GIT

Básico

ademir.mazer.jr@gmail.com - @nunomazer

Referência

Livro oficial: GIT Book

http://git-scm.com/book/pt-br

Como funciona?

Git considera que os dados são como um conjunto de **snapshots** (captura de algo em um determinado instante, como em uma foto)

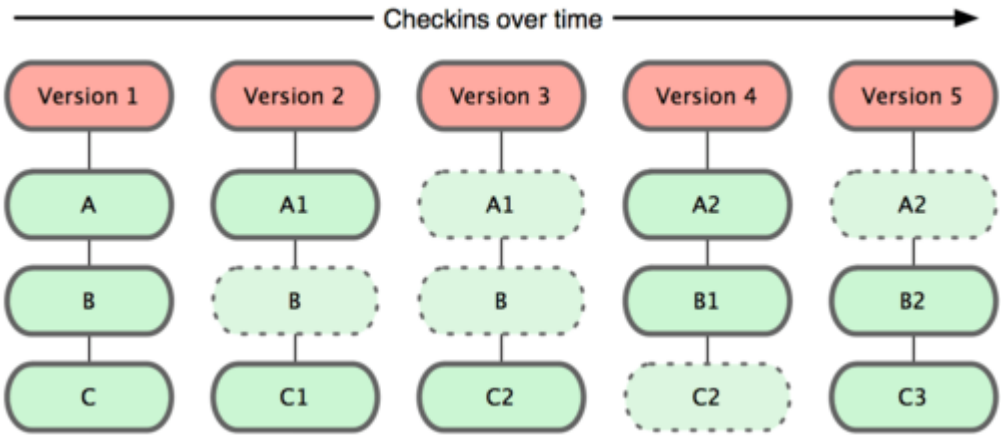
Como funciona?

Cada vez que você salva ou consolida (**commit**) o estado do seu projeto no Git, é como se ele tirasse uma foto de todos os seus arquivos naquele momento e armazenasse uma **referência** para essa captura.

Como funciona?

Para ser eficiente, se nenhum arquivo foi alterado, a informação não é armazenada novamente - apenas um link para o arquivo idêntico anterior que já foi armazenado.

Como funciona?



Características

- **Quase Todas Operações São Locais**
- **Git Tem Integridade** checksum
- **Git Geralmente Só Adiciona Dados**

Três estados

- **consolidado (committed):** quando estão seguramente armazenados em sua base de dados local
- **modificado (modified):** quando houve uma alteração que ainda não está consolidada
- **preparado (staged):** quando um arquivo modificado é marcado para fazer parte de um commit

Workflow

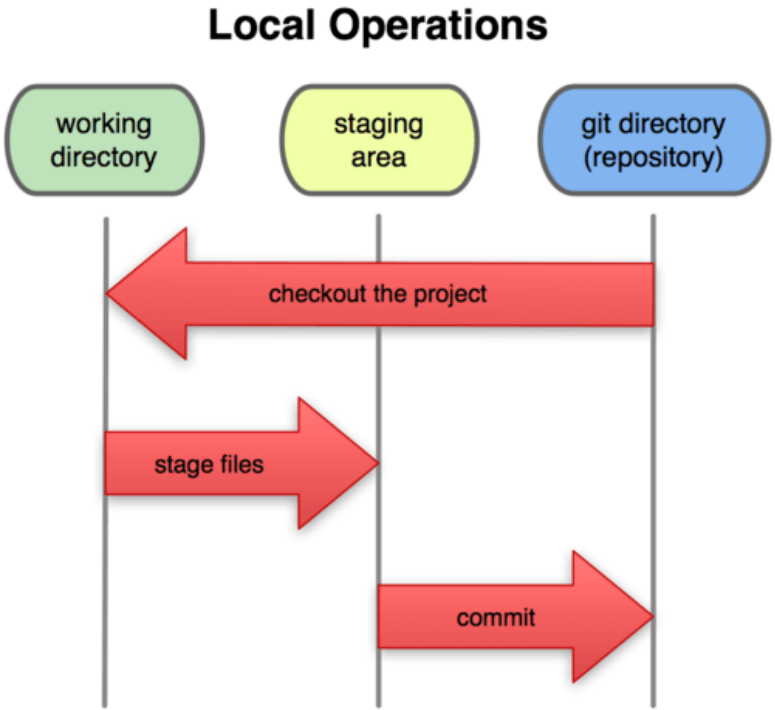
Fluxo básico pode ser descrito com os passos:

1. Você **modifica** arquivos no seu diretório de trabalho.

2. Você seleciona os arquivos, adicionando **snapshots** deles para sua área de preparação.

3. Você faz um **commit**, que leva os arquivos como eles estão na sua área de preparação e os armazena permanentemente no seu diretório Git.

Workflow

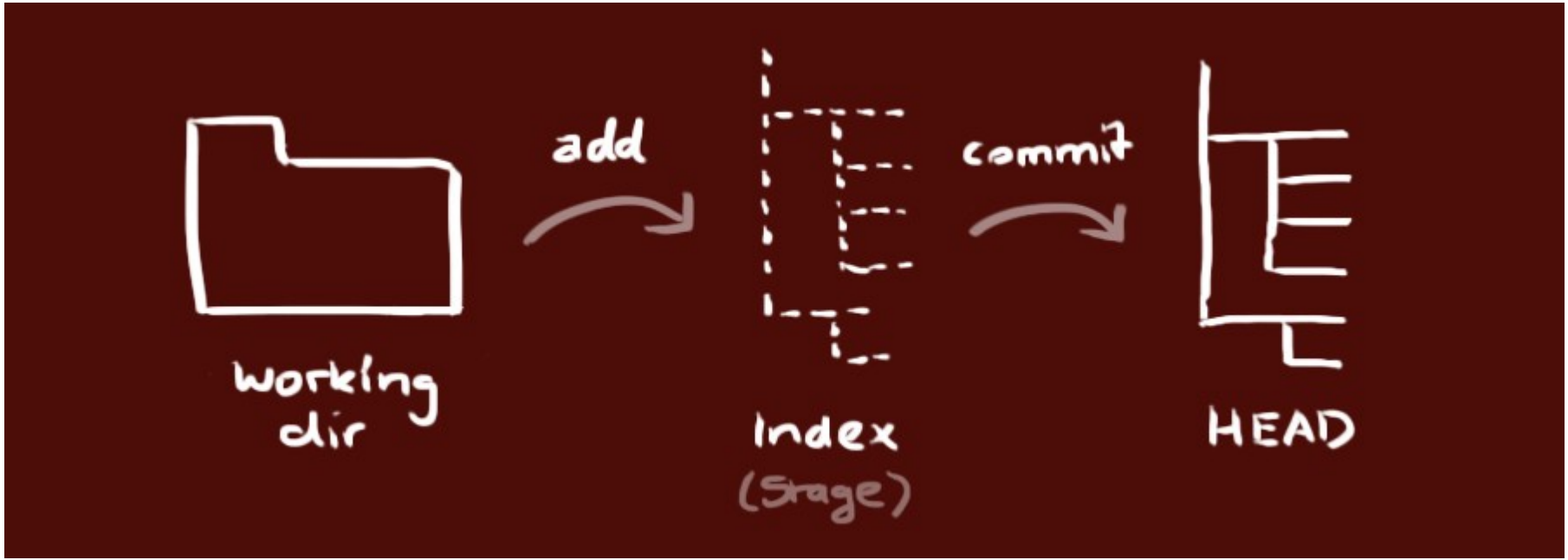

Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit

Workflow

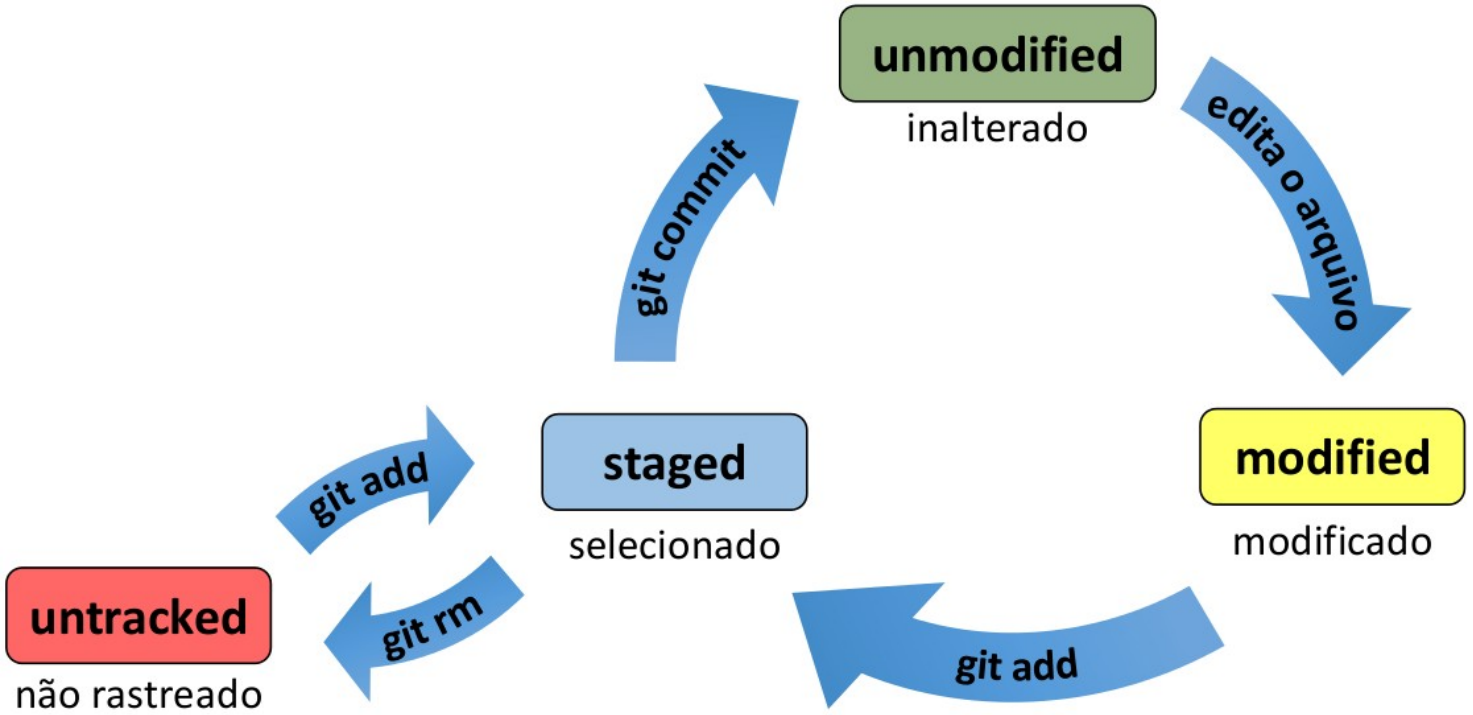
seus repositórios locais consistem em três "árvores" mantidas pelo git.

- **Working Directory** que contém os arquivos vigentes.
- **Index** que funciona como uma área temporária (**stage**)
- **HEAD** que aponta para o **último *commit*** (**confirmação**) que você fez.

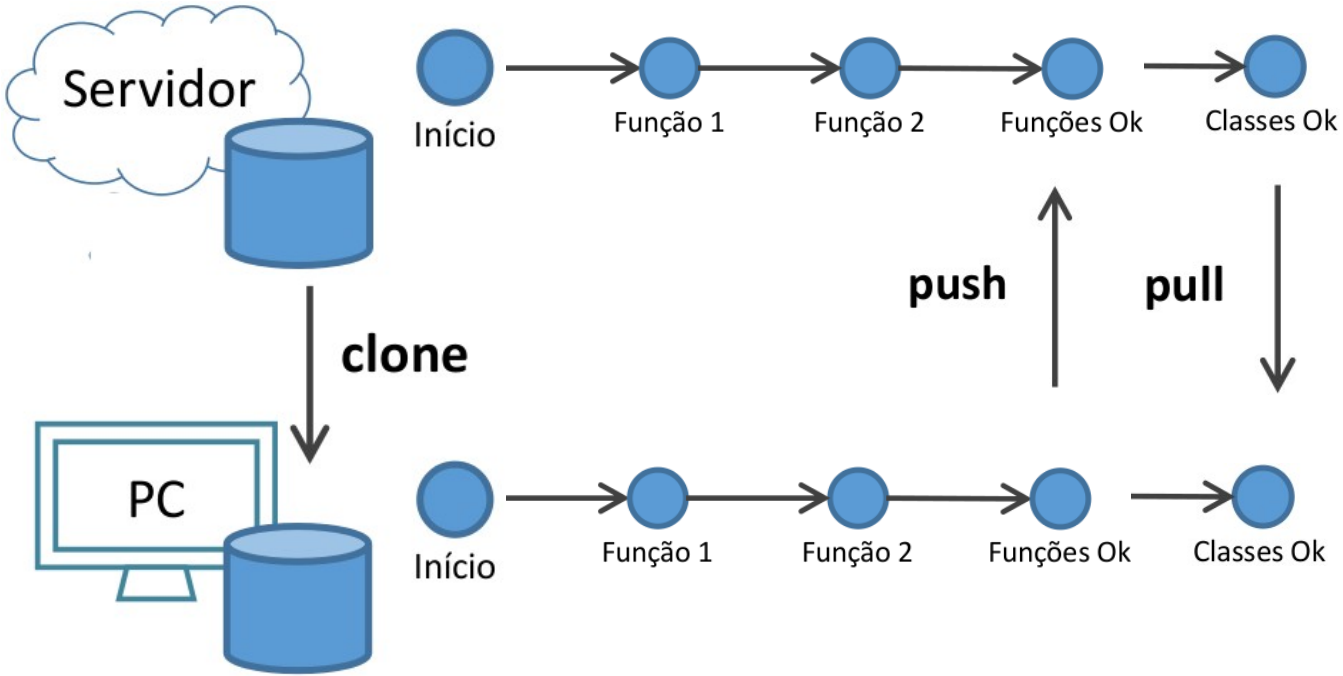
Workflow



Workflow


unmodified
inalterado
edita o arquivo
modified
modificado
git add
git add
staged
selecionado
git commit
git rm
untracked
não rastreado

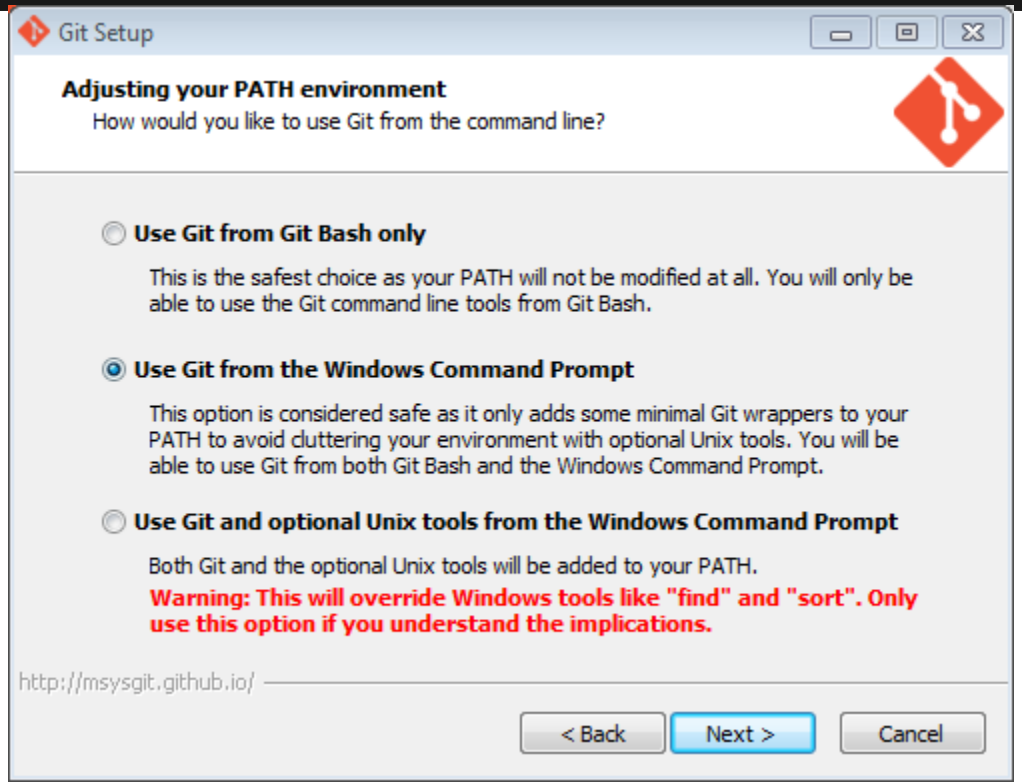
Workflow


Servidor
Início
Função 1
Função 2
Funções Ok
Classes Ok
push
pull
clone
PC
Início
Função 1
Função 2
Funções Ok
Classes Ok

Instalando - Windows

http://msysgit.github.io/

Instalando - Windows



Windows tip

Dica:

as **aspas simples** dos comandos nos slides devem ser *trocadas* por **aspas duplas** no Windows.

Configuração

arquivo /etc/gitconfig: Contém valores para todos usuários do sistema e todos os seus repositórios. Se você passar a opção **--system** para git config, ele lerá e escreverá a partir deste arquivo especificamente.

Configuração

arquivo ~/.gitconfig: É específico para seu usuário. Você pode fazer o Git ler e escrever a partir deste arquivo especificamente passando a opção **--global.**

Configuração

arquivo de configuração no diretório git (ou seja, **.git/config**) de qualquer repositório que você está utilizando no momento: Específico para aquele único repositório. Cada nível sobrepõem o valor do nível anterior, sendo assim valores em **.git/config** sobrepõem aqueles em /etc/gitconfig.

Configuração

Em sistemas Windows, Git procura pelo arquivo **.gitconfig** no diretório $HOME (**C:\Documents and Settins\$USER** para a maioria das pessoas)

Configuração

Definir seu nome e email:

```
git config --global user.name "John Doe"

git config --global user.email johndoe@example.com
```

```
git config --global color.ui true
```

Configuração

Listar o arquivo de configuração:

```
git config --list

git config user.name
```

Ajuda

```
git
git help
git help <verb>

git help config
```

Criando um repositório

Para criar um novo repositório:

- crie uma nova pasta (pode ser criado um novo repositório em uma pasta com arquivos existentes)
- acesse a nova pasta pela linha de comando
- digite o comando de criação de repositório

```
git init
```

Status do repositório

Para verificar o **status** do repositório em qualquer momento

```
git status
```

Obtendo um repositório

crie uma cópia de trabalho em um repositório local executando o comando:

```
git clone /caminho/para/o/repositório
```

quando usar um servidor remoto, seu comando será

```
git clone usuário@servidor:/caminho/para/o/repositório
```

Adicionar e confirmar

Você pode propor mudanças (adicioná-las ao **Index**) usando

```
git add <arquivo>
git add *
```

Este é o primeiro passo no fluxo de trabalho básico do git.

Adicionar e confirmar

Para realmente confirmar estas mudanças (isto é, fazer um ***commit***), use

```
git commit -m “comentários das alterações”
```

Agora o arquivo foi enviado para o **HEAD**, mas ainda não para o repositório remoto.

Ignorando arquivos

Para ignorar arquivos, crie padrões dentro de um arquivo **.gitignore** no raiz do repositório. Exemplo:

```
# Compiled source #
*.com
*.class
*.dll
*.exe
*.o
*.so
```

https://gist.github.com/octocat/9257657

Ignorando arquivos

Arquivos que já estão sendo rastreados não são afetados pelas regras do **.gitignore**

Enviando alterações

Suas alterações agora estão no **HEAD** da sua cópia de trabalho local. Para enviar estas alterações ao seu repositório remoto, execute

```
git push origin master
```

Altere *master* para qualquer ramo (***branch***) desejado, enviando suas alterações para ele.

Enviando alterações

Se você não clonou um repositório existente e quer conectar seu repositório a um servidor remoto, você deve adicioná-lo com

```
git remote add origin <servidor>
```

Agora você é capaz de enviar suas alterações para o servidor remoto selecionado.

Log do repositório

Se você deseja verificar o **log** de alterações de um repositório (tag, branch)

```
git log
```

Atualizar e mesclar

para atualizar seu repositório local com a mais nova versão do repositório remoto, execute o comando abaixo na sua pasta de trabalho para *obter* e *fazer* ***merge*** (mesclar) alterações remotas.

```
git pull
```

Atualizar e mesclar

para fazer merge de um outro branch ao seu branch ativo (ex. master), use

```
git merge <branch>
```

em ambos os casos o git tenta fazer o merge das alterações automaticamente.

Conflitos

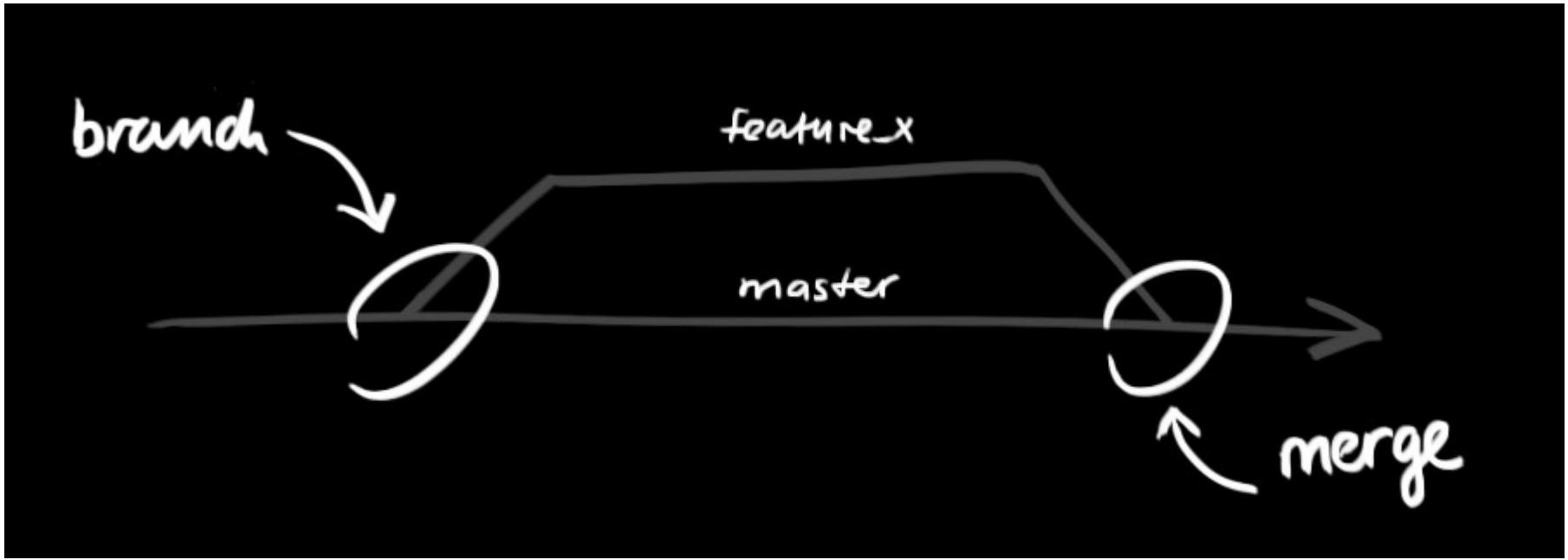
Quando um merge resulta em conflitos, você é responsável por fazer o merge destes ***conflitos*** manualmente editando os arquivos exibidos pelo git. Depois de alterar, você precisa marcá-los como merged com o comando abaixo antes de fazer o merge das alterações.

```
git add <arquivo>
```

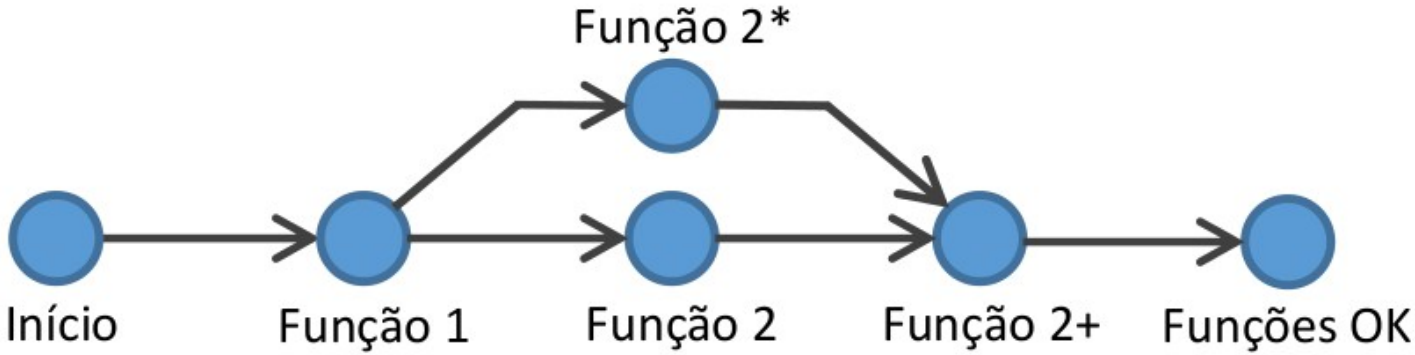
Ramificar

Branches ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch ***master*** é o branch "padrão" quando você cria um repositório. Use outros branches para desenvolver e mescle-os (*merge*) ao branch master após a conclusão.

Ramificar



Ramificar


Função 2*
Início
Função 1
Função 2
Função 2+
Funções OK


commit
branch
merge

Ramificar

crie um novo branch chamado "funcionalidade_x" e selecione-o usando

```
git checkout -b funcionalidade_x
```

agora os **commits** que fizer serão confirmados neste *branch*

Ramificar

retorne para o master usando

```
git checkout master
```

e remova o branch da seguinte forma

```
git branch -d funcionalidade_x
```

Ramificar

um branch *não está disponível a outros* a menos que você envie o branch para seu repositório remoto

```
git push origin <funcionalidade_x>
```

Resumo comandos básicos

- git add <arquivo ou pasta, ou . para add tudo>
- git commit -m “Explicação sobre o commit”
- git push {remote} {branch} {tag}
- git status
- git log {arquivo | pasta | tag | commit}
- git branch {nome}
- git checkout {branch | tag | commit | arquivo}
- git merge {branch}

Resumo comandos básicos

- git diff {arquivo | pasta | commit}
- git tag 1.0 (gera tag)
- git commit (commit na ultima tag)
- git push <remote> <branch> <tag>
- git log <tag>

Pratica 1

Crie um diretório

```
mkdir cursogit

cd cursogit
```

Pratica 1

Crie um repositório local

```
git init
```

Pratica 1

Crie um arquivo .gitignore

```
touch .gitignore (linux)


copy con .gitignore (windows)
F6 <ENTER> (para sair do editor)
```

Pratica 1

Adicione o arquivo .gitignore na *stage area* do repositório

```
git add .gitignore (somente o arquivo)

git add * (todos os arquivos)
```

Pratica 1

Verifique o status do repositório

```
git status
```

Pratica 1

Confirme as alterações (**commit**)

```
git commit -m ‘commit inicial’
```

Pratica 1

Verifique o status do repositório

```
git status
```

Pratica 1

Crie um arquivo texto chamado README.txt e insira algum texto nele com erro de grafia

Este é um texto de exempo.

Este arqv foi criado por mim.

Pratica 1

Verifique o status do repositório

```
git status
```

Pratica 1

Adicione o arquivo à *stage area* e confirme as alterações

```
git add *
git commit -m ‘Adicionado README do projeto’
```

Pratica 1

Altere o conteúdo do arquivo corrigindo os erros de grafia, após verifique o status

```
git status
```

Pratica 1

Tente realizar o commit

```
git commit -m ‘Erros de grafia corrigidos’
```

Pratica 1

Nâo funcionou pois necessita ser adicionado ao stage sempre que uma mudança ocorre

```
git add caminho/nome-do-arquivo
```

Pratica 1

Verifique o histórico de alterações

```
git log
```

Pratica 2

Confirmando alterações seletivas. Pequenos pontos de **estabilidade** devem ser confirmados.

Pratica 2

Crie mais 3 arquivos, simulando um programa fonte, com os nomes a, b e c de extensão .php

```
a.php
b.php
c.php
```

Pratica 2

Se um grupo de alterações realizada afetam a e c, faça um commit apenas com eles

```
git add a.php b.php
```

Pratica 2

Verifique o status

```
git status
```

Pratica 2

Faça o commit, inserindo informações de um código de tarefa

```
git commit -m ‘Correção da validação de campo código, task #333’
```

Pratica 2

Faça o commit do restante de arquivos modificados

```
git add *
git commit -m ‘Alteração da cor de fundo para azul’
```

Pratica 2

Verifique o log, com um formato de uma linha

```
git log --pretty=oneline
```

Pratica 3

Vamos voltar no tempo, mude o HEAD para o commit que tem o comentário sobre os erros de grafia corrigidos

```
git log --oneline
git checkout <hash - o suficiente para ser único>
```

Pratica 3

Liste os arquivos do diretório, você não deve ver os fontes a, b e c.php.
Verifique o status e o log

```
git status

git log
```

Pratica 3

Volte para o **branch master** e liste os arquivos do diretório, verifique o status e o log

```
git checkout master
ls ou dir *.*
git status
git log
```

Pratica 4

Crie uma tag para identificar uma configuração no HEAD atual (**master**)

```
git tag v1.0.0
```

Pratica 4

Altere o arquivo a.php com o conteúdo abaixo

```
<?php
    echo “teste”;
```

Pratica 4

Faça o commit

```
git commit -a -m ‘Imprimindo teste’
```

Pratica 4

Altere novamente o arquivo a.php, adicione o código abaixo e crie o arquivo d.php

```
echo “Curso de GIT”;
```

Pratica 4

Faça o commit

```
git commit -a -m ‘Imprimindo sobre o curso
```

Pratica 4

Crie a tag v2.0.0 e verifique o log das duas formas mostradas

```
git tag v2.0.0
git log
git log --no-walk --tags --pretty="%h %d %s"
```

Pratica 4

Crie **alias** (apelido) para comandos muito longos

```
git config --global alias.log-tags 'log --no-walk --tags --pretty="%h %d %s"'

git log-tags
```

Pratica 4

Verifique somente o log da tag v1.0.0

```
git log v1.0.0
```

Pratica 4

Você pode voltar à trabalhar na v1.0.0

```
git checkout v1.0.0
```

Pratica 4

Altere o arquivo b.php e faça o commit, depois volte ao master

```
git commit -a -m ‘Melhorado b.php’

git checkout master
```

Pratica 4

Por estar trabalhando em uma versão **detached** o commit ficou perdido, rode reflog para verificar

```
git log --oneline

git reflog
```

Pratica 5

À partir da v2.0.0 vamos criar um branch para trabalho isolado, evitando trabalhar em **detached**

```
git checkout -b v2-bugfix
git status
```

agora temos o branch **v2-bugfix** além de **master**

Pratica 5

Altere o arquivo c.php e faça o commit

```
git commit -a -m ‘Corrigido c, task #444’
```

Pratica 5

Adicione o arquivo e.php e altere novamente c.php, após faça o commit

```
git add e.php
git commit -a -m ‘Melhorado cadastro, #445’
```

Pratica 5

Volte ao branch master e adicione um arquivo f.php, faça commit e após altere-o, faça commit novamente

```
git checkout master
git add f.php
git commit -m ‘Adicionada função f.php’
```

Pratica 5

Navegue nos branchs com checkout

```
git checkout master
git log
git checkout v2-bugfix
git log
```

Pratica 5

Verifique as árvores

o primeiro exemplo apresenta hash dos commits, o segundo a data

```
git log --graph --pretty=oneline


git log --pretty=format:'%h %ad | %s%d [%an]' --
graph --date=short --all
```

Pratica 5

Lembre-se, crie alias para comandos longos

```
git config --global alias.hist "log --
pretty=format:'%h %ad | %s%d [%an]' --graph
--date=short --all"

git hist
```

Pratica 5

Mescle (**merge**) as alterações dos branchs

```
git checkout master
git merge v2-bugfix
git hist
```

???

ademir.mazer.jr@gmail.com @nunomazer

obrigado !!!

ademir.mazer.jr@gmail.com **@nunomazer**