

Aprendendo



git

Bismarck Gomes Souza Júnior

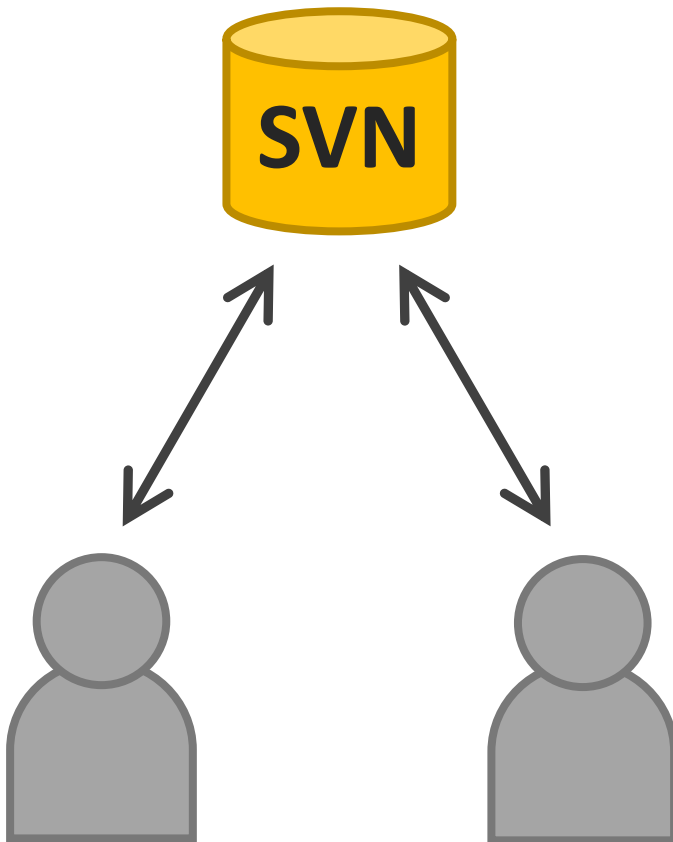
Março de 2014



Sistema de Controle de Versão

- Controle de histórico
- Trabalho em equipe
- Marcação e resgate de versões estáveis
- Ramificação do Projeto

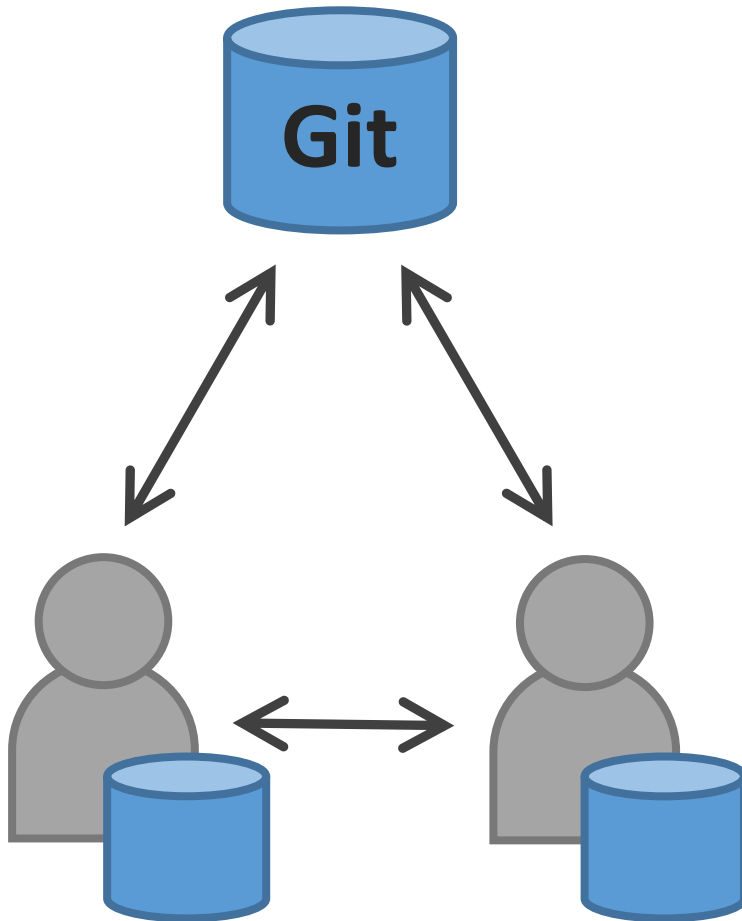
Sistema de Controle de Versão Centralizado



- Pouca autonomia
Ações necessitam de acesso ao servidor.
- Trabalho privado limitado
Versiona apenas arquivos no repositório.
- Risco de perda de dados
Tudo em um único repositório.

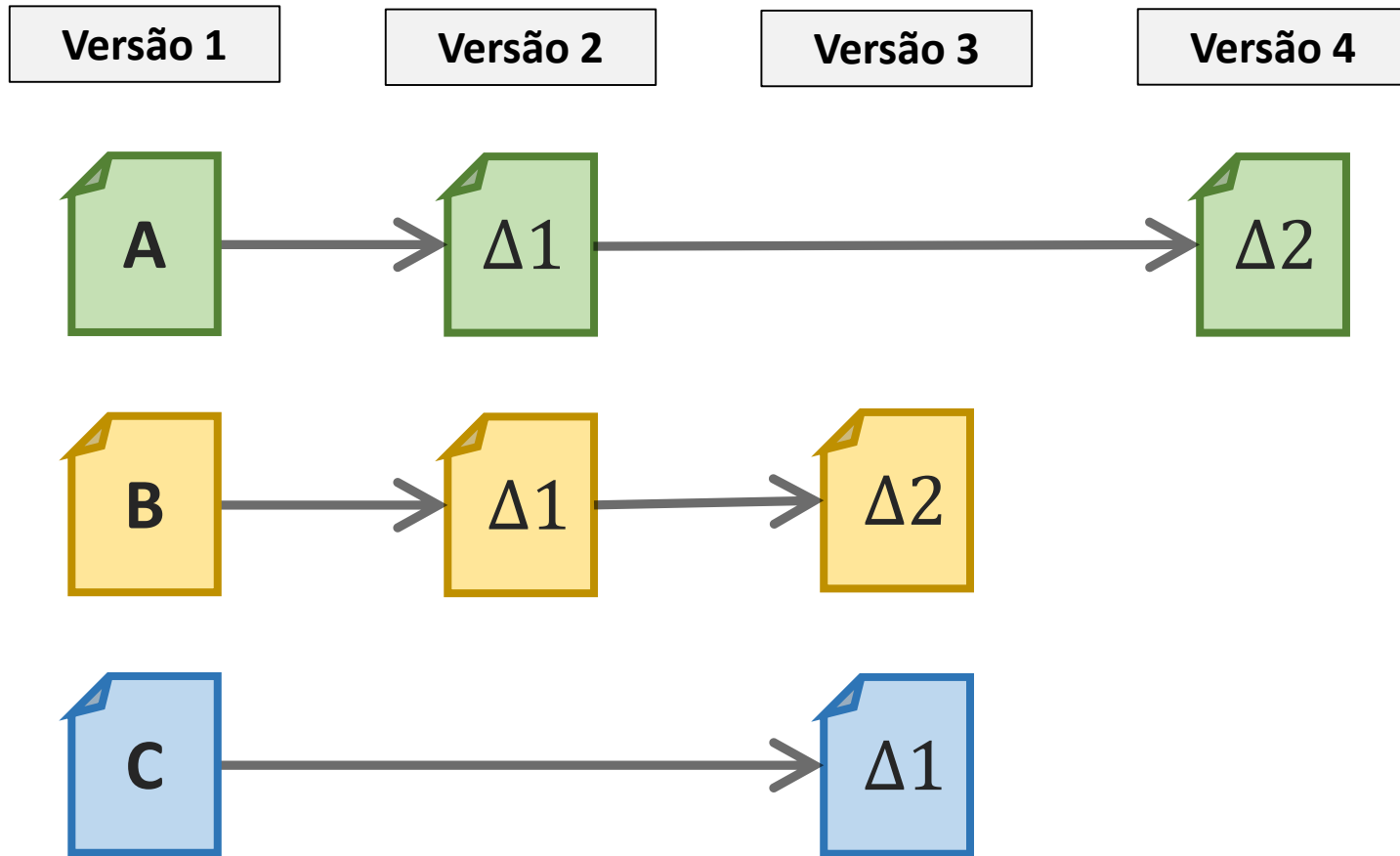


Sistema de Controle de Versão Distribuído

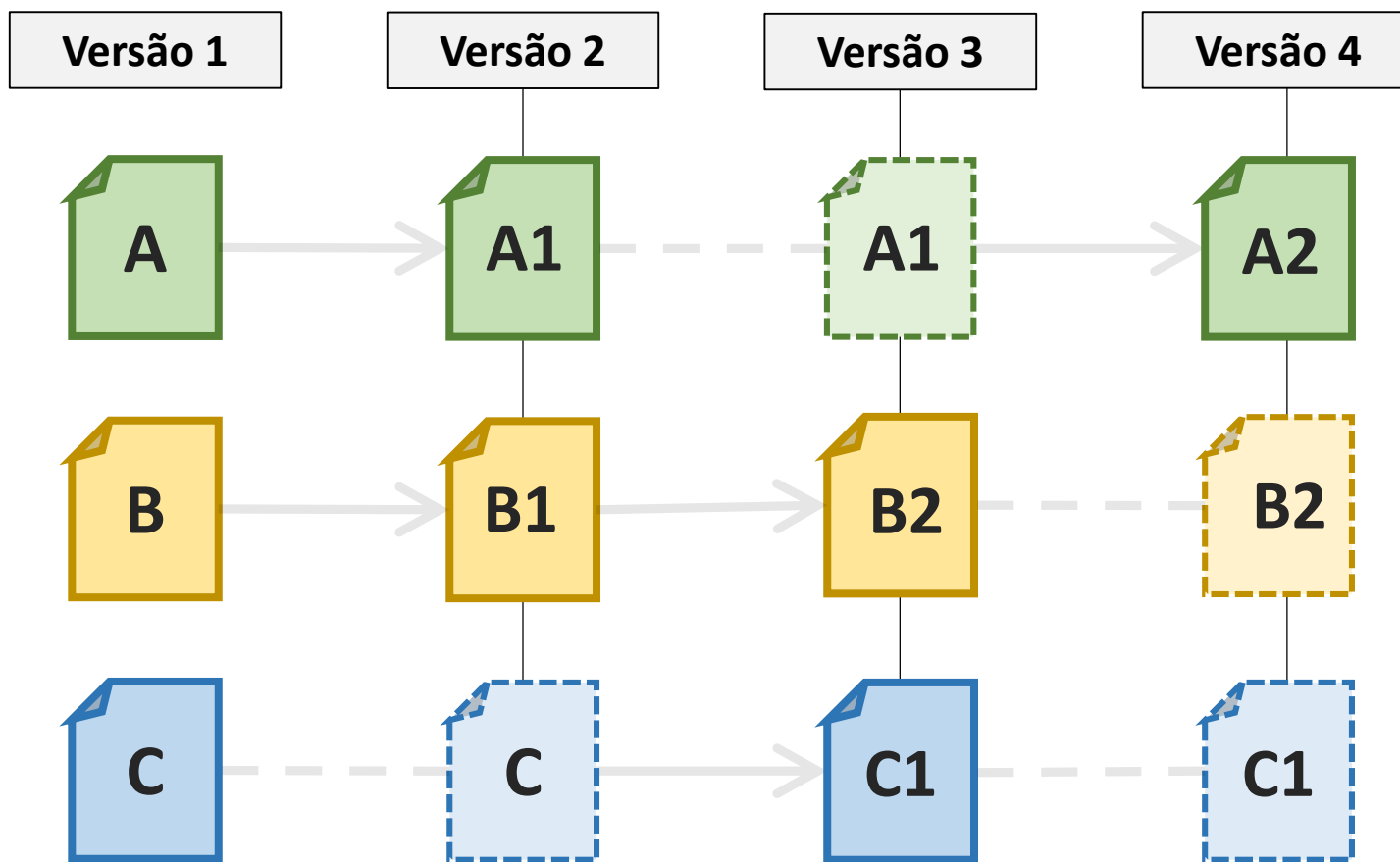


- **Autonomia**
Ações básicas “*off-line*”.
- **Rapidez**
Processos são locais.
- **Trabalho privado**
Trabalho local não afeta os demais.
- **Confiabilidade**
Todo repositório é um *backup*, ou seja, uma cópia completa do repositório, incluindo versões anteriores e histórico.

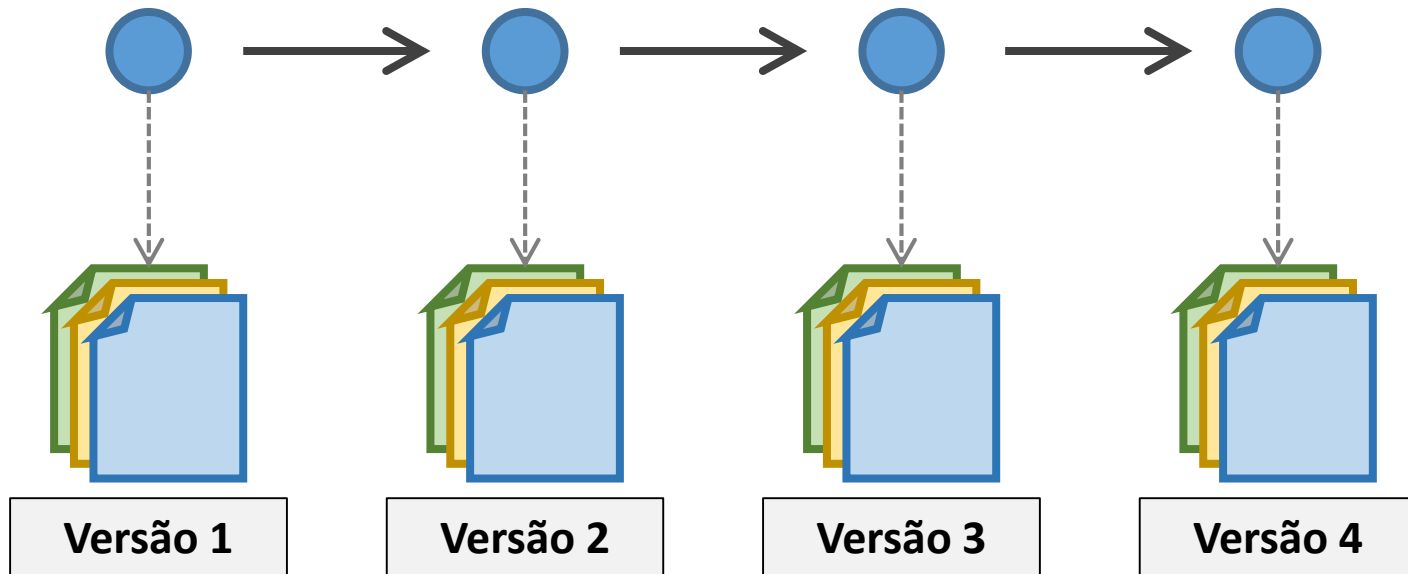
Sistema de Controle Convencional



Sistema de Controle do Git



Snapshots



Cada versão é uma “foto” do diretório de trabalho e será representado por um círculo azul denominado *commit*.



Sistema de Controle de Versão



Início



commit



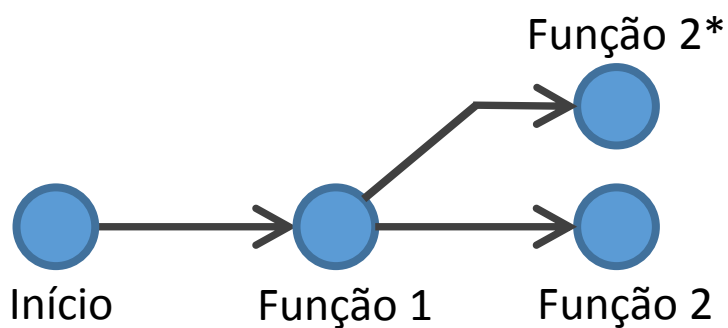
Sistema de Controle de Versão



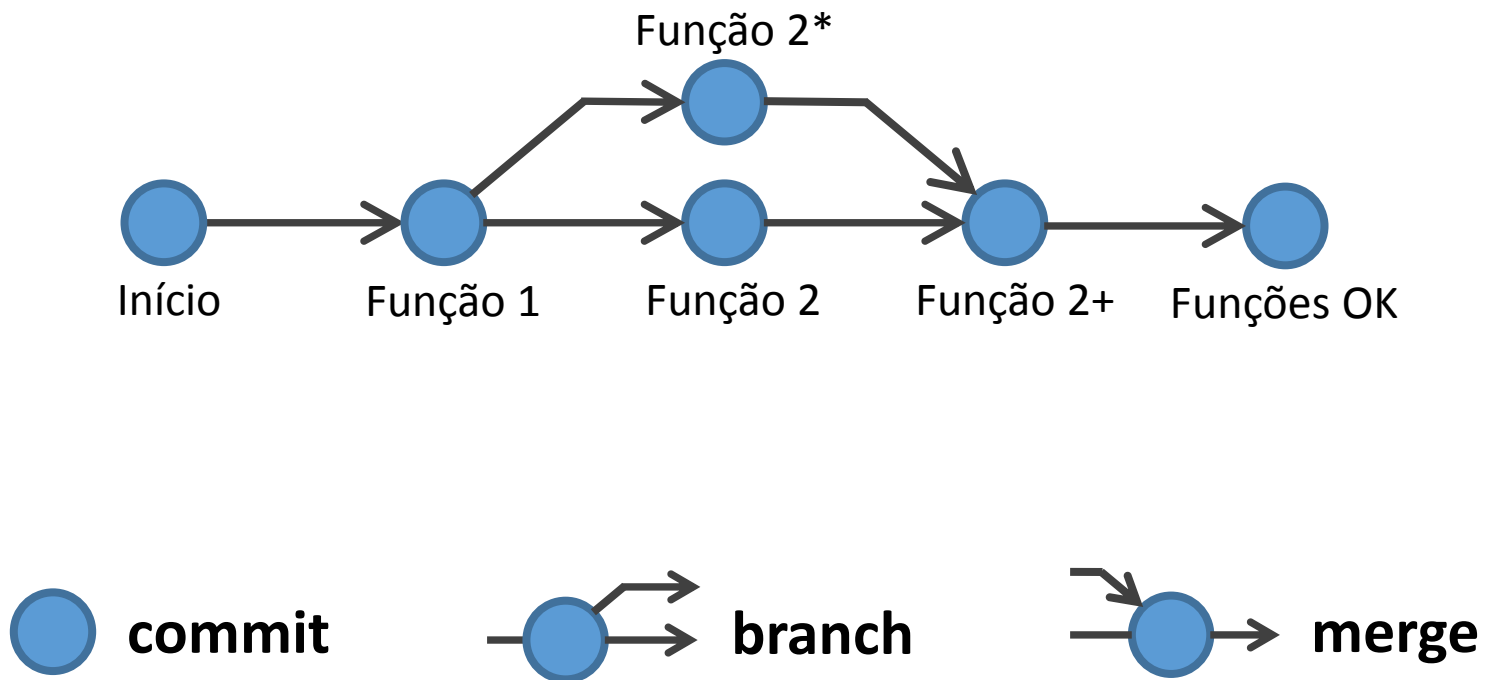
 **commit**

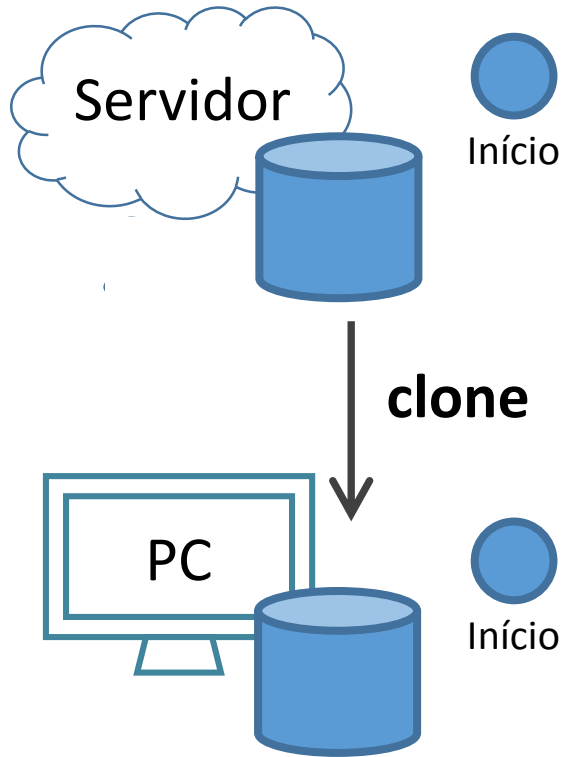


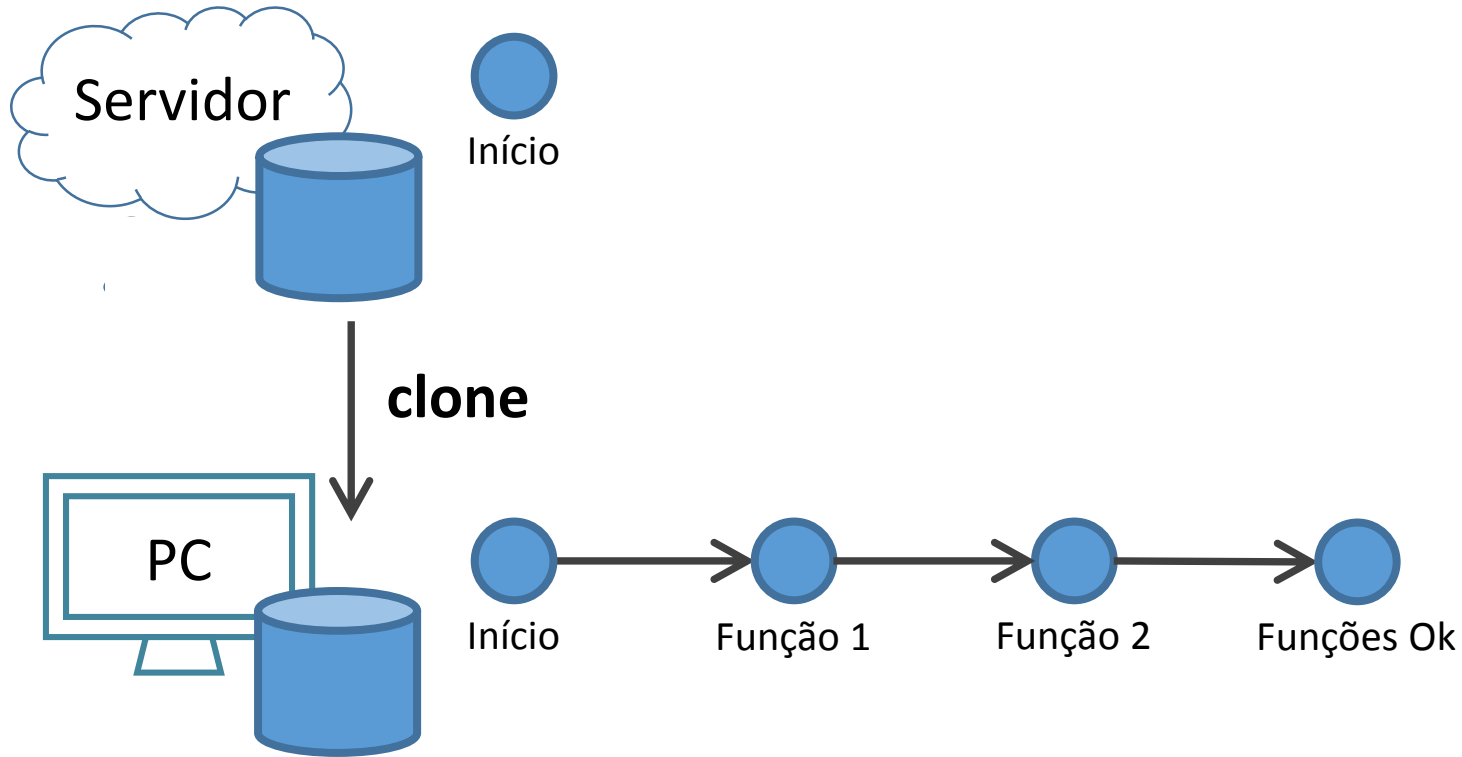
Sistema de Controle de Versão

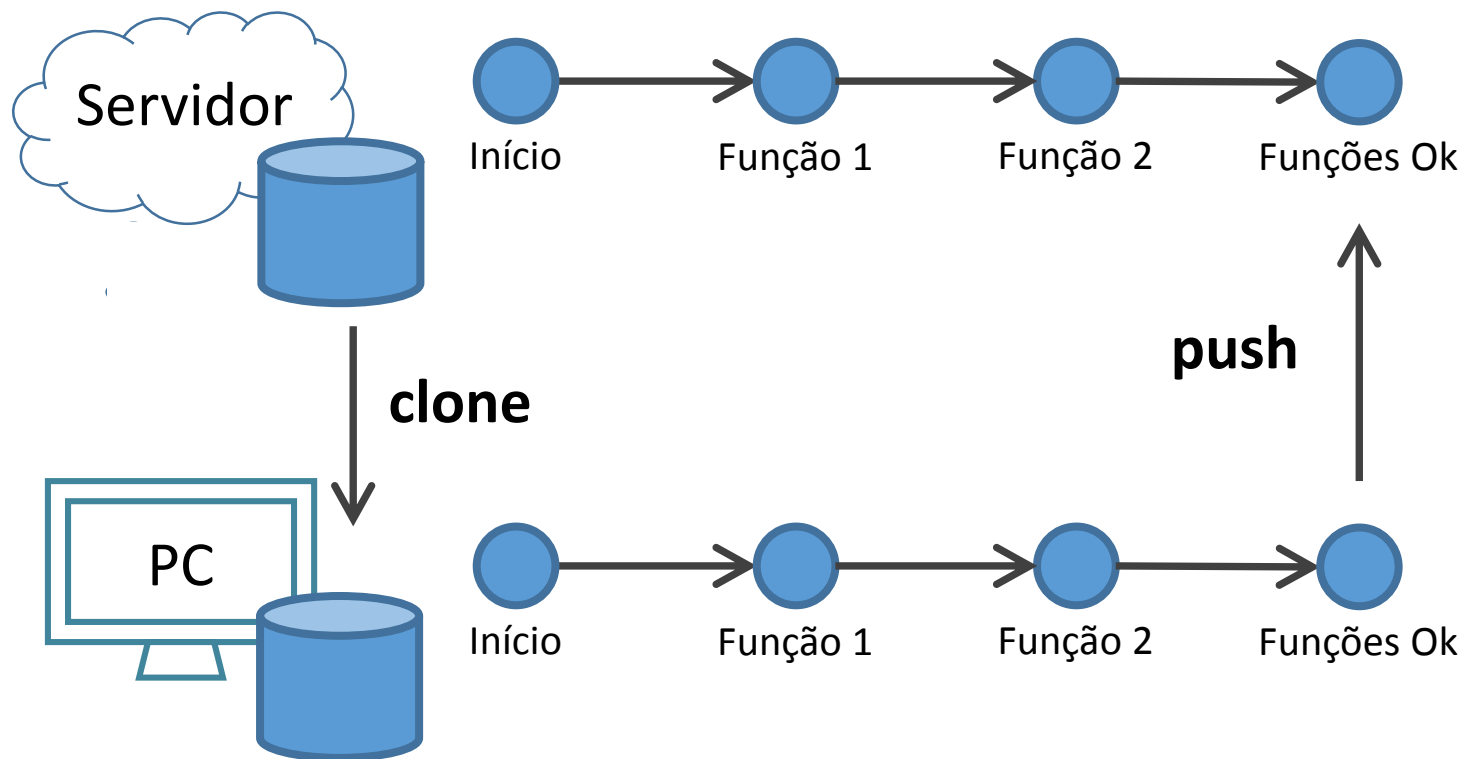


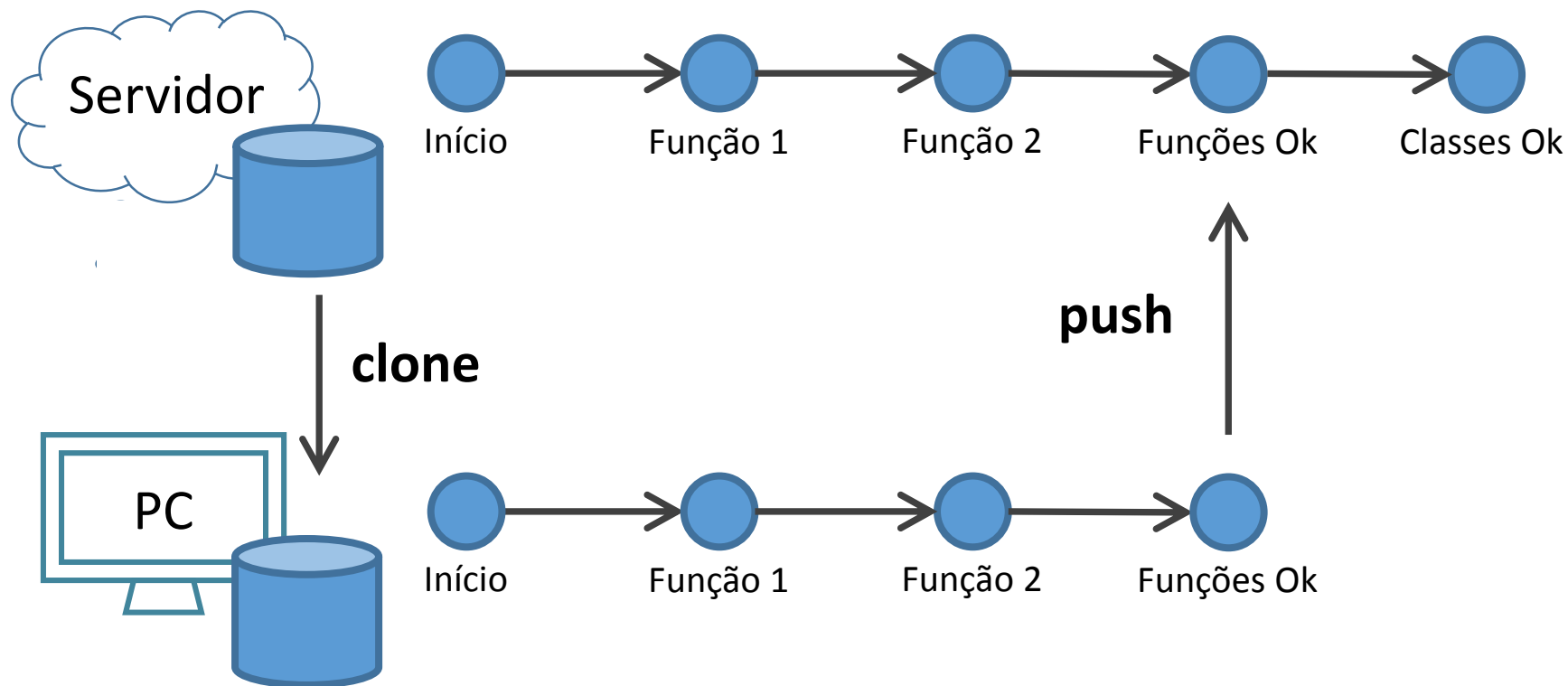
Sistema de Controle de Versão

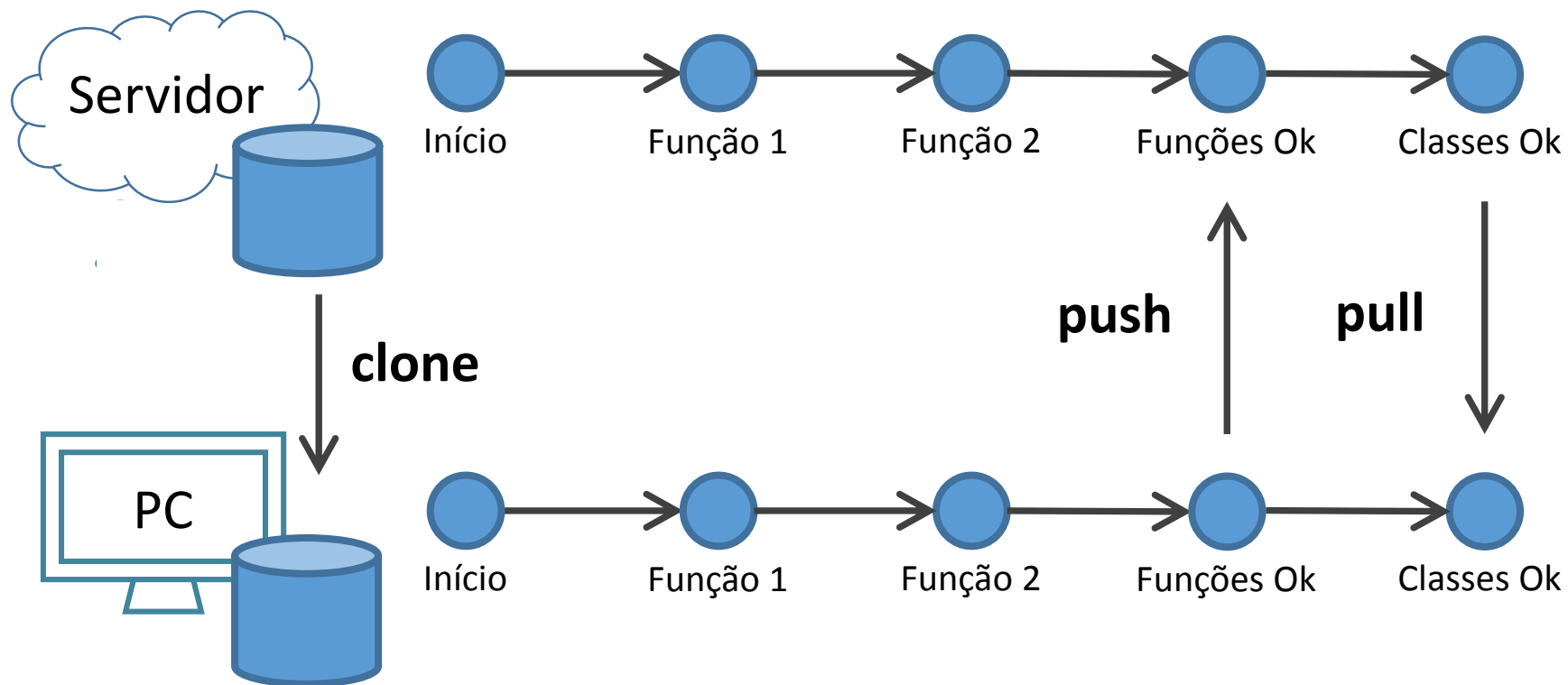














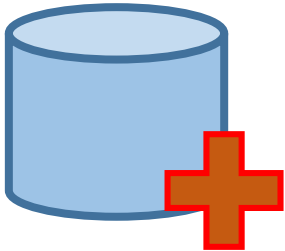
Servidores Para Hospedagem



Comandos Básicos



Criando um Repositório



```
$ git init
```

Transforma a diretório atual em um repositório git, criando o subdiretório “.git”.

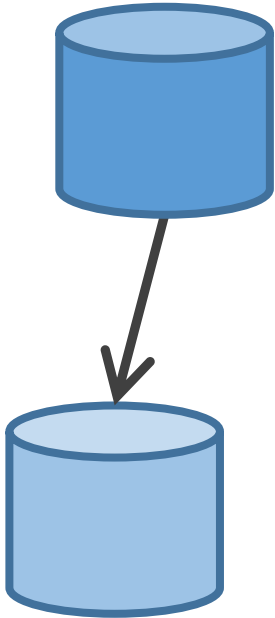


```
$ git init <dir>
```

Cria o diretório <dir> e transforma em um repositório git.



Clonando um Repositório



```
$ git clone <repo>
```

Clona o repositório <repo> para a máquina local.

```
$ git clone <repo> <dir>
```

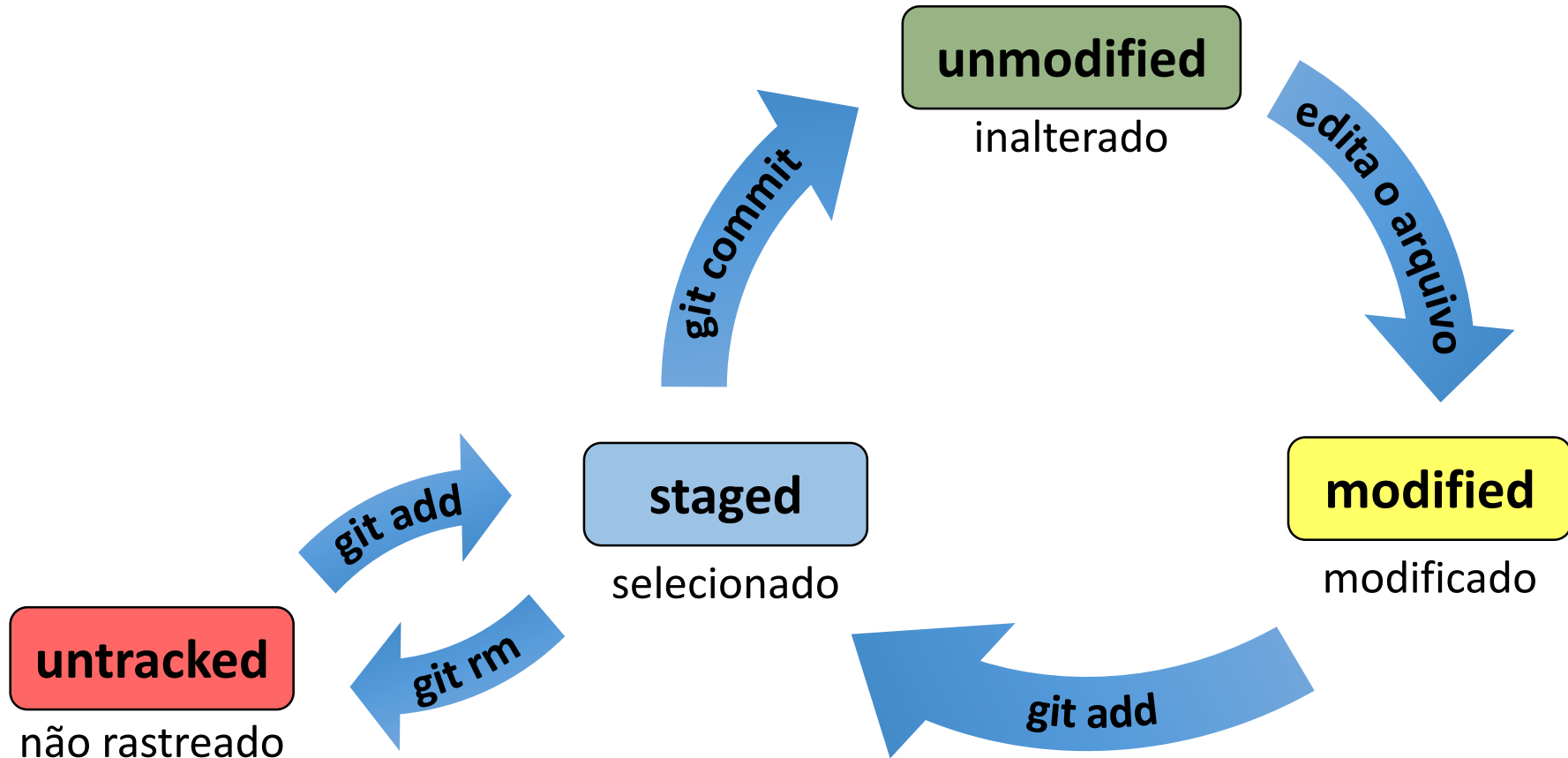
Clona o repositório <repo> para o diretório <dir>.

```
$ git clone git@github.com:user/Project.git
```

```
$ git clone https://github.com/user/Project.git
```



Tipos de Estado de um Arquivo





.gitignore

Arquivo que contém os arquivos que não serão visíveis pelo git.

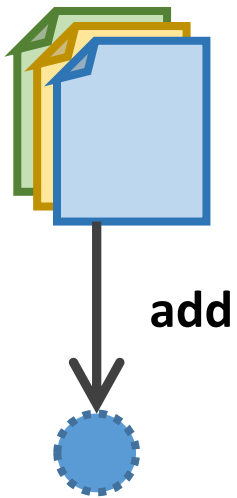
Arquivo .gitignore (exemplo)

```
Thumbs.db      #Arquivo específico
*.html         #Arquivos que terminam com ".html"
!index.html    #Exceção, esse arquivo será visível ao git
log/           #Diretório específico
**/tmp         #Qualquer diretório nomeado de "tmp"
```

- Arquivos que já estavam sendo rastreados não são afetados.



Preparando Para Salvar Alterações



*Stage Area
(Index)*

```
$ git add <arquivo|dir>
```

Adiciona as mudanças do arquivo <arquivo> ou do diretório <dir> para o próximo *commit*. O arquivo passa a ser rastreado.

```
$ git reset <arquivo>
```

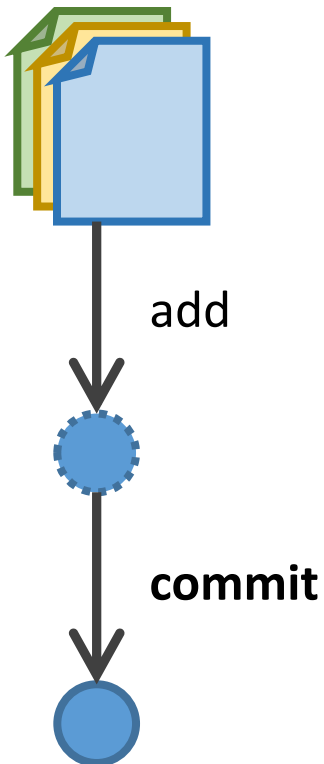
Remove as mudanças do arquivo <arquivo> para o próximo *commit*.

```
$ git rm --cached <arquivo>
```

Para de rastrear o arquivo <arquivo>.



Salvando Alterações



```
$ git commit
```

Realiza o *commit* e abre o editor para inserir uma mensagem.

```
$ git commit -a
```

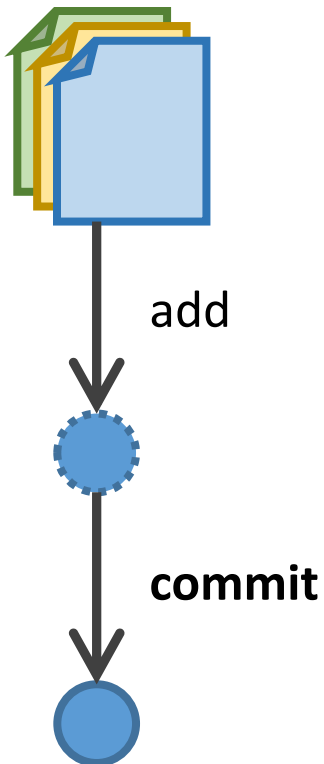
Adiciona as mudanças dos arquivos já rastreados e realiza o *commit*. O editor será aberto.

```
$ git commit -m "<msg>"
```

Realiza o *commit*, com a mensagem <msg>.



Salvando Alterações



```
$ git commit -am <msg>
```

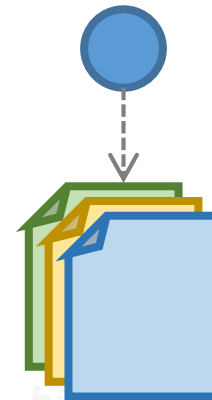
Adiciona as mudanças dos arquivos já rastreados e realiza o *commit* com a mensagem <msg>.

```
$ git commit --amend -m <msg>
```

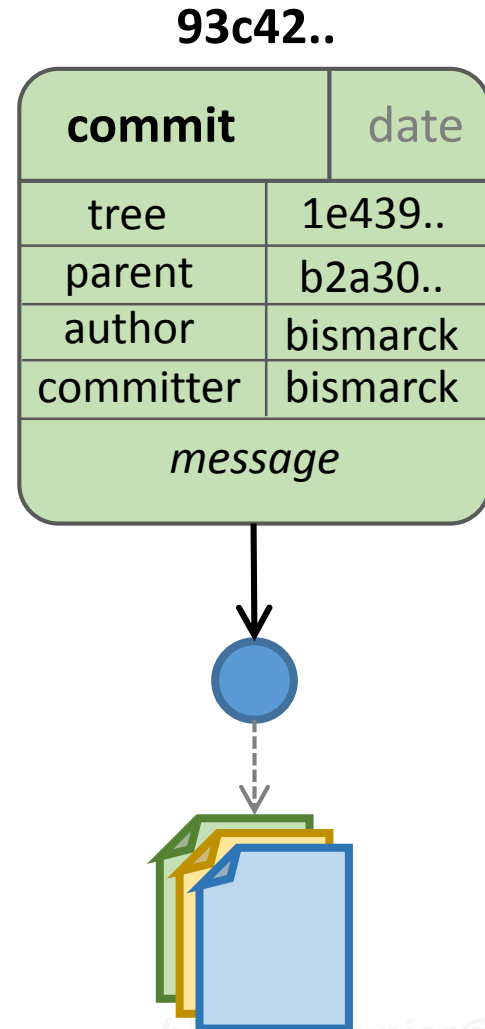
Substitui o último commit e altera a mensagem para <msg>.



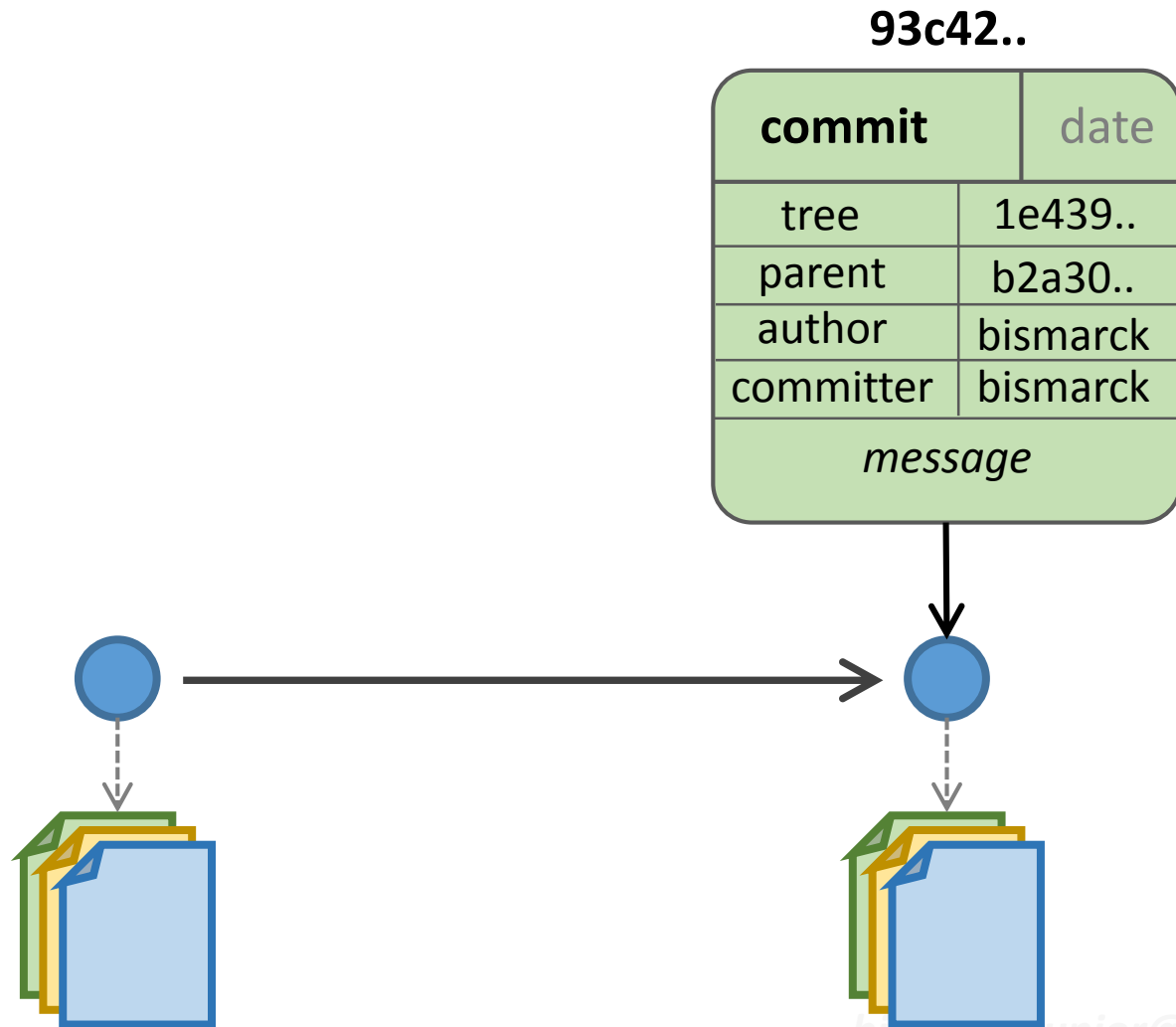
Commit



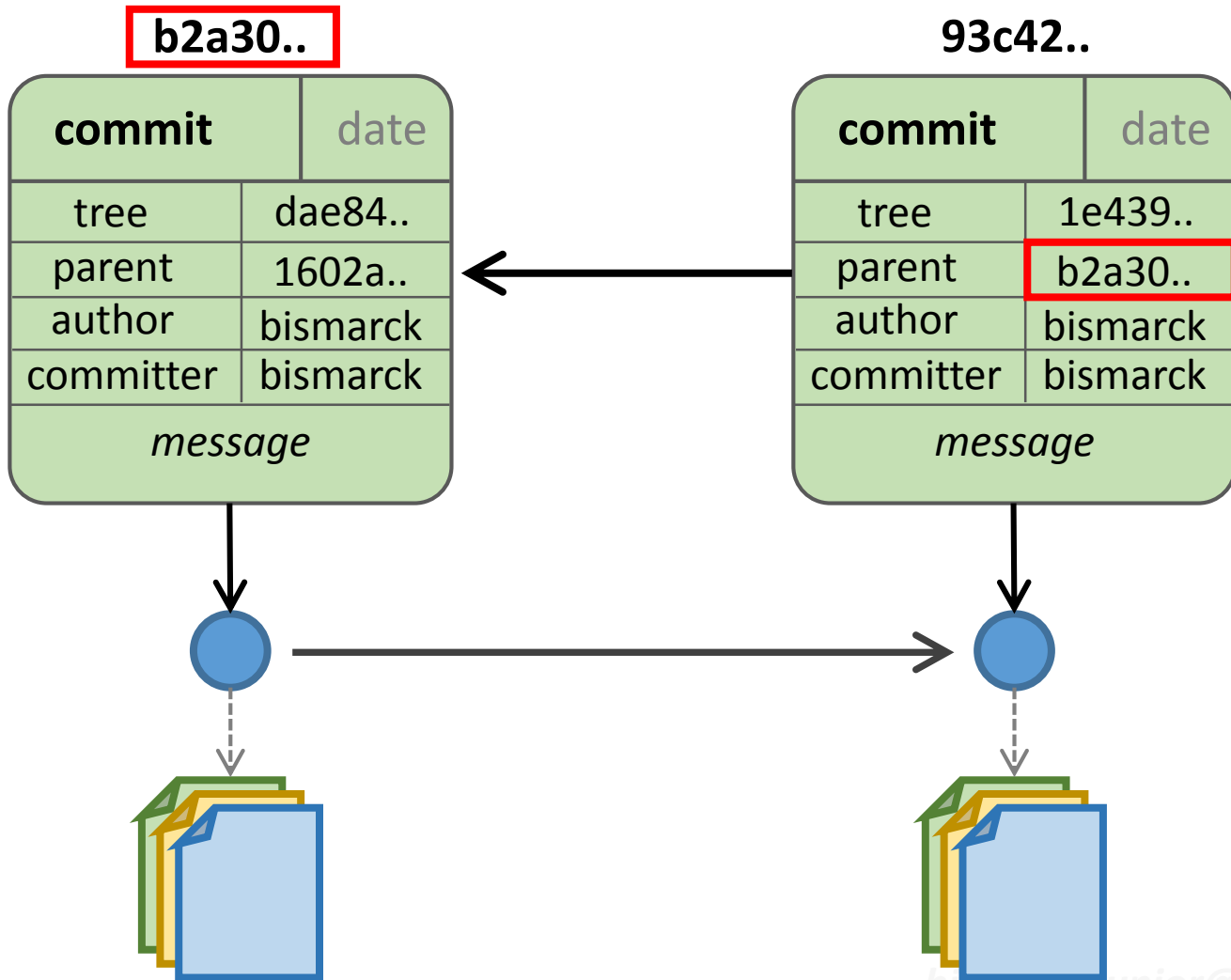
Commit



Commit

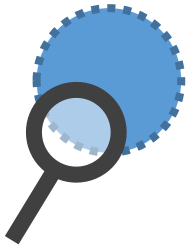


Commit





Analizando os Arquivos na Área Transitória



```
$ git status
```

Lista os arquivos que estão e que não estão na área transitória, e os arquivos que não estão sendo rastreados.

```
$ git status -s
```

Lista os arquivos de uma forma simplificada.

Tagging



```
$ git tag
```

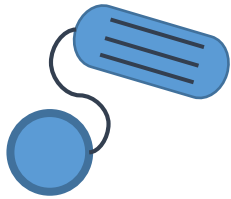
Lista as *tags* existentes.

```
$ git tag -l <tag>
```

Procura pela *tag* <tag>.

```
$ git tag -l 'v.0.*'
```

Tagging



```
$ git tag <tag> [<commit>]
```

Cria a *tag* <tag> para o último *commit* ou para o *commit* <commit>.

```
$ git tag -a <tag>
```

Cria a *tag* <tag> completa para o último *commit* e abre o editor para inserir uma mensagem.

```
$ git tag -a <tag> -m <msg>
```

Cria a *tag* <tag> completa para o último *commit* com a mensagem <msg>.



Versionamento



v.0.1.0

v[major] . [minor] . [patch]

[patch]: correção de *bugs*.

[minor]: incrementos de funcionalidades compatíveis com versões anteriores.

[major]: incrementos de funcionalidades incompatíveis com versões anteriores.

Versões teste: alpha (a), beta (b)

Ex: v0.1.9 < v0.1.10 < v0.2.0a < v0.2.0b < v0.2.0



Referência a *Commit*

<sha1>

Hash SHA-1 referente ao *commit*. Pode-se usar os primeiros caracteres.

Ex: b230 = b230e84a4c90d2f11ba85404e5fba93ce0a...

<tag>

Tag referente ao *commit*.

Ex: v0.1.2

<branch>

Último *commit* do *branch* <branch>.

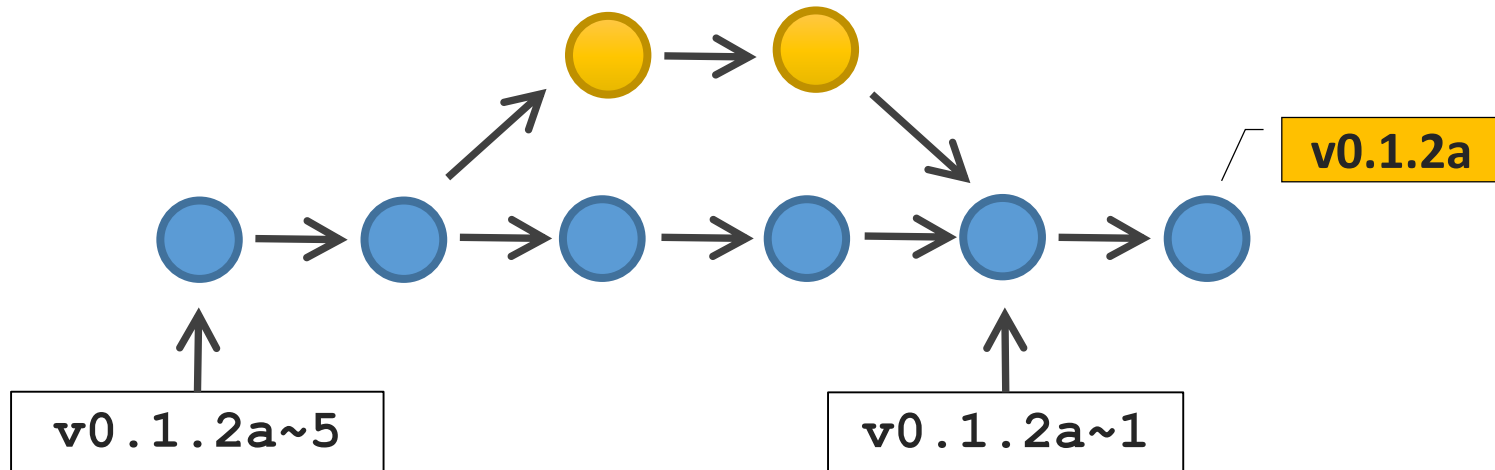
Ex: master



Referência a *Commit*

<commit>~<n>

O n-ésimo percussor do *commit* <commit>.

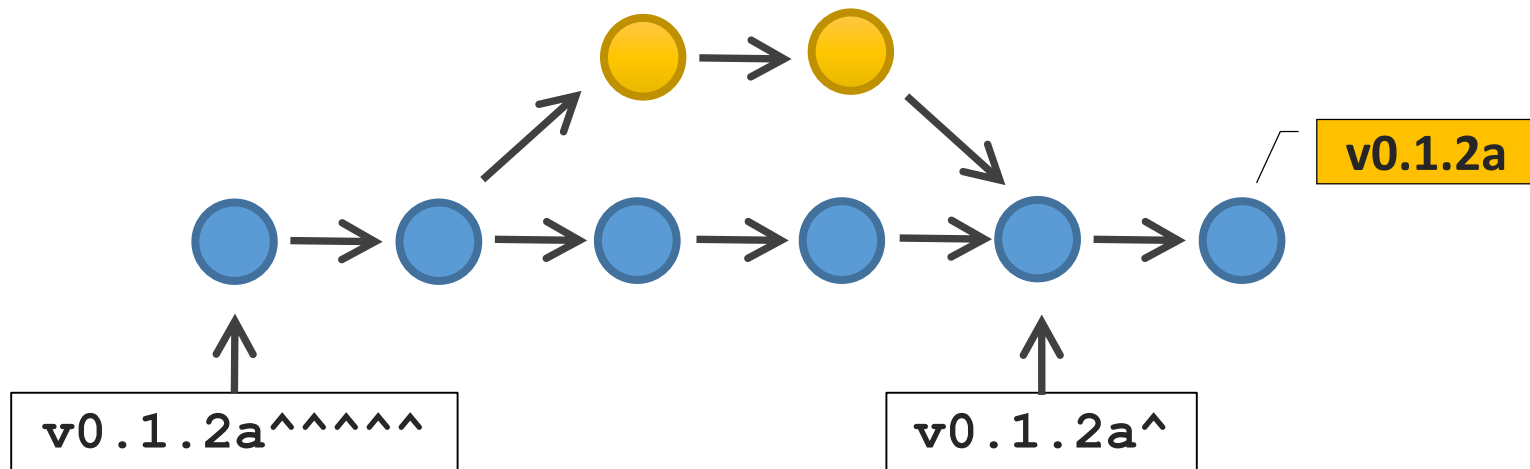




Referência a *Commit*

<commit>^1 ou <commit>^ ou <commit>~1

O primeiro percussor do *commit* <commit>.

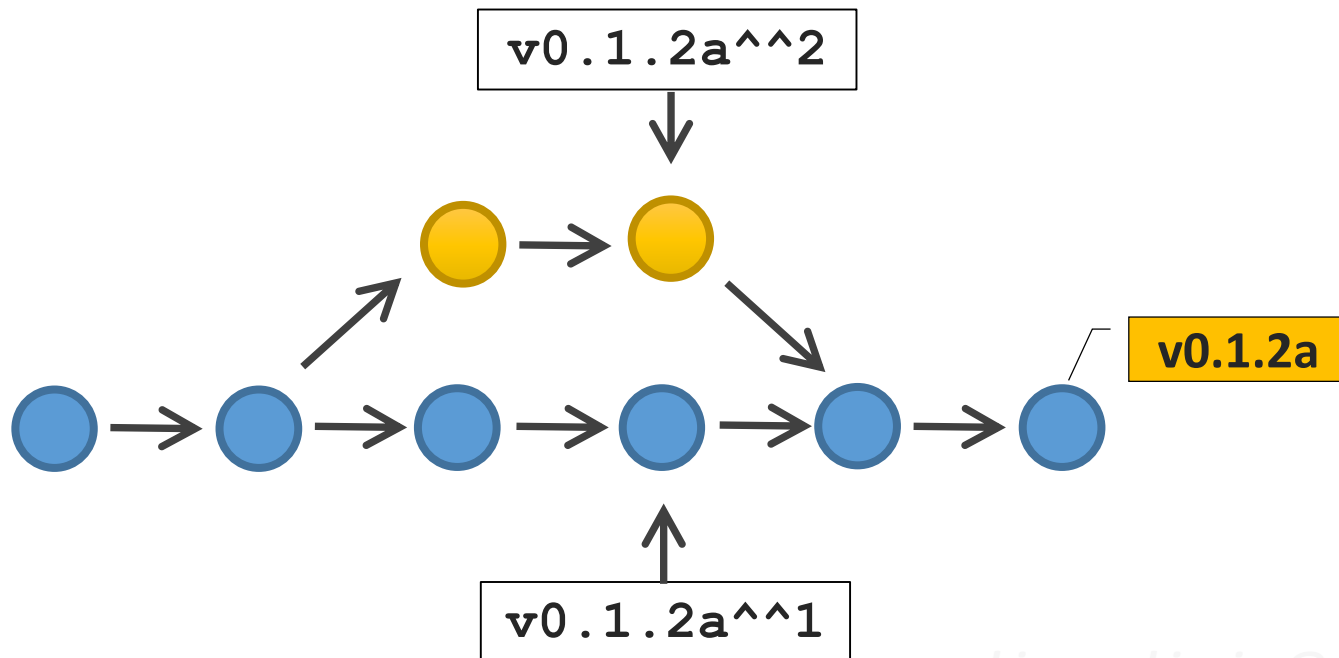




Referência a *Commit*

<commit>^2

O segundo percussor do *commit* <commit>. Utilizado em *commits* resultantes de um *merge*.



Analizando *Commits*



```
$ git show
```

Exibe o último *commit*.

```
$ git show <commit>
```

Exibe o *commit* referenciado por <commit>.

```
$ git show <commit>:<arquivo>
```

Exibe o arquivo <arquivo> no *commit* <commit>.

Analizando um Arquivo



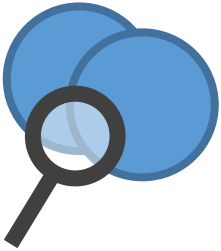
```
$ git blame <arquivo>
```

Exibe quem modificou cada linha do arquivo <arquivo>, incluindo data e *commit*.

```
$ git blame -L <n>,<m> <arquivo>
```

Exibe quem modificou as linhas de <n> a <m> do arquivo <arquivo>, incluindo data e *commit*.

Diferença Entre *Commits*

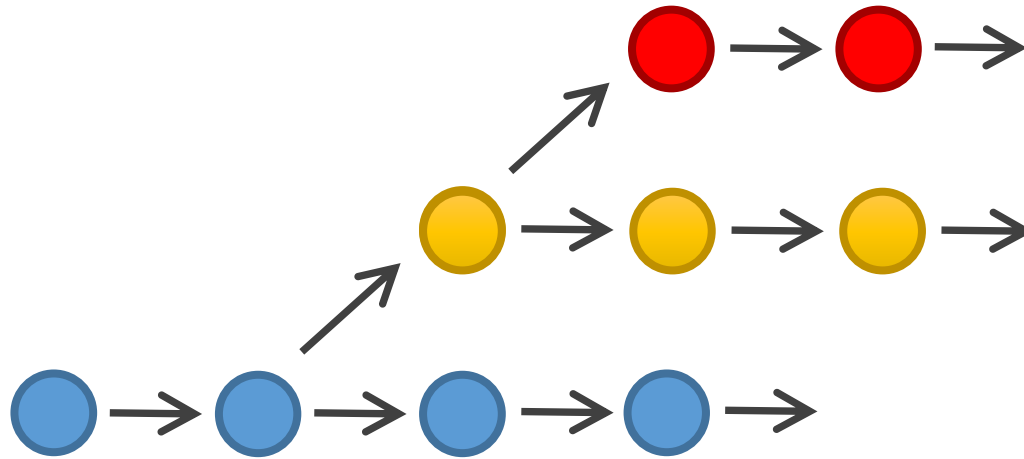


```
$ git diff <commit>
```

Exibe a diferença nos arquivos entre o *commit* <commit> e o diretório de trabalho.

```
$ git diff --cached <commit>
```

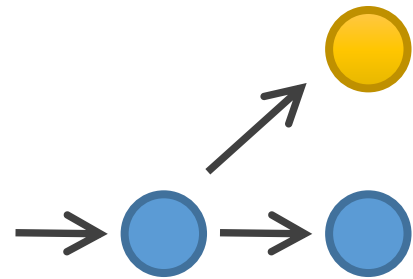
Exibe a diferença nos arquivos entre o *commit* <commit> e a área transitória.



Branches



Criando Ramificações



```
$ git branch [-a]
```

Exibe os *branches* existentes. Na forma completa, exibe também os *branches* remotos.

```
$ git branch <branch> [<base>]
```

Cria o *branch* <branch> a partir do *commit* <base>.

```
$ git checkout -b <branch>
```

Cria o *branch* <branch> e altera para ele.



Criando Ramificações

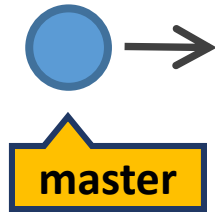


```
$ git add *
```

Adiciona os arquivos para o *index* (área transitória).



Criando Ramificações

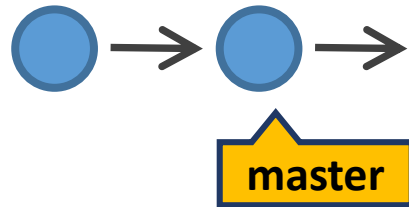


```
$ git commit
```

Realiza um *commit*.



Criando Ramificações

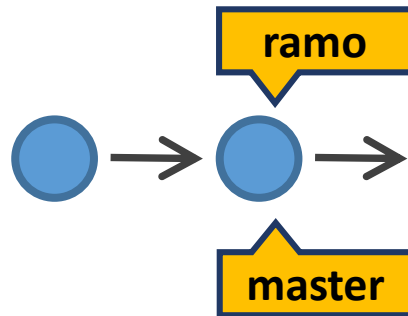


```
$ git commit -a
```

Adiciona os arquivos para o *index* e realiza um *commit*.



Criando Ramificações

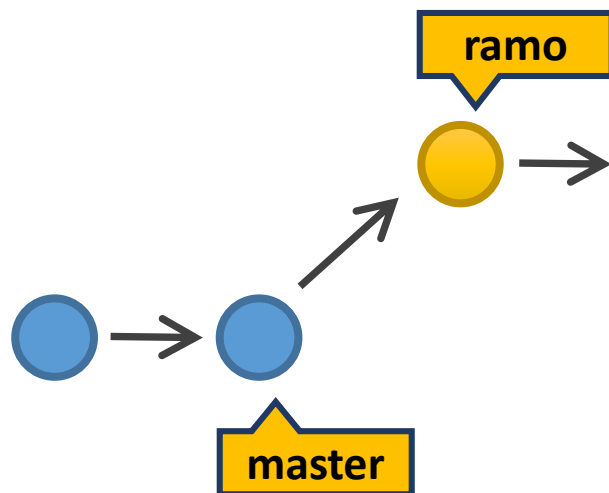


```
$ git checkout -b ramo
```

Cria o *branch* ramo e altera para ele, ou seja, os próximos *commits* serão no *branch* ramo.



Criando Ramificações



```
$ git commit -a
```

Realiza um *commit* no *branch* ramo.



Alternando em Ramificações

```
$ git checkout <branch>
```

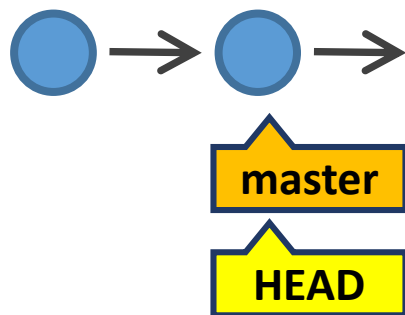
Altera para o *branch* <branch>.

```
$ git checkout -f <branch>
```

Altera para o *branch* <branch> “na força”, perdendo-se as informações não “commitadas”.



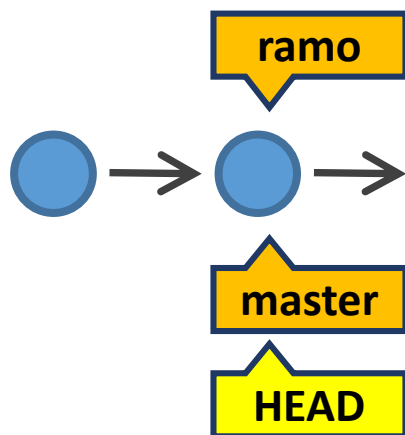
Alternando em Ramificações



HEAD: aponta para o *branch* atual.



Alternando em Ramificações

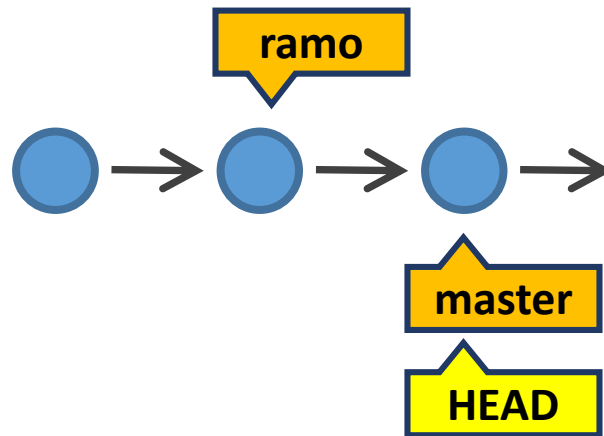


```
$ git branch ramo
```

Cria o *branch* ramo.



Alternando em Ramificações

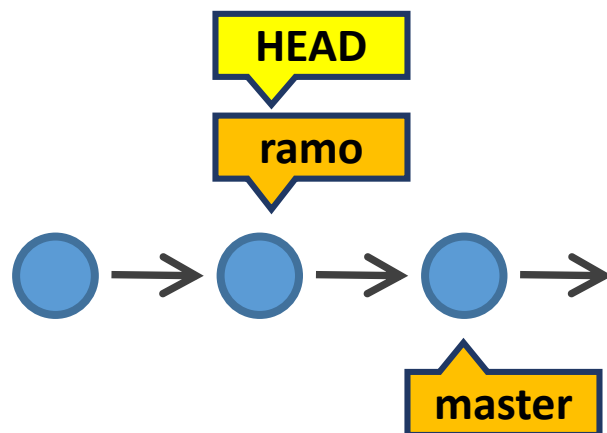


```
$ git commit -a
```

Realiza um *commit* no *branch* master.



Alternando em Ramificações

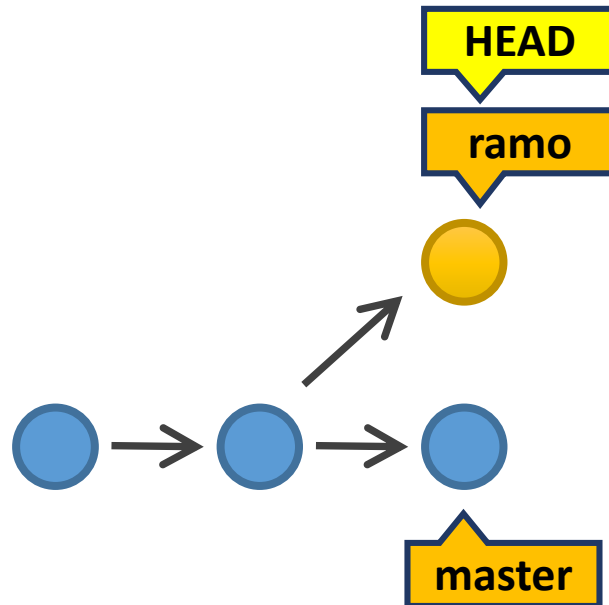


```
$ git checkout ramo
```

Alterna para o *branch* ramo.



Alternando em Ramificações

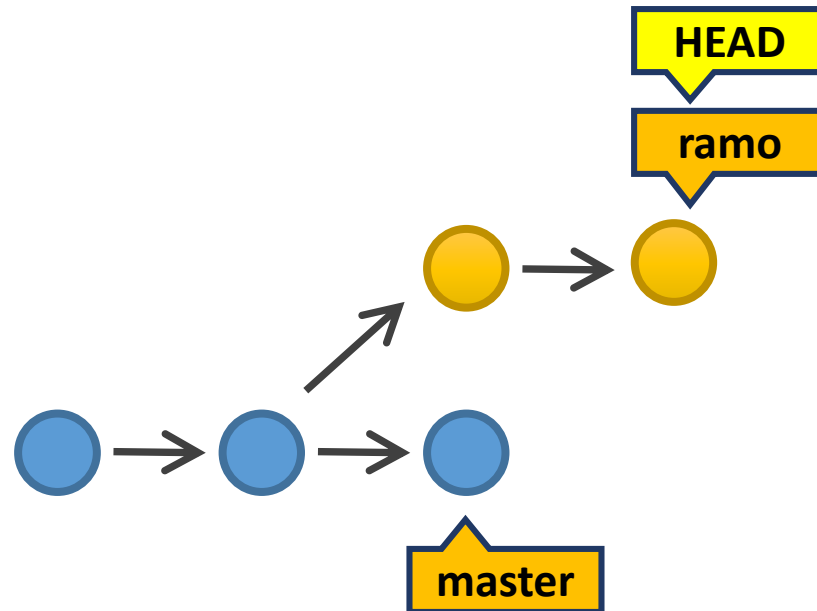


```
$ git commit -a
```

Realiza um *commit* no *branch* ramo.



Alternando em Ramificações



```
$ git commit -a
```

Realiza um *commit* no *branch* ramo.



Excluindo Ramificações



```
$ git branch -d <branch>
```

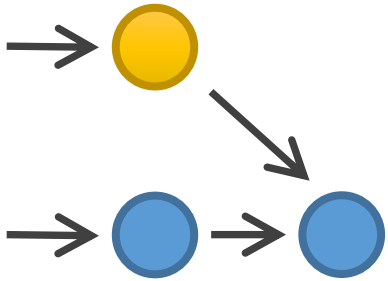
Exclui o *branch* <branch>. O *branch* já deve ter sido mesclado.

```
$ git branch -D <branch>
```

Exclui o *branch* <branch> mesmo não tendo sido mesclado.



Mesclando *Commits*



```
$ git merge <branch>
```

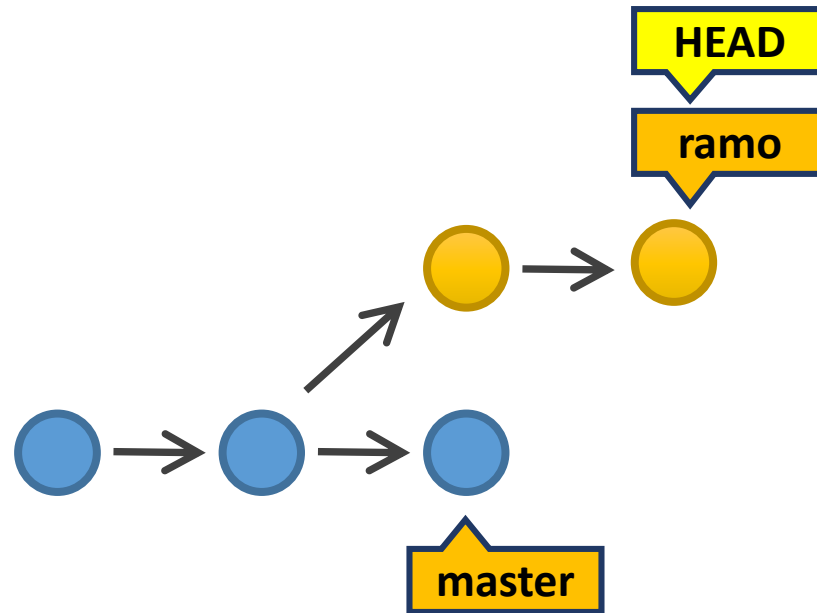
Mescla os *commits* do *branch* <branch> para o branch atual.

```
$ git merge <branch> --no-ff
```

Mescla os *commits* do *branch* <branch> para o branch atual sem *fast-forward*.

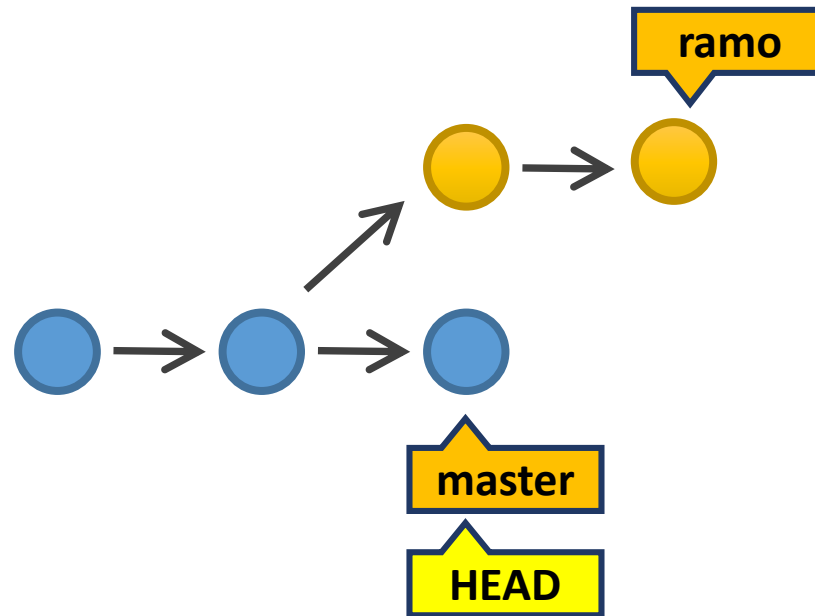


Mesclando *Commits*





Mesclando *Commits*

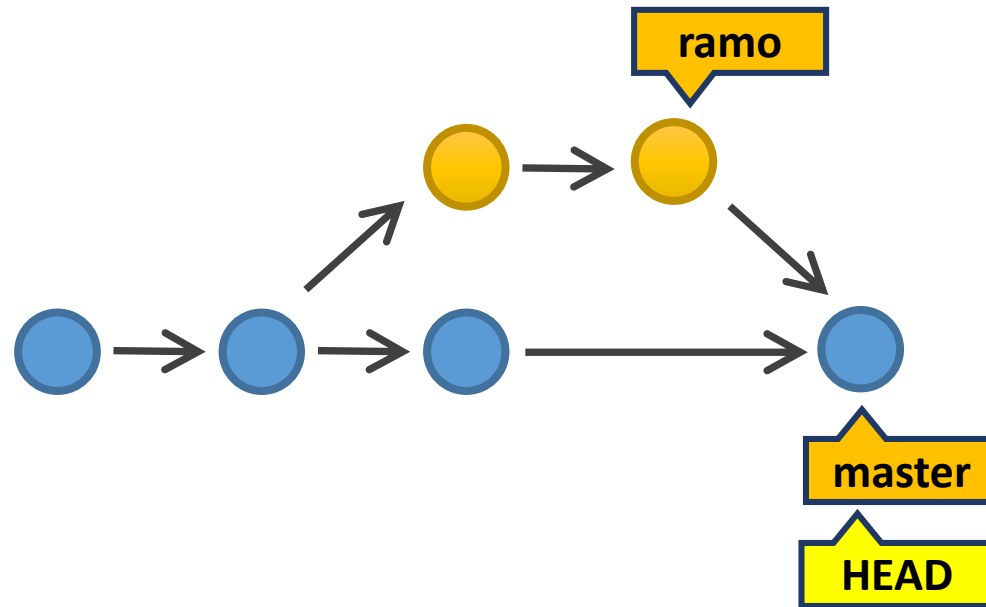


```
$ git checkout master
```

Alterna para o *branch* master.



Mesclando *Commits*

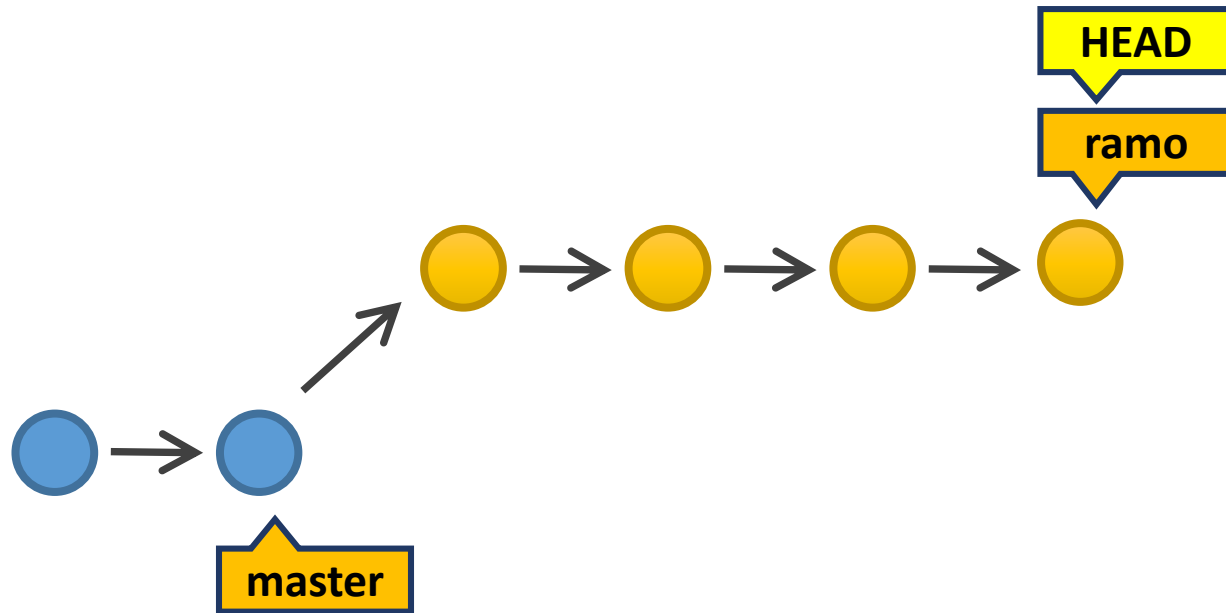


```
$ git merge ramo
```

Realiza um *merge* no *branch* master a partir do *branch* ramo.

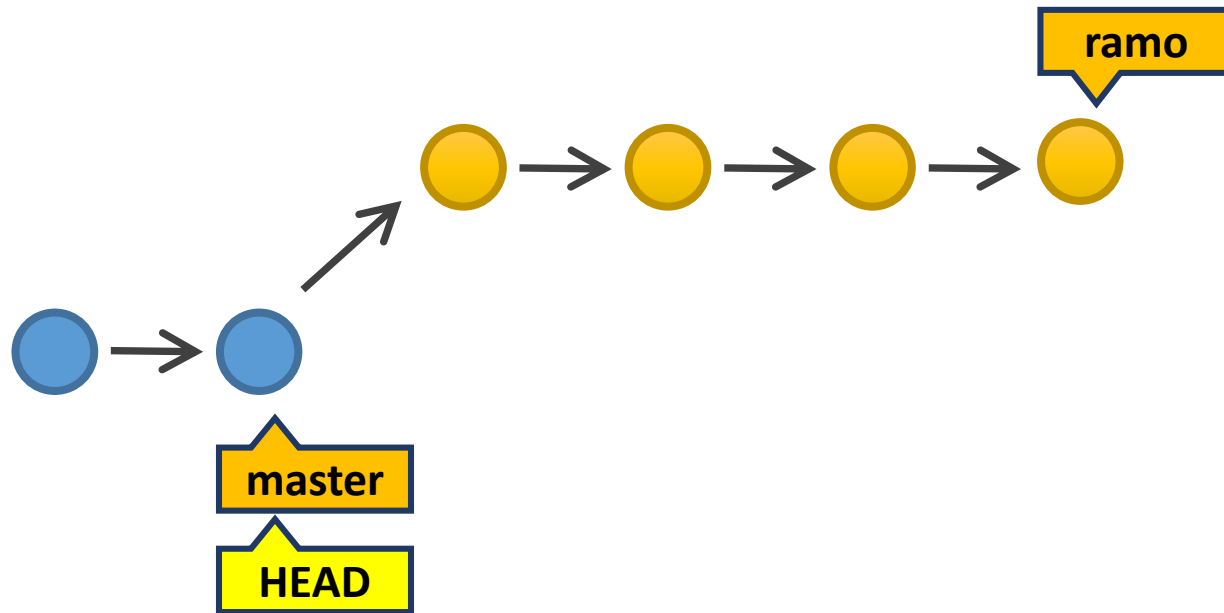


Mesclando *Commits* com *Fast-forward*





Mesclando *Commits* com *Fast-forward*

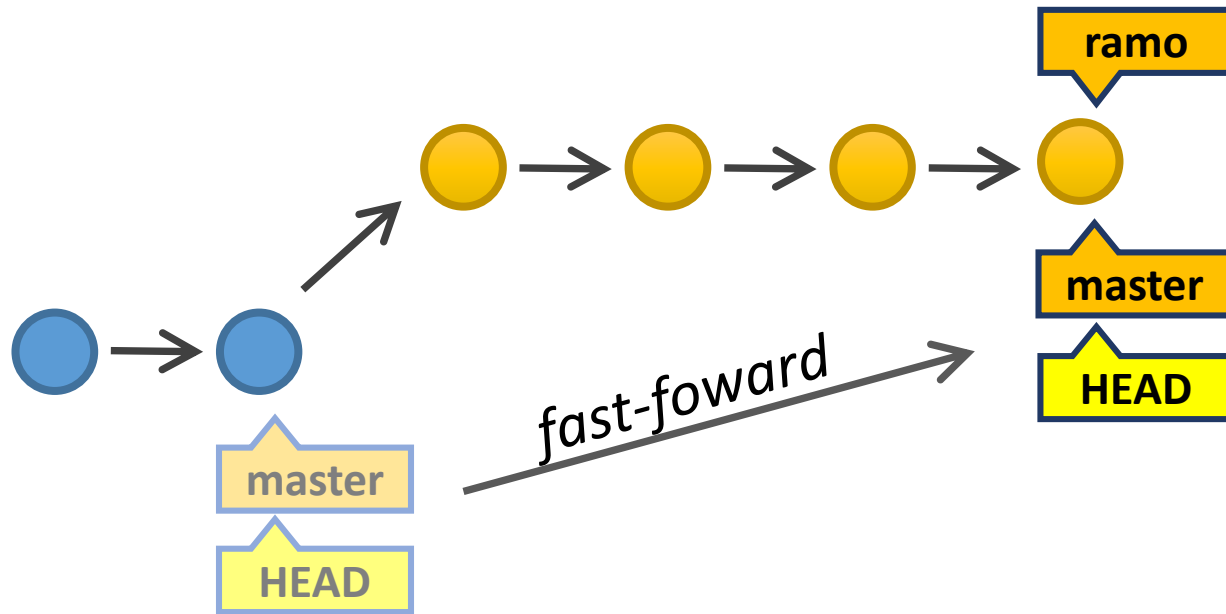


```
$ git checkout master
```

Alterna para o *branch* master.



Mesclando *Commits* com *Fast-forward*

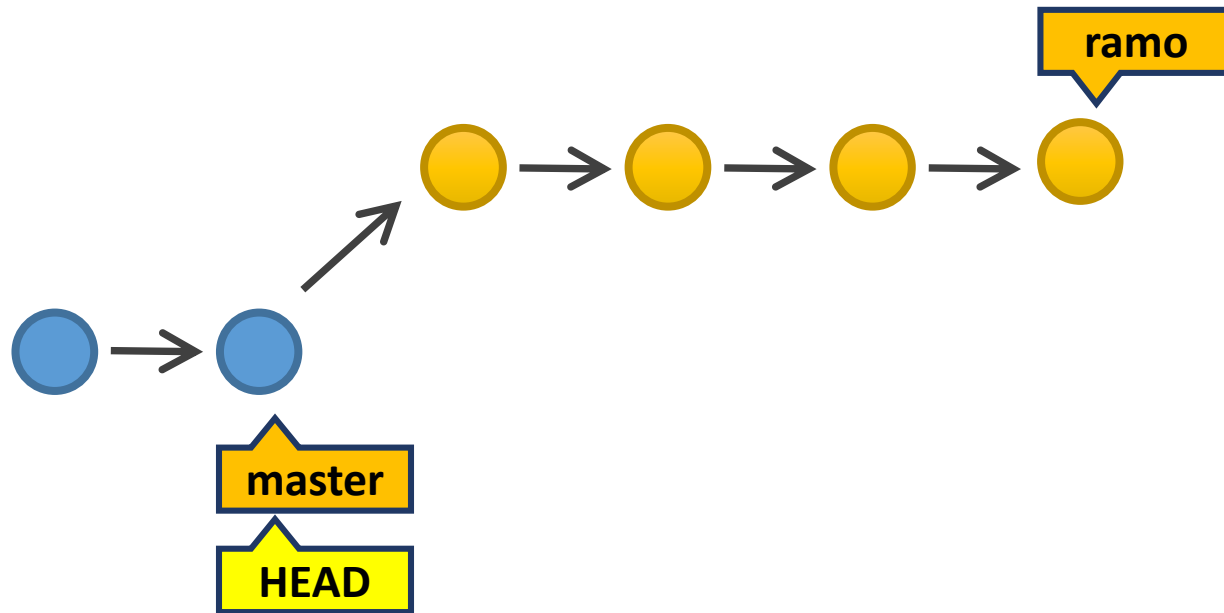


```
$ git merge ramo
```

Neste caso, não é necessário nenhum *commit* para realizar a mesclagem. Ocorre apenas um avanço rápido (*ff*).

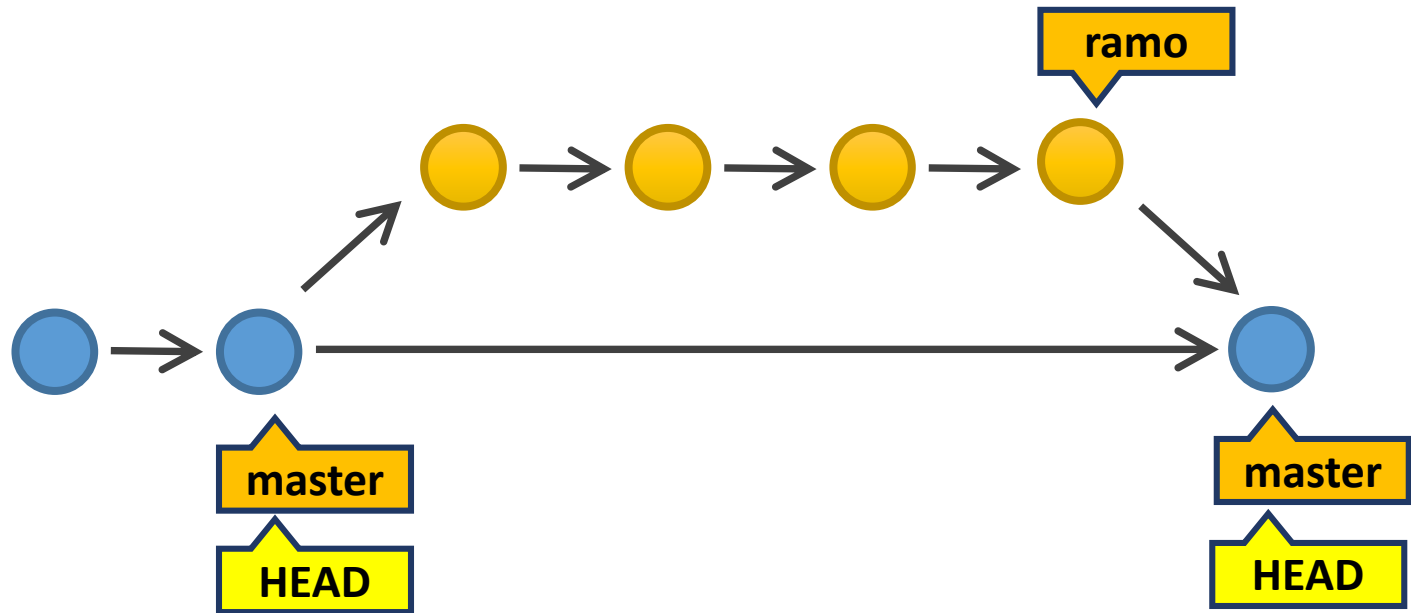


Mesclando *Commits* sem *Fast-forward*





Mesclando *Commits* sem *Fast-forward*



```
$ git merge ramo --no-ff
```

Realiza um *merge* com um *commit* obrigatoriamente.
Possibilita uma melhor visualização no histórico.



Mesclando *Commits*

master

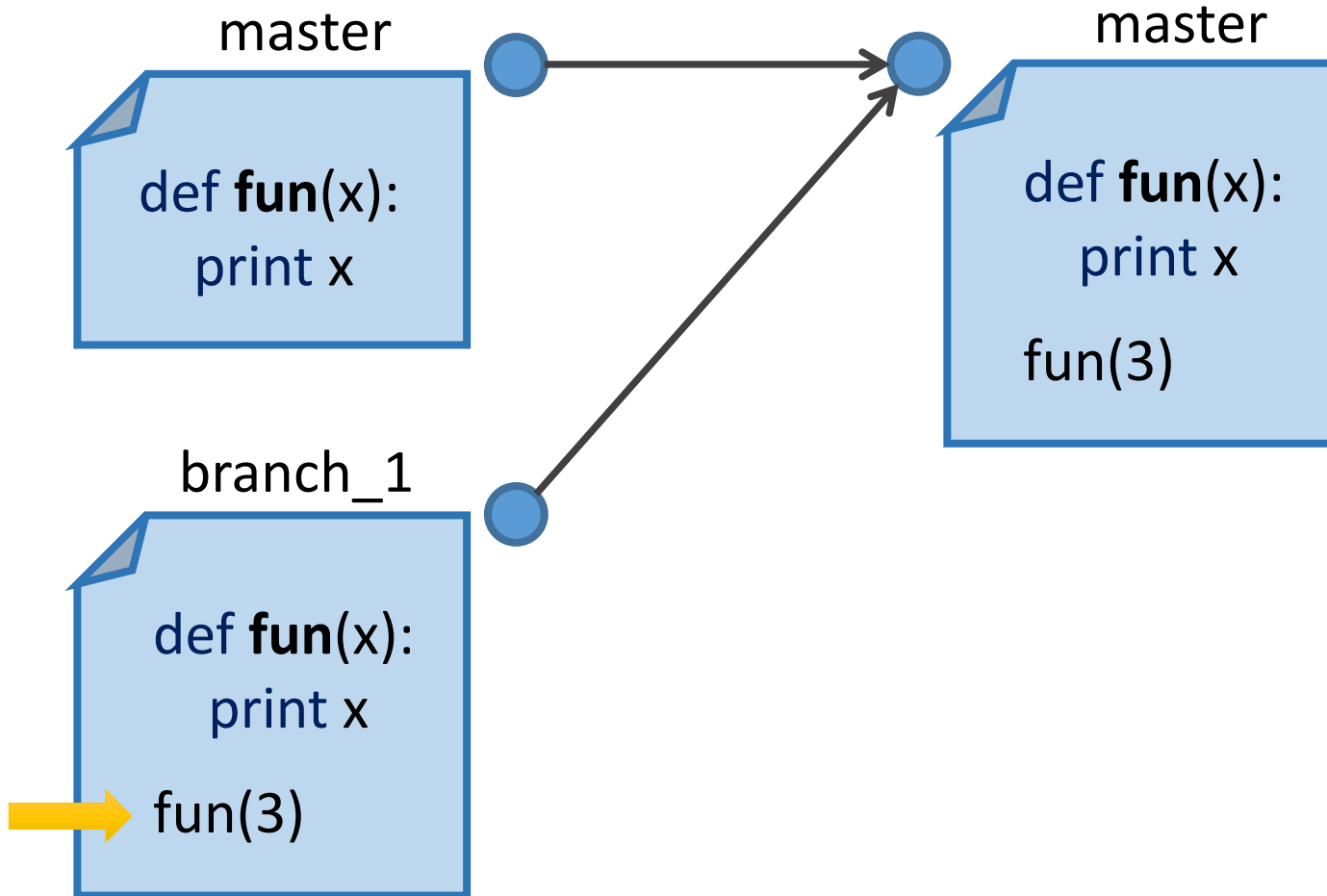
```
def fun(x):  
    print x
```

branch_1

```
def fun(x):  
    print x  
→ fun(3)
```



Mesclando *Commits*





Mesclando *Commits*

branch_2

A yellow arrow pointing from the left towards the code block.

```
def fun(x):  
    print x+x
```

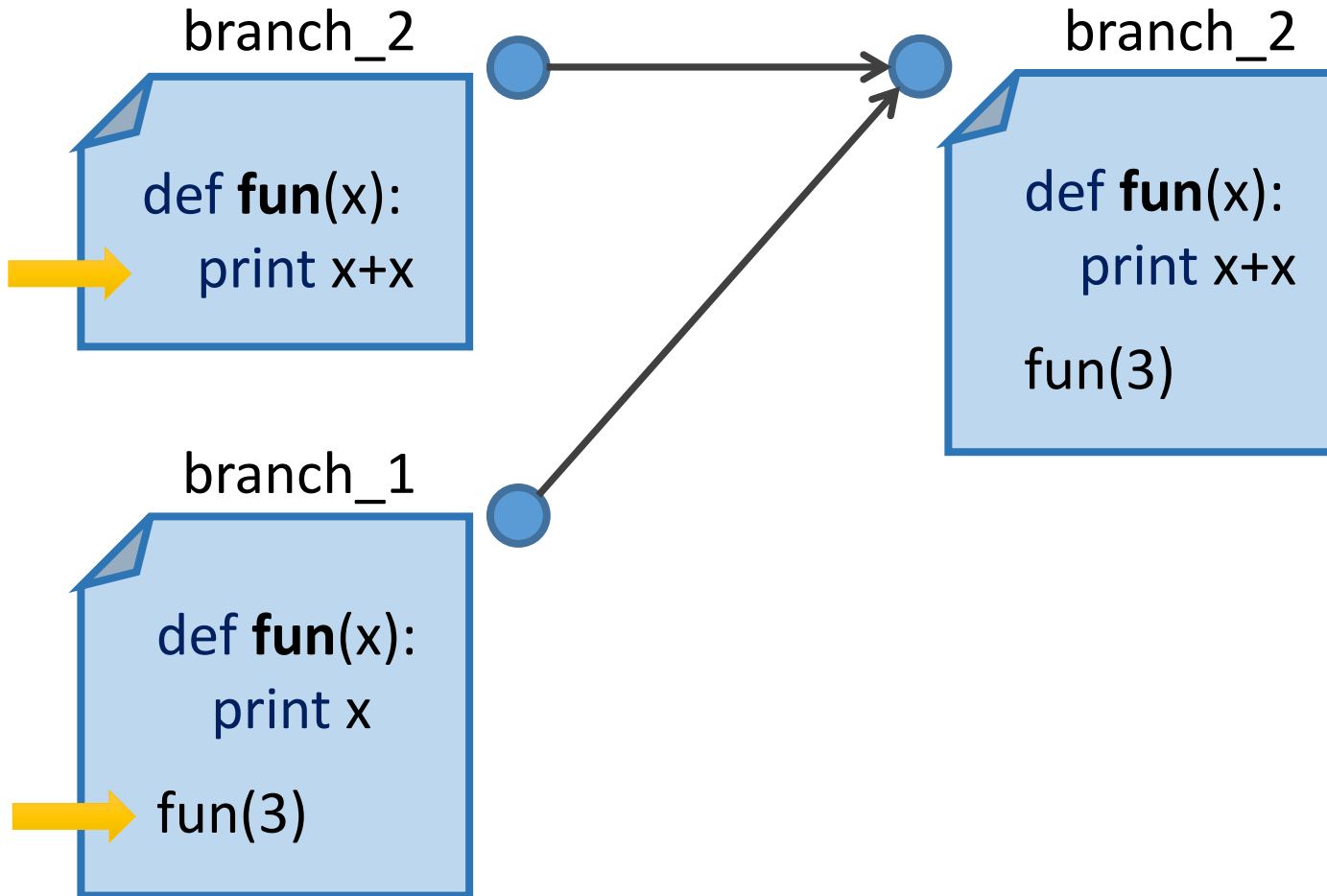
branch_1

A yellow arrow pointing from the left towards the code block.

```
def fun(x):  
    print x  
  
fun(3)
```



Mesclando *Commits*





Mesclando *Commits*

branch_2

A yellow arrow pointing from the left towards the code block.

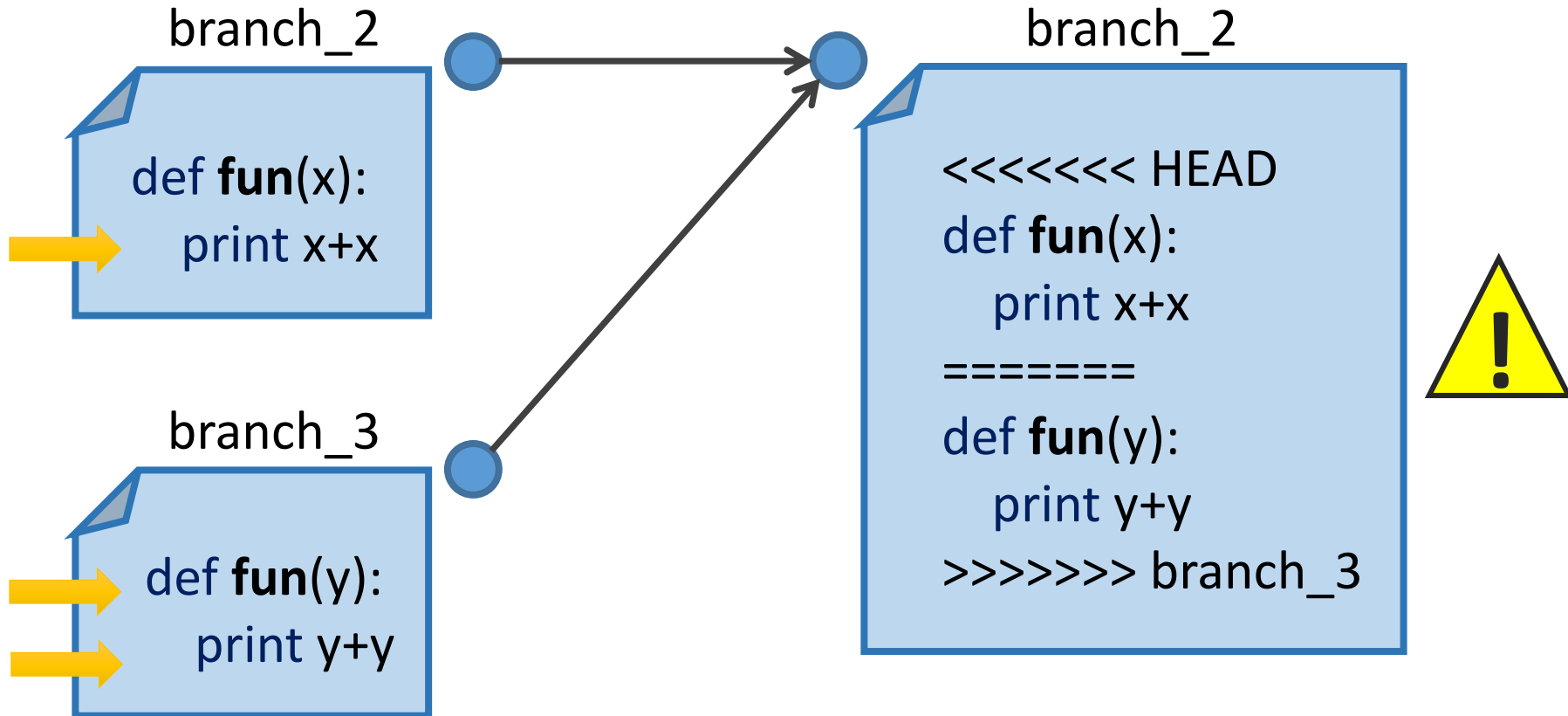
```
def fun(x):  
    print x+x
```

branch_3

Two yellow arrows pointing from the left towards the code block.

```
def fun(y):  
    print y+y
```

Mesclando *Commits*





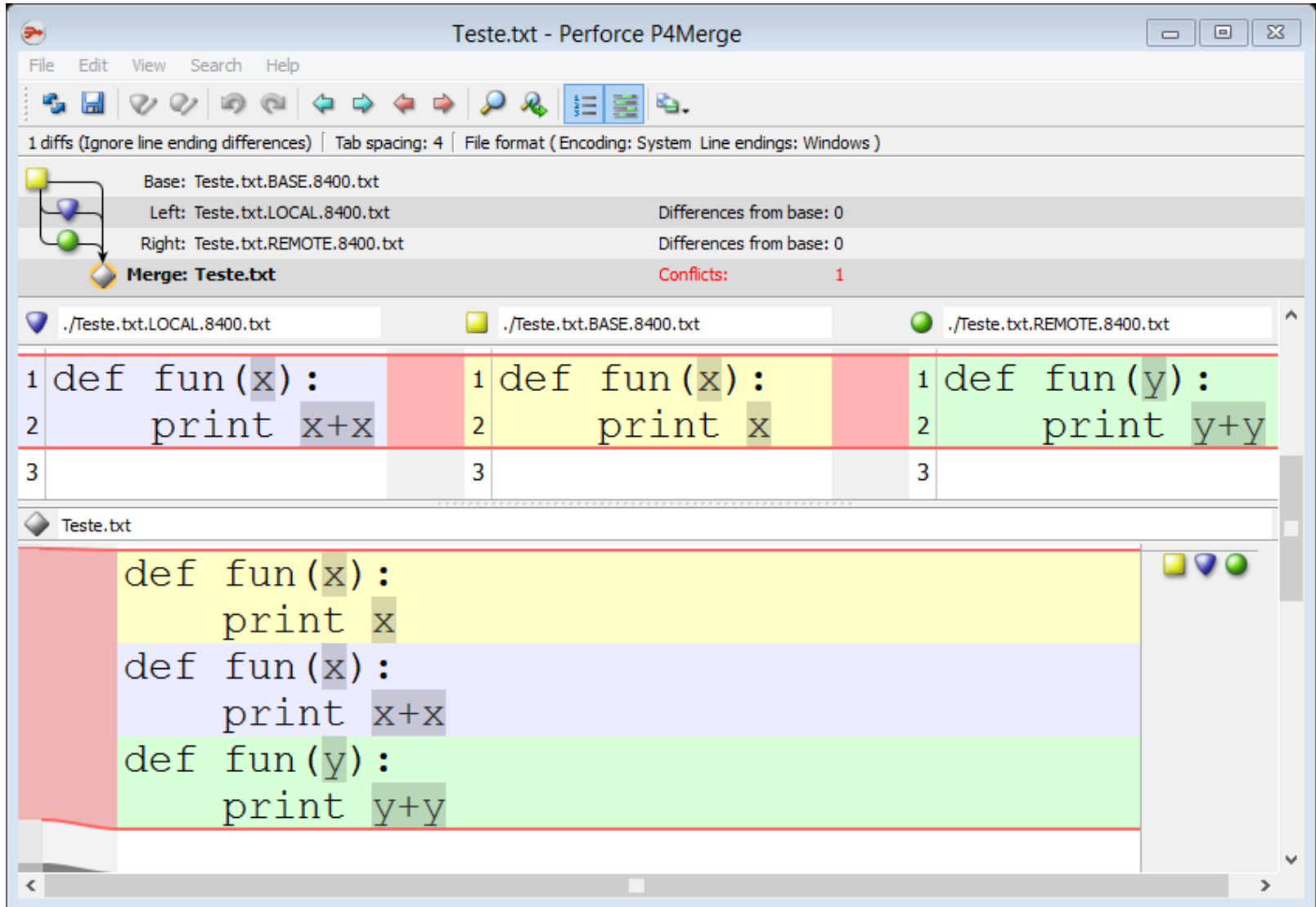
Resolvendo Conflitos

- Alterar o arquivo manualmente
- Utilizar uma interface gráfica
 - kdiff3, tkdiff, meld, xxdiff, vimdiff, [p4merge](#)

Com o p4merge configurado*, basta fazer:

```
$ git mergetool
```

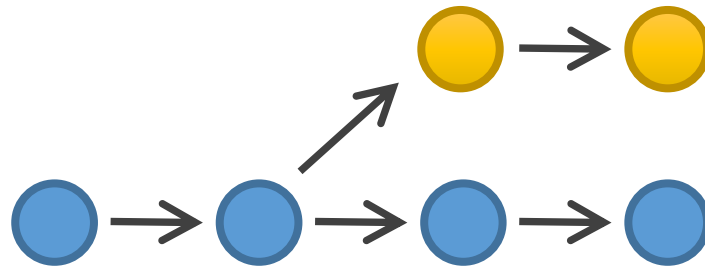
* Veja a seção Configurações



```
$ git commit -a
```



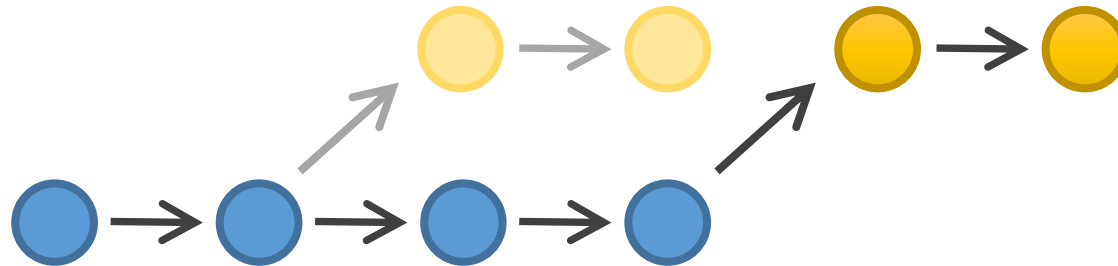

Rebase



```
$ git rebase <base> [-i]
```

Replica os *commits* do *branch* <base> para o atual. Na forma iterativa é possível escolher entre manter, omitir ou editar um *commit*.

Rebase

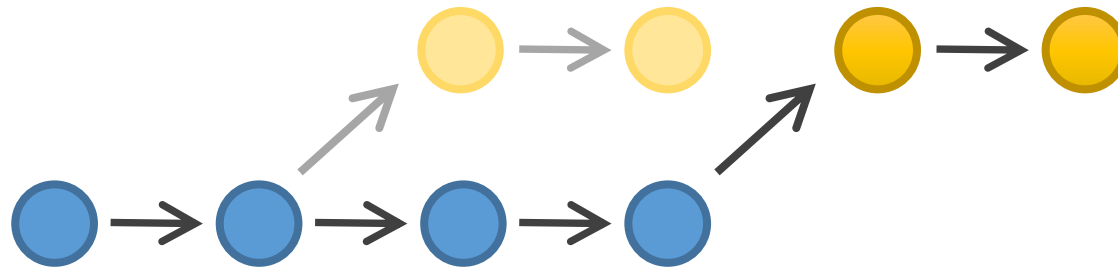


```
$ git rebase <base> [-i]
```

Replica os *commits* do *branch* <base> para o atual. Na forma iterativa é possível escolher entre manter, omitir ou editar um *commit*.



Rebase

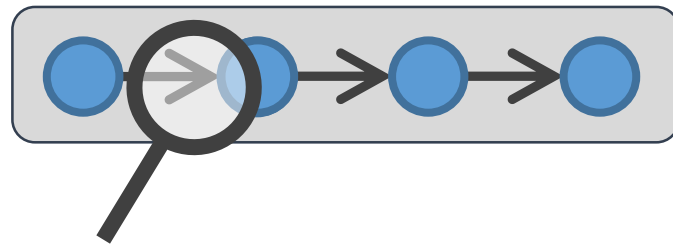


```
$ git rebase <base> [-i]
```

Caso haja algum conflito:

```
$ git mergetool
```

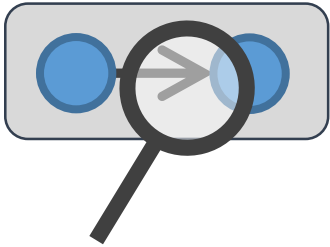
```
$ git rebase --continue
```



Analizando o Log



Analizando o *Log*



```
$ git shortlog
```

Exibe a primeira linha dos *commits* que cada autor enviou.

```
$ git shortlog -s
```

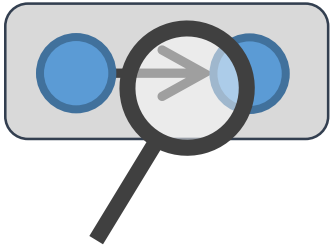
Exibe o número de *commits* que cada autor enviou.

```
$ git shortlog -n
```

Exibe, em ordem numérica, o número de *commits* que cada autor enviou.



Analizando o *Log*



```
$ git log
```

Exibe o *log* de *commits*.

```
$ git log -<n>
```

Exibe os últimos <n> *commits*.

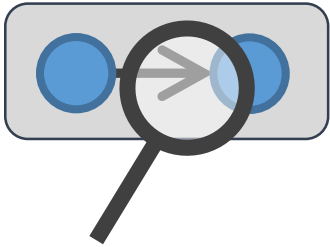
```
$ git log --since==<date>
```

Exibe os *commits* desde a data <date>.

Ex: “3.weeks”, “yesterday”, “3.minutes”



Analizando o *Log*



```
$ git log --graph
```

Exibe o *log* em forma de gráfico.

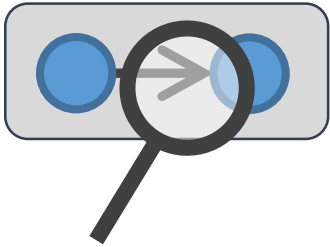
```
$ git log --oneline
```

Exibe o *log*, um *commit* (abreviado) por linha.

```
$ git log --all
```

Exibe o *log* de todas as *tags*, *branches*, ...

Analizando o *Log*



```
$ git log --decorate
```

Exibe o *log* destacando *branch*, *tags*, ...

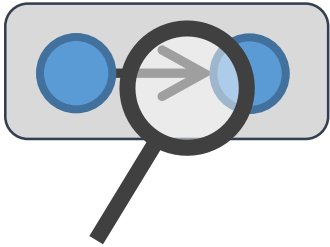
```
$ git log --author=<autor>
```

Exibe os *commits* realizados pelo autor <autor>.

```
$ gitk
```

Exibe o *log* em uma interface gráfica.

Analizando o *Log*



```
$ git log <arquivo>
```

Exibe o *log* de modificações do <arquivo>.

```
$ git log -- <arquivo>
```

Exibe o *log* de modificações do <arquivo> mesmo se ele tiver sido excluído.

```
$ git log <intervalo_commits>
```

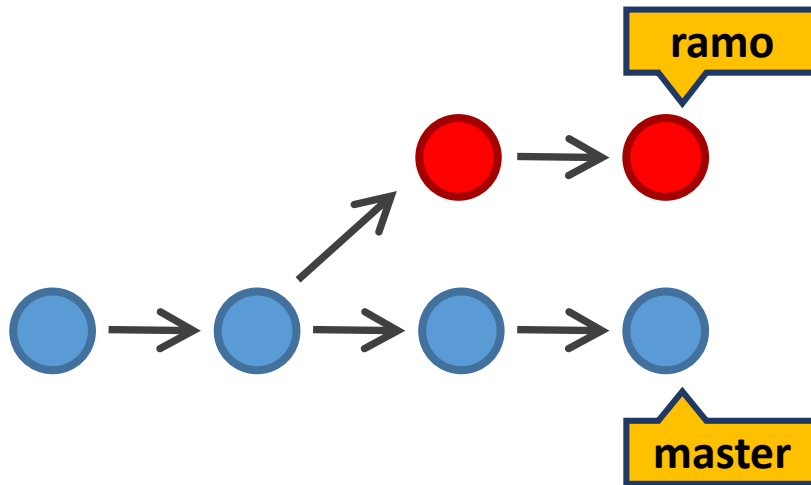
Exibe os *commits* no <intervalo_commits>.



Intervalo de *Commits*

<commit1>..<commit2>

Selecione os *commits* que são alcançados pelo *commit* <commit2>, mas não pelo *commit* <commit1>.

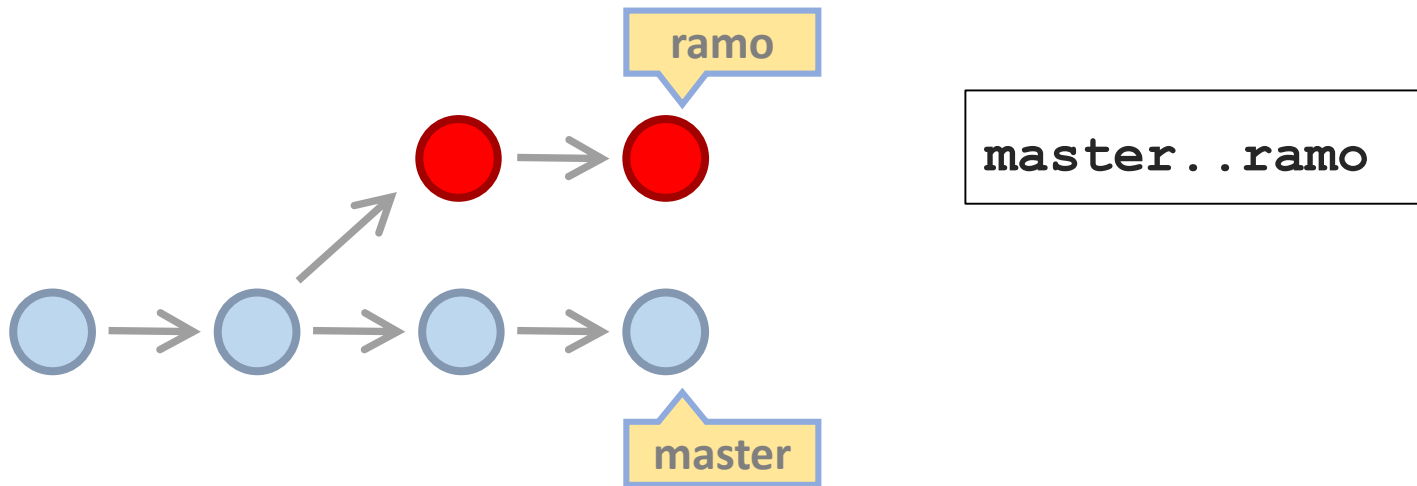




Intervalo de *Commits*

<commit1>..<commit2>****

Seleciona os *commits* que são alcançados pelo *commit* <commit2>, mas não pelo *commit* <commit1>.

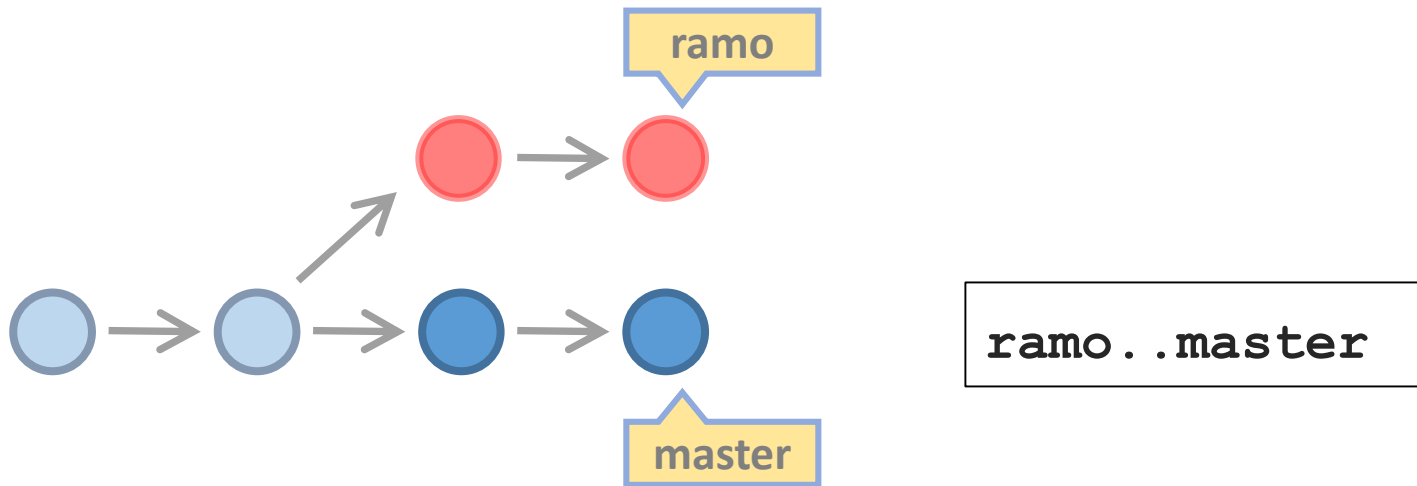




Intervalo de *Commits*

<commit1>..<commit2>****

Seleciona os *commits* que são alcançados pelo *commit* <commit2>, mas não pelo *commit* <commit1>.

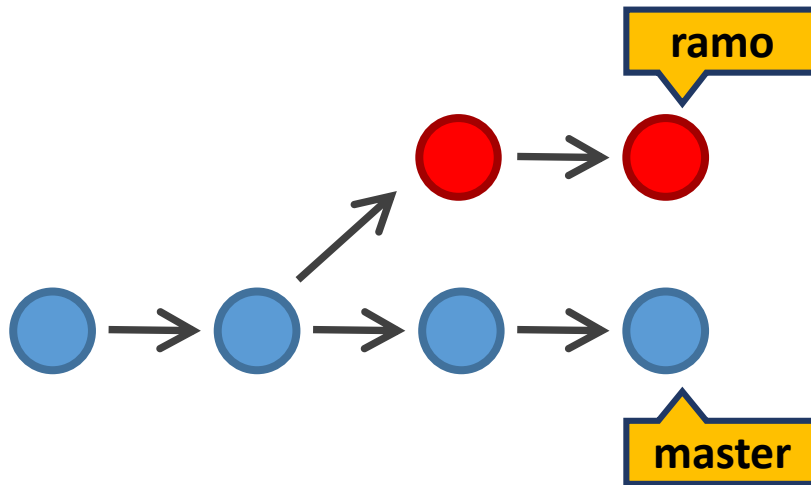




Intervalo de *Commits*

<commit1>...<commit2>

Seleciona os *commits* que são alcançados pelos *commits* <commit1> ou <commit2>, mas não pelos dois ao mesmo tempo.

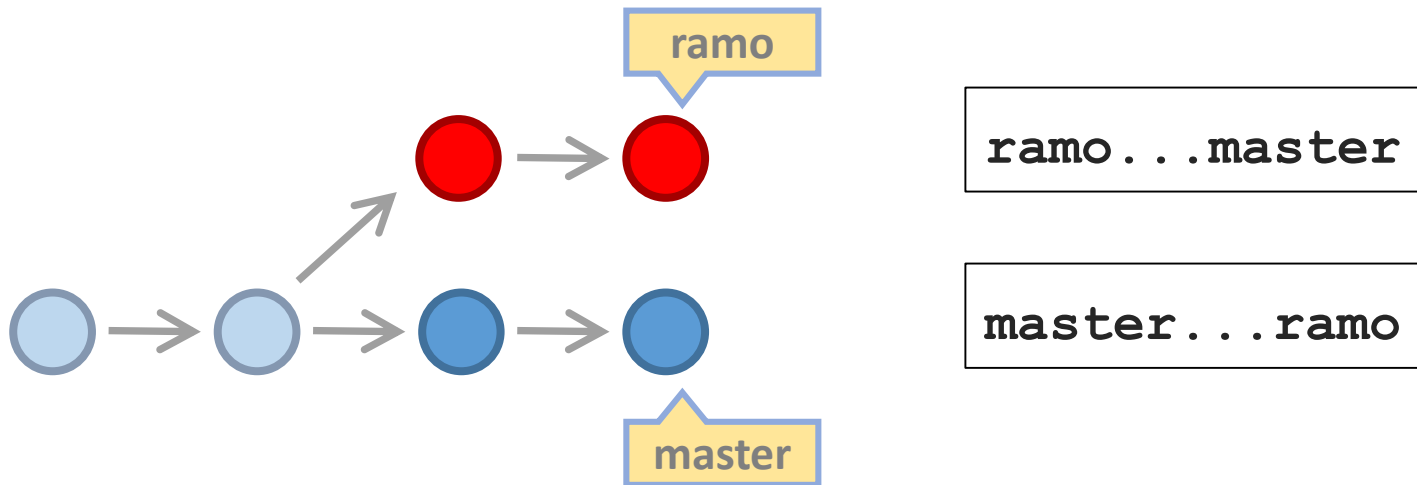


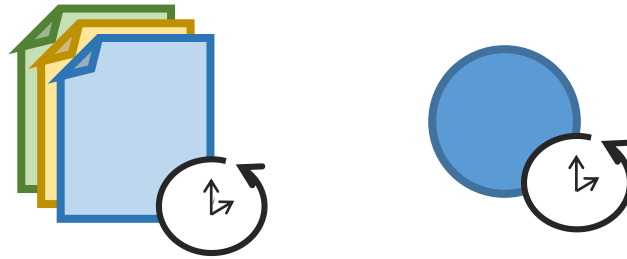


Intervalo de *Commits*

<commit1>...<commit2>

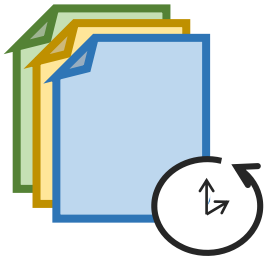
Seleciona os *commits* que são alcançados pelos *commits* <commit1> ou <commit2>, mas não pelos dois ao mesmo tempo.





Desfazendo Ações

Recuperando Arquivos



```
$ git checkout [--] <arquivo>
```

Recupera o arquivo <arquivo> do último *commit*.

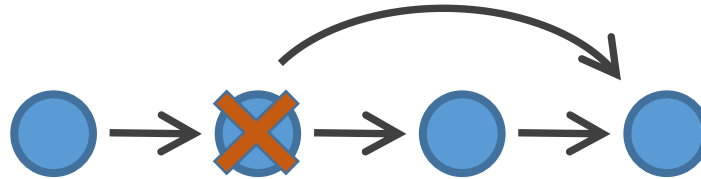
```
$ git checkout <commit> <arq>
```

Recupera o arquivo <arq> do *commit* <commit>.

```
$ git checkout <commit>
```

Recupera os arquivos do *commit* <commit>.

Revertendo *Commits*

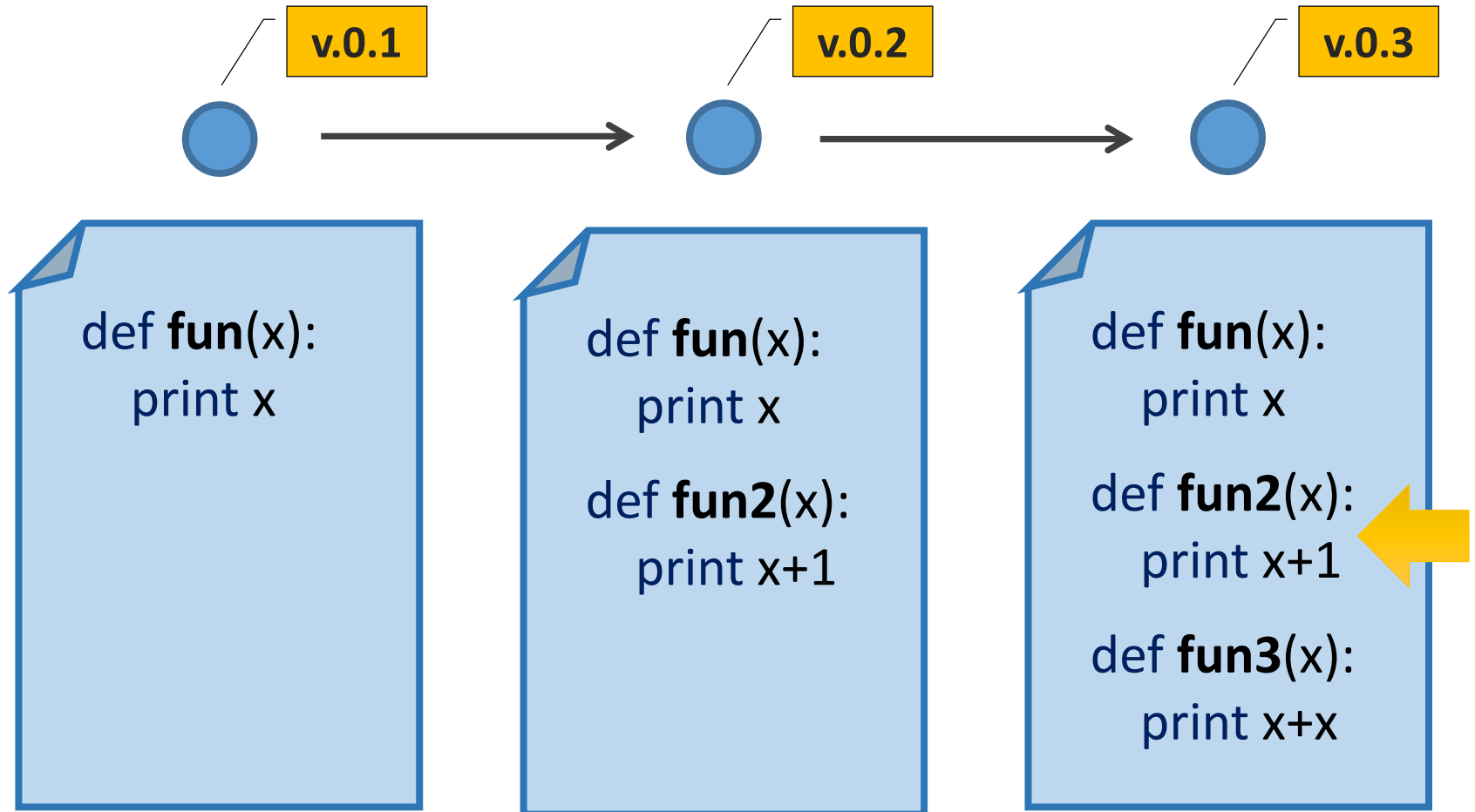


```
$ git revert <commit>
```

Cria um novo *commit* no *branch* atual que desfaz o que foi introduzido no *commit* <commit>.

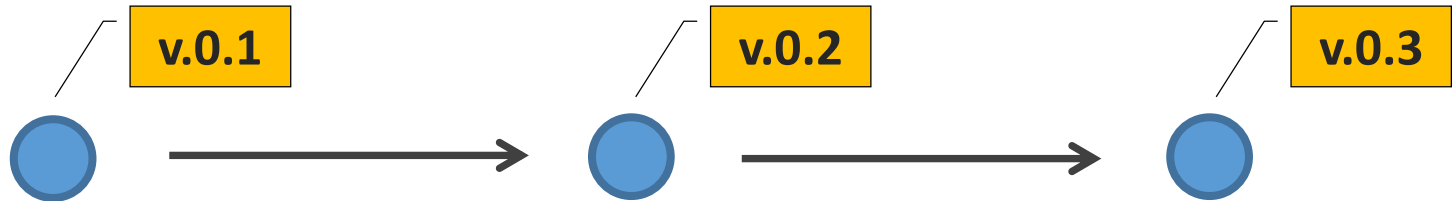
- Consertar um *bug* introduzido por um *commit*.
- Não remove o *commit* <commit>

Revertendo *Commits*





Revertendo *Commits*



```
$ git revert v.0.2
```



Conflitos aparecem!

```
$ git mergetool
```



Novo Documento de Texto.txt - Perforce P4Merge

File Edit View Search Help

2 diffs (Ignore line ending differences) | Tab spacing: 4 | File format (Encoding: System Line endings: Windows)

Base: Novo Documento de Texto.txt.BASE.56.txt
Left: Novo Documento de Texto.txt.LOCAL.56.txt Differences from base: 1
Right: Novo Documento de Texto.txt.REMOTE.56.txt Differences from base: 1
Merge: Novo Documento de Texto.txt Conflicts: 0

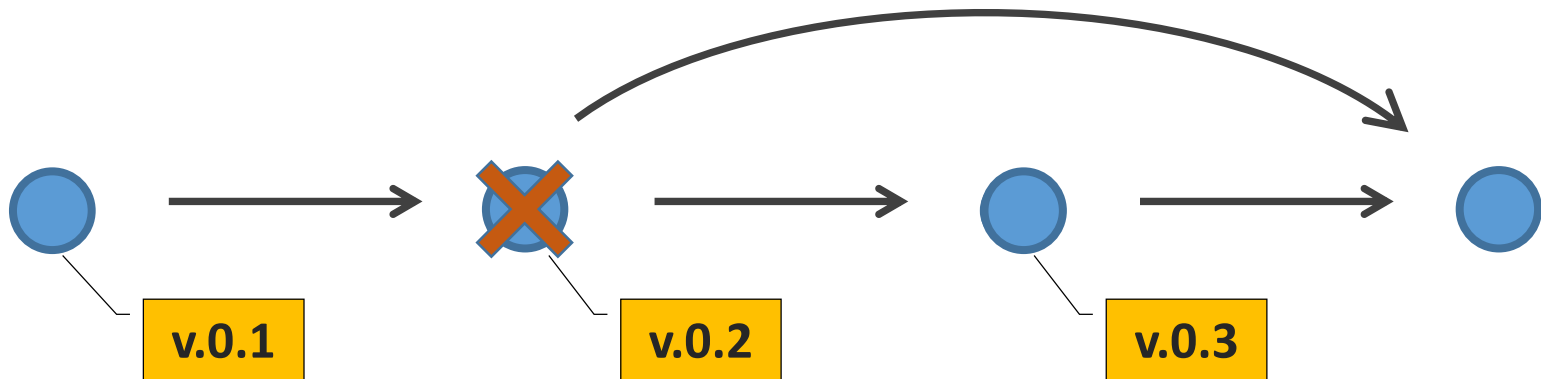
Novo Documento de Texto.txt.LOCAL.56.txt .\Novo Documento de Texto.txt.BASE.56.txt Novo Documento de Texto.txt.REMOTE.56.txt

1 def fun_1(x): 2 print x	1 def fun_1(x): 2 print x	1 def fun_1(x): 2 print x
3 4 def fun_2(x): 5 print x	3 4 def fun_2(x): 5 print x	3
6 7 def fun_3(x): 8 print x	6	

Novo Documento de Texto.txt

```
def fun_1(x):  
    print x  
  
def fun_2(x):  
    print x  
  
def fun_3(x):  
    print x
```

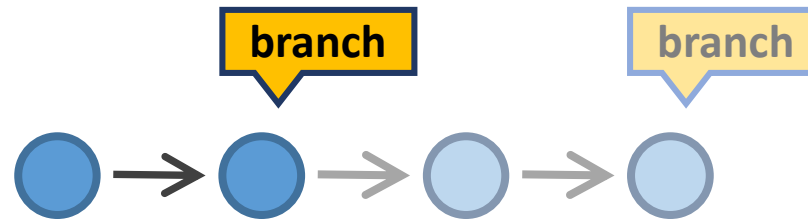
Revertendo *Commits*



```
$ git commit -am "Fixed bug in fun2"
```



“Excluindo” *Commits*

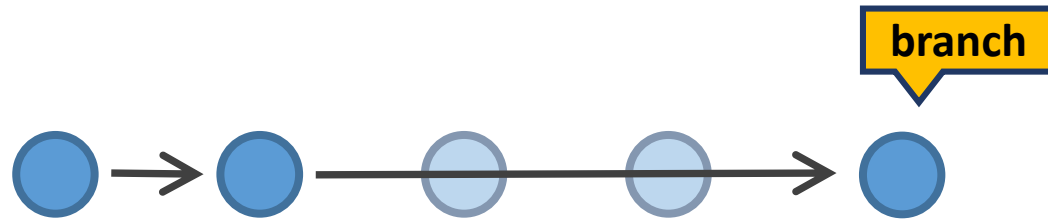


```
$ git reset --soft <commit>
```

Altera apenas o HEAD para o *commit* <commit>. Não altera a área transitória nem o diretório de trabalho.



“Excluindo” *Commits*

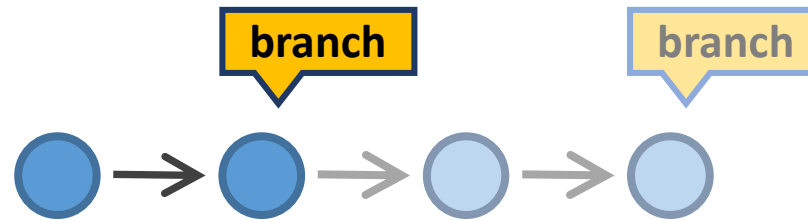


```
$ git reset --soft <commit>
```

```
$ git commit
```

Substitui os *commits* por um único *commit*. O diretório de trabalho não é alterado.

“Excluindo” *Commits*



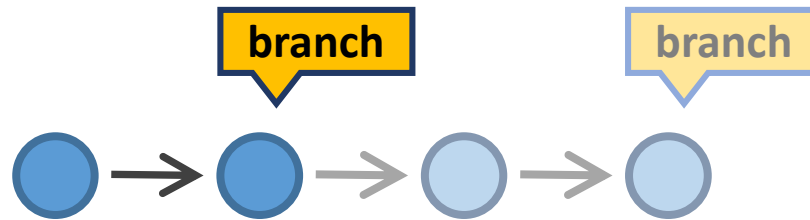
```
$ git reset --hard <commit>
```

Altera a área transitória e o diretório de trabalho para o *commit* <commit>.



O comando *git reset* é uma das poucas formas de se perder informação utilizando o *git*, pois os *commits* deixam de aparecer no *git log*.

“Excluindo” *Commits*



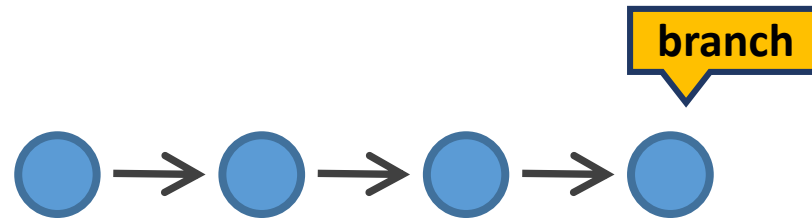
```
$ git reset [--mixed] <commit>
```

Altera apenas a área transitória para o *commit* <commit>. Não altera o diretório de trabalho.

É necessário um *git add* para selecionar os arquivos do diretório que irão para o próximo *commit*, caso contrário irá o arquivo da área transitória.

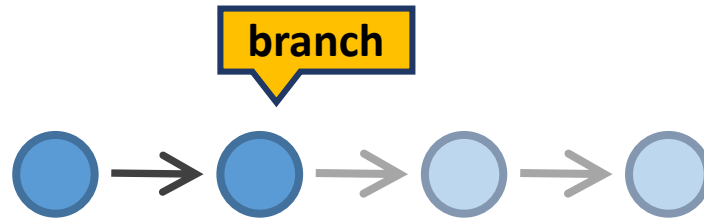


“Excluindo” *Commits*





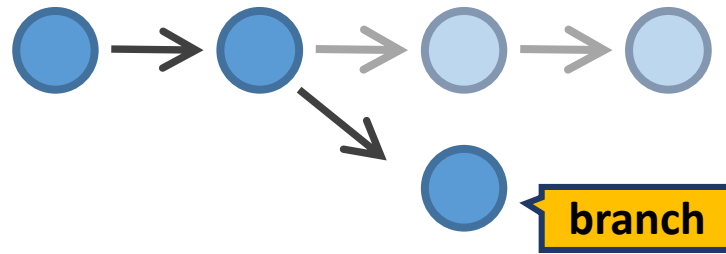
“Excluindo” *Commits*



```
$ git reset <commit>
```



“Excluindo” *Commits*



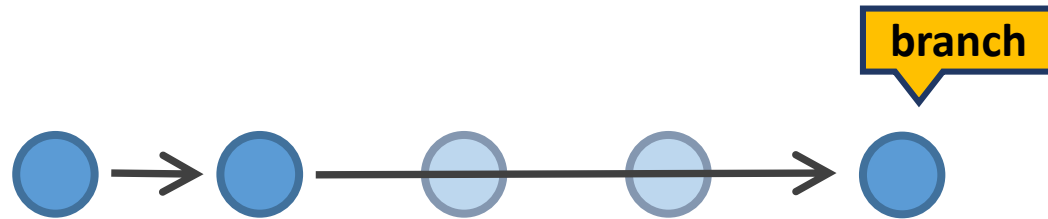
```
$ git reset <commit>
```

```
$ git commit
```

Mantém os arquivos da área transitória, ou seja, do *commit* <commit>.



“Excluindo” *Commits*






```
$ git reset <commit>
```

```
$ git add <arquivos>
```

```
$ git commit
```

Mantém os arquivos <arquivos> do diretório.

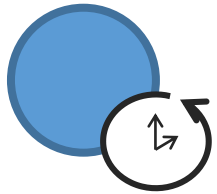
Resumo da Operação *Reset*

	 branch		
Reset	HEAD	Index	Diretório
--soft	Alterado	-	-
--mixed	Alterado	Alterado*	-
--hard	Alterado	Alterado	Alterado

* É possível modificar o *index* utilizando “git add”.



Recuperando *Commits*



```
$ git reflog
```

Exibe o histórico de *hashes* do repositório local.

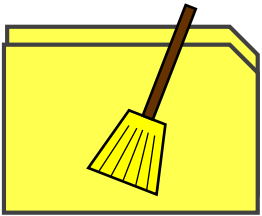
```
$ git reflog
```

```
$ git merge <commit>
```

Adiciona o *commit* <commit> ao *branch* atual.



Limpendo o Diretório

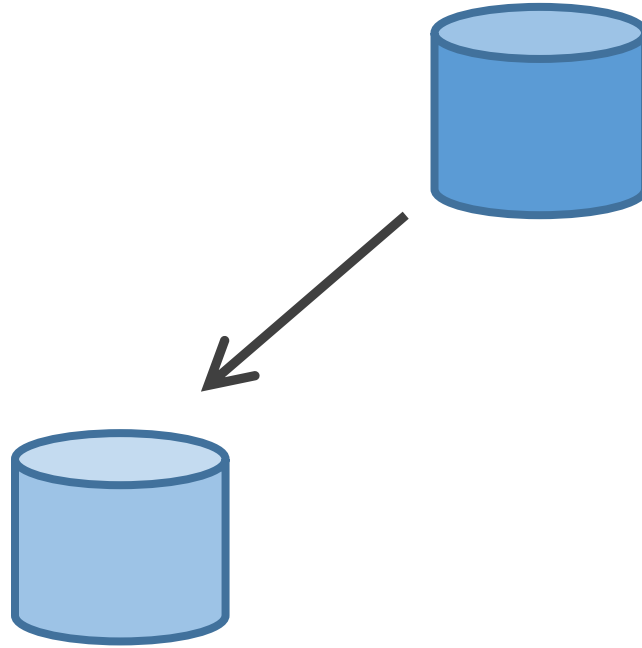


```
$ git clean [-f]
```

Exclui os arquivos que não estão sendo rastreados.
É possível forçar a exclusão.

```
$ git clean -n
```

Exibe os arquivos não rastreados que serão
excluídos.



Repositórios Remotos



Trabalhando com Repositórios Remotos

```
$ git remote -v
```

Lista os repositórios remotos e suas URLs. O repositório clonado é nomeado de *origin*.

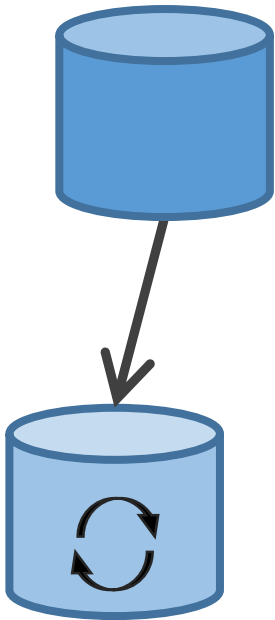
```
$ git remote add <nome> <url>
```

Usa <nome> ao invés da url <url> para se referir ao repositório remoto.

```
$ git remote add pendrive "E:/GitRepo"
```



Atualizando o Repositório Local



```
$ git fetch [<repo>]
```

Baixa todos os dados do repositório <repo>.

```
$ git fetch [<repo>] [<branch>]
```

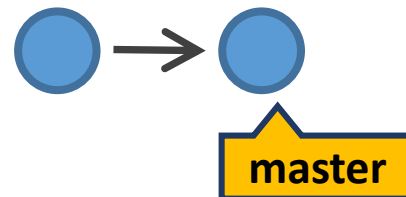
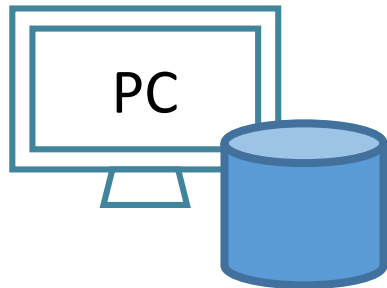
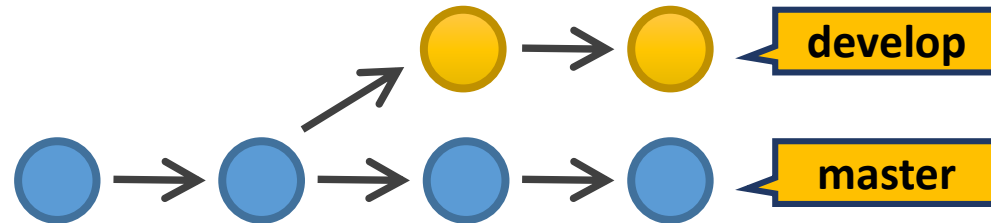
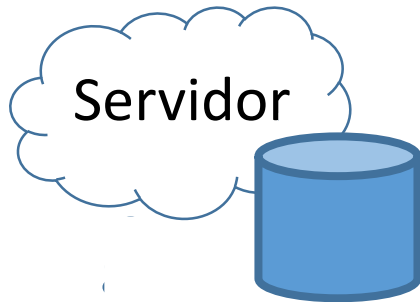
Baixa todos os dados do branch <branch> do repositório <repo>.

```
$ git pull [<repo>]
```

Atualiza todos os dados do repositório <repo>, ou seja, realiza um *fetch* seguido de um *merge*.

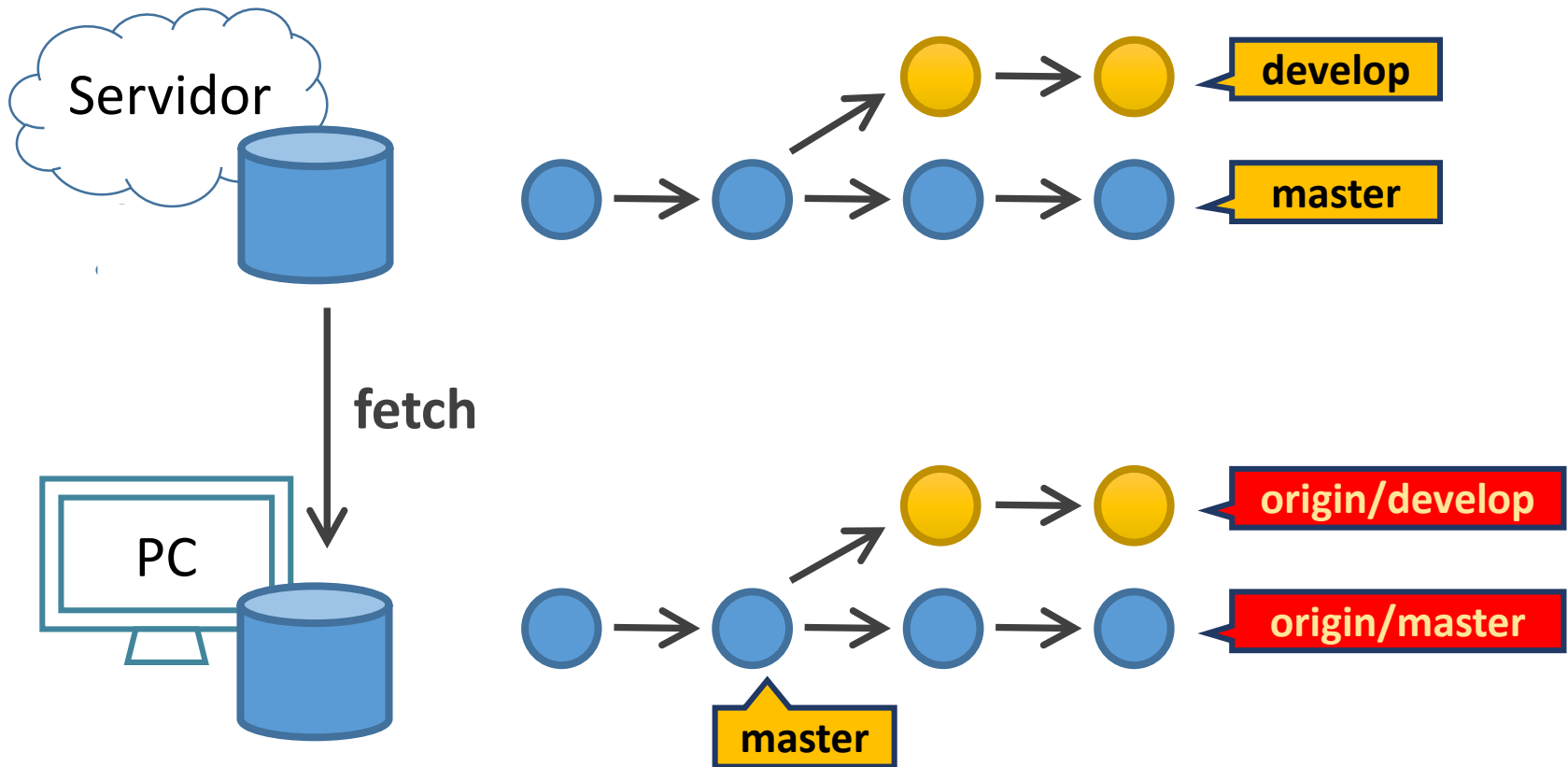


Usando *fetch*





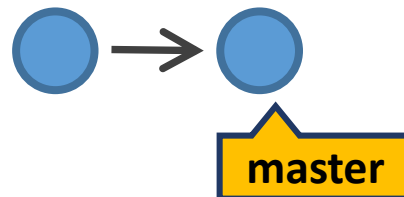
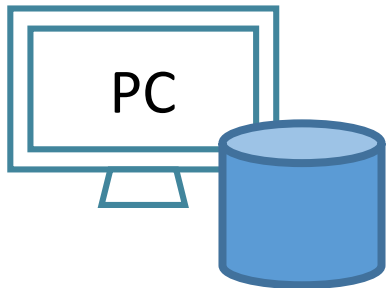
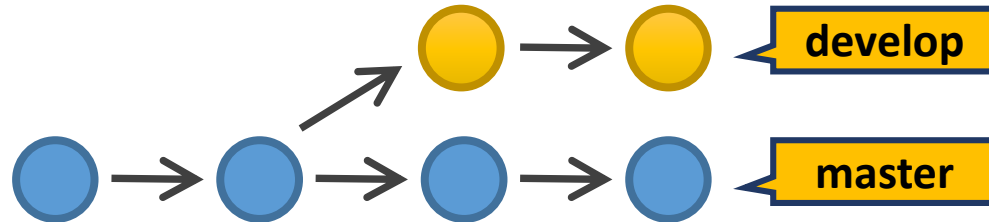
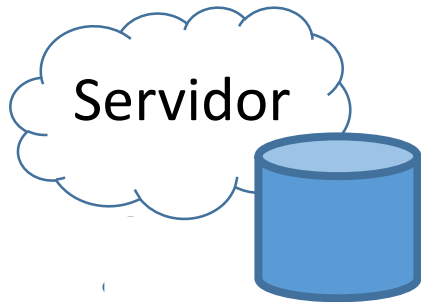
Usando *fetch*



```
$ git fetch #Baixa os dados do servidor
```

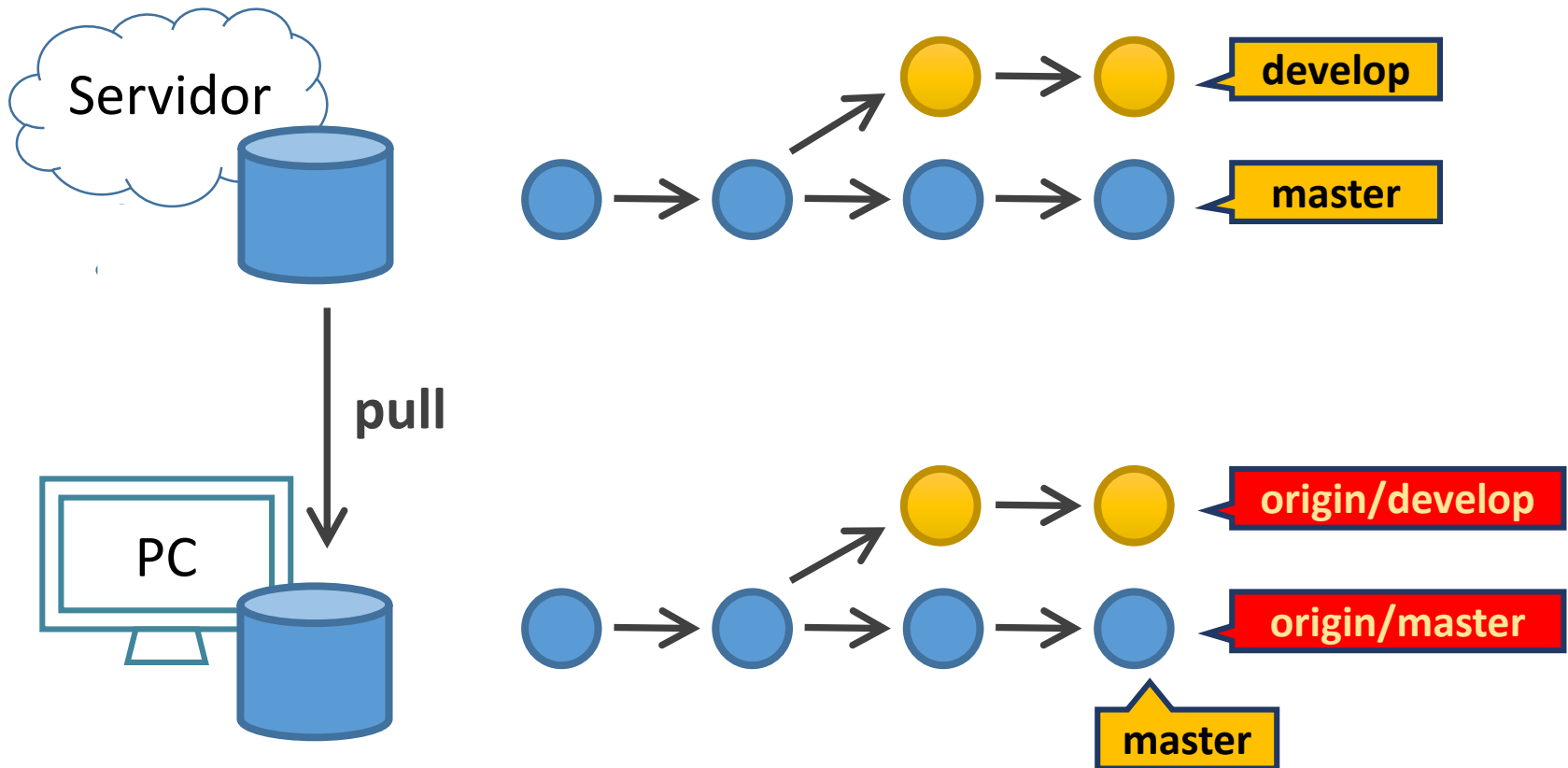


Usando *pull*





Usando *pull*



```
$ git pull #Atualiza o repositório local
```



Excluindo no Repositório Remoto

```
$ git push <repo> :<branch>
```

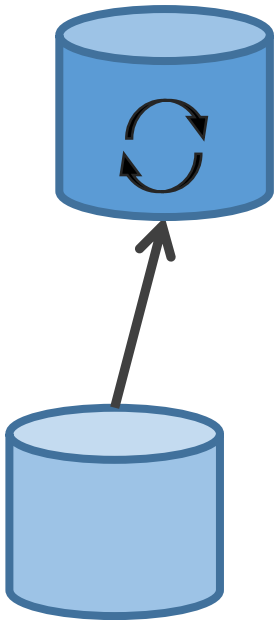
Exclui o *branch* <branch> do repositório <repo>.

```
$ git push <repo> :<tag>
```

Exclui a *tag* <tag> do repositório <repo>.



Enviando Para o Repositório



```
$ git push [<repo>] [<branch>]
```

Envia o branch <branch> para o repositório <repo>. Por padrão <repo> é *origin* e <branch> é o *branch* atual, mas pode ser configurado*.

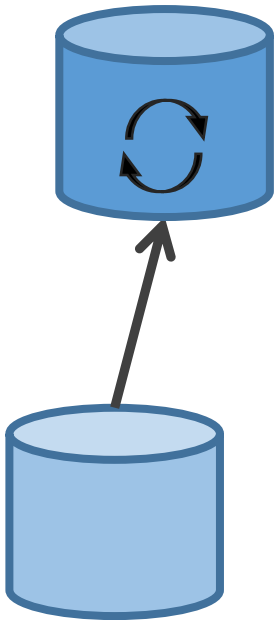
```
$ git push [<repo>] --all
```

Envia o todos os *branches* para o repositório <repo>.

* Veja a seção Configurações



Enviando Para o Repositório



```
$ git push [<repo>] --tags
```

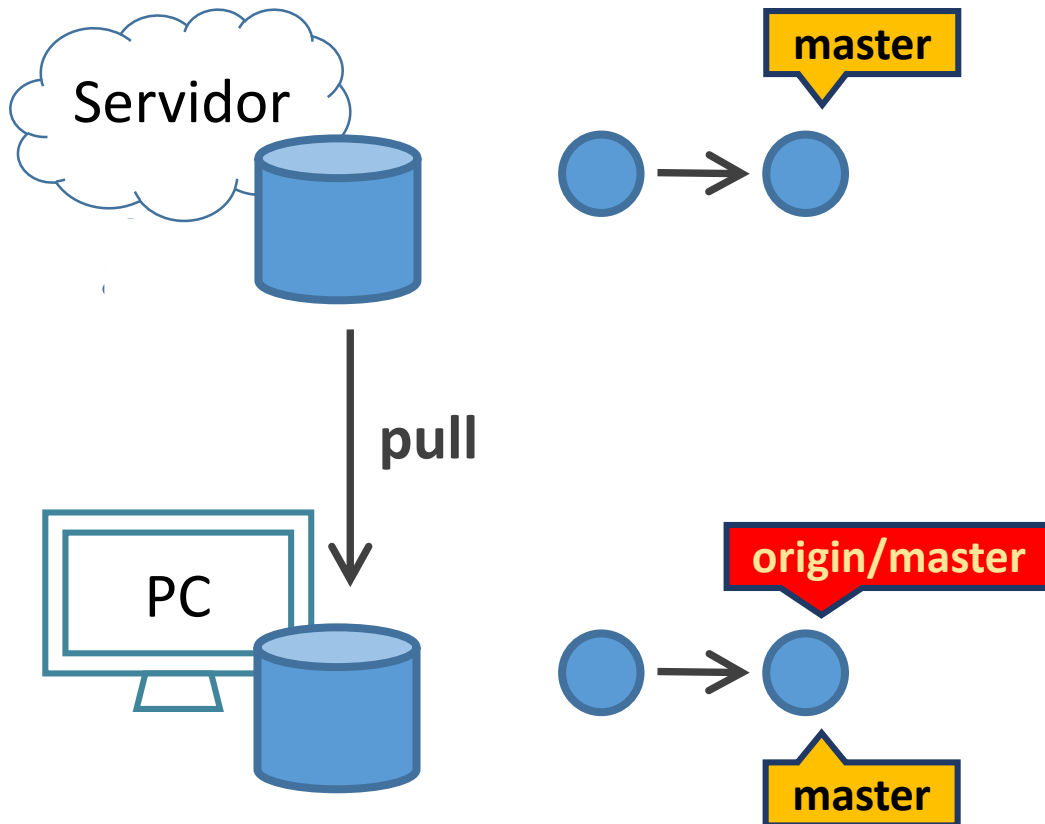
Envia todas as *tags* para o repositório <repo>.

```
$ git push <repo> <tag>
```

Envia a *tag* <tag> para o repositório <repo>.



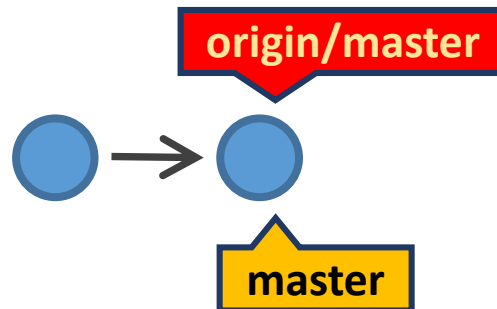
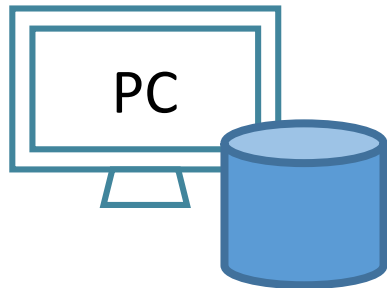
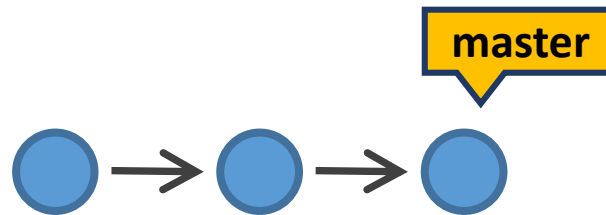
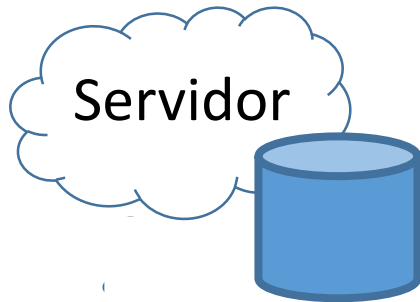
Exemplo de Aplicação



```
$ git pull #Atualiza o repositório local
```



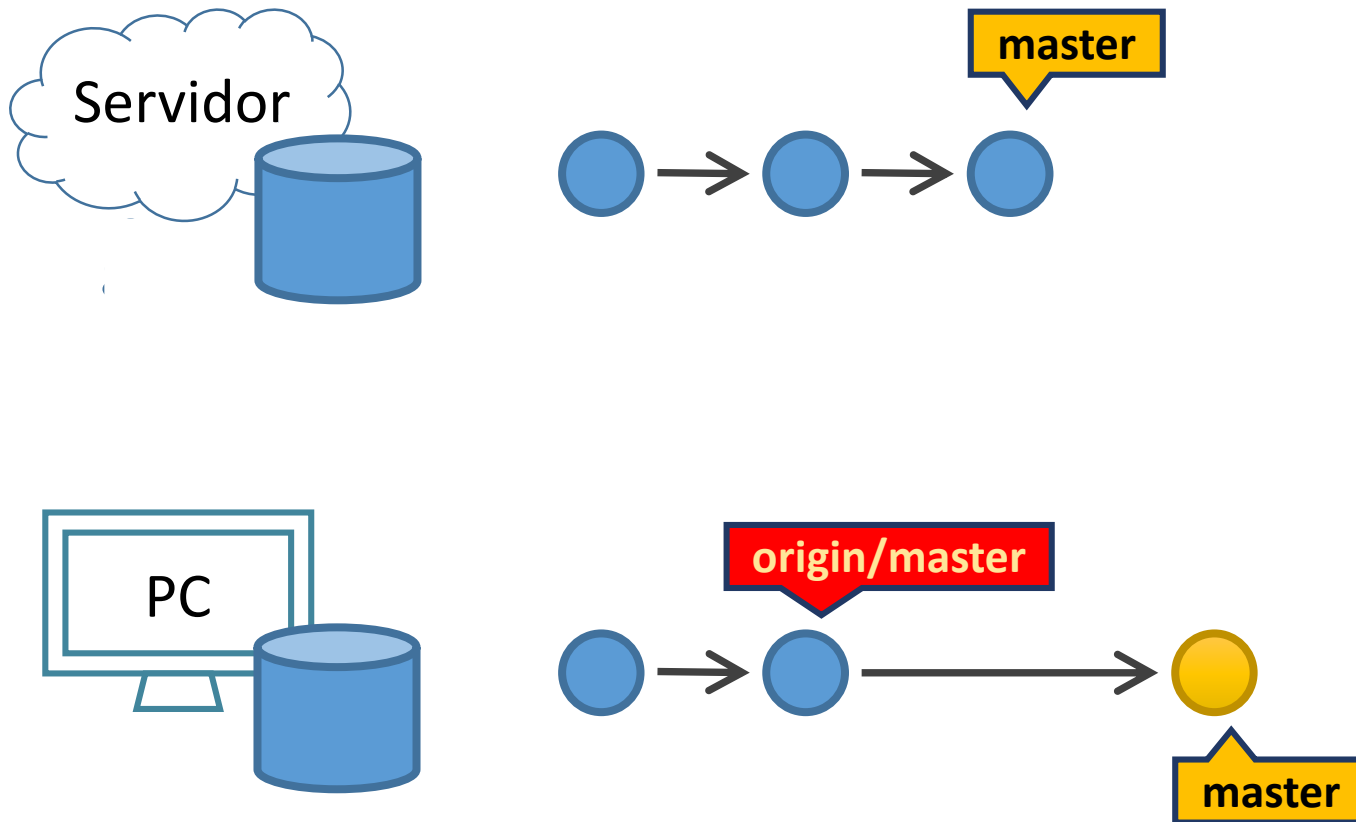
Exemplo de Aplicação



#Modifica o repositório remoto

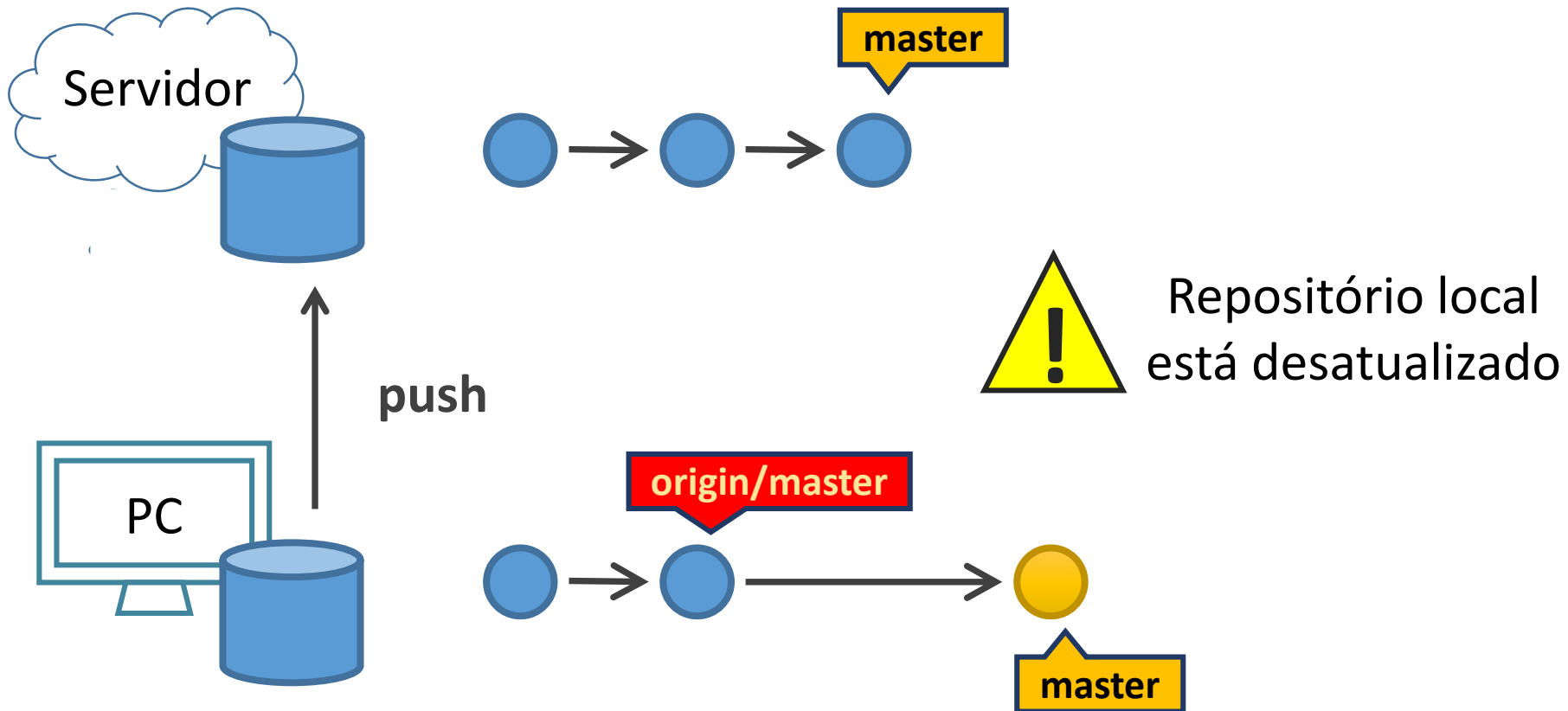


Exemplo de Aplicação



```
$ git commit -a #Altera o repositório local
```

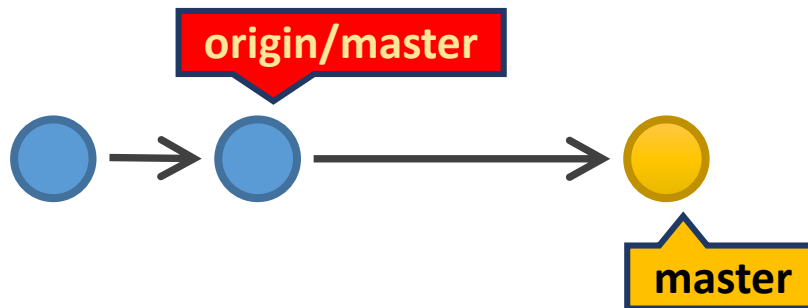
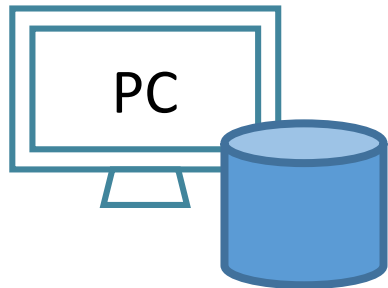
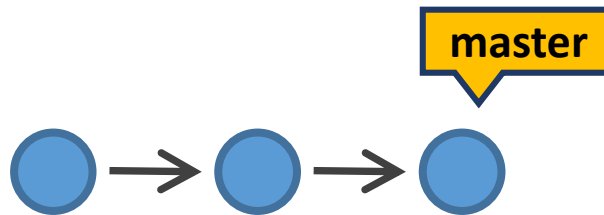
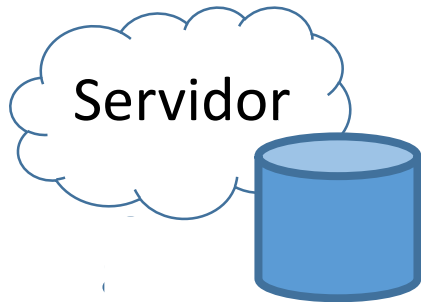
Exemplo de Aplicação



```
$ git push #Tenta atualizar o servidor
```

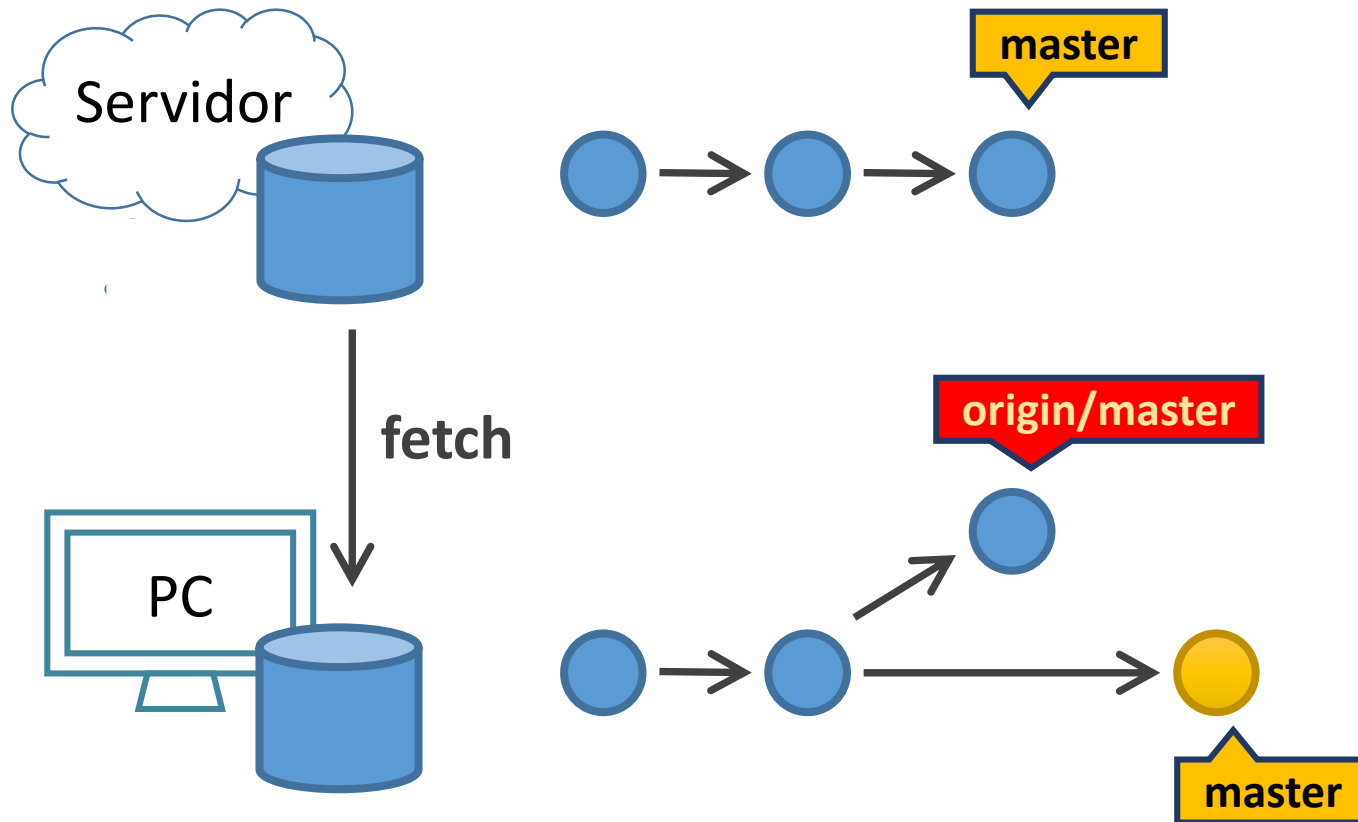


Solução 1: *fetch + rebase + push*





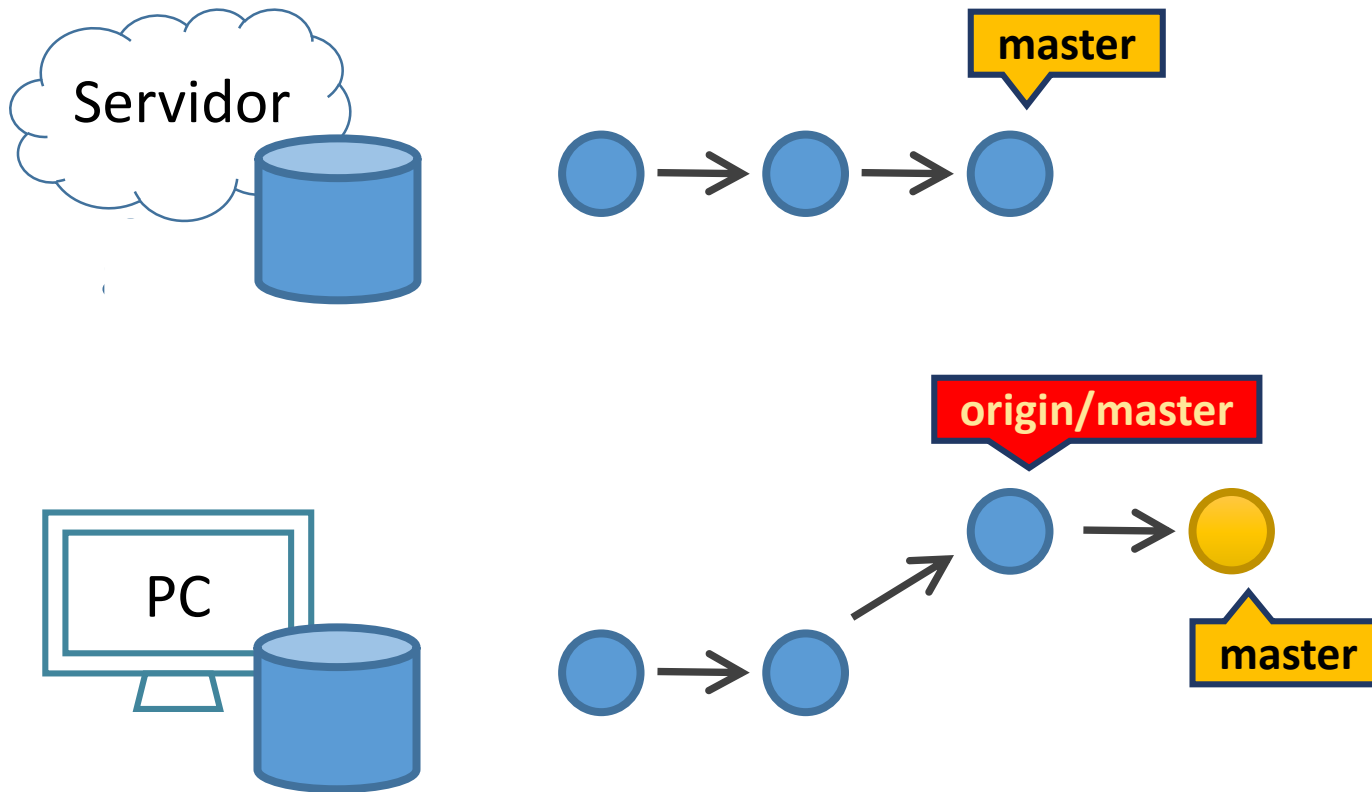
Solução 1: *fetch + rebase + push*



```
$ git fetch #Baixa os dados do servidor
```



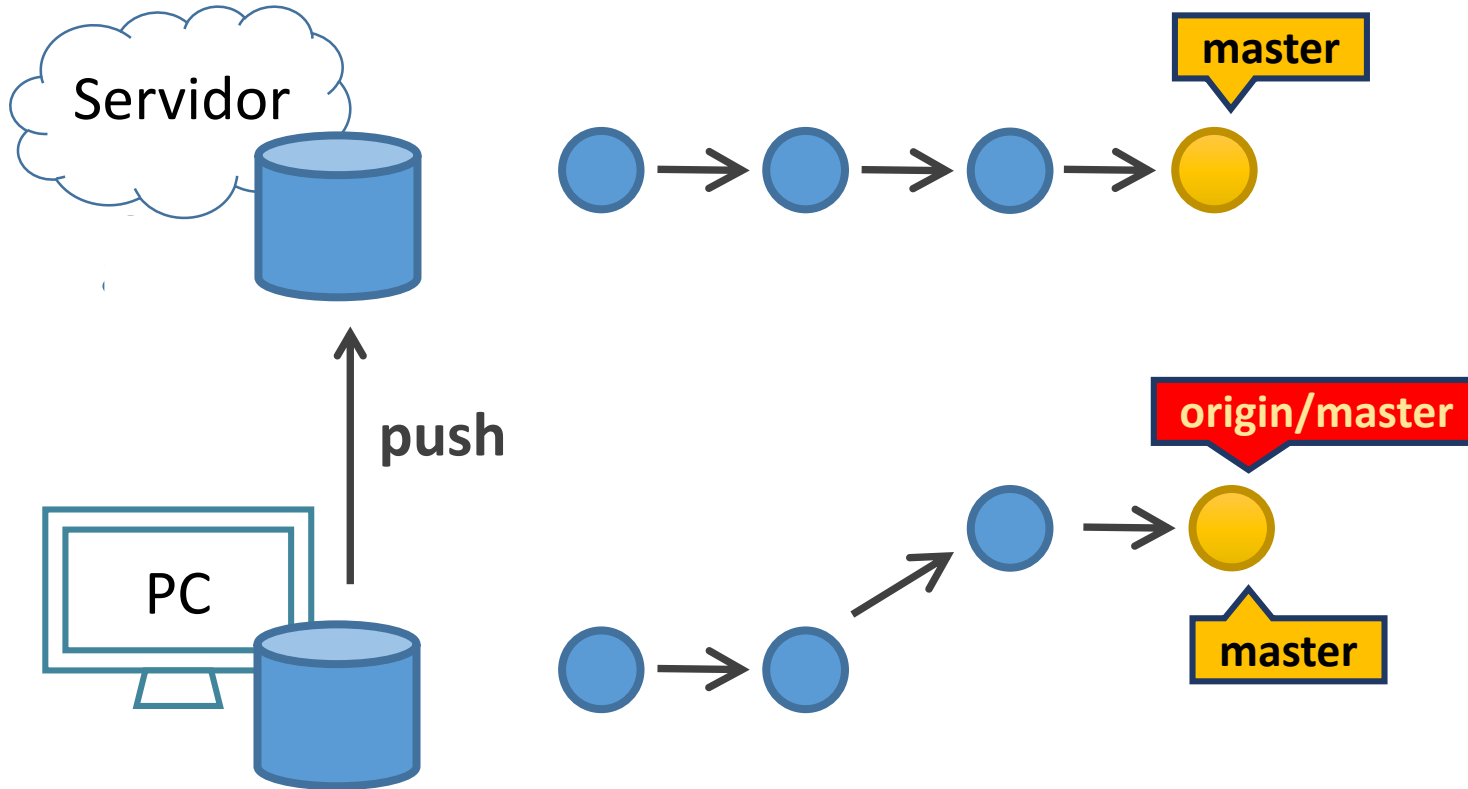

Solução 1: *fetch + rebase + push*



```
$ git rebase origin/master #Realiza o rebase
```

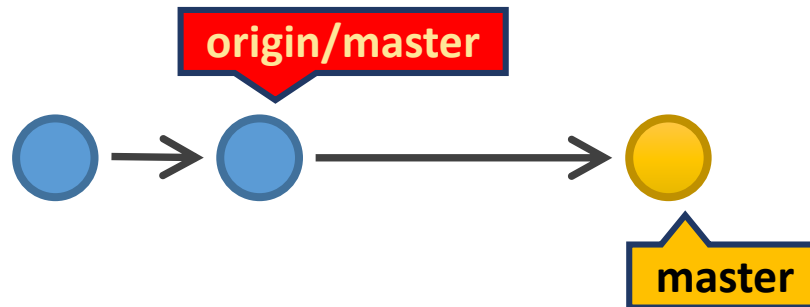
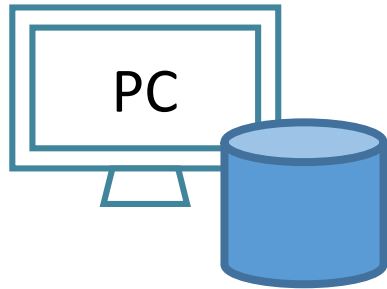
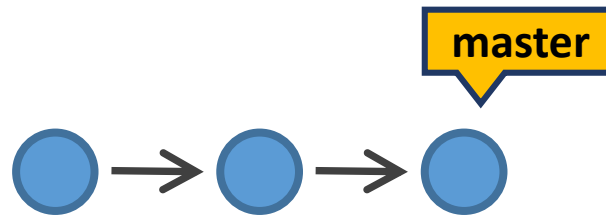
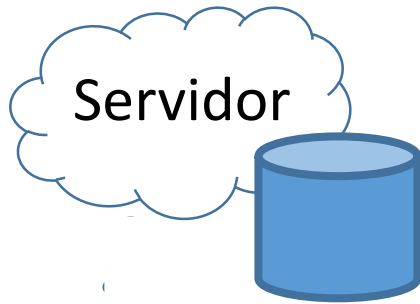


Solução 1: *fetch + rebase + push*



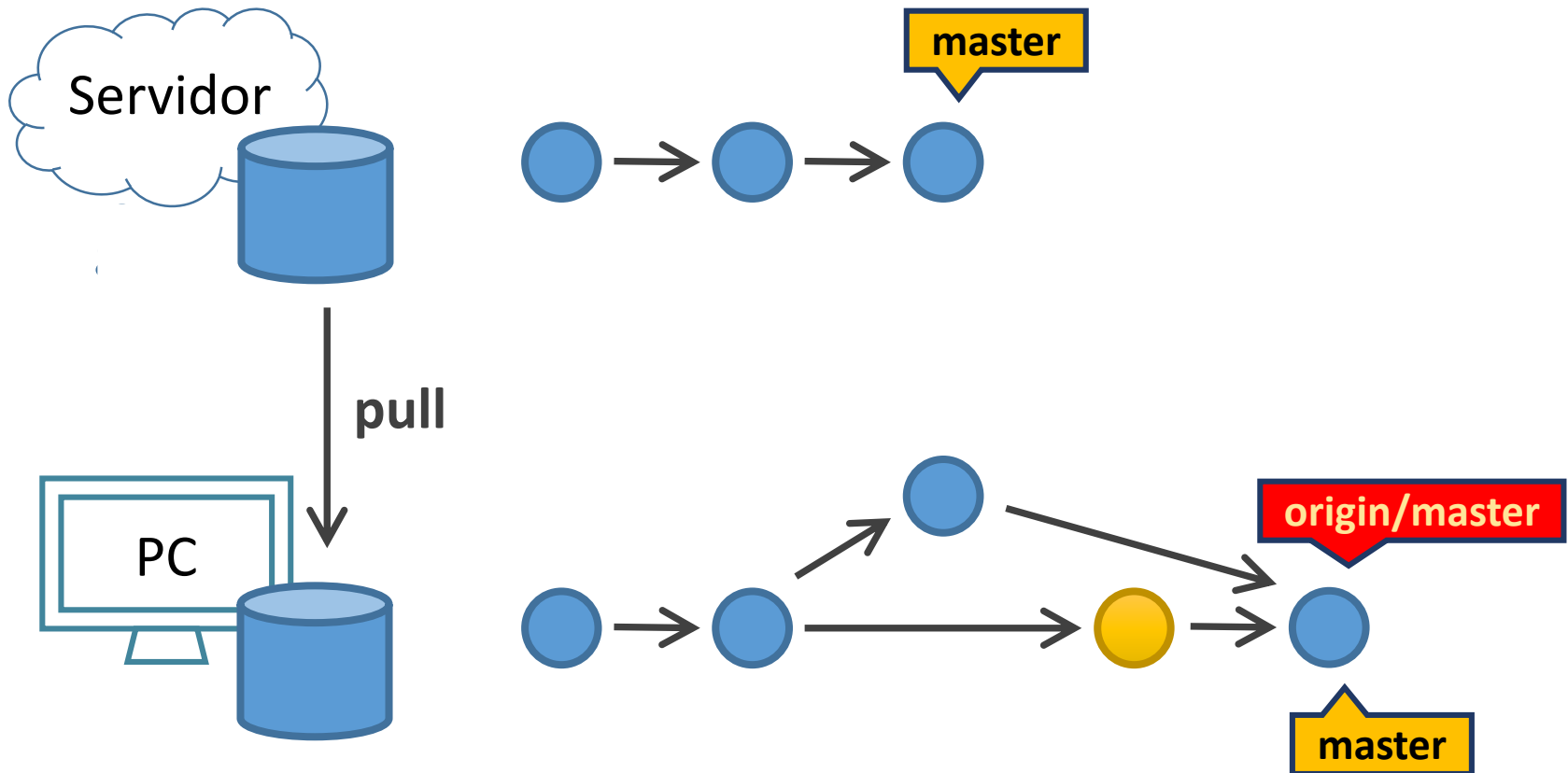
```
$ git push #Envia para o servidor
```

Solução 2: *pull + push*



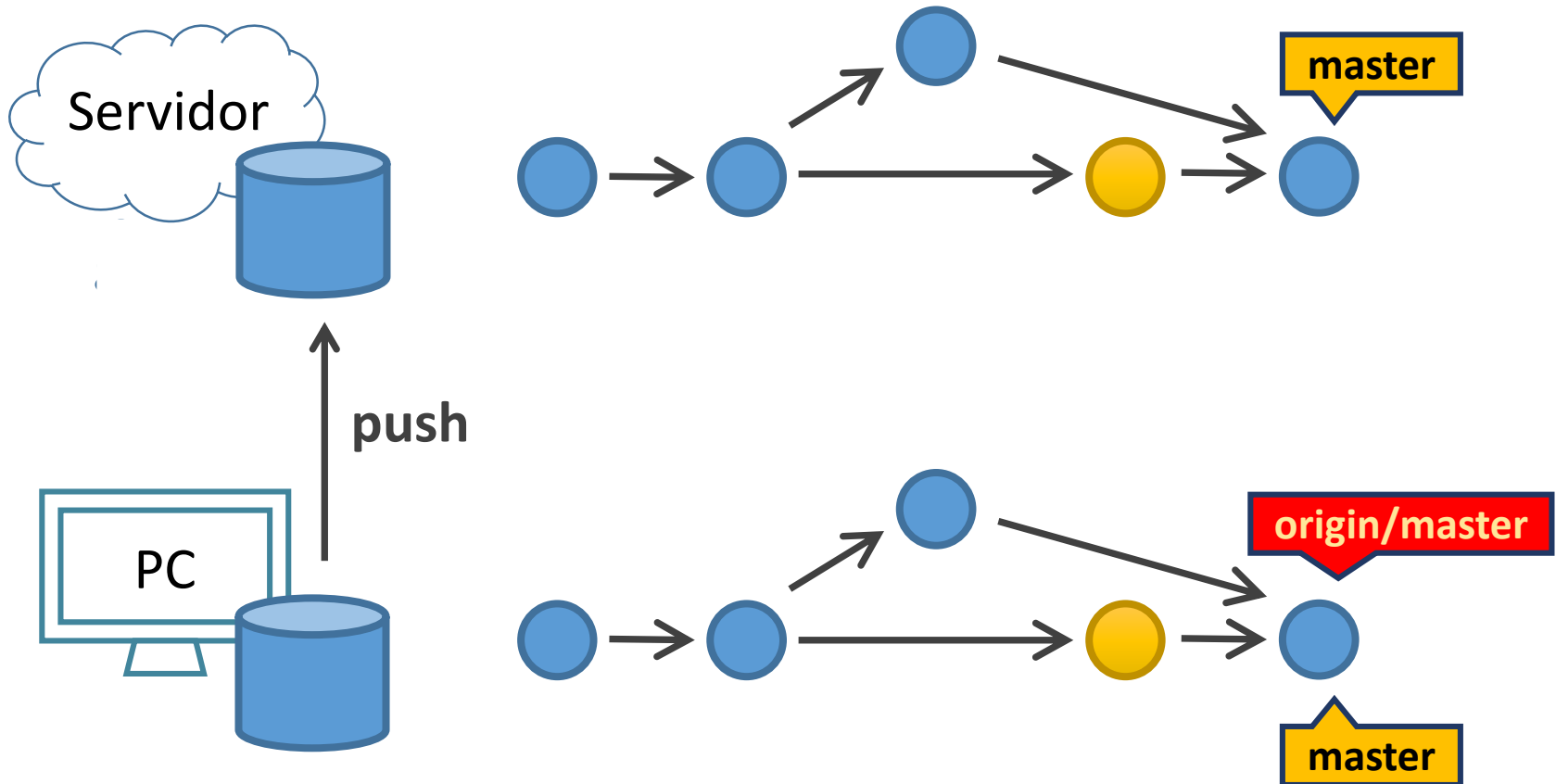


Solução 2: *pull* + *push*



```
$ git pull #Atualiza o repositório local
```

Solução 2: *pull + push*



```
$ git push #Envia para o servidor
```



Conflitos de Referência

Caso exista um *branch*, uma *tag* e/ou um repositório remoto com o mesmo nome <nome>, para evitar conflitos utilize a tabela abaixo.

Tipo	Referência
Branch	refs/heads/<nome>
Tag	refs/tags/<nome>
Repositório	refs/remotes/<nome>

```
$ git push origin refs/tags/issue13
```

Configurações Básicas



Configuração Inicial do Git

```
$ git config --global user.name <nome>
```

Atribui <nome> ao nome do usuário.

```
$ git config --global user.email <email>
```

Atribui <email> ao e-mail do usuário.

```
$ git config --global core.editor <editor>
```

Atribui <editor> como editor padrão. Ex.: notepad, emacs ...



Configurando o p4merge

```
$ git config --global merge.tool p4merge
```

Atribui p4merge como ferramenta de mesclagem.

```
$ git config --global mergetool.p4merge.cmd  
"p4merge.exe %BASE% %LOCAL% %REMOTE% %MERGED%"
```

Atribui o caminho e a forma de como executar o programa.

- Analogamente para *diff* e *difftool*.



Configuração do *Push*

```
$ git config --global push.default simple
```

Basicamente, envia apenas o *branch* atual, quando o branch não é especificado.

```
$ git config --global push.default nothing
```

Não envia nada, quando o branch não é especificado.

Outra opções: *current*, *upstream*, *matching*



Configuração do *Merge*

```
$ git config --global merge.ff false
```

Desativa o *fast-foward*, ou seja, cria sempre cria um *commit* na mesclagem.

*Cuidado: Quando o *fast-foward* está desativado, o comando *push (fetch +merge)* também irá sempre criar um *commit* ao atualizar um repositório.



Desconfigurando

```
$ git config --global --unset <key>
```

Desativa a chave <key>.

```
$ git config --global --unset merge.ff
```

```
$ git config --global --unset core.editor
```



Alias

```
$ git config --global alias.<abr> <cmd>
```

Substitui o comando <cmd> por <abr>.

```
$ git config --global alias.lol "log --graph  
--decorate --oneline"
```

```
$ git lol --all
```



Fim de Linhas em Arquivos

Windows: CRLF (*carriage-return and linefeed*)

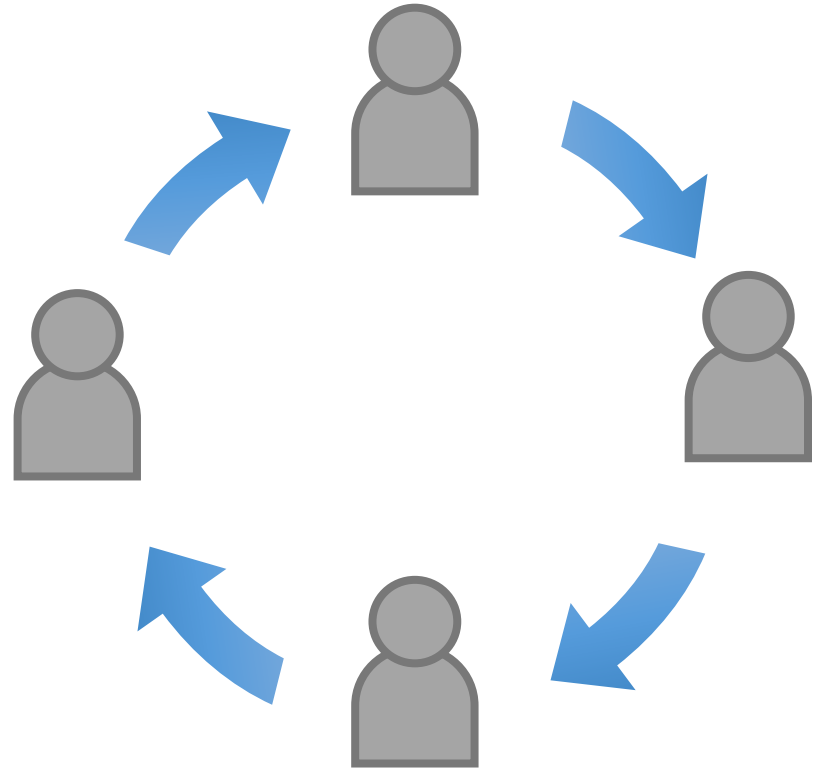
Linux: LF (*linefeed*)

```
$ git config --global core.autocrlf true
```

Converte CRLF para LF e de LF para CRLF automaticamente.
Configuração para Windows.

```
$ git config --global core.autocrlf input
```

Converte CRLF para LF durante um *checkout*. Configuração para Linux e Mac.



Fluxo de Trabalho



Fluxo de Trabalho



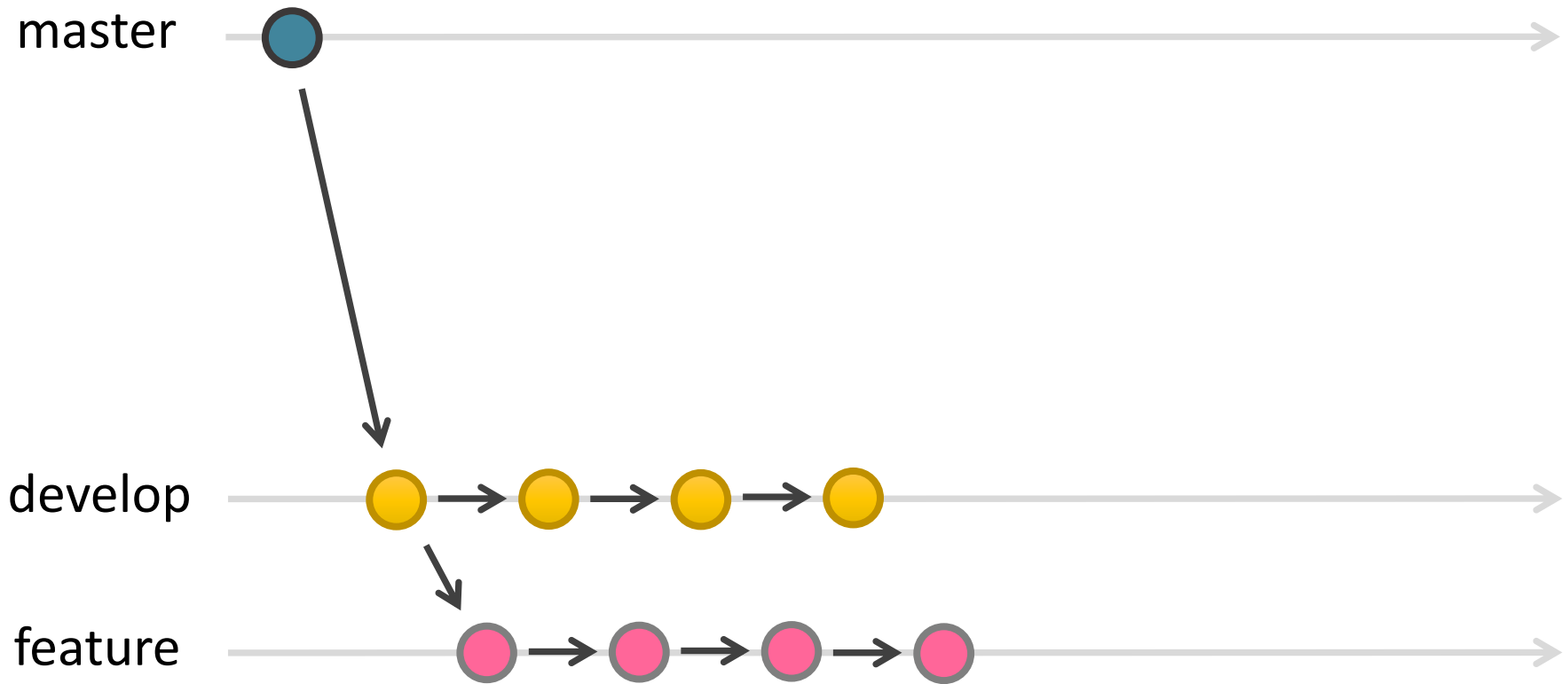


Fluxo de Trabalho



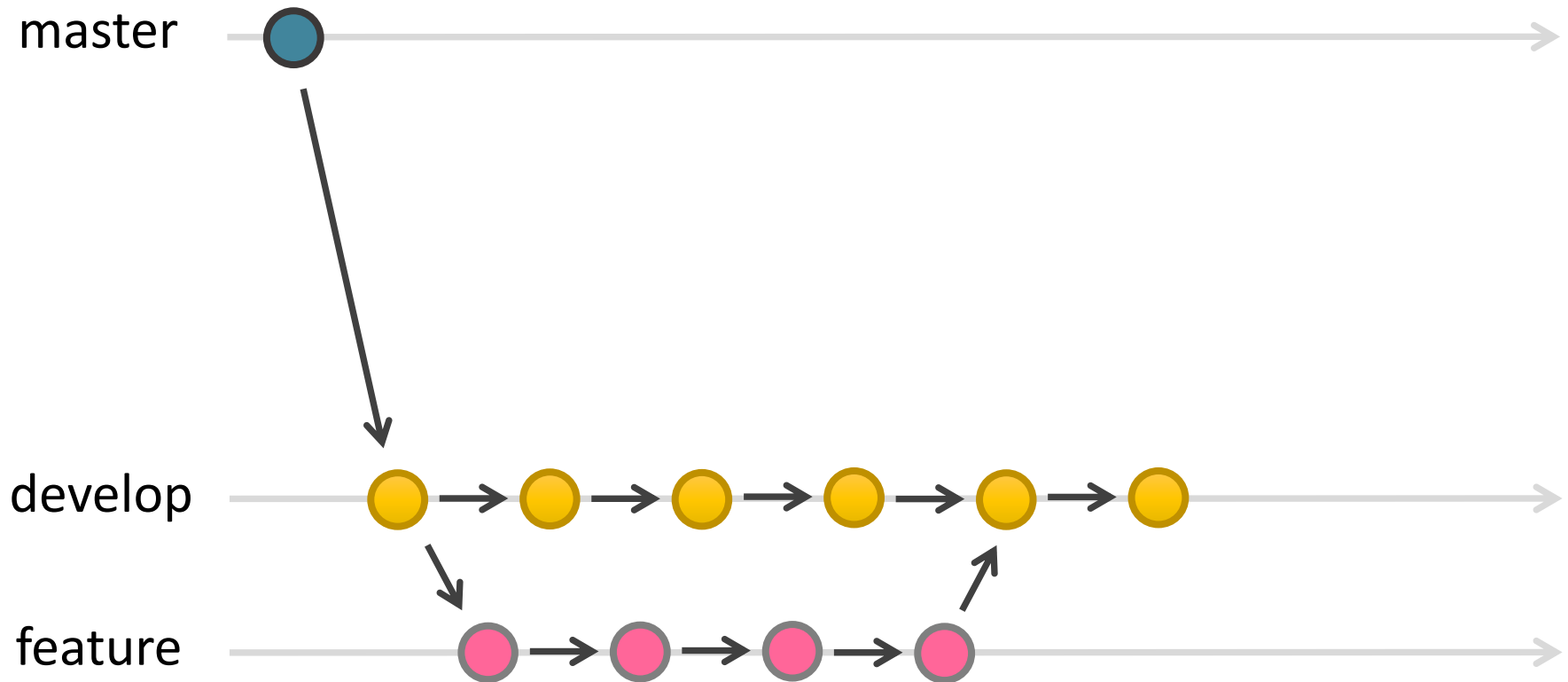


Fluxo de Trabalho



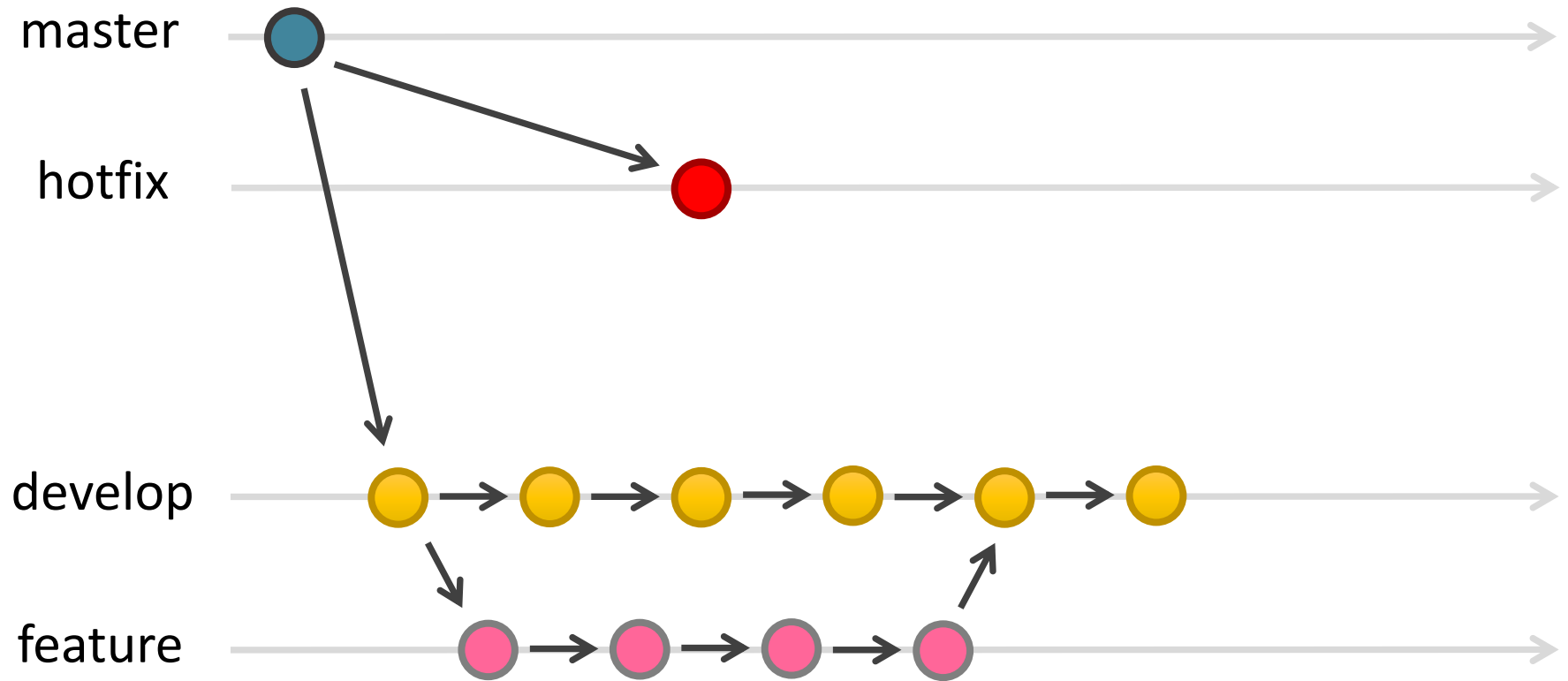


Fluxo de Trabalho



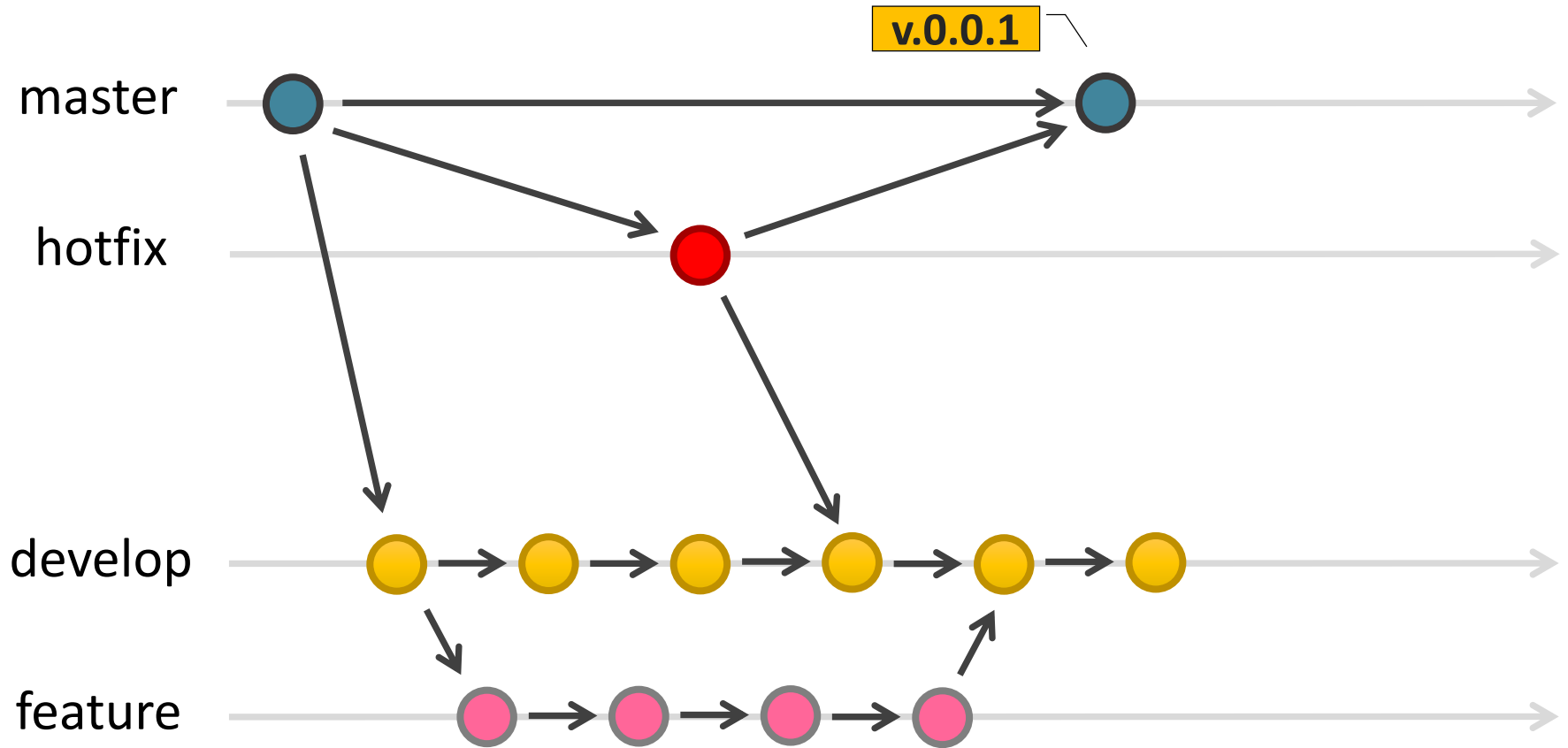


Fluxo de Trabalho



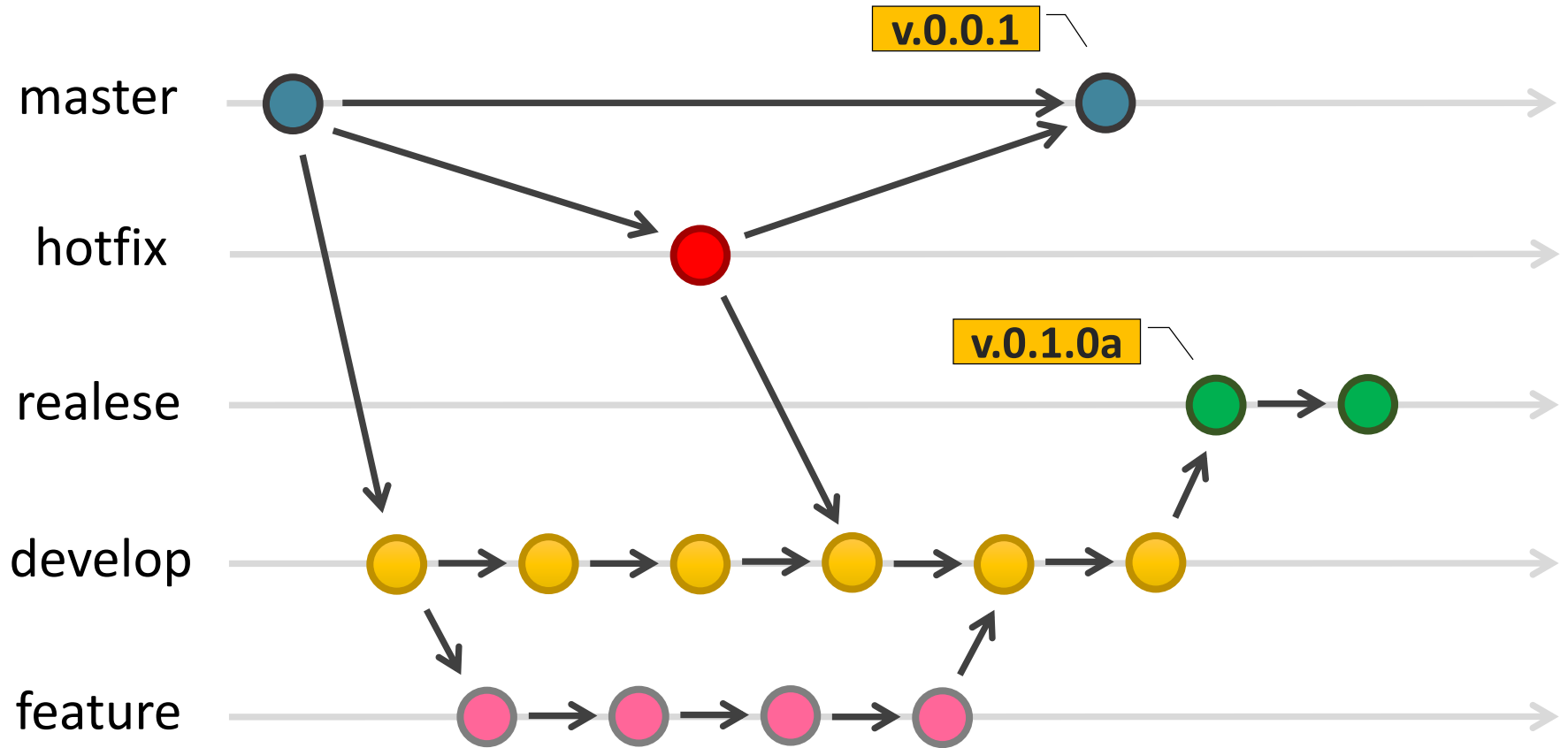


Fluxo de Trabalho



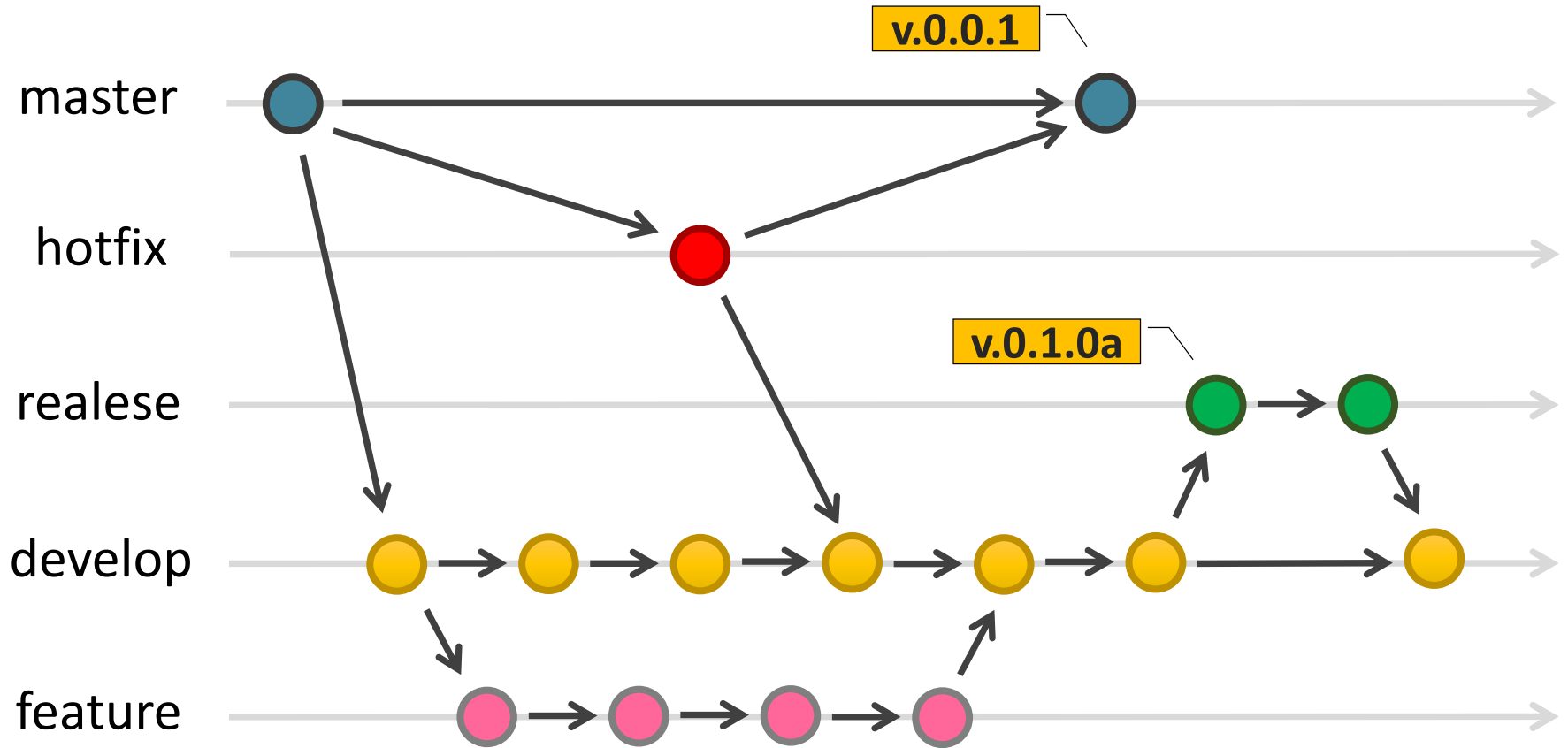


Fluxo de Trabalho



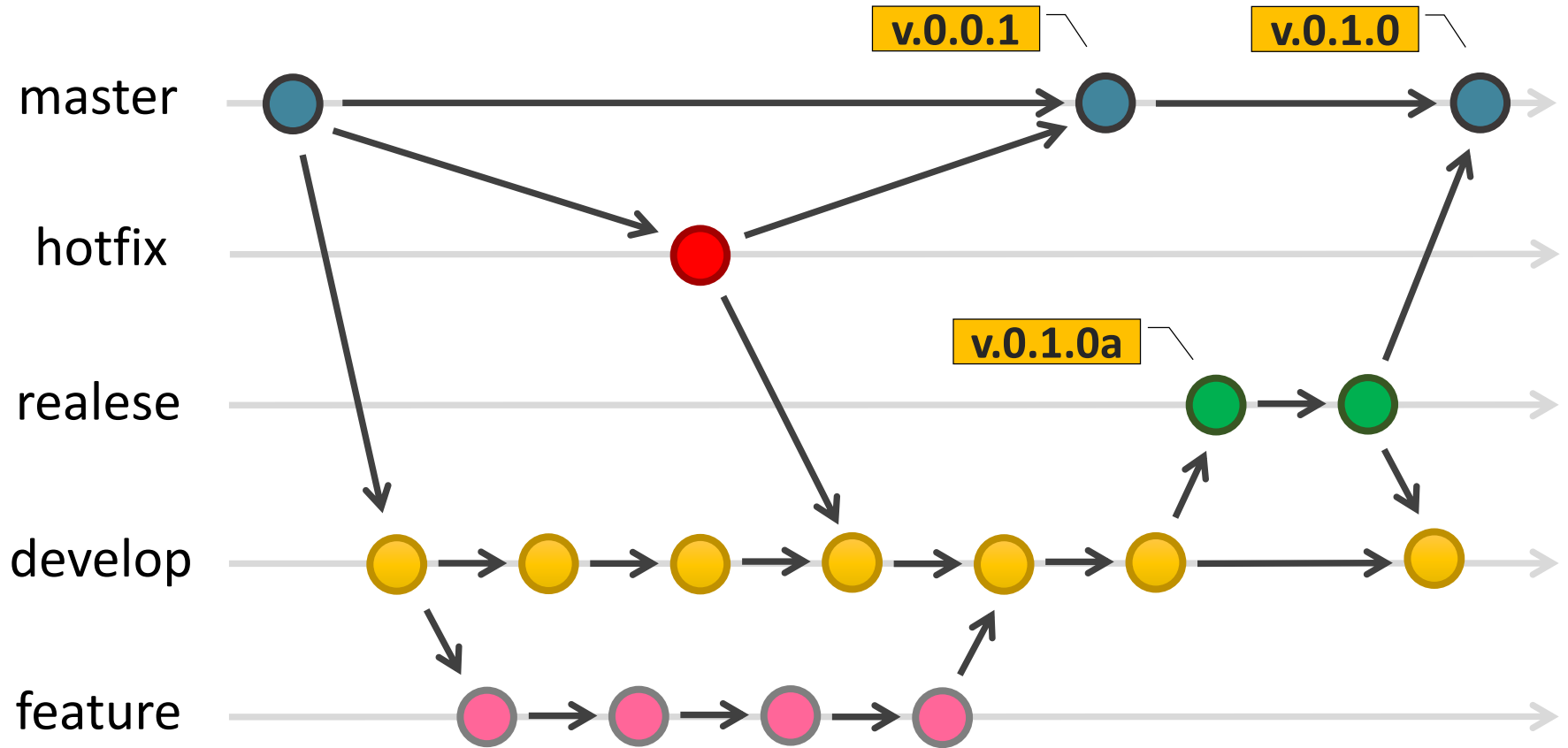


Fluxo de Trabalho





Fluxo de Trabalho





Fluxo de Trabalho

master	Versões estáveis.
hotfix	Correção de <i>bugs</i> da versão estável.
realese	Teste e correções de versões.
develop	Desenvolvimento.
feature	Implementação de funcionalidades.



Referências

- [Pro Git \(2009\), Scott Chacon](#)
- [Git Tutorials, Atlassian.com](#)
- [Git Tutorial, Lars Vogel, vogella.com](#)
- [A successful Git branch model, nvie.com](#)



Cursos

- <http://try.github.io/>
- <http://gitreal.codeschool.com/>

Dúvidas e Sugestões
bismarckjunior@outlook.com