

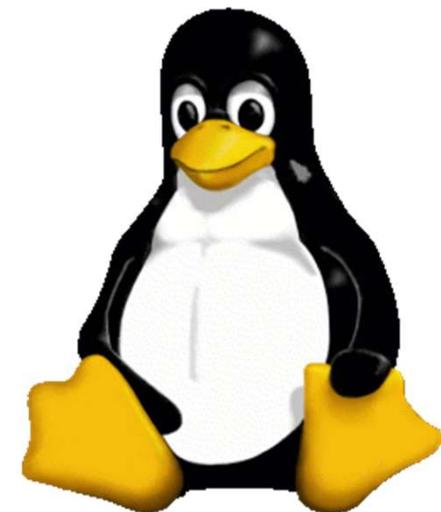


<http://linuxbackgrounds.blogspot.pt/2010/12/linux-freedom-wallpaper.html>

# Linux

# 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](http://en.wikipedia.org/wiki/List_of_operating_systems)
- Tem como mascote o TUX

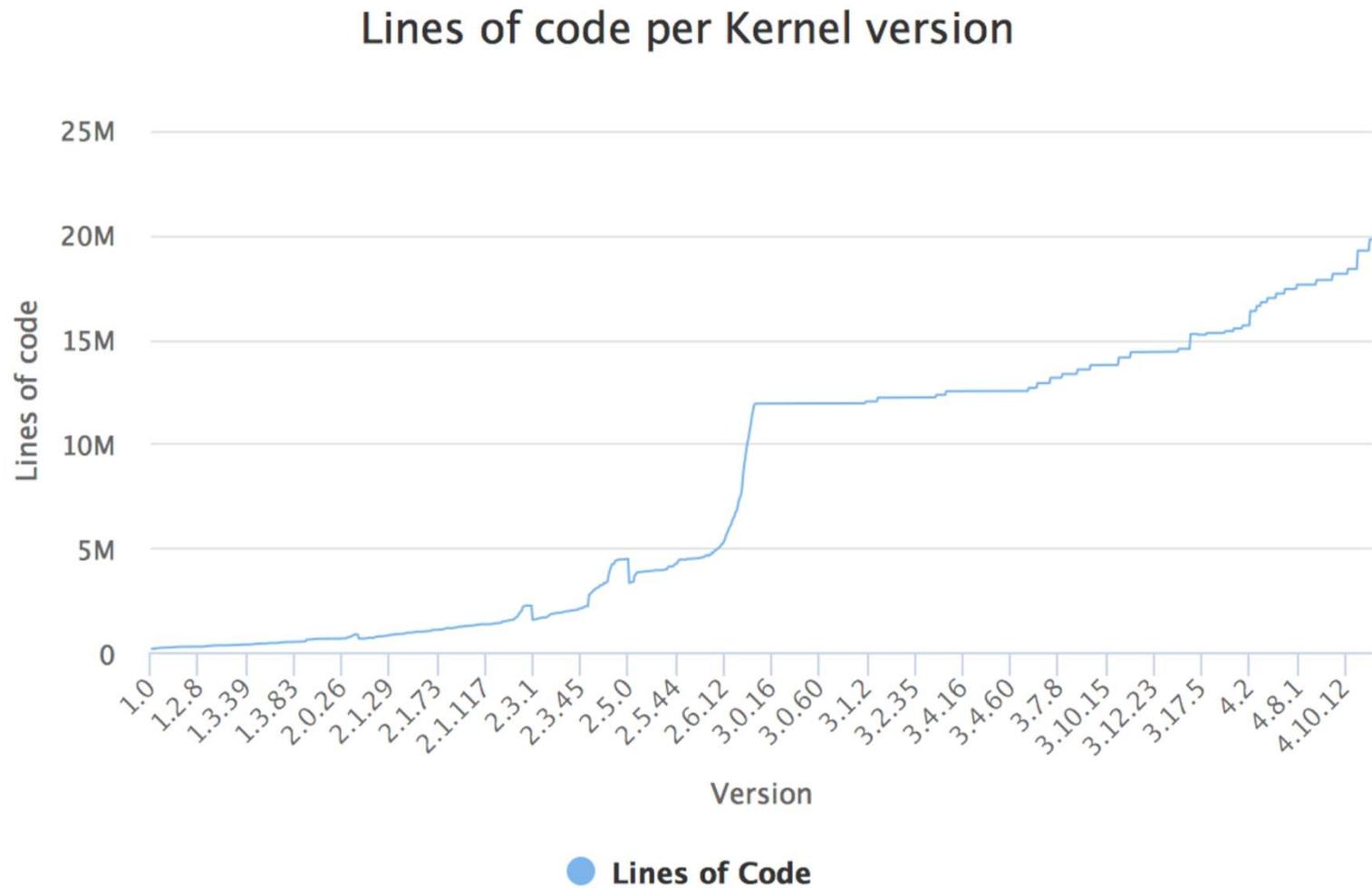


# Aparecimento



- 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



Highcharts.com

<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 <sup>(1)</sup>(<sup>2</sup>)
  - ▣ 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 1 Milhão de sites <sup>(1)</sup>
  - ▣ Outros: Google, Facebook
- Supercomputadores: cerca de 95% <sup>(2)</sup>

(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 <sup>(1)</sup>
  - ▣ 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

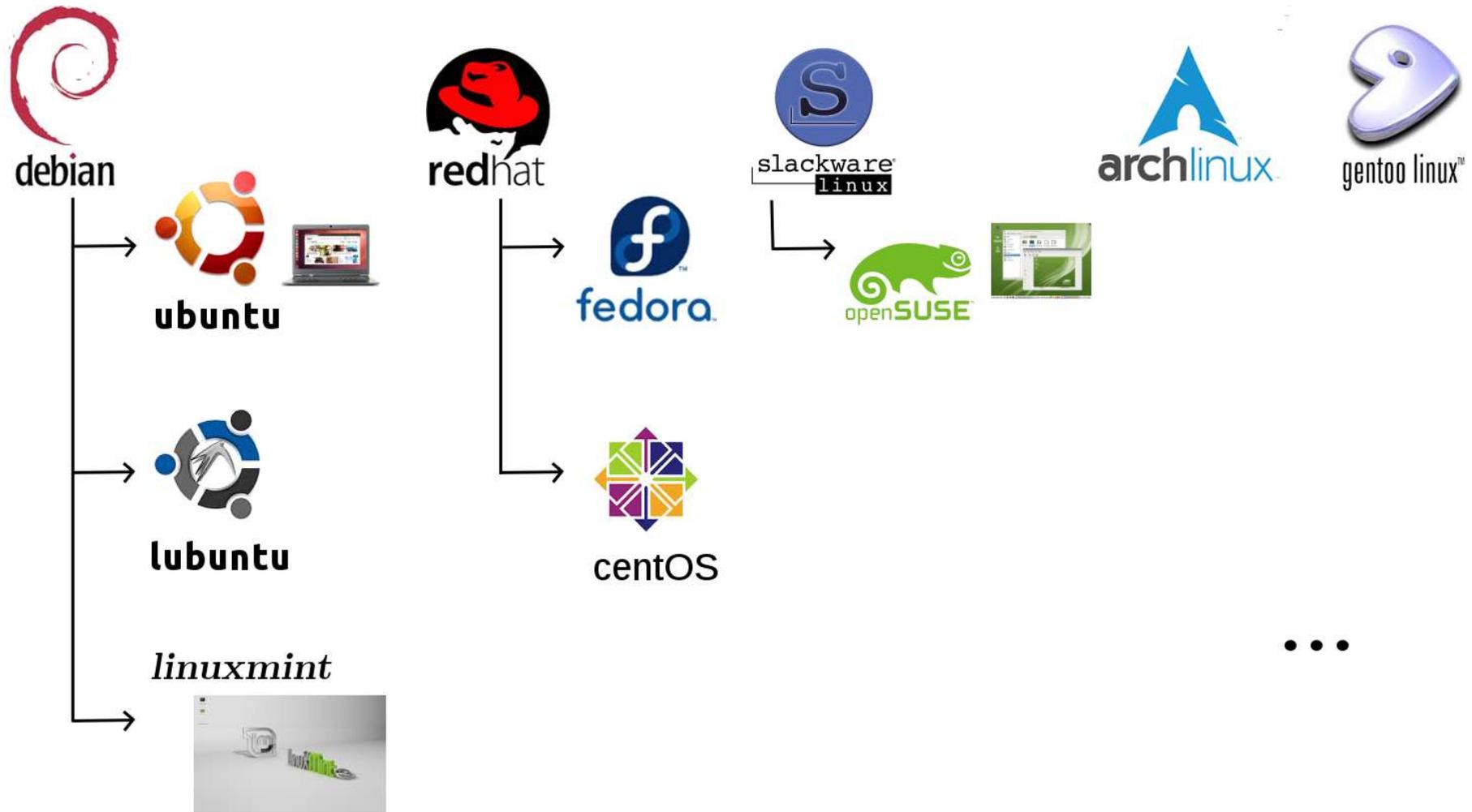
(foruns, telefone, email,  
etc..)

#### Comunidade

(utilizadores da mesma  
distribuição)

#### Documentação

# Distribuições



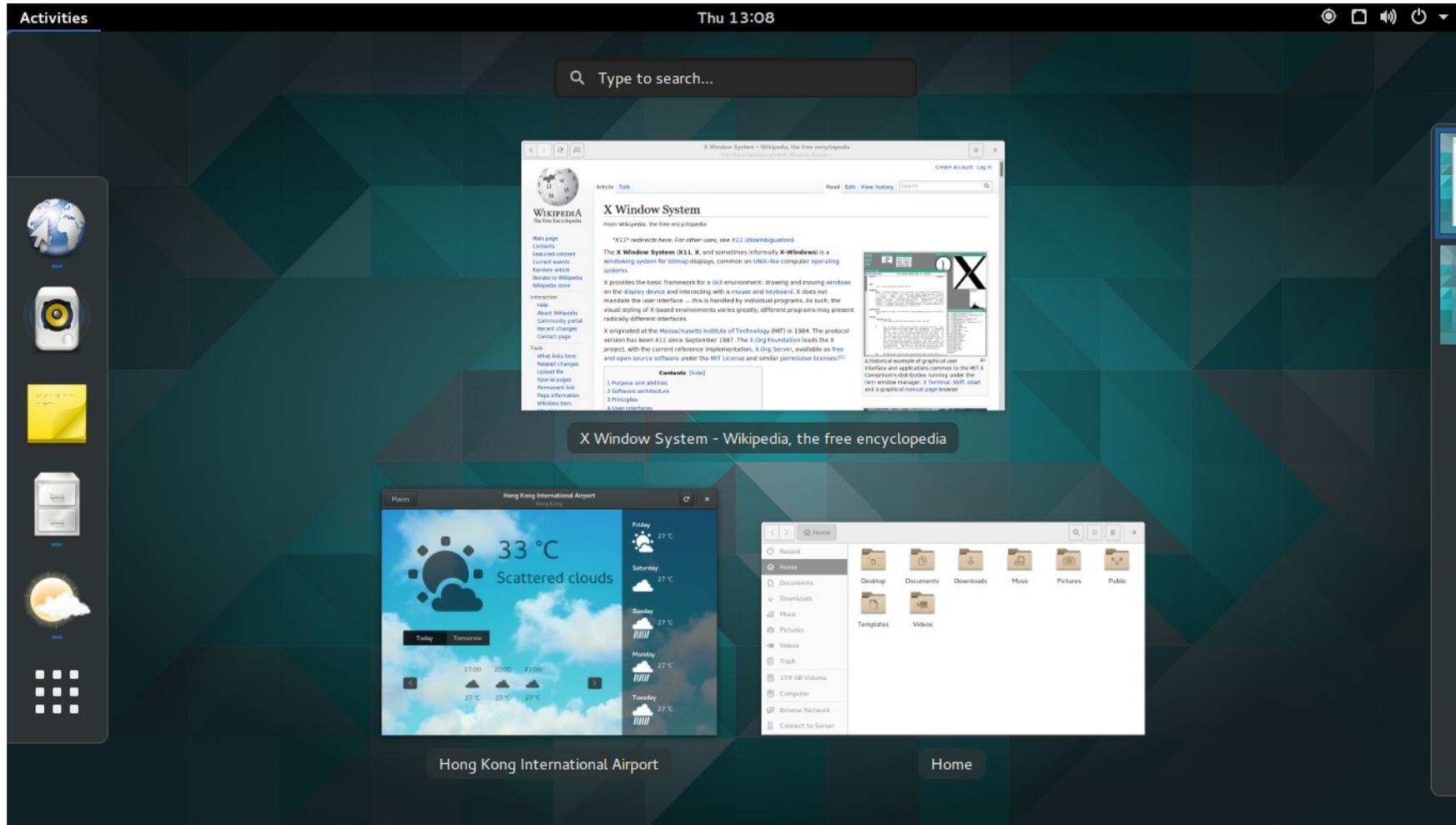
# Interface Gráfico



- 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

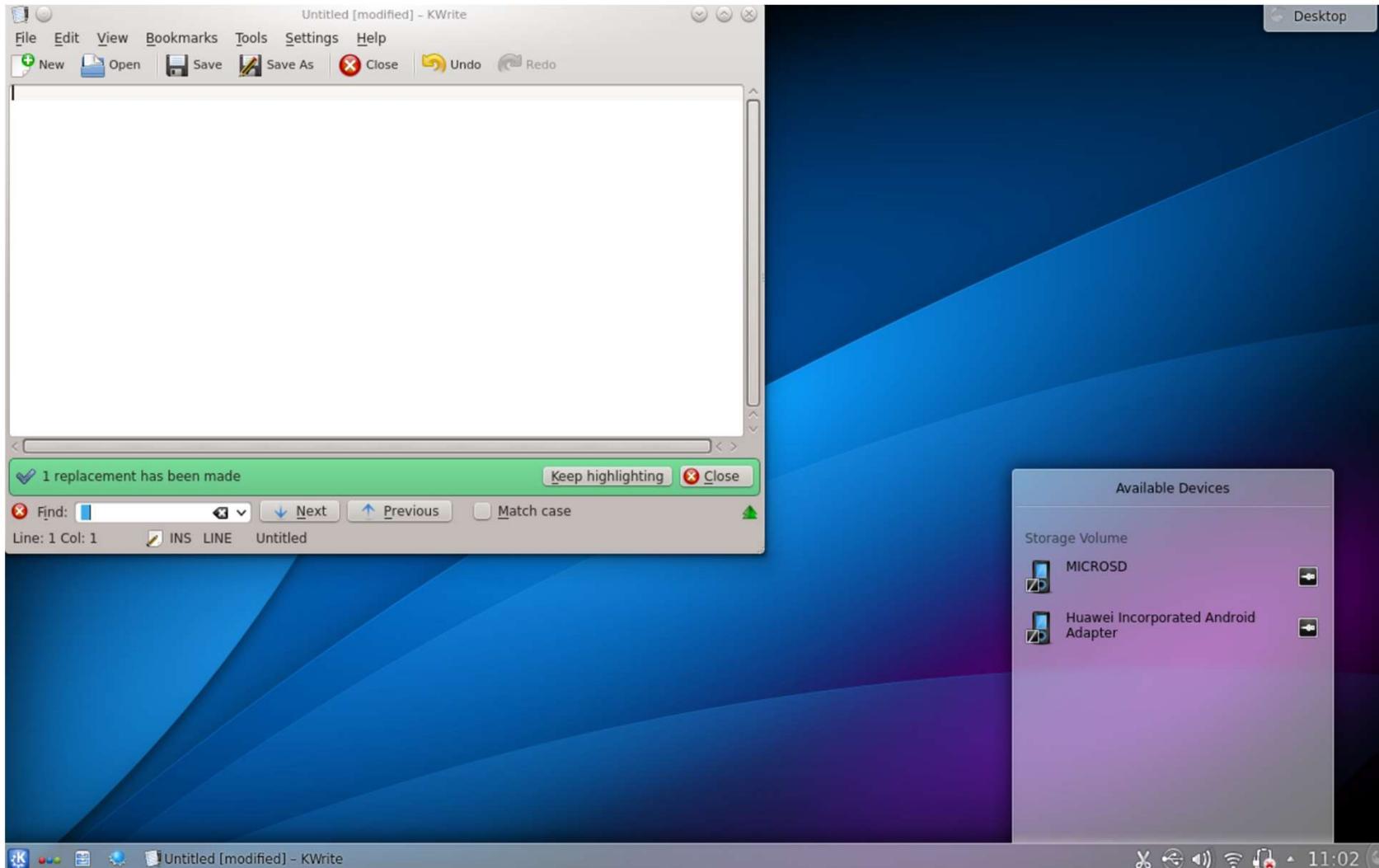
# GNOME

(menu actividades ao lado)

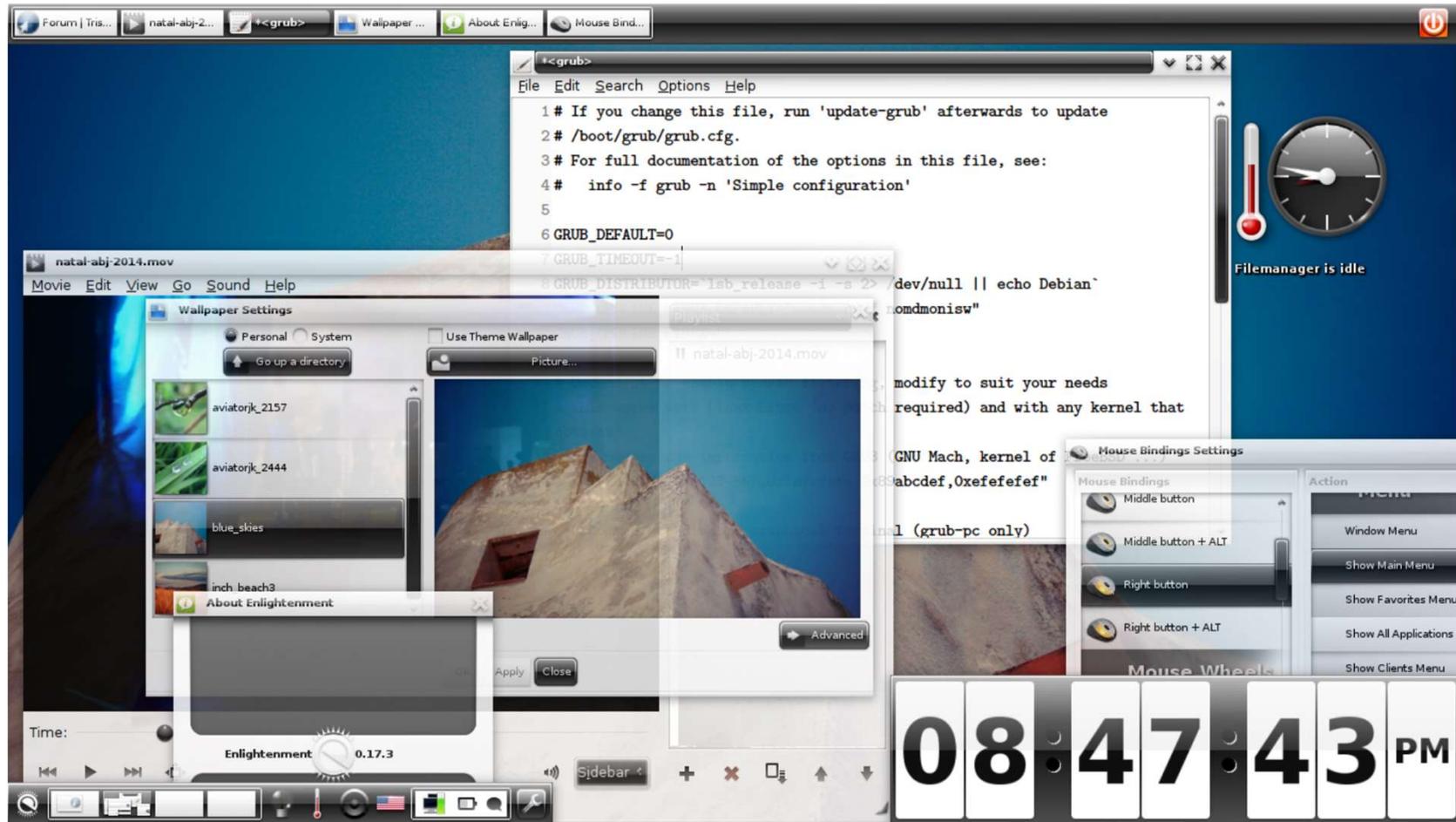


# KDE

(menu aplicações em baixo)



# Enlightenment



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

```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x.  2 root root 4096 May 14 00:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x.  3 root root 4096 May 18 16:03 db
drwxr-xr-x.  3 root root 4096 May 18 16:03 empty
drwxr-xr-x.  2 root root 4096 May 18 16:03 games
drwxrwx--T.  2 root gdm 4096 Jun  2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x.  2 root root 4096 May 18 16:03 local
lrwxrwxrwx.  1 root root  11 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx.  1 root root  10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x.  2 root root 4096 May 18 16:03 nis
drwxr-xr-x.  2 root root 4096 May 18 16:03 opt
drwxr-xr-x.  2 root root 4096 May 18 16:03 preserve
drwxr-xr-x.  2 root root 4096 Jul  1 22:11 report
lrwxrwxrwx.  1 root root   6 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt.  4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x.  2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                               | 2.7 kB      00:00
rpmfusion-free-updates/primary_db                    | 206 kB     00:04
rpmfusion-nonfree-updates                            | 2.7 kB      00:00
updates/metalink                                     | 5.9 kB      00:00
updates                                               | 4.7 kB      00:00
updates/primary_db                                   73% [=====] | 62 kB/s | 2.6 MB  00:15 ETA
```

# 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 comandos versátil e poderoso (*shell*)
  - ▣ Descritores de ficheiros (*file descriptors*)
  - ▣ Redirecionamento de entrada e/ou de saída de dados
  - ▣ Encadeamento de comandos (*pipe*)
  - ▣ 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* (**cs**) 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** no C – **System.in** no Java – **sys.stdin** no python) que está associado com o dispositivo convencional de entrada, que é o teclado
  - ▣ O descritor 1 representa o ficheiro de saída (**stdout** no C – **System.out** no Java – **sys.stdout** no python) que está associado com o dispositivo convencional de saída, que é o monitor
  - ▣ O descritor 2 representa o ficheiro de saída de erro (**stderr** no C – **System.err** no Java – **sys.stderr** no python), 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

$\$ \textit{comando} < \textit{ficheiro\_de\_entrada} \leftarrow$

# 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 **redirecionamento de saída**, utilizando o operador **>** da seguinte forma

*\$ comando > ficheiro\_de\_saída ↵*

□ Mas, o redirecionamento 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 redirecionamento 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 redirecionamento 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 redirecionamento da saída, no caso da Bourne *shell* **sh**, deve proceder-se da seguinte forma

*\$ comando > ficheiro\_de\_saída 2 > &1 ↵*

# 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/>