(../) The Unix Shell (../) (../02-filedir

Introducing the Shell

Overview

Teaching: 5 min
Exercises: 0 min
Questions

- What is a command shell and why would I use one?

Objectives

- Explain how the shell relates to the keyboard, the screen, the operating system, and users' programs.
- Explain when and why command-line interfaces should be used instead of graphical interfaces.

Background

At a high level, computers do four things:

- run programs
- store data
- communicate with each other, and
- interact with us

Computers can interact with us in many different ways, such as through a keyboard and mouse, touch screen interfaces, or using speech recognition systems. While touch and voice interfaces are becoming more commonplace, most interaction is still done using traditional screens, mice, touchpads and keyboards.

The **graphical user interface** (GUI) is the most widely used way to interact with personal computers. We give instructions (to run a program, to copy a file, to create a new folder/directory) with the convenience of a few mouse clicks. This way of interacting with a computer is intuitive and very easy to learn. But this way of giving instructions to a computer scales very poorly if we are to give a large stream of instructions even if they are similar or identical. For example if we have to copy the third line of each of a thousand text files stored in thousand different directories and paste it into a single file line by line. Using the traditional GUI approach of clicks will take several hours to do this.

This is where we take advange of the shell - a **command-line interface** to make such repetitive tasks automatic and fast. It can take a single instruction and repeat it as is or with some modification as many times as we want. The task in the example above can be accomplished in a few minutes at most.

The heart of a command-line interface is a **read-evaluate-print loop** (REPL). It is called so because when you type a command and press Return (also known as Enter) the shell reads your command, evaluates (or "executes") it, prints the output of your command, loops back and waits for you to enter another command.

The Shell

If you are familiar with the shell, go to this advanced course: ReproNim shell (http://www.repronim.org/module-reproducible-basics/01-shell-basics/)

The Shell is a program which runs other programs rather than doing calculations itself. Those programs can be as complicated as climate modeling software and as simple as a program that creates a new directory. The simple programs which are used to perform stand alone tasks are usually refered to as commands. The most popular Unix shell is Bash, (the Bourne Again SHell — so-called because it's derived from a shell written by Stephen Bourne). Bash is the default shell on most modern implementations of Unix and in most packages that provide Unix-like tools for Windows.

When the shell is first opened, you are presented with a **prompt**, indicating that the shell is waiting for input.

```
$
```

The shell typically uses `$` as the prompt, but may use a different symbol. In the examples for this lesson, we'll show the prompt as `$`. Most importantly: when typing commands, either from these lessons or from other sources, *do not type the prompt*, only the commands that follow it.

So let's try our first command, which will list the contents of the current directory:

```
$ ls
```

```
Desktop     Downloads   Movies      Pictures
Documents   Library     Music       Public
```

📌 Command not found

If the shell can't find a program whose name is the command you typed, it will print an error message such as:

```
$ ks
```

```
ks: command not found
```

Usually this means that you have mis-typed the command.

Is it difficult?

It is a different model of interacting than a GUI, and that will take some effort - and some time - to learn. A GUI presents you with choices and you select one. With a **command line interface** (CLI) the choices are combinations of commands and parameters, more like words in a language than buttons on a screen. They are not presented to you so you must learn a few, like learning some vocabulary in a new language. But a small number of commands gets you a long way, and we'll cover those essential few today.

Flexibility and automation

The grammar of a shell allows you to combine existing tools into powerful pipelines and handle large volumes of data automatically. Sequences of commands can be written into a *script*, improving the reproducibility of workflows and allowing you to repeat them easily.

In addition, the command line is often the easiest way to interact with remote machines and supercomputers. Familiarity with the shell is near essential to run a variety of specialized tools and resources including high-performance computing systems. As clusters and cloud computing systems become more popular for scientific data crunching, being able to interact with the shell is becoming a necessary skill. We can build on the command-line skills covered here to tackle a wide range of scientific questions and computational challenges.

Nelle's Pipeline: A Typical Problem

Nelle Nemo, a marine biologist, has just returned from a six-month survey of the North Pacific Gyre (http://en.wikipedia.org/wiki/North_Pacific_Gyre), where she has been sampling gelatinous marine life in the Great Pacific Garbage Patch (http://en.wikipedia.org/wiki/Great_Pacific_Garbage_Patch). She has 1520 samples in all and now needs to:

1. Run each sample through an assay machine that will measure the relative abundance of 300 different proteins. The machine's output for a single sample is a file with one line for each protein.
2. Calculate statistics for each of the proteins separately using a program her supervisor wrote called `goostats`.
3. Write up results. Her supervisor would really like her to do this by the end of the month so that her paper can appear in an upcoming special issue of *Aquatic Goo Letters*.

It takes her about two weeks to run her samples by the assay machine. Now she has the daunting task of analysing her results by running 'goostats'. The bad news is that if she has to run `goostats` by hand using a GUI, she'll have to select a file using an open file dialog 1520 times. At 30 seconds per sample, the whole process will take more than 12 hours (and that's assuming the best-case scenario where she is ready to select the next file as soon as the previous sample analysis has finished). This zero-breaks always-ready scenario is only achievable by a machine so it would likely take much longer than 12 hours, not to mention that the chances of her selecting all of those files correctly are practically zero. Missing that paper deadline is looking increasingly likely.

The next few lessons will explore what she should do instead. More specifically, they explain how she can use a command shell to run the `goostats` program, using loops to automate the repetitive steps e.g. entering file names, so that her computer can work 24 hours a day while she writes her paper.

As a bonus, once she has put a processing pipeline together, she will be able to use it again whenever she collects more data.

Key Points

- A shell is a program whose primary purpose is to read commands and run other programs.
- The shell's main advantages are its high action-to-keystroke ratio, its support for automating repetitive tasks, and its capacity to access networked machines.
- The shell's main disadvantages are its primarily textual nature and how cryptic its commands and operation can be.

^
(../)

>
(../02-
filedir

Edit on GitHub (https://github.com/neurodatascience/shell-novice/edit/gh-pages/_episodes/01-intro.md) / Contributing (https://github.com/neurodatascience/shell-novice/blob/gh-pages/CONTRIBUTING.md) / Source (https://github.com/neurodatascience/shell-novice/) / Cite (https://github.com/neurodatascience/shell-novice/blob/gh-pages/CITATION) / Contact (mailto:team@carpentries.org)

Using The Carpentries style (https://github.com/carpentries/styles/) version 9.5.2 (https://github.com/carpentries/styles/releases/tag/v9.5.2).