PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº17 - Dezembro de 2008

www.portugal-a-programar.org

e ainda...

Introdução ao Arduino / Performance e Optimização MySQL

ISSN 1647-0710

índice

equipa PROGRAMAR

coordenador
Miguel Pais

coordenador adjunto
Joel Ramos

editor
Pedro Abreu

redacção
Bruno Oliveira
Ciro Cardoso
João Antão
Miguel Rentes
Nuno Santos

colaboradores
David Ferreira
João Matos
José Fontainhas
Nuno João

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

COORDENADOR

Coordenador Adjunto desde a 4ª edição, é actualmente o Coordenador da Revista Programar. Frequenta o 2º ano do curso de Engenharia Informática e de Computadores no IST.

Miguel Pais

Boas notícias

Desde o início que a comunidade e todos os utilizadores envolvidos neste projecto sonham bem alto sobre ele, não apenas desejando o possível mas quase sempre o inatingível. Triste notar que dezassete edições são três anos e nada meramente inatingível tenha sido alguma vez tentado, conquistado, enfim oferecido a todos os utilizadores que alguma vez tenham dedicado uns minutos a este projecto. Não é que a participação na revista Programar viva ou apenas sobreviva de recompensas "douradas" ou mediáticas, pois até agora tais não houve e cada utilizador soube apreciar o resultado final e ter orgulho nele, mas uma prenda não faz mal a ninguém, de vez em quando.

É por isso que tenho o prazer de anunciar que a partir desta edição a Revista Programar será incluída no DVD da publicação sobre informática e novas tecnologias portuguesa PC Guia. Desde já aproveito para agradecer à PC Guia a confiança que depositou no projecto, comprovando a sua qualidade e relevância, ainda que direccionado a um grupo de utilizadores que, devido à sua dimensão, não encontra no mercado conteúdos adequados, só por este meio podendo ser fornecidos. A edição passada pode portanto ser encontrada na edição de Dezembro da PC Guia e as próximas edições seguir-se-ão.

Espero que todos sintam o maior orgulho possível com este pequeno feito, e caso sonhem ainda mais alto e achem que muito mais era possível, podem sempre sugerir através do mail da revista próximos feitos que gostassem de ver atingidos.

Como o título não é plural sem motivo, aproveitámos estes últimos dois meses para implementar algo há muito pedido, principalmente pelos autores dos artigos: um sistema de feedback. Nas habituais caixas de informação sobre o autor do artigo encontrará agora também um email para o qual pode enviar qualquer dúvida, sugestão ou agradecimento. Aconselhamos vivamente a que tal seja feito pois a revista não vive só do lado de cá e os autores precisam bastante de receber: sugestões, para que possam melhorar o seu trabalho; dúvidas, evidenciando um artigo que despertou a curiosidade de alguém; agradecimentos, enfim a prova final, se é que era necessária.

Em Janeiro faremos 3 anos. 2006 já lá vai tão perto e tão longe. A todos os que continuam a apoiar-nos o nosso muito obrigado.

Vem aí o e-Universidades

Depois do programa e-escola abranger os alunos de todos os anos, chega a vez dos universitários. Ao abrigo da Rede de Investigação e Aprendizagem Toshiba-Portugal, a empresa vai ter portáteis e-Universidades.

Com um investimento de 1 milhão de euros, durante cinco anos, a Toshiba, a Fundação para a Ciência e a Tecnologia (FCT), a Agência para a Sociedade do Conhecimento (UMIC) e a Agência Ciência Viva criaram a Rede de Investigação e Aprendizagem Toshiba Portugal.

A Toshiba apresentou também o e-Universidade-Toshiba, dirigido especificamente a alunos e docentes do ensino superior. O computador é semelhante ao que é distribuído no e-Escolas, um Satellite com processador Intel Dual Core T3200 a 2 GHz, 320 GB de disco, 3 GB de RAM, uma placa gráfica ATI Radeon HD 2400, ecrã de 15,4 polegadas e Windows Vista. Estes computadores vão ter suites de produtividade da Autodesk e da Microsoft e serão compatíveis com a rede wireless das universidades.

Os portáteis vão ter, como opção, acesso à banda larga móvel e duas opções de compra: 50 euros de entrada, mais 24 mensalidades de 25 euros ou sem entrada e 36 mensalidades de 20 euros. Em qualquer uma destas opções, acrescem 25 euros para despesas.

O lançamento destes computadores está para breve, faltando apenas acertar pormenores, nomeadamente com as entidades financiadoras do projecto, apresentado numa cerimónia onde estiveram presentes, entre outros, o ministro da Ciência, Tecnologia e Ensino Superior, Mariano Gago e o presidente e CEO da Toshiba Corporation, Atsutoshi Nishida.

Python 3.0 lançado

No passado dia 4 de Dezembro, foi lançada ao público a versão final do Python 3.0. Esta nova versão da linguagem é incompatível com as versões 2.x, e veio para melhorar ainda mais esta excelente linguagem.

Entre as novas alterações, podem-se destacar a remoção de sintaxe redundante, organização e limpeza da biblioteca standard e modificação dos tipos de strings (texto vs data em vez de unicode vs 8-bit).

Para mais informações, veja o anúncio na mailing list: http://mail.python.org/pipermail/python-list/2008-December/518408.html

Chrome abandona versão beta

Um novo passo foi dado no mercado dos browsers com a saída do Google Chrome da fase beta em tempo praticamente recorde. O produto foi melhorado nos últimos 100 dias e já conseguiu alcançar os 10 milhões de utilizadores, informa a Google no seu blog.

A última versão do software melhora a velocidade, protecção de privacidade e execução de conteúdos multimédia face às edições anteriores e, segundo a empresa, é mais seguro.

Até aqui, a Google continua a trabalhar nas versões do browser para Mac e Linux embora ainda não exista qualquer previsão para o lançamento dos produtos.

O final da versão beta vem contrariar uma das últimas notícias avançadas pela Google que dava como certo o lançamento da edição final do Chrome para todos os sistemas operativos só na primeira metade do próximo ano.

Subversion: Controlo total sobre o Software

O controlo de versões de um software sempre foi uma questão muito importante e delicada para qualquer programador que se preze. É actualmente impensável desenvolver projectos de software complexos e com requisitos que estão frequentemente a mudar, sem ter um sistema que controle tudo o que acontece ao software: as alterações ao código, a possibilidade de voltar sempre atrás até uma versão estável do software (óptimo quando ocorrem erros e se precisa de regredir no código), ver que alterações foram feitas no código globalmente ou apenas numa porção deste (incluindo as respostas às perguntas: quando? por quem? para que fim? em que ficheiros?), controlar eficazmente a fusão (em Inglês, merge) das alterações em vários módulos separados de software num mesmo produto unificado, e também controlar a própria documentação do software.

Sem dúvida que em qualquer projecto de software (por muito pequeno ou muito grande que seja), o controlo da informação é crucial para o sucesso do mesmo. Por isto mesmo, a tarefa de controlar as versões de um software não é de todo fácil, nem todos os sistemas de controlo de versões, ou VCS - Version Control System, são iguais e com as mesmas funcionalidades.

Neste artigo vamos ver como funciona o Subversion (também conhecido por SVN): um sistema de controlo de versões open source (software livre) de enorme sucesso actual. Vamos ver também como se instala o Subversion (cliente e servidor) para as plataformas Linux (distribuição Ubuntu) e Windows, como se configura o servidor HTTP da Apache (usufruindo logo à partida das vantagens da autenticação e autorização) para usar URLs dos repositórios SVN, como se usa o Subversion no dia-a-dia do trabalho de um programador e por fim, que ferramentas gráficas ou IDE's (Integrated Development Environment) existem à disposição de todos para serem usadas com o Subversion.

No final da leitura deste artigo, espera-se que se fique a conhecer e a saber usar o Subversion como uma ferramenta de trabalho muito útil para qualquer programador (e mesmo não-programador) no que toca à gestão das alterações do software, ou seja, controlando totalmente o percurso do software.

Porquê controlar o que acontece ao Software?

As funcionalidades básicas de qualquer sistema de controlo de versões são o de permitir anotar todas as alterações dos ficheiros ao longo do tempo e de unir as contribuições de vários programadores. Para suportar isto, o VCS tem de manter um histórico de todas as alterações que foram feitas ao longo do tempo por diferentes pessoas. Desta forma, é possível voltar atrás no código até uma revisão mais antiga, e ver que aspecto tinham os ficheiros nessa altura. Para além disso, o VCS tem de garantir sempre a correcta integração das alterações feitas no código.

Mas o que se ganha efectivamente com o controlo das versões de software? Especialmente, para uma equipa pequena que benefícios se podem tirar de um bom VCS que valham a pena perder tempo a aprender como usá-lo no dia-a-dia? Eis algumas das razões para usar sempre um VCS:

Integridade dos dados

Um bom VCS ajuda a proteger a integridade dos dados. Ao manter um histórico das revisões (que são os conjuntos atómicos de alterações) deixa de haver a preocupação de possivelmente se estar a apagar código que mais tarde vamos perder tempo a reproduzir e que afinal era mesmo importante. Com um bom VCS podemos voltar sempre a qualquer snapshot do código, inclusivamente a esse momento anterior a termos apagado código. Esta característica também é útil no backup de dados, porque se os programadores guardarem regularmente as suas alterações no repositório central (no SVN, isto designa-se de commit) que tem todo o código, então é fácil ter um processo automático que arquive todo este código do repositório, e assim resolve-se o problema do código não estar unicamente no computador de um programador (mesmo à espera que aconteça aquela falha inevitável no disco...).

Productividade

Ao libertar os programadores da tarefa árdua de ter que integrar manualmente todas as alterações que fizeram (e tipicamente em muitos ficheiros ao mesmo tempo), um VCS pode aumentar bastante a productividade na criação de software. De facto, com um bom VCS os programadores conseguem testar as alterações que fizeram e que outros programadores fizeram, resolver possíveis conflitos de código (que correspondem a mudanças nos mesmos ficheiros e nas mesmas linhas desses ficheiros) mesmo antes de incluir esse código no repositório central, e de uma forma automática. E, com mais tempo para programar, nasce melhor software (se o programador não se distraír das suas programações).

Gestão de Projectos

Um bom VCS consegue reunir rapidamente um conjunto de informações muito úteis para um gestor de projectos: quem alterou o quê, quando, porquê, quantas modificações fez no mesmo projecto (ou módulo), e se as alterações feitas vão de encontro à finalidade do projecto (por exemplo, se existirem várias modificações que não resolvem os bugs encontrados). Estas informações são importantes para os gestores de projectos e para os programadores que não são apenas programadores mas também líderes de pequenas equipas de software. Apesar de um VCS estar principalmente preocupado com a gestão das versões, as informações que fornece podem ser facilmente usadas por software mais específico da área de gestão de projectos.

Suporte a projectos de Engenharia de Software

Tipicamente, bons projectos de software são desenvolvidos com um processo de engenharia de software. Por engenharia de software, quero dizer a aplicação de políticas de desenvolvimento com o intuito de assegurar que o producto final do processo vai de encontro aos objectivos, dentro dos prazos previstos e com os melhores padrões possíveis de qualidade do software. Um bom processo de engenharia de software envolve um conjunto de passos, tais como o desenho bem detalhado do projecto na sua globalidade, a revisão dos componentes do software por parte dos analistas e engenheiros, a anotação de todos os bugs, e testes de qualidade ao software. Nenhum destes passos é explicitamente suportado por qualquer VCS actualmente, mas as características de muitos VCS (como por exemplo, os hook scripts e logs do SVN) são excelentes ajudas ao processo de engenharia de software. Vamos ver mais adiante o que o SVN nos fornece neste campo.

Ramificação do desenvolvimento

À medida que os projectos vão aumentando, há necessidade de ramificar o projecto. Isto é, vai surgir naturalmente a necessidade de um programador (ou conjunto de programadores) desenvolver features que não podem de maneira nenhuma minar o trabalho que já existe guardado num repositório de código central. Por exemplo, basta pensar num novo módulo que vai demorar tempo a ser desenvolvido e que vai ter repercussões nos restantes módulos. Neste caso, o ideal é o programador trabalhar num ramo privado, em que este ramo não é mais do que uma cópia de todo o código que já existia e no qual vai poder alterar à vontade, sem o problema de estar a interferir com o código de outros programadores. No final, quando todas as alterações estiverem feitas, o programador funde todo o seu código com o que existe no repositório central (podendo ter que resolver conflitos), e o ramo pode deixar de existir. Outro caso em que os ramos são importantes é no suporte a versões antigas do software. Cada uma das versões antigas poderá ser um ramo do código a dada altura (por exemplo, na altura em que sai uma nova versão), e a equipa de programadores pode testar a resolução dos bugs nesses ramos de código antigo e posteriormente entregar ao cliente uma versão corrigida desse software para aquela versão anterior. Nem todos os VCS possuem esta funcionalidade nem tão pouco fornecem uma alternativa. No entanto, o VCS analisado neste artigo fornece esta a outras funcionalidades importantes para o controlo do software.

Anotação dos logs

Para além de saber quem alterou o quê, é também importante saber o porquê. A cada alteração deverá ficar sempre associada uma mensagem que traduza a alteração que se fez e que pode posteriormente ser usada para construir um CHANGELOG ou ainda ser injectada num sistema de tracking, tal como o Trac (http://trac.edgewall.org/) de que falaremos mais abaixo.

Distribuição do trabalho

Actualmente, com a era da globalização e do alcance fácil de uma ligação à Internet, o trabalho à distância é uma realidade presente em cada vez mais empresas, seja o programador que está a aceder ao código do outro lado da cidade ou seja uma empresa sub-contratada do outro lado do planeta; um VCS deverá conseguir equilibrar esta distribuição de trabalho ao gerir todas estas alterações ao código pelos diferentes programadores e também unificar este mesmo código automaticamente. Combinando estas características com uma comunicação (que é fundamental) através de e-mail ou um cliente de IM (como o GTalk, MSN Messenger, Skype ou ICQ), é praticamente tão fácil de trabalhar remotamente como localmente ao lado dos restantes elementos da equipa.

Desenvolvimento rápido

As mais recentes metodologias de desenvolvimento de software apontam para uma forma de programar mais rápida, e flexível usando metodologias de *Extreme Programming* (XP) e de *Desenvolvimento Ágil* de software. Estas metodologias de desenvolver software, esperam por

iterações mais rápidas e sucessivas relativamente a metodologias mais "planeadas". Esta forma de programar é facilitada por um VCS que consiga manter e indicar todas as alterações por que passou o software nas sucessivas iterações. Para além disso, um repositório central de software pode ajudar na automatização dos builds frequentes que são precisos por um processo de desenvolvimento ágil.

O que é o Subversion?

O Subversion é um VCS open source que tem tido enorme sucesso mundial e sido escolhido não só para a gestão de pequenos projectos open source, como também para projectos comerciais de grandes empresas de software. É um VCS livre, a custo zero e que se pode alterar a gosto (http://subversion.tigris.org/license-1.html), e tem-se revelado como a melhor alternativa actual a VCS comerciais, como é o caso do Rational ClearCase da IBM, o Microsoft SourceSafe (já de si bastante decrépito; uma melhor alternativa actual é o Microsoft Team Foundation Server) ou ainda o Borland StarTeam.

É um sistema de controlo de versões que faz a gestão de ficheiros e directórios ao longo do tempo. Surgiu em 2000 pelas mãos da empresa CollabNet (ver http://en.wikipedia.org/wiki/CollabNet) com o objectivo principal de substituir o **CVS** - *Concurrent Versions System*, que até então era bastante usado, mas que tinha limitações claras na usabilidade.

A ideia principal do SVN é a de ter uma árvore de ficheiros colocados num repositório central. O repositório é como um servidor de ficheiros, com a excepção de que se lembra de todas as mudanças feitas nos ficheiros e directórios. Isto permite que versões antigas de software sejam recuperadas, ou apenas que se possa examinar o histórico das alterações feitas nos dados.

O Subversion pode também aceder a repositórios remotos, o que permite que possa ser utilizado por pessoas em diferentes máquinas, permitindo assim que haja uma maior colaboração no desenvolvimento de software. Este desenvolvimento pode, portanto, ser mais rápido, e uma vez que há versões no código desenvolvido, não há receio de que se perca qualidade no software uma vez que se houver uma mudança incorrecta no software, basta apenas retroceder nessa alteração. Simples e rápido.

O Subversion fornece:

• Versionamento de directórios: o CVS mostra apenas o histórico de ficheiros individuais, mas o Subversion implementa um sistema "virtual" de versões que pode tomar nota das alterações a árvores inteiras de directórios ao longo do tempo. Desta maneira, ficheiros e directorias têm versões;

• Histórico de versões: uma vez que o CVS é limitado nas versões dos ficheiros, operações como copiar ou renomear ficheiros não são suportadas. O CVS também não consegue substituir um ficheiro com uma determinada versão por outro ficheiro com o mesmo nome, sem que antes esse novo ficheiro tenha herdado todo o histórico do ficheiro antigo, e pode acontecer que este novo ficheiro não tenha nenhuma relação com o antigo do mesmo nome. Com o Subversion pode-se adicionar, apagar, copiar, e renomear tanto ficheiros como directorias. E cada novo ficheiro adicionado começa com um histório limpo apenas para aquele ficheiro;

• Commit's atómicos: um conjunto de modificações (designado de commit) vai para o repositório de uma só vez como um conjunto global, ou não vai de todo (precisamente como um commit em SQL, isto é, se o commit falhar durante a operação, nada é passado para o servidor). Isto permite que os programadores possam construir e aplicar as modificações ao software como conjuntos lógicos de modificações de software, prevenindo que possam ocorrer problemas quando apenas uma parte do conjunto de alterações é enviado com sucesso para o repositório;

• Versionamento de meta-dados: cada ficheiro e directoria tem um conjunto de propriedades – chaves e os valores das chaves – associadas. Pode--se criar e guardar qualquer par arbitrário chave/valor, que se deseje. As propriedades têm também versões ao longo do tempo, como para os conteúdos dos ficheiros; As propriedades fornecem um maior controlo e customização do código-fonte no repositório;

• Escolha de redes de acesso: o Subversion tem uma noção abstracta de acesso a um repositório, tornando mais fácil a criação de novos mecanismos de rede. O Subversion pode ligar--se ao servidor de HTTP da Apache como um módulo de extensão. Isto dá ao Subversion uma grande vantagem na estabilidade e interoperabilidade, e acesso instantâneo às funcionalidades existentes fornecidas por este servidor HTTP – autenticação, autorização, etc. Uma versão mais leve, sem se ligar ao servidor HTTP do Apache, do servidor de Subversion está também disponível (designada por svnserve). Este servidor (svnserve) comunica através de um protocolo que pode ser facilmente usado num túnel SSH;

• Tratamento consistente de dados: o Subversion percebe as diferenças nos ficheiros através de um algoritmo que diferencia os binários de texto, e que funciona tão bem para ficheiros de texto como para ficheiros binários. Ambos os tipos de ficheiros são comprimidos no repositório, e as diferenças são transmitidas em ambas as direcções através da rede de computadores;

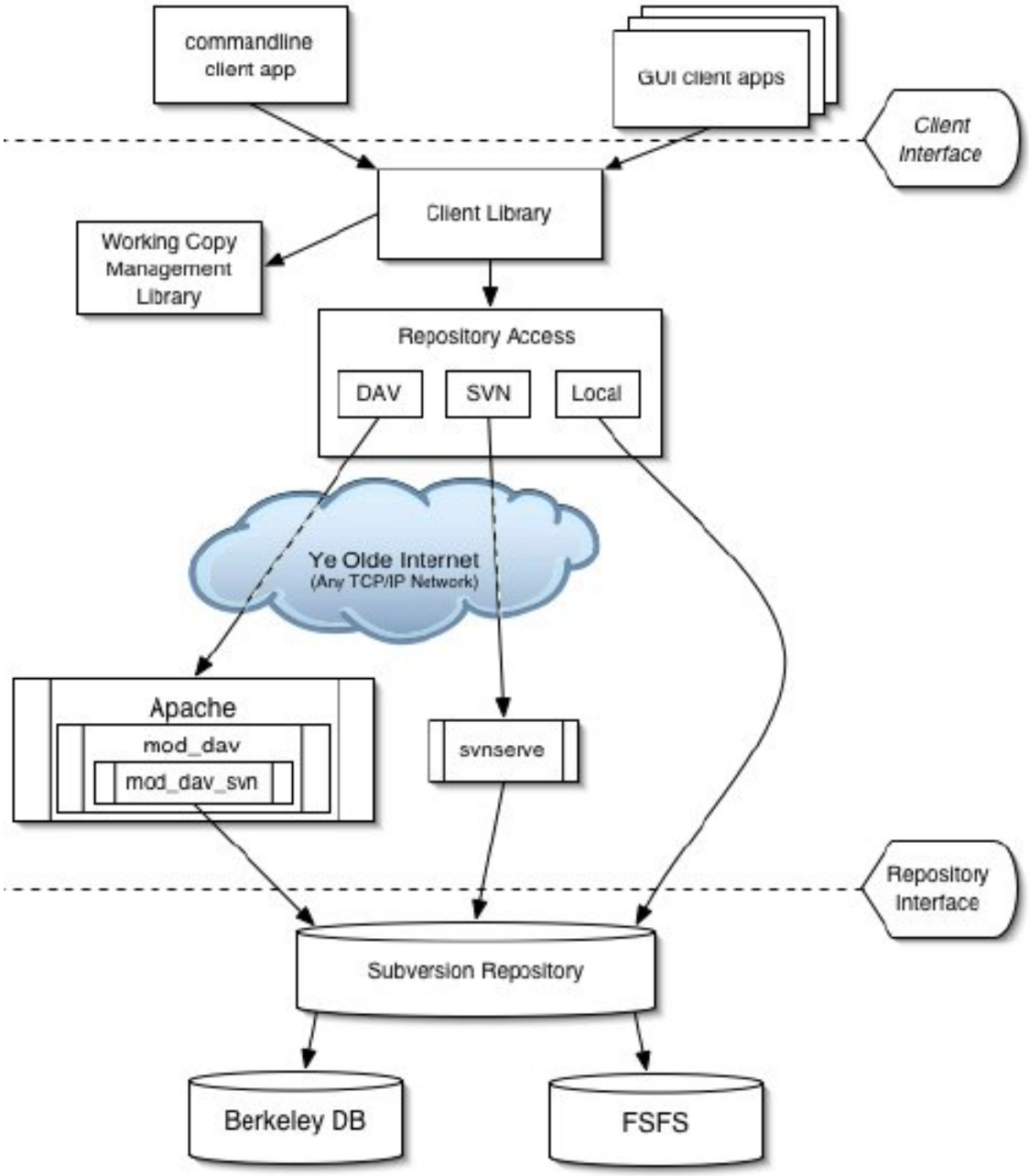
• Ramificação (branching) e Etiquetagem (tagging) eficientes: o custo de ramificar e etiquetar não são proporcionais ao tamanho do projecto. O Subversion cria

ramos e etiquetas ao copiar o projecto usando um mecanismo semelhante a um link em Linux. Por isso, estas operações levam um período de tempo bastante curto e constante. O que é excelente.

• Usabilidade: o Subversion é implementado numa colecção de bibliotecas C partilhadas com APIs bem definidas. Isto torna o Subversion extremamente fácil de manter e modificar por outras aplicações e linguagens de programação. Existe também actualmente uma biblioteca Java, SVNKit, que implementa o Subversion e que pode ser usada por qualquer aplicação Java.

Arquitectura do SVN

Na figura seguinte mostra--se a arquitectura do Subversion numa visão de alto-nível. Como se pode ver, o Subversion segue uma filosofia de implementação do tipo cliente-servidor. Numa extremidade está o repositório do Subversion que tem o projecto com as diferentes versões. Na outra extremidade está o cliente de Subversion, que gere as porções locais do projecto em determinada versão (chamadas cópias de trabalho – working copies). Entre estas extremidades há vários caminhos possíveis através das várias camadas de acesso ao repositório (RA – Repository Access). Alguns destes caminhos (ou percursos), atravessam redes de computadores e terminam em servidores que acedem ao repositório.



Outros caminhos não passam por redes de computadores e levam a um acesso directo ao repositório.

Existem dois tipos de filesystem para os repositórios SVN: FSFS (o mais recente e recomendado) e o Berkeley DB da Oracle. Com as versões mais recentes do SVN (desde a versão 1.4.6), o filesystem criado por omissão é FSFS.

Conceitos básicos do Subversion

O Subversion, tal como foi dito acima, é um sistema centralizado de partilha de informação. No seu núcleo estrutural encontra--se um repositório central de dados. Este repositório guarda a informação sob a forma de uma árvore de um sistema de ficheiros – tal como uma hierarquia típica de ficheiros e directorias. Qualquer número de clientes ligam--se ao repositório (directa ou indirectamente), e lêem ou escrevem para os ficheiros do repositório. Ao escrever, o cliente torna a informação disponível a outros utilizadores; ao ler, o cliente recebe informação de outros utilizadores. Recorde--se que o Subversion não permite apenas partilhar código--fonte de qualquer aplicação de software, mas sim todo o tipo de ficheiros (texto ou binários).

Apesar de parecer funcionar exactamente do mesmo modo que um normal servidor de ficheiros, o Subversion diferencia--se no facto de poder recordar cada alteração feita nos ficheiros, e cada alteração feita na árvore de uma directoria, como por exemplo, a adição, a remoção ou o re-arranjo de ficheiros e directorias.

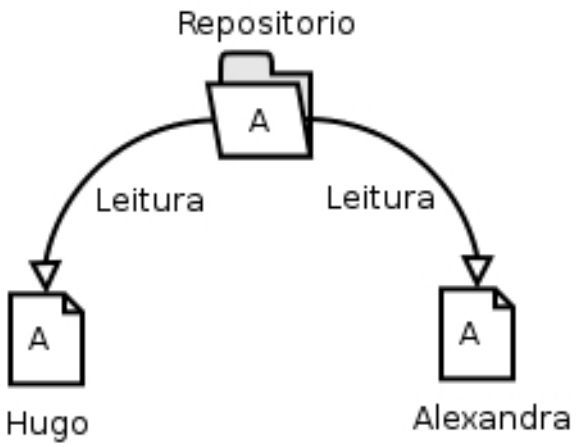
Quando um cliente lê dados do repositório, ele vê normalmente apenas a última versão da árvore do sistema de ficheiros. Porém, o cliente pode ainda ver os estados anteriores do sistema de ficheiros. Por exemplo, um cliente pode colocar questões sobre o histórico tais como, Que ficheiros tinha esta directoria na passada Quarta--feira? ou Quem foi a última pessoa a modificar este ficheiro, e que alterações é que essa pessoa fez?. Este tipo de questões são fundamentais para qualquer sistema de controlo de versões: sistemas desenhados para guardar e tomar nota das alterações aos dados ao longo do tempo.

O problema do controlo de versões

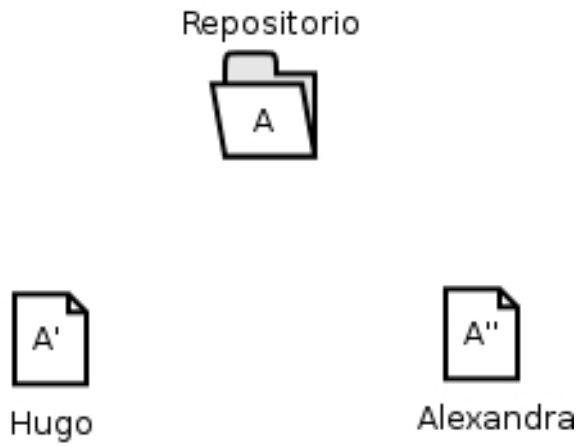
O objectivo de um sistema de controlo de versões é o de permitir a colaboração na edição e partilha de dados. Isto coloca um problema fundamental: como é que esse sistema permite que os utilizadores partilhem a informação, mas impeça que um utilizador altere acidentalmente um ficheiro que está a ser modificado por outro? Num repositório com um grande número de alterações a serem feitas diariamente, esta situação acontece com grande facilidade.

Consideremos a figura seguinte, em que temos dois programadores, o Hugo e a Alexandra, e em que cada um deles decide editar o mesmo ficheiro do respositório ao mesmo tempo. Se o Hugo guardar as suas alterações no repositório primeiro, então é possível que (passados uns minutos) a Alexandra possa acidentalmente passar por cima das alterações do Hugo na sua própria cópia do ficheiro. Embora as alterações feitas pelo Hugo não sejam perdidas (porque o sistema de versões lembra-se de cada alteração feita), estas não estarão presentes na nova versão do ficheiro criada pela Alexandra, pois ela nunca viu as alterações do Hugo. O trabalho do Hugo acaba por ser perdido efectivamente – ou pelo menos não aparece na última versão do ficheiro – e portanto o sistema de controlo de versões tem de evitar este tipo de problemas.

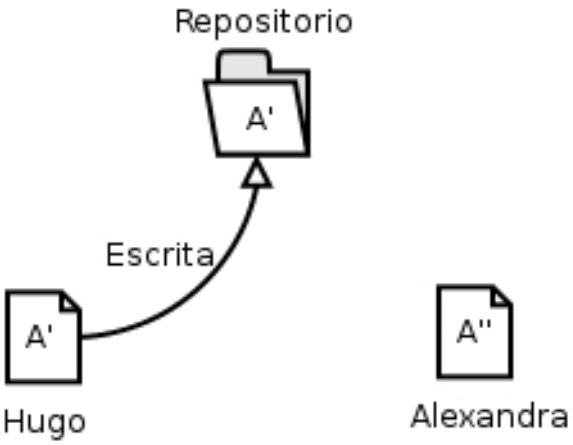
Os dois utilizadores lem o mesmo ficheiro



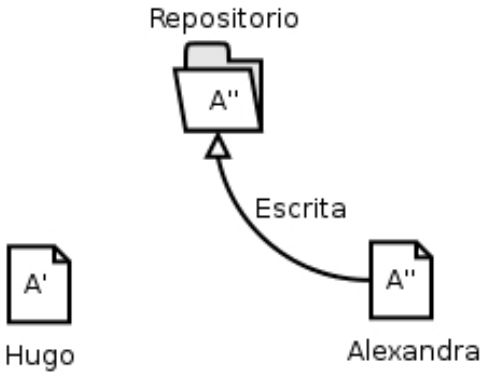
Ambos começam a editar as suas copias



O Hugo publica a sua versao de A primeiro



A Alexandra reescreve a versao do Hugo acidentalmente



A solução Bloqueia--Modifica--Desbloqueia (Lock--Modify--Unlock)

Muitos sistemas de controlo de versões (como por exemplo o SourceSafe da Microsoft) usam o modelo de bloqueia--modifica--desbloqueia para evitar o problema de haver mais do que um utilizador a modificar o mesmo ficheiro no repositório. Neste modelo, o repositório permite que só uma pessoa possa modificar um ficheiro em determinada altura. Esta política é gerida através de bloqueios (locks). O Hugo tem que bloquear um ficheiro antes de começar a fazer alterações nele. Se o Hugo bloquear um ficheiro, então a Alexandra não pode bloqueá-lo, e portanto não pode fazer nenhuma modificação nesse ficheiro. O que ela pode apenas fazer é ler o ficheiro, e esperar que o Hugo acabe as suas alterações e desbloqueie o ficheiro. Após o Hugo desbloquear o ficheiro, a Alexandra pode por sua vez bloquear e editar o mesmo ficheiro.

O problema com este modelo é que é muito restritivo, e muitas vezes torna--se uma dor de cabeça para os utilizadores, porque:

• **Bloquear pode causar problemas de administração**. Por vezes o Hugo pode bloquear um ficheiro e depois esquecer--se dele. Entretanto, e porque a Alexandra está à espera para editar o ficheiro, ela fica de mãos atadas à espera que o Hugo faça as alterações ao ficheiro. E é neste momento que o Hugo vai de férias. Perante este cenário, a Alexandra precisa de falar com um administrador para poder desbloquear o ficheiro do Hugo. Esta situação acaba por causar um atraso desnecessário.

 Bloquear pode causar serialização desnecessária. E se o Hugo estiver a editar o início de um ficheiro, e a Alexandra quiser editar apenas o fim desse mesmo ficheiro? Estas alterações que o Hugo e a Alexandra fizerem, não se sobrepõem e eles poderiam estar facilmente a editar o ficheiro ao mesmo tempo sem causar conflitos no ficheiro, assumindo que as alterações que os dois fizerem sejam propriamente conduzidas. Não há assim necessidade em haver turnos no bloqueio do ficheiro.

 Bloquear pode criar uma noção errada de segurança. Assumindo que o Hugo bloqueia e edita o ficheiro A, enquanto que a Alexandra bloqueia e edita ao mesmo tempo o ficheiro B. Mas suponhamos que os ficheiros A e B dependam um do outro, e que mudanças feitas nos dois ficheiros sejam semanticamente incompatíveis. A partir deste momento, os ficheiros A e B não funcionam mais correctamente. O sistema de bloqueios não consegue resolver este problema – e apesar disso forneceu uma noção errada de que havia segurança no manuseio dos ficheiros. É fácil de imaginar para o Hugo e a Alexandra que ao bloquearem um ficheiro, cada um deles está a começar uma tarefa independente e segura, e que não seja dada prioridade a uma discussão, antes de tudo, sobre as futuras mudanças incompatíveis que possam causar no repositório.

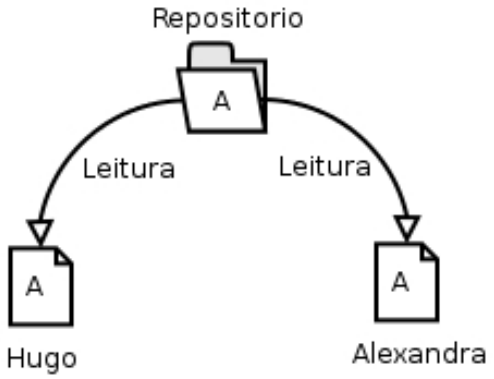
A solução Copia--Modifica--Unifica (Copy--Modify--Merge)

O Subversion, o CVS, e outros sistemas de controlo de versões usam o modelo copia--modifica--unifica como uma alternativa ao bloqueio. Neste modelo, cada utilizador contacta o servidor para aceder ao repositório, e cria uma cópia de toda a estrutura de ficheiros do projecto. Os utilizadores podem então trabalhar em paralelo, modificando as suas cópias privadas. Finalmente, as cópias privadas são unificadas numa só versão final. O sistema de controlo de versões ajuda muitas vezes nesta unificação, mas em último recurso, o utilizador é responsável por fazer com que a unificação ocorra correctamente sem conflitos.

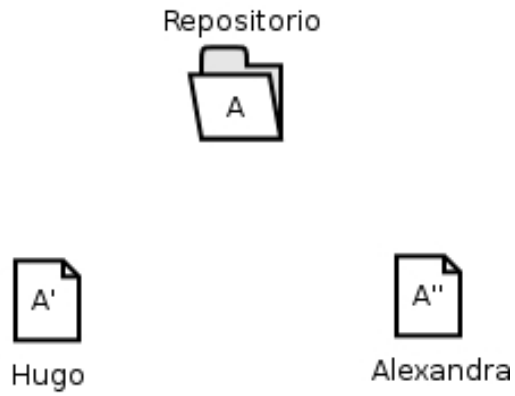
Eis um exemplo. Suponhamos que o Hugo e a Alexandra criam as suas próprias cópias de trabalho do mesmo

ficheiro, copiadas do repositório central. Trabalham concorrentemente (ao mesmo tempo, em paralelo) e fazem alterações ao mesmo ficheiro A nas suas cópias locais. A Alexandra guarda as suas alterações do ficheiro A no repositório antes do Hugo. Quando o Hugo tenta guardar as suas alterações mais tarde, o repositório informa--o que o ficheiro A dele está desactualizado. Noutras palavras, o ficheiro A no repositório foi alterado desde a última cópia que o Hugo fez do respositório. Neste ponto, o Hugo pede então ao cliente de Subversion para unificar quaisquer alterações que tenham havido no repositório com a sua cópia local do ficheiro A, guardando o resultado na sua cópia local. O mais provável é que as alterações da Alexandra não se sobreponham às do Hugo; então, desde que o Hugo tenha os dois conjuntos de alterações integrados, ele grava a sua cópia local de novo para o repositório. As figuras seguintes ilustram esta situação.

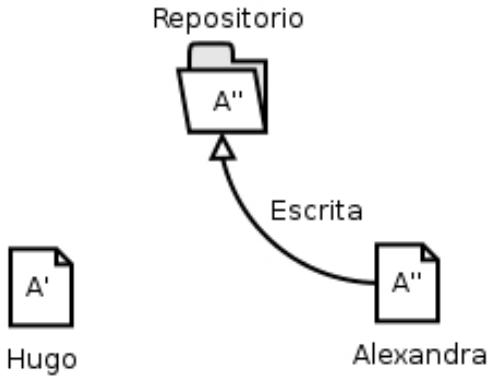
Os dois utilizadores copiam o mesmo ficheiro



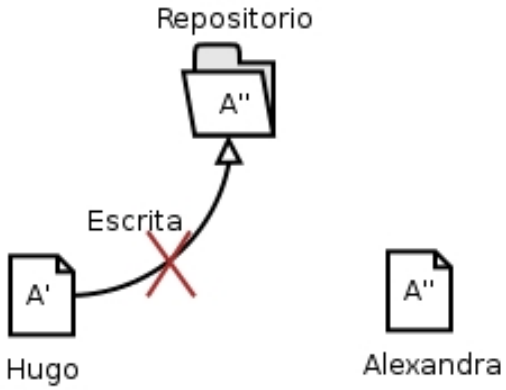
Ambos começam a editar as suas copias



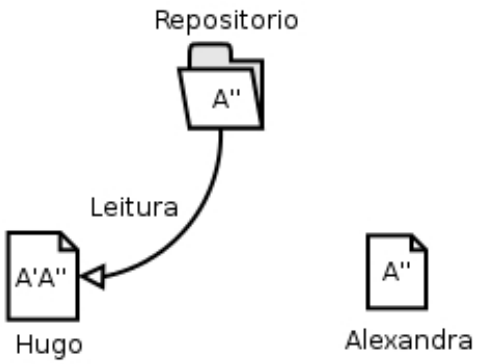
A Alexandra publica primeiro a sua versao



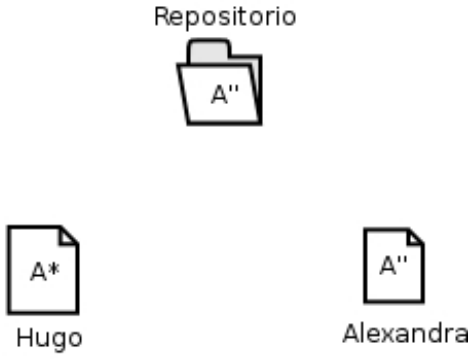
O Hugo apanha um erro de ficheiro "fora-de-prazo"

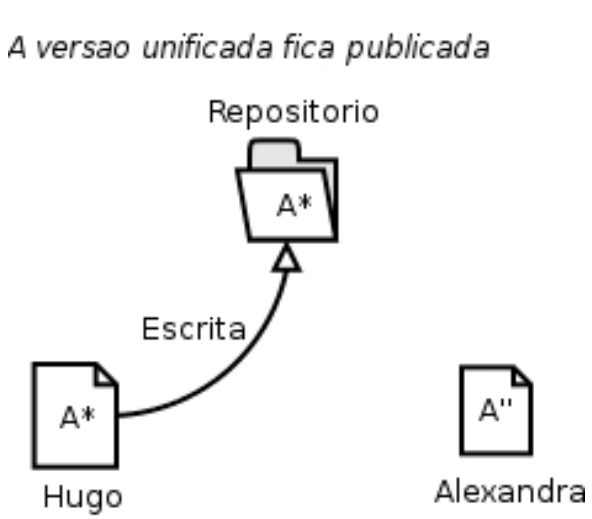


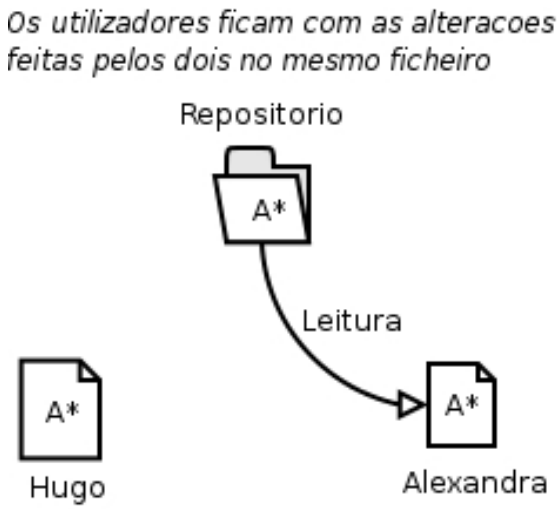
O Hugo compara a ultima versao com a sua copia



Uma nova versao unificada fica criada







Mas e se as alterações da Alexandra se sobrepõem às do Hugo? Esta situação é designada de conflito, e normalmente não é difícil de resolver. Quando o Hugo pergunta ao cliente de Subversion para unificar as últimas alterações do repositório com a sua cópia de trabalho, a sua cópia do ficheiro A é alterada para o estado de conflito através de uma flag: ele vai poder ver os dois conjuntos de alterações e poder escolher manualmente entre eles. Note--se que o software de controlo de versões não pode por si só resolver os conflitos automaticamente; só os utilizadores é que são capazes de entender e fazer as escolhas certas. A partir do momento em que o Hugo tenha resolvido manualmente os conflitos – após uma conversa com a Alexandra, por exemplo – ele pode guardar com segurança o ficheiro unificado de novo no repositório.

O modelo de copia--modifica--unifica pode parecer um pouco caótico, mas na realidade até é bastante simples de usar. Os utilizadores podem trabalhar em paralelo, sem nunca ter que esperar uns pelos outros. Quando se trabalha nos mesmos ficheiros, é interessante verificar que a maioria das alterações que são concorrentes (feitas ao mesmo tempo) não se sobrepõem; os conflitos são pouco frequentes. E o tempo que demora a resolver um conflito é muito menor do que o tempo gasto com um sistema de bloqueios. No centro da questão, tudo converge para um só factor importante: a comunicação entre utilizadores.

Quando os utilizadores comunicam mal, os conflitos aumentam sejam eles sintácticos ou semânticos. Não há nenhum sistema que force a que os utilizadores comuniquem na perfeição, e não há nenhum sistema que consiga detectar conflitos semânticos. Portanto, não adianta pensar que um sistema de bloqueios irá de alguma forma prevenir os conflitos; na prática, os bloqueios inibem a produtividade mais do que qualquer outro modelo de controlo de versões. Apesar disto, pode haver situações que seja de facto útil haver bloqueio de ficheiros. O modelo de copia--modifica--unifica é baseado na ideia de que os ficheiros alterados são passíveis de ser unificados contextualmente: ou seja, a maioria dos ficheiros no repositório são ficheiros de linhas de texto (tal como o código--fonte de um programa). Mas para ficheiros binários, como ficheiros de som ou ficheiros executáveis, é muitas vezes impossível unificar as alterações que geram conflitos. Nestas situações é realmente necessário que os utilizadores tomem a sua vez na alteração do ficheiro. Sem um acesso serializado, alguém acaba sempre por perder tempo com alterações que acabam por ser excluídas. Apesar de o CVS e o Subversion serem essencialmente sistemas do modelo copia--modifica--unifica, ambos reconhecem a necessidade de bloquear um ficheiro ocasionalmente e portanto fornecem mecanismos para o fazer. O bloqueio de ficheiros é suportado pelo Subversion usando a propriedade svn:needs-lock ao nível dos ficheiros.

Revisões

O conceito de Revisões

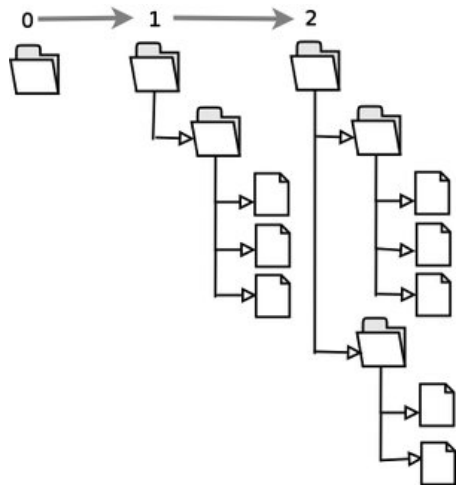
Um svn commit mostra as alterações a qualquer número de ficheiros e directorias numa só transacção atómica. Na cópia de trabalho pessoal de cada utilizador, podem-se alterar os conteúdos dos ficheiros, criar, apagar, renomear e copiar ficheiros e directorias, e só por fim fazer um commit a todo o conjunto de alterações englobando assim todas as alterações como se se tratasse de um só bloco de alterações.

No repositório, cada commit é tratado como uma operação atómica: ou todos os commits aos ficheiros acontecem de uma só vez ou nenhum deles acontece. O Subversion tenta manter sempre esta atomicidade para que não aconteçam crashes, problemas de rede ou outro tipo de situações que possam prejudicar o software desenvolvido e o seu uso correcto.

De cada vez que o repositório aceita um commit, isto cria um novo estado para a árvore do sistema de ficheiros,

chamado de revisão. A cada revisão é atribuído um número natural (1, 2, 3, ..., n) único e maior uma unidade do que o número de revisão anterior (incrementa sempre 1 unidade). A revisão inicial de um repositório acabado de criar tem o número de revisão 0, e consiste apenas numa directoria-raíz vazia.

A figura seguinte ilustra uma forma simples de vermos um repositório. Imagine-se um array de números de revisão, começando pelo 0, e aumentando da esquerda para a direita. Cada número de revisão tem uma árvore de ficheiros associada a essa revisão, e cada número de revisão é assim uma maneira de vermos como está um repositório após um commit, como se fosse uma fotografia dos ficheiros do repositório.



Números de Revisão Globais

Os números de revisão do Subversion aplicam-se a árvores inteiras de ficheiros, e não a ficheiros individuais. Cada número de revisão escolhe uma árvore inteira de ficheiros, que é um estado particular do repositório após algumas alterações terem sido aplicadas com o comando commit. Outra forma de se pensar é que a revisão N representa o estado dos ficheiros do repositório após o N-ésimo commit. Quando se fala, por exemplo, na revisão 7 de um ficheiro ficheiro.c está-se a referir ao ficheiro.c como este aparece na revisão 7. Note-se que em geral, as revisões N e M de um ficheiro podem não diferir: a revisão é feita globalmente à árvore do sistema de ficheiros e não aos ficheiros individuais; por isso, se um utilizador alterar alguns ficheiros e fizer um commit, os ficheiros não alterados passam também a ter um número de revisão superior apesar de não terem sido alterados por esse utilizador. O CVS usa a revisão por ficheiros individuais ao contrário do Subversion e é normal haver alguma dificuldade em usar o Subversion para utilizadores que estão habituados com o CVS.

É importante notar que cópias de trabalho nem sempre correspondem a uma única revisão do repositório; estas podem conter ficheiros de diferentes revisões. Por exemplo, suponha que se faz checkout de uma cópia de trabalho de um repositório cuja revisão mais recente é a 4:

```
ProjX/Makefile:4
       ficheiroA.c:4
       configure:4
       ...
```

Neste momento, esta cópia de trabalho corresponde exactamente à revisão 4 do repositório. No entanto, suponha que se faz uma alteração ao ficheiroA.c, e que se faz commit a essa alteração. Assumindo que não foram feitos mais commits, vai ser criada a revisão 5 do repositório, e a cópia de trabalho vai parecer-se com isto:

```
ProjX/Makefile:4
       ficheiroA.c:5
       configure:4
       ...
```

Suponha que, a partir deste momento, a Alexandra faz commit a uma alteração no ficheiro ficheiroA.c, criando assim a revisão 6. Se se fizer svn update para colocar a cópia de trabalho actualizada, então teremos:

```
ProjX/Makefile:6
       ficheiroA.c:6
       configure:6
       ...
```

A alteração que a Alexandra fez no ficheiroA.c vai aparecer na cópia de trabalho, e essa mudança vai também estar reflectida nos outros ficheiros que não foram alterados. Neste exemplo, o texto da Makefile é o mesmo para as revisões 4, 5 e 6, mas no entanto, o Subversion marca na cópia de trabalho a revisão 6 para a Makefile indicando assim que a revisão deste ficheiro ainda é a mais actual. Por isso, após se fazer um update à cópia de trabalho, isso corresponderá exactamente a uma única revisão no repositório.

Ter revisões diferentes e misturadas num repositório é normal

De cada vez que um utilizador faz svn commit, a sua cópia de trabalho acaba por ficar com uma mistura de revisões. As alterações a que se fez um commit são marcadas com um

número de revisão superior a tudo o resto. Após vários commits (sem updates pelo meio) a cópia de trabalho desse utilizador tem uma mistura de revisões diferentes. Mesmo que só haja um utilizador a alterar vários ficheiros, ele vai notar nesta característica do SVN. Para se ver a mistura de revisões existentes numa cópia de trabalho pode-se usar o comando svn status –verbose.

Frequentemente, novos utilizadores não estão a par de que a sua cópia de trabalho tem diferentes revisões. Isto pode ser confuso porque muitos clientes SVN são sensíveis à revisão de trabalho de um ficheiro que estejam a examinar. Por exemplo, o comando svn log é usado para mostrar um histórico das alterações a um ficheiro ou directoria, mas se o número de revisão desse ficheiro ou directoria for bastante antigo (possivelmente porque há já muito tempo que o utilizador não faz um svn update), então o histórico da versão mais antiga é que será mostrado ao utilizador.

Revisões misturadas são úteis

Se um projecto é suficientemente complexo, por vezes é útil fazer um retrocesso a algumas porções da cópia de trabalho para uma revisão mais antiga. Às vezes é útil testar uma versão mais antiga de um sub-módulo dentro de uma sub-directoria, ou talvez seja necessário saber quando um bug foi criado num ficheiro específico. Este é o aspecto de máquina do tempo de um sistema de controlo de versões - a característica que permite a um utilizador mover qualquer porção da cópia de trabalho para uma revisão mais antiga ou mais recente.

Limitações das revisões misturadas

No entanto, o uso de revisões diferentes e misturadas numa cópia de trabalho tem limitações à sua flexibilidade.

Em primeiro lugar, não se pode fazer commit quando se apaga um ficheiro ou uma directoria que não esteja com a revisão mais actual. Se uma versão mais recente do ficheiro existe no repositório, a tentativa de o apagar vai ser rejeitada, para prevenir que se apaguem acidentalmente alterações que o utilizador ainda não teve conhecimento.

Por fim, não se pode fazer commit numa alteração às propriedades de uma directoria (metadata) sem que esta esteja totalmente actualizada com a última revisão. A revisão de uma directoria de trabalho define um conjunto de entradas e de propriedades, e portanto, fazer commit a uma alteração às propriedades de uma directoria fora-do-prazo pode destruir propriedades que o utilizador ainda não saiba que existem.

Revisões: Números, Palavras-chave e Datas

Antes de se apresentar os exemplos de uso do SVN, convém reter mais algumas noções sobre os números de revisão, nomeadamente como se identificam as revisões num repositório.

As revisões são especificadas usando o switch - -revision (ou -r) seguido do número de revisão que se pretende (svn –r NUM-REV), ou especificando um intervalo separando duas revisões por dois pontos (svn –r NUM-REV1:NUM-REV2). Posteriormente, o Subversion identificar estas revisões pelo número, pela data ou por uma palavra-chave.

Números de Revisão

Quando se cria um repositório SVN, este começa a sua vida útil com a revisão zero e cada commit sucessivo aumenta o número de revisão em uma unidade. Após o commit ter terminado, o cliente de SVN informa o novo número de revisão:

```
$ svn commit --message "Corrigi o bug
XPTO!!"
Sending        ficheiroA.c
Transmitting file data .
Committed revision 12.
```

Se a qualquer momento no futuro se quiser referenciar esta revisão, pode-se referenciá-la como a revisão 12.

O cliente de SVN percebe um número de palavras-chave relacionadas com as revisões. Estas palavras-chave podem ser usadas em vez dos números de revisão com o switch –revision, e são depois resolvidas de novo para números específicos de revisão pelo Subversion, sendo elas as seguintes:

- HEAD: a última (mais recente) revisão no repositório;
- BASE: o número de revisão de um item na cópia de trabalho. Se o item foi modificado localmente, a versão BASE refere-se ao item sem essas modificação locais;
- COMMITTED: a mais recente revisão antes, ou igual a BASE, de um item ter sido alterado;
- PREV: a revisão imediatamente antes da última revisão na qual um item foi alterado (tecnicamente é igual a COMMITTED - 1).

Nota: as palavras-chave PREV, BASE e COMMITTED pode ser usadas para referenciar caminhos (paths) locais, mas não URLs.

Eis alguns exemplos do uso de palavras-chave de revisão:

```
$ svn diff --revision PREV:COMMITTED
ficheiroA.c
```

Mostra a última alteração feita ao ficheiroA.c (último commit)

```
$ svn log --revision HEAD
```

Mostra o log do último commit feito ao repositório

```
$ svn diff --revision HEAD
```

Compara o ficheiro de trabalho (com eventuais alterações locais) com a última versão no repositório

```
$ svn diff --revision BASE:HEAD
ficheiroA.c
```

Compara a versão do ficheiroA.c (sem alterações locais feitas pelo utilizador) com a última versão no repositório

```
$ svn log --revision BASE:HEAD
```

Mostra todos os logs dos commits feitos desde a última vez que o utilizador actualizou a cópia de trabalho local

```
$ svn update --revision PREV
ficheiroA.c
```

Retrocede a última alteração feita ao ficheiroA.c (a revisão de trabalho do ficheiroA.c é decrementada)

Estas palavras-chave permitem realizar várias operações importantes e rapidamente sem estar a ver especificamente quais os números de revisão que interessam da cópia de trabalho ou ainda qual é o último número de revisão da cópia de trabalho local.

Datas de Revisão

Quando se especifica um número de revisão ou uma palavra-chave de revisão, pode-se também especificar uma data dentro de { }. Pode-se ainda aceder a um intervalo de alterações no repositório usando em conjunto datas e números de revisão.

Eis alguns exemplos dos formatos de datas que o Subversion aceita. Não esquecer de usar { } à volta das datas.

```
$ svn checkout --revision {2006-10-25}
$ svn checkout --revision {15:30}
$ svn checkout --revision
{15:30:00.200000}
$ svn checkout --revision {"2006-10-25
15:30"}
$ svn checkout --revision {"2006-10-25
15:30 +0230"}
$ svn checkout --revision {2006-10-
25T15:30}
$ svn checkout --revision {2006-10-
25T15:30Z}
$ svn checkout --revision {2006-10-
25T15:30-04:00}
$ svn checkout --revision
{20061025T1530}
$ svn checkout --revision
{20061025T1530Z}
$ svn checkout --revision
{20061025T1530-0500}
```

Quando se especifica uma data de revisão, o Subversion encontra a versão mais recente do repositório naquela data:

```
$ svn log --revision {2006-10-25}
------------------------------------
--------------------------------
r12 | ira | 2006-10-25 15:30:00 -0600
(Wed, 25 Oct 2006) | 14 lines
...
```

Se se especificar uma única data sem incluir uma hora do dia (por exemplo, 2006-10-25), pode pensar-se que o Subversion deverá dar a última revisão que aconteceu no dia 25 de Outubro. Em vez disso, obtém-se uma revisão de dia 24 ou ainda anterior a este dia. Recorde-se que o Subversion encontra a revisão mais recente do repositório até à data que se introduziu. Quando se deu a data de 2006-10-25, o Subversion assumiu a hora de 00:00:00, por isso ao pesquisar pela revisão mais recente, não devolveu nada no dia 25. Se se quiser incluir o dia 25 na pesquisa, pode-se especificar o dia 25 com o tempo ({2006-10-25 23:59}), ou

apenas ({2006-10-26}).

Pode-se também usar um intervalo de datas. O Subversion encontra todas as revisões entre as duas datas inclusivé:

```
$ svn log --revision {2006-10-
14}:{2006-10-25}
```

Podemos também misturar datas e números de revisão, como mencionado em cima:

```
$ svn log --revision {2006-10-14}:1453
```

Nota: o timestamp de uma revisão é guardado como uma propriedade da revisão - é uma propriedade sem versão, que pode ser modificada. Uma vez que podem ser alterados os timestamps (representando assim um momento cronológico que não é correcto), ou até mesmo removidos, o Subversion não conseguirá converter correctamente datas para números de revisão se estas propriedades forem modificadas.

Estado das cópias de trabalho

Para cada ficheiro numa directoria de trabalho, o Subversion grava duas informações na pasta administrativa, .svn/:

• a revisão em que se está a trabalhar num determinado ficheiro (a chamada revisão de trabalho desse ficheiro), e
• um timestamp que guarda a data da última actualização feita ao repositório para esse ficheiro local;

Dadas estas duas informações, e ao comunicar com o repositório, o Subversion pode indicar em que estado um ficheiro local se encontra (de um total de 4 estados possíveis):

• Sem modificações, e actualizado: o ficheiro não tem modificações na directoria de trabalho, e não houve commits feitos ao repositório desde a revisão de trabalho desse ficheiro. Um svn commit a este ficheiro não vai alterar nada no seu conteúdo e um svn update também não vai alterar nada à revisão do ficheiro;

• Com modificações locais, e actualizado: o ficheiro foi alterado na directoria de trabalho, mas não foi feito nenhum commit ao repositório desde a sua revisão anterior. Há alterações locais às quais não foi feito nenhum commit, logo um svn commit vai colocar as alterações no repositório de SVN, e um svn update não vai alterar nada ao conteúdo desse ficheiro;

• Sem modificações locais, e desactualizado: o ficheiro não teve alterações na directoria de trabalho, mas foi modificado por outro utilizador no repositório. O ficheiro deverá eventualmente ser actualizado, para o colocar com a última revisão de acordo com a revisão no repositório. Um svn commit ao ficheiro não fará nada, mas um svn update vai colocar esse ficheiro com as últimas modificações feitas na cópia de trabalho;

• Com modificações locais, e desactualizado: O ficheiro foi alterado pelo utilizador na cópia de trabalho local, e no repositório por outro utilizador. Um svn commit do ficheiro vai falhar com o erro de desactualizado. O ficheiro deverá ser actualizado primeiro; um svn update vai tentar fundir ou unificar as alterações públicas do repositório com as alterações locais. Se o Subversion não conseguir fazer essa fusão de uma forma automática, então cabe ao utilizador resolver manualmente este conflito.

O comando svn status mostra o estado de um item numa cópia de trabalho e é bastante útil para ver o estado dos ficheiros numa directoria de trabalho local.

Cópias de trabalho com revisões diferentes

O Subversion tenta sempre manter uma certa flexibilidade, tanto quanto possível, para permitir que se possa trabalhar numa cópia de trabalho com ficheiros e directorias com diferentes números de revisão. Infelizmente, isto pode ser um pouco confuso, mas facilmente se percebe a utilidade de ter números de revisão diferentes numa cópia de trabalho.

Uma das regras fundamentais do Subversion é que um push ao repositório não causa um pull, e vice-versa. Ou seja, quando se submetem alterações ao repositório (push) isso não significa que se vá receber as alterações feitas pelos outros utilizadores (pull). E se um utilizador estiver a fazer modificações que ainda não tenham sido passadas ao repositório com um commit, fazendo svn update vai unificar as últimas alterações do repositório com as alterações em curso deste utilizador, sem que o utilizador seja obrigado a fazer commit às alterações que estava a fazer.

O efeito principal desta regra é que uma cópia de trabalho tem que ter um trabalho extra em tomar nota de todas as revisões diferentes que há na cópia de trabalho, e de ser flexível com as diferentes revisões para permitir que existam localmente. O que também complica neste processo é o facto de as próprias directorias terem também versões.

Por exemplo, suponha que se tem uma cópia inteira de trabalho com a revisão 10. Edita-se o ficheiro xpto.html e faz-se em seguida um svn commit, que cria a revisao 16 no repositório. Após o commit ter sido bem sucedido, muitos utilizadores pensarão que a cópia de trabalho local está toda

com a revisão 16, o que não acontece! Qualquer número de alterações pode ter sido feito ao repositório entre as revisões 10 e 16. O cliente de SVN não sabe de nenhumas dessas alterações ao repositório, uma vez que o utilizador local ainda não fez nenhum svn update, e um svn commit não traz para a cópia de trabalho as alterações feitas aos outros ficheiros entretanto. Por outro lado, se um svn commit fizesse um download automático das novas alterações que estavam no repositório, então isso iria colocar a cópia de trabalho local na revisão 16 - mas aí estaríamos a quebrar a regra de que um push e um pull são tarefas separadas. Portanto, a única acção segura que o cliente de SVN tem que fazer é de colocar o ficheiro xpto.html com a revisão 16. O resto da cópia de trabalho fica com a revisão 10. Só depois de se fazer um svn update é que as restantes alterações vão ser descarregadas, e toda a cópia de trabalho passa a ficar marcada com a revisão 16.

Subversion em acção

Nesta secção vai-se passar da ideia à acção, exemplificando na prática o uso do Subversion.

Cópias de trabalho

Como foi explicado acima, uma cópia de trabalho para o Subversion é apenas uma directoria no sistema local de trabalho de cada utilizador, que contém uma colecção de ficheiros. Cada utilizador pode editar os ficheiros como quiser, e se forem ficheiros de código--fonte de uma qualquer linguagem de programação, o utilizador pode compilar o programa da maneira habitual com os editores ou IDE's que preferir. A cópia de trabalho é uma área de trabalho privada: o Subversion nunca vai incorporar nesta área de trabalho as alterações das outras pessoas, ou colocar as alterações feitas nessa área de trabalho visíveis aos outros utilizadores, até que se diga ao explicitamente ao Subversion para o fazer. É possível ainda ter várias cópias de trabalho do mesmo projecto.

Após terem sido feitas alterações aos ficheiros na cópia de trabalho pessoal de cada utilizador e de se verificar que estas funcionam correctamente, o Subversion fornece os comandos para se publicar as alterações para os outros utilizadores que trabalham no mesmo projecto (ao escrever para o repositório). Se outras pessoas publicarem as suas próprias alterações, o Subversion fornece comandos para unificar essas alterações na cópia de trabalho de cada um (ao ler do repositório).

Uma cópia de trabalho contém também alguns ficheiros extra, criados e mantidos pelo Subversion, para ajudar na execução dos comandos do Subversion. Em particular, cada directoria na cópia de trabalho contém uma subdirectoria chamada .svn, que é também conhecida por directoria administrativa da cópia de trabalho. Os ficheiros na directoria administrativa ajudam o Subversion a reconhecer que ficheiros contêm alterações por publicar, e que ficheiros estão fora--do--prazo (antigos) em relação ao trabalho feito pelos outros utilizadores.

Um repositório típico de Subversion contém frequentemente ficheiros (ou código--fonte) de vários projectos; normalmente, cada projecto é uma sub-directoria na árvore do sistema de ficheiros do repositório. Por isso, a cópia de trabalho de um utilizador corresponde geralmente a uma sub-árvore particular do repositório global.

Por exemplo, suponhamos que temos um repositório Subversion que contém dois projectos de software, ProjX e ProjY. Cada projecto existe na sua própria sub--directoria. Para ter uma cópia de trabalho, um utilizador tem de fazer check out a uma sub--árvore do repositório (o termo check out pode parecer que se está a fazer algum bloqueio ou a reservar recursos, mas na verdade não; simplesmente cria uma cópia privada do projecto na máquina local do utilizador). Por exemplo, se o utilizador fizer check out ao ProjX/, fica com uma cópia local deste projecto:

```
$ svn checkout
http://svn.exemplo.com/repositorio/Proj
X
A   ProjX/Makefile
A   ProjX/classe.c
A   ProjX/LEIAME.txt
...
Checked out revision 29.
$ ls -A ProjX
Makefile   classe.c   LEIAME.txt ...
.svn/
```

A lista que começa com A significa que o Subversion está a adicionar esses ficheiros à cópia de trabalho pessoal do utilizador. O utilizador possui agora uma cópia pessoal da directoria ProjX/ do repositório, com uma entrada adicional – .svn – que contém informação extra necessária ao Subversion, como mencionado acima.

• Não apagar as pastas .svn; estas pastas ocultas são a porta de entrada da comunicação cliente-servidor de SVN.

Supondo que um utilizador fazia alterações no ficheiro classe.c, a directoria .svn/ lembra-se da data original de modificação deste ficheiro e do conteúdo que tinha nessa data, e o Subversion pode assim indicar que o ficheiro foi alterado. No entanto, o Subversion não torna as alterações do utilizador públicas até que o utilizador o indique. Publicar as alterações introduzidas em determinado ficheiro para ficarem visíveis aos outros utilizadores é feito através do comando commit ao repositório:

```
$ svn commit classe.c
Sending   classe.c
Transmiting file data .
Committed revision 30.
```

Agora que as alterações à classe.c foram colocadas no repositório, qualquer utilizador que faça 'check out à cópia de trabalho do ProjX/ vai poder ver as alterações introduzidas na última versão do ficheiro classe.c. Suponhamos agora que a Alexandra fez check out à sua cópia de trabalho do ProjX/ ao mesmo tempo que o Hugo. Quando o Hugo faz commit às alterações do ficheiro classe.c, a cópia de trabalho da Alexandra fica inalterada; o Subversion apenas modifica uma cópia de trabalho a pedido do utilizador. Para colocar o seu projecto actualizado, a Alexandra faz um pedido ao Subversion para fazer update à sua cópia de trabalho, usando para isso o comando update. Isto vai incorporar as alterações do Hugo no ficheiro classe.c, bem como outras alterações feitas pelos outros utilizadores do projecto desde que a Alexandra fez check out ao repositório de Subversion.

```
$ pwd
/repositorio/ProjX/
$ ls -A
.svn/ Makefile classe.c LEIAME.txt ...
$ svn update
U   classe.c
Updated to revision 30.
```

A saída do comando svn update indica que o Subversion fez uma actualização aos conteúdos do ficheiro classe.c. Note--se que a Alexandra não precisou de especificar a que ficheiros fazer o update; o Subversion usa a informação da directoria .svn, e outras informações contidas no repositório, para decidir que ficheiros precisam de ser actualizados na cópia de trabalho da Alexandra.

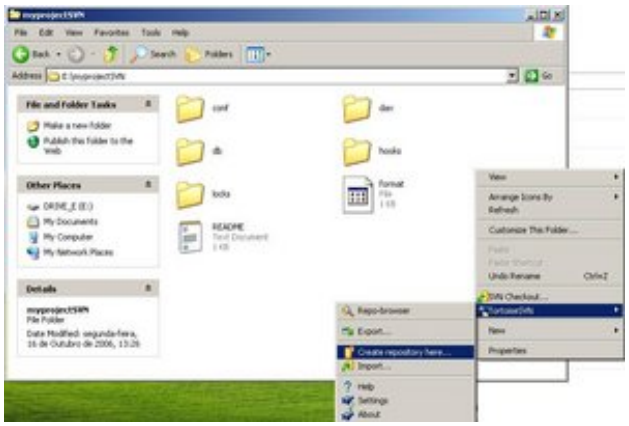
Começar a usar o Subversion

De seguida apresenta--se um exemplo de utilização do Subversion para que se consiga ter uma percepção mais abrangente das vantagens de utilização deste software de controlo de versões. O seguinte exemplo assume que já esteja instalado o svn – o cliente de linha de comandos do Subversion – e o svnadmin – a ferramenta administrativa – e que estejam prontos a usar. Assume também que se esteja a usar o Subversion 1.4 ou mais recente (executar svn - - version para verificar a versão).

O Subversion guarda todas versões do software num repositório central. Para começar, criar um novo repositório:

```
$   svnadmin   create
/usr/local/share/repositorio/
$ ls /usr/local/share/repositorio/
conf/   dav/   db/   format   hooks/
locks/  README.txt
```

Podemos também criar um repositório através do TortoiseSVN, começando por criar uma pasta e seleccionando a opção Create Repository here... com o botão direito do rato em cima da pasta para o repositório.



Este comando cria uma nova directoria (no exemplo acima, em /usr/local/share/repositorio) que contém um repositório de Subversion. Esta directoria contém para além de outros ficheiros, uma colecção de ficheiros de base de dados. Porém, não se acede aos ficheiros do software e às diferentes versões se explorarmos o repositório de Subversion acabado de criar. O Subversion não tem o conceito de "projecto". O repositório é apenas um sistema de ficheiros virtual com versões, uma árvore de ficheiros que pode conter qualquer tipo de ficheiro que se queira. Alguns programadores preferem guardar apenas um só projecto num repositório, enquanto que outros guardam vários projectos no repositório guardando cada um deles em directorias separadas. De qualquer forma, o repositório apenas guarda ficheiros e directorias, cabe aos programadores saber que directorias e ficheiros pertencem a determinado projecto. Assim, tudo o que realmente se está a falar quando se fala de repositórios de Subversion é de uma colecção de directorias e de ficheiros à qual o Subversion não conhece a que projecto pertence cada ficheiro ou directoria.

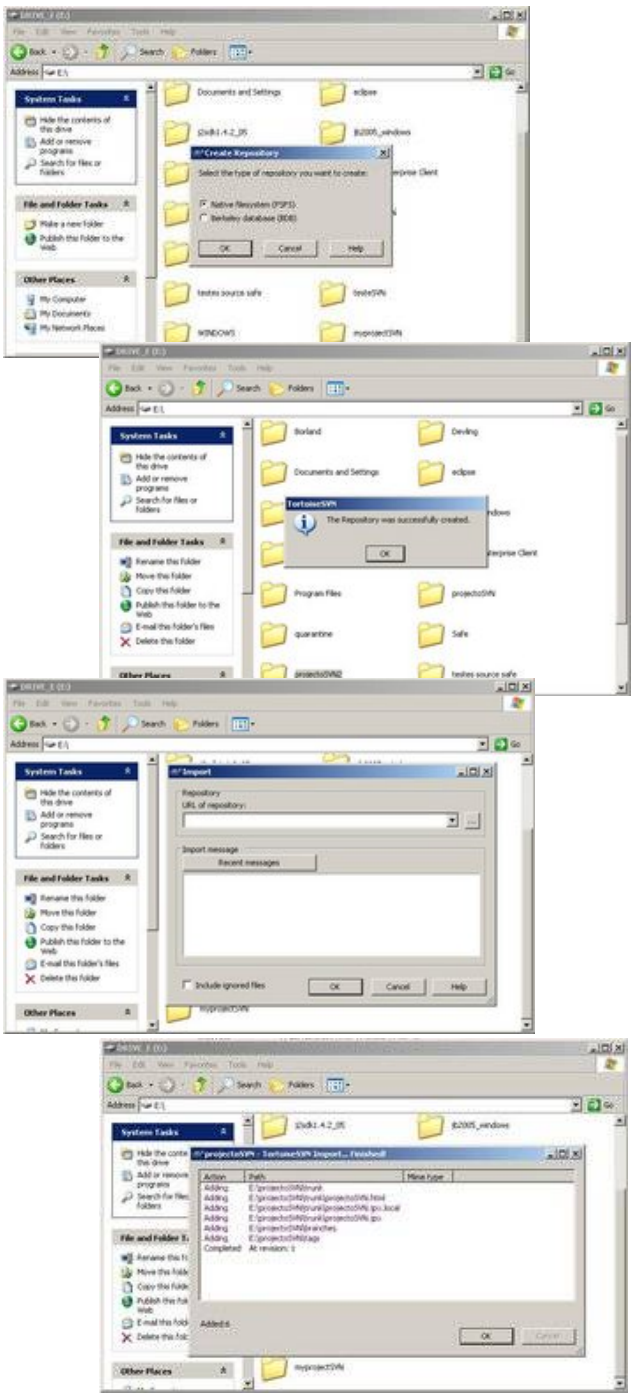
Supondo que o software a partir do qual se quer começar a depender do Subversion está numa pasta designada /usr/local/share/softwareXPTO/, começa--se por criar três directorias com os nomes branches, tags e trunk. A pasta trunk deverá conter todo o código de software e ficheiros dependentes, enquanto que as pastas branches e tags deverão estar vazias:

```
/usr/local/share/softwareXPTO/branches/
/usr/local/share/softwareXPTO/tags/
/usr/local/share/softwareXPTO/trunk/
                        ficheiroA.c
                        ficheiroB.h
                        ficheiroC.html
                        Makefile
                        configure
                        ...
```

Uma vez criadas estas sub-directorias, importa--se a pasta do software para o repositório criado com o comando svn import:

```
$ svn import
/usr/local/share/softwareXPTO/ \

file:///usr/local/share/repositorio/sof
twareXPTO \
-m "import inicial para o repositório"
Adding
/usr/local/share/softwareXPTO/branches
Adding
/usr/local/share/softwareXPTO/tags
Adding
/usr/local/share/softwareXPTO/trunk
Adding
/usr/local/share/softwareXPTO/trunk/fic
heiroA.c
Adding
/usr/local/share/softwareXPTO/trunk/fic
heiroB.h
Adding
/usr/local/share/softwareXPTO/trunk/fic
heiroC.html
Adding
/usr/local/share/softwareXPTO/trunk/Mak
efile
Adding
/usr/local/share/softwareXPTO/trunk/con
figure
...
Committed revision1.
```

Com o TortoiseSVN a operação é semelhante: depois de criar o repositório SVN, é só escolher a opção Import... a partir da directoria do software e escolher o repositório criado.

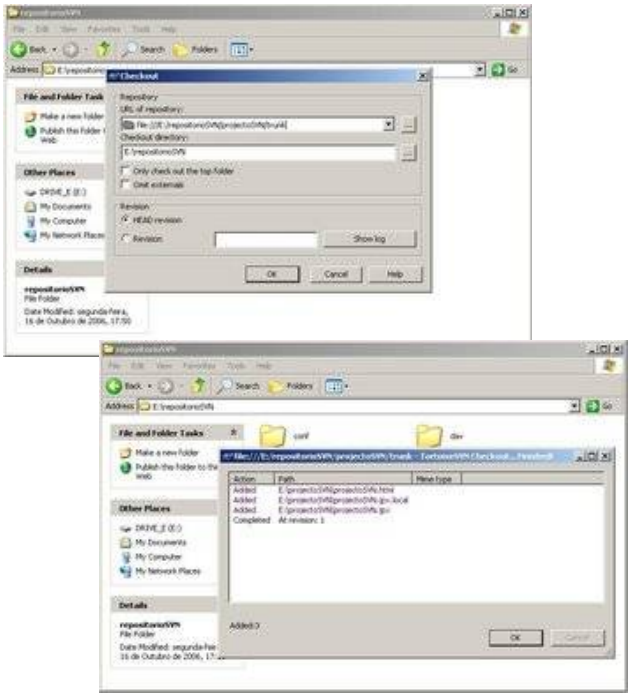


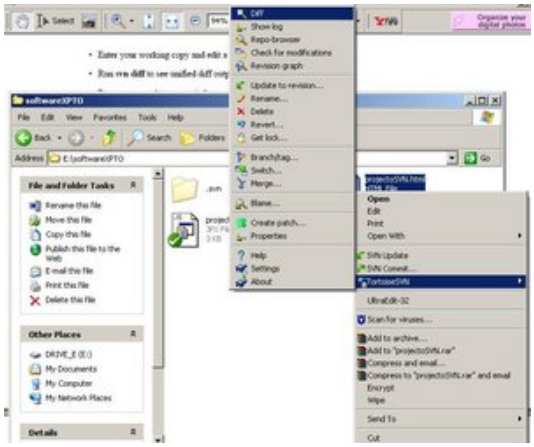
O repositório contém nesta altura a árvore de ficheiros pretendida. Como foi mencionado acima, não se acedem aos ficheiros do projecto de software olhando directamente para o conteúdo do repositório; o código fica armazenado numa base de dados dentro do repositório.

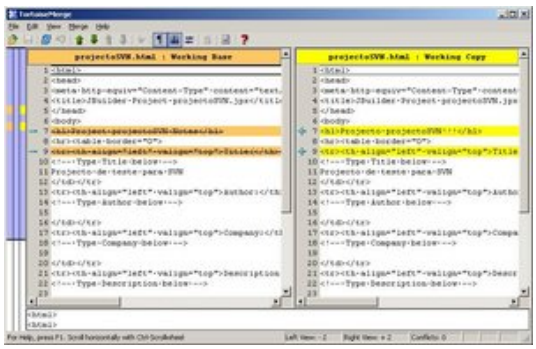
Para começar a manipular os dados no repositório, é necessário criar uma cópia de trabalho (working copy) dos dados, como se fosse uma área de trabalho privada. Para tal, pede--se ao Subversion para fazer um check-out da cópia de trabalho da directoria /usr/local/share/repositorio/software XPTO/trunk no repositório:

```
$ svn checkout
file:///usr/local/share/repositorio/sof
twareXPTO/trunk softwareXPTO
A    softwareXPTO/ficheiroA.c
A    softwareXPTO/ficheiroB.h
A    softwareXPTO/ficheiroC.html
A    softwareXPTO/Makefile
A    softwareXPTO/configure
...
Checked out revision 1.
```

Ou através do TortoiseSVN:







Após esta operação, temos uma cópia pessoal de parte do repositório numa directoria chamada softwareXPTO. Pode-se editar os ficheiros na cópia de trabalho e depois fazer commit a essas alterações de novo para o repositório.

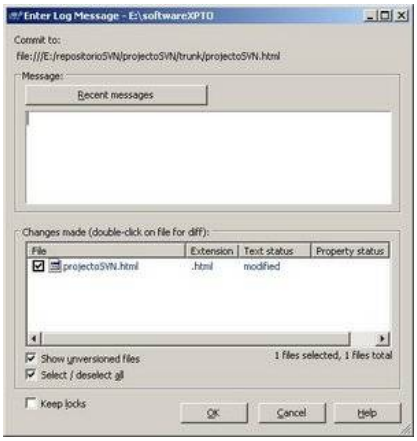
• Dentro da pasta da cópia de trabalho editar e alterar o conteúdo de um ficheiro;

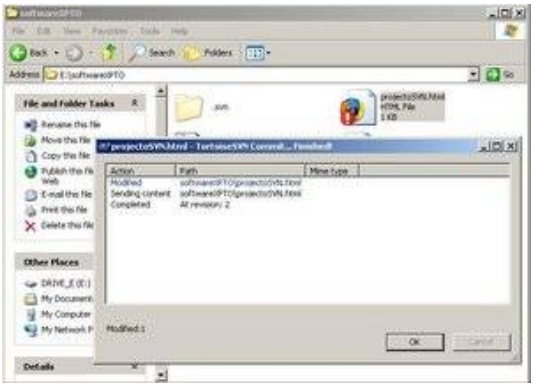
• Fazer svn diff para ver uma saída das diferenças unificadas das alterações feitas;

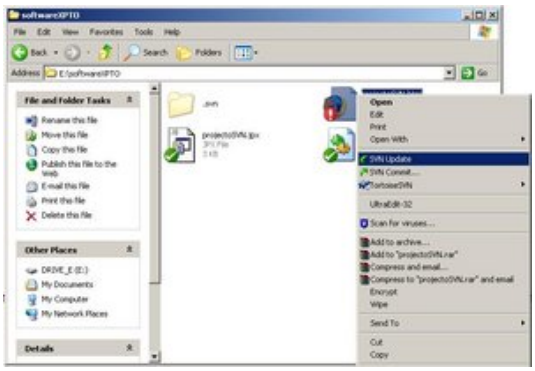
• Fazer svn commit para introduzir a nova versão dos ficheiros no repositório;

• Fazer svn update para colocar na cópia de trabalho a versão mais actual do repositório.

Através do TortoiseSVN:







A partir deste ponto pode tornar-se o repositório acessível a outros utilizadores e/ou programadores numa rede de computadores.

SOBRE O AUTOR

Miguel Rentes é licenciado em Engenharia Informática e de Computação pela FEUP desde 2005 e integra actualmente a equipa de Desenvolvimento da divisão de Inovação e Desenvolvimento/Gestão de Redes da Efacec Engenharia. Actualmente está a trabalhar na tese "Novas Metodologias de Controlo de Versões de Software" do Mestrado Integrado em Engenharia Informática e de Computação na FEUP. Fã do software livre e de tudo o que diga respeito à teoria da computação e segurança informática. Viciado em jogos de vídeo, livros e cinema.

miguel.rentes@portugal-a-programar.org

Miguel Rentes

Performance e Optimização MySQL

O MySQL tornou-se o **SGBD** (Sistema de gestão de base de dados) open source mais popular da actualidade, muito por culpa da sua boa performance, fiabilidade e facilidade de utilização.

Actualmente na versão 5.1 (última versão estável), o MySQL é usado por todo o mundo, seja em websites, ou sistemas críticos. È também utilizado em grandes empresas como a *Yahoo!*, *Google* e *Nokia*.

A sua portabilidade também é um factor de sucesso, pois funciona em mais de 20 plataformas incluindo Linux,Windows e OS/X, oferecendo uma importante flexibilidade ao programador. Também devido ao enorme número de WebHostings que o adoptaram, o MySQL tornou-se um dos SGBD's favoritos dos programadores web que pretendem construir um WebSite, procurando acima de tudo rapidez e performance, onde o downtime e outros problemas são intoleráveis, podendo levar a prejuízos consideráveis.

É importante referir que este artigo é orientado para programadores que já têm alguma experiência com o MySQL e pretendem ficar a conhecer a arquitectura de modo a poderem realizar queries com um custo computacional mais reduzido. O objectivo deste artigo passa por perceber o que se deve ou não fazer quando escrevemos uma query, para podermos avaliar o que é eficiente ou não de modo a podermos explorar as vantagens e evitando as fraquezas deste SGDB.

Arquitectura MySQL

A arquitectura do MySQL é um pouco diferente dos restantes SGBD's, possuindo particularidades como os motores de armazenamento (storage engines). Estes motores podem ser utilizados em run-time e é possível para tabelas específicas definir como a informação é armazenada e qual a sua performance, além de outras características que podemos considerar relevantes. Torna-se portanto importante perceber o que são e quais as razões que nos levam a optar por cada um dos motores.

Arquitectura Lógica

Como o objectivo é esclarecer e não confundir, o diagrama a seguir apresentado representa o essencial da arquitectura MySQL. A figura 1 é a imagem virtual que necessitamos de possuir para obter uma ideia da arquitectura lógica do MySQL.

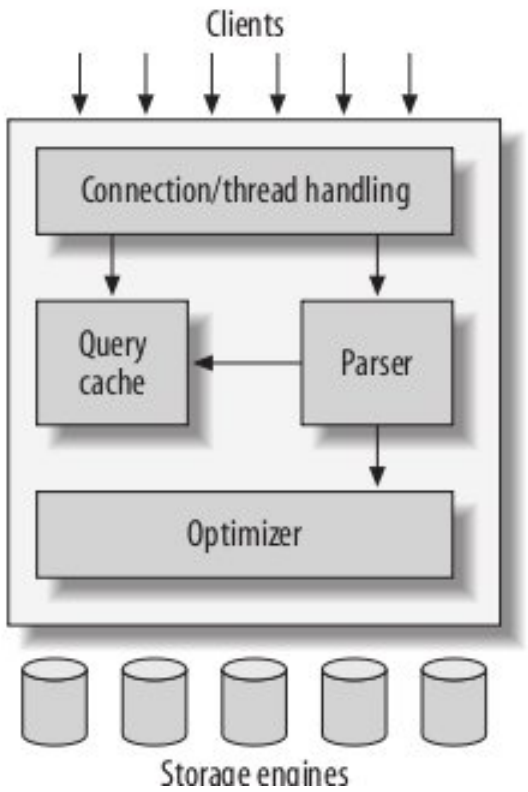


Figura 1 Arquitectura lógica MySQL

O nível mais alto do diagrama, que a título puramente indicativo vou chamar de nível 1, contém serviços que não são únicos ao MySQL, e que são baseados em cliente/servidor; é onde, por exemplo, são aceites as ligações ao SGBD e consequentemente efectuadas as autenticações.

O segundo nível é onde se situam as características e os pilares centrais do MySQL, incluindo o código para query parsing, análise, optimização, caching e funções (date,math,encryption por exemplo).

O terceiro nível contém os motores de armazenamento, que são responsáveis por armazenar e devolver toda a

informação contida na base de dados, possuindo cada um as suas vantagens e desvantagens.

O servidor de base de dados comunica com os motores através de uma API chamada **storage engine API** que esconde as diferenças dos motores de pesquisa e torna-os claramente transparentes para os níveis superiores do MySQL.

A API contém algumas funções de baixo nível para processar operações como *"begin transaction"* ou *"fetch the row that has primary key"*. Os vários motores não comunicam entre si, simplesmente respondem a pedidos do servidor.

Nota: A denominação de nivel 1,2 e 3 é puramente indicativa, de modo a que o leitor perceba que nível da imagem está a ser referido no texto.

Optimização e execução de queries

O MySQL analisa as queries para criar uma estrutura interna (em forma de árvore), e então aplica uma variedade de optimizações que incluem a reescrita da query, determinando a ordem de como as tabelas vão ser lidas, assim como escolhendo que índices vai utilizar.

Na fase de optimização não interessa qual foi o motor de base de dados que determinada tabela utiliza, mas o motor da base de dados afecta a forma como o servidor optimiza as queries.

O optimizador de queries pergunta ao motor de base de dados quais as suas capacidades, custo de determinadas operações, e estatísticas sobre a informação nas tabelas, de modo a poder maximizar a eficiência do processo.

Por exemplo, alguns motores de base de dados suportam tipos de índices que podem ser prestáveis para certo tipo de queries.

Antes mesmo de analisar a query, o servidor consulta o cache, no qual pode armazenar apenas as cláusulas SELECT, junto com os seus resultados. Se alguém executar uma query que já se encontra na cache, então o servidor não necessita de analisar, optimizar ou executar, pode simplesmente devolver o resultado previamente armazenado.

Motores de base de dados MySQL

O MySQL armazena cada base de dados (também chamada schema) como um subdirectório da directoria subjacente, ou seja o Sistema de Ficheiros. Quando criamos uma tabela, o MySQL armazena a definição da tabela num ficheiro com o nome da mesma e extensão **.frm**

Deste modo, quando é criado uma tabela chamada: *MinhaTabela*, é armazenado a definição da tabela em *MinhaTabela.frm*. Isto porque, o MySQL utiliza o *Sistema de Ficheiros* para armazenar os nomes da base de dados e a definição das tabelas, que são case sensitivity (faz distinção entre maiúsculas e minúsculas) e dependem da plataforma. Ou seja, no Windows os nomes das tabelas e das bases de dados são case insentitivity (não há distinção entre maiúsculas e minúsculas), e por isso "PESSOA" e "pessoa" nomeiam a mesma tabela. Já os sistemas Unix são case sensitivity, o que faz com que o exemplo anterior não seja válido, pois PESSOA é diferente de pessoa. Isto pode trazer problemas para quem gosta de trabalhar hoje no Windows e amanhã no Linux. Por isso sugiro que coloquem o nome de tabela e atributos sempre em minúsculas.

Cada motor de base de dados armazena a informação e os índices de forma distincta, mas o servidor apenas actua sobre a definição da tabela. Para determinar que motor de base de dados uma determinada tabela utiliza, podemos usar o comando: *SHOW TABLE STATUS*. Por exemplo, para examinar o motor de uma base de dados MySQL podemos fazer da seguinte forma:

```
SHOW TABLE STATUS LIKE 'user'

************ 1. linha ***************
Name: user
Engine: MyISAM
Row_format: Dynamic
Rows: 6
Avg_row_length: 59
Data_length: 356
Max_data_length: 4294967295
Index_length: 2048
Data_free: 0
Auto_increment: NULL
Create_time: 2002-01-24 18:07:17
Update_time: 2002-01-24 21:56:29
Check_time: NULL
Collation: utf8_bin
Checksum: NULL
Create_options:
Comment: Users and global privileges
1 row in set (0.00 sec)
```

O output mostra que é uma tabela com motor MyISAM (**Engine: MyISAM**) .

O motor MyISAM

O MyISAM é o motor por omissão do MySQL e oferece um bom equilíbrio entre performance e funcionalidades úteis como compressão de dados. Contudo o MyISAM não suporta transacções, controlo concorrencial ou locks a nível de linha.

Armazenamento

O *MyISAM* armazena cada tabela em dois ficheiros: um ficheiro de dados e um ficheiro de índice. Os dois ficheiros têm extensão *.MYD* e *.MYI*, respectivamente. O MyISAM é um formato independente de plataformas, ou seja, pode-se copiar os dados e os índices de um processador Intel para uma arquitectura PowerPc ou arquitectura Sun SPARC sem qualquer problema.

As tabelas do MyISAM podem conter registos estáticos e dinâmicos. Registos estáticos são registos que não sofrem variações em termos de tamanho, ou seja, os seus campos têm todos o mesmo tamanho (os do mesmo tipo de dados). Por exemplo, uma declaração do tipo char(8) é um campo estático com um máximo de 8 caracteres.

Já um registo dinâmico contém um tamanho variável. Se for declarado VARCHAR , indica que o campo não é de tamanho fixo e pode variar em termos de tamanho. O número de registos que o MyISAM suporta é limitado pelo espaço disponível em disco, e o tamanho máximo dos ficheiros é limitado pelo sistema operativo.

Resumidamente podemos afirmar que o MyISAM possui as seguintes vantagens:
- Simplicidade: É um motor fácil de entender e para o qual é fácil desenvolver aplicações, possuindo também ferramentas gratuítas muito boas, como o mysqkhitcopy. (script perl que utiliza LOCK TABLES, FLUSH TABLES e cp ou scp para fazer um backup rápido de um banco de dados.).
- Optimização: Existem sistemas optimizados especialmente para este motor, pois este é um motor que embora não apresente características que imponham a integridade referencial (como o uso do CASCADE), é um motor desenhado para um retorno eficiente dos dados.
- Utilização de recurso: é o motor que melhor aproveita os recursos do sistema, tornando-se por isso um excelente motor para servidores com capacidades limitadas.

Motor InnoDB

O motor *InnoDB* providencia uma transacção segura das tabelas. É incluído por omissão no *MySQL 5.0*, e possui capacidades de *commit*, *rollback* e recuperação de crashs. Possui também boas capacidades concorrênciais multi-utilizador, e boa performance.

Suporta *chaves estrangeiras* e *cascades* , reforçando a integridade dos dados. O motor InnoDb foi desenhado para processar grandes quantidades de informação com boa performance.

É utilizado em sites que requerem alta performance. Sites como o Slashdot.org corre InnoDB. Contudo o InnoDB é um motor complexo e não tão acessível como o MyISAM.

Se uma aplicação efectua poucas inserções e não possui acessos concorrenciais, então o motor MyISAM é o mais rápido; contudo, o InnoDB possui várias vantagens:
- Actualizações e inserções não bloqueiam a leitura
- Possibilita transacções
- Suporta crash recovery
- Suporta uma maior integridade dos dados

Motor MEMORY

Formalmente conhecido como *HEAP* engine , este motor é útil quando necessitamos de acesso rápido aos dados que não sofrem alterações e que não necessitem de ficar armazenados depois de um restart.

Toda a informação é armazenada em memoria (RAM, ou seja memoria volátil), portanto são queries que não necessitam de esperar por operações de I/O. Neste motor a estrutura de uma tabela persiste depois do restart, mas a informação não.

Este motor é útil:
- Para a "pesquisa" ou "mapeamento" de tabelas, como por exemplo uma tabela que mapeia códigos postais para indicar nomes de residências
- Para caching periódico de dados
- Para armazenar resultados temporários de análise de informação

Motor Archive

É usado para armazenar grandes quantidades de informação sem a utilização de índices. Na práctica é um motor simples para inserções rápidas e de boa compressão e dados.

Motor Federated

O motor Federated foi adicionado na versão 5.0.3 do MySQL e é um motor que armazena dados numa base de dados remota, contudo é ainda um motor pouco desenvolvido e possui problemas com queries agregadas, junções e outras operações básicas.

Motor NDB Cluster

Conhecido também como *NDB* é um motor usado pelo *MySQL Cluster* (versão especial do MySQL adaptada para sistemas distribuídos) para implementar tabelas que estão distribuídas por vários computadores.

É actualmente suportado por diversas distribuições Unix, e está previsto ser compatível com o Windows nas versões posteriores do Mysql Cluster. Este motor não é suportado pelas versões tradicionais do MySQL, como a 5.1.

Motor Falcon

O motor falcon é um motor demasiado recente, desenhado para servidores com múltiplos processadores 64 bits, mas também corre em sistemas mais modestos. O objectivo deste motor é correr totalmente as transacções em memória e assim tornar rollbacks e operações de recovery extremamente rápidas. Falcon é ainda um motor em desenvolvimento, de tal forma que ainda não é descrito no site do MySQL.

Existem outros motores não abordados neste artigo, como tal ainda muito ficou por falar. Mesmo em relação aos motores abordados, apenas foram descritas as características base de cada um.

Selecionar o motor correcto
Quando desenhamos uma aplicação baseada em MySQL, temos que decidir qual é o melhor motor de base de dados para enfrentar complicações futuras no projecto. É necessário analisar se o motor padrão (MyISAM) ofere-se todas as características que necessitamos como transacções ou acessos concorrenciais por exemplo.

Como é possível escolher um motor de base de dados por tabela, necessitamos de ter uma ideia clara de como cada tabela será usada. É necessário entender os requisitos da aplicação e o seu potencial de crescimento.

Com esta informação é possível começar a tomar boas opções sobre os motores que vão realizar a tarefa.

Contudo não é necessariamente uma boa ideia utilizar um motor de base de dados diferente para cada tabela. Se nos for possível afastar disso, mais simples se torna a tarefa
.
Pequeno sumário dos motores abordados:

Storage engine	MySQL version	Transactions	Lock granularity	Key applications	Counter-indications
MyISAM	All	No	Table with concurrent inserts	SELECT, INSERT, bulk loading	Mixedread/write workload
MyISAM Merge	All	No	Table with concurrent inserts	Segmented archiving, data warehousing	Many global lookups
Memory (HEAP)	All	No	Table	Intermediate calculations, static lookup data	Large datasets, persistent storage
InnoDB	All	Yes	Row-level with MVCC	Transactional processing	None
Falcon	6.0	Yes	Row-level with MVCC	Transactional processing	None
Archive	4.1	Yes	Row-level with MVCC	Logging, aggregate analysis	Random access needs, updates,
Federated	5.0	N/A	N/A	Distributed data sources	Any but the intended use

Optimização de queries

A principal causa para a fraca performance de uma query é esta estar a trabalhar com demasiada informação. No geral a maioria das queries com fraco desempenho podem ser optimizadas acedendo a uma menor quantidade de dados. Podemos analisar uma querie com fraco desempenho em dois passos:

1. Descobrir se a aplicação com que acedemos á base de dados não está a **devolver** mais informação do que aquela que é necessária, ou seja, devolver demasiadas linhas e colunas.

2. Descobrir se o MySQL está a **analisar** mais linhas do que aquelas que necessita

Estamos a devolver mais informação do que aquela que necessitamos?

Muitas queries devolvem mais informação do que

necessitam, o que faz com que essa informação seja lixo. Este tipo de prática é um trabalho extra para o MySQL que consome memória e recursos de CPU.

Aqui ficam alguns erros típicos:
- Devolver mais campos do que é necessário é um dos principais erros quando executamos uma querie.
- Outro erro comum é assumir que o MySQL fornece resultados sobre a procura, em vez de calcular e devolver o resultado completo conjunto. Ou seja, são utilizadas técnicas como a declaração da cláusula *SELECT* que retorna muitas linhas, e em seguida retira-se apenas as linhas necessárias. Por exemplo o MySQL retorna 100 linhas e o utilizador apenas precisa de 10. Muitas vezes é assumido erradamente que o MySQL devolve aqueles 10 resultados e para de executar a consulta, mas o que realmente o MySQL faz é gerar um conjunto completo de resultados, e descarta aqueles que não são necessários. Uma solução é utilizar a cláusula LIMIT que limita o número de linhas devolvidas.

Exemplo:

Sintaxe da clausula LIMIT:

LIMIT <opcional> <limite>

• <opcional> -Opcionalmente especificar o numero de linhas a evitar no resultado

• <limit> - Especificar o limite máximo de linhas do resultado

```
SELECT p.nome,p. morada, p.telefone, p.bi
FROM pessoa p
ORDER BY p.bi DESC LIMIT 10
```

Devolve o nome, morada, telefone e BI da tabela pessoa , ordenando as linhas pelo numero de bi de forma descendente com o limite de 10 linhas.

```
SELECT p.nome,p. morada, p.telefone, p.bi
FROM pessoa p
ORDER BY p.bi LIMIT 0, 50
```

Faz o mesmo que a anterior mas devolve o intervalo de linhas especificado, ou seja, da linha 0 até á linha 50.

Devolver várias colunas com uma junção de várias tabelas:

Exemplo:
Se pretendermos devolver todos os actores que dão voz no filme Shrek não devemos elaborar a query da seguinte forma:

```
SELECT *
FROM actor
INNER JOIN filmes_actor USING(actor_id)
INNER JOIN filmes USING(film_id)
WHERE filmes.titulo= "Shrek" ;
```

Esta querie devolve todas as colunas de todas as 3 tabelas e terá uma performance péssima. Mas também podemos escrever a querie da seguinte forma:

```
SELECT actor_id
FROM actor
WHERE actor_id IN (SELECT actor_id
     FROM filmes
     WHERE titulo="Shrek");
```

Evitando as junções construimos uma query optimizada, mais simples e eficiente, que devolve menos colunas.

Devolução de todas as colunas

Sempre que utilizamos a clausula *SELECT* * ,devemos desconfiar da sua necessidade. Será mesmo necessário devolver todas as colunas? Provalmente não.

Devolver todas as colunas pode prejudicar optimizações baseadas em índices, para além que utilizam demasiadas operações de *I/O*, assim como grande consumo de memória e CPU. Alguns SGBD's não permitem o uso do *SELECT* * para prevenir os problemas acima mencionados.

Contudo por vezes é necessário devolver todos os campos, e nem sempre é mau utilizar esta técnica, dependendo claro do tipo de aplicação que desenvolvemos. Utilização de buffers ou outro tipo de cache pode fornecer algumas vantagens. Mas para uma aplicação usual é necessário ter cuidado com a sua utilização. O mais importante é ter noção das implicações ao utilizar determinadas operações.

Estará o MySQL a examinar demasiada informação?

Quanto temos a nossa querie que apenas devolve a informação que necessitamos, podemos examinar a querie para observar a informação que esta trata.

As principais métricas de custo computacional de uma querie são:
- Tempo de execução
- Número de linhas examinadas
- Número de linhas retornadas

Nenhuma destas métricas por si só é a forma perfeita de analisar o custo de uma querie, mas juntas podem reflectir aproximadamente qual é o custo computacional para

executar a query e traduzir qual a velocidade de execução da mesma.

O tempo de execução é afectado pelo motor utilizado, acesso concorrencial e claro o hardware utilizado.

Esta análise torna possível verificar o quanto eficiente as queries estão a encontrar a informação que necessitamos.

Idealmente, o número de linhas examinadas deve ser o número de linhas retornadas, mas é uma prática que é raramente possível.

Formas de restruturar queries

Quando optimizamos queries problemáticas, o nosso objectivo deve ser encontrar alternativas para retornar o resultado pretendido, sem claro possuir o mesmo resultado em termos de performance do MySQL.

Ocasionalmente é possível transformar queries em formas equivalentes e melhorar a performance.

Queries complexas *vs* Muitas queries

Uma questão importante no desenho de uma querie, é quando é preferível dividir uma querie complexa em pequenas queries. A abordagem tradicional é executar todo o trabalho com o menor número de queries possível. Esta abordagem é historicamente superior porque o custo das comunicações de rede é alto, assim como a sobrecarga da rede quando executamos muitas queries.

Mas, esta abordagem não se aplica ao MySQL, porque o MySQL foi desenhado para trabalhar de uma forma muito eficiente com as ligações de rede. O MySQL está também preparado para responder a pequenas queries muito rapidamente.

O MySQL pode correr mais de 50.000 queries simples por segundo e tratar de 2.000 queries por segundo de um simples invocador (utilizador) numa rede gigabit, portanto executar múltiplas queries não é necessariamente uma má prática.

Apesar disso a resposta de conecção continua lenta comparada com o número de linhas que o MySQL pode tratar por segundo internamente.

Assim continua a ser uma boa ideia utilizar um número minimo de queries possível, mas por vezes pode-se tornar uma querie muito mais eficiente decompondo-a e executando várias pequenas queries. A titulo de exemplo, quando tratamos de junções torna-se muito útil decompôr essa querie em queries mais simples e evitar o menor número de junções possiveis, pois esta operação é bastante custosa para o SGBD.

Também é possivel fazer optimizações recorrendo ás linguagens de programação.

Por exemplo decompondo uma clausula DELETE.Vamos assumir então o seguinte exemplo:

```
DELETE FROM mensagens
WHERE created < "12-11-2008";
```

Esta querie elimina todas as mensagens cuja data de criação é menor que 12-11-2008.

Poderiamos fazer algo parecido em pseudo-código:

```
rows_affected = 0
do {
    rows_affected = do_query(
    "DELETE FROM mensagens WHERE created <
12-11-2008)
} while rows_affected > 0
```

Que retorna o mesmo resultado.

Eliminar 10,000 linhas de uma só vez é tipicamente uma tarefa muito cara computacionalmente, e pode ser optimizada por múltiplas queries de modo a minimizar o impacto no servidor. Motores transicionais beneficiam da utilização de pequenas operações (InnoDB e MyISAM por exemplo).

Decomposição de junções

Muitos sites com alta performance utilizam uma decomposição de junções. Pode-se decompor uma junção correndo várias queries orientadas a uma tabela em vez de uma junção multi-tabela, e então executar uma junção sobre essas queries.

Por exemplo:

```
SELECT *
FROM utilizador
JOIN mensagens ON
remetente_id=utilizador_id
JOIN servidor_mail ON
servidor_mail_id=mensagem_servidor_id
WHERE utilizador_nome='bruno';
```

Esta querie faz uma junção da tabela utilizadores com a

tabela mensagens e tabela servidor_email, tendo como objectivo retornar todas as mensagens do utilizador bruno juntamente com o nome do servidor mail utilizado.

Seria útil, por exemplo, para sabermos quais as pessoas com o nome bruno que nos enviaram um mail.

Pode-se executar estas queries em alternativa:

```
SELECT *
FROM utilizador
WHERE utilizador_nome='bruno';
SELECT *
FROM mensagens
WHERE remetente_id=1234;
SELECT *
FROM servidor_mail
WHERE id in (123,456,567,9098,8904);
```

Pode parecer um desperdício este número de queries caso pretendamos por exemplo devolver o id do remetente com o nome bruno, mas este tipo de estrutura tem algumas vantagens:

- *Armazenamento em cache:* as aplicações podem beneficiar de um maiorrendimento das várias queries pois alguns dados necessários podem já estar em cache.
- Para tabelas baseadas no motor MyISAM este tipo de procedimentos são mais
eficientes, também devido ao controlo concorrencial, pois os locks entre tabelas são mais breves.
- Beneficia de uma base de dados distribuída, ou seja a possibilidade de possuir várias tabelas em servidores diferentes.
- Reduz a redundância de acessos

Optimização da função agregadora COUNT ()

A função *COUNT()* funciona de duas formas bem diferentes:
- Conta valores e linhas, e se for definido um nome de uma coluna ou outra expressão dentro do parêntesis, o *COUNT()* conta quantas vezes essa expressão contém o valor.
- Ou então conta simplesmente o número de linhas do resultado. Esta é a forma que o MySQL utiliza quando sabe que a expressão dentro do parêntesis não pode ser *NULL*.

O exemplo mais claro é quando é usado o *COUNT (*)*, que é uma forma especial de utilizar o *COUNT()*. È geralmente utilizado quando pretendemos contar o número de linhas de uma tabela devolvida.

Forma	Descrição
COUNT (*)	Devolve o número de linhas que resulta de um SELECT.
COUNT (coluna)	Devolve o número de ocorrências na coluna diferentes de NULL.
COUNT (DISTINCT coluna)	Devolve o número de ocorrências (sem repetições devido ao DISTINCT) na coluna.

Um dos erros mais frequentes é específicar o nome de uma coluna dentro de um parêntesis quando pretendemos contar o número de linhas.

Quando pretendemos saber o número de linhas de um resultado, devemos usar sempre COUNT (*). Isto comunica claramente a nossa intenção e evita fraco desempenho. Vejamos a seguinte querie:

Forma1:

```
SELECT COUNT(*)
FROM cidade
WHERE ID > 5;
```

Que conta todas as linhas da tabela cidade cujo id seja superior a 5.

Se utilizarmos o comando SHOW STATUS , observamos que pesquisou 4.079 linhas. Se negarmos as condições e subtrairmos o numero de cidades cujo ID's são menores ou iguais a 5 ao número total de cidades, podemos reduzir esta pesquisa a 5 linhas:

Forma 2:

```
SELECT (SELECT COUNT(*)
     FROM cidade) - COUNT(*)
FROM Cidade WHERE ID <= 5;
```

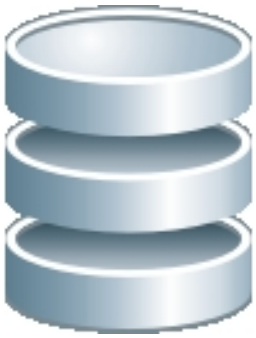
Ou seja subtraímos todas as cidades ás que têm o ID menor ou igual a 5, e assim obetermos o resultado pretendido.

Esta versão lê menos linhas porque a subquery evita uma comparação massiva de todos os registos com id maior que 5.

Forma utilizada	Linhas analisadas
Forma 1	Mais de 4000
Forma 2	5

Conclusão

Foram assim abordados os principais conceitos do MySQL, como algumas formas de optimizar queries, porém muito ficou dizer sobre a optimização de queries em ambiente MySQL, assim como da sua organização. Deixo uma análise mais profunda para uma futura continuação deste artigo, onde as optimizações podem ser levadas um pouco mais além.



Referências:

- Database Management Systems 3rd Edition -McGraw-Hill

- SQL – Luís Damas – FCA

- High Performance MySQL Optimization, Backups, Replication, Load Balancing & More - Jeremy Zawodny, Derek J. Balling

- http://www.mysql.com/

- http://www.mysqlperformanceblog.com/

SOBRE O AUTOR

Bruno Oliveira é estudante do 3º ano de Engenharia Informática da ESTGF, bem como Trabalhador Independente na área de programação Web. Tem especial interesse pela área de Base de Dados e programação orientada a objectos.

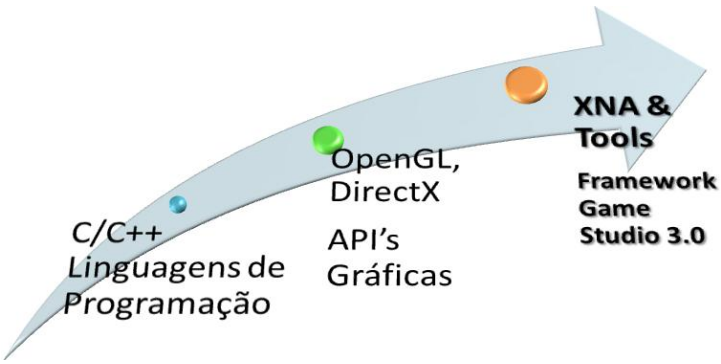
bruno.oliveira@portugal-a-programar.org

Bruno Oliveira

Desenvolvimento de jogos utilizando a plataforma XNA

Introdução

Foi há exactamente 50 anos atrás que se deu uma revolução, partiu pela mão de um senhor chamado William Higinbotham, um físico Americano. Usando a tecnologia disponível na sua altura, este desenvolveu um jogo, chamado Tennis for two que simulava uma partida de ténis entre dois jogadores usando um osciloscópio e um computador analógico. A ideia tinha o objectivo de evitar o aborrecimento dos visitantes do centro de Física onde ele trabalha. Esse objectivo foi cumprido e ultrapassado: deu origem à bem conhecida indústria dos videojogos. Essa indústria, que cresceu gradualmente com o desenvolvimento contínuo dos videojogos, não parou e é, hoje em dia, uma das mais reconhecidas a nível mundial, movimentando milhões de pessoas.



O futuro não pára e consequentemente a tecnologia também não: hoje em dia temos à nossa disponibilidade várias tecnologias e ferramentas que nos permitem produzir os nossos próprios jogos.

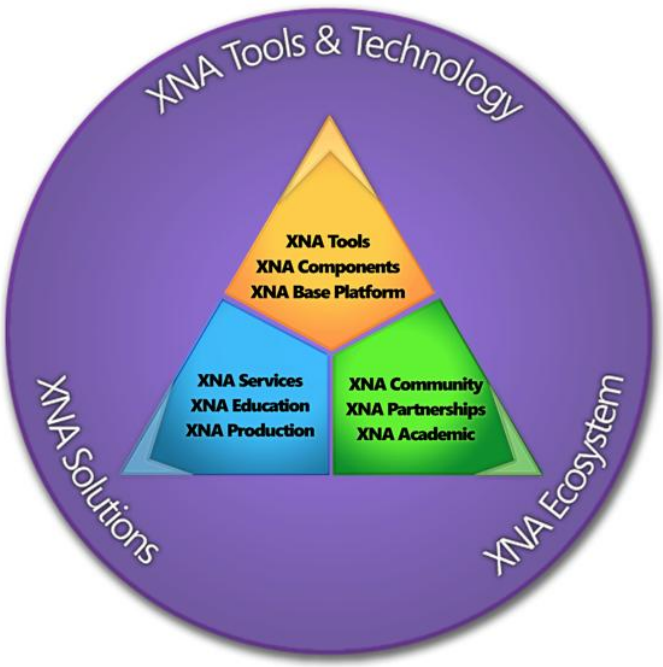
XNA diz-vos alguma coisa? Bem, a sigla XNA não tem um acrónimo associado, significa XNA's Not Acronymed. A plataforma XNA fornece-nos um pacote de ferramentas para o desenvolvimento de jogos, esta tecnologia está a criar a sua própria revolução nesta área. A tecnologia XNA foi construída com base num conjunto de API's (Aplication Programming Interfaces) desenvolvidas através da .NET Framework 3.5, que focam um desenvolvimento e design de jogos de forma bastante simples, tentando ao máximo através destas o uso de reutilização de código.

XNA para quem?

O XNA é para todas as pessoas que querem começar a dar os primeiros passos em desenvolvimento de jogos ou mesmo para utilizadores mais experientes que querem evoluir certos aspectos. O seu lema diz-nos isso mesmo: consumers into creators.

Quem já tentou ou começou a dar os primeiros passos em desenvolvimento de jogos sabe que a tecnologia disponível hoje em dia impõe-nos certas barreiras difíceis de ultrapassar para atingirmos os nossos objectivos; seja em termos de segurança, funcionamento em múltiplas plataformas ou mesmo de usabilidade. A tecnologia XNA oferece-nos soluções para podermos ultrapassar algumas restrições.

Com XNA podemos criar os nossos próprios jogos para a XBOX360, PC ou para o leitor multimédia Zune, programando em linguagem C#. A tecnologia XNA trabalha com um conjunto de parceiros, o que permite o uso de várias ferramentas para melhorar o desenvolvimento nossos jogos, como por exemplo Torque-X, Autodesk, Softimage, Turbo Squid e Allegorithmic.

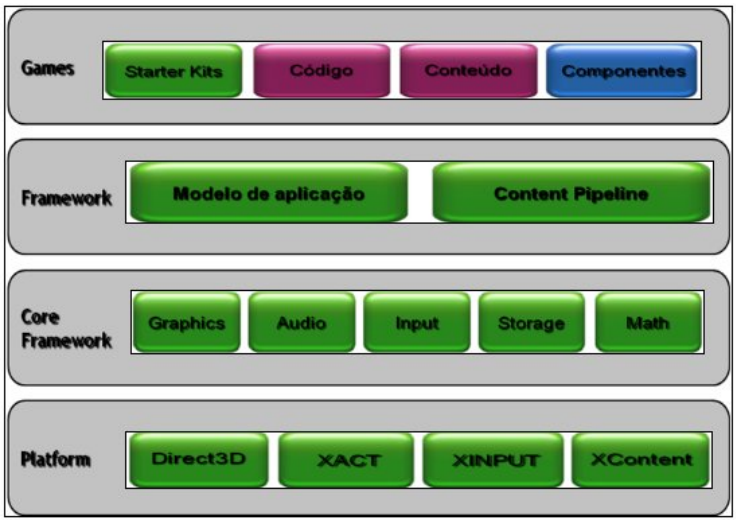


XNA Game Studio & Framework

Utilizando o XNA Game Studio, uma extensão do IDE do Visual C# 2008 Express Edition e Visual Studio 2008, podemos começar a dar os primeiros passos e desenvolver os nossos jogos. O XNA Game Studio, que se encontra na versão 3.0, aproveita quase todas as funcionalidades do Visual Studio 2008 (Ex: Intellisense, LINQ). Esta última

versão introduz vários melhoramentos tanto para gamers como para developers. Incorpora starter kits que são jogos completos de código aberto com as respectivas texturas, sons, etc. onde é possível fazer modificações para melhor compreender a plataforma XNA.. Nesta versão é possível incorporar directamente do VS2008 ficheiros .mp3, .wav, .wmv, entre outros. Podemos também fazer deployment dos jogos criados para a XBOX360 e para o Zune.

O XNA Game Studio 3.0 funciona com base no XNA Framework, que nos permite o desenvolvimento de jogos de forma simples pois facilita o acesso ao hardware gráfico, aos controladores (rato, teclado ou game pad) utilizados, aos recursos de áudio e até mesmo aos dispositivos de armazenamento de informação dos nossos jogos. Nesta imagem encontramos de forma geral os elementos que constituem a XNA Framework.



A XNA Framework foca a simplicidade de construção de jogos, é constituída por várias camadas:

Platform

Consiste no conjunto de API's com que a Framework foi construída, Direct3D 9, XACT, XINPUT e XContent.
Core Framework
Esta é a "primeira" camada da XNA Framework fornece várias áreas funcionais que as outras camadas irão extender. Aqui encontramos várias áreas de funcionalidades agrupadas em secções como gráficos, áudio, input, math e armazenamento.

Extended Framework

O principal objectivo desta camada é tornar o desenvolvimento dos jogos eficaz e objectivo. Através do Application Model não precisamos de nos preocupar com detalhes da criação de janelas, timers e outros items, pois este abstrai-nos da plataforma de hardware em que irá correr o nosso jogo. O Content Pipeline, é usado para a gestão de conteúdos do nosso jogo, sejam esses imagens, modelos 3D ou áudio.

Games

Esta é a camada superior, consiste no nosso código de jogo, conteúdos, starter kits, templates e alguns componentes.
Como podem reparar na imagem, a verde encontramos os elementos que são fornecidos pela plataforma XNA: o programador apenas tem que se concentrar nos conteúdos e no código do jogo. A comunidade XNA, que está em crescente desenvolvimento, fornece-nos alguns componentes que podemos incorporar durante o desenvolvimento.

XNA EM 2D

Pretende-se explicar o funcionamento de um simples jogo em 2D manipulando a tecnologia XNA. O jogo consiste num jogador conseguir apanhar objectos e fugir de um inimigo dentro do cenário.

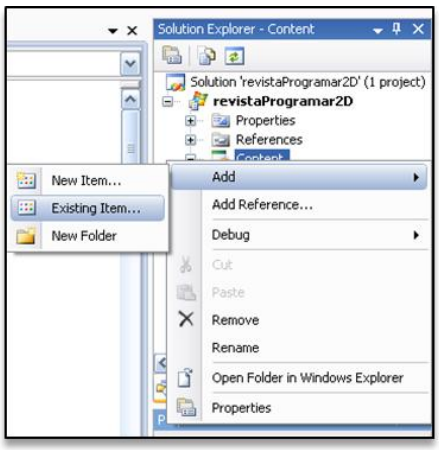
Requisitos:

- Microsoft Visual Studio C# 2008 Express ou Microsoft Visual Studio 2008,
- XNA Game Studio 3.0.

Iniciamos o Visual Studio 2008 e escolhemos a opção de iniciar novo projecto, seleccionando assim a opção de desenvolvimento com o XNA Game Studio 3.0. Temos aqui à disposição o tipo de plataforma para o qual queremos desenvolver o nosso jogo: XBOX360, Zune ou PC. É também possível a escolha de um starter kit e o desenvolvimento de bibliotecas para jogos mais complexos e serem adicionadas a estes mais tarde ou mesmo partilhadas entre outros utilizadores.

Neste caso selecionei a opção Windows Game (3.0) atribuindo o nome revistaProgramar2D e a localização onde o meu projecto irá ficar guardado.

De seguida, são carregados alguns ficheiros criados automaticamente pelo XNA Game Studio que já contêm algum código.



Como podem reparar no canto superior direito encontramos o Solution Explorer, que funciona em esquema de árvore, permite adicionar, remover e organizar ficheiros e conteúdos de jogo. É visível o nosso projecto (revistaProgramar2D) já com alguns ficheiros como o Game1.cs e o Program.cs. Antes de explicar o que está dentro destes e como iremos desenvolver o nosso jogo, vamos carregar os conteúdos que serão usados no nosso jogo. Tal como o ficheiro Courier New.SprintFont criado por mim previamente que contém uma descrição XML da fonte que define o estilo da fonte a ser escrita e actualizada no ecrã durante o decorrer do jogo. Este pode ser alterado consoante o estilo que queremos apresentar. Carregamos também alguns ficheiros de imagem definidos para representar cada um dos objectos do jogo.

Para carregar estes ficheiros basta clicar na secção content e adicioná-los, como é demonstrado na imagem acima. Assim que são carregados para o nosso projecto irão aparecer no Solution Explorer.

Após carregar os conteúdos, neste momento, vamo-nos preocupar-nos com os ficheiros: Game1.cs e Program.cs. Serão estes que iremos programar para o funcionamento do nosso jogo.

Program.cs

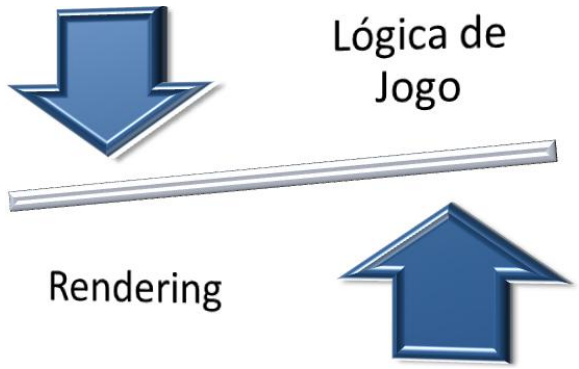
O ficheiro Program.cs executa o Game Loop.

```
using System;

namespace revistaProgramar2D
{
    static class Program
    {
            static void Main(string[]
args)
        {
              using (Game1 game = new
Game1())
            {
                  game.Run(); //Executa
o Game Loop
            }
        }
    }
}
```

Game Loop

O Game Loop é um ciclo infinito que é usado para desenhar os objectos no ecrã e executar a lógica de jogo.

- O Game Loop é 1/60, ou seja, trabalha a 60 loops por cada segundo, embora este valor possa ser alterado.
- No Game Loop é executada a lógica (Actualizar o estado do jogo, recebe novos inputs do jogador, actualiza a posição dos enimigos, etc.) e o Rendering (Mostra visualmente os (novos) estados.) do jogo.



Game1.cs

No início deste ficheiro encontramos os Namespaces que nos permitem aceder à XNA Framework:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using
Microsoft.Xna.Framework.GamerServices;
using
Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

O ficheiro Game1.cs contém a classe central do nosso jogo que deriva da classe principal do XNA Microsoft.XNA.Framework.Game.

Aqui são declarados alguns dos objectos que nos vão permitir desenhar no espaço 2D do nosso jogo. O XNA disponibiliza-nos alguns que nos ajudam nesta tarefa, tal como o GraphicsDeviceManager. Este é um objecto fulcral, pois é o gestor do dispositivo gráfico.

O Texture2D que na realidade representa uma imagem, ou seja, é um sprite.

O SpriteBatch, que se encarrega de desenhar as sprites no ecrã. Para desenhar as sprites é necessário usar o método

draw ou drawString (para as SpritesFonts) como iremos ver mais à frente em código apresentado. A performance dos jogos está ligado ao número de SpriteBatchs.

O objecto Rectangle, é usado para definir os sprites, permite verificar se por exemplo dois Rectangle's se interceptam, verificando se existem colisões entre estes.

O Vector2 é um objecto muito importante, pois permite representar um ponto/vector (X,Y) podendo efectuar diversas operações com vectores.

```
namespace revistaProgramar2D
{

    public class Game1 :
Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager
graphics;
        SpriteBatch spriteBatch;

        Vector2 velocidade;

        SpriteFont fonte;

        Texture2D fundoTex;
        Rectangle fundoRect;

        Texture2D revistaTex;
        Rectangle revistaRect;

        Texture2D jogadorTex;
        Rectangle jogadorRect;

        Texture2D enimigoTex;
        Rectangle enimigoRect;


        int ler;
      ...
```

Através da herança desta classe podemos abstrair-nos de diversos recursos que o nosso jogo irá necessariamente usar.

```
public Game1()
        {
            graphics = new
GraphicsDeviceManager(this);
            Content.RootDirectory =
"Content";
        }
```

A classe Game1 apresentada é automaticamente escrita pelo XNA após a criação de um novo projecto. O Content.RootDirectory = "Content" define a directoria raiz do ContentManager, usado para carregar todos os conteudos do jogo. Aqui definimos a pasta Content como raiz.
Muitas vezes os conteúdos para desenvolvimento de um jogo, não estão num formato suportado o que implica perda de tempo de desenvolvimento em conversões ou emulações de conteúdos para que estes possam devidamente ser usados. O Content Pipeline importa estes ficheiros, compila-os e permite o uso destes nos formatos suportados.

Irá tambem ser criado um novo objecto graphics do tipo de objecto GraphicsDeviceManager que permitirá o controlo e configuração do dispositivo gráfico especificado para o jogo, neste caso o monitor do nosso PC. Esta classe é carregada assim que o Game1 é inicializado no ficheiro program.cs.

De seguida teremos que implementar um conjunto de métodos básicos para o desenvolvimento do nosso jogo:

```
protected override void Initialize()
        {
         //Inicialização dos conteúdos
lógicos
   //Define um novo spritebach object
ao GraphicsDevice para que funcione
correctamente assim que o jogo se
inicia.
   spriteBatch = new
SpriteBatch(GraphicsDevice);

          Random rand = new Random();
          velocidade = new
Vector2((float)(rand.NextDouble() *
10) - 5, (float)(rand.NextDouble() *
10) - 5);

          ler = 0;


graphics.PreferredBackBufferHeight =
700;

graphics.PreferredBackBufferWidth =
900;
          graphics.ApplyChanges();

          Window.Title = "Revista
Programar";

          base.Initialize();
        }
```

As linhas de código graphics.PreferredBackBufferHeight = 700 e graphics.PreferredBackBufferWidth = 900 definem o tamanho da janela do nosso jogo. Torna-se bastante útil, se estivermos a desenvolver um jogo para a XBOX360 uma vez que evitamos deformações com widesreen.

Este método é executado apenas uma vez e antes do Game Loop se iniciar. Tem como função atribuir valores, parâmetros e condições de entrada no Game Loop

Os dois próximos métodos apresentados são utilizados para carregar/descarregar os conteúdos gráficos no jogo e são invocados quando necessário.

```
protected override void LoadContent()
{

// Carrega conteúdos

fonte =
Content.Load<SpriteFont>("fonte");
fundoTex =
Content.Load<Texture2D>("fundo");
jogadorTex =
Content.Load<Texture2D>("jogador");
enimigoTex =
Content.Load<Texture2D>("enimigo");
revistaTex =
Content.Load<Texture2D>("revista");


fundoRect = new Rectangle(0,
0,graphics.GraphicsDevice.Viewport.Widt
h,
graphics.GraphicsDevice.Viewport.Height
);

jogadorRect = new
Rectangle(graphics.GraphicsDevice.Viewp
ort.Width / 2,
graphics.GraphicsDevice.Viewport.Height
 / 2, jogadorTex.Width,
jogadorTex.Height);

Random rand = new Random();

enimigoRect = new
Rectangle(rand.Next(graphics.GraphicsDe
vice.Viewport.Width -
enimigoTex.Width),
rand.Next(graphics.GraphicsDevice.Viewp
ort.Height - enimigoTex.Height),
enimigoTex.Width, enimigoTex.Height);

revistaRect = new
Rectangle(rand.Next(graphics.GraphicsDe
vice.Viewport.Width -
revistaTex.Width),
rand.Next(graphics.GraphicsDevice.Viewp
ort.Height -
revistaTex.Height),revistaTex.Width,
revistaTex.Height);
}
```

Pequena explicação: O XNA atribui os conteúdos previamente importados às variáveis no código. Para que o nosso código utilize as texturas, sons, e outros conteúdos que incluímos na nossa solução apenas é preciso usar o método Content.Load<type>(...) que recebe como parâmetro o nome do ficheiro atribuido pelo XNA Game Studio na altura da importação (ver as propriedade "Asset Name" do conteúdo). Por exemplo, se importamos uma nova textura para \Content\Texturas\NovaTex então poderemos escrever: aMinhaNovaTex = Content.Load<Texture2D>("Texturas\\NovaTex");"

```
protected override void
UnloadContent()
{
}
```

Os dois métodos seguintes são chamados de métodos de Loop uma vez que são invocados por cada game Loop e são invocados pela ordem pela qual os vamos descrever.

```
protected override void
Update(GameTime gameTime)
```

Este método é usado para executar a lógica de jogo que pode passar por definir novas posições dos objectos, detectar colisões, tratar os inputs do teclado, etc

O código apresentado demonstra alguns desses exemplos para o nosso jogo:

```
//Tecla esc para sair do jogo
if
(GamePad.GetState(PlayerIndex.One).Butt
ons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Esca
pe))
this.Exit();

// Movimentos do jogador
KeyboardState keys =
```

```
Keyboard.GetState();

            if
(keys.IsKeyDown(Keys.Up))
                jogadorRect.Y -= 6;
            if
(keys.IsKeyDown(Keys.Down))
                jogadorRect.Y += 6;
            if
(keys.IsKeyDown(Keys.Left))
                jogadorRect.X -= 6;
            if
(keys.IsKeyDown(Keys.Right))
                jogadorRect.X += 6;

// Intersecção com o inimigo, ele
circula aleatoriamente pelo ecrã.
if
(jogadorRect.Intersects(enimigoRect))
{
ler -= 1;
Random rand = new Random();
enimigoRect.X =
rand.Next(graphics.GraphicsDevice.Viewp
ort.Width - enimigoTex.Width);
enimigoRect.Y =
rand.Next(graphics.GraphicsDevice.Viewp
ort.Height - enimigoTex.Height);

velocidade = new
Vector2((float)(rand.NextDouble() *
10) - 1, (float)(rand.NextDouble() *
10) - 1);
}

enimigoRect.X += (int)velocidade.X;
enimigoRect.Y += (int)velocidade.Y;

int xpos =
(int)MathHelper.Clamp(enimigoRect.X,
0.0f,
graphics.GraphicsDevice.Viewport.Width
- enimigoTex.Width);
int ypos =
(int)MathHelper.Clamp(enimigoRect.Y,
0.0f,
graphics.GraphicsDevice.Viewport.Height
 - enimigoTex.Height);

 if (enimigoRect.X != xpos)
     velocidade.X *= -1;
 if (enimigoRect.Y != ypos)
     velocidade.Y *= -1;

enimigoRect.X = xpos;
enimigoRect.Y = ypos;
```

```
// Intersecção com a revista
programar, soma mais um ponto!

if
(jogadorRect.Intersects(revistaRect))
{
ler += 1;
Random rand = new Random();
revistaRect.X =
rand.Next(graphics.GraphicsDevice.Viewp
ort.Width - revistaTex.Width);
revistaRect.Y =
rand.Next(graphics.GraphicsDevice.Viewp
ort.Height - revistaTex.Height);
}
base.Update(gameTime);
}
```

Este último método é onde são executadas as rotinas de desenho do jogo. (Game rendering)

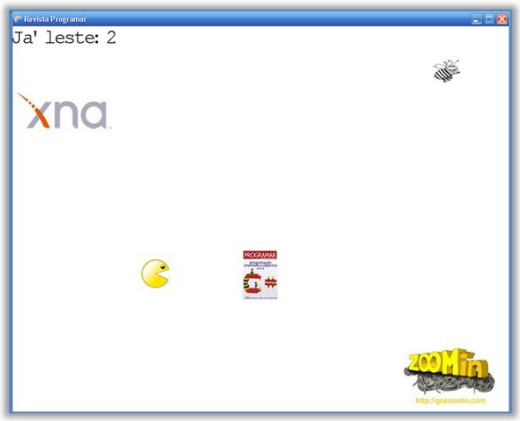
```
protected override void Draw(GameTime
gameTime)
```

Exemplo de código para o nosso jogo:

```
{
graphics.GraphicsDevice.Clear(Color.Whi
te);
//Iniciar o SpriteBatch.
spriteBatch.Begin();

spriteBatch.Draw(fundoTex, fundoRect,
Color.White);
spriteBatch.Draw(jogadorTex,
jogadorRect, Color.White);
spriteBatch.Draw(enimigoTex,
enimigoRect, Color.White);
spriteBatch.Draw(revistaTex,
revistaRect, Color.White);

//Passar a string actualizada para o
ecrã.
spriteBatch.DrawString(fonte, "Ja'
leste: " + ler.ToString(),
Vector2.Zero, Color.Black);

spriteBatch.End();
//Terminar o SpriteBatch.

base.Draw(gameTime);
}
```

Todas as instruções para desenhar sprites têm que estar incluídas entre o spriteBatch.Begin() e o spriteBatch.End() pois o XNA vai automaticamente optimizar a renderização juntando todos sprites para mandar de uma só vez para a placa gráfica. Neste exemplo simples não fazia mal ter vários Begin(), End() mas cada um deles incrementa o número total de batches o que reduz a performance.

Compilando e executando o nosso projecto podemos ver qual é o resultado final, temos assim o nosso jogo a correr. Este é apenas um pequeno exemplo de um simples jogo em 2D usando a tecnologia XNA.



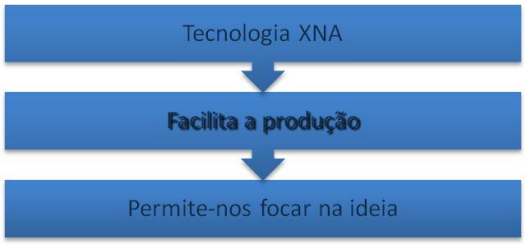
Jogos desenvolvidos com XNA

Este jogo, City Rain, foi desenvolvido no âmbito do concurso Imagine Cup, na categoria de Game Development, foi um dos 6 jogos finalistas do concurso no ano anterior. O jogo foca no desenvolvimento de uma cidade não esquecendo o ambiente sustentável desta e a sua ecologia.

Conclusão

Tendo todo este conjunto de ferramentas e tecnologias à nossa disposição, o que irá caracterizar os nossos jogos para que tenham sucesso?

Bons gráficos? Bons efeitos de áudio? Uma boa história de jogo? Sim, todos esses são elementos válidos, mas acima de tudo é uma boa ideia! Ideias como a do senhor William Higinbotham.



Usando a plataforma XNA apenas temos que dar uso à nossa imaginação, definir o funcionamento do jogo, os conteúdos, elementos deste e desenvolver a nossa ideia. Quem sabe se não poderemos estar a desenvolver o próximo jogo a ser jogado em todo o mundo por milhões de gamers?

Obrigado, aos meus colegas MSP's Ricardo Costa, Gonçalo Chaves, Bruno Tavares e Bruno Silva pela partilha de conhecimento, que não só me ajudou a escrever este artigo, mas também incentivou ao gosto por esta tecnologia e área.

Recursos

XNA Developer Center
http://msdn.com/xna
Creators Club Online
http://creators.xna.com/
Ziggyware
http://www.ziggyware.com/

SOBRE O AUTOR

Natural de Ameixoeira (Castelo Branco), João Filipe Antão é finalista do curso de Engenharia Informática da Universidade da Beira Interior e um Microsoft Student Partner. Partilha os estudos em trabalhos como fotógrafo freelancer, músico em vários projectos e o movimento Eco Projecto - Serra da Estrela.

joao.antao@portugal-a-programar.org

João Antão

Fundamentos de Segurança em Redes (Firewall)

Na primeira parte deste artigo, vimos a importância de encararmos a segurança, não apenas como uma camada, uma única solução de segurança, mas sim como um conjunto de camadas que criam uma solução de segurança mais adequada ás necessidades da rede. Entre essas camadas que constitui a solução de segurança vimos a importância do desenho da rede e o saber aplicar diversas técnicas como por exemplo, Subnetting e Switching. Apesar disto tudo é necessário no entanto recorrer a diversos dispositivos essenciais para a protecção de uma rede e na continuação deste artigo vamos ver brevemente alguns dispositivos e soluções de segurança mais comuns:

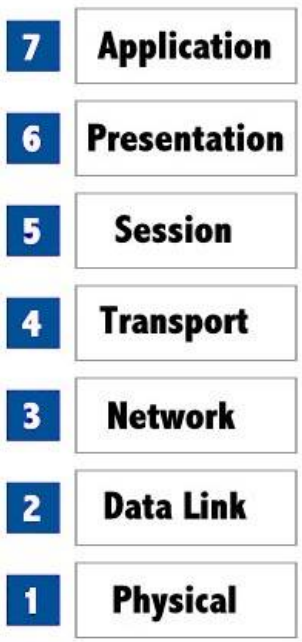
Firewall

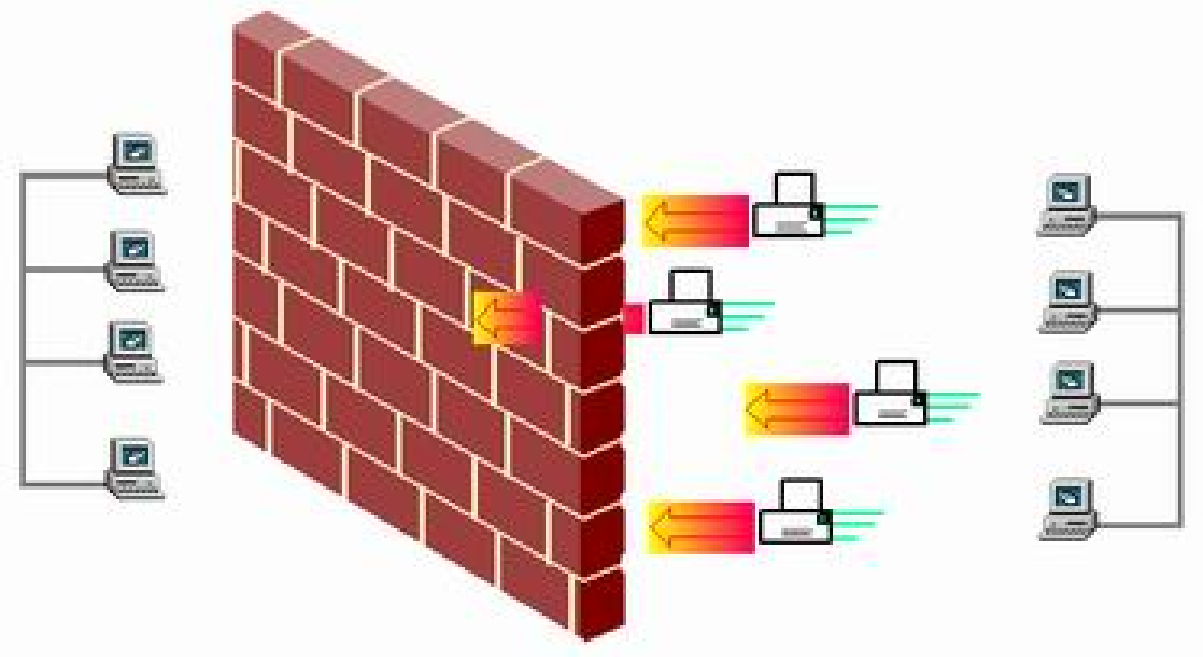
Esta talvez seja logo a primeira solução que vêm á mente quando falamos em segurança num computador ou numa rede. Existem duas categorias de firewall: software e hardware. No caso do software, a firewall é instalada numa máquina preparada para isso e fica localizada entre a rede interna e a rede pública. Um firewall baseado em hardware corre numa appliance que contém um sistema operativo especificamente desenhado para esse tipo de tarefas. Claro que estes últimos têm um melhor desempenho mas são muito mais complexos na sua configuração e não são tão flexíveis. Mas basicamente podemos dizer que um firewall é um programa ou a combinação de um software com hardware (appliance) que filtra a informação que vem da Internet para dentro da rede ou apenas um PC, ou seja contém regras que especificam o que pode ou não passar em termos de tráfego. Portanto se um pacote é indicado ou marcado pela firewall não passa para dentro da rede interna. É uma boa medida de defesa contra actividades maliciosas o colocar ou implementar uma firewall num ponto de conexão entre a rede interior com o mundo lá fora.

Configuração

A configuração de um firewall assenta em alguns aspectos básicos e que nos dão grande margem de manobra no que toca a criar regras. Algumas das coisas que podemos usar para configurar um firewall são: Endereços de IP - um endereço IP é algo como 205.45.78.1 e pode ser usado na configuração de um firewall, restringindo ou bloqueando tráfego baseado no endereço IP de origem ou destino; Protocolos - um protocolo é maneira como por exemplo dois computadores comunicam entre si. Alguns dos protocolos usados nos filtros de uma firewall podem incluir TCP(Transmission Control Protocol) , UDP(User Datagram Protocol), ICMP(Internet Control Message Protocol), IP(Internet Protocol), HTTP(Hyper Text Transfer Protocol), SMTP(Simple Mail Transport Protocol), FTP (File Transfer Protocol)e muitos outros protocolos; Portas - serviços usam portas e certas aplicações utilizam determinadas portas para comunicar entre si. Por isso estas portas podem ser usadas para bloquear determinados serviços e ou aplicações; Também em alguns casos se pode usar palavras ou frases para filtragem, em que o Firewall com base nessas palavras examina a informação no pacote. A quando a configuração do firewall poderemos começar por bloquear todo o tipo de tráfego e aos poucos criarmos regras para aquele tráfego que poderá passar. Ter em atenção que é importante colocar filtros ou regras especificas antes dos filtros ou regras gerais, por que se isso não acontecer o filtro geral poderá deixar passar um pacote para depois mais á frente ele ser descartado por um filtro específico. Outro aspecto importante é implementar um filtro para o tráfego que saí indicando que apenas poderá sair tráfego cujo endereço de origem seja de dentro da rede, é uma boa medida para parar spyware e botnet. Outra medida é usar listas para definir quais as aplicações, endereços de IP são de confiança e quais não são. De seguida vamos ver alguns tipos de firewall:

- Packet filtering firewalls





Filtragem de pacotes é um tipo de firewall que trabalha geralmente na Camada 2 (Data Link) e na Camada 3 (Network). Esses filtros baseados em componentes presentes no firewall, examinam os dados que passam para dentro e fora da rede. Os pacotes são comparados com regras já definidas que vão indicar se o pacote poderá passar ou ser descartado. Essas regras podem incluir definições tais como o IP de destino ou de origem, os números das portas usadas e os protocolos como já vimos anteriormente. A grande vantagem de firewall que fazem filtragem com base nos pacotes é a rapidez a que essa filtragem pode ser feita. É comum usar-se este tipo de dispositivos no perímetro da rede como primeira linha de defesa. No entanto, apesar de ser uma solução rápida, eficiente e rentável tem como é lógico falhas. Muitos firewall's de filtragem de pacotes não conseguem detectar spoofed IP ou ARP endereços. Daí o facto de que este tipo de firewall's serem usados principalmente como forma de prevenir ataques Deniel-of-Service e não contra ataques mais intensos. Além de que outras funcionalidades como por exemplo a autenticação, não pode ser suportado dado que trabalha em camadas mais altas que a do firewall.

- Stateful packet filtering

Existe um outro tipo de firewall chamado de stateful packet. A Camada 4 (Transporte) no modelo OSI trabalha usando conexões, recorrendo para isso ao endereço de origem e a porta usada e o endereço de destino e a porta usada, usando por exemplo o protocolo Transmission Control Protocol mais conhecido por TCP. Como é que a Camada 4 está envolvida com este tipo de Firewall? Bem, um Firewall que use este método de filtragem armazena em tabelas todas as conexões estabelecidas para fora e para dentro. Quando ocorre uma requisição do lado de fora para estabelecer uma conexão, os parâmetros da mesma são verificados nas tabelas e se certas regras forem satisfeitas poderá ocorrer uma "conexão legítima". Este tipo de técnica usa como vimos o protocolo TCP e outros métodos de controlo dos dados para filtrar. A informação da conexão é mantida em tabelas que são controladas dinamicamente. Cada conexão é introduzida na tabela e depois de ser validada, os pacotes são encaminhados com base nas regras definidas naquela conexão em particular. Deixem-me dar um exemplo, como sabem existe portas standards para diversas aplicações. Se uma aplicação, vamos supor usa geralmente as portas o a 1023 e se por ventura na conexão um dos parâmetros é a porta ser a 2503 será lógico da parte do Firewall bloquear esse acesso. Acontece no entanto que esta é uma filtragem que já ocorre numa camada mais elevada envolvendo muito mais informação e portanto o desenho e a implementação das regras deve ser quase prefeito. Quando tal não é feito correctamente e são implementadas demasiadas restrições torna-se complicado ou praticamente impossível utilizar a rede e os recursos nela existentes. Este tipo de filtragem é mais avançada que o Firewall Packet Filtering e também faz um melhor trabalho, mas existe um contra. Qual? Bem, como podemos ver este tipo de firewall utiliza uma arquitectura complexa ao usar tabelas dinâmicas agregadas a outras funcionalidades complexas. Comparando este tipo de firewall com a firewall que utiliza uma filtragem de pacotes, vemos que o Stateful packet filtering carece da robustez e flexibilidade. Além de que influenciam bastante a rapidez com que os pacotes são analisados á medida que vão aumentando as conexões e os pacotes a analisar assim como também as tabelas.

- Proxy Firewall

É uma aplicação que geralmente usa mais a Camada 7 (Aplicação) do modelo OSI para correr. Um proxy como é geralmente conhecido é usado como mecanismo de protecção e filtragem entre redes seguras, internas ou privadas e redes públicas. Em termos de autenticação, é bastante bom a utilização de um proxy dado que podemos usar esquemas de autenticação mais avançadas e que podem ser implementadas para fortificar a segurança. Em muitos casos, estes proxies podem ser usados como complementos das firewall principais. A desvantagem deste tipo de firewall é a rapidez. Não nos esqueçamos que os proxies trabalham em camadas superiores o que envolve trabalharem com muita quantidade de dados e informação que tem de ser processada sacrificando assim alguma velocidade. No entanto, proxies oferecem a melhor opção de segurança a usar.

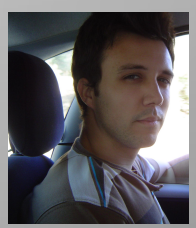
Desvantagens?!

Sim como vimos em cada um dos casos existem desvantagens e vantagens entre eles. Mas mesmo assim existe algumas desvantagens inerentes ao instalar e usar um firewall. Não basta chegar e instalar um firewall no caso de uma rede complexa, é necessário primeiro fazermos um estudo. Sim o estudo de uma rede, assim como o seu desenho é importante, apesar de muitas vezes ser deixado completamente de lado. É necessário ver qual a arquitectura da rede assim como as vulnerabilidades da mesma e daí fazermos as escolhas certas e qual será o tipo de firewall mais apropriado. Também não é um sistema infalível dado que poderá ser contornado por atacantes que terão então acesso à rede e logicamente mais fácil acesso aos recursos da mesma. Outra desvantagem é quando a configuração da firewall não é feita correctamente e por vezes aqueles que por direito teriam acesso à rede ficam impedidos disso. Claro que numa rede doméstica estas são situações que não terão o impacto que teriam se fosse uma empresa. Mais uma vez esta pequena análise prova a importância de não depositarmos a nossa confiança apenas num aplicativo ou numa protecção. No próximo artigo veremos outras maneiras de nos protegermos e que poderão ser implementadas junto com o Firewall.

A ver:

http://www.endian.com/en/community/

SOBRE O AUTOR



Residente em Lisboa, Ciro Cardoso é um grande amante do vasto mundo que é a informática em especial a segurança. Está actualmente a trabalhar na próxima etapa do percurso Cisco com a certificação CCNP. Gosta bastante da área de redes, configuração e implementação de protocolos.

ciro.cardoso@portugal-a-programar.org

Ciro Cardoso

Introdução ao Arduino

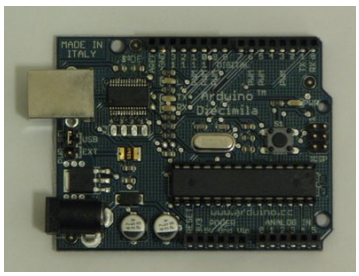
Introdução

É objectivo deste artigo dar a conhecer o Arduino. Esta "ferramenta" com enormes potencialidades, que pode ser utilizada por todos, tenham ou não conhecimentos de electrónica devido à sua enorme simplicidade de utilização.

O Arduino tem ainda muita margem de desenvolvimento, começando como um pequeno projecto educacional evoluindo até aos dias de hoje. Dentro das suas vantagens pode-se encontrar o facto de ser open-source, correndo em ambiente Linux, Macintosh e Windows tendo ainda o aliciante de ser bastante económico comparativamente com "ferramentas" de iguais funcionalidades disponíveis no mercado.

Para apresentar este tema com maior simplicidade, o que não significa menor rigor, torna-se necessário dividi-lo em duas partes distintas: hardware e software e fazer a sua ligação.

É importante referir que o Arduino abre imensas portas, devendo haver por parte do leitor interesse em procurar, conhecer e aprender.



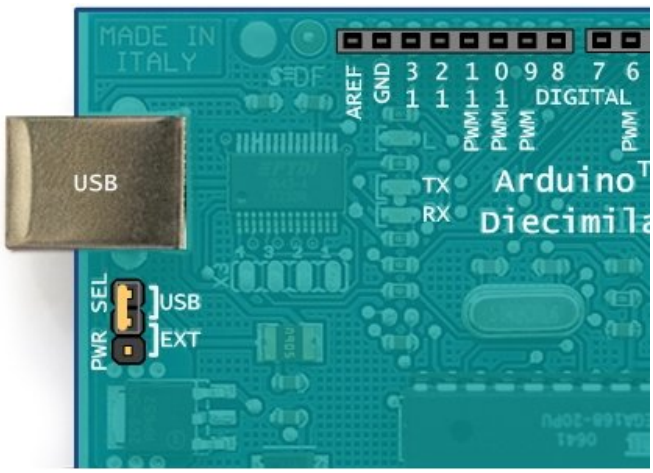
Hardware

O Arduino pode ter várias apresentações sendo a base deste artigo o Arduino Diecimilia (baseado num microprocessador Atmega168 da Atmel).

Convém dar especial atenção a este capítulo sobre hardware, pois é sobre ele que toda a programação se vai apoiar.

Não se torna necessário, como foi referido anteriormente, possuir elevados conhecimentos de electrónica, já que o único requisito realmente relevante é a vontade de aprender.

Observemos então pormenorizadamente a seguinte figura:



Pela análise da figura atrás apresentada, que representa uma placa - *Arduino Diecimilia*, podemos constatar o seguinte "pin out":
- 3 pinos de **GND** (Ground);
- 1 pino de alimentação de 3.3V (3V3) e um de 5V;
- Possui um pino denominado Vin, que possibilita o uso da tensão colocada à entrada (Pwr), antes de passar pelo controlador de tensão, sendo esta funcionalidade programável por software.
- Um pino de reset, que à semelhança do botão presente no Arduino, e como o próprio nome indica faz o reinício do Arduino, ou seja, executa o programa a partir do inicio novamente, através da aplicação de um sinal de entrada. Voltando a executar o bloco de instruções da função setup, como referido mais à frente no artigo.
- 14 portas digitais (0 ao 13) configuráveis como input ou output. Com a possibilidade de em seis destas (5,6,9,10,11) usar PWM – Pulse Width Modulation. Esta potencialidade é muito importante, pois através da variação da largura do impulso pode-se "simular" tensões entre 0 e 5V. Os pinos digitais 0 e 1 possibilitam, ainda, quando configurados, o envio de informação em série. Permitindo outra interface, dependendo do fim pretendido.
- 6 Portas analógicas, possuindo um conversor analógico digital de 10 bits. E fazendo as contas:

$$2^{10} = 1024$$

Como a tensão máxima de referência, por definição, se encontra nos 5v correspondendo ao valor 1023, obtemos a seguinte resolução:

$$5 \div 1024 = 0{,}00488\ V = 5\ mV$$

O que significa que só se conseguirá "detectar" variações superiores a 5 mV. Ou seja, o valor lido pelo Arduino só se altera a cada 5 mV de variação do sinal analógico de entrada.

Em caso de aplicação de sensores, como por exemplo de sensores de temperatura do tipo termo-pares, que podem ter variadíssimas apresentações e que funcionam na casa dos mV, correspondendo 5 mV em certos casos a subidas de temperatura da ordem dos 80 ºC. Torna-se essencial encontrar uma solução, sem recorrer a electrónica externa.

Assim, para tal existe uma porta de entrada denominado AREF, que significa "Analog Reference". Este pino permite mudar a referência analógica do standard 5V para o valor introduzido. Ficando todas as entradas analógicas com a referência introduzida.

Simplificando, se se introduzir no pino AREF a tensão de 2V obtém-se a seguinte resolução:

$$2 \div 1024 = 1.953 \text{ mV} = 2 \text{ mV}$$

É importante ter em conta que todas as portas ficam com esta referência, sendo necessária também a sua configuração por Software.

É igualmente de referir que após configurar o Arduino para o uso do pino AREF, ele deixa de ter disponíveis os pinos de 3.3V e 5V. E que estando estes desligados, será então necessário recorrer a alimentação externa.

O Arduino possui capacidade de operar alimentado, quer pela porta USB ou por uma entrada Pwr. Sendo recomendada a sua utilização entre os 7 e os 12V, que possibilita uma operação do tipo "Standalone".

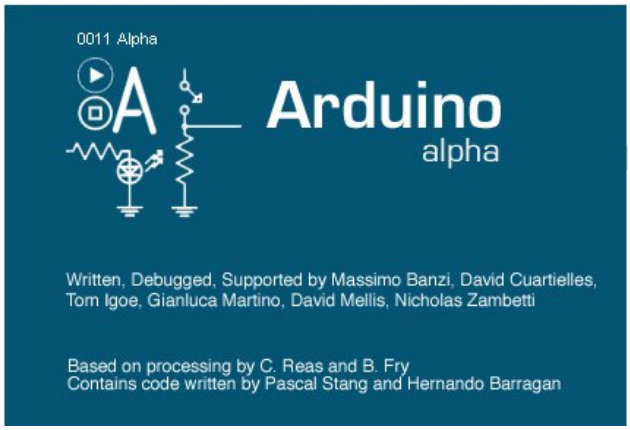
Poderíamos referir aspectos de funcionamento do próprio microprocessador, o que não se torna relevante para o aproveitamento das capacidades do Arduino. O que não significa que não seja objecto de interesse. Devendo o leitor, caso queira, aprofundar os seus conhecimentos nesta matéria fazendo uma pequena pesquisa.

Havendo para isso muita documentação disponível online, sendo uma paragem obrigatória o site oficial do Arduino - http://www.arduino.cc/.

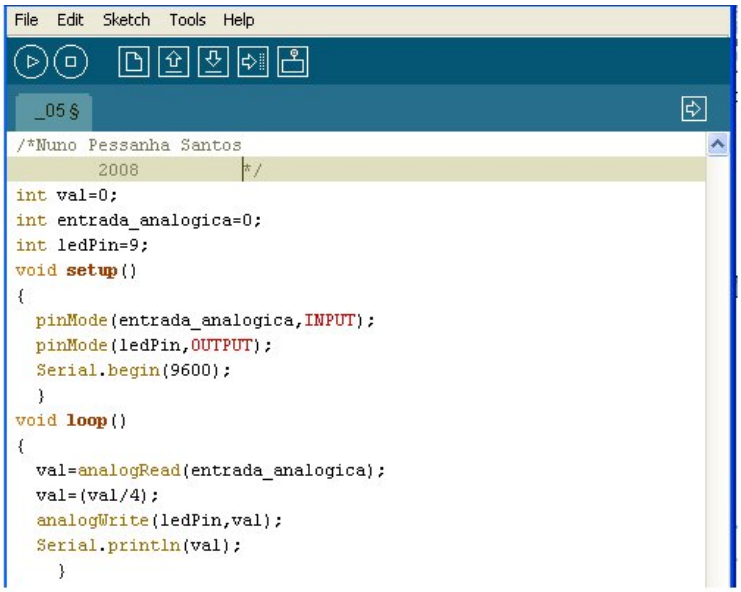
Software

O ambiente de desenvolvimento pode ser obtido através do site oficial do Arduino, correspondendo a última versão ao Arduino 0012.

Sendo a sua distribuição completamente livre (sendo "Open-Source", como já foi referido anteriormente).



A linguagem usada nesta aplicação é uma versão simplificada de C, possuindo o mesmo tipo de regras e funções básicas.



Na figura acima apresentada, é possível visualizar o ambiente de desenvolvimento do Arduino, muito intuitivo e fácil de usar. Para carregar um programa é simplesmente necessário, elaborá-lo e com um simples click no botão Upload to I/O Board, o programa é compilado e enviado para o Arduino.

Isto é possível ser feito sem recorrer a Hardware externo, pois o Arduino possui um Bootloader de origem. Ferramenta esta que quando ligada, possibilita que o Arduino receba os comandos enviados pela porta USB.

No entanto, se não chegarem dados, o último programa carregado é executado. E no caso de ser a primeira utilização do Arduino, o único programa em memória será ele mesmo.

Constituição do Código

O código é constituído por dois blocos de funções distintas

Função setup

```
void setup() {
    Instruções 1;
}
```

Função loop

```
void loop() {
    Instruções 2;
}
```

Função Setup

A função Setup é executada quando o programa começa, sendo este bloco de instruções apenas executado uma vez. É executado quando o Arduino é ligado ou quando se efectua o reset.

É usada normalmente como a função responsável por inicializar variáveis, definir os pinos (I/O), definição de bibliotecas entre outros.

Exemplo 1:

```
int buttonPin = 5;
void setup()
{
    Serial.begin(9600);
    pinMode(buttonPin, INPUT);
}
```

Dentro do bloco de instruções está a ser configurado o modo de comportamento do pino 5, definido como buttonPin (linha 1). No exemplo apresentado, está definido que o pino 5 está configurado como INPUT. Contudo é bastante fácil configurá-lo como OUTPUT, ficando:

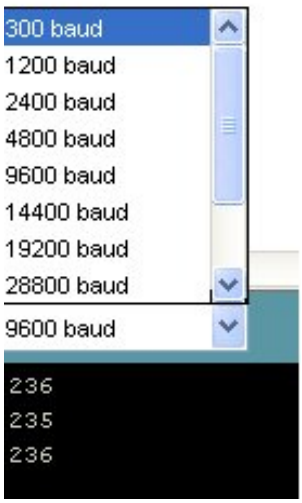
Exemplo 2:

```
pinMode(buttonPin, OUTPUT);
```

A função *Serial.begin(int taxa_bps)*, é usada para definir a taxa de transmissão em série. Tipicamente para comunicar com o computador, temos taxas de 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200 bps.

Sendo comum definirem-se valores diferentes dos apresentados, para comunicar com outros dispositivos.

As taxas de transmissão atrás referidas são as suportadas pelo Software padrão do Arduino.



Ao utilizar esta funcionalidade (comunicação pela porta série), que permite ler facilmente valores, p.ex. de sensores, é também possível, através de qualquer Software que faça leituras da porta série, tratar e guardar esses dados, bem como apresentá-los sobre a forma de gráficos ou tabelas, conforme o pretendido.

Um exemplo de um programa muito comum, que possibilita a leitura pela porta série é o Matlab. Este permite o armazenamento e tratamento de dados, como referido anteriormente.

Exemplo 3:

```
s = serial('COM7', 'BaudRate',19200);
fopen(s)
fprintf(s, 'hello arduino!');
end
fclose(s)
```

Definido o número da porta série, neste caso COM7, e o BaudRate como 19200 bps, na situação apresentada (linha 1). É de seguida enviada a seguinte mensagem 'hello arduino!' (linha 3). Terminando com um fecho da ligação fclose(s) (linha 5).

Sendo o Matlab apenas um exemplo de muitos softwares que existem disponíveis e que possibilitam Rx/Tx através da porta série (p.ex. HyperTerminal).

É ainda possível obter mais informações sobre este tema na página oficial Arduino.

Função Loop

Após a execução da função Setup, a função Loop é iniciada. E como o próprio nome indica, faz um loop sucessivo e permite que o programa faça acções pré-estabelecidas.

Um exemplo deste tipo de acções pré-estabelecidas é a leitura de botões ou sensores, apenas possível através desta característica da função loop.

Exemplo 4:

```
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    Serial.println('H');
  else
    Serial.println('L');
  delay(1000);
}
```

O exemplo acima está a fazer a leitura de um *buttonPin* (linha 1), quando este apresenta um valor lógico 1 (HIGH) é enviado pela porta série o valor 'H' (linha 2). Caso essa condição não se verifique é enviado pela porta série o valor 'L'. (linha 4)

Comportando-se o ciclo if apresentado no exemplo 4, como um ciclo if de programação usado p.ex: na linguagem C.

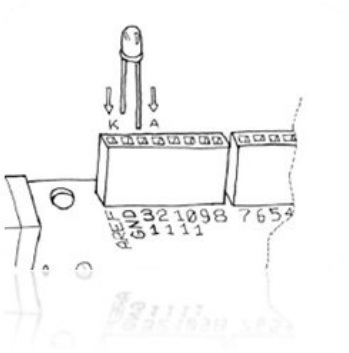
Para executar as instruções *Serial.println(data)* na função Loop, é necessário efectuar a declaração na função Setup da função *Serial.begin(int taxa_bps)*. Definindo o bit rate (bps) desejado para a comunicação.

Hardware vs Software

Torna-se necessário, à semelhança do 1º programa que se faz em C, o famoso "Hello World", efectuar também um programa inicial. Programa esse que não vai apresentar no ecrã a string "Hello World", fazendo, no seu lugar, um simples led piscar.
O programa para o efectuar é o seguinte:

Exemplo 5:

```
int ledpin = 13;
void setup()
{
    pinMode(ledPin, OUTPUT);
}
Void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitaWrite(ledPin, LOW);
    delay(1000);
}
```



Analisando o exemplo 5:

Na linha 1, foi efectuado a declaração da variável *ledpin*, sendo esta declarada como sendo do tipo integer.

As variáveis podem ser do tipo:
- char
- byte
- int
- unsigned int
- long
- unsigned long
- float
- double

Char - Ocupa um byte de memória a guardar um caracter.

Byte- Bytes guardam um número de 8 bits, vão do 0 ao 255. Não guardam números negativos.

Int - Guardam um valor de 2 byte, inteiro.

Unsigned int - Continua a ter um tamanho de 2 bytes, sendo um int, não contendo parte negativa.

Long - Permite guardar um número inteiro de 4 bytes. (32 bits)

Unsigned long - Semelhante ao long, não possuindo valores negativos. Aumentando assim o intervalo de valores positivo.

Float - Usada para designar números com componente decimal, são guardados em 4 bytes. (32 bits)

De seguida é inicializada a função Setup (linha 2), que permite definir o pino 13 como OUTPUT (linha 3).

Após a declaração das portas como INPUT/OUTPUT, tendo em conta a montagem desejada, inicia-se o ciclo Loop, que vai conter a base do nosso programa.

No caso apresentado, recorrendo à função digitalWrite (pino,valor), é possível atribuir o valor lógico 1 (HIGH) ou o (LOW) a uma saída definida.

O programa apresentado contém ainda a função delay (ms), que faz uma pausa no programa pelo tempo definido. Continuando depois com as instruções procedentes à instrução delay (ms).

O exemplo apresentado permite fazer com que um led pisque de 2 em 2 segundos. Led esse que está ligado ao pino 13 como configurado. O pino 13 possui uma resistência ligada em paralelo, não havendo qualquer implicação da ligação do led directamente a esta porta. O que não acontece nos restantes pinos de saída, sendo este exemplo apenas indicado para o uso da porta 13.

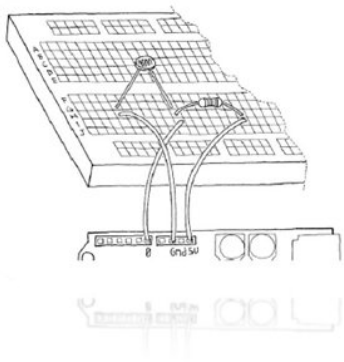
De seguida, vamos ver outro exemplo, que possibilita a leitura de valores analógicos, p. ex. de um LDR, um acelerómetro analógico, entre outros.

Exemplo 6:

```
int val=0;
void setup()
{
    pinMode(13, OUTPUT);
}
Void loop(){
    Val = analogRead(0);
    digitalwrite(13, HIGH);
    delay(val);
    digitaWrite(13,LOW);
    delay(val);
}
```

Analisando o exemplo 6:

Neste exemplo, vai-se proceder apenas à explicação do bloco de instruções da função Loop, pois a função Setup é em tudo semelhante à do exemplo 5.
A função analogRead(pino), lê o valor analógico e guarda-o, neste caso, na variável val. Este valor está compreendido entre 0 e 1023, sendo esta a resolução do conversor analógico digital (10 bits). A operação de conversão leva cerca de 0.0001s, pelo que a taxa máxima de leitura é de 10000 amostras por segundo.



Através da variação da entrada analógica, é possível fazer variar, no caso apresentado, o delay (ms) da montagem.

Sendo os exemplos 5 e 6 bastantes acessíveis à compreensão, é imperativo focar um ponto muito importante que possibilita responder à questão: *Como enviar dados através da porta série para o meu Arduino?*

Exemplo 7:

```
int val=0;
int ledPin=9;
void setup()
{
  pinMode(ledPin,OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  if (Serial.available() > 0)
  {
    val = Serial.read();
    Serial.print("Eu recebi:");
    Serial.println(val,BYTE);
    analogWrite(ledPin,val);
  }
}
```

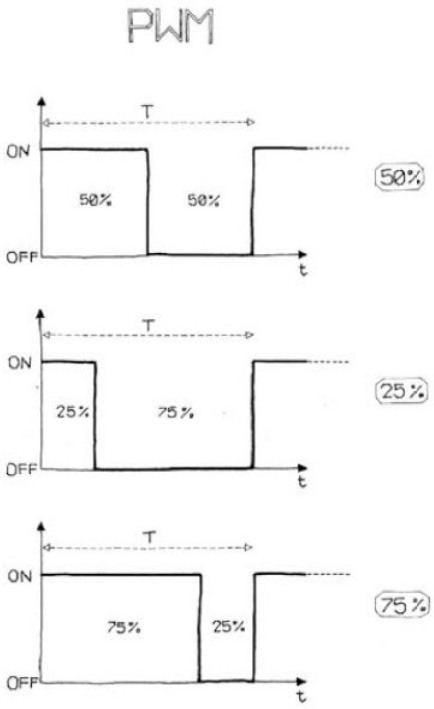
Analisando o exemplo 7

O ciclo if apresentado permite utilizando a função Serial.available() (linha 10), detectar se existem dados a serem enviados para o Arduino acima de um determinado valor. Em caso afirmativo executa o bloco de instruções (linha 11 à 16).

O valor lido através da função Serial.read() (linha 12) é guardado, neste caso, na variável val (linha 12). Sendo depois enviada através da função Serial.println(data) (linha 14) pela porta série e apresentada e/ou guardada por algum programa que possibilite leituras da porta série. Como é o caso do Software Arduino que possibilita essas mesmas leituras, mas não o seu armazenamento.

Através da função AnalogWrite (pino,valor) é possível utilizar a funcionalidade das saídas digitais PWM (Pulse Width Modulation). O parâmetro de entrada, valor da função analogWrite, varia entre 0 e 255, pelo que ao 0 corresponderá 0V e ao valor 255, respectivamente, 5V.

Por exemplo se o valor for 128, o valor de saída será 5V durante metade do tempo e 0V durante a outra metade. Originando um valor eficaz de aproximadamente 2.5V.



Conclusões

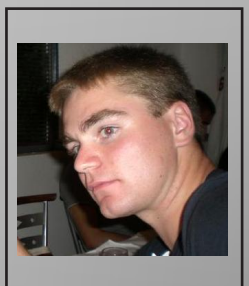
Este artigo tem como objectivo dar a conhecer uma "ferramenta" que como o leitor já deve ter reparado, possui enormes potencialidades.

O trabalho aqui exposto constitui uma ínfima parte do que existe para conhecer e aprender sobre o tema, sendo este documento uma tentativa para despertar o interesse e busca de conhecimento, que tantas vezes faltam nos dias de hoje.

Sendo a melhor solução adquirir um exemplar, caso tenha o leitor manifesto interesse em descobrir mais sobre esta "ferramenta".

Quanto a pormenores das funções utilizadas pelo software oficial Arduino, deve o leitor recorrer ao site oficial Arduino (http://www.arduino.cc) ou a tutoriais disponíveis online. Pois o saber não ocupa espaço de memória.

SOBRE O AUTOR



Nuno Pessanha Santos é um apaixonado pela área da electrónica e das telecomunicações, estando actualmente a frequentar na Escola Naval o primeiro ano do mestrado integrado no ramo de Armas e Electrónica na classe de Engenheiros Navais.

nuno.santos@portugal-a-programar.org

Nuno Santos

TopCoder

http://www.topcoder.com/

O TopCoder é um site que promove a competição como forma de criar software. Durante os vários passos deste processo, como design, arquitectura, desenvolvimento, testes, os programadores competem pela apresentação da melhor solução. No final, os vencedores são recompensados com prémios monetários e participação em torneios presenciais; os clientes obtêm software à medida das necessidades e qualidade garantida - afinal, é feito pelos melhores. Existem ainda provas quase semanais de algoritmia, com finais presenciais em torneios, uma categoria para alunos do secundário e um fórum utilizado por toda a comunidade.

Last.fm API

http://www.last.fm/api

A API do Last.fm permite retirar toda a informação contida nesta rede social de muita aceitação. É, assim, possível associar a um certo utilizador todo o seu perfil musical e encontrar outros utilizadores na sua aplicação com perfis semelhantes, por exemplo. Tudo o que é possível fazer com esta API está disponível neste site, assim como o seu download para variadas linguagens de programação.

Snipt

http://www.snipt.net/

O Snipt é um pastebin com agregamento de snippets por utilizador e tags, útil para organizar pequenos códigos que usamos do dia a dia, seja ele para gestão de um servidor como código para fazer dump a uma base de dados.

Codepad

http://www.codepad.org/

O Codepad é um pastebin tradicional, com as funcionalidades adicionais de compilação e execução do código copiado. Após inserir o código numa das linguagens suportadas (C, C++, D, Haskell, Lua, OCaml, PHP, Perl, Python, Ruby, Scheme e TCL), é apresentada uma página com o código devidamente colorido e o output que a execução do programa produziu, bem como eventuais erros de compilação ou execução. Útil quando não há compilador ou interpretador à mão, funciona também em muitos telemóveis.

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar
@portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org