# Linux: Ferramentas de Programação
# 1 ª Semana de Informática

v1.1 — 25 de outubro, 2004


Camila Coelho
Linux User #215987

Reinaldo Sanches
Linux User #213980

# Sumário



# 1 Introdução

Esse documento foi desenvolvido especialmente para a 1ª Semana de Informática da Faculdade Comunitária de Campinas. Neste mini-curso temos como objetivo apresentar as principais ferramentas livres e IDEs de desenvolvimento disponíveis para Linux. O público-alvo inclui iniciantes em programação e interessados na utilização dos recursos básicos oferecidos pelas ferramentas de desenvolvimento open-source. Está fora do escopo deste curso o que diz respeito a práticas de programação ou linguagens, que serão utilizadas apenas para fins demonstrativos. O código fonte em LaTeX 2$_\varepsilon$ desse documento poderá ser obtido posteriormente, no link: http://linuxbr.org/.

# 2 Linha de comando

Uma das principais características de sistemas Un*x-like, como o Linux, é a disponibilidade de ferramentas poderosas na linha de comando (ou ***C****ommand* ***L****ine* ***I****nterface*). Essas facilidades são especialmente importantes para administradores de sistema e programadores.

As ferramentas GNU encontradas nas distribuições Linux apresentam funcionalidades muitas vezes indisponíveis em ferramentas gráficas. A flexibilidade da linha de comando, que permite a utilização de operadores lógicos (&& e ||), redirecionamento ('>', '»') e "pipe" ('|'), além dos recursos de programação nativos da "shell" — o interpretador de comandos, possibilita operações que seriam trabalhosas e demoradas em GUIs.Entre as ferramentas básicas estão: *ls, rm, more, wc, cat, od, cut, tail*, etc. No Slackware Linux, por exemplo, estes programas são integrantes do pacote "GNU Core Utils", que nada mais é que a composição dos pacotes GNU *fileutils*, *sh-utils* e *textutils*.

A shell, por sua vez, é a interface primária entre o usuário e o kernel, pois é através dela que o usuário pode interagir com o sistema, executando os comandos disponíveis na distribuição GNU/Linux. Em geral, a shell que acompanha estas distribuições é a *Bourne-again Shell*, mais conhecida por **bash**. Bash é uma versão aprimorada da shell tradicional em sistemas Unix, a "Bourne Shell" (ou somente "sh"), incluindo também recursos de outras shells como a Korn Shell (ksh) e a C-Shell (csh).

O bash apresenta diversos recursos para criação de scripts, chamados de **shell scripts**, que são desenvolvidos geralmente para automatizar processos ou facilitar operações que envolvam diferentes utilitários e parâme-

tros.

Para desenvolver scripts simples, os editores mais utilizados são o **vi(m)**, o **emacs** e o **nano**. Estes editores apresentam recursos de coloração de código (*syntax highlighting*) para scripts bash, entre outros.

## 2.1 O editor vi (M)

O editor **vi** é o editor tradicional de sistemas Unix, encontrado virtualmente em qualquer versão do sistema, de Solaris a HP-UX ou AIX. Esse fato explica sua grande popularidade. Seu uso mais comum é para a edição rápida de textos ou programas, pois o vi (modo linha de comando) é muito leve, podendo ser gravado em um disquete.

No Linux, é comum encontrarmos uma versão aperfeiçoada do tradicional vi, o **vim**: *Vi iMproved*. O vim apresenta basicamente três modos de operação: o modo de comandos (*default*), o modo de edição, e o modo **ex**. Ao iniciar o vim, ele encontra-se no primeiro modo, o que difere de outros editores que iniciam no que seria o modo de edição. Para iniciar a edição de um texto, geralmente chama-se o comando de inserção: a tecla '**i**'. Outros comandos úteis para o vi(m):

Tabela 1: Modo de comandos do vi

| | |
|---|---|
| a | insere o texto após a palavra selecionada (append) |
| x | deleta a palavra selecionada |
| dd | deleta a linha |
| u | desfaz a última modificação |
| v | modo de seleção |
| y | copia a área selecionada |
| p | cola a área selecionada |
| / | busca por uma string no texto |

Vamos criar um script chamado "numero.sh". Para isso, informe o nome do script como parâmetro do editor:

```
$ vim -N numero.sh
```

Utilizamos o parâmetro "-N" para desabilitar recursos de compatibilidade com o vi antigo. Entre em modo de inserção e digite:

```
#!/bin/bash
echo "hello world!"
```

O comando "echo" retorna uma mensagem na tela. Para gravar o script, entramos em *modo ex*. Para isso tecle ESC : (a tecla "ESC" seguida de dois pontos).

Na realidade, o modo "vi" é o modo visual do editor de linha "ex"[3]. Em modo "ex", podemos agora gravar nosso arquivo. Para isso, digite **w**.

Podemos ativar também opções do vi para facilitar o desenvolvimento, destacando-se (modo "ex"):

**set (no)number** : Ativa/Desativa a numeração de linhas.

**syntax on** : Ativa o modo "syntax highlighting".

**set autoindent(ai)** : Ativa a formatação automática do código.

**vi** : Retorna para o modo de comando padrão.

Em modo de comando também há comandos que podem auxiliar o programador, como o **%**, que retorna o parênteses, colchete, ou chaves correspondente ao caracter selecionado. Outro comando, especialmente relevante para programadores C é o **gd** (*Goto Declaration*[4]), que leva o cursor de volta à declaração da variável selecionada.

Em modo de edição, um atalho importante é a combinação **CTRL-P**, que procura uma palavra semelhante e completa a sentença em questão.

Ao criar ou abrir um arquivo de edição no vi, ele na realidade gera um “buffer”, e a edição ocorre neste “buffer”. Alguns comandos importantes para trabalhar com os “buffers” no vi:

**edit** : Inicia a edição em um outro buffer.

**write** : Grava o buffer com um novo nome e retorna para o buffer atual.

**ls** : Lista os buffers.

**buffer #** : Altera de buffer de edição.

**q** : Encerra o buffer atual.

**x** : Grava os buffers e sai do editor.

Finalmente, para sair do editor, digite **q**. Como apenas temos um “buffer” ativo, o vi encerrará.

Comandos em modo “ex” devem ser precedidos por “:” e podem ser concatenados. Por exemplo, a sequência “wq” grava e encerra o editor. O comando “x” faz o mesmo que esta combinação. Um comando comum também é o “q!”, que encerra o editor sem gravar os “buffers” modificados. O “!” omite os avisos do vi.

Para rodar o script que criamos, é preciso indicar que o script é “executável”, e em seguida, o caminho para a execução:

```
$ chmod +x numero.sh
$ ./numero.sh
```

A mensagem “hello world” deverá aparecer na tela, indicando o sucesso da execução do script. Para confirmar isso, a variável do bash “$?” armazena o código de retorno da última execução (o código **0** equivale ao sucesso da operação).

Para que o vi mantenha as configurações efetuadas, crie um arquivo **.vimrc**, manualmente, ou com o comando **:mkv**, e adicione as opções (*set, syntax, etc.*) desejadas.

## 2.2 Bash — shell scripts

O bash disponibiliza diversos componentes e estruturas comuns em linguagens de programação para o desenvolvimento de scripts para o atendimento de diversas necessidades. Abaixo, exemplificamos os principais recursos de um shell script:

1. Interatividade

   Para aguardar um “input” do usuário, utiliza-se o comando “read”. Veja um exemplo abaixo:

```
#!/bin/bash
#
# numero.sh: Meu primeiro script.

echo "Digite um número:"
read var
echo "O número é:" \$var
```

Listing 1: Shell Script

Um exemplo comum de “shell script” recebe parâmetros de entrada e executa ações baseadas nestes comandos. Nosso script também pode receber parâmetros, como no exemplo:

```
#!/bin/bash
# numero.sh: soma o número informado por 2.

expr $1 + 2
```

A variável de ambiente “$1” refere-se ao primeiro parâmetro, “$2” ao segundo, e assim por diante. Nesse caso, ao executar o script, devemos informar logo em seguida o parâmetro, no caso, o número a ser somado:

```
$ ./numero.sh 15
```

Para referências de outras variáveis úteis do Bash, a melhor fonte é o manual (*manpage*):

```
$ man bash
```

A seção *PARAMETER* da *manpage* do bash apresenta informações completas a respeito de tratamento de parâmetros e variáveis de ambiente.

2. Condições *If/Else/Case*

   No exemplo anterior, caso o parâmetro seja omitido, o resultado será simplesmente 2. Porém, muitas vezes é necessário determinar se o parâmetro foi informado para efetuarmos a operação. Para verificar se um parâmetro foi realmente informado, utilizamos a expressão “if”:

```
#!/bin/bash
  if [ $# -ne 1 ]
  then
     echo "Utilização: ./numero.sh NUMERO"
  else
     expr $1 + 2
  fi
```

   Desta forma, alertamos sobre a utilização correta do script **se** a quantidade de parâmetros ($#) não for igual (parâmetro “ne”, ou *not equal*) a “1”. Outras expressões condicionais podem ser encontradas na seção “CONDITIONAL EXPRESSIONS” da *manpage* do **bash**.

   Como pode ser observado, nosso programa não verifica se a entrada é um caracter ou um número. No Bash, por padrão, não há diferenciação de tipos. Uma forma de se verificar se o argumento é numérico seria através de uma expressão regular, como no exemplo abaixo:

```
#!/bin/bash
case "$1" in
     [0-9]*)
          expr $1 + 2 ;;
     -h)
          echo "Utilização: ./numero.sh NUMERO";;
     *)
          echo "Parâmetro não é um número." ;;
esac
```

   Dessa forma, se o valor for numérico, efetua-se a soma. Se o parâmetro for a string “-h”, retorna a ajuda de utilização. Para todos os outros, a mensagem “Parâmetro não é um número” é exibida.

3. Loops “While” e “For”

   O Bash também possui comandos para *loops*, ou laços condicionais. O “for” por exemplo, é geralmente utilizado para a iteração entre palavras ou items, como pode ser observado a seguir:

```
for i in ‘seq 1 10‘;
do
   echo $i
done
```

O “while” é semelhante a outras linguagens, considerando as expressões condicionais do Bash (*not equal, lesser than, etc.*): [3]

```
COUNTER=0
while [  $COUNTER -lt 10 ]; do
    echo "O contador é: " $COUNTER
    let COUNTER=COUNTER+1
done
```

Além de shell scripts, há diversas *linguagens script* que podem ser úteis para efetuar tarefas no sistema. Entre as mais comuns estão o Perl, o AWK, e a linguagem Tcl/Tk.

Para identificar um script como relativo a uma shell (Bash, Korn) ou a alguma linguagem de script, utiliza-se a notação “#!”, seguido do caminho do interpretador. Veja um exemplo de script Tcl/Tk, e note a especificação do caminho para o interpretador correto, no caso, o **wish** (adaptado de [7]):

```
#!/usr/bin/wish

set apachectl "/usr/local/apache/bin/apachectl"

proc start {} {
  global apachectl
  exec \$apachectl start &
}

proc stop {} {
  global apachectl
  exec \$apachectl stop &
}

proc screen {} {
  frame .top -borderwidth 10
  pack .top -fill x
  button .top.start -text "Start Apache" -command start
  button .top.stop -text "Stop Apache" -command stop
  pack .top.start .top.stop -side left -padx 0p -pady 0
}
screen
```

Listing 2: Tcl Script

Este script apresenta uma forma conveniente de se iniciar ou restartar o Apache HTTP Server, usando a biblioteca gráfica “tk”.

Particularmente, scripts desenvolvidos em shell possuem uma grande flexibilidade, permitindo a execução simultânea de programas das mais diferentes formas, graças à união dos recursos do **bash** e o poder das ferramentas GNU.

# 3 IDEs Open Source

Uma IDE (*Integrated Development Environment*) caracteriza-se por um conjunto de ferramentas de programação (editor, compilador, debug, etc.) integradas em um único ambiente. Nesta seção abordaremos o editor **GNU/Emacs**, o framework de desenvolvimento **Eclipse** e, finalmente, a IDE/RAD do ambiente KDE, o **KDevelop**.

## 3.1 O editor Emacs

No Linux, é comum encontrarmos diversos editores que possuam facilidades para trabalhar com linguagens de programação. A presença do GCC em praticamente todas as distribuições leva o Linux a ser um sistema de especial interesse para desenvolvedores. Nesta seção abordaremos um editor especializado para programadores: o **Emacs**. O editor **GNU Emacs** foi desenvolvido por RMS e pela FSF (Figura 1), sendo o primeiro programa integrante do que seria o pacote GNU.


Figura 1: FSF Logo

O Emacs roda tanto em linha de comando quanto em modo gráfico. O modo gráfico é bastante intuitivo, mesmo para não programadores. É possível redimensionar “buffers”, alterar o “buffer” clicando na barra de status com o botão direito do mouse, selecionar, copiar e colar textos de forma visual.

Assim como o *vim*, o Emacs também trabalha com múltiplos “buffers”. Ao abrir o emacs sem nenhum parâmetro, ele iniciará o buffer de “Scratch” (rascunho - nunca é gravado) e de “Messages” (log de todas as mensagens do “mini-buffer”). O comando **list-buffer** (“C-x C-b”) lista todos os buffers disponíveis. Na própria listagem é possível finalizar um buffer, alternar de buffer, etc.

A última linha do buffer, logo abaixo da “linha de modo”, é chamada de **mini-buffer**, onde as mensagens do sistema são listadas, os comandos, etc. A linha de modo seria o equivalente à barra de status encontrada em muitos programas, que apresenta informações a respeito do nome do arquivo, número da linha, o “major mode” ativo, entre outras. Para sair do “mini-buffer” (cancelar um comando, por exemplo), digite C-g.

O comando “M-x” utiliza o mini-buffer para a execução de comandos e modos disponíveis no Emacs pelo nome (tecle “TAB” para obter uma listagem dos mesmos). Neste caso, “M” é a tecla “Meta”, muitas vezes representada pela tecla “Alt” ou “ESC”. No caso, “M-x” significa que para acessar o menu de comandos, deve-se teclar “Alt” seguido de “x”.

Para programadores, o Emacs apresenta recursos especiais, como os modos de execução chamados **major modes**, específicos para cada linguagem. Os modos incluem recursos de formatação e coloração de código e extensões para o próprio Emacs, incluindo novos menus e teclas de atalho para realizar as mais diferentes operações.

Por exemplo, ao se abrir um código C, o Emacs apresenta o menu:

`File Edit Options Buffers Tools` **C** `Help`

Ao abrir uma página em HTML, o menu passa a ser:

`File Edit Options Buffers Tools` **HTML SGML** `Help`

Da mesma forma, a linha de modo (status) é alterada mostrando este fato. Por exemplo, ao editar este texto no Emacs, em tex-mode, tenho as seguintes informações na minha barra de status (chamada de **linha de modo**):

```
--11:---F1  manual.tex     Wed Oct 20 10:15PM 0.20   (LaTeX Abbrev)--L384--C0--65
```

Além das opções no menu, cada modo possui atalhos específicos, isto é, algumas teclas assumem novos comportamentos. Por exemplo, em modo HTML a combinação “C-c /” fecha uma “tag”, e assim por diante.

Entre os vários recursos para programação disponíveis no Emacs, destacam-se os modos de compilação, debug, comparação de códigos e CVS. Em modo **compile**, o Emacs abrirá um novo buffer com as mensagens de compilação retornadas pelo compilador escolhido. Ao utilizar o “compile”, é possível navegar pelas linhas que causaram o erro através da combinação **C-‘** (crase) ou pelo comando **next-error**. [5]

Para chamar o modo “compile”, uma das formas é através da tecla **M-x compile**. Por *default*, o comando de compilação é o “make -k”, podendo ser substituido por qualquer outro no próprio “mini-buffer”, ou de forma definitiva, através da configuração do **.emacs**, que será vista mais adiante.

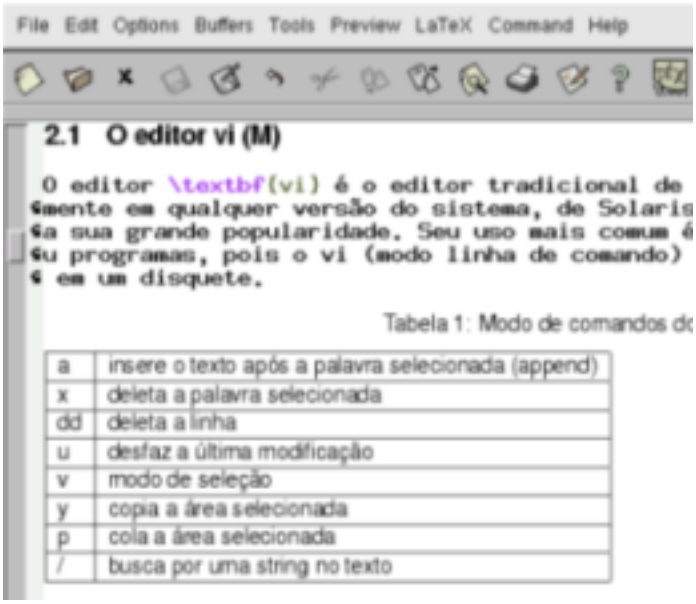


Figura 2: Emacs em modo AuCTeX

O modo “debug” pode ser acessado através do comando `(M-x) gdb`. É importante ressaltar que, no caso de um fonte em C, será preciso compilá-lo com o parâmetro **-g** do **gcc** para que sejam adicionados símbolos de debug específicos do gdb. No código fonte, adicione “breakpoints” com a sequência **C-x SPC**, onde “SPC” refere-se à tecla de espaço, e “C-x” às teclas “CTRL” e “x”.

De volta ao modo debug, defina o nome do programa a ser depurado com o comando “file” e execute o programa com o comando “run”. Para inspecionar o conteúdo de uma variável, por exemplo, o comando é “print”, seguido do nome da variável. Para executar o programa passo-a-passo, digite “n” (ou “next”), e assim por diante. A referência completa do **gdb** pode ser encontrada em sua respectiva “manpage”. Observe que, na medida em que o gdb executa um comando, o cursor no “buffer” do código fonte é posicionado na linha em questão. Isto demonstra a grande integração entre o gdb e o Emacs.

Outra ferramenta muito útil que acompanha o Emacs é o **ediff**. Através do comando “M-x ediff”, o Emacs compara 2 arquivos e apresenta de forma visual a diferença entre eles. Além do modo ediff, o Emacs também apresenta o modo “diff” tradicional, para a geração de patches, etc.

Outras opções do Emacs, especialmente relevantes para edição de código, podem ser ativadas no Menu de opções (“Options”), ou através do menu de comandos:

Tabela 2: Opções do Emacs

| | |
|---|---|
| Syntax highlighting | (global-font-lock-mode 1) |
| Habilita expansão automática de texto | (abbrev-mode 1) |
| Realce de parênteses correspondentes | (show-paren-mode 1) |
| Define comportamento a uma tecla (ex: f3) | (global-set-key [f3] ’eshell) |
| Quebra de linha | (auto-fill-mode 1) |

Estas configurações são gravadas no diretório $HOME do usuário, no arquivo **.emacs**, ao selecionar a opção “Save Options” do Menu. Os comandos da tabela correspondem a comandos de **elisp** (Emacs Lisp): a linguagem de desenvolvimento de extensões e do próprio arquivo de configuração do Emacs.

Para alterar o compilador padrão do modo **compile**, por exemplo, poderíamos adicionar o seguinte ao nosso `.emacs`:

```
(add-hook 'c-mode-hook
   (lambda ()
        (set (make-local-variable 'compile-command)
        (concat "gcc " buffer-file-name " -o "
        (file-name-sans-extension buffer-file-name))
)))
```

Essa função, em elisp, define o "gcc" como compilador padrão (compile-command) do modo C (c-mode-hook), e automaticamente se encarrega de preencher o nome do arquivo com o nome do buffer ativo (buffer-file-name).

Destacamos também algumas configurações (Tabela 2), entre elas a expansão automática de texto. Há duas opções para a digitação automática de texto no Emacs: a criação de um arquivo de "abreviaturas", e utilizar o recurso de "abreviação dinâmica" (**dabbrev**). Por padrão, o recurso de abreviação dinâmica está associado à combinação **M-/**.

Para criar "abreviaturas" pré-definidas, uma das formas é via modo de comando: **C-x a l** (ou "g", para global). Essa sequência cria uma abreviatura para a palavra previamente digitada. Para utilizar a abreviatura, o comando é **C-x a e**, ou (expand-abbrev), podendo ser associado a qualquer tecla. Ao sair do editor, ele perguntará se deseja gravas as abreviaturas para uso posterior. As abreviaturas podem ser globais ou locais. Caso sejam locais, serão específicas para um determinado modo (as abreviaturas de C não interferem nas abreviaturas de Java, etc.).

O recurso de configuração de teclas também é bastante útil para as mais diversas operações. Para definir a tecla F12 para o recurso de "auto-complete", por exemplo, adicione as seguintes linhas ao `.emacs`:

```
(global-set-key [f12]     'dabbrev-expand)
(define-key esc-map [f12] 'dabbrev-completion)
```

Na opção de coloração de códigos, é possível customizar as cores, bastando acessar o modo **list-faces-display**. Selecione o comando a ser alterado e clique com o 3º botão (ou os 2 botões simultâneos) para customizar a sua respectiva cor.

Finalmente, é possível abrir um shell e executar os comandos do pacote GNU sem sair do emacs, graças ao "eshell" (**M-x eshell**).

Como pode ser observado, o Emacs oferece recursos bastante sofisticados dependendo da linguagem que se estiver trabalhando. Muitas possuem o suporte nativo do Emacs, como o HTML, Java, e claro, C/C++, mas muitos outros modos do Emacs podem ser encontrados na Internet. O grande número de "undos" armazenados em memória, o suporte a várias entradas no "clipboard", a interface gráfica de simples interação, a possibilidade de edição rápida através de teclas de atalho e os inúmeros recursos embutidos no Emacs fazem dele um "canivete suiço", e é tido por alguns como o diferencial entre um usuário "newbie" e um guru do Linux.

Abaixo segue uma tabela com uma referência rápida de alguns comandos do Emacs.

Tabela 3: Modo de comandos do Emacs

| | |
|---|---|
| C-g | finaliza ação (ESC ESC ESC) |
| C-x C-s | salva o buffer ativo |
| C-x C-c | finaliza o editor |
| C-x b | muda de buffer |
| C-x 1 | apenas 1 buffer (unsplit) |
| C-x u | desfaz última alteração |
| M-w/C-y | copiar/colar área selecionada (C-SPC) |
| C-s | busca uma string |
| C-x g | goto line |
| C-x k | finaliza buffer |

Em geral, as teclas Control e Meta (alt) ativam o modo de comando do Emacs (C-x, C-c, M-x, etc).

Para mais informações, consulte o "help" do próprio Emacs, com o comando: **C-h i** (modo info), seguido de **m** para menu, e a seção que desejar saber mais. Por exemplo: `C-h i m emacs <ENTER> m programs`. Também é possível obter informações via "apropos", como a listagem dos modos do Emacs: `C-h a -mode$`.

## 3.2 Eclipse Framework

Nas seções anteriores apresentamos as ferramentas mais básicas de programação, encontradas em praticamente todas as distribuições Linux. Entretanto, há um projeto Open-Source que tem apelo de mercado significativo: o projeto **Eclipse**.

O Eclipse é uma IDE de programação e um framework, a partir do momento em que apresenta recursos de implementações para o próprio Eclipse (pontos de extensão) e a possibilidade de se desenvolver ferramentas externas utilizando o Eclipse como base. O Eclipse usa o paradigma de componentização, sendo todo baseado em plugins: módulos que adicionam funcionalidades ao sistema. Um plugin geralmente adiciona uma perspectiva ao Eclipse, ou um tipo de arquivo ao "wizard" de criação de arquivos, ações (menus e ícones) e recursos de configuração no menu de "Preferências".

O projeto Eclipse (www.eclipse.org) disponibiliza versões da IDE para Linux, Windows, Solaris, MacOSX, etc. Ressalta-se a necessidade de bastante memória para a plataforma, pois ela é bastante robusta e pode consumir uma parcela significativa de recursos da máquina.

O Eclipse vem por padrão com o plugin **JDT** — *Java Developer Toolkit*, além de suporte a CVS e outras facilidades para o desenvolvimento dos próprios plugins. Ele é tão extensível que pode funcionar não apenas como uma IDE, mas também como um leitor de sites RSS (com o plugin "All The News"), uma ferramenta de modelagem (com o plugin UML, disponível no site do projeto), habilitar o suporte a proxy (com o plugin "X-Parrots"), interface com o banco de dados (com o plugin "Quantum"), etc. É possível encontrar os mais diversos plugins para Eclipse no site: http://www.eclipse-plugins.info.

Vamos aos recursos do Eclipse para programadores:

- IDE multi-plataforma, por ser desenvolvida em Java/SWT (*Standard Widget Toolkit*).
- Validação de código *inline* e *auto-complete* de códigos configurável.
- Extensibilidade, por ser orientado a componentes. Isso permite sua integração com ferramentas externas de programação e "containers", especialmente relevante para programadores Java (integração com o Tomcat, JBoss, ferramentas de mapeamento, etc).
- Suporte a diversas linguagens de programação. Destaca-se o suporte nativo à linguagem Java, e à linguagem C/C++, através do plugin **CDT**, também integrante do projeto oficial.
- Modo de depuração sofisticado, com perspectiva de Debug específica para cada linguagem suportada.
- Facilidades para programação O.O. (Perspectivas O.O.).

Para instalar o Eclipse, basta descompactá-lo para algum diretório do sistema. Para instalar plugins, o procedimento é o mesmo: basta descompactar o plugin para seus respectivos diretórios: /plugins (os pacotes componentes do plugin) e /features (recursos de atualização, se houver). A partir da versão 3.x do Eclipse é possível instalar os plugins através do menu `Help > Install Features > New Features.`, caso o plugin possua módulo de atualização pela Internet.

A área de trabalho do Eclipse é chamada de **Workbench**, onde podemos visualizar a estrutura do projeto, a janela de mensagens (compilação, erros, etc), isto é, tudo o que é relevante para iniciarmos o desenvolvimento de nossos aplicativos. Para isso, o primeiro passo consiste na criação de um "Projeto", através dos "wizards" disponíveis no menu `File > New > New Project`. Depois de criado o projeto, o Eclipse irá sugerir a alteração da perspectiva, para que se inicie a criação de classes (para projetos O.O., como Java), fontes (File), etc.

O Eclipse trabalha com o conceito de **perspectivas**, ou seja, visões diferentes da IDE para determinadas tarefas e linguagens. Por exemplo, ao se trabalhar com um projeto Java, utiliza-se a perspectiva Java, que por sua vez apresentará no "workbench" o editor Java, a estrutura do projeto organizada em pacotes e APIs, hierarquia de super e subtipos, opções de "refactoring", ou seja, todos os elementos relevantes a um programador Java.

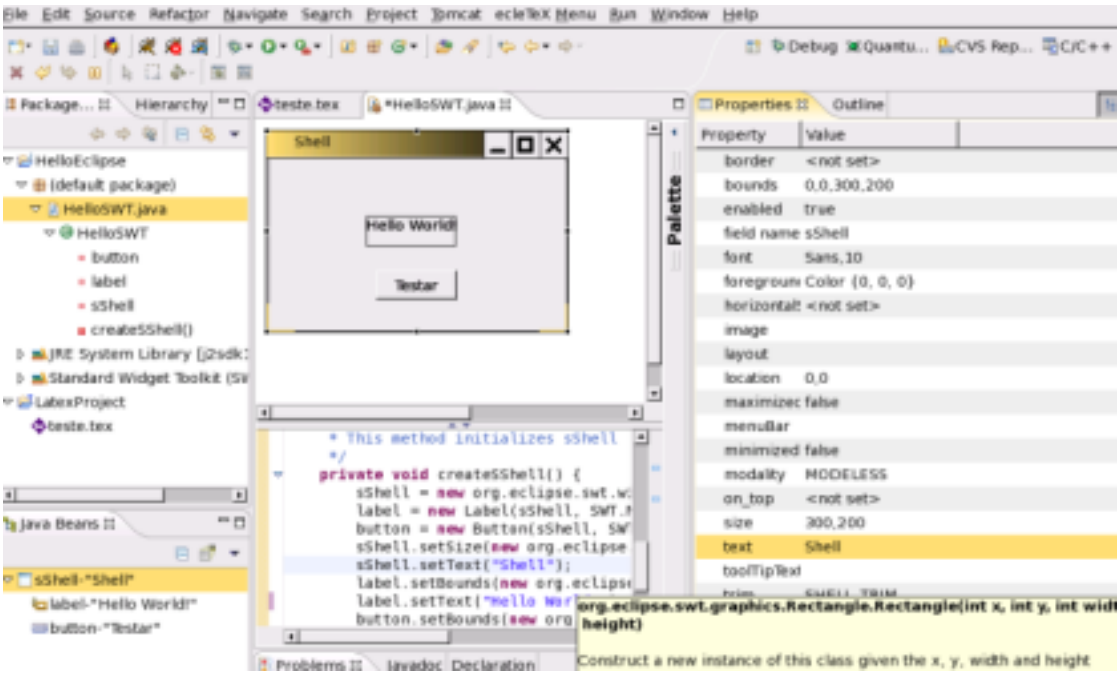


Figura 3: Eclipse com diversos plugins (VE, Tomcat, CDT, etc.)

Um recurso importante para iniciantes de Eclipse são os tutoriais apresentados logo no início do programa. Eles levam o usuário a passar por diversos recursos da IDE, e ao mesmo tempo, desenvolver programas que vão desde uma classe a um plugin, e até a uma interface SWT, que é a própria biblioteca gráfica utilizada no desenvolvimento do Eclipse. Este recurso é altamente recomendável para quem está iniciando na plataforma, pois os tutoriais são breves (levam menos de 1 hora) e completos, apresentando de forma bastante explicativa os diversos aspectos da ferramenta.

Depois de desenvolvido o programa, o Eclipse apresenta diferentes modos de execução (menu “Run”, e “Run as”), o que permite a especificação da classe “main”, argumentos, “classpath” e outras configurações específicas de ambiente que possam ser necessárias para a execução do programa.

Além disso, o Eclipse guarda um histórico local de todas as mudanças que ocorreram em um determinado fonte, acessadas via menu do projeto (`Compare With > Local History`, muito útil para acompanhar as modificações no decorrer do desenvolvimento.

Em suma, o Eclipse apresenta recursos sofisticados para desenvolvedores, em especial para programadores Java. As facilidades no desenvolvimento O.O. são úteis não apenas para programadores experientes, mas também para iniciantes, pois seus recursos facilitam bastante o aprendizado.

Entretanto, o Eclipse é bastante robusto e seu foco são projetos mais complexos. Além disso, ainda exige bastante da máquina, especialmente memória. Sendo assim, para pequenos projetos as ferramentas que acompanham a distribuição, como o jed e o kate, muitas vezes são suficientes. Para iniciantes, há também ferramentas voltadas ao aprendizado, como é o caso do BlueJ (www.bluej.org), uma ferramenta multi-plataforma desenvolvida para a aprendizagem da linguagem Java.

## 3.3 KDevelop

O ambiente de desenvolvimento que acompanha o KDE, o **KDevelop**, não poderia deixar de ser mencionado entre as principais ferramentas de desenvolvimento encontradas atualmente no Linux.

O KDevelop é uma IDE de programação bastante completa, que permite o desenvolvimento de aplicações gráficas ou console para Linux. Em uma analogia simplista, o KDeveloper seria o equivalente para Linux da ferramenta VC++. Entretanto, o KDevelop apresenta suporte a diversas linguagens (Bash, C/C++, Java, Pascal,

Perl, PHP, Python, Ruby, entre outras) e bibliotecas (GTK, wxWidgets, ncurses, etc.).

Entre seus recursos estão: gerenciamento de projetos, *auto-complete*, *wizards*, *templates*, *debugger*, etc. Porém, a disponibilidade de cada um varia de linguagem para linguagem. No momento, apenas a linguagem C/C++ apresenta o suporte completo da ferramenta, afinal ela foi especialmente desenvolvida para a confecção de programas C++ para o KDE, com um adicional suporte para a API grafica QT.

Da mesma forma que o projeto Eclipse, o KDevelop também possui uma arquitetura baseada em **plugins**. Por fim, o KDevelop também é flexível no que diz respeito ao suporte a controle de versões: CVS, Perforce, Subversion e Clearcase.

O KDevelop 3 possui a seguinte estrutura de interface:

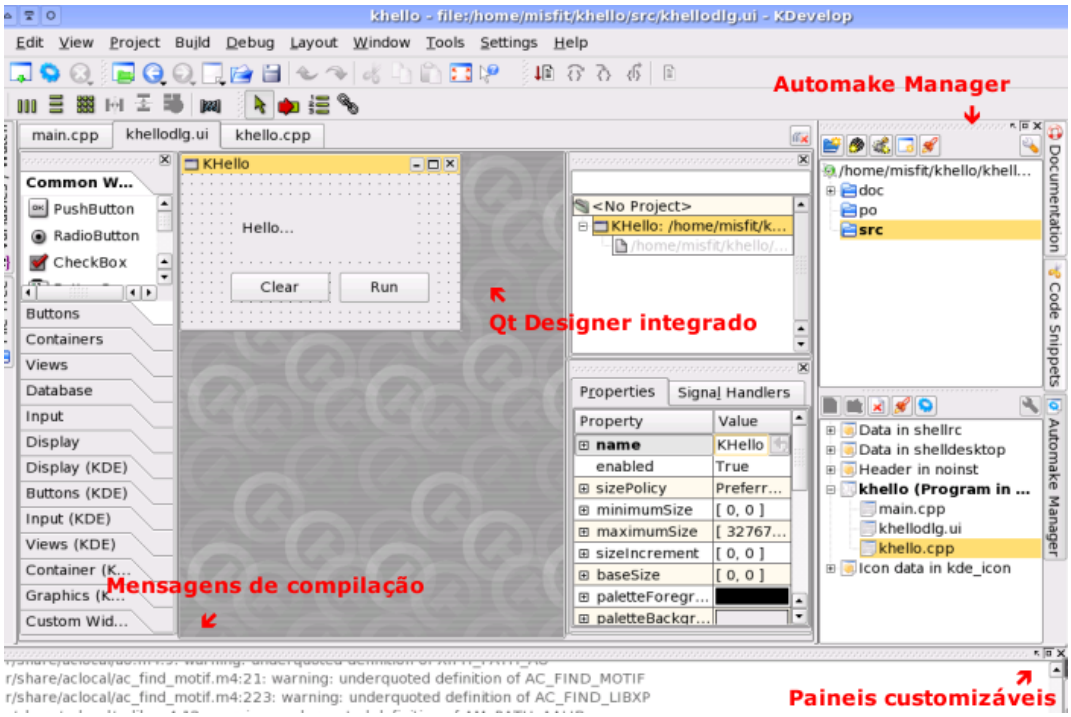


Figura 4: KDevelop

Para criar uma aplicação, a forma mais simples é utilizar o “Wizard”. Selecione Projeto > Novo Projeto, no Menu do KDevelop. O “wizard” apresentará uma série de “templates”, ou modelos. Para um projeto C++, isso envolve a criação de uma função main(), além de construtores/destrutores para a classe. No caso, selecione C++ > KDE > Simple KDE App, e chame-o de KHello, por exemplo.

Tente compilar. Caso não funcione, provavelmente o KDevelop não encontrou os caminhos das variáveis de ambiente QTDIR e KDEDIR, que podem ser configuradas em Project > Project Options > Configure.

Selecione Project > Build Configuration > default para sair do modo “debug” neste primeiro momento do desenvolvimento. Em “Project Options” selecione Make Options > “Abort on first error”.

## 3.4 Qt Designer

Com a integração ao QtDesigner, o KDevelop passa a ser uma verdadeira ferramenta RAD (Rapid Application Development). Apesar do QtDesigner incluir também um editor de código, utilizaremos o KDevelop para tanto.

Uma interface gráfica, ou “Form”, possui a extensão **.ui**, para que possa ser processada pelo “uic”, o programa que gera o “.cpp” e “.h” correspondentes à interface. O KDevelop responsabiliza-se por este processo. Sendo assim apenas precisamos criar o “Form”.

Selecione File > New, e o template “Widget”. Chame-o de khellodlg.ui, e selecione a checkbox “Add to project”. Confirme a tela seguinte, e ele deverá ser adicionado ao *Automake Manager*. Clique duas vezes no novo arquivo para editar a interface. Depois de adicionar a “ui”, será preciso rodar “Automake” antes de compilar o programa.

A primeira coisa a ser alterada é o nome do Form. Chame-o de “KHello”. Agora vamos adicionar **widgets**, ou componentes, à nossa aplicação. Insira um “Label” e também um botão (“Pushbuttom”).

No desenho da interface, muitas vezes é útil adicionar “Spacers”, para que o layout se mantenha independente do tamanho da janela, já que os **widgets** não são dimensionados. [6]

Isso encerra o desenvolvimento da interface.

### 3.4.1 Sinais e Slots

Uma particularidade das aplicações Qt são os *signals/slots* utilizados para a comunicação entre os objetos da tela, ao invés de “callbacks”, como encontrado em outras bibliotecas gráficas.

Por exemplo, um widget pode ter um sinal para o evento *click()* associado a um slot *clear()* de uma label, e assim por diante.

Após desenvolver o layout, o estágio final envolve a criação das conexões de “signals/slot” (via ícone, menu, ou a tecla de atalho F3). Depois de selecionar a opção “Signal”, selecione o botão criado anteriormente e arraste até o fim do formulário. Em seguida, clique no botão “Edit Slots”, e digite o nome “helloSlot()”. Finalmente, para efetuar a conexão, associe ao botão o sinal “clicked” e o “slot” que acabamos de criar.

## 3.5 Considerações Finais

Neste momento, de volta ao “Automake Manager”, nosso projeto consistirá de 3 arquivos: khello.cpp, main.cpp e khellodlg.ui. Delete o “khello.cpp”, clique com o botão direito na khellodlg.ui e selecione “Subclass Wizard”. Preencha a opção “Reformat source” e crie um novo khello.cpp, com as mesmas propriedades da classe original.

No main.cpp, ao invés de KMainWindow (aplicação KDE), como nossa aplicação é QT, comente as linhas referentes a KApplication (KCmdLineArgs e a condição de RESTORE, antes de “app.exec”).

Para implementar a função de nosso botão, ou nosso “slot”, complemente a função “helloSlot”, gerada automaticamente pelo KDevelop.

Pronto. Para compilar nosso programa lembre-se de seguir os passos:

1. *Build > Run automake & friends*
2. *Build > Run Configure*
3. *Build > Build Project*
4. *Build > Execute Program*

Isso finaliza a criação de nossa aplicação. Em versões mais novas do KDeveloper nossa aplicação poderia ter sido criada pelo “wizard” `Simple Designer Application`, sem nenhuma intervenção no código.

# 4 Controle de versão

Em um projeto de software, é essencial que haja algum tipo de controle de fontes, para possibilitar que múltiplas pessoas trabalhem no mesmo conjunto de arquivos, sem sobrescrever o trabalho do outro. A forma mais simples de controle é através de *locking* de arquivos: se alguém abrir um arquivo para edição, ninguém poderá editá-lo até que o arquivo seja liberado. Em aplicações complexas, a manutenção das alterações é um fator essencial, seja para ter um acompanhamento do processo, seja para debug, evitando a passagem bugs despercebidos.

Com um sistema adequado de desenvolvimento com controle de versões, caso haja um problema na aplicação, é possivel retornar para a versão anterior, funcional, e assim permitir que os desenvolvedores trabalhem em solucoes melhores para o problema sem afetar o projeto como um todo.

## 4.1 CVS

CVS é um sistema de controle de versão tradicional em sistemas Linux. A utilização do CVS basicamente envolve um “check-out”, e posteriores “check-ins” de código pelo desenvolvedor. O **checkout** cria a pasta do projeto em seu computador, e é efetuado na primeira vez que um desenvolvedor irá trabalhar com um projeto previamente armazenado em um repositório. Após alterações nos programas, efetua-se o **commit**, e a cada “check-in” uma nova versão é criada, documentada e armazenada no repositorio. Na realidade, apenas as diferenças entre as versões são de fato armazenadas no repositório. Os arquivos no repositório não são “backups”, mas sim arquivos (com a extensao “,v”), que guardam apenas as modificações no projeto original e meta-dados para a recuperação da versão.[2]

A criação de um sistema CVS consiste na definição de um **repositório**, que será o responsável por armazenar todas as revisões dos seus arquivos. O uso mais comum do CVS é a manutenção de versões de programas (código fonte), porém ele pode ser utilizado para gerenciar qualquer tipo de arquivo, de editoração de imagens, criação de videos a listas de supermercados.

Um repositório também pode armazenar múltiplos projetos (chamados **módulos**). O CVS pode rodar como “pserver” (serviço) ou localmente. A forma mais simples é configurar um repositório local e efetuar o acesso via SSH, o que elimina a necessidade de configuração de um servidor, apenas de um cliente e um repositório.

- Para configurar um repositório CVS, são necessários 3 passos:

1. Definir o meio de acesso ao CVS

   ```
   $ export CVS_RSH=ssh
   ```

   A forma de conexão via RSH/SSH (EXT) apenas trabalha com as portas default.

2. Definir o repositório: setar a variável de ambiente CVSROOT

   ```
   $ export CVSROOT=:ext:user@localhost:/home/username/projetos/cvs
   ```

   Onde o caracter de “:” age como separador entre o método de acesso (ext, server ou local), o IP do repositório e o caminho no *filesystem*, respectivamente.

3. “Iniciar” o repositório

   ```
   $ cvs -d /home/user/projeto/cvs init
   ```

- Para adicionar um módulo ao repositório:

```
$ export CVSROOT=:ext:user@localhost:/home/username/projetos/cvs
$ cd nome_do_projeto/
$ cvs import projeto username start
```

Onde “projeto” será o nome do módulo, “username” a *vendor tag*, ou o dono do projeto, e por último a *release tag*, um nome representativo à primeira versão do projeto.

- E finalmente, utilizando o CVS.

1. Efetuar o primeiro checkout, caso ainda não tenha participado do projeto.

   ```
   $ cvs checkout projeto
     (ou simplesmente, cvs co projeto)
   ```

2. Gravar as alterações no repositório: “commit”

```
$ cvs commit
```

3. Adicionar arquivos ao repositório: “add”

```
$ cvs add nome_arquivo
```

4. Manter o repositório em sincronia e verificar alterações: “update” e “diff”

```
$ cvs update
$ cvs -Q diff
```

Caso alterações sejam feitas no mesmo arquivo, o CVS fará o “merge”. Caso as alterações tenham sido feitas no mesmo local, o CVS marcará os arquivos como “conflitantes”, e desta forma você terá que fazer o “merge” manual nas regiões marcadas.

Vale lembrar que os principais editores e IDEs (como Emacs e Eclipse) possuem recursos de CVS nativos, isto é, os comandos envolvidos no check-out/check-in são transparententes ao desenvolvedor.

### 4.2 Subversion

Além do CVS, há uma ferramenta que tem ganho uma certa popularidade no controle de versões: o Subversion (SVN). Este sistema tem como proposta superar limitações e falhas encontradas no CVS original. Ele propõe-se a ser um substituto ao CVS, com operação similar, de forma a não exigir um novo aprendizado de quem já utilizava CVS em seus projetos.

Entre os recursos adicionais apresentados pelo Subversion, destacam-se [1]:

- Versionamento de diretórios.
- Histórico real de versões: renomeações e cópias são logadas, não apenas o conteúdo dos arquivos.
- Commits atômicos: caso haja um erro na transferência ao repositorio, interrompa o processo e retome ao estado consistente.
- Versionamento de meta-dados: possibilidade de armazenar qualquer par arbitrário de chave/valor (ou propriedades) de arquivos e diretórios.

Subversion requer a APR (*Apache Portable Runtime Library*), o que possibilita sua compilação em qualquer S.O. suportado pelo httpd server apache. Ele pode rodar como um módulo do Apache (mod_dav_svn) ou “standalone” (svnserver), via SSH.

## Referências

[1] B. C. et al. Version Control with Subversion. Web: http://svnbook.red-bean.com/, 2004.

[2] K. Fogel. Open Source Development With CVS. Web: http://cvsbook.red-bean.com/, 2000.

[3] M. G. BASH Programming - Introduction HOW-TO. Web: http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html, 2000.

[4] S. Heroor. C editing with VIM - HOWTO. Web: http://www.tldp.org/HOWTO/C-editing-with-VIM-HOWTO/, 2001.

[5] S. Mahajan. The Emacs Editor. Web: http://www.aims.ac.za/resources/tutorials/emacs/.

[6] A.-M. Mahfouf. Qt Designer and KDevelop-3.0 for beginners. Web: http://women.kde.org/articles/tutorials/kdevelop3/.

[7] S. J. Peralta. Scripting Graphical Commands with Tcl/Tk Mini-HOWTO. Web: http://www.tldp.org/HOWTO/Scripting-GUI-TclTk/, 2003.