GWDG AG-C

Do’s and Don’ts for Bash Scripting

Dr. Freja Nordsiek

Table of contents

Introduction

- Bash is the most pervasive shell on Linux
- Bash is often the default shell
- Previous GöHPC Coffee on on 22.05.2024 (`https://pad.gwdg.de/s/pCUIWnKrR`)
- We will cover some similar and some different material today

What is a shell?

- Language interpreter
- Used interactively on command line (REPL) or run a script
- Optimized to
  - Run other programs
  - Pass programs arguments
  - Connect program inputs and outputs together
  - Work with environmental variables
- Often has minimal programming capabilities

Unix Shells – sh (/bin/sh)

- Thompson shell (the original Unix shell)
- Bourne shell (added a lot)
- POSIX shell (standardized additions)
    - Incorporated many features from ksh (Korn Shell)
    - Many mostly-compliant shells
    - System symlinks one to `/bin/sh`

Unix Shells – non-POSIX shells (for reference)

- Korn Shell (ksh)
  - Contribued a lot to POSIX shell and others
  - Feature poor compared to modern shells
  - **Limited availability on GWDG clusters (being phased out)**

Unix Shells – non-POSIX shells (for reference)

- Korn Shell (ksh)
  - Contributed a lot to POSIX shell and others
  - Feature poor compared to modern shells
  - **Limited availability on GWDG clusters (being phased out)**
- C Shells (csh, tcsh, etc.)
  - More C-like syntax
  - Came up with a lot of ideas used by other shells now
  - **Limited availability on GWDG clusters (being phased out)**
  - **Don’t use**

Unix Shells – non-POSIX shells (for reference)

- Korn Shell (ksh)
  - Contribued a lot to POSIX shell and others
  - Feature poor compared to modern shells
  - **Limited availability on GWDG clusters (being phased out)**
- C Shells (csh, tcsh, etc.)
  - More C-like syntax
  - Came up with a lot of ideas used by other shells now
  - **Limited availability on GWDG clusters (being phased out)**
  - **Don't use**
- And many many many others
  - Possibility that some might become modules in upcoming software stack (like fish)

Unix Shells – POSIX Shells

- Almquist Shells (ash)
  - Many
  - Minimal (POSIX and not much else)
  - Debian Almquist Shell (dash)
  - **Not on SCC and NHR clusters**

Unix Shells – POSIX Shells

- Almquist Shells (ash)
  - Many
  - Minimal (POSIX and not much else)
  - Debian Almquist Shell (dash)
  - **Not on SCC and NHR clusters**
- Z Shell (zsh)
  - Mostly compatible with Bash
  - Focuses on interactive use
  - **Should be everywhere on SCC and NHR clusters**

Unix Shells – POSIX Shells

- Almquist Shells (ash)
  - Many
  - Minimal (POSIX and not much else)
  - Debian Almquist Shell (dash)
  - **Not on SCC and NHR clusters**
- Z Shell (zsh)
  - Mostly compatible with Bash
  - Focuses on interactive use
  - **Should be everywhere on SCC and NHR clusters**
- **Bourne-Again Shell (bash)**
  - Today’s topic
  - **Everywhere on SCC and NHR clusters**

How Every Program Is Run

```
PROGRAM ARG1 ... ARGN [REDIRECTIONS]
```

- `PROGRAM` is either
  - Path to program executable file
  - Executable file found in one of the directories in PATH variable
- Zero or more arguments separated by spaces
- `REDIRECTIONS` determine program's stdin, stdout, stderr
- Program's exit code stored in variable `?` (literally)

File Handles

- File handles are numbers that refer to open
  - files
  - devices
  - pipes
  - pseudo-files
  - etc.
- Every program has the following file handles
  - **0** – standard input (stdin)
  - **1** – standard output (stdout)
  - **2** – standard error (stderr)
  - And any others that the calling program decides to pass
- Many features to control stdin, stdout, stderr

Redirections from/to files/devices – basics

```
[HANDLE]DIRECTION[WHERE]
[HANDLE]DIRECTION WHERE
```

`DIRECTION` is the IO direction

IO Operation	`DIRECTION`	Default `HANDLE`
Input	<	0 (stdin)
Output (clobber)	>	1 (stdout)
Output (append)	>>	1 (stdout)

`WHERE` is what to read-from/write-to

- Path to file/device
- `&HANDLE` for file handle `HANDLE`

Redirections from/to files/devices – examples

Stdin from file

```
wc -l < FOO
```

Stdout to file (clobber and append)

```
ls /bin > FILE
ls /usr 1>> FILE
```

Discard stderr by sending to `/dev/null`

```
ls /proc 2>/dev/null
```

Stderr to stdout (will write to file with name 1 if you forget the &)

```
ls /bar 2>&1
ls /baz 2> &1
```

Pipes – One Program's Output to Another's Input

```
PROG_A | PROG_B
```

- Runs PROG_A and PROG_B at the same time
- Stdout of PROG_A connected to stdin of PROG_B

Can string together as many programs as one wants

```
ls -1 2>&1 | grep foo | wc -l
```

Exit code is that of the last program in the pipe sequence, unless one runs "`set -o pipefail`" in which case it is the the first non-zero exit code in the sequence.

Capture Stdout in String

What if you need a program's output in

- An Bash variable
- An environmental variable
- As an argument to another command

Bash (and all POSIX shells) provide syntax

```
$(COMMAND)
```

```
NUMBER_LINES=$(wc -l thesis.md)
stat $(dirname $(echo "$PATH" | cut -d ':' -f 1))
```

Almost Everything is a String to Bash

The Situation

- Pretty much everything is a string to Bash
- Insufficient care with strings is common cause of bugs

Almost Everything is a String to Bash

The Situation

- Pretty much everything is a string to Bash
- Insufficient care with strings is common cause of bugs
- **BE CAREFUL**

Almost Everything is a String to Bash

The Situation

- Pretty much everything is a string to Bash
- Insufficient care with strings is common cause of bugs
- **BE CAREFUL**
- Unless you quote, Bash considers everything separated by a un-escaped character in `IFS` to be a separate string.

Almost Everything is a String to Bash

The Situation

- Pretty much everything is a string to Bash
- Insufficient care with strings is common cause of bugs
- **BE CAREFUL**
- Unless you quote, Bash considers everything separated by a un-escaped character in `IFS` to be a separate string.

IFS Variable

- Default value is space, tab, and newline.
- The C-string `"a b\tc\nd"` is 4 separate strings to Bash
- If `IFS=+`, the C-string `"foo\nbar"` is 1 string to Bash
- If `IFS=+`, the C-string `"foo+bar"` is 2 separate strings to Bash

Quoting – basics

- Quoting a string forces Bash to ignore the value of `IFS`
- Otherwise, you must escape every character that is also in `IFS`

Quoting – basics

- Quoting a string forces Bash to ignore the value of `IFS`
- Otherwise, you must escape every character that is also in `IFS`
- Single quotes for literal string with no interpretation
  - Impossible to insert single quotes
  - Newline must be an actual newline

Quoting – basics

- Quoting a string forces Bash to ignore the value of `IFS`
- Otherwise, you must escape every character that is also in `IFS`
- Single quotes for literal string with no interpretation
    - Impossible to insert single quotes
    - Newline must be an actual newline
- Double quotes for interpreted string
    - Variable, command, arithmetic, etc. expansion
    - Must escape certain characters

Quoting – basics

- Quoting a string forces Bash to ignore the value of IFS
- Otherwise, you must escape every character that is also in IFS
- Single quotes for literal string with no interpretation
  - Impossible to insert single quotes
  - Newline must be an actual newline
- Double quotes for interpreted string
  - Variable, command, arithmetic, etc. expansion
  - Must escape certain characters
- When in doubt, quote (**especially for untrusted inputs**)
  - Malicious input could delete all your data or take over your account
  - Bugs can cause a lot of damage
  - Especially dangerous with sudo and su
    - **AVOID AT ALL COSTS!!!**

Quoting – single quotes

```
$> echo 'ab"
> c$HOMEd
> e$(ls /sys)f'
ab"
c$HOMEd
e$(ls /sys)f
```

Quoting – double quotes

Special Characters

The following characters have special meaning and must be escaped to ignore it

- " – ends the string
- $ – starts an expansion
- \ – escape the next character (itself or one of the above)

```
$> echo "a\"b $(cat ~/.bashrc | wc -l)  \$HOME=$HOME"
a"b 57  $HOME=/home/fnordsi1
```

Expansions - variable

Expression	Result
`"$PATH"`	Value of PATH
`"${PAT}H"`	Value of PAT followed by `'H'`
`"${FOO:-something}"`	Value of FOO if it exists, otherwise `'something'`

For more possibilities, see `https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html`

Expansions - command and arithmetic

Command expansion covered previously

`$(COMMAND)`

Integer arithmetic

Expression	Result
`"$((1 + 2))"`	3
`"$((2 * (4 + 2 / 1)))"`	10
`"$((2 ** 8))"`	256

For more operators, see `https://www.gnu.org/software/bash/manual/html_node/Shell-Arithmetic.html`

IFS Examples – normal

```
$> FOO="foo bar baz"
$> for A in $FOO ; do
> echo "$A"
> done
foo
bar
baz
```

IFS Examples – change to newline

```
$> FOO="foo bar baz"
$> IFS='
> '
$> for A in $FOO ; do
> echo "$A"
> done
foo bar baz
```

Don't forget to return IFS to its original value.

```
OLDIFS="$IFS"
IFS='
'
...
IFS="$OLDIFS"
```

HERE Documents – passing strings as stdin

```
COMMAND <<ENDMARKER
CONTENTS
ENDMARKER
```

- Choose whatever `ENDMARKER` you want (common choice is `EOF`)
- Behaves like double quotes

```
wc -l <<EOF
foo
bar
baz
EOF
```

Will produce the output 3 because there are 3 lines

HERE Documents – create a file

```
cat > test.txt <<EOF
my \$HOME directory is
$HOME
EOF
```

Creates a file `test.txt` with the contents

```
my $HOME directory is
/home/fnordsi1
```

Useful Links

- Full Bash manual – `https://www.gnu.org/software/bash/manual`
- Previous GöHPC Coffee – `https://pad.gwdg.de/s/pCUIWnKrR`