Basics of working with UNIX

Most modern computers (and similar devices, e.g. tablets or telephones) are equipped with complex software that makes up the operating system. You've definitely used Microsoft operating systems from the Windows family. You may also have come across Android (i.e. a version of Unix) from Google or iOS from Apple. In most cases, the system has a so-called GUI (Graphical User Interface). These systems are basically similar and launching a web browser or disk browsing for example, are not a challenge in any of them.

However, not every computer has a GUI, this for example applies to large computer systems used in numerical calculations (e.g. determining the aerodynamic properties of a car using the Fluent program). In this case, it is not possible to use the mouse and watch something on the screen, because the computer is in the server room, sometimes in another country. To use such a remote computer, we must connect to it using a special program that will allow us to issue commands in text mode.

For the needs of this laboratory, everyone received a card with a login and password. They are valid until the end of the semester and you can use them to log into our school server `info3.meil.pw.edu.pl` also from outside the campus.

If you're using Windows software, it's most convenient to use the free PuTTy program to connect. After starting, provide the following data:

- `Host Name` - host name (`info3.meil.pw.edu.pl`),
- `Port` - The number of the port from which we want to connect (`22`),
- `Connection Type` - connection type (`SSH`).

After clicking `Open`, a black window will appear asking for the login and then for the password. Note: the characters in the password you enter are not marked in any way (e.g. asterisks) and it is not visible. After logging in, you'll see information about the date, license, system version, etc. ending in:

```
Last login: Thu Feb 21 06:23:38 2013 from xx.xx.xx.xx
stud01@eto:~$
```

When logging in from a Linux system, we use the `ssh` command:

```
ssh stud01@info3.meil.pw.edu.pl
```

Port 22 is the default port used by the command, you do not need to specify it explicitly.

The line, which will appear:

```
stud01@eto:~$
```

so called the prompt and means that as the user `stud01` we are logged in to the `eto` computer. Between the characters `:` and `$` there is a directory in which we are currently located. In this case we are in the home directory. The sign `~` is an abbreviation whose expansion is `/home/students/stud01` - the home directory.

Exercises

The first fight

Enter the command `date` in the console and press enter. The computer displays the current (in his opinion) date and time. The line ending with '$' will be displayed below again, indicating that the computer is waiting for new commands. Use the up and down arrows to scroll through the command history. Clicking the `Ctrl + R` keys will activate the option to search commands in history. The classic prompt will be replaced by

```
(reverse-i-search)`':
```

We search by entering the next characters from the command sought, and the system will show a hint under the colon. Another click of `Ctrl + R` will show the next suggestion. Another feature is the ending of names. If you enter `dat` and press `tab` 2x, a list of commands starting with `dat` will be displayed. If there is only one such command, the name will be completed. Remember these tricks - they make work in text mode much easier.

Navigating directories

Working in text mode, we always work in a directory, so-called current directory. If we run a program, e.g. a simple program that reads data from a file from Computer Science I, it will read the files in this directory. Any program that you will use and which needs a file or directory name (e.g. for copying) can be given in two forms. The first is the so-called absolute path, starting with / e.g.

```
/home/students/stud01
/usr/bin/bash
/etc
```

Check which directory you are in by entering the `pwd` command.

To change the directory, the `cd` command is used, e.g.

```
stud01@eto:~$ cd /tmp
stud01@eto:/tmp$ pwd
/tmp
stud01@eto:/tmp$
```

To return to the home directory, enter `cd ~`. The `~` character always means the user's home directory.

In addition to the absolute paths, you can specify a relative path, in which:

- `file` means a file in current directory
- `directory/` means a directory, which is a subdirectory of the current directory
- `../` means the parent directory,
- `/` is the root directory (beginning of each absolute path),
- `.` and `./`represent the current directory (the one returned by the `pwd` command).

Experiment now with browsing the catalogs. If entering paths makes you bored, try the `mc` program. It allows, among others for graphical navigation through the directory tree. Exit the program by clicking the 'F10' key or by typing 'exit'.

Creating and deleting directories

To create directories, use the `mkdir` command, e.g.

```
$ mkdir directory_name
```

and to check the contents of the current directory, the command `ls`. Create A, B, C and D directories, each inside the previous one. To do this, you'll have to create directory A, go to it, then create B, etc.

You cannot delete a directory that has content with `rmdir`. To do this, use the `rm -r` command. The characters `-r` after the program name `rm` are the program argument and mean that the directory is to be removed recursively. Most commands have similar arguments, for example `cp -r` copies directory with contenst, and `ls -l` shows the contents of a given directory as a list containing various information about the files.

Basic operation of files and directories

The `echo` command prints the string that is given as its argument to the screen. This can be used to create the first file (with the meaning of the symbol '>>' will be in the next class)

```
$ echo "First file" >> file.txt
```

To display the contents of the file on the screen, use the `cat` command

```
$ cat file.txt
```

Copying and moving

To copy, use the command `cp [WHAT] [WHERE]`. Create a directory now and copy your file to it. It should look like this:

```
$ cp file.txt directory
```

To move / rename a file or directory, use the `mv [WHAT] [WHERE]` command. Go to the new directory and change the file name. Then delete the file with the `rm [WHAT]` command.

You do not always have to provide the full name of the file / directory that you want to use as the program argument. Clicking on the `tab` button will complete the name. If the hint is not unambiguous, after pressing the `tab` key twice, the console will display the names of all files / directories starting with the entered characters.

Help

The vast majority of the text mode commands have decent documentation available immediately, e.g .:

```
$ man rm
$ rm --help
```

In the case of the `man` command, we get more extensive documentation. The text is scrolled with the arrows. To end browsing, press `Q`. Check the instructions for the `who`, `whoami`, `finger` and `date` commands. Check how they work.

Tar program

The tar program is used to package and extract a directory tree and files into one file. The resulting file may not necessarily be smaller than the original files. Only using compression will reduce the size. Prepare several files for packaging first:

```
$ mkdir a
$ cd a
$ mkdir b
$ echo asdasd >> ./b/c
$ cat ./b/c
asdasd
```

Now pack them and then view the contents of the archive using the program `mc`.

```
$ tar -cf test.tar b
$ ls
b  test.tar
$ ls -l
```

Check the contents of the directory you have packed, then delete it and unpack the archive.

```
$ ls
b  test.tar
$ rm -r b
$ ls
test.tar
$ tar -xf test.tar
$ ls
b  test.tar
```

Check the volume of the created archive with the `ls -la` command. Then package the same files with the additional `z` flag. Adding this parameter starts compression with the `gzip` program. Change the archive extension to `tar.gz`. Check if the resulting file is smaller.

Simple scripts

The most important aspect of working in text mode is the ability to create scripts, i.e. subsequent commands stored in the file executed as if we were typing them from the keyboard. You will learn more about advanced scripts in the next laboratories, but a first one you will write today.

Simple and convenient text editors are `nano` and `vim`. Run the `nano` program with the `nano script.sh` command and write the first script to it:

```
#!/bin/bash
echo 1
echo 2
```

Then change the permissions to allow our script to run:

```
$ chmod u+x skrypt.sh
$ ./skrypt.sh
```

The `chmod` command is used to change the file permissions. `u` means the user (i.e. you) to whom we want to give +the right to run `x` scripts / programs.

Variables

Bash supports variables just like C language. There is no typing here. To create a variable containing text, write:

```
var="Text"
```

However, the result of a program can be saved to the variable as follows:

```
variable=$(pwd)
```

To display the variable's value, write:

```
echo $variable
```

Second script

Write a script which will prepare the directory structure:

- AA
  - BB
    - file.txt
  - CC
    - DD
    - file.txt

The file `file.txt` should contain the date returned by the command `date`.

Modify the script so that the name of each directory begins with the prefix passed to the script as an argument.

- before_AA
  - before_BB
    - file.txt
  - before_CC
    - before_DD
    - file.txt

To run the script with arguments, one can enter these arguments after the script name when executing:

```
./script.sh arg1 arg2
```

We access the arguments `arg1` and `arg2` from within the script you can use the variables `$1 - $9` (in this case the variables `$1` and `$2`).

Loops

Prepare a script containing:

```
#!/bin/bash

for i in *.txt
do
  cp $i $1_$i
done
```

and run it in a directory containing files with the extension `.txt`. How does it work? Remember the script's argument! What happens when you don't give the argument?

Write a script that creates a directory with the name given as the first argument to the script `$1` and copies all the `.txt` files into it, adding the prefix given as the second argument `$2` to their name. What happens if you don't provide arguments to the script?

GUI

If you're using Linux, you can copy data from a remote computer to a local computer using the `scp` command. Being logged in to the `eto` server all the time, create a `copyme` file in your home directory. Log out using the command 'exit' or the keyboard shortcut 'Ctrl + D'.

Open the console on the local computer and download the created file using the command

```
scp stud01@info3.meil.pw.edu.pl:~/copyme ./
```

In general, the syntax has the form

```
scp login@host:sciezka_do_pliku gdzie_zapisac
```

The `scp` program works like `cp`, except that the target (or source) is on another computer that supports ssh connections. The `WinSCP` program was created for computers with the Windows family system, which allows data to be copied in graphic mode.

Dessert! *

Pre-props

Check what the `write` program is for using the `man` command. Also compare the results of the commands:

```
$ who
$ who | awk '{print $1}'
$ who | awk '{print $2}'
```

Note the | symbol, which means redirecting the output of one program to the other. This notation is called a "pipe".

Spammer script

Create a script:

```
#!/bin/bash

for u in $( who | awk '{print $1}' )
do
   echo $u
done
```

and check how it works. Then modify it to "spam" all logged in users.