Introduction to UNIX

Start MobAgent – Windows
Login to Lisa – with key pair (3)
Login to Lisa – with key pair (4)

Manage your SSH keys

Here you can manage your public ssh keys. Please note that these keys are not used by the git server.

<table>
<thead>
<tr>
<th>Title</th>
<th>Fingerprint</th>
</tr>
</thead>
</table>

Add key
Login to Lisa – with key pair (5)
Login to Lisa – with key pair (6)

* multiple SURFsara services (compute, data storage, front-end storage)  *
* consecutive multiple impacts cannot be prevented. See the preliminary  *
* downtime schedule on our site: https://userinfo.surfsara.nl/movedc.   *
* The move will affect computing and data storage services in the period of *
* August 1 until November 15, 2016.                                     *
*                                                                 *
* State-of-the-art data center                                          *
* SURFsara is moving its infrastructure to this new state-of-the-art data *
* center so we can meet the needs of our clients in a fast, cost-effective *
* and sustainable way. The building, a 72 meter high tower, has a total *
* equipment floor of 5000 m2 and a designed PUE (power usage effectiveness) *
* of 1.22. SURFsara will rent 800 m2 for the national HPC data center.   *
*                                                                 *
* Questions?                                                            *
* Call or email your advisor, or contact our helpdesk: helpdesk@surfsara.nl.
*                                                                 *
***************************************************************************

Accounting information:
Your account is about to expire in 87 day(s)

Filesystem Quota Used Avail Use% Server
/home/sdemo001 200.0 GB 6.3 MB 199.99 GB 0%  fs12
sdemo001@login1:~$
Hands on – Let’s Play!

Next slides copied from previous years (with thanks to)
• Willem Vermin (retired)
• John Donners (colleague)
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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w</code></td>
<td>who is logged in</td>
</tr>
<tr>
<td><code>whoami</code></td>
<td>who am I?</td>
</tr>
<tr>
<td><code>Whoami</code></td>
<td>an unknown command</td>
</tr>
<tr>
<td><code>abc</code></td>
<td>name of the system</td>
</tr>
</tbody>
</table>
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
```
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:

```bash
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 permission bits 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
```

Type
User
Group
Other

Remove x-bit from mydir

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

Will fail
Content of files

Type:

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

```
wiltest@login4:~/lsatutorial/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:~/lsatutorial/simple$
```

Another command: `cat`

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`

- 'pipe': output of cat goes to input of bc
  
  `cat bcin | 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/

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:
  a b c d e f g h i j k l m n o p q r s t u v w x y z
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  0 1 2 3 4 5 6 7 8 9
  _ -. 

- 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
eenv
print environment
eenv
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
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

What is happening:

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

-l parm  prod.c

etc
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
```

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`  
  assign string "John" to shell variable v

- `echo $v1`
  three spaces

- `v2="John and Mary"`
  assign string with spaces to v

- `echo $v2`
- `echo "$v2"`

Notice the different results

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

Use `{example}` to prevent $exampleb

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

Create environment variable:

```bash
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:

```bash
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 will 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:

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

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

```bash
cp script script1
```

and edit script1 to become:

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

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

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

$1 will become the first parameter you give, $2 the second.
Thank you for listening!

UNIX

is user friendly
it's just selective about who its friends are
Thank you for listening!

UNIX is user friendly
it's just selective about who its friends are