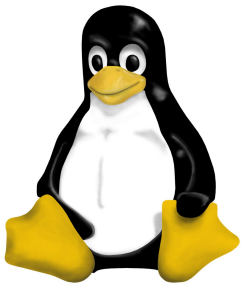
Introduction in Unix

Linus Torvalds

Ken Thompson & Dennis Ritchie



My name: John Donners

John.Donners@surfsara.nl

Consultant at SURFsara

And Cedric Nugteren

Cedric.Nugteren@surfsara.nl

Consultant at SURFsara

What will you learn:

Make a connection with a unix system

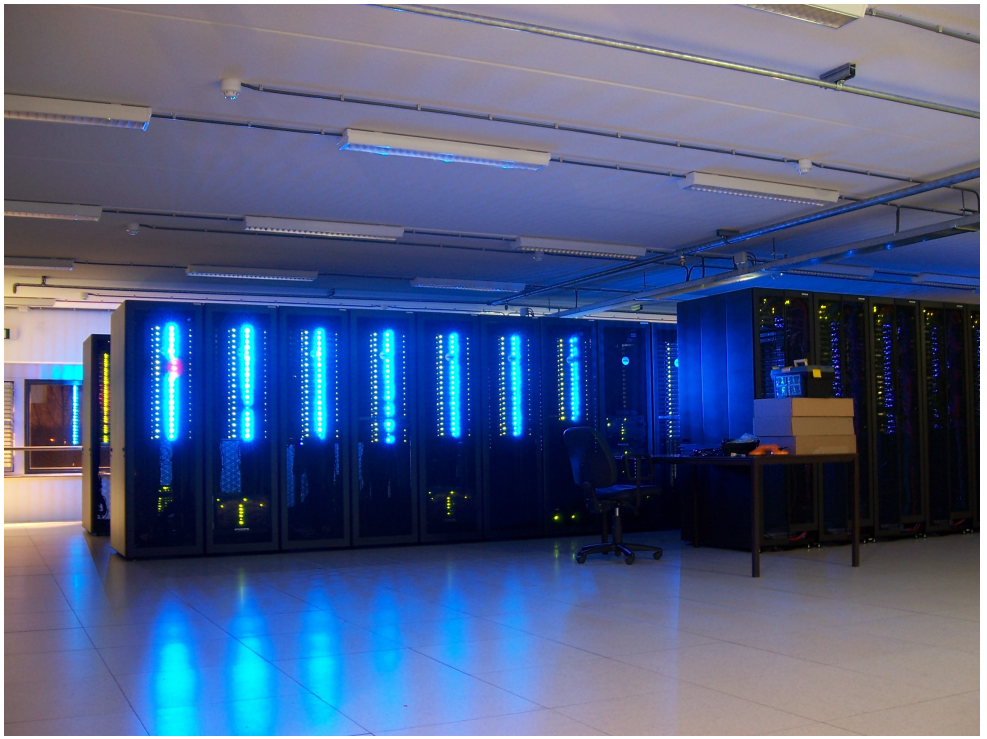
Learn something about the basics of

- command line
- shell
- common unix commands

We follow the tutorial at

https://surfsara.nl/systems/lisa/tutorial

The LISA system



Some 584 boxes, 2 of which for interactive use

Logging in to Lisa

From windows: use winscp and putty

http://winscp.net/eng/download.php

From Mac: use Terminal

From Linux: use (gnome-)terminal or xterm

In examples:
- user: wiltest
- password: *****

Putty:
- system: lisa.surfsara.nl
- port 22, ssh
- login as wiltest
- password *****

Terminal:
- ssh wiltest@lisa.surfsara.nl
- password: *****

After successful login:

```
wiltest@login4:~$
```

Login name

Name of the system

The system is ready to process your commands

First command

You type:

`date`

followed by an 'Enter'

```
wiltest@login4:~$ date
Mon Mar 15 14:26:17 CET 2010
wiltest@login4:~$
```

More commands

`w`	who is logged in
`whoami`	who am I?
`Whoami`	
`abc`	an unknown command
`uname`	name of the system

Say goodbye

```
logout
```

Behind the screens

Q: What is interpreting
the things you type in
at the $ prompt?
A: a program called 'shell'

You type:
`date`
The shell tries to find a program called 'date' and takes care that the system executes it.

An interactive program

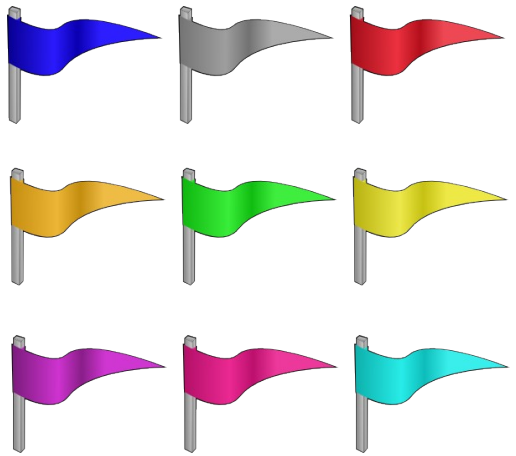
You type:

```
bc
4+7
5*9+10
quit
```

The program 'bc' is listening to your keyboard, not the shell. bc has no prompt.

Flags, parameters

The working of most programs can be influenced by flags (parameters), for example try:



```
date -u
bc -q
```

Flags, parameters 2

The shell gets this:

```
date -u
```

Shell locates program 'date',
puts '-u' in a place where
the program 'date' can find it,
and starts the program 'date'.
'date' finds the flag '-u' and
changes its internal workings.

Try:

```
date -x
```

Files and folders

We made some examples for you

Type:

```
svn export https://subtrac.surfsara.nl/userdoc/examples/lisatutorial
```

ls (list subdirectory)

Type:

ls

```
wiltest@login4:~$ ls
lisatutorial
wiltest@login4:~$
```

ls -l

```
wiltest@login4:~$ ls -l
total 3
drwx------ 3 wiltest wiltest 3 2010-03-15 15:45 lisatutorial
wiltest@login4:~$
```

directory = folder

ls and cd (change directory)

Type:

```
ls -l lisatutorial
```

```
wiltest@login3:~$ ls -l lisatutorial
total 2
drwx------ 2 wiltest wiltest 3 2010-03-22 14:05 c-example
drwx------ 2 wiltest wiltest 3 2010-03-22 14:05 jobs
drwx------ 2 wiltest wiltest 4 2010-03-22 14:05 scripts
drwx------ 2 wiltest wiltest 5 2010-03-22 14:05 simple
```

```
cd lisatutorial
```

```
wiltest@login4:~$ cd lisatutorial
wiltest@login4:~/lisatutorial$
```

Notice the change in prompt

~ = home directory

/ separates directory names

More cd and ls

Type:

```
cd
ls -l
```

Conclusion:

`cd`

brings you back to your home directory

```
cd lisatutorial
ls
cd simple
ls -l
```

```
wiltest@login4:~/lisatutorial/simple$ ls -l
total 5
-rw------- 1 wiltest wiltest 129 2010-03-15 15:39 bcin
-rw------- 1 wiltest wiltest 221 2010-03-15 15:39 file1
-rw------- 1 wiltest wiltest  94 2010-03-15 15:39 file2.txt
```

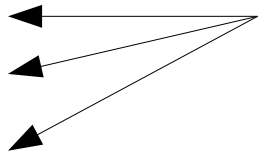
Q: Why 5 and not 3?
A: Not shown . and .. (current and parent directory)

Hidden files

File names, starting with '.', are not shown by default.

Use the '-a' flag of ls to make them visible:

```
ls -a -l
ls -l -a
ls -la
```



Often, you can combine flags

Where am I? **pwd** print working directory

Type:

```
pwd
```

Creating a directory

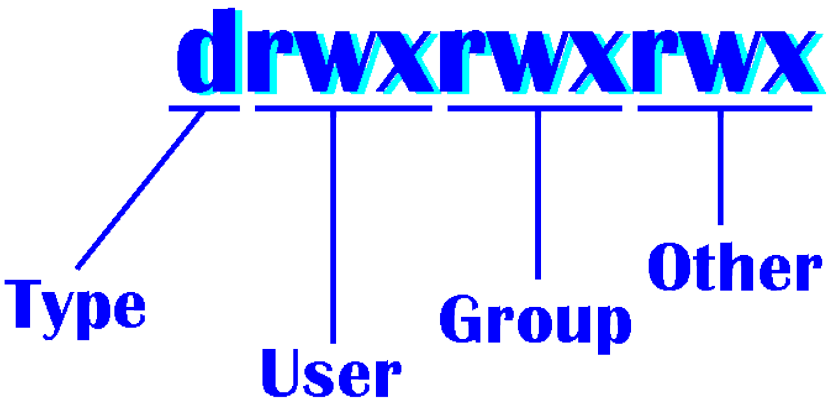
```
mkdir mydir
ls -l

drwx------ 2 wiltest wiltest   2 2010-03-19 13:49 mydir
```

Permission bits!

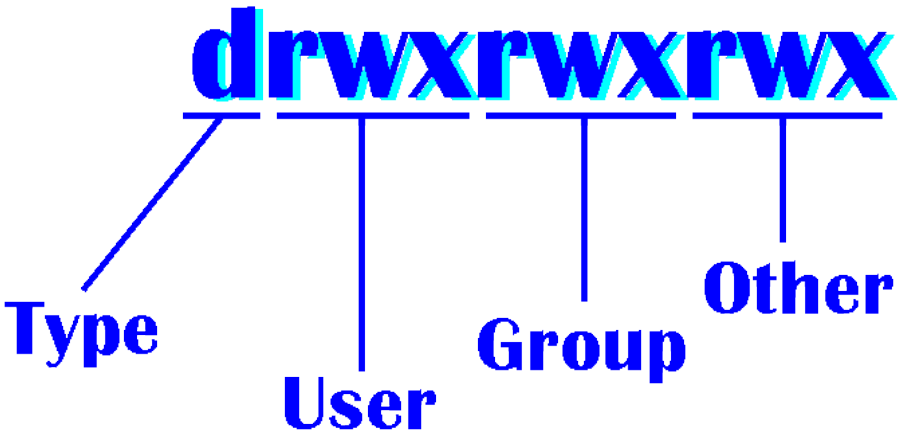
Each file or directory has 9 permissionbits associated

nr 0: '-' normal file
nr 123 read, write execute for owner
nr 456 idem for group
nr 789 idem for others

nr 0: 'd' directory
nr 123 'which files', 'create files', 'cd to' for owner
etc



Permission bits, example, chmod



Type:

```
cd
cd mydir
cd
chmod -x mydir
ls -ld mydir
cd mydir
chmod +x mydir
cd mydir
```

Remove x-bit from mydir

-d flag of ls: show properties of mydir, not the contents

Will fail

Content of files

Another command: **cat**

Type:

```
cd
cd lisatutorial/simple
cat file1
cat file2.txt
cat bcin
```

```
wiltest@login4:~/lisatutorial/simple$ cat bcin
# this is a file for bc.
# note: bc ignores lines starting with #
# let's make a complicated computation:
3+128*9877-123*(45+98)
wiltest@login4:~/lisatutorial/simple$
```

This looks like something you could feed into bc ...

Standard input, output and error

Every program (bc, shell, …) has three predefined input/output files associated:

Standard input (stdin): normally your keyboard
Standard output (stdout): normally your screen
Standard error (stderr): normally your screen

stderr is for error messages (in general)

In the 'simple' directory, type:

```
bc < bcin
```

Instructs the shell, that bc should take stdin from file 'bcin'

More redirection

In the 'simple' directory:

redirection of stdin to a file

```
bc < bcin
```

redirection of stdout to a file

```
bc < bcin > bcout

cat bcout

cat bcin | bc
```

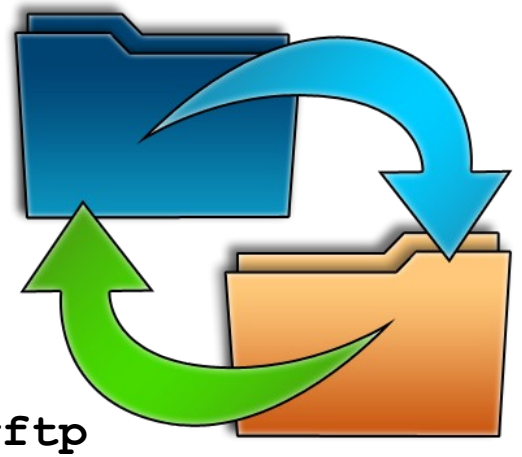
'pipe': output of cat goes to input of bc

```
cat bcin | cat | cat | cat | bc | bc | cat
```

Transporting files

Create on your system a simple text file 'example.txt':

- Ubuntu: use gedit
- Mac OS: use TextEdit
- Windows: use notepad



Install a file transfer program:

- Ubuntu: gftp `sudo apt-get install gftp`
- Mac OS: cyberduck from http://cyberduck.ch/
- Windows: winscp from http://winscp.net/eng/download.php

Use 'ssh2' or 'scp' as protocol

Click yourself a new directory on Lisa, and put 'example.txt' in that directory.

Try to download a file from Lisa to your system.

Tips about file names

- Do NOT use spaces: my file.txt → my-file.txt

- Use only these characters:
  `abcdefghijklmnopqrstuvwxyz`
  `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
  `0123456789`
  `_-.`

- Suffixes less important than in Windows,
  but it is wise to use **`.txt`** for simple text files, **`.jpg`** for jpeg files, etc.

- Upper and lower case do matter:
  Myfile != myfile

Some commands

bc *calculator*	**bc**
cat *concatenate*	**cat one**
	cat one two > three
cd *change directory*	**cd lisatutorial**
chmod change perm. bits	**chmod +x script**
cp *copy*	**cp one two**
env *print environment*	**env**
less *view file*	**less myfile**
man *manual*	**man bash**
ls *list subdirectory*	**ls**
	ls -l
	ls -d mydirectory
mv *rename*	**mv foo bar**
nano *editor*	**nano myfile**
pwd *where am I*	**pwd**

Redirection: > < >> |

Create simple text files

Methods:

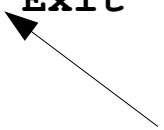
- create a file on your system, and copy to Lisa
or
- create file on lisa using an editor, for example 'nano'

Type:

```
nano
```

```
GNU nano 2.2.4                    New Buffer
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```



Cntrl-X

Inspect files with less

Less is a browser for text files

" Less is more than more "

Example, type

```
env
```

and then

```
env | less
```

Commands in less:

```
- q       quit
- space   one page ahead
- g       to start of file
- G       to end of file
- /text   search text
- n       next occurrence
- N       previous occurrence
```

… and many more

Wildcards

CD to the directory lisatutorial/scripts

type:

```
ls
ls -l p*
ls -l p*d
ls -l pr*
echo pr*
ls -l *.c

etc
```


What is happening:

The shell is expanding **`p*`**, and presents the results to **`ls`**. **`ls`** sees these parameters:

```
-l parm  prod.c
```

Scripting

Exercise

Create the following file, and call it 'script'.

```
#!/bin/bash
echo "3 + 6" | bc > bcout
cp bcout result.txt
echo "contents of result.txt"
cat result.txt
```

compute 3 + 6
result in file bcout

copy bcout to result.txt

A script is a kind of a workflow:
shell commands are executed.
Conditionals etc available.

Type:

```
chmod +x script
./script
```

Shell variables

Type:

```
v1=John

echo $v1

v2="John   and Mary"
echo $v2
echo "$v2"

example="a"
w=${example}b
echo $w
```

assign string "John" to shell variable v

three spaces

assign string with spaces to v

Notice the different results

Use {example} to prevent $exampleb

Allowed characters in variable names: a-z A-Z 0-9 _

Environment variables

Create environment variable:

```
export ev=horse
```
or
```
ev=horse
export ev
```

Environment variables are copied to the environment of programs (scripts) you start. Shell variables are not copied.

Cd to **`lisatutorial/scripts`**
Type:

```
myvar=one
yourvar=two
export yourvar
cat envtest
./envtest
```

The PATH environment variable

Type:

`echo $PATH`

you get something like:

```
/sara/sw/modules-ng-64/wrappers/sara:/bin:/usr/bin:
                    /usr/bin/X11:/usr/local/bin
```

The shell will search for programs in the directories:

```
/sara/sw/modules-ng-64/wrappers/sara
/bin
/usr/bin
/usr/bin/X11
/usr/local/bin
```

NOTE: no `'.'` in PATH

The HOME environment variable

Type:

`echo $HOME`
you wil get something like:
`/home/wiltest`

Type:

`echo ~`

Now it will be clear what the `'~'` in your prompt means.

Example: extend the PATH variable to search also for programs in **`$HOME/bin`**:

`PATH=$HOME/bin:$PATH`

Some more scripting

This example we already made:

```
#!/bin/bash
echo "3 + 6" | bc > bcout
cp bcout result.txt
echo "contents of result.txt"
cat result.txt
```

This script is written for the program **bash**, the shell. Other script eating programs exist: python, perl, ruby and many more

copy this example (called 'script') to script1:

```
cp script script1
```

and edit script1 to become:

```
#!/bin/bash
echo "$1 + $2" | bc > bcout
cp bcout result.txt
echo "contents of result.txt"
cat result.txt
```

$1 will become the first parameter you give, **$2** the second

And activate the script: `./script1 5 7`

Compilers

In short: a compiler takes as input a human readable and human written text, containing instructions what to do.

The output of the compiler is a file, which contains instructions for the processor: so it will run very fast.

Example:

cd to lisatutorial/c-example

Have a look at the file prod.c :

```
// this program reads from standard input two numbers.
// these numbers are multiplied and put on standard output.

#include <stdio.h>
int main()
{
  double a,b,c;
  scanf("%lf %lf",&a,&b);  // read a and b from stdin
  c = a*b;                 // compute the product
  printf("%lf\n",c);       // print the product
  return 0;
}
```

Compilers 2

Type:

```
gcc  -o prod prod.c

echo 3 8 | ./prod
```

This example was in the language 'C'. There are more languages: Fortran, C++, Fortran90, Java and more.