Minicurso de git

Lucas Castro (Foo)

O que é git?

- git foi feito para controlar versões, ajudando a manter a organização e integridade dos projetos
- Permite fazer “checkpoints” e ramificações do seu código, trabalho, relatório ou qualquer outro projeto
- Facilita o desenvolvimento em equipe
- Simples de usar e com ampla documentação

Como funciona?

Utilizem (e divulguem) o manual de sobrevivência disponível aqui: github.com/IcbcFoo/Minicurso-git

Sua primeira vez aqui?

- O git mantém informações sobre quem realiza as modificações em um projeto
- A primeira vez que você utiliza git em uma nova máquina, você precisa se identificar
- Estes comandos falam para o git que existe um usuário desta conta nessa máquina, que é o “Foo”, com email “foo@email.com”

 # Modifica o nome

$ git config --global user.name Foo

 # Modifica o email

$ git config --global user.email foo@email.com

 # Lista os usuários

$ git config --global --list

Criando um repositório

- Repositório nada mais é do que um diretório que possui controle de versões do git
- Todo o controle de versões feito pelo git está no diretório oculto *.git*, dentro do repositório (basta saber que ele está ali, você nunca vai mexer dentro dele)

 # Cria um diretório qualquer

$ mkdir teste_git

 # Entra no diretório e inicializa um repositório git

$ cd teste_git

$ git init

 # Agora existirá um .git neste diretório

$ ls -a

Adicionando arquivos e dando commits

- Toda mudança no diretório que você quer que se mantenha na sua árvore precisa ser informada ao git
- Isso inclui a primeira vez que você cria um arquivo ou quando você modifica algum já existente
- As alterações só são salvas quando você realiza um *commit*
- Imaginem o commit como o save de um jogo, o git guarda o estado atual do repositório, dá um nome e coloca na lista de commits

 # Cria um arquivo

$ echo “CACo <3” >> teste.txt

 # Notifica ao git que ele deve observar o arquivo

$ git add teste.txt

 # Salva a mudança em um commit

$ git commit -m “meu commit”

Vendo o movimento do repositório

- É possível verificar o estado atual do repositório com o comando *git status*
- Também é possível verificar a lista de commits e outras informações sobre *branches* (veremos daqui a pouco) com o comando *git log*
- O uso desses dois comandos tem que ser algo frequente, para você sempre saber o que está mudando antes de cada commit

 # Verifica o estado atual do repositório

$ git status

 # Verifica o histórico de commits e onde cada branch se encontra (veremos branches depois)

$ git log

Exercício 1

Lembrando de tudo:

- Criar um repositório para o exercício com: *$ git init*
- Adicionar arquivos (ou mudanças neles) com: *$ git add <nome do arquivo>*
- Dar commit com: *$ git commit -m <mensagem do commit>*
- Ver a situação atual com: *$ git status*
- Utilizem o manual

Exercício:

Criem um programa (bem bobo, o objetivo é mexer com o git) com uma função chamada “soma” que some dois inteiros e outro programa com uma função “printa” que escreve os números de 1 a 10 no terminal utilizando um repositório git. Comecem o segundo programa apenas quando o primeiro já estiver “commitado”.

Faça quantos adds e commits quiserem, lembrem-se de utilizar o git status para ver mudanças.

Branches

- Branches são ramificações nas versões do seu repositório
- O ramo principal é o *master*, você pode criar outros branches e alternar entre eles. O branch no qual você se encontra no momento é apelidado de *HEAD*
- Pense no master como o tronco de uma árvore e os demais branches como os galhos que saem do tronco
- Depois que você cria um branch e muda sua HEAD pra ele, é a mesma coisa para adicionar modificações e dar commit

 # Cria um branch chamado branch_top

$ git branch branch_top

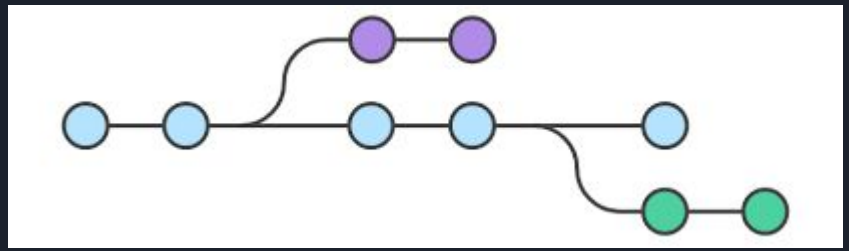
 # Altera seu HEAD para o branch_top

$ git checkout branch_top

 # Alterações e commits feitos neste ponto serão adicionados ao ramo branch_top, não ao ramo master

 # Lembre-se que é possível ver informações sobre os branches e commits com:

$ git log



Merges

- O merge serve para fundir os seus branches
- Normalmente, quando você terminou o que estava fazendo no branch, você da merge com o master (ou outro branch de sua escolha) e deleta o branch finalizado, como se cortando o galho da árvore que não deve crescer mais
- Podem existir problemas de conflito que você precisa resolver manualmente ou dizer ao git qual versão ele deve utilizar (leia mais no manual de sobrevivência)

 # Mudamos para o branch que desejamos manter, no caso o master

$ git checkout master

 # Mandamos ele se fundir com o branch_top

$ git merge branch_top

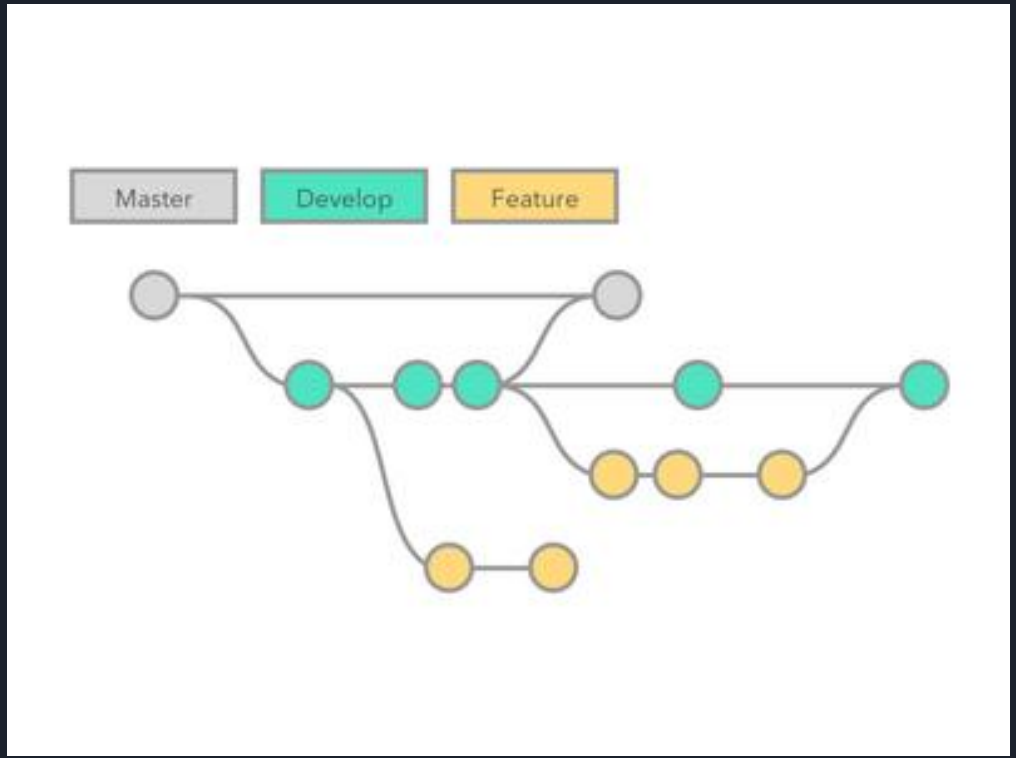
 # Se ocorreram conflitos, você precisa resolvê-los e tentar novamente ou dizer ao git qual vesão é para ele utilizar

 # Deleta o branch finalizado

$ git branch -d branch_top

Branches e Merges

- Cada bolinha representa um commit
- Existem 4 branches
- O branch develop é criado a partir do master, é mergeado ao master alguns commits depois, mas não é excluído
- Existem 2 branches feature. Não são o mesmo branch, só representam que features novas são feitas em branches separados do master ou do de desenvolvimento



Fonte: https://dribbble.com/shots/2564987-git-branches

Exercício 2

Lembrando de tudo:

- Criar um repositório para o exercício com: *$ git init*
- Criar branch com: *$ git branch <nome do branch>*
- Alterar o HEAD com: *$ git checkout <nome do branch>*
- Dar merge com: *$ git merge <nome do branch>*
- Deletar branchs com: *$ git branch -d <nome do branch>*
- Vejam detalhes sobre os commits com: *$ git log*
- Utilizem o manual

Repita o exercício 1, mas agora utilizando um branch para fazer a função “soma” e outro branch para fazer a função “printa”, em ambos façam merge com o master e deletem os branches destas “features” quando elas estiverem completas.

C O F F E E

Repositórios Remotos

- É um repositório onde você “publica” um repositório local. Normalmente pode ser acessado por várias pessoas e é utilizado para trabalhar em equipe. No git, ele é chamado de origin
- Existem vários sites que oferecem repositórios remotos, como Github, GitLab, BitBucket
- É possível vincular um repositório remoto à um repositório local e copiar versões de um para o outro
- Caso você esteja iniciando um repositório local a partir de um remoto, o ideal é cloná-lo

 # Assumindo que o repo remoto já tenha sido criado

 # Vinculamos um repo remoto

$ git remote add origin <url do repo>

 # Merge de um branch (normalmente o master) com o da a origin

$ git push origin <nome do branch>

 # Merge de um branch da origin com o branch local

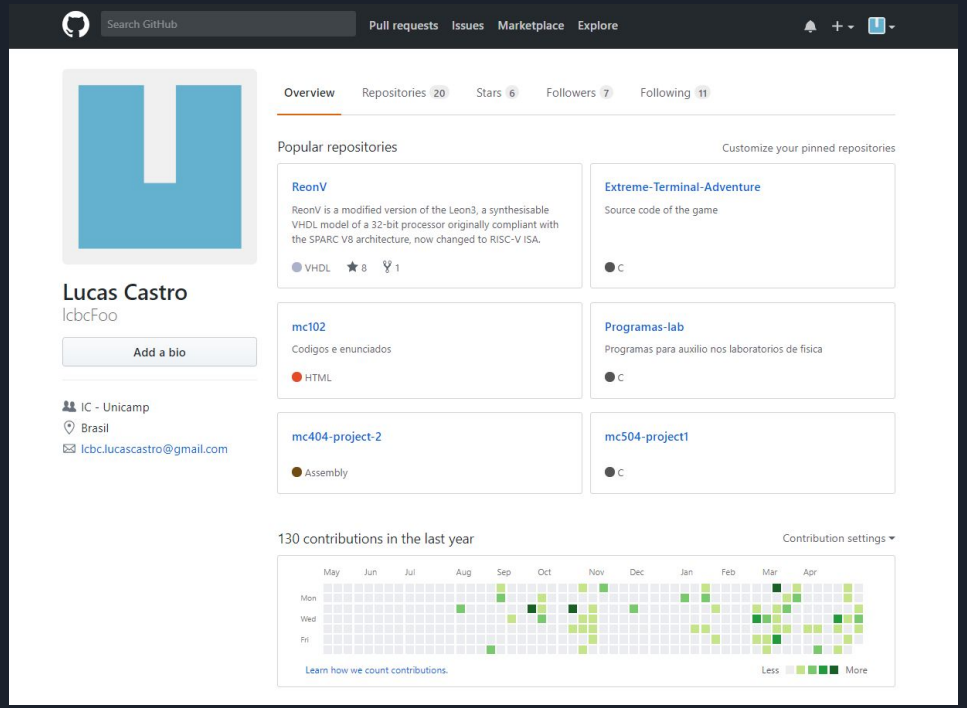
$ git pull origin <nome do branch>

 # Clona um repositório remoto (mas não o adiciona como origin!)

$ git clone <url do repo remoto>

Github

- Praticamente um facebook de repositórios
- Possui opção de repositórios fechados e abertos (fechados apenas na versão paga ou com conta da Unicamp)
- Interface bem amigável e vários guias para iniciantes
- Facilidade para desenvolvimento em grupo, reportar bugs, gerenciar branches de diferentes pessoas e várias outras funcionalidades



Exercício 3

Lembrando:

- Adicionamos a origin — *$ git remote add origin <url do repo>*
- Damos merge em um repositório da origin — *$ git push origin <nome do branch>*
- Damos merge em repo local a partir da origin — *$ git pull origin <nome do branch>*

Aquecimento:

- Criem uma conta no Github (olhem no manual como criar uma conta com o email da Unicamp)
- Criem um repositório remoto
- Adicionem este repositório como origin no repositório do exercício 2 e deem push do branch master (*$ git push origin master)*
- Verifiquem no Github o estado do repositório remoto

Exercício:

Criem um novo repositório local e clonem o repositório remoto do aquecimento (e também o adicionem como origin). Adicione uma função “subtração” que subtrai dois inteiros (pode utilizar branches se quiser treinar). Quando o trabalho no repositório local estiver concluído, deem *push* na origin e vejam se o repositório no Github recebeu a mudança.

Exercício 4 (Bônus)

Escolha uma dupla. Um dos membros da dupla deve criar um repositório no Github e adicionar o outro como contribuidor. Refaçam o exercício 2, agora cada membro fazendo uma das funções do código e colocando no repositório remoto. No final, vocês devem ter um programa com as 2 funções do exercício 2, tanto no repositório remoto como em ambos repositórios locais.

Aproveitem para testar as funcionalidades do Github (tentem dar uns pull requests, adicionarem issues se der tempo), criar branches e darem merges e coisas do tipo.

Obrigado! Espero que tenham gostado!!