Manual do Editor do Linux From Scratch

Versão 20210713

Gerard Beekmans (Versão Inicial)
Pierre Labastie (Versão Git)

Manual do Editor do Linux From Scratch: Versão 20210713

por Gerard Beekmans (Versão Inicial) e Pierre Labastie (Versão Git)

Resumo

Este manual ensina a você como editar adequadamente o Livro LFS.

Dedicação

Este livro é dedicado para todos(as) os(as) editores(as) do LFS, quem tem mantido o projeto do LFS vivo ao longo de todos esses anos.

Índice

Bem vindo(a) ao Manual do Editor do LFS

Este manual ensinará a você o que você precisa saber para editar adequadamente o Livro do LFS. Ele cobrirá alguns problemas básicos como acessar o Git via SSH e os comandos do Git que você usará. Ele também discutirá a sequência apropriada de fazer coisas como atualizar um pacote e como trabalhar com a base de dados de rastreio de defeito.

--
Gerard Beekmans
gerard@linuxfromscratch.org
Pierre Labastie
pierre@linuxfromscratch.org

Capítulo 1. Desenvolvimento do LFS

1.1. Introdução

O desenvolvimento do LFS toma lugar usando três sistemas principais. Primeiro, as listas de discussão <lfs-book@lists.linuxfromscratch.org>; <lfs-dev@lists.linuxfromscratch.org>; e (em menor extensão) <lfs-support@lists.linuxfromscratch.org>. Segundo, o sistema de rastreio de defeito Trac; e terceiro, o servidor Git onde o próprio livro é armazenado. Todos esses serviços são fornecidos pelo servidor **linuxfromscratch.org**, também conhecido como **rivendell.linuxfromscratch.org** ou, mais usualmente, **rivendell**. Esse único servidor fornece listas de discussão, hospedagem web, hospedagem Git, Trac e basicamente tudo o que nós usamos para trabalhar no projeto LFS.

O livro LFS é escrito usando Docbook-XML 4.5 e é dividido em um número de arquivos XML. Os diretórios na árvore do fonte representam os capítulos do livro e cada arquivo XML dentro de um diretório tipicamente contém o texto para uma seção dentro daquele capítulo. Essa estrutura é concebida para habilitar um(a) colaborador(a) a rapidamente localizar a área particular do livro que eles(as) querem editar.

O sistema de bilhetagem Trac para o LFS pode ser encontrado em *https://wiki.linuxfromscratch.org/lfs*. Para a finalidade de estar apto(a) para adicionar, remover e editar tíquetes, você precisa adicionar uma conta e ter certeza de que você está logado(a) quando desejar realizar uma tal ação. Você pode consultar e ler a base de dados do Trac sem se logar ou ter um(a) usuário(a) criado(a). Observe que todas as mensagens de tíquetes são copiadas para a lista de discussão <lfs-book@lists.linuxfromscratch.org> e que todos(as) os(as) editores(as) deveriam estar inscritos(as) nessa e na lista <lfs-dev@lists.linuxfromscratch.org> no mínimo.

Veja-se Usando o Trac para informação acerca de como ter um(a) usuário(a) criado(a) com as permissões de atualização apropriadas para o sistema Trac.

Finalmente, existe o servidor Git o qual será discutido nos capítulos seguintes.

1.2. Registro das mudanças

Esta é a versão 20210713 do Manual do(a) Editor(a) do Linux From Scratch, datada de 13 de julho de 2021.

Abaixo está uma lista das mudanças feitas desde a versão Git mais recente do livro, primeiro um sumário, então um registro detalhado.

- Adicionado:
  - Adicionadas folhas de estilo do LFS atuais.
  - Adicionada entrada para git log.
  - Adicionada entrada para git status.
  - Adicionada entrada para git push.
  - Adicionada entrada para git clone.
  - Adicionada entrada para git pull.
- Removido:
  - Removida a entrada para svn diff.
  - Removida a entrada para svn update.
  - Renomeadas todas as outras entradas svn xxx para git xxx.

- Removidas folhas de estilo antigas.

- Entradas de Registro das Mudanças:
  - 13 de julho de 2021
    - [thomas] Adicionado capítulo 8: Processos para construir os livros.
  - 29 de abril de 2021
    - [ken] Adicionado capítulo 7: Avisos de Segurança.
  - 31 de março de 2021
    - [bdubbs] Mudanças na redação.
  - 10 de janeiro de 2021
    - [pierre] capítulo01/introdução: Uns poucos consertos mais no nome do servidor e na versão do DocBook que nós estamos usando.
    - [pierre] folhas de estilo: remover anotações $LastChangedBy e $Date que são inúteis no git.
    - [pierre] Capítulo 4: adaptar "commit it!" para o git.
  - 09 de janeiro de 2021
    - [pierre] Capítulo 4: adaptar "check relevant files" para o git.
    - [pierre] Capítulo 4 atualizando o registro das mudanças: mostra como ter uma entrada de registro das mudanças correspondendo à somente uma revisão do livro.
    - [pierre] Capítulo 4: adaptar a introdução para o git.
    - [pierre] Capítulo 3: reescrever o comando push para o git.
  - 08 de janeiro de 2021
    - [bdubbs] Capítulo 3: copiar folhas de estilo e imagens a partir do livro lfs.
    - [pierre] Capítulo 3: reescrever o comando status para o git.
    - [pierre] Capítulos 2 e 3: remover referências à ramificação master e substituir com trunk.
    - [pierre] Capítulo 3: remover diff.xml. Adicionar log.xml e status.xml.
    - [pierre] Capítulo 3: Adicionar atributos xreflabel para comandos do git; reescrever a introdução usando xref.
    - [pierre] Capítulo 3: reescrever os comandos diff, merge, pull, push e rebase para o git.
  - 07 de janeiro de 2021
    - [pierre] Capítulo 3: adicionar push.xml.
    - [pierre] reescrever os comandos add, checkout, clone, commit, rm e mv para o git.
  - 05 de janeiro de 2021
    - [pierre] Capítulo 3: Reescrever a introdução para o git.
    - [pierre] Capítulo 3: Remover update.xml, moving.xml e delete.xml. Adicionar clone.xml, pull.xml, delmov.xml e rebase.xml (modelos).
    - [pierre] Folhas de estilo: Remover e usar aquelas do livro lfs.
    - [pierre] Makefile: Criar variáveis de forma que usar um esquema local seja mais fácil. Consertar alguns comandos obsoletos.

  - [pierre] Capítulo 2: Mudar nomes de arquivos, reescrever coisas de forma que os comandos se refiram ao git em vez do svn. Consertar nomes de host.
- 04 de janeiro de 2021
  - [pierre] introdução: Consertar nomes de host. Mudar de subversion para git. Mudar o estilo de codificação de XML.
  - [pierre] prólogo: reescrita leve. Mudar estilo de codificação de XML para o que nós temos nos outros livros.
  - [pierre] etiqueta Last_SVN_Version.

Parte I. Gerenciamento Básico do Fonte

Capítulo 2. Acesso Git

2.1. Introdução

O servidor Git rivendell fornece repositórios para todos os projetos *LFS (e alguns outros). O repositório no qual nós estamos interessados para edição LFS é (não surpreendentemente) o repositório LFS. Uma lista completa dos módulos os quais estão disponíveis pode ser encontrada usando a interface do navegador do fonte do Trac em *https://wiki.linuxfromscratch.org/lfs/browser*.

Existem dois tipos de acesso Git para a árvore do LFS. Primeiro, existe o acesso somente leitura anônimo o qual qualquer pessoa pode usar. Segundo, existe o acesso leitura e escrita concedido para editores(as) ativos(as).

> **Observação para Usuários(as) do Subversion**
>
> Subversion é um sistema de controle de versão centralizado, o que significa que existe um repositório único no servidor, e que usuários(as) podem *copiar* esse repositório para um diretório de trabalho na máquina local deles(as). Os(As) usuários(as) com acesso de escrita podem então *comitar* as modificações deles(as) para o repositório no servidor.
>
> Git é um sistema de controle de versão distribuído, o que significa que todos(as) os(as) usuários(as) tem os repositórios próprios deles(as). O repositório no servidor não é nada mais que os outros, exceto que um acesso público é concedido até ele. A ação de copiar o repositório a partir do servidor para a máquina local é chamada de *clonagem*. Copiar e comitar assim ocorre somente localmente. A ação de sincronizar o repositório público com o local é chamada de *pull* (público para local) ou *push* (local para público para usuários(as) com acesso de escrita).

2.2. Acesso Anônimo

Para obter acesso anônimo, simplesmente use o comando seguinte (observe que isto assume que você está usando o bash ou um shell similar):

```
git clone git://git.linuxfromscratch.org/lfs.git lfs-git
```

Isso copiará o repositório público para um subdiretório chamado `.git` no diretório `lfs-git` e então copiará a ramificação padrão para esse diretório. Ele [o comando] também configurará o repositório local para rastrear a ramificação padrão do repositório público, de forma que você pode atualizar seu repositório local simplesmente executando (depois de mudar para o diretório `lfs-git`):

```
git pull
```

Observe que para o livro LFS, a ramificação padrão é chamada de *trunk*.

2.3. Acesso SSH Git (para editores(as))

Para editores(as), o acesso é ligeiramente mais complicado. Você primeiro precisa gerar um par de chaves ssh. Para gerar as chaves execute:

```
ssh-keygen -t ed25519
```

Nota

Dos(as) Editores(as) não é exigido ter uma conta no servidor, porém ela [a conta] talvez seja útil.

Quando perguntado(a) onde salvar elas, provavelmente é melhor deixá-las em .ssh (como `id_ed25519` e `id_ed25519.pub`). Quando perguntado(a) por uma frase senha, apenas pressione enter, a menos que você queira dar a frase *a cada* vez que você sincronizar para o servidor. Entretanto, dado que a mesma frase senha será usada quando você se logar no *rivendell* via ssh, talvez seja aconselhável ter alguma segurança no lugar.

Tendo gerado suas chaves, envie o `~/.ssh/id_ed25519.pub` para um(a) administrador(a) do LFS para a finalidade de ter ela adicionada ao `~git/.ssh/authorized_keys` *no rivendell*. Se você estará obtendo uma conta de login no servidor, [então] o(a) administrador(a) usará a mesma chave para permitir a você o acesso ssh.

Sua cópia local do `id_ed25519` e `id_ed25519.pub` deveria permanecer intocado por esse processo.

Uma vez que esse processo esteja completo, tente copiar a revisão mais recente do livro LFS executando (a partir da sua máquina local):

```
git clone git@git.linuxfromscratch.org:lfs.git lfsbook
```

Se tudo for bem, [então] você baixará uma cópia do repositório atual para o `lfsbook/.git` e você copiará a ramificação padrão, a qual é chamada de *trunk*. Você também terá acesso de escrita, de forma que, de agora em diante, seja extra cuidadoso(sa). Observe que *nenhuma* mudança será feita até que você emita um comando **`git push`**.

O acima está bom para obter a ramificação padrão, porém você talvez queira trabalhar em uma ramificação pública diferente. Para configurar uma ramificação local que rastreie uma ramificação pública chamada *new_branch*, apenas emita:

```
git checkout --track origin/<new_branch>
```

Observação para Usuários(as) do Subversion

Contrariamente ao Subversion, onde você precisa copiar uma cópia funcional nova da ramificação remota, com o Git quase nada é baixado. O diretório local agora reflete a ramificação nova, porém o diretório da ramificação antiga não mais existe.

Se você precisar trabalhar na ramificação padrão chamada *trunk*, [então] apenas comite suas mudanças (se alguma) para a ramificação nova (o commit é local), e mude de volta para trunk com:

```
git checkout trunk
```

Dado que a ramificação nova já está configurada, se você quiser novamente trabalhar na ramificação nova, apenas faça:

```
git checkout <new_branch>
```

Como com o acesso anônimo, você pode atualizar seu repositório local simplesmente emitindo um **cd** para o diretório LFS e executando:

```
git pull
```

Capítulo 3. Comandos Básicos do Git

3.1. Introdução

Vamos nos tornar familiarizados(as) com o conjunto básico de comandos os quais todos(as) os(as) editores(as) usarão em uma quase sempre base diária. Existem muitas mais opções disponíveis que as listadas aqui, de forma que você desejará ler a documentação do Git (*Pro Git*) em algum ponto. **git `help`** também fornece uma referência rápida útil para os comandos do Git. As vezes **git status** mostra um par de comandos que um(a) usuário(a) poderia querer executar no atual estado do repositório.

> **Nota para os(as) Usuários(as) do Subversion**
>
> Como já foi dito, o repositório Git está na máquina local. Mais precisamente, o diretório mantém uma cópia funcional de alguma ramificação do repositório, e o próprio repositório está armazenado sob o diretório `.git` juntamente com várias definições de configuração. O comando git checkout é usado para alternar entre ramificações.
>
> Além do repositório local, o git mantém uma área de teste que é conhecida como o índice. O comando git add povoa essa área de teste e git commit é usado para transferir o conteúdo do índice para o repositório. Isso permite um bom controle do que é comitado, porém muitos atalhos podem ser usados. Ao contrário do subversion, **git add** não é usado apenas para adicionar arquivos, mas também para armazenar modificações para os arquivos existentes.

3.2. git clone

`git clone`. Esse comando é usado para copiar um repositório público para a máquina local e fazer um checkout da ramificação padrão (*trunk* para o LFS). Você deveria precisar fazer isso somente uma vez. Várias opções permitem a criação de um repositório "raso" que não contém o histórico completo ou todas as ramificações. Veja-se **git help clone**.

Exemplos

Para clonar o repositório do LFS com acesso somente leitura:

```
git clone git://git.linuxfromscratch.org/lfs.git lfsbook
```

Para clonar com acesso leitura/escrita:

```
git clone @git.linuxfromscratch.org:lfs.git lfsbook
```

3.3. git pull

`git pull`. Esse comando sincroniza seu repositório local. Se você tiver feito mudanças locais, [então] o Git tentará mesclar quaisquer mudanças no servidor com as mudanças que você tiver comitado *na sua máquina*. Se as mudanças no servidor sobrepuserem com as mudanças locais não comitadas, [então] a mesclagem é cancelada, e a árvore de trabalho permanecerá intacta.

Diferente de **svn `up`**, quando o Git mescla seus commits locais com as mudanças no servidor, ele produzirá um commit de mesclagem. Muitos commits de mesclagem talvez causem confusão no histórico. Como uma alternativa, uma pessoa pode passar `--rebase` para **git `pull`**, dizendo ao Git para fazer o rebase de seus commits locais nas mudanças no servidor. Leia-se git rebase para os detalhes acerca de rebasing.

Para tornar um histórico limpo, os(as) editores(as) deveriam usar rebasing em vez de mesclagem para sincronizar as mudanças no servidor se você tiver alguns commits locais. Para tornar `--rebase` o padrão para **git pull**, emita: **git config pull.rebase true**.

Você deveria sempre fazer um **git `pull`** manual antes de tentar enviar mudanças para a finalidade de assegurar que não existem conflitos com mudanças que tenham sido feitas desde que você iniciou seu trabalho. Observe que **git `push`** advertirá você se existir um conflito e você se esquecer se realizar um **git `pull`**.

3.4. git add

`git add [`*`arquivos-modificados`*`]`. Isso prepara as modificações para todos os `arquivos-modificados` (incluindo arquivos criados recentemente) para o índice. Se você especificar um diretório em `arquivos-modificados`, [então] todos os arquivos modificados nesse diretório ou nos subdiretórios dele serão preparados. A mudança não está no repositório local até que você faça um **`git commit`**. Nós tocaremos mais nisso conforme continuemos no capítulo.

Esse comando pode ser usado de uma maneira muito flexível usando-se a opção `-p`. Você pode selecionar cada pedaço do diff que está preparado dentro do índice. Isso permite testar uma mudança completa e então divide os commits em pequenos pedaços "atômicos" que são mais fáceis para revisar e entender.

3.5. git rm e git mv

`git rm`. Isso remove um arquivo e deixa o Git saber acerca dele. Várias opções permitem selecionar quando o arquivo é deletado somente no índice ou em ambos no índice e no diretório de trabalho. Observe que você precisa comitar (e possivelmente realizar um push) para a remoção aparecer no repositório.

`git mv`. Isso renomeia ou move um arquivo e deixa o Git saber acerca dele. Novamente, você precisa comitar (e possivelmente realizar um push) para a mudança aparecer no repositório.

3.6. git status

`git status`. Esse provavelmente é o comando que você usará mais frequentemente. Ele imprime os arquivos que foram modificados no diretório de trabalho, e que não estão no índice; ou que estão no índice e não comitados ainda. Ele também mostra o estado do repositório local respectivo ao remoto.

Uma característica muito útil é a de que ele mostra uma lista de comandos que podem ser usados no contexto atual.

3.7. git commit

git `commit`. Esse comando armazena suas mudanças para o repositório *local*. Normalmente ele comita mudanças preparadas no índice, porém várias opções permitem que você contorne a preparação. Observe que nada é mudado no servidor até que você emita **`git push`**. As opções `-m` e `-F` podem ser usadas para passar uma mensagem de registro para o comando. Se você não especificar uma opção *-m "MENSAGEM"* ou *-F "Nomearquivo"*, [então] o Git abrirá o editor padrão e solicitará que você digite uma mensagem de registro. O editor padrão é especificado pelas variáveis de ambiente, `GIT_EDITOR`; ou `VISUAL`; ou `EDITOR` (verificadas nessa ordem); ou por um parâmetro de configuração em `~/.git/config`. Por exemplo, para configurar seu editor padrão para o vim, execute:

```
export GIT_EDITOR=vim
```

ou:

```
git config --global core.editor vim
```

Não use mensagens de registro vazias (veja-se posteriormente neste documento acerca da política que as rege).

3.8. git log

`git log`. Gera o registro do commit da ramificação atualmente no diretório de trabalho. Várias opções permitem personalizar o nível da informação dada para cada commit. Os(As) Editores(as) são fortemente aconselhados(as) a executar esse comando antes de realizar um push para o repositório remoto ou mesclagem. Se o histórico for muito complicado, [então] talvez seja necessário executar git rebase para a finalidade de simplificá-lo. Seus(uas) coeditores(as) odiarão você se você mudar o histórico depois de realizar um push dele.

> **Nota para Usuários(as) do Subversion**
>
> A saída é automaticamente passada para um paginador, de forma que não é necessário canalizá-la para o **less**. Além disso, nenhum acesso ao repositório remoto é necessário, de forma que usualmente é muito mais rápido obter um registro longo.

3.9. git push

`git push`. Esse comando sincroniza o repositório público no servidor com o seu local. Você deveria ter certeza de que você quer publicar todos os commits que você tenha feito. As vezes, talvez seja útil executar **`git rebase -i`** para a finalidade de simplificar o histórico. Também, para prevenir conflitos quando se realizar um push, é melhor primeiro usar **`git pull`** antes de realizar um push e resolver quaisquer conflitos localmente.

3.10. git checkout

`git checkout`. Esse comando é usado para tornar uma ramificação do repositório local visível no diretório de trabalho.

> **Nota para Usuários(as) do Subversion**
>
> **`git checkout`** as vezes é confuso para usuários(as) do subversion, pois ele é muito diferente do **`svn checkout`**. Aquele está mais próximo do **`svn switch`**. Quando usar o Git, você se encontrará criando ramificações bastante frequentemente, e usar **`git checkout`** para alternar entre elas é um comando bastante comum.

3.11. git merge

`git merge`. Isso é útil para aplicar mudanças originárias de outra ramificação (ou várias ramificações) para a atual. Por padrão, as mudanças resultantes são comitadas. Isso pode ser evitado com a opção `--no-commit`, a qual dá a você uma chance para inspecionar e ajustar ainda mais o resultado da mesclagem antes de comitar.

Em alguns casos, a mesclagem resulta em conflitos que tem de ser resolvidos manualmente. Observe que as edições deveriam mesclar apenas para uma ramificação limpa, isto é, após comitar todas as mudanças.

> **Nota para Usuários(as) do Subversion**
>
> Ao contrário do subversion, não existe opção `--dry-run` para **`git merge`**. Porém, um repositório sempre pode ser restaurado para um estado anterior com **`git reset --hard <commit-id>`**.

3.12. git rebase

`git rebase`. Isso é útil para ter a ramificação atual iniciada a partir de outro ponto no histórico. Por exemplo, se você tiver:

```
                            B1---B2---B3    my_branch
                           /
                          A---C1---C2       trunk
```

realizar um rebase de my_branch para a trunk produzirá:

```
                            B'1---B'2---B'3 my_branch
                           /
                     A---C1---C2            trunk
```

onde o estado alcançado depois do commit B'3 é o mesmo que era em B3.

Porém, o comando mais útil é o **`git rebase -i`** que permite a você reescrever o histórico quando você tiver feito commits confusos.

Capítulo 4. Comitando Mudanças - Política

4.1. Introdução

Aqui está uma lista do sumário das coisas a fazer antes de comitar mudanças:

- Teste as instruções que você está adicionando.
- Atualize `general.ent` com a nova data.
- Atualize `chapter01/changelog.xml`.
- Verifique se todos os arquivos relevantes foram adicionados com **`git add`** ou removidos com **`git rm`**.
- Verifique a validade do XML.
- Verifique se o livro renderiza adequadamente.
- Comite; volte ao início se você tiver outro conjunto de mudanças que você desejar fazer.
- Realize um push assim que você terminar com todas as suas mudanças.
- Atualize o Trac para refletir as mudanças.

Você normalmente deveria restringir um commit a um conjunto de mudanças, de forma que se você estiver atualizando as versões de três pacotes, [então] usualmente você fará isso em três commits. Claramente, o que forma um "conjunto" é uma questão para seu julgamento. Sempre pense que um conjunto terá que ser mesclado para (ao menos) a ramificação multilib. Realizar um push não é obrigatório depois de cada mudança se você planeja breve fazer outra. Mais uma vez, os(as) editores(as) deveriam executar **`git pull`** antes de realizar um push para a finalidade de resolver qualquer conflito localmente.

Os(As) editores(as) do livro LFS preferem que as mudanças para a data em general.ent sejam feitas como parte de uma mudança *real*.

4.2. Teste as instruções

Isso talvez pareça *realmente* óbvio, porém é muito fácil cometer um erro de digitação em mudanças de comando de instalação que fazem com que a instalação quebre. Todos nós cometemos; você provavelmente também cometerá eventualmente. Entretanto, verifique duplamente para minimizar a chance.

4.3. Atualizando o general.ent

Os seguintes elementos deveriam ser descomentados e atualizados no arquivo `general.ent` quando um lançamento (incluindo lançamentos -rc) for feito e as duas linhas acerca do `version.ent` deveriam ser comentadas:

```
<!ENTITY version         "10.2-rc1">
<!ENTITY versiond        "10.2-rc1">
<!ENTITY releasedate     "August 26th, 2021">
<!ENTITY copyrightdate   "1999-2021">
```

Para instantâneos de desenvolvimento, essas entidades são geradas automaticamente a partir da informação de commit do Git. Assim, o `general.ent` não precisa ser atualizado.

4.4. Atualize o chapter01/changelog.xml

As atualizações do registro das mudanças *sempre* deveriam ser fornecidas, com a exceção dos pequenos consertos dos erros de digitação. Você não precisa adicionar "consertado um pequeno erro de digitação em XXX" ao registro das mudanças, do contrário ele crescerá demais.

As atualizações do registro das mudanças precisam estar no seguinte formato:

```
<listitem>
  <para>Dia, Mês e Ano</para>
  <itemizedlist>
    <listitem>
      <para>[nome de usuário(a)] - O que você mudou.</para>
    </listitem>
<!-- caso se aplique somente para a versão sysv (adapte para systemd) -->
    <listitem revision="sysv">
      <para>[nome de usuário(a)] - O que você mudou somente para a versão sysv.</p
    </listitem>
    <listitem>
      <para>[nome de usuário(a)] - Entrada anterior do registro
      das mudanças originária do mesmo dia, por você ou outro(a)
      editor(a).</para>
    </listitem>
  <itemizedlist>
</listitem>
```

Exemplo:

```
<listitem>
  <para>03 de março de 2006</para>
  <itemizedlist>
    <listitem>
      <para>[renodr] - Atualização para o attr-2.5.1. Conserta
      <ulink url="&lfs-ticket-root;4833">#4833</ulink>.</para>
    </listitem>
  </itemizedlist>
</listitem>
```

As entradas do registro das mudanças sempre estão no topo da entrada do registro das mudanças adicionada anteriormente.

4.5. Verifique todos os Arquivos Relevantes

Se você estiver adicionando arquivos, [então] você precisa executar um comando **git add** em cada um deles. Quando você remover arquivos, você sempre deveria fazer isso com **git rm**. Mover ou renomear arquivos é feito com **git mv**. Adicionar um diretório e todos os filhos dele é feito com **git add <nome_diretório>**.

Se você pensa que está pronto para [efetivar o] commit, [então] execute **git status** para ver o estado da cópia de trabalho. O processo normal é primeiro [executar] **git add** [para adicionar] os arquivos que estão modificados ao índice, então executar **git commit** para armazenar o conjunto de mudanças no repositório local. A saída do git status se parece com isto:

```
On branch trunk
Your branch is up to date with 'origin/trunk'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README

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:   TODO
```

onde a [mensagem] "Changes to be committed" aparece em verde, enquanto que a [mensagem] "Changes not staged for commit" aparece em vermelho.

4.6. Comentando alguma coisa

As vezes, uma mudança tornará alguma coisa não mais apropriada, porém você pensa que ela seria necessária no futuro. Se você comentá-la da maneira normal e iniciar o comentário com um marcador `FIXME:`, [então] isso será realçado sempre que o arquivo for editado, ao menos se você usar o vim.

Se você encontrar um *FIXME:* em um comentário, [então] não assuma automaticamente que ele possa ser deletado porque as instruções seguintes agora tenham sido comentadas.

Observe que não existe exemplo aqui - a renderização do html retira os comentários e uma caixa vazia parece idiota. Como sempre, renderize a sua cópia local depois que você comentar alguma coisa, de forma que você possa ver o resultado!

4.7. Consigne!

Tão logo você tiver certeza de que tudo renderiza e que sabe quais arquivos deseja comitar, você está pronto(a). Primeiro comite localmente:

- Procedimento normal: permite comitar somente o que você quer
  - Execute **git add *<arquivo1> <arquivo2> ...***
  - Depois que verificar que todos foram organizados corretamente (use **git status** e (ou) **git diff --staged**), execute **git commit** e preencha a mensagem de registro na janela do editor. Ou, para uma mensagem curta de registro, execute **git commit -m "*mensagem de registro*"**.
- Atalho para comitar todos os arquivos modificados de uma só vez:
  - Execute **git commit -a** e escreva a mensagem de registro na janela do editor. Como o acima, você pode usar a opção `-m` para mensagens curtas de registro.

Agora envie seu trabalho: **git push**. Um registro de transação de todos os commits que compõem o envio será enviado por correio eletrônico para a lista de discussão `<lfs-book@lists.linuxfromscratch.org>`, de forma que outros(as) editores(as) possam ver imediatamente o que você fez. As mensagens de correio eletrônico de commits contém algumas informações básicas (registro, mudanças para quais arquivos), incluindo uma saída no formato *diff -u*.

Acerca das mensagens de registro: Devido à maneira como o **git log** exibe mensagens de registro, por favor, mantenha o tamanho de cada linha abaixo de setenta e dois (72) caracteres. Uma boa mensagem de commit é composta de um sumário curto (não mais que cinquenta caracteres); uma linha em branco; então detalhes, razões, etc. em linhas de menos que setenta e dois (72) caracteres. As mensagens de registro *nunca* deveriam estar vazias. Ainda que a mensagem seja apenas 'pequeno conserto de erro de digitação', isso servirá. Outras mensagens usuais são 'atualização para o pacote-x.y.z' ou 'consertadas as instruções de instalação do pacote foo'. Essa última usualmente também precisa de linhas de esclarecimento conforme dito acima.

Exemplo: Se você modificou o `general.ent`; o registro das mudanças; e consertou um erro de digitação na instalação do bash do capítulo 6, [então] o fluxo de trabalho poderia ser:

```
git add general.ent chapter01/changelog.xml chapter06/bash.xml
git commit -m 'consertado um erro de digitação no bash do capítulo 6'
```

4.8. Atualize o Trac

A parte final de atualizar o livro é atualizar o bilhete. Isso usualmente é tão fácil quanto ir para a wiki (*https://wiki.linuxfromscratch.org/lfs*); ir ao bilhete e escolher resolver, mudando a resolução para consertado.

Capítulo 5. Usando o Trac

5.1. Introdução

Este capítulo cobre as coisas que você precisa fazer quando estiver usando o Trac para registrar defeitos novos no sistema e consertando/atualizando defeitos excepcionais.

Nós assumimos que você já se logou no Trac antes de fazer qualquer coisa delineada nas seções seguintes.

5.2. Adicionando comentários

- Vá para o bilhete que você quer adicionar comentários
- Adicione sua informação adicional na caixa *Comment*.
- Conforme o último passo, clique no botão *Submit changes* para comitar suas mudanças para a base de dados. Um registro disso será enviado para a lista de discussão lfs-book.

5.3. Registre um defeito novo

Você encontrou um defeito ou outra pessoa encontrou um defeito e você decidiu adicioná-lo ao Trac. Ou você quer registrar uma solicitação para uma característica que alguma pessoa tenha pedido ou uma tarefa de algum outro tipo.

- Vá para *https://wiki.linuxfromscratch.org/lfs/newticket*
- Selecione o tipo de tíquete (defeito por bugs; ou aprimoramento; ou tarefa).
- O *Sumário curto* e a *Descrição completa* são obrigatórios.
- Selecione a *Prioridade*. Use seu próprio julgamento acerca do quão importante é consertar esse defeito. Se você não tiver certeza, [então] apenas deixe o padrão. As prioridades periodicamente são reavaliadas e mudadas de qualquer maneira.
- Selecione o *Componente* - as únicas opções são Book ou Bootscripts.
- Selecione a *Severidade* conforme apropriado, algo de trivial até crítica ou bloqueadora.
- Se você não quiser vincular-se imediatamente, [então] deixe o campo *Assign To* em branco. Será vinculado ao `<lfs-book@linuxfromscratch.org>` até que um(a) editor(a) mude isso para ele(a) próprio(a).
- Se você desejar, [então] selecione um *Marco histórico*: ou a próxima versão; ou futuro.
- Selecione a *Versão* apropriada. Quase sempre você escolherá a versão *SVN*. Não faz sentido relatar um defeito contra uma versão antiga se ela não mais está no Subversion. Se ela estiver, então a SVN é mais nova que uma versão estável do livro lançada anteriormente. As versões estão lá basicamente somente para pessoas que não editam o livro e quem quer relatar um defeito contra a versão do livro que tem.
- Preencha os campos *Keywords* e *Cc:* se você desejar.
- Se você estiver usando qualquer formatação no campo Descrição completa, [então] você talvez deseje clicar no botão *Preview*.
- Como o último passo, clique no botão *Submit ticket* para comitar as suas mudanças para a base de dados. Um registro disso será enviado para a lista de discussão `<lfs-book@linuxfromscratch.org>`.

5.4. Vinculando Tíquetes

Quando você estiver pronto(a) para começar a trabalhar em um problema que esteja no Trac, o passo um é vinculá-lo a você próprio(a). Isso informa a todos(as) os(as) editores(as) que você está trabalhando no problema.

- Vá para o problema sobre o qual você quer trabalhar.
- Selecione o botão de rádio *Accept ticket*.
- Clique no botão *Submit changes*.
- Enquanto trabalhar no problema, as vezes você talvez queira adicionar comentários a ele. Sinta livre para fazer isso.

5.5. Sinalize um Problema como Consertado

Tão logo você tenha consertado um problema e tenha comitado as mudanças relevantes no repositório Subversion, você precisa sinalizar o tíquete como consertado no Trac.

- Vá para o tíquete que você consertou e o qual está vinculado a você.
- Selecione o botão de rádio “resolve as”. Então selecione a resolução adequada. Usualmente você selecionará *fixed* aqui, porém existem ocasiões quando você seleciona outros, como casos onde um problema relatado é (*invalid*); ou nós sabemos acerca dele, porém não o consertaríamos (*wontfix*); e assim por diante.
- Clique no botão *Submit changes* para comitar as mudanças para a base de dados.

Capítulo 6. Procedimentos de Atualização de Pacote

6.1. Introdução

Atualizar um pacote no Livro LFS consiste do seguinte:

- Atualizar o Trac para anunciar a disponibilidade de uma versão nova de um pacote.
- Atualizar o livro para refletir a versão nova do pacote.

6.2. Atualizando o Trac

Quando um pacote novo é lançado pelos(as) mantenedores(as) dele, nós sinalizamos isso no Trac. Você não precisa atualizar o livro pouco depois que um pacote for lançado, porém ao menos anunciá-lo.

- Vá para a página de consulta do Trac em *https://wiki.linuxfromscratch.org/lfs/report*
- Escolha Active Tickets e procure pelo nome do pacote.
- Crie um tíquete novo para a versão nova (conforme discutido anteriormente sob registrar um defeito novo). Sinalize-o como um aprimoramento. Adicione qualquer informação interessante, por exemplo, a partir das notas de lançamento. Quando você tiver colocado toda a informação que você desejar registrar, clique em *Submit ticket* na maneira normal.
- Se existir um tíquete aberto existente para uma versão anterior do pacote, [então] resolva isso como *wontfix* com uma observação de que ele foi substituído pelo tíquete novo. Alternativamente, selecione *Modify Ticket* e atualize o Summary e a Description para refletir a versão nova.

6.3. Atualize o Livro

Quando você for atualizar o livro para adicionar um pacote novo a ele, aqui está como você faz isso:

- Vincule-se ao tíquete para esse pacote novo.
- Construa um sistema LFS de teste para ter a certeza de que esse pacote compila adequadamente em um ambiente LFS. Não use o seu ambiente regular da estação de trabalho - ele talvez esteja suficientemente diferente para afetar os procedimentos de construção. Um pacote talvez dependa de alguma coisa que você tenha instalado, porém a qual não vem com o Livro LFS. Tenha em mente também que alguns pacotes são usados mais que uma vez no processo de construção do LFS.
- Atualize as instruções de instalação no livro, se necessário.
- Quando o pacote compilar adequadamente e o pacote também funcionar (não der falha de segmentação ou exibir outros erros quando se tentar executar aplicativos a partir dele), então abra o `packages.ent` em um editor.
- Encontre a entidade *package-version* e atualize o valor dela para a versão nova.
- Atualize a lista dos arquivos instalados pela versão nova do pacote. Isso pode ser obtido trivialmente usando o **find** imediatamente antes de e imediatamente depois que instalar a versão nova e comparar com o **diff** as duas saídas geradas.

6.4. Renderize o Livro e Verifique os Links

Agora que você está pronto(a), tenha a certeza de que o livro renderiza adequadamente e verifique os links no *Capítulo 3* para ter a certeza de que os links do pacote novo são válidos.

Se isso der certo, então você está pronto(a) com a atualização do pacote. Para mais informação acerca de como construir os arquivos HTML a partir dos fontes do livro, consulte o *Capítulo 8*.

Comite as suas mudanças para o repositório do Git.

Capítulo 7. Avisos de Segurança

7.1. Introdução

Em ambos LFS e BLFS nós tentamos avisar ao(à)s nossos(as) usuários(as) das vulnerabilidades de segurança. As vezes, vulnerabilidades são mencionadas em um anúncio de lançamento; outras vezes elas são divulgadas posteriormente.

Os avisos atuais apontam para os livros de desenvolvimento, com os avisos anteriores apontando para os livros numerados (em vez de 'estáveis'). Provavelmente você precisará criar alguns links simbólicos onde quer que você renderize os livros para que ambas as versões dos livros LFS e BLFS atuais apontem para suas cópias locais, de forma que você possa verificá-las antes de enviar. Se você tiver renderizações locais dos livros lançados atualmente, [então] também poderá criar um link para eles. Como `consolidated.html` está localizado no livro BLFS, você precisará criar um link simbólico especial para chegar a ele a partir dos seus avisos locais do LFS.

Por razões históricas, os links no rivendell apontam para 'svn' e 'systemd' para os livros atuais.

Normalmente, emitiremos um aviso de segurança assim que tivermos uma correção testada no livro apropriado (LFS ou BLFS), ou seja, uma versão mais recente ou um patch. No entanto, as vezes, uma vulnerabilidade permanece aberta por muito tempo e nós talvez optemos por sugerir uma solução alternativa, como "não use essa característica do pacote".

Nós tentamos fornecer detalhes suficientes para habilitar os(as) usuários(as) a decidirem se precisam atualizar o mais rápido possível, incluindo uma avaliação de vulnerabilidade e quaisquer detalhes que possamos encontrar. Se o(a) desenvolvedor(a) fornecer uma avaliação da gravidade, [então] normalmente usaremos essa. Do contrário, precisamos procurar por aí por detalhes. Se os detalhes de um CVE estiverem visíveis publicamente, [então] existiria uma classificação em *https://nvd.nist.gov/vuln/search*, apesar de, as vezes, esse sítio rotular os problemas como Moderados quando uma classificação de Alta seria apropriado. Se em dúvida, [então] nossa classificação padrão é Alta.

Na teoria, um(a) usuário(a) de uma versão anterior recente do livro pode examinar nossas vulnerabilidades atuais e anteriores e ver cada vulnerabilidade corrigida a qual afeta um pacote. Na prática, nem sempre fica claro que uma versão mais recente corrigiu a vulnerabilidade - se isso vier à tona depois que tenhamos feito um lançamento com uma versão mais recente do pacote, [então] tais itens seriam ignorados na presunção de que a maioria dos(as) usuários(as) já consultou as vulnerabilidades anteriores e é improvável que note adições novas - por exemplo, uma vulnerabilidade no flac-3.2 foi corrigida no flac-3.3, porém não mencionada publicamente. Em tais casos, nós podemos adicionar uma seção 'Avisos Tardios' entre os avisos para o livro atual e os avisos anteriores - veja-se 10.0-102 no `consolidated.html` e a entrada correspondente no `10.0.html`. Nesses casos, a melhor maneira para alertar os(as) usuários(as) é a de anunciar o aviso nas listas lfs-support e blfs-support.

De tempos em tempos, nós talvez fiquemos cientes de que os detalhes de um aviso deveriam ser mudados, por exemplo, aconteceria de ele ser inválido por alguma razão (por exemplo, nunca em uma versão lançada; ou aplicável somente ao Windows). Quando isso acontecer, nós podemos modificar o item existente e mudar a entrada 'Data' para 'Atualizado' com a data atual.

Para atualizações diárias dos avisos, os arquivos estão organizados em:

- `blfs/advisories/consolidated.html` (compartilhado entre o LFS e o BLFS).
- `lfs/advisories/NN.N.html` (para quaisquer avisos do LFS desde o lançamento (NN.N) atual).
- `bfs/advisories/NN.N.html` (para quaisquer avisos do BLFS desde o lançamento (NN.N) atual).

Existem duas partes para se criar um aviso:

- Pegar o próximo número disponível para criar uma entrada no `consolidated.html` contendo os detalhes necessários.

- Adicionar detalhes resumidos do aviso para esta versão do LFS ou do BLFS.

Grande parte de qualquer item novo será copiado a partir de um modelo ou a partir de um aviso anterior. É importante investir tempo quando revisar o que você criou, para ver se ele lê e vincula corretamente.

7.2. Consolidated.html

A página consolidada fornece uma lista dos nossos avisos desde que começamos a criá-los. Eles são numerados dentro dos lançamentos, em cada caso os mais recentes primeiro. Essa página existe em `blfs/advisories`, porém ela é comum a ambos LFS e BLFS.

Crie os detalhes para o próximo número de aviso disponível. A parte do topo da página contém um monte de itens comentados os quais podem ser usados como um modelo, porém se existirem avisos anteriores para esse pacote, [então] talvez seja mais fácil copiar os detalhes a partir de um daqueles.

Bem como o número, e o link sa-NN.N-nnn, cada entrada tem uma data e uma gravidade. Onde o(a) desenvolvedor(a) tenha atribuído uma gravidade, normalmente vamos com essa. Se uma fonte tal como NVD tenha atribuído uma gravidade, [então] considere se ela é apropriada (nosso padrão é 'Alta' mesmo para Negação de Serviço em aplicativos não servidor).

Onde um ou mais CVEs tenham sido divulgados, talvez seja útil vincular ao NVD se isso fornecer detalhes; ou, do contrário, a qualquer outra fonte confiável a qual explique a vulnerabilidade (isto é, Google para ela). Se os detalhes não estiverem públicos, [então] você não estaria apto(a) a encontrar um link externo apropriado. Porém, para pacotes os quais fornecem seus próprios avisos, é bom vincular a eles.

Renderize a página, verifique se os links (externos para aplicativos; e internos para os livros sysv e systemd) funcionam corretamente e que o que você escreveu e colou faz sentido.

Seria benéfico confirmar a mudança para consolidated.html neste ponto, de forma que a titularidade da propriedade do número do aviso seja sua (por exemplo, se você pensa que outra pessoa talvez esteja também atualizando avisos). Nesse caso, provavelmente é melhor fazer somente um se você tiver vários; e, então, atualizar a página para o lançamento atual.

7.3. Avisos desde o lançamento anterior.

Existem páginas separadas para o LFS e para o BLFS onde nós listamos nossos avisos desde nosso lançamento anterior. Essas páginas estão em ordem alfabética, com os avisos mais recentes vindo antes dos mais antigos para o mesmo pacote.

Em ambos LFS e BLFS, a página é rotulada com o número do nosso lançamento atual, de forma que 10.1.html a partir de março de 2021 até que lancemos nossa próxima versão.

Você verá que existe uma <h3>Nome do Pacote<h3> comentada como um guia. A linha <h4> deveria ser copiada a partir da entrada que você criou no `consolidated.html`. Siga isso com um parágrafo curto sumarizando o problema e como corrigi-lo (ou, em casos excepcionais, outras ações para contornar o problema). Adicione uma frase vinculando ao aviso no `consolidated.html`.

Alguns pacotes obtém informação extra, por exemplo, o Thunderbird tem um parágrafo itálico explicando como as vulnerabilidades são em um contexto semelhante a navegador. Se esse for o primeiro aviso para o Thunderbird neste lançamento, [então] copie essa parte a partir dos avisos anteriores. Similarmente, no LFS, o glibc obtém informação diferente, pois o único caminho de atualização suportado no LFS é construir um LFS novo: dê a informação e deixe que os(as) usuários(as) decidam o quão séria(s) é(são) a(s) vulnerabilidade(s) para o caso de uso deles(as) (muitas são bastante antigas).

7.4. O que fazer quando nós aprontamos um lançamento.

Quando um lançamento novo dos livros está pronto (isto é, os diretórios novos do livro para esta versão tendo sido povoados), existem observações nos arquivos `Notes-re-release.txt`:

Bem como para atualizar o `index.html` para o lançamento novo, e criar uma página nova para a versão nova - inicialmente declarando que "Não existem vulnerabilidades de segurança conhecidas" - os avisos para o que é, agora, o lançamento anterior precisam ser corrigidos.

Isso é porque os pacotes naquele lançamento foram testados para as versões e pacotes naquele lançamento, e, de tempos em tempos, os pacotes serão descartados; ou serão substituídos por versões bifurcadas; ou mesmo movidos para uma parte diferente do livro. Assim, os links deveriam ser mudados, de forma que, em vez de apontarem para o livro de desenvolvimento, eles agora apontem para a versão que nós lançamos recentemente.

Parte II. Automatização

Capítulo 8. Processos

8.1. Introdução

Este capítulo descreve os processos acerca de como converter os fontes XML do DocBook para HTML, os quais são apresentados para os(as) usuários(as) no sítio da web. Os processos são basicamente os mesmos em uma máquina local quando o(a) usuário(a) clona os XMLs para um repositório local e quer criar os arquivos HTML lá.

Os pré-requisitos são um conjunto pequeno de ferramentas, nomeadamente *make*; *DocBook XML*; *DocBook XSLT*; e as ferramentas *xsltproc*. Essas ferramentas são usadas para converter XML do DocBook para outros formatos; o formato alvo mais usado é o HTML, o qual pode ser usado para apresentar os livros de uma maneira formatada bem legal para o(a) usuário(a). Outros formatos seriam PDF, porém esse não é parte deste capítulo.

Este capítulo está dividido nas seguintes partes:

- Básicos acerca da conversão

  Descreve os passos básicos para converter os XMLs para HTML, os quais são comuns para o LFS e para o BLFS.

- Especiais no LFS ou no BLFS

  Observações acerca dos especiais a se ter em mente quando se converter o livro LFS ou o BLFS.

- Processos no servidor

  Observações para editores(as) e administradores(as) mantendo a conversão do livro no servidor principal do LFS.

8.2. Básicos

8.2.1. Renderizando livros

Quando um livro tenha sido tirado (clonado), mude para o diretório de nível superior do livro. A criação dos arquivos HTML é iniciada com o seguinte comando:

```
make
```

Por padrão, a versão sysv do livro será renderizada. Para criar a versão systemd, emita:

```
make REV=systemd
```

Os únicos valores permitidos para REV são `sysv` ou `systemd`. Qualquer outro valor - enquanto que sysv é o padrão quando nada for especificado - causará um erro.

O local da saída gerada é dependente do tipo do livro; será diferente quando se construir o LFS; o BLFS; ou qualquer outro livro. Usualmente, a saída gerada é um diretório no diretório HOME do(a) usuário(a) atual. Na época da escrita deste texto, os diretórios eram

Tipo	*Diretório*
LFS (sysv)	~/lfs-book
LFS (systemd)	~/lfs-systemd
BLFS (sysv)	~/public_html/blfs-book

BLFS (systemd)	~/public_html/blfs-systemd
Guia dos(as) Editores(as) (este livro)	~/lfs-editors-guide-output

Para especificar um diretório diferente de saída gerada, use o parâmetro BASEDIR:

```
make BASEDIR=<caminho/para/colocar/os/htmls>
```

Certamente, uma combinação de REV e BASEDIR é permitida.

8.2.1.1. Outros parâmetros comuns

Para controlar se o processo de construção deveria exibir todos os comandos executados em detalhes, use um parâmetro V não vazio:

```
make V=1
```

O processo de construção dos livros LFS ou BLFS exige a produção de alguns dados temporários. O parâmetro RENDERTMP define onde armazenar tais arquivos temporários. O padrão é `$HOME/tmp` e pode ser configurado para qualquer outro diretório, porém deveria ser diferente do BASEDIR:

```
make RENDERTMP=<caminho/para/colocar/arquivos/temporarios>
```

8.2.2. Scripts de inicialização / Arquivos de serviço

PARA FAZER: Descreva o que fazer quando os scripts de inicialização tiverem sido mudados. Edite o registro das mudanças; atualize o general.ent no LFS; ...; copie para o anduin; ...

8.3. Especiais

8.3.1. Especiais

Esta seção descreve algumas extensões e variações disponíveis para alguns tipos de livros.

8.3.1.1. LFS - Versão multi biblioteca

Existe uma ramificação especial do livro LFS que inclui instruções adicionais para aqueles(as) usuários(as) que gostam de construir um sistema que suporta binários de 32 bits e não somente de 64 bits. Para mais detalhes acerca de como e porque, consulte aquele livro; aqui somente os parâmetros exigidos para construir o livro XML são discutidos.

Para a finalidade de usar a versão multi biblioteca do livro, o repositório precisa ser comutado para a ramificação `multilib` usando-se o git:

```
git checkout multilib
```

Para controlar se e qual versão multi biblioteca do livro será criada, use o parâmetro ARCH.

```
make ARCH=ml_32
```

Os valores possíveis para ARCH são `ml_32` para incluir instruções para suportar os binários m32; ou `ml_x32` para a arquitetura mx32. `ml_default`, que é usado quando ARCH não for dado, inclui nenhuma das instruções adicionais e o livro resultante é praticamente idêntico ao livro quando a ramificação `trunk` é usada. `ml_all` inclui ambas as instruções `ml_32` e `ml_x32`.

8.4. Processos no servidor principal

8.4.1. Scripts

Os scripts úteis para automatizar a renderização diária dos livros estão localizados em `/usr/local/bin`.

A seguir está uma lista dos scripts que seriam de interesse para renderizar e instalar os livros no servidor principal:

- build-lfs-edguide.sh

  (sem parâmetros)

  Esse script cria o Guia dos(as) Editores(as) como um "paginador único" - significando que o guia inteiro está em uma página HTML.

  O resultado pode ser visualizado em *https://www.linuxfromscratch.org/lfs/LFS-EDITORS-GUIDE.html*.
- check-blfs-files.sh

  ...
- render-blfs-book-systemd.sh

  ...
- render-blfs-book.sh

  ...
- render-lfs-book-dev.sh

  ...
- render-lfs-book-systemd.sh

  ...
- update-hints.sh

  ...
- update-patches.sh

  ...
- update-website.sh

  ...

8.4.2. Tarefas agendadas

Existem várias tarefas agendadas (tarefas executadas automaticamente em uma hora específica). Essas tarefas automatizaram a renderização dos livros, bem como o manuseio de arquivos como patches, scripts de inicialização e assim por diante.

A seguir está uma lista das tarefas agendadas definidas no servidor principal:

...