GERENCIAMENTO DE CONFIGURAÇÃO

Professor Paulo Honório

Aula - 05/08/2023

Agenda

- Apresentação
- Introdução
- Problemas corriqueiros
- Gerenciamento de configuração
- Funções básicas
- Terminologias
- Atividades do processo
- Mitos
- Ferramentas
- Benefícios de GC
- Questionário
- Práticas
- Guias
- Extra

Apresentação

- Paulo Honório Filho
  - *Doutorando no Programa de Pós-Graduação em Engenharia Elétrica na UFC*
  - *Pesquisador no Laboratório de Condicionadores de Energia*
    https://lce.ufc.br
  - ***Lattes:*** *https://lattes.cnpq.br/7658949372926009*
  - ***LinkedIn:*** *https://www.linkedin.com/in/paulo-hon%C3%B3rio-filho-4827236b/*

MUDANÇAS NO DESENVOLVIMENTO DE SOFTWARES

Introdução

- A certeza que temos de considerar é que, na programação de sistemas, as mudanças sempre irão ocorrer:
  - *A legislação mudas, dentre outros motivos*
  - *Os interesses de quem está adquirindo a solução muda*
  - *O ambiente em o programa irá rodar muda*
  - *O entendimento dos usuários sobre suas necessidades muda*
- Tais mudanças precisam ser acompanhadas para não gerar desordem durante o desenvolvimento do software e em toda a vida útil dele.

Problemas corriqueiros

- O erro corrigido ontem, apareceu novamente hoje!
- A versão entregue ao cliente não foi a versão correta!
- O que você está desenvolvendo agora, já está pronto...
- Não estou vendo a sua correção
- Quais as novas funcionalidades, desde a última versão?
- Quem fez a correção na funcionalidade XPTO?

Problema da Quebra de Comunicação

Problema da Quebra de Comunicação

- Falhas de comunicação nos times de desenvolvimento
- Principais causas:
  - Culturas diferentes, Incompatibilidade de vocabulários
  - Equipes geograficamente distribuídas
  - Conhecimento técnico, Dificuldade de expressão
- Consequências
  - Os sistemas produzidos não atendem aos requisitos
  - Desperdício da força de trabalho

Problema dos Dados Compartilhados

Desenvolvedor A

Desenvolvedor B

Problema dos Dados Compartilhados

- De acordo com o cenário:
  1. O desenvolvedor **A** modifica o componente compartilhado
  2. Mais tarde, o desenvolvedor **B** realiza alterações no mesmo
  3. Ao tentar compilar o componente, erros são apontados pelo compilador, mas nenhum deles ocorre na parte que **B** alterou
  4. O desenvolvedor **B** não tem a menor ideia sobre a causa do problema

Problema dos Dados Compartilhados

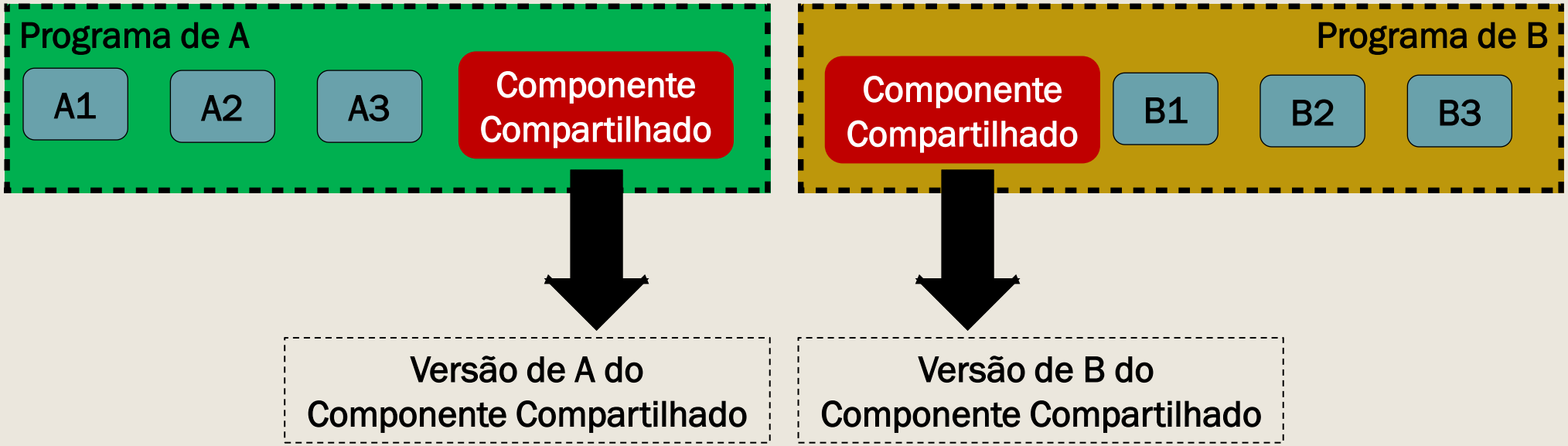
- Solução:
  - Cada desenvolvedor trabalha em uma cópia "local" do componente

- Consequência:
  - Resolve o Problema dos Dados Compartilhados, porém cria um novo problema

Problema da Manutenção Múltipla

Desenvolvedor A

Desenvolvedor B



Problema da Manutenção Múltipla

- Ocorre quando cada desenvolvedor trabalha com uma cópia "local" do que seria o mesmo componente

- Dificuldade para saber:

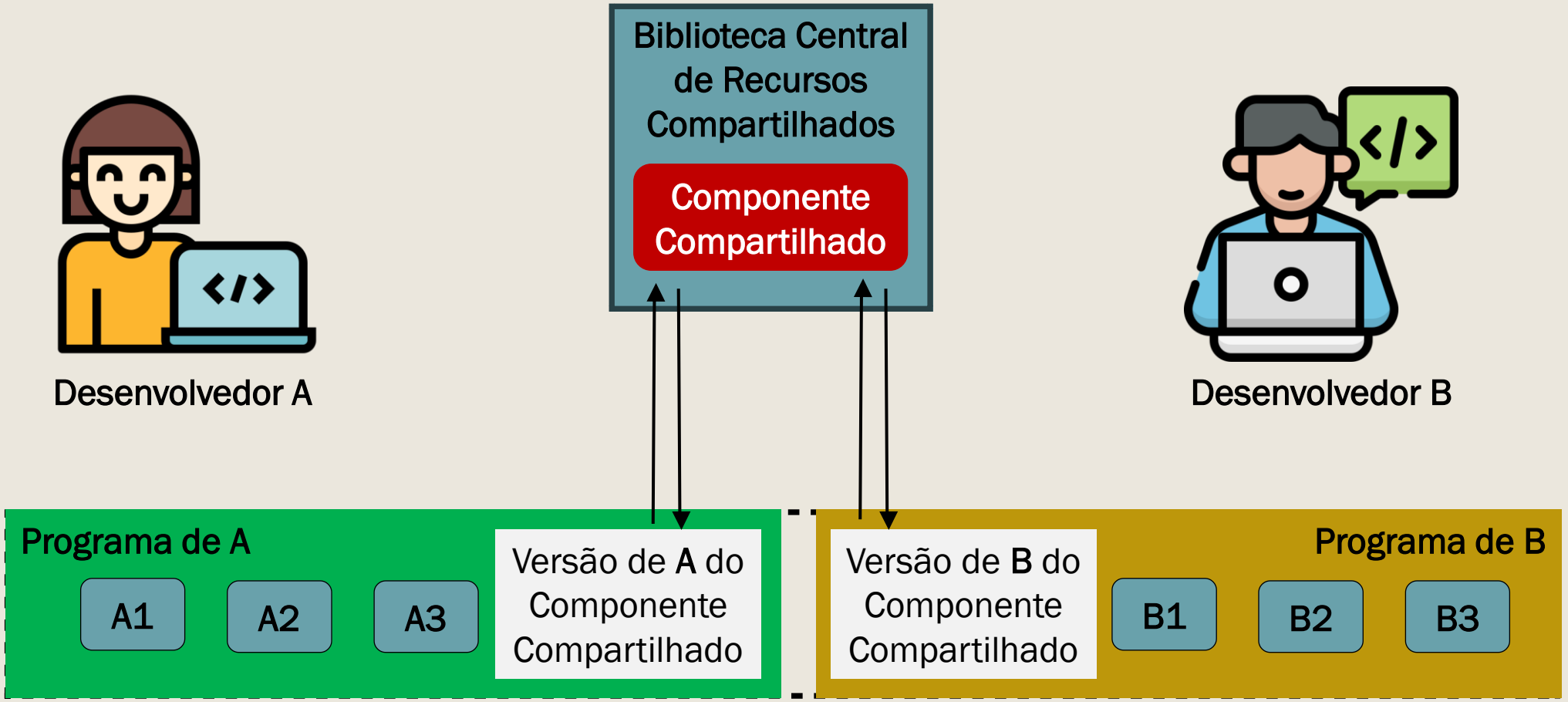
  - Que funcionalidades foram implementadas em quais versões do componente?

  - Que defeitos foram corrigidos?

Problema da Manutenção Múltipla

- Solução:
  - Biblioteca central de componentes compartilhados
  - Uso: Cada componente é copiado para a biblioteca sempre que alterado
- Consequência:
  - Cria um novo problema: Atualização Simultânea

Problema da Atualização Simultânea



Problema da Atualização Simultânea

- De acordo com o primeiro cenário
  - O desenvolvedor **A** encontra e corrige um defeito em sua versão do componente compartilhado
  - Uma vez corrigido, o componente modificado é copiado para a biblioteca central
  - O desenvolvedor **B** encontra e corrige o mesmo defeito em sua versão do componente, por não saber que **A** já havia corrigido
  - O trabalho de **A** é desperdiçado

Problema da Atualização Simultânea

- De acordo com o segundo cenário
  - O desenvolvedor **A** encontra e corrige um defeito em sua versão do componente compartilhado
  - Uma vez corrigido, o componente modificado é copiado para a biblioteca central
  - O desenvolvedor **B** encontra e corrige um outro defeito em sua versão do componente, sem saber do defeito corrigido por **A**

Problema da Atualização Simultânea

- Ainda no segundo cenário:
  - O desenvolvedor B copia sua versão do componente para a biblioteca central
  - Além de o trabalho de A ser desperdiçado, a versão do componente que se encontra na biblioteca central continua apresentando um defeito
  - O desenvolvedor A julga o problema como resolvido

Qual seria a solução?

- O Problema da Atualização Simultânea não pode ser resolvido simplesmente copiando componentes compartilhados para uma biblioteca central
- Algum mecanismo de controle é necessário para gerenciar a entrada e saída dos componentes

GERENCIAMENTO DE CONFIGURAÇÃO
OU
CONFIGURATION MANAGEMENT - CM

Gerenciamento de Configuração

- É um conjunto de atividades de apoio que permite a absorção controlada das mudanças inerentes ao desenvolvimento de software, mantendo a estabilidade na evolução do projeto

"É um conjunto de atividades projetadas para controlar as mudanças pela identificação dos produtos do trabalho que serão alterados, estabelecendo um relacionamento entre eles, definindo o mecanismo para o gerenciamento de diferentes versões destes produtos, controlando as mudanças impostas, e auditando e relatando as mudanças realizadas."

MAXIM, B. R.; PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional. 2021.

Oque é CM?

- Em outras palavras... Gerenciamento de Configuração objetiva responder as seguintes questões:
  - O que mudou e quando?
  - Por que mudou?
  - Quem fez a mudança?
  - Podemos reproduzir esta mudança?
  - O sistema continua íntegro mesmo depois das mudanças?

Funções básicas

Identificação

Documentação

Controle

Auditoria

Terminologias

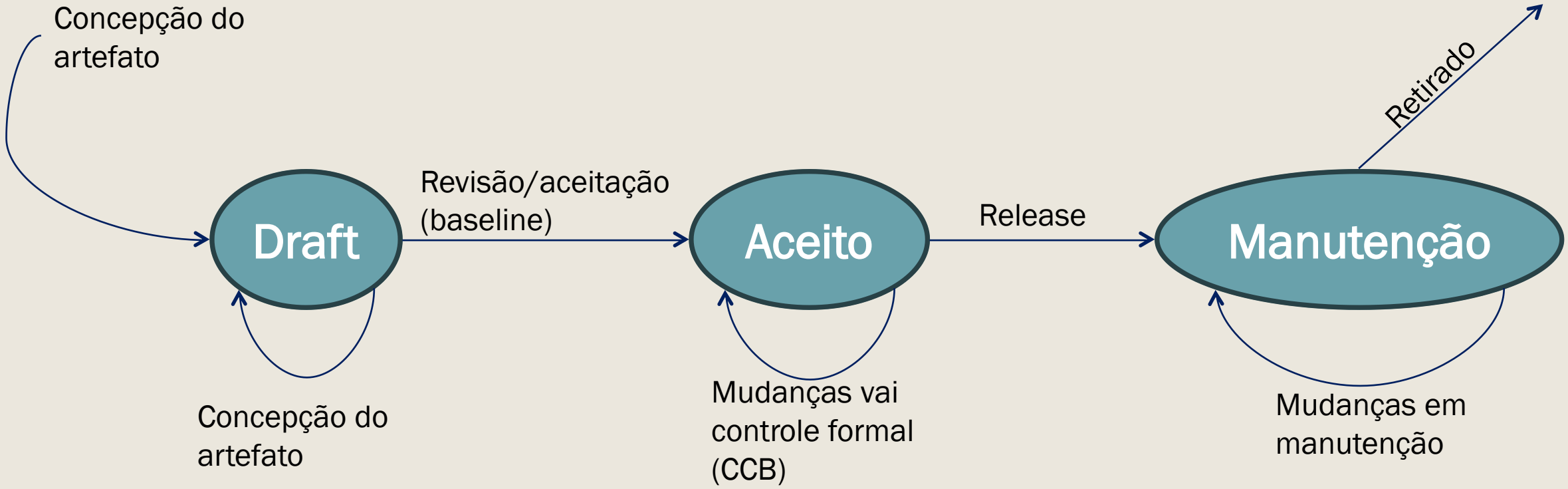
- Item de configuração, Baseline
- Commit
- Repositório, Tag
- Trunk/Branch, Merge
- Build, Release

Item de Configuração (IC)

- Durante o desenvolvimento, todos os elementos produzidos que precisam sofrer controle de versões e de mudanças são os ICs

- Todo IC deve ser gerenciado. Ex:

Item de Configuração (IC)



Baseline

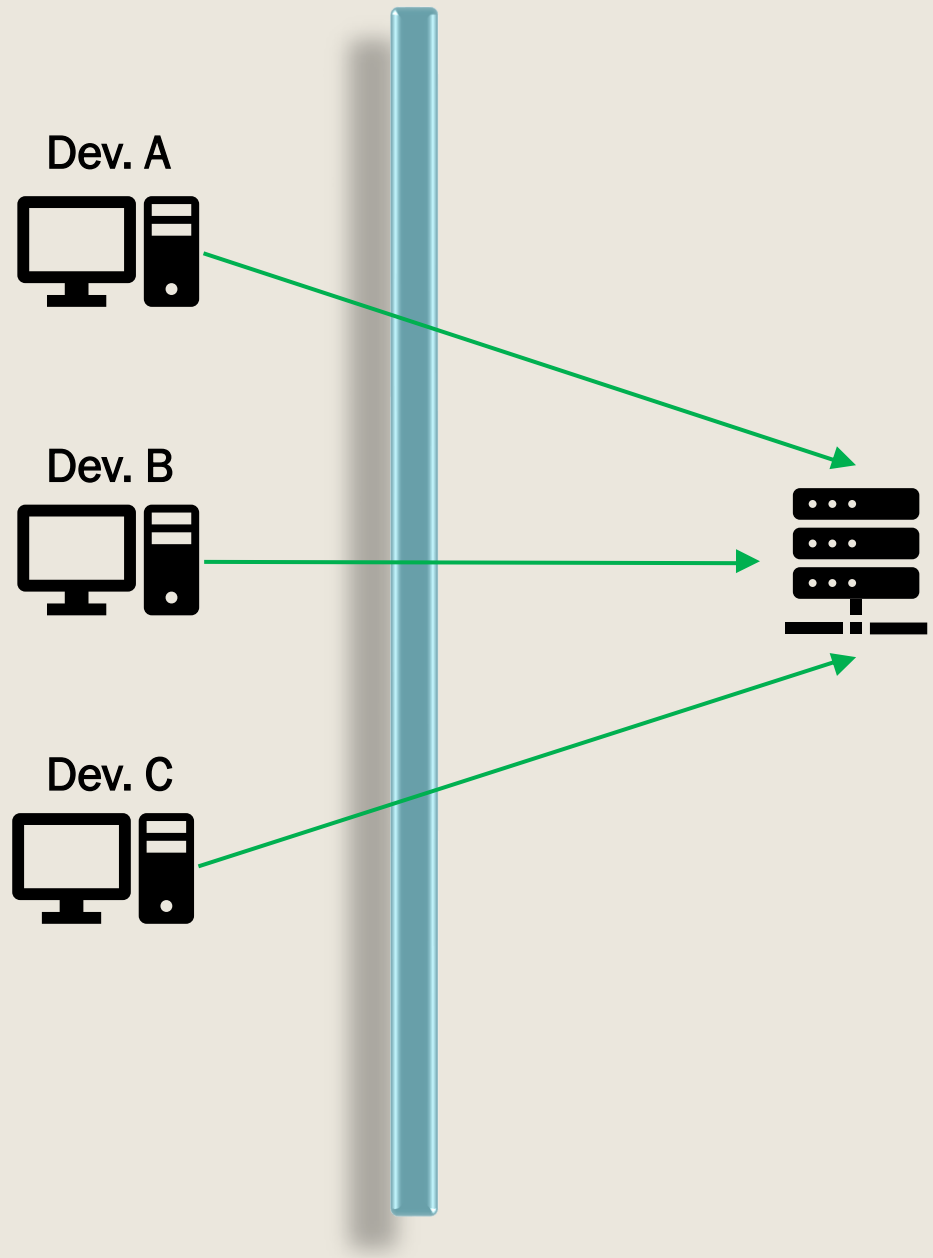
- Uma ‘imagem’ de uma versão de cada artefato no repositório do projeto de desenvolvimento de software
- Um marco de referência (checkpoint)
- Caracterizado pela entrega de um ou mais itens de configuração
- Reprodutibilidade, rastreabilidade e elaboração de relatórios

Ex: versão 1.0

Repositório

- Local (físico e lógico) onde os ICs são armazenados
- Pode conter diversas versões do sistema
- Utiliza mecanismos de controle de acesso

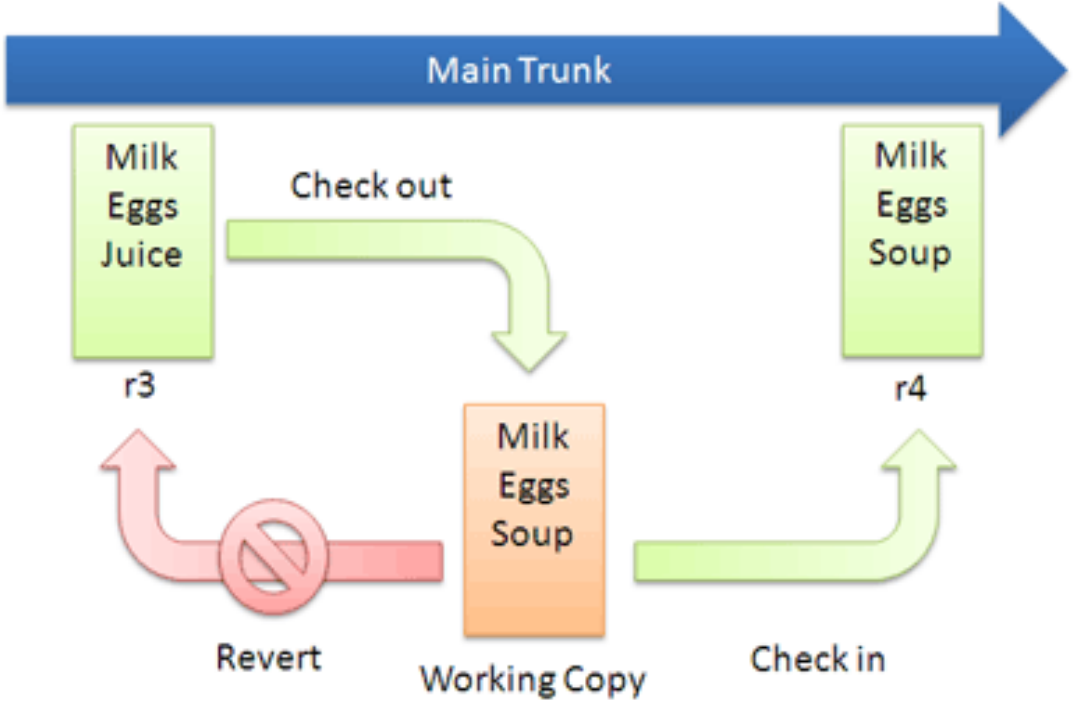
Repositório



Repositório

- Lock
  - *Resolve o problema de atualização simultânea*
  - *Garante que apenas o usuário que detém o lock pode alterar o arquivo*
  - *Problema: serializa o desenvolvimento*

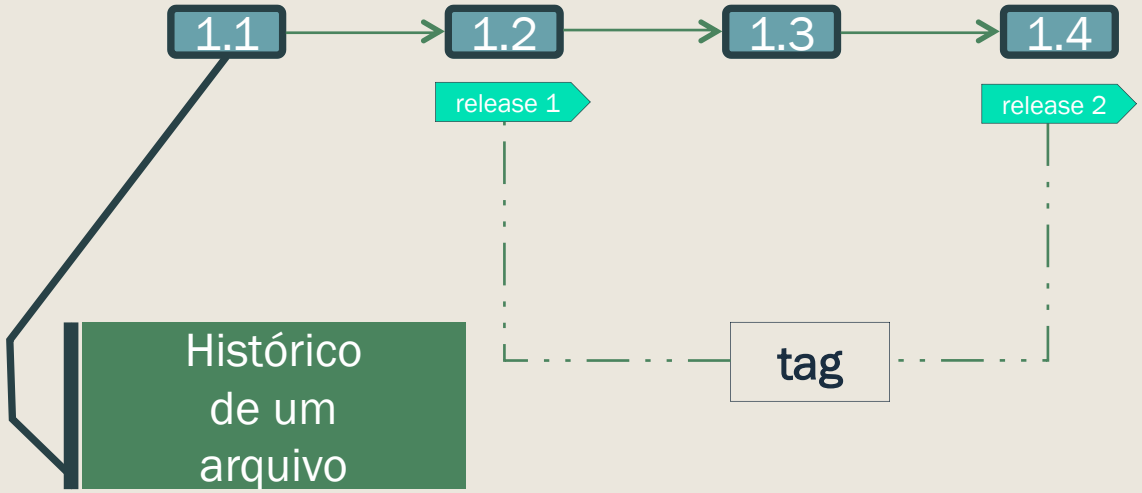
Check-ing & Check-out



REPOSITÓRIO

Tag

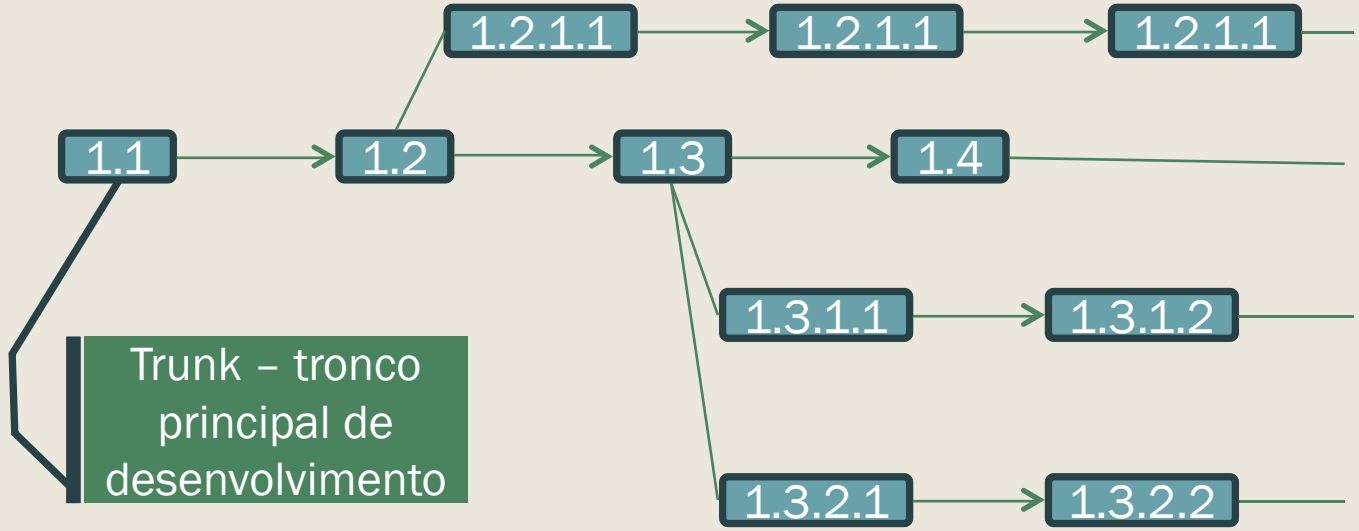
- Rótulos que são associados a conjuntos de arquivos
- Um **tag** referência um ou mais arquivos em um ou mais diretórios



Trunk/Branch

- Trunk
  - *O trunk deve ser a base do projeto, no qual o desenvolvimento tem progresso*
- Branch
  - *São ramificações laterais de versões que se originam de uma revisão da linha principal de desenvolvimento (trunk)*

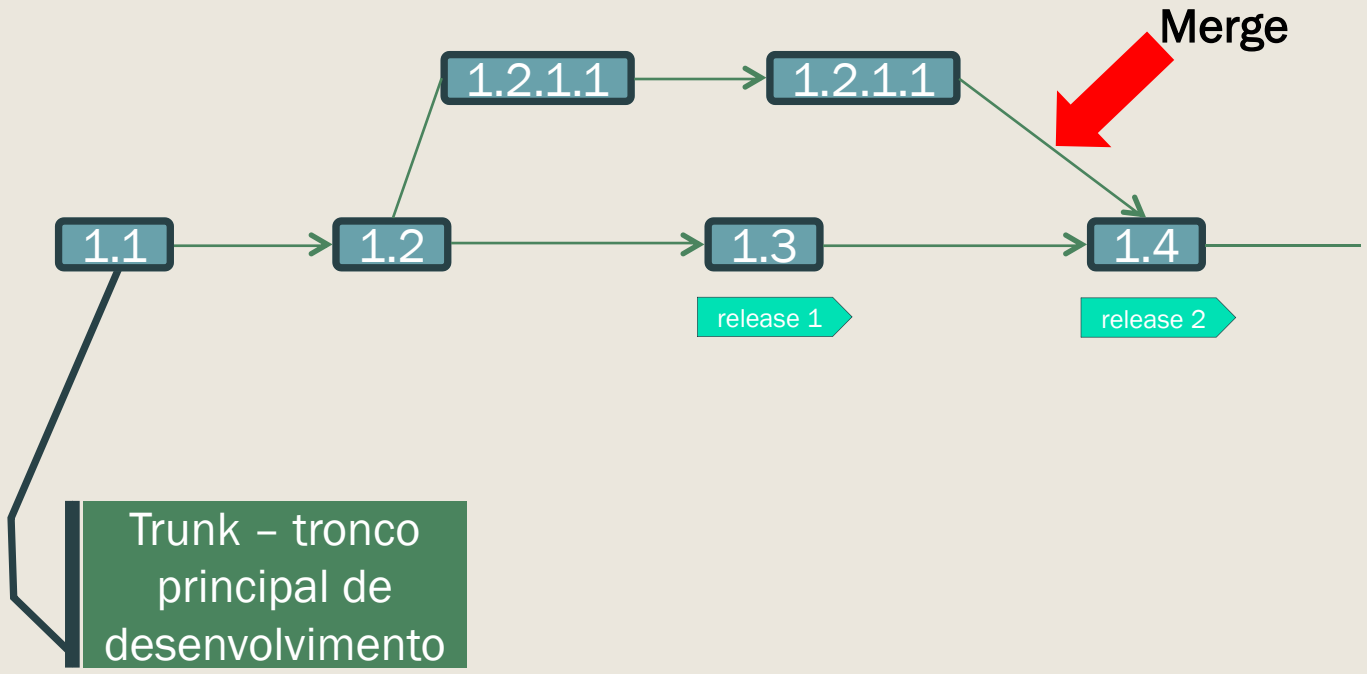
Trunk/Branch



Merge

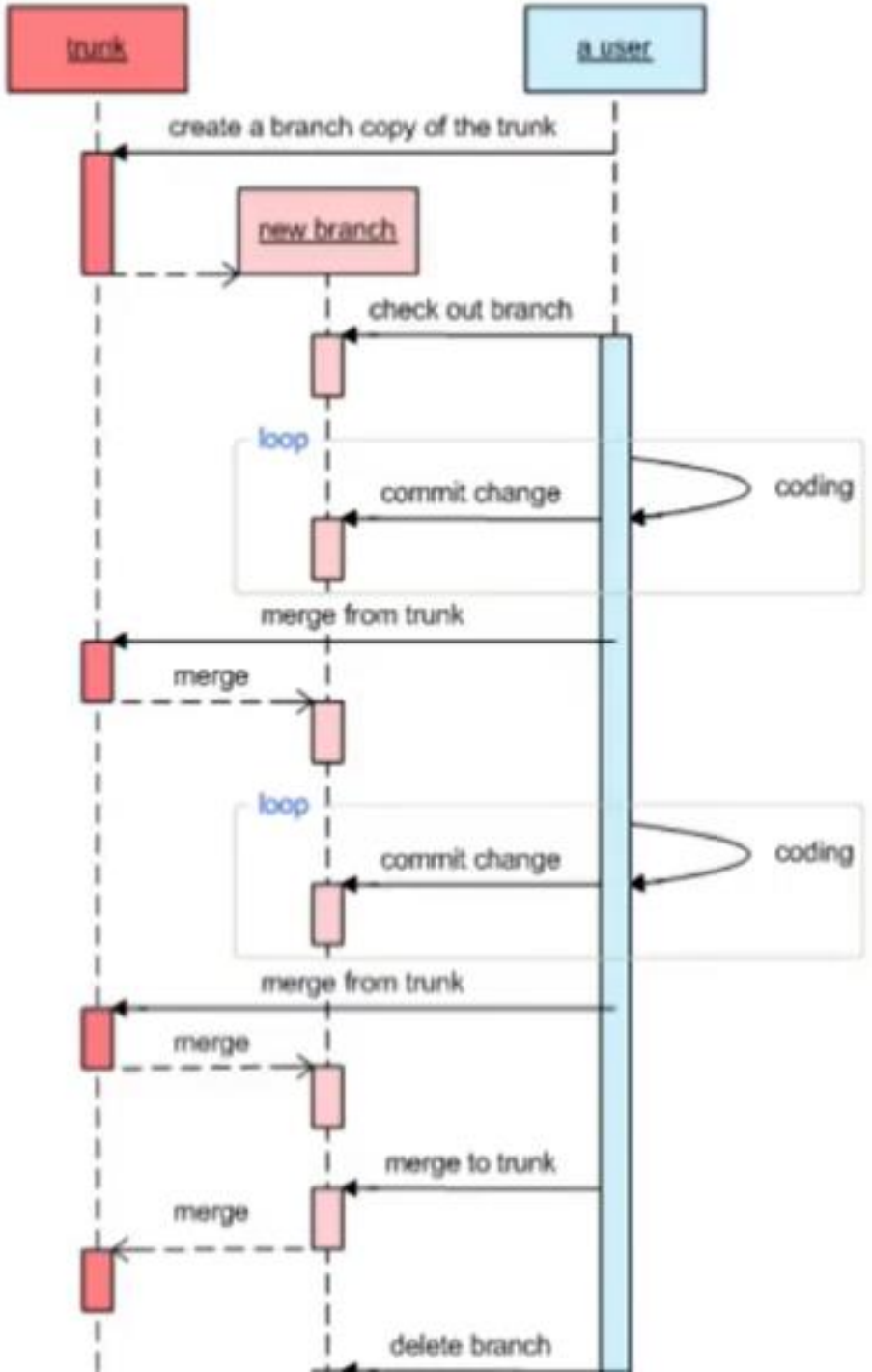
- Unificação de diferentes versões de um mesmo item de configuração
- Integração dos itens de configuração de um Branch com os itens de configuração do fluxo principal
- Algumas ferramentas fornecem um mecanismo automático para realização de merges

Merge


Merge
1.2.1.1
1.2.1.1
1.1
1.2
1.3
1.4
release 1
release 2
Trunk – tronco principal de desenvolvimento

Merge

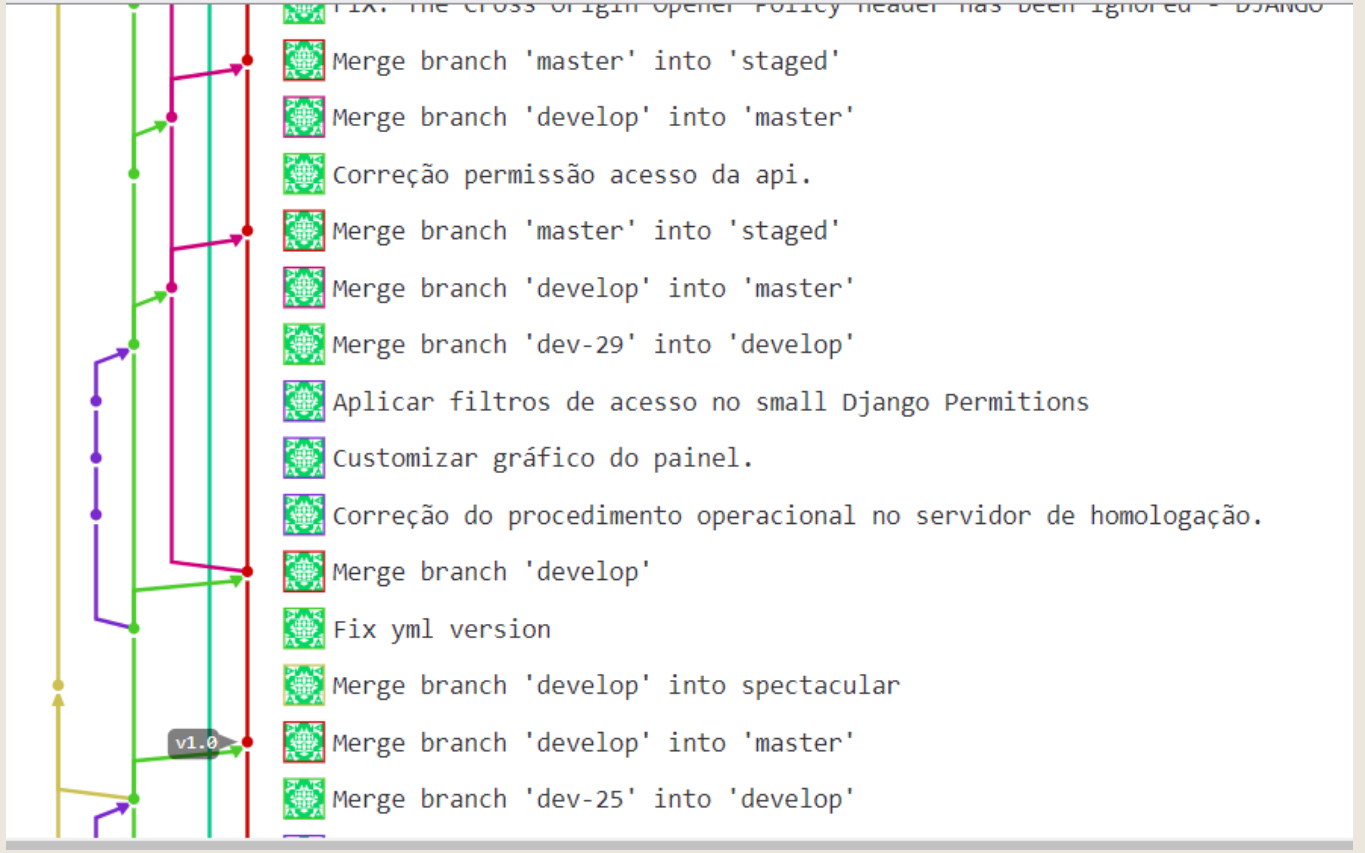
- Boa Prática
  - *Fazer merge do trunk com o Branch periodicamente*
  - *Isto torna o processo de merge mais gerenciável*



Merge

- Boa Prática
  - *Fazer merge do trunk com o Branch periodicamente*
  - *Isto torna o processo de merge mais gerenciável*

Merge


Merge branch 'master' into 'staged'
Merge branch 'develop' into 'master'
Correção permissão acesso da api.
Merge branch 'master' into 'staged'
Merge branch 'develop' into 'master'
Merge branch 'dev-29' into 'develop'
Aplicar filtros de acesso no small Django Permitions
Customizar gráfico do painel.
Correção do procedimento operacional no servidor de homologação.
Merge branch 'develop'
Fix yml version
Merge branch 'develop' into spectacular
v1.0
Merge branch 'develop' into 'master'
Merge branch 'dev-25' into 'develop'

Build

- Representa uma versão ainda incompleta do sistema em desenvolvimento, mas com certa estabilidade
- Costuma apresentar limitações conhecidas
- Espaço para integração de funcionalidades
- Inclui não só código fonte, mas documentação, arquivos de configuração, case de dados, etc.
- A política de geração dos build deve ser vem definida na estrutura do ambiente

Release

- Identificação e empacotamento de artefatos entregues ao cliente (interno ou externo) ou ao mercado
- Uma release implica no estabelecimento de um novo baseline, de produto
- Produto de software supostamente sem erros
- Processo interativo/incremental produz, em geral, mais de um release

ATIVIDADES DO GC

Atividades do GC

- Planejamento
  - *Definir ferramentas, equipamentos, estruturas de diretórios, repositórios e áreas de backup*
  - *Definir política para utilização do ambiente*
  - *Definir os papéis e responsabilidades*

Atividades do GC

- Planejamento
  - *Definir os baselines que serão estabelecidos*
  - *Definir o cronograma das atividades*
  - *Definir as políticas, procedimentos e padrões que guiarão as atividades*

Atividades do GC

- Identificação de Configuração
  - *Determinar os itens de configuração do software em desenvolvimento*
- Controle de Configuração
  - *Controlar as mudanças no software (contínuo)*
- Relatório de Status (Status accounting)
  - *Saber o status de todos os componentes em desenvolvimento, e saber quais são afetados por quais mudanças*

Atividades do GC

- Auditoria
  - *Verificar se o que foi desenvolvido atende aos requisitos de uma determinada baseline*
  - *Verificar se a rastreabilidade entre os itens de configuração está sendo mantida*
  - *Assegurar que a integridade do acesso do projeto está sendo preservada*

Mitos

- GC é papel exclusivo do Engenheiro de Configuração
- GC só precisa ser adotado por desenvolvedores
- GC serve apenas para a obtenção de certificações
- GC é perda de tempo, codificação é mais importante
- Somente código precisar ser versionado
- Uso o CVS/SVN. E assim faço GC corretamente, etc.

Ferramentas

- Controle de Versão
  - *Sistema para gerenciar as versões dos ICs (e diretórios) que permite:*
    - Controle do histórico (metadados e conteúdo)
    - Trabalho concorrente e isolado (branches)
    - Tagging e recuperação de versões
    - Relatório de Status (status accounting)

Ferramentas

- As mais utilizadas
  - *SVN – Subversion*
  - *CVS – Concurrent Version System*
  - *GitHub, BitBucket*
- Outras opções
  - *GitLab, OS: Mercurial, GIT, etc*
  - *Comerciais: ClearCase, Visual Source Safe*

Ferramentas

- Controle de Mudanças
  - *Registro e Gerenciamento de mudanças no software que permite:*
    - Acompanhamento dos estados de cada requisição
    - Controle de acesso
    - Histórico de Alteração
    - Automatizar o processo de submissão e gestão de Solicitações de Mudanças

Ferramentas

- Mais utilizadas
  - *Mantis (php)*
  - *Bugzilla (perl)*
- Outras opções
  - *Trac (python)*
  - *Atlassian Jira (j2ee)*

- Geração de Builds
  - *Automatiza o processo de geração de builds*
- Algumas opções
  - *Ant, Gnu Make, Maven, Shell/Batch Script ...*

Benefícios de GC

- Aumento de produtividade no desenvolvimento
- Redução nos custos de manutenção
- Redução de defeitos
- Maios rapidez na identificação e correção de problemas

Benefícios de GC

- Releases mais controlados
- Reuso de itens de software
  - *Artefatos, componentes*
- Automatização de processos
  - *Geração de releases, construção de builds, testes*

DUVIDAS?

Questionário

- Prazo 10/08/2023!!!
- https://forms.office.com/r/HvJBjhYV6s

PRÁTICAS

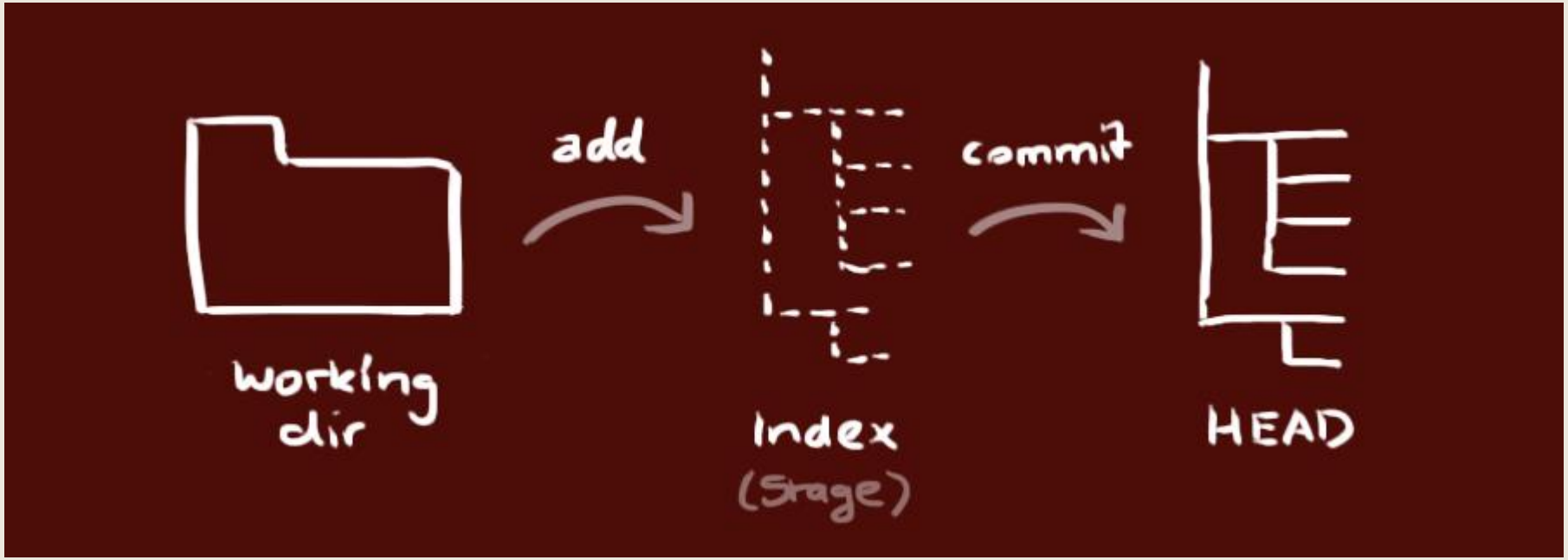
Atividade 1
Configuração de ambiente Git

Paços:

1. Cadastrar uma conta no GitHub
   - https://github.com/
2. *Efetuar FORK do repositório da aula*
   - https://github.com/oluap-honorio/aula_gc_unifametro_2023
3. Instalação do Git
   - https://git-scm.com/downloads
4. Criar um novo repositório local
   - **git init**
5. Criar um arquivo de texto
   - **codigo.txt**
6. Verificar ser status do repositório local
   - **git status**

Entendendo as arvores no Git

- *Working Directory*: contém os arquivos vigentes
- *Index*: que funciona como uma área temporária (*Stage*)
- *Head*: que aponta para o último commit (confirmação) no repositório local.



Atividade 2
Adicionar arquivos no Git

Paços:

1. Para adicionar o arquivo ao controle de versão
   - **git add “codigo.txt”**
2. Para confirmar vamos olhar novamente o status do repositório
   - **git status**
3. Gerar novos arquivos ao repositório
   - **Pegar em https://dontpad.com/aula-cm-2023**
4. Adicionar os arquivos em massa
   - **git add .**
5. E vamos olhar mais uma vez o status

Atividade 3
Criar versões do código - commit

Paços:

1. Criar a primeira versão do repositório local
   - **git commit –m “A3: Commit inicial”**
2. Configurar o autor dos commits
   - **git config –global user.email “mesmo email da repositório remoto”**
   - **git config –global user.name “Seu nome”**
3. Repetir o paço 1
   - **git commit –m “A3: Commit inicial”**
4. Caso já tenha um autor configurado na máquina diferente de você atualize os dados com o paço 2 e atualize o autor do último commit
   - **git commit –-amend**

Atividade 4
Enviar alterações para a o repositório remoto

Paços:

1. Criar um repositório remoto
   - **aula_gc_unifametro_2023**
2. Copiar o link e aplicar a referência no local
   - **git remote add origin <url>**
3. Enviar as alterações para o repositório remoto
   - **git push**
4. Vincular o branch remoto a ser atualizado
   - **git push –set-upstream origin master**
5. Atiualizar a página do repositório no GitHub
6. Solicitar o pull request da atividade 4

Atividade 5
Ciclo de atualização de código

Paços:

1. Altere o arquivo codigo.txt; Salve e verifique o repositório.
   - **git status**
2. Adicione as mudanças no Stage
   - **git add .**
3. novamente
   - **git status**
4. Commit descrevendo a alteração realizada no arquivo de texto
   - **git commit –m “A5: <descrição>”**
5. Enviar para repositório remoto
   - **git push**

Atividade 6
Verificar histórico de atualizações

Paços:

1. Listar o histórico de ações no repositório local
   - **git reflog**
2. Listar o histórico de commits no repositório local
   - **git log**
   - **git log --online**

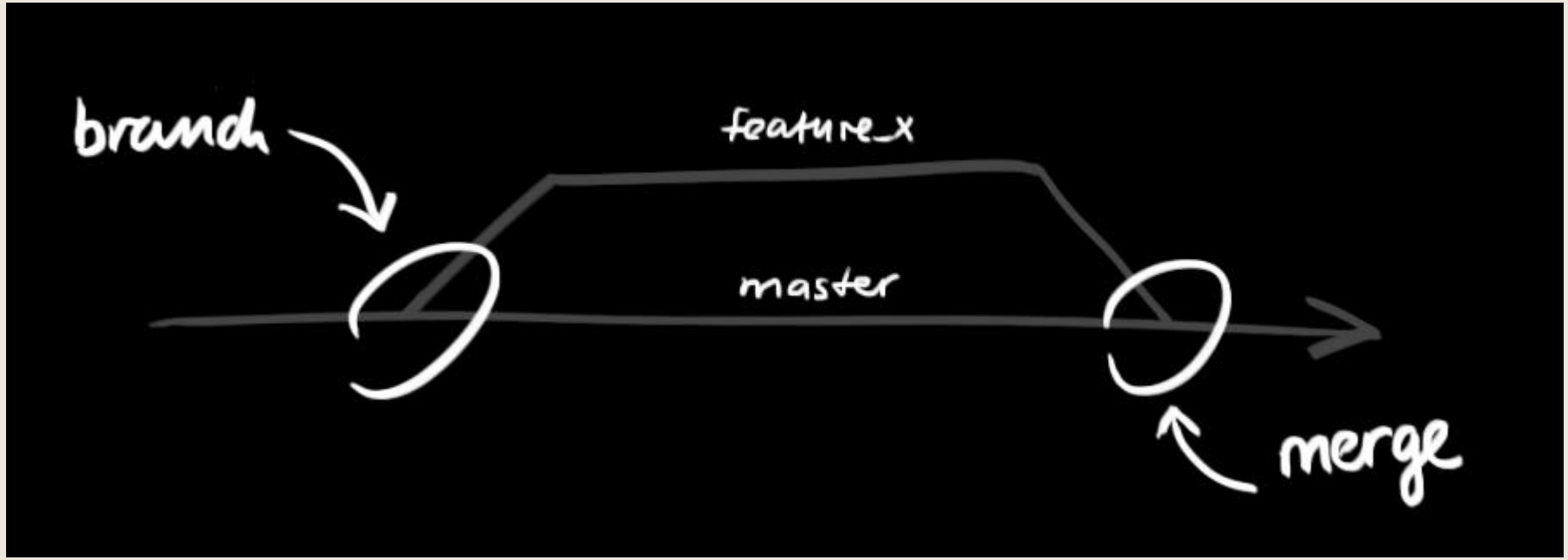
Atividade 7
Alterando versões do código

Paços:

1. Listar as versões do código
   - **git reflog**
2. Mover para uma versão específica
   - **git reset --hard <id>**
3. Uma outras forma de fazer a navegação
   - **git reset --hard HEAD-<quantidade de regressão>**
4. O que faz o seguinte comando
   - **git reset --soft HEAD-1**

Ramificando

- *Branches (“galhos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch master é o branch "padrão" quando você cria um repositório. Use outros branches para desenvolver e una-os (merge) ao branch master após a conclusão.*



Atividade 8
Construindo branch

Paços:

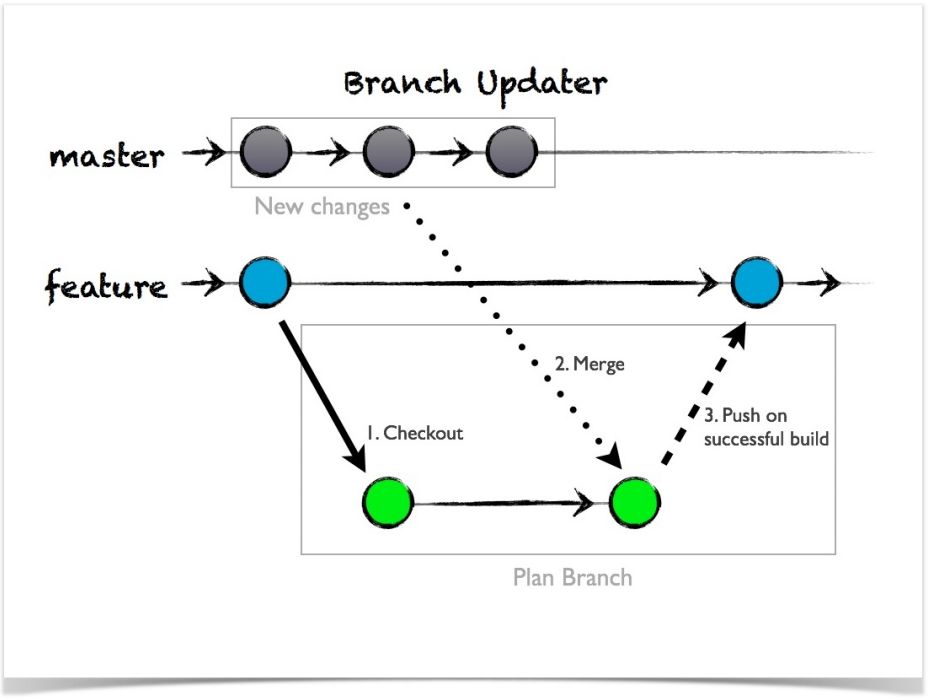
1. Listar os branch no repositório local
   - **git branch**
2. Criar um branch
   - **git branch dev**
3. Checar a lista de branch
   - **git branch**
4. Navegar entre branch
   - **git checkout dev**
5. Check novamente
   - **git branch**
6. Altere o arquivo de texto
7. Adicione as alterações no Stage
8. Efetue o commit com a descrição
9. Envie as alterações para o repositório remoto
   - **git push**
10. Indicar o branch de upstream
    - **git push –set-upstream origin dev**
11. Refaça o envio
    - **git push**
12. Verificar o repositório remoto no GitHub
13. Note que o código na master não fio alterado pois o commit foi no branch **dev**

Atividade 9
Unindo o código com merge

Paços:

1. Navegar para o branch que irá receber o código
   - **git checkout master**
2. [BOA PRÁTICA] Atualize os brunches compartilhados
   - **git pull**
3. Efetuar o merge com o branch alvo
   - **git merge dev**
4. Enviar as alterações
   - **git push**
5. Verificar o repositório remoto no GitHub
6. Solicitar o pull request da atividade 9

Ações realizadas no trabalho em equipe



1. git pull do branch principal
2. Gerar um novo branch a partir do principal
3. Trabalhar e adicionar as funcionalidades no novo branch
4. Finalizar o trabalho no branch temporário
5. git checkout no branch principal
6. git pull
7. merge o código do branch temporário
8. git push do branch principal para o repositório remoto

Atividade 10
Trabalhando com Pull Request

Paços:

1. Navegar para o branch principal
   - **git checkout master**
2. Gere um novo branch a partir do principal
   - **git checkout –b taskA10**
3. Edite o arquivo do código.
4. Efetue commit com a descrição
5. Envie para repositório remoto
   - **git push origin HEAD:taskA10**
6. Verifique a página de controle de pull request no GitHub
7. Solicitar o pull request para que as mudanças sejam revisadas

Atividade 11
gitignore

Paços:

1. Na pasta raiz do projeto adicione o arquivo de texto .gitignore
2. Liste neste arquivo de texto pastas e arquivo que não devem ser mapeados pelo git.
3. Valide se os arquivos listados são desconsiderados no status do repositório local
   - **git status**

Atividade 12
Tratamento de conflitos

Paços:

1. Navegar para o branch principal
   - **git checkout master**
2. Gere um novo branch a partir do principal
   - **git checkout –b taskA12**
3. Insira na primeira linha do arquivo do código:
   - { background: #fff }
4. Adicione a modificação e Efetue commit com a descrição
5. Retorne ao branch principal
6. Edite novamente o arquivo na primeira linha:
   - { background: #000 }
7. Adicione a modificação e Efetue commit com a descrição
8. Atualize o branch principal
   - **git pull**
9. Una as mudanças do takA10 com a master
   - **git merge taskA10**
10. Tratar o conflito editando novamente o arquivo de código
11. Adicionar as correções no Stage e efetuar commit sem uma descrição:
    - **git commit**
12. Verificar o histórico no repositório local
    - **git log –online**
13. Atualizar o reportório remoto
14. E solicitar o pull request para que as mudanças sejam revisadas

Guias

- Livro da comunidade Git
- Pro Git
- Pense como um Git
- Ajuda do GitHub
- Um guia visual do Git

Extra - comandos uteis

1. git cherry-pick
2. git rebase
3. git revert
4. git clean
5. git stash
6. gitk

DUVIDAS?

Referências

- MAXIM, B. R.; PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional. 2021.
- Descrição do workflow de gerência de configuração e mudanças do RUP
- DE ARAUJO, Alessandro Cruvinel Machado; DE VASCONCELOS, Alexandre Marcos Lins. Adaptando o RUP para o Desenvolvimento de Sistemas de Informação Web.
- CRNKOVIC, Ivica; ASKLUND, Ulf; DAHLQVIST, Annita Persson. Implementing and integrating product data management and software configuration management. Artech House, 2003.
- BAYS, Michael E. Software release methodology. Prentice-Hall, Inc., 1999.
- CHACON, Scott; STRAUB, Ben. Pro git. Springer Nature, 2014.
- Material de aula https://www.youtube.com/@CrescencioLima

BONS ESTUDOS!

MATERIAL ADAPTADO DO CANAL HTTPS://WWW.YOUTUBE.COM/@CRESCENCIOLIMA