Advanced Shell Scripting for Bioinformatics

Stephen A. Sefick

2017-02-23

Outline

Topic

Motivation: why write a bash script?

Motivation: why write a bash script?

1. Reusable

Motivation: why write a bash script?

1. Reusable
   - Do something once; do it a thousand times

Motivation: why write a bash script?

1. Reusable
    - Do something once; do it a thousand times
2. Remember what you did

Motivation: why write a bash script?

1. Reusable
    - Do something once; do it a thousand times
2. Remember what you did
    - analysis documentation

Motivation: why write a bash script?

1. Reusable
    - Do something once; do it a thousand times

2. Remember what you did
    - analysis documentation
    - reproducible research

Motivation: why write a bash script?

1. Reusable
   - Do something once; do it a thousand times
2. Remember what you did
   - analysis documentation
   - reproducible research
   - literate programming

Motivation: why write a bash script?

1. Reusable
   - Do something once; do it a thousand times
2. Remember what you did
   - analysis documentation
   - reproducible research
   - literate programming
3. Deploy-able on desktop or supercomputer

Motivation: why write a bash script?

1. Reusable
   - Do something once; do it a thousand times

2. Remember what you did
   - analysis documentation
   - reproducible research
   - literate programming

3. Deploy-able on desktop or supercomputer

4. Computer Programming is JUST DARN FUN!!!

Basic Bash Scripting

Basic Bash Scripting

1. What is a script?

Basic Bash Scripting

1. What is a script?
    - a "recipe" for the computer

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer
2. A collection of instructions in an interpreted language

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer

2. A collection of instructions in an interpreted language
   - Bash, R, Python, Perl, etc.

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer

2. A collection of instructions in an interpreted language
   - Bash, R, Python, Perl, etc.
   - Instructions executed at run-time

Basic Bash Scripting

1. What is a script?
    - a "recipe" for the computer
2. A collection of instructions in an interpreted language
    - Bash, R, Python, Perl, etc.
    - Instructions executed at run-time
    - with language specified in the shebang line

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer

2. A collection of instructions in an interpreted language
   - Bash, R, Python, Perl, etc.
   - Instructions executed at run-time
   - with language specified in the shebang line

3. Human-readable (next step literate programming)

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer

2. A collection of instructions in an interpreted language
   - Bash, R, Python, Perl, etc.
   - Instructions executed at run-time
   - with language specified in the shebang line

3. Human-readable (next step literate programming)
   - comment, comment, comment!!!

Basic Bash Scripting

1. What is a script?
   - a "recipe" for the computer

2. A collection of instructions in an interpreted language
   - Bash, R, Python, Perl, etc.
   - Instructions executed at run-time
   - with language specified in the shebang line

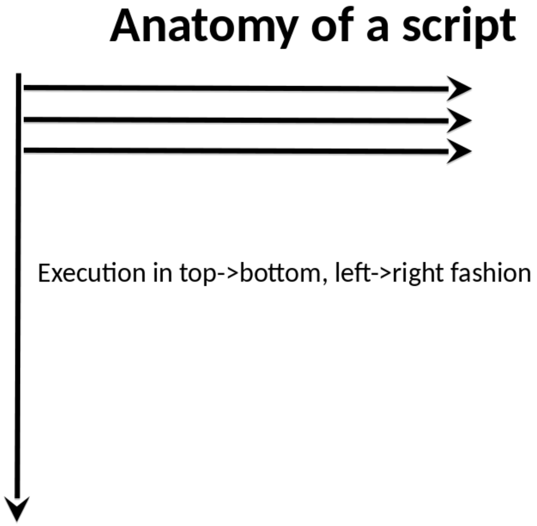
3. Human-readable (next step literate programming)
   - comment, comment, comment!!!
   - worst enemy in 6 months?

Basic Bash Scripting

1. What is a script?
    - a "recipe" for the computer
2. A collection of instructions in an interpreted language
    - Bash, R, Python, Perl, etc.
    - Instructions executed at run-time
    - with language specified in the shebang line
3. Human-readable (next step literate programming)
    - comment, comment, comment!!!
    - worst enemy in 6 months?
    - YOU ARE!!!!

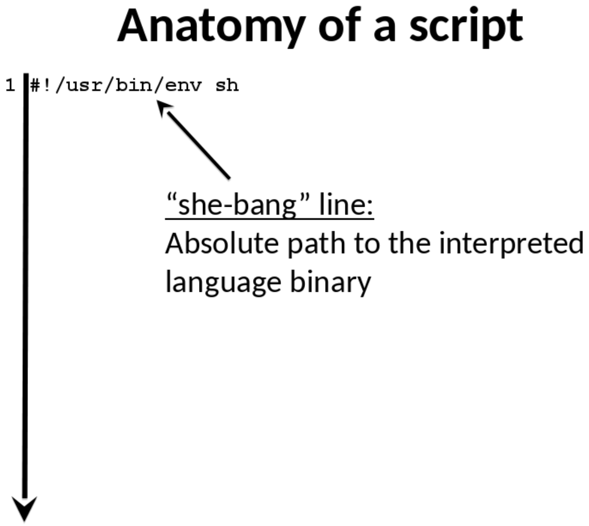
Anatomy of a script

Anatomy of a script


Anatomy of a script

Execution in top->bottom, left->right fashion

Anatomy of a script

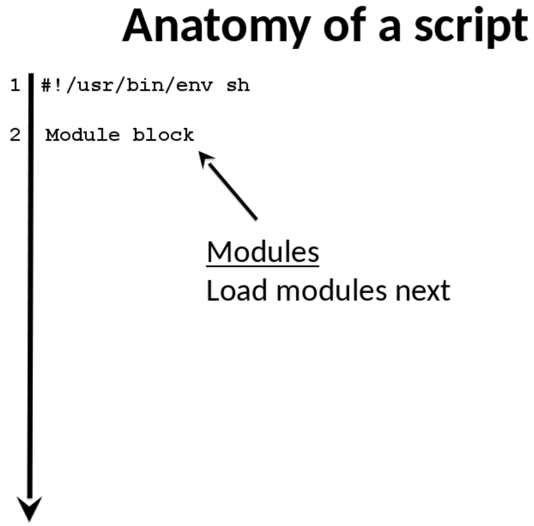


Anatomy of a script

```
#!/usr/bin/env sh
```

"she-bang" line:
Absolute path to the interpreted language binary

Anatomy of a script



Anatomy of a script

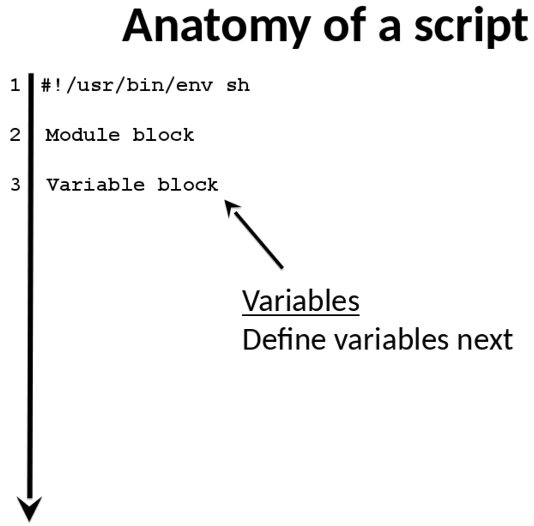
```
1 #!/usr/bin/env sh

2 Module block
```

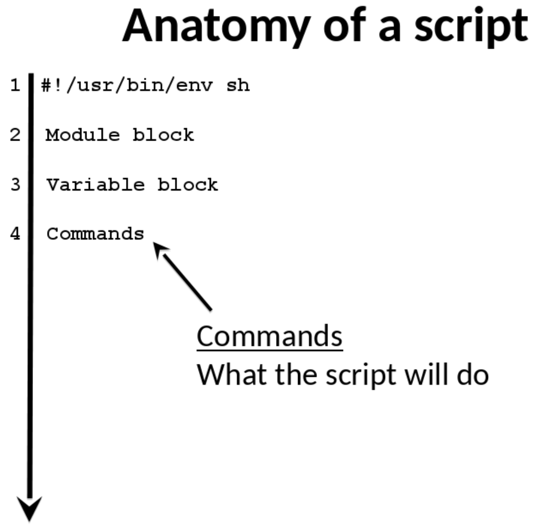
Modules

Load modules next

Anatomy of a script



Anatomy of a script



Anatomy of a script

```
1 #!/usr/bin/env sh
2  Module block
3  Variable block
4  Commands
```

<ins>Commands</ins>
What the script will do

What are variables?

1. Environmental variables

What are variables?

1. Environmental variables
    - used by 1 or more applications

What are variables?

1. Environmental variables
   - used by 1 or more applications
   - $HOME, $PATH, $SHELL, etc.

What are variables?

1. Environmental variables
    - used by 1 or more applications
    - $HOME, $PATH, $SHELL, etc.
2. User defined variables

What are variables?

1. Environmental variables
   - used by 1 or more applications
   - $HOME, $PATH, $SHELL, etc.
2. User defined variables
   - can be anything (assume env variable names are reserved)

What are variables?

1. Environmental variables
   - used by 1 or more applications
   - $HOME, $PATH, $SHELL, etc.
2. User defined variables
   - can be anything (assume env variable names are reserved)
   - best practice

What are variables?

1. Environmental variables
    - used by 1 or more applications
    - $HOME, $PATH, $SHELL, etc.
2. User defined variables
    - can be anything (assume env variable names are reserved)
    - best practice
        - indicative of what it stores

What are variables?

1. Environmental variables
   - used by 1 or more applications
   - $HOME, $PATH, $SHELL, etc.
2. User defined variables
   - can be anything (assume env variable names are reserved)
   - best practice
     - indicative of what it stores
     - contains no special characters (i.e., $)

What are variables?

1. Environmental variables
   - used by 1 or more applications
   - $HOME, $PATH, $SHELL, etc.
2. User defined variables
   - can be anything (assume env variable names are reserved)
   - best practice
     - indicative of what it stores
     - contains no special characters (i.e., $)
     - separated by underscores raw_counts

Why are variables important?

Why are variables important?

1. tidy code

Why are variables important?

1. tidy code
2. easily readable

Why are variables important?

1. tidy code
2. easily readable
3. less mistakes

Why are variables important?

1. tidy code
2. easily readable
3. less mistakes
    - type once use multiple times

Variable creation

1. directly

```
wd=${HOME}/analysis_directory
echo ${wd}
```

```
/home/ssefick/analysis_directory
```

Variable creation

1. directly

```
wd=${HOME}/analysis_directory
echo ${wd}
```

```
/home/ssefick/analysis_directory
```

1. dynamically

```
##direct
input_dir=input
##dynamic
files=(‘ls ${input_dir} | grep sh$‘)
echo ${files[@]}
```

```
awesome_script1.sh awesome_script2.sh awesome_script3.sh
```

Topic

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl
2. What we know about the problem

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl
2. What we know about the problem
   - more than 1 mislabeled file

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl
2. What we know about the problem
   - more than 1 mislabeled file
   - Contained in a folder called input

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl
2. What we know about the problem
   - more than 1 mislabeled file
   - Contained in a folder called input
3. What is a sensible way to go about this?

An example: work smarter not harder

1. I accidentally named files with .sh and not .pl
2. What we know about the problem
   - more than 1 mislabeled file
   - Contained in a folder called input
3. What is a sensible way to go about this?
4. Let's write a little script to fix it

Let's build a script interactively

Parts of a script

1. Shebang -language specific

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block
   - all HPC modules (e.g., module load)

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block
   - all HPC modules (e.g., module load)
3. Variable block

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block
    - all HPC modules (e.g., module load)
3. Variable block
    - all variables

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block
   - all HPC modules (e.g., module load)
3. Variable block
   - all variables
4. Commands

Let's build a script interactively

Parts of a script

1. Shebang -language specific
2. Module block
   - all HPC modules (e.g., module load)
3. Variable block
   - all variables
4. Commands
   - instruction set

input

1. I think the files are in "input"

Let's check

```
########################
wd=`ls input | grep sh$`
echo ${wd}
```

```
awesome_script1.sh awesome_script2.sh awesome_script3.sh
```

Shebang and module block

shebang

1. top of file

```
#########################
#!/usr/bin/env sh
#########################
```

Shebang and module block

shebang

1. top of file

```
#########################
#!/usr/bin/env sh
#########################
```

module block

1. Load all modules here
2. Easily find/remember what modules loaded
3. No modules to load because I built this presentation on a PC

```
######################
##Modules
##module load something
######################
```

Variable Block

1. Easily find input/output directories
2. Tidy programming

variable block

```
#######################
##Variables
input_dir=input
out_dir=output
##input/output files arrays
files=(‘ls ${input_dir} | grep sh$‘)
out_files=(‘echo ${files[@]} | sed s/sh/pl/g‘)

##parameter expansion
##parameter/patten/string
input_with_path=( "${files[@]/#/${input_dir}/}" )
output_with_path=( "${out_files[@]/#/${out_dir}/}" )
#######################
```

Commands

For loop

```
########################
##commands
##make the output dir
mkdir -p ${out_dir}

##for loop
##use length to iterate in order to index input output arrays
for ((i=0; i<${#input_with_path[@]}; i++)); do

    echo ${i}

    echo cp ${input_with_path[${i}]} ${output_with_path[${i}]}

done
######################
```

Understanding the script

```
files=(`ls ${input_dir} | grep sh$`)
```

awesome_script1.sh
awesome_script2.sh
awesome_script3.sh

```
input_with_path=( "${files[@]/#/${
    input_dir}/}" )
```

input/awesome_script1.sh
input/awesome_script2.sh
input/awesome_script3.sh

```
out_files=(`echo ${files[@]} | sed s/
    sh/pl/g`)
```

awesome_script1.pl
awesome_script2.pl
awesome_script3.pl

```
output_with_path=( "${out_files[@]/#/$
    {out_dir}/}" )
```

output/awesome_script1.pl
output/awesome_script2.pl
output/awesome_script3.pl

Understanding the for loop

```
for ((i=0; i<${#input_with_path[@]}; i++)); do
    ${i}
    cp ${input_with_path[${i}]} ${output_with_path[${i}]}
done
echo ${i}
```

```
0
1
2
```

```
cp ${input_with_path[${i}]} ${output_with_path[${i}]}
```

```
cp    input/awesome_script1.sh    output/awesome_script1.pl
cp    input/awesome_script2.sh    output/awesome_script2.pl
cp    input/awesome_script3.sh    output/awesome_script3.pl
```

Topic

History to script

1. blast history to script from last class

History to script

1. blast history to script from last class
2. Use some information from today

History to script

1. blast history to script from last class
2. Use some information from today
3. Why in the world would we want to redo an analysis?

History to script

1. blast history to script from last class
2. Use some information from today
3. Why in the world would we want to redo an analysis?
    - reproducible research artifact

History to script

1. blast history to script from last class
2. Use some information from today
3. Why in the world would we want to redo an analysis?
    - reproducible research artifact
4. ~20 min

History to script

1. blast history to script from last class
2. Use some information from today
3. Why in the world would we want to redo an analysis?
    - reproducible research artifact
4. ~20 min
    - HW5

Topic

Gold standard: literate programming

"Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."
-Knuth

Some Tools

LaTeX

10 suggestions from the paper

1. For every result, keep track of how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all custom scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected
9. Connect textual statements to underlying results
10. Provide public access to scripts, runs, and results