Capítulo 1. Introdução ao Ubuntu Linux

Frederico Schmitt Kremer [1], Luciano da Silva Pinto [1]

[1] *Laboratório de Bioinformática e Proteômica, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas*

Objetivos: Apresentar as principais características e funcionalidades dos sistemas operacionais UNIX / Linux, incluindo exemplos de linhas de comando de uso geral.

Sistema operacional Linux / UNIX

No final dos anos 60, os pesquisadores Ken Thompson, Dennis Ritchie, Douglas McIlroy e Peter Weiner desenvolveram um **sistema operacional** multiusuário e multitarefa denominado **UNIX**, enquanto trabalhavam nos laboratórios Bell da AT&T, a principal empresa de telefonia dos Estados Unidos. Ao longo dos anos, este sistema se tornou amplamente difundido tanto no meio no meio corporativo quanto acadêmico, sobretudo devido a sua flexibilidade e robusteza, além de por já trazer um conjuto rico de ferramentas e funções para processamento de dados. O desenvolvimento deste sistema foi um marco na computação, não apenas pelas suas funcionalidades, mas por ter sido durante o seu desenvolvimento, dentro do mesmo grupo de trabalho, que surgiu a **linguagem C**, a base para a maioria das linguagens de programação modernas, os interpretadores de comandos **AWK** e **Sed**, que por sua vez são a base da linguagem de programação Perl (amplamente usada em Bioinformática), e por ter popularizado o uso das **expressões regulares** através de ferramentas como o **grep**.

Por ter sido desenvolvido dentre de um laboratório privado, por muito otempo UNIX foi um **programa proprieatário** e de **código fechado**, o que significa que apenas os seus desenvolvedores poderiam modificar o seu código fonte e seu uso era mediado por licenças, muitas das vezes pagas. Para muitos programadores, o modelo de código fechado trás diversas desvantagens para a manutenção do programa, uma vez que certos erros podem demorar muito tempo para serem corridigos, além de tornar o usuário completamente dependente dos desenvolvedores originais.

Nos anos 80, com o advento de movimentos de apoio ao chamado **software livre**, como o **projeto GNU** (*GNU is Not UNIX*, GNU não é UNIX“), muitos programadores iniciaram o desenvolvimento de novos sistemas operacionais mantendo em mente o conceito de **código aberto**, ou seja, de que o código-fonte de um programa deve estar disponível para que outras usuários possam não apenas usar livremente o programa, mas também entender como ele funciona e modifica-lo, caso necessário. Neste contexto, **Linus Torvalds**, um programador finlandês, iniciou o desenvolvimento de um **núcleo de sistema** (*kernel*) basedo no UNIX (*UNIX-like*), denominado Linux (“*Linus UNIX*”).
Ao mesmo tempo, colaboradores do projeto GNU criaram um pacote de programas que imitava boa parte das funcionalidades presentes no UNIX, incluindo os programas para compilação de códigos, processadores de texto e gerenciamento de arquivos. Estes programas foram rapidamente incorporados ao Linux, sendo hoje elementos praticamente inseparáveis de seu ecossistema.

Por ser de código aberto e de livre distribuição, qualquer usuário pode baixar o código do Linux e modificar o sistema de acordo com as suas necessidades. Desta forma, com o passar dos anos, diferentes grupos de desenvolvimento criaram **distribuições** modificadas do sistema usando como base o kernel Linux, sendo estas denominadas "**distros**". Atualmente existe diversas distros disponíveis, sendo **Ubuntu**, **Fedora** e **OpenSUSE** algumas das mais populares.

Além de ser utilizado com *notebooks* e *desktops*, o Linux é amplamete difundido em praticamente todos os dispositivos. O sistema operacional Android, por exemplo, amplamente usado em *smartphones*, foi criado com base no kernel escrito por Torvalds. O Linux também é amplamente utilizado em servidores, estações de trabalho, *data centers* e *clusters* de processamento de alto desempenho, visto que sua flexibilidade e possibilidade de customização / adaptação permite que este seja migrado para diferentes arquiteturas de processadores.

A fato de ter um código aberto também implica em diversas vantagens no que diz respeito a segurança. Quanto mais colaboradores contribuem para a escrita e correção do seu código-fonte, menor a chance de erros passarem despercebidos, o que inclui possíveis vulnerabilidades. Esta maior segurança é também um dos principais motivos para o Linux e outros sistemas operacionais UNIX-like serem amplamente adotados em servidores *Web* e servidores de sistemas bancários, por exemplo.

Ubuntu Linux

O Ubuntu é uma distro do Linux desenvolvida pela empresa **Canonical** e pode ser obtido gratuitamente através do seu *website* (www.ubuntu.com/), sendo disponibilizado em versões para *desktop* / *notebook*, servidor e *smartphone*. Atualmente o Ubuntu é uma das distribuições mais populadores para uso doméstico, acadêmico e empresarial, sobretudo por ter um grande grupo desenvolvimento e suporte, uma **interface amigável** mesmo para usuários com pouco experiência em informática, suporte uma grande variedade de **arquiteturas de *hardware*** e por possuir um rico **repositório de programas**.

Shell

As sistemas operacionais Linux, da mesma forma que o Microsoft Windows e o Apple OS, podem ser utilizados tanto através de uma interface gráfica (GUI) quando através de uma interface de linha de comando (CLI). Interface de linha de comando, apesar de pouco intuitivas em um primeiro momento, oferecem muitas vantagens para a **manipulação de arquivos**, **automatização de rotinas** e **gerenciamento de processos**.
Os comandos passados através de uma interface CLI são processados por um interpretador, denominado *shell*. No caso do Windows, por exemplo, o "Prompt de comando" ("cmd.exe", ou popularmente chamado de "DOS"), e o Windows Powershell são exemplos de shells já disponíveis no sistema e que podem ser usados para automarizar algumas tarefas. Já no caso do Linux, de acordo com a distribuição que está sendo utilizada, diferentes interpretadores de comandos podem estar configurados. No caso do Ubuntu, por exemplo, o sistema utiliza como padrão o interpretador Bash (https://www.gnu.org/software/bash/), que oferece diversas funcionalidades, o que

inclui um ambiente rico como suporte inclusive à variáveis e estruturas lógicas de programação (*shellscript*), que serão melhor apresentadas no **próximo capítulo**.

Em sistemas que utilizam a interface gráfica como padrão o uso do shell é normalmente feito através de um programa deminado **terminal**, que serve de intermediário entre o usuário e o interpretador de comandos.

Usuários

Como dito anteriormente, o Linux é um sistema multi-usuário, o que significa que ele permite que diferentes usuários utilizem o mesmo sistema, na mesma máquina, simultaneamente, e tendo cada um **permissões e restrições específicas**. Para cada usuários que é configurado no sistema, uma pasta é criada no diretório `/home/`. O acesso e o direito de manipulação destas pastas e seus conteúdos é restrito ao seu manipulador, a menos que este libere para os demais. Além disso, cada sistema possui um usuário que detêm **acesso total** as configurações do sistema, sendo denominado `root`. Virtualmente, qualquer ação que afeta o sistema de forma permanente precisa de permissões de `root` para ser executada. Exemplos destas ações são instalação / remoção de programas, configurações de redes, atualização do sistema operacional e **alteração de diretórios de acesso restrito.**

Um usuário com direitos de `root` possui direitos de manipular até mesmo pastas que são naturalmente restritas ao sistema, o que faz com que a utilização destas permissões exija cuidado. Não são raros os casos de má utilização das permissões de `root`, que pode inclusive levar a completa inutilização do sistema operacional. Desta forma, estas devem sempre serem utilizadas com **parcimônia**.

Iniciando uma sessão

Atualmente, pelo menos as distribuições mais "populares", toda sessão do Linux é inicializada já com a interface gráfica carregada, sendo mostrada a **Área de Trabalho** (*desktop*) e os arquivos e pastas que nela estão armazenados (Figura 1). Cada usuário possui sua própria área de trabalho, e como falado anteriormente, o acesso à esta é permitido, *a priori*, apenas ao seu respectivo usuário e ao `root`.

Após a sessão do usuário ser inicializada, é possível acessar o **terminal** através das teclas de atalho `Ctrl + Alt + T` (Figura 2). Ao iniciar uma sessão no terminal, sempre o usuário será direciono para a **sua respectiva pasta**. Desta forma, caso o nome do usuário seja "`user1`", a sua pasta de usuário será consequentemente "`/home/user1/`", e será este endereço (ou caminho) que o terminal usará como ponto de partida.

Figura 1. Exemplo da tela de entrada de um *desktop* com um sistema Linux Ubuntu utilizando o ambiente gráfico XCFE.

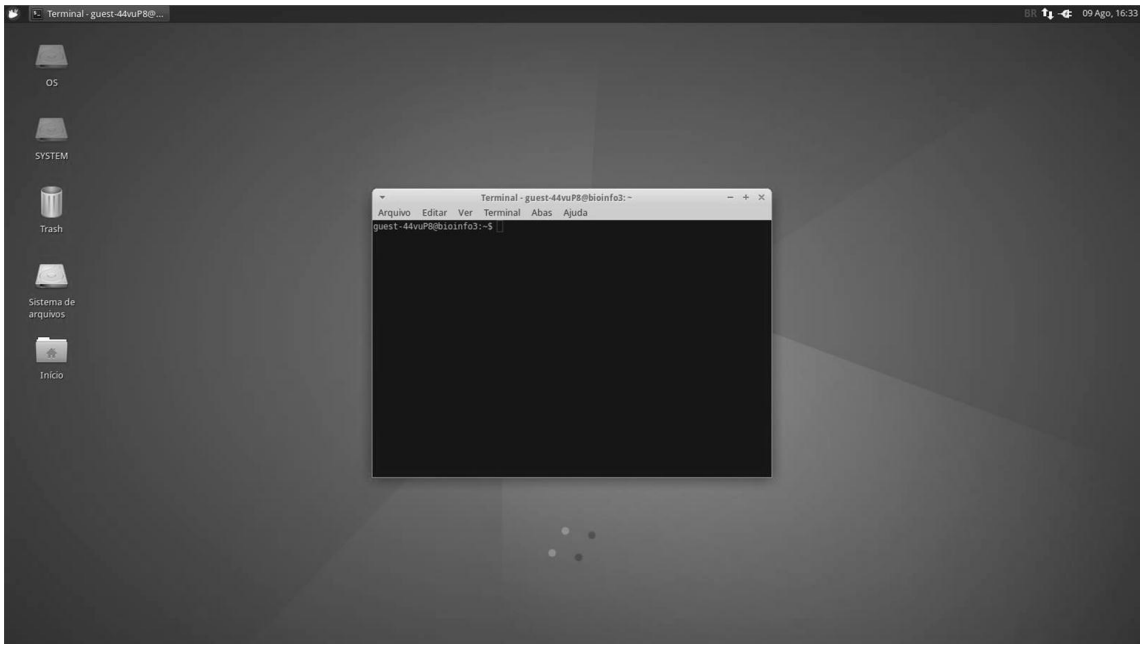

Figura 2. *Desktop* de um sistema Linux Ubuntu com o terminal aberto.

Quem sou eu? (comando: `whoami`)

Caso não estejamos utilizando a nossa própria máquina, é possível que tenhamos dúvida de qual o nome do usuário que está sendo utilizado no momento. Em alguns casos, também pode ser interessante obter essa informação para que um determina programa saiba se uma determinada ação é permitida ou não. No Linux, através da linha de comandos, é possível se obter esta informação com o comando comando `whoami` (eng:"*Who am i?*"; pt: "Quem sou eu?").

```
$ whoami
```

```
user1
```

Aonde estou? (comando: `pwd`)

Da mesma forma que é possível se saber "quem somos", também é possível se saber "aonde estamos". Quando estamos utilizando a linha de comando, frequentemente temos de passar de uma pasta para outra, criar novos diretórios, deletar diretórios, e assim por diante. É possível se saber o caminho do diretório no qual estamos através do comando `pwd` ("pwd" → eng:"*print working directory*"; pt: "*imprimir diretório de trabalho*").

```
$ pwd
/home/user1/
```

Listando o conteúdo de uma pasta (comando: `ls`)

O comando `ls` serve para **listar** o conteúdo de uma determinada pasta, o que inclui os diretórios, arquivos e *links* que estão nela localizados.

```
$ ls
Desktop             Documents             Downloads
Images              Music                 data.txt
```

O resultado deste comando é uma listagem e forma de **grade**, apenas com o nome de cada elemento que está presente no diretório. Para maiores informações sobre estes elementos é possível se utilizar o **argumento** `-l`.

```
$ ls -l

drwxr-xr-x  2 user1 user1   4096 Out  8  2015 Desktop
drwxr-xr-x  2 user1 user1   4096 Out  8  2015 Documents
drwxr-xr-x  2 user1 user1   4096 Out  8  2015 Downloads
drwxr-xr-x  2 user1 user1   4096 Out  8  2015 Images
drwxr-xr-x  2 user1 user1   4096 Out  8  2015 Music
-rwxr-xr-x  2 user1 user1     20 Dec  1  2015 data.txt
```

Mudando de diretório (comando: `cd`)

Como foi falado, através da interface de linhas de comandos é possível se navegar através dos diretórios do sistema, sendo esta função é realizada através do comando `cd` ("cd" → eng:"*change directory*"; pt: "*mudar de diretório*"). Para usar este comando é necessário se passar como argumento o nome do **diretório de destino**, podendo este ser passado no forma de um caminho **absoluto** ou **relativo**. Um caminho relativo tem este nome pois parte sempre do atual diretório de trabalho, enquanto que o caminho absoluto usa como referencial a **raiz do disco**.

Entrando em um diretório através do **caminho relativo**:

```
$ cd Desktop
$ pwd
/home/user1/Desktop
```

É possível também se utilizar também o comando `cd` para voltar um ou mais diretórios, através da notação `cd ..`

```
$ pwd
/home/user1/Desktop
$ cd ..
/home/user1
```

Entrando em um diretório através do **caminho absoluto**:

```
$ cd /home/user1/Downloads
$ pwd
/home/user1/Downloads
```

Criando uma nova pasta (comando: mkdir)

O comando `mkdir` (eng:"make directory"; pt: "criar diretório") serve para criar um novo diretório.

```
$ ls
Desktop             Documents             Downloads
Images              Music                 data.txt
$ mkdir NewDirectory
$ ls
Desktop             Documents             Downloads
Images              Music                 NewDirectory
data.txt
```

Removendo arquivos e diretórios (comando: `rm`)

O comando `rm` (eng:"remove"; pt: "remover") serve para deletar arquivos, diretórios e *links* a partir de seus respectivos nomes. A remoção de um arquivo pode ser feita através do comando `rm [nome do arquivo]`.

```
$ ls
Desktop             Documents             Downloads
Images              Music                 NewDirectory
data.txt
$ rm data.txt


$ ls
Desktop             Documents             Downloads
Images              Music                 NewDirectory
```

A remoção de um diretório segue um modelo parecido, mas é necessário informar para o interpretador de comandos que não só o diretório deve ser removido, mas também todos os arquivos e demais diretórios que nele estão presentes. Para isso, é necessário se deixar implícito que a remoção deve ser feita de forma **recursiva**, através do argumento `-r`.

```
$ ls
Desktop             Documents             Downloads
Images              Music                 NewDirectory
$ rm -r NewDirectory
$ ls
Desktop             Documents             Downloads
Images              Music
```

Concatenando arquivos (comando: `cat`)

O comando `cat` (eng:"concatenate"; pt: "concatenar") permite que o conteúdo de diversos arquivos seja mostrado sequencialmente na tela. Para entender isso, tomemos como exemplo dois arquivos:

[1]. **Nome:** `file_1.txt`
Conteúdo: `Este é o conteúdo do arquivo 1.`

[2]. **Nome:** `file_2.txt`
Conteúdo: `Este é o conteúdo do arquivo 2.`

Para se utilizar o comando `cat` é necessário se fornecer o nome dos arquivos que serão impressos na tela, cada um na forma de um argumento separado. Sendo assim, para se unir o conteúdo destes dois arquivos deve ser utilizado o seguinte comando:

```
$ cat file_1.txt file_2.txt
Este é o conteúdo do arquivo 1.
Este é o conteúdo do arquivo 2.
```

Comandos como o `cat` são normalmente acompanhados da geração de um novo arquivo, ao invés de mostrar simplesmente o resultado na tela. Para que o resultado de um programa seja enviado para um arquivo é possível se utilizar a seguinte notação:

```
$ ls
file_1.txt file_2.txt
$ cat file_1.txt file_2.txt > file_3.txt
$ ls
file_1.txt file_2.txt file_3.txt
```

No Linux, uma mensagem impressa na tela pode ser de dois tipo: `stdout` (eg:"standard output"; pt: "saída padrão"). e `stderr` (eng:"*Who am i?*"; pt: "Quem sou eu?").. Mensagens `stdout`

Movendo e renomeando arquivos e diretórios (direcionador: `mv`)

O comando `mv` (eng:“move”; pt: “mover”) serve para mover um arquivo ou pasta de lugar, ou atribuir um novo nome, sendo o seu **caminho** alterado em ambos os casos. O mv recebe dois argumentos, sendo o primeiro o **caminho inicial** e o segundo o **caminho de destino**.

Renomeando um arquivo:

```
$ ls
arquivo.txt
$ mv arquivo.txt dados.txt
$ ls
dados.txt
```

Movendo um arquivo para outra pasta:

```
$ ls
arquivo.txt minha_pasta/
$ mv arquivo.txt minha_pasta/
$ ls minha_pasta/
arquivo.txt
```

Criando uma cópia de um arquivo ou diretório (comando: `cp`)

O comando `cp` (eng:“copy”; pt: “copiar”) permite gerar uma cópia de um detercopiar o conteúdo e as permissões de acesso de um arquivo ou diretório. O comando recebe dois argumentos, sendo o primeiro o nome do arquivo que será copiado e o segundo o caminho para o destino da cópia. O destino pode tanto ser um caminho direto de arquivo como também um diretório, para o qual o arquivo será copiado mantendo-se seu nome original.

Copiando um arquivo:

```
$ ls
arquivo.txt
$ cp arquivo.txt novo_arquivo.txt
$ ls
arquivo.txt novo_arquivo.txt
```

Copiando um arquivo para outro diretório:

```
$ ls
arquivo.txt minha_pasta/
$ cp arquivo.txt minha_pasta/
$ ls
arquivo.txt
$ ls minha_pasta/
arquivo.txt
```

O `cp` também pode ser utilizado para se copiar o conteúdo inteiro de uma pasta para outro local, entretanto, da mesma forma que o comando `rm`, o `cp` requer o argumento `-r` para que a execução seja **recursiva**.

```
$ ls
pasta1
$ ls pasta1/
arquivo1.txt arquivo2.txt
$ cp -r pasta1 pasta2
$ ls pasta2/
arquivo1.txt arquivo2.txt
```

Obtendo o arquivo da internet (comando: `wget`)

Um comando bastante útil presente no Linux é o `wget` (eng:"web + get"; pt: "web + pegar"), que permite fazer o download de um arquivo disponível na rede através de sua URL.

```
$ wget http://labbioinfo.ufpel.edu.br/arquivo.txt
$ ls
arquivo.txt
```

Descompactando um arquivo (comando: `tar`; `gzip`)

O `tar` (eng:"tape archive"; pt: "arquivo de fita *tape*") é um comando que permite juntar vários arquivos em um único, tendo sido criado na época em que os computadores utilizadam rolos de fita magnética (*tape*) para armezenar as informações. Como uma mesma fita poderiam guardar vários arquivos, o formato tar foi criado para padronizar a representação do conteúdo total destas fitas e da delimitação de cada arquivo. Hoje em dia, o `tar` é comumente utilizado em conjuto com **ferramentas de compacção** para reduzir o espaço ocupado por determinados conjuntos de arquivos, como os de código fonte, para tornar a transferência dos dados mais prática.

Para descompactar um arquivo na extensão .tar é utilizada a linha de comando:

```
tar -xvf arquivo.tar
```

O resultado do `tar` é um arquivo com tamanho um pouco maior que a soma do tamanho das suas parte. Desta forma, o arquivo não é de fato "**compactado**", e sim apenas **unido**. Para reduzir o espaço em disco ocupado por este tipo de arquivo, muitas vezes após, ou mesmo durante, a sua execução é também feita a compactação através de ferramentas como o `gzip` ("GNU zip"). O resultado de uma compactação de um arquivo .tar pelo gzip costuma ter a extensão `.tar.gz`, ou apenas `.tgz`.

Para descompactar um arquivo `.tar.gz` / `.tgz` é usado o comando:

```
$ tar -xzvf minha_pasta.tar.gz
$ ls
$ minha_pasta/
```

```
$ ls minha_pasta/
arquivo1.txt  arquivo2.txt  arquivo3.txt
```

O gzip também pode ser utilizado para compactar e descompactar arquivos individualmente. Para descompactar um arquivo, basta utilizar o comando:

```
$ ls
arquivo.txt.gz
$gzip –d arquivo.gz
$ ls
arquivo.txt
```

Fornecendo permissões de `root` (`sudo`; `sudo su`)

Como foi falado, determinadas ações exigem permissões específicas para sendo executadas. Para executar um comando que exige permissões de `root` é possível se conceder temporariamente esta permissão através do comando `sudo`, seguido do comando de interesse. Após chamar o comando, uma mensagem será mostrada pedindo a senha do `root`.

```
$ cp program.exe /usr/local/bin/
permission denied
$ sudo cp program.exe /usr/local/bin/
password:
```

É possível também se aplicar as permissões de root irrestritamente a todos os comandos que forem executados através do comando `sudo su`. Desta forma, não será mais necessário se escrever sudo para cada comando que necessitar de permissões especiais.

```
$ sudo su
password:
$ cp program.exe /usr/local/bin/
$ rm /usr/local/bin/program.exe
```

Compilação (comandos: `./configure`; `make`; `make install`;)

Como dito anteriormente, o Linux nasceu dentro no contexto da filosofia **software livre**, que, dentre outras coisas, prega que a distribuição de um determinado programa de computador deve ser irrestrita e acompanhada de seu respectivo código-fonte. Por conta disso, hoje em dia muitos dos programas desenvolvidos para Linux são distribuidos na forma de códigos que devem ser **compilados** para que o ferramenta seja de fato utilisável. Diferente do Microsoft Windows, o Linux pode ser executado de um amplo espectro de arquiteturas de *hardware*, sendo que cada uma exige um processo de compilação específico.

O processo de compilação consiste em converter um código escrito em uma linguagem de programação (“entendível por pessoas”) em uma **linguagem de máquina** (“entendível pelo computador”). O código de máquina é sempre específico para o sistema operacional e a arquitetura de *hardware* para a qual ele foi desenvolvido, não

sendo executável, a princípio, em demais máquinas. Por causa disso, programas compilados para Windows não funcionam (diretamente) no Linux, e visse-versa.

Existem diversas formas de se compilar um programa no Linux, mas muitos dos programas que são disponibilisados na forma de código-fonte seguem alguns "caminhos comuns" para isso. Muitos dos programas, por exemplo, vem acompanhados de um arquivo `INSTALL`, `README` e `LICENSE`, que contêm informações sobre como deve ser realizado o processo de instalação, sobre a ferramenta propriamente dita e sobre os direitos do **usuário final**, respectivamente.

Como existe um grande número de variantes do Linux, muitos programas são companhados de um *script* de configuração para a instalação, normalmente chamado `configure`. Veremos mais sobre o que são *scripts* no próximo capítulo, mas por hora é importante saber apenas que estes são pequenos programas de computador não-compilados. Muitos vezes a execução deste script é necessária para se realizar a compilação do programa em si. Para executar estes scripts, basta se escrever `./configure`, ou `bash configure`.

```
$ cd pasta_do_programa/
$ ./configure
```

Além dos *scripts* de configuração, também existem *scripts* que gerenciam o processo de compilação do programa. O gerenciador de compilação mais comum em sistemas Linux é o `make`, sendo os seus scripts de compilação denominados "Makefiles". Normalmente, pastas contendo códigos-fonte possuem um `Makefile`, ou este será gerado a partir dos *scripts* de configuração. Para executar estes scripts, basta se chamar o programa `make` dentro da pasta que o contêm.

```
$ cd pasta_do_programa/
$ ls
Makefile   src/     bin/
$ make
```

Muitos programas também oferecem a opção "`make install`", que serve para realizar a configuração dos programas após a compilação. Entretanto, para executar esta opção é necessário se ter permissões de `root`.

Sem permissões de `root`:

```
$ make install
[mensagem de erro] permission denied
```

Com permissões de root:

```
$ sudo make install
```

E o que mais?

O Linux oferece uma grande variedade de ferramentas, além de todas os programas que podem ser instalados. Para aprender mais sobre o sistema, seus comandos, e até mesmo alguns pacotes de programas adicionais, recomendamos o curso *Learn the Command Line*, do Codecademy, que é gratuíto e apresenta várias funcionalidades interessantes da linha de comandos. Em caso de dúvidas sobre a sintaxe de algum comando, é possível se utilizar o site explainshell para entender melhor os diferentes argumentos e a função de cada programa. Por fim, fóruns de discussão como Stackoverflow, Ask Ubuntu e Ubuntu forums.