GNU/LINUX TUTORIAL

PART III: BASH SCRIPTING

§9: WILDCARDS AND PATTERNS

PATTERNS 🤖

- `*`: zero or more characters, e.g. `foo.txt` matches `*.txt`.
- `?`: one character, e.g. `123.txt` matches `???.txt`.
- Escaped sequences `\*` and `\?`.
- `[abc]`, `[!abc]`, `[abc]*`, `[!abc]*`: character classes.
- `{foo,bar}` : foo or bar.
- See the docs for more.

EXAMPLES 🪄

```
*.bak          # anything ending in .bak
*.{txt,pdf}    # files ending in .txt or .pdf
???.bak        # matches foo.bak and not foobar.bak
```

PITFALLS 💣

- Bash expands wildcards and passes to the command various arguments.

```
echo {a,b,c}{d,e,f}
ad ae af bd be bf cd ce cf
```

- If there are files named `-r` and `-f`, then `rm *` can mistake files for arguments. Better option is `rm ./*`.

§10: VARIABLES

- Global scope by default.
- `NAMING_CONVENTION` for global vars.

```
$ FOO="World!"
$ echo Hello, $FOO
Hello, World!
```

HOW GLOBAL? 🤔

Children won't see bindings by default!

```
$ TEST="Can you hear me?"
$ echo $TEST
Can you hear me?
$ bash
> echo $TEST  # TEST is not defined in this shell


> exit
```

EXPORTS 💾

```
$ export TEST="HEY?"
$ bash
> echo $TEST
HEY?
> exit
```

- Run `export` to see exported vars.

USE OF QUOTATION MARKS ⚠

```
FOO=World
BAR=Hello, $FOO!    # WRONG: sees World as a command
BAR="Hello, $FOO!"  # substitutes World! for $FOO
BAR='Hello, $FOO!'  # this is literally Hello, $FOO!
```

SOME PREDEFINED VARIABLES 💡

- `$$` : current PID
- `$?` : exit status of last command
- `$HOSTNAME`, `$USER`, `$HOME`

$PATH VARIABLE 🔎

- List of directories with executables for commands.
- E.g. `/usr/local/bin:/usr/bin:/bin`
- To edit your `$PATH`, add to the end of `~/.profile` file `PATH="/foo/bar/baz:$PATH"`

ESCAPING $ AND OTHER THINGS 🧨

```
curl host/api/Resource/\$operation
curl 'host/api/Resource/$operation'
curl host/api/Resource?foo=true\&bar=true
curl 'host/api/Resource?foo=true&bar=true'
```

SUBSTITUTION OF COMMAND ⚙️

```
echo "Today is $(date +%F)"

FOO="Today is $(date +%F)"
echo $FOO

echo $USER is running $(lsb_release -ds)

FILE=/etc/apt/sources.list
echo $(basename $FILE) is located in $(dirname $FILE)
```

ARITHMETIC 🧮

```
$ echo $((2*3*4*5*6*7*8*9*10))
3628800
$ echo $((2**10))
1024
```

WORKING WITH TEXT 🗒️

```
text="foobar"
${text/bar/baz}  # foobaz
${text^^}        # FOOBAR
${text:1}        # oobar
${text:3}        # bar
${text:1:4}      # ooba
${text:(-2)}     # ar
```

EXAMPLE: BATCH PROCESSING 🗂️

```
for file in *.jpg; do
  convert "$file" "${file/.jpg/.png}"
done
```

(Will see loops later.)

§11: REDIRECTION

BASIC REDIRECTION OF STDIN / STDOUT

```
command > file          # redirect STDOUT to file
command >> file         # append to the end
command < file          # send file to the command
command1 | command2     # pipeline
```

USELESS CAT 😹

- Don't do `cat foo.txt | command` : it's the same as `command < foo.txt`.
- *nix convention: normally a command accepts either a file as argument, or reads from STDIN. This is to allow piping.
- https://porkmail.org/era/unix/award

EXAMPLES 💡

```
echo "Hello" > test.txt
echo "Hello again" >> test.txt
sort -u /etc/passwd | head
```

§12: SCRIPTING BASICS

INSTALL BASIC EDITOR 🗒️

- `sudo apt install nano`
- `vim` or `emacs`.

SCRIPTS 📋

- Script is a file read line by line by the interpreter.
- Should have execution rights (`x`).
- Interpreter is specified in the first line (shebang): `#!<path-to-the-interpreter>`

SHEBANGS 💣

```
#!/bin/bash
#!/usr/bin/python3
#!/usr/bin/perl
#!/usr/bin/node
```

GENERAL CONSIDERATIONS ✅

- Bash scripts must be simple and short.
- Otherwise, consider Perl, Python, Node.js, etc.
- Perl is good for more complex programs. Syntax- and feature-wise, it's `bash` + `sed` + … on acid.

CREATING FIRST SCRIPT 👶

- `nano hello`

```
#!/bin/bash


echo "Hello, World!"
```

- `chmod +x hello`
- `./hello`

LINE ENDINGS ⛔

- Scripts must have correct line endings `<LF>`.
- With `<CR><LF>` endings, `<CR>` is treated as a part of shebang.

COMBINING COMMANDS 👨‍🔧

- `foo && bar` : executes `bar` if `foo` returns `0` (success),
- `foo || bar` : executes `bar` if `foo` does not return `0` (failure).
- `true` : command that returns `0` (success).
- `false` : command that returns `1` (failure).

ARGUMENTS 🤬

- `$0`, `$1`, `$2`, `$3`, ...
- `$@` : all arguments
- `$#`

EXAMPLES

For further syntax and examples:

https://github.com/abeshenov/linux-tutorial/tree/main/scripting