Praticando Git

Sumário

Prefácio

TODO

Capítulo 1

Introdução

1.1 Sobre este livro

Este livro é, na verdade, um **tutorial prático** *(mão na massa e direto ao ponto)* para o aprendizado de Git. Com o seu auxílio, você estará praticando o uso do Git num ambiente Linux (Fedora 23). Talvez também seja possível você executar os mesmos comandos que apresento aqui em outros ambientes: distruições Linux diferentes, Mac OS X, Windows (com o auxílio do Cygwin). Mas, nesse caso, você poderá precisar adaptar os comandos que eu mostrarei aqui.

Siga, passo a passo, a execução dos comandos que eu apresento. Nesse livro , a maioria dos comandos serão digitados em um shell Bash. Para ter maior agilidade, você poderá simplesmente copiar/colar os comandos que apresento aqui, diretamente no teu shell.

Para ter, em tua máquina, o mesmo ambiente (Fedora 23) que eu utilizo para demonstrar o os comandos neste livro, você deverá instalar o VirtualBox, o Vagrant e, por fim, iniciar e configurar uma box (máquina virtual gerenciada pelo Vagrant) contendo esse sistema operacional. Os passos para isso são apresentados no próximo tópico.

Este livro é GRATUITO. Isso mesmo: você não precisa pagar nada por ele! Porque? Eu creio que já há muito, muito material mesmo, sobre Git, GitHub e assuntos correlatos, na Internet (veja as referências que cito ao final do livro). Sendo assim, este material é apenas mais um, que utilizo para apresentar os assuntos em questão, a minha maneira, seguindo o uso de ferramentas livres.

O código fonte deste livro também está disponível num repositório hospedado no GitHub. Ele pode ser utilizado para a geração deste livro utilizando o mesmo processo adotado por professores da Universidade Federal da Paraíba (UFPB) para a construção dos livros criados para os cursos de computação dessa instituição. Este livro segue o template asciidoc-book-template-with-rake-and-github podendo, dessa forma, ser construído através do site de geração dos livros que utiliza o processo especificado pela UFPB. A construção deste livro também pode ser realizada de maneira off-line, sem acesso ao site da UFPB, pelo uso do projeto producao-computacao-ead-ufpb-box.

1.2 Sobre o autor (Paulo Jerônimo)

Neste momento, eu (Paulo Jerônimo) atuo como gerente de projetos na instituição Cebraspe (Cespe/-UnB). Também sou desenvolvedor, com maior tempo de trabalho utilizando soluções e middleware desenvolvidos em Java e colocados em produção em plataforma Linux. Fora dos horários em que estou trabalhando para o Cebraspe atuo, ainda, como instrutor de cursos oficiais da Red Hat. Mais sobre

o meu histórico profissional pode ser encontrado em meu currículo http://j.mp/curriculopj. Meus trabalhos públicos mais recentes estão disponíveis, na minha conta no GitHub, em https://github.com/-paulojeronimo.

1.3 Público alvo

Este livro é para qualquer pessoa que deseje aprender como controlar versões de seus arquivos (código fonte de programas, de documentos, etc), de forma eficaz e eficiente, e trabalhar de forma colaborativa com outras pessoas.

O foco principal desse material está na realização de atividades através de uma linha de comando. Contudo, após aprendendido os conceitos do Git através de seus comandos, fazer uso desses conceitos em ferramentas gráficas torna-se algo muito simples.

Capítulo 2

Construindo o ambiente para a execução dos comandos deste livro

2.1 Instalando o VirtualBox

Para que você possa executar os comandos deste livro da mesma forma que eu o faço, você precisará de uma máquina virtual com o Linux Fedora 23 instalado.

Neste livro, eu utilizo o VirtualBox para fazer o gerenciamento de máquinas virtuais. Antes de prosseguir, siga os procedimentos de instalação dessa ferramenta para o teu sistema operacional.

Para exemplificar, os subtópicos a seguir referenciam vídeos que demonstram a instalação do VirtualBox em alguns sistemas operacionais.

2.1.1 No Windows

TODO

2.1.2 No Fedora

TODO

2.1.3 No OS X

TODO

2.2 Instalando o Vagrant

A forma mais simples e rápida de se gerenciar uma máquina virtual (criando, instalando e configurando pacotes e arquivos do sistema, etc) em diferentes ferramentas de virtualização (VirtualBox, VMware, KVM, Parallels, etc) é através do Vagrant.

Antes de prosseguir, faça a instalação do Vagrant conforme os procedimentos necessários para o teu sistema operacional.

Para exemplificar, os subtópicos a seguir referenciam vídeos que demonstram a instalação do Vagrant em alguns sistemas operacionais.

2.2.1 No Windows

TODO

2.2.2 No Fedora

TODO

2.2.3 No OS X

TODO

2.3 Construindo, acessando e configurando a box Vagrant

2.3.1 Construindo

Após ter instalado o Vagrant, crie um diretório chamado `git-box` e, dentro dele, crie um arquivo `VagrantFile` com o seguinte conteúdo:

```
Vagrant.configure(2) do |config|
  config.vm.box = "boxcutter/fedora23"
  ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'
  config.vbguest.auto_update = false
end
```

Em seguida, estando dentro do diretório `git-box`, inicie a box:

```
vagrant up
```

2.3.2 Acessando e configurando

Acesse o shell da box:

```
vagrant ssh
```

Já no shell da box, execute o seguinte comando:

```
cat << 'EOF' | bash
for f in fedora fedora-updates
do
    echo "Configurando o arquivo /etc/yum.repos.d/${f}.repo ..."
    sudo sed '
        s,^\(metalink=\),#\1,g
        s,^#\(baseurl=\)http://download.fedoraproject.org/pub/fedora ↩
           ,\1http://fedora.c3sl.ufpr.br,g
```

```
    ' -i /etc/yum.repos.d/${f}.repo
done
EOF
```

Os comandos acima configuram o gerenciador de pacotes (`dnf`) para utilizar o *mirror* público do Fedora disponível na UFPR. Isso evita que o Fedora fique tentando descobrir qual *mirror* está mais próximo de você e, consequentemente, seja mais rápido na instalação dos pacotes. Obviamente, se você estiver fora do Brasil, o comando acima talvez precise ser ajustado! ;)

Para executar alguns comandos apresentados neste livro, será necessária a instalação de alguns pacotes. Faça a instalação desses através do seguinte comando:

```
sudo dnf -y install tree vim
```

Algumas variáveis de ambiente deverão estar disponíveis no shell, toda vez que fizermos acesso a ele. Para isso, faremos ajustes no arquivo `~/.bashrc`, conforme o comando a seguir:

```
f=~/.bashrc; echo -e "export PS1='$ '\nexport TREE_CHARSET=ascii" >>  ↩
   $f; source $f
```

Executando os comandos acima sua box estará pronta para a execução dos comandos que virão a seguir, nos próximos tópicos desse livro.

2.3.3 Instalando o Asciinema

O Asciinema é um software que possibilita gravar e reproduzir tudo o que está sendo realizado num shell em execução. No contexto deste livro, ele poderá ser utilizado para reproduzir a tela dos comandos digitados e executados em cada um dos tópicos deste livro.

Para utilizá-lo faça sua instalação:

```
sudo dnf -y install asciinema
```

Crie, executando o comando abaixo, uma função que será bastante útil para você visualizar a execução dos comandos apresentados neste livro:

```
cat >> ~/.bashrc <<'EOF'
play() { asciinema play /vagrant/livro-git.asciinema/$1
EOF
```

Capítulo 3

Instalando e configurando o Git

3.1 Instalando

Estando no shell da box criada, para instalar o Git, execute:

```
sudo dnf -y install git
```

Confira a versão do Git que foi instalada:

```
git --version
```

Deverá ser esta (ou superior):

```
git version 2.5.0
```

3.2 Pedindo um help

Vá para o seu diretório `$HOME` e, em seguida, solicite a ajuda do git para saber os comandos disponíveis:

```
cd && git help
```

Digite o comando abaixo e leia o que comando `git config` faz:

```
git help config
```

3.3 Configurando o básico

Execute os comandos abaixo substituindo o meu nome e email pelas tuas informações. Estes comandos criarão o arquivo `~/.gitconfig`:

```
git config --global user.name "Paulo Jerônimo"
git config --global user.email "pj@paulojeronimo.info"
```

Veja o conteúdo do seu arquivo (`.gitignore`) após a execução dos comandos acima:

```
cat .gitconfig
```

Em minha execução, a saída do comando abaixo apresenta este conteúdo:

```
[user]
   name = Paulo Jerônimo
   email = pj@paulojeronimo.info
```

Capítulo 4

Gerenciando arquivos num repositório Git

4.1 Criando um novo repositório

Vá para o diretório `/vagrant` e crie o diretório `repo1` que conterá o primeiro repositório git desse tutorial:

```
cd /vagrant/; mkdir repo1 && cd $_; pwd
```

Inicialize esse repositório:

```
git init
```

Essa inicialização criará o diretório `.git`. Esse é o único diretório que será criado pelo Git em seu projeto. Isso é um diferencial em relação a outras ferramentas de controle de versão, como CVS e Subversion, que criam subdiretórios de controle para cada diretório que você cria em seu projeto.

O conteúdo do diretório `.git` não é importante nesse momento. Mas veja, executando o comando abaixo, que ele conterá vários arquivos gerenciados pelo Git:

```
tree -a
```

Observe a saída do comando:

```
.
`-- .git
    |-- branches
    |-- config
    |-- description
    |-- HEAD
    |-- hooks
    |   |-- applypatch-msg.sample
    |   |-- commit-msg.sample
    |   |-- post-update.sample
    |   |-- pre-applypatch.sample
    |   |-- pre-commit.sample
    |   |-- prepare-commit-msg.sample
    |   |-- pre-push.sample
    |   |-- pre-rebase.sample
    |   `-- update.sample
```

```
    |-- info
    |   `-- exclude
    |-- objects
    |   |-- info
    |   `-- pack
    `-- refs
        |-- heads
        `-- tags

10 directories, 13 files
```

4.2 Verificando o estado do repositório

Execute o comando abaixo e você verá que o próprio Git lhe dirá que passos podem ser realizados em seguida:

```
git status
```

Observe a saída do comando:

```
On branch master

Initial commit

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

Note que o próprio git dá a dica do que pode ser feito! (Na saída acima, ele diz que você pode criar ou copiar arquivos para o repositório e executar um `git add`).

Esse comando será um dos que você mais utilizará. Ele apresentará o estado em que se encontram os arquivos que você estará gerenciando através do Git.

4.3 Criando um novo arquivo no diretório de trabalho

O Git trabalha com três áreas: o *working directory* (diretório de trabalho), a *staging area* (índice) e o *git directory* (repositório).

Novos arquivos são criados no diretório de trabalho e são *untracked* (desconhecidos pelo Git) até serem adicionados ao índice.

Crie um novo arquivo:

```
echo 'Olá Git!' > OlaGit.txt
```

Verifique o estado do repositório:

```
git status
```

Observe a saída do comando:

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        OlaGit.txt

nothing added to commit but untracked files present (use "git add" to track)
```

4.4 Adicionando arquivos ao índice

O índice (*staging area*) é uma área intermediária do Git para a qual enviamos arquivos que irão para o repositório no próximo *commit*.

O índice aponta para arquivos no diretório de trabalho.

O comando `git add` (sinônimo para `git stage`) permite adicionar arquivos nessa área.

Então, adicione o arquivo `OlaGit.txt`:

```
git add OlaGit.txt
```

Verifique o estado do repositório (`git status`):

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   OlaGit.txt
```

4.5 Comitando

Comitar (verbo abrasileirado a partir do termo em inglês *commit*), no contexto do Git, é adicionar arquivos ao repositório.

Toda vez que é feito um *commit* os arquivos que estão no índice saem dessa área e são levados para o *git diretory* (repositório).

Efetue o seu primeiro *commit*:

```
git commit -m 'Primeiro commit para o aprendizado do Git'
```

Note a saída do comando:

```
[master (root-commit) d5e4a3d] Primeiro commit para o aprendizado do  ↩
   Git
 1 file changed, 1 insertion(+)
 create mode 100644 OlaGit.txt
```

Note o estado do repositório (`git status`):

4.6 Adicionando mais mudanças

Crie alguns novos arquivos, executando os comandos abaixo:

```
echo 'Ola PJ!' > OlaPJ.txt
echo 'Eu sou um bom aluno!' > EuAluno.txt
echo 'Eu sou um arquivo errado!' > ArquivoErrado.txt
d=dir1; mkdir $d; echo "Olá de dentro de $PWD/$d" > $d/OutroArquivo. ↩
   txt
d=dir2; mkdir $d; echo "Olá de dentro de $PWD/$d" > $d/OutroArquivo. ↩
   txt
echo "Olá de dentro de $PWD/$d" > $d/MaisOutroArquivo.log
```

Veja a árvore de diretórios/arquivos:

```
tree
```

Observe a saída do comando (como ficou o sistema de arquivos):

```
.
|-- ArquivoErrado.txt
|-- dir1
|   `-- OutroArquivo.txt
|-- dir2
|   |-- MaisOutroArquivo.log
|   `-- OutroArquivo.txt
|-- EuAluno.txt
|-- OlaGit.txt
`-- OlaPJ.txt

2 directories, 7 files
```

Note o estado do repositório (`git status`):

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ArquivoErrado.txt
    EuAluno.txt
    OlaPJ.txt
    dir1/
    dir2/

nothing added to commit but untracked files present (use "git add" to  ↩
   track)
```

Adicione os arquivos `.txt` do diretório corrente:

```
git add *.txt
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoErrado.txt
    new file:   EuAluno.txt
    new file:   OlaPJ.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    dir1/
    dir2/
```

Adicione os arquivos `.txt` que estejam em qualquer subdiretório (note o uso das aspas):

```
git add "*.txt"
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoErrado.txt
    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
    new file:   dir2/OutroArquivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    dir2/MaisOutroArquivo.log
```

Crie mais um arquivo em `dir2`:

```
echo "Mais um arquivo em $PWD/$d." > $d/MaisUmArquivo.etc
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoErrado.txt
    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
```

```
    new file:   dir2/OutroArquivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    dir2/MaisOutroArquivo.log
    dir2/MaisUmArquivo.etc
```

Adicione todos os arquivos de `dir2`:

```
git add dir2/
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoErrado.txt
    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
    new file:   dir2/MaisOutroArquivo.log
    new file:   dir2/MaisUmArquivo.etc
    new file:   dir2/OutroArquivo.txt
```

4.7 Removendo um arquivo do índice

Ooops! O ArquivoErrado.txt naõ era para ser adicionado! Às vezes, por engano, enviamos um arquivo para a área de índice. O Git permite-nos desfazer isso.

O próprio comando `git status` nos dá a dica sobre como proceder (como apresentado em sua última execução).

Execute os comando abaixo para remover o ArquivoErrado.txt do índice e verificar o estado:

```
git reset HEAD ArquivoErrado.txt
```

Note o estado do repositório (`git status`):

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

    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
    new file:   dir2/MaisOutroArquivo.log
    new file:   dir2/MaisUmArquivo.etc
    new file:   dir2/OutroArquivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
    ArquivoErrado.txt
```

Fazer um `git reset HEAD` sem informar o arquivo removerá todos os que foram adiconados ao índice. Execute:

```
git reset HEAD
```

Note o estado do repositório (`git status`):

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ArquivoErrado.txt
    EuAluno.txt
    OlaPJ.txt
    dir1/
    dir2/

nothing added to commit but untracked files present (use "git add" to  ↩
   track)
```

4.8 Adicionando arquivos de maneira interativa

O comando `git add -i` nos oferece a possibilidade de adicionar arquivos de maneria iterativa. Execute-o:

```
git add -i
```

Conforme apresentado na saída a seguir, informe o que é solicitado. Isso nos trará a situação desejada (todos arquivos adicionados, exceto o `ArquivoErrado.txt`):

```
           staged     unstaged path


*** Commands ***
  1: status   2: update   3: revert   4: add untracked
  5: patch    6: diff     7: quit     8: help
What now> 4
  1: ArquivoErrado.txt
  2: EuAluno.txt
  3: OlaPJ.txt
  4: dir1/OutroArquivo.txt
  5: dir2/MaisOutroArquivo.log
  6: dir2/MaisUmArquivo.etc
  7: dir2/OutroArquivo.txt
Add untracked>> 2-7
  1: ArquivoErrado.txt
* 2: EuAluno.txt
* 3: OlaPJ.txt
* 4: dir1/OutroArquivo.txt
```

```
* 5: dir2/MaisOutroArquivo.log
* 6: dir2/MaisUmArquivo.etc
* 7: dir2/OutroArquivo.txt
Add untracked>>
added 6 paths

*** Commands ***
  1: status   2: update   3: revert   4: add untracked
  5: patch    6: diff     7: quit     8: help
What now> q
Bye.
```

Note o estado do repositório (`git status`):

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

    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
    new file:   dir2/MaisOutroArquivo.log
    new file:   dir2/MaisUmArquivo.etc
    new file:   dir2/OutroArquivo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ArquivoErrado.txt
```

4.9 Comitando novamente

Faça o seu segundo commit:

```
git commit -m 'Segundo commit, mais arquivos adicionados'
```

Observe a saída do comando:

```
[master 3db9884] Segundo commit, mais arquivos adicionados
 6 files changed, 6 insertions(+)
 create mode 100644 EuAluno.txt
 create mode 100644 OlaPJ.txt
 create mode 100644 dir1/OutroArquivo.txt
 create mode 100644 dir2/MaisOutroArquivo.log
 create mode 100644 dir2/MaisUmArquivo.etc
 create mode 100644 dir2/OutroArquivo.txt
```

4.10 Visualizando o log

O comando log possiblita-nos visualizar os commits realizados. Execute:

```
git log
```

Observe a saída do comando:

```
commit 3db9884f3d9120eb0887933e0ce1983251e4fa64
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:01:09 2016 +0000

    Segundo commit, mais arquivos adicionados

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git
```

O comando `git log` tem várias opções. Execute:

```
git help log
```

A opção `--summary` mostra um resumo do que entrou no commit. Execute:

```
git log --summary
```

Observe a saída do comando:

```
commit 3db9884f3d9120eb0887933e0ce1983251e4fa64
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:01:09 2016 +0000

    Segundo commit, mais arquivos adicionados

 create mode 100644 EuAluno.txt
 create mode 100644 OlaPJ.txt
 create mode 100644 dir1/OutroArquivo.txt
 create mode 100644 dir2/MaisOutroArquivo.log
 create mode 100644 dir2/MaisUmArquivo.etc
 create mode 100644 dir2/OutroArquivo.txt

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

 create mode 100644 OlaGit.txt
```

Veja o log com a opção `--name-status` para obter uma lista dos arquivos foram adicionados, modificados ou removidos em cada *commit*:

```
git log --name-status
```

Observe a saída do comando:

```
commit 3db9884f3d9120eb0887933e0ce1983251e4fa64
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:01:09 2016 +0000

    Segundo commit, mais arquivos adicionados

A       EuAluno.txt
A       OlaPJ.txt
A       dir1/OutroArquivo.txt
A       dir2/MaisOutroArquivo.log
A       dir2/MaisUmArquivo.etc
A       dir2/OutroArquivo.txt

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

A       OlaGit.txt
```

Veja o log com ainda mais detalhes ao passar o parâmetro `--stat`. Isto apresentará estatísticas relativas ao *commit*:

```
git log --stat
```

Observe a saída do comando:

```
commit 3db9884f3d9120eb0887933e0ce1983251e4fa64
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:01:09 2016 +0000

    Segundo commit, mais arquivos adicionados

 EuAluno.txt               | 1 +
 OlaPJ.txt                 | 1 +
 dir1/OutroArquivo.txt     | 1 +
 dir2/MaisOutroArquivo.log | 1 +
 dir2/MaisUmArquivo.etc    | 1 +
 dir2/OutroArquivo.txt     | 1 +
 6 files changed, 6 insertions(+)

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

 OlaGit.txt | 1 +
 1 file changed, 1 insertion(+)
```

4.11 Consertando um commit que deveria conter um arquivo a mais

Oh não! :P O segundo commit era pra conter um `ArquivoCerto.txt`! Não se apavore! :D Creia (aprenda) e o Git te ajudará! ;)

Renomeie e altere o conteúdo do `ArquivoErrado.txt`:

```
mv ArquivoErrado.txt ArquivoCerto.txt
sed -i 's/errado/certo/g' ArquivoCerto.txt
```

Veja o conteúdo do `ArquivoCerto.txt`:

```
cat ArquivoCerto.txt
```

Adicione o arquivo ao índice:

```
git add ArquivoCerto.txt
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoCerto.txt
```

Refaça o commit (note o uso da opção `--amend`):

```
git commit -m 'Segundo commit, mais arquivos adicionados' --amend
```

Observe a saída do comando:

```
[master ae86ff0] Segundo commit, mais arquivos adicionados
 Date: Fri Mar 18 21:01:09 2016 +0000
 7 files changed, 7 insertions(+)
 create mode 100644 ArquivoCerto.txt
 create mode 100644 EuAluno.txt
 create mode 100644 OlaPJ.txt
 create mode 100644 dir1/OutroArquivo.txt
 create mode 100644 dir2/MaisOutroArquivo.log
 create mode 100644 dir2/MaisUmArquivo.etc
 create mode 100644 dir2/OutroArquivo.txt
```

Verifique o log do repositório:

```
git log --name-status
```

Observe a saída do comando:

```
commit ae86ff0ef262dff52c22127f87f6a640b639c5df
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:01:09 2016 +0000

    Segundo commit, mais arquivos adicionados
```

```
A       ArquivoCerto.txt
A       EuAluno.txt
A       OlaPJ.txt
A       dir1/OutroArquivo.txt
A       dir2/MaisOutroArquivo.log
A       dir2/MaisUmArquivo.etc
A       dir2/OutroArquivo.txt

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

A       OlaGit.txt
```

4.12 Desfazendo o último commit

4.12.1 Forma 1

Outra alternativa para resolver o problema anterior seria descartar todo o segundo commit e refazê-lo novamente. Para fazer isso, execute:

```
git reset --soft HEAD^
```

Note o estado do repositório (`git status`):

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

    new file:   ArquivoCerto.txt
    new file:   EuAluno.txt
    new file:   OlaPJ.txt
    new file:   dir1/OutroArquivo.txt
    new file:   dir2/MaisOutroArquivo.log
    new file:   dir2/MaisUmArquivo.etc
    new file:   dir2/OutroArquivo.txt
```

Observe que os arquivos voltaram para o índice.

Observe também o log (`git log`) e note que apenas o primeiro commit ficou:

```
commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git
```

Agora, refaça o commit:

```
git commit -m 'Segundo commit, mais arquivos adicionados'
```

Observe a saída do comando:

```
[master 73dd7e0] Segundo commit, mais arquivos adicionados
 7 files changed, 7 insertions(+)
 create mode 100644 ArquivoCerto.txt
 create mode 100644 EuAluno.txt
 create mode 100644 OlaPJ.txt
 create mode 100644 dir1/OutroArquivo.txt
 create mode 100644 dir2/MaisOutroArquivo.log
 create mode 100644 dir2/MaisUmArquivo.etc
 create mode 100644 dir2/OutroArquivo.txt
```

4.12.2 Forma 2

Outra forma (apenas com uma sintaxe diferente) é executar o comando a seguir:

```
git reset HEAD~1 --soft
```

Se quiséssemos voltar dois commits, poderíamos informar `HEAD~2` (no lugar de `HEAD~1`) ou `HEAD^^` (na primeira forma).

Antes de continuar, observe o estado do repositório, o log, e refaça o *commit* (como feito na forma 1)!

4.13 Desfazendo o último commit e todas as mudanças

Com `git reset --soft HEAD^` (ou `git reset HEAD~1 --soft`) desfazemos o último commit deixando os arquivos no índice. Ou seja, com esse comando o Git não perde completamente todas as informações do último commit. Talvez você queria que, realmente, o Git esqueça esse commit fazendo tudo voltar a ser como era após o primeiro commit. Contudo, isso não afetará arquivos *untracked*, ou seja, os que estão que estão em no diretório de trabalho.

Para comprovar, primeiro faça um backup desse diretório para não perder o que já foi feito:

```
(cd ..; d=repo1; tar cvfz $d.tar.gz $d)
```

Crie um novo arquivo e verifique o estado:

```
echo 'Eu sou um novo arquivo!' > NovoArquivo.txt
```

Note o estado do repositório (`git status`):

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    NovoArquivo.txt

nothing added to commit but untracked files present (use "git add" to ↩
   track)
```

Execute:

```
git reset --hard HEAD^
```

Observe a saída do comando:

```
commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git
```

Note o estado do repositório (`git status`):

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    NovoArquivo.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Note que agora temos apenas um commit e que voltamos ao estado em que estávamos após tê-lo realizado. Então, o `reset --hard HEAD^` volta tudo ao estado que era antes do último commit sem excluir os arquivos que estão no diretório de trabalho.

Poderíamos, nesse ponto, ter o desejo de apagar todos arquivos que foram criados e que não estão no índice e nem no repositório. É fácil fazer isso executando o comando `git clean`. Execute:

```
git clean -f
```

Observe a saída do comando:

```
Removing NovoArquivo.txt
```

Note o estado do repositório (`git status`):

```
On branch master
nothing to commit (working directory clean)
```

Você poderia voltar o backup feito para continuar o tutorial. O comando, para isso, seria este (não execute):

```
cd ..; d=repo1; rm -rf $d && tar xvfz $d.tar.gz && cd $d
```

Contudo, você não precisa fazer isso (e nem precisaria ter feito um backup). Há uma alternativa no Git ... (próximo tópico)

4.14 Recuperando um commit desfeito

O Git armazena um log de tudo o que vai sendo realizado no repositório.

Execute o comando abaixo para visualizar, na saída apresentada, o número do commit perdido:

```
git reflog
```

Observe a saída do comando:

```
d5e4a3d HEAD@{0}: reset: moving to HEAD^
ffa73cb HEAD@{1}: commit: Segundo commit, mais arquivos adicionados
d5e4a3d HEAD@{2}: reset: moving to HEAD~1
73dd7e0 HEAD@{3}: commit: Segundo commit, mais arquivos adicionados
d5e4a3d HEAD@{4}: reset: moving to HEAD^
ae86ff0 HEAD@{5}: commit (amend): Segundo commit, mais arquivos  ↩
   adicionados
3db9884 HEAD@{6}: commit: Segundo commit, mais arquivos adicionados
d5e4a3d HEAD@{7}: commit (initial): Primeiro commit para o aprendizado ↩
    do Git
```

Para recuperar o commit perdido antes do último `git reset HEAD^ --hard` perceba que, pela saída acima, o hash desse commit é ffa73cb. Então, execute o comando a seguir para recuperá-lo (adapte-o conforme a saída que você obter):

```
git merge ffa73cb
```

Observe a saída do comando:

```
Updating d5e4a3d..ffa73cb
Fast-forward
 ArquivoCerto.txt          | 1 +
 EuAluno.txt               | 1 +
 OlaPJ.txt                 | 1 +
 dir1/OutroArquivo.txt     | 1 +
 dir2/MaisOutroArquivo.log | 1 +
 dir2/MaisUmArquivo.etc    | 1 +
 dir2/OutroArquivo.txt     | 1 +
 7 files changed, 7 insertions(+)
 create mode 100644 ArquivoCerto.txt
 create mode 100644 EuAluno.txt
 create mode 100644 OlaPJ.txt
 create mode 100644 dir1/OutroArquivo.txt
 create mode 100644 dir2/MaisOutroArquivo.log
 create mode 100644 dir2/MaisUmArquivo.etc
 create mode 100644 dir2/OutroArquivo.txt
```

Agora você sabe que backups são essenciais mas, no caso do Git, ele próprio se vira para nos ajudar na recuperação de arquivos perdidos! =)

Veja novamente o log para ter certeza de que estamos no segundo commit:

```
git log --stat
```

Observe a saída do comando:

```
commit ffa73cb0a4f49fec57b1ff2f7234974b3e89dfce
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:14:45 2016 +0000
```

```
    Segundo commit, mais arquivos adicionados

 ArquivoCerto.txt          | 1 +
 EuAluno.txt               | 1 +
 OlaPJ.txt                 | 1 +
 dir1/OutroArquivo.txt     | 1 +
 dir2/MaisOutroArquivo.log | 1 +
 dir2/MaisUmArquivo.etc    | 1 +
 dir2/OutroArquivo.txt     | 1 +
 7 files changed, 7 insertions(+)

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

 OlaGit.txt | 1 +
 1 file changed, 1 insertion(+)
```

Perceba que a árvore de diretórios/arquivos foi recuperada:

```
tree
```

Observe a saída do comando:

```
.
|-- ArquivoCerto.txt
|-- dir1
|   `-- OutroArquivo.txt
|-- dir2
|   |-- MaisOutroArquivo.log
|   |-- MaisUmArquivo.etc
|   `-- OutroArquivo.txt
|-- EuAluno.txt
|-- OlaGit.txt
`-- OlaPJ.txt
```

4.15 Voltando um arquivo no diretório de trabalho para sua última versão no repositório

Altere o arquivo `EuAluno.txt` acrescentando uma nova linha:

```
echo 'Sou atencioso' >> EuAluno.txt
```

Verifique a mudança (`cat EuAluno.txt`):

```
Eu sou um bom aluno!
Sou atencioso
```

Note o estado do repositório (`git status`):

```
On branch master
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:   EuAluno.txt

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

Peça ao git para desfazer o que você fez:

```
git checkout -- EuAluno.txt
```

Verifique o conteúdo do arquivo (`cat EuAluno.txt`):

```
Eu sou um bom aluno!
```

Note o estado do repositório (`git status`):

```
On branch master
nothing to commit (working directory clean)
```

4.16 Visualizando diferenças

Altere, novamente, o arquivo `EuAluno.txt`, adicionando um novo conteúdo:

```
echo 'Sou atencioso. Aprenderei Git! \o/' >> EuAluno.txt
```

Verifique as diferenças:

```
git diff
```

Observe a saída do comando:

```
diff --git a/EuAluno.txt b/EuAluno.txt
index eed3ade..534585a 100644
--- a/EuAluno.txt
+++ b/EuAluno.txt
@@ -1 +1,2 @@
 Eu sou um bom aluno!
+Sou atencioso. Aprenderei Git! \o/
```

4.17 Visualizando diferenças através de uma ferramenta

A saída do comando diff não é tão intuitiva quanto a visualização das diferenças entre dois arquivos através de uma ferramenta apropriada para isso. Por esse motivo, o git consegue delegar essa tarefa.

Execute:

```
git difftool
```

Observe a saída do comando:

```
This message is displayed because 'diff.tool' is not configured.
See 'git difftool --tool-help' or 'git help config' for more details.
'git difftool' will now attempt to use one of the following tools:
kompare emerge vimdiff

Viewing (1/1): 'EuAluno.txt'
Launch 'vimdiff' [Y/n]:
2 files to edit
```

O que ele faz é executar a ferramenta `vimdiff` que, por sua vez, apresenta as diferenças entre a versão que está no repositório e a que está no diretório de trabalho.

Para tornar o uso do vimdiff a opção padrão (e global), você deve realizar configurações no git, com os seguintes comandos:

```
git config --global diff.tool vimdiff
git config --global difftool.prompt false
```

Se as configurações são realizadas de forma global, como acima, além de ler o conteúdo do arquivo `~/.gitconfig` você também pode executar o comando a seguir:

```
git config -l
```

A saída desse último comando, entretanto, leva em conta tanto as configurações globais, quanto as locais ao repositório (arquivo `.git/config`).

Se você, agora, repetir o comando `git difftool`, notará que nada será questionado. Obviamente, outras ferramentas de visualização entre dois arquivos podem ser configuradas.

4.18 Visualizando diferenças em arquivos que estão no índice

Adicione o arquivo `EuAluno.txt` ao índice:

```
git add EuAluno.txt
```

Note o estado do repositório (`git status`):

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

    modified:   EuAluno.txt
```

Observe que não haverá saída para o comando a seguir:

```
git diff
```

Para pegar as diferenças do repositório em relação a arquivos que já foram para o índice, adicione o parâmetro `--staged` ao comando anterior:

```
git diff --staged
```

Observe a saída do comando:

```
diff --git a/EuAluno.txt b/EuAluno.txt
index eed3ade..534585a 100644
--- a/EuAluno.txt
+++ b/EuAluno.txt
@@ -1 +1,2 @@
 Eu sou um bom aluno!
+Sou atencioso. Aprenderei Git! \o/
```

Obviamente, você também pode executar o comando `git difftool` com o parâmetro `--staged` (experimente).

4.19 Alterando um arquivo que já foi para o índice

Ops! Você errou depois de adicionar o arquivo ao índice! :(Você queria um salto de linha após o "atencioso.". Então, conserte isso. Peça ao git para voltar o arquivo:

```
git reset HEAD EuAluno.txt
```

Observe a saída do comando:

```
Unstaged changes after reset:
M  EuAluno.txt
```

Retire o arquivo do índice:

```
git checkout -- EuAluno.txt
```

Finalmente, modifique o arquivo:

```
echo -e 'Sou atencioso.\nAprenderei Git! \o/' >> EuAluno.txt
```

Reveja o seu conteúdo (`cat EuAluno.txt`):

```
Eu sou um bom aluno!
Sou atencioso.
Aprenderei Git! \o/
```

Adicione-o, novamente, ao índice:

```
git add EuAluno.txt
```

Nota: Não é, extritamente, necessário remover um arquivo do índice para fazer uma alteração em seu conteúdo. Dessa forma, evitamos a necessidade de um `git checkout` para, em seguida, fazer um `git add`.

Note o estado do repositório (`git status`). A seguir, refaça o commit:

```
git commit -m 'Terceiro commit, EuAluno, já estou aprendendo git'
```

Observe a saída do comando:

```
[master d33aa12] Terceiro commit, EuAluno, já estou aprendendo git
 1 file changed, 2 insertions(+)
```

Verifique o log:

```
git log --name-status
```

Observe a saída do comando:

```
commit d33aa12c6d25383582d4a31a93c2a40141b652bd
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 22:00:42 2016 +0000

    Terceiro commit, EuAluno, já estou aprendendo git

M       EuAluno.txt

commit ffa73cb0a4f49fec57b1ff2f7234974b3e89dfce
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 21:14:45 2016 +0000

    Segundo commit, mais arquivos adicionados

A       ArquivoCerto.txt
A       EuAluno.txt
A       OlaPJ.txt
A       dir1/OutroArquivo.txt
A       dir2/MaisOutroArquivo.log
A       dir2/MaisUmArquivo.etc
A       dir2/OutroArquivo.txt

commit d5e4a3d0dfdcd29ad3e8af2b7e45f4c6414b6f70
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 20:31:51 2016 +0000

    Primeiro commit para o aprendizado do Git

A       OlaGit.txt
```

4.20 Removendo um arquivo do repositório

Remova o arquivo `dir2/OutroArquivo.txt`:

```
git rm dir2/OutroArquivo.txt
```

Observe a saída do comando:

```
rm 'dir2/OutroArquivo.txt'
```

Note o estado do repositório (`git status`):

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

    deleted:    dir2/OutroArquivo.txt
```

Note que o arquivo também foi removido do diretório de trabalho:

```
tree
```

Faça o quarto commit:

```
git commit -m 'Quarto commit, agora aprendi a excluir coisas do  ↩
   repositório'
```

Observe a saída do comando:

```
[master 39e569e] Quarto commit, agora aprendi a excluir coisas do  ↩
   repositório
 1 file changed, 1 deletion(-)
 delete mode 100644 dir2/OutroArquivo.txt
```

Verifique novamente o log:

```
git log --name-status
```

Observe a saída do comando (aqui ela está truncada para mostrar apenas o último commit):

```
commit 39e569ec5873a8112c98f854bc821f075882cb95
Author: Paulo Jerônimo <pj@paulojeronimo.info>
Date:   Fri Mar 18 22:03:46 2016 +0000

    Quarto commit, agora aprendi a excluir coisas do repositório

D       dir2/OutroArquivo.txt
...
...
```

4.21 Ignorando arquivos

O arquivo `.gitignore` é usado pelo Git para saber que arquivos devem ser ignorados em suas operações.

Supondo, então, que não você não deseja os arquivos/diretórios `*~`, `./tmp` e `./target` em seu repositório, você pode criar o conteúdo para o arquivo `.gitignore` com o seguinte comando:

```
cat > .gitignore <<-EOF
*~
tmp/
target/
EOF
```

Note que, agora, ao criar um diretório `target` com alguns arquivos dentro de seu projeto, esse diretório será ignorado pelos próximos comandos do Git.

Crie alguns diretórios e arquivos, com os seguintes comandos:

```
d=target; mkdir $d
touch $d/{f1.txt,f2.txt}
touch ArquivoCerto.txt~
d=tmp; mkdir $d
touch $d/lixo
```

Veja a árvore de diretórios/arquivos:

```
tree
```

Note o estado do repositório (`git status`):

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

 .gitignore
nothing added to commit but untracked files present (use "git add" to ↩
   track)
```

Observe que a saída do comando acima informa que o Git está reconhecendo apenas o arquivo `.gitignore` no diretório. Então, os outros estão sendo ignorados por ele (como queríamos) ;).

Adicione esse arquivo (`.gitignore`) ao repositório:

```
git add .gitignore
```

Faça um novo commit:

```
git commit -m 'Quinto commit, agora aprendi a ignorar arquivos com o . ↩
   gitignore'
```

Talvez você queira que essa configuração de arquivos a serem ignorados também funcione para outros repositórios (não apenas o que você está trabalhando agora). Nesse caso, um arquivo global também pode ser configurado para o seu usuário. Faça isso, executando os seguintes comandos:

```
cp .gitignore ~
git config --global core.excludesfile ~/.gitignore
```

Verifique essa configuração, com o comando abaixo:

```
git config -l
```

Também pode ser útil, em algumas situações, mandar o Git remover arquivos presentes no diretório de trabalho e que são ignorados por ele. Se você quiser apenas ver que arquivos deveriam ser removidos (sem realmente removê-los), execute:

```
git clean -n -d -x
```

Observe a saída do comando:

```
Would remove ArquivoCerto.txt~
Would remove target/
Would remove tmp/
```

Estando certo de que deseja realmente remover os arquivos listados, é só executar esse mesmo comando substituindo o parâmetro `-n` pelo `-f`:

```
git clean -f -d -x
```

Observe a saída do comando:

```
Removing ArquivoCerto.txt~
Removing target/
Removing tmp/
```