bash scripting – arguments and variables

Adapted from materials by Dr. Carrier

Wikimedia

Jealousy

Think about other commands:

```
echo “Hello world!”
```

```
cat file.txt
```

Wouldn’t it be nice if we could take in arguments too?

Arguments (positional parameters)

We don’t actually need to change anything to accept arguments!

Arguments (positional parameters)

We don’t actually need to change anything to accept arguments!

We just need to use them:

Arguments (positional parameters)

We don’t actually need to change anything to accept arguments!

We just need to use them:

- `$0` - command used to call our program

Arguments (positional parameters)

We don’t actually need to change anything to accept arguments!

We just need to use them:

- `$0` - command used to call our program
- `$1`, `$2`, etc. give first, second, etc arguments

Arguments (positional parameters)

We don’t actually need to change anything to accept arguments!

We just need to use them:

- `$0` - command used to call our program
- `$1`, `$2`, etc. give first, second, etc arguments
- Numbers > 9 need braces: `${10}`

Arguments - Special cases

How many arguments do we have?

```
$#
```

Arguments - Special cases

How many arguments do we have?

`$#`

Access all arguments

As an array: `$@`

As a string: `$*`

Variables

We can also create our own variables

Variables

We can also create our own variables

Assignment:

```
age=72
```

```
filname=foo.txt
```

```
message="This is a test"
```

Variables

We can also create our own variables

Assignment:

```
age=72
```

```
filname=foo.txt
```

```
message=”This is a test”
```

Note that spacing is important!

Variables

We can also create our own variables

Assignment:

```
age=72
```

```
filname=foo.txt
```

```
message=”This is a test”
```

Note that spacing is important!

Accessing: Again use `$name`:

```
echo “I am $age years old!”
```

Variables

You can use `$(cmd)` to run a command and grab its input

Variables

You can use `$(cmd)` to run a command and grab its input

Combine with variables:

```
num_lines=$(grep “dog” file.txt | wc -l)
```

A note from ol’ Dr. Ferg

You can access variables like this:

```
$filename
```

Or like this:

```
${filename}
```

Why would we prefer one or the other?

A note from ol’ Dr. Ferg

You can access variables like this:

```
$filename
```

Or like this:

```
${filename}
```

Why would we prefer one or the other?

What if we want to print “(filename)_backup”?

A note from ol’ Dr. Ferg

You can access variables like this:

```
$filename
```

Or like this:

```
${filename}
```

Why would we prefer one or the other?

What if we want to print “(filename)_backup”?

Generally, I *always* use `${var}` for my variables to avoid any confusion.

Quotes

Double quotes: expand variables inside

Single quotes: Take everything as a literal

Math

We do have access to “arithmetic expansion” (math)

Math

We do have access to “arithmetic expansion” (math)

Must be inside $(( ))

Math

We do have access to “arithmetic expansion” (math)

Must be inside $(( ))

```
$((2 + 2))
```

```
$((2025 - age))
```

Math

We do have access to “arithmetic expansion” (math)

Must be inside $(( ))

```
$((2 + 2))
```

```
$((2025 - age))
```

Common operators:

+, -, *, /, %, ++, --, **