Controle de versão de software

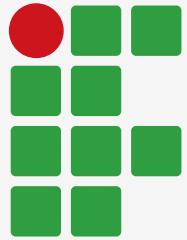
GIT

Hugo Marcondes

Departamento Acadêmico de Eletrônica

DAELN

hugo.marcondes@ifsc.edu.br



INSTITUTO FEDERAL

Santa Catarina

Câmpus Florianópolis

Agenda

- O que é controle de versão
- Comandos Básicos GIT
- Branching, Merging e Tags
- Utilizando repositórios remotos

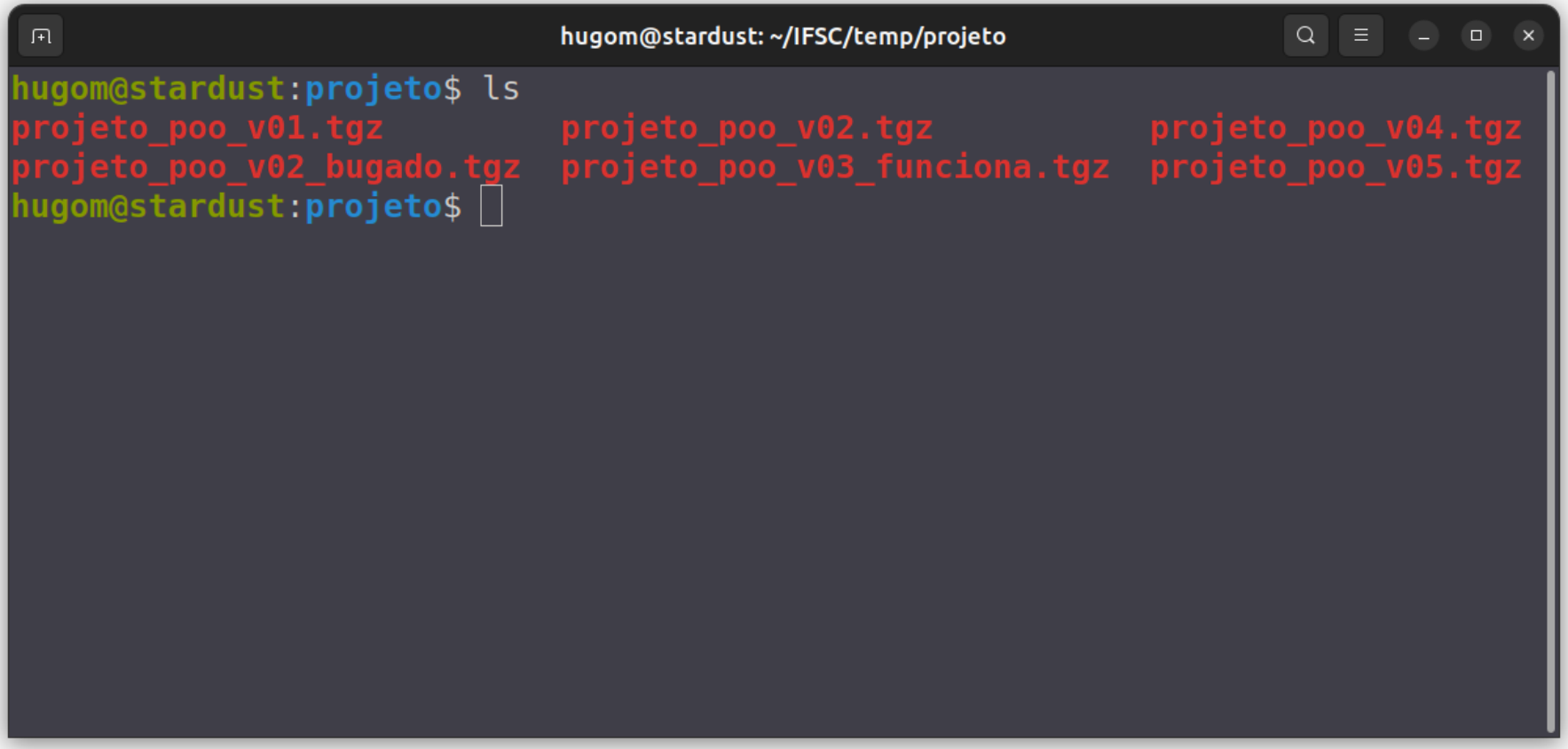
O que é controle de versão

Controle de versões

- Controle de versão é uma ferramenta que permite que você:
- **Colabore**
  - Crie projetos com outras pessoas, sejam eles relacionados ao desenvolvimento de software ou não
- **Monitore** e **Reverta** as modificações
  - Erros acontecem. Com o controle de versão é possível monitorar todas as modificações realizadas no projeto, e também reverter as mesmas.

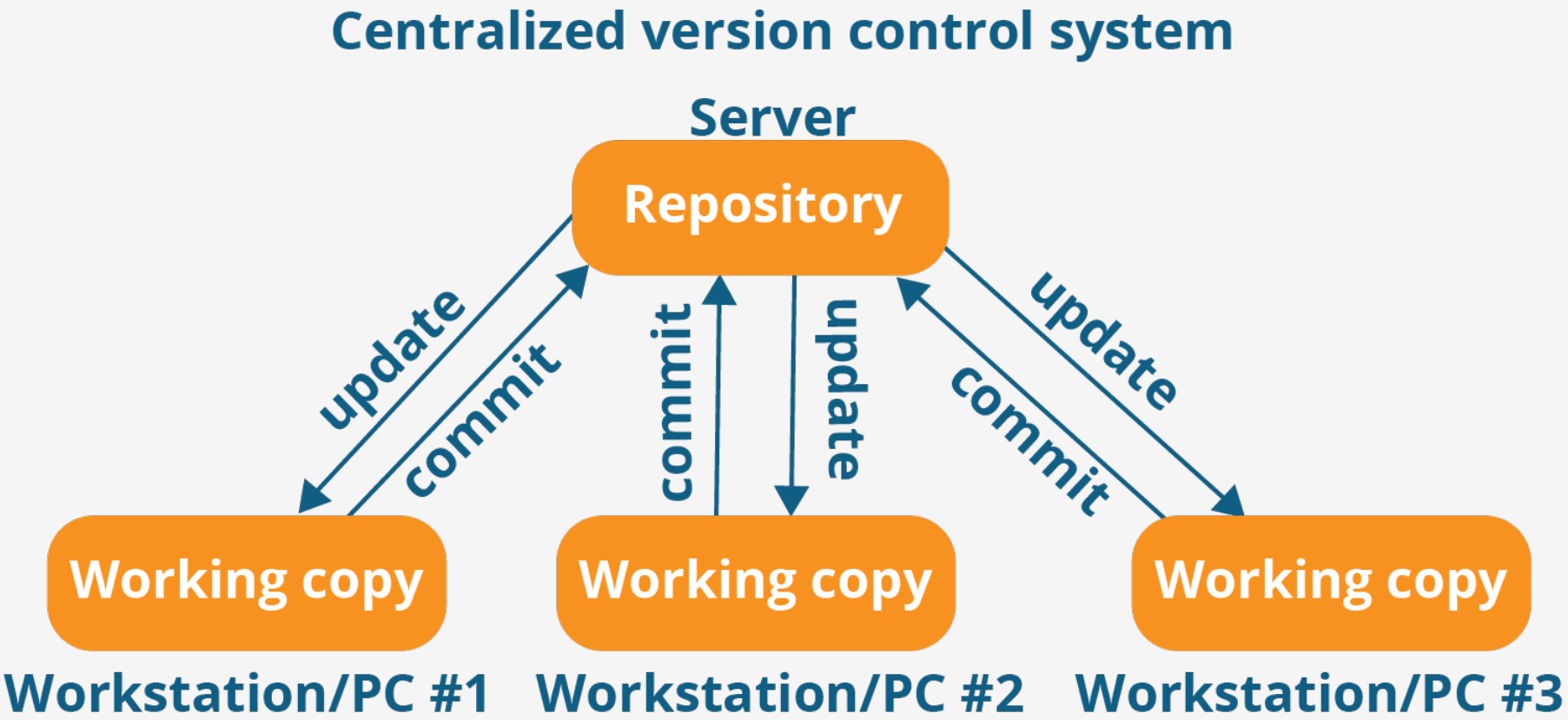
Você já gerencia versões dos seus trabalhos, não?

- Você tem arquivos assim?



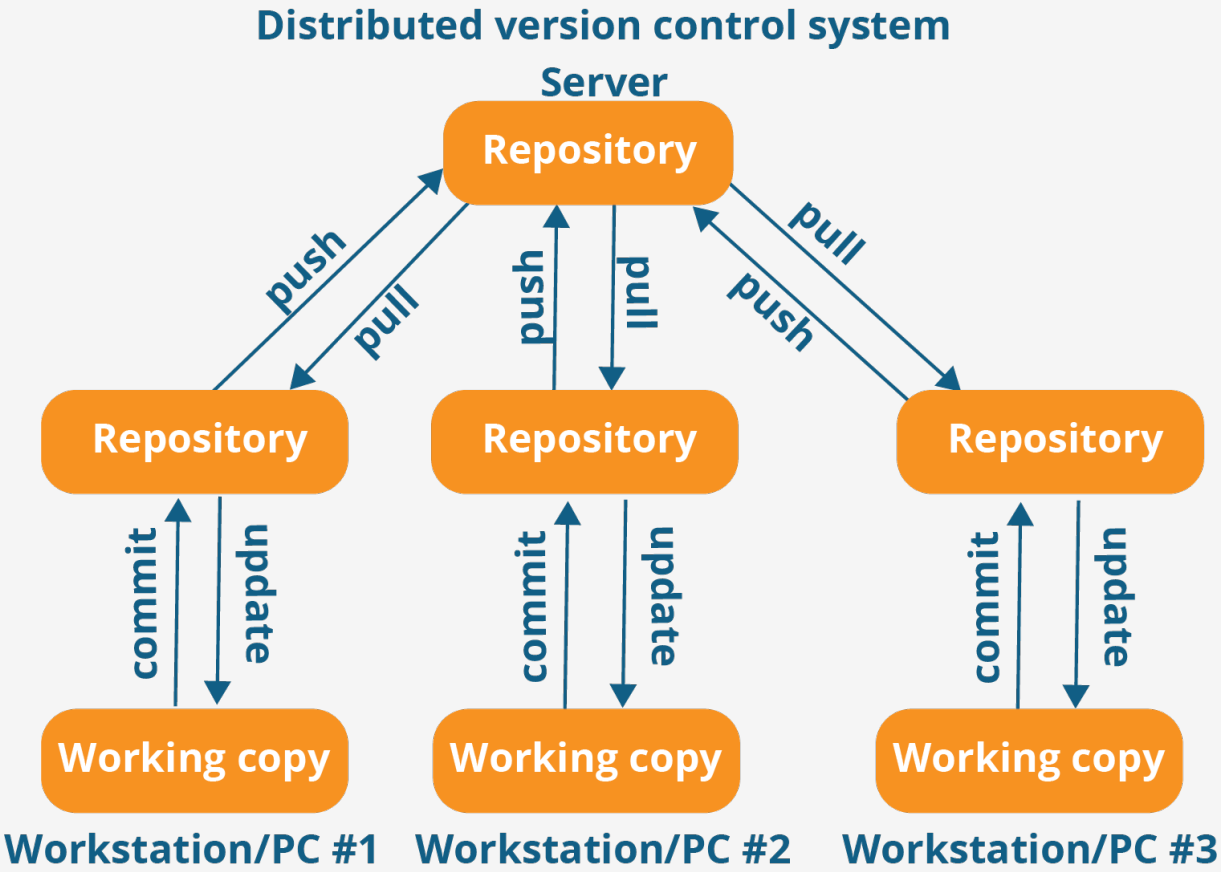
Tipos de Controle de Versão

- Centralizados
  - Exemplos: CVS, Subversion (SVN), Perforce



Tipos de Controle de Versão

- Distribuidos
  - Exemplos: GIT, Mercurial



GIT

- Uma necessidade no desenvolvimento do kernel do Linux
- 1991 — 2002: Uso de patches e coleções de arquivos
- 2002 — 2005: BitKeeper (DVCS)
- 2005: BitKeeper deixa de ser disponibilizado sem custos.
- Linux Torvalds então decide desenvolver o próprio DVCS para uso no desenvolvimento do kernel do Linux focando nos seguintes princípios:
  - Velocidade
  - Simplicidade
  - Forte suporte ao desenvolvimento não linear (diversos “branchs” paralelos)
  - Totalmente distribuído
  - Ser escalável para projetos grandes, como o kernel do Linux

Vocabulário do Git

- Um **repositório** é um diretório onde você armazena todos os arquivos que vc deseja rastrear as modificações
- Um ramo (**branch**) é o nome dado a uma linha separada de desenvolvimento de um repositório, que possui o seu próprio histórico
- Um **commit** é um objeto que armazena informações sobre um conjunto de modificações no repositório. Os commits são feitos em um determinado branch
- **HEAD** refere-se ao commit (geralmente o mais recente) para qual o diretório local do repositório aponta no branch atual

Instalando o Git

- Ferramenta em linha de comando — $ git
- Diversos clientes, alguns integrados ao próprio gerenciador de arquivos do SO
- https://git-scm.com/

Comandos Básicos GIT

Principais comandos Git

- init
- add
- commit
- status
- log
- branch
- checkout
- fetch
- merge
- pull
- push
- clone

Criando um repositório local

- Qualquer diretório do seu sistema pode armazenar um repositório Git, para isso basta utilizar o comando “**git init**”

1. Crie um diretório no seu sistema
2. Execute o comando “**git init**”
3. Verifique o estado do seu repositório com o comando “**git status**”

- Os comandos básicos para realizar a gerenciamento dos arquivos no repositório são
  - $ git **add**
  - $ git **commit**
  - $ git **log**

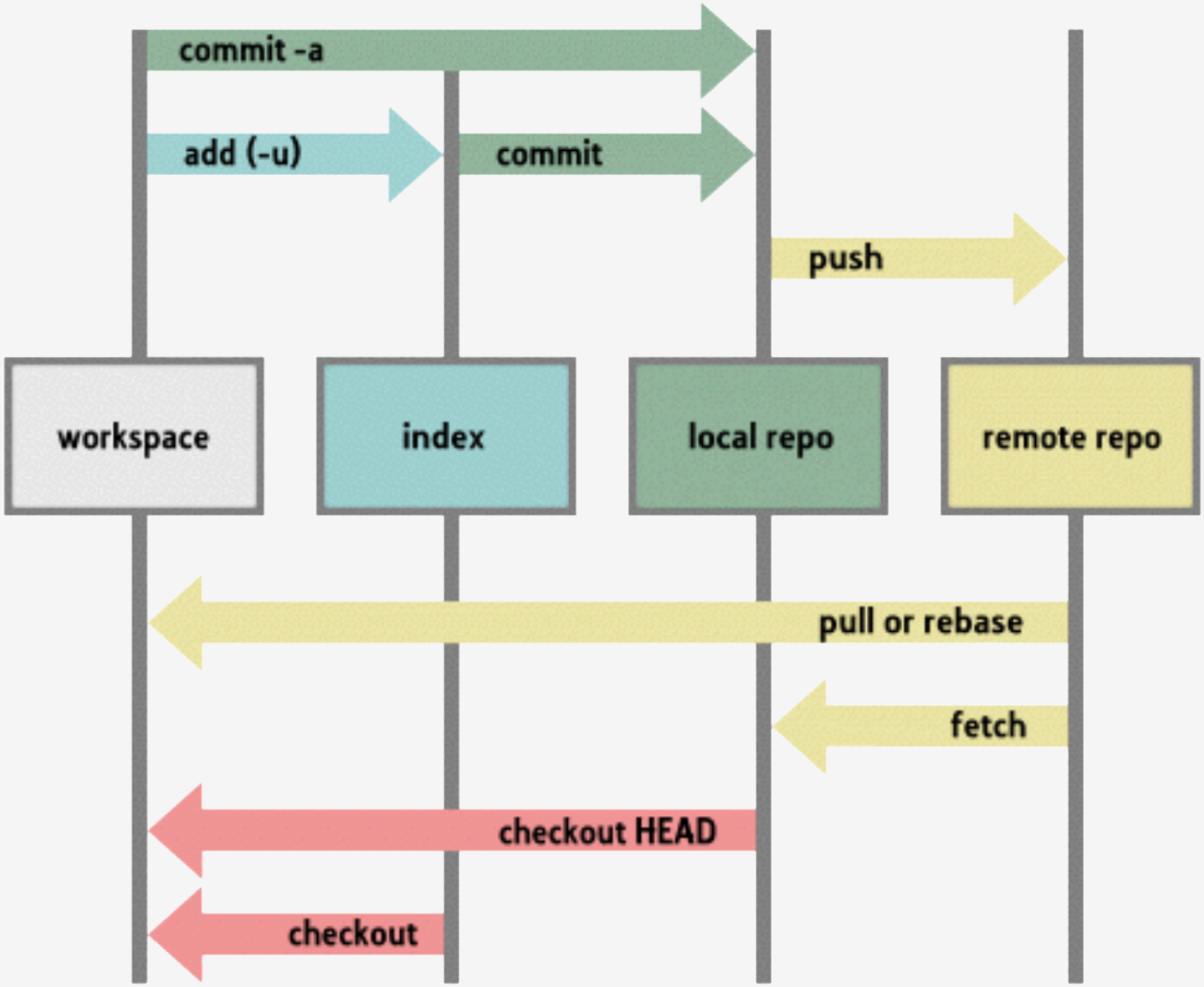
O que fizemos?

- Quando adicionamos (**add**) um novo arquivo, dizemos ao Git para iniciar o monitoramento das modificações naquele arquivo
- Neste estágio, o arquivo é chamado de “**staging** file”. Um snapshot das nossas modificações do arquivo estão na área conhecida como “**staging area**” (também conhecido como **index** ou **cache**). Neste estágio, as modificações estão prontas para serem salvas
- O comando **commit** salva as modificações realizadas no arquivo, não o arquivo como um todo. Cada operação de commit, possui um ID único de forma que podemos rastrear todas as modificações feitas, quando foram feitas e por quem.

Explore um pouco os comandos básicos

- Realize modificações no arquivo criado em seu repositório.
- Adicione outros arquivos (experimente arquivos não textuais) em seu projeto e realize o commit dessas modificações
- Modifique mais que um arquivo por vez, e pratique a realização de commits colocando no staging apenas um dos arquivos, ou ambos.
- Experimente utilizar o comando “**git diff**” quando possui modificações “**unstaged**”
- Verifique o histórico de modificações com o comando “**git log**” conforme você adiciona mais commits (utilize a opção –oneline)

Os “domínios” do Git



Para que devemos usar controle de versão?

- Para **QUALQUER** coisa, mas **NÃO** para **TUDO**!

Gitignore

- É possível selecionar quais arquivos serão ou não rastreados com o Git, e desta forma conseguimos deixar de fora o que não deve ser rastreado.
  - Arquivos backup, objeto, temporários derivados da compilação, etc. . .
- Através do arquivo ġitignore na raiz do diretório do seu projeto.
  - Padrões de nomes de arquivos em cada linha
- https://gitignore.io — Sugestões de arquivos ġitignore

Lidando com os erros...

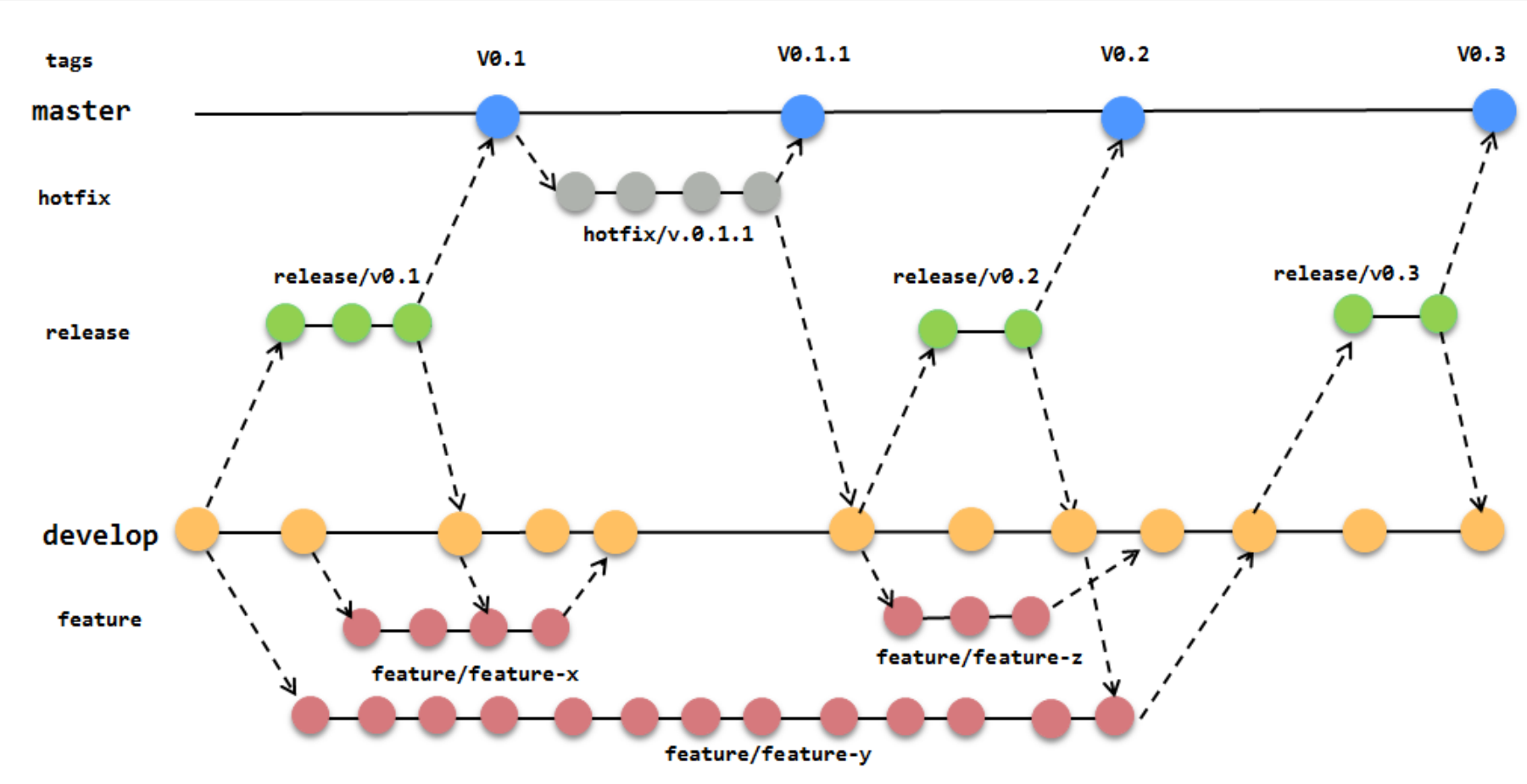
- Como desfazer mudanças que não deram certo?
- O processo irá depender do estágio em que a sua modificação está
  - Se você ainda não adicionou a sua modificação no staging, basta usar o comando "**git checkout**"
  - Se o arquivo já está no staging, utilize o comando "**git reset**"
    - Remove o arquivo do staging, mas mantêm na cópia local
  - Se quiser voltar a ultima versão "commitada", utilize o comando **reset** e depois **checkout**
  - Para desfazer um commit
    - **git reset –soft <commit_ID>**
    - **git reset –hard <commit_ID>**

Branching, Merging e Tags

Branching

- Um ramo (**branch**) é essencialmente uma outra cópia do seu repositório em que você pode isolar as modificações feitas, e manter a cópia original sem modificações.
  - Depois você pode decidir em combinar (**merge**) essas modificações, descartá-las integralmente, totalmente ou em parte em sua cópia original (**master**), ou simplesmente descartá-las integralmente
- Ramos são ideais para a implementação de novas funcionalidades ou resolução de bugs.

Branching


tags
V0.1
V0.1.1
V0.2
V0.3
master
hotfix
hotfix/v.0.1.1
release/v0.1
release/v0.2
release/v0.3
release
develop
feature
feature/feature-x
feature/feature-z
feature/feature-y

Branching

- Desenvolvimento de um novo código sobre a mesma base.
- Conduzir experimentos sem afetar o trabalho no ramo principal (master branch)
- Incorporar mudanças ao ramo principal apenas se estiverem prontas, ou descartá-las completamente de forma muito fácil
- Ramos podem ser criados através do comando de **checkout**

```
$ git checkout -b <branch_name>
```

- O comando **branch** é utilizado listar e remover os ramos

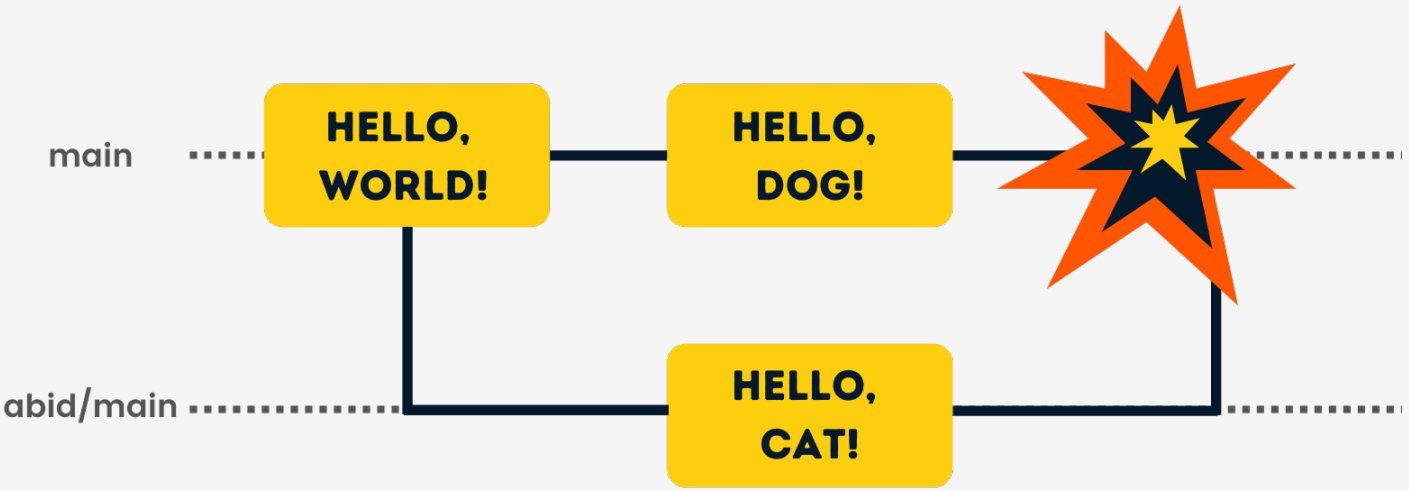
Merging

- Uma vez criado, o branch é desenvolvido e pode receber commits normalmente.
- Uma vez que a funcionalidade está concluída é necessário realizar junção ao ramo principal do repositório (master). Este processo é chamado de **merging**.
- É realizado através do comando **merge**

1. Mudo para o branch que irá receber as modificações
2. Executo o comando merge passando como parâmetro, o branch que quero incorporar

```
$ git checkout master
$ git merge <branch>
```


Conflitos



- Quando um conflito é identificado, o arquivo é anotado com as informações sobre o conflito.
- Conflitos devem ser resolvidos manualmente
- Uma vez resolvidos basta adicionar os arquivos no stage e realizar o commit.

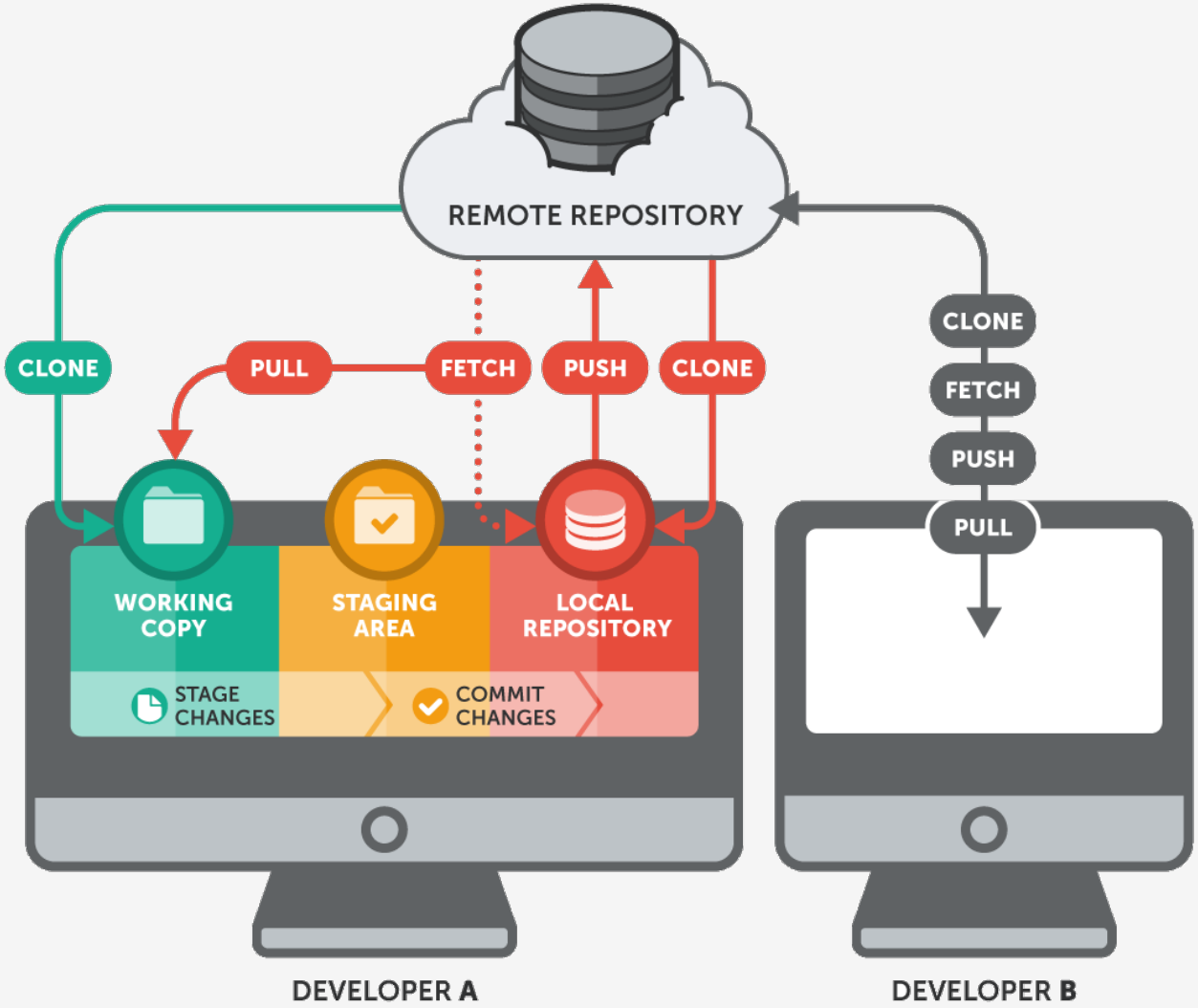
Tagging

- Não use ramos, quando você deseja apenas nomear um determinado commit do seu repositório (i.e, não haverá desenvolvimento naquele ramo)
- As tags podem ser criadas com o comando tag

```
$ git tag [-a] <tag_name> [<commit_id>]
```

- A opção **[-a]** define se a tag é “**anotada**” ou “**lightweight**”
- As tags podem ser acessadas normalmente com o comando “**checkout**”
- Para listar as tags de um repositório utilize o comando sem parâmetros

The BIG picture



Utilizado repositórios remotos


github
SOCIAL CODING

GitHUB

- Serviço online que oferece a hospedagem de repositórios baseados em Git
- Mais de 20 milhões de usuários, compartilhando mais de 57 milhões de repositórios!
- Grande integração com web
  - Possibilidade de acesso ao repositório através do browser
  - É possível até editar arquivos através do página do GitHUB

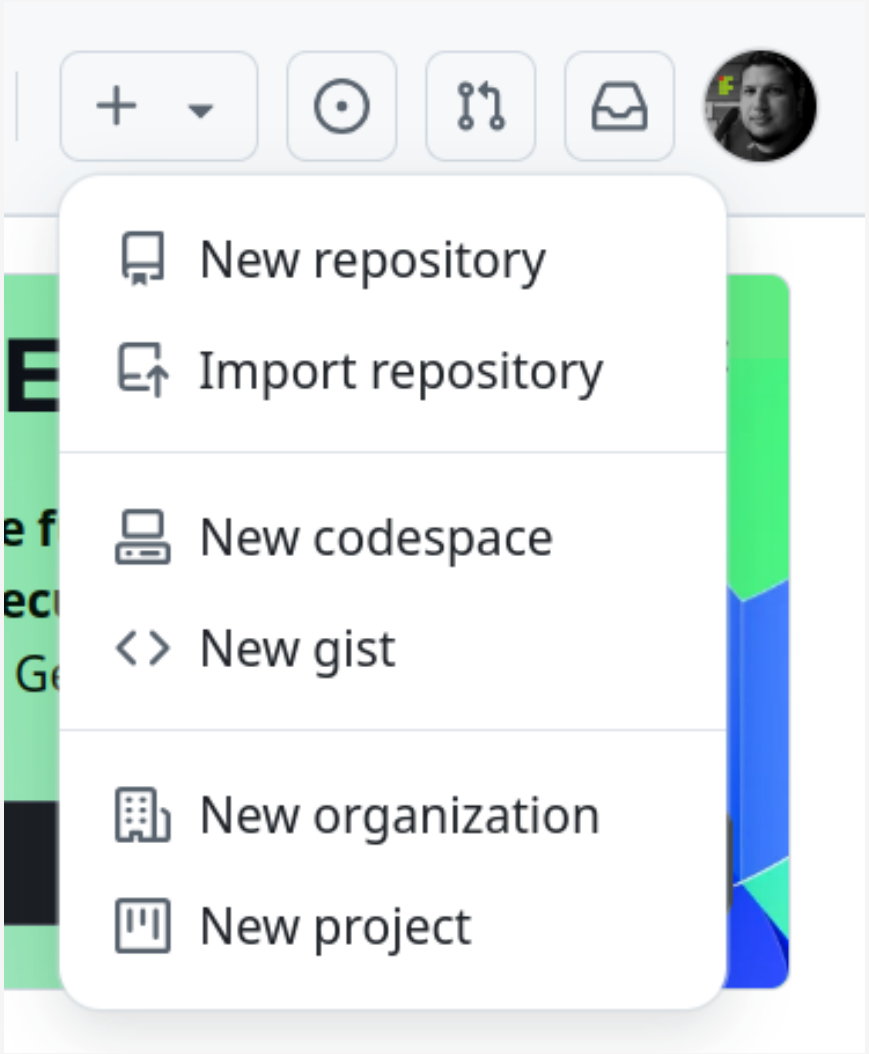
O que fazer no GitHUB?

- Hospedagem de repositórios públicos ou privados (com custos)
  - Grandes projetos OpenSource hospedam/espelham no GitHUB
  - Permite que outros colaborem e **utilizem o seu projeto**
  - **Aprenda** acessando o repositório de outros usuários
  - **Contribua** com projetos OpenSource

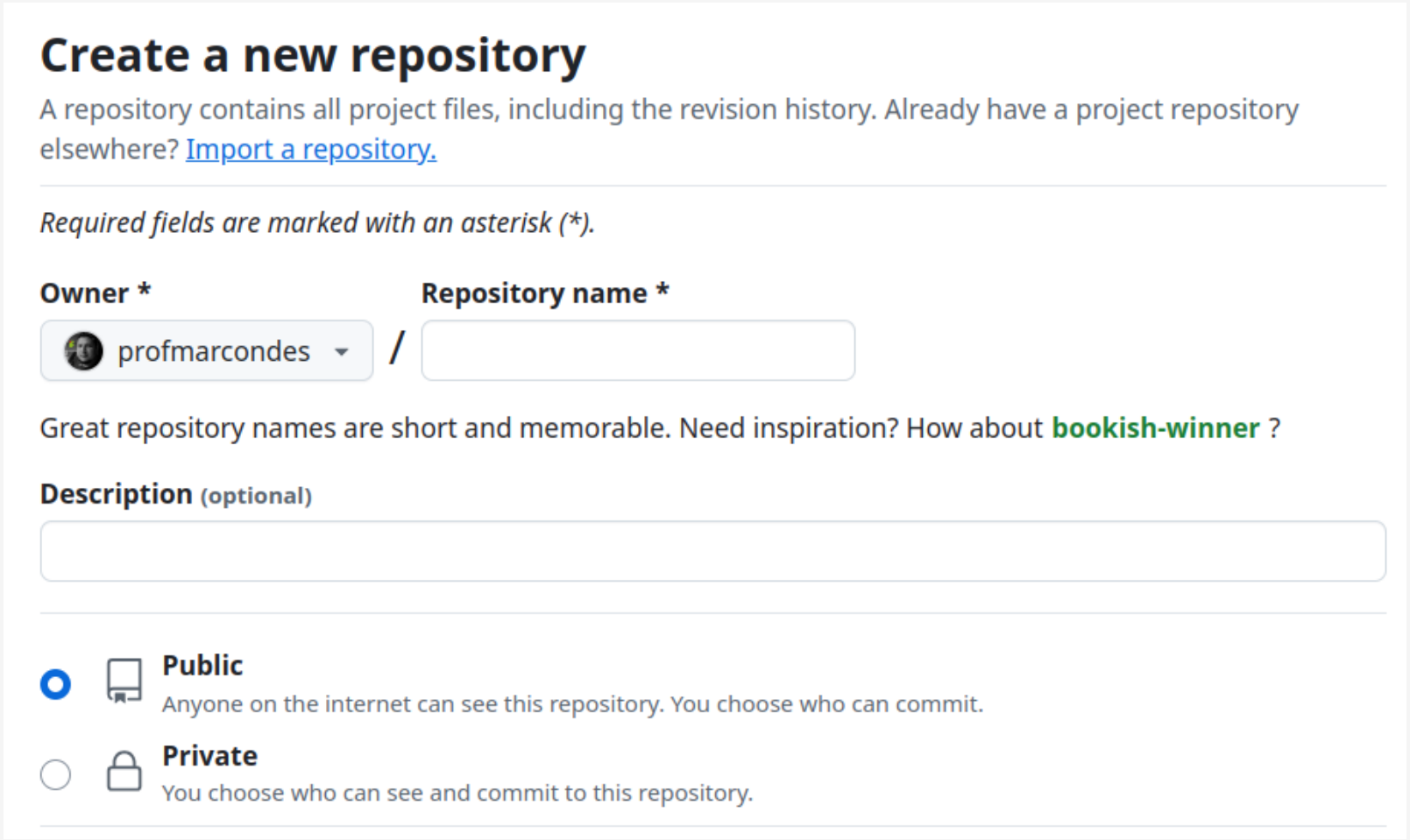
- Não estranhe se uma empresa solicitar o seu perfil no GitHUB para um entrevista de emprego. **O seu perfil pode contar muitos pontos na seleção de uma vaga!**

Mas como integrar o meu repositório do GiHUB?

- Antes de mais nada é necessário criar um repositório no GitHUB
- Caso não tenha ainda, uma conta no GitHUB primeiro



Criando o repositório no GitHUB



Utilizando o repositório no GitHUB

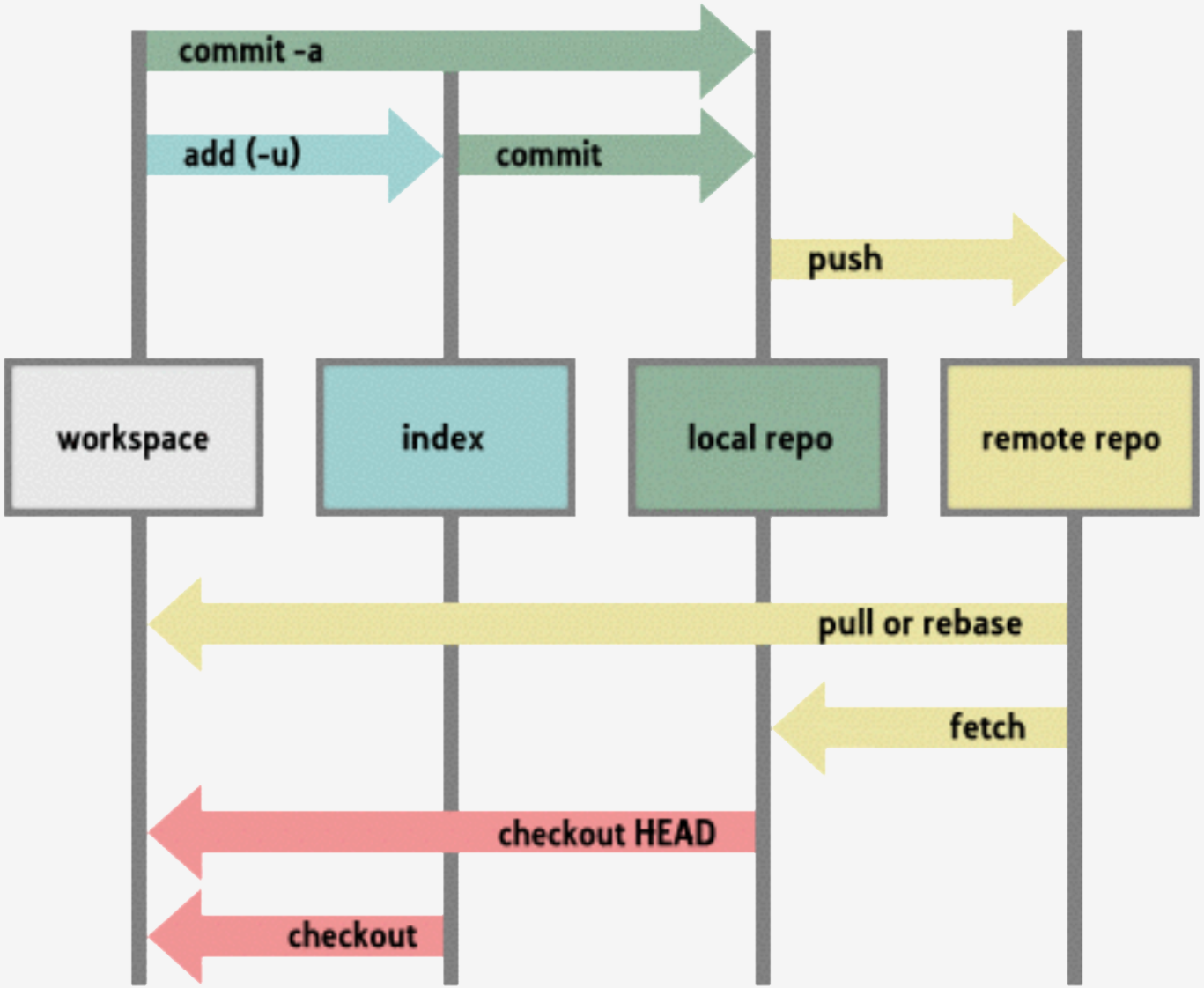
- O repositório criado no GitHUB é simplesmente uma repositório remoto. Para utiliza-lo no seu repositório local, você precisa configurá-lo através do comando “git remote”

```
$ git remote add origin <remote_url>
```

- A url remota deve ser de acordo com o serviço que está utilizando
  - Git Credentials: `https://github.com/<user>/<repo>.git`
  - SSH Key: `git@github.com:<user>/<repo>.git`
- Neste comando, você apenas configurou em seu diretório local a existência de um repositório remoto chamado “origin”
- Para “enviar” o conteúdo para o seu repositório local, utilize o comando “push”

```
$ git push -u origin master
```


Sincronizando o repositório remoto



Tags em repositórios remotos

- Os tags não são enviados para o repositório remoto com o comando “**push**”.

- Para compartilhar uma tag em específico através do repositório remoto, utilize o comando

```
$ git push origin <tag>
```

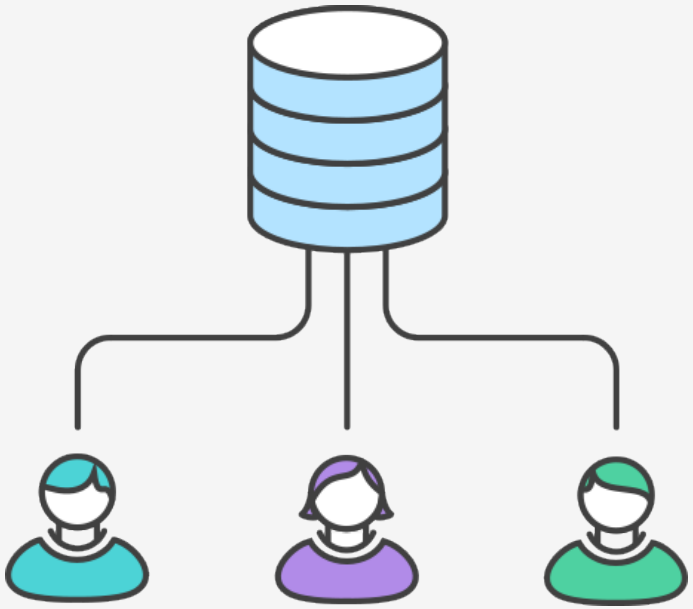
- Caso queira compartilhar todas as tags criadas em seu repositório local, utilize a opção --tags

```
$ git push origin --tags
```


Git Workflows

- Lembre-se, o Git é apenas uma ferramenta
- Diferentes fluxos de trabalho
- https://www.atlassian.com/git/tutorials/comparing-workflows
- Centralized Workflow
- Feature Branch Workflow
- Gitflow Workflow
- Forking Workflow

Repositório Centralizado



- Muito semelhante a um VCS (CVS, SVN), mas ainda assim, com algumas vantagens (eg. Velocidade)
- É um bom workflow para iniciar no Git, e depois avançar para workflows mais avançados.

Repositório Centralizado

- Sempre realize um **pull** do repositório central antes de realizar um **push**
  - Garantir que o seu repositório local está sincronizado com modificações de outros usuários
- Gerenciando conflitos
  - Utilize o parâmetro rebase no comando pull

```
$ git pull --rebase
```

  - Facilita a resolução de conflitos (caso ocorram)
  - Continuando o rebase

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

  - Abortando o rebase

```
$ git rebase --abort
```

Conclusão

Conclusão

- **Git** é um software para controle de versão

Conclusão

- **Git** é um software para controle de versão
- **Github** é uma comunidade online onde você pode colaborar com outras pessoas em projetos utilizando o Git

Conclusão

- **Git** é um software para controle de versão
- **Github** é uma comunidade online onde você pode colaborar com outras pessoas em projetos utilizando o Git
- Um **repositório** é um diretório onde você armazena todos os arquivos que vc deseja rastrear as modificações

Conclusão

- **Git** é um software para controle de versão
- **Github** é uma comunidade online onde você pode colaborar com outras pessoas em projetos utilizando o Git
- Um **repositório** é um diretório onde você armazena todos os arquivos que vc deseja rastrear as modificações
- **Local** refere se a versão do repositório que está em seu computador / local de desenvolvimento

Conclusão

- **Git** é um software para controle de versão
- **Github** é uma comunidade online onde você pode colaborar com outras pessoas em projetos utilizando o Git
- Um **repositório** é um diretório onde você armazena todos os arquivos que vc deseja rastrear as modificações
- **Local** refere se a versão do repositório que está em seu computador / local de desenvolvimento
- **Clonar** um repositório significa copiar um repositório remoto (e.g. GitHUB) para um repositório local

Conclusão

- **Git** é um software para controle de versão
- **Github** é uma comunidade online onde você pode colaborar com outras pessoas em projetos utilizando o Git
- Um **repositório** é um diretório onde você armazena todos os arquivos que vc deseja rastrear as modificações
- **Local** refere se a versão do repositório que está em seu computador / local de desenvolvimento
- **Clonar** um repositório significa copiar um repositório remoto (e.g. GitHUB) para um repositório local
- **origin** refere-se a versão do repositório remoto que deu origem ao repositório local

Conclusão

- Um ramo (**branch**) é o nome dado a uma linha separada de desenvolvimento de um repositório, que possui o seu próprio histórico

Conclusão

- Um ramo (**branch**) é o nome dado a uma linha separada de desenvolvimento de um repositório, que possui o seu próprio histórico

- Um **commit** é um objeto que armazena informações sobre um conjunto de modificações no repositório. Os commits são feitos em um determinado branch

Conclusão

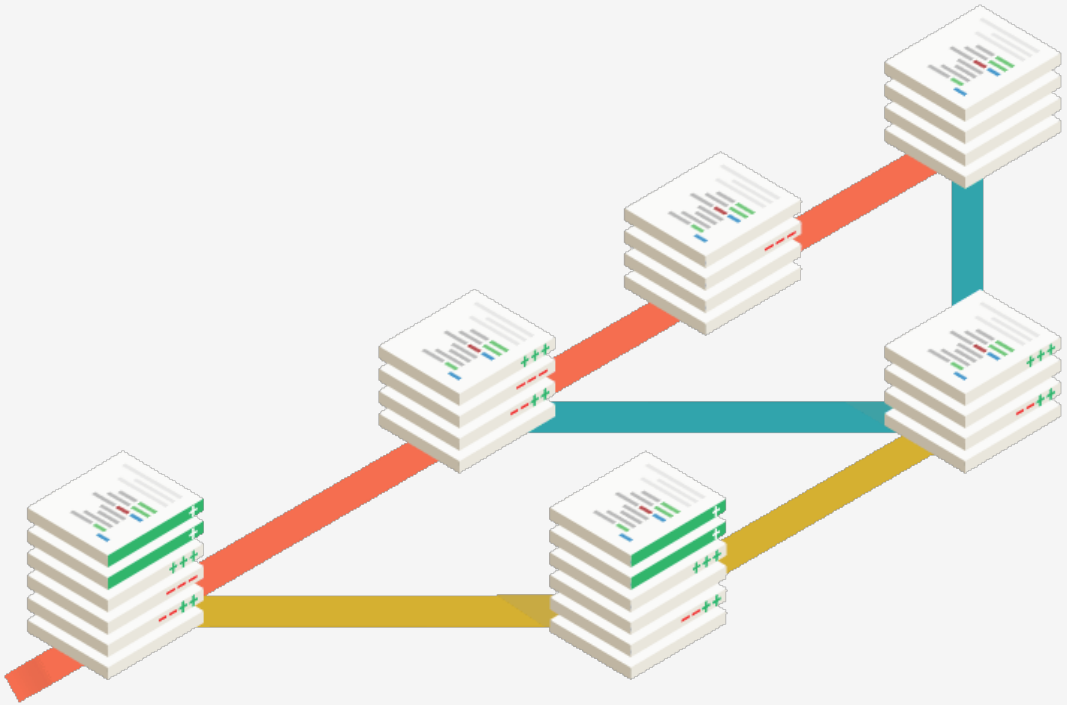
- Um ramo (**branch**) é o nome dado a uma linha separada de desenvolvimento de um repositório, que possui o seu próprio histórico

- Um **commit** é um objeto que armazena informações sobre um conjunto de modificações no repositório. Os commits são feitos em um determinado branch

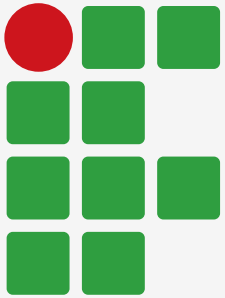
- **HEAD** refere-se ao commit (geralmente o mais recente) para qual o diretório local do repositório aponta no branch atual

Links Úteis

- Documentação Oficial
- ProGit — Livro
- GIT Immersion
- Tutoriais — Atlassian
- SourceTree — GUI para o Git (Windows e Mac)
- Git Cheat Sheet



That’s all folks!



INSTITUTO FEDERAL
Santa Catarina

Câmpus
Florianópolis