# Git e GitHub

github
SOCIAL CODING

@hsilvest

# O que é Git?

"Git é um sistema de controle de versão distribuida, rápido e escalável"

*Tradução do Manual*

Basicamente é um versionador de arquivos, é utilizado principalmente para gerenciar versões de softwares desenvolvidos por um ou mais desenvolvedores, com ele podemos implementar novas funcionalidades e tudo é registrado em um histórico, o qual podemos retroceder sempre que necessário, os integrantes de um projeto podem enviar correções, atualizacões, etc. As alterações enviadas para o projeto principal não  comprometem o mesmo pois cabe ao dono do projeto a inclusão ou não das alterações efetuadas.

# Um Pouco de História

O git foi desenvolvido inicialmente por Linus Torvalds (Criador do Linux), pela necessidade de ter um software capaz de controlar a versão do kernel do linux.

Mais informacões no link do projeto Git abaixo:

http://git.or.cz/

# Termos

Repository: Local onde fica todos os arquivos do projeto, inclusive os históricos e versões.

Commit: Colecão de alteracões realizadas, é como se fosse um “checkpoint” do seu projeto, sempre que necessário vc pode retroceder ate algum commit.

# Termos

Branch: É uma ramificacão do seu projeto, cada branch representa uma versão do seu projeto, e podemos seguir uma linha de desenvolvimento a partir de cada branch.

---

Fork: Basicamente é uma bifurcação, uma cópia de um projeto existente para seguir em uma nova direção.

Merge: É a capacidade de incorporar alterações do git, onde acontece uma junção dos branchs.

# Começando

Primeiro vamos criar um novo diretório e em seguida iniciar nosso repository git:

---

```
mkdir NovoProjeto
cd NovoProjeto
```

```
git init
```

*Todos os arquivos dentro do diretório NovoProjeto faz parte do nosso repository*

# Configurando

Agora vamos indicar as informações do desenvolvedor, fazemos isto para identificar os desenvolvedores em cada commit específico:

---

git config user.name “Henrique Silvestre”
git config usar.email “hs.ccti@gmail.com”

*Nesse ponto já podemos ver que uma pasta “.git” foi criada, lá ficam os registros de todo o nosso projeto.*

# Exemplificando

Vamos criar um arquivo vazio para exemplificarmos:

```
touch exemplo.txt
```

Agora informamos ao git que vamos adicionar o arquivo que acabamos de criar ao proximo commit:

```
git add exemplo.txt
```

Mas se quisessemos que todos os arquivos fossem adicionados podiamos ter usado o seguinte comando:

```
git add .
```

# Exemplificando

O git é inteligente o suficiente para apenas realizar commits se detectar alguma alteração nos arquivos que foram indicados pelo (git add), dessa forma o único commit que ele realiza sem alguma alteração é o primeiro.

# Nosso Primeiro Commit...

git commit -a -m “Nosso primeiro commit”

---

Passamos o parâmetro -a para informar que todos os arquivos alterados devem ser incluidos do commit.

Passamos o paramêtro -m para adicionar uma mensagem explicativa para o commit.

# Comandos Úteis

Depois que fizemos nosso primeiro commit existem alguns comandos que podemos utilizar para verificar o mesmo, são eles:

---

git log // listar todos os commits já realizados.

git status // mostrar os arquivos alterados desde o último commit.

git show // mostrar as alterações realizadas no último commit.

# Branchs

Todo novo repositorio git tem um branch default chamado “master”.

---

Para listar todos nossos branchs:

git branch

Se vc não tiver mais branchs a saida será:

*master

# Branchs

Vamos criar um novo branch chamado “working” pegando como base o branch corrente e vamos continuar no branch corrente:

---

git branch working

Agora vamos criar um novo branch chamado “outro” como base o branch “working” e vamos nos manter no branch corrente:

git branch outro working

# Branchs

Como já vimos, nosso repositório ja tem por default o branch “master” dessa forma quando falamos “branch corrente” nos referimos ao branch que estamos trabalhando no momento. Se não criamos nenhum então estaremos trabalhando no master.

# Branchs

Para criar um novo branch e automaticamente sairmos do branch corrente e irmos para o novo branch usamos o “checkout”:

---

git checkout -b “outro”

Por hora devemos ter os seguintes branchs:

master
*outro

*o branch sinalizado com “*” é o branch corrente.*

# Branchs

Mudando de branch corrente:

git checkout master

---

Agora a saida fica assim:

*master
outro

Para deletarmos um branch fazemos dessa maneira:

branch -d outro

# Branchs

Não podemos deletar um branch corrente, primeiro temos que mudar para outro branch e depois deletarmos o que desejamos.

# Exemplificando Merge

Digamos que o arquivo exemplo.txt foi alterado no branch “outro”.

---

git checkout outro // alterando para o branch outro.

git add . // indicamos todos os arquivos do commit.

git commit -a -m “alteração exemplo.txt” // commit.

# Exemplificando Merge

Agora queremos que nossa alteração no exemplo.txt seja refletida a nossa branch master.

---

Primeiro vamos para o branch que queremos as alterações refletidas:

git checkout master

Agora realizamos o merge (mesclagem/fusão) das alterações do branch outro para o current branch:

git merge outro

# Exemplificando Merge

Não precisamos passar o branch master como parâmetro pois já estamos nele através do:

git checkout master

---

Lembram?

A saida deverá ser algo como isto:

Updating 642caa9..851gb82

Fast forward

exemplo.txt | 1 +

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

# Exemplificando Merge

Confirmando a alteração no arquivo:

cat exemplo.txt

---

Nesse momento toda a alteração realizada no branch “outro” sera refletida no branch master inclusive nossos commits que fizemos no branch outro e agora esta no master.

# Exemplificando Merge

Todo branch possui um branch pai, que é a base a partir de então, sabendo disso e se precisassemos fazer um merge em um branch master que já tivesse alguma alteração depois que o branch outro foi criado? Dessa forma o branch outro não estava refletindo a ultima versão de master, como proceder então?

# Exemplificando Merge

Nesse caso a melhor forma seria o “rebase” ou seja pegar o ultimo commit do branch master, e trazer para o branch outro e aplicar todos os seus commits nele, feito isso agora nosso branch outro está refletindo a ultima versão do master

Poderiamos fazer o merge também, mas talvez poderia causar conflitos e a última coisa que queremos são conflitos.

# Exemplificando Merge

Então vamos fazer o rebase da branch outro para refletir a ultima versão da branch master:

---

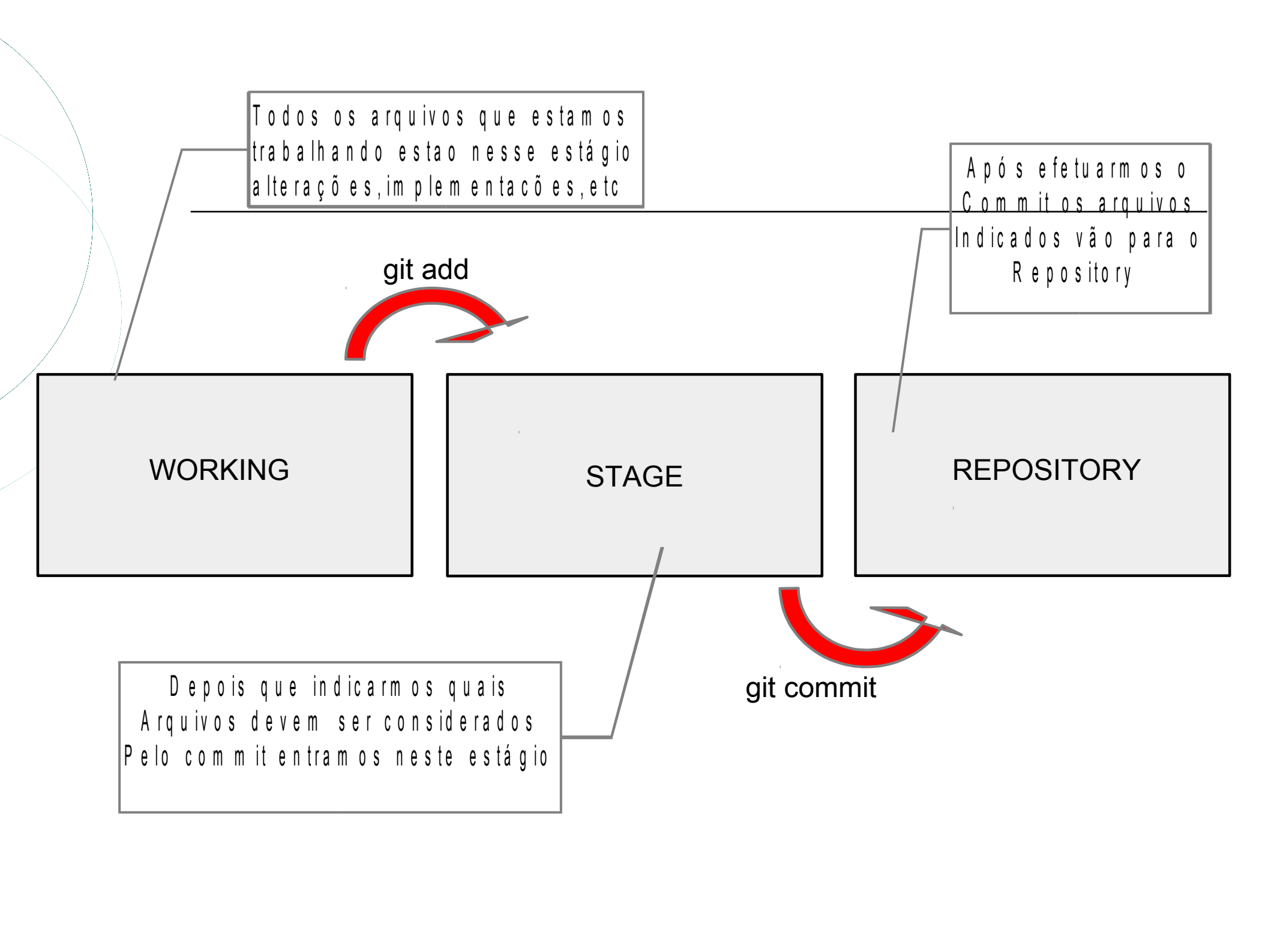
git checkout outro

git rebase master

Agora podemos realizar o merge:

git checkout master

git merge outro


Todos os arquivos que estamos trabalhando estao nesse estágio alteraçõ es, implementacõ es, etc
Após efetuarmos o Commit os arquivos Indicados vão para o Repository
git add
WORKING
STAGE
REPOSITORY
Depois que indicarmos quais Arquivos devem ser considerados Pelo commit entramos neste estágio
git commit

# Repositórios Remotos

A idéia é a mesma de um repositório local, mas os remotos como o proprio nome diz, não estão na sua máquina, estão hospedados em servidores na internet ou outra máquina qualquer.

Existem vários sites que servem como repositórios online, nesta apresentação usaremos o www.github.com pois é um dos melhores na minha opnião.

# Repositórios Remotos

A primeira coisa que precisamos é criar uma conta no github e em seguida configurar a chave ssh para que consigamos nos comunicar com o github.

Para auxilio na configuracao da ssh keys  ou outra configuracao do github siga o passo-a-passo do próprio site indicado abaixo:

http://help.github.com/set-your-user-name-email-and-github-tok

# Clonando um Projeto

Para criar um clone (cópia de todo o projeto, incluindo todos commits) devemos utilizar o seguinte comando:

git clone https://github.com/MeuUsuario/MeuProjeto

# Clonando um Projeto

Logo após poderemos ver que foi criado um diretório com o nome do projeto e dentro dele a cópia do projeto

Veja o repositório remotos disponível agora:

---

```
cd MeuProjeto
git branch –r

origin/HEAD
origin/master
origin/working
```

# Clonando um Projeto

Devemos criar um branch local baseado em algum branch remoto antes de começar a efetuarmos nossas alterações.

git checkout –b outro origin/working

# Clonando um Projeto

Agora vamos supor que nós efetuamos várias alterações neste repositório e durante esse tempo o projeto principal também foi alterado não correspondendo mais a base que nós possuimos agora, e então desejamos sincronizar com a última versão disponível do projeto.

# Clonando um Projeto

Primeiramente recuperamos a versão recente do projeto:

---

git fetch

Agora efetuamos o merge com o branch atual:

git merge origin/master

# Conclusão

Esta apresentação visa esclarecer os conceitos básicos do git e github, mas existem infinidades de conceitos que podem ser pesquisados mais profundamente, material na internet não falta, espero ter ajudado a clarear alguns conceitos! Qualquer correção, ou observações comentem.