UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM DESIGN DIGITAL

PEDRO HENRIQUE PEREIRA DE OLIVEIRA

UM ESTUDO SOBRE A ADERÊNCIA A RECOMENDAÇÕES DE USABILIDADE EM INTERFACES DE LINHA DE COMANDO

QUIXADÁ
2026

PEDRO HENRIQUE PEREIRA DE OLIVEIRA

UM ESTUDO SOBRE A ADERÊNCIA A RECOMENDAÇÕES DE USABILIDADE EM INTERFACES DE LINHA DE COMANDO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Design Digital do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Design Digital.

Orientadora:
Prof[a]. Ma. Leonara de Medeiros Braz

QUIXADÁ
2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O49e Oliveira, Pedro Henrique Pereira de.
Um Estudo Sobre a Aderência a Recomendações de Usabilidade em Interfaces de Linha de Comando / Pedro Henrique Pereira de Oliveira. – 2026.
64 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Design Digital, Quixadá, 2026.
Orientação: Profa. Ma. Leonara de Medeiros Braz.

1. Interface de Linha de Comando. 2. Usabilidade. 3. Avaliação Heurística. I. Título.
CDD 745.40285

PEDRO HENRIQUE PEREIRA DE OLIVEIRA

UM ESTUDO SOBRE A ADERÊNCIA A RECOMENDAÇÕES DE USABILIDADE EM INTERFACES DE LINHA DE COMANDO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Design Digital do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Design Digital.

Aprovada em: 20/01/2026

BANCA EXAMINADORA

Profª. Ma. Leonara de Medeiros Braz (Orientadora)
Universidade Federal do Ceará (UFC)

Profª. Drª. Rainara Maia Carvalho
Universidade Federal do Ceará (UFC)

Profª. Ma. Simone de Oliveira Santos
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Agradeço à minha orientadora, Profª. Ma. Leonara de Medeiros Braz, cujo acompanhamento e interesse foram fundamentais. Registro também minha gratidão aos membros da banca examinadora, Profª. Drª. Rainara Maia Carvalho e Profª. Ma. Simone de Oliveira Santos, pelas valiosas observações que contribuíram para o aprimoramento deste trabalho. À Universidade Federal do Ceará, agradeço pelo suporte institucional e pelas condições acadêmicas que possibilitaram este estudo. De forma especial, agradeço também à minha família pelos indispensáveis apoio e incentivo.

RESUMO

Apesar da prevalência do uso das interfaces gráficas do usuário (GUI, *graphical user interface*) pelo público geral, interfaces de linha de comando (CLI, *command line interface*) continuam sendo amplamente utilizadas em contextos acadêmicos e profissionais, principalmente por sua flexibilidade e expressividade. Esses benefícios, porém, estão associados a dificuldades de uso e aprendizagem, especialmente para usuários iniciantes. Embora existam recomendações de usabilidade específicas para CLIs, como o guia *Command Line Interface Guidelines* (CLIG), viu-se necessária a investigação da abrangência deste em relação a princípios gerais de usabilidade, representados pelas heurísticas de Jakob Nielsen. Através de uma análise exploratória das diretrizes do guia, composta por uma comparação entre ele e as heurísticas e um questionário, foi confirmada a necessidade de uma investigação mais detalhada. Deu-se isso através de uma segunda comparação entre as duas listas de recomendações, dessa vez de maneira mais minuciosa. Perceberam-se lacunas conceituais no guia em relação às heurísticas e, como forma de mitigar essas lacunas, foram elaboradas quatro diretrizes adicionais para o guia CLIG.

Palavras-chave: interface de linha de comando; usabilidade; avaliação heurística.

ABSTRACT

Notwithstanding the prevalence of graphical user interfaces (GUI) among the general public, command-line interfaces (CLI) continue to find extensive employment in academic and professional contexts, owing chiefly to their flexibility and expressive power. These advantages, however, are attended by difficulties of use and acquisition, particularly on the part of novice practitioners. Although specific recommendations concerning CLI usability exist, most prominently those set forth in the *Command Line Interface Guidelines* (CLIG), it was deemed requisite to examine the extent to which they comport with general principles of usability, as embodied in the heuristics of Jakob Nielsen. An exploratory analysis of the guidelines, comprising a comparison with the heuristics and a corroborating questionnaire, confirmed the necessity of a more meticulous examination, subsequently effected through a second comparison of the two corpora of recommendations. In this inquiry, certain conceptual lacunæ were discerned in the guide; to attend to these, four additional guidelines were devised for incorporation into the CLIG.

Palavras-chave: command-line interface; usability; heuristic evaluation.

LISTA DE FIGURAS

Figura 1 – Modelo da aceitabilidade de um sistema. 12
Figura 2 – Exemplo da não-continuidade da exibição do estado do sistema. 13
Figura 3 – Fluxograma da metodologia adotada por este trabalho. 22
Figura 4 – Distribuição aproximada das diretrizes CLIG por heurística 23
Figura 5 – Respostas aos treze primeiros itens da terceira seção 25
Figura 6 – Distribuição das diretrizes CLIG/heurística após a segunda comparação. 27

LISTA DE QUADROS

Quadro 1 – Comparação entre os trabalhos relacionados e este. 21
Quadro 2 – Organização do questionário. 24
Quadro 3 – Visão geral dos mapeamentos CLIG–Nielsen 46

SUMÁRIO

1	**INTRODUÇÃO**	9
2	**FUNDAMENTAÇÃO TEÓRICA**	11
2.1	**Usabilidade**	11
2.2	**Interfaces de linha de comando**	12
2.3	**Heurísticas de usabilidade de Jakob Nielsen**	14
2.4	**Command Line Interface Guidelines**	16
3	**TRABALHOS RELACIONADOS**	18
3.1	**Schmitzhaus (2024)**	18
3.2	**Schröder e Cito (2021)**	18
3.3	**Cabot (2017)**	20
3.4	**Análise comparativa**	21
4	**METODOLOGIA**	22
5	**ANÁLISE EXPLORATÓRIA E QUESTIONÁRIO**	23
6	**COMPARAÇÃO ENTRE NIELSEN E CLIG**	27
7	**RECOMENDAÇÕES**	33
8	**CONSIDERAÇÕES FINAIS**	38
	REFERÊNCIAS	40
	APÊNDICE A – Questionário	42
	APÊNDICE B – Comparação entre Nielsen e CLIG	46

1 INTRODUÇÃO

A forma como computadores são operados passou por transformações profundas desde a popularização dos primeiros computadores pessoais. Modelos como o TRS-80, o Commodore PET e o Apple II (lançados 1977) ofereciam interpretadores de BASIC em ROM e acesso a seus sistemas operacionais em disco[1] através de interfaces de linha de comando (CLI, *command line interface*) (Welsh; Welsh, 2007). A computação doméstica, no entanto, só se tornaria realmente acessível ao público leigo com o lançamento do Macintosh 128k em 1984 (Sensomatic, 2025). O emprego de uma interface gráfica do usuário (GUI, *graphical user interface*) por este ampliou o público dos computadores, mas trouxe também limitações. Entre estas estão a menor eficiência de uso e dificuldades na execução de operações mais abstratas (Gentner; Nielsen, 1990) (Norman, 2007).

A despeito da ampla suplantação das interfaces de linha de comando em favor das interfaces gráficas pelo público geral (Preece *et al.*, 2023), diversas variantes de CLIs permanecem em uso extensivo por diferentes tipos de profissionais, bem como usuários comuns com mais experiência. Usuários frequentes desse tipo de interface incluem administradores de sistema, desenvolvedores de *software*, pesquisadores e cientistas de dados (Schröder; Cito, 2021); além desses, podem ser mencionados profissionais de áreas técnicas e exatas, como economistas, físicos, matemáticos, engenheiros e arquitetos (Hunt *et al.*, 2001) (Autodesk, 2010).

Dentre as características das CLIs mais favorecidas pelos profissionais supracitados, está a capacidade de compor funcionalidades, pelas quais ações podem ser executadas em sequência com o objetivo de formar estruturas de controle por intermédio de sintaxe conectiva como *encadeamentos* (do inglês *pipelines*) (Cabot, 2017). Outra característica é a habilidade de criar *scripts*[2], que permite a elaboração de programas por usuários finais, propiciando a estes ferramentas sofisticadas para automatizar seus trabalhos, diminuindo, assim, esforço físico e cognitivo (Ibid.).

No entanto, esses benefícios são associados à relativa dificuldade de uso e aprendizagem, sendo as CLIs geralmente não recomendadas para usuários iniciantes (Dix *et al.*, 2004). Nas CLIs mais populares, comandos devem ser lembrados, pois não são dadas dicas que indiquem quais deles podem ser utilizados (Ibid.). Isso tende a introduzir nesse tipo de interface um problema em que é necessário o reconhecimento ao invés da lembrança, no qual usuários devem memorizar as particularidades da sintaxe e combinações de argumentos, ao invés de receberem pistas sobre o funcionamento através de algum símbolo reconhecível, como em GUIs (Nielsen; Molich, 1990 apud Schröder; Cito, 2021).

[1] Sistemas operacionais em disco (DOS, *disk operating system*) são sistemas operacionais que necessitam dispositivos de armazenameno de acesso direto como armazenamento secundário

[2] De forma geral, em computação, um *script* é um conjunto de instruções relativamente curto e simples que permite automatizar processos que, de outra forma, seriam manuais.

Além disso, é perceptível uma lacuna no debate acadêmico sobre a forma como usuários manuseiam CLIs (Heron, 2015) (Cabot, 2017). Isso pode contribuir para a persistência de práticas pouco eficazes, que dificultam a aprendizagem e o uso eficiente de ferramentas baseadas nesse tipo de interface. Considerando-se a importância das CLIs, a análise crítica da usabilidade destas representa uma oportunidade de fomentar discussões e subsidiar avanços tanto teóricos quanto práticos na área.

Nesse contexto, surge o questionamento sobre a existência de características implícitas às CLIs que ocasionam as limitações de usabilidade supracitadas. Mesmo que existam recomendações específicas para mitigar essas limitações — sendo destas a mais compreensiva e, potencialmente, a mais conhecida, o guia *Command Line Interface Guidelines* (CLIG) — é possível que estas não cubram de forma balanceada princípios gerais de usabilidade. Assim, julga-se ser necessária uma análise dessas recomendações à luz de critérios gerais de usabilidade amplamente adotados, como os originalmente propostos por (Nielsen, 1993c). Essa abordagem pode revelar aspectos subexplorados e contribuir para o aprimoramento da experiencia de uso no tipo de interface em questão.

Tomou-se como objetivo principal deste trabalho examinar a aplicabilidade de principios gerais de usabilidade às CLIs. A partir deste, derivaram-se três objetivos específicos: **1.** identificar características das CLIs que podem comprometer a adoção de boas práticas de usabilidade, **2.** analisar as diretrizes do guia CLIG à luz das heurísticas de Nielsen e **3.** propor recomendações para mitigar possíveis lacunas de correspondência.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Usabilidade

Como definido por Hewett *et al.* (1992), *Interação Humano-Computador* (IHC) é o campo de estudo responsável pelo projeto, avaliação e implementação de sistemas computacionais interativos para uso humano e investigação dos principais fenômenos relacionados a esses sistemas. Preece *et al.* (1994) mencionam que o termo foi adotado em meados da década de 1980 para descrever o então novo campo responsável por compreender como o uso de computadores poderia enriquecer a vida pessoal e profissional das pessoas.

Inicialmente, a ênfase recaía sobre as capacidades e limitações de usuários humanos, buscando entender os processos psicológicos envolvidos na interação com os sistemas computacionais. No entanto, à medida que a área evoluiu, tornou-se evidente que outros fatores também influenciavam essa interação e, portanto, deveriam ser considerados (Preece *et al.*, 1994). Questões como treinamento, práticas de trabalho, gestão, aspectos organizacionais e riscos à saúde passaram a ser vistos como elementos essenciais para o sucesso ou fracasso da utilização dos sistemas computacionais (Ibid.).

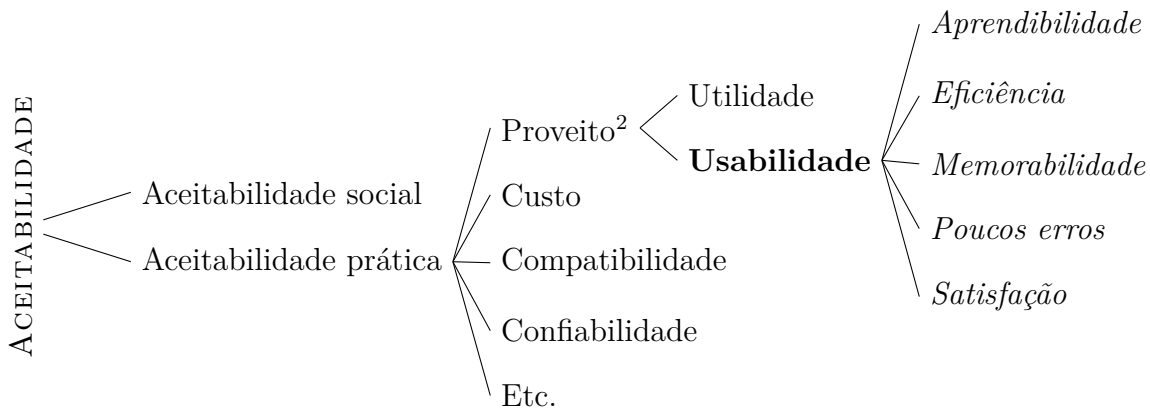
São listados por Barbosa *et al.* (2021) quatro critérios de qualidade de uso em IHC: usabilidade, experiência do usuário, acessibilidade e comunicabilidade. A *usabilidade* é o critério de qualidade de uso mais conhecido e, por conseguinte, o que é considerado com mais frequência (Ibid.). A norma ISO/IEC 25010:2011(E) (2011) define usabilidade como o grau com o qual um produto ou sistema pode ser usado por usuários específicos para alcançar determinados objetivos com eficicácia, eficiência e satisfação em um determinado contexto de uso. Assim como o documento da norma, Nielsen (1993c) também descreve a usabilidade como tendo multiplos componentes, associados a cinco atributos[1]:

- *Aprendibilidade*: O sistema deve ser fácil de aprender de forma a permitir que o usuário rapidamente possa realizar alguma tarefa;
- *Eficiência*: O sistema deve ser eficiente para ser usado, de forma a, uma vez que o usuário aprendeu seu funcionamento, é possível um alto nível de produtividade;
- *Memorabilidade*: O funcionamento do sistema deve ser fácil de lembrar, possibilitando o retorno do usuário casual após algum período sem o utilizar sem ter que aprender tudo novamente;
- *Erros*: O sistema deve ter uma baixa taxa de erros, permitindo usuários cometer poucos erros durante o uso do sistema, e se recuperarem de erros caso eles sejam cometido. Erros catastróficos não devem ocorrer;
- *Satisfação*: O sistema deve ser agradável de usar, para que o usuário esteja subjetivamente satisfeito ao usar ele — eles *gostam* do sistema.

[1] No original *learnability*, *efficiency*, *memorability*, *errors* e *satisfaction*, respectivamente (tradução nossa).

É salientado ainda que, apenas por meio da definição de usabilidade — descrita pelo autor como um conceito abstrato — em termos desses cinco componentes mais precisos e mensuráveis, é possível chegar a uma disciplina de engenharia, onde a usabilidade não é apenas discutida, mas sim sistematicamente abordada, melhorada, avaliada e mensurada. Além disso, Nielsen (1993c) destaca que a usabilidade é apenas uma das preocupações inseridas no problema maior da *aceitabilidade* de um sistema, que é, em suma, um grupo de critérios que avalia se um sistema é bom o suficiente para satisfazer todos os as necessidades e requisitos do usuário e potenciais partes interessadas, como clientes do usuário ou gerentes (Ibid.). A Figura 1 mostra um modelo da aceitabilidade de um sistema.

Figura 1: Modelo da aceitabilidade de um sistema.



Fonte: Adaptado de Nielsen (1993c). Tradução dos termos pelo autor.

2.2 Interfaces de linha de comando

Nielsen (1993c) ressalta que a usabilidade é aplicável a todos os aspectos de um sistema com o qual humanos podem interagir. Desses aspectos, possivelmente um dos mais perceptíveis é a interface do usuário, visto que esta é o único meio de contato entre o usuário e o sistema (Barbosa *et al.*, 2021). Como definido por Card *et al.* (1983), a *interface do usuário* é qualquer mecanismo usado pelo usuário como intermédio de seu diálogo com computadores — sejam estes componentes de *hardware*, como teclados e telas, ou de *software*, como os próprios programas de computador. Os diversos tipos de interface podem ser agrupados por função, dispositivo de entrada/saída, plataforma para a qual ele é projetado e estilo de interação (Preece *et al.*, 2023). Shneiderman *et al.* (2018) mencionam cinco estilos primários de interação: manipulação direta, interação por menus, interação por formulários, linguagem natural e linguagem de comando[3].

[2] Originalmente *usefulness*.
[3] Tradução dos termos por Barbosa *et al.* (2021).

Interfaces de linguagem de comando, popularmente também conhecidas como *interface de linha de comando* (CLI, acrônimo inglês para *command-line interface*)[4] são interfaces em que, para realizar ações, o usuário deve digitar comandos e aguardar a execução deles que, muitas vezes, é instantânea (Shneiderman *et al.*, 2018). Tal imediatez, aliada à natureza efêmera das CLIs, forma a essência da interação nesse tipo de interface (Ibid.). Ao contrário do que ocorre em, por exemplo, interfaces de manipulação direta, o estado do sistema não é exibido continuamente. Isto é, por padrão[5], após a execução de determinado comando, o retorno exibido é referente ao estado do sistema no momento da execução desse comando. No exemplo ilustrado pela Figura 2, o usuário precisou executar o comando `ls` para visualizar a estrutura atual do diretório após a criação ou remoção do arquivo `grass`.

Figura 2: Exemplo da não-continuidade da exibição do estado do sistema.

```
[11:26:32] ~/documentos/ufc/sem12/tcc (main v) ls
apresentacao  atividades  LICENSE   tcc
arquivos      IDEIAS.md   NOTAS.md  textos
[11:26:38] ~/documentos/ufc/sem12/tcc (main x) touch grass
[11:26:38] ~/documentos/ufc/sem12/tcc (main x) ls
apresentacao  atividades  IDEIAS.md  NOTAS.md  textos
arquivos      grass       LICENSE    tcc
[11:26:44] ~/documentos/ufc/sem12/tcc (main v) rm grass
[11:26:50] ~/documentos/ufc/sem12/tcc (main v) ls
apresentacao  atividades  LICENSE   tcc
arquivos      IDEIAS.md   NOTAS.md  textos
```

Fonte: Elaborado pelo autor (2025).

Tal como ocorre muitas vezes na comunicação em linguagem natural, a interação em CLIs costuma seguir um padrão *pergunta–resposta*, no qual mensagens submetidas pelo remetente — aqui o usuário —, a não ser quando em branco, são necessariamente respondidas pelo destinatário, isto é, o sistema, mesmo que por meio de mensagens de erro. Prasad et al. apontam que, particularmente nos seus primórdios, interfaces gráficas utilizaram-se intensivamente de metáforas — áreas de trabalho, arquivos, pastas e lixeiras — o que foi uma grande vantagem desse tipo de interface sobre as CLIs. No entanto, as CLIs acabaram por, acidentalmente, incorporar também uma metáfora: a interação nela pode ser vista como uma conversação. Os autores mencionam as seguintes maneiras de como essa conversação usuário-sistema pode acontecer:

[4] Shneiderman *et al.* (2018) usam ambos termos indiscriminadamente.

[5] Utilitários como o programa `watch` permitem a execução periódica de comandos, simulando, assim, a exibição contínua do estado do sistema.

- O usuário sucessivamente executa um comando, recebe um erro e altera o comando, até que o resultado desejado é alcançado;
- A execução de algum comando para configurar uma ferramenta seguida do aprendizado de que comandos executar para de fato começar a usar ela;
- A execução de vários comandos para configurar uma operação seguida de um comando final para executar ela — por exemplo, múltiplos `git add`s seguidos por um `git commit`;
- Exploração de um sistema — por exemplo, executando os comandos `cd` e `ls` múltiplas vezes para ter uma ideia da estrutura de diretórios, ou `git log` e `git show` para explorar o histórico de um arquivo;
- A realização de um *dry-run*[6] de uma operação complexa antes de realmente a executar.

Exemplos de CLIs são os interpretadores de comando presentes em diversos sistemas operacionais: por exemplo, os que estão presentes em sistemas tipo UNIX — Bourne shell (`sh`), Bash, Z shell (`zsh`), entre outros — e os disponíveis para Microsoft Windows — PowerShell, Prompt de Comando (`cmd.exe`), Subsistema do Windows para Linux (WSL[7]), entre outros. Além destes, algumas aplicações costumam ter sua interação mediada por CLIs, como o programa de análise numérica MATLAB e o programa de desenho assistido por computador AutoCAD. Mecanismos de busca, considerado que estes são muitas vezes usados para obtenção direta de respostas de forma textual, podem ser vistos também como interfaces de linha de comando (Norman, 2007).

2.3 Heurísticas de usabilidade de Jakob Nielsen

Publicadas pela primeira vez por Molich e Nielsen (1990) e atualizadas por Nielsen (1993c), essa lista é composta por um conjunto de diretrizes de usabilidade, que descrevem características desejáveis da interação e da interface, chamadas por Nielsen de *heurísticas* (Ibid.). Os princípios delas são bastante amplos e podem ser aplicados a praticamente qualquer tipo de interface, incluindo tanto interfaces textuais quanto interfaces gráficas (Nielsen, 1990e). Como mencionado por Molich e Nielsen (1990), a lista original foi composta de forma a corresponder a experiências pessoais dos autores, sendo que os princípios delas correspondem a princípios similares descritos por outros — a saber, o *Apple Human Interface Guidelines* (APPLE COMPUTER, INC., 1987). O conjunto inicial de heurísticas descrito por Nielsen (1993c) a ser utilizado em seu método de avaliação heurística é enumerado abaixo, com cada princípio seguido de sua respectiva descrição[8]:

[6] Simulação do que um comando faz sem de fato realizar alterações.

[7] Do acrônimo inglês para *Windows Subsystem for Linux*. O WSL é um componente do Microsoft Windows que permite utilizar um ambiente Linux diretamente no Windows, eliminando a sobrecarga de uma máquina virtual e oferecendo uma alternativa ao uso de *dual boot*.

[8] Tradução de Barbosa *et al.* (2021).

1. *Visibilidade do estado do sistema*: o sistema deve sempre manter os usuários informados sobre o que está a contecendo através de *feedback* (respostas às ações do usuário) adequado e no tempo certo;
2. *Correspondência entre o sistema e o mundo real*: o sistema deve utilizar palavras, expressões e conceitos que são familiares aos usuários, em vez de utilizar termos orientados ao sistema ou jargão dos desenvolvedores. O *designer* deve seguir as convenções do mundo real, fazendo com que a informação apareça em uma ordem naural e lógica, conforme esperado pelos usuários;
3. *Controle e liberdade do usuário*: os usuários frequentemente realizam ações equivocadas no sistema e precisam de uma "saída de emergência" claramente marcada para sair do estado indesejado sem ter de percorrer um diálogo extenso. A interface deve permitir que o usuário desfaça e refaça suas ações;
4. *Consistência e padronização*: os usuários não devem ter de se perguntar se palavras, situações ou ações diferentes significam a mesma coisa. O *designer* deve seguir convenções da plataforma ou do ambiente computacional;
5. *Reconhecimento em vez de memorização*: o *designer* deve tornar os objetos, as ações e opções visíveis. O usuário não deve ter que se lembrar para que serve um elemento de interface cujo símbolo não é reconhecido diretamente; nem deve ter de se lembrar de informação de uma parte da aplicação quando tiver passado para uma outra parte dela. As instruções de uso do sistema devem estar visíveis ou facilmente acessíveis sempre que necessário;
6. *Flexibilidade e eficiência de uso*: aceleradores[9] — imperceptíveis aos usuários novatos — podem tornar a interação do usuário mais rápida e eficiente, permitindo que o sistema consiga servir igualmente bem os usuários experientes e inexperientes. Exemplos de aceleradores são botões de comando em barras de ferramentas ou teclas de atalho para acionar itens de menu ou botões de comando. Além disso, o designer pode oferecer mecanismos para os usuários customizarem ações frequentes;
7. *Projeto estético e minimalista*: a interface não deve conter informação que seja irrelevante ou raramente necessária. Cada unidade extra de informação em uma interface reduz sua visibilidade relativa, pois compete com as demais unidades de informação pela atenção do usuário;
8. *Prevenção de erros*: melhor do que uma boa mensagem de erro é um projeto cuidadoso que evite que um problema ocorra, caso isso seja possível;
9. *Ajude os usuários a reconhecerem, diagnosticarem e se recuperarem de erros*: as mensagens de erros devem ser expressas em linguagem simples (sem códigos indecifráveis), indicar precisamente o problema e sugerir uma solução de forma construtiva;

[9] Originalmente *shortcuts*.

10. *Ajuda e documentação*: embora seja melhor que um sistema possa ser utilizado sem documentação, é necessário oferecer ajuda e documentação de alta qualidade. Tais informações devem ser facilmente encontradas, focadas na tarefa do usuário, enumerar passos concretos a serem realizados e não ser muito extensas.

Em seu contexto original, Molich e Nielsen (1990) empregaram esses princípios para classificar problemas de usabilidade de um sistema fictício criado como parte de uma pesquisa cujo objetivo foi investigar se profissionais de processamento de dados industriais seriam capazes de identificar problemas sérios de interface em diálogos. No exercício, publicado pela edição danesa da revista *Computerworld*, os setenta e sete participantes tiveram que avaliar um diálogo humano-computador para indicar problemas de usabilidade e possíveis soluções para estes. O sistema fictício adotado na pesquisa foi um diálogo em um terminal de texto, composto por uma tela de 24 linhas por 80 caracteres e um teclado; sem cores, mouse ou interface gráfica.

2.4 Command Line Interface Guidelines

O conjunto de diretrizes *Command Line Interface Guidelines* (CLIG)[10]é um guia que visa auxiliar no desenvolvimento de programas para CLI mais eficazes. Em suas recomendações, os autores buscaram adaptar princípios da filosofia UNIX, cujo cerne é a ideia de que programas pequenos e simples com interfaces limpas podem ser combinados na construção de sistemas maiores (Kernighan; Pike, 1984). Como mencionado no guia, programas UNIX tradicionamente eram escritos para serem usados por outros programas. Na atualidade, porém, muitos programas para CLIs são usados primariamente — ou mesmo exclusivamente — por humanos. Logo, é afirmado que se um programa será utilizado primariamente por usuários humanos, o comportamento destes deve ser considerado no desenvolvimento de comandos para esses utilitários (Prasad *et al.*, 2025).

As noventa e nove diretrizes do CLIG são divididas em dezesseis seções, cujos conteúdos são descritos abaixo[11]. É explicitado no guia que as seções subsequentes à primeira são compostas por diretrizes complementares: "(...) as demais (diretrizes) são apenas desejáveis. Se você tiver tempo e energia para implementar essas coisas, seu programa será um tanto melhor que o programa médio" (Ibid.).

1. *O básico*: diretrizes fundamentais que devem ser seguidas e que, caso sejam entendidas erroneamente, o programa desenvolvido poderá ser de dificil utilização;
2. *Ajuda*: fornecimento de ajuda pelo programa — em quesito de documentação interna e do próprio uso — e como essa ajuda pode ser estruturada;

[10]Disponível em https://clig.dev. No Quadro 3, disponível no Apêndice B, são mostradas as descrições breves de cada diretriz do guia.
[11]Tradução dos títulos das seções feitas pelo autor.

3. *Documentação*: documentação do programa, isto é, ajuda mais extensa e detalhada, e canais pelos quais ela é fornecida;
4. *Saída*: saída retornada pelo programa, e como ela deve ser fornecida e estruturada;
5. *Erros*: mensagens de erro e como elas devem ser fornecidas e estruturadas;
6. *Argumentos flags flags*: forma como argumentos e *flags* devem ser empregados pelo programa;
7. *Interatividade*: diretrizes relacionadas ao uso de *prompts* e elementos interativos pelo programa e como estes devem se comportar;
8. *Subcomandos*: diretrizes relacionadas à maneira como subcomandos devem ser empregados e como eles devem ser estruturados;
9. *Robustez*: diretrizes relacionadas à robustez do programa — anteriormente no guia descrita como uma qualidade objetiva (entrada inesperada deve ser tratada graciosamente, operações devem ser consistentes) ou subjetiva (não é desejável que um programa pareça frágil);
10. *Planejamento do futuro*: diretrizes relacionadas a adaptações a mudanças futuras que podem ocorrer tanto no desenvolvimento do programa, quanto ao ambiente em que ele é utilizado;
11. *Sinais e caracteres de controle*: diretrizes relacionadas ao uso de sinais e caracteres de controle pelo programa;
12. *Configuração*: diretrizes relacionadas à configuração do programa por parte do usuário e como ela pode ser propiciada pelo desenvolvedor;
13. *Variaveis de ambiente*: diretrizes relacionadas a variáveis de ambiente e como elas podem ser empregadas pelo programa;
14. *Nomeação*: diretrizes relacionadas à forma como comandos são nomeados;
15. *Distribuição*: diretrizes relacionadas à maneira como programas são distribuídos;
16. *Estatísticas*: diretrizes relacionadas à coleta de estatísticas sobre o uso do programa.

Mesmo não sendo explicitado no texto do CLIG, é aparente a influência de certos pontos das heurísticas primeiramente propostas por Molich e Nielsen (1990), visto que o guia recorre repetidamente a princípios como retorno imediato, prevenção de erros e documentação concisa. Apesar de maior parte dos exemplos serem direcionados a terminais de sistemas tipo UNIX, acredita-se que as recomendações do guia sejam válidas para outras CLIs.

3 TRABALHOS RELACIONADOS

3.1 Schmitzhaus (2024)

Partindo do fato de que em CLIs o aspecto da usabilidade é frequentemente desconsiderado, o que resulta em interfaces que tendem a ser de difícil utilização, em seu trabalho de conclusão de curso *Usabilidade em Interfaces de Linha de Comando* Schmitzhaus se propõe a analisar essa problemática a partir de uma abordagem teórico-prática fundamentada em conceitos da IHC. Tendo isso em consideração, o autor avalia a usabilidade das ferramentas Git, Docker e AWS CLI à luz das diretrizes do guia *Command Line Interface Guidelines* (CLIG), cujos detalhes já foram descritos neste trabalho. A análise enfatiza critérios como clareza das mensagens de ajuda, coerência na organização de subcomandos, padronização de argumentos e retorno textual das operações.

Os resultados apontam que, embora essas ferramentas apresentem pontos positivos — como comandos de ajuda bem estruturados e documenação acessível — há também falhas recorrentes, como terminologia inconsistente e ausência de mensagens de erro explicativas. Sendo assim, concluiu-se que, embora muitas das diretrizes sejam seguidas, ainda há espaço para melhoras, especialmente em aspectos relacionados à curva de aprendizado, clareza de mensagens e consistência na estrutura dos comandos.

O trabalho de Schmitzhaus aproxima-se da proposta deste estudo ao aplicar diretrizes específica (o guia CLIG) na análise da usabilidade de ferramentas populares de linha de comando. No entanto, enquanto sua abordagem permanece concentrada no emprego imediato dessas diretrizes como instrumento avaliativo, o presente projeto busca questionar a suficiência e abrangência de tais orientações à luz de princípios mais amplos de usabilidade. Com isso, pretende-se refletir criticamente sobre possíveis lacunas que possam comprometer a experiência do usuário em CLIs.

3.2 Schröder e Cito (2021)

No artigo *An empirical investigation of command-line customization* Schröder e Cito realizam uma análise empírica em larga escala sobre práticas de personalização em CLIs, com foco no uso de *aliases* — isto é, pseudônimos ou apelidos para comandos ou sequências de comandos — em *shells* UNIX. A partir da mineração de mais de 2,2 milhões de definições de *aliases* extraidas do GitHub, os autores identificaram padrões recorrentes de personalização adotados por usuários experientes.

Usando métodos de codificação indutiva[1], os *aliases* foram classificados em **1.** *atalhos* (apelidos para comandos frequentes ou complexos), **2.** *modificações* (ajustes no comportamento de comandos existentes) e **3.** *scripts* (encadeamento de comandos para automação de tarefas repetitivas ou transformação de dados).

A pesquisa destaca como esse tipo de prática pode expor limitações de usabilidade de CLIs convencionais, especialmente no que tange à necessidade de memorização e à inadequação configurações padrão às necessidades de seus usuários. Os autores discutem uma série de implicações ligadas à usabilidade das CLIs analisadas, sendo estas:

- *Correção de erros*: com base na análise dos dados coletados, os autores afirmam que é possível a correção de erros recorrentes;
- *Descoberta de fluxos de trabalho*: além da correção de erros, os autores antecipam como os dados coletados podem ser úteis para fornecer dicas que podem apresentar ao usuário padrões comuns e fluxos de trabalho adaptáveis;
- *Descoberta de falhas conceituais*: é afirmado pelos autores que personalização pode ser um indicador de problemas conceituais subjacentes, manifestados por frustrações de usabilidade que necessitam adaptação pelo próprio usuário;
- *Padrões contextuais*: levando em consideração o fato de que 14.48% das personalizações analisadas sobrescreviam configurações padrão, os autores afirmam que nem em todos os casos o contexto assumido corresponde à realidade do usuário. Logo, os autores sugerem que talvez seja possível o fornecimento de padrões diferentes para diferentes tipos de usuário;
- *Interatividade vs. composição de scripts*: ao serem analisadas definições de *aliases*, percebeu-se que muitas delas tentavam melhorar a legibilidade da saída de comandos que, tradicionamente, não foram pensadas para serem lidas por humanos. Os autores afirmam que alterações desse tipo definidas por padrão podem ser problemáticas para *scripts*, mas notam que é possível a detecção do contexto para adaptação da saída de determinado comando.

Além da análise feita, os autores também forneceram o conjunto de dados coletados e formas de replicar a coleta automatizada utilizadas por eles para recolhimento de informações sobre arquivos de configuração.

A pesquisa de Schröder e Cito oferece evidências empíricas valiosas sobre como usuários moldam suas próprias experiências em CLIs por meio da personalização, revelando limitações de usabilidade percebidas na configuração padrão dessas interfaces. As conclusões apresentadas pelos autores podem ajudar a sustentar a hipótese de que deficiências de usabilidade podem levar usuários a buscar soluções alternativas. Assim, o trabalho contribui indiretamente para reforçar a relevância de se reavaliar a cobertura oferecida por diretrizes específicas para CLIs.

[1] *Codificação indutiva* é uma técnica para análise qualitativa de dados que é utilizada quando se conduz uma pesquisa exploratória sem expectativas preliminares sobre os temas dos dados analisados.

3.3 Cabot (2017)

Em *Re-imagining the Command Line Experience for Problem Solving*, Cabot discorre sobre como se pode ser tirado mais proveito de CLIs como ferramentas para compor uma vasta gama de funcionalidades visando a resolução de problemas. A autora parte da constatação de que, embora esse tipo de interface ofereça poder e controle expressivo, elas impõem uma carga cognitiva elevada ao exigir que os usuários especifique ações por meio de comandos sintaticamente rígidos e pouco previsíveis. A questão analisada no texto é como o *problema de exploração* — caracterizado pela autora como a necessidade de usuários precisarem realizar ações exploratórias com o objetivo de executar comandos válidos — pode ser abordada e, por meio disso, determinar como CLIs podem suportar a resolução de problemas não convencionais.

Ao serem analisados trabalhos anteriores relacionados a tipos de interface em relação a estratégias de resolução de problemas, conclui-se que ações exploratórias de um sistema são fundamentais para toda interação humano-máquina. Assim, usuários de CLIs seriam beneficiados também por melhor suporte à descoberta de comandos e reflexos dos efeitos de ações de comandos anteriores. Para melhor fundamentação dessas descobertas, são realizadas entrevistas com um grupo de usuários experientes de CLI e, com base nisso, são identificadas qualidades que usuários desse tipo de interface valorizam, quais efeitos de suas experiências e a quais problemas CLIs são mais sucetíveis.

Considerando essas informações, a autora especifica uma CLI aprimorada, com foco em resolução de problemas. São propostas algumas versões que buscaram atender os problemas encontrados sugerindo funcionalidades como integração com inteligência artificial, mecanismos de sugestão e visualização de informação. Posteriormente as versões da CLI aprimorada foram avaliadas por usuários, e o melhor protótipo implementado. Em seguida, a CLI aprimorada que foi implementada é analisada em mais detalhe, com objetivo de avaliar sua efetividade.

Ao refletir sobre a resolução de problemas em CLIs, Cabot destaca como a rigidez sintática e a imprevisibilidade de comandos dificultam ações exploratórias — fator crítico quando o usuário não possui um caminho claro a seguir. Como forma de mitigar essa demanda cognitiva, a autora propõe recursos que favorecem a descoberta de comandos e a visualização de efeitos, sinalizando o potencial de adaptações no desenho dessas interfaces para melhor apoiar o raciocinio do usuário. Esse olhar para os obstáculos enfrentados durante a exploração reforça a importância de critérios de usabilidaide que vão além da eficiência de execução, incluindo também a compreensão, previsibilidade e adaptabilidade das interações.

3.4 Análise comparativa

O Quadro 1 sintetiza os aspectos centrais dos três trabalhos discutidos na seção anterior e este trabalho, evidenciando seus focos principais, métodos, amostras e limitações.

Quadro 1: Comparação entre os trabalhos relacionados e este.

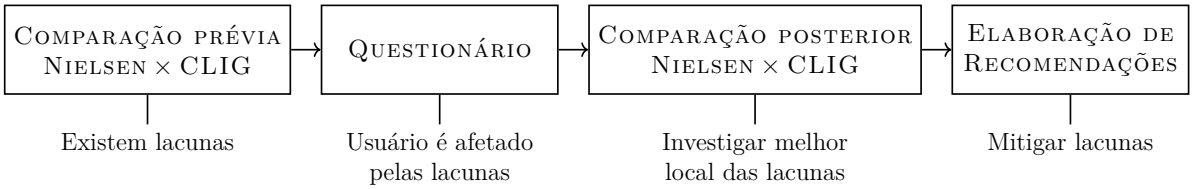
Critério	**Schmitzhaus (2024)**	**Schröder e Cito (2021)**	**Cabot (2017)**	**Este trabalho (2025)**
Foco principal	Avaliar a usabilidade de CLIs populares com base no CLIG	Investigar padrões de personalização em CLIs	Redesenhar a CLI para melhor suporte a resolução de problemas	Identificar lacunas no CLIG e propor análise crítica
Método	Análise teórico-descritiva	Mineração de dados e codificação indutiva	Entrevistas, prototipagem, testes	Questionário, revisão, comparação
Amostra	(Não se aplica)	Usuários do GitHub	Grupo escolhido de usuários avançados de CLI	Estudantes universitários da área de tecnologia
Limitações	Restrito a três ferramentas e às diretrizes do CLIG	Não propõe melhorias diretas para as CLIs estudadas	Amostra limitada de usuários e foco em protótipo específico	Falta de métricas quantitativas

Fonte: Elaborado pelo autor (2025)

4 METODOLOGIA

Este estudo foi conduzido através de uma abordagem qualitativa, de caráter exploratório e interpretativo. A metodologia foi estruturada em etapas complementares, que combinam análise documental, coleta exploratória de dados com usuários e síntese interpretativa dos resultados. Os processos metodológicos adotados são ilustrados na Figura 3.

Figura 3: Fluxograma da metodologia adotada por este trabalho.



Fonte: Elaborado pelo autor (2026)

Realizou-se uma análise exploratória do guia CLIG, na qual suas noventa e nove diretrizes foram mapeadas às dez heurísticas de usabilidade de Jakob Nielsen. O mapeamento foi conduzido de forma interpretativa, estabelecendo correspondências conceituais entre diretrizes e heurísticas. Aplicou-se, então, um questionário eletrônico com usuários de interfaces de linha de comando. O questionário teve caráter exploratório e abordou aspectos relacionados a dificuldades de uso, aprendizado, prevenção de erros e eficiência. As respostas obtidas foram analisadas qualitativamente.

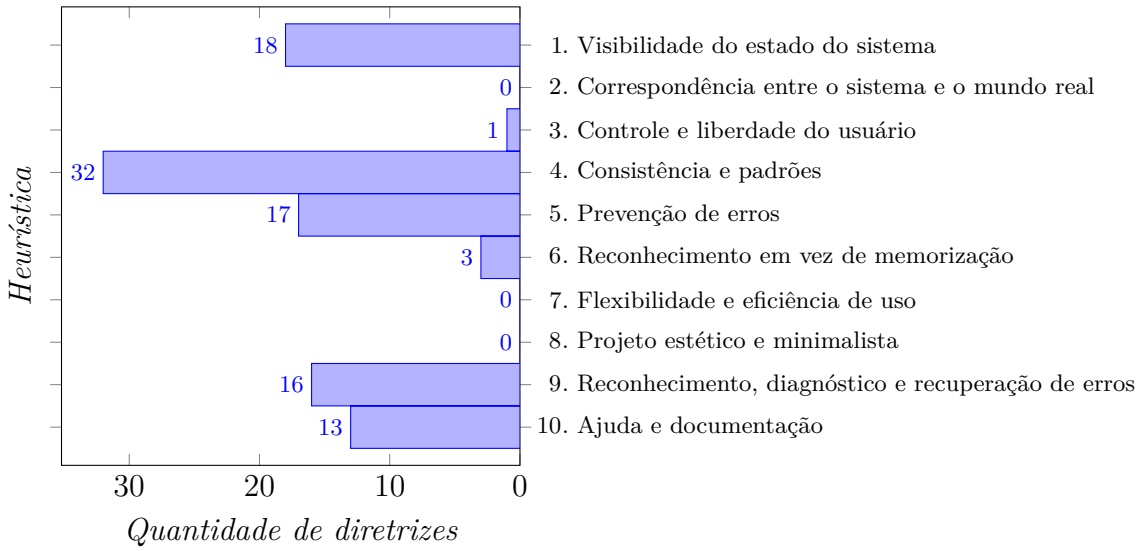
Adotou-se a análise exploratória em conjunto com o questionário sob a razão de que — assumindo-se a universalidade das heurísticas de Nielsen — é plausível conjecturar que se **1.** há uma aparente lacuna de correspondência entre as diretrizes do guia CLIG e essas heurísticas, e que **2.** usuários de CLIs mostram ter dificuldades relacionadas a conceitos abordados por heurísticas pouco correspondidas pelo guia, então, tendo em mente a já mencionada relevância deste, o planejamento e avaliação da usabilidade de utilitários de linha de comando podem ser prejudicados.

A partir dos resultados das etapas anteriores, fio realizada uma comparação sistemática entre o guia CLIG e as heurísticas de Nielsen. As correspondências entre diretrizes e heurísticas foram estabelecidas de forma interpretativa, com o registro das associações identificadas. Com base nessa comparação, foram elaboradas recomendações complemetares ao guia CLIG. As recomendações foram estruturadas de forma semelhante às diretrizes existentes, incluindo descrições sucintas e detalhamentos quando pertinente.

5 ANÁLISE EXPLORATÓRIA E QUESTIONÁRIO

Realizou-se uma análise do guia *Command Line Interface Guidelines*, na qual as noventa e nove diretrizes deste foram mapeadas às dez heurísticas de Jakob Nielsen. As correspondências entre as duas listas de recomendações foi feita de forma interpretativa, buscando relacionar o conceito abordado por determinanda diretriz a alguma heurística. O propósito dessa análise inicial foi não mais que verificar se há aspectos das heurísticas pouco ou não abordados pelo guia CLIG. Percebendo-se a possível presença dessas lacunas, julgou-se necessária a confirmação do impacto real delas em usuários e uma análise mais detalhada, para que fossem documentadas as razões de cada correspondência; esses processos que são descritos nos parágrafos abaixo e no capítulo seguinte. Na Figura 4 é exibida a distribuição da quantidade de diretrizes por heurística.

Figura 4: Distribuição aproximada das diretrizes CLIG por heurística



Fonte: Elaborado pelo autor (2025)

Aplicou-se um questionário eletrônico que auxiliou, por meio da coleta de informações de uso diretamente de usuários de CLIs, numa síntese das dificuldades enfrentadas por estes, com foco nas lacunas de correspondência CLIG-Nielsen percebidas após a análise inicial do guia. Apesar de no questionário não terem sido incluídos itens que tratam sobre informações pessoais mais específicas, como a ocupação do participante, acredita-se que a maioria da amostra consultada foi composta por estudantes universitários de áreas relacionadas à tecnologia. Supõe-se isso ao ser considerado que o questionário foi divulgado especialmente em grupos de WhatsApp compostos por membros com esse perfil.

O questionário foi estruturado em três seções, cujos detalhes são ilustrados no Quadro 2. Os itens referentes à primeira e segunda seções foram perguntas de múltipla escolha. Já os da terceira seção contaram com apenas duas alternativas (SIM/NÃO) e, excetuando-se o último, todos foram ordenados tematicamente em quatro subseções. Foram obtidas treze respostas ao questionário, que está disponível no Apêndice A. A Figura 5 ilustra as respostas da terceira seção do questionário que, como descrito a seguir, constituiu a parte principal dele.

Quadro 2: Organização do questionário.

SEÇÃO	PROPÓSITO
1. Informações básicas (3 itens)	Coletar informações acerca da área de atuação e nível técnico geral do respondente, e identificar se é usuário de terminal/prompt de comando.
2. Informações técnicas (3 itens)	Identificar o sistema operacional primário do respondente, há quanto tempo usa o terminal/prompt de comando e a frequência com que usa.
3. Experiência com CLIs (14 itens)	Investigar sobre aspectos da usabilidade relacionados às heurísticas de Nielsen que, após a comparação, foram percebidas como as que possuem menos correspondências no guia CLIG, e aspectos das CLIs geralmente vistos como empecilhos à adoção de boas práticas de usabilidade nesse tipo de interface.
	SUBSEÇÃO **3.1.** Compreensão e interpretação de mensagens (3 itens); **3.2.** Aprendizado e familiaridade com comandos (7 itens); **3.3.** Erros e segurança de uso (2 itens); **3.4.** Uso, domínio de atalhos e personalização (1 item).

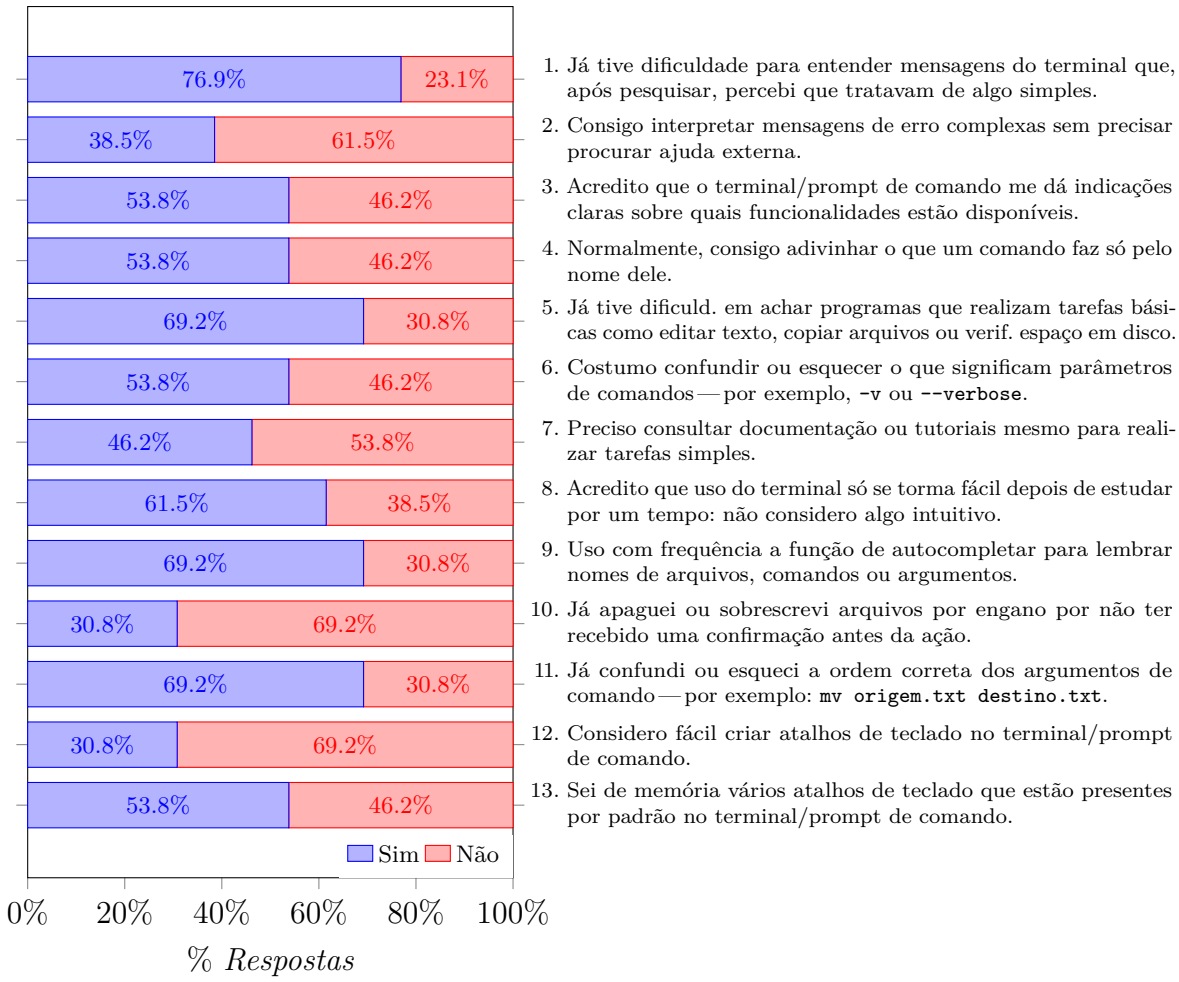
Fonte: Elaborado pelo autor (2025).

As respostas coletadas foram dadas por usuários, em sua maioria, autodeclarados como usuários com nível técnico geral intermediário (61.5%). As duas opções de sistemas operacionais primários que obtiveram respostas foram Linux (61.5%) e Windows (38.5%), o que indica que boa parte dos respondentes ou utilizam interpretadores de comando tipo UNIX ou algum dos comumente presentes em sistemas Windows. Apesar de questionários serem mais indicados no tratamento quantitativo de informações, por conta da relativa pouca quantidade de respostas, treze, foi realizada uma análise qualitativa das informações coletadas.

A primeira subseção, composta por três itens que incorporaram conceitos abrangidos pela segunda, sexta e oitava heurísticas de Nielsen[1], buscou compreender dificuldades enfrentadas relacionadas a mensagens exibidas pelo terminal/prompt de comando. Com base nas respostas ao primeiro e segundo itens desta subseção, pode-se inferir que, de forma geral, as mensagens exibidas por boa parte dos programas de

[1] *Correspondência entre o sistema e o mundo real*, *Reconhecimento em vez de memorização*, e *Projeto estético e minimalista*, respectivamente.

Figura 5: Respostas aos treze primeiros itens da terceira seção



Fonte: Elaborado pelo autor (2025)

terminal não são claras o suficiente, fazendo-se necessária a consulta de ajuda externa. É interessante notar que boa quantidade dos respondentes (76.9%) declarou dificuldade em decifrar mensagens aparentemente complicadas que, após consulta externa, se tratavam de algo simples. Isso pode indicar uma provável dificuldade na transferência de conhecimentos prévios ao uso de utilitários baseados em CLI: há possibilidade que o usuário consiga entender certo conceito, mas o programa não dá pistas precisas de como isso pode ocorrer. Em relação ao terceiro item[2], aproximadamente a metade dos respondentes (53.8% respostas positivas contra 46.2% respostas negativas) indicaram ser claras as indicações das funcionalidades do terminal. Foi notado que dos quatro usuários autodeclarados avançados, apenas um deles (25%) julgou serem claras essas indicações, em contraste aos cinco de oito (62.5%) usuários intermediários que afirmaram o mesmo. É possível que usuários menos experientes não sejam capazes de indicar com precisão seu nível técnico.

[2] Consigo interpretar mensagens de erro complexas sem precisar de ajuda externa.

A segunda subseção, cujos seis itens trataram temáticas relacionadas à segunda e sexta heurísticas de Nielsen, teve como objetivo investigar as dificuldades enfrentadas pelos usuários no processo de aprendizado de comandos e parâmetros, especialmente no que diz respeito ao reconhecimento, memorização e previsibilidade das funcionalidades. De acordo com as respostas desta seção, apesar de que cerca da metade (53.8%) dos respondentes acreditarem ser capazes de descobrir o que comandos fazem com base apenas em seu nome, uma quantidade expressiva deles (60.2%) declarou já ter tido alguma dificuldade ao tentar localizar programas básicos no terminal. Essa informação pode apontar uma disassociação entre a previsibilidade dos nomes e a capacidade de encontrar ferramentas úteis.

A terceira subseção — na qual estiveram contidos dois itens associados à segunda e, principalmente, sexta heurística de Nielsen — dispôs-se a explorar a frequência e as causas de erros críticos (como perda de dados) cometidos por usuários durante o uso de comandos, bem como a presença (ou ausência) de mecanismos de segurança e prevenção de erros na interface. Apesar da aparente cautela dos respondentes ao realizarem ações críticas, indicada pelos 69.2% de respostas negativas ao primeiro item desta seção[3], pôde-se notar uma possível falta de clareza por parte de programas de terminal ao especificar a ordem de argumentos, denotada pelos 69.2% de respostas afirmativas ao segundo item[4] da terceira seção.

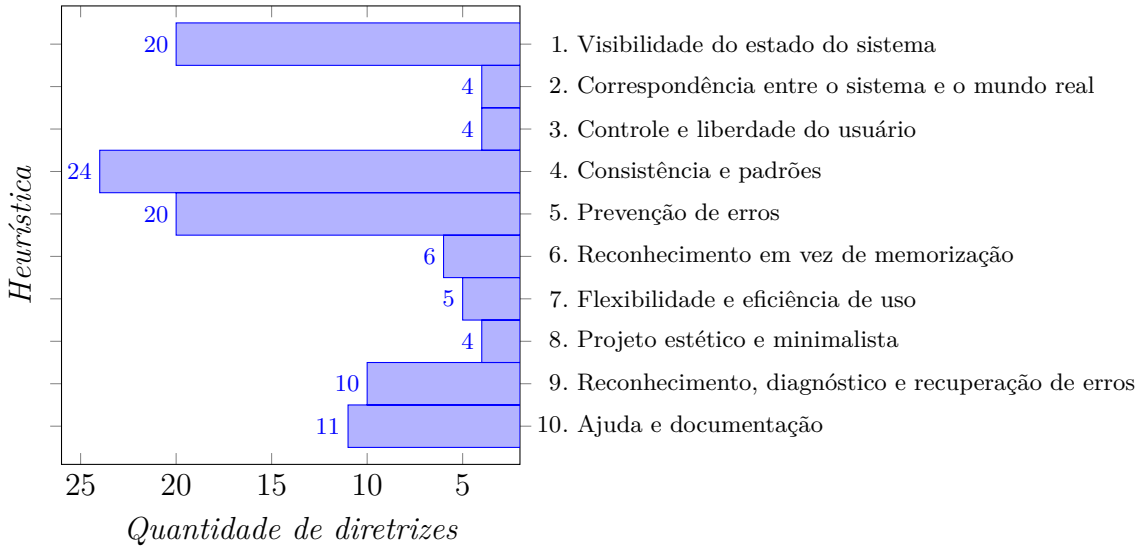
A quarta seção, que incluiu dois itens relacionados à sétima heurística de Nielsen[5], tentou examinar o grau de domínio dos usuários sobre atalhos de teclado e customizações que podem tornar o uso do terminal mais eficiente. Apesar de mais da metade (53.8%) dos respondentes ter afirmado saber de memória atalhos de teclado que estão presente por padrão no terminal/prompt de comando, apenas 30.8% da amostra considera fácil a criação de novos atalhos. Isso pode indicar que, mesmo que CLIs — em especial as utilizadas pela amostra analisada — sejam bastante flexíveis e eficientes, ainda há espaço para melhoras em relação a indicação de como personalizações podem ser realizados pelos próprios usuários.

[3] Já apaguei ou sobrescrevi arquivos por engano por não ter recebido uma confirmação antes da ação.
[4] Já confundi ou esqueci a ordem correta dos argumentos de comando — por exemplo: `mv origem.txt destino.txt`.
[5] *Flexibilidade e eficiência de uso.*

6 COMPARAÇÃO ENTRE NIELSEN E CLIG

Neste capítulo é descrita a comparação posterior realizada entre as dez heurísticas de Jakob Nielsen e o guia *Command Line Interface Guidelines*. A comparação foi feita de maneira similar à análise preliminar realizada anteriormente. Ou seja, realizou-se uma distribuição das noventa e nove diretrizes do guia CLIG entre as dez heurísticas de Nielsen de forma interpretativa, buscando relacionar o conceito abordado por determinada diretriz a alguma heurística. Diferencia-se a comparação prévia da que é descrita a seguir devido ao fato de que a escolha de cada correspondência desta Nielsen–CLIG foi documentada numa planilha, cujo conteúdo formatado em texto corrido está disponível no Apêndice B. Na Figura 6 é ilustrado o gráfico com as quantidades de correspondência por heurística.

Figura 6: Distribuição das diretrizes CLIG/heurística após a segunda comparação.



Fonte: Elaborado pelo autor (2025)

A correspondência entre muitos pares de itens não ocorreu de forma direta. Casos disso são onde ações propostas por determinada diretriz foram vistas como facilitadores *indiretos* da realização de sugestões dadas por certa heurística. Por exemplo, a diretriz 2.9.[1], *use formatação em textos de ajuda*, que foi alocada à décima heurística, *ajuda e documentação*, trata da formatação adequada do texto de ajuda disponibilizado por programas, que é exemplificada no guia com a utilização de negrito quando necessário. No entanto, a décima heurística trata sobre a própria disponibilização de ajuda e documentação por parte do programa. Julgou-se, então, a diretriz 2.9. como um correspondente indireto à décima heurística sob a razão de que a devida formatação do texto de ajuda auxilia na legibilidade deste.

[1] Segunda diretriz da nona seção.

Foram também identificadas relações diretriz–heurística não-binárias, visto que certas diretrizes tratam diretamente de conceitos relativos a múltiplas heurísticas. Além disso, foram encontradas diretrizes cujos conceitos abordados possivelmente estão fora do escopo das heurísticas de Nielsen. Dentre estas estão algumas diretrizes da décima quarta seção do guia CLIG, que tratam da ergonomia por trás da nomeação de comandos.

6.1 Visibilidade do estado do sistema

Ao todo, relacionadas a esta heurística, identificaram-se vinte e uma diretrizes. Destas, quatro pertencem à primeira seção do guia CLIG, *O básico*, do total de cinco itens que a compõem. Dado de que, no contexto das CLIs, o estado do sistema não costuma ser exibido de forma contínua[2] — isto é, após a execução de determinado comando, o retorno exibido refere-se ao estado do sistema no momento em que o comando foi executado — é pertinente notar como os autores do guia julgaram como básico o ponto abordado pela primeira heurística de Nielsen. Essas quatro diretrizes trataram sobre a indicação do estado do sistema ao usuário através do retorno explícito do êxito da execução do programa (saída zero) e envio do retorno ao fluxo[3] adequado.

Relativos à segunda seção do guia, *Ajuda*, apenas uma diretriz correspondente à primeira heurística foi identificada. Nela, é reforçada a importância de indicar ao usuário como proceder quando o programa necessita receber entrada através de encadeamento ou argumentos. O guia exemplifica aqui um ponto onde deve-se dar preferência à praticidade imediata sobre a bagagem histórica: tradicionalmente, o usuário não recebe retorno algum em casos como este, visto que, tecnicamente, não ocorreram erros. Comportamentos como esse tendem a causar confusão em usuários ainda não acostumados com as convenções geralmente adotadas em CLIs, em especial as de sistemas tipo UNIX.

Foram identificadas onze diretrizes relacionadas à primeira heurística na quarta seção do guia, *Saída*, do total de dezessete. Acredita-se que tal concentração ocorre graças ao mesmo motivo da que ocorre na primeira seção. Isto é, visto que o retorno textual é a principal maneira oferecida por CLIs de externalizar o estado atual do sistema, é plausível a ênfase em fornecer formas de fazer de isso adequadamente. Desta maneira, pode-se considerar a quarta seção do guia como uma extensão da primeira. Pontos abordados nas correspondências aqui verificadas incluem a exibição de indicadores de sucesso, o esclarecimento sobre quais partes do sistema um programa consegue acessar, melhora na visualização do estado do sistema por meio de indicadores visuais (cores, símbolos e arte ASCII) e a facilidade na visualização e navegação de *logs*[4] com muitas linhas. Um

[2] No entanto, utilitários como o programa `watch` permitem a execução periódica de comandos, *simulando*, assim, a exibição contínua do estado do sistema.
[3] Em inglês *stream*.
[4] Registro sequencial de eventos ocorridos em um sistema.

aspecto aqui também tratado é a legibilidade *pela máquina* da saída de programas, cujas diretrizes relacionadas foram anexadas à primeira heurística em virtude do frequente uso de encadeamento, que pode ser adotado na composição de utilitários que assistem a verificação do estado do sistema.

Na nona seção do guia, *Robustez*, foram encontradas quatro diretrizes relacionadas à visualização do estado do sistema, do total de oito. Tópicos aqui inseridos são relativos à responsividade do sistema por meio da exibição rápida do progresso realizado e à praticidade na visualização do estado do sistema, aqui exemplificada pela visualização paralela de informações. Já relativos à décima segunda seção, *Configuração*, um único correspondente foi localizado, que trata de assunto similar a uma das diretrizes da quarta seção: o acesso de arquivos do sistema por programas, mas aqui sendo enfatizado o acesso a arquivos de configuração.

6.2 Correspondência entre o sistema e o mundo real

Identificaram-se, no total, cinco diretrizes relacionadas a esta heurística, todas na quarta seção do guia CLIG. As diretrizes identificadas estão relacionadas à melhora na clareza do retorno dado por programas através de artifícios como arte ASCII, cores, símbolos e emojis. Inseriram-se aqui diretrizes relacionadas a esses tópicos sob a justificativa de que recursos como os mencionados são alguns dos poucos que, no contexto das CLIs, podem ser empregados no estabelecimento de metáforas visuais reconhecíveis ao usuário; neste ponto, especificamente, estão inclusos arte ASCII, símbolos e emojis. Além de ser sugerido o uso de convenções, como o uso de vermelho para indicar erros, o uso cuidadoso de cores é ressaltado: se *tudo* está em "uma cor diferente", tanto a cor utilizada passa a não ter significado claro, quanto o texto torna-se menos legível.

6.3 Controle e liberdade do usuário

Em relação à terceira heurística, perceberam-se cinco correspondências, dispersamente divididas entre quatro seções do guia CLIG: apenas uma correspondência na sexta (*Argumentos e flags*), sétima (*Interatividade*) e décima quinta (*Distribuição*) seções, e duas na décima primeira seção (*Sinais e caracteres de controle*). Pontos claramente relacionados ao tempo de encerramento de processos pelo usuário — aqui representados pelo pressionamento das teclas `CTRL-C` — são abordados. Pontos discutidos nos itens aqui inclusos são bastante similares aos tratados por Nielsen (1993c): o usuário deve sempre conseguir cancelar ações em andamento. Além da finalização de programas, também é sugerido na décima quinta seção a facilidade de desistalar estes.

6.4 Consistência e padrões

Vinte e quatro correspondências à quarta heurística foram verificadas, que estão distribuídas entre onze seções do guia. A grande concentração de diretrizes nesta heurísca pode apontar para uma tendência ao mantimento de padrões já existentes no contexto das CLIs. Na primeira seção, os itens mapeados para esta heurística tratam do uso de retornos padrão pelo programa e o direcionamento destes retornos para o fluxo adequado. As correspondências da segunda e terceira seções trataram da adoção de convenções específicas. A quarta seção, *Saída*, onde foram identificadas três diretrizes relacionadas, trata da conformidade a convenções relacionadas ao retorno de programas legível por máquinas e do uso adequado do fluxo de erros `stderr`.

Na sexta e setima seção, com duas correspondências cada, foram encontrados itens relacionados à consistência externa[5] que estão em conformidade com padrões adotados em sistemas tipo UNIX. Já na oitava seção, notaram-se correspondências relacionadas à consistência interna[6], neste caso relacionada à consistência entre subcomandos de determinado utilitário. Também na décima seção há uma única correspondência relacionada à consistência interna, aqui ligada à previsão de que ações tomar no caso de futuras atualizações de um programa. As correspondências verificadas nas seções décima segunda e décima terceira são relacionadas à adoção de conveções específicas de consistência externa que estão em conformidade com padrões de sistemas tipo UNIX. Na décima quarta seção, perceberam-se duas correspondências relacionadas a nomeação de programas.

6.5 Prevenção de erros

Foram mapeadas vinte diretrizes do guia CLIG à quinta heurística, sendo os itens encontrados dispostos entre oito seções. Relacionados à quarta seção, a única correspondência identificada trata sobre a prevenção de erros por meio da explicitação de ações que fogem dos limites internos de programas. Na sexta seção são abordados em oito itens pontos como a clareza da entrada dada a programas, o fornecimento de boas configurações padrão que são adequadas para a maioria dos usuários, a confirmação antes da realização de ações críticas, segurança da entrada de informações sensíveis e a redução de ambiguidades. Na sétima e oitava seções também são abordados assuntos similares aos dois últimos mencionados.

Na décima seção, *Preparação para o futuro*, é sugerido o mantimento de funcionalidades e comportamentos antigos de programas quando possível e a prevenção de problemas provindos de atualizações futuras. A décima terceira seção trata sobre prevenção de erros relacionados a variáveis de ambiente.

[5] *Consistência externa* refere-se à importância de alinhar o projeto de uma nova aplicação a convenções existentes e padrões familiares à audiência foco.

[6] *Consistência interna* trata da consistência do projeto e comportamento de uma aplicação através de telas ou recursos.

6.6 Reconhecimento em vez de memorização

Seis correspondências relacionadas à sexta heurística foram identificadas, que estão divididas entre a primeira, segunda, quarta, sexta e décima quarta seções. Verificou-se aqui apenas um mapeamento para cada seção. Na primeira seção, a correspondência encontrada sugere a adoção de bibliotecas de análise (*parsing*) de argumentos que permitem, dentre outras funcionalidades, o fornecimento de sugestões de correção ortográficas, algo que potencialmente pode facilitar o aprendizado de comandos, mesmo através de erros. Esse aspecto está relacionado ao ponto abordado nas diretrizes mapeadas à sexta heurística localizadas nas seções segunda e quarta: a sugestão de comandos, que, como ressaltado pelo guia, promove o aprendizado do uso do programa e a descoberta de novas funcionalidades. O item da quarta seção recomenda a sugestão de próximos passos após a primeira execução de um comando; já o da segunda seção trata, especificamente, da sugestão após algum erro cometido pelo usuário.

Relacionado à sexta seção, identificou-se um item que sugere o fornecimento do nome de *flags* também em suas versões por extenso — algo útil no reconhecimento da funcionalidade de *flags* após um período sem examinar ajuda externa, o que ocorre, especialmente, ao ser consultado o código-fonte de algum *script*. A correspondência identificada na décima quarta seção sugere a adoção de nomes de programa memoráveis. Apesar de não ser elaborado sobre o significado disso, acredita-se que aqui os autores se refiram à adoção de nomes relacionados à funcionalidade do programa.

6.7 Flexibilidade e eficiencia de uso

Cinco correspondências relacionadas à sétima heurísticas foram identificadas, que estão divididas entre a sexta, sétima (ambas com uma) e décima (três). Boa parte das correspondências alocadas a esta heurística estão relacionadas à maior comodidade na composição de *scripts*, que, além de ser um dos aspectos fundamentais da interação com CLIs, é um facilitador da eficiência e flexibilidade de uso do sistema. Sugestões dadas pelo guia que promovem tal conveniência estão relacionadas à não obrigatoriedade da solicitação de informações por meio de *prompt* ou outros elementos interativos, ao mantimento do comportamento antigo de *flags* mesmo após atualizações e ao uso de nomes de subcomandos por extenso.

6.8 Projeto estético e minimalista

Quatro correspondências relacionadas à oitava heurística foram identificadas, que estão divididas entre a segunda e quarta seções, ambas com dois itens cada. Os itens da segunda seção estão relacionados à concisão de informações dada pelo programa —

mais especificamente quanto ao texto de ajuda exibido por padrão — e à localização e organização de exemplos de uso do programa que, se muito extensos, podem poluir o texto de ajuda. Os itens da quarta seção referem-se à saída sucinta após sucesso na execução de um programa (por padrão, nada é exibido nesses casos) e ao uso adequado de cores.

6.9 Reconhecimento, diagnóstico e recuperação de erros

Dez correspondências relacionadas à nona heurística foram identificadas, que estão divididas entre a primeira (dois itens), quarta (dois itens), quinta (quatro itens) e nona (dois itens) seções. Os itens da primeira seção são relacionados ao retorno adequado para sucesso ou erro e direcionamento de *logs* e mensagens erros para o fluxo adequado. Os itens da segunda seção estão ligados à clareza da saída dada por programas no geral — isto é, independente de serem mensagens de erro ou não — e à comodidade na visualização de *logs* com muitas linhas.

Os itens da quinta seção relacionam-se à clareza das mensagens de erro, tanto no que se refere à legibilidade do conteúdo destas por humanos, quanto à organização e formatação do texto dessas mensagens e ao fornecimento de informações que realmente são relevantes para o diagnóstico de erros. Já o item encontrado na nona seção é pertinente à própria continuação e um processo pelo usuário após falha no ponto da execução anterior à falha.

6.10 Ajuda e documentação

Onze correspondências relacionadas à décima heurística foram identificadas, que estão divididas a primeira (um item), segunda (sete itens) e terceira (três itens) seções. A correspondência verificada na primeira seção sugere a adoção de bibliotecas de análise de argumento que, muitas vezes, permitem o tratamento de texto de ajuda disponibilizado pelo programa. Os itens encontrados na segunda seção estão relacionados à própria disponibização de ajuda extensa quando solicitado e versão *web* da documentação, fornecimento de exemplos de uso, clareza da formatação do texto de ajuda e o oferecimento de ajuda contextual após uso inadequado do programa. Os itens da terceira seção referem-se ao fornecimento de documentação em versão *web* e local — tanto no próprio texto de ajuda do programa, quanto através de páginas do utilitário de documentação `man`.

7 RECOMENDAÇÕES

Após a realização da análise comparativa descrita no capítulo anterior, foi elaborada uma lista que contém, genericamente, as maneiras de aplicar as heurísticas de Nielsen no contexto das CLIs já mencionadas pelo guia CLIG. Através da interpretação das sugestões descritas por Nielsen (1993c), foram listadas maneiras complementares de executar essa aplicação, adaptando, quando possível, os conceitos tratados pelo autor à realidade da interação em interfaces textuais. Além da descrição de 1994 das heurísticas — à qual foi dada preferência, por conta de que, muitas vezes, as sugestões mencionadas são mais compatíveis com recursos presentes em CLIs — também foram consultados os exemplos dados pela última versão das heurísticas, mantida pelo sítio do Nielsen Norman Group (NN/g). Na lista abaixo são ilustradas essas maneiras juntamente às já adotadas. São indicadas com um asterisco (*) as maneiras de aplicação adicionais e em **negrito** as maneiras às quais fora elaboradas novas diretrizes. Foram elaboradas diretrizes relacionadas a segunda, sétima, oitava e décima heurísticas.

1. Visibilidade do estado do sistema
    - Indicação de sucesso ou erro
    - Indicação de ações futuras após comportamento inesperado
    - Saída legível por máquina
    - Indicação do estado do sistema
    - Explicitação de ação crítica
    - Clareza através de indicadores visuais
    - Eficiência na visualização do estado do sistema
    - Responsividade
2. Correspondência entre o sistema e o mundo real
    - Saída legível por humanos
    - Metáforas
    - **Internacionalização***
3. Controle e liberdade do usuário
    - Saída acessível
    - Mecanismo para desfazer ação
4. Consistência e padrões
    - Consistência externa
    - Consistência interna

5. Prevenção de erros
    - Explicitação de ação crítica
    - Clareza da entrada
    - Redução de ambiguidades
    - Bons padrões
    - Segurança de informações sensíveis
    - Prevenção de erros futuros
    - Consistência interna
6. Reconhecimento em vez de memorização
    - Incentivo à exploração do sistema
    - Reconhecimento de opções disponíveis
7. Flexibilidade e eficiência de uso
    - Facilitamento da composição de *scripts*
    - **Incentivo à personalização***
8. Projeto estético e minimalista
    - Concisão de informações
    - **Uso adequado de cores**
9. Reconhecimento, diagnóstico e recuperação de erros
    - Reconhecimento de erros
    - Recuperação de erros
10. Ajuda e documentação
    - Disponibilização de ajuda e documentação
    - Disponibilização de exemplos de uso
    - **Ajuda contextual**

A partir das categorias de formas de aplicação complementares e determinadas categorias existentes, foram elaboradas quatro diretrizes adicionais para o guia CLIG. Buscou-se estruturar as diretrizes suplementares de maneira similar à das existentes. Isto é, cada uma delas é composta por uma descrição breve da diretriz seguida de detalhes específicos. Foram, quando possível, disponibilizados exemplos de implementação ou sugestões de utilitários que auxiliam na concretização de determinada diretriz. Todas as novas diretrizes foram alocadas a seções que já constam no guia. Na lista abaixo são descritas as novas diretrizes agrupadas por heurística correspondente. As diretrizes foram alocadas a seções onde julgou-se mais adequado, levando em consideração os conceitos abordados por cada uma delas.

2. Correspondência entre o sistema e o mundo real
   Internacionalização
   4.17. Suporte internacionalização e localização. Internacionalização e localização são maneiras de determinado programa para diferentes línguas, regionais e requisitos técnicos do locale alvo. Considere usar ferramentas como `gettext`[1] e ICU[2].
4. Flexibilidade e eficiência de uso
   Incentivo à personalização
   12.5. Sugira a criação de um arquivo de configuração exemplo localmente durante o processo de instalação. Isso tende a induzir o usuário a explorar e personalizar as opções de configuração do programa.
8. Projeto estético e minimalista
   Uso adequado de cores
   4.20. Garanta que mensagens possam ser interpretadas sem o uso de cores. Codificação por meio de cores devem suplementadas por meio de indicações redundantes — por exemplo, mensagens breves, caracteres especiais e emojis. Por exemplo, ao invés indicar um erro apenas por meio de uma mensagem em vermelho, explicite no início da mensagem que ela descreve um erro.
10. Ajuda e documentação
    Ajuda contextual
    12.6. Comente em detalhe o arquivo de configuração exemplo. Ao ser feito isso, o usuário é poupado de ter que verificar a documentação principal do programa sempre que quiser obter informações rápidas sobre determinada opção.

Apesar da importância de, sempre que possível, o texto de interfaces ser exibido na língua nativa do usuário (Nielsen, 1993c), percebeu-se não há menção da internacionalização pelo guia CLIG. Acredita-se que isso ocorre por conta da ampla adoção do inglês como lingua franca no contexto do desenvolvimento de *software*. Muitos usuários, porém, tendem a ter mais domínio em sua própria lingua nativa do que na lingua inglesa (Projeto GNU, 2025). Foi elaborada, então, a diretriz 4.17., relacionada ao conceito da internacionalização, que sugere o suporte à internacionalização e localização (i18n e l10n) pelo programa; isto é, a adaptação do programa para que este possa ter mensagens traduzidas com mais facilidade por desenvolvedores.

Como forma de encorajar a personalização por parte do usuário, foi proposto na diretriz 12.5. sugerir a criação de um arquivo de configuração exemplo local — isto é, no diretório pessoal do usuário — durante o processo de instalação do programa. Recomendou-se essa diretriz sob a justificativa de que, ao ser executada essa sugestão, o usuário pode ser instigado a alterar as configurações padrão do programa para adequar estas a suas

[1] `gettext` é um sistema de internacionalização e localização comumente utilizado no desenvolvimento de programas que suportam múltiplos idiomas.
[2] *International Components for Unicode* (ICU) é um projeto de bibliotecas de C/C++ e Java para suporte de Unicode e internacionalização.

necessidades: além de se tornar claro, no processo de sugestão da criação do arquivo, que é possível a alteração da configuração padrão, ainda é oferecida a comodidade de editar as opções do programa sem a exigência de permissões especiais, como ocorre caso seja necessário editar arquivos localizados em diretórios do sistema.

Associada à sugestão da diretriz descrita acima está a da diretriz 12.6., que orienta comentar em pormenor arquivos de configuração exemplo. Levando em consideração que, ao ser feito isso, o usuário não precisará necessariamente consultar a documentação sempre que desejar alterar determinada opção de um programa — trechos relevantes da documentação podem ser disponibilizados no local adequado do arquivo — essa ação constitui uma forma de oferecer ajuda contextual no momento do ajuste das preferências de utilitários.

Em conformidade com as sugestões relacionadas ao uso adequado de cores de Rice (1991), reforçadas posteriormente por Nielsen (1993c), foi elaborada a diretriz 4.20. Como mencionado pelos autores, boa parte da população (cerca de 5% da população mundial (Conselho Federal de Medicina, 2004)) é portadora de daltonismo. Sugere-se então na diretriz 4.20. que seja possível a interpretação mensagens mesmo sem o uso de cores, e que codificação baseada em cor deve ser suplementada por outros elementos gráficos, que, no contexto das CLIs podem ser mensagens textuais por extenso, caracteres especiais ou mesmo emojis.

7.1 Discussão

Ao serem interpretadas as heurísticas propostas por Nielsen (1993c), notou-se que a aplicação de algumas delas tende a assumir a presença de interfaces com determinados recursos, apesar de o autor afirmar que as recomendações podem ser seguidas na avaliação ou desenvolvimento de qualquer tipo de interface. Dentre esses recursos está a possibilidade de exibição rápida ou, preferencialmente, contínua de elementos manipuláveis pelo usuário. Como já mencionado, em CLIs o estado do sistema não é exibido continuamente. Logo, não é difícil notar que, em tese, alguns conceitos relacionados à primeira e sexta heurística são de difícil adaptação a CLIs.

De modo geral, apesar de a aplicação das heurísticas ser possível no contexto de CLIs — embora em abrangência menor do que a possível em, por exemplo, interfaces de manipulação direta — a ausência de certos recursos nesse tipo de interface dificulta a adoção de parte das recomendações propostas. Isso é especialmente evidente em casos onde o que é desejado por certa heurística é, com efeito, o contrário de algo inerente à forma de interação com CLIs, como no exemplo dado.

Dessa maneira, excetuando-se aspectos intrínsecos ao paradigma, acredita-se que seja possível, por parte de desenvolvedores, a implementação de mecanismos que facilitem a execução das prescrições das heurísticas. No entanto, deve ser ressaltado que certos fatores impedem essa implementação. Incluem-se aqui fatores históricos relacionados à forma como programas que aderem à filosofia UNIX são planejados. Como exemplo destes, podem ser mencionados boa parte dos utilitários fornecidos pela coleção de programas GNU `coreutils`[3] ou equivalentes, presentes em virtualmente todas os sistemas tipo UNIX e, consequentemente, utilizados por um grande número de usuários humanos e *scripts*, muitas vezes presentes em sistemas legados.

[3] O *GNU Core Utilities* ou `coreutils` é uma coleção de programas que implementa diversos comandos padrão em sistemas tipo UNIX, como `ls`, `rm` e `cat`.

8 CONSIDERAÇÕES FINAIS

Este estudo propôs-se a examinar a aplicabilidade de princípios gerais de usabilidade, representados aqui pelas heurísticas propostas por Nielsen (1993c), a interfaces de linha de comando. Para investigar se e como essa aplicabilidade ocorre, buscou-se identificar em que pontos as heurísticas são adaptáveis aos recursos de CLIs e, especialmente, características desse tipo de interface que comprometem a adoção das heurísticas. Nesse segundo tópico, o foco foi identificar características fortemente ligadas à forma de interação em CLIs, cujos prejuízos não são facilmente contornáveis apenas por meio de mecanismos implementáveis por programas. Um candidato identificado a essa categoria de características é a não-continuidade da exibição do estado do sistema. Fatores históricos relacionados a forma como programas são tradicionalmente planejados e utilizados, que tendem a impossibilitar a adoção de boas práticas de usabilidade, também foram considerados, apesar dos prejuízos causados por estes serem, possívelmente, contornáveis via implementação.

A identificação de características de CLIs que comprometem a adoção das heurísticas ocorreu por meio da verificação se recursos requisitados pelas recomendações são possíveis no tipo de interface em questão. Já a identificação de conceitos das heurísticas que são adaptáveis a CLIs deu-se pela análise comparativa Nielsen-CLIG, complementada pela interpretação dos exemplos e conceitos abordados por Nielsen (1993c). Esse complemento foi empregado na elaboração de diretrizes adicionais para o guia CLIG relacionadas à internacionalização e ao incentivo à personalização. Além das diretrizes relacionadas a esses conceitos presentes nas lacunas conceituais, foram também desenvolvidas outras ligadas ao reconhecimento de opções disponíves, uso adequado de cores e ajuda contextual. Pretende-se posteriormente encaminhar as recomendações escritas ao repositório `git` das diretrizes CLIG para que as sugestões sejam analisadas pelos autores do guia.

No que se refere à metodologia e à validade dos resultados, o trabalho apresenta algumas limitações que devem ser consideradas na interpretação das conclusões. As análises comparativas foram conduzidas de forma interpretativa e por apenas uma pessoa, o que pode ter influenciado a interpretação das duas listas de recomendações inspecionadas. A primeira análise, além de não ter sido documentada em detalhe, foi menos minuciosa, o que pode ter impactado a seleção dos conceitos abordados no questionário. A amostra deste foi pequena e relativamente homogênea, o que dificulta a representação a representação de diferentes perfis de usuários de CLIs. Além disso, as recomendações elaboradas basearam-se em uma fonte principal não específica para CLIs. Investigações futuras poderiam considerar espaços mais amplos frequentados por usuários desse tipo de interface, como fóruns *on-line*, a fim de levantar um conjunto mais abrangente de formas de aplicação das heurísticas em CLIs. Por fim, as diretrizes propostas não foram validadas com usuários, não sendo possível, assim, verificar sua efetividade.

Possibilidades de trabalhos futuros incluem a averiguação de como outros critérios de uso em IHC se comportam no contexto das CLIs. Por exemplo,quanto à acessibilidade, características como a base textual e a interação por teclado sugerem que esse tipo de interface é acessível a portadores de deficiências visuais. No entanto, estudos realizados por (Sampath *et al.*, 2021) com usuários cegos revelaram uma série de problemas na interação com CLIs. Apesar do fornecimento de um conjunto de sugestões para a melhora da acessibilidade nesse contexto, acredita-se que ainda há espaço para aprimoramentos. Dentre estes está a comprovação da eficácia dessas sugestões, visto que os autores não mencionaram a realização da avaliação das recomendações desenvolvidas após os estudos. Cabot (2017) propõe também uma série de trabalhos futuros. Alguns desses são o estudo de mecanismos de sugestão em CLIs, o aproveitamento dos benefícios de CLIs em produtos de *software* voltado para o público geral e a personalização de CLIs pelo usuário. Outro assunto tratável é a própria validade das recomendações do CLIG. Acredita-se que as sugestões do guia foram elaboradas com base em experiências dos próprios autores do texto — em sua maioria engenheiros experientes — cujo nível de instrução pode ter limitado a representação das dificuldades enfrentadas por usuários iniciantes.

Em síntese, este trabalho buscou contribuir no debate acadêmico acerca da interação com CLIs. Como mencionado, a abrangência desse debate, até o momento, permanece limitada, apesar da relevância do tipo de interface estudado em diversos contextos. Espera-se que as contribuições deste estudo sirvam como ponto de partida para investigações futuras que aprofundem o entendimento da interação com CLIs, bem como no desenvolvimento de melhorias práticas da experiência de uso nesse tipo de interface.

REFERÊNCIAS

APPLE COMPUTER, INC. **Apple Human Interface Guidelines**: The Apple Desktop Interface. Cupertino, CA, EUA, 1987.

AUTODESK. **AutoCAD Architecture 2011 User's Guide**. [*S. l.*: *s. n.*], 2010.

BARBOSA, S. D. J.; SILVA, B. S. da; SILVEIRA, M. S.; GASPARINI, I.; DARIN, T.; BARBOSA, G. D. J. **Interação Humano-Computador e Experiência do Usuário**. Rio de Janeiro, RJ, Brasil: Autopublicação, 2021. ISBN 978-65-00-19677-1.

CABOT, R. B. **Re-imagining the Command Line User Experience for Problem Solving**. Bacharel em Ciência da Computação – Universidade de Bath, Bath, Reino Unido, mai. 2017.

CARD, S. K.; MORAN, T. P.; NEWELL, A. **The Psychology of Human-Computer Interaction**. Hillsdale, NJ, EUA: Lawrence Erlbaum Associates, 1983. 4 p. ISBN 0-89859-243-7.

Conselho Federal de Medicina. **Daltonismo: distúrbio atinge 5% da população mundial**. 2004. Disponível em: https://clig.dev/. Acesso em: 3 jan. 2026.

DIX, A.; FINLAY, J.; ABOWD, G. D.; BEALE, R. **Human-Computer Interaction**. 3. ed. Harlow, Essex, Inglaterra: Pearson Education Limited, 2004. ISBN 978-0-13-046109-4.

GENTNER, D.; NIELSEN, J. The anti-mac interface. **Communications of the ACM**, v. 39, n. 8, p. 70–82, ago. 1990.

HERON, M. J. A case study into the accessibility of text-parser based interaction. In: ASSOCIATION FOR COMPUTING MACHINERY. **Proceedings of the 7th ACM SIGCHI symphosium on engineering interactive computing system (EICS'15)**. Duisburgo, Alemanha, 2015. p. 74–83.

HEWETT, T. T.; BAECKER, R.; CARD, S.; CAREY, T.; GASEN, J.; MANTEI, M.; PERLMAN, G.; STRONG, G.; VERPLANK, W. **ACM SIGCHI Curricula for Human-Computer Interaction**. Nova York, NY, EUA, 1992.

HUNT, B. R.; LIPSMAN, R. L.; ROSENBERG, J. M. **A Guide to MATLAB for Beginners and Experienced Users**. Cambridge, Reino Unido: Cambridge University Press, 2001. ISBN 978-0-511-07792-0.

ISO/IEC 25010:2011(E). **Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models**. Geneva, Suíça, 2011.

KERNIGHAN, B. W.; PIKE, R. C. **The Unix Programming Environment**. Englewood Cliffs, NJ, EUA: Prentice-Hall, Inc., 1984. ISBN 0-13-937699-2.

MOLICH, R.; NIELSEN, J. Improving a human-computer dialogue. **Communications of the ACM**, v. 33, n. 3, p. 338–348, mar. 1990.

NIELSEN, J. Traditional dialogue design applied to modern user interfaces. **Communications of the ACM**, v. 33, n. 10, p. 109–118, out. 1990e.

NIELSEN, J. **Usability Engineering**. Mountain View, CA, EUA: Academic Press, 1993c. 26 p. ISBN 0-12-518406-9.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. In: ASSOCIATION FOR COMPUTING MACHINERY. **CHI '90 Proceedings**. [*S.l.*], 1990. p. 249–256.

NORMAN, D. A. The next ui breakthrough: Command lines. **ACM Interactions**, p. 44–45, mai. 2007.

PRASAD, A.; FIRSHMAN, B.; TASHIAN, C.; PARISH, E. **Command Line Interface Guidelines CLIG**. 2025. Disponível em: https://clig.dev/. Acesso em: 4 jan. 2026.

PREECE, J.; ROGERS, Y.; SHARP, H. **Interaction Design**: Beyond Human-Computer Interaction. 6. ed. Hoboken, Nova Jersey, EUA: John Wiley & Sons, Inc., 2023. ISBN 978-1-119-90110-5.

PREECE, J.; ROGERS, Y.; SHARP, H.; BENYON, D.; HOLLAND, S.; CAREY, T. **Human-Computer Interaction**. [*S.l.*]: Addison-Wesley Publishing Company, 1994. 7 p. ISBN 0-201-62769-8.

Projeto GNU. **GNU gettext utilities**. 2025. Disponível em: https://www.gnu.org/software/gettext/manual/gettext.html. Acesso em: 5 jan. 2026.

RICE, J. F. Display color coding: 10 rules of thumb. **IEEE Software**, v. 8, p. 86–88, jan. 1991.

SAMPATH, H.; MERRICK, A.; MACVEAN, A. Accessibility of command line interfaces. In: ASSOCIATION FOR COMPUTING MACHINERY. **CHI '21: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems**. Yokohama, Japão, 2021. p. 1–10. ISBN 978-1-4503-8096-6.

SCHMITZHAUS, C. **Usabilidade em Interfaces de Linha de Comando**. Tecnólogo em Sistemas para Internet – Universidade Federal de Santa Maria, Santa Maria, RS, BR, dez. 2024.

SCHRÖDER, M.; CITO, J. An empirical investigation of command-line customization. **Empirical Software Engineering**, v. 27, n. 30, p. 338–348, dez. 2021.

SENSOMATIC. **The graphical user interface**: Time for a paradigm shift? 2025. Disponível em: https://www.sensomatic.com/chz/gui/history. Acesso em: 25 nov. 2025.

SHNEIDERMAN, B.; PLAISANT, C.; COHEN, M.; JACOBS, S.; ELMQVIST, N.; DIAKOPOULOS, N. **Designing the User Interface**: Strategies for Effective Human-Computer Interaction. [*S.l.*]: Pearson, 2018. 329 p. ISBN 1-292-15391-1.

WELSH, T.; WELSH, D. **Priming the Pump**: How TRS-80 Enthusiasts Helped Spark the PC Revolution. Ferndale, Michigan, EUA: The Seeker Books, 2007. ISBN 978-0-9793468-0-4.

APÊNDICE A – QUESTIONÁRIO

O conteúdo deste apêndice foi extraido do questionário realizado com a plataforma de gerenciamento de questionários eletrônicos *Google Forms* (https://forms.google.com). A legenda de símbolos adotada é a seguinte: ☐: alternativa de item em que múltiplas opções poderiam ser selecionadas; ◯: alternativa de item em que apenas uma opção poderia ser selecionada; ⊜: item com apenas duas alternativas (SIM/NÃO). A resposta a todos os itens foi obrigatória, exceto aos da quarta seção.

Descrição do formulário

Este formulário faz parte de uma pesquisa sobre a usabilidade em interfaces de linha de comando. As perguntas a seguir abordam sua experiência com o uso do terminal/prompt de comando, incluindo tarefas comuns, compreensão de mensagens, comandos e atalhos.

Termo de consentimento

Esta é uma pesquisa acadêmica realizada no contexto da atividade de Trabalho de Conclusão de Curso I, vinculada ao curso de Design Digital da Universidade Federal do Ceará.

Objetivos da pesquisa

Esta pesquisa tem como objetivo investigar a experiência de uso de interfaces de linha de comando, com foco na usabilidade percebida pelos usuários — especialmente em relação à facilidade de uso, reconhecimento de comandos, compreensão de mensagens, prevenção e reversão de erros, e uso de atalhos.

Atividades envolvidas

A participação consiste em responder a um formulário sobre seu uso de interfaces de linha de comando. As perguntas são de múltipla escolha e baseiam-se em situações comuns ao utilizar esse tipo de interface. O tempo estimado para o preenchimento é de aproximadamente 5 (cinco) minutos.

Riscos e benefícios

A pesquisa não apresenta riscos previsíveis à sua integridade ou bem-estar. Os benefícios são indiretos e estão relacionados à sua contribuição para estudos que visam melhorar a usabilidade de ferramentas em linha de comando.

Privacidade e confidencialidade

As respostas fornecidas serão utilizadas exclusivamente para fins acadêmicos. Nenhuma informação pessoal identificável será divulgada. Os dados serão tratados com sigilo e armazenados de forma segura.

Contato

Para dúvidas ou mais informações sobre a pesquisa, entre em contato pelo e-mail: pedroholiveira@alu.ufc.br.

A.1 Informações básicas

1. Qual sua área de atuação?
   - ☐ Tecnologia da Informação / Computação
   - ☐ Design / Comunicação / Artes
   - ☐ Engenharias / Exatas
   - ☐ Ciências Humanas / Sociais / Educação
   - ☐ Outro: _____
2. Qual seu nível técnico geral?
   - ○ Leigo
   - ○ Básico
   - ○ Intermediário
   - ○ Avançado
3. ⊜ Você usa o terminal/prompt de comando?[1]

A.2 Informações técnicas

1. Qual seu sistema operacional primário?
   - ○ Linux
   - ○ macOS
   - ○ FreeBSD
   - ○ Windows
   - ○ Outro: _____
2. Há quanto tempo você utiliza o terminal/prompt de comando?
   - ○ Menos de 6 meses
   - ○ Entre 6 meses e 1 ano
   - ○ Entre 1 ano e 2 anos
   - ○ Entre 2 anos e 5 anos
   - ○ Mais de 5 anos

[1] Pergunta eliminatória. Caso assinalasse NÃO, o respondente seria direcionado para o fim do questionário.

3. Com que frequência você usa o terminal/prompt de comando?
   - ○ Raramente: menos de uma vez por mês
   - ○ Ocasionalmente: algumas vezes por mês
   - ○ Regularmente: várias vezes por semana
   - ○ Frequentemente: diariamente ou quase todos os dias

A.3 Experiência de uso com CLIs

1. ⦵ Já tive dificuldade para entender mensagens do terminal/prompt de comando que, após pesquisar, percebi que tratavam de algo simples.
2. ⦵ Consigo interpretar mensagens de erro complexas sem precisar procurar ajuda externa.
3. ⦵ Acredito que o terminal/prompt de comando me dá indicações claras sobre quais funcionalidades estão disponíveis.
4. ⦵ Normalmente, consigo adivinhar o que um comando faz só pelo nome dele.
5. ⦵ Já tive dificuldade em localizar programas que realizam tarefas básicas como editar texto, copiar arquivos ou verificar espaço em disco.
6. ⦵ Costumo confundir ou esquecer o que significam parâmetros de comandos — por exemplo, `-v` ou `--verbose` (mostrar saída detalhada de um comando).
7. ⦵ Preciso consultar documentação ou tutoriais mesmo para realizar tarefas simples.
8. ⦵ Acredito que uso do terminal/prompt de comando só se torma fácil depois de estudar por um tempo: não considero algo intuitivo.
9. ⦵ Uso com frequência a função de autocompletar para lembrar nomes de arquivos, comandos ou argumentos.
10. ⦵ Já apaguei ou sobrescrevi arquivos por engano por não ter recebido uma confirmação antes da ação.
11. ⦵ Já confundi ou esqueci a ordem correta dos argumentos de comando — por exemplo: `mv origem.txt destino.txt`.
12. ⦵ Considero fácil criar atalhos de teclado no terminal/prompt de comando.
13. ⦵ Sei de memória vários atalhos de teclado que estão presentes por padrão no terminal/prompt de comando.
14. Por que você usa o terminal/prompt de comando em vez da interface gráfica para realizar determinadas tarefas?
    - ☐ Consome menos recursos do sistema (memória, CPU).
    - ☐ É mais rápido ou eficiente para certas tarefas.
    - ☐ Permite automatizar processos com scripts.
    - ☐ Algumas tarefas só são viáveis via terminal/prompt de comando.
    - ☐ É necessário para o meu trabalho ou estudos.

- ☐ É mais fácil de documentar ou compartilhar comandos.
- ☐ Oferece mais controle ou opções avançadas.
- ☐ Acostumei-me ou prefiro por hábito.
- ☐ Uso por aprendizado ou para praticar comandos.
- ☐ Algumas ferramentas só têm versões para terminal.
- ☐ Outro: _____

A.4 Sugestões e contato

1. Conte-nos algo que você gostaria de acrescentar: impressões, experiências ou pontos que não tenhamos abordado.

   ---

2. Você gostaria de participar de futuras pesquisas sobre usabilidade de terminais e interfaces de linha de comando? Se sim, insira abaixo seu e-mail para contato.

APÊNDICE B – COMPARAÇÃO ENTRE NIELSEN E CLIG

É ilustrada no Quadro 3 uma visão geral dos mapeamentos entre as diretrizes do CLIG e as heurísticas de Nielsen. Em azul são indicadas correspondências diretas e em laranja as indiretas. No quadro são mostradas as descrições breves de cada diretriz na horizontal à esquerda e na parte superior a numeração de cada heurística. A planilha onde a análise foi originalmente documentada está disponível em texto corrido na lista após o quadro para melhor legibilidade. Na lista, as correspondências foram agrupadas em três níveis: **1.** heurística, **2.** seção do CLIG e **3.** correspondência. Quando dois ou mais mapeamentos próximos receberam a mesma justificativa, eles foram agrupados — por exemplo, na primeira heurística e primeira seção do guia, as diretrizes 1.3. e 1.4. A numeração entre parênteses no início de cada heurística ou seção do CLIG indica a quantidade de correspondências: à esquerda quantas foram identificadas, à direita o total de itens da seção.

Quadro 3: Visão geral dos mapeamentos CLIG–Nielsen

			Heurísticas de Nielsen									
			1	2	3	4	5	6	7	8	9	10
Command-Line Interface Guidelines	1. O básico	1.1. Use a command-line argument parsing library where you can						■				■
		1.2. Return zero exit code on success, non-zero on failure	■			■					■	
		1.3. Send output to `stdout`	■			■						
		1.4. Send messaging to `stderr`	■			■					■	
	2. Ajuda	2.1. Display extensive help text when asked										■
		2.2. Display concise help text by default								■		
		2.3. Show full help when `-h` and `-help` is passed				■						
		2.4. Provide a support path for feedback and issues										
		2.5. In help text, link to the web version of the documentation										■
		2.6. Lead with examples										■
		2.7. If you've got loads of examples, put them somewhere else								■		■
		2.8. Display the most common flags and commands at the start of the help text										■
		2.9. Use formatting in your help text										■
		2.10. If the user did something wrong and you can guess what they meant, suggest it						■				
		2.11. If your comm. is expect. to have someth. piped to it & `stdin` is an interactv. term., display help immediat. & quit	■									■

Quadro 3: Visão geral dos mapeamentos CLIG–Nielsen (continuação)

Command-Line Interface Guidelines			1	2	3	4	5	6	7	8	9	10
	3. Docum.	3.1. Provide web-based documentation										■
		3.2. Provide terminal-based documentation										■
		3.3. Consider providing man pages				■						■
	4. Saída	4.1. Human-readable output is paramount		■							■	
		4.2. Have machine-readable output where it does not impact usability	■									
		4.3. If human-readable output breaks machine-readable output, use `-plain` to display output in plain, tabular text format	■			■						
		4.4. Display output as formatted JSON if `-json` is passed	■			■						
		4.5. Display output on success, but keep it brief	■			■				■		
		4.6. If you change state, tell the user	■									
		4.7. Make it easy to see the current state of the system	■									
		4.8. Suggest commands the user should run										
		4.9. Actions crossing the boundary of the program's internal world should usually be explicit	■					■				
		4.10. Increase information density — with ASCII art!	■	■								
		4.11. Use color with intention	■	■						■		
		4.12. Disable color if your program is not in a terminal or the user requested it										
		4.13. If `stdout` is not an interactive terminal, don't display any animations										
		4.14. Use symbols and emoji where it makes things clearer	■	■								
		4.15. By default, don't output information that's only understandable by the creators of the software		■								
		4.16. Don't treat `stderr` like a log file, at least not by default					■					
		4.17. Use a pager (e.g., `less`) if you are outputting a lot of text	■								■	
	5. Erros	5.1. Catch errors and rewrite them for humans									■	
		5.2. Signal-to-noise ratio is crucial									■	
		5.3. Consider where the user will look first									■	
		5.4. If there is an unexpected or unexplainab. error, provide debug & traceback info., & instruct. on how to submit a bug									■	
		5.5. Make it effortless to submit bug reports										
	6. Argumentos	6.1. Prefer flags to args					■					
		6.2. Have full-length versions of all flags					■	■				
		6.3. Only use one-letter flags for commonly used flags										
		6.4. Multiple arguments are fine for simple actions against multiple files					■					
		6.5. If you've got two or more arguments for different things, you're probably doing something wrong										
		6.6. Use standard names for flags, if there is a standard				■						
		6.7. Make the default the right thing for most users					■					
		6.8. Prompt for user input					■					
		6.9. Never *require* a prompt			■				■			

Quadro 3: Visão geral dos mapeamentos CLIG–Nielsen (continuação)

			1	2	3	4	5	6	7	8	9	10
Command-Line Interface Guidelines	6. Argumentos	6.10. Confirm before doing anything dangerous					■					
		6.11. If input or output is a file, support `-` to read from `stdin` or write to `stdout`				■						
		6.12. If a flag can accept an optional value, allow a special word like “none”					■					
		6.13. If possible, make arguments, flags and subcommands order-independent										
		6.14. Do not read secrets directly from flags					■					
	7. Interatividade	7.1. Only use prompts or interactive elements if `stdin` is an interactive terminal (a TTY)				■			■			
		7.2. If `-no-input` is passed, don’t prompt or do anything interactive										
		7.3. If you’re prompting for a password, don’t print it as the user types					■					
		7.4. Let the user escape			■	■						
	8. Subcom.	8.1. Be consistent across subcommands				■						
		8.2. Use consistent names for multiple levels of subcommand				■						
		8.3. Don’t have ambiguous or similarly-named commands					■					
	9. Robustez	9.1. Validate user input					■					
		9.2. Responsive is more important than fast	■									
		9.3. Show progress if something takes a long time	■									
		9.4. Do stuff in parallel where you can, but be thoughtful about it	■									
		9.5. Make things time out	■									
		9.6. Make it recoverable									■	
		9.7. Make it crash-only									■	
		9.8. People are going to misuse your program					■					
	10. Planejando o futuro	10.1. Keep changes additive where you can				■	■		■			
		10.2. Warn before you make a non-additive change					■		■			
		10.3. Changing output for humans is usually OK							■			
		10.4. Don’t have a catch-all subcommand										
		10.5. Don’t allow arbitrary abbreviations of subcommands							■			
		10.6. Don’t create a “time bomb.”					■					
	11. Sinais	11.1. If a user hits Ctrl-C (the INT signal), exit as soon as possible			■							
		11.2. If a user hits Ctrl-C during clean-up operations that might take a long time, skip them			■		■					
	12. Configuração	12.1. Use flags										
		12.2. Use flags and probably environment variables too										
		12.3. Use a command-specific, version-controlled file										
		12.4. Follow the XDG-spec				■						
		12.5. If you auto. modif. config. that is not your program’s, ask the user for consent & tell them exactly what you’re doing	■									
		12.6. Apply configuration parameters in order of precedence										

Quadro 3: Visão geral dos mapeamentos CLIG–Nielsen (continuação)

			1	2	3	4	5	6	7	8	9	10
Command-Line Interface Guidelines	13. Variáveis de ambiente	13.1. Environment variables are for behavior that *varies with the context* in which a command is run					■					
		13.2. For max. portabil., environ. var. names must only contain uppercase letters, numbers, and underscores				■						
		13.3. Aim for single-line environment variable values					■					
		13.4. Avoid commandeering widely used names				■						
		13.5. Check general-purpose environment variables for configuration values when possible				■						
		13.6. Read environment variables from `.env` where appropriate				■						
		13.7. Don't use `.env` as a substitute for a proper configuration file				■						
		13.8. Do not read secrets from environment variables					■					
	14. Nomeação	14.1. Make it a simple, memorable word						■				
		14.2. Use only lowercase letters, and dashes if you really need to		■		■						
		14.3. Keep it short		■		■						
		14.4. Make it easy to type		■								
	15. Dist.	15.1. If possible, distribute as a single binary										
		15.2. Make it easy to uninstall				■						
	16. Est.	16.1. Do not phone home usage or crash data without consent										
		16.2. Consider alternatives to collecting analytics										

Fonte: Elaborado pelo autor (2026)

I. (20/99) Visibilidade do estado do sistema

1. (3/5) O básico

1.2. Return zero exit code on success, non-zero on failure.

Indicação de sucesso ou erro

Por meio da saída zero, o usuário ou outros programas recebem retorno que indica se uma operação falhou ou sucedeu, mantendo transparência sobre o que aconteceu.

1.3. Send output to `stdout`.

1.4. Send messaging to `stderr`.

Indicação de sucesso ou erro

Quando a saída de um programa é enviada para o fluxo adequado, usuários e outros programas podem ver ou capturar os resultados da execução de um programa com segurança. Isso previne confusão sobre o local onde o retorno é recebido e reforça a percepção sobre o estado atual do sistema.

2. (1/11) Ajuda
   2.11. If your command is expecting to have something piped to it and `stdin` is an interactive terminal, display help immediately and quit.
   Indicação de ações futuras após comportamento inesperado
   Explicitar que o comando necessita receber entrada por meio de uma *pipeline* contribui para que o usuário não precise adivinhar isso. O comando `cat`, que é usado como exemplo no gua, não dá retorno algum ao ser executado sem argumentos, o que pode fazer o usuário supor que algo de errado aconteceu.
4. (11/17) Saída
   4.2. Have machine-readable output where it does not impact usability.
   4.3. If human-readable output breaks machine-readable output, use `--plain` to display output in plain, tabular text format for integration with tools like `grep` or `awk`.
   4.4. Display output as formatted JSON if `--json` is passed.
   Saída legível por máquina
   Quando a saída pode ser enviada para outros programas — isto é, por meio de encadeamento, que, para que funcione corretamente, a saída deve ser legível por máquina — a visualização do estado do sistema, caso esta seja resultado da utilização da saída de um programa anterior em um outro programa.
   4.5. Display output on success, but keep it brief.
   Indicação de sucesso ou erro
   A não exibição de algum indicador após socesso pode causar confusão sobre o estado atual do sistema. O usuário pode supor que algo deu errado na execução de determinada ação.
   4.6. If you change state, tell the user.
   4.7. Make it easy to see the current state of the system.
   Indicação do estado do sistema
   Transparência na mudança de estado do sistema é essencial para que o usuário saiba que ações tomar em seguida.
   4.9. Actions crossing the boundary of the program's internal world should usually be explicit.
   Explicitação de ação crítica
   Usuário deve sempre ser informado quando programa o que está fora do funcionamento interno dele.

4.11. Use color with intention

Clareza através de indicadores visuais

Uso adequado de cores pode ajudar a expor o estado do sistema de forma mais clara e eficaz.

4.14. Use symbols and emoji where it makes things clearer

Clareza através de indicadores visuais

Uso adequado de símbolos e emojis pode ajudar a expor o estado do sistema de forma mais clara e eficaz.

4.17 Use a pager (e.g. `less`) if you are outputting a lot of text.

Eficiência na visualização do estado do sistema

Paginadores auxiliam na visualização de grandes quantidades de texto por oferecerem melhor navegação do que a visualização da saída direto no terminal.

9. (4/8) Robustez

9.2. Responsive is more important than fast.

9.3. Show progress if something takes a long time.

Responsividade

Retorno rápido auxilia na redução de incertezas, o que implica na interpretação de que o sistema realizou uma ação ou não.

9.4. Do stuff in parallel where you can, but be thoughtful about it.

Eficiência na visualização do estado do sistema

Visualização paralela auxilia na exibição mais eficaz do estado atual do sistema.

9.5. Make things time out.

Responsividade

Indicação de *tempo esgotado* auxilia na interpretação do estado atual do sistema.

12. (1/6) Configuração

12.5. If you automatically modify configuration that is not your program's, ask the user for consent and tell them exactly what you're doing.

Explicitação de ação crítica

Informar o usuário caso haja modificações em configurações externas ao programa é importante para que o usuário saiba o estado atual do sistema.

II. (04/99) Correspondência entre o sistema e o mundo real

4. (04/17) Output

4.1. Human-readable output is paramount.

Saída legível por humanos

Uso de linguagem legível por humanos auxilia na aprendizagem e lembrança de como determinado programa funciona.

4.10. Increase information density — with ASCII art!

4.11. Use color with intention.

4.14. Use symbols and emoji where it makes things clearer.

Metáforas

Arte ASCII, cores, símbolos e emojis são alguns dos símbolos existentes no mundo real que podem facilitar a comunicação sistema-usuário em CLIs.

III. (4/99) Controle e liberdade do usuário

7. (1/4) Interactivity

7.4. Let the user escape.

Saída acessível

Permitir que o programa seja facilmente encerrado pelo usuário promove liberdade deste e controle sobre o sistema.

11. (2/2) Signals and control characters

11.1. If a user hits `CTRL-C` (the INT signal), exit as soon as possible.

11.2. If a user hits `CTRL-C` during clean-up operations that might take a long time, skip them.

Saída acessível

Terminar programa ao usuário pressionar `CTRL-C` promove liberdade deste.

15. (1/2) Distribution

15.2. Make it easy to uninstall.

Mecanismo para desfazer ação

Facilidade para desinstalar programa favorece controle do usuário sobre o sistema.

IV. (24/99) Consistência e padrões

1. (3/4) O básico

1.2. Return zero exit code on success, non-zero on failure.

Consistência externa

Retorno adequado é o padrão em sistemas UNIX.

1.3. Send output to `stdout`.

1.4. Send messaging to `stderr`.

Consistência externa

Retorno da saída no fluxo adequado é o padrão em sistemas UNIX.

2. (1/11) Help

2.3. Show full help when `-h` and `--help` is passed.

Consistência externa

A grande maioria dos programas de linha de comando utiliza esse padrão para exibir ajuda.

3. (1/3) Documentation

3.3. Consider providing man pages.

Consistência externa

Fornecer página `man` é padrão em sistemas UNIX.

4. (4/11) Output

4.3. If human-readable output breaks machine-readable output, use `--plain` to display output in plain, tabular text format for integration with tools like `grep` or `awk`.

Consistência externa

Saída que pode ser usada para outros programas influencia positivamente no mantimento da consistência entre utilitários.

4.4. Display output as formatted JSON if `--json` is passed.

Consistência externa

Uso de formato padrão potencialmente pode ajudar no mantimento de consistência entre sistemas.

4.16. Don't treat `stderr` like a log file, at least not by default.

Consistência externa

O padrão é que o fluxo `stderr` seja usado para exibir mensagens que de fato refletem algum erro.

6. (2/14) Arguments and flags

6.6. Use standard names for flags, if there is a standard.

Consistência externa

Uso de nomes padrão para argumentos e *flags* ajuda na manutenção da consistência externa entre programas.

6.11. If input or output is a file, support `-` to read from `stdin` or write to `stdout`.

Consistência externa

Permitir que a saída de algum programa seja a entrada de outro é padrão em sistemas UNIX.

7. (2/4) Interactivity
   7.1. Only use prompts or interactive elements if `stdin` is an interactive terminal (a TTY).

   Consistência externa

   O comportamento de solicitar prompts ou fazer uso de elementos interativos apenas em terminais interativos costuma ser o padrão mais amplamente adotado.

   7.4. Let the user escape.

   Consistência externa

   É padrão em programas de linha de comando permitir o encerramento destes com certas combinações de teclas (a saber, `CTRL-C` ou `CTRL-D`)
8. (2/3) Subcommands
   8.1. Be consistent across subcommands.
   8.2. Use consistent names for multiple levels of subcommand.

   Consistência interna

   Adoção dessas práticas contribui com a consistência interna de utilitários.
10. (1/6) Future-proofing
   10.1. Keep changes additive where you can.

   Consistência interna

   Manter o comportamento de flags já existentes contribui para a consistência interna do utilitário.
12. (1/6) Configuration
   12.4. Follow the XDG-spec.

   Consistência externa

   Seguir a especificação XDG para a localização de arquivos de configuração contribui para a consistência externa.
13. (5/8) Environment variables
   13.2. For maximum portability, environment variable names must only contain uppercase letters, numbers, and underscores (and mustn't start with a number).

   Consistência externa

   Padronização dos nomes de variáveis de ambiente contribui para a consistência externa.

   13.4. Avoid commandeering widely used names.

   Consistência externa

   Não se apropriar de nomes de variáveis de ambiente amplamente utilizados contribui para a consistência externa.

13.5. Check general-purpose environment variables for configuration values when possible.

Consistência externa

Uso de variáveis de propósito geral contribui para a consistência externa.

13.6. Read environment variables from `.env` where appropriate.

Consistência externa

Ler variáveis de um arquivo `.env` costuma ser o padrão.

13.7. Don't use `.env` as a substitute for a proper configuration file.

Consistência externa

Uso de arquivos de configurações costuma ser o padrão.

14. (2/4) Naming

14.2. Use only lowercase letters, and dashes if you really need to.

Consistência externa

A grande maioria dos comandos não costuma utilizar letras maiúsculas ou traços.

14.3. Keep it short

Consistência externa

Programas costumam ter nomes curtos; nomes com pouquíssimos caracteres são reservados para utilitários usados frequentemente.

V. (20/99) Prevenção de erros

4. (1/17) Output

4.9. Actions crossing the boundary of the program's internal world should usually be explicit.

Explicitação de ação crítica

Explicitação desse tipo de ação potencialmente pode previnir erros ou outras consequências inesperadas.

6. (8/14) Arguments

6.1. Prefer flags to args.

Clareza da entrada

Uso de *flags* ao invés de argumentos posicionais auxilia na prevenção de erros por exigir a inserção da entrada no programa de forma clara.

6.2. Have full-length versions of all flags.

Redução de ambiguidades

O fornecimento de versões por extenso de *flags* auxilia na eliminação de ambiguidades. Pode ser útil especialmente na documentação de *scripts*, cujo código torna-se mais descritivo do que se fossem utilizadas *flags* abreviadas.

6.4. Multiple arguments are fine for simple actions against multiple files.
Redução de ambiguidades
Argumentos com funcionalidades ambíguas podem ser fontes de erros; logo, devem ser evitados.

6.7. Make the default the right thing for most users.
Bons padrões
Deslizes por parte dos usuários podem ser prevenidos por meio do fornecimento de bons comportamentos padrões de um programa.

6.8. Prompt for user input.
Segurança de informações sensíveis
Solicitação opcional de prompt ao usuário pode previnir erros, especialmente ao inserir informações sensíveis, como senhas.

6.10. Confirm before doing anything dangerous.
Explicitação de ação crítica
Confirmação antes de ações críticas potencialmente previne a execução de erros.

6.12. If a flag can accept an optional value, allow a special word like “none”.
Redução de ambiguidades
Prevenir ambiguidades potencialmente pode evitar erros.

6.14. Do not read secrets directly from flags.
Segurança de informações sensíveis
Senhas fornecidas por *flags* são facilmente vazadas; aconselhar o não fornecimento de senhas dessa maneira pode previnir erros relacionados à segurança.

7. (1/4) Interactivity

7.3. If you’re prompting for a password, don’t print it as the user types.
Segurança de informações sensíveis
Exibir senhas potencialmente traz vulnerabilidades de segurança ao usuário. Logo, sugerir que esse comportamento não seja adotado contribui para a prevenção de vazamento de senhas.

8. (1/3) Subcommands

8.3. Don’t have ambiguous or similarly-named commands.
Redução de ambiguidades
Ações ambíguas potencialmente são fontes de confusão, que podem levar a erros.

9. (2/8) Robustness
   9.8. People are going to misuse your program.
   Prevenção de erros futuros
   Planejamento do programa para usos não convencionais potencialmente previne erros.
10. (3/6) Future-proofing
   10.1. Keep changes additive where you can.
   Consistência interna
   Manter o comportamento anterior de um programa auxilia na prevenção de erros por não manter o comportamento antigo.
   10.2. Warn before you make a non-additive change.
   Explicitação de ação crítica
   Advertir quando mudanças não aditivas — por exemplo, que alteram o funcionamento de flags existentes — pode previnir erros. Usuários saberão que devem tomar ações.
   10.6. Don't create a "time bomb."
   Prevenção de erros futuros
   Planejamento do futuro do utilitário pode previnir erros de futuros usuários.
11. (1/2) Signals and control characters
   11.2. If a user hits `CTRL-C` during clean-up operations that might take a long time, skip them.
   Explicitação de ação crítica
   Avisar a usuário sobre o que irá ocorrer caso ele pressione `CTRL-C` novamente — por exemplo, ações destrutivas — contribui para a prevenção de erros.
13. (3/8) Environment variables
   13.8. Do not read secrets from environment variables.
   Segurança de informações sensíveis
   Evitar a leitura de senhas a partir de variáveis de ambiente torna elas menos visíveis e, logo, pode previnir erros relacionados à segurança.

VI. (6/99) Reconhecimento em vez de memorização

1. (1/4) O básico
   1.1. Use a command-line argument parsing library where you can.
   Incentivo à exploração do sistema
   Uma das funcionalidades que bibliotecas de análise (*parsing*) de argumentos, a autocompleção, auxilia na lembrança de nomes de comandos.

2. (1/11) Help

2.10. If the user did something wrong and you can guess what they meant, suggest it.

Incentivo à exploração do sistema

Sugerir a opção correta após um erro ser cometido pelo usuário contribui para que este não necessariamente precise memorizar opções de certo comando tal como descritas na documentação. Ao ser apenas sugerida a opção correta, mas não executada, o usuário terá também oportunidade de aprender com seu erro.

4. (1/17) Output

4.8. Suggest commands the user should run.

Incentivo à exploração do sistema

Sugestão de comandos potencialmente pode ajudar no reconhecimento de funcionalidades.

6. (1/14) Arguments

6.2. Have full-length versions of all flags.

Reconhecimento de opções disponíveis

É mais fácil reconhecer a funcionalidade de determinada *flag* quando é utilizado o nome completo dela.

14. (1/4) Naming

14.1. Make it a simple, memorable word.

Reconhecimento de opções disponíveis

Palavras fáceis de memorizar e idealmente relacionadas à funcionalidade de um programa auxiliam no reconhecimento desta.

VII. (5/99) Flexibilidade e eficiência de uso

6. (1/14) Arguments and flags

6.9. Never *require* a prompt.

Facilitamento da composição de scripts

Permitir a inserção de informação por meio de *flags* contribui para a facilidade de composição de *scripts*.

7. (1/4) Interactivity

7.1. Only use prompts or interactive elements if `stdin` is an interactive terminal (a TTY).

Facilitamento da composição de scripts

O emprego obrigatório de prompts ou elementos interativos potencialmente atrapalha na execução de *scripts*.

10. (4/10) Future-proofing

10.1. Keep changes additive where you can.

Facilitamento da composição de scripts

Manter o comportamento de flags existentes contribui para que *scripts* antigos continuem funcionando como esperado.

10.2. Warn before you make a non-additive change.

Facilitamento da composição de scripts

Avisar que flags ou outras funcionalidades antigas serão alteradas contribui para que usuários saibam que devem fazer as devidas alterações em *scripts* existentes.

10.5. Don't allow arbitrary abbreviations of subcommands.

Facilitamento da composição de scripts

Abreviações explícitas contribuem para a composição de *scripts*.

VIII. (4/99) Projeto estético e minimalista

2. (2/11) Help

2.2. Display concise help text by default.

Concisão de informações

Ajuda concisa por padrão pode ajudar a focar nas funcionalidades essenciais do programa.

2.7. If you've got loads of examples, put them somewhere else.

Concisão de informações

Exibir primeiramente apenas exemplos essenciais pode ajudar o usuário a focar nas funcionalidades principais.

4. (2/17) Output

4.5. Display output on success, but keep it brief.

Concisão de informações

Manter saída breve, restrita ao essencial, pode ajudar o usuário a focar nas informações realmente relevantes.

4.11. Use color with intention.

Uso adequado de cores

Uso de cor não planejado pode atrapalhar no foco em informações essenciais.

IX. (10/99) Reconhecimento, diagnóstico e recuperação de erros

1. (2/4) O básico

1.2. Return zero exit code on success, non-zero on failure.

Reconhecimento de erros

Identificar se o programa foi executado com sucesso ou não deixa claro se será necessário o tratamento de erros.

1.4. Send messaging to `stderr`.

Reconhecimento de erros

Receber saída através do fluxo adequado pode ajudar no diagnóstico de erros.

2. (1/11) Ajuda

2.10. If the user did something wrong and you can guess what they meant, suggest it.

Recuperação de erros

A sugestão do comando correto é uma maneira de recuperação de uma ação errônea, no caso a execução de um comando errado.

4. (2/17) Output

4.1. Human-readable output is paramount.

Reconhecimento de erros

Saída legível potencialmente pode ajudar no diagnóstico de erros.

4.17. Use a pager (e.g. `less`) if you are outputting a lot of text.

Reconhecimento de erros

A melhor navegação dos *logs* proporcionada por paginadores pode ajudar no diagnóstico e prevenção de erros.

5. (4/5) Errors

5.1. Catch errors and rewrite them for humans.

Reconhecimento de erros

Mensagens de erro legíveis por humanos auxiliam no diagnóstico e tratamento de erros.

5.2. Signal-to-noise ratio is crucial.

Reconhecimento de erros

Agrupamento de erros do mesmo tipo pode auxiliar na identificação e diagnóstico deles.

5.3. Consider where the user will look first.

Reconhecimento de erros

Destaque adequado de mensagens de erro contribui para a identificação e diagnóstico destes.

5.4. If there is an unexpected or unexplainable error, provide debug and traceback information, and instructions on how to submit a bug.

Recuperação de erros

Fornecimento de formas de rastrear a origem de erros contribui para a identificação e diagnóstico destes.

9. (1/8) Robustness

9.6. Make it recoverable.

Recuperação de erros

Conseguir facilmente retornar para o estado em que o programa estava antes de falhar contribui para a recuperação de erros.

X. (11/99) Ajuda e documentação

1. (1/4) O básico

1.1. Use a command-line argument parsing library where you can.

Disponibilização de ajuda e documentação

Bibliotecas desse tipo podem fornecer tratamento de texto de ajuda.

2. (7/11) Help

2.1. Display extensive help text when asked.

Disponibilização de ajuda e documentação

Ajuda extensiva contribui para a documentação dos usos do programa.

2.5. In help text, link to the web version of the documentation.

Disponibilização de ajuda e documentação

Versão web da documentação facilita a navegação nela.

2.6. Lead with examples.

2.7. If you've got loads of examples, put them somewhere else.

Disponibilização de exemplos de uso

Documentação com exemplos potencialmente pode ajudar no aprendizado de certo programa.

2.8. Display the most common flags and commands at the start of the help text.

Disponibilização de exemplos de uso

Mostrar ajuda relacionada a funcionalidades comuns no início da documentação pode ajudar usuários a encontrar rapidamente o que provavelmente irão precisar.

2.9. Use formatting in your help text.

Clareza do texto de ajuda

Formatação adequada do texto contribui para que usuários naveguem com mais facilidade na documentação de um programa.

2.10. If the user did something wrong and you can guess what they meant, suggest it.

Ajuda contextual

Sugestões podem após erro podem servir como ajuda contextual.

2.11. If your command is expecting to have something piped to it and `stdin` is an interactive terminal, display help immediately and quit.
Ajuda contextual
Exibir a documentação imediatamente após um uso inadequado de um programa contribui para que o usuário aprenda como usar ele da forma correta.

3. (3/3) Documentation
   3.1. Provide web-based documentation.
   Disponibilização de ajuda e documentação
   Documentação, web além de ser encontrável por motores de busca, tende a ser mais facilmente navegável.
   3.2. Provide terminal-based documentation.
   Disponibilização de ajuda e documentação
   Documentação diretamente no terminal é acessível de forma mais rápida do que, e.g., páginas `man` ou documentações on-line.
   3.3. Consider providing man pages.
   Disponibilização de ajuda e documentação
   Fornecimento de páginas `man` faz com que a documentação de um programa seja acessível localmente.