



git

Pedro Castilho
1 de junho de 2016

Introdução

O que é Git?

- Sistema de versionamento

O que é Git?

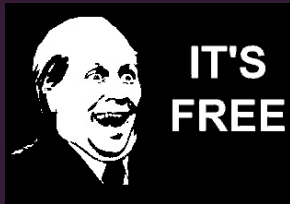
- Sistema de versionamento
- Distribuído

O que é Git?

- Sistema de versionamento
- Distribuído
- Livre e de código aberto

O que é Git?

- Sistema de versionamento
- Distribuído
- Livre e de código aberto

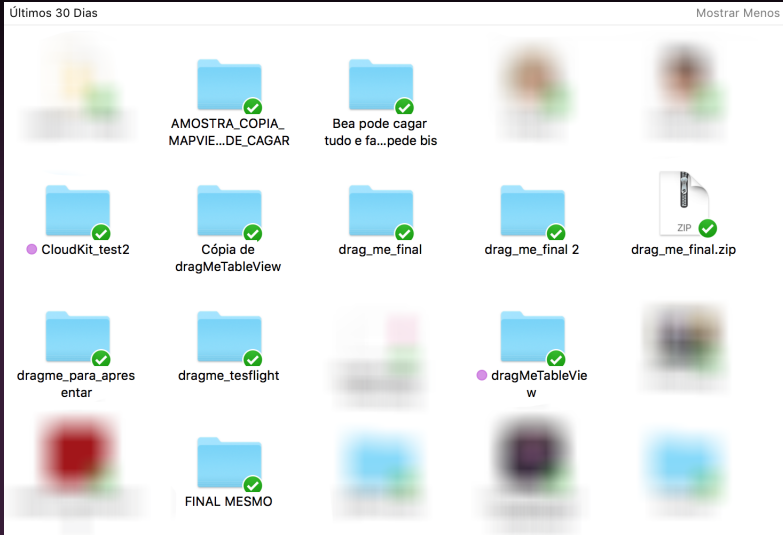


Por que usar Git?

Por que usar Git?

O que acontece quando você *não* usa git:

Por que usar Git?



Por que usar Git?

O que acontece quando você *usa* git:

Por que usar Git?



Como usar Git?

Como usar Git?

```
1. ~/src/blog.metaintellect.com (zsh)
blog.metaintellect.com git:master > gs
# On branch master
# Changes to be committed:
#   modified:   .themes/classic/sass/base/_theme.scss
#
# Changes not staged for commit:
#   modified:   _posts/2012-10-12-make-powershell-and-git-suck-less-on-windows.markdown
#
blog.metaintellect.com git:master > git commit -am 'The italics ar now in different color'
[master 9086eb0] The italics ar now in different color
 2 files changed, 246 insertions(+), 2 deletions(-)
blog.metaintellect.com git:master > gp
Counting objects: 17, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 4.38 KiB, done.
Total 9 (delta 5), reused 0 (delta 0)
To git@github.com:xajler/blog.metaintellect.com.git
 1a5ad5e..9086eb0 master -> master
blog.metaintellect.com git:master > |
```

Como usar Git?

- git foi pensado como programa de linha de comando

Como usar Git?

- git foi pensado como programa de linha de comando
- só na linha de comando todas as opções estão disponíveis

Como usar Git?

- git foi pensado como programa de linha de comando
- só na linha de comando todas as opções estão disponíveis
- **existem** interfaces gráficas

Como usar Git?

- git foi pensado como programa de linha de comando
- só na linha de comando todas as opções estão disponíveis
- existem interfaces gráficas
- (vou citar algumas no final)

Um aviso

**DON'T
PANIC**

Baixando o Git

Baixando o Git

Se você tem o Xcode, você tem git!

Baixando o Git

- Abra o terminal (Command + Espaço, escreva "terminal", Enter)
- Digite `git --version`

Atualizando o Git

Pergunte-me como!

(Dica: é usando o Homebrew)

git help

- O git contém ajuda para todos os comandos.

git help

- O git contém ajuda para todos os comandos.
- Digite `git help` para ver a lista.

git help

- O git contém ajuda para todos os comandos.
- Digite `git help` para ver a lista.
- (Lembre-se, não entre em pânico)

git help

- O git contém ajuda para todos os comandos.
- Digite `git help` para ver a lista.
- (Lembre-se, não entre em pânico)
- Ajuda de comandos específicos:

git help

- O git contém ajuda para todos os comandos.
- Digite `git help` para ver a lista.
- (Lembre-se, não entre em pânico)
- Ajuda de comandos específicos:
- Digite `git help config`

git help

- O git contém ajuda para todos os comandos.
- Digite `git help` para ver a lista.
- (Lembre-se, não entre em pânico)
- Ajuda de comandos específicos:
- Digite `git help config`
- Digite `q` para sair da ajuda.

Configurando o Git

git config

- Vamos começar configurando seu usuário no git.

git config

- Vamos começar configurando seu usuário no git.
- Digite o seguinte no terminal:

git config

- Vamos começar configurando seu usuário no git.
- Digite o seguinte no terminal:
- `git config user.name "Seu nome"`

git config

- Vamos começar configurando seu usuário no git.
- Digite o seguinte no terminal:
- `git config user.name "Seu nome"`
- `git config user.email seu_email@provedor.com`

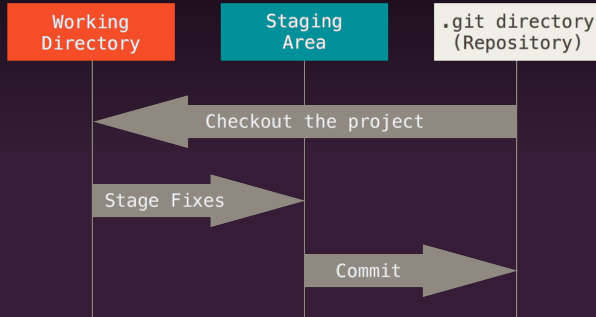
git config

- Vamos começar configurando seu usuário no git.
- Digite o seguinte no terminal:
- `git config user.name "Seu nome"`
- `git config user.email seu_email@provedor.com`
- Agora digite `git config --list`

Dúvidas?

Trabalhando com repositório local

Working directory, Staging area, e Repositório



git init

- Crie um novo projeto do Xcode (ou use a pasta de um projeto velho)

git init

- Crie um novo projeto do Xcode (ou use a pasta de um projeto velho)
- No terminal, vá para a pasta do projeto

git init

- Crie um novo projeto do Xcode (ou use a pasta de um projeto velho)
- No terminal, vá para a pasta do projeto
- Na pasta do projeto, digite `git init`

git init

- Crie um novo projeto do Xcode (ou use a pasta de um projeto velho)
- No terminal, vá para a pasta do projeto
- Na pasta do projeto, digite `git init`
- Se tudo der certo, agora você tem um repositório nessa pasta!

git init

git init: cria um novo repositório

git status

- Na mesma pasta, digite `git status`

git status

- Na mesma pasta, digite `git status`
- Deve surgir uma lista com vários “Untracked files”

git status

- Na mesma pasta, digite `git status`
- Deve surgir uma lista com vários “Untracked files”
- Esses arquivos estão no “Working Directory”

git status

- Na mesma pasta, digite `git status`
- Deve surgir uma lista com vários “Untracked files”
- Esses arquivos estão no “Working Directory”
- Vamos colocá-los na “Staging Area”.

git status

git status: mostra o estado atual do
“Working Directory” e da “Staging Area”.

git status

- Ainda na mesma pasta, escolha um arquivo da lista.

git status

- Ainda na mesma pasta, escolha um arquivo da lista.
- Digite `git add nome_do_arquivo`

git status

- Ainda na mesma pasta, escolha um arquivo da lista.
- Digite `git add nome_do_arquivo`
- Digite `git status` novamente.

git status

- Ainda na mesma pasta, escolha um arquivo da lista.
- Digite `git add nome_do_arquivo`
- Digite `git status` novamente.
- O arquivo está na “Staging Area”.

git status

- Queremos que todos os arquivos da pasta sejam versionados pelo git.

git status

- Queremos que todos os arquivos da pasta sejam versionados pelo git.
- Digite `git add -A` .

git status

- Queremos que todos os arquivos da pasta sejam versionados pelo git.
- Digite `git add -A` .
- Esse comando coloca todos os arquivos da pasta atual na “Staging Area”.

git status

git add: adiciona arquivos
à "Staging Area".

git reset

- Escolha um dos arquivos na “Staging Area”.

git reset

- Escolha um dos arquivos na “Staging Area”.
- Digite `git reset nome_do_arquivo`

git reset

- Escolha um dos arquivos na “Staging Area”.
- Digite `git reset nome_do_arquivo`
- Digite `git status` novamente.

git reset

- Escolha um dos arquivos na “Staging Area”.
- Digite `git reset nome_do_arquivo`
- Digite `git status` novamente.
- O arquivo saiu da “Staging Area”.

git reset

git reset: tira arquivos
da "Staging Area".

git commit

- Vamos mover os arquivos da “Staging Area” para o repositório.

git commit

- Vamos mover os arquivos da “Staging Area” para o repositório.
- Digite `git commit -m "Primeiro commit"`

git commit

- Vamos mover os arquivos da “Staging Area” para o repositório.
- Digite `git commit -m "Primeiro commit"`
- Digite `git status`

git commit

- Vamos mover os arquivos da “Staging Area” para o repositório.
- Digite `git commit -m "Primeiro commit"`
- Digite `git status`
- Sua working area está limpa - exceto o arquivo que removemos.

git commit

- Vamos adicionar o arquivo que falta ao repositório.

git commit

- Vamos adicionar o arquivo que falta ao repositório.
- Usem `git add` e `git commit`.

git commit

- Vamos adicionar o arquivo que falta ao repositório.
- Usem `git add` e `git commit`.
- Dica: Você sempre precisa adicionar uma mensagem em um commit.

git commit

git commit: adiciona os arquivos da
“Staging Area” ao repositório.

git log

- Podemos visualizar o histórico de commits usando `git log`.

git log

- Podemos visualizar o histórico de commits usando `git log`.
- Digite `git log` agora.

git log

```
Terminal
commit 51263d833fb070e212a381242d0582ea3394b7cd
Author: Pedro Castilho <castil.px@gmail.com>
Date: Mon May 30 09:13:09 2016 -0300

    Bringing forms back

commit 8f568b15f927fe9579d4d6c1f5a320cc5b09d4
Author: Pedro Castilho <castil.px@gmail.com>
Date: Mon May 30 03:18:36 2016 -0300

    Improved appearance of lists

commit c2572634a21a75e09489e7077cbfd22f6ac99f59
Author: Pedro Castilho <castil.px@gmail.com>
Date: Mon May 30 02:59:55 2016 -0300

    Fixed page reloading error

commit 387667ed12d7506dfcb45dfff2dcf1fc756c805da
Author: Pedro Castilho <castil.px@gmail.com>
Date: Mon May 30 01:49:19 2016 -0300

    Added build instructions to README

commit fba73e998830b84356a1e2cf33efd725aba33b9
Author: Pedro Castilho <castil.px@gmail.com>
Date: Mon May 30 01:19:53 2016 -0300

    Added navbar

commit 65ff1227499d0b13ee023e76390274c0045b693c
Author: Pedro Castilho <castil.px@gmail.com>
Date: Sun May 29 18:38:18 2016 -0300

    Added a lot of CSS, made UI responsive

commit 2bc72ae68d7fe507672d0b8cc6db76183df89c
Author: Pedro Castilho <castil.px@gmail.com>
Date: Sat May 28 20:18:21 2016 -0300

    Added finance component, improved build automation

commit 48eb4f8a07b03a9f36db34db31f1f137c803663
Author: Pedro Castilho <castil.px@gmail.com>
Date: Sat May 28 11:59:40 2016 -0300

    Automated build process and added package management

commit 9d3e208a0c5b495094677e411473bf15b3a3ce8
Author: Pedro Castilho <castil.px@gmail.com>
Date: Sat May 28 03:00:50 2016 -0300

    Reorganizing tests

commit e8b500bb2710d961ae79052a448a40021f3c1652
Author: Pedro Castilho <castil.px@gmail.com>
Date: Fri May 27 23:08:45 2016 -0300

    Added money type and tests
```


git log

- Podemos visualizar o histórico de commits usando `git log`.
- Digite `git log` agora.

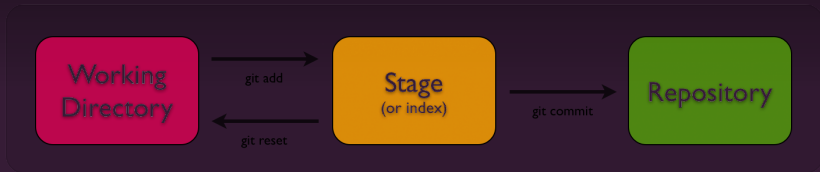
git log

- Podemos visualizar o histórico de commits usando `git log`.
- Digite `git log` agora.
- Digite `q` para sair da lista de commits.

git log

git log: mostra o histórico de commits.

O que vimos até agora



git checkout

- Modifique mais algum arquivo e dê commit na sua modificação.

git checkout

- Modifique mais algum arquivo e dê commit na sua modificação.
- Podemos recuperar qualquer versão antiga do arquivo.

git checkout

- Modifique mais algum arquivo e dê commit na sua modificação.
- Podemos recuperar qualquer versão antiga do arquivo.
- Use `git log` novamente e encontre o ID do commit anterior.

git checkout

- Modifique mais algum arquivo e dê commit na sua modificação.
- Podemos recuperar qualquer versão antiga do arquivo.
- Use `git log` novamente e encontre o ID do commit anterior.
- Digite `git checkout id_do_commit nome_do_arquivo`

git checkout

- Modifique mais algum arquivo e dê commit na sua modificação.
- Podemos recuperar qualquer versão antiga do arquivo.
- Use `git log` novamente e encontre o ID do commit anterior.
- Digite `git checkout id_do_commit nome_do_arquivo`
- Digite `git status`

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.
- Digite `git checkout id_do_commit`

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.
- Digite `git checkout id_do_commit`
- Digite `git status`

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.
- Digite `git checkout id_do_commit`
- Digite `git status`
- Nossa “Working Area” agora está no estado de um commit antigo.

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.
- Digite `git checkout id_do_commit`
- Digite `git status`
- Nossa “Working Area” agora está no estado de um commit antigo.
- Nós podemos salvar esse estado, mas por enquanto vamos voltar ao commit anterior.

git checkout

- `git checkout` também pode ser usado para retornar o projeto inteiro a um commit anterior.
- Digite `git checkout id_do_commit`
- Digite `git status`
- Nossa “Working Area” agora está no estado de um commit antigo.
- Nós podemos salvar esse estado, mas por enquanto vamos voltar ao commit anterior.
- Digite `git checkout master`

git checkout

git checkout: restaura versões antigas de arquivos ou do projeto inteiro.

HEAD: identificador especial que indica a versão do projeto que você está visualizando.

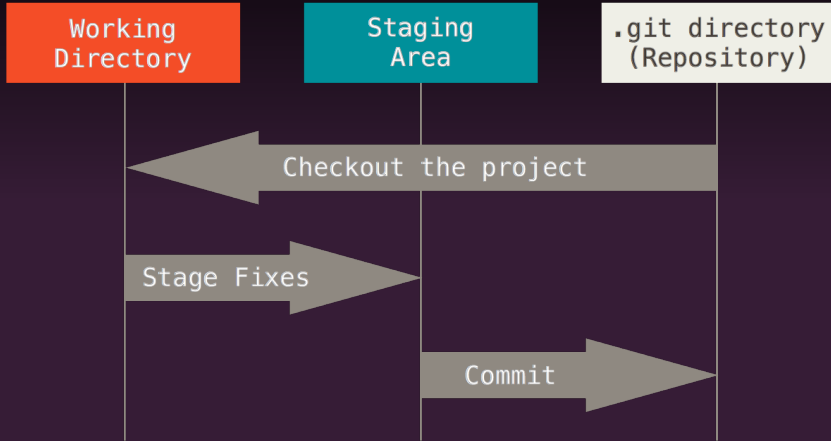
git checkout

master: nome padrão do git para a versão principal do projeto.

git checkout

Uma última dica: `git checkout -- nome_do_arquivo` restaura o arquivo à versão dele no HEAD.

O que vimos até agora



Quando tudo dá errado

- Fazer um commit toda vez que você faz alguma alteração relevante no projeto pode salvar sua vida.

Quando tudo dá errado

- Fazer um commit toda vez que você faz alguma alteração relevante no projeto pode salvar sua vida.
- Os seus commits são salvos **permanentemente**.

Quando tudo dá errado

- Fazer um commit toda vez que você faz alguma alteração relevante no projeto pode salvar sua vida.
- Os seus commits são salvos **permanentemente**.
- Caso no futuro algo dê errado, você pode simplesmente restaurar o projeto.

Dúvidas?

Trabalhando com branches

git branch

- Às vezes queremos fazer modificações sem quebrar o que já temos pronto.

git branch

- Às vezes queremos fazer modificações sem quebrar o que já temos pronto.
- O git te permite criar um “ramo” de trabalho isolado do resto do projeto.

git branch

- Às vezes queremos fazer modificações sem quebrar o que já temos pronto.
- O git te permite criar um “ramo” de trabalho isolado do resto do projeto.
- Digite `git branch`

git branch

- Às vezes queremos fazer modificações sem quebrar o que já temos pronto.
- O git te permite criar um “ramo” de trabalho isolado do resto do projeto.
- Digite `git branch`
- Agora digite `git branch mybranch`

git branch

- Às vezes queremos fazer modificações sem quebrar o que já temos pronto.
- O git te permite criar um “ramo” de trabalho isolado do resto do projeto.
- Digite `git branch`
- Agora digite `git branch mybranch`
- Digite `git branch` novamente.

git branch

git branch: lista branches existentes e cria
branches novos.

git checkout (de novo)

git checkout (de novo)

- Vamos mudar para o branch que criamos.

git checkout (de novo)

- Vamos mudar para o branch que criamos.
- Digite `git checkout mybranch`

git checkout (de novo)

- Vamos mudar para o branch que criamos.
- Digite `git checkout mybranch`
- Digite `git branch`

git checkout (de novo)

- Vamos mudar para o branch que criamos.
- Digite `git checkout mybranch`
- Digite `git branch`
- Modifique algum arquivo e dê commit

git checkout (de novo)

- Vamos mudar para o branch que criamos.
- Digite `git checkout mybranch`
- Digite `git branch`
- Modifique algum arquivo e dê commit
- Digite `git checkout master`

git checkout (de novo)

- Vamos mudar para o branch que criamos.
- Digite `git checkout mybranch`
- Digite `git branch`
- Modifique algum arquivo e dê commit
- Digite `git checkout master`
- As modificações são limitadas ao branch onde são feitas.

git checkout (de novo)

git checkout: também altera a versão do projeto para branches diferentes.

git merge

- Suponhamos que terminamos uma feature nova, e queremos integrá-la ao projeto principal.

git merge

- Suponhamos que terminamos uma feature nova, e queremos integrá-la ao projeto principal.
- Para isso, precisamos combinar 2 branches.

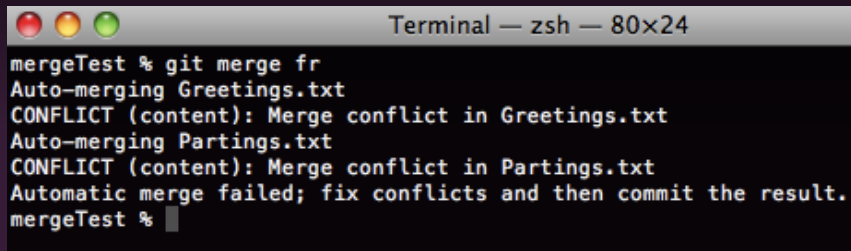
git merge

- Suponhamos que terminamos uma feature nova, e queremos integrá-la ao projeto principal.
- Para isso, precisamos combinar 2 branches.
- Modifique as mesmas linhas arquivo que você modificou no branch `mybranch`.

git merge

- Suponhamos que terminamos uma feature nova, e queremos integrá-la ao projeto principal.
- Para isso, precisamos combinar 2 branches.
- Modifique as mesmas linhas arquivo que você modificou no branch `mybranch`.
- Digite `git merge mybranch`

Conflitos



A terminal window titled "Terminal — zsh — 80x24" with three window control buttons (red, yellow, green) on the left. The terminal output shows a git merge command being executed, followed by two merge conflicts in "Greetings.txt" and "Partings.txt". The message "Automatic merge failed; fix conflicts and then commit the result." is displayed, followed by a prompt "mergeTest %" with a cursor.

```
mergeTest % git merge fr
Auto-merging Greetings.txt
CONFLICT (content): Merge conflict in Greetings.txt
Auto-merging Partings.txt
CONFLICT (content): Merge conflict in Partings.txt
Automatic merge failed; fix conflicts and then commit the result.
mergeTest %
```

Conflitos

```
23  
24 class String  
25 <<<<<< HEAD:lib/jekyll/core_ext.rb  
26   def cutoff(desired = 5)  
27     =====  
28     def cutoff(desired = 400)  
29 >>>>>> conflicts:lib/jekyll/core_ext.rb  
30     return self if self.length <= desired
```

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`
- O conflito fica salvo - você pode repetir a resolução do conflito se a anterior não for satisfatória.

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`
- O conflito fica salvo - você pode repetir a resolução do conflito se a anterior não for satisfatória.
- Voltem para o conflito usando `git checkout`

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`
- O conflito fica salvo - você pode repetir a resolução do conflito se a anterior não for satisfatória.
- Voltem para o conflito usando `git checkout`
- Usem `git checkout -b novo_branch` para criar um novo branch com o conflito

Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`
- O conflito fica salvo - você pode repetir a resolução do conflito se a anterior não for satisfatória.
- Voltem para o conflito usando `git checkout`
- Usem `git checkout -b novo_branch` para criar um novo branch com o conflito
- Resolvam o conflito de forma diferente no novo branch

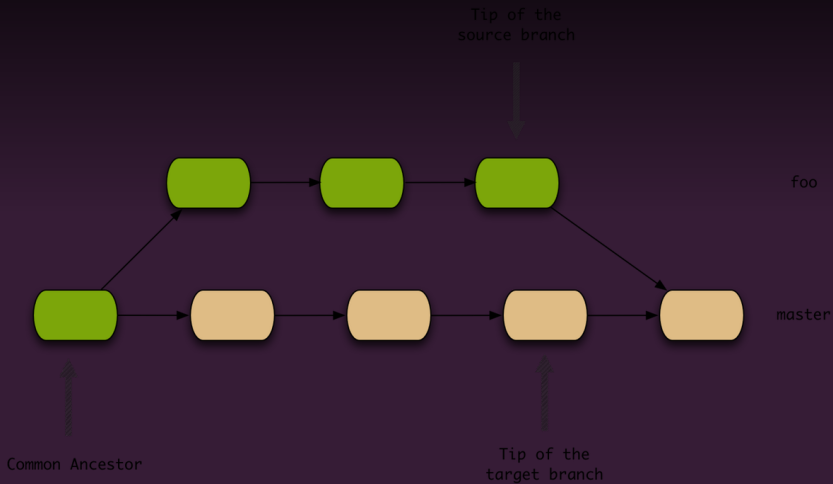
Conflitos

- Após resolver o conflito, comitem a versão com o conflito resolvido.
- Digitem `git log`
- O conflito fica salvo - você pode repetir a resolução do conflito se a anterior não for satisfatória.
- Voltem para o conflito usando `git checkout`
- Usem `git checkout -b novo_branch` para criar um novo branch com o conflito
- Resolvam o conflito de forma diferente no novo branch
- Deem merge do novo branch com `master` e resolvam o conflito novamente

Conflitos

git merge: combina duas versões diferentes do projeto.

O que vimos até agora



Dúvidas?

Trabalhando com repositório remoto

Provedores de repositórios



GitHub

Provedores de repositórios



git clone

- Vocês vão todos salvar uma cópia deste slide para terem como referência depois.

git clone

- Vocês vão todos salvar uma cópia deste slide para terem como referência depois.
- Vocês vão fazer isso usando o git!

git clone

- Vocês vão todos salvar uma cópia deste slide para terem como referência depois.
- Vocês vão fazer isso usando o git!
- Digitem `git clone https://github.com/pcstl/git-workshop.git`

git clone

git clone: copia código de repositórios remotos.

git remote

- Na pasta onde vocês baixaram os slides, digitem `git remote`

git remote

- Na pasta onde vocês baixaram os slides, digitem `git remote`
- Vocês vão ver um link com o nome `origin`

git remote

- Na pasta onde vocês baixaram os slides, digitem `git remote`
- Vocês vão ver um link com o nome `origin`
- Esse é um repositório *remoto* ligado a seu projeto.

git remote

- Na pasta onde vocês baixaram os slides, digitem `git remote`
- Vocês vão ver um link com o nome `origin`
- Esse é um repositório *remoto* ligado a seu projeto.
- Você também pode adicionar repositórios remotos com `git remote add nome_do_repositório link_do_repositório`

git remote

- Na pasta onde vocês baixaram os slides, digitem `git remote`
- Vocês vão ver um link com o nome `origin`
- Esse é um repositório *remoto* ligado a seu projeto.
- Você também pode adicionar repositórios remotos com `git remote add nome_do_repositório link_do_repositório`
- Repositórios remotos podem ser usados para *centralizar* o projeto.

git pull

- Suponhamos que um membro do seu time adicionou features novas ao repositório remoto e você quer pegar elas.

git pull

- Suponhamos que um membro do seu time adicionou features novas ao repositório remoto e você quer pegar elas.
- Para fazer isso, use `git pull nome_do_repositório nome_do_branch_remoto`

git pull

- Suponhamos que um membro do seu time adicionou features novas ao repositório remoto e você quer pegar elas.
- Para fazer isso, use `git pull nome_do_repositório nome_do_branch_remoto`
- Esse comando irá fazer um merge entre o branch em que você está e o branch que você escolheu no repositório remoto.

git pull

- Suponhamos que um membro do seu time adicionou features novas ao repositório remoto e você quer pegar elas.
- Para fazer isso, use `git pull nome_do_repositório nome_do_branch_remoto`
- Esse comando irá fazer um merge entre o branch em que você está e o branch que você escolheu no repositório remoto.
- Conflitos podem ocorrer da mesma forma que em um merge normal.

git push

- O contrário do `git pull` é o `git push`.

git push

- O contrário do `git pull` é o `git push`.
- Ele manda suas alterações para o repositório remoto.

git push

- O contrário do `git pull` é o `git push`.
- Ele manda suas alterações para o repositório remoto.
- Se o repositório estiver mais atualizado que sua versão, o push falha.

git push

- O contrário do `git pull` é o `git push`.
- Ele manda suas alterações para o repositório remoto.
- Se o repositório estiver mais atualizado que sua versão, o push falha.
- Para evitar isso, dê pull antes do push.

git push

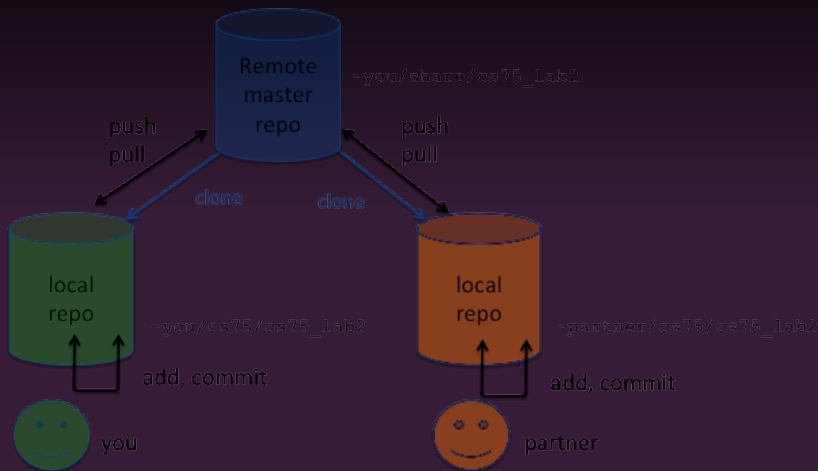
- O contrário do `git pull` é o `git push`.
- Ele manda suas alterações para o repositório remoto.
- Se o repositório estiver mais atualizado que sua versão, o push falha.
- Para evitar isso, dê pull antes do push.
- `sintaxe: git push nome_do_repositório nome_do_branch_local`

git push

git pull: puxa atualizações do repositório remoto para sua versão local.

git push: manda atualizações da sua versão local para o repositório remoto.

O que vimos até agora



Dúvidas?

Colaboração

Fluxo de trabalho

- O ideal é que pessoas diferentes do grupo trabalhem em coisas diferentes para evitar conflitos.

Fluxo de trabalho

- O ideal é que pessoas diferentes do grupo trabalhem em coisas diferentes para evitar conflitos.
- Mantenham a versão “oficial” no repositório remoto - assim, se der merda, vocês sempre podem puxar de lá.

Fluxo de trabalho

- O ideal é que pessoas diferentes do grupo trabalhem em coisas diferentes para evitar conflitos.
- Mantenham a versão “oficial” no repositório remoto - assim, se der merda, vocês sempre podem puxar de lá.
- Nunca mandem código que não funciona para o remoto - trabalhem localmente até funcionar.

Fluxo de trabalho

- O ideal é que pessoas diferentes do grupo trabalhem em coisas diferentes para evitar conflitos.
- Mantenham a versão “oficial” no repositório remoto - assim, se der merda, vocês sempre podem puxar de lá.
- Nunca mandem código que não funciona para o remoto - trabalhem localmente até funcionar.
- Desenvolvam features diferentes em branches separados - só deem merge quando a feature estiver pronta.

Fluxo de trabalho

- O ideal é que pessoas diferentes do grupo trabalhem em coisas diferentes para evitar conflitos.
- Mantenham a versão “oficial” no repositório remoto - assim, se der merda, vocês sempre podem puxar de lá.
- Nunca mandem código que não funciona para o remoto - trabalhem localmente até funcionar.
- Desenvolvam features diferentes em branches separados - só deem merge quando a feature estiver pronta.
- Em último caso, checkout na última versão que funcionava!

Interfaces gráficas

GitKraken

The screenshot displays the GitKraken application interface. The top bar shows the current repository is 'GitKraken' on the 'master' branch. The left sidebar lists local and remote branches, with 'LOCAL' branches including '0.1.40' and '0.1.39'. The main area shows a commit history graph with a list of commits on the right. The selected commit is 'Bump to version 0.1.40' by 'John Haley'.

Commits:

- Fix unstage and staging
- Keep rename detection stage/unstage all
- Bump to version 0.1.40
- Merge pull request #597 from johnhaley81/fix-dispatch...
- Merge pull request #594 from Mr-Wallettricker-ref-ran...
- Fix "waitfor" bug in dispatcher
- Merge pull request #596 from johndavidsparrowhigh-p...
- Reveal custom variable script and switch
- Merge pull request #595 from johndavidsparrowhigh-p...
- Resolved edge case where Rethreads could overlap
- Universe removal of in-app invite wording
- Bump to version 0.1.39
- Merge pull request #591 from Mr-Wallettricker-graph-ref...
- Merge pull request #590 from strajcoldiv-be-gone
- Merge pull request #588 from Mr-Wallettricker-renderer-app...
- Merge pull request #568 from Mr-Wallettricker-ref-ran...
- Merge pull request #589 from johndavidsparrowhigh-p...
- Merge pull request #592 from implausibleFahNSFW
- JS tidy up in form-validation.js
- Javascript update for universe
- Universe page
- added maxwait to updateworker debounce
- Update NSFW for memory leak
- Fix NSFW segfault
- Fix flickering GraphiteColumn every time ... 6 days ago
- Preventing page reload on default pull click
- Eliminate console spam when conflicts exist in a stanell...
- Upgrading to react-bootstrap v0.24.5

Diff View (Bump to version 0.1.40):

0 added, 0 deleted, 2 modified

package.json

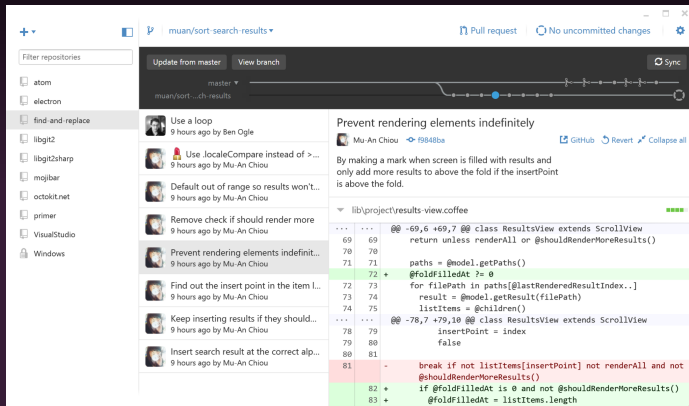
```
1 1 {
2 2   "name": "gitkraken",
3 3   "version": "0.1.39",
4 4   "version": "0.1.40",
5 5   "dependencies": {
6 6     "atom-keymap": {
7 7       "version": "5.1.11",
```

package.json

```
1 1 {
2 2   "name": "gitkraken",
3 3   "productName": "GitKraken",
4 4   "version": "0.1.39",
5 5   "version": "0.1.40",
6 6   "description": "An intuitive git cli
7 7   "main": "./src/appBootstrap/main.js"
```

[Provide Feedback](#)

Github Desktop



Buscando ajuda

Sites úteis

- Documentação oficial do git → <https://git-scm.com/documentation>
- Tutoriais do GitHub → <https://guides.github.com/>
- Tutoriais da Atlassian → <https://www.atlassian.com/git/tutorials/>

Dúvidas?