

DEVOPS

DIA 2 DE 10





Augusto Nunes
PSM, MCT, OCA, OCP, MCSD



augustonunes



Augusto Nunes



augusto.cbn

GIT FUNDAMENTOS



GIT – SUMÁRIO

- Visão Geral
- Principais Comandos
- Git no Azure DevOps
- Laboratório



GIT

VISÃO GERAL

GIT – VISÃO GERAL

- Sistema de versionamento distribuído.
- Gratuito e de código aberto.
- Possibilidade de implementar diversos fluxos de trabalho.
- Trabalha com “staging areas” por padrão.

GIT – VISÃO GERAL

- O git trabalha desconectado, ou seja, as operações são executadas localmente, em um primeiro momento.
- Nascido para manter a gestão do Kernel Linux, sendo assim, suporta grandes repositórios.
- Escrito em C, o que reduz o overhead de linguagens de alto nível.

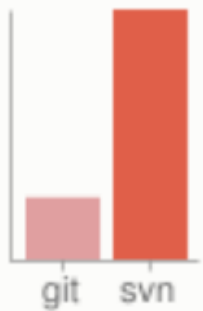
GIT – VISÃO GERAL - BENCHMARKS

* Unidade de medida em segundos

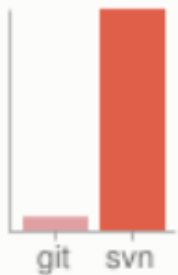
Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push a thousand 1 kB images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19 kB of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits – 9.4 MB of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c – 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x
Clone	Clone and shallow clone(*) in Git vs checkout in SVN	21.0	107.5	14.0
Size (MB)	Size of total client side data and files after clone/checkout (in MB)		181.0	132.0

GIT – VISÃO GERAL - BENCHMARKS

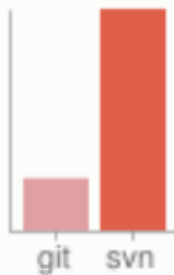
Commit A



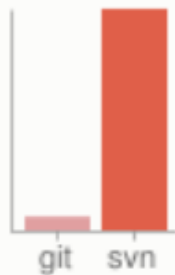
Commit B



Diff Curr



Diff Rec



Diff Tags



Clone



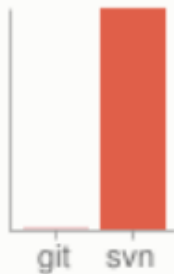
Log (50)



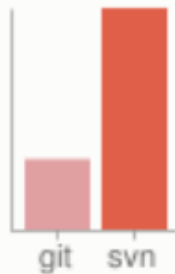
Log (All)



Log (File)



Update



Blame

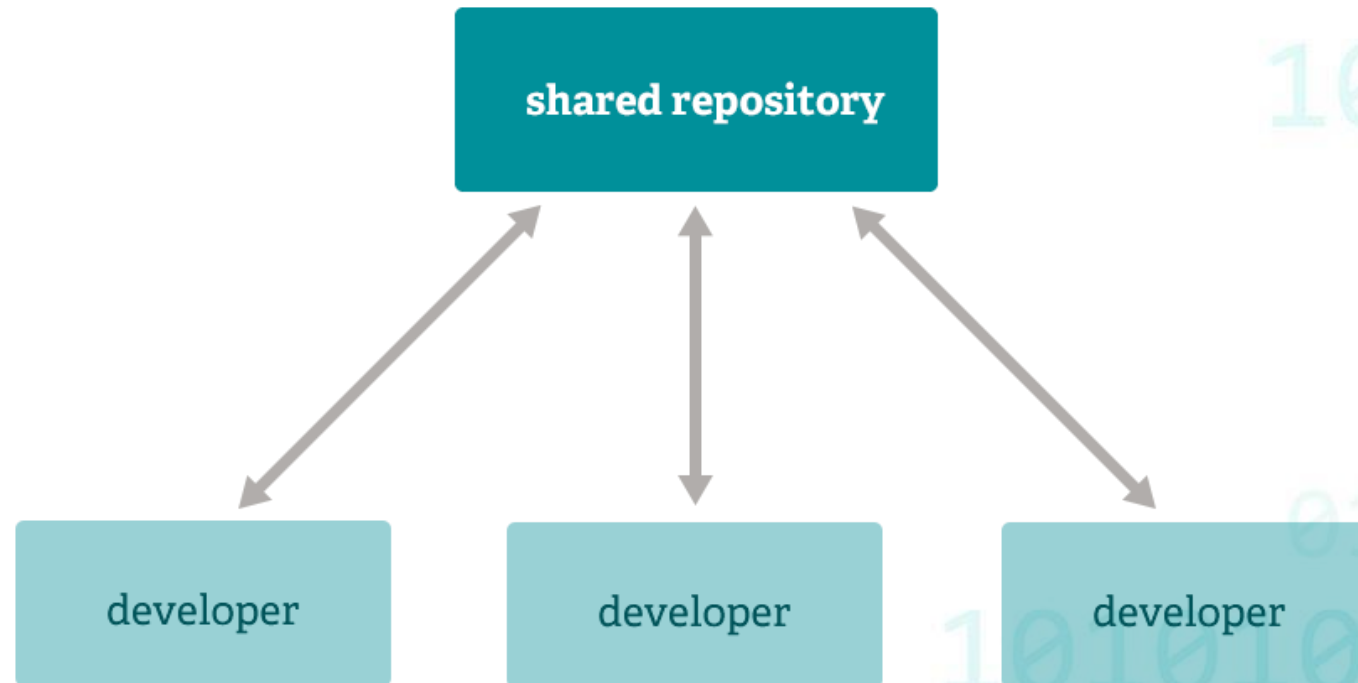


Size



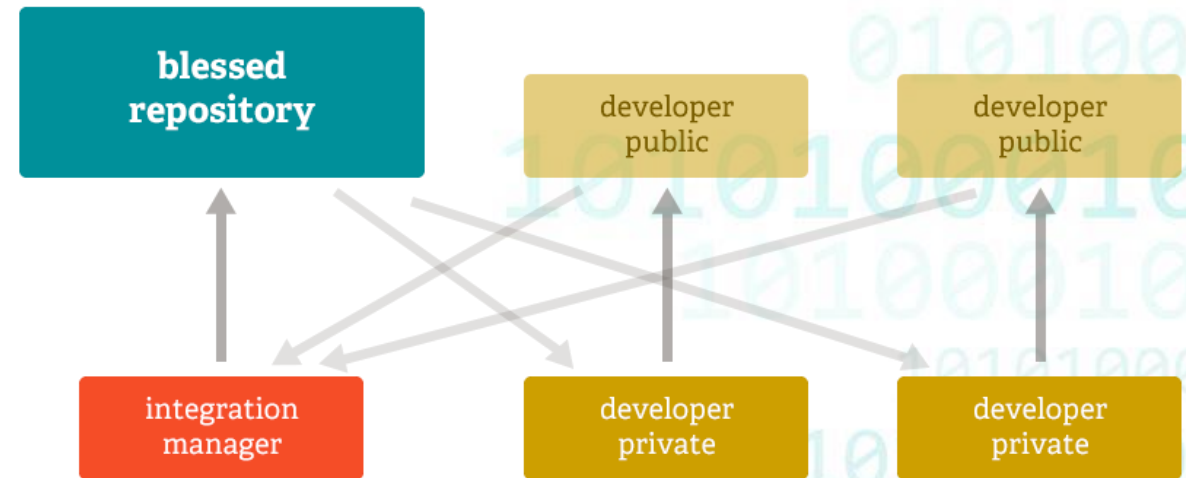
GIT – VISÃO GERAL – FLUXOS DE TRABALHO

Shared Repository, onde todos os membros da equipe realizam as integrações para um mesmo destino.



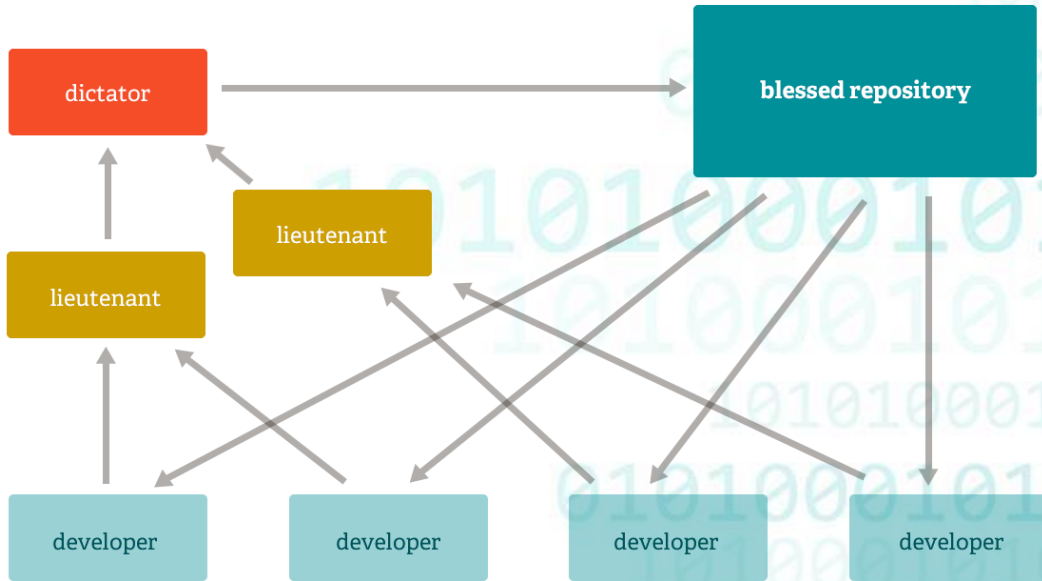
GIT – VISÃO GERAL – FLUXOS DE TRABALHO

Integration Manager, os desenvolvedores fazem um clone do repositório principal, trabalham em seus próprios correspondentes remotos e solicitam a um responsável que integre as suas alterações de volta a origem.



GIT – VISÃO GERAL – FLUXOS DE TRABALHO

Dictator and Lieutenants, onde o modelo é dividido em subsistemas. Cada subsistema possui um responsável (**lieutenant**), que valida os pushes submetidos pelos desenvolvedores. Os **lieutenants** por sua vez, submetem pushes ao **dictator**, que é o responsável por realizar as integrações com o **blessed repository**.



```
graph TD; D[dictator] --> BR[blessed repository]; BR --> D; L1[lieutenant] --> D; L2[lieutenant] --> D; D --> L1; D --> L2; Dev1[developer] --> L1; Dev2[developer] --> L2; Dev3[developer] --> L1; Dev4[developer] --> L2; BR --> Dev1; BR --> Dev2; BR --> Dev3; BR --> Dev4;
```

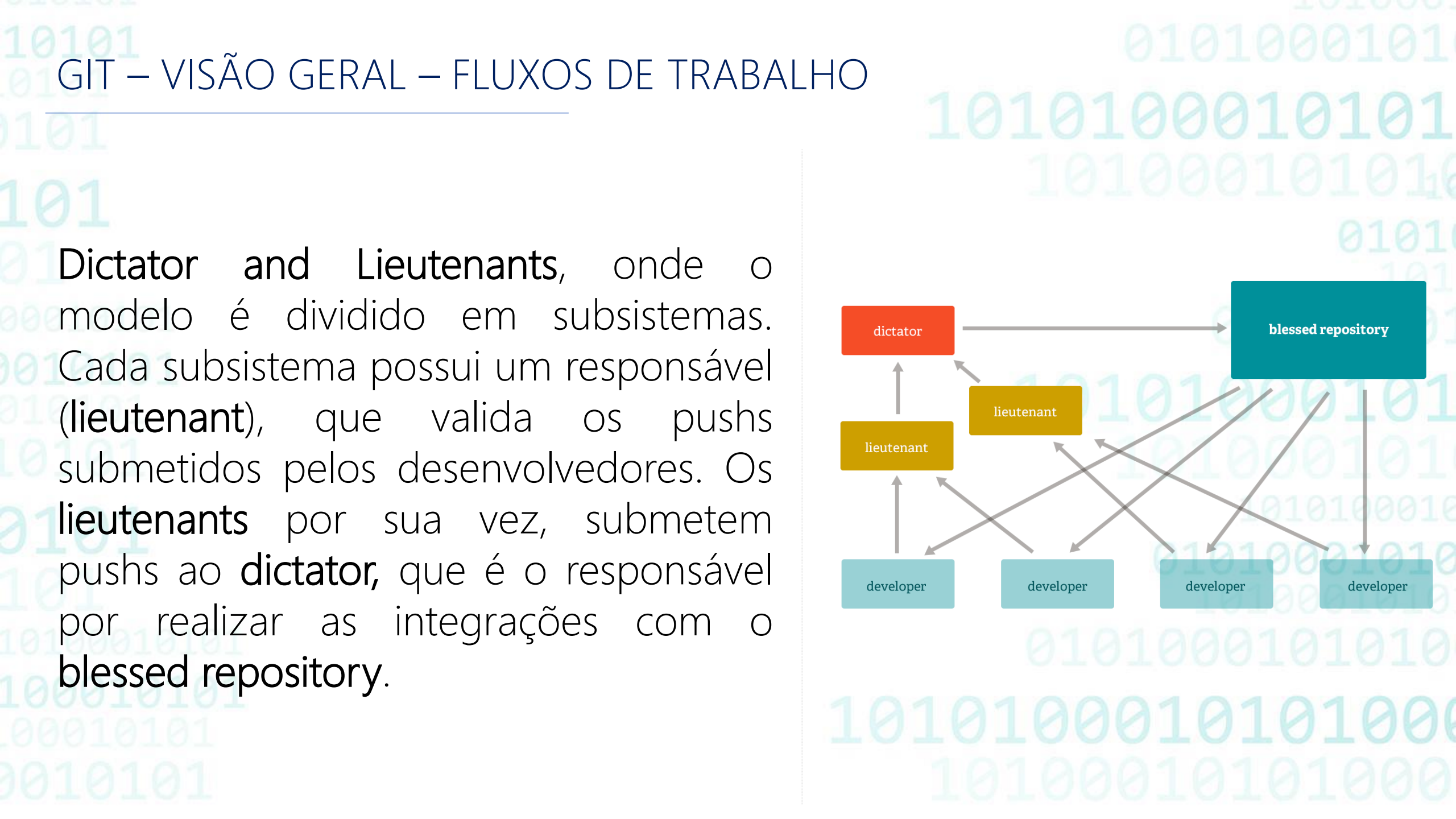
O diagrama ilustra o fluxo de trabalho Dictator and Lieutenants. No topo à esquerda, um retângulo laranja rotulado "dictator" está conectado por uma seta horizontal para a direita a um retângulo verde-azulado rotulado "blessed repository". Abaixo do "dictator", há dois retângulos amarelos rotulados "lieutenant". Abaixo desses, há quatro retângulos azuis rotulados "developer". As setas indicam o fluxo de trabalho: os desenvolvedores enviam pushes para os seus respectivos lieutenants; os lieutenants enviam pushes para o dictator; o dictator realiza as integrações com o blessed repository; e o blessed repository envia os pushes de volta para os desenvolvedores.

GIT – VISÃO GERAL – FLUXOS DE TRABALHO

Dictator and Lieutenants, onde o modelo é dividido em subsistemas. Cada subsistema possui um responsável (**lieutenant**), que valida os pushes submetidos pelos desenvolvedores. Os **lieutenants** por sua vez, submetem pushes ao **dictator**, que é o responsável por realizar as integrações com o **blessed repository**.

```
graph TD; D[dictator] --> BR[blessed repository]; BR --> D; L1[lieutenant] --> D; L2[lieutenant] --> D; D1[developer] --> L1; D2[developer] --> L2; D3[developer] --> L1; D4[developer] --> L2; BR --> D1; BR --> D2; BR --> D3; BR --> D4;
```

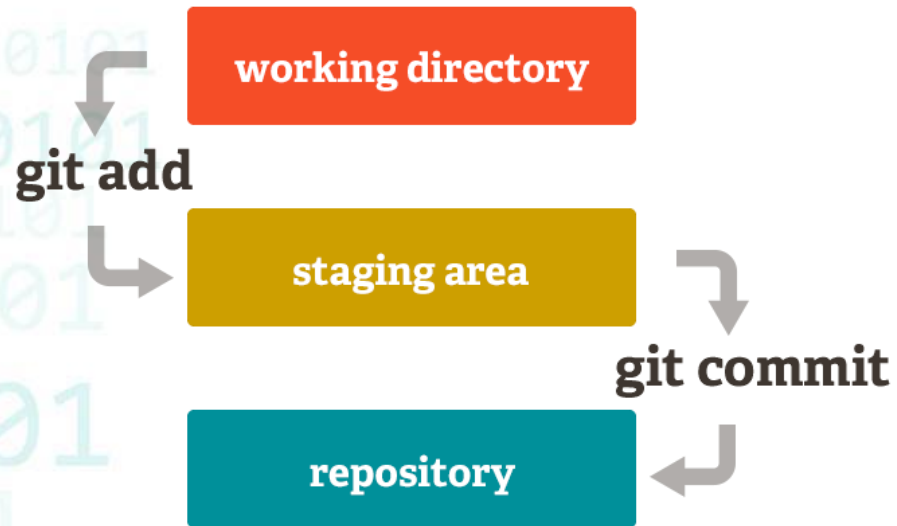
O diagrama ilustra o fluxo de trabalho Git Dictator and Lieutenants. No topo, há um retângulo vermelho rotulado "dictator" e um retângulo verde rotulado "blessed repository". Abaixo do "dictator", há dois retângulos amarelos rotulados "lieutenant". Na base, há quatro retângulos azuis rotulados "developer". As setas indicam o fluxo de trabalho: os desenvolvedores enviam pushes para seus respectivos lieutenants; os lieutenants enviam pushes para o dictator; o dictator envia pushes para o blessed repository; e o blessed repository envia pushes de volta para todos os desenvolvedores.



GIT – VISÃO GERAL – INTEGRIDADE

- Todos os dados mantidos nos repositórios Git são criptografados.
- Para todos os dados trafegados, uma combinação de “hash” é gerada, a fim de, garantir que eles não sejam alterados durante o trajeto.

GIT – VISÃO GERAL – STAGING AREA



O Git possui uma área intermediária, entre o seu diretório de trabalho e o seu repositório local. Com isso, é possível realizar uma operação de **commit** para uma lista de arquivos de seu interesse, sem se preocupar com os outros arquivos que foram alterados durante o trabalho.



GIT

PRINCIPAIS COMANDOS

GIT – PRINCIPAIS COMANDOS – GIT CONFIG

```
c:> git config  
c:> git config --list  
c:> git config --list --show-origin  
c:> git config [nome da chave]  
c:> git config [nome da chave] [valor]  
c:> git config --global [nome da chave] [valor]
```

O comando **git config** é utilizado para obter e atualizar as configurações do Git, de forma global, ou para um repositório específico.

GIT – PRINCIPAIS COMANDOS – GIT INIT

```
c:> git init
```

O comando **git init** é utilizado para transformar, um diretório padrão do sistema de arquivos, em um repositório Git.

GIT – PRINCIPAIS COMANDOS – GIT REMOTE

```
c:> git remote add origin [url do repositório remoto]  
c:> git remote -v
```

add origin possibilita vincular o repositório local a um remoto
-v permite visualizar o endereço do repositório remoto

O comando **git remote** é utilizado vincular um repositório Git local a um remoto.

GIT – PRINCIPAIS COMANDOS – GIT LOG

```
c:> git log  
c:> git log -p -3  
c:> git log --pretty=oneline
```

O comando **git log** permite visualizar o histórico de commits .

GIT – PRINCIPAIS COMANDOS – GIT STATUS

```
c:> git status
```

O comando **git status** exibe a situação dos arquivos e diretórios contidos no repositório Git.

GIT – PRINCIPAIS COMANDOS – GIT ADD

```
c:> git add .  
c:> git add [nome completo do arquivo ou diretório]
```

O comando **git add** inclui os arquivos e diretórios informados na área de staging (index).

GIT – PRINCIPAIS COMANDOS – GIT COMMIT

```
c:> git commit -m "mensagem amigável"  
c:> git commit -am "mensagem amigável"
```

- m permite a inclusão de uma mensagem
- a inclui as alterações na área de staging

O comando **git commit** efetiva as alterações, contidas na área de staging, diretamente no repositório local.

GIT – PRINCIPAIS COMANDOS – GIT PUSH

```
c:> git push  
c:> git push origin [branch específico]
```

O comando **git push** envia as alterações, contidas no repositório local, ao correspondente remoto.

GIT – PRINCIPAIS COMANDOS – GIT PULL

```
c:> git pull  
c:> git pull origin [branch específico]
```

O comando **git pull** obtém atualizações do repositório remoto e os transfere para o branch de trabalho.

GIT – PRINCIPAIS COMANDOS – GIT CLONE

```
c:> git clone [url do repositório remoto]  
c:> git clone [url do repositório remoto] -b [nome do branch]
```

O comando **git clone** realiza a cópia de um repositório remoto para um diretório local, transformando essa área do sistema de arquivo em um diretório git.

GIT – PRINCIPAIS COMANDOS – GIT CHECKOUT

```
c:> git checkout [branch, hash, tag]  
c:> git checkout -b [branch]
```

-b cria um novo branch

O comando **git checkout** permite trocar o branch de trabalho ou realizar a movimentação para um ponto de restauração.



GIT

AZURE DEVOPS

GIT – AZURE DEVOPS – REPOSITÓRIOS – VISÃO GERAL

- Possui uma instância privada do Git.
- Fácil integração com os ambientes de desenvolvimento (IDE).
- Segurança aprimorada, através de permissões de acesso e políticas de branch.
- Interface gráfica que reconhece o código fonte. Fácil para realizar pesquisas semânticas.

GIT – AZURE DEVOPS – REPOSITÓRIOS – VISÃO GERAL

- Interface gráfica para administração dos branches, tags e visualização da árvore de commits. Permite o acompanhamento do histórico de alterações.

GIT – AZURE DEVOPS – POLÍTICAS DE BRANCHES

Através das políticas de branches, é possível criar uma série de regras de validação, com o objetivo de garantir a integridade do repositório. Com isso, para que os commits sejam sincronizados ao branch alvo, algumas regras deverão ser obedecidas.

Dessa forma, é criada uma barreira que impede que commits sejam enviados diretamente a determinados branches.

Branch policies for master

 Save changes  Discard changes

Protect this branch

- Setting a Required policy will enforce the use of pull requests when updating the branch
- Setting a Required policy will prevent branch deletion
- Manage permissions for this branch on the [Security page](#)

☒ Require a minimum number of reviewers

Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers

- ☒ Allow requestors to approve their own changes
- ☐ Prohibit the most recent pusher from approving their own changes
- ☐ Allow completion even if some reviewers vote to wait or reject
- ☐ Reset code reviewer votes when there are new changes

☐ Check for linked work items

Encourage traceability by checking for linked work items on pull requests.

☐ Check for comment resolution

Check to see that all comments have been resolved on pull requests.

☒ Limit merge types

Control branch history by limiting the available types of merge when pull requests are completed.

Allowed merge types:

- ☐ Basic merge (no fast-forward)
Preserves nonlinear history exactly as it happened during development.
- ☒ Squash merge
Creates a linear history by condensing the source branch commits into a single new commit on the target branch.

GIT – AZURE DEVOPS – POLÍTICAS

Garantia de vinculação a um requisito de software.

Regras para garantir um número mínimo de revisores.



Integração com ferramentas de terceiros.

Vinculação a pipelines de integração contínua.

GIT – PULL REQUESTS (PR)

A sincronização com o branch alvo ocorrerá apenas quando todas as políticas forem satisfeitas.

Garantia de integridade do repositório.



Interface gráfica para gerenciar as requisições de mudança.

Não é possível enviar commits para os branches principais.



GIT

LABORATÓRIO 2