

# Workflows de trabalho utilizando git

---

Advanced Institute for Artificial Intelligence – AI2

<https://advancedinstitute.ai>

“

*Receita ou recomendação sobre como usar o Git para realizar o trabalho de maneira consistente e produtiva*

- ☐ Incentivam os usuários a aproveitar o Git de modo eficiente e consistente
- ☐ Dado o foco do Git em flexibilidade, não há nenhum processo padronizado de como interagir com o Git;
- ☐ É importante ter certeza de que a equipe toda esteja de acordo com como o fluxo de mudanças será aplicado

- Ao avaliar um fluxo de trabalho para sua equipe, o mais importante é considerar a cultura da equipe
- Algumas coisas a considerar ao avaliar um fluxo de trabalho do Git são:
  - **Este fluxo de trabalho é dimensionado com o tamanho da equipe?**
  - **É fácil desfazer erros com este fluxo de trabalho?**
  - **Este fluxo de trabalho impõe alguma nova sobrecarga cognitiva desnecessária à equipe?**

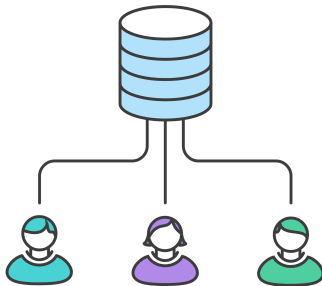


# Centralized Workflow

---

# Fluxo de trabalho centralizado

- ☐ Fluxo de trabalho do Git para equipes em transição do SVN
- ☐ Usa um repositório central para servir como único ponto de entrada para todas as mudanças no projeto



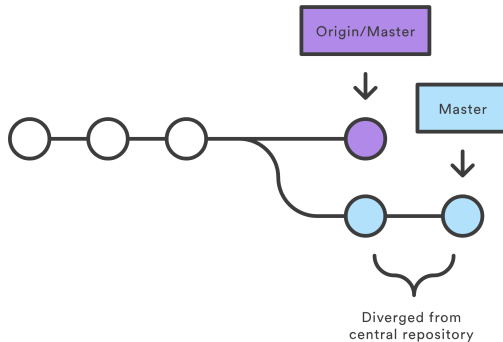
- ❑ Não requer nenhuma outra ramificação além de `master`
- ❑ **O mais simples dos fluxos de trabalho**
- ❑ Como funciona:
  - clone do repositório central
  - Em cópias locais são feitas edições dos arquivos e confirmação das mudanças (`git add` & `git commit`)
  - Fazer push da *branch* `master` local para a cópia remota;

## Exemplo:

```
1 $ git clone <algum repositório>
2 # Edição de arquivos;
3 $ git add <arquivos modificados>
4 $ git commit -m "descrição rápida de modificações"
5 $ git push origin master
```

# Fluxo de trabalho centralizado

## □ Conflitos:





## □ Resolvendo conflitos (exemplo):

- John trabalha nas suas mudanças



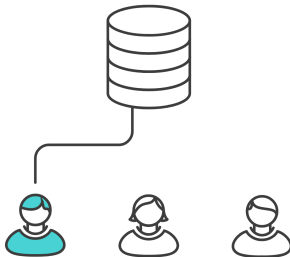
- Resolvendo conflitos (exemplo):
  - Mary trabalha nas suas mudanças



# Fluxo de trabalho centralizado

## □ Resolvendo conflitos (exemplo):

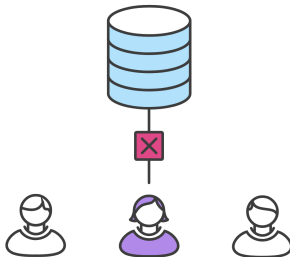
- John publica suas mudanças



```
1 $ git push origin master
```

# Fluxo de trabalho centralizado

- Resolvendo conflitos (exemplo):
  - Mary tenta publicar suas mudanças



```
1 $ git push origin master
```

# Fluxo de trabalho centralizado

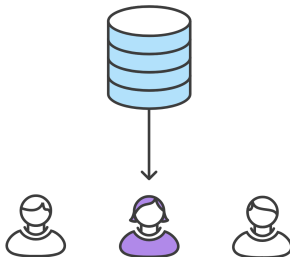
## ❑ Resolvendo conflitos (exemplo):

- Mary tenta publicar suas mudanças

```
1  ! [rejected]          master -> master (fetch first)
2  error: failed to push some refs to '/Users/rocknroll/github/meu-
   repositorio'
3  hint: Updates were rejected because the remote contains work that you
   do
4  hint: not have locally. This is usually caused by another repository
   pushing
5  hint: to the same ref. You may want to first integrate the remote
   changes
6  hint: (e.g., 'git pull ...') before pushing again.
7  hint: See the 'Note about fast-forwards' in 'git push --help' for
   details.
```

# Fluxo de trabalho centralizado

- Resolvendo conflitos (exemplo):
  - Mary faz o rebase de suas mudanças

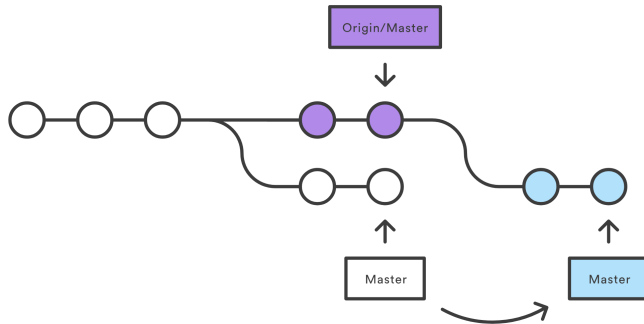


```
1 $ git pull --rebase origin master
```

# Fluxo de trabalho centralizado

## □ Resolvendo conflitos (exemplo):

- -rebase diz ao Git para mover todos os commits de Mary para a ponta da ramificação mestre



## ❑ Resolvendo conflitos (exemplo):

- Se Mary e John estiverem trabalhando em recursos não relacionados, é improvável que o processo de rebase gere conflitos.
- se gerar, o Git vai pausar o rebase na confirmação atual e enviar a seguinte mensagem, juntamente com algumas instruções relevantes:

```
1  CONFLICT (content): Merge conflict in <file>
2  Resolve all conflicts manually, mark them as resolved with
3  "git add/rm <conflicted_files>", then run "git rebase --continue".
```



## ❑ Resolvendo conflitos (exemplo):

- Verificando o status no repositório de Mary

```
1 $ git status
2 Unmerged paths:
3   (use "git restore --staged <file>..." to unstage)
4   (use "git add <file>..." to mark resolution)
5 ^^Iboth modified:   <file>
```

## ❑ Resolvendo conflitos (exemplo):

- Verificando o conflito no repositório de Mary

```
1 <<<<<< HEAD
2 <Local File Content>
3 =====
4 <Remote File Content>
5 >>>>>> Remote Repo Commit Message
6 <Common ContentL>
```

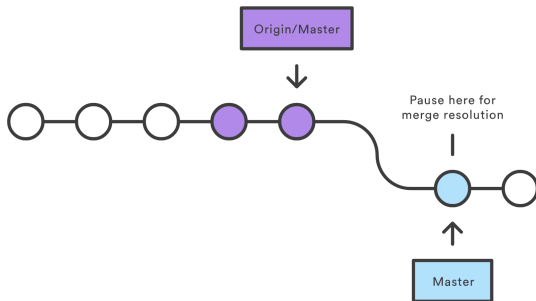
- Resolvendo conflitos (exemplo):
  - Continuando a operação de rebase

```
1 $ git rebase --continue
```

# Fluxo de trabalho centralizado

## □ Resolvendo conflitos (exemplo):

- Resultado do rebase no histórico de commits





# Feature Branch Workflow

---

# Fluxo de trabalho de Branch por Feature

- ☐ Focado no modelo de branch
- ☐ Desenvolvimento de *features* deve ocorrer em um branch dedicado
- ☐ Facilita o trabalho em uma *feature* específica sem interromper a principal base de código
- ☐ **branch principal nunca vai conter um código quebrado**
- ☐ *pull requests* para iniciar discussões em torno de um branch
- ☐ Dão a outros desenvolvedores a oportunidade de aprovar um recurso antes que ele seja integrado ao projeto oficial

# Fluxo de trabalho de Branch por Feature

- ❑ O *branch* principal representa o histórico oficial do projeto
- ❑ desenvolvedores criam um novo branch sempre que começam a trabalhar em uma nova *feature*
- ❑ Os branches dos recursos devem ter nomes descritivos:
  - issue-#1061
  - carregamento-de-dados
- ❑ Dar um objetivo claro e bastante focado a cada *branch*
- ❑ Os branches de *features* devem ser enviados para o repositório remoto

# Fluxo de trabalho de Branch por Feature

## □ Passo-a-passo

- Certificar-se que o branch master está atualizado com a cópia remota
- `git reset -hard origin/master` vai limpar todas as modificações locais feitas na branch master

```
1 $ git checkout master
2 $ git fetch origin
3 $ git reset --hard origin/master
```



# Fluxo de trabalho de Branch por Feature

## □ Passo-a-passo

- Criar nova branch baseada na `master` para a nova *feature*

```
1 $ git checkout -b nova-feature
```

## □ Passo-a-passo

- Escrever modificações, adicioná-las e realizar o *commit*

```
1 $ git add <arquivo>
2 $ git commit -m "Mensagem de Commit"
```

# Fluxo de trabalho de Branch por Feature

## □ Passo-a-passo

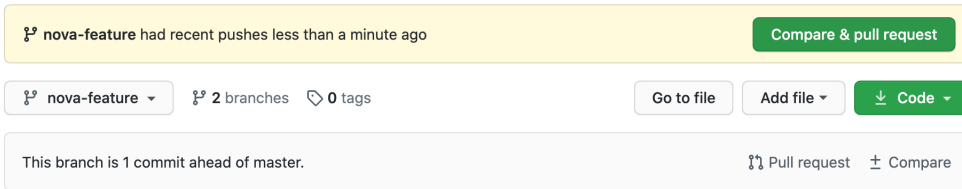
- Enviar branch do recurso para repositório remoto

```
1 $ git push -u origin nova-feature
```

# Fluxo de trabalho de Branch por Feature

## □ Passo-a-passo

- Criando *pull-request*




# Fluxo de trabalho de Branch por Feature

## □ Passo-a-passo


- Criando *pull-request*

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: master ▼ ← compare: nova-feature ▼

✓ **Able to merge.** These branches can be automatically merged.



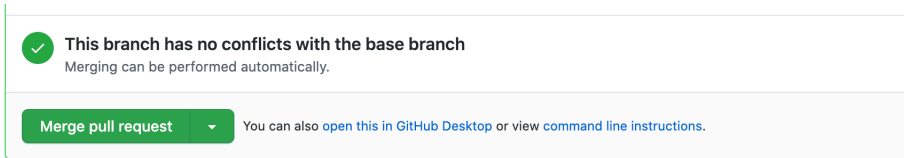
Adicionado o planeta Marte

2

# Fluxo de trabalho de Branch por Feature

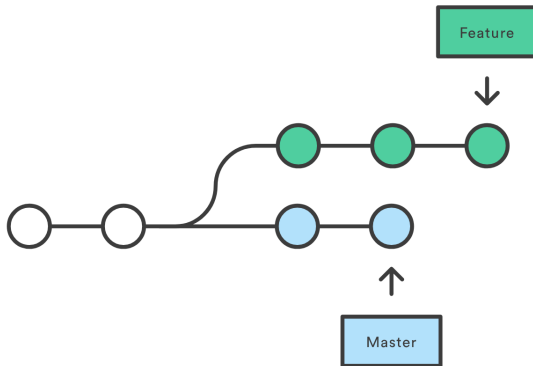
## □ Passo-a-passo

- Aceitando *pull-request*



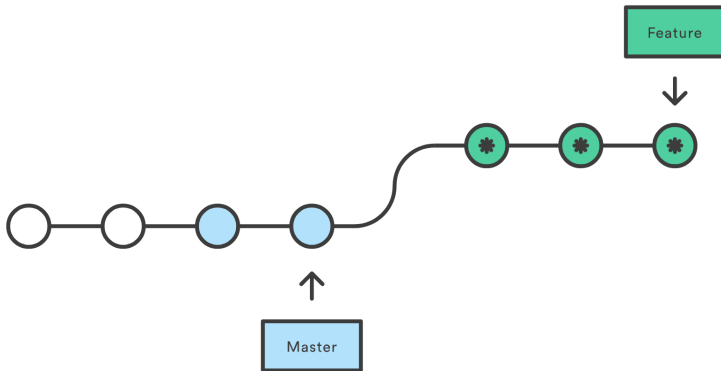
# Fluxo de trabalho de Branch por Feature

- Caso a branch da nova *feature* esteja atrás da *master*



# Fluxo de trabalho de Branch por Feature

- rebase para atualizar branch

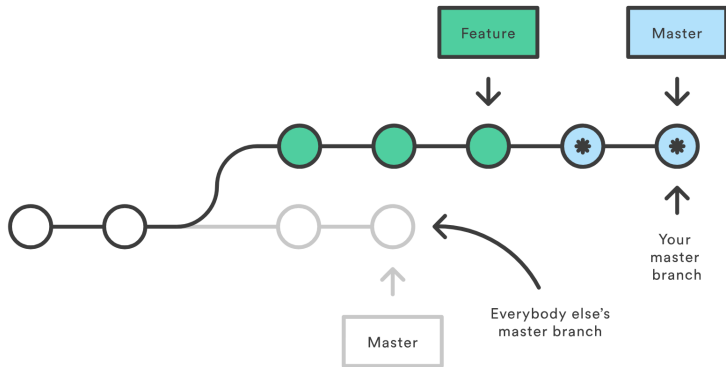




# Fluxo de trabalho de Branch por Feature

## □ A regra de ouro para o rebase

- Nunca realizar rebase em branches públicas!





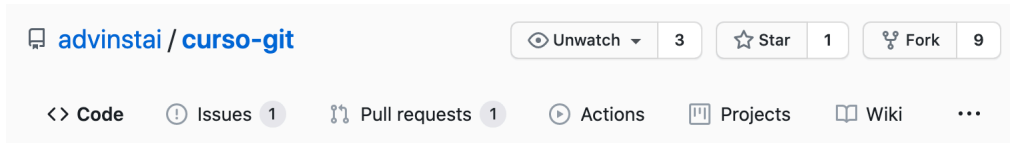
# Forking Workflow

---

- No Forking Workflow o usuário possui dois repositórios: um **privado (pessoal)** e um **público (principal/central)**.
- Conceito de “validação e aprovação”
  - Focado na qualidade do projeto e no nível de abertura para contribuições;
  - Colaboradores não precisam necessariamente de permissões para dar push no repositório oficial
  - Submissão de *Pull Requests* para o repositório principal
  - Após a revisão, correções são feitas no fork privado e agregando alterações ao *pull request*

## ❑ Como funciona?

- Criando um fork no github.



## □ Como funciona?

- Clonar seu fork

```
1 $ git clone <repositorio do seu fork>
```

## □ Como funciona?

- Adicionar novo repositório remoto (e.g., upstream)

```
1 $ git remote add upstream <repositorio de onde você realizou o fork>
```

## □ Como funciona?

- Criar branch para nova *feature*

```
1 $ git checkout -b nova-feature
2 # Edit some code
3 $ git commit -a -m "<Mensagem de Commit>"
```

## □ Como funciona?


- Enviar branch para o seu repositório privado remoto

```
1 $ git push origin nova-feature
```



## ☐ Como funciona?

- Criando um pull request



 novo-readme had recent pushes less than a minute ago

Compare & pull request




# Fluxo de trabalho baseado em Forks

- ❑ Como funciona?
  - Aprovando um pull request



## Primeira versão do README #4

 **Open** raphaelmcobe wants to merge 1 commit into `advinstai:master` from `raphaelmcobe:novo-readme` 



Conversation **0** Commits **1** Checks **0** Files changed **1**


 **raphaelmcobe** commented now Member  


*No description provided.*

  Primeira versão do README dfb7dd0

See more commits by pushing to the `novo-readme` branch on `raphaelmcobe/curso-git`.

  Continuous integration has not been set up  
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

 **This branch has no conflicts with the base branch**  
Merging can be performed automatically.

**Merge pull request**  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

## □ Como funciona?

- Atualizando branch com novas alterações enquanto você trabalha em sua funcionalidade

```
1 $ git rebase upstream/master
```