Introduction to the Shell

Ralf B. Schäfer

2020-11-04

What is a Shell?

- Interface between user and operating system (OS)
- Allows user to execute tools and programs via command line interface (CLI)
- Standard interface in the past, nowadays complemented by graphical user interface (GUI)

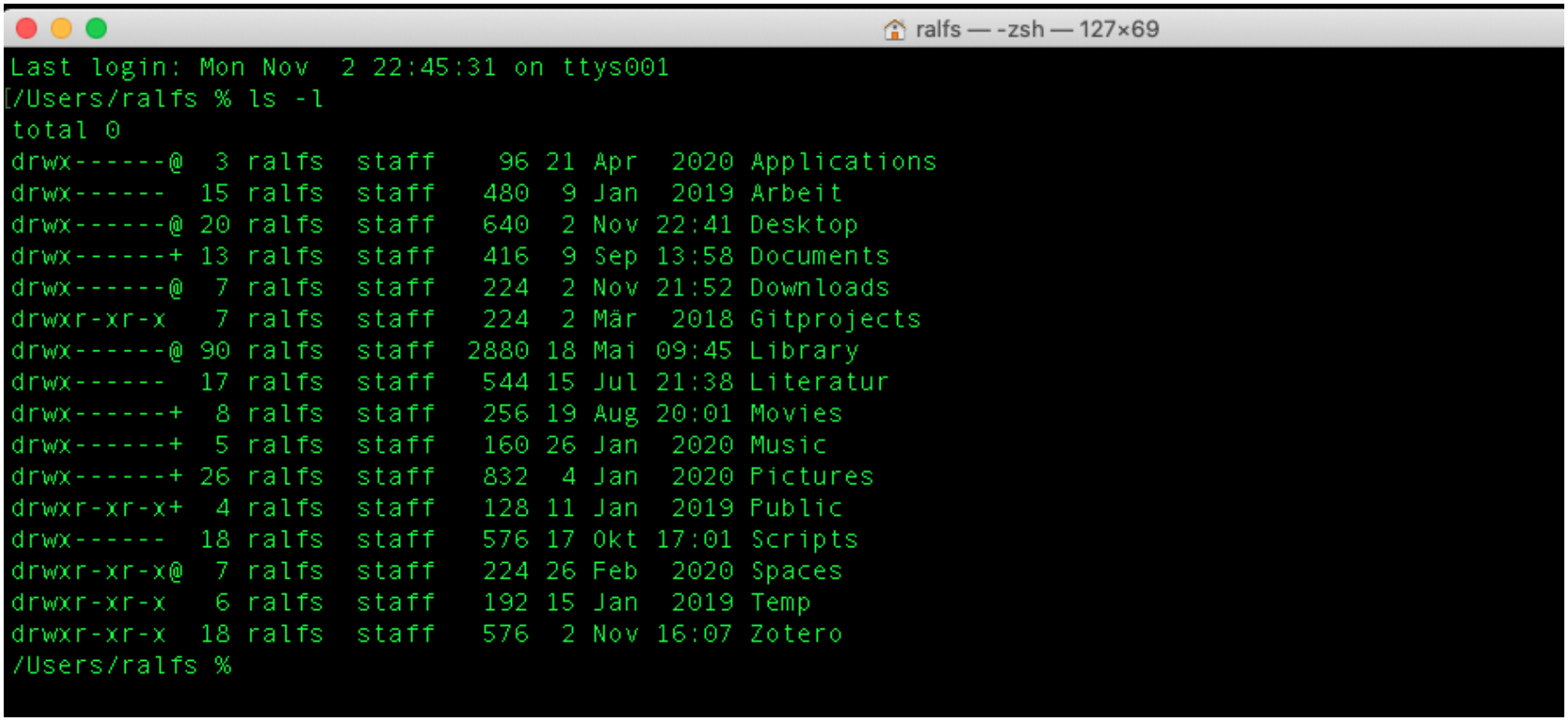


Figure 1: Example of Terminal window to access the Shell

What can I do with the Shell?

- System administration and file management (and messing up the system!)
- Automation of (repeated) tasks
- Creating reproducible analyses (as opposed to GUI-based)
- Access clouds, clusters and remote computers

In comparison to the GUIs of modern OSs, the Shell is quite similar across different OSs.

A motivating example

Task: Split 12 publications into single pages and convert each page into a png file that can be added to a text document (habilitation thesis)

(Semi-)manual solution:

1. Create several directories by hand
2. Find (and potentially buy) software to split pdfs, apply to each pdf
3. Then find (and potentially buy) software to convert pdf to png and apply to each of ~150 pdfs

A motivating example

Task: Split 12 publications into single pages and convert each page into a png file that can be added to a text document (habilitation thesis)

Automated solution using the Shell (may require installation of (free) additional tools depending on OS):

1. Create directories

```
for file in *.pdf; do mkdir "${file%\.*}"; done
```

2. Split pdfs into single pages

```
pdfseparate File.pdf %d.pdf
```

3. Convert pages into figures

```
for file in *.pdf; do sips -s format png $file
--out "${file%\.*}".png && rm -f "$file"; done
```

A motivating example

Automation saved time (of boring work) and money (no software purchase required)

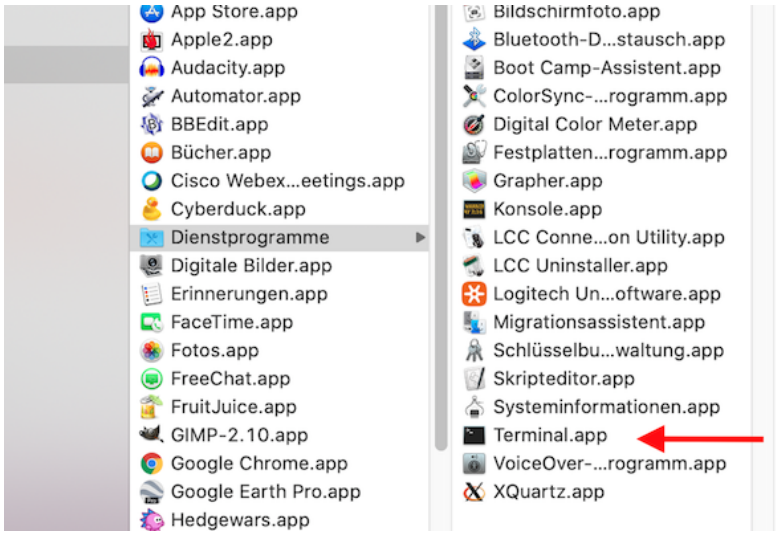
However, at times manual solutions can be more efficient:

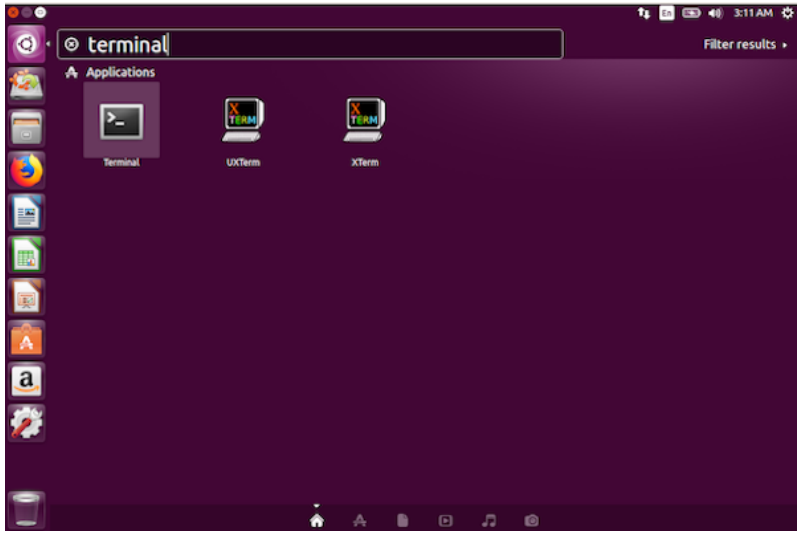
Figure 2: https://xkcd.com/196/

How can I use the Shell?

You need to launch a Terminal Window to access the Shell

- Linux/OS X : Start Terminal from programs
- Windows: Start bash.exe or Cygwin terminal (depending on installation)





Basic Shell syntax

Consists of command and arguments preceded by `-` or `--`

```
cd ~
ls -a -l
```

`cd` is the command to **c**hange the **d**irectory, `~` is a short cut for the user directory

`ls` is the command to list files in the directory, the argument `-a` displays hidden files, i.e. files preceded by a `.` and `-l` formats the output as list

`-a` and `-l` can be combined:

```
ls -al
```

Which commands exist?

- Different types of shells (e.g. Bourne, Bash (Bourne Again Shell), zsh - the Z Shell (extension of bash))
  - Bourne Shell reference manual
  - Bash reference manual
  - zsh reference manual
- Additional programs (and therefore commands) are available in most OSs or can be installed (and then called via CLI (e.g. tesseract))
- Wealth of information on the web including forums, cheat sheets, online tutorials

Help for specific commands

- Manual for specific commands (exit with `q`), examplarily for `ls`:

```
man ls
```

- For many commands, short manual and syntax help is available via:

```
command --help
command -help
```

Input and output

- Commands are generally connected to standard input (called `stdin`), standard output (`stdout`) and error log (`stderr`)
- < and > are used to assign specific files to input or output

```
command < input_file
command > output_file
command < input_file > output_file
```

How does the system know where a file is located?

- As in R (remember `setwd`), we execute commands within a working directory (set with `cd`)
- absolute path: full path on device, begins with `/`
- relative path: relative path on device, does not begin with `/`

For example, the file `/Users/ralfs/Example.png` can be addressed via full path or, assuming our working directory is `/Users/ralfs/`, with `Example.png`.

Specific concepts: Pipes

Pipes (implemented via |) are a powerful tool to combine commands, where the output of a command is fed into the following command

```
command_1 | command_2 | ... | command_N
```

Note that this concept has been implemented in R in the dyplr package using %>% (for details check out this tutorial)

Specific concepts: Loops

Loops are programming constructs that repeat one or multiple commands for each object in a given list

```
for object in list
do
  command using object
done
```

You may know the concept from R:

```
for(i in list){
   variable <- command(i)
    }
```

Special characters

Several characters are evaluated to have a special meaning, for an overview follow this link.

A few important ones:

`\n` - newline

`\` - next character is interpreted as normal character (escape)

`*` - wildcard that matches zero or more characters

`?` - wildcard that matches exactly one character

`[characters]` - the characters in brackets need to be matched

`[!characters]` - the characters in brackets must not be matched

Defining variables

Defining variables is done via =, where variable names are typically in UPPERCASE letters:

```
VAR1="Text"
VAR2=Number
VAR3="$(command)"
command $VAR1
```

A defined variable is called with `$`, `$( )` is used to run a command in another command.

Running Shell commands from within R

Shell commands can be run from within R using the `system2()` function.

For example, to list the files in a directory you can use:

```
system2(command = "ls")
```

Setting additional arguments:

```
system2(command = "ls"
        args    = c("-l", "-a"))
```

Redirecting output:

```
system2(command = "ls"
        args    = c("-l", "-a"),
        stdout  = "output.txt")
```

Want to learn more?

Some online tutorials:

https://linuxcommand.org/lc3_learning_the_shell.php

https://swcarpentry.github.io/shell-novice/

https://www.learnshell.org

https://www.tutorialspoint.com/unix/shell_scripting.htm

https://www.shellscript.sh

https://explainshell.com