Let's use the terminal (bash)!

Daniel Heimgartner

May 13, 2025

https://linuxcommand.org/lc3_learning_the_shell.php

```
echo 'echo "Hello bash"!' > foo.sh  # write to a file
cat foo.sh                          # inspect the file content
sh foo.sh                           # execute the file
rm foo.sh                           # remove the file
```

1 WHY?

What is bash

- bash stands for *Bourne Again SHell*, i.e., it is a shell. All clear?
- Simply put, a shell is a program that takes commands/user input from the keyboard and gives them to the operating system (OS) to perform.
- In the old days, this was the only available user interface on a Unix-like system (your parents probably know how to use a terminal).
- Nowadays, we have *graphical user interfaces* (GUIs).
- So: There are two ways to interact with a computer: Via the shell and via a GUI.
- bash is actually a scripting language. I.e., you can declare variables, write functions, use control flow, etc. (e.g., https://github.com/dheimgartner/MOBISarchive/blob/main/main.sh)
- However, most of the stuff you actually run in the terminal is not directly bash but an executable program (e.g., `ls`, `cat README.md`, `man man`, `man which`, `which man`, etc.).

What is a terminal?

Most OS usually come with a desktop environment which is pretty much a GUI for interacting with the OS itself. However, in principle, you don't need one – everything can be achieved via the shell!

- A terminal is usually a point of access (e.g., flight terminal).
- The terminal used to be a physical device allowing you to "access" (interact with) the computer.
- Nowadays, it's simply a digital program (*terminal emulator*) – a program that opens a window and lets you interact with the shell.
- The thing that runs in the terminal is a shell (and in most cases bash or some variant of it).

What are the benefits?

- Ever used the Euler cluster? Most servers don't have a desktop environment/(GUI)-based operating system (you have to interact with it via the terminal!)
- Much clearer recipes/instructions how to accomplish stuff (e.g., "Right click > then navigate to "create new folder" > Enter the name of the folder" vs. `mkdir foo`)
- Kind of a pre-requisite to use git!
- Scaffold the project structure and add new files, quickly change files (e.g., using vim), get an overview over the project structure (e.g., `tree .`), etc.
- Most importantly: Quickly use R! Remember from last time? `usethis::create_package()` scaffolds the package (just type `R` to start the R-interpreter and run the function!).
- Understand how the "computer" works (e.g., find the source code of an executable you like and inspect it)...
- Automate stuff (you can think of bash as a way to glue stuff together)

What will I learn?

- Get a mental picture of what the terminal is and what it does when interacting with it
- Learn the most basic (built-in) bash commands (mostly, navigating the file system)
- How to get help; access manual pages
- Goal: 1. Navigate to a place in the file system, 2. Create a new repository, 3. Start the R interpreter (and understand that this simply executes the R executable), 4. Scaffold an R-package (`usethis::create_package()`), 5. Stop the R interpreter, 6. Inspect the newly created directory structure, 7. Navigate out of the project directory and 8. Delete it again.
- Learn where to learn more!

2 LET'S GET STARTED

```
command -options arguments

# e.g.,
ls -a foo
# list (ls) all files (-a) in the directory called foo
```

where `command` is the name of the command, `-options` is one or more adjustments to the command's behavior, and `arguments` is one or more "things" upon which the command operates.

What are commands?

Commands can be one of 4 different kinds:

1. An executable program (e.g., a compiled binary but also a program written in a scripting language such as R, Python, etc.)

2. A command built into the shell itself (e.g., `cd`)
3. A shell function (miniature shell script incorporated into the environment)
4. An alias (command built from other commands)

man

The most important part (about programming) is where to get help. Man page is shorthand for manual page which explains the existence of the `man` command. Man pages vary somewhat in format but generally contain a title, a synopsis of the command's syntax, a description of the command's purpose, and a listing and description of each of the command's options. Man pages, however, do not usually include examples, and are intended as a reference, not a tutorial. Let's try it out and invoke the command on itself

For shell built-ins like `cd` we actually have to use `help`. Many executable programs support a "–help" option, e.g., `R --help`

```
man man   # learn about man
man bash  # learn about bash
```

How to navigate man pages?

The jargon of man pages might be a little difficult to understand and most programs feature many, many options. You might find `tldr` handy (needs to be installed first)... It's fine to just google for help too! Or ask some AI – but then it might be a good idea to consult `man` and check the suggested options and arguments...

"press h for help or q for quit" – It's always a good idea to read first! ;)

- Use vim keybindings (`j` for down and `k` for up, or use arrows).
- Use `/pattern` to search.
  - `n` for the next appearance and `N` to cycle back.

pwd

`pwd` stands for (at least in my head) present working directory.

```
pwd
/home/daniehei
```

ls

Let's list all the files in the current working directory

```
ls -a .
# output omitted
```

mkdir

Let's create/make a new directory

```
mkdir foo
```

cd

And change the directory (i.e., set the working directory to `foo`)

```
cd foo
ls -a .
.  ..
```

To understand the file system (as well as absolute and relative paths) you need to understand the following symbols

- `/` The root directory (at least on Linux). Everything is a subdirectory of the root directory (on Windows you have different drive letters that split the file system into different trees, e.g., `C:`).

- ~ The home directory
- . The current directory
- .. The parent directory

touch

Let's create a new file

```
touch bar.txt
```

tree

Let's inspect the hierarchical structure of the `foo` directory (not that interesting at the moment...)

```
tree .
.
└── bar.txt
```

Let's start the R interpreter and scaffold a new project/package and then inspect again

```
R
usethis::create_package("tuesdays", rstudio = TRUE)
quit()  # quit the R interpreter (alternatively, use Ctrl-D)
tree .
.
├── bar.txt
└── tuesdays
    ├── DESCRIPTION
    ├── NAMESPACE
    ├── R
    └── tuesdays.Rproj

2 directories, 4 files
```

cat

Inspect the content of a file

```
cat(tuesdays/DESCRIPTION)
Package: tuesdays
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person("First", "Last", , "first.last@example.com", role = c "aut", "cre"),
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.3.2
```

cp

For some reason, we want to copy the `DESCRIPTION` file to the current directory

Use `tab` *for autocompletion. E.g., type* `cat tuesdays/DES` *and hit* `tab`.

```
cp ./tuesdays/DESCRIPTION .
```

mv

Let's move the `bar.txt` file into the `tuesdays` directory

```
mv bar.txt tuesdays  # same as ./tuesdays (./ is implicit)
```

rm

Let's remove/delete all of this stuff

```
cd ..
rm -rf foo
```

3 A QUICK DEMO: WRITING SHELL SCRIPTS

- https://linuxcommand.org/lc3_writing_shell_scripts.php
- To successfully write a shell script, we have to do three things:
  1. Write a script
  2. Give the shell permission to execute it
  3. Put it somewhere the shell can find

Resources

TODO

- bash: Highly recommended to work through https://linuxcommand.org/lc3_learning_the_shell.php (Shotts, 2019)
- While you are at it: Learn how to write shell scripts https://linuxcommand.org/lc3_writing_shell_scripts.php (Shotts, 2019)
- Use a package manager to install command line tools or any (GUI) application (e.g., RStudio)!
  - For Mac: brew https://brew.sh/
  - For Windows: chocolatey https://chocolatey.org/

REFERENCES

Shotts W (2019). *The Linux Command Line*. 2nd edition. Turnaround, London.