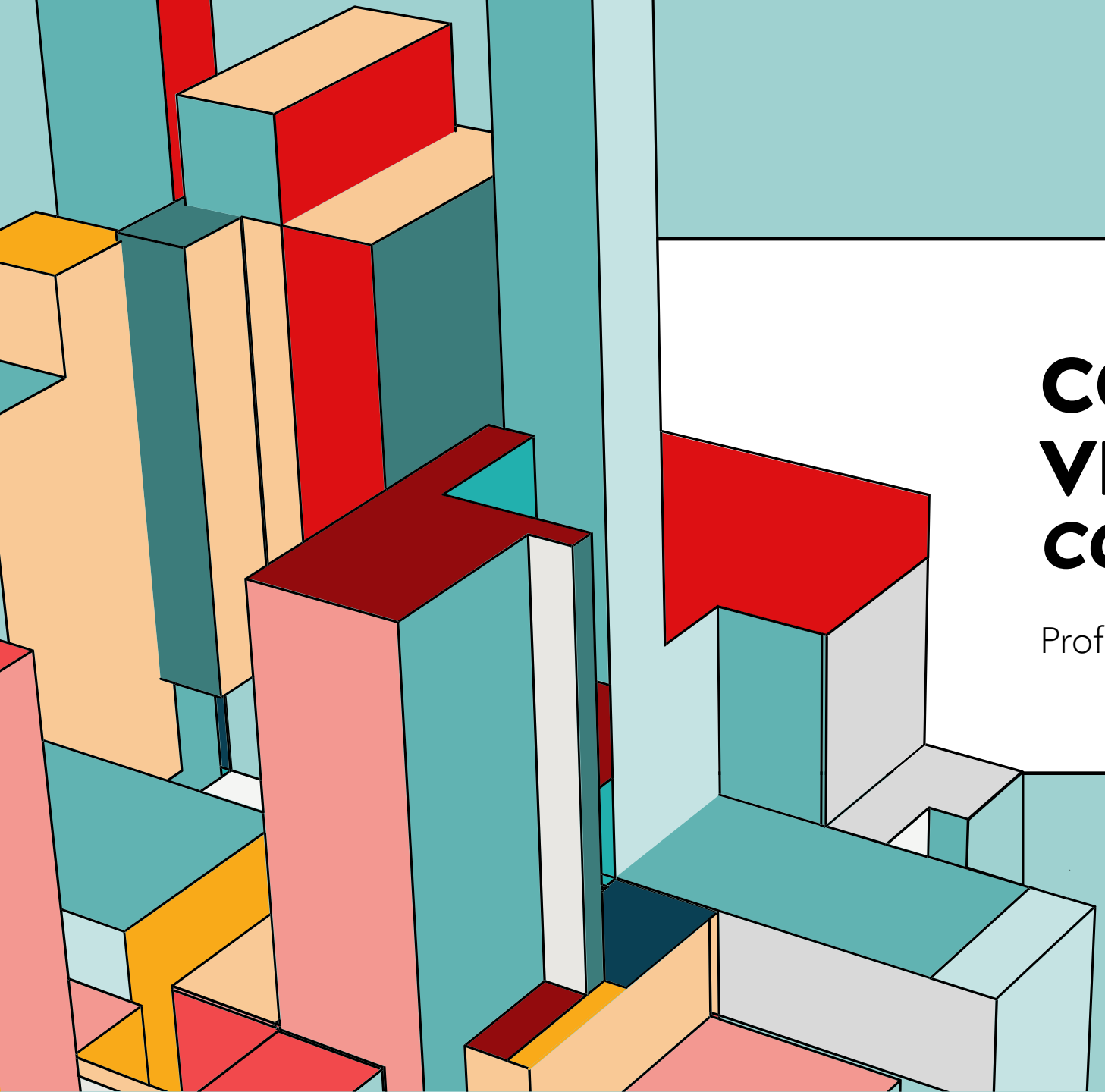


CONTROLE DE VERSÃO (GIT)
COMANDOS BÁSICOS

Prof. Dr. Dilermando Piva Jr

COM O GIT FUNCIONA?

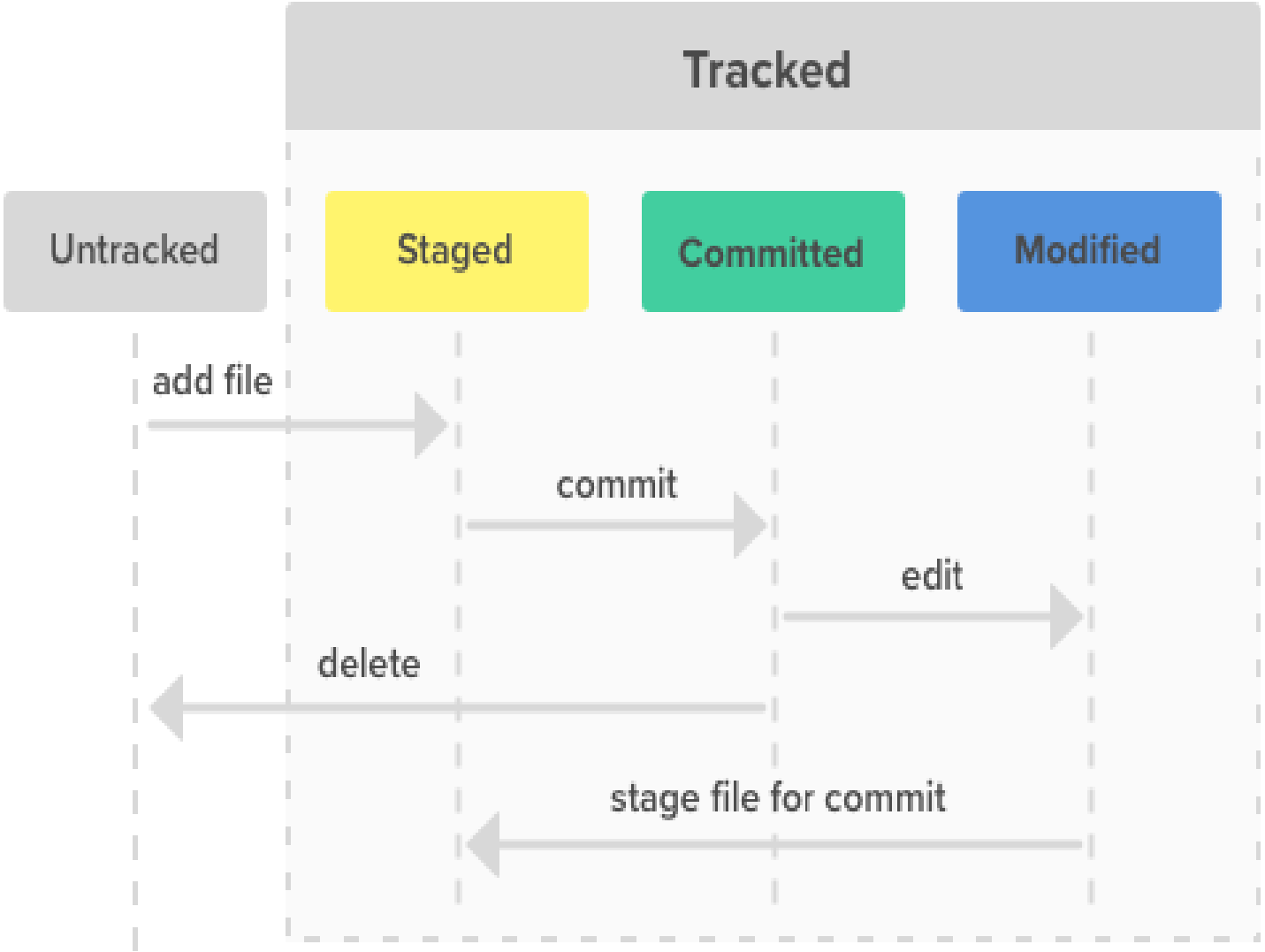
Por ser um sistema de gerenciamento de controle de versão de código/documentos, o git possui recursos avançados de monitoramento e acompanhamento de arquivos que fazem parte de um projeto.

Após ativo em um projeto, o git começa a **monitorar** as mudanças ocorridas neste projeto e nos mostra através de um poderoso sistema de **estágios** (ou fases) em que estágio ou fase se encontram os arquivos.

É este sistema de estágios/fases que iremos conhecer e entender seus significados ...

CICLO DE VIDA

O GIT utiliza um poderoso sistema de gerenciamento e controle de estágios/fases para os arquivos que fazem parte de um projeto.

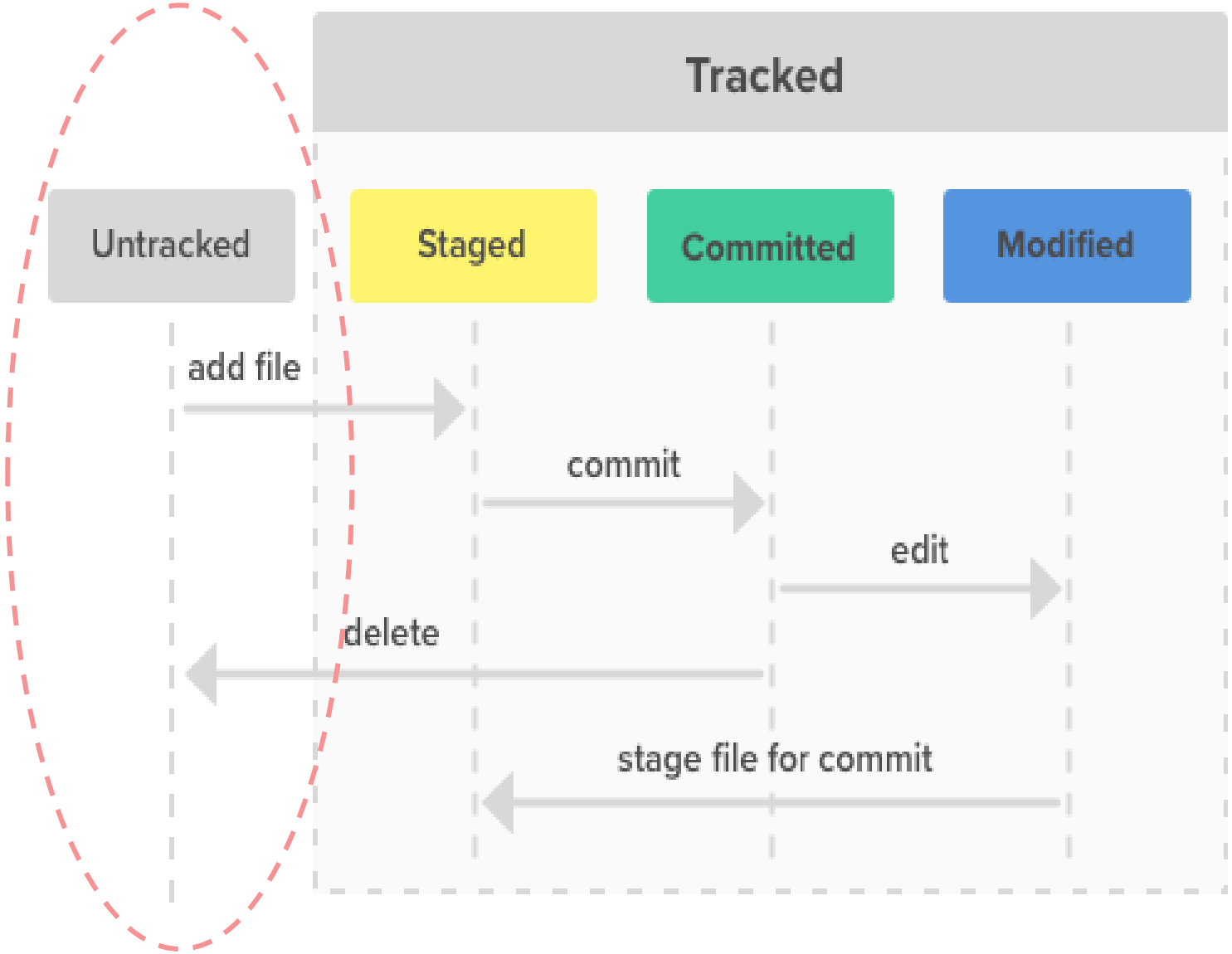


CICLO DE VIDA
UNTRACKED

Quando criamos um novo arquivo ou incluímos um ou mais arquivos para serem gerenciados pelo git, os arquivos são incluídos com o status “untracked”
→ Não rastreado

Isso significa que, mesmo o(s) arquivo(s) existindo, ele(s) ainda não foi(ram) adicionado(s) ao monitoramento do git

Nesse status, o git não estará controlando seu versionamento.

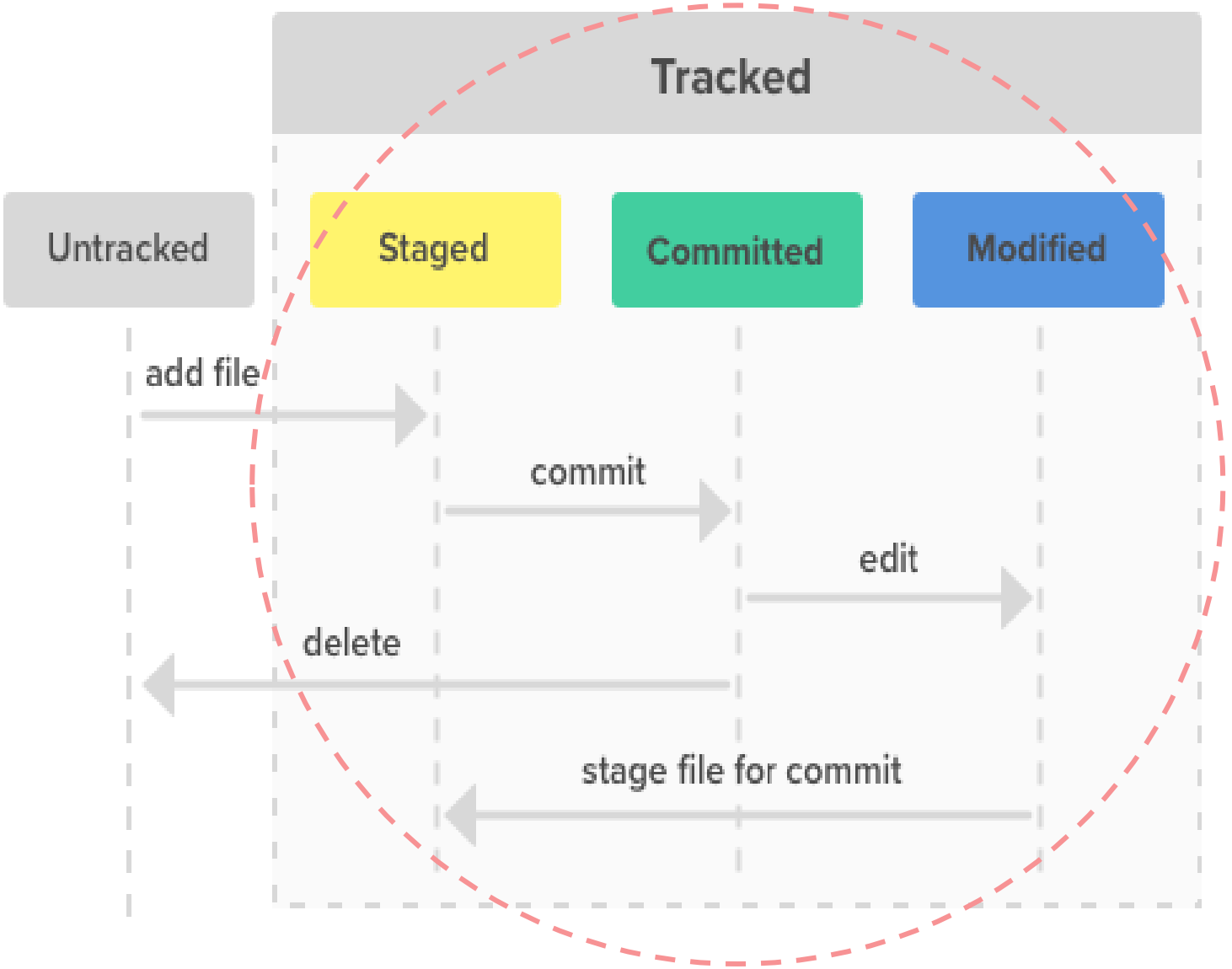


CICLO DE VIDA
TRACKED

Quando adicionamos arquivos ao monitoramento do git, o status deste arquivo passa a ser “tracked”
→ Rastreado

A partir de então o git passa a controlar as mudanças neste arquivo

Os arquivos com status “tracked” são conhecidos como “new file” (novos arquivos)

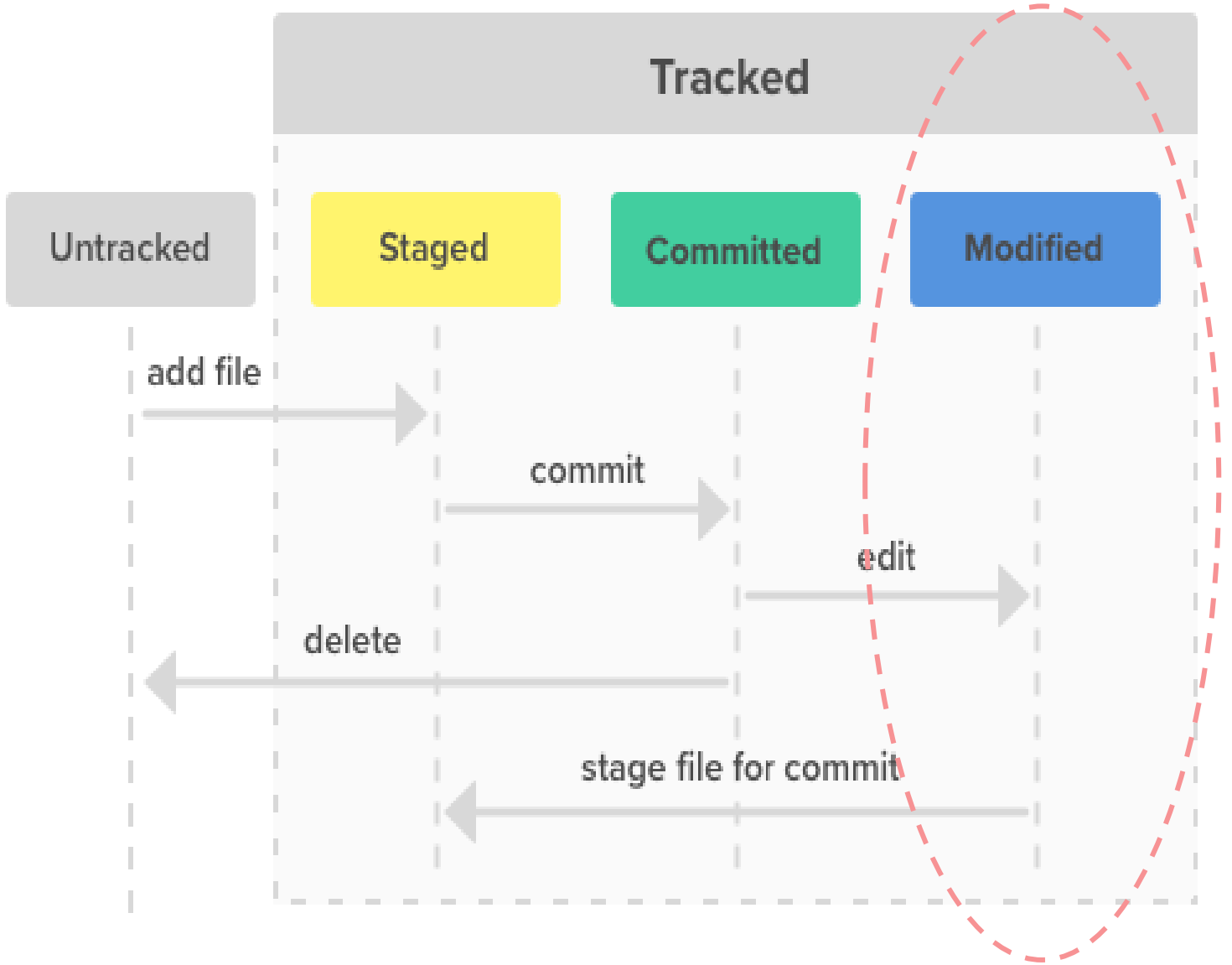


CICLO DE VIDA
MODIFIED

Quando modificamos um arquivo e ele está sendo rastreado, seu status muda para “modified”
→ Modificado

Atenção:
É nesta etapa que pode ocorrer conflitos. Ex: você e outra pessoa estão trabalhando no mesmo arquivo... No final, ambas as modificações devem ser “juntadas” para ter apenas uma modificação;

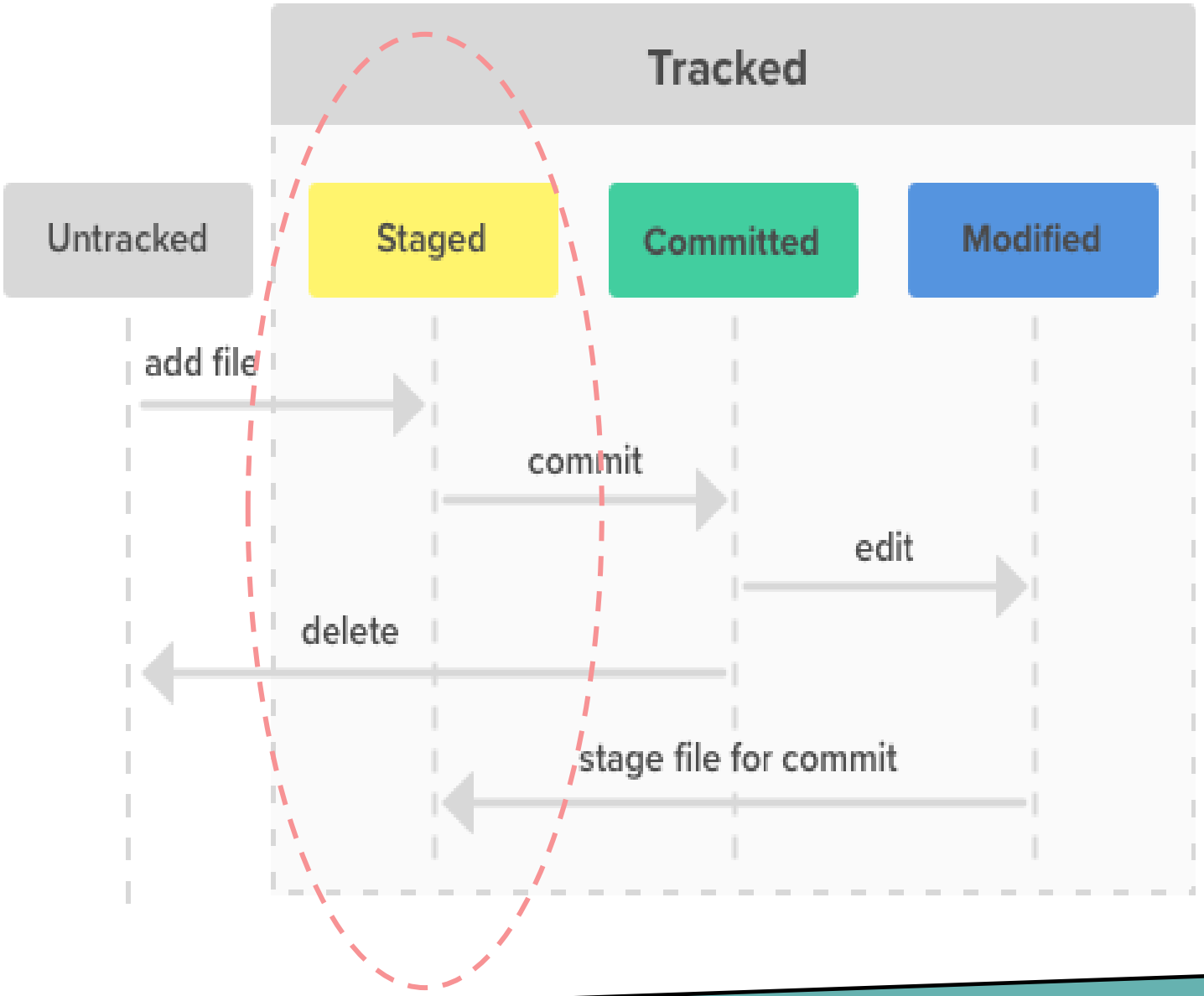
Mágica do MERGE.



CICLO DE VIDA
STAGED

Quando um arquivo está pronto, ou seja, ele está sendo rastreado e já foi modificado e finalizado e está pronto para ser enviado para o repositório, seu status passa para “staged”
→ Preparado

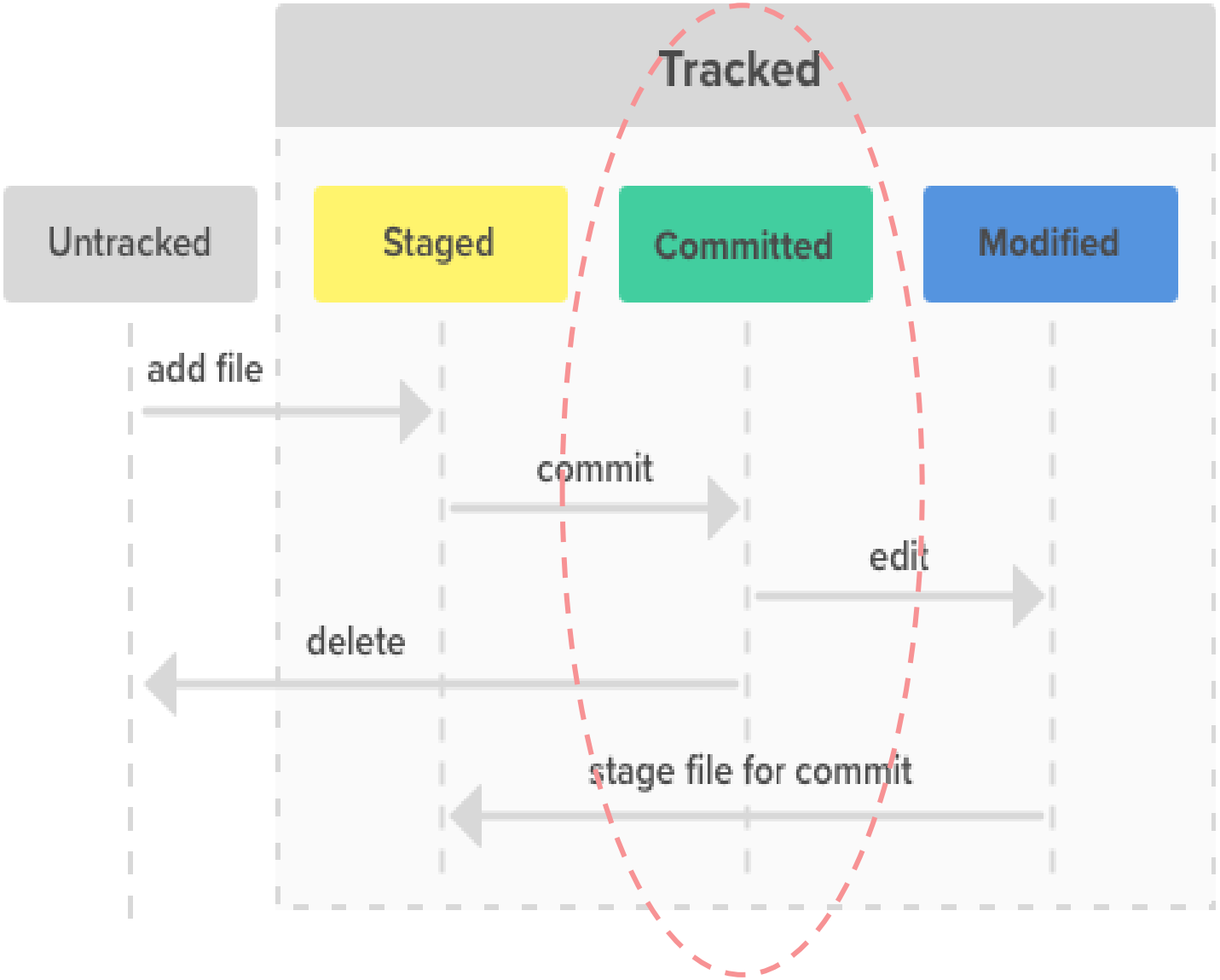
Atenção:
Os arquivos no estágio “staged” são enviados para o repositório através de commits.



CICLO DE VIDA
COMMITTED

Depois que os arquivos são envidados para o repositório por intermédio de commits (ou seja, são atualizados) seu status passa para “committed”
→ Enviados / Entregues / Atualizados

Atenção:
Esses arquivos são aqueles que guardam as últimas atualizações (os mais atualizados).

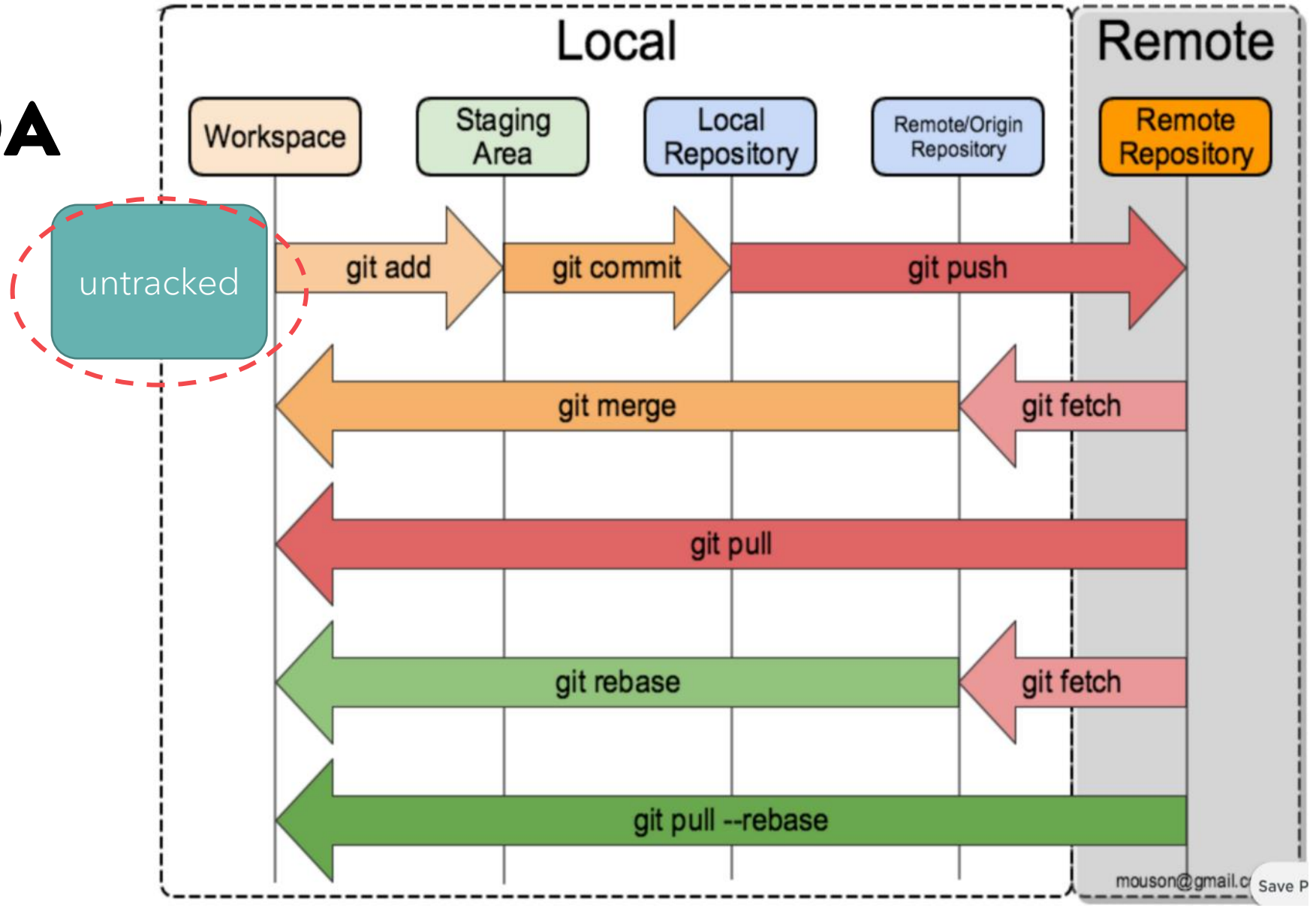


CICLO DE VIDA

Para inicializar o gerenciamento de versionamento pelo GIT:

git init

Ao fazer isso, todos os arquivos passam para o Workspace do projeto. Eles ainda estão no estágio UNTRACKED.



CICLO DE VIDA

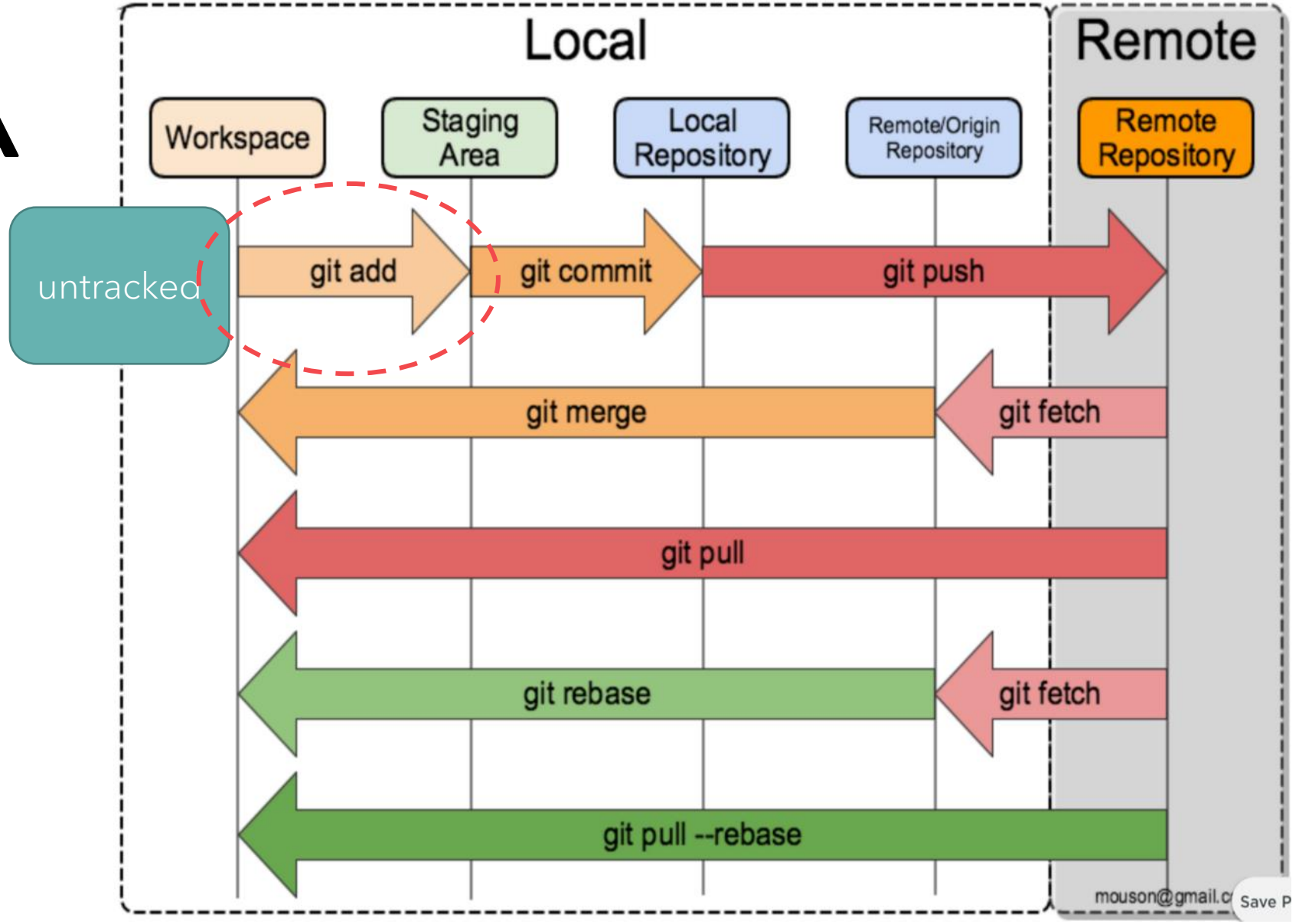
Para que os arquivos passem a ser rastreados, temos que adicioná-los:

git add .

Adicionamos todos os arquivos
Ou

git add arquivo.py

Adicionamos um arquivo específico



CICLO DE VIDA

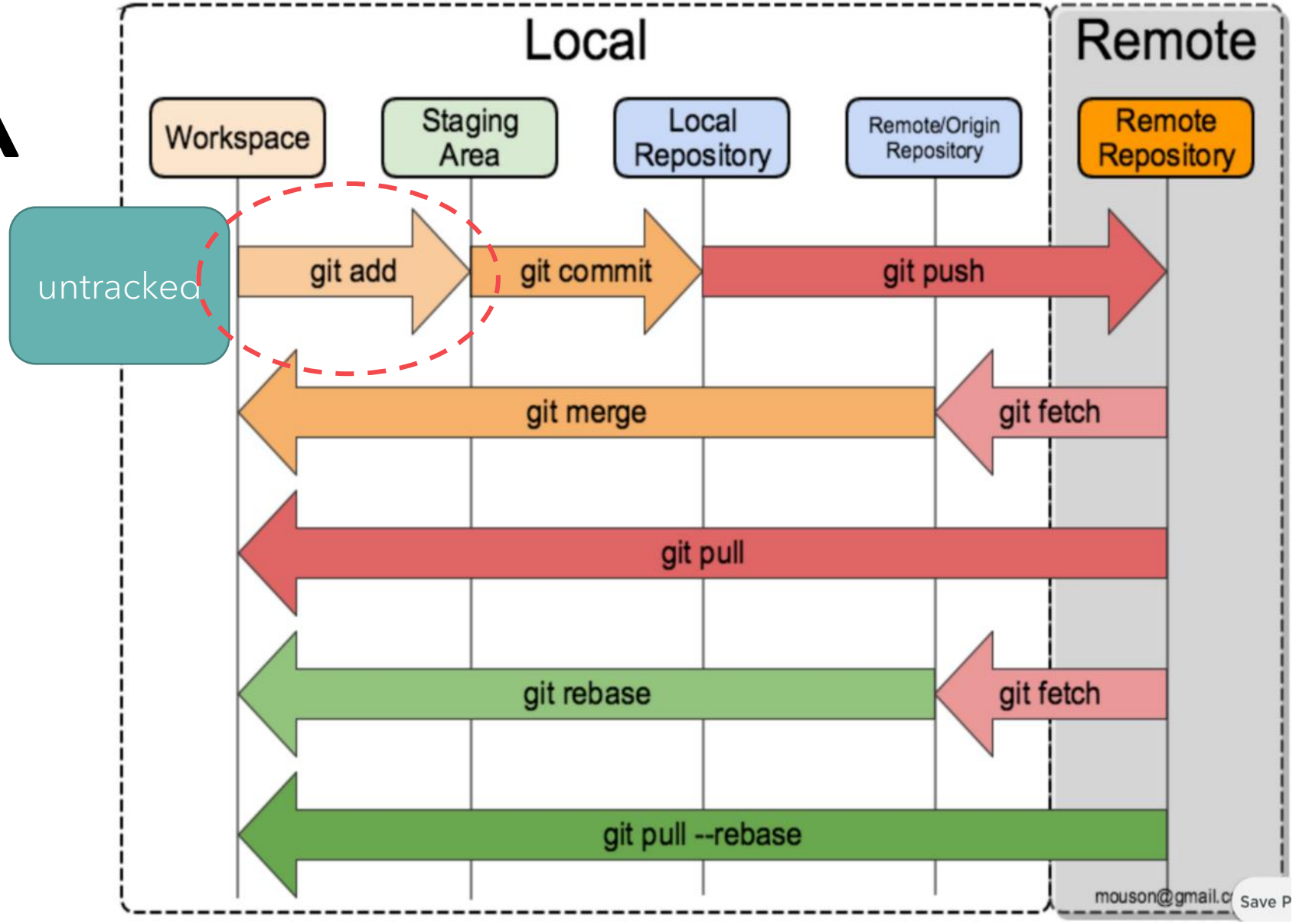
Toda vez que fizermos uma modificação no(s) arquivo(s), temos que fazer novamente a adição desse arquivo.

git add .

Adicionamos todos os arquivos
ou

git add arquivo.py

Adicionamos um arquivo específico



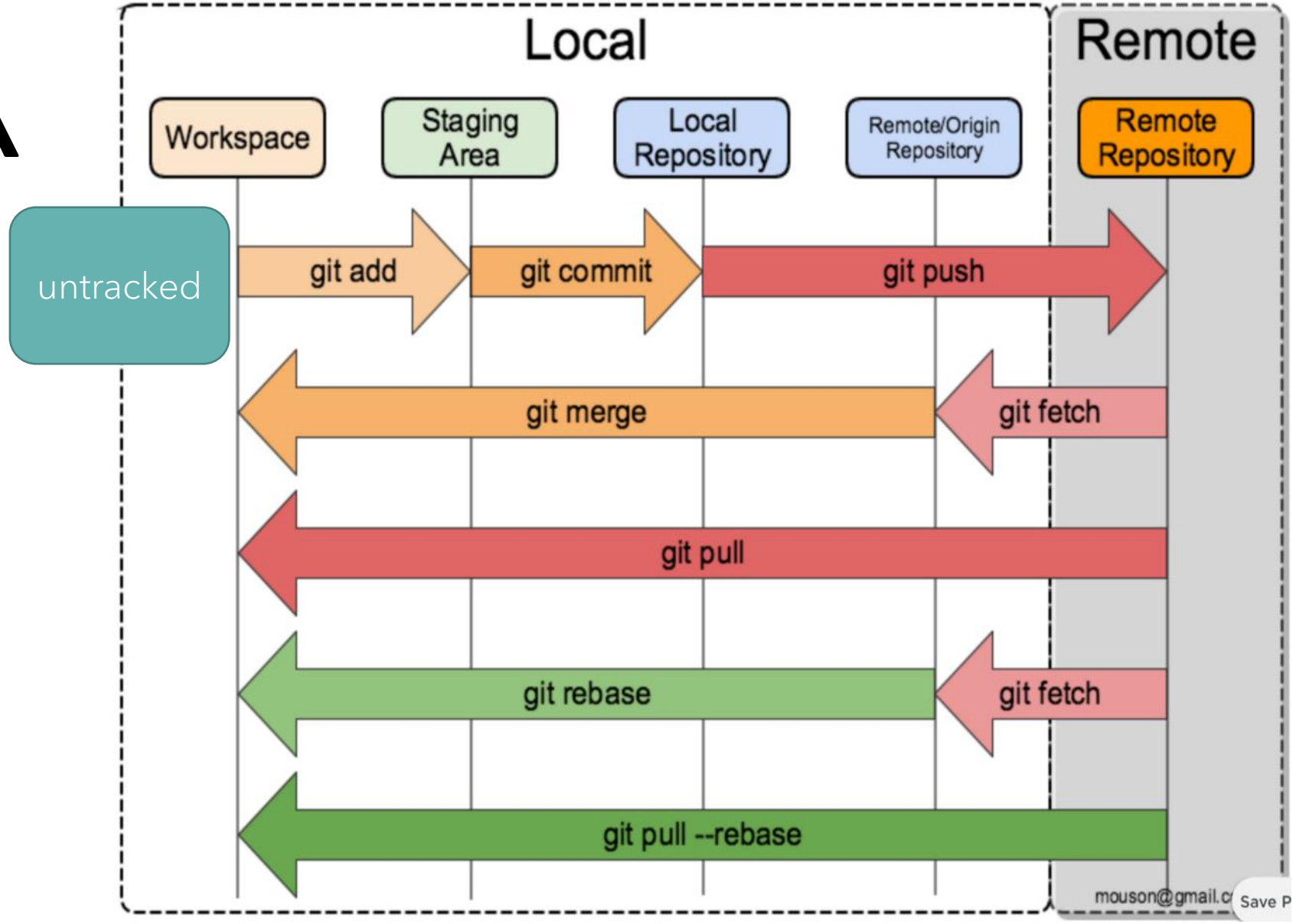
CICLO DE VIDA

Para verificar o STATUS do processo de gerenciamento de versionamento dos arquivos utilize o comando:

git status

Utilize sempre esse comando para ver como está o processo.

git status é um de seus melhores amigos no versionamento.



CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

.gitignore usa padrões globbing

Padrão	Exemplo de correspondências	Explicação*
**/logs	logs/debug.log logs/monday/foo.bar build/logs/debug.log	Você pode prefixar um padrão com um asterisco duplo para combinar diretórios em qualquer lugar no repositório.
*.log	debug.log foo.log .log logs/debug.log	Um asterisco é um curinga que corresponde a zero ou mais caracteres.
debug?.log	debug0.log debugg.log *but not* debug10.log	Um ponto de interrogação corresponde exatamente a um caractere.

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

.gitignore usa padrões globbing

Padrão	Exemplo de correspondências	Explicação*
debug[0-9].log	debug0.log debug1.log *but not* debug10.log	Os colchetes também podem ser usados para corresponder a um único caractere de um intervalo especificado.
debug[!01].log	debug2.log *but not* debug0.log debug1.log debug01.log	O ponto de exclamação pode ser usado para estabelecer correspondência com qualquer caractere, exceto um do conjunto especificado.
debug[a-z].log	debuga.log debugb.log *but not* debug1.log	Os intervalos podem ser numéricos ou alfabéticos.

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

.gitignore usa padrões globbing

Padrão	Exemplo de correspondências	Explicação*
logs	logs logs/debug.log logs/latest/foo.bar build/logs build/logs/debug.log	Se você não anexar uma barra, o padrão vai combinar os arquivos e o conteúdo dos diretórios com esse nome. Na correspondência de exemplo à esquerda, os diretórios e arquivos chamados *logs* são ignorados
logs/	logs/debug.log logs/latest/foo.bar build/logs/foo.bar build/logs/latest/debug.log	Acrescentar uma barra indica que o padrão é um diretório. Todo o conteúdo de qualquer diretório no repositório que corresponda a esse nome – incluindo todos os seus arquivos e subdiretórios – vai ser ignorado

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

.gitignore usa padrões globbing

Padrão	Exemplo de correspondências	Explicação*
ogs/**/debug.log	logs/debug.log logs/monday/debug.log logs/monday/pm/debug.log	Um asterisco duplo corresponde a zero ou mais diretórios.
ogs/*day/debug.log	logs/monday/debug.log logs/tuesday/debug.log *but not* logs/latest/debug.log	Os curingas também podem ser usados em nomes de diretórios.
logs/debug.log	logs/debug.log *but not* debug.log build/logs/debug.log	Os padrões que especificam um arquivo em um determinado diretório são relativos à origem do repositório. (Você pode prefixar uma barra se quiser, mas não acontece nada de especial.)

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

Exemplo de arquivo .gitignore p/ Python

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

```

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

Exemplo de arquivo .gitignore p/ Python

```
# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

```

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

Exemplo de arquivo .gitignore p/ Python

```
# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since the code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version

```

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

Exemplo de arquivo .gitignore p/ Python

```
# pdm
#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.
#pdm.lock
#   pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include it
#   in version control.
#   https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

```

CICLO DE VIDA

O processo de rastreamento é feito em todos os arquivos que estiverem dentro do diretório especifico no início do processo com o comando **git init**.

Todavia, existe como falar para o GIT ignorar o rastreamento de alguns arquivos.

Isso é feito utilizando o arquivo oculto:

.gitignore

Exemplo de arquivo .gitignore p/ Python

```
# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can
#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.  For a more nuclear
#  option (not recommended) you can uncomment the following to ignore the entire idea folder.
#.idea/
```

TAREFAS...

- Crie um diretório (do seu projeto)
- Inicialize o GIT neste diretório.
- Crie um ou mais arquivos fontes neste diretório
- Faça a adição desses arquivos
- Depois faça o commit (para o repositório local).

OBRIGADO

Prof. Dr. Dilermando Piva Jr.

https://piva.pro.br

piva.jr@fatec.sp.gov.br

https://pypro.com.br