Instalação de software

- Basicamente duas formas de se utilizar a instalação e gerenciamento de aplicativos em ambientes GNU/Linux
- Pacotes contendo binários prontos (i.e.: Já compilados)
- Pacotes contendo código fonte a ser compilado

Instalando a partir de código fonte

– Procedimento natural para qualquer desenvolvedor de software.

– Obter compilador necessário para transformar código fonte em código objeto.

– Facilidade de se poder personalizar ou parametrizar o software a partir do próprio código

Instalando a partir de código fonte

– Dadas as características das plataformas GNU/Linux, essa é uma prática muito comum.

– Desafio é tentar manter consistência, dependências e atualizações.

– Manter estrutura de diretórios em consonância com FHS.

/bin /usr/share/man /var/

/usr/bin/usr/local/bin /etc

Instalando a partir de código fonte

– Não existem regras. Cada software vai ter seus códigos objetos e bibliotecas. Para tanto, ferramentas auxiliam e orientam o processo de instalação

– Caso de aplicativos simples, não há necessidade ferramentas de auxílio.

Atividade

- Como criar um programa simples, do estilo “hello world” e compilá-lo com gcc.

Instalando a partir de código fonte

```
#include <sdtio.h>

Void main()
{

printf (“Olá Mundo cruel\n”);

}
```

```
#> gcc -o hello hello.c
```

Um código simples e sem dependências, pode ser facilmente gerado, chamando seu compilador. Nesse caso o GNU C Compiler.

Múltiplos Códigos

- ##############def.h#############
  #ifndef QUAD
  #define QUAD
  int quadrado (int nro);
  #endif
- ##############funcao.c#############
  #include <stdio.h>
  int quadrado (int nro)
  { return nro*nro; }
- ##############main.c#############
  #include <stdio.h>
  #include "def.h"
  int main(void)
  { int x;
  printf ("Forneca um valor\n");
  scanf("%d",&x);
  printf("O quadrado de %d eh %d\n", x,quadrado(x));
  return 0; }

Compilando múltiplos códigos

#> gcc -c funcao.c -o funcao.o

#> gcc funcao.o def.h main.c -o quadrado

Porém, se você alterar def.h ? O que fazer?

Instalando a partir de código fonte

– Entretanto, se dois arquivos são dependentes, para se compilar é necessários chamá-los em uma mesma linha.

#> gcc -o handshake hello.c bye.c

– Um código simples com poucas dependências, pode ser facilmente gerado.

– Extrapolar a ideia para códigos complexos, contendo diversos módulos é que se torna complicado.

Make e Makefile

– Algumas ferramentas e procedimentos padrões são adotados para facilitarem o dia a dia do administrador de sistema no que se refere ao controle e instalação do software.

– É o caso do programa make e dos scripts de configure

Make e Makefile

- Procedimento anterior ao Makefile (compilação) é a configuração.

– Script de configure: faz uma verificação e definição de parâmetros a serem utilizados durante a compilação. Grande parte dos códigos abertos distribuídos para Unix/Linux vai conter script de configure.

Ex:
#> ./configure –help
#> ./configure base_dir=/usr/local/loki
#> ./configure arch=i686

Make e Makefile

– Makefile: arquivo de sintaxe pré estabelecida que define regras e alvos para submeter ao compilador uma conjunto de código de fontes que possui diversas dependências e flexibilidade de modo de compilação. Esse arquivo é interpretado pela ferramenta make.

#> make
#> make test
#> make clean
#> make install
#> make minimo

Atividade

- Realizar Download do código fonte do **bashish** ou **dhcpd** e realizar o script de configure e se possível realizar o **make** e **make** para **teste**

- É possível encontrar outros alvos nos Makefiles desses softwares?

- É possível desinstalar os softwares previamente instalados a partir desse Makefiles ?

Sintaxe do makefile

```
# Linha de regra
 alvo: dependencia1 dependencia2 …
    Comando1
    Comando 2
```

- Onde o alvo pode representar um arquivo ou resultado a ser gerado. A dependência para um arquivo ou regra qualquer pode ser opcional (são arquivos recursivamente buscados em outros alvos)
- Comandos são tabulados por um “tab”
- Primeira regra existente no arquivo é considerada a regra padrão

Sintaxe do makefile

– Arquivo de Makefile, em geral é escrito com 'M' maiúsculo.
– Pode conter variáveis (x=string) declaradas e aquelas especiais e reservadas.
– Não exige estar atrelado à linguagem do compilador. Vide exemplo no artigo complementar (Linux Journal).
– Algumas regras são convenções de boas práticas como por exemplo:

- clean
- test
- install
- uninstall

Exemplo de makefile

```
#conta.c
    float conta (float x, float y)
    {
        return x*y;
    }


#atributos.c
    #include <stdio.h>
    int main()
    {
        float conta(float a, float b);
        float a,b,c;

        printf("Forneca uma valor para o primeiro fator\n");
        scanf("%f",&a);
        printf("Forneca uma valor para o segundo fator\n");
        scanf("%f",&b);
        c = conta(a,b);
        printf("O resultado é %f \n",c);
        return 0;
    }
```

Exemplo de makefile

```
CC=gcc

CFLAGS=-Wall

conta : conta.o atributos.o
    $(CC) -o conta conta.o atributos.o

conta.o:
    $(CC) -c conta.c -o conta.o

atributos.o:
    $(CC) -c atributos.c -o atributos.o

clean:
    rm -f conta conta.o atributos.o

install:
    cp conta /usr/local/bin/
```

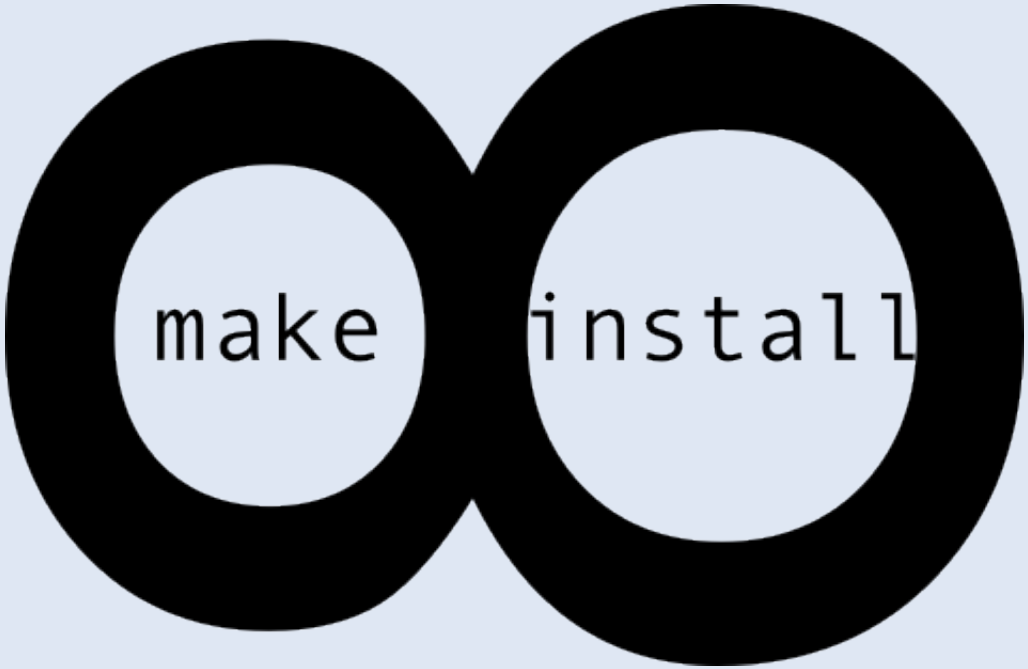
Exemplo de makefile

```
#> make

#> make conta

#> make clean

#> make install
```



Atividade

- Criar um makefile para que seja possível criar um arquivo compactado (zip, gzip, etc) de 5 originais (file1, file2, file3, file4, file5)
- Caso um dos arquivos não exista, o Makefile deve criá-los através do comando touch ou através da cópia de um outro arquivo qualquer
- Outras regras a serem colocadas são **test** e **clean**

Resolução

```
compacto.tar.gz: file1 file2 file3 file4 file5
	tar -czvf compacto.tar.gz  file1 file2 file3 file4 file5
file1:
	touch file1
file2:
	touch file2
file3:
	touch file3
file4:
	touch file4
File5:
	touch file5
clean:
	rm -f compacto.tar.gz
```