Using Shells and Scripting (slides)

In the intro portion of the workshop you will learn:

- What are shells?
- Which one(s) to use?
- What is scripting and what are its applications?
- How to write scripts.
- How to use some useful tools in scripts.

What are shells?

The program that is started after logging on a Linux machine and that allows users to type commands (aka the login shell). For historical reasons there are several shells:

1. The Bourne shell (`sh`) or the Bourne again shell (`bash`)
2. The C shell (`csh`) or the tC shell (`tcsh`)
3. The Korn shell (`ksh`)
4. The Z shell (`zsh`)

You can check the Introduction to the Command Line for Genomics Carpentries lesson for an intro to `bash`.

Additional information

“man pages”

Everything about each shell is explained in the “man pages”, there are books written about scripting and there is a ton of info on the web too.

Reference slide deck

We have included a reference slide deck, with extended information on tools covered in this presentation, and additional scripting tools and examples.

Which shell(s) to use?

On Linux machines `sh` and `bash` are the same program, `csh` and `tcsh` are also the same program.

- `bash` is the shell most used by biologists (and used to install packages for historical reasons).
- `csh` is the shell most used by astronomers.

  The `csh` was written after `sh` to use a syntax more like the C programming language (hence the name), while the `tcsh` is an improvement to the C shell for terminal use (hence the `t`).
- a few people and systems use the Korn shell.

You can write scripts in any shell syntax you care to, independently of what your login shell is.

We will focus on the `bash` shell in our lessons here.

What is scripting and what are its applications?

Scripting vs. Programming

- Programming languages are sets of instructions compiled to produce executables with machine level instructions (a more "complicated" process).
- By contrast, scripting refers to sets of instructions that are parsed and executed by a program, hence
  - there is no need to compile the script (convenient)
  - but, *in principle*, they run more slowly than executables.
- Scripts can combine existing modules or components, while programming languages are used to build more sophisticated/complicated applications from scratch.

Scripting vs. Programming (cont'd)

- Some scripting languages are parsed to check for syntax errors before executing them, but *not* shell scripts.
- Scripts are used to help run applications, on Hydra a job file is a (simple) script.
- You can write complex scripts, and scripts can invoke (i.e., start) other scripts.

How to write scripts?

- A script is a text file that holds a list of commands, and thus can be written with any type of editor (`nano`, `vi`, `emacs`).
- The commands in the script must follow the syntax of the shell used to parse the script.
- A script can take arguments (options) and thus be more general, it allows you to define variables, use expressions or execute commands.
  - A variable is a mechanism to hold a value and refer to it by its name, or a way to modify how some commands behave - variables can also be one dimensional arrays (lists)
  - An expression allows a user to perform simple arithmetic or use commands to create values held by variables (for example: set the variable `num` to hold the number of lines in a file).

How to write scripts (cont'd)

- Scripting syntax allows for "flow control" namely it allows for
  - tests and logical operators - `if` statements
  - loops - `for` statements
  - more flow control: `case`, `while`, `until` and `select`
  - `bash` also allows users to define functions (not covered)
  - the precise syntax is shell specific, i.e. `[ba]sh` syntax is different from `[t]csh` syntax.
- Scripts allow for I/O redirection
  - input: aka `stdin`
  - output: aka `stdout`
  - error: aka `stderr`
  - pipes: redirecting output of one command to be the input to another command

Sophisticated shell scripting is akin to programming, we can't & won't teach programming today.

We will focus on basic skills and make time for hands-on learning.

There is more than one way to bake a cake, which makes things confusing at first.

Script Variables

What are variables

- A variable is a character string to which a value is assigned.
- The assigned value can be a number, some text, a filename, a device, or any other type of data.
- The value of a variable is obtained by using $ followed by the variable name.

Variable Examples

bash syntax

```
filename=/here/goes/nothing.txt
string='hello class'
let num=33
echo $filename $string $num
```

Result

```
/here/goes/nothing.txt hello class 33
```

Quotes when setting variables

Single quotes ' vs double quotes "

```
blah='tell me more'
name0=hello
name1='hello $blah'
name2="hello $blah"
echo name1=$name1
echo name2=$name2
```

In case of doubt, use { and }

```
blah='tell me more'
name1='hello $blah'
name2="hello ${blah}"
echo name1=${name1}
echo name2=${name2}
```

Output:

```
name1=hello $blah
name2=hello tell me more
```

scripting hands-on

In the hands-on portion of the workshop you will learn how to:

- Run a set of commands: Scripts
- Simplify and avoid errors
- Test assumptions
- Generalize a script to be used for multiple files or executions: Arguments
- Ease hands-on time by repeating a command for every file: for loops
- Re-use arguments by manipulating the text
- Get parameters from another file

Log in to Hydra

If you need a reminder about how to log into Hydra and how to change your password, check out our Intro to Hydra tutorial:
`https://github.com/SmithsonianWorkshops/Hydra-introduction/`