



Universidade Federal Fluminense



Git: mantendo versões

e sua sanidade mental

THADEU PENNA

thadeupenna@id.uff.br

1. Motivação

1.1 Lição 1:

Debugar é ruim.

Testar é legal.

1.2 Análise de um chute

2. Git Yourself:

Git do Eu Sozinho

2.1 Diff e Patch

2.2 VCS

2.3 Git na Prática

2.4 Modificando o Código e Salvando as Versões

2.5 Resumo

2.6 Branches

3. GitHub

Eu e você, a dois, a três, escancarando de vez

3.1 Criando Forks

Motivação

ACABOU DE APRENDER UMA LINGUAGEM DE PROGRAMAÇÃO



FASES DO DESENVOLVIMENTO



Será que consigo ?



Vou fazer



Como eu faço ?



Saquei como fazer



Terminei



Não funcionou



Vai dar certo ?



Erro de novo



#@\$#+!!



Não vou conseguir



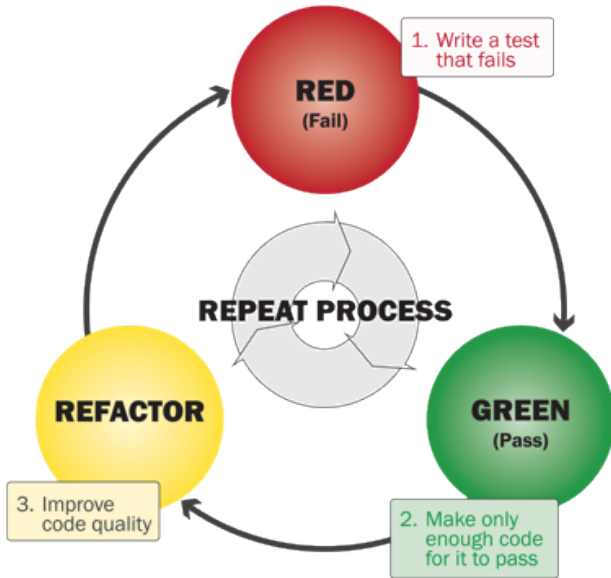
Será que vou ser aceito ?



CAÇA AOS BUGS

Lição 1:
Debugar é ruim.
Testar é legal.

TDD: TEST DRIVEN DEVELOPMENT



TDD: MUDAR O CÓDIGO SEM MEDO DE CAUSAR DESASTRES



We tend to forget
that baby steps still
move you forward.

@PeacefulMindPeacefulLife

Análise de um chute

FOLHA SECA DO ZICO

EXEMPLO: SIMULAR A BOLA EM UM CHUTE

1. Do que depende ?
velocidade inicial, ângulo(s), vento, gravidade, rotação (efeito)
2. Como atualizo ?
função para mover a bola. Testo se em $t \neq 0$, $x \neq 0$.
3. Move na horizontal ?
Veja se $x = vt$
4. Move na vertical ?
Veja se $y \neq 0$
5. Como descubro o alcance ?
Veja se $y = 0$ se $\theta = 90^\circ$. 45° é máximo ?
6. adiciono 3D. Testa $x - y$, $x - z$, etc.
7. adiciono o vento. Testa se aumenta o alcance
8. adiciono a rotação da bola. Testo a movimentação.

Git Yourself: Git do Eu Sozinho

CONTROLE DE VERSÕES

Mais usado: Nenhum controle. Você pode facilmente destruir um programa que funcionava antes.



Diff e Patch

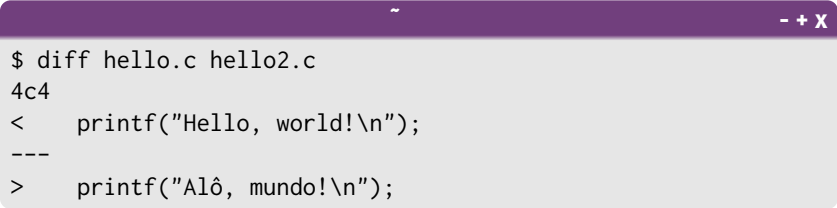
DIFF E PATCH: OS PRIMÓRDIOS

Escreva um programa que imprima `''Hello, world!''` na tela.

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello, world!\n");
5 }
```

Salve o arquivo como `hello.c`

Altere o programa para escrever `"Alô, mundo"` e salve como `hello2.c` (pode fazer o mesmo em qualquer linguagem). Rode




```
$ diff hello.c hello2.c
4c4
<     printf("Hello, world!\n");
---
>     printf("Alô, mundo!\n");
```

```
~ - + X
$ diff -uNr hello.c hello2.c
--- hello.c      2017-10-23 18:33:35.945704069 -0200
+++ hello2.c     2017-10-23 18:33:59.881753109 -0200
@@ -1,5 +1,5 @@
#include <stdio.h>

int main() {
-   printf("Hello, world!\n");
+   printf("Alô, mundo!\n");
}
```

diff funciona entre diretórios também.

- ➔ -u: cria cabeçalho e linhas próximas
- ➔ -N: arquivos inexistentes como vazios
- ➔ -r: recursivo, i.e., procura dentro de diretórios.
- ➔ diff --help para mais opções

A terminal window with a dark purple title bar containing a tilde (~) and window controls (-, +, X). The terminal has a light gray background and displays the following commands and their outputs:

```
$ cat hello.c
$ diff -uNr hello.c hello2.c > hello.patch
$ patch -p1 hello.c < hello.patch
$ cat hello.c
```

➔ -b: faz backup do original

É possível manter a história de todas as versões usando patch e diff. É possível mas não é divertido.

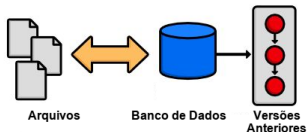
Imagine se pudesse ter apenas o arquivo da versão mais atual na sua vista, mas pudesse acessar as anteriores quando quisesse. Para isso, existe o git (mas não só para isso).

VCS

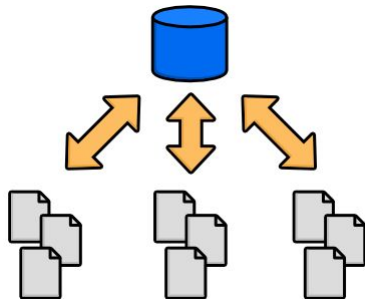
VCS: version control system. Gerenciar as mudanças que acontecem nos seus programas. É fundamental para grandes projetos, com colaborações, mas ajuda mesmo em pequenos códigos. Exemplos: CVS, SVN, Mercurial, BitKeeper e Git.

- ➔ Criado por Linus Torvalds, em 2005.
- ➔ Manutenção do kernel Linux
- ➔ Hoje: 19.786.000 linhas de código
- ➔ Mais de 2000 contribuidores ativos (Linus contribui com menos de 0,3% do código). 15 mil no total.
- ➔ BitKeeper fechou o código
- ➔ Pode ser usado sem conexão de rede.

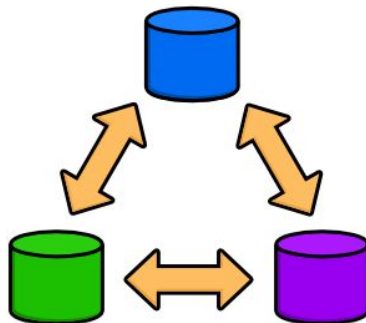
MODELOS



Local



Centralizado: conflitos /
merge



Distribuído

Git na Prática

INICIALIZANDO O GIT

Desenvolvido em 2005, pelo Linus Torvalds. O kernel Linux tem cerca de 20 milhões de linhas de código, mantido por 15 mil colaboradores.

TERMINAL

- + X

```
$ mkdir Hello
$ cd Hello
$ git init
  Initialized empty Git repository in ...
$ ls -a
.  ..  .git
$ git status

$ git config --global user.name "Thadeu Penna"
$ git config --global user.email thadeupenna@id.uff.br
$ git config --global core.editor vim
```

Use vim, nano, gedit, etc.

Os comandos `git config` configuram o git globalmente, isto é, valem para outros repositórios a serem criados.

Modificando o Código e Salvando as Versões

ADICIONANDO ARQUIVOS

Crie um arquivo `hello.c` ou copie o arquivo anterior para este diretório.

TERMINAL

- + X

```
$ git status
```

```
$ git add hello.c
```

```
$ git status
```

```
No ramo master
```

Submissão inicial.

Mudanças a serem submetidas:

(utilize `"git rm --cached <arquivo>..."` para não apresentar)

```
new file:   hello.c
```

```
$ git log
```

```
fatal: your current branch 'master' does not have any  
commits yet
```

OS ESTÁGIOS DO GIT

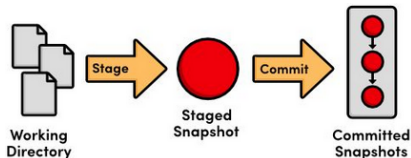


O PRIMEIRO COMMIT

TERMINAL

- + X

```
$ git status
$ git commit -m "Criação do arquivo"
[master (root-commit) 9df8c85] Criação do arquivo
1 file changed, 5 insertions(+)
create mode 100644 hello.c
$ git status
$ git log
$ git log --oneline
```



O SEGUNDO COMMIT

Altere o arquivo `hello.c` para escrever em português. Salve o arquivo.

TERMINAL

- + X

```
$ git status
Changes not staged for commit:
modified:   hello.c
$ git add hello.c
$ git status
(use "git reset HEAD <file>..." to unstage)
modified:   hello.c
$ git commit -m "Traduzido para o português"
[master 27be570] Traduzido para o Português
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log --oneline
27be570 Traduzido para o Português
9df8c85 Criação do arquivo
```

GIT CHECKOUT

Temos dois commits até agora (os números serão diferentes em cada máquina). O arquivo `hello.c` contém a versão em português. Verifique as saídas dos comandos

TERMINAL

- + X

```
$ git log --oneline
27be570 Traduzido para o Português
9df8c85 Criação do arquivo
$ git checkout 9df8c85
HEAD is now at 9df8c85... Criação do arquivo

$ cat hello.c
$ git log --oneline
```

O que aconteceu com o `hello.c` ?

Observem o aviso. O git é bem pedagógico. O HEAD está agora no primeiro commit. HEAD é a posição atual do sistema modificado e o master é o do último commit.

Verifique as saídas dos comandos

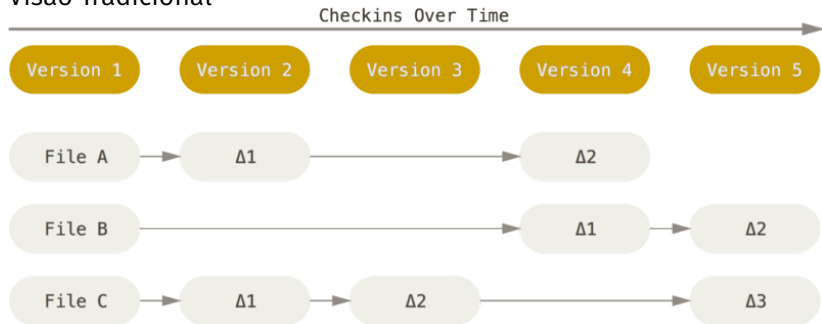
TERMINAL

- + X

```
$ git status
HEAD detached at 9df8c85
nada a submeter, diretório de trabalho vazio
$ git checkout master
Previous HEAD position was 9df8c85... Criação do arquivo
Switched to branch 'master'
$ cat hello.c
```

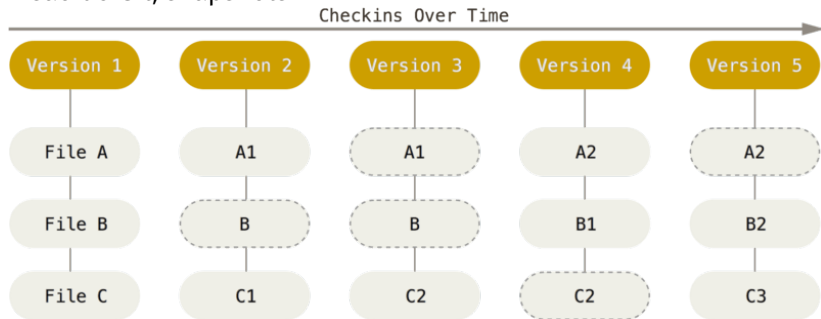

ENTENDENDO O GIT

Visão Tradicional



ENTENDENDO O GIT

Visão do Git, snapshots



GIT DIFF - O QUE MUDOU ?

Suponha que agora você queira ver as diferenças entre dois checkouts.

TERMINAL

- + X

```
$ git log --oneline
27be570 Traduzido para o Português
9df8c85 Criação do arquivo
$ git status
No ramo master
$ git diff 9df8c85
--- a/hello.c
+++ b/hello.c
@@ -1,5 +1,5 @@
#include <stdio.h>

int main() {
-   printf("Hello, world!\n");
+   printf("Alô, mundo!\n");
```

REVERTENDO COMMITS I

➔ Se você esqueceu de adicionar um arquivo no último commit

```
~ - + X  
$ git add outroarquivo.dat  
$ git commit --amend
```

➔ Você acabou de modificar o arquivo, mas se arrependeu.

```
~ - + X  
$ git status  
(utilize "git add <arquivo>..." para atualizar  
o que será submetido)  
(utilize "git checkout -- <arquivo>..." para descartar  
mudanças no diretório de trabalho)  
  
modified:   hello.c  
$ git checkout -- hello.c
```

REVERTENDO COMMITS II

➔ Você se arrependeu do último commit

```
~ - + X  
$ git log --oneline  
f950da4 Vírgula  
...  
$ git revert f950da4
```

➔ Adicionou um arquivo para o staged e se arrependeu

```
~ - + X  
$ git add dados.dat  
$ git status  
Mudanças a serem submetidas:  
  (use "git reset HEAD <file>..." to unstage)  
  
  new file:   dados.dat  
  
$ git reset HEAD dados.dat
```

ETIQUETANDO OS COMMITS

```
$ git tag -a v1.0 -m "Versão estável"  
$ git status  
$ git checkout -a v1.0
```

- git config --global --list Lista configurações globais
- git status -s Status resumido(silent)
- git add . Adiciona todo o diretório
- git commit -a -m "ok" Adiciona e faz o commit
- git rm dados.dat Remove dados.dat do git e do diretório
- git clean -f Limpa arquivos modificados
- git clean -n Mostra o que vai ser apagado
- git log -p Mostra as mudanças
- git tag -l -n1 Versões e mensagens
- .gitignore Arquivo com especificações para ignorar arquivos (*.o, *.aux, por exemplo). Um tipo por linha.
- git help O mais útil deles
- <http://git-scm.com> Documentação sobre o Git.

Resumo

1. Cria diretório
2. `git init`
3. `git add`
4. `git commit -m ""`
5. `git status`
6. `git log`

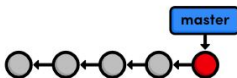
Branches

Branch é um ramo independente de desenvolvimento. Imagine que duas pessoas tenham que mexer no código, mas criando funções independentes. Cada uma cria um branch e depois os mesmos são ligados ao tronco principal, o `master`. Ou mesmo, você quer tentar duas saídas diferentes para um mesmo problema mas não está certo de que escolha será a melhor. Se não der certo, voltamos ao programa original.

Aqui é que reside o grande poder de organização do git. A página inicial do curso tem um relâmpago com branches, para ressaltar a importância deste procedimento.

CRIANDO UM BRANCH

Imagine que você tenha criado um arquivo e “commitado” algumas modificações.



O nosso programa já imprime “Alô, mundo!”. Vamos criar um branch para imprimir os números pares, mas você não quer perder o que já funciona tão bem.

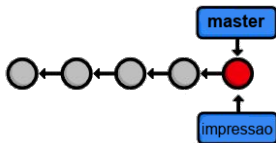
TERMINAL

- + X

```
$ git branch
* master
$ git branch pares
$ git branch
  pares
* master
$ git checkout pares
Switched to branch 'pares'
```

MODIFICANDO UM BRANCH

No momento estamos assim:



```
1 #include <stdio.h>
2
3 void main() {
4     int i;
5
6     printf("Oi Mundo!\n");
7     for (i=0;i<=10;i+=2) printf("%d\n",i);
8 }
```

MODIFICANDO NO BRANCH

```
~ - + X
$ git commit -m "Imprime pares"
$ git log --oneline
9488aec Imprime pares
27be570 Traduzido para o Português
....
$ git branch
* pares
master
$ git checkout master
Switched to branch 'master'
$ git log --oneline
27be570 Traduzido para o Português
....
```

Observe que, no master, não há referência ao que aconteceu no branch pares.

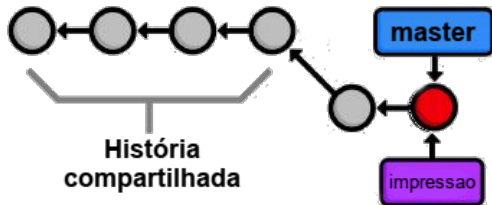
Se o programa funcionar, é hora de adicionar a modificação ao master.

TERMINAL

- + X

```
$ git branch
pares
* master
$ git merge pares
Updating d054ccd..9488aec
Fast-forward
hello.c | 5 ++++-
1 file changed, 4 insertions(+), 1 deletion(-)
$ git status
No ramo master
nada a submeter, diretório de trabalho vazio
$ git log --oneline --decorate --all --graph
```

FAST-FORWARD MERGE



TERMINAL

- + X

```
$ git branch -d impressao
Deleted branch impressao (was 79628c5).
$ git branch
* master
```


ENTENDENDO O MERGE

Use o `git revert` para voltar a versão sem impressão dos pares.
Crie dois branches (par e impar), um para mostrar os pares e outro para mostrar os ímpares.

O que daria o merge de um e depois o outro ?

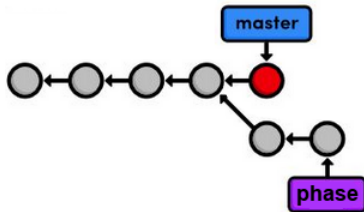
```
1 #include <stdio.h>
2
3 void main() {
4     int i;
5
6     printf("Oi Mundo!\n");
7     for (i=0;i<=10;i+=2) printf("%d\n",i);
8 }
```

ENTENDENDO O MERGE

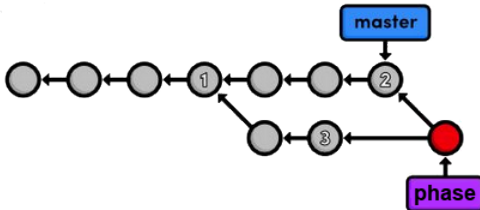
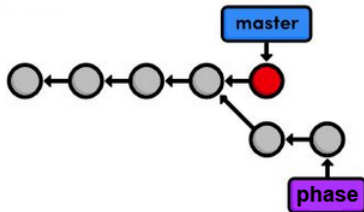
Use o `git revert` para voltar a versão sem impressão dos pares. Crie dois branches (par e impar), um para mostrar os pares e outro para mostrar os ímpares, através de uma função. O que daria o merge de um e depois o outro ?

```
1 #include <stdio.h>
2
3 void imprime_pares(int n) {
4     int i;
5
6     for(i=0; i<=n; i+=2) printf("%d\n", i);
7 }
8
9 void main() {
10    printf("Alô Mundo!\n");
11    imprime_pares(10);
12 }
```

3-WAY MERGE



3-WAY MERGE



TERMINAL

- + X

```
$ git merge impar  
Mesclagem automática de sistemas.c  
CONFLITO (conteúdo): conflito de mesclagem em hello.c  
Automatic merge failed; fix conflicts and then commit  
the result.
```

Abra o arquivo e encontre o local de conflito.

Sim, mensagens de erro devem ser lidas cuidadosamente.

Edite-o, adicione-o e faça o novo commit.

COMANDOS ÚTEIS PARA ARQUIVAR

`git archive master --format=zip -o ../backup.zip`

`git archive master -o ../backup.tar.gz` Copia todos os arquivos do ramo master. Não copiará a informação do git.

`git bundle create repo.bundle master` Cria um arquivo `repo.bundle`, incluindo as informações do `.git`.

`git clone repo.bundle outrolugar -b master` Recria toda a árvore do git, salva em `repo.bundle`, no diretório `outrolugar`.

GitHub
Eu e você, a dois, a três,
escancarando de vez

O GitHub <https://github.com> é o repositório mais usado para manter projetos open-source. Outro site interessante é o Bitbucket <https://bitbucket.org/>. Você pode hospedar seus projetos lá, sem custos, desde que os projetos sejam abertos. O BitBucket permite hospedagem de projetos privados gratuitamente também.

É possível rodar o git em um diretório compartilhado, tipo Google Drive ou Dropbox, podendo acessá-lo a partir de máquinas diferentes. O GitHub vai além disso, facilitando colaborações.



Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside millions of other developers.

Username



Email




Password



Use at least one letter, one numeral, and seven characters.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.

 [thadeupenna](#) / [massamola](#)

Unwatch ▾1

★ Star0

🍴 Fork0

<> Code

🕒 Issues0

🔗 Pull requests0

📁 Projects0

📖 Wiki

📶 Pulse

📊 Graphs

⚙️ Settings

Física Computacional. Problemas do tipo massa-mola.

Edit

[Add topics](#)

🕒 1 commit

🌿 1 branch

📦 0 releases

👤 1 contributor

📄 GPL-3.0

Branch: master ▾





New pull request


Create new file

Upload files

Find file

Clone or download ▾

 thadeupenna Initial commit		Latest commit d42c348 12 minutes ago
 .gitignore	Initial commit	12 minutes ago
 LICENSE	Initial commit	12 minutes ago
 README.md	Initial commit	12 minutes ago

 **README.md**

massamola

Física Computacional. Problemas do tipo massa-mola.

<https://guides.github.com/features/mastering-markdown>

Seções

```
# This is an <h1> tag
## This is an <h2> tag
##### This is an <h6> tag
```

Fontes

It's very easy to make some words **bold** and other words *italic* with Markdown. You can even [link to Google!](http://google.com)

Imagens

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

URL é do tipo <https://github.com/thadeupenna/massamola.git>
Muito Fácil

TERMINAL

- + X

```
$ git remote add origin URL
$ git clone URL

$ git pull origin master
warning: no common commits
remote: Counting objects: 5, done.

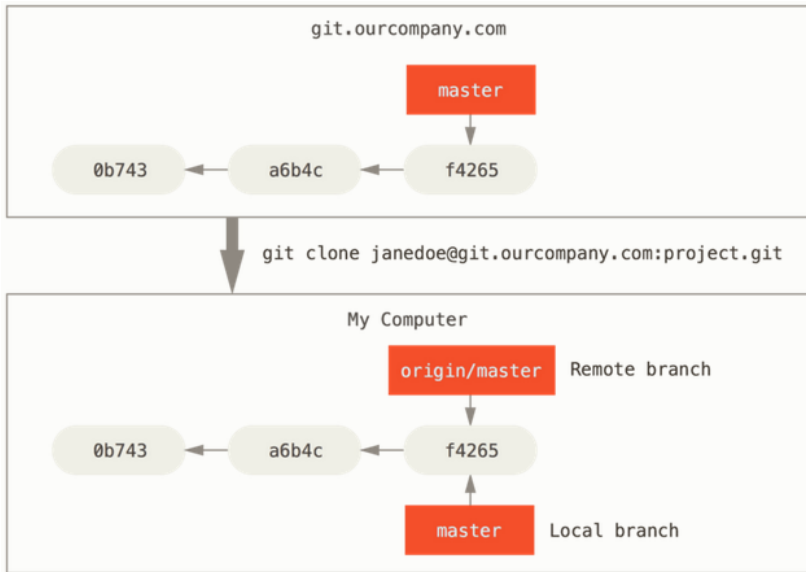
$ git push -u origin master
Counting objects: 36, done.
```

pull pega do github

push joga no github

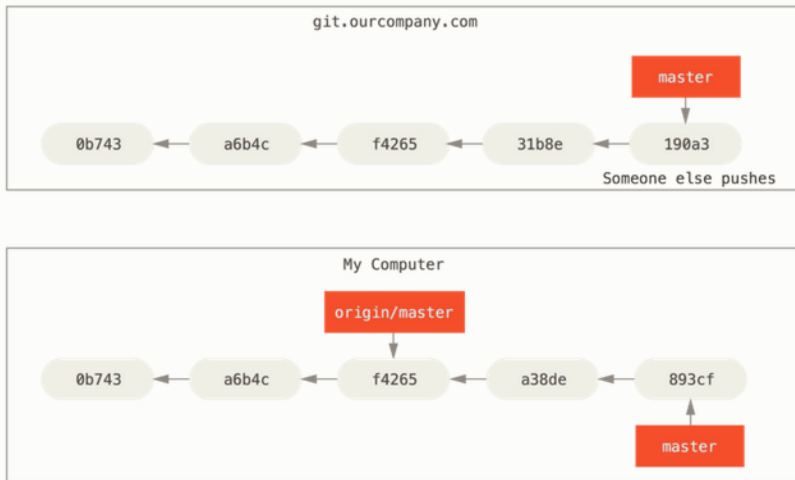
GIT REMOTE

Git Clone

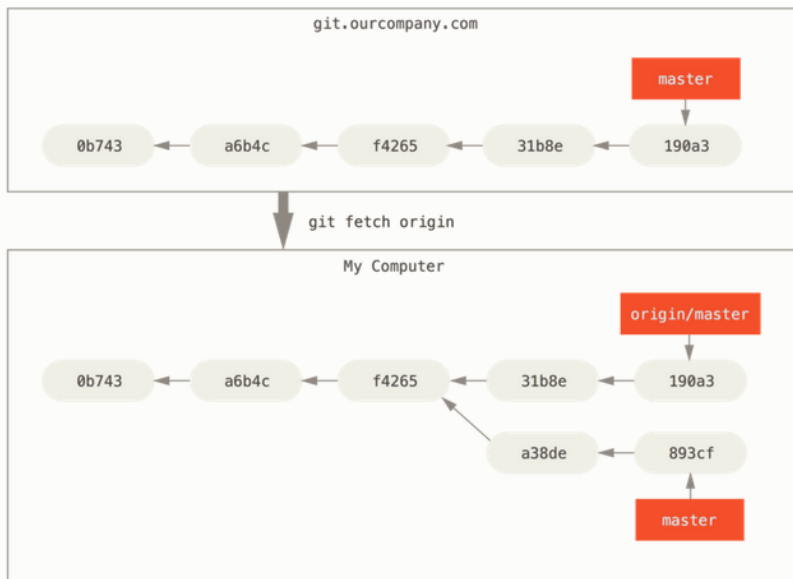


GIT REMOTE

Modificações em ambos repositórios



GIT REMOTE



CONFLITOS REMOTOS

Modifique o README.me no Github e no local, diferentemente. Tente sincronizar os dois. Por exemplo, escreva sandbox no GitHub e caixa de areia no local.

TERMINAL

- + X

```
$ git pull origin master
remote: Counting objects: 3, done.
Updating d884d04..66aeb1b
error: Your local changes to the following files would be
overwritten by merge:
README.md
Please, commit your changes or stash them before you can merge.
```

Como resolver isso ?

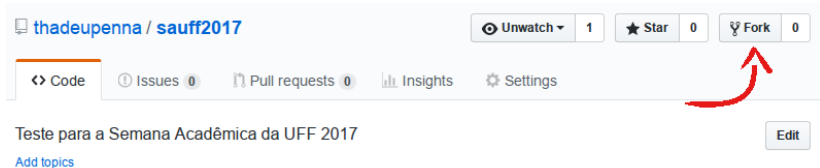
ADICIONANDO COLABORADORES

- ➔ Clique em **Settings**.
- ➔ **Collaborators**. Adicione o colaborador pelo login do github ou pelo email. Colaboradores podem quase tudo, menos apagar o repositório.

Criando Forks

CRIANDO UM FORK

Seus programas estão públicos no GitHub. Qualquer um pode propor alterações, cabe a você aceitá-las, ou não.



- ➔ Crie um fork do sauff2017
- ➔ Clone o seu repositório
- ➔ modifique o arquivo hello.c para imprimir seu nome
- ➔ Commite e envie para o GitHub.
- ➔ Faça um Pull Request e preencha os dados.
- ➔ Para continuar atualizando, adicione o repositório original com outro nome que não o origin (upstream, por exemplo).

`git log --oneline --decorate --all --graph` colore a saída do git log.

`git tag -d v1.0` Apaga a tag.

`git push --tags` Faz o upload das tags

`git remote show origin` Informações sobre o repositório remoto.

`git fetch URL` Pega os arquivos que ainda não foram atualizados mas não realiza um merge com o que você tem. Você deve fazer o merge manualmente.

`git config --global credential.helper 'cache --timeout=3600'`
Não pergunta senha e usuário por uma hora .