

GIT E GITHUB

Gabriel Avila



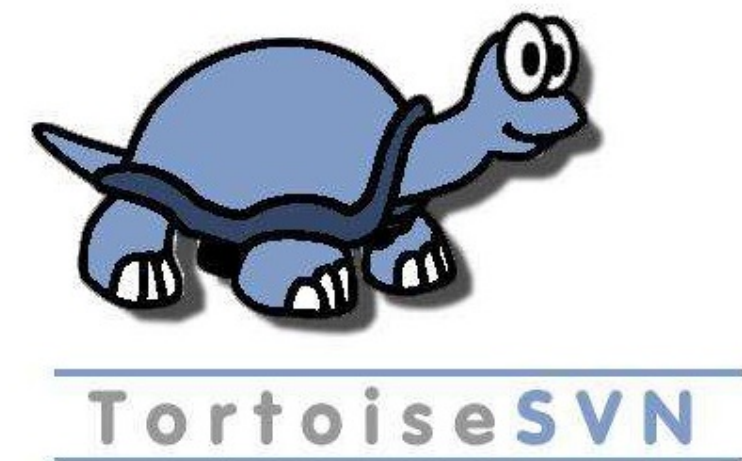
SVN

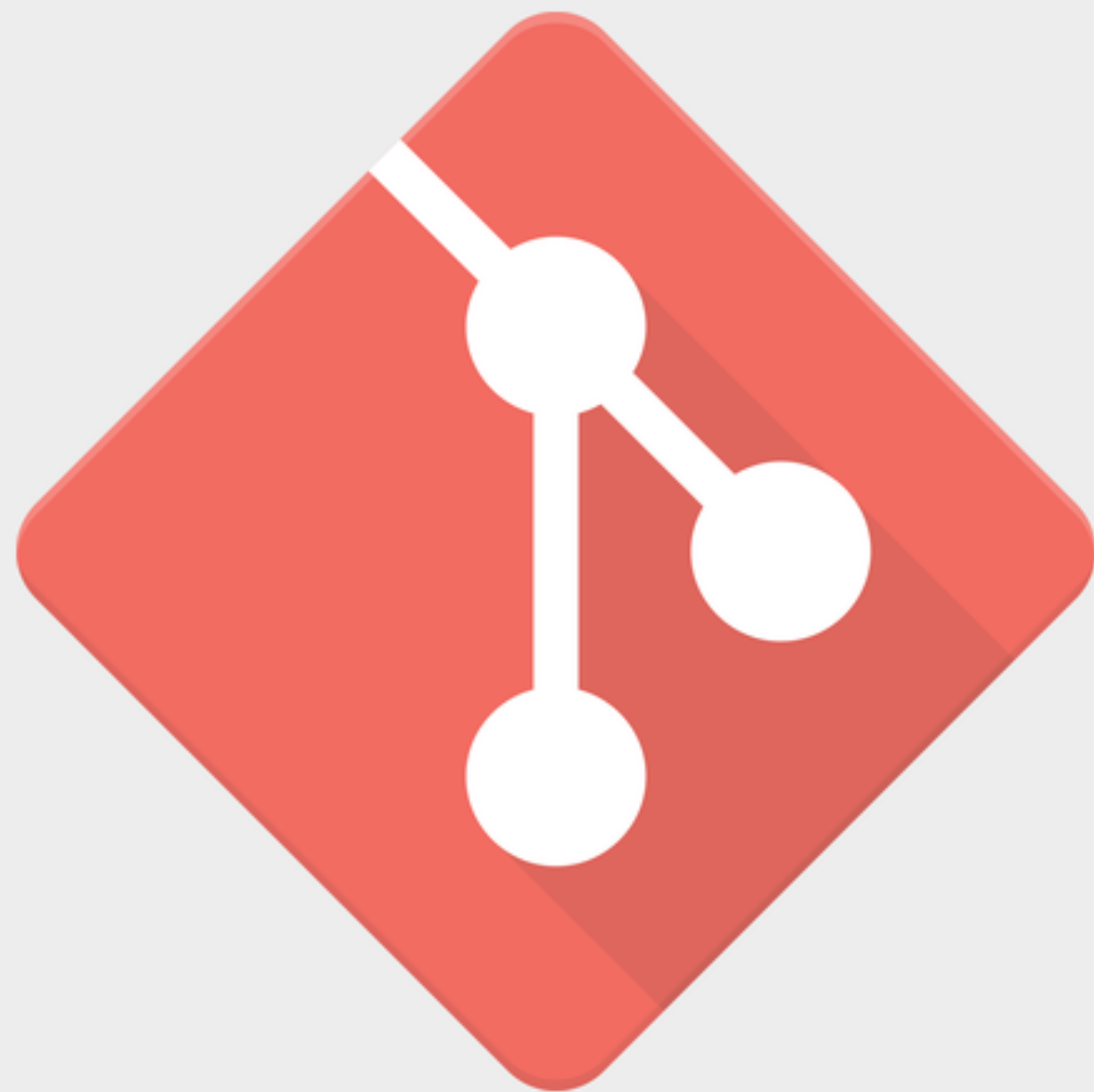
Old School.

Subversion (SVN) é um sistema de controle de versão também. Baseado na existencia de um repositório, que mantém todo o histórico e versões.

Para trabalhar com o SVN precisa ter uma cópia no seu workspace, assim poderá enviar alterações em arquivos.

Pela praticidade de usar Git, hoje em dia é quase inaceitável utilizar SVN.





--- GIT

O Git é um sistema de controle de versão distribuído, que permite que indivíduos e equipes controlem as mudanças em seu código ao longo do tempo. Ele é usado em conjunto com plataformas como o GitHub para permitir o compartilhamento e colaboração em projetos de software.

Principais plataformas que usam GIT:

- GitHub
- GitLab
- BitBucket





GITHUB

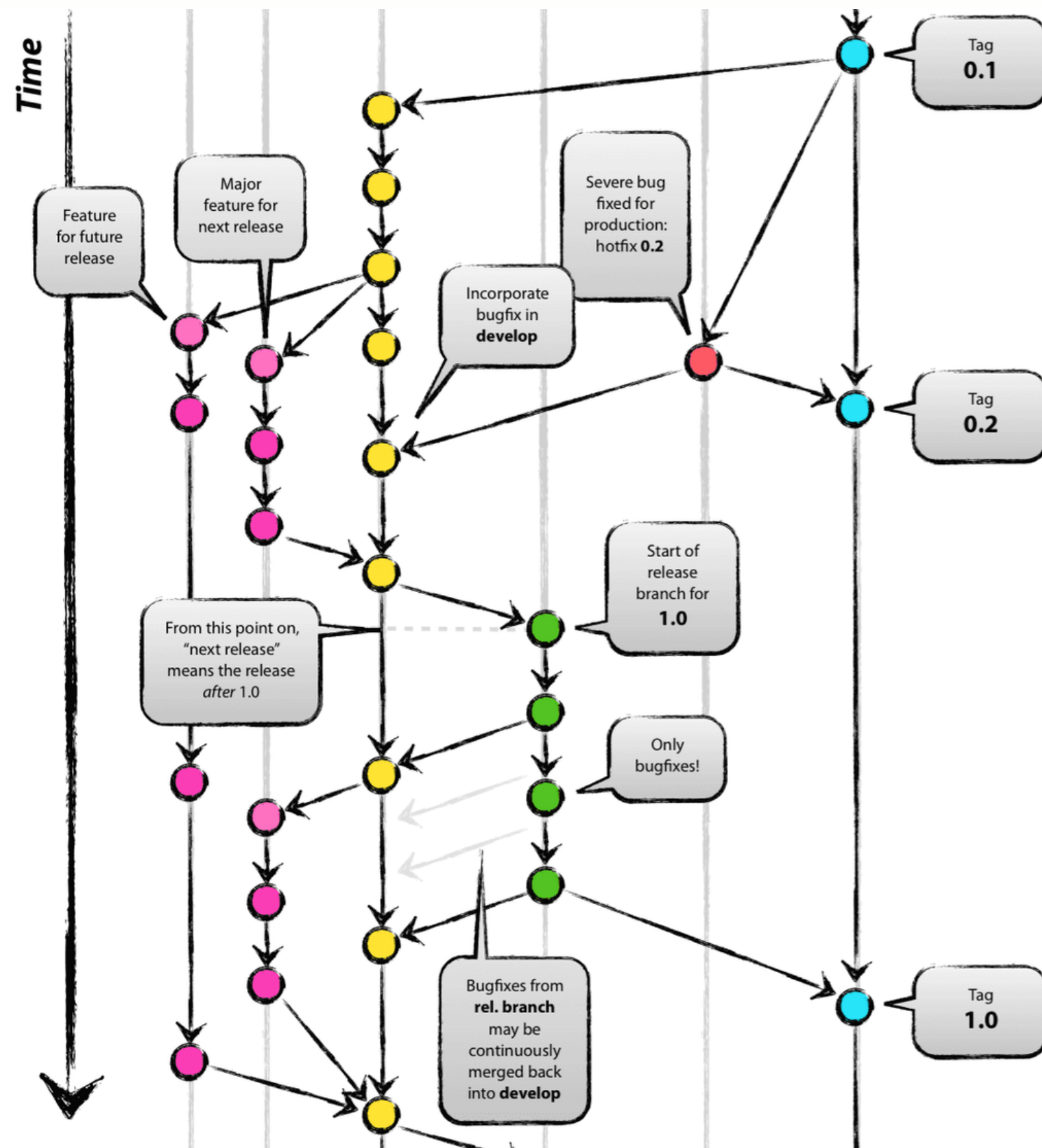
O GitHub é uma plataforma de hospedagem de código-fonte e colaboração que permite a indivíduos e equipes compartilhar e colaborar em projetos de software.

Ele fornece controle de versão usando o Git, bem como recursos adicionais como gerenciamento de problemas, controle de acesso, gestão de tarefas e wikis.

Atualmente muito mais que código, uma rede social?

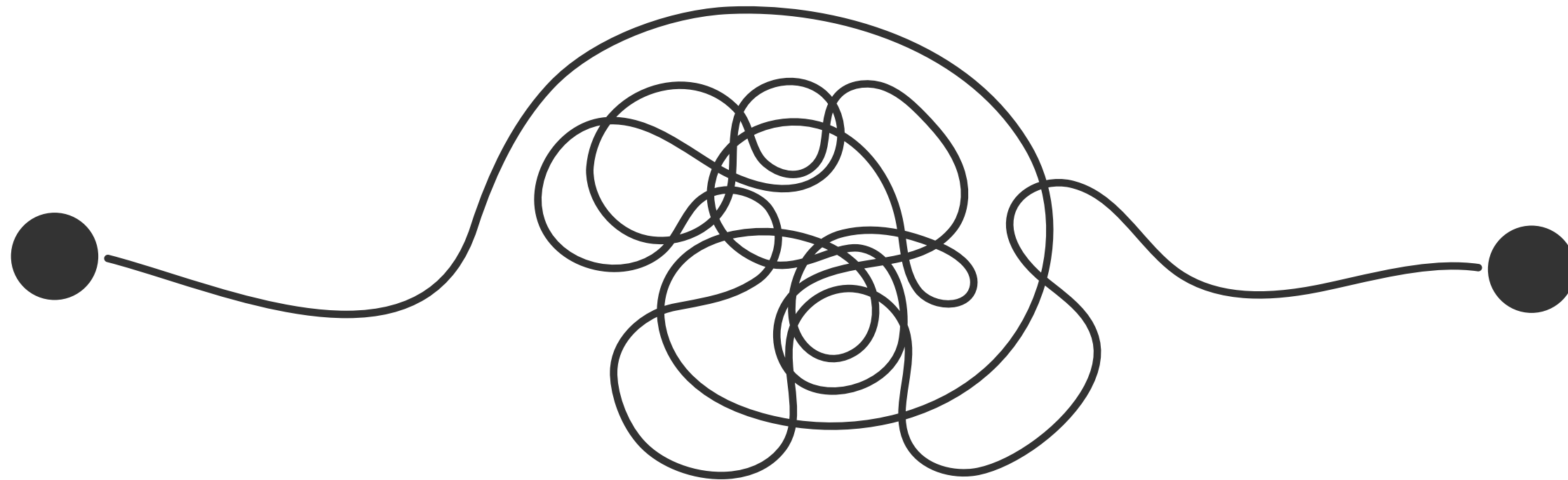
GitHub > Linkedn ??

Fato é todos devemos ter um github bem feitinho.



PALAVRAS CHAVE

- **Repositório:** é o local onde os arquivos e o histórico de um projeto são armazenados.
- **Commit:** é uma alteração no projeto que é registrada no histórico. Ele tem uma mensagem e é identificado por um código hash exclusivo.
- **Branch:** é uma ramificação independente do histórico de commits do Git.
- **Merge:** é o processo de combinar duas ou mais branches do Git em um único histórico de commits.
- **Pull Request:** é uma solicitação para que as alterações feitas em um branch de um repositório sejam mescladas ao branch principal.
- **Clone:** é a ação de criar uma cópia local de um repositório Git.
- **Push:** é a ação de enviar as alterações locais feitas em um repositório Git para um repositório remoto.
- **Pull:** é a ação de baixar as alterações mais recentes de um repositório.
- **Fork:** é a ação de criar uma cópia completa de um repositório Git hospedado em um servidor remoto.
- **Feature:** nome de branch comumente utilizado para definir novas implementações.
- **Fix/Hotfix:** diz respeito ao nível do erro encontrado e a urgência dele para liberação.
- **Release:** produto pronto para liberação, em fase de testes ainda.



— DESAFIO FUNCIONAL

Criar a conta ou rever a sua já existente.

Instalando o que precisa, e escolhendo o melhor Git Client ou mesmo o próprio GIT.

1. Conhecer a interface
2. Criando nosso primeiro projeto
3. Trazendo ele pro local workspace
4. Já vamos de primeiros comandos

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

GIT THE RIGHT WAY

Vamos de links:

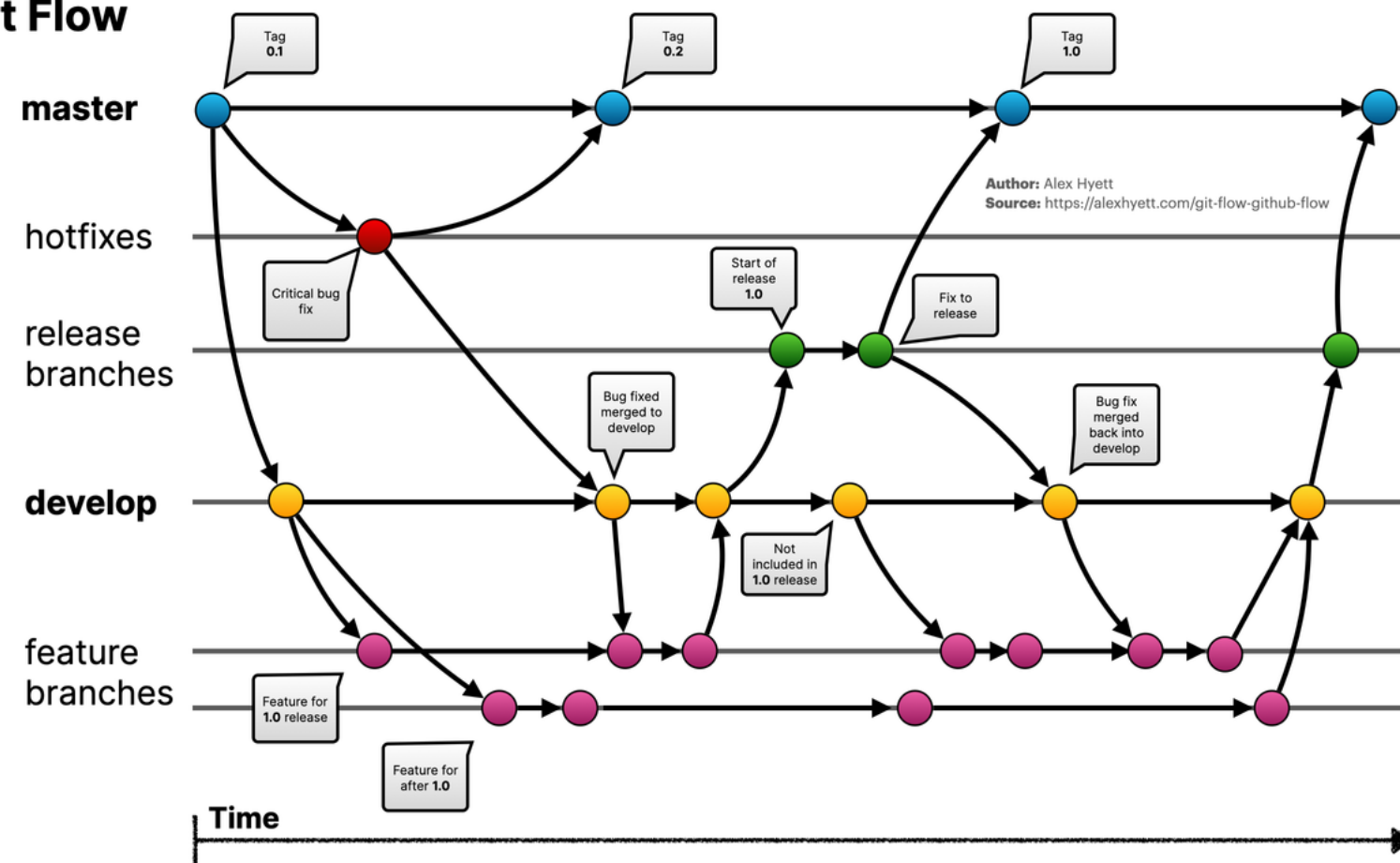
https://danielkummer.github.io/git-flow-cheatsheet/index.pt_BR.html

<https://cbea.ms/git-commit/>

Onde podemos organizar e padronizar o GIT e seu uso, fica por conta de 2 pontos:

- GIT FLOW -> Onde temos uma característica específica de como trabalhar com branches e fluxos.
- COMMIT -> Dentro do commit temos a possibilidade de trazer um texto livre, mas precisamos seguir algumas regras para melhorar e tornar bem intuitivo.

Git Flow



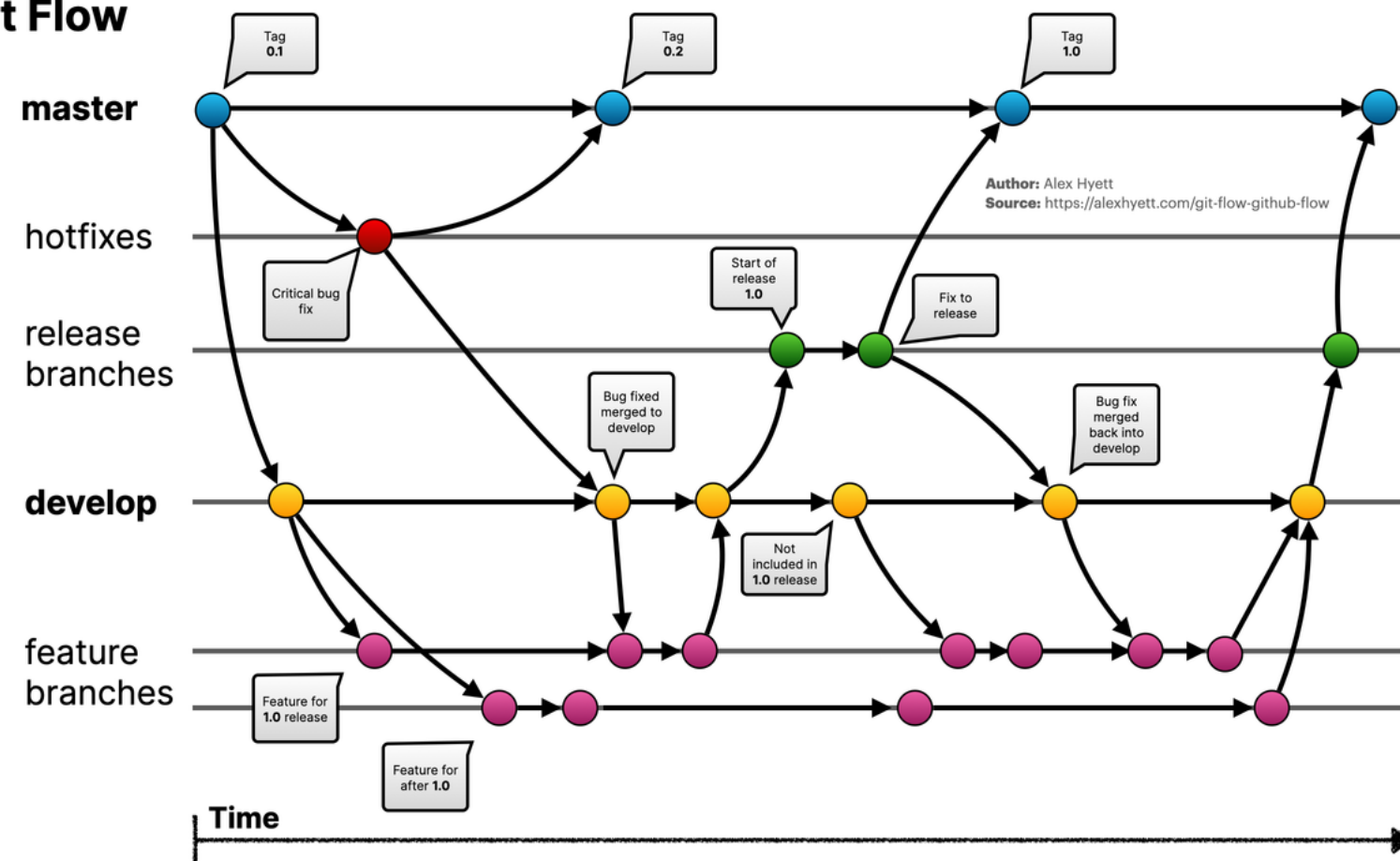
GIT FLOW

Git flow é um modelo de fluxo de trabalho (workflow) para o gerenciamento de branches em projetos de desenvolvimento de software com o Git, que tem como objetivo fornecer uma estrutura organizada e consistente para a criação e manutenção de branches em um projeto.

O modelo Git flow é baseado em duas branches principais: a branch **master**, que contém o código estável e pronto para produção, e a branch **develop**, que contém o código em desenvolvimento.

- Feature: criada a partir da branch develop para desenvolver novas funcionalidades;
- Release: criada a partir da branch develop para preparar a próxima versão de produção;
- Hotfix: criada a partir da branch master para corrigir problemas críticos na produção;
- Support: criada a partir da branch master para fornecer suporte a versões legadas do software.

Git Flow



GIT FLOW

Vantagens:

1. Estrutura mais organizada;
2. Melhor controle de qualidade;
3. Fácil manutenção de versões;
4. Integração contínua;

Desvantagens:

1. Complexidade elevada, precisa que todos conheçam sobre o assunto;
2. Branches de mais para serem controladas;
3. Dificuldade na manutenção, é moroso a liberação de hotfix e muitas vezes pode causar problemas;
4. Demora na entrega, é necessário seguir o processo corretamente;
5. Não é adequado para entrega contínua, pois não necessariamente o fluxo é o melhor para liberação;
6. Devido à sobrecarga necessária para liberar, isso pode levar a um acúmulo de dívida técnica;

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO


GIT COMMIT

Vamos ao jeito certo.

Quais são as dicas:

1. Separar o título do corpo do commit;
2. Limite de 50 caracteres no título do commit;
3. Utilizar as letras maiúsculas que forem necessárias;
4. Não utilizar pontos;
5. Utilizar o modo imperativo no título;
6. Quebrar linhas no corpo do commit;
7. Utilizar a mensagem para explicar o que foi feito e como foi feito;

PRINCIPAIS COMANDOS



git CHEAT SHEET

INITIALIZATION \$ git init Create or reinitialize an repository \$ git clone ["url"] Clone a repository	MERGE & REBASE \$ git merge <branch> Merge <branch> into your current HEAD \$ git rebase <branch> Rebase your current HEAD onto <branch> \$ git rebase --abort Abort rebase \$ git rebase --continue Continue a rebase after resolving conflicts	BRANCHES \$ git branch List all branches \$ git checkout <branch> Switch HEAD branch \$ git branch <new-branch> Create a new branch based on your current \$ git branch -d <branch> Delete a local branch
LOCAL CHANGES \$ git status Changes files in your working directory \$ git diff Changes to track files \$ git add . Add all current changes to next commit \$ git add -p <file> Add some changes in <file> to next commit \$ git commit -a Commit all local changes in tracked files \$ git commit Commit previously staged changes \$ git commit --amend Change last the commit	UNDO \$ git reset --hard HEAD Discard all local changes in your working directory \$ git checkout HEAD <file> Discard all local changes in a specific file \$ git revert <commit> Revert a commit (by producing a new commit with contrary changes)	UPDATE E PUBLISH \$ git fetch Download all changes from <remote>, but don't integrate into HEAD \$ git pull <remote> <branch> Download changes and directly merge/integrate into HEAD \$ git push <remote> <branch> Publish local changes on a remote branch



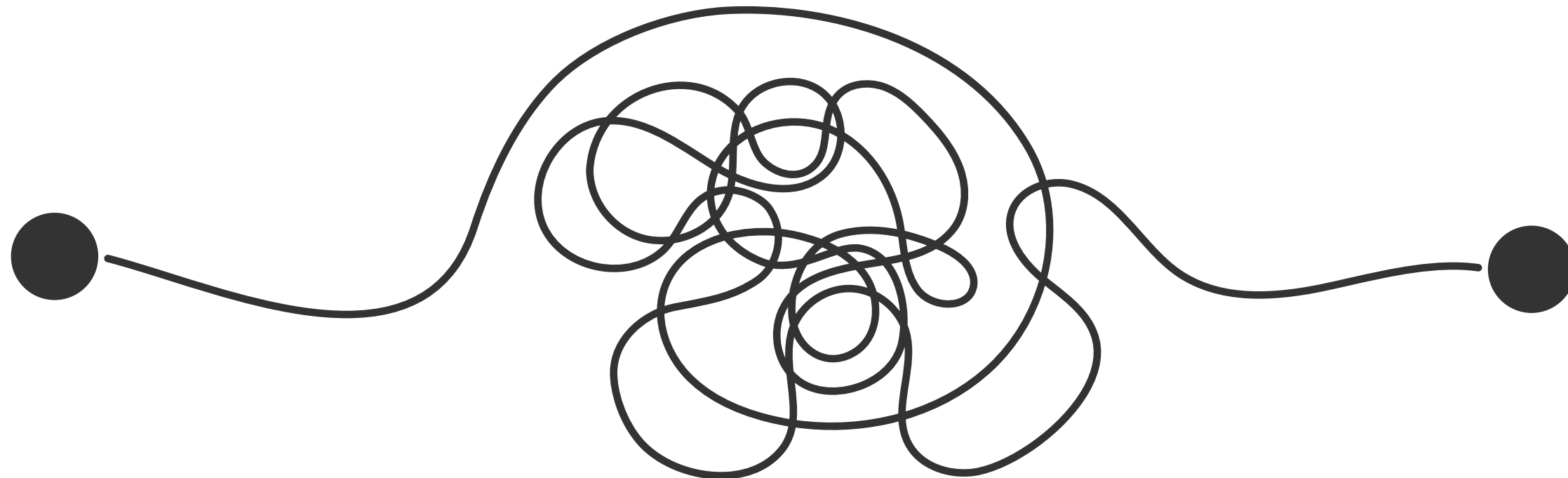
PRÁTICA DE CODE REVIEW

Code review é a pratica de revisar o código do coleguinha, sendo o principal objetivo, encontrar erros, apontar possíveis melhorias de qualidade e/ou performance, ou mesmo de boas práticas de programação.

Tudo isso visando compartilhar conhecimento dentro da equipe.

Mas e isso dentro do GIT, como fazemos?

Pull Request... Bora ver



— VAMOS A PRATICA

HANDS ON