BASH/SHELL Workshop for Data Science

Session III

Agenda

- Why should Economists care about BASH?
- Some powerful commands for text manipulation
- Regular expressions
- Writing loops in BASH
- Managing with your data workflow in BASH

Why Shell!?

When to use shell

- When you want to have a high-level structural glimpse of data.
- When you want break complex projects into simpler subtasks, of chaining together components and utilities.
- It is often used in first stage of project development to get understanding of pitfalls before proceeding to other high-level languages such as python or C++.
- Example –

a. Sometimes you may want to run a Stata job as a background Unix process.

b. You may not want to disturb original proprietary source of data and create symbolic link to it for your local tasks.

c. Scripting languages especially suited for parsing text files and command output. May be embedded singly or in combination in pipes and shell scripts.

When not to use shell

- When you need cross−platform portability - Use Python wrappers instead.
- Projects involving heavy−duty math operations, especially floating point arithmetic, arbitrary precision calculations, or complex numbers.
- Project consists of subcomponents with interlocking dependencies. (Bash is limited to serial file access).
- When need multi−dimensional arrays. (Use Pandas library instead).
- Need data structures, such as linked lists or trees or need to run statistical models.
- Need to generate visualizations or manipulate graphics or GUIs – Use tools such R shiny/d3.js/seaborn/Plotly, etc.
- Need direct access to system hardware.
- Need to use libraries or interface with legacy code.
- Proprietary, closed−source applications (shell scripts are essentially Open Source).

Bash vs. Python vs. Other ….. how to pick?
Bash as a catalyst to other tools

- It is even possible to combine a Bash script and Perl script within the same file. Depending on how the script is invoked, either the Bash part or the Perl part will execute.
- One powerful example of BASH one-liner for text parsing & manipulation:

  This uses bash + perl, to find all .yaml files in a directory, and replace all subtree occurrences with lesson instead of session.
  - for i in `find . -name '*.md'`; do perl -p -i -e `s/Session/Lesson/g' $i;done

Now let’s come back to bash syntax and commands

...there are dark corners in the Bourne shell, and people use all of them.
-Chet Ramey

A Brief Introduction to Regular Expressions

- An expression is a string of characters. Those characters that have an interpretation above and beyond their literal meaning are called metacharacters. A quote symbol, for example, may denote speech by a person, ditto, or a meta−meaning for the symbols that follow.
- Regular Expressions are sets of characters and/or metacharacters that UNIX endows with special features.
- The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters (a substring or an entire string).
- The asterisk * matches any number of repeats of the character string or RE preceding it, including zero.

Special characters

- # : comments
- command substitution: [backticks] `command` makes available the output of command for setting a variable. This is also known as backticks or backquotes.
- wild card: [asterisk] The * character serves as a "wild card" for filename expansion
- Command separator: [Semicolon ;] Permits putting two or more commands on the same line. E.g. echo hello; echo there.

exit command

- Every command returns an exit status (sometimes referred to as a return status).
- A successful command returns a 0, while an unsuccessful one returns a non−zero value that usually may be interpreted as an error code.
- Well−behaved UNIX commands, programs, and utilities return a 0 exit code upon successful completion, though there are some exceptions.
- Try
- #!/bin/bash
- echo hello echo $? # Exit status 0 returned because command successful.
- ls ulb # No such file or directory.
- echo $? # Non−zero exit status returned.

Conditioning and looping

- At the heart of any programming language are iteration and conditional execution.
- Iteration is the repetition of a section of code until a condition changes.
- Conditional execution is making a choice between two or more actions (one of which may be to do nothing) based on a condition.
- In the shell, there are three types of loop (while, until, and for) and three types of conditional execution (if, case, and the conditional operators && and ||, which mean AND and OR, respectively).

Let’s create a data pipeline:

Download tutorials

loop_simple_examples.sh

if_condition_examples.sh

prepare_data_pipeline_for_a_specific_project.sh

Appendix

The ls and tree command will show the certain folder structure that has been created by executing the tutorial.

For loop can be simple or complex depending on the directory structure.

Some notes:

- $basenode command: It strips the directory from the filename (essentially strips the branches of the tree i.e. directory and gives you base leaf i.e. filename.)