Stash

ENTENDENDO CENÁRIO DE BLOQUEIO NO SWITCH

No repositório de João criar uma nova branch e alternar para a mesma

```
$ git switch -c forms
```

Gerar um novo commit

```
$ echo "function save(){...}" >> forms.js
$ git add forms.js
$ git commit -m "add save form"
```

Incluir uma nova alteração no arquivo forms.js

```
$ echo "function update(){...}" >> forms.js
```

Visualizar o estado do repositório

```
$ git status
```

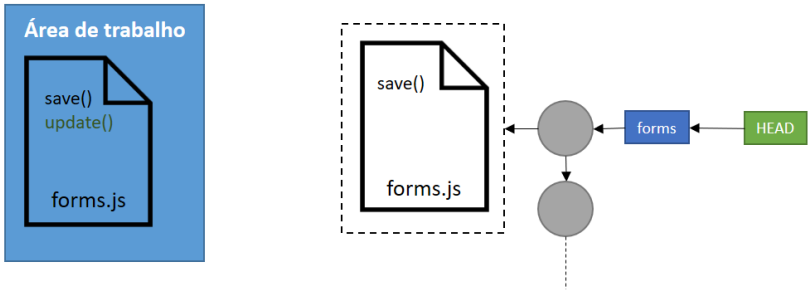
Tentar retornar para a branch master

```
$ git switch master
```

Nota: *Na branch master o arquivo forms.js não existe. Como existem modificações não versionadas, se o git remover este arquivo tais modificações serão perdidas. Nesta situação o Git solicita o commit dessas alterações ou um "stash"*

ENTENDENDO OPERAÇÕES DO COMANDO STASH

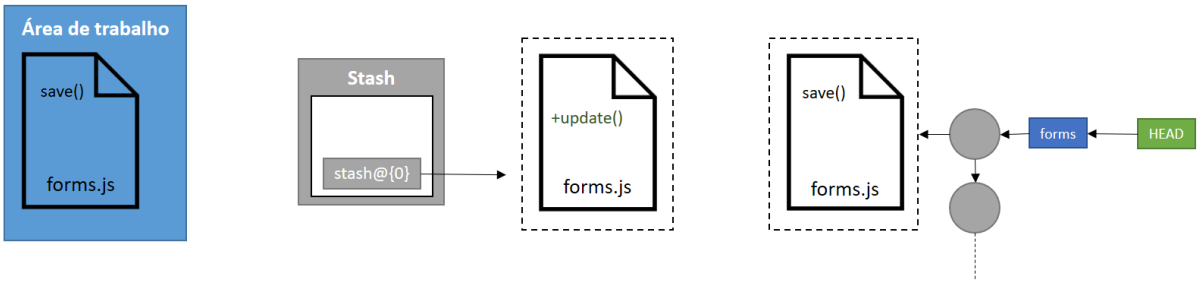
Estado inicial do repositório de João



Adicionar as alterações não versionadas na pilha

```
$ git stash
```

Nota: *o comando acima é equivalente a git stash push*



Visualizar o estado do repositório

```
$ git status
```

Visualizar as entradas na pilha

```
$ git stash list
```

Tentar retornar para a branch master

```
$ git switch master
```

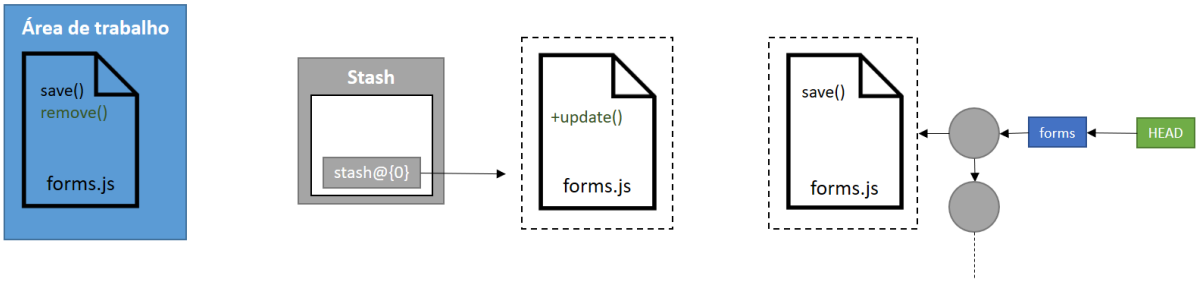
Nota: *pronto, salvando o trabalho em andamento na pilha foi possível alternar para uma outra branch!*

Retornar para a branch forms

```
$ git switch forms
```

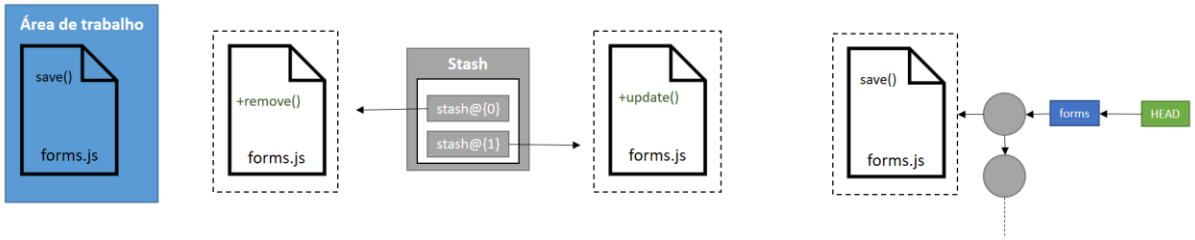
Incluir uma nova alteração no arquivo forms.js

```
$ echo "function remove(){...}" >> forms.js
```



Adicionar as novas alterações não versionadas na pilha

```
$ git stash
```



Visualizar as entradas na pilha

```
$ git stash list
```

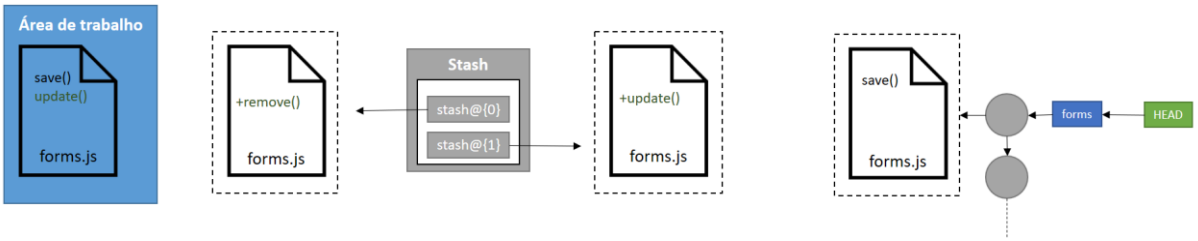
Visualizar o arquivo forms.js

```
$ cat forms.js
```

Aplicar as alterações da primeira entrada na pilha (de baixo para cima)

```
$ git stash apply stash@{1}
```

Nota: *aplica as alterações da entrada especificada na área de trabalho, mas não remove da pilha*



Visualizar as entradas na pilha

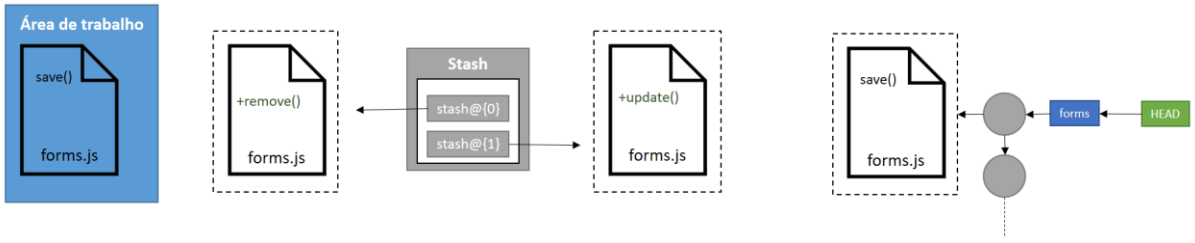
```
$ git stash list
```

Visualizar o arquivo forms.js

```
$ cat forms.js
```

Descartar as alterações do arquivo forms.js

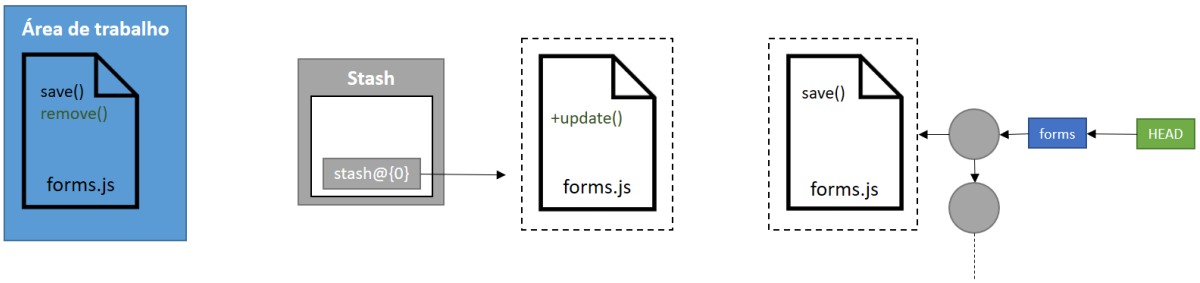
```
$ git restore forms.js
```



Aplicar a alteração salva na pilha novamente na área de trabalho

```
$ git stash pop
```

Nota: *remove a entrada no topo da pilha e aplica as alterações na área de trabalho*



Visualizar as entradas na pilha

```
$ git stash list
```

Visualizar o arquivo forms.js

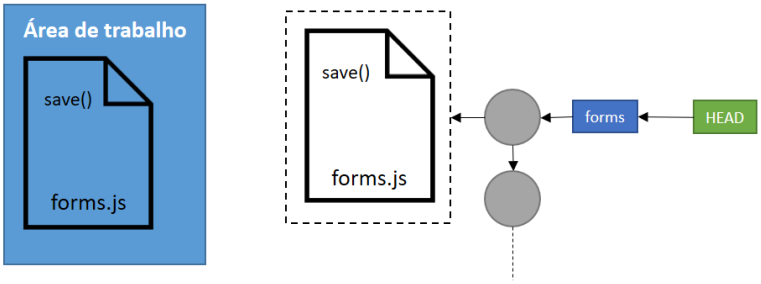
```
$ cat forms.js
```

Limpar as entradas na pilha

```
$ git stash clear
```

Descartar as alterações do arquivo forms.js

```
$ git restore forms.js
```



Stash - internals

ENTENDENDO O FUNCIONAMENTO INTERNO DO GIT STASH

No repositório de João verificar os arquivo forms.js e courses.js

```
$ cat forms.js
$ cat courses.js
```

Incluir uma nova função no arquivo courses.js

```
$ echo "function subscription(){...}" >> courses.js
```

Nota: *repare que este arquivo existe no repositório*

Adicionar a alteração acima na área de preparo

```
$ git add courses.js
```

Incluir uma nova alteração no arquivo forms.js

```
$ echo "function update(){...}" >> forms.js
```

Criar um novo arquivo fields.js

```
$ echo "function save(){...}" >> fields.js
```

Verificar o estado do repositório

```
$ git status
```

Nota:

- *A alteração do arquivo courses.js consta na área de preparo*
- *A alteração do arquivo forms.js consta na área de trabalho*

- *O novo arquivo fields.js ainda não é rastreado pelo Git (arquivo no estado untracked)*

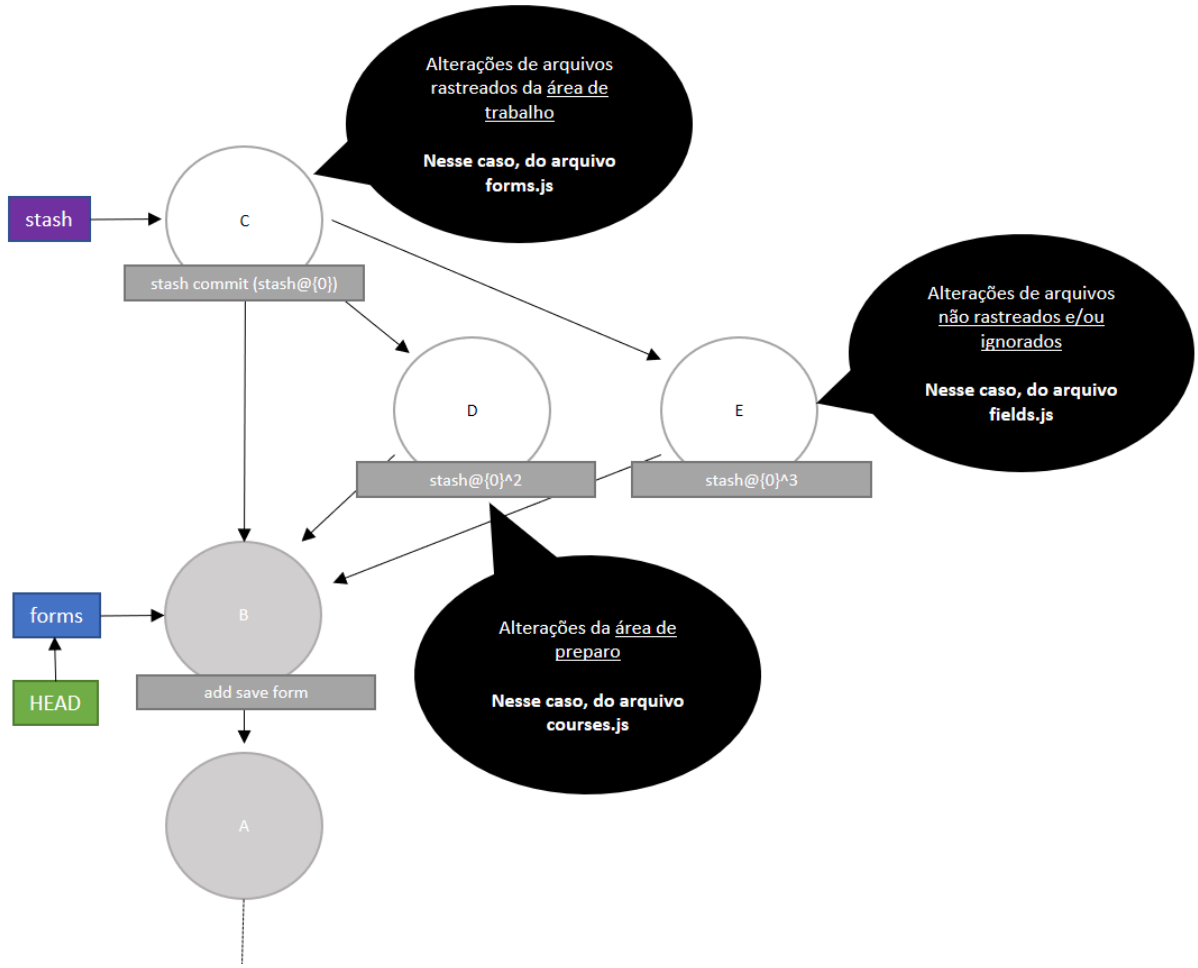
Adicionar as alterações acima e o novo arquivo na stash

```
$ git stash -u
```

Nota: *a opção -u indica para considerar arquivos no estado untracked*

Visualizar as entradas na pilha

```
$ git stash list
```



Visualizar a primeira entrada da pilha

```
$ git show stash@{0}
```

Nota: *a referência stash, localizada em .git\refs\stash é uma referência pro objeto de commit que é o topo da pilha (stash@{0})*

Imprimir o conteúdo do objeto de commit acima

```
$ git cat-file -p <HASH_COMMIT>
```

Visualizar o primeiro pai

```
$ git show <HASH_PRIMEIRO_PAI>
```

Nota: *o primeiro pai é o commit atual no momento do stash, ou seja "add save form"*

Visualizar o segundo pai

```
$ git show <HASH_SEGUNDO_PAI>
```

Nota: *o segundo pai contém o conjunto de alterações presentes na área de preparo no momento do stash, ou seja é exibido a alteração no arquivo courses.js*

Visualizar o terceiro pai

```
$ git show <HASH_TERCEIRO_PAI>
```

Nota: *o terceiro pai contém o conjunto de alterações dos arquivos não rastreados e os arquivos ignorados no momento do stash, ou seja é exibido a inclusão do novo arquivo fields.js*

Limpar as entradas na pilha

```
$ git stash clear
```

Stash - conflitos

RESOLVENDO CONFLITOS NO GIT STASH

No repositório de João verificar o histórico da branch atual

```
$ git hist
```

Avaliar o conteúdo forms.js

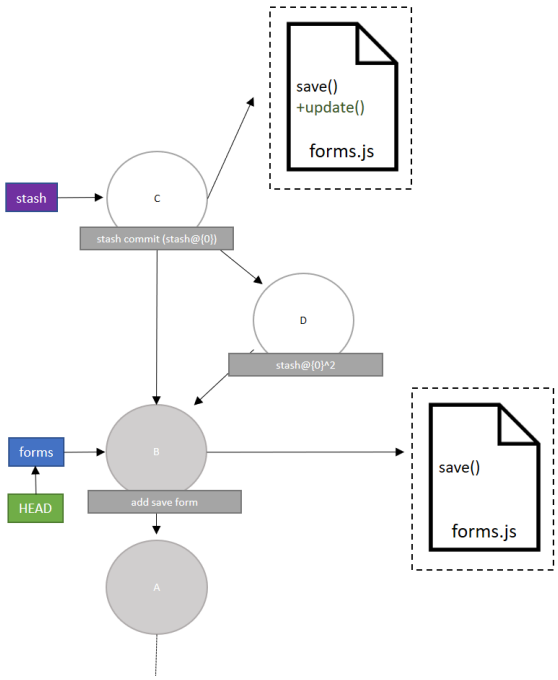
```
$ cat forms.js
```

Incluir uma linha 2 no arquivo forms.js

```
$ echo "function update(){...}" >> forms.js
```

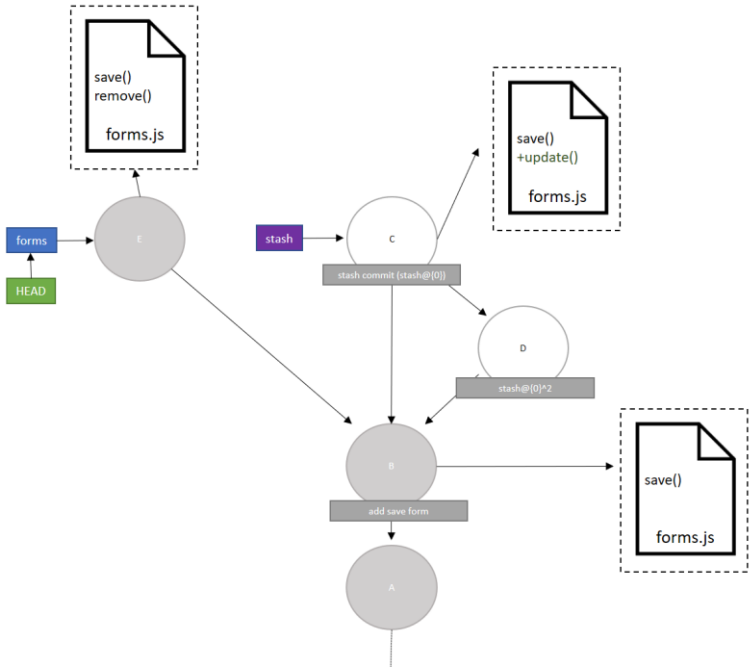
Adicionar a alteração acima na pilha

```
$ git stash
```



Gerar um novo commit

```
$ echo "function remove(){...}" >> forms.js
$ git add forms.js
$ git commit -m "add remove form"
```



Desempilhar a alteração salva na pilha

```
$ git stash pop
```

Nota: *a alteração da stash envolve adicionar a linha 2, porém a linha dois foi adicionada no commit acima. Ou seja, um conflito é gerado!*

Resolver o conflito aceitando ambas as alterações
Realizar o commit

```
$ git commit -m "add update form"
```

Visualizar o último commit

```
$ git show
```

Cherry-pick

ENTENDENDO O COMANDO CHERRY-PICK

No repositório de João gerar dois novos commit

```
$ echo "function formatNumber(){...}" >> utils.js
$ git add utils.js
$ git commit –m “add format number"
$ echo "function save(){...}" >> fields.js
$ git add fields.js
$ git commit –m “add save fields"
```

Retornar para a branch master

```
$ git switch master
```

Remover a branch forms

```
$ git branch -D forms
```

Nota: *a opção -D deve ser maiúscula para forçar a remoção. Isso porque nesse caso a branch possui commits que ainda não foram incorporados na branch origem (master)*

Criar uma nova branch “projects” e alternar para a mesma

```
$ git switch -c projects
```

Buscar a hash do commit “add format number" no reflog

```
$ git reflog
```

Copiar a hash do commit “add format number"

Aplicar o commit “add format number" na branch atual

```
$ git cherry-pick <HASH_COPIADA>
```

Verificar o histórico

```
$ git hist
```

Verificar o existência do arquivo utils.js

```
$ ls
```

Diff

ENTENDENDO O COMANDO DIFF

No repositório de João gerar um novo commit

```
$ echo "function save(){...}" >> projects.js
$ git add projects.js
$ git commit –m “add save projects"
```

Verificar as alterações não versionadas

```
$ git diff
```

Nota: *o comando acima é equivalente a git diff HEAD. Por isso, está sendo utilizado para mostrar as diferenças entre a área de trabalho e o último commit*

Inclua uma alteração no arquivo projects.js

```
$ echo "function update(){...}" >> projects.js
```

Verificar as alterações não versionadas

```
$ git diff
```

Inclua uma alteração no arquivo utils.js

```
$ echo "function formatText(){...}" >> utils.js
```

Verificar as alterações não versionadas

```
$ git diff
```

Nota: *todas as alterações de todos os arquivos serão exibidas*

Verificar as alterações não versionadas do arquivo utils.js

```
$ git diff utils.js
```

Nota: *todas as alterações do arquivo utils.js serão exibidas*

Adicionar todas as alterações para a área de preparo

```
$ git add .
```

Verificar as alterações não versionadas

```
$ git diff
```

Nota: *para comparar com a área de preparo deve-se utilizar git diff –staged*

Realizar um commit

```
$ git commit -m "add update project"
```

Verificar as alterações não versionadas

```
$ git diff
```

Verificar as diferenças entre o penúltimo commit e o último commit

```
$ git diff head~..head
```

Nota: *pode ser usado com qualquer referência ou até commit específico*

Verificar as diferenças entre o antepenúltimo commit e o último commit

```
$ git diff head~2..head
```

Grep e pesquisa no log

ENTENDENDO O COMANDO GREP

No repositório de João, realizar uma busca pelo termo "save()"

```
$ git grep "save()"
```

Realizar uma busca pelo termo "save()", solicitando a linha de ocorrência de cada resultado

```
$ git grep -n "save()"
```

Nota: *a opção -n é equivalente a --line-number*

Realizar uma busca pelo termo "save()", solicitando apenas quais arquivos contém tal termo e quantas correspondências existem em cada arquivo

```
$ git grep -c "save()"
```

Nota: *a opção -c é equivalente a --count*

Tentar realizar uma busca por "Save()" com S maiúsculo

```
$ git grep "Save()"
```

Nota: *por padrão, maiúsculo é diferente de minúsculo*

Realizar uma busca por "Save()", ignorando a diferença entre maiúsculas e minúsculas

```
$ git grep -i "Save()"
```

Nota: *a opção -i ignora a diferenciação entre maiúsculas e minúsculas*

PESQUISANDO NO LOG

Exibir os commits no histórico onde a mensagem de commit contém a palavra "crud"

```
$ git hist --grep="crud"
```

Nota: *lembrar que git hist é um alias para:*
log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

Logo, git hist --grep="crud" é equivalente a:
log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short ***--grep="crud****"*

Exibir os commits onde "format" foi adicionado ou removido no conteúdo de um arquivo

```
$ git hist -G"format"
```

Avaliar os commits dos resultados acima

```
$ git show <hash commit 1>
$ git show <hash commit 2>
```

Blame

ENTENDENDO O COMANDO BLAME

Avaliar quem e quando foi alterado cada linha do arquivo login.js

```
$ git blame login.js
```

Nota: *esse comando mostra qual commit e autor modificou pela última vez cada linha de um arquivo*

Avaliar quem e quando foi alterado a linha 3 do arquivo login.js

```
$ git blame login.js -L 3,3
```

Nota: *a opção -L permite especificar um intervalo de linhas no arquivo que você deseja examinar*

Bisect

GERANDO COMMITS

No repositório de João gerar um novo commit

```
$ echo "function remove(){...}" >> projects.js
$ git add projects.js
$ git commit –m “add remove projects"
```

Editar a primeira linha do arquivo projects.js para "function save(){...bug...}"
Gerar um novo commit

```
$ git add projects.js
$ git commit -m "update query on save project"
```

Gerar um novo commit

```
$ echo "function addCache(){...}" >> projects.js
$ git add projects.js
$ git commit –m “add cache on projects"
```

Gerar um novo commit

```
$ echo "function log(){...}" >> projects.js
$ git add projects.js
$ git commit –m “add log on projects"
```

Editar a primeira linha do arquivo projects.js para "function save(){...bug...addCache()}"
Gerar um novo commit

```
$ git add projects.js
$ git commit -m "add cache on save project"
```

Verificar o histórico

```
$ git hist
```

BUSCANDO BUG ATRAVÉS DO BISECT

Iniciar a busca para descobrir qual commit originou o bug

```
$ git bisect start
```

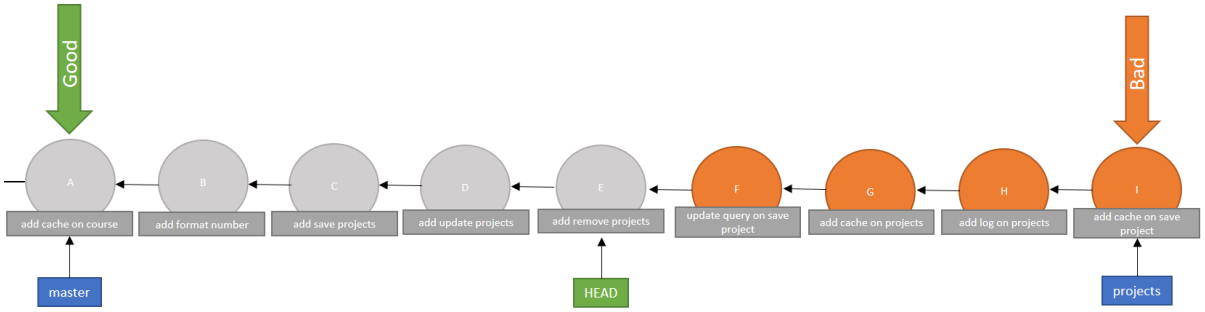
Marcar o commit atual como um commit ruim

```
$ git bisect bad
```

Marcar o commit “add format number” como um commit bom

```
$ git bisect good master
```

Pronto, o Git iniciou a busca. Agora sua área de trabalho foi modificada para uma versão no meio entre a ruim e a boa



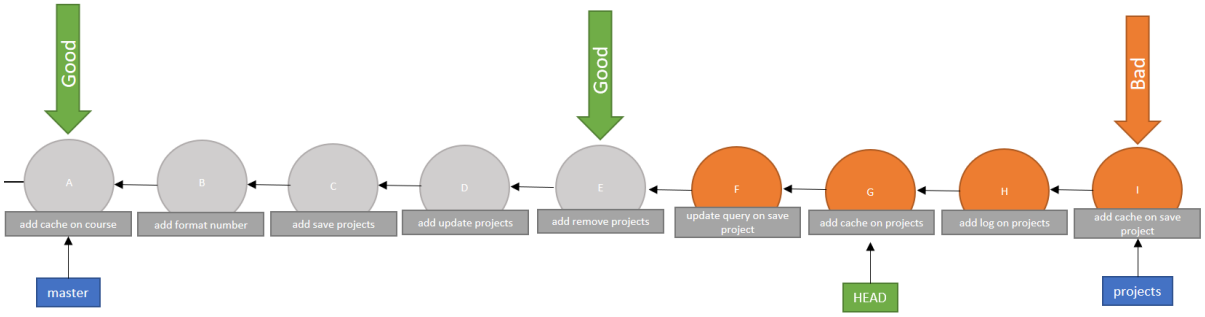
Verificar o arquivo projects.js

```
$ cat projects.js
```

Nota: *o arquivo ainda não possui nenhum bug*

Marcar o commit atual como bom

```
$ git bisect good
```



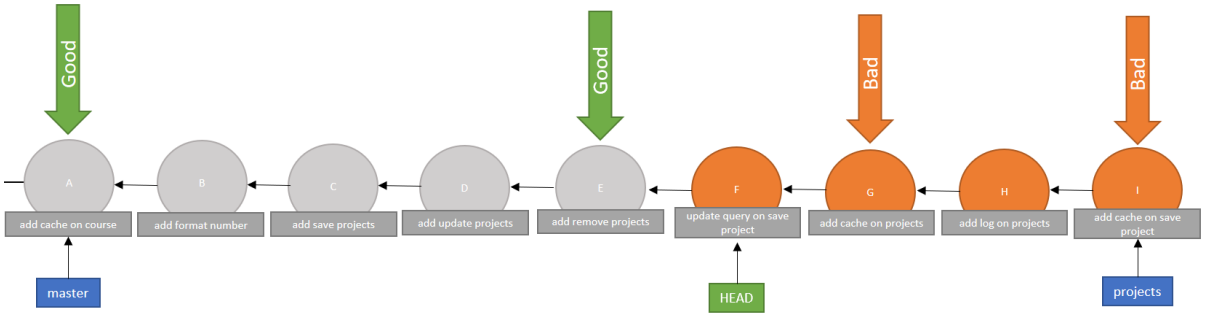
Verificar o arquivo projects.js

```
$ cat projects.js
```

Nota: *o arquivo contém o bug*

Marcar o commit atual como ruim

```
$ git bisect bad
```



Verificar o arquivo projects.js

```
$ cat projects.js
```

Nota: *o arquivo contém o bug*

Marcar o commit atual como ruim

```
$ git bisect bad
```

Nota: *a busca pela origem do bug é concluída e todas as informações desse commit são exibidas, como autor e data*

Após encontrado o commit é necessário limpar o estado do bisect e retornar ao HEAD original

```
$ git bisect reset
```

Remover o bug do arquivo projects.js

Gerar um novo commit

```
$ git add projects.js
$ git commit –m “add fix on save user”
```

SINCRONIZANDO REPOSITÓRIOS

Publicar tal branch para o repositório remoto

```
$ git push -u origin projects
```

No Bitbucket:

- Criar um pull request da branch projects para a branch master
- Aprovar pull request e realizar o merge com a estratégia fast forward

No repositório de João e Maria:

```
$ git switch master
$ git pull
```

Está gostando deste curso?

Compartilhe sua experiência nas redes sociais com a tag ***#rsantanatech*** *para que eu possa interagir com a sua postagem.*

Acompanhe nas redes sociais
e fique por dentro de todos os conteúdos.