http://linuxbackgrounds.blogspot.pt/2010/12/linux-freedom-wallpaper.html

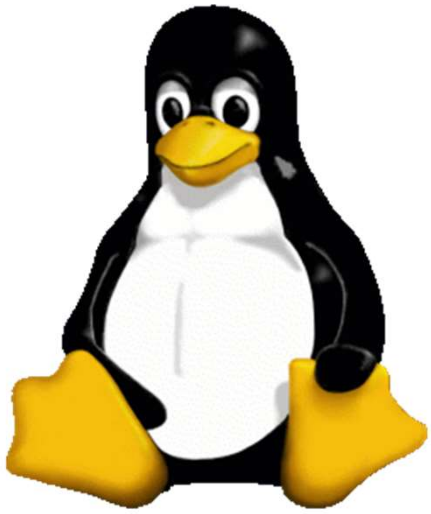
Linux

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Laboratórios de Informática

O que é?

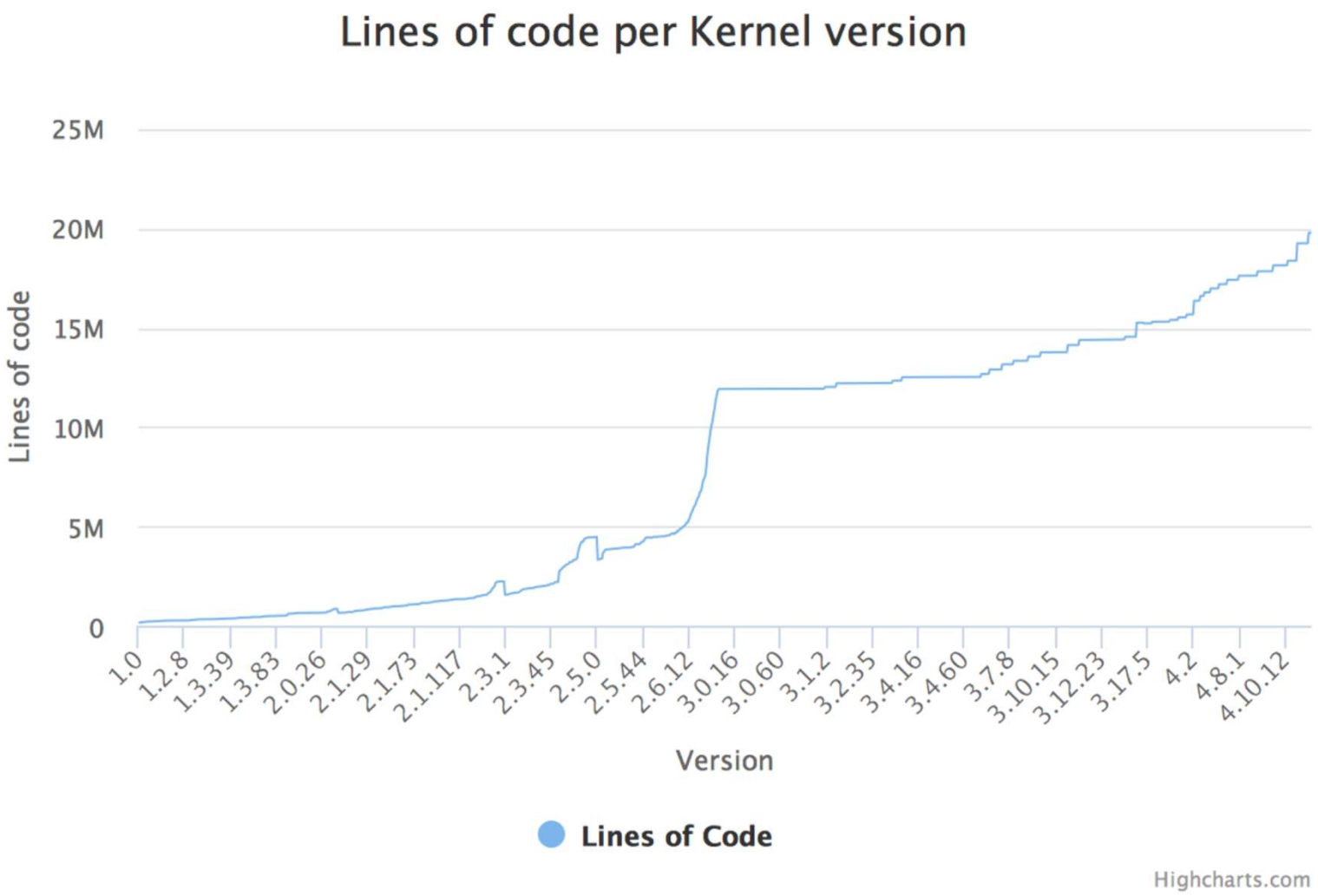
- Linux é um Sistema Operativo
- Entre outros como o Windows, macOS, iOS…
  - Mais: http://en.wikipedia.org/wiki/List_of_operating_systems
- Tem como mascote o TUX



Aparecimento

- Adoptado pela comunidade Open Source
  - É Software Livre
- Desenvolvido de forma distribuída
  - Qualquer um pode contribuir
  - Linus Torvalds mantém controlo sobre novas versões
- Herda conceitos de funcionamento do UNIX
  - Tal como macOS

Evolução



https://www.linuxcounter.net/statistics/kernel

Onde existe?

- Modelo em que código fonte é livre
  - Todos podem usar, alterar
  - Significa que pode existir em qualquer dispositivo

- No mercado [1][2]
  - desktop/laptop: ~1.46%
  - Mobile: 87.8%
  - Servidores: >50%

- Grande utilização nas universidades, datacenters
  - Permite aprender, adaptar, evoluir
  - Sem custos
  - Baixo consumo de recursos

1) http://www.gartner.com/newsroom/id/3516317
2) https://w3techs.com/technologies/details/os-unix/all/all

Quem já usou Linux fora da UA?

TODOS !

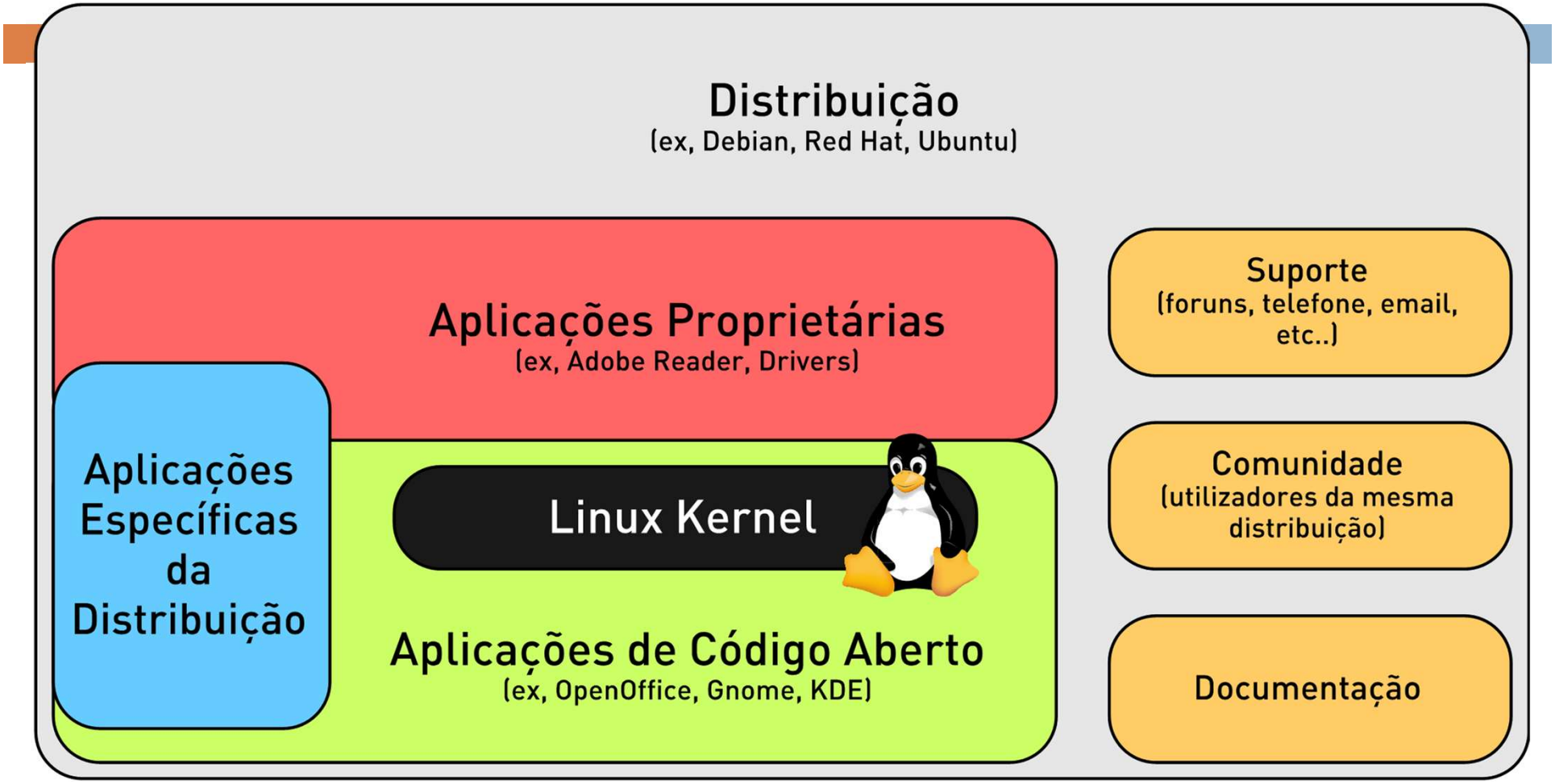
Linux Kernel

- Dispositivos diversos: Tomtom GPS, Media Players, Smart TV, Automóveis, …
- Equipamentos de rede: Routers Wifi
- Smartphones: **Android**, Bada, Tizen, Meego
- Servidores Web: 98.3% do top 1Milhão de sites [1]
  - Outros: Google, Facebook
- Supercomputadores: cerca de 95% [2]

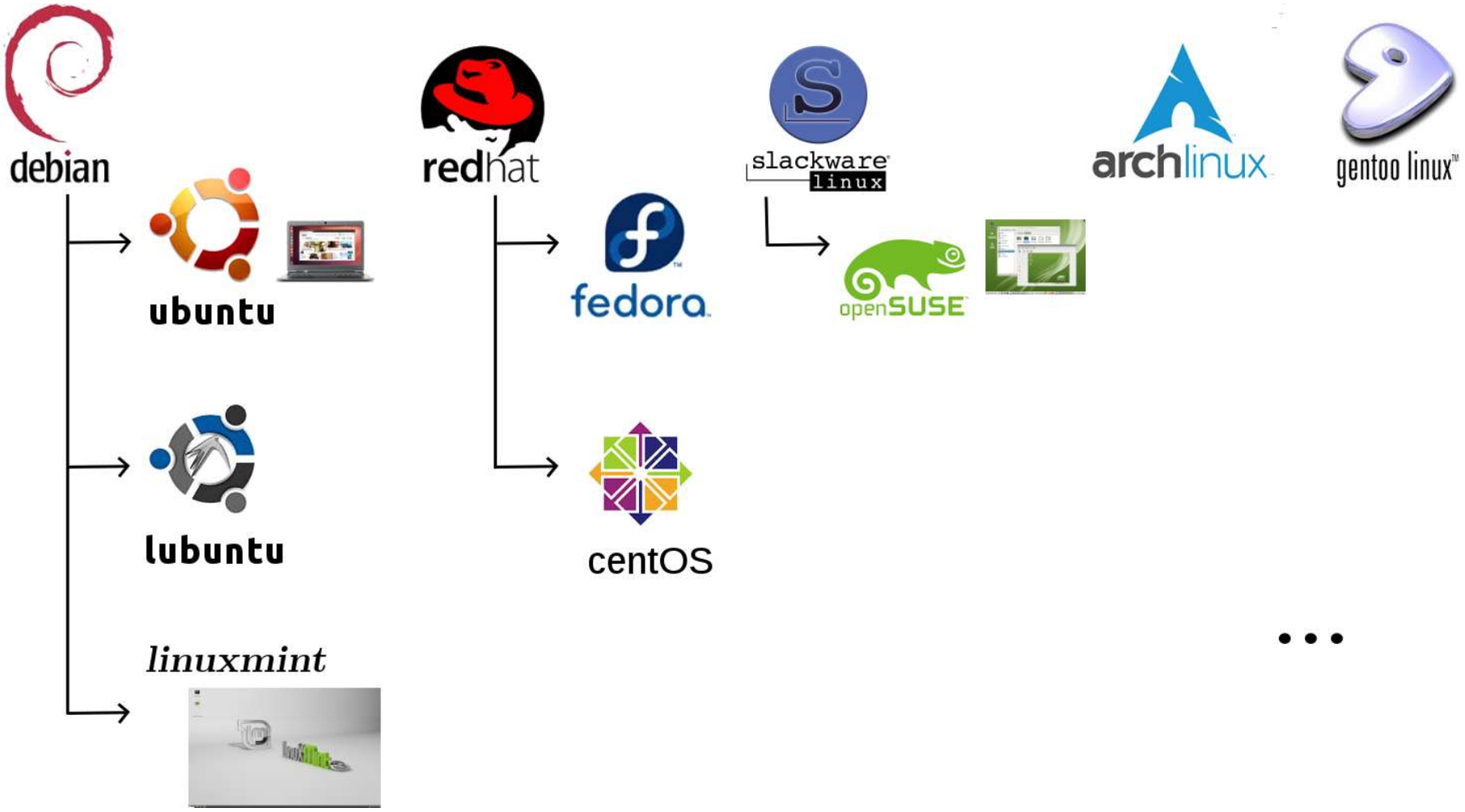
(2) http://www.w3cook.com (2) http://www.top500.org

Linux vs Distribuições

- Linux é instalado em equipamentos
- Ou fornecido em distribuições
  - Ex: Debian, Ubuntu, Caixa Mágica, etc..
- + de 470 distribuições [1]
  - Originais: Debian, Slackware, Red-Hat
    - Uma distribuição não é o Linux!


Distribuição
(ex, Debian, Red Hat, Ubuntu)
Aplicações Proprietárias
(ex, Adobe Reader, Drivers)
Aplicações Específicas da Distribuição
Linux Kernel
Aplicações de Código Aberto
(ex, OpenOffice, Gnome, KDE)
Suporte
(foruns, telefone, email, etc..)
Comunidade
(utilizadores da mesma distribuição)
Documentação

Distribuições



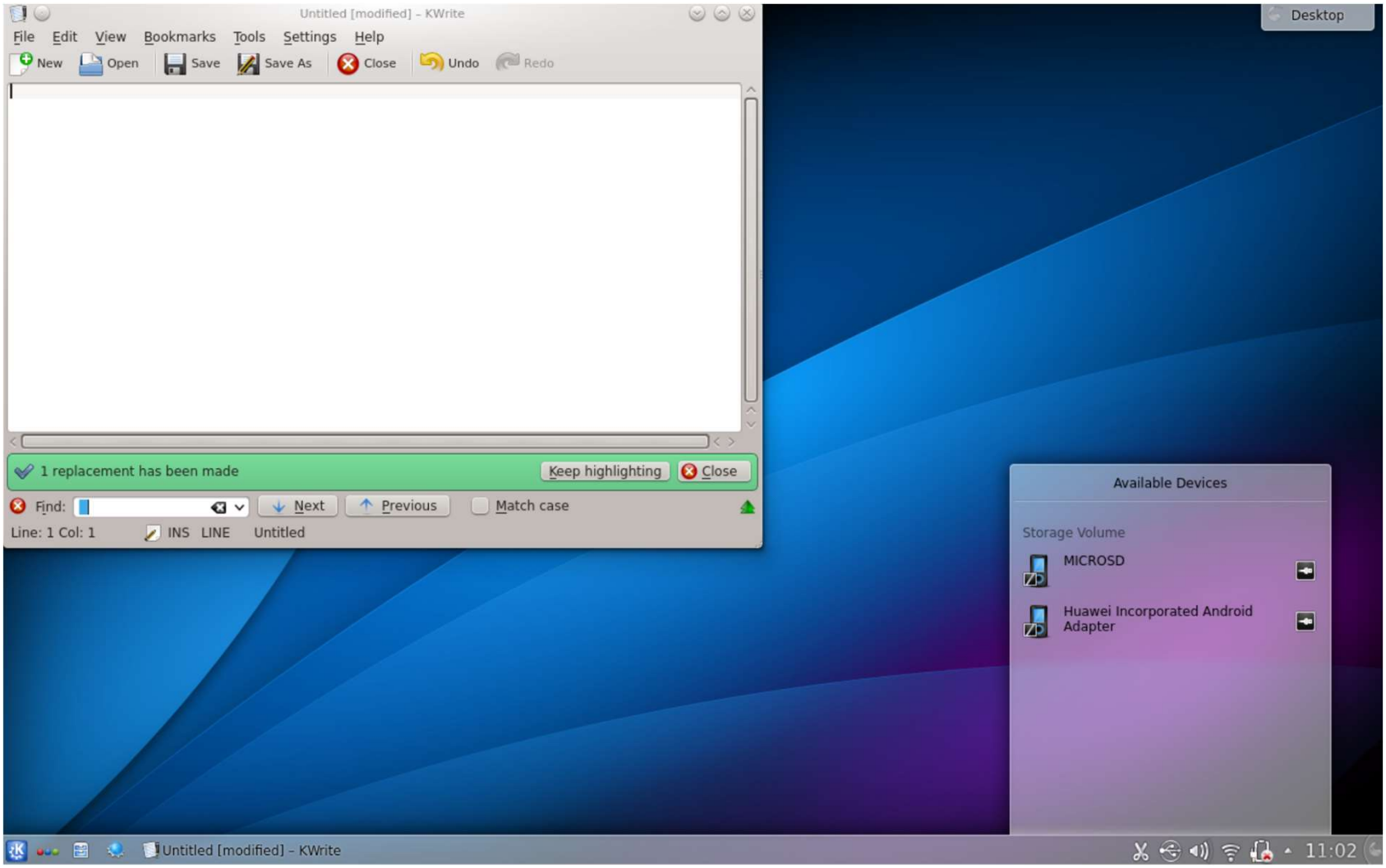
Interface Gráfico

- Semelhante ao presente noutros sistemas operativos
  - Point and Click suportado por janelas
- Diferentes ambientes gráficos:
  - Enlightenment, GNOME, XFCE, KDE, LXDE
  - Com diferentes aplicações em cada um

GNOME
(menu actividades ao lado)

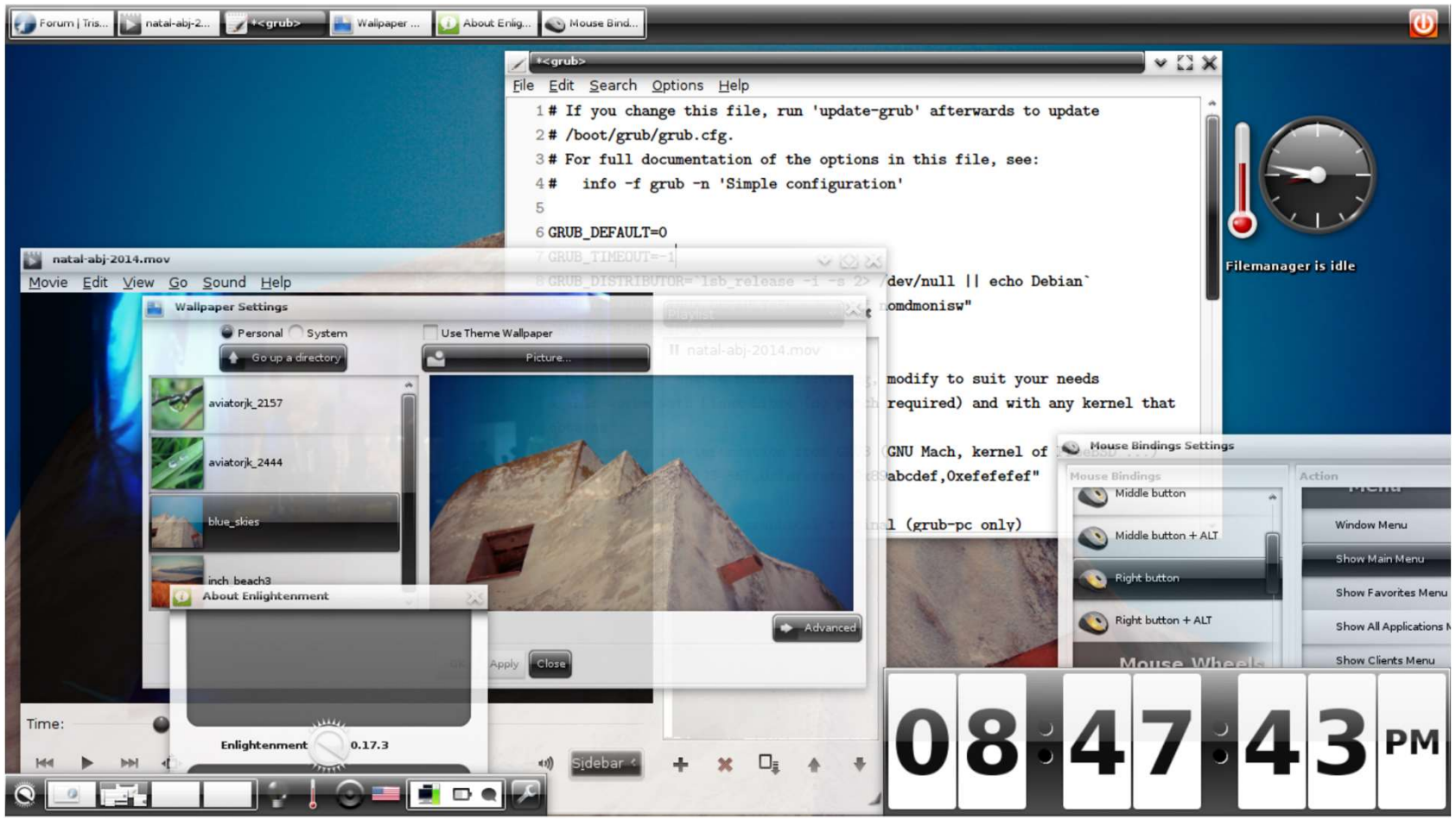
http://en.wikipedia.org/wiki/File:GNOME-Shell-3.10.png

KDE
(menu aplicações em baixo)



http://en.wikipedia.org/wiki/File:KDE_4.png

Enlightenment

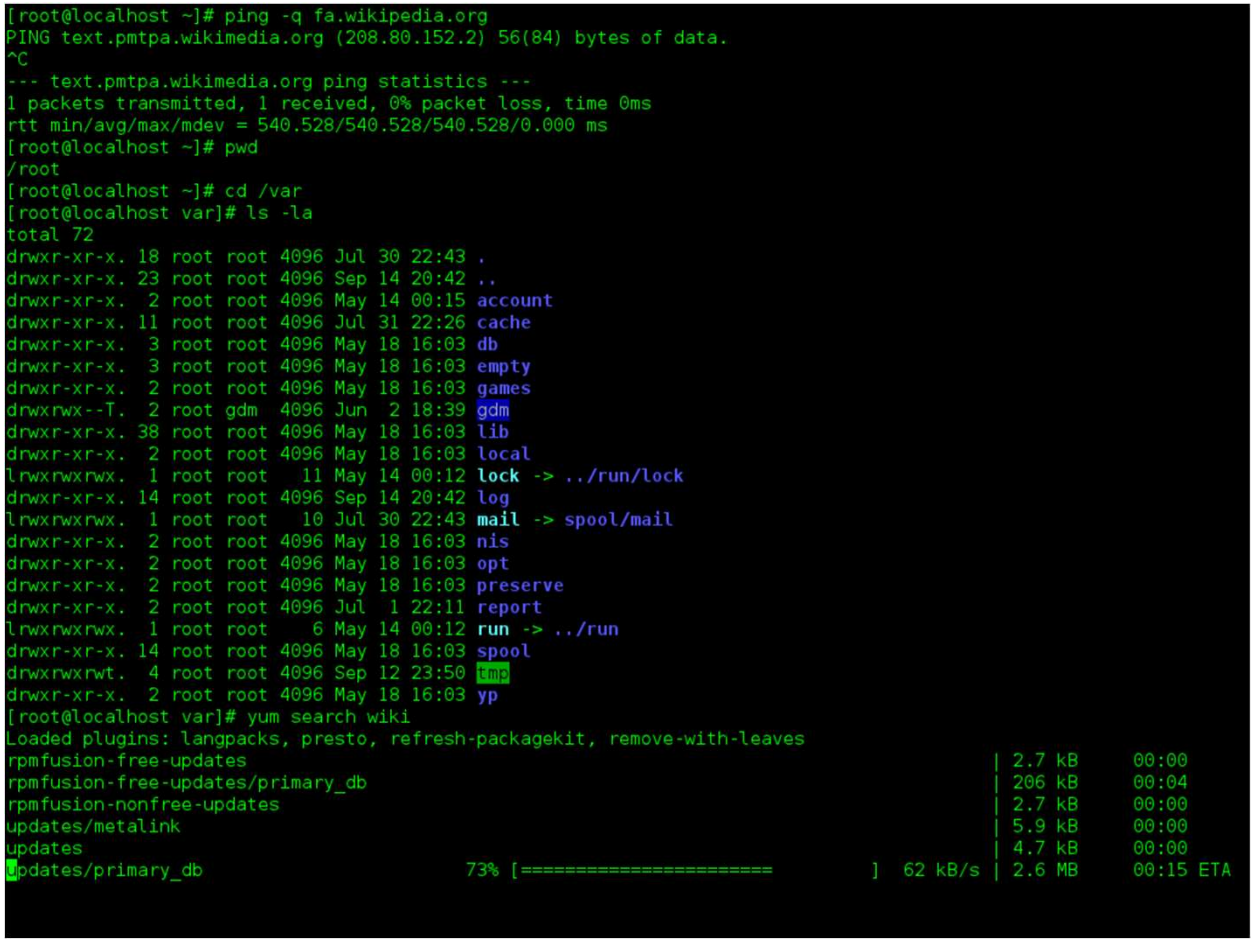


https://upload.wikimedia.org/wikipedia/commons/8/8c/Shot-2014-12-22_20-47-54.png

Interface de Texto

- Interface primordial
  - Pois não existia suporte gráfico
- Também chamado de Consola
  - Command Line Interface (CLI)
- Ainda muito utilizado
  - Power Users
  - **Administração e Desenvolvimento**
  - **Servidores**

Interface de Texto



By ZxxZxxZ (Own work) [GPL (http://www.gnu.org/licenses/gpl.html)], via Wikimedia Commons

Formato de um comando

- Um comando Linux é constituído por três partes
  - O primeiro identificador, a seguir ao *prompt* ($), indica o comando propriamente dito
  - Existem comandos que têm opções para configurar a sua execução. Para indicar a existência de opções, que são identificadas por caracteres minúsculos e maiúsculos, estas são precedidas pelo carácter -
  - A seguir ao comando ou às opções, caso elas existam, segue-se a lista de argumentos, que normalmente é constituída por um ou mais nomes de ficheiros, ou directorias, a ser processados pelo comando

$ *comando [- lista_de_opções] [lista_de_argumentos]* ↵

Interface de Texto

- Principais características da interface de texto
  - Interpretador de commandos (*shell*) versátil e poderoso
  - Descritores de ficheiros (*file descriptors*)
  - Redirecionamento de dados e de saída de dados
  - *Pipes* para encadear comandos
  - Execução de programas em *foreground* e em *background*

Shell

- A *shell* além de ser um interpretador de comandos, também é uma linguagem de programação, com instruções condicionais e repetitivas e variáveis

- Com estas instruções, usando comandos Unix, a comunicação entre processos (*pipe*) e os redirecionamentos de entrada e de saída de dados, é possível criar programas (*shell scripts*) de administração de sistema para automatizar tarefas

- Muito utilizado por programadores experientes e administradores de sistemas Unix

Shell

- A primeira *shell* do Unix (**sh**) foi inicialmente criada por Thompson e depois desenvolvida por Stephen Bourne, nos laboratórios AT&T. Ela é normalmente conhecida por Bourne *shell* e foi lançado em 1977 com o Unix Versão 7

- A C *shell* (**csh**) foi desenvolvido por Bill Joy na Universidade de Berkeley e é a shell mais utilizada nos sistemas BSD. Deriva originalmente da sexta edição do Unix, ou seja da Thompson *shell*

Shell

- Esta *shell* caracteriza-se por permitir uma maior interação do utilizador com o sistema, providenciando mecanismos que facilitam a trabalho por vezes repetitivo de desenvolvimento de software, tais como o mecanismo de história e o controlo de tarefas

- A Korn *shell* (**ksh**) foi desenvolvido por David Korn nos laboratórios AT&T – fusão das melhores características das *shells* já existentes

- A *shell* mais usada no sistema operativo Linux, é a Bourne-Again *shell* (**bash**) que é uma evolução retro-compatível muito mais interativa da Bourne *shell*

Descritores de ficheiros

- Quando um comando é posto em execução, tem associado três ficheiros representados por números inteiros que se designam por descritores de ficheiros
  - O descritor 0 representa o ficheiro de entrada (**stdin** na linguagem C – **System.in** na linguagem Java) que está associado com o dispositivo convencional de entrada, que é o teclado
  - O descritor 1 representa o ficheiro de saída (**stdout** na linguagem C – **System.out** na linguagem Java) que está associado com o dispositivo convencional de saída, que é o monitor
  - O descritor 2 representa o ficheiro de saída de erro (**stderr** na linguagem C – **System.err** na linguagem Java), que também está associado com o dispositivo convencional de saída

Redirecionamento de entrada

- Existem comandos cuja entrada é recebida pelo teclado. A *shell* também permite redireccionar a entrada de um comando do teclado para um ficheiro anteriormente criado, o que se designa por **redirecionamento de entrada**, utilizando o operador **<** da seguinte forma

$ *comando* < *ficheiro_de_entrada* ↵

Redirecionamento de saída

- Por defeito, o resultado de um comando é enviado para o monitor. No entanto, há a possibilidade de redireccionar esse resultado para um ficheiro para posterior análise, o que se designa por **redireccionamento de saída**, utilizando o operador **>** da seguinte forma

$$\$\ comando > ficheiro\_de\_saída \hookleftarrow$$

- Mas, o redireccionamento de saída, para um ficheiro já existente, faz com que o seu conteúdo seja apagado. Para juntar o resultado de um comando a um ficheiro já existente, deve utilizar-se em alternativa o operador de redireccionamento de saída **>>**

Redirecionamento de saída de erro

- Existem comandos que, para além de produzirem um resultado, produzem também resultados que não são entendidos como saída, mas sim como indicador da ocorrência de situações de erro

- A implementação do redireccionamento de saída de erro depende da *shell* e da intenção do utilizador de juntar ou não a saída de erro com a saída. No caso da Bourne *shell*, se pretendermos armazenar a saída de erro num ficheiro separado, deve proceder-se da seguinte forma

$ *comando [> ficheiro_de_saída] 2 > ficheiro_de_saída_de_erro* ↵

Redirecionamento de saída de erro

- Mas, se pretendermos armazenar a saída de erro no mesmo ficheiro do redireccionamento da saída, no caso da Bourne *shell* **sh**, deve proceder-se da seguinte forma

$$\$\ comando\ >ficheiro\_de\_saída\ \ 2>\&1\ \hookleftarrow$$

Encadeamento de comandos - *pipe*

- A shell também permite a comunicação entre comandos, possibilitando encaminhar a saída de um comando para a entrada de outro comando. Este mecanismo designa-se por *pipe*. Para isso utiliza-se o operador | da seguinte forma

$ *comando1* | *comando2* ↵

- Num *pipe*, o redireccionamento de entrada é feito no comando inicial e o redireccionamento de saída é feito no comando final da seguinte forma

$ *comando1* <*ficheiro_de_entrada* | ... | *comandoN* >*ficheiro_de_saída* ↵

Encadeamento de comandos - *pipe*

- Para forçar que a saída de erro de um comando atravesse o *pipe*, no caso da Bourne *shell* **sh**, utiliza-se a notação 2>&1 a seguir ao comando pretendido, da seguinte forma

$ *comando1* | *comando2* **2>&1** | **...** | *comandoN* ↵

Execução de programas

- Quando a *shell* é utilizada no modo interactivo, o utilizador tem de esperar que um comando termine para poder executar outro comando, o que se designa por execução no "primeiro plano" (***foreground***)

$ *comando* ↵

- Para comandos de longa duração, existe a possibilidade de os mandar executar sem ter de se esperar pela sua finalização, podendo assim continuar a executar outros comandos, o que se designa por execução nos "bastidores" (***background***), utilizando o operador **&** a finalizar o comando da seguinte forma

$ *comando* **&** ↵

Para Referência

- DistroWatch: http://www.distrowatch.com
- Linux Kernel: http://www.kernel.org
- GLUA: http://glua.ua.pt
- GNU/Linux Distribution Timeline: http://futurist.se/gldt/