

# Uma Suave Introdução ao Git



Parte dos slides foram  
cedidos pelo Leonardo Murta

# Software



programa.py

# Evolução










programa.py

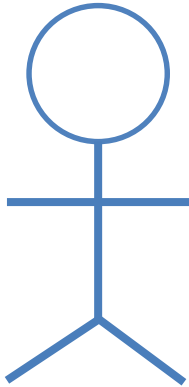


programa.v2.py

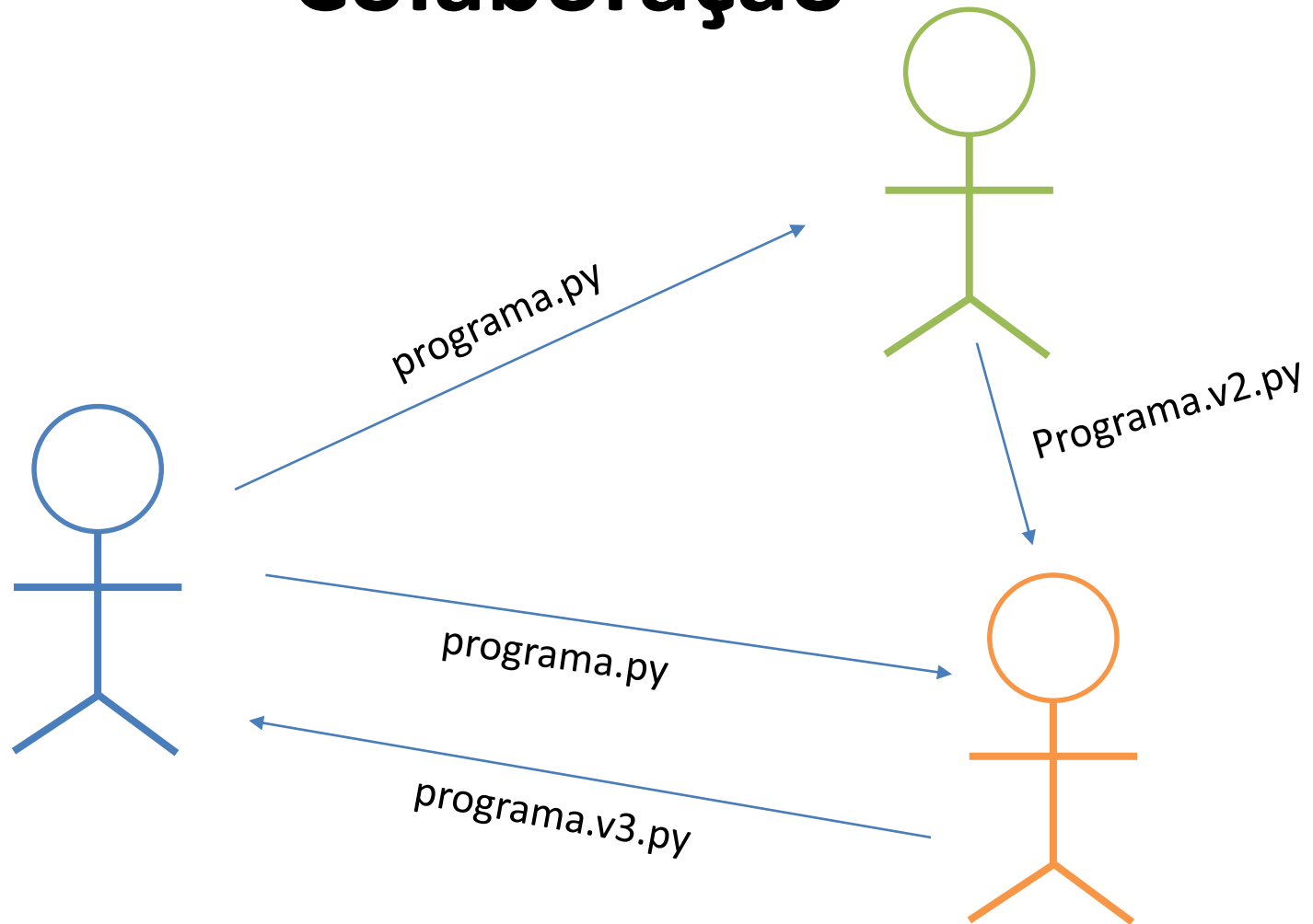
# Evolução

-  programa.py
-  programa.v2.py
-  programa.v3.alternativo.py
-  programa.v3.py
-  programa.v4.final.de.verdade.py
-  programa.v4.final.py
-  programa.v4.final2.py

# Colaboração

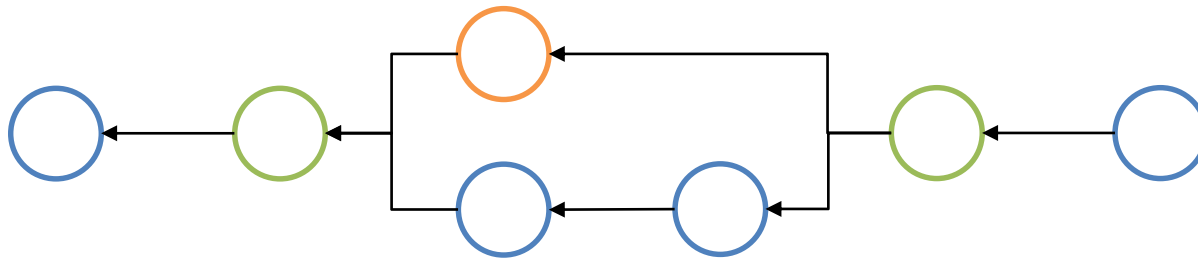


# Colaboração



# Sistema de Controle de Versão

- É necessário controlar versões de artefatos de software
  - Evolução
  - Colaboração



- Git

# Git vs GitHub





# Git – Instalação

- Linux (debian/ubuntu)
  - \$ sudo apt install git
- Windows
  - <https://gitforwindows.org/>
- Mac
  - \$ brew install git
  - <https://sourceforge.net/projects/git-osx-installer/>
  - XCode

# Conceitos básicos: *help*!

- `git help`
  - Oferece ajuda geral sobre o git
- `git help <comando>`
  - Oferece ajuda sobre um comando específico do git
- Demais comandos dão dicas do que pode ser feito (leia com atenção as saídas dos comandos!)



# \$ git help help

## git-help(1) Manual Page

### NAME

git-help - Display help information about Git

### SYNOPSIS

```
git help [-a | --all [--no-verbose]] [-g | --guide]
        [-i | --info | -m | --man | -w | --web] [COMMAND | GUIDE]
```

### DESCRIPTION

With no options and no COMMAND or GUIDE given, the synopsis of the *git* command and a list of the most commonly used Git commands are printed on the standard output.

If the option `--all` or `-a` is given, all available commands are printed on the standard output.

If the option `--guide` or `-g` is given, a list of the useful Git guides is also printed on the standard output.

If a command, or a guide, is given, a manual page for that command or guide is brought up. The *man* program is used by default for this purpose, but this can be overridden by other options or configuration variables.

If an alias is given, git shows the definition of the alias on standard output. To get the manual page for the aliased command, use `git COMMAND --help`.

Note that `git --help ...` is identical to `git help ...` because the former is internally converted into the latter.

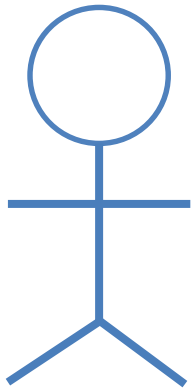
To display the `git(1)` man page, use `git help git`.

This page can be displayed with `git help help` or `git help --help`



# Git – Configuração

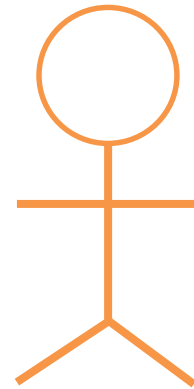
- `$ git config --global user.name <seu nome>`
  - Configura o nome do usuário
- `$ git config --global user.email <seu email>`
  - Configura o email do usuário



joao <joao@email.com>



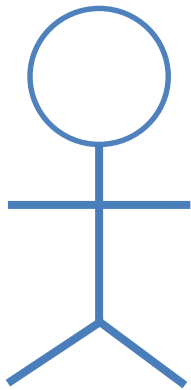
maria <maria@email.com>



pedro <pedro@email.com>

# Git – Configuração

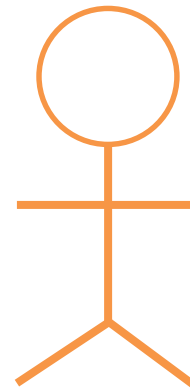
- `$ git config --global user.name <seu nome>`
  - Configura o nome do usuário
- `$ git config --global user.email <seu email>`
  - Configura o email do usuário



joao <joao@email.com>



maria <maria@email.com>



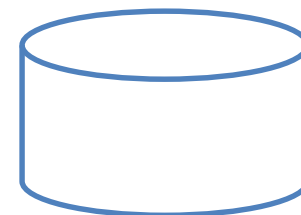
pedro <pedro@email.com>

# Hierarquia de Configurações

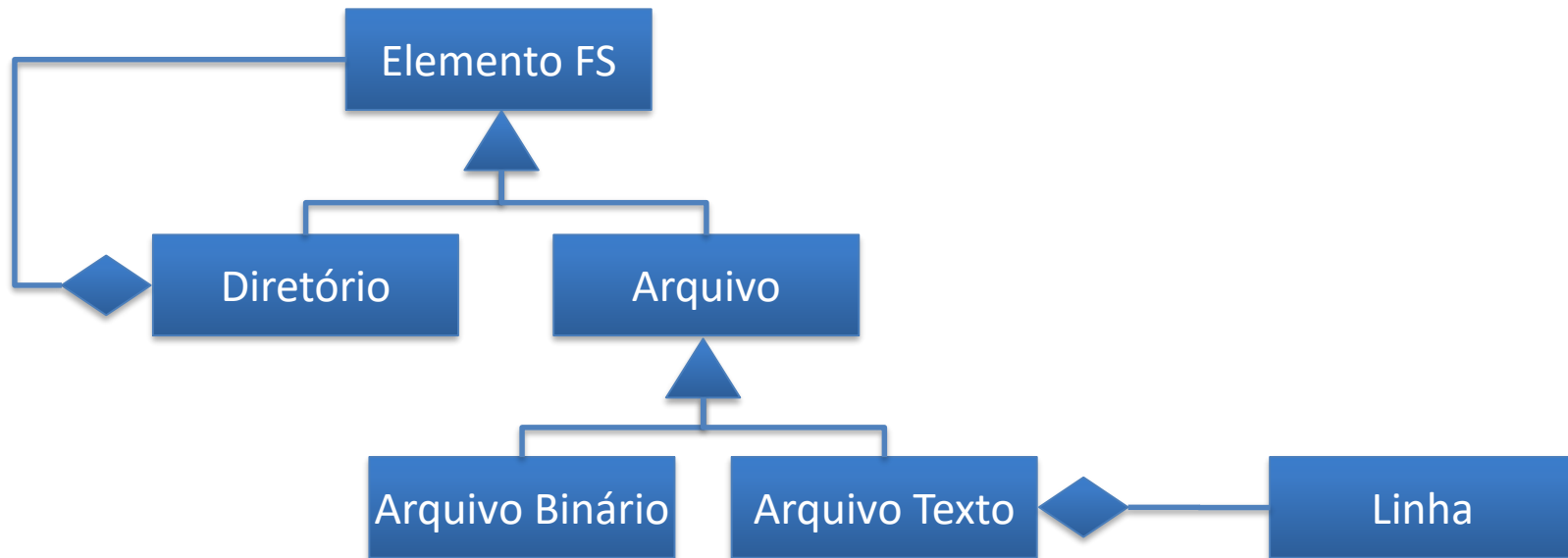
- **--system**
  - Configuração para o sistema inteiro
- **--global**
  - **Configuração para o usuário logado**
- **--local**
  - Configuração para o repositório do projeto

# Repositório do Git

- Local onde ficam as versões e histórico dos artefatos monitorados do projeto



# O que são artefatos?







# \$ git init <nome>

- Cria um repositório Git no diretório

```
joao@DESKTOP-N2MKCOJ MINGW64 ~
```

```
$ git init suaveintro
```

```
Initialized empty Git repository in C:/Users/joao/suaveintro/.git/
```

↑  
Repositório



# \$ vi suaveintro/programa.py

```
print("hello")
```

programa.py



# \$ git status

```
$ git status  
On branch master
```

```
No commits yet
```

```
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    programa.py
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

- Esse programa ainda não está no repositório!
- Está apenas no espaço de trabalho (não rastreado)
- Ainda não existe uma versão atribuída a ele no git

# Mas afinal, o que são versões?

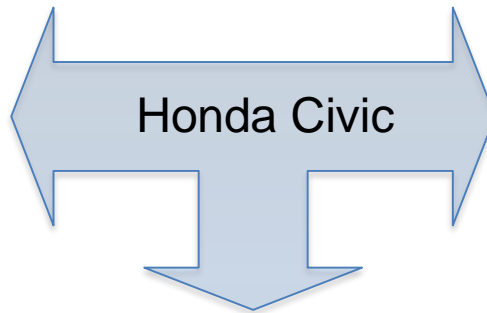
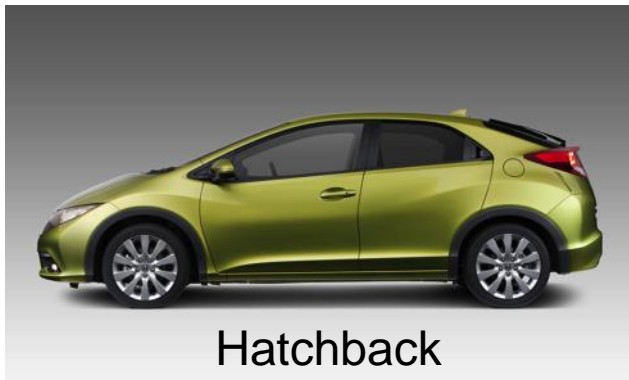


(Conradi and Westfechtel 1998)

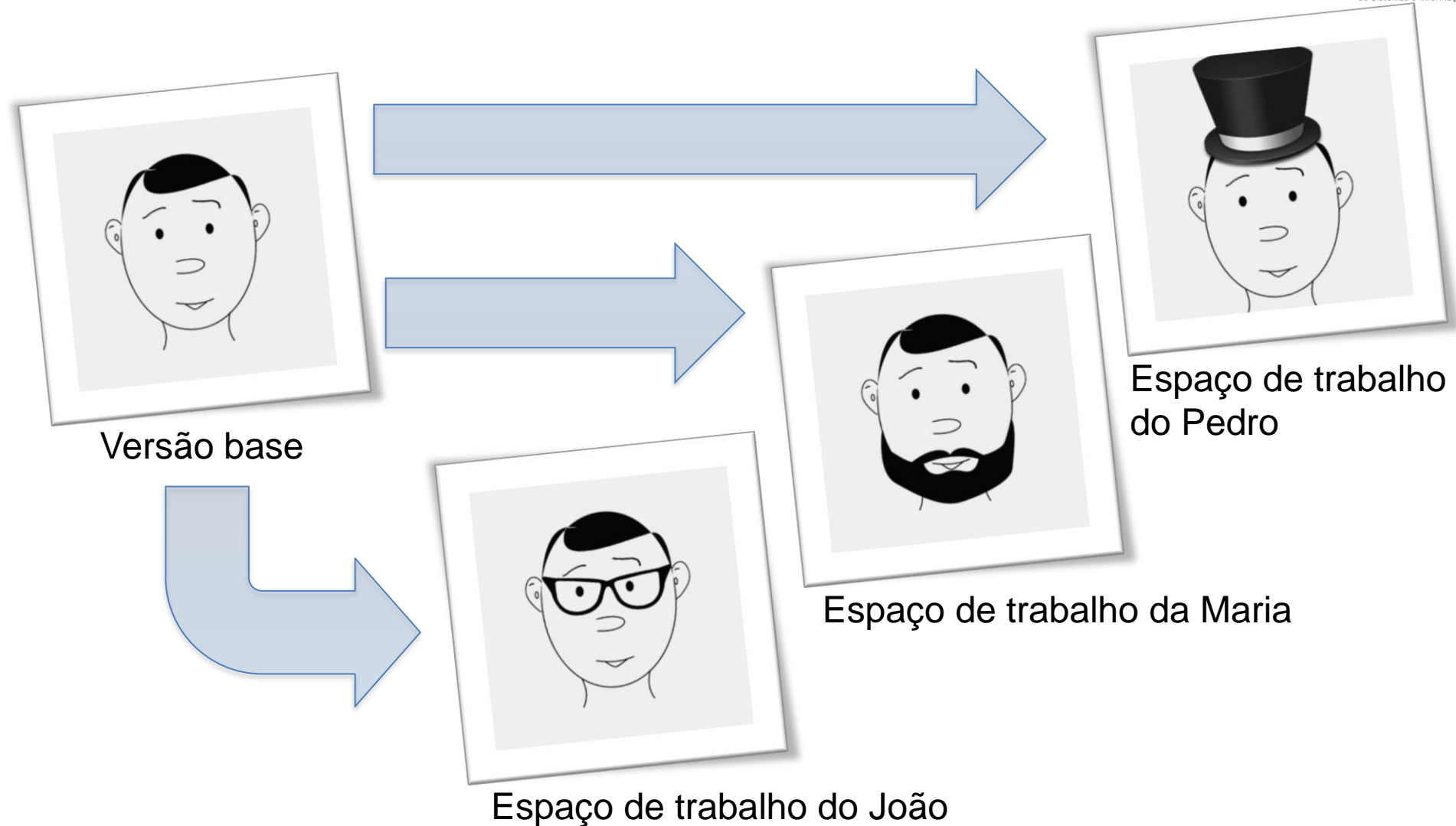
# Revisões



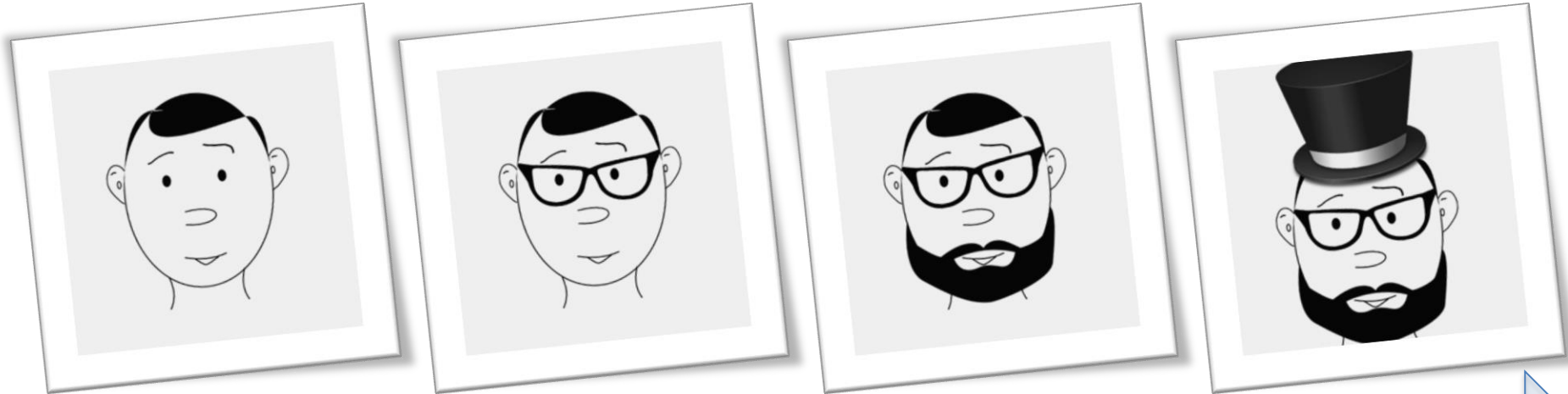
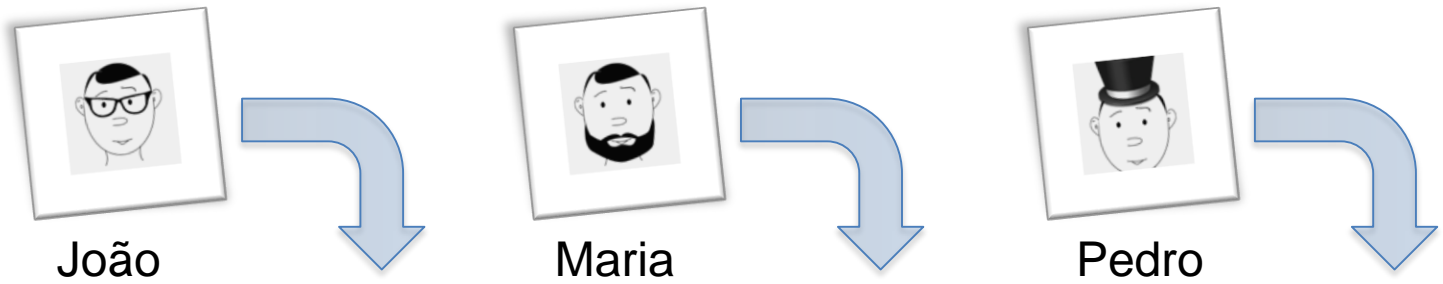
# Variantes



# Cooperação (versões rascunho)

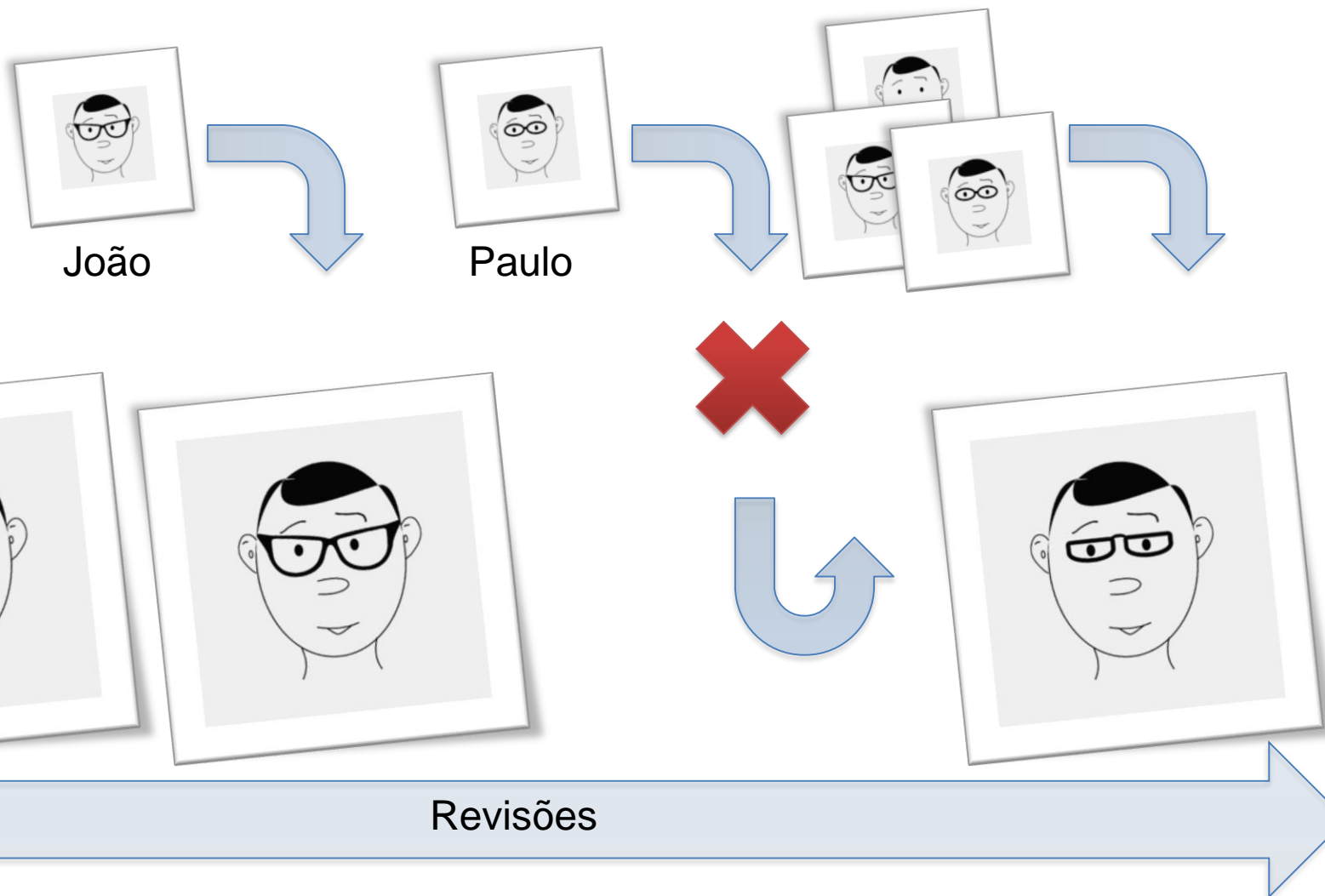


# Versões de rascunho podem ser combinadas (operação de *merge*)

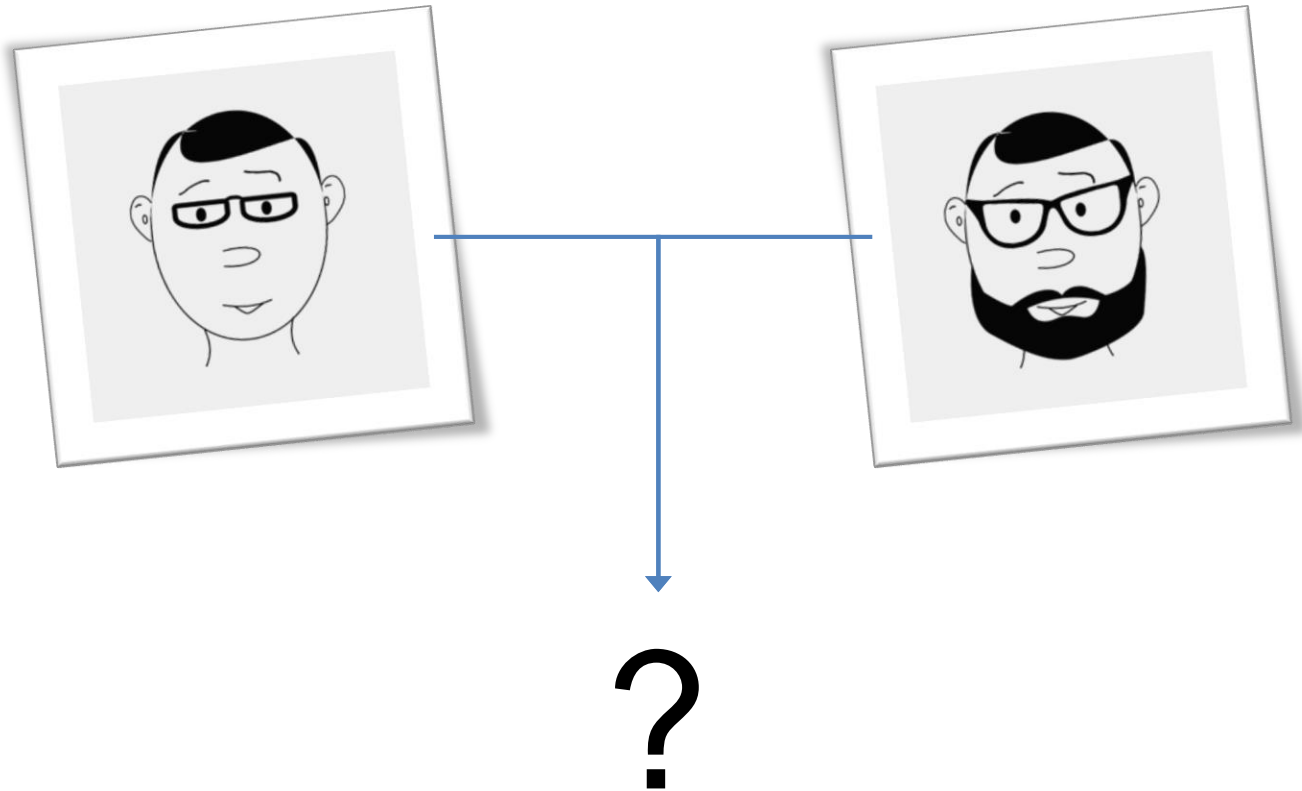




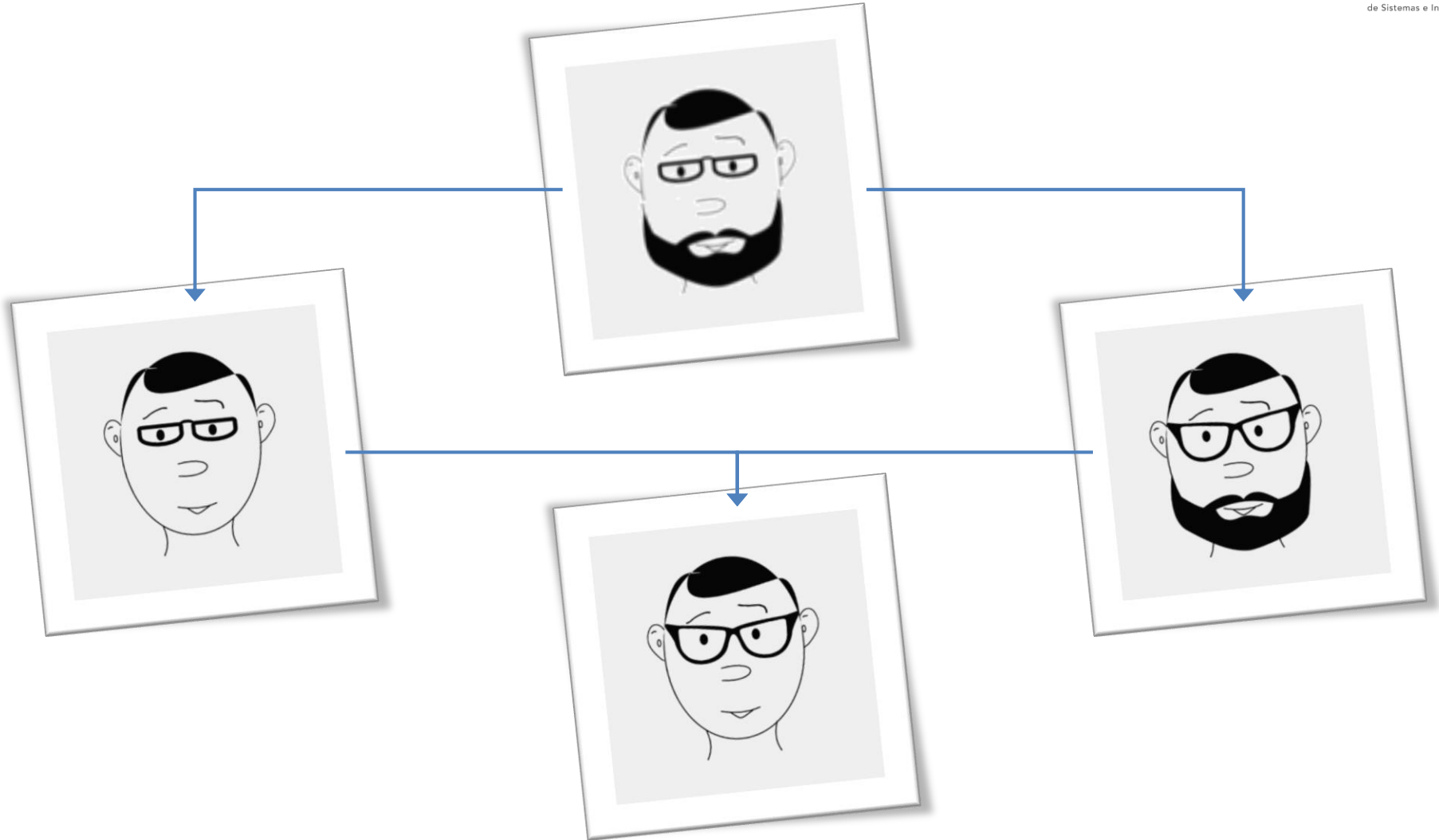
# Conflitos podem ocorrer durante o *merge*



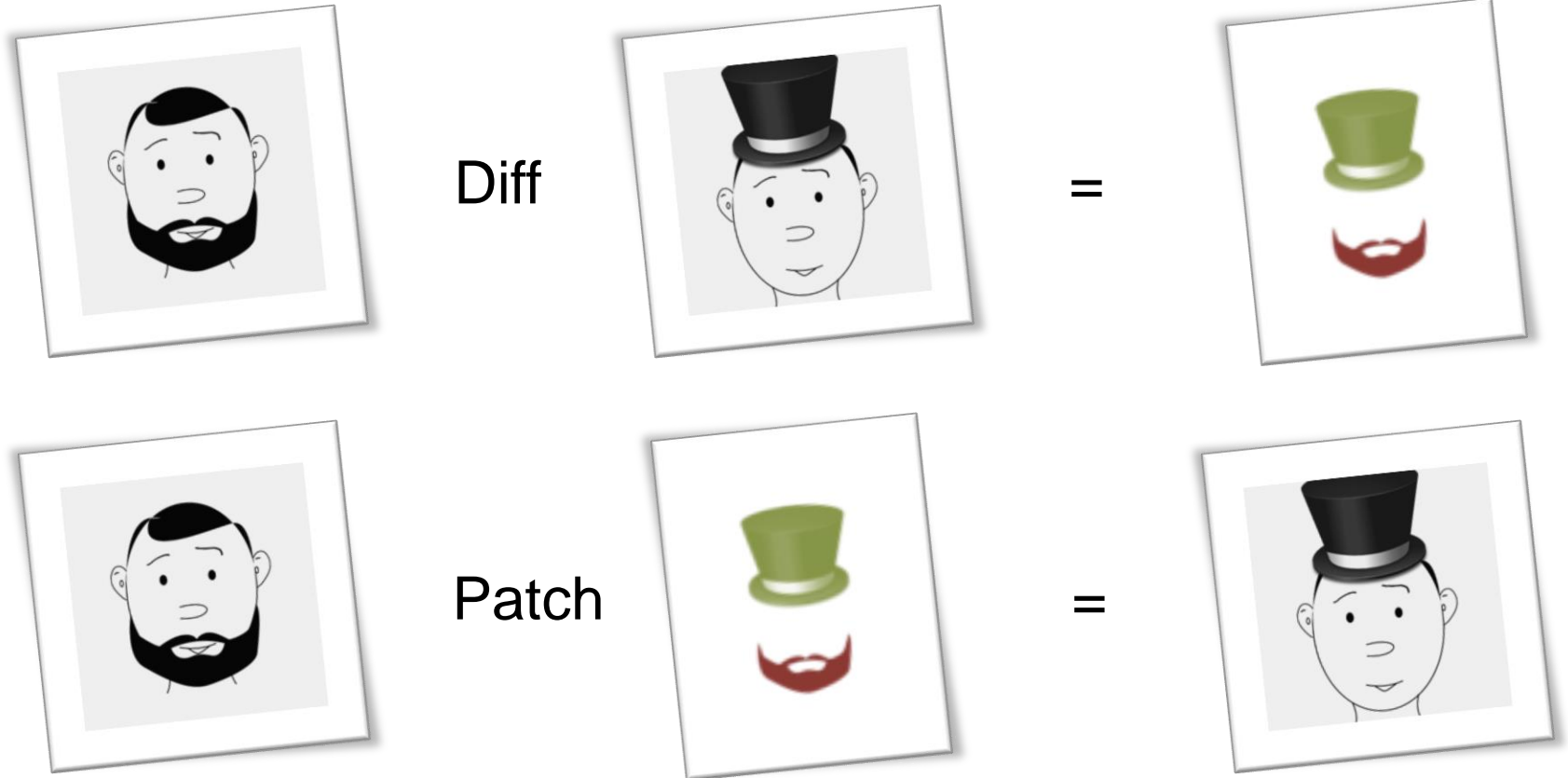
# 2-way merge



# 3-way merge

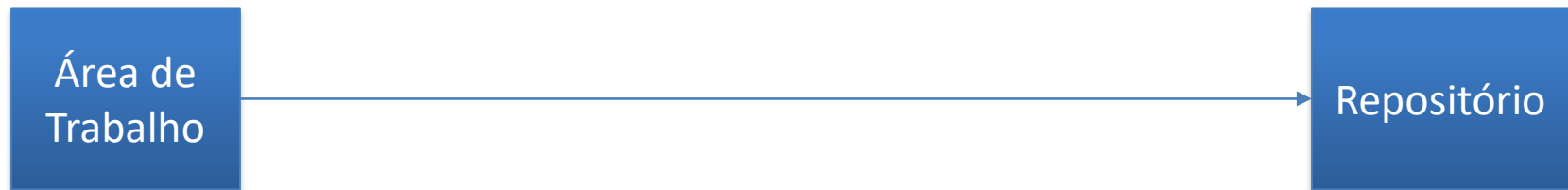


# Outras duas operações importantes...



... para guardar, transferir e compreender versões.

# Voltando pro Repositório



programa.py

# Conceitos básicos: *staging area*

- Área onde são colocados os arquivos que pretendemos enviar para o repositório



programa.py

# Conceitos básicos: *staging area*

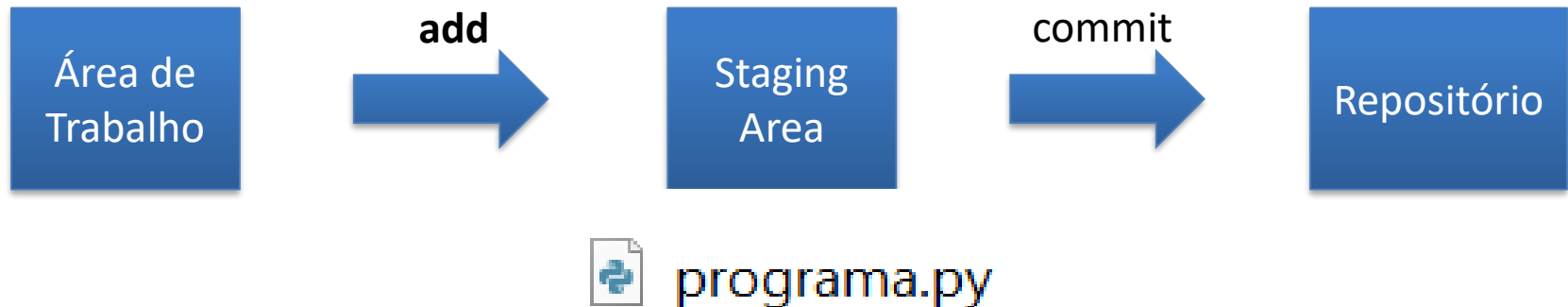
- Área onde são colocados os arquivos que pretendemos enviar para o repositório





# \$ git add <path>

- Adiciona um arquivo na *staging area* para ser enviado ao repositório no próximo *commit*







# \$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: programa.py

- Ainda não está no repositório, mas está pronto para ser enviado



# \$ git commit -m "mensagem"

- Envia os arquivos que estão na *staging area* para o repositório

```
$ git commit -m "adiciona versao inicial"  
[master (root-commit) f9edfd0] adiciona versao inicial  
1 file changed, 1 insertion(+)  
create mode 100644 programa.py
```

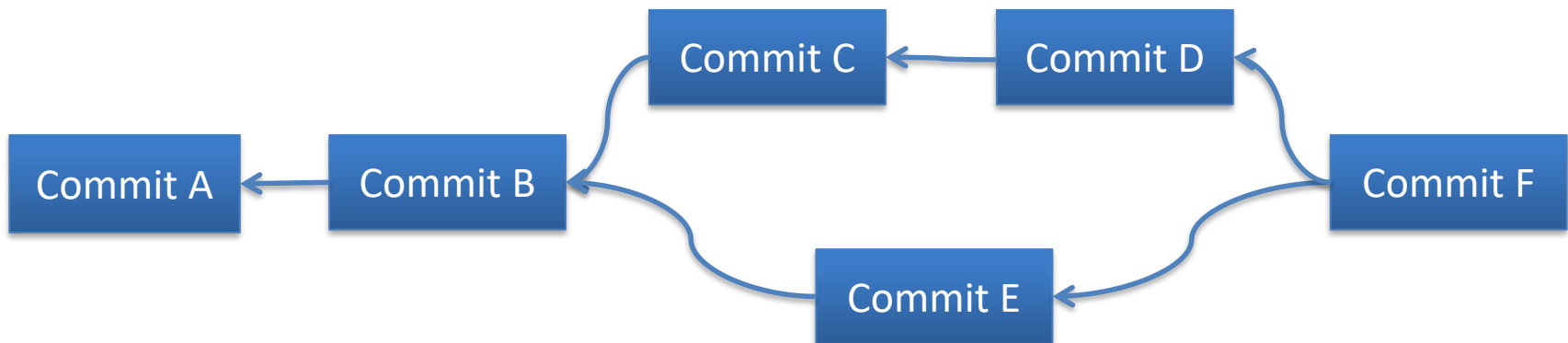
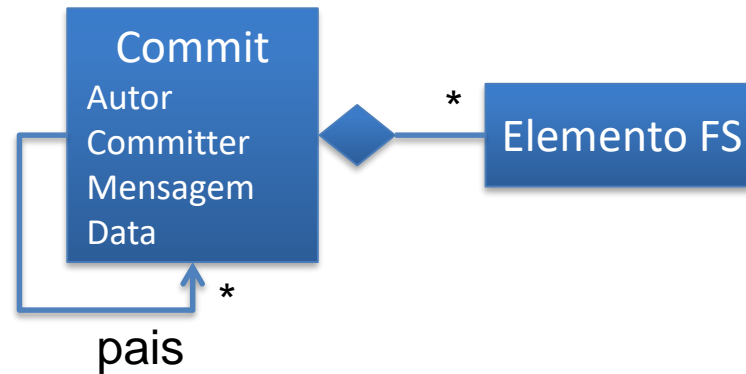


# Conceitos básicos: *commit* id

- Cada sistema de controle de versão usa uma estratégia diferente para identificar *commits*
  - Número sequencial por arquivo (CVS)
  - Número sequencial por repositório (Subversion)
  - Hash (Git e Mercurial)



# Como é versionado?





# \$ git log

- Inspeciona o histórico do repositório local

```
commit f9edfd02fb0dba729bf193a2f1ea21a4c9818758 (HEAD -> master)
Author: Joao Pimentel <joaofelipenp@gmail.com>
Date:   Thu Jun 4 16:38:31 2020 -0300
```

adiciona versao inicial

- \$ git log [--graph] [--decorate=short] [--name-status]



# \$ git show

## ■ Inspecciona um *commit*

```
$ git show f9edfd02
commit f9edfd02fb0dba729bf193a2f1ea21a4c9818758 (HEAD -> master)
Author: Joao Pimentel <joaofelipenp@gmail.com>
Date: Thu Jun 4 16:38:31 2020 -0300
```

adiciona versao inicial

```
diff --git a/programa.py b/programa.py
new file mode 100644
index 0000000..11b15b1
--- /dev/null
+++ b/programa.py
@@ -0,0 +1 @@
+print("hello")
```



# \$ vi suaveintro/programa.py

```
print("hello")  
print("world")
```

programa.py



# \$ git diff

- Compara o espaço de trabalho com a *staging area* ou com alguma versão do repositório

```
$ git diff
warning: LF will be replaced by CRLF in programa.py.
The file will have its original line endings in your working directory
diff --git a/programa.py b/programa.py
index 11b15b1..3ef823c 100644
--- a/programa.py
+++ b/programa.py
@@ -1,2 @@
 print("hello")
+print("world")
```





# Continuando

- \$ git add .
- \$ git commit -m "adiciona palavra world"
- \$ git log --graph

```
$ git log --graph --decorate=short --name-status
* commit 28b9f77dcf9c4e5ff34425a87b8bead4207d4b8d (HEAD -> master)
   Author: Joao Pimentel <joaofelipenp@gmail.com>
   Date:   Thu Jun 4 17:30:35 2020 -0300

       adiciona palavra world

M     programa.py

* commit f9edfd02fb0dba729bf193a2f1ea21a4c9818758
   Author: Joao Pimentel <joaofelipenp@gmail.com>
   Date:   Thu Jun 4 16:38:31 2020 -0300

       adiciona versao inicial

A     programa.py
```

# Interface gráfica

- É possível fazer todos esses passos de forma visual
- Dentre várias ferramentas, vamos praticar com...



# Interface gráfica

The screenshot displays the Git GUI interface for a repository named 'suaveintro'. The top menu bar includes File, Edit, View, Repository, Actions, Tools, and Help. Below the menu is a toolbar with icons for Commit, Pull, Push, Fetch, Branch, Merge, Stash, Discard, Tag, Git-flow, Remote, Terminal, and Explorer. The left sidebar shows the 'WORKSPACE' section with 'File Status', 'History' (selected), and 'Search'. Below this are 'BRANCHES' (showing 'master'), 'TAGS', 'REMOTES', and 'STASHES'. The main area shows the commit history for the 'master' branch, sorted by 'Date Order'. The table below represents the commit history shown in the interface:

Graph	Description	Date	Author	Commit
	adiciona palavra world	4 Jun 2020 17:30	Joao Pimentel <jp>	28b9f77
	adiciona versao inicial	4 Jun 2020 16:38	Joao Pimentel <jp>	f9edfd0

Below the commit history, the 'Diff' view for the file 'programa.py' is shown. It displays the changes in 'Hunk 1 : Lines 1-2':

```

1 1  ...print("hello")
2 2  ...print("world")
  
```

The 'Commit:' section shows the commit hash '28b9f77dcf9c4e5ff34425a87b8bead4207d4b8d' and the parent commit 'f9edfd02fb'. The file 'programa.py' is listed below the diff.

# Conceitos básicos: apelidos

- A versão base do seu espaço de trabalho
  - *HEAD*
- O ramo principal do seu repositório
  - *master*



# Marcando versões especiais

- `$ git tag`
  - Lista os rótulos existentes
- `$ git tag <nome do rótulo> [commit id]`
  - Cria um rótulo sobre um dado commit (HEAD por default)
- `$ git tag -d <nome do rótulo>`
  - Remove um rótulo



# Repositório local com ramos

- `git branch --all -v`
  - Lista os ramos existentes no repositório
- `git branch <nome do ramo>`
  - Cria um ramo à partir da versão indicada no HEAD
- `git branch -d <nome do ramo>`
  - Remove um ramo
- `git checkout <commit id ou nome do ramo>`
  - Troca a versão base do espaço de trabalho



# \$ git merge <ramo>

- Combina um ramo com o ramo corrente

```
$ git merge palavras
Auto-merging programa.py
CONFLICT (content): Merge conflict in programa.py
Automatic merge failed; fix conflicts and then commit the result.
```

```
joao@DESKTOP-N2MKCOJ MINGW64 ~/suaveintro (master|MERGING)
```

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   programa.py
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```



# \$ vi suaveintro/programa.py

```
<<<<<<< HEAD
print("hello world!")
=====
palavras = ["hello", "world"]
print(" ".join(palavras))
>>>>>>> palavras
```

→

```
palavras = ["hello", "world!"]
print(" ".join(palavras))
```

programa.py

- \$ git add .
- \$ git commit -m "Merge entre master e palavras"



# Histórico

File Edit View Repository Actions Tools Help

suaveintro

Commit Pull Push Fetch Branch Merge Stash Discard Tag Git-flow Remote Terminal Explorer

WORKSPACE

File Status

History

Search

BRANCHES

master

palavras

TAGS

v1

v2

REMOTES

STASHES

All Branches Show Remote Branches Date Order

Graph	Description	Date	Author	Commit
	<b>merge entre master e palavras</b>	4 Jun 2020 17:57	Joao Pimentel <joa	9cd3c78
	Adiciona exclamacao	4 Jun 2020 17:50	Joao Pimentel <joa	19f9822
	palavras cria lista de palavras	4 Jun 2020 17:47	Joao Pimentel <joa	987d116
	v2 passa a escrever em uma só linha	4 Jun 2020 17:38	Joao Pimentel <joa	d908724
	v1 adiciona palavra world	4 Jun 2020 17:30	Joao Pimentel <joa	28b9f77
	adiciona versao inicial	4 Jun 2020 16:38	Joao Pimentel <joa	f9edfd0

Sorted by file status

programa.py

Commit:

9cd3c789f7bdd1c54a4c2ea4f35fe490180b0949  
[9cd3c78]  
Parent: 10f08220d8\_087d1168d2

programa.py

Hunk 1 : Lines 1-2

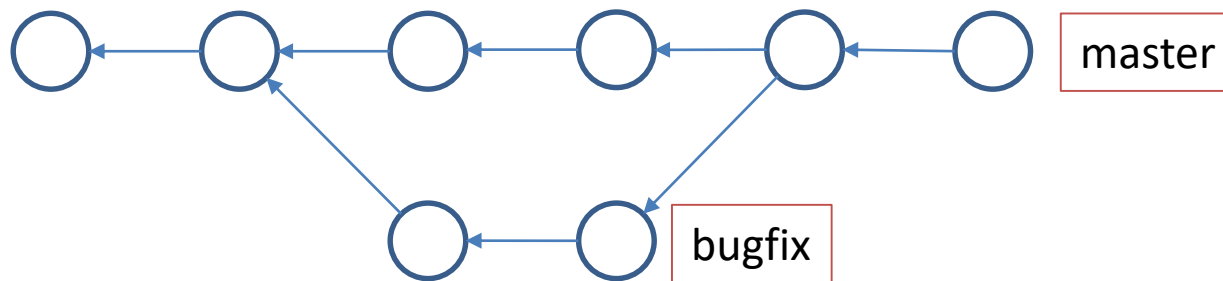
Reverse hunk

```
-- print("hello world!")
+ palavras = ["hello", "world!"]
+ print("-".join(palavras))
```

# Pra que usar ramos?

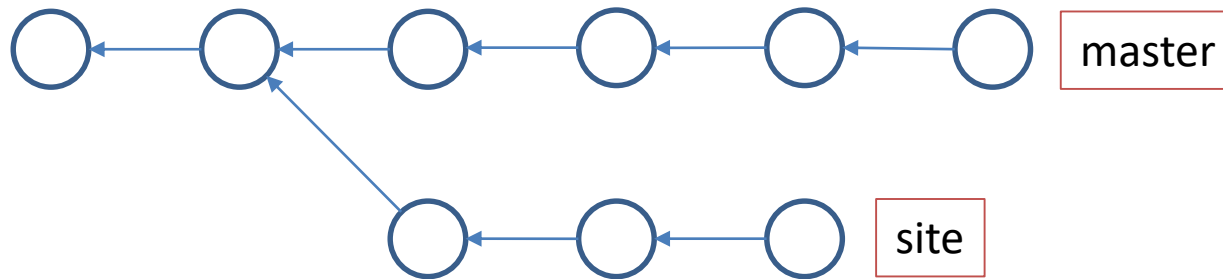
# Pra que usar ramos?

- Corrigir bugs e implementar funcionalidades



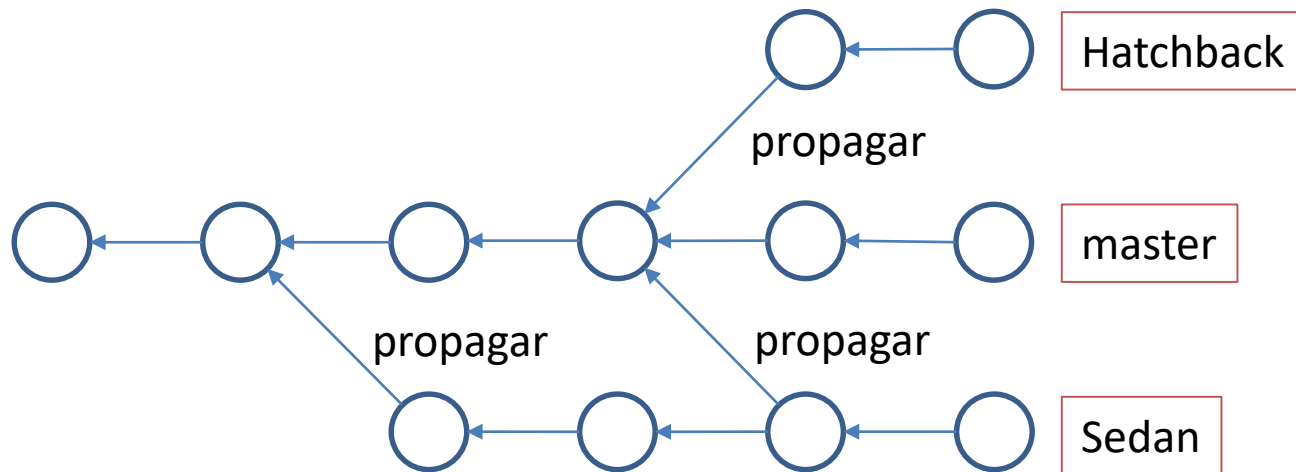
# Pra que usar ramos?

- Separar projetos



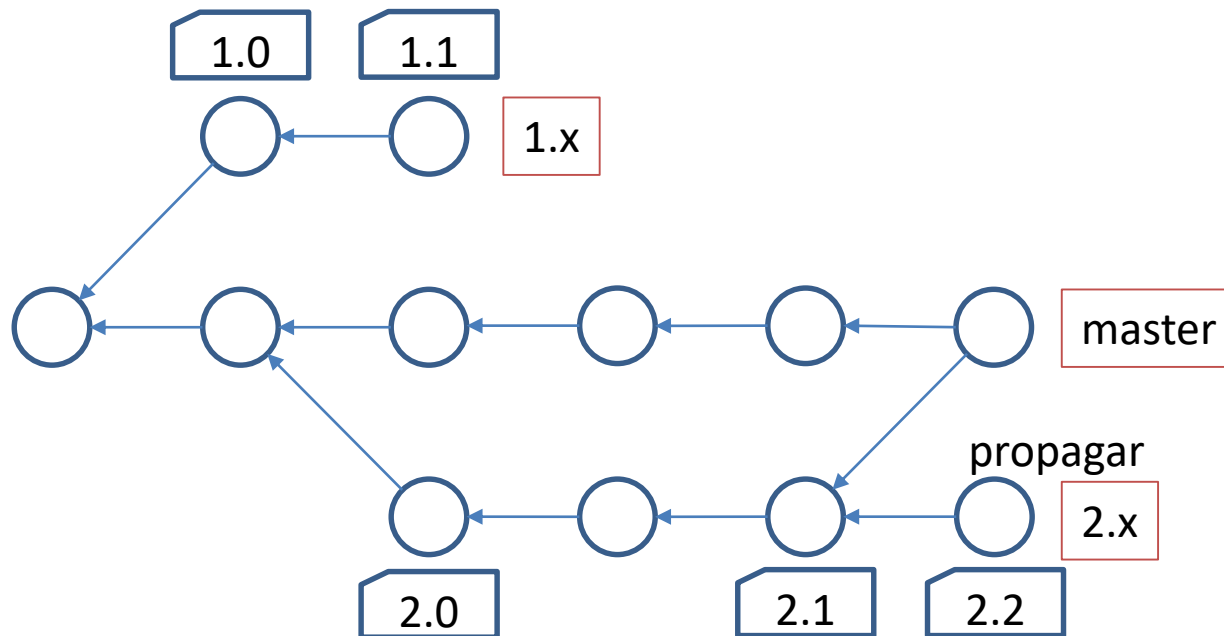
# Pra que usar ramos?

- Manter versões alternativas



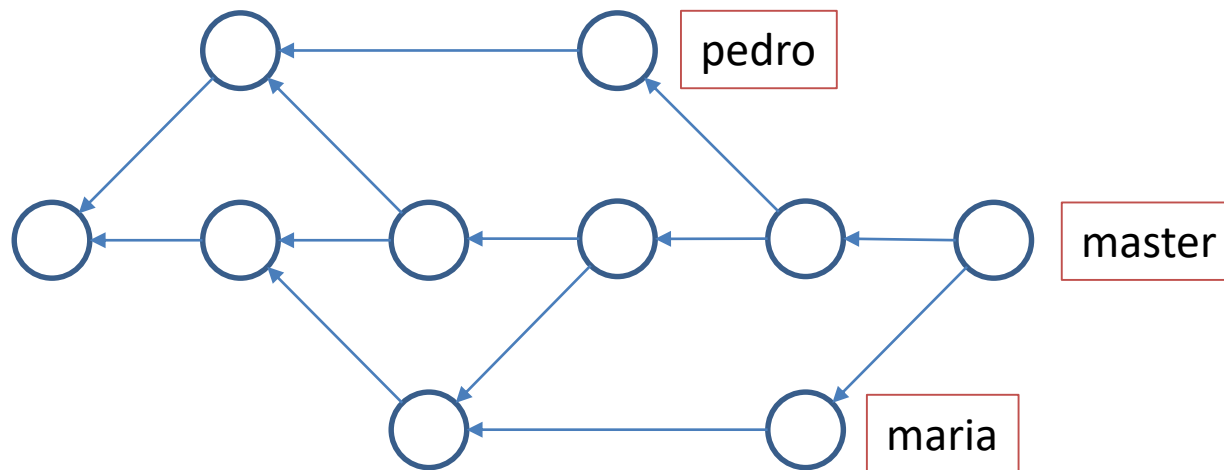
# Pra que usar ramos?

- Suportar versões antigas



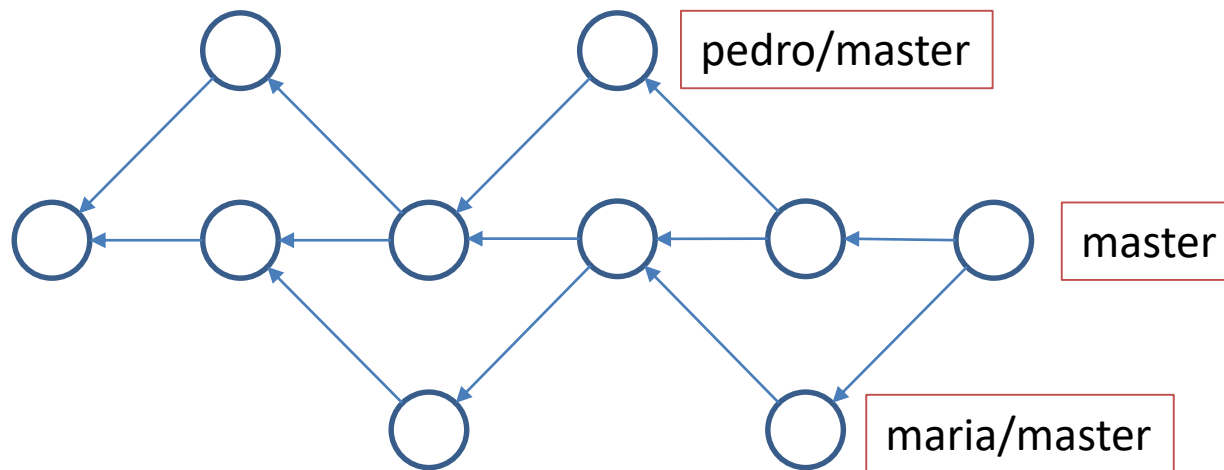
# Pra que usar ramos?

- Colaborar



# Pra que usar ramos?

- Colaborar





# Git vs GitHub



# GitHub



## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)



Owner

Repository name \*

 JoaoFelipe ▾ / introsuave 

Great repository names are short and memorable. Need inspiration? How about **supreme-octo-waffle?**

Description (optional)

- ☐  **Public**  
Anyone can see this repository. You choose who can commit.
- ☒  **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

- ☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾


Add a license: **None** ▾



Creating repository...

# GitHub

## Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** **SSH** `https://github.com/JoaoFelipe/introsuave.git` 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


## ...or create a new repository on the command line

```
echo "# introsuave" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/JoaoFelipe/introsuave.git
git push -u origin master
```



## ...or push an existing repository from the command line

```
git remote add origin https://github.com/JoaoFelipe/introsuave.git
git push -u origin master
```



## ...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)



# Repositórios remotos

- `git remote -v`
  - Listar os repositórios remotos cadastrados
- `git remote add <nome> <url>`
  - Adiciona um novo repositório remoto
- `git remote remove <nome>`
  - Remove um repositório remoto existente



# Sincronizando repositórios

- `git pull`
  - Atualiza o repositório local e o espaço de trabalho em relação a um repositório remoto
- `git push`
  - Atualiza o repositório remoto em relação ao repositório local



# \$ git clone <url>

- Cria um repositório local copiando o histórico de um repositório remoto
  - Define o remote *origin* como sendo o repositório remoto

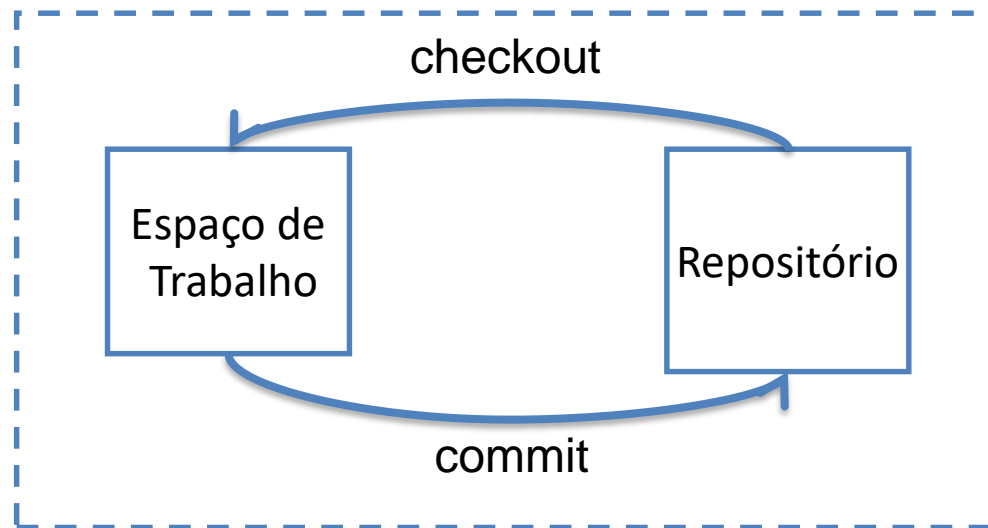
```
$ git clone https://github.com/JoaoFelipe/introsuave.git suaveintro2
Cloning into 'suaveintro2'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 18 (delta 1), reused 18 (delta 1), pack-reused 0
Unpacking objects: 100% (18/18), done.
```

# Histórico dos sistemas de controle de versões (VCS)

- Anos 70/80 – Sistemas locais
  - SCCS (1972)
  - RCS (1982)

# Histórico dos sistemas de controle de versões (VCS)

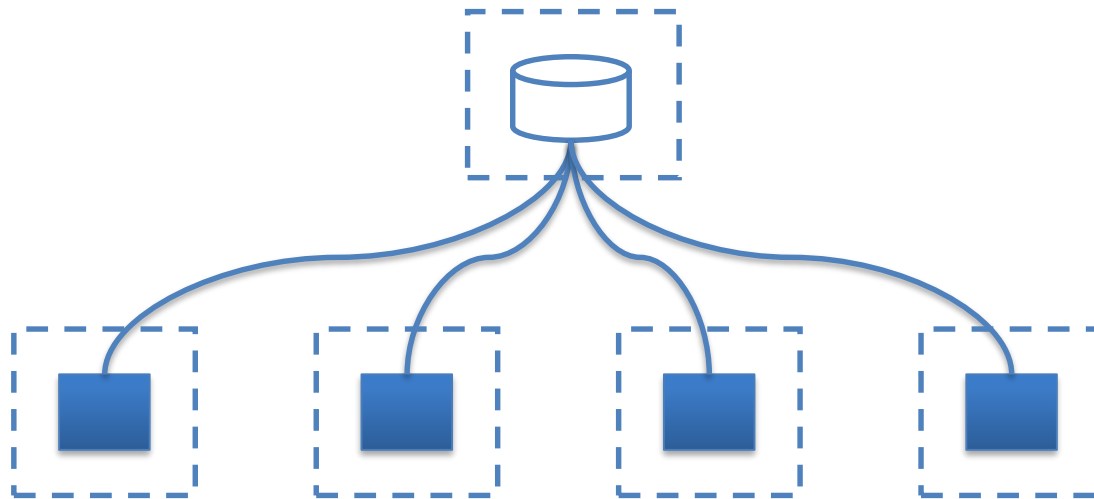
- Anos 70/80 – Sistemas locais
  - SCCS (1972)
  - RCS (1982)





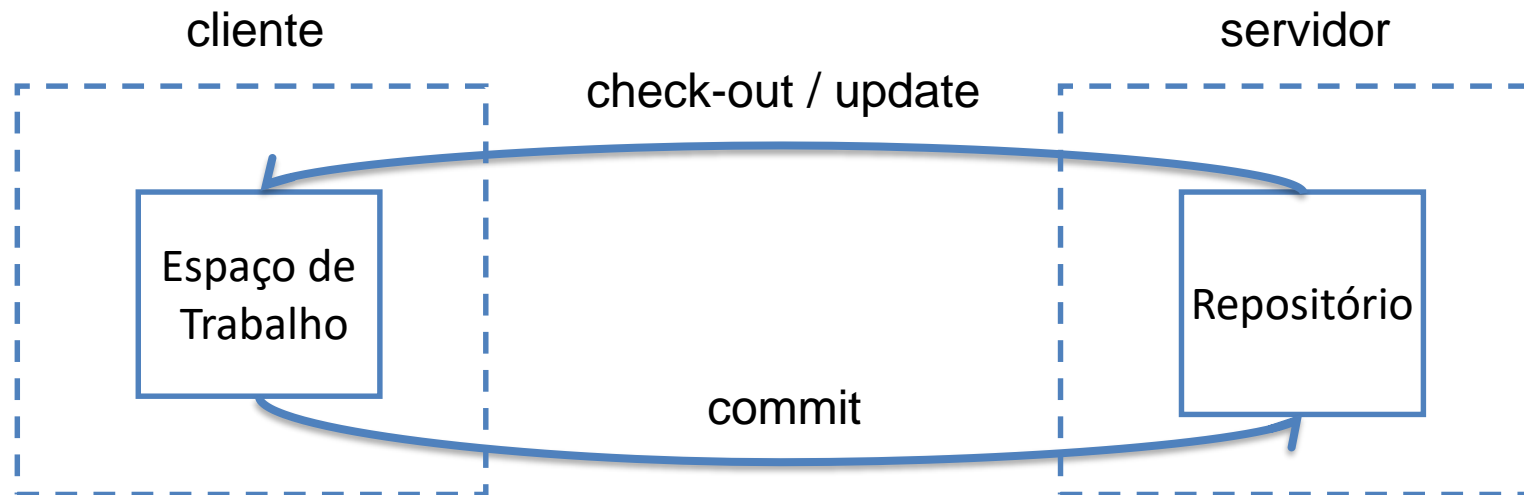
# Histórico dos sistemas de controle de versões (VCS)

- Anos 80/90 – Sistemas cliente-servidor
  - CVS (1986)
  - Subversion (2000)



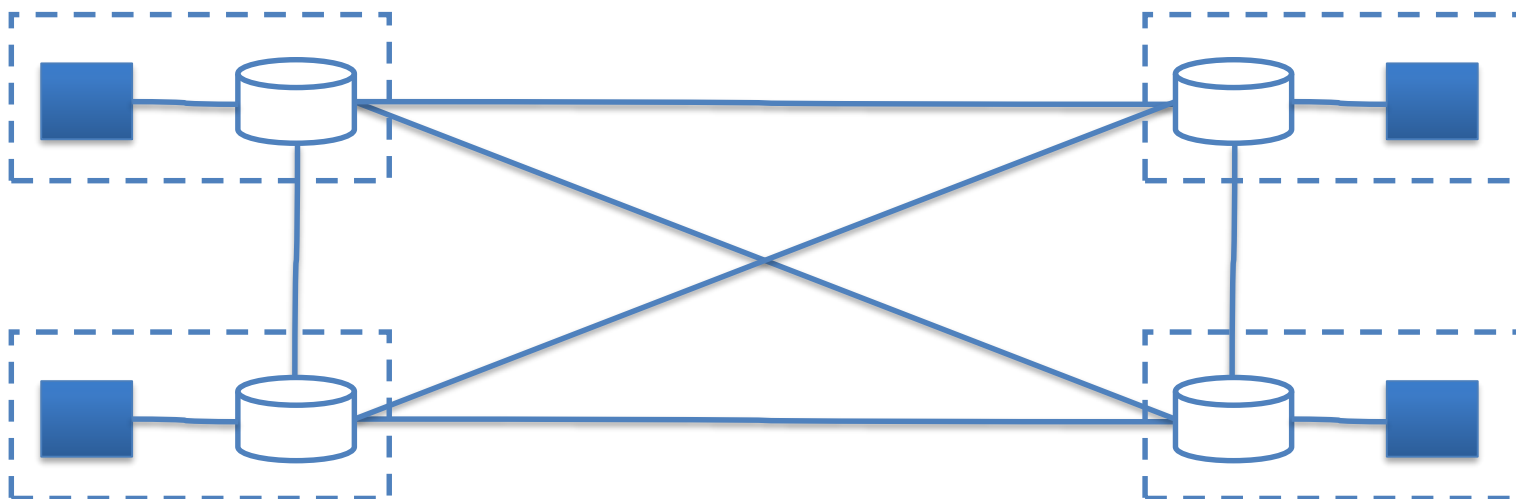
# Histórico dos sistemas de controle de versões (VCS)

- Anos 80/90 – Sistemas cliente-servidor
  - CVS (1986)
  - Subversion (2000)



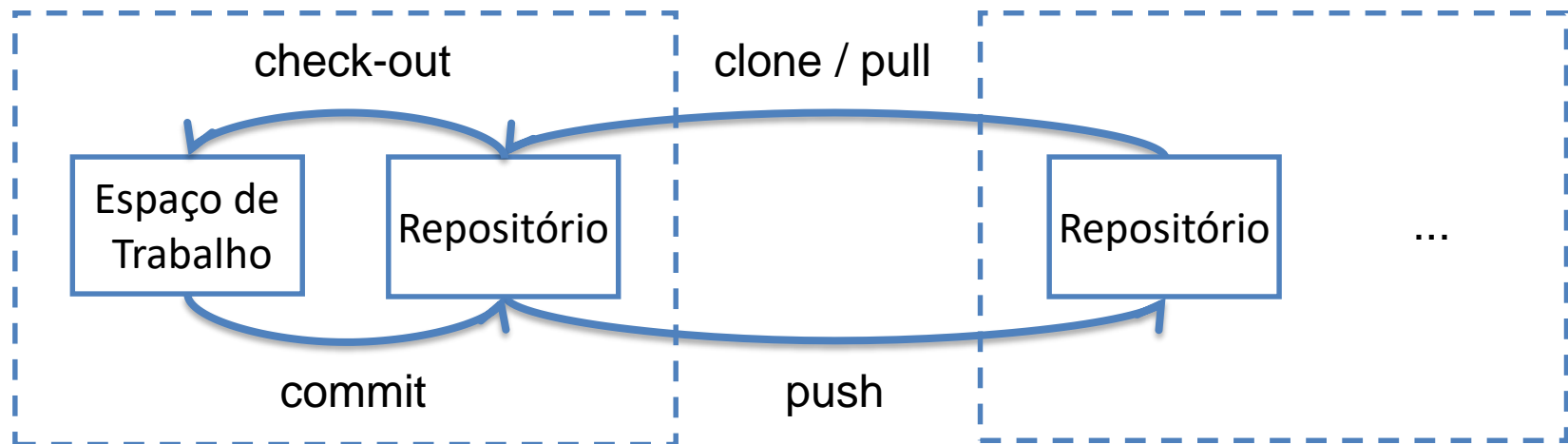
# Histórico dos sistemas de controle de versões (VCS)

- Anos 2000 – Sistemas peer-to-peer
  - Git (2005)
  - Mercurial (2005)



# Histórico dos sistemas de controle de versões (VCS)

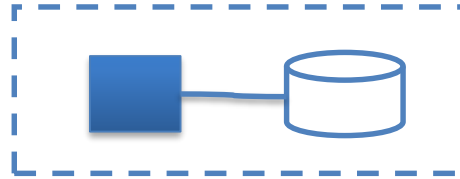
- Anos 2000 – Sistemas peer-to-peer
  - Git (2005)
  - Mercurial (2005)



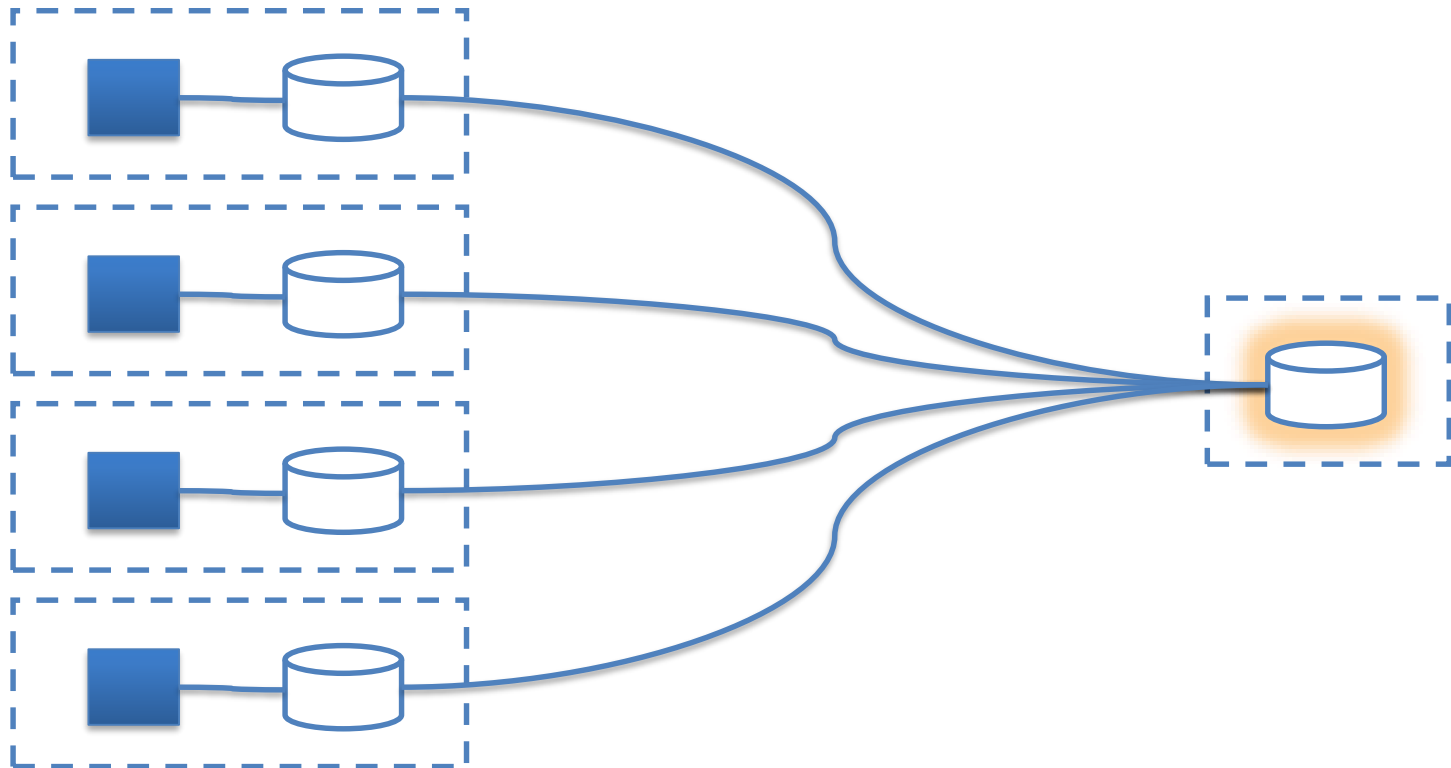
# Formas de adoção

- Apesar de ser peer-to-peer, normalmente é definido um “workflow” para adoção de DVCS em função de características do projeto
  - Individual
  - Cliente-servidor
  - Gerente de integração
  - Ditador/tenentes

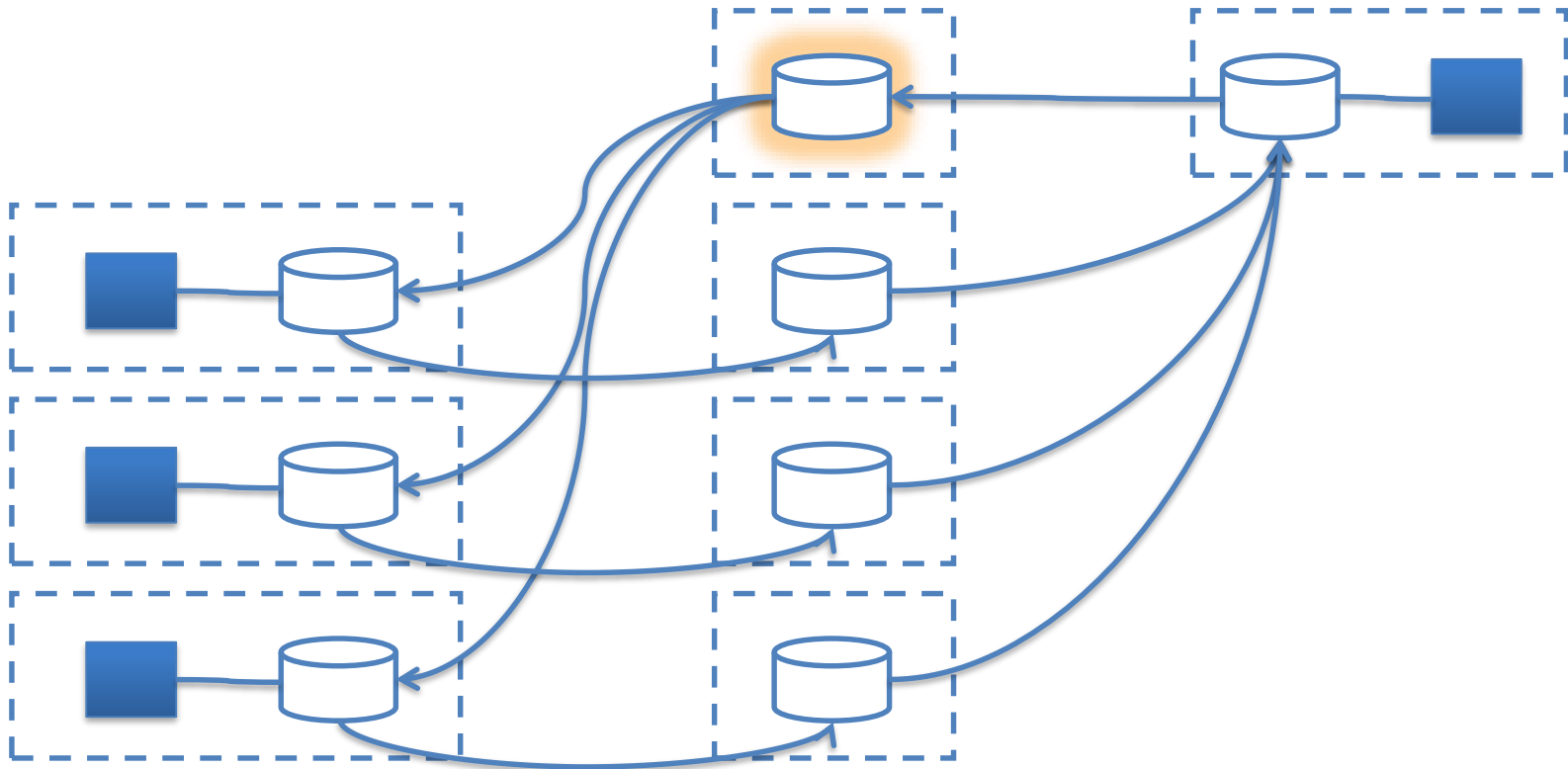
# Individual



# Cliente-servidor

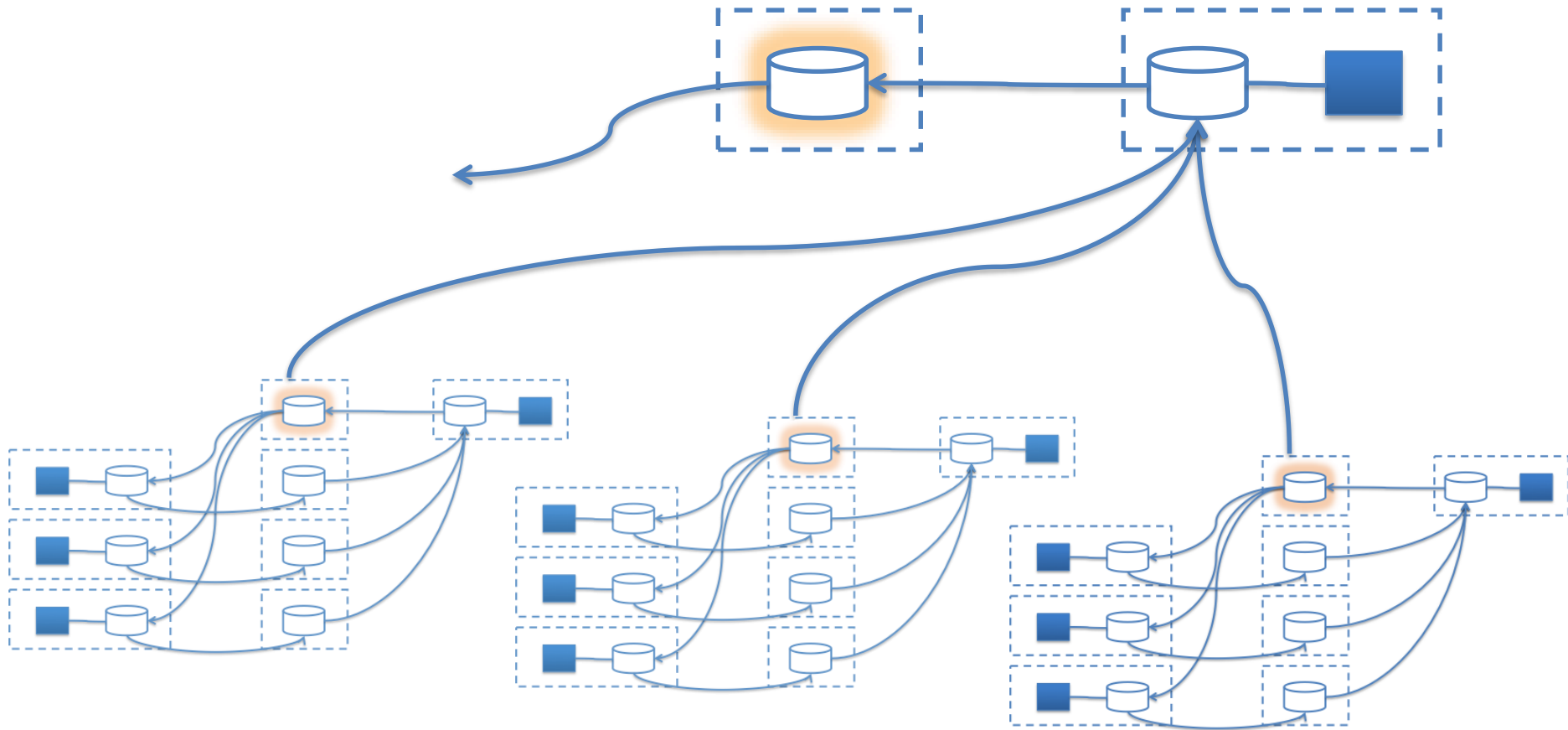


# Gerente de integração (fork + pull request)





# Ditador/tenentes (pull request em cascata)



# Principais referências bibliográficas

- Conradi, R. and Westfechtel, B. Version Models for Software Configuration Management. ACM Computing Surveys, v.30, n.2, p. 232-282, 1998.
- Chacon, S. Pro Git. Apress, 1ª edição, 2009.

# Uma Suave Introdução ao Git



Parte dos slides foram  
cedidos pelo Leonardo Murta