Comandos básico en Linux y uso de Makefile

Sistemas Operativos
DC - FCEN - UBA

1C - 2024

Pre-requisitos (repasar o aprender en casa)

Supondremos que no deberían tener problemas para:

- Conectarse a través de `ssh`.
- Moverse entre directorios y buscar archivos.
- Operar con archivos y directorios.
- Distinguir "allá" de "acá" (camino o path relativo y absoluto).
- Editar un archivo de texto.
- Escribir un `hello.c`, compilarlo, y ejecutarlo.
- Guardar la salida a un archivo.
- Distinguir entra la salida normal y la de errores.
- Filtrar líneas de texto.
- Buscar comandos.
- Buscar **syscalls**.
- Buscar ayuda.
- Buscar **stdlib**.
- Buscar en el manual.
- Buscar en **Google**.

Shell

- Intérprete de comandos
- Interfaz de texto.
- Ejemplos: `sh`, `csh`, `ksh`, `bash`.
- `$SHELL --version`

Manejo de archivos y directorios

Ver y ejercitar en casa

- Directorio absoluto: `/home/usuario`.
- Directorio relativo: `../` o `./` o nada.

Comandos:

- `ls` Lista archivos (directorio actual u otro parámetro).
- `cd` Cambia el directorio (parámetro o Home).
- `cp`, `mv` Copia/Mueve un archivo o directorio.
- `rm` Remueve un archivo/directorio.
- `mkdir` Crea un directorio.
- `rmdir` Elimina un directorio.

Manejo de archivos y directorios

Ver y ejercitar en casa

- `cat` Muestra por `stdout` el contenido de un archivo.
- `less` Muestra por `stdout` el contenido de un archivo (paginado).
- `echo` Escribe en `stdout` lo indicado por parámetro.
- `head` Escribe en `stdout` las primeras líneas de un archivo.
- `tail` Escribe en `stdout` las últimas líneas de un archivo.

Busqueda, Matching y Procesamiento

Ver y ejercitar en casa

- `find` Busca dentro del arbol de directorio.

```
find /home -name "*.c" -exec ls -al {} \;
```

- `grep` Busca coincidencias de cadenas de caracteres dentro de los archivos.

```
grep "hola" archivo.txt
```

- `awk` Procesamiento de texto.
  - `awk '{ print $1 }' archivo.txt`
  - `ls -l | awk '{ print $1 }'`
  - `ls -l | awk '{total += $5} END {print total}'`

Obtención de información

- `man` Muestra las páginas del manual.
- `apropos` Buscador en todas las páginas del manual.
- `pwd` Print Working Directory.
- `who` Quién está logueado.
- `uptime` Cuánto tiempo lleva prendido el sistema.
- `uname -a` Qué kernel de Linux se está ejecutando.

Manejo de entrada/salida

Archivos especiales:

- `stdout` Salida estándar.
- `stderr` Salida estándar (errores).
- `stdin` Entrada estándar.

Re-direcciones:

- `>` Redirige `stdout` a un archivo.

```
ls > lista_de_archivos.txt
```

- `2>` Redirige `stderr` a un archivo.

```
ls 2> lista_de_archivos.txt
```

- `&>` Redirige `stdout` y `stderr` a un archivo.

```
ls &> lista_de_archivos.txt
```

- `>>` Redirige `stdout` a un archivo en modo **append**.

```
ls >> lista_de_archivos.txt
```

- `&>>` Redirige `stdout` y `stderr` a un archivo en modo **append**.

```
ls &>> lista_de_archivos.txt
```

- `<` Redirige `stdin`.

```
sort < lista_de_archivos.txt
```

- `|` Pipe. Copia `stdout` a `stdin`.

Ayuda con los comandos de Shell

http://explainshell.com/

Permisos

Ejemplo:

```
$ ls -hl
-rw-rw-r-- 1 user group  445 mar 14 16:12 archivo
drwxrwxr-x 2 user group 4,0K mar 14 19:31 directorio
```

Permisos:

- `r` Read
- `w` Write
- `x` eXecute

Entidad:

- `u` User
- `g` Group
- `o` Others

Comandos:

- `chown baader:so archivo.txt`
- `chmod u+x archivo.txt`

Editores

- `vi`/`vim` Vi/Vim
- `nano` Nano

¿Cómo incorporar `make` a mi TP?

1. Crear un archivo de texto llamado `Makefile` que **describa**:
   - Los **targets** deseados.
     (`all`, `cliente`, `servidor`, `clean`, `entrega`, ...)
   - Los archivos involucrados.
     (archivos de código fuente, objeto, libs ...)
   - **Las dependencias** entre estas entidades.

2. Listo. Bastará situarse en el directorio del Makefile y decir

   ```
   make <target>
   ```

   para que `make` haga su magia.

Estructura genérica de un componente de Makefile

Léase: "El target se puede generar con estos comandos una vez generadas o conseguidas las dependencias."

```
targets ... : dependencias ...
Tab         comandos
Tab         ...
```

donde

target es el objetivo, puede ser un archivo de salida que la regla que se declare sea capaz de generar, o una acción determinada (e.g. `clean`) que se lleva a cabo cuando se invoca (e.g. `make clean`);

dependencias es uno o varios archivos de entrada necesarios para poder generar el target;

comando es una acción que, al ser ejecutada en un shell con todos las dependencias satisfechas, invoca los programas necesarios y genera el target.

Importante: todo renglón "comando" **debe** comenzar con exactamente un espacio del tabulador.

¿Se acuerdan?

```
suma.asm        nasm -f elf -o suma.o suma.asm
resta.asm       nasm -f elf -o resta.o resta.asm
producto.asm    nasm -f elf -o producto.o producto.asm
main.c          gcc -o main main.c suma.o resta.o \
                producto.o
```

Ejemplo

- La suma:

```
suma.o: suma.asm
	nasm -f elf -o suma.o suma.asm
```

- La resta:

```
resta.o: resta.asm
	nasm -f elf -o resta.o resta.asm
```

- El producto:

```
producto.o: producto.asm
	nasm -f elf -o producto.o producto.asm
```

Ejemplo

- El ejecutable:

```
main: main.c suma.o resta.o producto.o
	gcc -o main main.c suma.o resta.o producto.o
```

El primer `Makefile`

```
suma.o: suma.asm
        nasm -f elf -o suma.o suma.asm

resta.o: resta.asm
        nasm -f elf -o resta.o resta.asm

producto.o: producto.asm
        nasm -f elf -o producto.o producto.asm

main: main.c suma.o resta.o producto.o
        gcc -o main main.c suma.o resta.o producto.o
```

Makefile: comodines y variables

```
suma.o: suma.asm
        nasm -f elf -o suma.o suma.asm

resta.o: resta.asm
        nasm -f elf -o resta.o resta.asm

producto.o: producto.asm
        nasm -f elf -o producto.o producto.asm

main: main.c suma.o resta.o producto.o
        gcc -o main main.c suma.o resta.o producto.o
```

```
%.o: %.asm
        nasm -f elf -o $@ $<
```

- %: Es un **comodín**.
- $@: El **target**.
- $<: La primera dependencia.
- $^: Todas las dependencias.

El segundo `Makefile`

```
%.o: %.asm
	nasm -f elf -o $@ $<

%.o: %.c
	gcc -c -o $@ $<

main: main.c suma.o resta.o producto.o
	gcc -o $@ $^
```

`Makefile`: Targets especiales

Existen una serie de targets que se utilizan normalmente:

1. `make`: Sin especificar un etiqueta de **target**, se usa la primera etiqueta.
2. `make clean`: Elimina los archivos binarios
3. `make all`: Compila todo.
4. `make dist`: Genera un archivo comprimido con todo el contenido compilado.
5. `make install`: Instala lo compilado

Sólo haremos el 1, 2 y 3

Makefile: All Clean

```
.PHONY: all clean

BIN = main

ASMSRC = suma.asm resta.asm producto.asm
OBJ = $(ASMSRC:.asm=.o)
SRC = $(ASMSRC)


all: main

clean:
	rm -f $(BIN) $(OBJ)

main: main.c suma.o resta.o producto.o
	gcc -o $@ $^

%.o: %.asm
	nasm -f elf -o $@ $<

%.o: %.c
	gcc -c -o $@ $<
```

.PHONY nos indica que `all` y `clean` NO son archivos

Referencias (Makefiles)

- `man make`
- Manual completo de GNU `make`
  `http://www.gnu.org/software/make/manual/`

¿Preguntas?