CTEDS

Capacitação Tecnológica em Engenharia e Desenvolvimento de Software

Controle de Versão

(com Git)

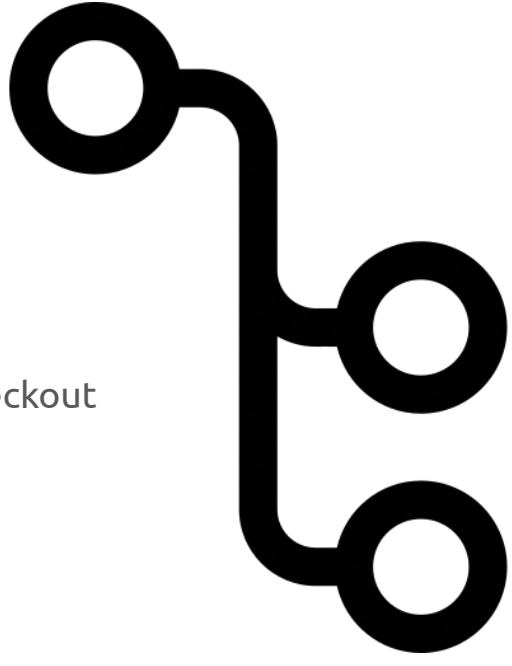
Professor: Bruno Albertini [PCS/EPUSP] *balbertini@usp.br*

31 de Agosto de 2022

Resumo

O que é o controle de versão?

- Histórico
- Funcionamento interno (*snapshots*)
- Boas práticas em VCS
- Git básico:
  - Conceitos (Working Tree, Staging Area, .git)
  - Locais: init, config, add, rm, mv, commit, log, reset/restore, checkout
  - Remotos: clone, push, (fetch, pull)
  - .gitignore



📝 Exercício 8: *fetch* e *pull*

Faça o seguinte exercício (no seu repositório):

1. Através da interface web, altere um arquivo texto (não crie)
2. Na sua WT, faça um *fetch*
3. Verifique o conteúdo do arquivo
4. Na sua WT, faça um *pull*
5. Verifique o conteúdo do arquivo

Qual a diferença entre o *fetch* e o *pull*? Lembre-se de colocar a resposta no arquivo RESPOSTAS.md.

Nota: se for a primeira vez que está fazendo um *fetch*, pode ser necessário configurar o comportamento dos *branches*: git config --global pull.rebase "false"

Analisando os *branches*

35. Na sua WT, veja para onde ela está apontando com:

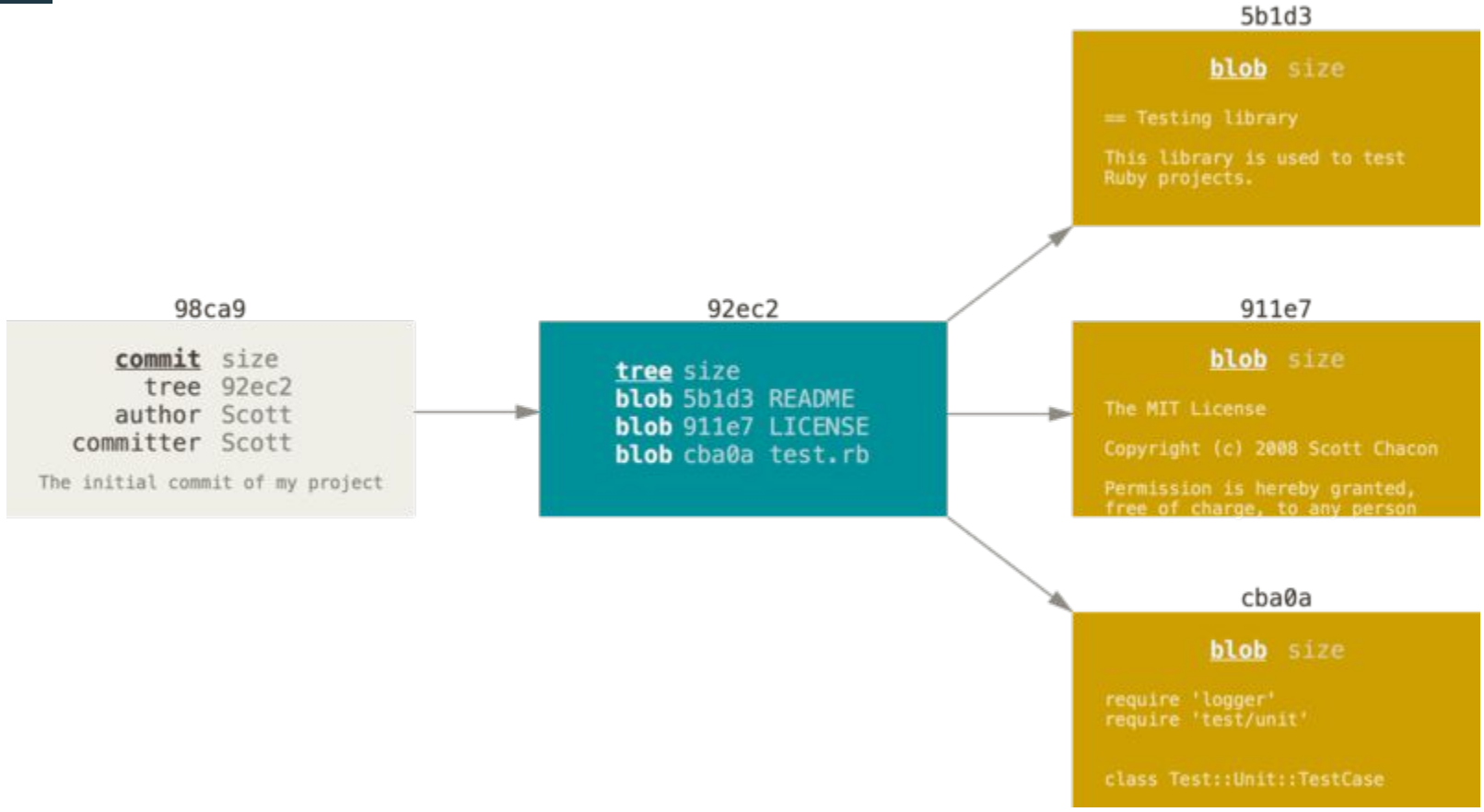
    git remote show origin

36. Veja seus *branches*
37. Para testar depois (opcional):

    git remote rename <branchantigo> <branchnovo>

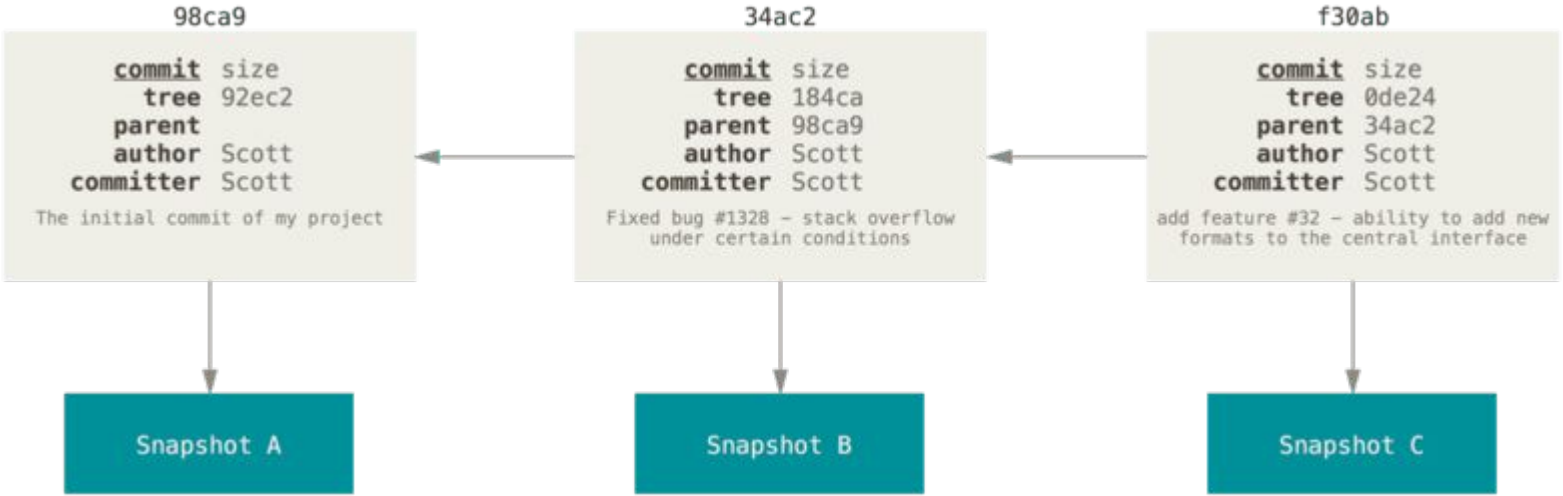
    git remote remove <branch>

Branch (conceitos)



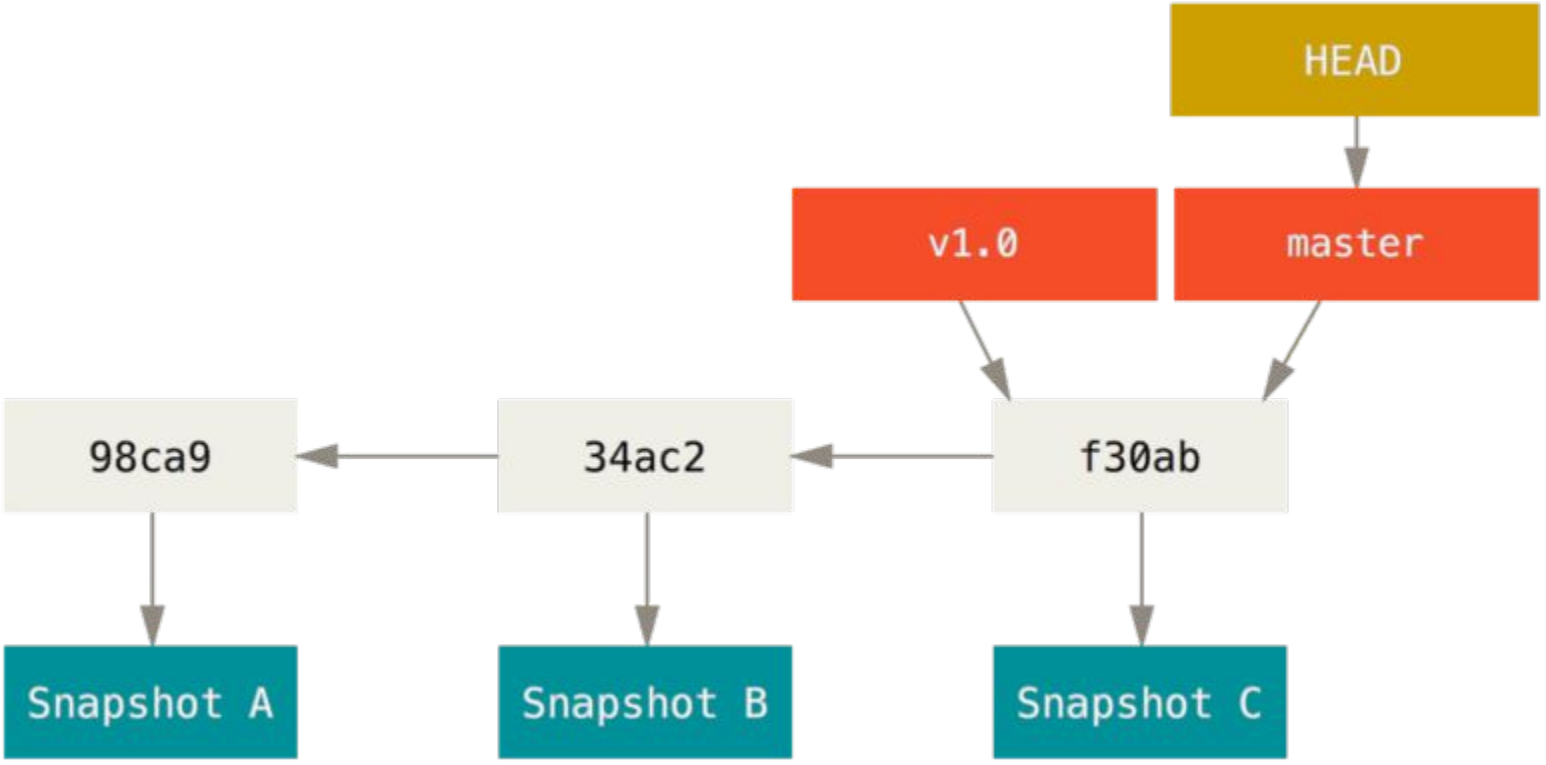
Fonte: [2]

Branch (conceitos)



Fonte: [2]

Branch (conceitos)

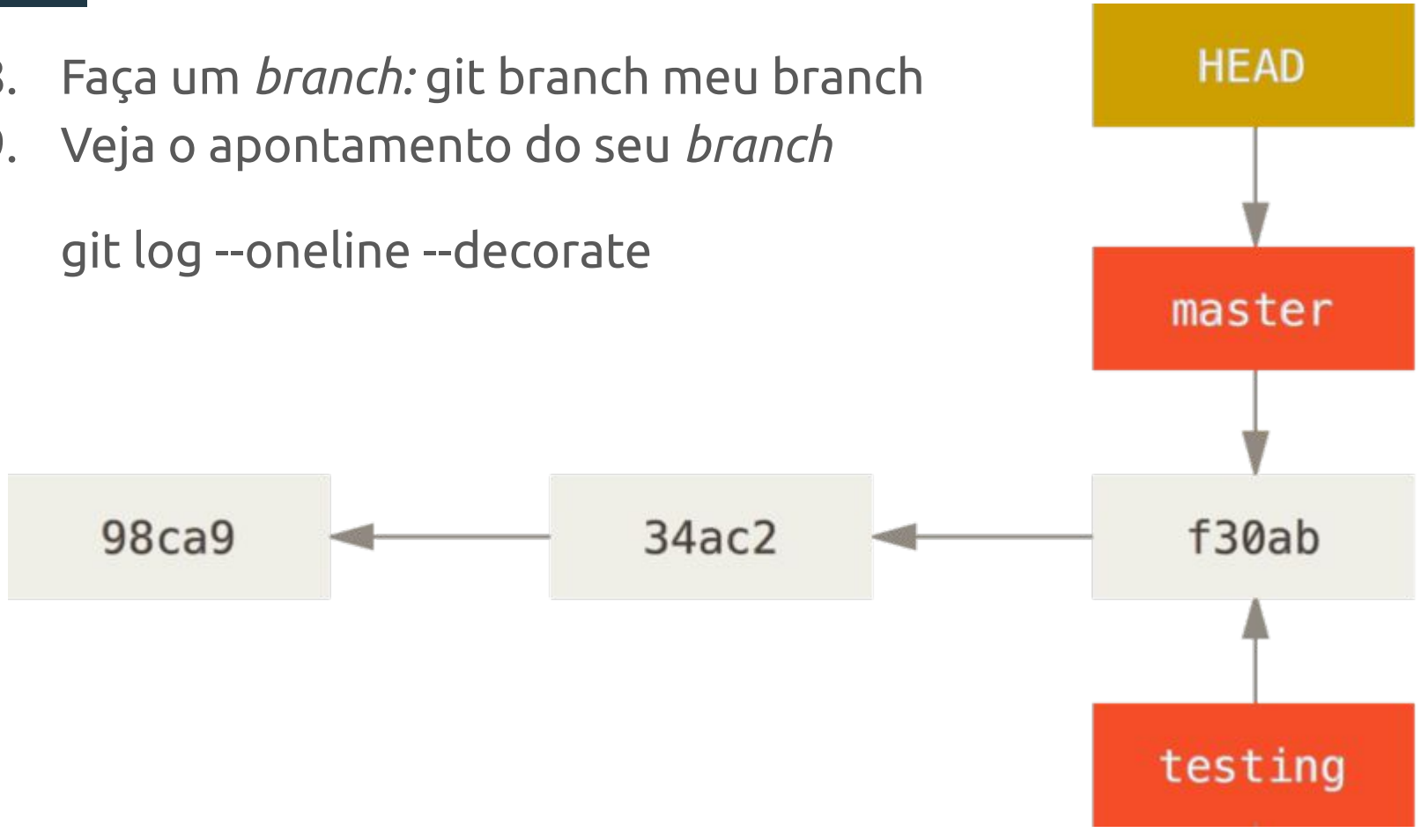


Fonte: [2]

Adicionando arquivos

38. Faça um *branch:* git branch meu branch
39. Veja o apontamento do seu *branch*

    git log --oneline --decorate



Fonte: [2]

📝 Exercício 9: mude para seu *branch*

Faça o seguinte exercício (no seu repositório):

1. Crie um *branch* se não criou no exemplo anterior
2. git checkout <branch>
3. Repita o git log --oneline --decorate
4. Adicione um arquivo texto (add e commit)
5. git checkout main (ou master dependendo de como inicializou)
6. Repita o git log --oneline --decorate

O que faz o *checkout*? Para onde foi o arquivo que criou? Lembre-se de colocar a resposta no arquivo RESPOSTAS.md.

merge

40. Crie um *branch* novo e mude para ele (git checkout -b mb2)
41. Modifique um arquivo (*commit*) (git commit -a -m 'Msg3')
42. Volte para o branch main (git switch main), verifique o arquivo
43. Faça o *merge*: (git merge mb2)
44. Apague o *branch* (git branch -d mb2)

Sugestão: entre os comandos, observe a lista ligada com:

git log --oneline --decorate --graph --all

📝 Exercício 10: *merge*

Faça o seguinte exercício (no seu repositório):

1. Garanta que está no main
2. Faça merge do *branch* do exercício 10 no main
3. Veja o log *git log --oneline --decorate --graph --all)
4. Apague o branch do exercício 10
5. Veja o log novamente

O que houve com o arquivo que foi alterado no *branch* do exercício 10?
Lembre-se de colocar a resposta no arquivo RESPOSTAS.md.

Conflitos

45. Crie e troque para um branch (git checkout -b mb3)
46. Modifique um arquivo texto (git commit -a -m 'Teste conflito')
47. Volte para o branch main (git switch main), verifique o arquivo
48. Modifique um arquivo texto NO MESMO LUGAR
49. Tente o merge: git merge mb3
    - Veja as diferenças: git diff mb3
50. Veja o estado: git status

Sugestão: entre os comandos, observe a lista ligada com:

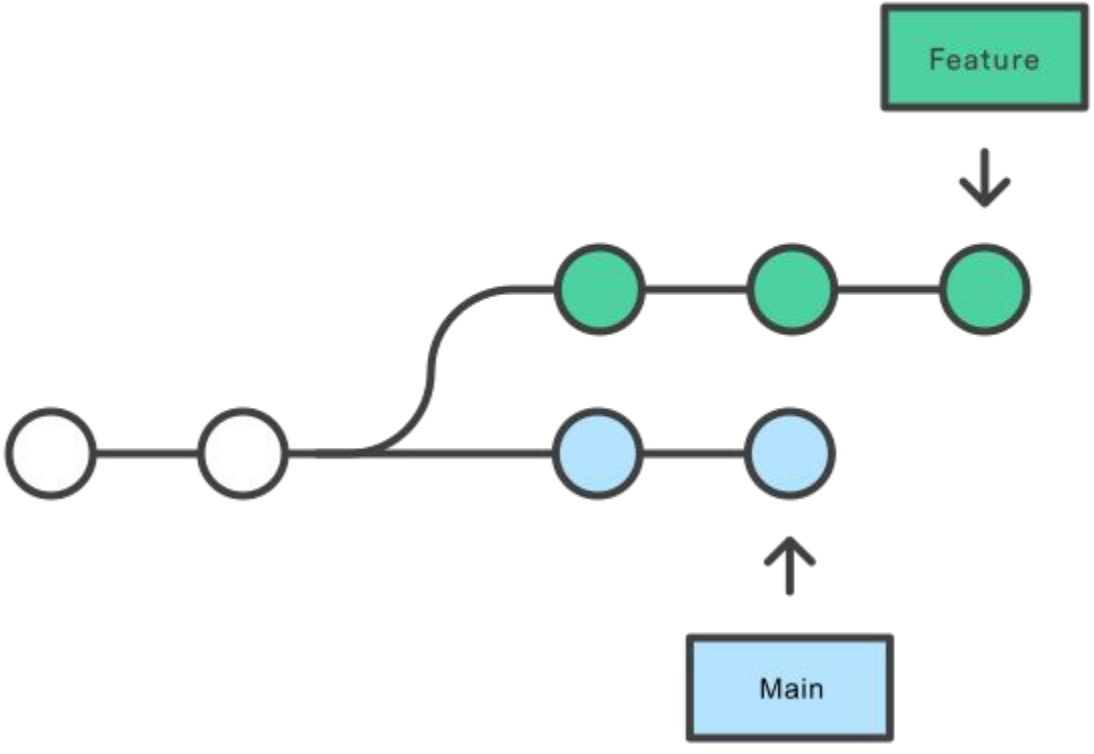
git log --oneline --decorate --graph --all

rebase

55. Apague todos os *branches* exceto o main
56. Crie um *branch*, edite um arquivo
    - git checkout -b mb4
    - git commit -a -m 'Teste rebase'
57. Volte para o main
58. Adicione um arquivo (add, commit)
59. Faça um rebase (git rebase main)
60. Volte para o main e faça rebase (git switch main, git rebase mb4)

branch vs. *rebase*

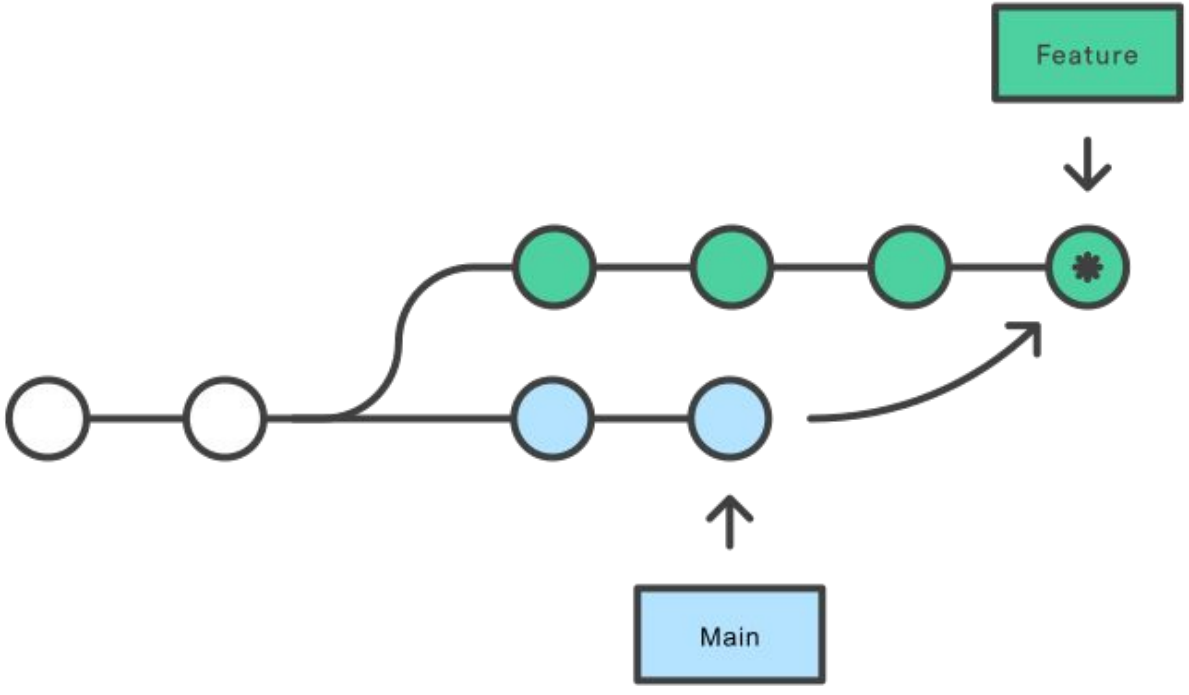
A forked commit history



Fonte: [1]

branch vs. *rebase*

Merging main into the feature branch

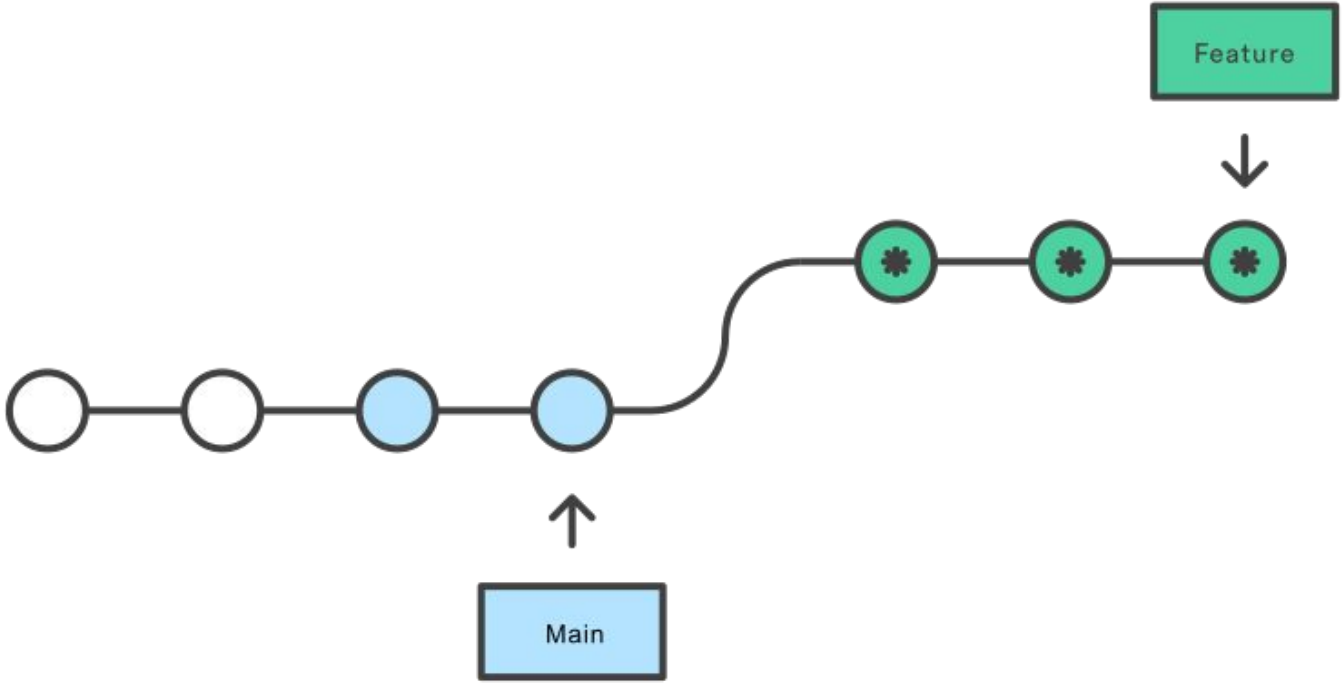


Merge Commit

Fonte: [1]

branch vs. rebase

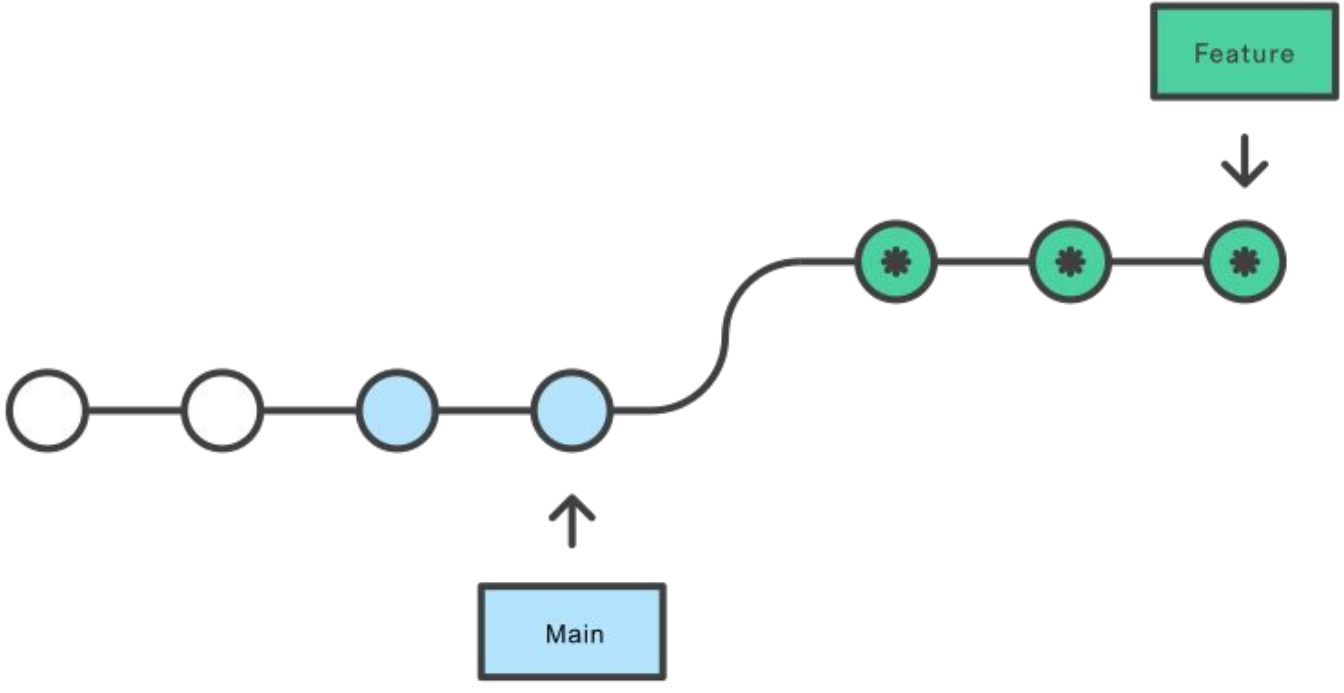
Rebasing the feature branch onto main



Fonte: [1]

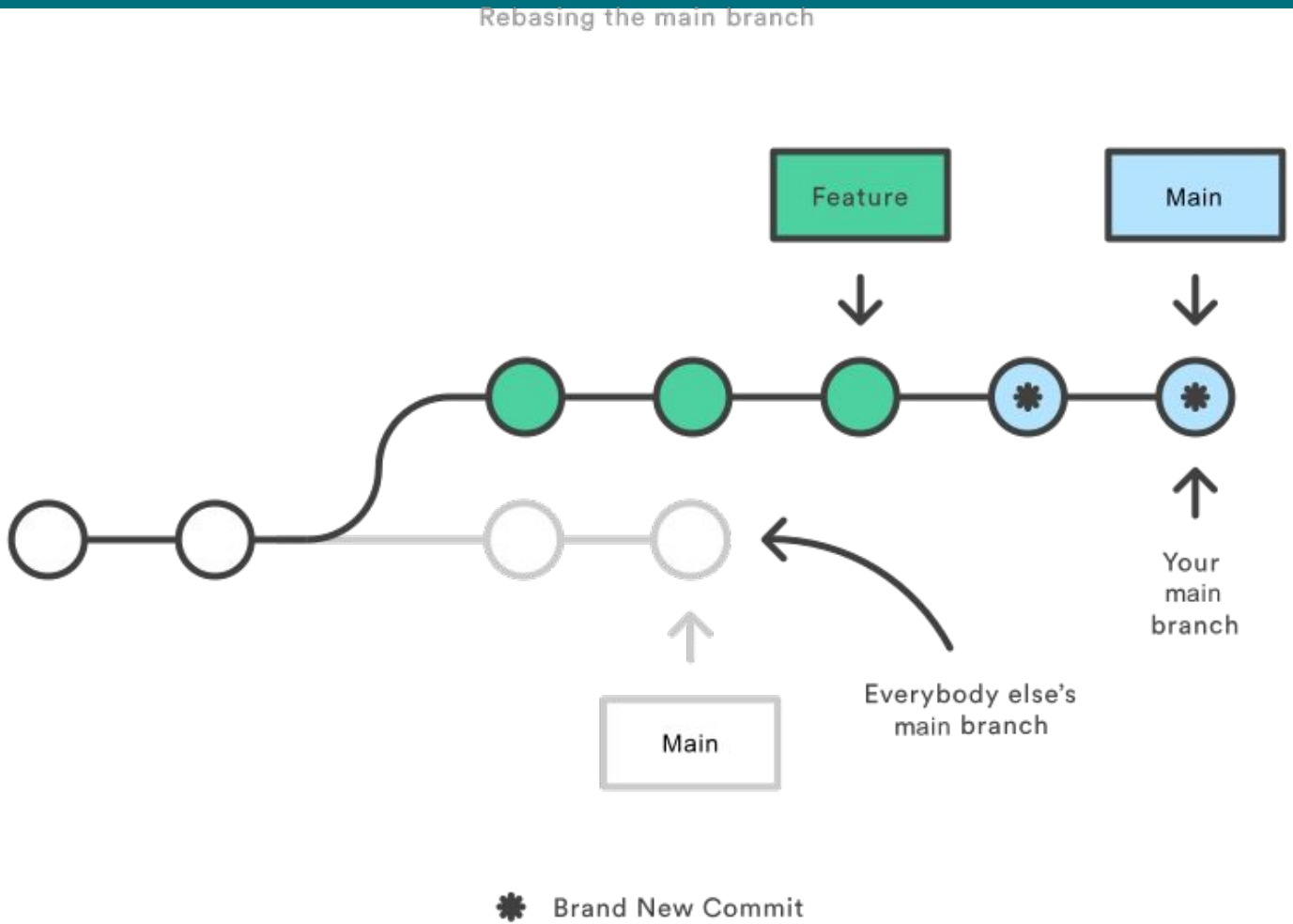
branch vs. rebase

Rebasing the feature branch onto main



Fonte: [1]

branch vs. *rebase* (NÃO FAZER)



Fonte: [1]

stash

Pilhas de trabalhos não concluídos, inclui arquivos modificados e toda área de *staging*. Útil quando você quer mudar de *branch* sem commitar.

61. Apague todos os *branches* exceto o main
62. Modifique um arquivo existente (só modifique)
63. Crie um arquivo novo e faça *staging*
64. Crie e mude para um *branch*
65. Veja o *status*
66. Volte para o main

stash

Pilhas de trabalhos não concluídos, inclui arquivos modificados e toda área de *staging*. Útil quando você quer mudar de *branch* sem commitar.

67. (no main) Faça *stash* (git stash)
68. Mude para seu *branch*
69. Veja o *status*
70. Modifique um arquivo e faça *stash*
71. Veja sua pilha (git stash list)
72. Volte para o main e desfaça a pilha (git stash pop)

📝 Exercício 11: *stash*

Faça o seguinte exercício (no seu repositório):

1. Garanta que está no main e não há *branches*
2. Faça uma modificação e *stash* (2x)
3. Faça git stash pop stash@{0}
4. Faça commit

O conteúdo do arquivo modificado é da primeira ou da segunda modificação?
Lembre-se de colocar a resposta no arquivo RESPOSTAS.md.

Para testar

Teste depois estes comandos (opcional):

- Apagar *stash*: git stash drop
- Cria *stash* mas mantém SA: git stash --keep-index
- Cria *stash* incluindo arquivos não adicionados: git stash -u
- Cria *branch* do topo da pilha: git stash branch <meubranch>

Submódulos

São projetos Git dentro de um repositório

- Clona-se diferente (git submodule add <endereço>)
  - Não cria WT (só referências)
  - Cria .gitmodules que precisa ser “commitado”
- O init simplesmente cria a WT (git submodule init)
- Não faz update com o repositório principal
  - git submodule update [--remote]

Git LFS

Large File Storage

- Armazena um ponteiro para um repositório externo
  - Não trabalha com blob como em arquivos normais
  - O servidor deve suportar Git LFS
- Ùtil para:
  - Arquivos muito grandes
  - Arquivos binários

Git LFS

- Habilite o LFS no repositório: git lfs install
- Para adicionar um tipo de arquivo: git lfs track <tipoarquivo>
  - Todos os arquivos deste tipo serão enviados para o LFS e não para o repositório comum (que terá um link)
- *staging*, *commit* e *push* agora vão para o LFS para este tipo de arquivo, *pull* também funciona
- Modificações são consideradas novas cópias (mesmo que 1 bit)
- Você pode desabilitar com git lfs uninstall

Ferramentas

- Há vários servidores capazes de manter uma cópia “de referência” para trabalho colaborativo
  - GitHub, GitLab, BitBucket, etc.
  - Você pode instalar o seu próprio servidor
  - Não é necessário um servidor, mas é mais fácil
- Todas as plataformas oferecem serviços além do VCS:
  - Wiki (documentação)
  - Issues (bug tracker)
  - Merges com permissão limitada (pull-request)

📝 Exercício 12: *issues*

Faça o seguinte exercício (no seu repositório):

1. Pela interface web, abra um *issue* no seu projeto, anote o número.
2. No seu repositório local, crie ou modifique um arquivo, faça *staging*.
3. Faça commit com uma mensagem “Fixes #1”

O que aconteceu com o *issue*? Se o mesmo for feito em um *branch*, o que acontece?

Lembre-se de colocar a resposta no arquivo RESPOSTAS.md.

Você agora sabe git!

Não se esqueça de:

- Enviar seu arquivo RESPOSTAS.md para o servidor até segunda;
- Eventualmente criar um arquivo de duvidas.md e colocar na raiz;
- A prova será liberada sexta-feira e há todo o final de semana para responder (prova bico).

Até 22h (ou último aluno na sala):

- Dúvidas sobre todos os comandos de hoje;
- Exercícios extras
- Desafio (5 níveis)

Bruno Albertini [PCS/EPUSP]

balbertini@usp.br

Discord e GitHub: balbertini

Não há tarefa obrigatória fora de aula!

Extras:

https://www.w3schools.com/git/exercise.asp
https://github.com/juanfresia/git-challenge
https://learngitbranching.js.org/

Bibliografia

(todos os links acessados em julho de 2022)

💻 [1] Atlassian/Bitbucket Git Tutorial, Merging vs. Rebasing (link).

📚 [2] Chacon, Scott. Pro Git (link).

💻 [3] Introdução com o Git no Azure Repos (link).