Linux, Git e Versionamento

Sumário

[O que é Linux? 1](#_Toc76546208)

[Distribuições Linux 2](#_Toc76546209)

[Entendendo os diretórios da pasta raiz do sistema 3](#_Toc76546210)

[Principais Comandos do Terminal Linux 5](#_Toc76546211)

[Entendendo as informações que aparecem no Bash 5](#_Toc76546212)

[Comandos relacionados aos diretórios do sistema 5](#_Toc76546213)

[Comandos relacionados à arquivos 6](#_Toc76546214)

[Comandos para instalar/atualizar/remover programas 6](#_Toc76546215)

[Outros comandos 7](#_Toc76546216)

[Tipos de comandos do Terminal 7](#_Toc76546217)

[Dicas 7](#_Toc76546218)

# O que é Linux?

Antes de utilizarmos um sistema operacional baseado em Linux ou mesmo aprendermos os comandos para trabalhar com ele, dentre outras coisas mais práticas, é super importante que possamos entender primeiro o que é Linux, bem como entender os principais termos que estão ao redor do ecossistema do Linux.

É muito comum que as pessoas se refiram ao Linux como sendo um sistema operacional. No entanto, tecnicamente, o Linux é apenas o ***kernel*** que é utilizado para a criação de diferentes sistemas operacionais. Em outras palavras, o Linux é uma tecnologia ou um componente que pode estar contido em um sistema operacional.

O termo *kernel*, portanto, é o mais importante nessa definição de Linux e, por isso, vamos entendê-la da melhor maneira possível. Ao pé da letra, o termo *kernel* pode ser traduzido como núcleo, o que pode fazer sentido quando entendemos o que essa camada/componente do sistema operacional representa.

Todo sistema operacional possui um *kernel* (Windows, Linux, MacOS, Android, etc), ou núcleo; e esse componente do sistema operacional é responsável por gerenciar o hardware, ou seja, é nele que estão os *drivers* e, por isso, ele é responsável por gerenciar a interação entre os aplicativos do sistema e o seu hardware. Em outras palavras, é o *kernel* que gerencia e permite a interação entre o sistema operacional e os componentes de hardware do seu laptop, desktop, smartphone, smartwatch e até mesmo os nossos periféricos, como a webcam, o teclado, mouse ou fones de ouvido. Tudo isso só funciona como deveria porque o *kernel* está alocando todos os recursos necessários para tais operações.

O Linux foi criado por um estudante de computação da Universidade de Helsinki, na Finlândia, chamado Linus Torvalds. O objetivo dele era aprimorar o kernel do Minix, um sistema operacional Unix-like escrito por Andrew Tanenbaum. A primeira versão do kernel Linux foi publicada no final do ano de 1991.

O grande diferencial do Linux em relação aos demais kernels é Linus Torvalds decidiu deixar o código fonte do sistema aberto, o que resultou no paradigma do **software livre**. Para que um software seja considerado livre, é necessário que ele obedeça à quatro liberdades, as quais são:

Liberdade 1: Liberdade de executar o programa para qualquer propósito.

Liberdade 2: Liberdade para estudar como o programa funciona e adaptá-lo às suas necessidades.

Liberdade 3: Liberdade de redistribuir cópias de forma que você possa ajudar outras pessoas.

Liberdade 4: Liberdade para melhorar o programar e disponibilizar as melhoras para o público, de forma que toda a comunidade possa se beneficiar delas.

*Uma informação que é importante destacar é que o conceito de software livre se opõe às diretrizes do software proprietário, mas não às do software que é vendido almejando lucro (software comercial).*

## Distribuições Linux

As distribuições Linux são, de fato, sistemas operacionais que foram criados utilizando o kernel Linux. Cada distribuição Linux, portanto, possui a sua própria "cara", sua tecnologia de interface gráfica, seu desktop environment, seu gerenciador de pacotes (falaremos mais sobre isso adiante), dentre outras coisas.

Além disso, diferentes "distros" (um apelido carinhoso para as distribuições Linux) podem demonstrar diferentes focos e, portanto, serem mais adequadas para determinados tipos de usuários. Por exemplo, algumas distros focam mais em estabilidade, como é o caso do Debian, por exemplo. Por outro lado, ele demora mais tempo para atualizar os softwares. Já outras, como o Manjaro, OpenSUSE ou KDE Neon, visam disponibilizar as versão mais recentes dos softwares e ambiente gráfico e, por isso, podem apresentar algumas pequenas instabilidades. Outras são focadas em desenvolvimento, jogos, dentre outras coisas.

Outro ponto importante a destacar é que, cada distro se utiliza de um ambiente gráfico. Diferentes ambientes gráficos vão nos levar a diferentes *workflows*, diferentes usabilidade. Ambientes gráficos com poucos recursos visuais (como animações) tendem a ser mais leves para a GPU e consomem menos memória RAM. Os principais ambientes gráficos são o GNOME, KDE, XFCE, MATE Deepin Desktop, Pantheon, Cinnamon e Bugdie. Algumas distros disponibilizam versões com diferentes ambientes gráficos, como é o caso do Ubuntu.

Falando em Ubuntu, essa é a distribuição Linux mais popular nos desktops. Por esse motivo, utilizaremos ela nas nossas aulas. No entanto, existem outras distribuições (você pode, inclusive, optar por outra, caso queira). Como exemplo, podemos citar: Debian, Manjaro, Linux Mint, Zorin OS, Elementary OS, Pop!\_OS, OpenSUSE, RegataOS, Kali, MX Linux, Fedora, Deepin, Solus, dentre outras.

Estrutura de Arquivos do Linux (Ubuntu)

Muita gente está acostumada apenas com a estrutura de arquivos do Windows, e por isso se surpreendem ao abrir o disco de instalação do Ubuntu e notar que não existe um disco C:.

Entender essa estrutura de arquivos vai te ajudar a conseguir administrar melhor os servidores e máquinas que utilizam alguma distro Linux, o que pode te favorecer na hora de lidar com problemas no momento do deploy de alguma aplicação. Vamos entender o que cada pasta dessas significa.

Antes disso, algumas informações importantes:

As pastas que apresentam um ícone de seta são apenas links simbólicos, ou seja, são pastas que fazem referência a outras pastas.

As pastas/arquivos que contém um ícone de "X" neles não podem ser sequer acessados para leitura sem que você esteja como usuário root. De um modo geral, os arquivos desse diretório não podem ser alterados, mas alguns deles podem ser acessados sem privilégios de root.

Esse padrão de estrutura de diretórios é compartilhado entre a maioria das distribuições Linux.

# Entendendo os diretórios da pasta raiz do sistema

**/bin** - Refere-se à binary ou binário. Este diretório contém executáveis de softwares do sistema. De um modo geral, você pode comparar com a pasta "Arquivo de Programas" do Windows, com a diferença de que temos apenas os executáveis dos programas. Seria como se só tivéssemos os arquivos .exe do Windows.

**/boot** - Uma pasta muito importante, portanto, nunca mexa nessa pasta, caso não seja realmente necessário. Ela contém os arquivos necessários para o seu sistema iniciar (bootloader, GRUB, etc).

**/cdrom** - Um diretório um tanto legado. Porém, se você tiver um driver de CD/DVD no seu computador, a imagem desse disco será criada aqui. Ou seja, ao inserir o disco, os seus dados irão aparecer nessa pasta.

**/dev** - Abreviação de devices. Nessa pasta você consegue encontrar arquivos que correspondem ao seu hardware. Esses arquivos contêm as configurações que determinam como o seu hardware funciona (ou deve funcionar). Por isso, quando você instala uma distro, o nome de cada partição é /dev/sdaX, em que X é um número inteiro. O que acontece é que o seu HD/SSD é catalogado também nessa pasta.

Uma curiosidade sobre esse diretório, é que nele existe uma espécie de "buraco negro", que seria o /dev/null. Se você mover qualquer arquivo/pasta para esse diretório ele vai, simplesmente, sumir (literalmente).

**/etc** - Pode significar edit to config ou, de fato, a palavra etcetera (que abreviamos para etc). A ideia é que nessa pasta fica todas as configurações que são gerais, ou seja, de todos os usuários. As configurações que são específicas de um único usuário ficam em outro diretório que veremos adiante.

**/home** - Vem de "lar" mesmo. É uma das pastas mais importantes para um usuário, porque contém os dados de cada um dos usuários do sistema. Ou seja, nessa pasta você deve encontrar uma pasta com o nome do usuário que você utilizou na instalação do sistema. Dentro da pasta com o nome do usuário você vai encontrar as pastas Documentos, Downloads, Imagens, Vídeos, etc, para o mesmo usuário.

Além disso, essa pasta do usuário contém outras informações bem importantes que estão como arquivos ocultos. Esses arquivos se referem as configurações de cada aplicativo e do sistema como um todo, para aquele usuário específico.

**/lib, /lib32, /lib64, /libx32** - Todas essas pastas contêm bibliotecas de softwares e do sistema. Ao instalar um software, esse pode adicionar as libs (bibliotecas) necessárias para a sua execução dentro dessa pasta do sistema. As libs podem ser comparadas aos arquivos dll do Windows.

**/media** - É nessa pasta que irão aparecer todos os dispositivos removíveis que serão montados no sistema, ou seja, pen drives, HDs externos, dentre outros.

**/mnt** - É um diretório muito parecido com o /media. É abreviação de mount. É pensado para ser um ponto de montagem de unidades de disco feitas pelo usuário manualmente.

**/opt** - O nome se refere à optional. De modo geral, esse diretório contém arquivos que são criados por softwares proprietários que desejam organizar todas as suas informações principais em um mesmo diretório.

**/proc** - Vem de processes (processos). Um diretório no qual você encontra informações sobre o sistema e processos que são executados por ele. Esses arquivos não existem, de fato, na sua máquina; eles são criados sempre que você inicia o seu computador e, com isso, os processos vão iniciando. Cada processo possui um arquivo dentro desse diretório.

**/root** - Igual ao diretório /home, mas focado no usuário root.

**/run** - Vem de runtime. É um outro diretório que, na verdade, não existe fisicamente no seu computador, mas é criado quando o sistema inicia, e é apagado sempre que o computador desliga. Guarda informações desde o último boot, como usuários que foram logados, por exemplo.

**sbin** - Bem parecido com o bin. A sigla representa system binary. Refere-se a binários de programas que só podem ser acessados pelo administrador do sistema, como o comando useradd que só pode ser utilizado como usuário root. Uma forma legal de associar, seria pensando no s como sendo sudo, ou seja, seriam sudo binaries.

**/snap** - Um diretório bem recente e contém os dados dos pacotes snaps que foram instalados no sistema.

**/srv** - É a abreviação de services. Se você estiver rodando algum Servidor Web ou FTP, você pode armazenar nessa pasta os arquivos que podem ser acessíveis por outros usuários.

**/sys** - É a abreviação de system. É onde estão os drives do sistema e por meio dela você pode interagir com o kernel do sistema.

**/tmp** - É uma pasta de arquivos temporários. É aqui onde programas pode armazenar arquivos que serão utilizados apenas durante uma sessão. Esses arquivos serão excluídos após uma reinicialização. Mas não é bom salvar arquivos muito pesados ou muitos arquivos aqui, porque a sua memória pode ficar muito cheia.

**/usr** - Pode significar user ou unix system resources. Anteriormente os usuários do sistema ficavam nessa pasta, porém, ela mudou um pouco de função e dentro dela ficam arquivos de configuração dos usuários, mas que não são tão vitais para o funcionamento deles.

**/var** - A abreviação vem de variables. Portanto, dentro dessa pasta estarão contidos os arquivos que se espera que irão aumentar de tamanho com o tempo, como as runtimes dos pacotes flatpacks, por exemplo.

# Principais Comandos do Terminal Linux

Agora veremos os comandos mais importantes e que, portanto, você deve conhecer do Terminal Linux. Temos que reconhecer que a maioria das atividades que precisamos realizar em uma distribuição Linux já é possível de ser feita por meio da interface, afinal o Linux evoluiu bastante nesse quesito de interface nos últimos anos. No entanto, conhecer e dominar o terminal Linux é importante, porque em alguns casos você só vai ter o terminal para interagir (como é o caso dos servidores, por exemplo) e, além disso, com o tempo você vai ver que algumas atividades podem ser realizadas com mais facilidade e agilidade por meio do próprio Terminal, o que vai te garantir mais produtividade, além de trazer mais detalhes a respeito dos processos que estão por trás da sua tarefa.

Por isso, vamos conhecer um pouco dessa ferramenta e os seus principais comandos. Mas, antes, gostaria de destacar que alguns desses comandos podem ser diferentes, dependendo de qual distribuição você está utilizando. Para o Ubuntu e quaisquer outras distribuições que sejam baseadas nele ou no Debian, todos os os comandos apresentados aqui irão funcionar perfeitamente.

## Entendendo as informações que aparecem no Bash

Nosso primeiro passo será entender as informações que aparecem assim que a gente inicia o terminal (aproveitando, você pode iniciar o terminal com as teclas Ctrl + Alt + T). Note que, ao iniciá-lo, já temos algumas informações.

As informações que aparecem são:

1. **Nome do usuário** que você definiu no momento da instalação. Observe que no caso da imagem acima o nome de usuário é walisson. A informação do nome do usuário também pode ser obtida por meio do comando whoami.
2. O arroba (@) é símbolo que pode ser traduzida como "está logado em".
3. A informação da máquina na qual você está logada(o). Nesse caso, pop-os.
4. O til (~) representa o diretório /home. Ou seja, ao abrir o terminal, você sempre vai estar no diretório home.
5. O $ indica que você está utilizando o terminal no modo de usuário comum, e não root. Se você estiver utilizando no modo root, o $ será substituído por um #. Para entrar no modo root, você pode digitar o comando sudo su.

## Comandos relacionados aos diretórios do sistema

1. ls: comando utilizado para listar (daí a singla ls) os arquivos que estão contidos no seu diretório atual.
2. ls -l: lista os arquivos do diretório atual com mais informações, como data de modificação, usuário que modificou/criou o arquivo, dentre outras informações.
3. ls -a: lista todos os arquivos, incluindo os arquivos ocultos (que iniciam com um ponto).
4. cd: significa change directory, ou seja, é um comando para alterar o diretório em que você se encontra. Se executado sem um complemento, irá te levar para o diretório /home. O complemento desse comando pode ser qualquer caminho (path) para uma outra pasta do disco; ou apenas .. para retornar para a pasta anterior à atual.
5. mkdir: comando para criar uma pasta (make directory).
6. rmdir: comando para remover uma pasta (remove directory). Funciona apenas se a pasta estiver vazia.
7. rm: comando utilizado para remover pastas ou arquivos. Para remover uma pasta/direrório você deve utilizar o comando da seguinte forma: rm -r nome\_da\_pasta.
8. pwd: comando para exibir o diretório (estrutura das pastas) atual. Vem do inglês, print working directory.

## Comandos relacionados à arquivos

1. touch: comando para criar um arquivo qualquer. Por isso, você deve utilizar esse comando sempre acompanhado do nome do arquivo e a extensão dele. Por exemplo: o comando touch exemplo.py irá criar o arquivo exemplo.py.
2. nano: comando utilizado para editar um arquivo dentro do próprio terminal. Sempre deve vir acompanhado do nome do arquivo, por exemplo: o comando nano exemplo.py irá abrir o arquivo exemplo.py no modo de edição.
3. cat: comando utilizado para exibir no terminal o conteúdo do arquivo. Sempre deve vir acompanhado do nome do arquivo, por exemplo: o comando cat exemplo.py irá exibir o conteúdo do arquivo exemplo.py.
4. rm: remover um arquivo. Por exemplo, para remover o arquivo exemplo.py, podemos utilizar o comando rm exemplo.py.
5. mv: mover um arquivo de um diretório para outro diretório. Por exemplo, o comando mv arquivo1.txt Documentos/ vai **mover** o arquivo1.txt para a pasta Documentos.
6. cp: copiar um arquivo de um diretório para outro diretório. Por exemplo, o comando cp arquivo1.txt Documentos/ vai **copiar** o arquivo1.txt para a pasta Documentos.

## Comandos para instalar/atualizar/remover programas

1. sudo apt update: Comando utilizado para atualizar a lista de repositórios do sistema. O apt é o gerenciador de pacotes do Ubuntu e Debian. Esse comando irá atualizar a lista de repositórios para verificar se existem atualizações de softwares para serem feitas ou irá apenas manter a lista atualizada para que, quando você instalar um novo pacote, ele já esteja em sua versão mais recente.
2. apt list --upgradable: esse comando irá listar os pacotes/softwares que estão disponíveis para atualizar.
3. man apt: mostra um manual detalhado do gerenciador de pacotes apt.
4. apt --help: mostra os principais comandos do apt, bem como uma breve descrição de cada um deles.
5. sudo apt upgrade: atualiza o sistema instalando e/ou atualizando os pacotes/softwares.
6. sudo apt full-upgrade: atualiza o sistema instalando, atualizando e/ou removendo os pacotes/softwares.
7. sudo apt install pacote: comando para instalar um pacote/software no sistema. Para ver a lista de todos os pacotes disponíveis para instalação no apt você pode utilizar o comando apt list, ou pode pesquisar por um pacote específico com o comando apt search pacote\_desejado.

## Outros comandos

1. free: este comando irá apresentar quanto o seu sistema está utilizando de RAM e quanto está disponível. Para ficar melhor de visualizar os valores da memória, basta utilizar a flag -h, ou seja, free -h trará a mesma informação, mas uma forma mais legível para **h**umanos.
2. df: do inglês, disk free. Sendo assim, esse comando exibe quanto você tem de espaço disponível em disco.
3. du: do inglês, disk usage. Sendo assim, esse comando exibe cada arquivo ou diretório está ocupando no disco de armazenamento. Para ficar resumido e mais legível para humanos, você pode utilizar a flag -sh. Por exemplo, du -sh Documentos/.

## Tipos de comandos do Terminal

1. Built-in: comandos internos do bash.
2. Comandos do sistema operacional, que pode ser definido como variável do Bash ou ZSH.
3. Alias: apelidos que você pode dar para comandos, a fim de facilitar a sua utilização desses comandos. Para visualizar os alias que você já tem no seu sistema você pode utilizar o comando alias.

*Para saber qual é o tipo de comando que você está utilizando, você pode utilizar o comando type, por exemplo:*

* *O comando type ls retorna "ls is aliased to 'ls --color=auto' "*
* *O comando type cp retorna "cp é um comando interno do interpretador".*

## Dicas

* Você pode limpar o terminal utilizando o comando clear, ou apenas o atalho Ctrl + L.
* Você pode utilizar as setas do teclado (para cima e para baixo) para navegar pelo seu histórico de comandos utilizados.

# Git e GitHub

Hoje em dia, em geral, ninguém programa sozinho. O git é uma ferramenta criada para facilitar na colaboração distribuição de códigos. O git pode ser usado em servidores privados, mas no geral, os códigos são enviados para o GitHub, um site que hospeda os códigos e que também tem uma interface gráfica simples, que facilita a visualização de mudanças feitas por outras pessoas.

## Começando

O git é instalado como qualquer outro programa, seja por um instalador do Windows ou pelo package manager do linux, pelo comando sudo apt-get install git (dependendo da sua distribuição). Depois de instalado, ele já pode ser usado em um terminal, independente do sistema operacional. Uma das primeiras coisas a se fazer é configurar sua conta do GitHub. Isso é importante para seus commits serem mais facilmente identificados. Para isso, é preciso executar 2 comandos:

git config --global user.name "Seu Nome"

git config --global user.email seuemail@exemplo.com

Depois disso, já podemos começar a usar o git.

## Importando um repositório

Cada projeto tem seu repositório, como é chamada a \*"pasta"\* que guarda os arquivos no servidor. Para começar a trabalhar em qualquer repositório, precisamos importar ele no seu computador. Há dois modos de fazer isso: usando HTTPS e usando SSH. Uma vantagem de HTTPS é que não é necessário configurar nada previamente no sistema e no perfil do GitHub. Uma desvantagem, porém, é que será necessário digitar seu usuário e senha toda vez que aplicar as mudanças locais no repositório online (git push). Já usando SSH, não será necessário digitar as informações de sua conta toda vez que executar git push, mas é preciso configurar algumas coisas no computador antes (lembrando que, provavelmente, só será necessário fazer essa configuração uma vez, e não é difícil).

### Usando HTTPS

1. Navegue até a pasta em que você quer importar o repositório, usando o comando cd (uso: cd <caminho/para/pasta>).
2. git clone <URL\_do\_repositório>. Esse URL pode ser obtido indo até o repositório no GitHub, e clicando em Clone or Download. Por padrão, a janela mostrada é clonar com HTTPS, mas caso esteja escrito Clone with SSH no título dessa janela, clique em Use HTTPS.

No caso desse repositório, o comando seria:

git clone https://gitlab.com/letscode-academy/sitelc.git

### Usando SSH

Para usar SSH, é necessário gerar uma chave SSH (ou usar uma existente) e adicionar essa chave à sua conta no GitHub. [Esse guia](https://help.github.com/articles/connecting-to-github-with-ssh/) explica como fazer isso e testar se a configuração está correta. Feita essa configuração, o processo de clonar o repositório é basicamente o mesmo.

1. cd <caminho/para/pasta>.
2. git clone <URL\_do\_repositório>. Esse URL pode ser obtido indo até o repositório no GitHub e clicando em Clone or Dowload. Em seguida, clique em Use SSH (se o título dessa janela for Clone with SSH, já está certo). O endereço mostrado é o URL do repositório necessário.

No caso desse repositório, o comando seria:

git clone git@gitlab.com:letscode-academy/sitelc.git

## Trabalhando com o repositório

Agora que você já tem uma cópia do repositório no seu computador, já podemos começar a trabalhar com o código, fazendo as alterações que quisermos. No entanto, é sempre bom executar um git pull (que baixa as últimas alterações no repositório) antes de começar a trabalhar(para evitar conflitos por seu repositório estar desatualizado). Depois que tivermos feitas as mudanças e estivermos satisfeitos com o código, podemos enviá-lo para o GitHub.

O git trabalha da seguinte forma: você tem uma cópia local do repositório, na qual ficam os arquivos e você faz alterações. Depois, você adiciona essas mudanças para a staging area, onde elas ficam prontas para serem enviadas em um "bloco", o commit, para o repositório. Caso você queira enviar também as mudanças para um repositório remoto, é preciso usar o comando push.



Primeiro, precisamos adicionar as nossas mudanças para a staging area. Para vermos essas mudanças, usamos os comandos git status (que mostra um resumo) ou git diff (que mostra as mudanças linha por linha). Para realmente adicionar no commit, usamos o comando git add. Com esse comando, podemos selecionar especificamente quais arquivos ou pastas queremos adicionar no commit, ou podemos usar git add . ou git add -A para adicionar todos de uma vez. Usando o comando git diff --cached ou git diff --staged, podemos ver as mudanças feitas apenas nos arquivos que já estão na staging area, ou seja, que vão ser enviados no próximo commit.

Depois disso, para criar um commit, usamos o comando git commit. Esse comando abre uma janela do editor de texto padrão do sistema operacional, para adicionar uma mensagem de resumo do commit. Esses editores costumam ser meio difíceis de usar, então, pessoalmente, gosto de o modificador -m, que permite enviar a mensagem diretamente. O comando fica assim:

git commit -m "Essa é a mensagem do commit"

Os commits são locais, ou seja, eles são só da sua máquina. Eles normalmente são enviados um por um para o servidor, mas você pode usá-los pra se organizar, sem enviar. Depois que você tiver feito quantos commit quiser, e achar que está pronto para enviar seu código para o GitHub, basta executar o comando git push. No caso de algum erro, provavelmente seu repositório local está desatualizado, e isso pode ser resolvido com um pull. Caso haja problemas no auto-merge, eu ensino a resolver na parte final da aula. Caso não dê nenhum erro, o seu código já está no GitHub. Em resumo, um commit fica normalmente assim:

git add .

git commit -m "Mensagem"

git push

PS: Podemos adicionar diretamente ao commit o modificador -a, que faz a função do add. No entanto, isso só funciona para modificações de arquivos, sendo que para arquivos que foram criados nesse commit o comando add ainda é necessário

## Usando branches

O que passei até agora é o básico do GitHub. É o necessário para trabalhar em uma única "versão" do código por vez. No entanto, no git existe a possibilidade de se fazer alterações no código (e compartilhar com outros usuários), sem comprometer um código que já está funcionando. O código do branch master (o principal, e o único que existe por padrão) deve ser sempre estável e funcional, com testes sendo feitos em outros branches.



Para criar um novo branch, primeiro execute um pull, para garantir que os arquivos estão atualizados. Então, execute git checkout -b [Nome do branch]. O modificador -b serve para criar um novo branch. Agora, você está em um branch diferente do master, podendo fazer mudanças e commits sem alterar o código original. Para mandar esse branch para o GitHub e criar um branch remoto, usamos o comando git push origin -u [Nome do branch]. Depois da primeira vez, podemos usar o push direto(desde que estejamos no branch certo).

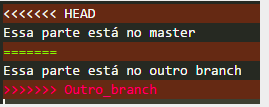
Para mudar de branch, podemos usar o comando git checkout [Nome do branch]. Tome cuidado antes de fazer isso, qualquer mudança que você tiver feito e não estiver em um commit será jogada fora. Para evitar isso, crie um commit ou use o comando git stash que guarda todas as mudanças em um commit temporário, que pode ser recuperado facilmente, por exemplo, com o comando git cherry-pick [código do commit], que "pula" o estado do repositório para um certo commit. Uma lista de modificações do repositório pode ser acessada com o comando git reflog, e dele, você seleciona o commit criado pelo stash.

Se você estiver satisfeito com o trabalho de um branch, e quiser juntá-lo ao master, basta voltar para o master e usar o comando git merge [Nome do branch]. No entanto, caso a mesma parte de um mesmo arquivo tenha sido modificada nos 2 branches, o auto-merge vai falhar, e te avisar quais os arquivos problemáticos. (o mesmo pode acontecer quando você altera coisas sem antes dar pull). Nesses arquivos, ficam as duas versões diferentes daquela parte. Você pode escolher uma delas, e então usar o add e criar um commit para resolver o problema. Agora, você já pode deletar o seu branch antigo, pois tudo está no master.

Com o comando git branch -a podemos ver todos os branches, locais ou remotos, do nosso repositório. Para deletar um branch local, usamos git branch -d [Nome do branch]. Já para deletar um branch remoto, usamos git push origin :[Nome do branch](note os dois pontos).

## Resolvendo conflitos de merge

Quando você tenta dar um merge em dois branches ou dar pull nas mudanças remotas do seu repositório, o git tenta misturar as duas versões do arquivo. No entanto, se tiver alterações na mesma parte do programa nos dois arquivos, o auto-merge vai falhar, e você vai ter que resolver os problemas manualmente. O git vai te avisar quais os arquivos problemáticos, e em cada um deles vai criar uma estrutura assim:



A parte de cima é a que está no seu arquivo original, e a parte de baixo é a que estava no outro arquivo, seja o de um outro branch ou o que veio de um repositório remoto depois de um pull. Para resolver, você tem que escolher a versão que você quer, ou uma mistura das duas. Então, você precisa remover os marcadores do conflito (<<<<<<<, ======= e >>>>>>>). Depois de fazer isso em todos os conflitos, você precisa usar o git add e o git commit para criar um commit com essas mudanças, e o conflito está resolvido.

## Aliases

Aliases são atalhos para comandos do git. Esses atalhos são úteis principalmente para comandos muito longos ou comandos que você não considera intuitivos. Há duas formas de criar aliases:

1. Usando o comando git config --global alias.<atalho> <comando>.
2. Editando o arquivo .gitconfig no diretório home (~), adicionando linhas dessa forma:

[alias]

<atalho1> = <comando1>

<atalho2> = <comando2>

(...)

Exemplos: Executar o comando git config --global alias.graph log --all --graph --decorate --color --oneline ou editar o arquivo .gitconfig, adicionando o abaixo:

[alias]

graph = log --all --graph --decorate --color --oneline

permite executar o comando git log --all --graph --decorate --color --oneline digitando apenas git graph.