Effective Programming Practices for Economists

3. Introduction to the Shell

Hans-Martin von Gaudecker

Department of Economics, Universität Bonn

universitätbonn

Introducing the Shell

- What is the shell and what does it do?
- Why you want to use the shell
- When you want to use the shell

What and why

- At a high level, computers do four things
  - run programs
  - store data
  - communicate with each other
  - interact with us

What and why

- Serious Graphical User Interfaces (GUIs) weren't available until the 1990s
- Before then, interaction took place via a Command Line Interface (CLI)
- CLIs are REPLs

CLIs are REPLs

- REPL means *read-evaluate-print-loop*
- Type a command $\rightarrow$ press enter (or return) $\rightarrow$ computer reads it $\rightarrow$ executes it $\rightarrow$ prints its output
- Although it may seem as if the user talked to the computer directly, there is a program inbetween called the *command shell*

Command Shell

- It's a program like any other
- Figures out what to run given what the user inputs
- Effectively a program to run other programs
- The most popular Unix shell is bash (**B**ourne **A**gain **SH**ell)

Command Shell

- The shell allows us to combine existing tools in powerful ways with only a few keystrokes and to set up pipelines to handle large volumes of data automatically
- The command line is often the easiest way to interact with remote machines
- As clusters and cloud computing become more popular for scientific data crunching, being able to drive them is becoming a necessary skill
- Useful for teaching/learning: Much clearer what happens in which order relative to point-and-click.

Files and Directories

- We want to:
  - Explain the similarities and differences between a file and a directory
  - Translate an absolute path into a relative path and vice versa
  - Construct absolute and relative paths that identify specific files and directories
  - Explain the steps in the shell's read-run-print cycle
  - Identify the actual command, flags, and filenames in a command-line call

Files and Directories

- The part of the Operating System that handles files and directories is called the filesystem
- We differentiate between files which hold information and directories (or folders) which hold files
- The structure of the filesystems on Windows and Linux / OS X differ considerably (the latter two as well, but not for what we will need)
- We will focus on the OS X Shell for now but later mention some (of the many) differences
- Several commands are frequently used to interact with these structures

Basic Bash

The dollar sign stands for a prompt waiting for input

```
$
```

Type `whoami` and press Enter to see how the current user is named

```
$ whoami
hmg
```

Basic Bash

- When we type `whoami` the shell finds the program
- The program is run
- The output of the program is shown
- A new prompt is displayed, indicating that it's ready for new commands

Basic Bash

If we want to know where we are we can type `pwd` (print working directory) which yields

```
$ pwd

/Users/me
```

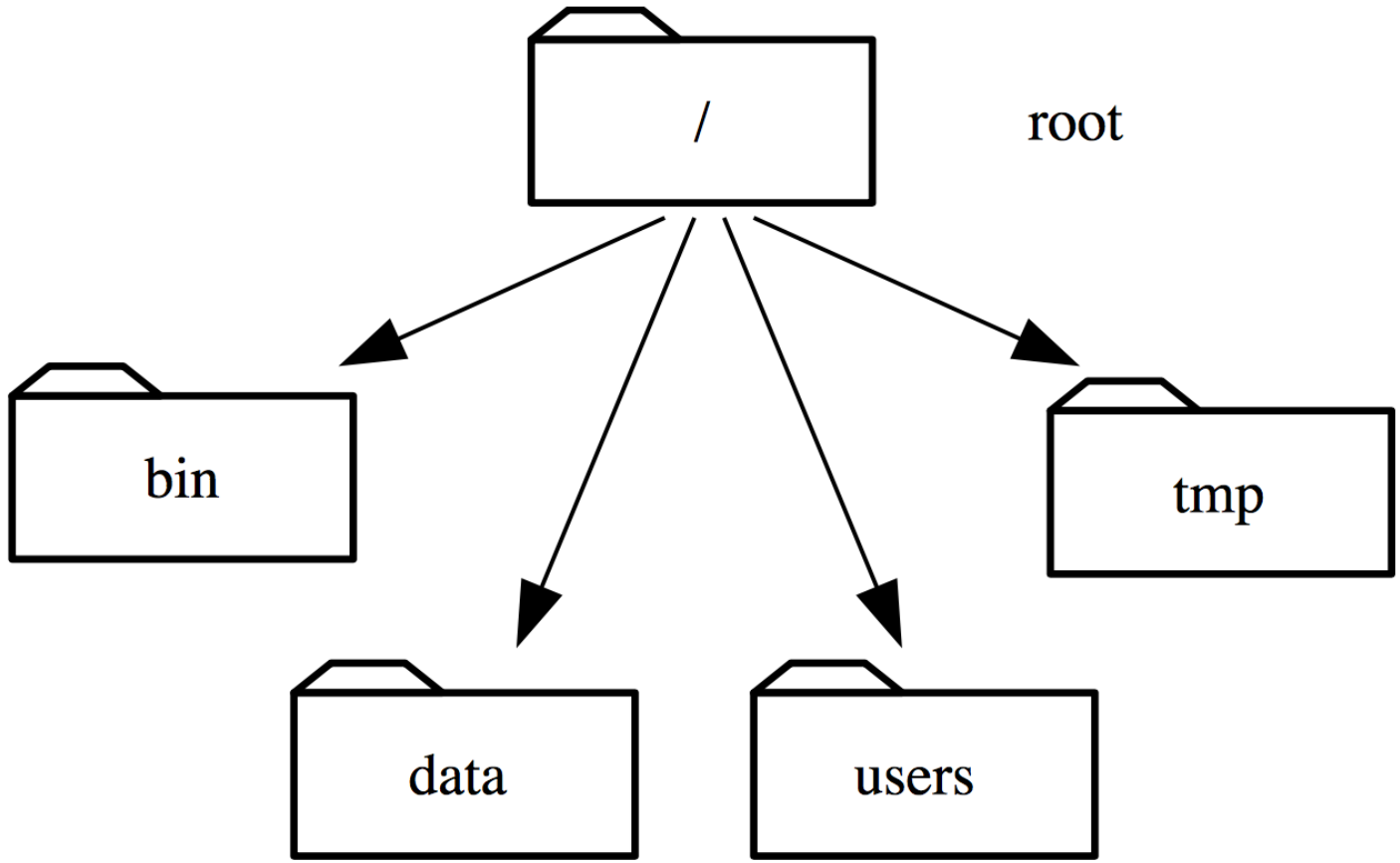
In this case, we are in our home directory.

Directory structure

- To understand what our home directory is, let's look at directory structure
- It is organized as a tree with the root directory called `/` at the very top
- Everything else is contained in it
- `/` refers to the leading slash in `/Users/me`

Directory structure

On OS X systems

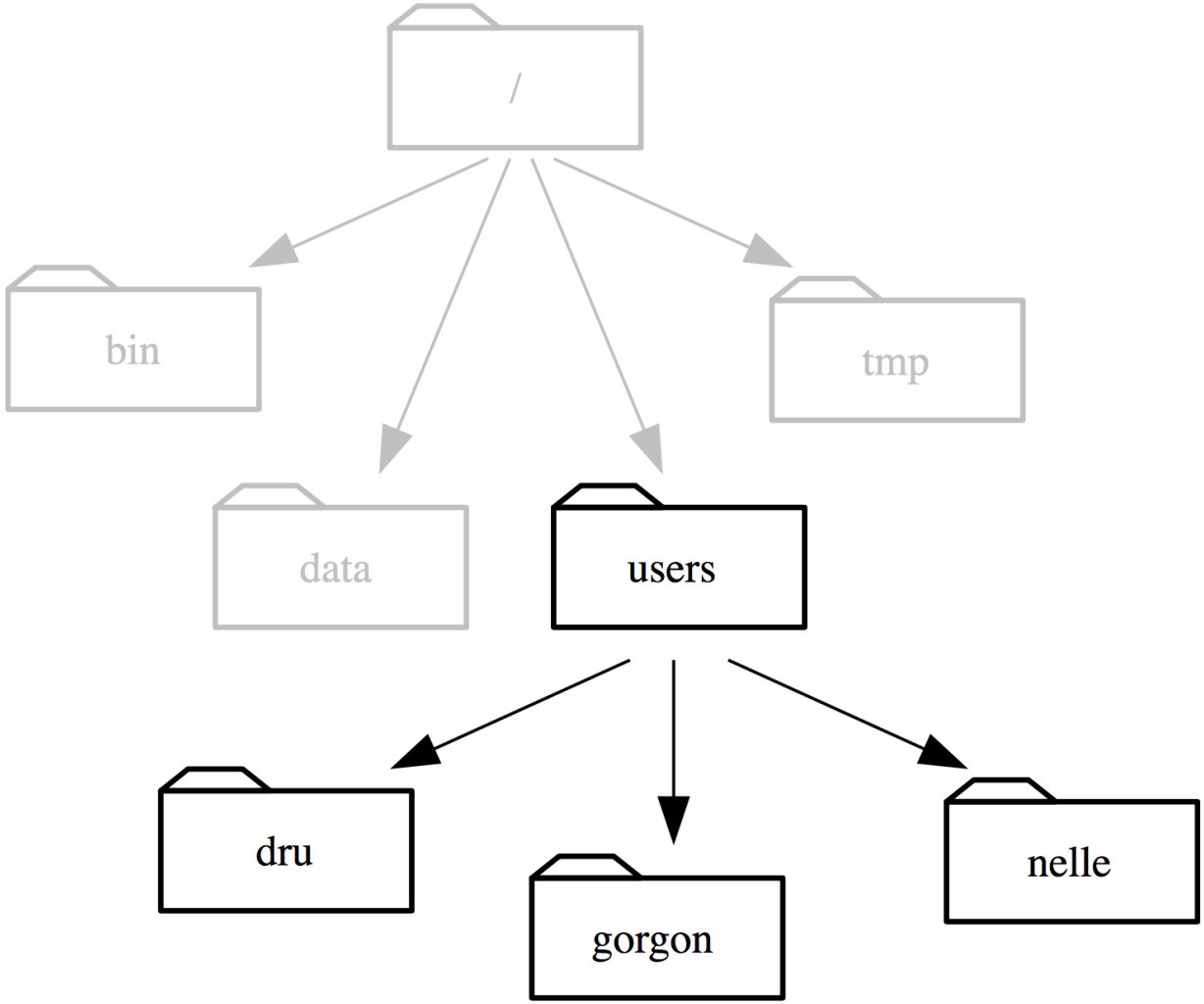


Directory structure

- Underneath `/Users` the data of the other user accounts on the machine is stored
- E.g. `/Users/another-user-on-this-machine`
- We know, that if our current folder is `/Users/me`, we are inside `/Users` because of the first part of its name. Similarly `/Users` resides in the root `/`

Directory structure

inside the `Users` folder



Directory structure

Let's see what happens when we run `ls` inside our home folder (on OS X)

```
$ ls

Applications  Desktop  bin  Documents  Library  Projects
Downloads  Music  anaconda3
```

Important commands in the shell

- There are a couple important commands that are important for manipulating items in the shell and moving around
- We will look at some of the most important ones like `cd`, `ls`, `rm`, `mv`, `cat`, `less`, `grep`, `nano`

cd

Lets first look at `cd` which allows us to change directories

```
$ pwd

/Users/me

$ ls

Desktop  Documents  Downloads ...

$ cd Documents && ls

important_doc_1  important_doc_2  important_doc_3 ...

$ pwd

/Users/me/Documents

$ cd ..

$ pwd

/Users/me
```

cd

- `.` denotes our current working directory (output of `pwd`)
- `..` the directory one above in the tree structure

ls

As we have seen before `ls` shows us what's contained inside directories. We can use parameters to specify how `ls` should work

```
$ ls -F
```

Display a slash ('/') immediately after each pathname that is a directory, an asterisk ('*') after each that is executable

ls

- Often we want to use something like `ls -laF`. This will include files starting with a `.` (so called *dotfiles*, that often hold configuration) as well as showing a so called long list (including hidden files for example) and mark directories accordingly.

- We can also write `ls some-sub-directory/` to receive the contents of the sub directory

rm

We can use `rm` to remove files. We can write

```
$ rm somefile.txt
```

or

```
$ rm some-sub-folder/somefile.txt
```

to remove a file in a subfolder. We could also use `rm -r` (for recursive) to remove directories

```
$ rm -r some-directory/
```

cat & less

We can print out files into the shell using `cat` or view them using `less`. Simply supply the file as an

```
$ cat <somefile>
```

```
$ less <somefile>
```

man

For much about every shell command distributed with Linux / OS X so called man pages exist. They represent documentation or help files. To get regarding some command just enter

```
$ man <somecommand>
```

|, grep, head, tail

We can for example use `cat`, `|` (called pipe) and `grep` to filter output for matching text

```
$ cat <somefile> | grep <somesearchterm>
```

Use `head` or `tail` to display the beginning or end of some file. This can be helpful when viewing server logs or similar. Usage is similar just (optionally) supply `n` for the number of lines and a filename

```
$ head -n <somefile>
```

```
$ tail -100 some-log-file.log
```

Summary of elementary commands

command	name
`cd`	**c**hange **d**irectory
`ls`	**ls**t directory contents
`rm`	re**m**ove directory entries
`less`	opposite of more
`cat`	con**cat**enate and print files
`grep`	file pattern search, (**g**eneral **r**egular **e**xpression **print**)
`pipe`	Postfix delivery to external command
`tail`	display the last part of a file

Looking further

- With `|` and `grep` we can do some things that would require signifant effort without the shell
- `|` can chain two functions (commands) together. The output of the first command serves as input for the second one
- `grep` can look for patterns in files and serve as a great tool to search
- We can use `ls -la | grep '\.log$'` to see all log files in the current folder
- Or `cat some_log_file.log | grep -10 'error'` to see all the errors in a log file (with 10 lines of context)

Differences between operating systems

- Shells of most Unix derivatives (Linux, OS X) are fairly similar and the basic above mentioned tools are available
- The Windows shell differs considerably from this
  - On Windows we can utilize the so called Power Shell which resembles the Linux shell to some extent
  - For all Git related matters there exists a system independent Git bash for interacting with Git and not having to worry about shell issues
  - On Windows systems path names start with \ instead of /

Why work with a shell?

- Close interaction with the operation system and compilers/interpreters
- Automation using scripts (with bash or Python or both)
- Working with (remote) servers via ssh (all you get is a bash)
- Greater control and flexibility

When to use the shell

- “When not surfing the internet or doing email” –unknown
- The learning curve can be steep in the beginning
- Trade-off because simple things may take long in the beginning
- Essential in a programming context, over time basically all administrative tasks become less time intensive than using a GUI

Some simple recipes

- The following slides have some recipes for situations you may find yourself stuck in
- No need for an in-dept understanding of those commands

How to get out of vim

- Some commands (e.g. `git commit` without a message) will trigger a default editor if user input is required. This happens to be `vim` in most cases
- It is helpful to remember two things
  - If you just want to leave: Press '`Esc`', followed by '`:q`' (a colon followed by q for quit)
  - If you want to make changes to the opened file:
    1. Press '`i`' for insert mode and make your changes
    2. Press '`Esc`' followed by '`:wq`' (for write & quit)

Using Sublime as default text editor

Enter the following commands to make Sublime Text your default editor

```
$ export EDITOR='subl -w '

$ export TEXEDIT='subl'

$ alias subl="subl --new-window"
```

Those settings will only be active for the current session. If you want to make them persistent, you can copy those terms into your `.bash_profile` / `.bashrc` in your home directory

.bashrc and Git bash

- When using Git bash on Windows you can also use a `.bashrc` file
- Create a `.bashrc` file under `~/.bashrc`, where ~ denotes your home directory
- ~ is usually your `C:\Users\<your user name>` folder. Typing `echo ~` in the git bash terminal will tell you what that folder is

Acknowledgements

- This course is designed after and borrows a lot from the `Software Carpentry` course designed by `Greg Wilson` for scientists and engineers.

- The Software Carpentry course material is made available under a `Creative Commons Attribution License`, as is this course's material.

License for the course material

[Links to the full `legal` text and the source text for `this` page.] You are free:

- **to Share** to copy, distribute and transmit the work
- **to Remix** to adapt the work

Under the following conditions:

- **Attribution** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

- **Waiver** Any of the above conditions can be waived if you get permission from the copyright holder.
- **Public Domain** Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- **Other Rights** In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to `this` web page.