Operating Systems

SSH, make, C and other bits needed for NSWI004

Vojtěch Horký

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University in Prague
Czech Republic

October 13, 2017

Running the Terminal

Running the Terminal

Where to find it

- GUI menu: *System* or *Utilities* or *Accessories*
- Application name: *Terminal* or *RXVT* or *Console*

Some other tips

- Ensure you use readable font (face and size) – you will be using it a lot
- Use tabs and multiple windows
- `export TERM=xterm` when keyboard/output behaves in a funny way

Remote Login etc.

Logging to a Remote Machine via SSH

```
ssh remotelogin@remote.machine.hostname
```

First login

```
The authenticity of host 'u-pl15 (195.113.21.145)' can't be
established.
ECDSA key fingerprint is
SHA256:U6u6eLekctQDr9uy4CKZJeDFjcCWqCI/v9owL1NODcE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'u-pl15,195.113.21.145' (ECDSA) to the
list of known hosts.
```

Configuring SSH Login

Put the following into `~/.ssh/config`

```
Host osy
    User USERNAME
    PreferredAuthentications publickey,password
    HostName uniform.ms.mff.cuni.cz
```

and you can login simply by typing `ssh osy`

Password-less login

SSH supports public/private key login.
You need to generate public/private key pair first.

Do I have one?

`ls ~/.ssh` look for `id_rsa` and `id_rsa.pub` or similar pair.

Generating a new one

```
ssh-keygen -t RSA
```

Pass-phrase improves security of the key but also slows-down its usage (unless you use SSH agent).

Password-less login (cont.)

Setting up the password-less login to uniform

1. Get your public key (`cat ~/.ssh/id_rsa.pub`)
2. Login to `uniform.ms.mff.cuni.cz` normally
3. Copy the public key to the end of `~/.ssh/authorized_keys` file
   `echo 'public-key-here' >> ~/.ssh/authorized_keys`
4. Test password-less login

Issues

- Check that `~/.ssh` has `rwx------` permissions
- Check that `authorized_keys` has at most `rw-r--r--`
- Check that you copied the public key correctly (single line etc.)

Password-less login on Windows

1. Install PuTTY (http://www.putty.org/)
2. Run PuTTY Key Generator
   - RSA key
   - Save public and private key pair
3. PuTTY configuration
   - Session – Host Name: `LOGIN@uniform.ms.mff.cuni.cz`
   - Copy public key to `authorized_keys` (right click inserts)
   - Connection – SSH – Auth – Private key file

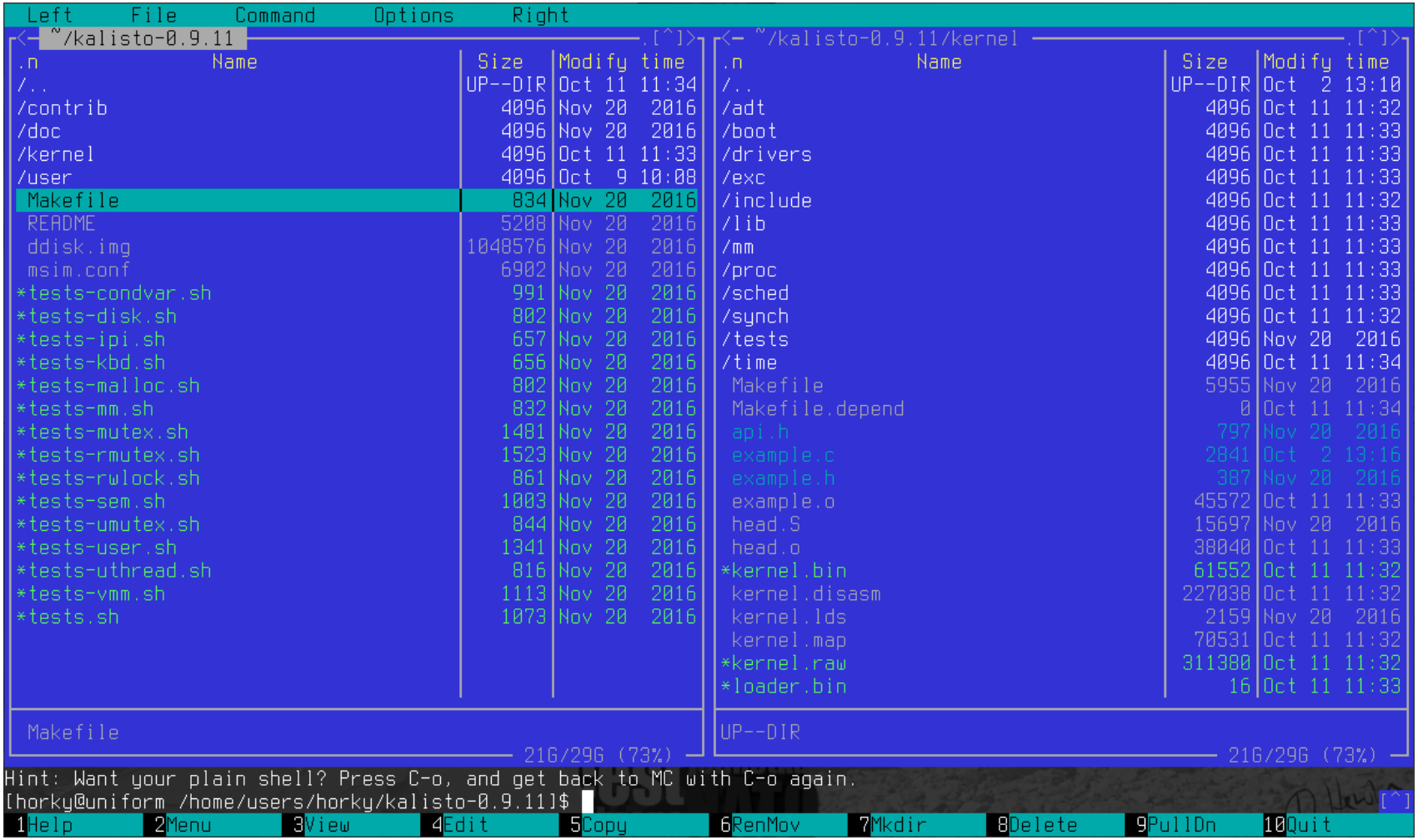
Using the Filesystem

The Basics

- `cd`
- `ls`, `ls -l`
- `find`
- `chmod`

Midnight Commander

Remember at least the `mc` command :-)



Downloading, Unpacking, ...

For downloading, use `wget`.

To unpack, use `tar` with proper switches.

- `x` to extract
- `f` to work with file (must use)
- `j` to work with `.bz2`
- `z` to work with `.gz`

```
tar xjf kalisto-0.9.11.tar.bz2
```

Accessing Remote Files via SSH FS

- `mkdir remote-fs`
- `sshfs osy:kalisto-0.9.11/ remote-fs/`
- Work with files on `uniform` locally in `remote-fs` directory
- But compile remotely!
- Do not forget to unmout: `fusermount -u remote-fs`

Inspecting Text Files

- `cat FILENAME` to dump contents to terminal
- `less FILENAME` to scroll through file (or pipe)
  - Use `/` to search
  - Use `q` to quit
- `grep` to search for a pattern

Using Make Build System

Running Make

If there is `Makefile` in the current directory, just type `make`.

Common targets

- `make clean` Remove all generated files.
- `make doc` Generate documentation.
- `make FILENAME` Regenerate particular file.
- `make -B …` Force rebuild.

Anatomy of a `Makefile`

```
all: program

.PHONY: all clean

clean:
	rm -f *.o program

program: main.o
	gcc main.o -o program

main.o: main.c
	gcc -g -Wall -c -o main.o main.c
```

Anatomy of a `Makefile` II

```
all: program

.PHONY: all clean

clean:
	rm -f *.o program

program: main.o
	gcc main.o -o $@

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

Anatomy of a `Makefile` II$\frac{1}{2}$

```
all: program
.PHONY: all clean
clean:
	rm -f *.o program

program: main.o
	gcc main.o -o $@

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

Anatomy of a Makefile III

```
CC = gcc
CFLAGS = -g -Wall
LD = gcc

all: program
.PHONY: all clean
clean:
	rm -f *.o program

program: main.o
	$(LD) main.o -o $@

%.o: %.c
	$(CC) $(CFLAGS) -c -o $@ $<
```

Anatomy of a `Makefile` IV

```
CC = gcc
CFLAGS = -g -Wall
LD = gcc
SOURCES = main.c
OBJECTS := $(addsuffix .o,$(basename $(SOURCES)))

all: program
.PHONY: all clean
clean:
        rm -f *.o program

program: $(OBJECTS)
        $(LD) $(OBJECTS) -o $@

%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<
```

Anatomy of a Makefile V

```
CC = gcc
CFLAGS = -g -Wall
LD = gcc
SOURCES = main.c
OBJECTS := $(addsuffix .o,$(basename $(SOURCES)))
DEPEND = Makefile.depend
...

%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<

-include $(DEPEND)
$(DEPEND):
    -makedepend -f - -- $(CCFLAGS) -- $(SOURCES) > $@ 2> /dev/null
    -[ -f $(DEPEND).prev ] && diff -q $(DEPEND).prev $@ \
        && mv -f $(DEPEND).prev $@
```

From C++ to C

Things You Cannot Use

- Classes and namespaces
- STL and templates in general
- Function overloading
- Exceptions, RTTI and type casting
- `new`, static initialization
- Streams

There are ways to bypass these limitations. Not all of them are nice.

Actually, it is possible to write OS kernel in C++ and use namespaces, exceptions etc. But the OS has to provide run-time support for these constructs. Kalisto provides no such support at the moment.

Missing Classes and Namespaces

- `object.function(…)` is actually `classname_function(object, …)`
- Prefix identifiers with namespace name
  - pthreads are a pretty good example
  - Rest of POSIX is definitely not

Missing STL and Templates

- Templates can be (lamely) emulated with X macros
- Generic data structures are possible
  - Simplified linked list in Kalisto
  - Full-fledged generic RB-tree, B+ trees or hash tables in HelenOS

Using `list_t`

```
#include <adt/list.h>

/* Structures that are in a list. */
struct my_struct { link_t link; ... }

/* List declaration and initialization. */
static list_t my_list;
list_init (&my_list);

/* Adding to a list. */
struct my_struct *x = malloc(sizeof(struct my_struct));
link_init (&x->link);
list_append (&my_list, &x->link);

/* Iterate through items, it points to my_struct */
list_foreach (my_list, struct my_struct, link, it) { ... }
```

No Function Overloading

- `_Generic` macro in C11
- Wrapper functions with different names

Error Handling

- Function always returns an error code
  - `EOK` or `0` on success
  - Other values passed through parameters
- Error is signalled by negative response, valid handle is always positive
  - `open()` could have had behaved in this way too
- Error is signalled via `errno`

Always check for errors. Especially in OS code!

Only `malloc` is available

- Check for errors
- Use `sizeof`
- Initialize afterwards

Static Initialization

- Unavailable directly
- Compiler extensions (`__attribute__((constructor))`)
- For OS, better to call them directly (ensures proper ordering)

Type Casting

No run-time support, static cast only.

Streams and I/O

- No `<<` and `>>` operators for I/O
- `printf` for formatted output
- `FILE *` and `fopen`, `fread`/`fwrite` and `fclose`

```
int i = 42;
const char *s = "Hello";
size_t x = sizeof(i);

printf("i = %d [%zuB], s = \"%s\"\n", i, x, s);
// i = 42 [4B], s = "Hello"
```

Beware of Undefined Behaviour

```
int i = INT_MAX + 1
```

- We might expect it wraps around to `INT_MIN`
- That is what the CPU instruction probably does
- But C standard says this is undefined so
  - it may work as we expect
  - or the whole program can do anything

Practical use? Optimizations ...

For example, knowing that `i++` on `int i = 0` may never overflow (because it is undefined) compiler it can safely assume that `i > 0`.

Other Bits

`no previous prototype for` function with no arguments

Function with no arguments has to be declared as (notice `void` parameters):

```
void driver_init(void)
```

Idiom for multi-command macro

```
#define SHORTCUT(x,y) do { cmd1(x); cmd2(y) } while (0)
```

Debugging macro

```
#define dprintf(fmt, ...) \
    printf("[DEBUG %s:%d %s()] " fmt, \
        __FILE__, __LINE__, __FUNCTION__, \
        #__VA_ARGS__)
```

Questions? Comments?