidades delegadas para Recepção (não use o ctrl+c, ctrl+v).

*Obs: Ao terminar a implementação, envie o código para o github, assim sua dupla poderá baixar sua implementação e acoplar a atual. Lembre dos passos para o envio de alterações no projeto do github: add>commit>push; bem para receber as alterações feitas: pull.*

Dicas importantes:

* **Usem controle de versão**, é importante aprender a criar software em grupo, **ninguém é lobo solitário/forever alone**;
* **Dividam a implementação** entre P1 e P2, os **commits serão avaliados**, caso apenas um membro da dupla faça commits, a pontuação será única dele;
* **Utilizem o diagrama abaixo** para assinatura de métodos, é importante para não ocorrerem conflitos entre as classes (nome de variáveis, métodos, etc.);
* No projeto baixado do diretório remoto do professor, vocês encontrão uma classe de testes simples, é preferível que ela fosse preenchida e utilizada para a validação e verificação do sistema de vocês;

Link para o diagrama: <https://goo.gl/41cfB1>

*Boa Implementação!*

*Francisco Neto e Gabriel Vinha*