

INSTITUTO FEDERAL
SANTA CATARINA

Controle de versão utilizando o GIT

Hugo Marcondes

hugo.marcondes@ifsc.edu.br



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

- 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 ?

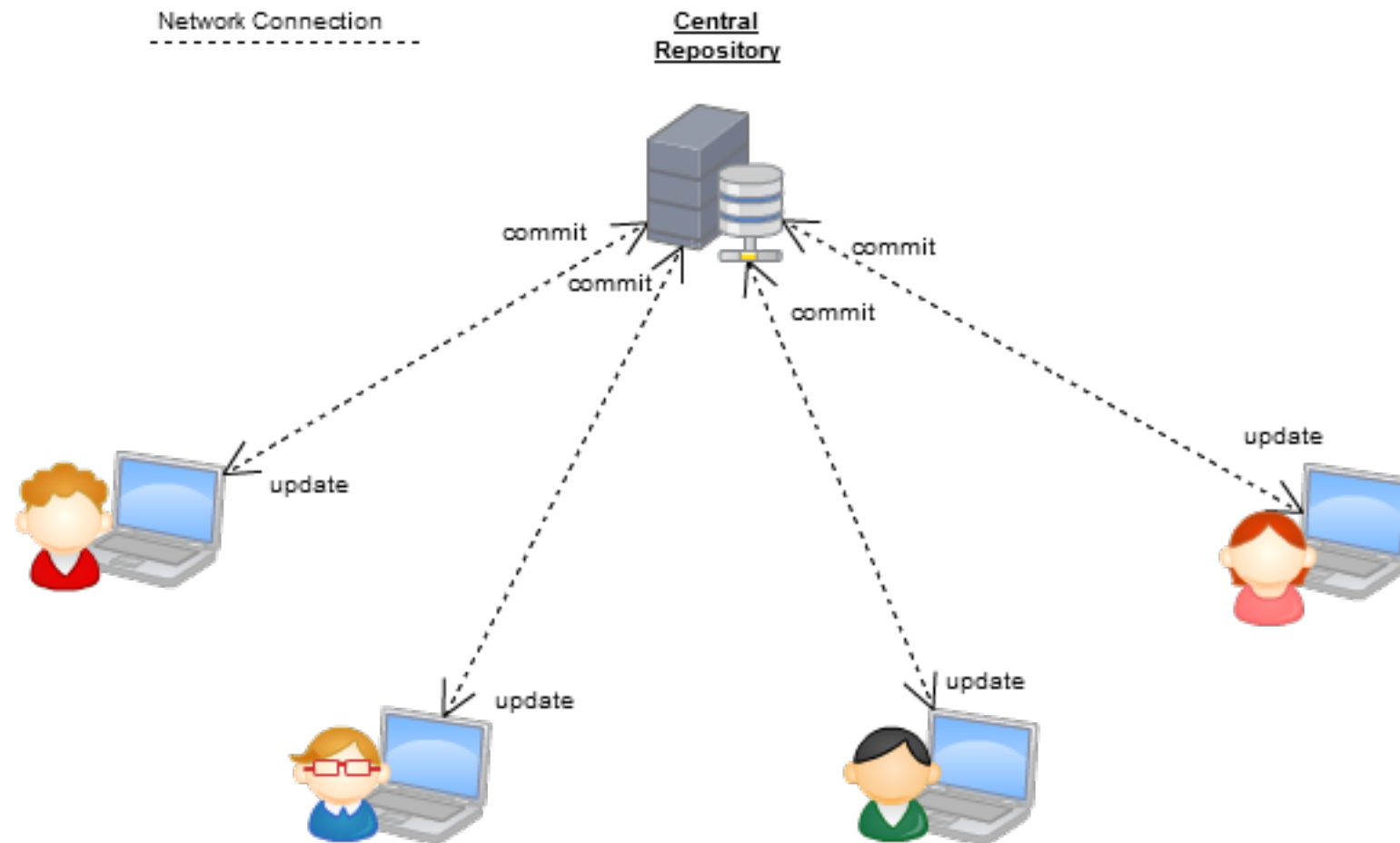


INSTITUTO FEDERAL
SANTA CATARINA

- Você tem arquivos assim ?

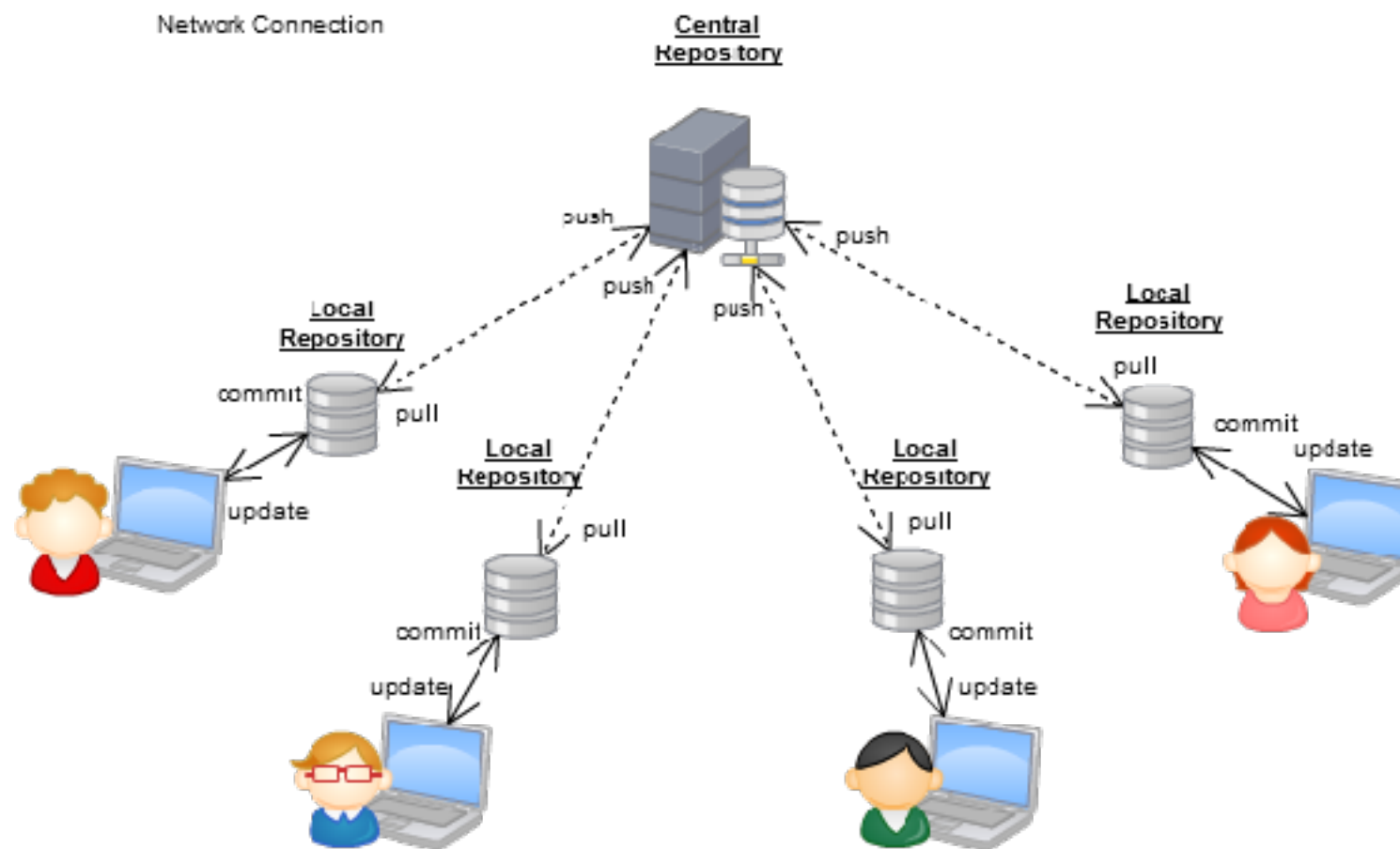
```
ap - bash - 90x9
100% Zoom Add Slide Play Keynote Live Table Chart Text Shape Media Comment Collaborate
[Hugos-MacBook-Pro:ap hugom$ ls
projeto_poo_20170412.tgz      projeto_poo_funcionou.tgz    projeto_poo_v01.tgz
projeto_poo_bug_corrigido.lgz projeto_poo_ultima-versao.lgz projeto_poo_v02.lgz
Hugos-MacBook-Pro:ap hugom$
```

- Centralizados



Exemplos: CVS, Subversion (SVN), Perforce

- Distribuídos



Exemplos: GIT, Mercurial

- 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 "branches" paralelos)
 - Totalmente distribuído
 - Ser escalável para projetos grandes, como o kernel do Linux

- 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

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


The screenshot shows the Git website homepage. At the top, the Git logo is followed by the tagline "—fast-version-control". A search bar is on the right. The main text describes Git as a free and open source distributed version control system. Below this, it mentions Git is easy to learn and has a tiny footprint with lightning fast performance. A diagram on the right shows a branching model with stacks of code and arrows indicating commits and branches. At the bottom, there are four sections: "About", "Documentation", "Downloads", and "Community". On the right side, there is a section for the latest source release, 2.14.1, with a button for "Downloads for Mac".

git —fast-version-control

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint** with **lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.

 **Learn Git in your browser for free with Try Git.**

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.14.1
Release Notes (2017-08-04)
[Downloads for Mac](#)

The screenshot displays the SourceTree application window for a repository named 'ccf-store (Git)'. The interface is divided into several panels:

- Left Panel:** Contains navigation sections for 'FILE STATUS' (Working Copy), 'BRANCHES' (develop, feature, master), 'TAGS', 'REPOSITORIES' (origin, develop, feature-cart-style, front-end-setup, HEAD, feature, master, heroku), 'STASHES', 'SUBMODULES', and 'SOURCES'.
- Commit History (Top Right):** A table listing recent commits with columns for Graph, Description, Commit ID, Author, and Date. The selected commit is 'Added yaml files that provided structured pricing data for the pricing algorithm to utilize' by Sarah Anderson, dated June 11, 2014.
- Commit Details (Bottom Left):** Provides information for the selected commit, including the commit ID (f804c3e), parent ID (2a321e4), author (Sarah Anderson), and date (June 11, 2014). It also shows the commit message: 'Added yaml files that provided structured pricing data for the pricing algorithm to utilize'.
- File Diff (Bottom Right):** Shows the changes in the file 'spruce_extended_options/config/drawer_option_types.yml'. The diff highlights additions (green) and deletions (red) in the file's content, which includes a list of options like 'Main Options', 'Spec Material Thickness', 'Bottom Material', etc.

Comandos Básicos do Git





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

- 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**

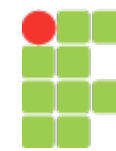
O que fizemos ?



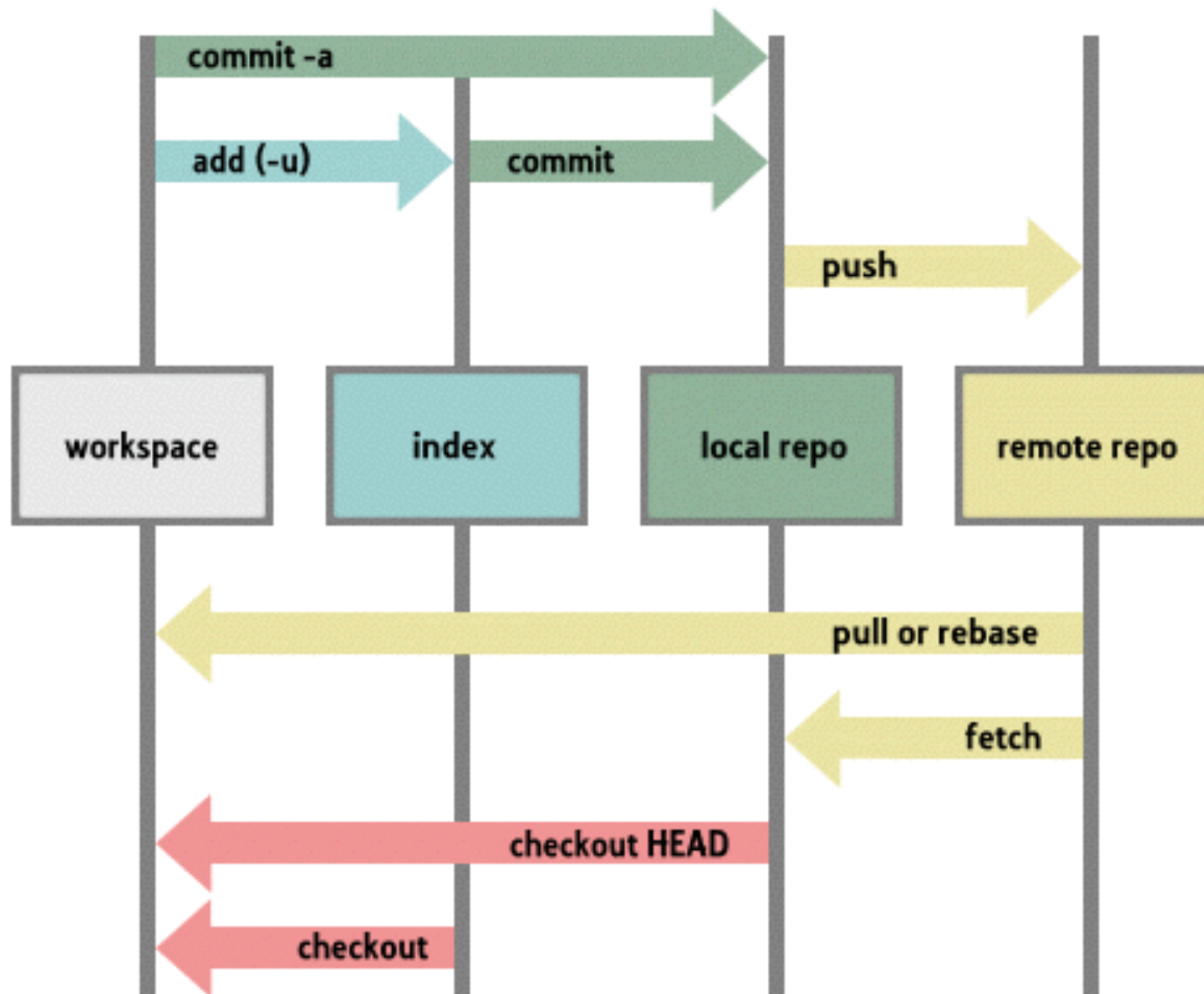
- Quando adicionamos (**add**) um novo arquivo, dizemos ao Git para iniciar o monitoramento das modificações naquele arquivo.
- Neste estágio, o arquivo é chamado de "**staging** file". Um snapshot das nossas modificações do arquivo estão na área conhecida como "**staging area**" (também conhecido como **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.

- 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 `--online`)

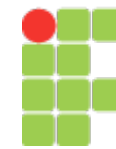
Os “domínios” do Git



INSTITUTO FEDERAL
SANTA CATARINA



Para que devemos usar controle de versão ?



INSTITUTO FEDERAL
SANTA CATARINA

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



- 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 .gitignore na raiz do diretório do seu projeto.
 - Padrões de nomes de arquivos em cada linha
- <https://gitignore.io> - Sugestões de arquivos .gitignore

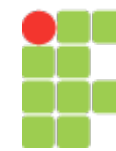
- Como desfazer mudanças que não deram certo ?
- O processo irá depender do estágio em que a sua modificação está
 - Se você ainda não adicionou a sua modificação no staging, basta usar o comando "`git checkout`"
 - Se o arquivo já está no staging, utilize o comando "`git reset`"
 - Remove o arquivo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

Branching e Merging

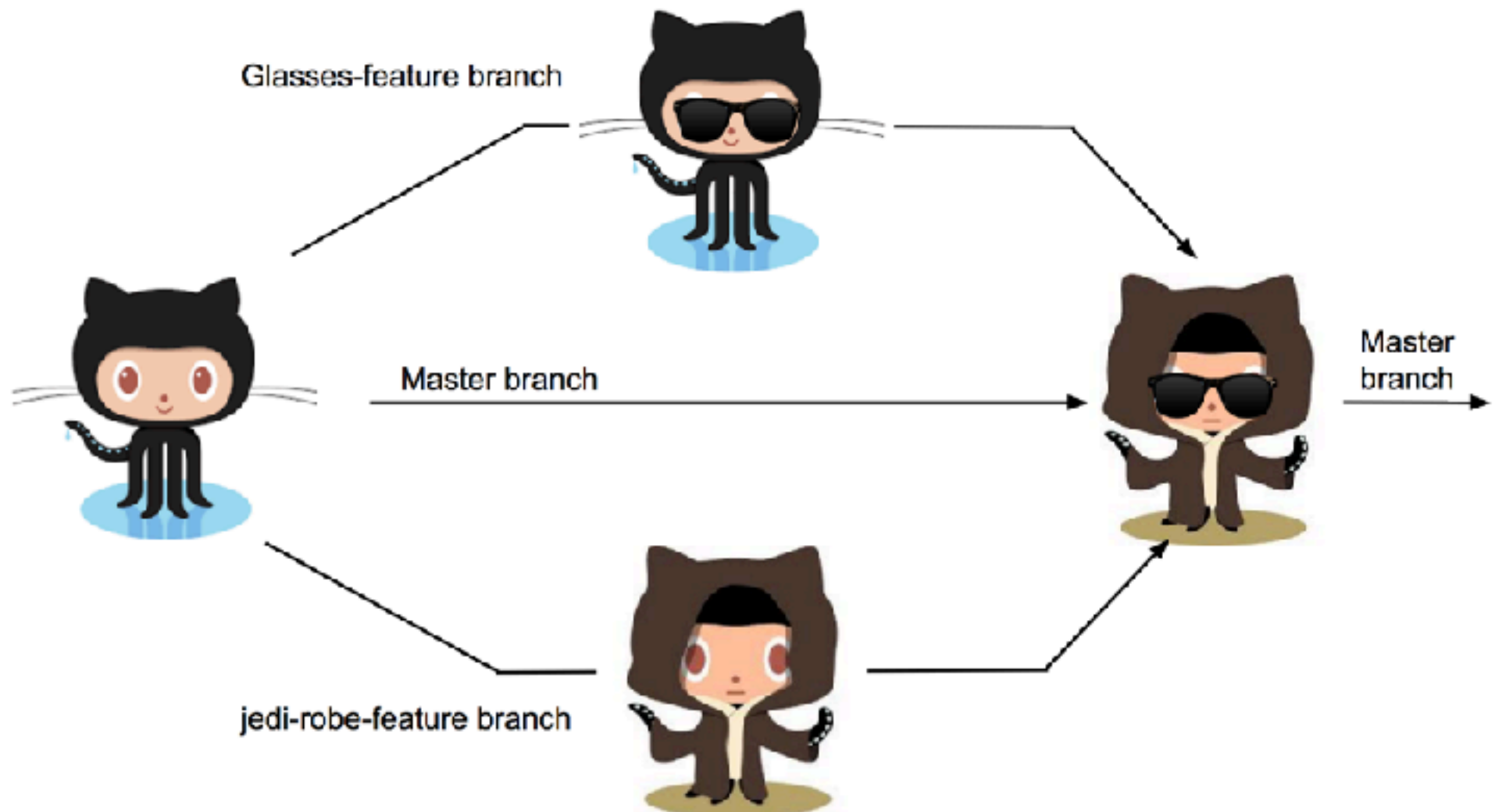


- Um ramo (**branch**) é essencialmente uma outra cópia do seu repositório em que você pode isolar as modificações feitas, e manter a cópia original sem modificações.
- Depois você pode decidir em combinar (**merge**) essas modificações totalmente ou em parte em sua cópia original (**master**), ou simplesmente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

Branching



INSTITUTO FEDERAL
SANTA CATARINA

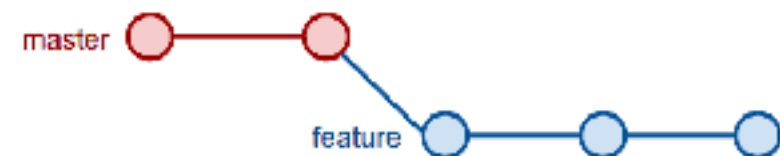


Credit to Jon Gilbert & DBC for the idea of this slide

- 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**

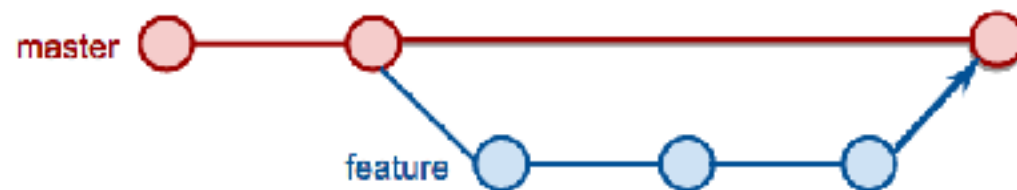
```
$ git checkout -b <branch_name>
```

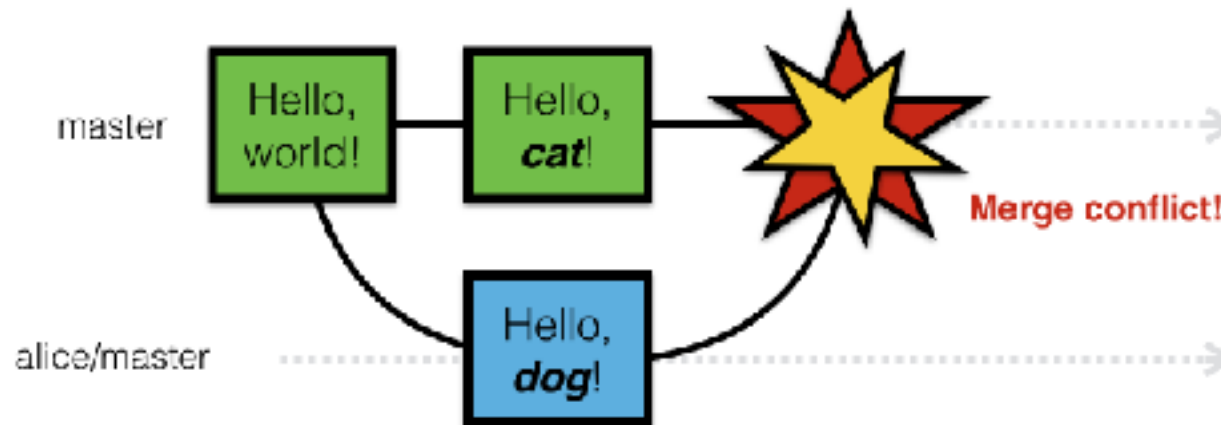
- O comando **branch** é utilizado listar e remover os ramos



- 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>
```





- 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.

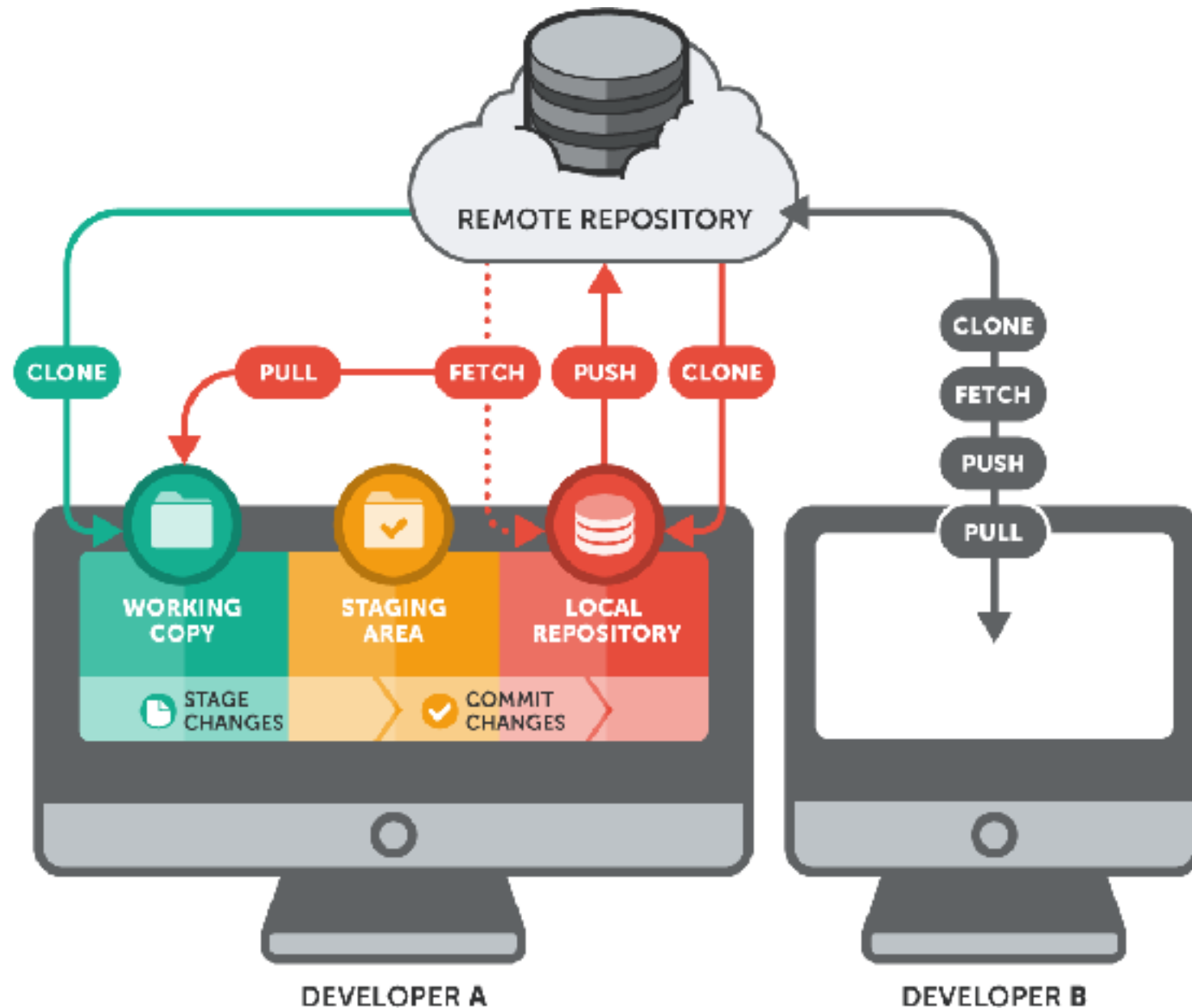
- Não use ramos, quando você deseja apenas nomear um determinado commit do seu repositório (i.e, não haverá desenvolvimento naquele ramo)
- As tags podem ser criadas com o comando tag

```
$ 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



INSTITUTO FEDERAL
SANTA CATARINA





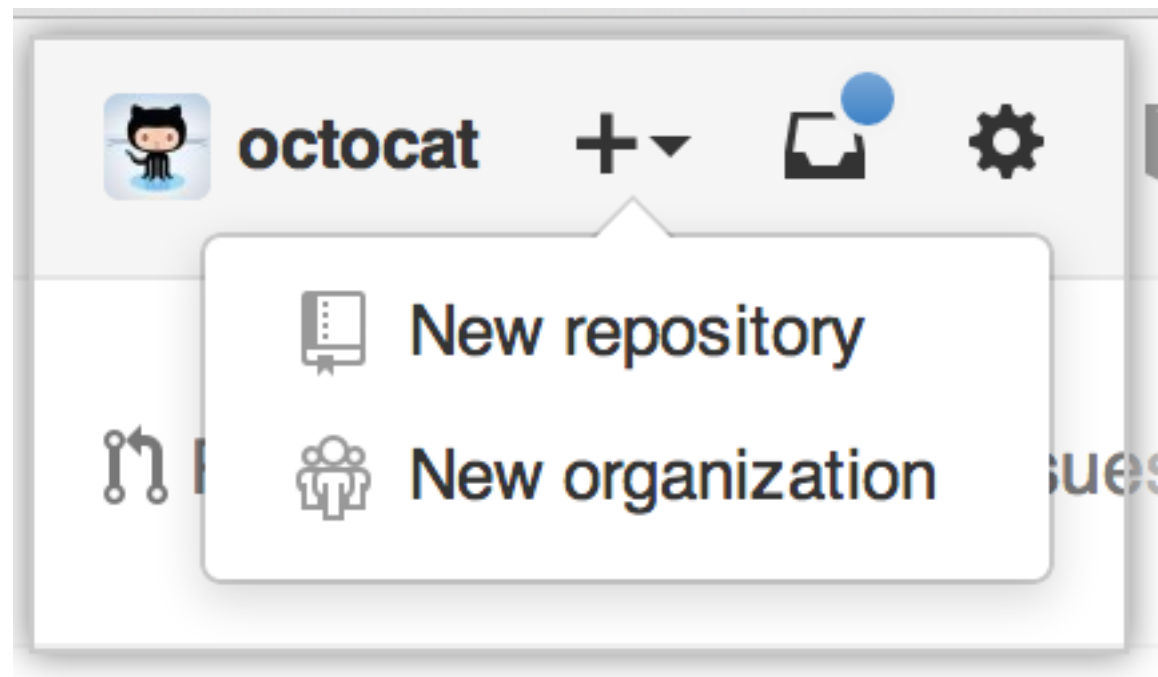
- Serviço online que oferece a hospedagem de repositórios baseados em Git
- Mais de 20 milhões de usuários, compartilhando mais de 57 milhões de repositórios !
- Grande integração com web
 - Possibilidade de acesso ao repositório através do browser
 - É possível até editar arquivos através do página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



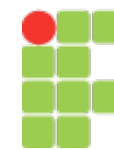
O que fazer no GitHub ?

- 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 !**

- 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




INSTITUTO FEDERAL
SANTA CATARINA


Owner **Repository name**

PRIVATE   **github** /

Great repository names are short and memorable. Need inspiration? How about **flaming-tyrion**.

Description (optional)

☐  **Public**
Anyone can see this repository. You choose who can commit.

☒  **Private**
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately.

Add .gitignore: **None**

Create repository

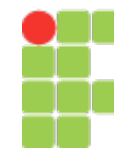
- O repositório criado no GitHub é simplesmente uma repositório remoto. Para utiliza-lo no seu repositório local, você precisa configurá-lo com os “git remote”
- No caso do GitHub, utilize os seguintes comandos

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

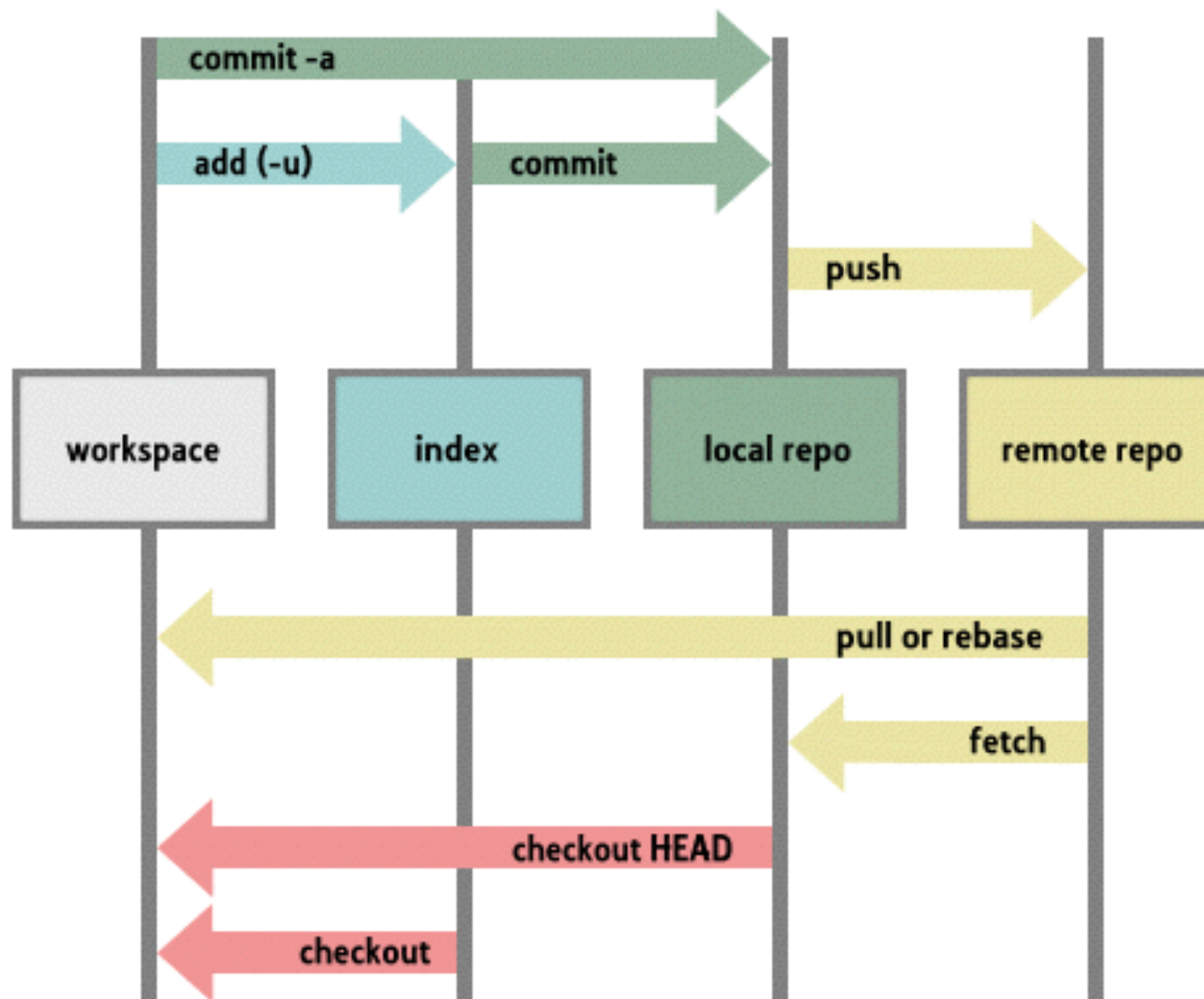
<remote_url> = <https://github.com/<user>/<repo>.git>
- Neste comando, você apenas configurou em seu diretório local a existência de um repositório remoto chamado “origin”. Para “enviar” o conteúdo para o seu repositório local, utilize o comando “push”

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

Sincronizando o repositório remoto



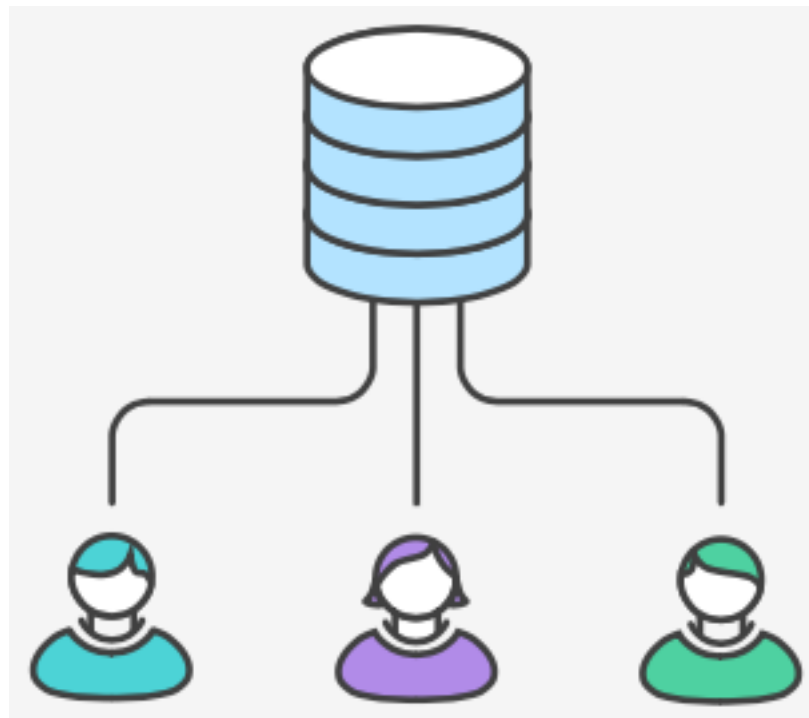
INSTITUTO FEDERAL
SANTA CATARINA



- 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`

- 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



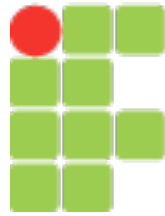
- Muito semelhante a um VCS (CVS, SVN), mas ainda assim, com algumas vantagens (eg. Velocidade)
- É um bom workflow para iniciar no Git, e depois avançar para workflows mais avançados.

- 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 para trazer possíveis modificações no origin/master, antes de realizar o push
 - Facilita a resolução de conflitos (caso ocorram)
 - **\$ git rebase --continue** : para continuar o rebase após resolver o conflito
 - **\$ git rebase --abort** : abortar a operação de rebase

- **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

- 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

- [Documentação Oficial](#)
- [Git Cheat Sheet](#)
- [ProGit - Livro](#)
- [Tutoriais - Atlassian](#)
- [SourceTree - GUI para o Git \(Windows e Mac\)](#)
- [Tutoriais - Atlassian - Workflows](#)
- [GIT Immersion](#)



INSTITUTO FEDERAL
SANTA CATARINA

Controle de versão utilizando o GIT

Hugo Marcondes

hugo.marcondes@ifsc.edu.br

