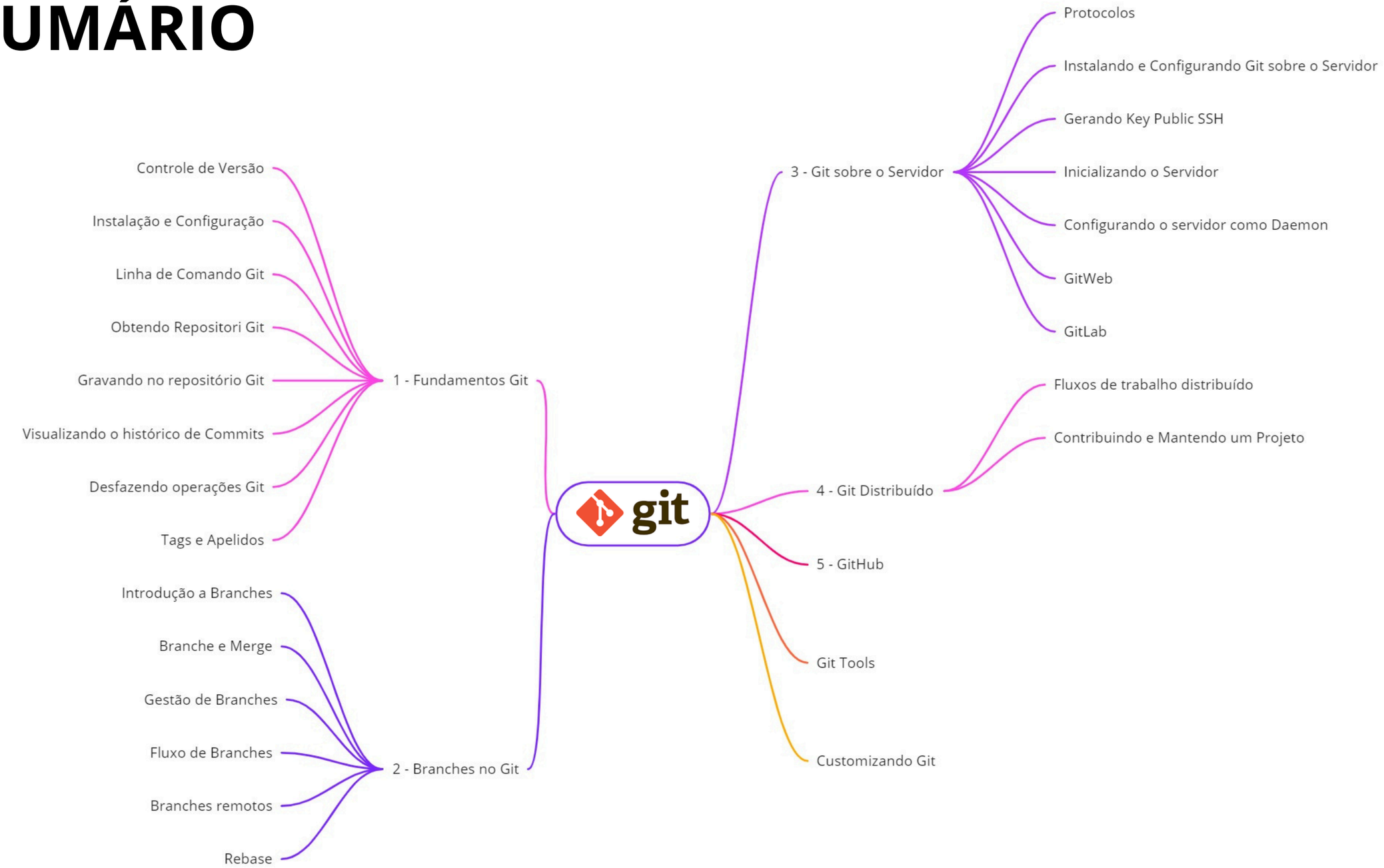
Controle de Versão

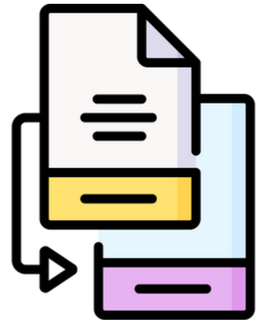
DESVENDANDO GIT E GITHUB

SUMÁRIO


git
1 - Fundamentos Git
Controle de Versão
Instalação e Configuração
Linha de Comando Git
Obtendo Repositori Git
Gravando no repositório Git
Visualizando o histórico de Commits
Desfazendo operações Git
Tags e Apelidos
2 - Branches no Git
Introdução a Branches
Branche e Merge
Gestão de Branches
Fluxo de Branches
Branches remotos
Rebase
3 - Git sobre o Servidor
Protocolos
Instalando e Configurando Git sobre o Servidor
Gerando Key Public SSH
Inicializando o Servidor
Configurando o servidor como Daemon
GitWeb
GitLab
4 - Git Distribuído
Fluxos de trabalho distribuído
Contribuindo e Mantendo um Projeto
5 - GitHub
Git Tools
Customizando Git

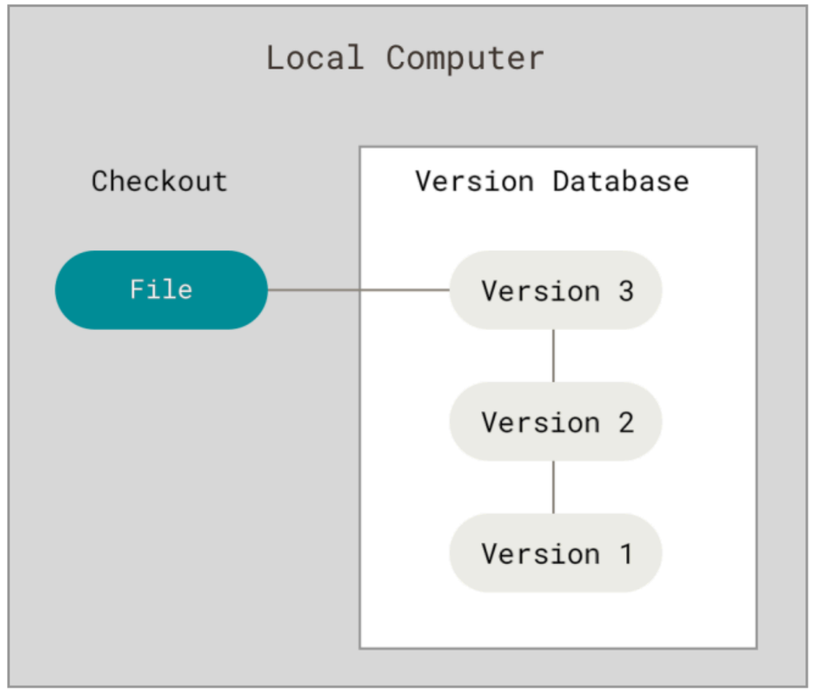
Controle de Versão

"Controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde"



Sistema Locais de Controle de Versão

Banco de dados simples que mantém todas às alterações dos arquivos sob controle de revisão.



Sistemas Centralizados de Controle de Versão

Um único servidor que contém todos os arquivos de controle de versão compartilhados entre diversos colaboradores ou desenvolvedores. O ponto positivo deve-se a um único ponto de controle e a desvantagem se este ponto falhar deixa de haver a colaboração compartilhada.

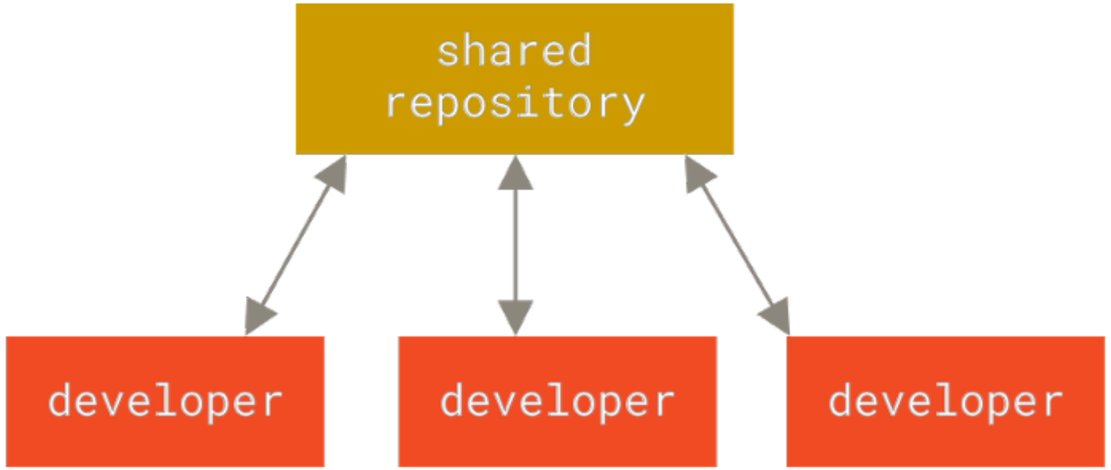


Figura 2. Controle de versão centralizado.

Sistemas Distribuídos de Controle de Versão

Duplicam localmente o repositório completo, de modo que se um servidor específico falhar e esses sistemas estiverem colaborando por meio dele, qualquer um dos repositórios de um dos clientes podem ser copiado de volta para o servidor para restaurá-lo. Cada clone é de fato um backup completo de todos os dados. É ideal para trabalhar repositórios remotos de modo a permitir que se clone o mesmo projeto de diferentes grupos de pessoas

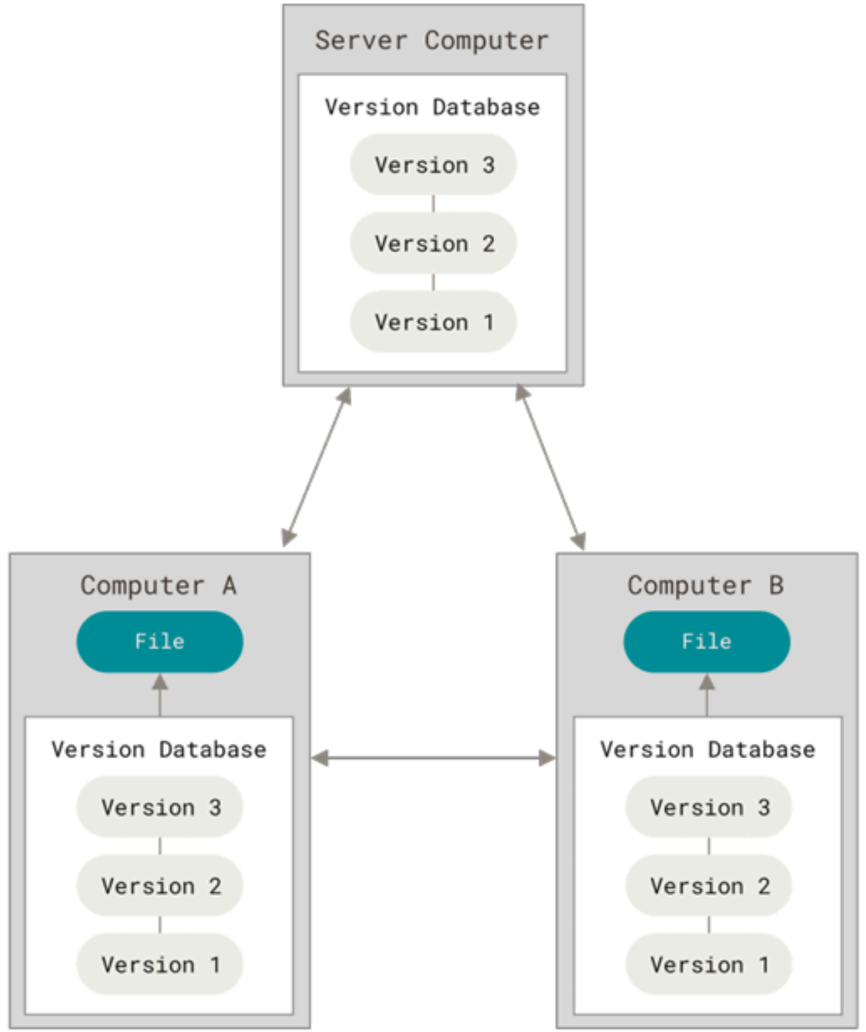


Figura 3. Controle de versão distribuído.

Introdução ao Git

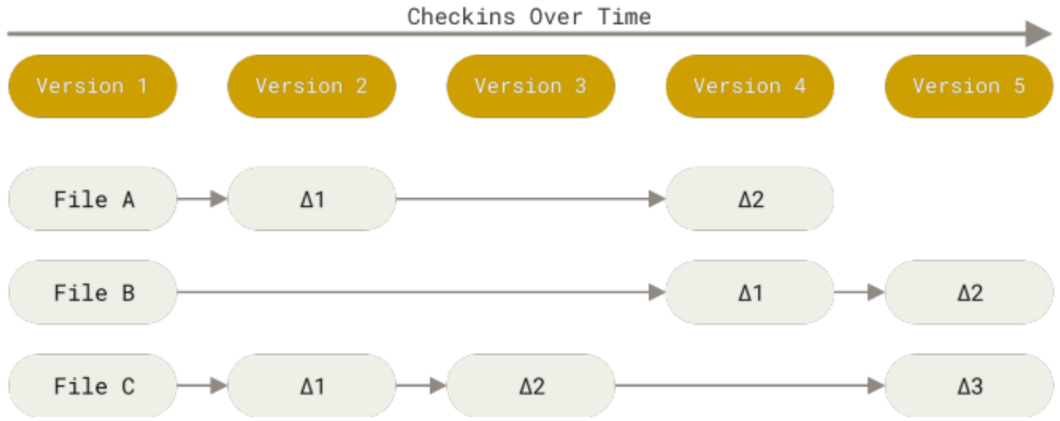
- Projeto simples a complexos - Ex: Kernel Linux
- Suporte a ramificações paralelas
- Distribuído
- Operações locais
- Integridade

Git vs Outros Sistemas de Controle de Versão

Outros

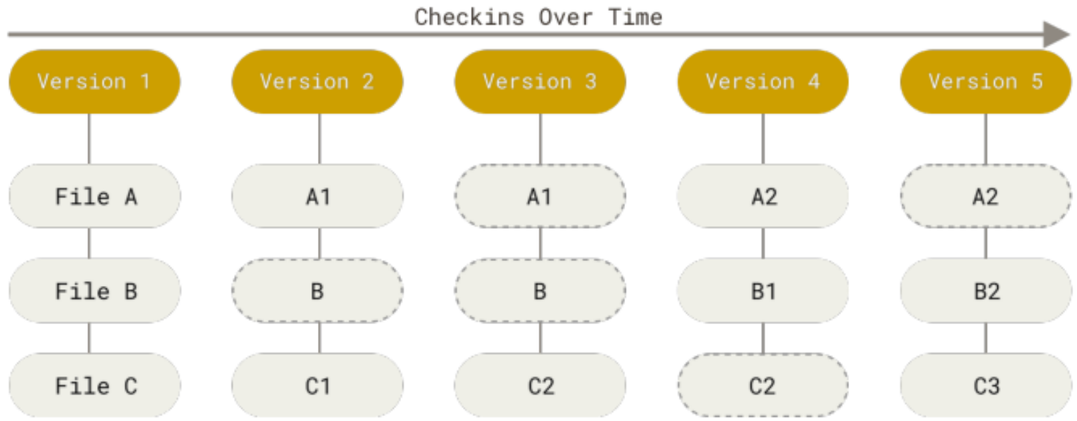
Tratam a informação como um conjunto de arquivos e registra uma lista de mudanças para cada arquivo ao longo do tempo.

- CSV
- Subversion
- Perforce
- Bazaar



Git

O Git trata seus dados mais como um conjunto de imagens de um sistema de arquivos em miniatura. Toda vez que você fizer um commit, ou salvar o estado de seu projeto no Git, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para esse conjunto de arquivos. Para ser eficiente, se os arquivos não foram alterados, o Git não armazena o arquivo novamente, apenas um link para o arquivo idêntico anterior já armazenado. O Git trata seus dados mais como um fluxo do estado dos arquivos



Operações locais

- Não é necessário outro computador de rede para realizar as operações locais, evitando a demora causada pela latência da rede como na maioria dos outros sistemas de controle de versão
- O fato do Git não depender de servidores para realizar às operações locais, permite a execução de commits sem conexão de rede até conseguir alguma.

Integridade

- o Git passa por uma soma de verificações (checksum) antes de ser armazenado e é referenciado por esse checksum. Isto significa que é impossível mudar o conteúdo de qualquer arquivo ou pasta sem que Git saiba
- não perderá informação durante a transferência e não receberá um arquivo corrompido sem que o Git seja capaz de detectar
- o Git utiliza um mecanismo de soma de verificação chamado: hash SHA-1. Esta é uma sequência de 40 caracteres composta de caracteres hexadecimais (0-9 e-f) e é calculada com base no conteúdo de uma estrutura de arquivo ou diretório no Git. Um hash SHA-1.


```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Instalação e Configuração do GIT

Instalando Git

Instalando no Linux

```
$ sudo yum install git-all
```

```
$ sudo apt-get install git-all
```

Instalando no Mac

Atualização do próprio Git

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

Instalador Git para Windows

https://git-scm.com/download/win

Configuração Inicial do Git

1 - Comando git config

git config: permite ver e atribuir variáveis de configuração que controlam todos os aspectos de como o Git aparece e opera;

```
$ git config --global user.name "Fulano de Tal"
$ git config --global user.email fulanodetal@exemplo.br
```

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

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

2 - Variáveis de configuração Git

- **Todos os usuários:** /etc/gitconfig válido para todos os usuários no sistema e todos os seus repositórios. Se você passar a opção --system para o **git config**, ele lê e escreve neste arquivo.
- **Usuário espcífico:** ~/.gitconfig ou ~/.config/git/config Somente para o seu usuário. Você pode fazer o Git ler e escrever neste arquivo passando a opção --global
- **Repositório em uso:** config no diretório Git (ou seja, .git/config) de qualquer repositório que você esteja usando: específico para este repositório

3 - Arquivo .gitconfig

```
HP-DANIEL+danie@hp-daniel MINGW64 ~
$ pwd
/c/Users/danie

HP-DANIEL+danie@hp-daniel MINGW64 ~
$ ls .gitconfig
.gitconfig

HP-DANIEL+danie@hp-daniel MINGW64 ~
$ cat .gitconfig
[user]
        name = Daniel Corrêa da Silva
        email = daniel@devtech-edu.net
```

4 - Status do repositório

```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-e
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

5 - Testando as configurações

```
HP-DANIEL+danie@hp-daniel
$ git config --list
```

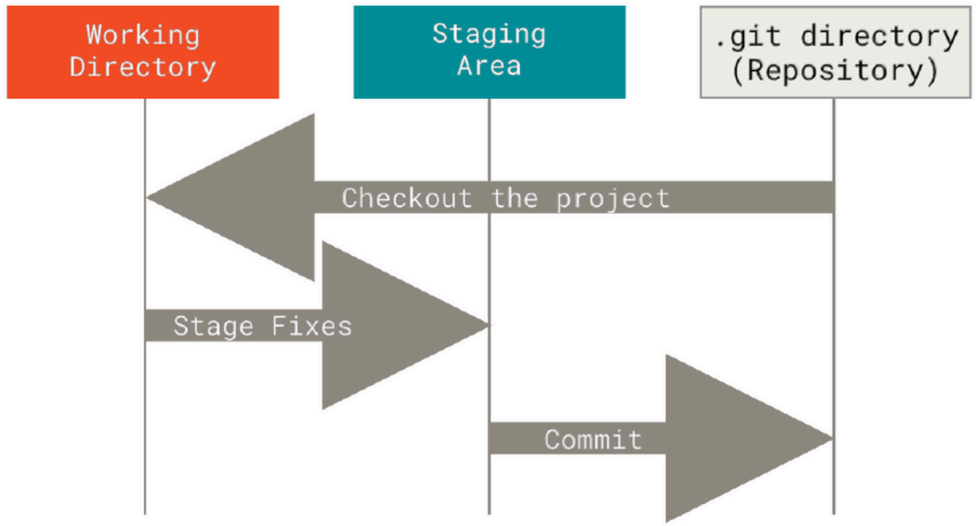
```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/dev-tech/sites/dev
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bu
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Daniel Corrêa da Silva
user.email=daniel@devtech-edu.net
```

Gravando Alterações em Seu Repositório

Os três estados do Git

No Git, os arquivos podem estar em 3 estados:

- **Estado Committed:** significa que os dados estão armazenados de forma segura em seu banco de dados local.
- **Estado Modificado:** significa que você alterou o arquivo, mas ainda não fez o commit no seu banco de dados.
- **Estado Preparado:** significa que você marcou a versão atual de um arquivo modificado para fazer parte de seupróximo commit.



Quando faz algo no Git, quase sempre dados são adicionados no banco de dados do Git. Perde-se dados- caso não tenha feito o commit, após o commit armazena-se o estado atual das alterações

- **Diretório Git:** é onde o Git armazena os metadados e o banco de dados de objetos de seu projeto. Clone de um repositório de outro computador;
- **Diretório de trabalho:** é o diretório do seu sistema de arquivos onde você está trabalhando em seus arquivos. É onde você pode criar, editar e excluir arquivos, e onde as alterações são feitas antes de serem registradas no repositório Git. É uma simples cópia de uma versão do projeto. Esses arquivos são pegos do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar;
- **Área de preparo:** é um arquivo, geralmente contido em seu diretório Git, que armazena informações sobre o quem entrará em seu próximo commit. É por vezes referido como o “índice”, mas também é comum referir-se a ele como área de preparo (staging area).

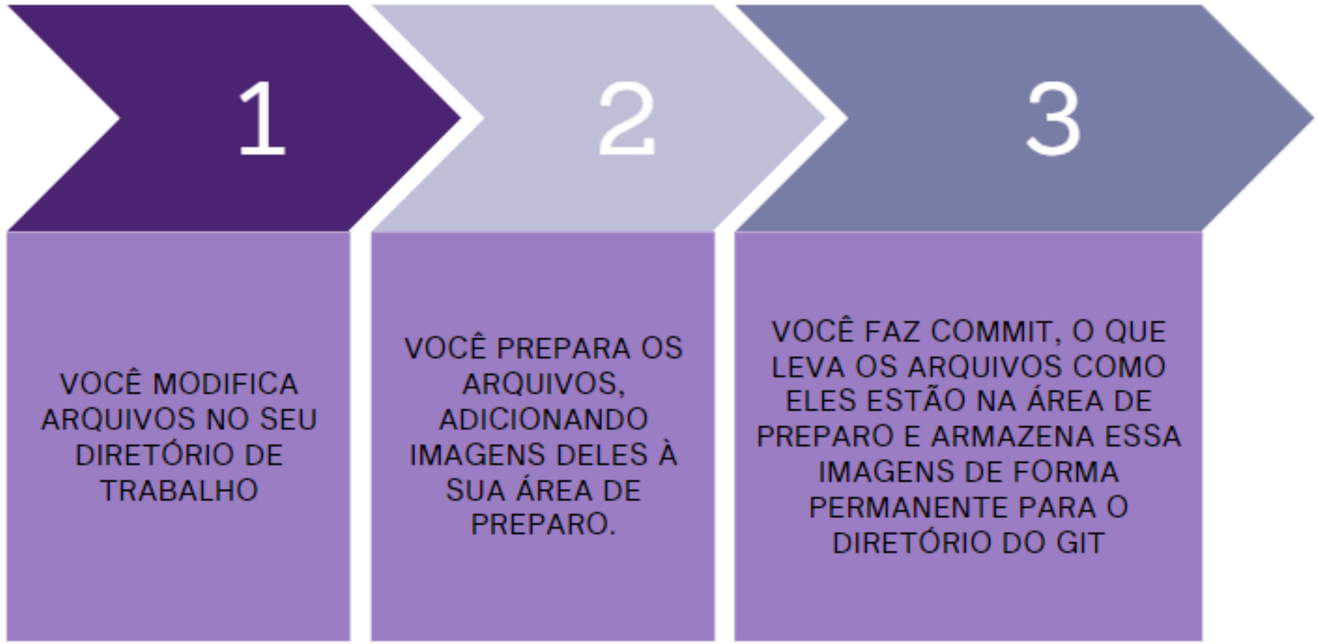
DIRETÓRIO / ARQUIVOS GIT	DESCRIÇÃO
hooks/	contém scripts que podem ser executados automaticamente pelo Git em eventos específicos, como a criação de um commit ou o recebimento de um push.
info/	contém um arquivo chamado exclude, que lista padrões de arquivos e diretórios que devem ser ignorados pelo Git.
objects/	é o diretório onde o Git armazena os objetos do repositório. Os objetos podem ser blobs (arquivos), árvores (diretórios) ou commits.
refs/	contém referências para commits específicos. As referências podem ser branches (ramificações), tags ou HEAD.
HEAD	é um arquivo que aponta para a referência atual em que o repositório está trabalhando.
config	é um arquivo de configuração do Git, que contém informações como o nome e endereço de email do usuário, e as configurações padrão do repositório.
description	é um arquivo que contém uma descrição curta do repositório.
index	é o arquivo que representa a área de preparação do Git, onde as alterações que serão incluídas no próximo commit são armazenadas.

Gravando alterações em seu repositório

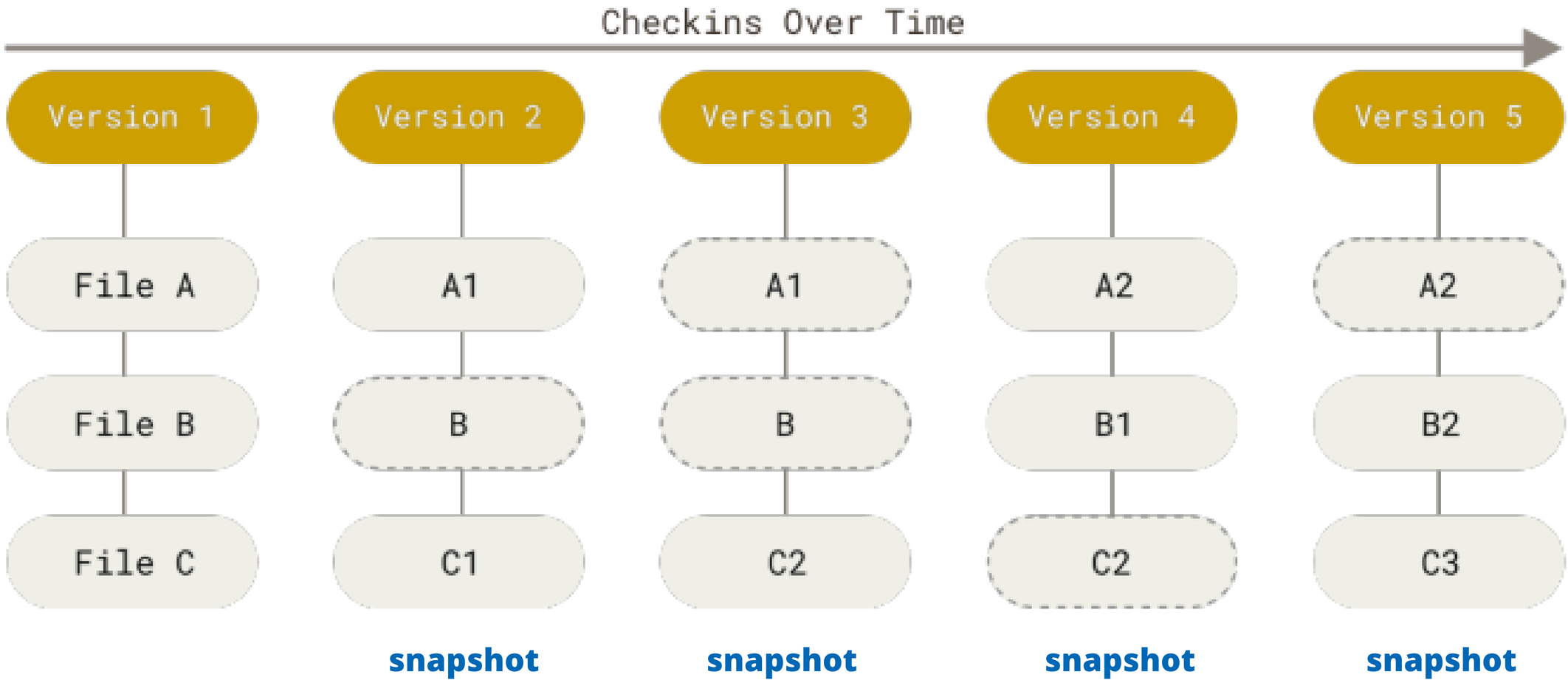
Seu diretório de trabalho pode estar em um dos seguintes estados: rastreado e não-rastreado:

- **Arquivos rastreados:** são arquivos que foram incluídos no último **snapshot**; eles podem ser não modificados, modificados ou preparados (adicionados ao stage). Em resumo, *arquivos rastreados são os arquivos que o Git conhece*.
- **Arquivo não rastreados:** são todos os outros - quaisquer arquivos em seu diretório de trabalho que não foram incluídos em seu último snapshot e não estão na área de stage.

Quando você clona um repositório pela primeira vez, todos os seus arquivos serão rastreados e não modificados já que o Git acabou de obtê-los e você ainda não editou nada.



snapshot: uma imagem/cópia instântanea do repositório



snapshot: uma cópia instântanea do repositório

Configuração Inicial do Git

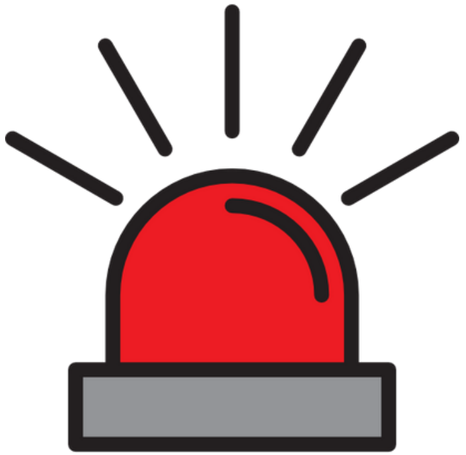
1 - Inicializando um repositório Git

Isso cria um novo subdiretório chamado .git que contém todos os arquivos necessários de seu repositório – um esqueleto de repositório Git. Neste ponto, nada em seu projeto é monitorado ainda.

```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu
$ git init
Initialized empty Git repository in F:/dev/repositorios-git/devtech-edu/.git/

HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu (master)
$ ls

HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu (master)
$ ls -a
./  ../  .git/
```

Dentro do diretório Git:

```
IEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu/.git (GIT_DIR!)

config  description  hooks/  info/  objects/  refs/
```



ATENÇÃO

Se remover a pasta .git perderá todas as informações sobre o repositório e os controles sobre o versionamento dos arquivos

Clonando um repositório existente

Caso você queira obter a cópia de um repositório Git existente – por exemplo, um projeto que você queira contribuir – o comando para isso é git clone. o Git recebe uma cópia completa de praticamente todos os dados que o servidor possui. Cada versão de cada arquivo no histórico do projeto é obtida por padrão quando você executa git clone.

```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu (master)
$ git clone https://github.com/libgit2/libgit2
```

Isso cria um diretório chamado libgit2, inicializa um diretório .git dentro dele, recebe todos os dados deste repositório e deixa disponível para trabalho a cópia da última versão

```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu/.git (GIT_DIR!)
$ pwd
/f/dev/repositorios-git/devtech-edu/.git

HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/devtech-edu/.git (GIT_DIR!)
$ ls
HEAD  config  description  hooks/  info/  objects/  refs/
```

Rastreando arquivos novos

Para começar a rastrear um novo arquivo, você deve usar o comando git add. Para começar a rastrear o arquivo README, você deve executar o seguinte

```
1    git add README
```

```
1    git status
2    git status -s
```

Executando o comando status novamente, você pode ver que seu README agora está sendo rastreado e preparado (staged) para o commit:

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

O comando git add recebe o caminho de um arquivo ou de um diretório. *Se for um diretório, o comando adiciona todos os arquivos contidos nesse diretório recursivamente.*

```
1    git rm nome_arquivo
```

Preparando arquivos modificados

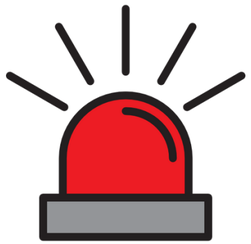
Se você modificar o arquivo CONTRIBUTING.md, que já era rastreado e então executar git status novamente, você deve ver algo como:

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

O arquivo CONTRIBUTING.md aparece sob a seção “Changes not staged for commit” — que indica que um arquivo rastreado foi modificado no diretório de trabalho, mas ainda não foi mandado para o stage (preparado). Para mandá-lo para o stage, você precisa executar o comando git add.

O git add é um comando de múltiplos propósitos: serve para começar a rastrear arquivos e também para outras coisas, como marcar arquivos que estão em conflito de mesclagem como resolvidos.

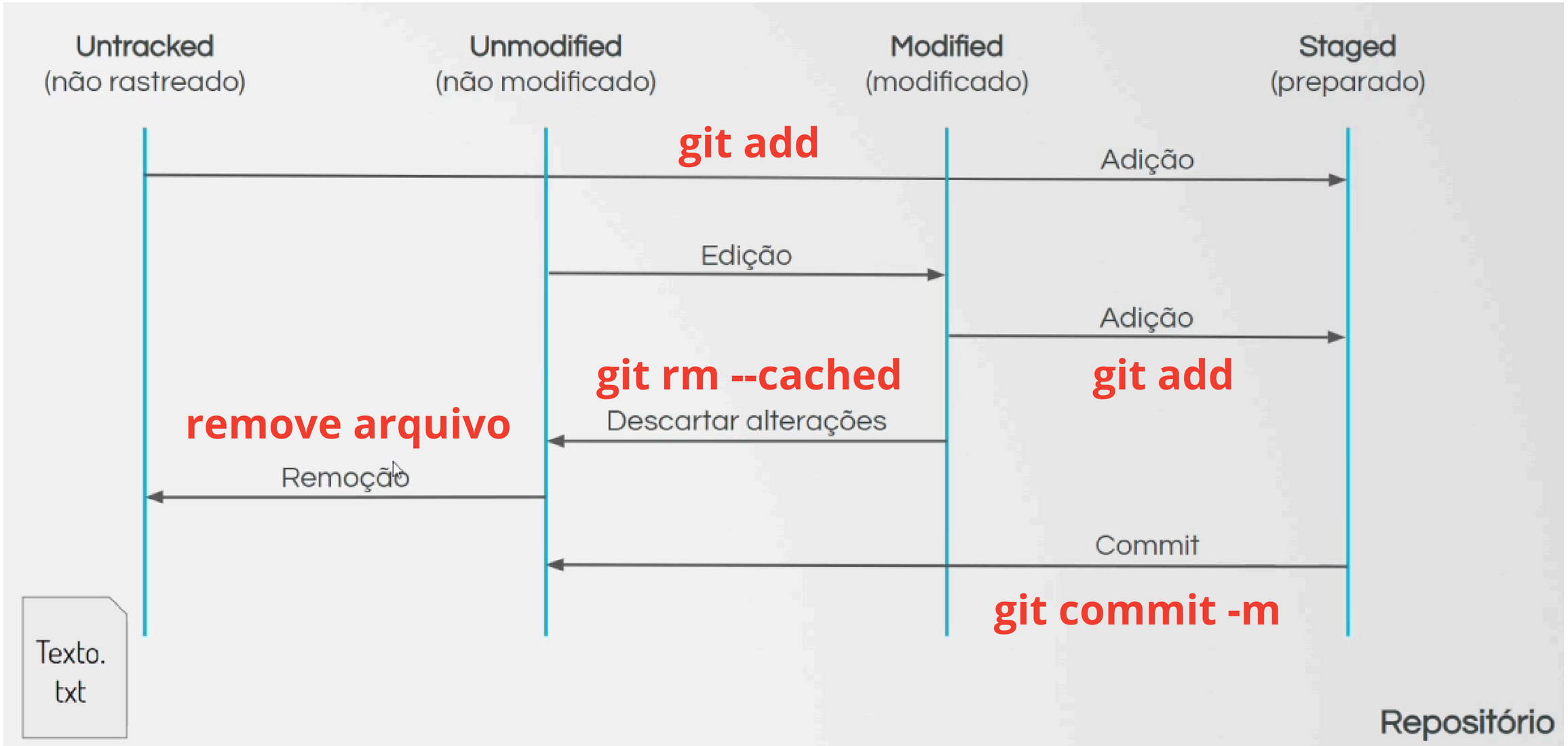
Pode ser útil pensar nesse comando mais como “adicione este conteúdo ao próximo commit”.



Atenção com a seguinte situação:

cria-se um novo arquivo e adiciona para a área de preparação (staged) e em seguida modifica o arquivo.

```
HP-DANIEL+danie@hp-daniel MINGW64 /f/dev/repositorios-git/proj-devtech-edu (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        new file:   config.json
        new file:   teste.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   teste.txt
```


Untracked
(não rastreado)
Unmodified
(não modificado)
Modified
(modificado)
Staged
(preparado)
git add
Adição
Edição
Adição
git rm --cached
git add
remove arquivo
Descartar alterações
Remoção
Commit
git commit -m
Texto.
txt
Repositório

Visualizando alterações

Você quer saber exatamente o que você alterou, não apenas quais arquivos foram alterados — você pode usar o comando git diff. Você deseja responder as seguintes perguntas:

- que você alterou mas ainda não mandou para o stage (estado preparado)?
- E o que está no stage, pronto para o commit?

```
1    git diff
2    git diff --cached
3    git diff --staged
```

Esse comando compara o que está no seu diretório de trabalho com o que está no stage. O resultado permite que você saiba quais alterações você fez que ainda não foram mandadas para o stage.

Se você já tiver mandado todas as suas alterações para o stage, a saída do git diff vai ser vazia. Se você quiser ver as alterações que você mandou para o stage e que entrarão no seu próximo commit, você pode usar git diff --staged.

Enviando as alterações com o Commit

Agora que sua área de stage está preparada do jeito que você quer, você pode fazer commit das suas alterações. Lembre-se que qualquer coisa que ainda não foi enviada para o stage — qualquer arquivo que você tenha criado ou alterado e que ainda não tenha sido adicionado com git add — não entrará nesse commit. Esses arquivos permanecerão no seu disco como arquivos alterados.

```
1    git commit -m "Story 182: Fix benchmarks for speed"
```

```
[master 463dc4f] Story 182: Fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

Veja que a saída do comando fornece algumas informações: em qual branch foi feito o commit (master), seu checksum SHA-1 (463dc4f), quantos arquivos foram alterados e estatísticas sobre o número de linhas adicionadas e removidas.

Removendo Arquivos

Para remover um arquivo do Git, você tem que removê-lo dos seus arquivos rastreados (mais precisamente, removê-lo da sua área de stage) e então fazer um commit. O comando **git rm** faz isso, e também remove o arquivo do seu diretório de trabalho para que você não o veja como um arquivo não-rastreado nas suas próximas interações.

Se você simplesmente remover o arquivo do seu diretório, ele aparecerá sob a área “Changes not staged for commit” (isto é, fora do stage) da saída do seu git status:

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    PROJECTS.md

no changes added to commit (use "git add" and/or "git commit -a")
```

```
1    git rm README.md
```

Para removê-lo da sua área de stage e manter o arquivo no seu disco rígido, execute o comando:

```
1    git rm --cached README.md
```

Movendo Arquivos

Diferentemente de outros sistemas de controle de versão, o Git não rastreia explicitamente a movimentação de arquivos. Se você renomear um arquivo no Git, ele não armazena metadados indicando que determinado arquivo foi renomeado. Se você quiser renomear um arquivo no Git, você pode executar alguma coisa como:

```
$ git mv arq_origem arq_destino
```

```
1    git mv README.md README
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

Visualizando o histórico de commits

Quando clona um repositório, o git possibilita visualizar o histórico pré-existente de um repositório. O comando abaixo mostra o log de commits em ordem cronológica inversa; isto é, o commit mais recente aparece primeiro.

```
git log
git log -p # mostra os diffs de cada commit
git log --stat # exibe uma estatistica
git log --pretty=oneline # exibe em uma linha
```

```
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

Opções	Descrição
`-p`	Mostra o patch introduzido com cada commit.
`--stat`	Mostra estatísticas de arquivos modificados em cada commit.
`--shortstat`	Exibe apenas a linha informando a alteração, inserção e exclusão do comando --stat.
`--name-only`	Mostra a lista de arquivos modificados após as informações de commit.
`--name-status`	Mostra também a lista de arquivos que sofreram modificação com informações adicionadas / modificadas / excluídas.
`--abbrev-commit`	Mostra apenas os primeiros caracteres da soma de verificação SHA-1 em vez de todos os 40.
`--relative-date`	Exibe a data em um formato relativo (por exemplo, “ 2 semanas atrás ”) em vez de usar o formato de data completo.
`--graph`	Exibe um gráfico ASCII do histórico de branches e merges ao lado da saída do log.
`--pretty`	Mostra os commits em um formato alternativo. As opções incluem oneline, short, full, fuller e format (onde você especifica seu próprio formato).

Visualizando o histórico de commits

A opção format

Opção	Descrição da saída
%H	Hash do commit
%h	Hash do commit abreviado
%T	Hash da árvore
%t	Hash da árvore abreviado
%P	Hashes dos pais
%p	Hashes dos pais abreviado
%an	Nome do Autor
%ae	Email do Autor
%ad	Data do Autor (o formato segue a opção --date=option)
%ar	Data do Autor, relativa
%cn	Nome do Committer
%ce	Email do Committer
%cd	Data do Committer
%cr	Data do Committer, relativa
%s	Comentário

Visualizando o histórico de commits

Veja alguns exemplos:

```
git log --pretty=format:"%h - %an, %ar : %s"
```

```
ca82a6d - Scott Chacon, 6 years ago : changed the version number
085bb3b - Scott Chacon, 6 years ago : removed unnecessary test
a11bef0 - Scott Chacon, 6 years ago : first commit
```

```
git log --pretty=format:"%h %s" --graph
```

```
* 2d3acf9 ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | 5a09431 add timeout protection to grit
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Limitando o retorno do comando Log:

```
git log --since=2.weeks
```

```
$ git log --pretty="%h - %s" --author=gitster --since="2008-10-01" \
   --before="2008-11-01" --no-merges -- t/
5610e3b - Fix testcase failure when extended attributes are in use
acd3b9e - Enhance hold_lock_file_for_{update,append}() API
f563754 - demonstrate breakage of detached checkout with symbolic link HEAD
d1a43f2 - reset --hard/read-tree --reset -u: remove unmerged new paths
51a94af - Fix "checkout --track -b newbranch" on detached HEAD
b0ad11e - pull: allow "git pull origin $something:$current_branch" into an unborn branch
```

Desfazendo coisas

Quando você executa um commit muito cedo e possivelmente esquecendo de adicionar alguns arquivos ou você escreveu a mensagem do commit de forma equivocada. Se você quiser refazer este commit, execute o commit novamente usando a opção --amend:

```
git commit --amend
```

Esse comando pega a área stage e a usa para realizar o commit. Se você não fez nenhuma alteração desde o último commit (por exemplo, se você executar o comando imediatamente depois do commit anterior), então sua imagem dos arquivos irá ser exatamente a mesma, e tudo o que você alterará será a mensagem do commit

Seja cuidadoso, porque nem sempre você pode voltar uma alteração desfeita. Essa é uma das poucas áreas do Git onde pode perder algum trabalho feito se você cometer algum engano.

```
git commit -m 'initial commit'
git add forgotten_file
git commit --amend
```

Retirando um arquivo do Stage

Suponhamos que você alterou dois arquivos, e deseja realizar o commit deles separadamente, porém você acidentalmente digitou git add * adicionando ambos ao stage. Como você pode retirar um deles do stage? O comando git status lhe lembrará de como fazer isso:

```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

Suponhamos que você alterou dois arquivos, e deseja realizar o commit deles separadamente, porém você acidentalmente digitou git add * adicionando ambos ao stage. Como você pode retirar um deles do stage? O comando git status lhe lembrará de como fazer isso:

```
1    git reset README.md
```

Desfazendo as Modificações de um Arquivo

E se você se der conta de que na verdade não quer manter as modificações do arquivo? Como você pode reverter as modificações, voltando a ser como era quando foi realizado o último commit

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

```
1    git checkout -- README.md
```

Se você gostaria de manter as modificações que fez no arquivo, porem precisa tirá-lo do caminho por enquanto, sugerimos utilize Branches; esta geralmente é a melhor forma de fazer isso. Utilize Data Recovery para recuperação de dados em situações deletou-se alguma Branches ou commits (--amend)

Trabalhando de Froma Remota

Repositórios Remotos

Repositórios remotos são versões de seu repositório hospedado na Internet ou em uma rede qualquer. Você pode ter vários deles, cada um dos quais geralmente é ou somente leitura ou leitura/escrita. Colaborar com outras pessoas envolve o gerenciamento destes repositórios remotos, fazer pushing(atualizar) e pulling(obter) de dados para e deles quando você precisar compartilhar seu trabalho. Gerenciar repositórios remotos inclui saber como adicioná-los remotamente, remover aqueles que não são mais válidos, gerenciar vários branches(ramos) e definí-los como rastreados ou não e muito mais.

Exibindo seus repositórios remotos

Para ver quais servidores remotos você configurou, você pode executar o comando abaixo, especificando -v, que mostra as URLs que o Git tem armazenado pelo nome abreviado a ser usado para ler ou gravar naquele repositório remoto.

```
1    git remote -v
```

```
$ git remote -v
bakkdoor  https://github.com/bakkdoor/grit (fetch)
bakkdoor  https://github.com/bakkdoor/grit (push)
cho45     https://github.com/cho45/grit (fetch)
cho45     https://github.com/cho45/grit (push)
defunkt   https://github.com/defunkt/grit (fetch)
defunkt   https://github.com/defunkt/grit (push)
koke      git://github.com/koke/grit.git (fetch)
koke      git://github.com/koke/grit.git (push)
origin    git@github.com:mojombo/grit.git (fetch)
origin    git@github.com:mojombo/grit.git (push)
```

O mesmo repositório local pode apontar para um ou mais repositórios remotos

fetch : buscar/obter o repositório
push : enviar/atualizar o repositório remoto

Adicionando Repositórios Remotos

Para adicionar/apontar para um rempositório remoto, deve-se executar os seguintes comandos:

```
git remote add pb https://github.com/paulboone/ticgit
```

O mesmo repositório local pode apontar para um ou mais repositórios remotos

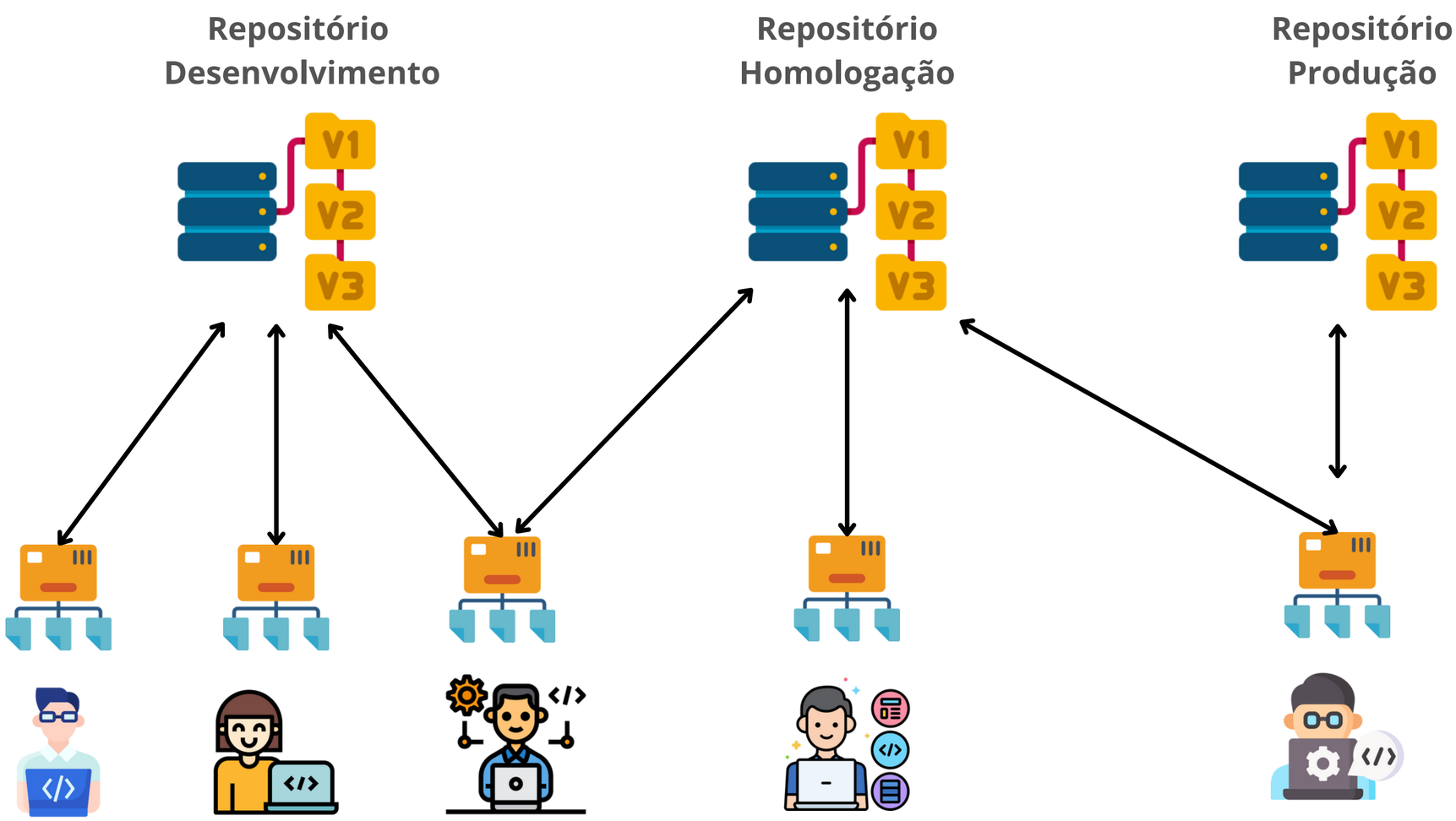
```
git remote set-url https://github.com/paulboone/ticgit
```

Para mudar a url do servidor remoto, execute o comando:

```
git remote rename origin novo_nome
```

```
# Define a URL do repositório remoto para fetch
git remote set-url --fetch origin https://github.com/user/repo.git

# Define a URL do repositório remoto para push
git remote set-url --push origin git@github.com:user/repo.git
```

Cenário



Repositórios Remotos

Repositórios Locais

Buscando e Obtendo de seus Repositórios Remotos

O comando vai até aquele projeto remoto e extrai todos os dados daquele projeto que você ainda não tem. Depois que você faz isso, você deve ter como referência todos as ramificações (branches) daquele repositório remoto, que você pode mesclar (merge) com o atual ou inspecionar a qualquer momento. É importante notar que o comando git fetch só baixa os dados para o seu repositório local - ele não é automaticamente mesclado (merge) com nenhum trabalho seu ou modificação que você esteja trabalhando atualmente.

git fetch [remote-name]

Você deve mesclá-los manualmente dentro de seu trabalho quando você estiver pronto.
Se o branch atual é configurado para rastrear um branch remoto, você pode usar o comando git pull para buscar (fetch) e então mesclar (merge) automaticamente aquele branch remoto dentro do seu branch atual.

git pull [remote-name] [branch-name]

Enviando para seu Repositório Remoto

Quando você tem seu projeto em um ponto que deseja compartilhar, é necessário enviá-lo para o servidor remoto. O comando para isso é simples:

git push [remote-name] [branch-name]

Se você quiser enviar sua ramificacão (branch) master para o servidor origin (novamente, a clonagem geralmente configura ambos os nomes para você automaticamente), então você pode executar isso para enviar quaisquer commits feitos para o servidor:

```
1    git push origin main
```

Este comando funciona apenas se você clonou de um servidor ao qual você tem acesso de escrita (write-access) e se ninguém mais utilizou o comando push nesse meio-tempo. Se você e outra pessoa clonarem o repositório ao mesmo tempo e ela utilizar o comando push e, em seguida, você tentar utilizar, seu envio será rejeitado. Primeiro você terá que atualizar localmente, incorporando o trabalho dela ao seu, só assim você poderá utilizar o comando push.

Inspecionando o Servidor Remoto

Se você quiser ver mais informações sobre um servidor remoto em particular, você pode usar o comando?

git remote show [nome-remoto]

Ele lista a URL para o repositório remoto, bem como as informações de rastreamento do branch. O comando, de forma útil, comunica que se você estiver no branch master e executar git pull, ele irá mesclar (merge) automaticamente no branch master do servidor após buscar (fetch) todas as referências remotas. Ele também lista todas as referências remotas recebidas.

Ao executar este comando com um nome abreviado específico, como origin, obterá algo assim:

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master                               tracked
    dev-branch                           tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

...or create a new repository on the command line

```
echo "# proj-portalFabrica" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/daniel-fatesg/proj-portalFabrica.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/daniel-fatesg/proj-portalFabrica.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Criando Tags

Tags

O Git possibilita marcar pontos específicos e importantes do histórico do rastreamento do repositório. Desenvolvedores utilizam o conceito de **tag** do Git para implementar a funcionalidade de marcar esses pontos, denominados releases - exemplo v1.0, etc. Portanto, é importante saber como listar as tags existentes, como criar novas tags e quais são os diferentes tipos de tags.

Para listar as tags exixtentes, basta executar um dos comandos abaixo:

```
git tag
git tag -l "v1.8.5*"
```

O Git usa dois tipos de tags: **Leve** e **Anotada**. Uma tag do tipo **leve** é muito parecida com um branch que não muda – Ela apenas aponta para um commit em especifico. Tags **anotadas**, entretanto, são um armazenamento completo de objetos no banco de dados do Git. Elas têm checksum, contem marcações de nome, email e data; têm uma mensagem de tag; e podem ser assinadas e asseguradas pela GPG (GNU Privacy Guard). É geralmente recomendado que você crie tags anotadas assim você tem todas as informações

```
# Tag do tipo anotada
git tag -a v1.4 -m "my version 1.4"

# Tag do tipo leve
git tag v1.4-]

# Para criar tags após o commit informe checksum
git tag -a v1.2 9fceb02
```

git show: tags do tipo leve e anotados

O comando git show não exibe informações extras quando a tag é do tipo leve quando comparado ao tipo anotado.

```
$ git show v1.4
tag v1.4
Tagger: Ben Straub <ben@straub.cc>
Date:   Sat May 3 20:19:12 2014 -0700

my version 1.4

commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number
```

```
$ git show v1.4-lw
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number
```

Compartilhando Tags

Por padrão, o comando git push não envia as tags para os servidores remoto. Você terá que explicitamente enviar as tags para o servidor de compartilhamento depois de tê-las criado. Esse processo é semelhante a compartilhar branches remotos – você pode executar:

git push origin [tagname]

```
git push origin v1.5

# Enviar muitas tags
git push origin --tags
```

```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.5 -> v1.5
```

```
$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.4 -> v1.4
 * [new tag]         v1.4-lw -> v1.4-lw
```

Compartilhando Tags

Você não pode realizar o checkout de uma tag no Git, uma vez que elas não podem ser movidas. Se você quer deixar uma versão do seu repositório idêntica a uma tag especifica em seu diretório de trabalho, você pode criar um novo branch em uma tag especifica com o comando:

git checkout -b [branchname] [tagname]

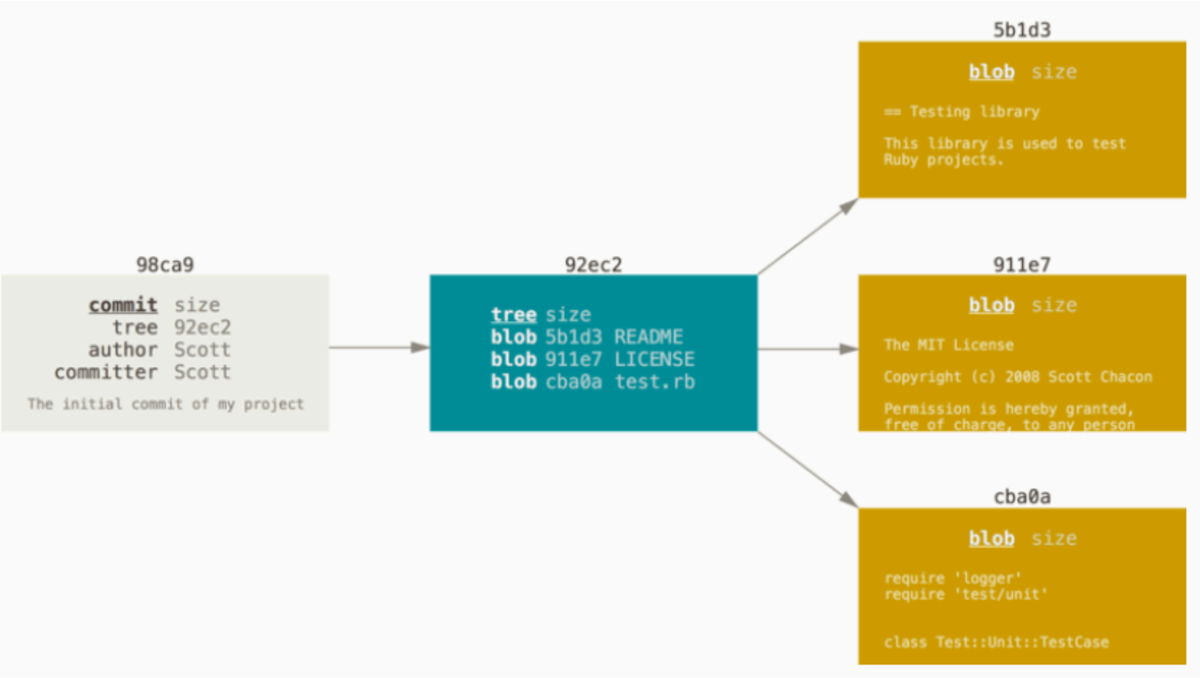
```
1    git checkout -b version2 v2.0.0
```

Branchs no Git

Introdução a Branchs

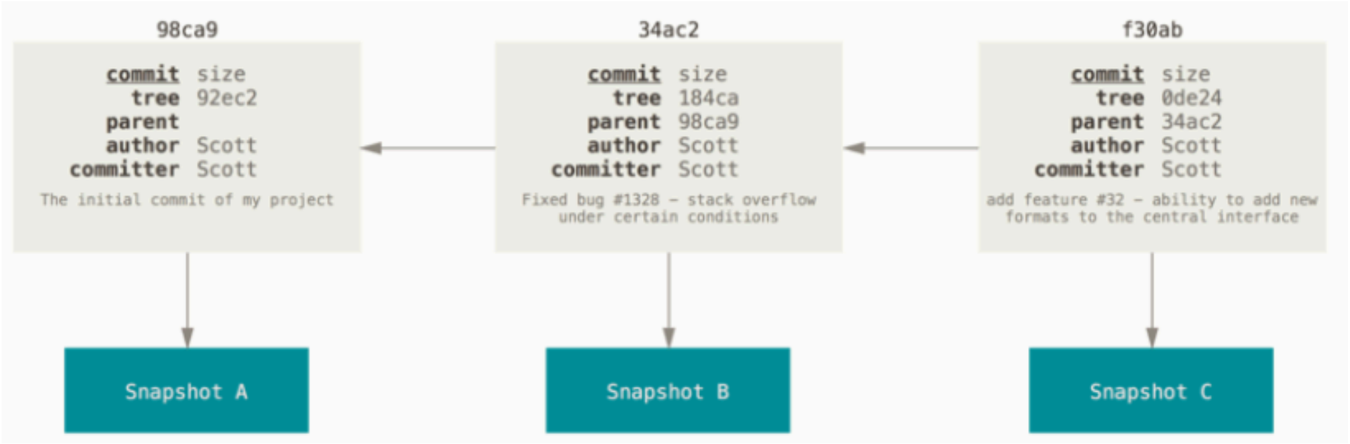
Branches (Ramificação): significa que você diverge da linha principal de desenvolvimento e continua a trabalhar sem alterar essa linha principal.
Para realmente entender como o Git trabalha com Branches, precisamos dar um passo atrás e examinar como o Git armazena seus dados.

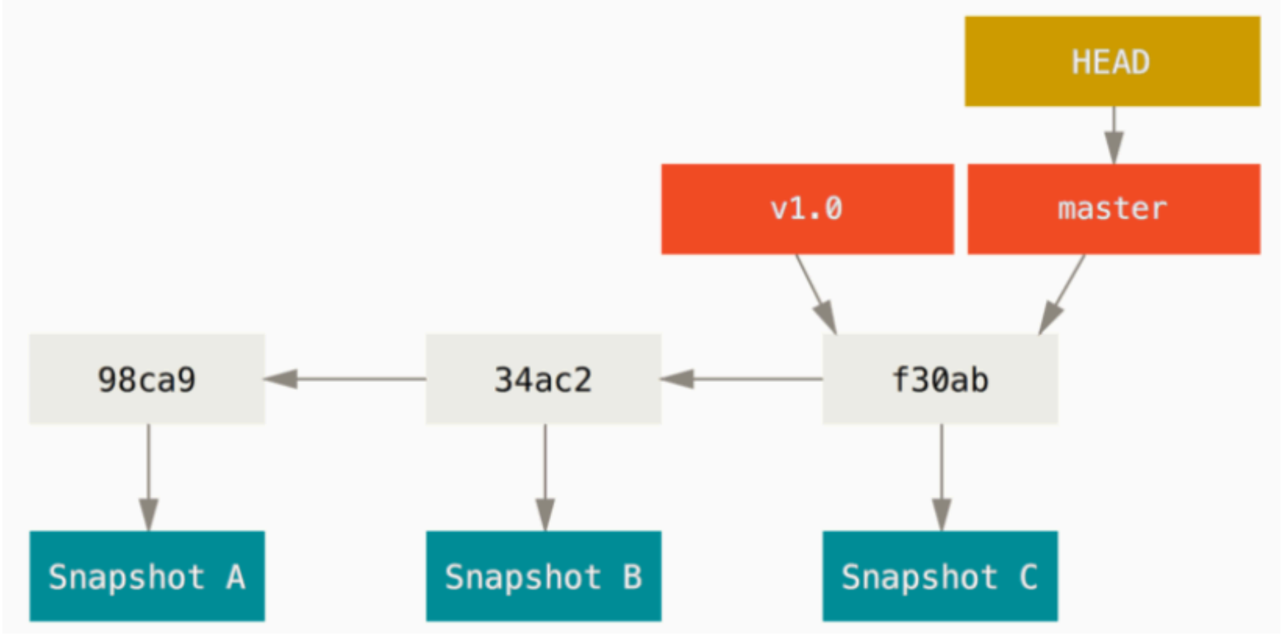
o Git armazena um objeto de commit que contém um ponteiro para o snapshot do conteúdo que você testou. Este objeto também contém o nome do autor e o e-mail, a mensagem que você digitou e ponteiros para o commit ou commits que vieram antes desse commit (seu pai ou pais): sem pai para o commit inicial, um pai para um commit normal, e vários pais para um commit que resulta de uma fusão de dois ou mais branches.



Introdução a Branchs

Um branch no Git é simplesmente um ponteiro móvel para um desses commits. O nome do branch padrão no Git é master. Conforme você começa a fazer commits, você recebe um branch master que aponta para o último commit que você fez. Cada vez que você faz um novo commit, ele avança automaticamente.





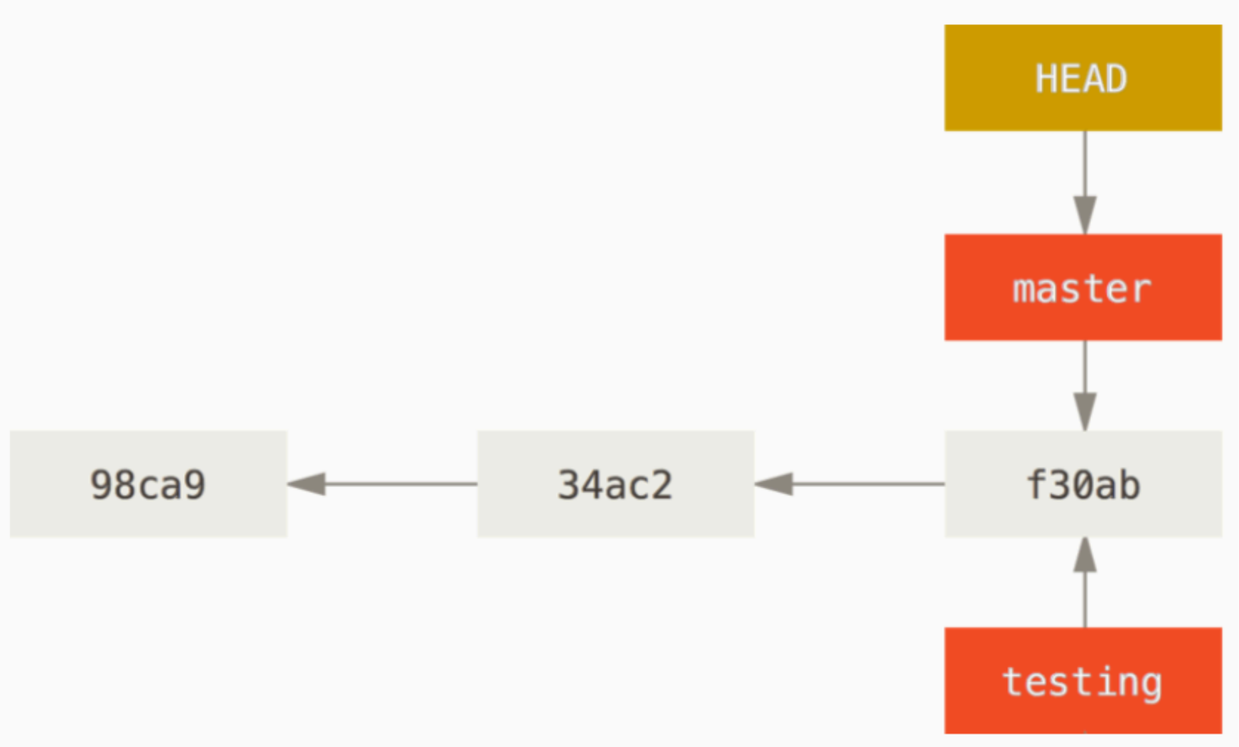
Criando Branch

O que acontece se você criar um novo branch? Bem, fazer isso cria um novo ponteiro para você mover.

git branch [nome-branch]

Como o Git sabe em qual branch você está atualmente? Ele mantém um ponteiro especial chamado HEAD. Digamos que você crie um novo branch chamado: testing. Você faz isso com o comando git branch :

```
1    git branch testing
```

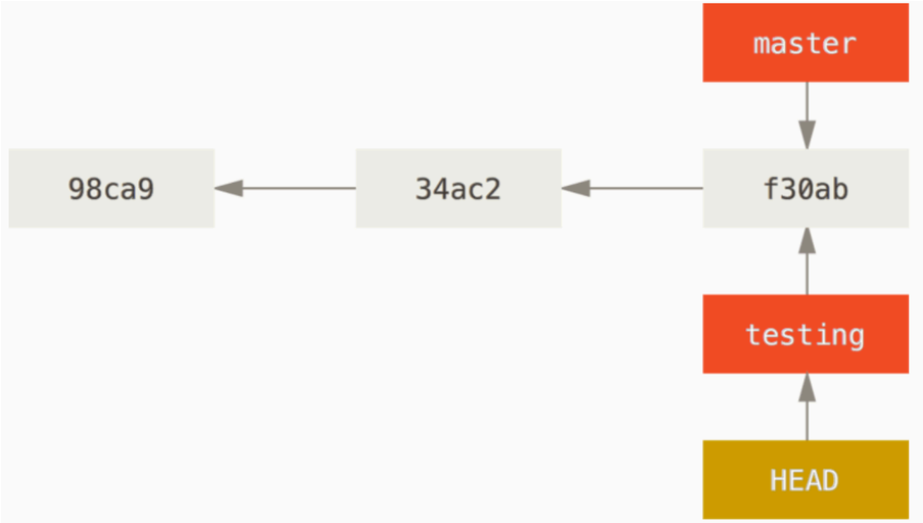


Alternando entre Branch

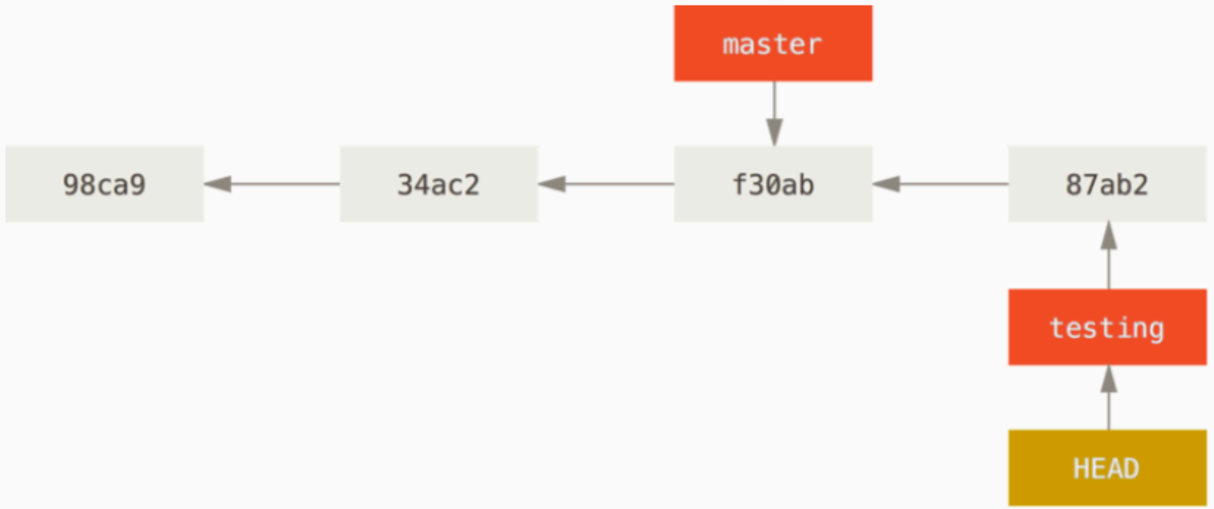
Para mudar para um branch existente, você executa o comando git checkout. Vamos mudar para o novo branch testing:

git checkout [nome-branch]

```
1    git checkout testing
```

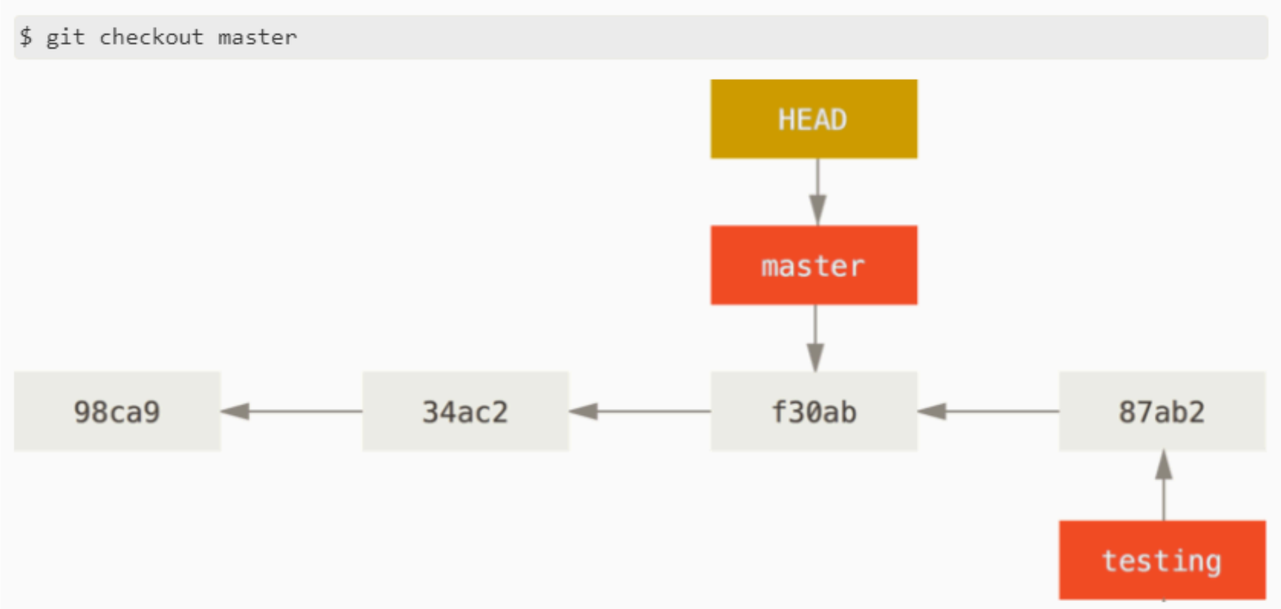



```
$ vim test.rb
$ git commit -a -m 'made a change'
```



Alternando entre Branch

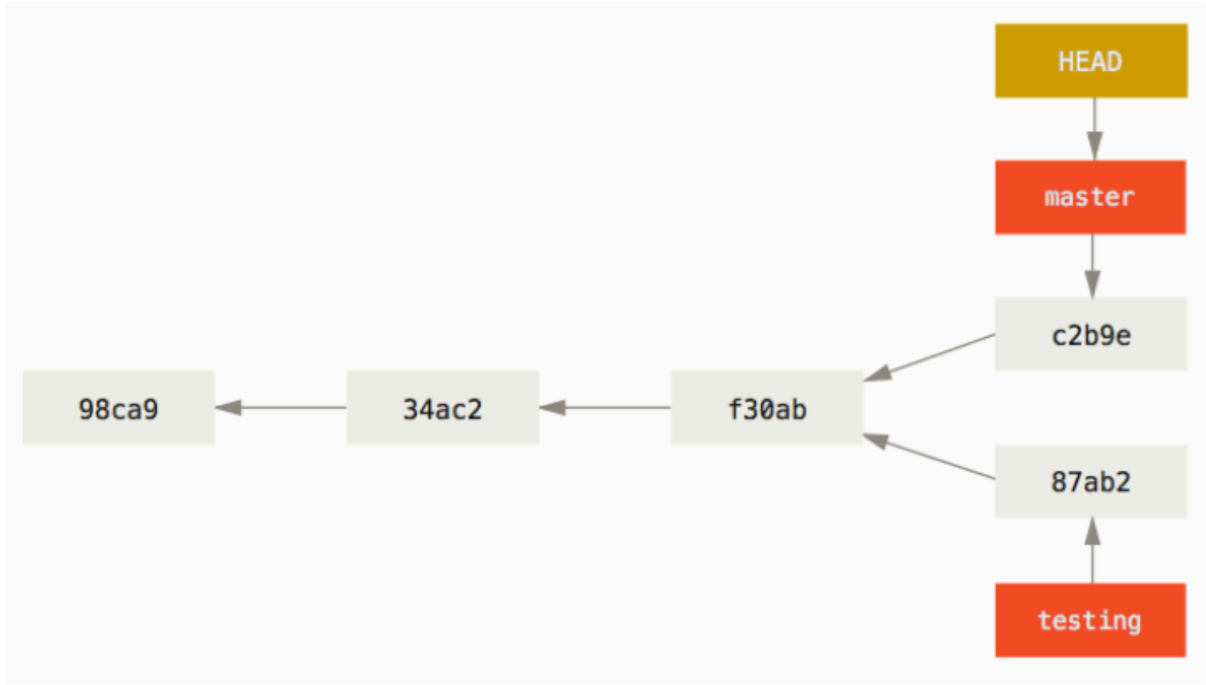
A troca de branches muda os arquivos em seu diretório de trabalho
É importante notar que quando você muda de branches no Git, os arquivos em seu diretório de trabalho mudam. Se você mudar para um branch mais antigo, seu diretório de trabalho será revertido para se parecer com a última vez que você fez commit naquele branch. Se o Git não puder fazer, ele não permitirá que você faça a troca.



Alternando entre Branch

Agora o histórico do seu projeto divergiu (consulte Histórico de diferenças). Você criou e mudou para um branch, fez algum trabalho nele e, em seguida, voltou para o seu branch principal e fez outro trabalho. Ambas as mudanças são isoladas em branches separados: você pode alternar entre os branches e mesclá-los quando estiver pronto.

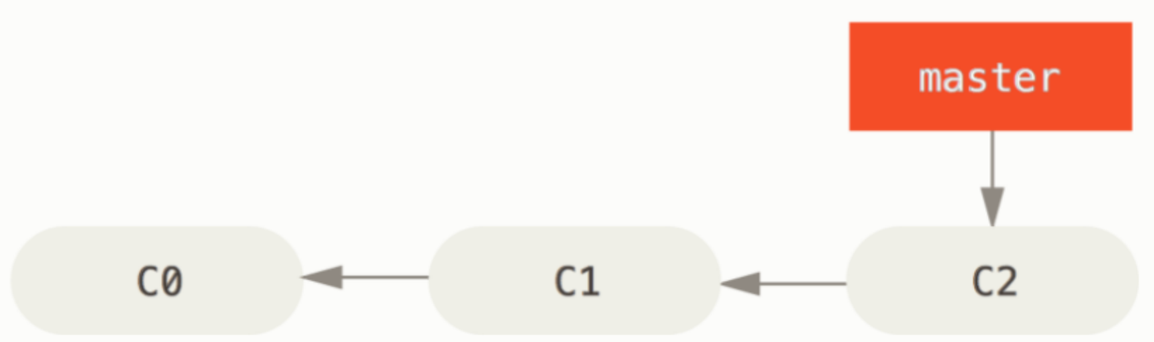
```
vim test.rb
git commit -a -m 'made other changes'
```




```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) made other changes
| * 87ab2 (testing) made a change
|/
* f30ab add feature #32 - ability to add new formats to the
* 34ac2 fixed bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```

Estudo de caso - Web Site

1. Trabalhar um pouco em um website.
2. Criar um branch para um nova história de usuário na qual você está trabalhando.
3. Trabalhar um pouco neste novo branch.

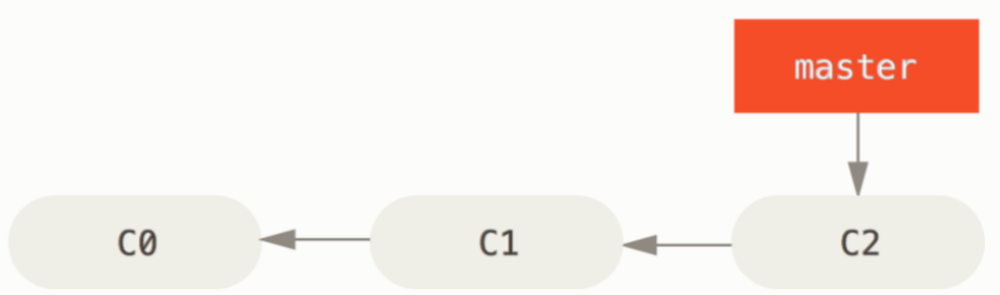


Nesse ponto, você vai receber uma mensagem dizendo que outro problema é crítico e você precisa fazer a correção. Você fará o seguinte:

1. Mudar para o seu branch de produção.
2. Criar um novo branch para fazer a correção.
3. Após testar, fazer o merge do branch de correção, e fazer push para produção.
4. Voltar para sua história de usuário original e continuar trabalhando.

Estudo de caso - Web Site

Projeto Web Site:

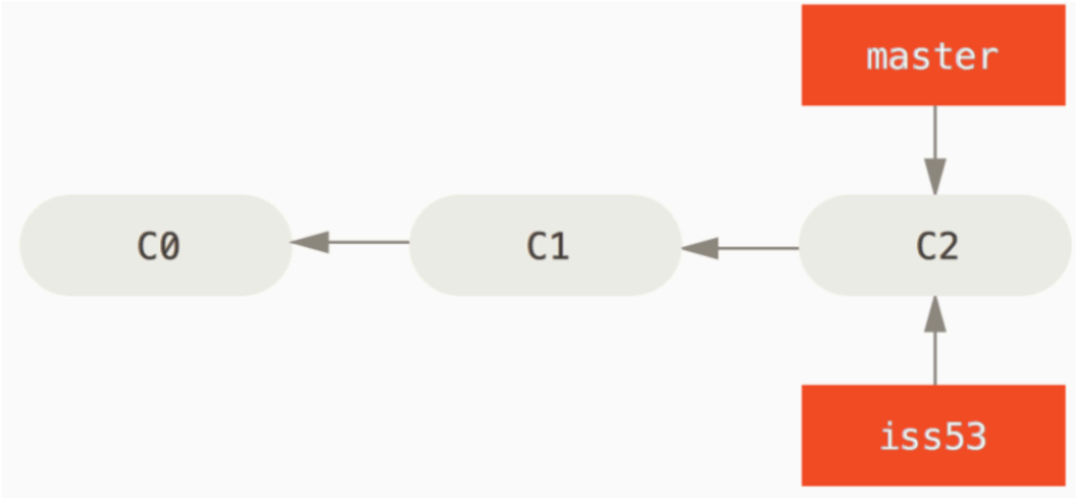



```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

```
$ git branch iss53
$ git checkout iss53
```

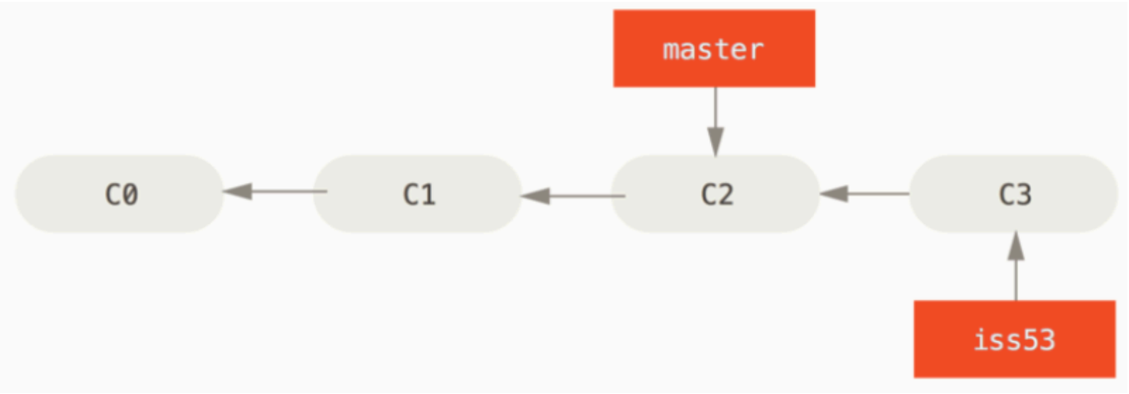
Você decidiu que você vai trabalhar no chamado #53 em qualquer que seja o sistema de gerenciamento de chamados que a sua empresa usa.

Para criar um novo branch e mudar para ele ao mesmo tempo, você pode executar o comando git checkout com o parâmetro -b:



Estudo de caso - Web Site

Você trabalha no seu website e adiciona alguns commits. Ao fazer isso, você move o branch iss53 para a frente, pois este é o branch que está selecionado, ou checked out(isto é, seu HEAD está apontando para ele):



Agora você recebe a ligação dizendo que há um problema com o site, e que você precisa corrigí-lo imediatamente. Com o Git, você não precisa enviar sua correção junto com as alterações do branch iss53 que já fez. Você também não precisa se esforçar muito para desfazer essas alterações antes de poder trabalhar na correção do erro em produção. Tudo o que você precisa fazer é voltar para o seu branch master.

Entretanto, antes de fazer isso, note que se seu diretório de trabalho ou stage possui alterações ainda não commitadas que conflitam com o branch que você quer usar, o Git não deixará que você troque de branch. O melhor é que seu estado de trabalho atual esteja limpo antes de trocar de branches.