Versionamento de *software*

Amanda Souza Zírpolo e João Guilherme Coutinho Beltrão

Universidade Federal Fluminense

12 de novembro de 2023

O que é versionamento

Definição

O versionamento de *software* é o processo de atribuir um nome único ou uma numeração única para indicar o estado de um programa de computador. Esses números são geralmente atribuídos de forma crescente e indicam o desenvolvimento de melhorias ou correção de falhas no *software*.

Versionamento

Benefícios

- Análise de ameaças
- Reparação de *bugs*
- Alterações de arquitetura
- Potencialização do trabalho colaborativo
- Atualizações estéticas

Sistemas de controle de versão

Definição

Um sistema de controle de versões ou SCV é um *software* que tem a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer.

Sistemas de controle de versão

Vantagens

- O programador tem total controle e fácil acesso a todas as versões anteriores de seu código e quais são as diferenças de cada versão.
- A equipe de programadores consegue se ramificar e facilmente juntar os códigos no final do projeto.
- Auxilia na organização do projeto como um todo.

Tipos de SCV

SCV centralizado

No SCV centralizado, quando o programador salva um estado do código, o SCV envia o arquivo para um repositório local, conectado ao mesmo servidor que o computador está usando.

SCV distribuído

No SCV distribuído, o salvamento do estado do código pelo programador pode ser feito em um repositório local dentro do próprio computador. Posteriormente, esse arquivo será direcionado a um repositório remoto.

Tipos de SCV

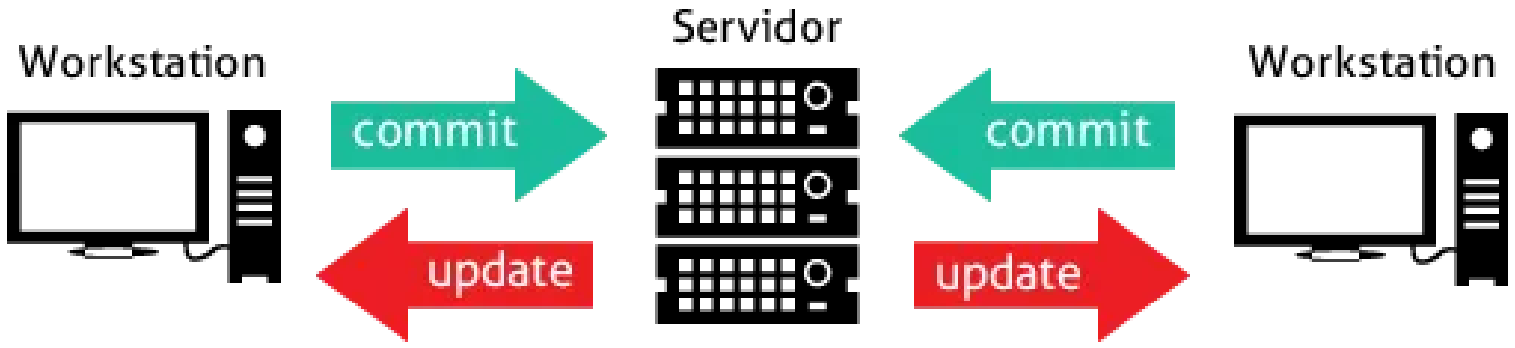

Figura 1: Sistema de controle de versões centralizado.

Tipos de SCV

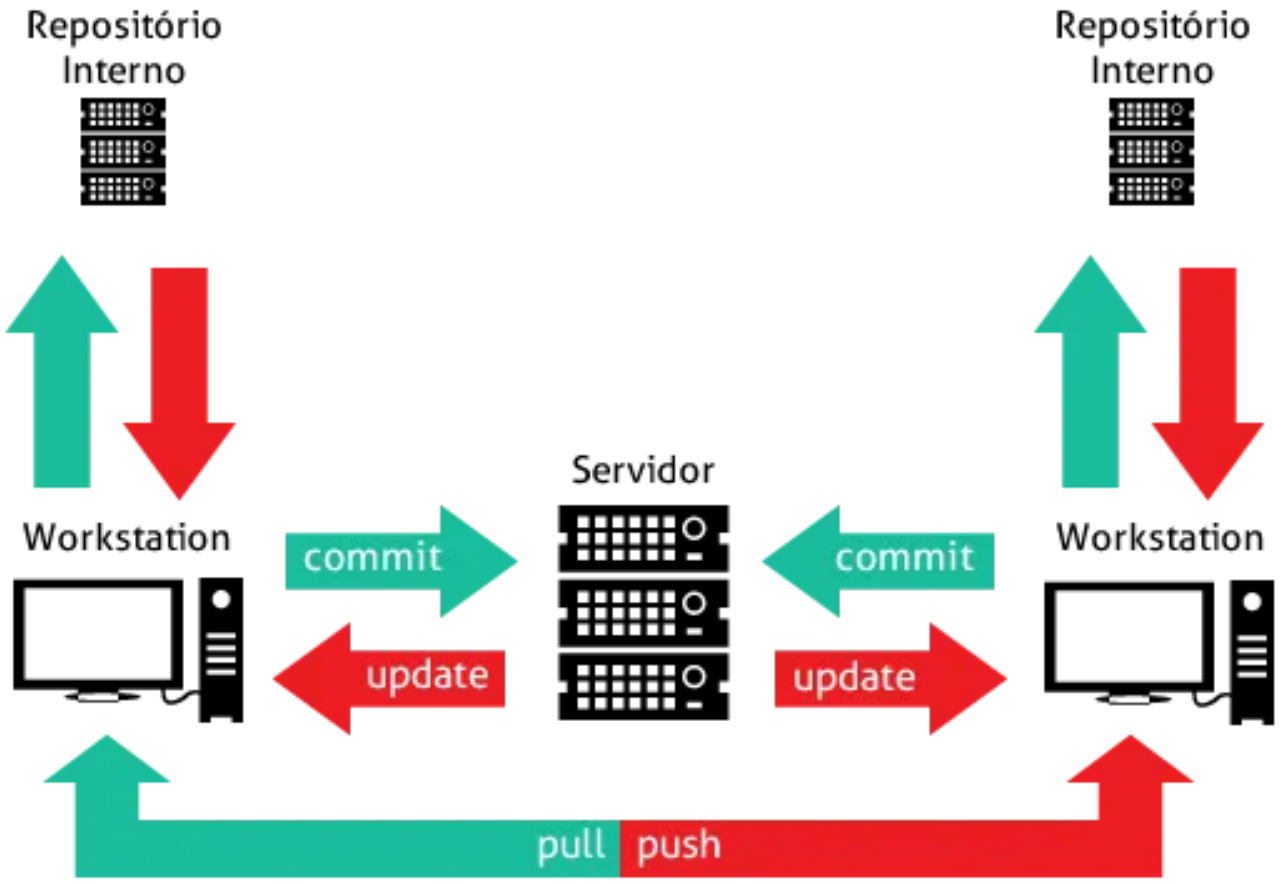

Figura 2: Sistema de controle de versões distribuído.

GIT

Definição

O *Git* é um sistema de controle de versão distribuído, criado, em 2005, por Linus Torvalds.

GIT

História

Foi desenvolvido por Linus Torvalds para o desenvolvimento do *kernel* do Linux, depois de um desentendimento com o criador do *BitKepper*.
Depois do seu desenvolvimento, em menos de um mês, o *Git* já estava sendo adotado por várias áreas diferentes da indústria.

GIT

Vantagens

- É *open source*, diferentemente do *BitKepper*.
- Possui um desempenho muito superior aos outros SCV da época.
- É distribuído.

download

Link para fazer o *download* do instalador

https://git-scm.com/download/linux

Instalação

Para a instalação no Linux, se for desejado instalar através de um instalador binário, pode-se geralmente fazê-lo por meio da ferramenta básica de gerenciamento de pacotes que vem com a distribuição usada. No Fedora, por exemplo, pode-se executar o seguinte comando: *$ sudo yum install git-all*.

Vinculação de usuário

Depois da instalação, abra o terminal e digite os seguintes comandos: *git config –global user.name* “inserir nome” e *git config –global user.e-mail* “inserir e-mail” .

Comandos:

Init

Usando o comando *git init*, o usuário cria ou reinicializa um repositório local para o arquivo que estiver sendo acessado no momento.

Status

Com o comando *git status*, o *Git* irá mostrar se ocorreu alguma alteração no repositório e o que se pode fazer com elas.

Comandos:

```
joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git init
Reinitialized existing Git repository in C:/Users/joaog/PET/.git/

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git status
On branch master
Your branch is up to date with 'testes/master'.
```

Figura 3: Comandos *git init* e *git status*.

Comandos:

Add

Com o comando *git add "nome do arquivo"*, o programador adiciona uma das modificações mostradas pelo status na fila de commits.

```
joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git add PET

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git status
On branch master
Your branch is up to date with 'testes/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   PET/Exemplo.txt
```

Figura 4: Comando *git add*.

Comandos:

Commit

Com o comando *git commit, o Git faz o "commit"* das mudanças na fila de commits. Isso significa enviá-los para o repositório local. Porém, para fazer o commit é necessário enviar um comentário junto a ele. Existem duas formas de fazer isso. Uma delas é usando o comando mostrado acima, onde o *Git* levará a uma outra tela, onde deverá ser adicionada a mensagem e, em seguida, selecionar "esc" e digitar ":wq" para sair. A outra forma é usar *git commit -m "comentario sobre a mudança"*, onde o *Git* já enviará o comentário junto ao commit.

Comandos:

```
joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git commit -m Exemplo
[master fc5f385] Exemplo
 1 file changed, 1 insertion(+)
 create mode 100644 PET/Exemplo.txt

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git status
On branch master
Your branch is ahead of 'testes/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Figura 5: Comando *git commit*.

Comandos:

Log

Usando o comando *git log*, o usuário consegue visualizar o histórico de todos os commits feitos no repositório acessado.

Diff

Com o comando *git diff*, o usuário consegue ver os estados das mudanças feitas no repositório inteiro ou em um arquivo específico (colocando o nome do arquivo depois do comando) no momento e após o último commit.

Comandos:

```
joaog@DESKTOP-HKHI60K MINGW64 ~/PET (novo)
$ git log
commit db746b45496a6458fd77eaece5505bb1f1884234 (HEAD -> novo)
Author: joaoguibelt <78770894+joaoguibelt@users.noreply.github.com>
Date:   Mon May 3 19:15:01 2021 -0300

    exemplo

commit caf46d2db219bccb71f71cabc5c132309a6898b4
Author: joaoguibelt <78770894+joaoguibelt@users.noreply.github.com>
Date:   Mon May 3 19:12:52 2021 -0300

    exemplo

commit d809c96dc9768de4de2e127c561a06c4195737c0 (origin/novo)
Author: joaoguibelt <78770894+joaoguibelt@users.noreply.github.com>
Date:   Tue Apr 6 20:39:36 2021 -0300

    documento

commit 6e79c424ef163704359c1bc6dabe7da78d012380
Author: joaoguibelt <78770894+joaoguibelt@users.noreply.github.com>
Date:   Tue Apr 6 20:37:50 2021 -0300
```

Figura 6: Comando *git log*.

Comandos:

```
PS C:\Users\USER\OneDrive\Área de Trabalho\PET-Tele\setel_2023\pet_tele> git diff
diff --git a/setel.py b/setel.py
index dd6a8f4..5eba8ea 100644
--- a/setel.py
+++ b/setel.py
@@ -3,4 +3,7 @@ print("PET-Tele!!!")
 print("oiiii")

 print("alterandooooooooooooooooooooooo")
-print("estou mudando o codigo")
\ No newline at end of file
+print("estou mudando o codigo")
+
+print("gosto muito de estudar")
+print("quero me tornar engenheira")
\ No newline at end of file
```

Figura 7: Uso do comando *git diff*.

Comandos:

Push

Usando o comando *git push*, o *Git* envia o conteúdo do repositório local para o repositório remoto, dependendo de qual tipo de repositório foi configurado.

Pull

Com o comando *git pull*, o *Git* busca e mescla o repositório remoto com o repositório local, deixando-os iguais.

Comandos:

```
joaog@DESKTOP-HKHI60K MINGW64 ~/PET (master)
$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 363 bytes | 121.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/joaoguibelt/testes
   1f1a5fa..fc5f385  master -> master
```

Figura 8: Comando *git push*.

Comandos:

```
joaog@DESKTOP-HKHI6OK MINGW64 ~ (master)
$ cd PET

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (novo)
$ git pull
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), 5.36 KiB | 24.00 KiB/s, done.
From https://github.com/joaoguibelt/testes
   45bca41..2dd2ef4  master     -> origin/master
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=<remote>/<branch> novo
```

Figura 9: Comando *git pull*.

Comandos:

Help

Com o comando *git help "comando"*, o programador é redirecionado para uma página de manual sobre o comando escolhido. Pode-se usar também o comando *git help*. Dessa forma, o *Git* dará a lista de todos os comandos e uma pequena frase sobre o que cada um deles faz.

Branchs

Definição

Elas representam ambientes de desenvolvimento diferentes para o mesmo projeto, onde um usuário pode trabalhar no mesmo código que outro programador, mas sem que ambos interfiram um no trabalho do outro.

Comandos branch:

- Para criar uma nova *branch* pelo *Git*, é só usar o comando *git branch "nome da branch"*.
- Para checar as *branchs* ativas, deve-se usar o comando *git branch*.
- Quando as *branchs* forem checadas, aparecerá um asterisco ao lado daquela que estiver sincronizada para receber os códigos. Para trocar isso, deve-se usar o comando *git checkout "nome da branch a sincronizar"*.
- Para as *branchs* novas terem permissão de fazer os *commits* primeiro deve-se usar *git push origin "nome da nova branch"*.

Comandos *branch*:

```
joaog@DESKTOP-HKHI6OK MINGW64 ~ (master)
$ cd PET

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git branch
* master
  novo

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (master)
$ git checkout novo
Switched to branch 'novo'

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (novo)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'testes/master'.
```

Figura 10: Comandos de *branch*.

Merge

Definição

É a junção das *branchs* periféricas com a *branch* principal.

Para fazer isso usa-se o comando *git merge* para fundir a *branch* atual com a principal.

```
PS C:\Users\USER\OneDrive\Área de Trabalho\PET-Tele\setel_2023\pet_tele> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\USER\OneDrive\Área de Trabalho\PET-Tele\setel_2023\pet_tele> git merge pet
Updating 13b62e2..337585b
Fast-forward
 setel.py | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Figura 11: Comando *git merge*.

Pull requests

Os *pull requests* são pedidos que o usuário pode fazer, para conseguir fazer alterações em repositórios remotos, quando ele não possui permissão para tal.

Comandos:

Reset

O comando *git reset* serve para fazer o ponteiro (chamado de *HEAD*) que aponta para o estado atual do código retornar para os estados de *commits* anteriores.

Restore

O comando *git restore "nome do arquivo"* é usado para restaurar um arquivo para sua versão anterior, descartando qualquer mudança feita nele que não foi adicionada para a linha de *commits*.

Comandos:

Revert

O comando *git revert "código do commit"* reverte o *commit* referenciado, cancelando qualquer alteração feita nele. O código do *commit* pode ser conseguido usando o comando *git log*.

Comandos:

Bisect

A técnica de *bisect* é usada para descobrir em qual *commit* ocorreu o erro indesejável no código. Primeiramente, deve-se usar o comando *git bisect start*, para o *Git* começar o processo.

É preciso informar ao *Git* que o *commit* atual possui um erro, usando o comando *git bisect bad*. Depois deve-se informar um *commit* que esteja sem erro com o comando *git bisect good "código do commit"*.

Comandos:

```
joaog@DESKTOP-HKHI6OK MINGW64 ~ (master)
$ cd PET

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (novo)
$ git bisect start

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (novo|BISECTING)
$ git bisect bad

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET (novo|BISECTING)
$ git bisect good 1bd2a2c318d1b5c17dfff7c1e24de19bb5723647
Bisecting: 9 revisions left to test after this (roughly 3 steps)
[70fdb696a2b949639b9615afb2cbb71440062f94] documento

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET ((70fdb69...)|BISECTING)
$ git bisect bad
Bisecting: 5 revisions left to test after this (roughly 2 steps)
[9917427760553d64c64bc01a60483032f7a76459d] Merge branch 'master' into novo

joaog@DESKTOP-HKHI6OK MINGW64 ~/PET ((9917427...)|BISECTING)
$
```

Figura 12: Comando *git bisect*.

Repositórios remotos

Como já foi dito, um SCV distribuído versiona os códigos enviando-os primeiramente para um repositório local e, em seguida, para o remoto. Nesse caso, o *Git* desempenha o papel do repositório local. Porém, para o remoto é preciso usar outra plataforma.

Repositórios remotos

(a) GitHub

(b) GitLab

Figura 13: Repositórios remotos GitHub e GitLab.

GitHub vs GitLab

Diferenças

- Velocidade vs Confiabilidade
- Integralização
- Amplo suporte de programas externos vs *DevOps* completo

Conectando com GitHub

Chave SSH

O *Secure Shell* (SSH) é um protocolo que garante trocas seguras de informação por meio de chaves geradas virtualmente. Com essas chaves, é possível fazer a comunicação entre os repositórios locais e os remotos.

Conectando com GitHub

Criando chave

Deve-se abrir o terminal e digitar o comando *ssh-keygen -t ed25519 -C "e-mail da conta do github"*.
Terminando o processo, a chave deve ser ligada a um *ssh-agent*.

Conectando com GitHub

Agente

Para que não seja necessário informar a senha toda vez que a chave for usada, é preciso ligá-la a um agente.

Primeiramente, deve-se iniciar o *ssh-agent*, digitando o comando *eval "ssh-agent -s"*. Em seguida, deve-se adicionar a chave a esse agente, com o comando *ssh-add∼/.ssh/id ed25519*. Se a chave não foi salva no local padrão, deve-se substituir *"id ed25519"* pelo nome do local onde encontra-se a chave.

Por fim, é necessário adicionar a chave ao *GitHub*, o que é explicado a seguir.

Conectando com GitHub

Adicionar chave ao *GitHub*

Primeiro é preciso digitar o comando *clip <~/.ssh/id ed25519.pub* e copiar o código que será retornado. Como anteriormente, se a chave não foi salva no local padrão, deve-se substituir *"id ed25519"*, pelo local onde ela está armazenada.

Agora, no *GitHub*, é necessário acessar o perfil e selecionar a opção *"settings"*. Depois, em *"SSH and GPC keys"*, selecionar *"new SSH key"*, adicionar um nome e colar o código salvo na caixa *key*. Para finalizar, deve-se selecionar *"add SSH key"* e inserir a senha do *GitHub* se for solicitado.

Instalando

https://github.com/shiftkey/desktop