Shell Scripting

For PHP Developers

ABOUT ME

ANDREW WOODS

- Software Engineer at Paramount
- Former Organizer Seattle PHP
- Live in NYC

EVERYONE SHOULD USE CLI

- Navigate around
- run commands
- change permissions
- create pipelines
- file redirection

SCRIPTING IS FOR YOU

- You’re here
- The way of wizards
- It’s a powerful skill
- You still have a question

In 2024, Is it worth it to learn Shell Scripting

YES

THE OPAL FRAMEWORK

- Pet Project of mine on Github
- Bash Dotfiles Framework
- In mid-to-late 2023 started on v3
- Inspired the content for my tek talks

GETTING STARTED

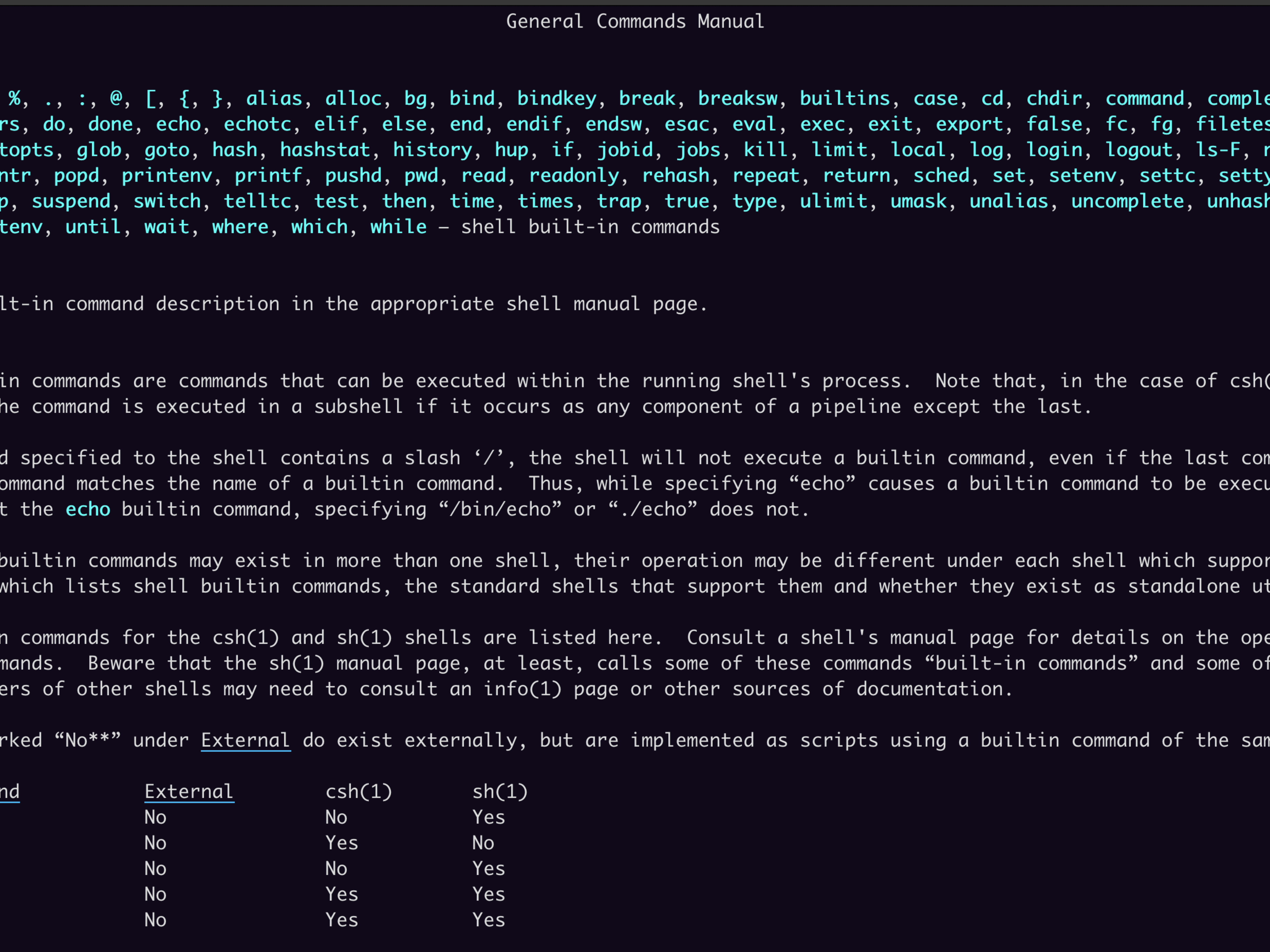
- Current version of Bash = 5.2
- Mac version of Bash = 3.2
- 4 prompts: $PS1, $PS2, $PS3, $PS4

COMMANDS TO KNOW

- `sed`
- `awk`
- `tr`
- `cut`
- `paste`

GETTING HELP

```
man alias
```



```
                                   General Commands Manual



 %, ., :, @, [, {, }, alias, alloc, bg, bind, bindkey, break, breaksw, builtins, case, cd, chdir, command, comple
rs, do, done, echo, echotc, elif, else, end, endif, endsw, esac, eval, exec, exit, export, false, fc, fg, filetes
topts, glob, goto, hash, hashstat, history, hup, if, jobid, jobs, kill, limit, local, log, login, logout, ls-F, n
ntr, popd, printenv, printf, pushd, pwd, read, readonly, rehash, repeat, return, sched, set, setenv, settc, setty
p, suspend, switch, telltc, test, then, time, times, trap, true, type, ulimit, umask, unalias, uncomplete, unhash
tenv, until, wait, where, which, while – shell built-in commands



lt-in command description in the appropriate shell manual page.



in commands are commands that can be executed within the running shell's process.  Note that, in the case of csh(
he command is executed in a subshell if it occurs as any component of a pipeline except the last.

d specified to the shell contains a slash ‘/’, the shell will not execute a builtin command, even if the last com
ommand matches the name of a builtin command.  Thus, while specifying “echo” causes a builtin command to be execu
t the echo builtin command, specifying “/bin/echo” or “./echo” does not.

builtin commands may exist in more than one shell, their operation may be different under each shell which suppor
which lists shell builtin commands, the standard shells that support them and whether they exist as standalone ut

n commands for the csh(1) and sh(1) shells are listed here.  Consult a shell's manual page for details on the ope
mands.  Beware that the sh(1) manual page, at least, calls some of these commands “built-in commands” and some of
ers of other shells may need to consult an info(1) page or other sources of documentation.

rked “No**” under External do exist externally, but are implemented as scripts using a builtin command of the sam

nd            External        csh(1)      sh(1)
              No              No          Yes
              No              Yes         No
              No              No          Yes
              No              Yes         Yes
              No              Yes         Yes
```

GETTING MORE HELP

- `help alias`
- `man bash`
- `tldr or cheat`

INS AND OUTS

- `0 = stdin`
- `1 = stdout`
- `2 = stderr`

```
ls -1rt | tail -n 10

alias | grep git

ls -1 *.md | pbcopy
```

WRITE TO STDOUT

```
echo 'hello world'
```

```
cat README.md
```

```
$ cat << TEK
> This is line one
> Second line
> TEK
```

STDERR

```
echo 'An error message' >&2

echo 'An error message' > /dev/stderr

std_error ‘An error message’
```

WHY USE STDERR

- To give the user an error message
- To report information that shouldn't appear in the content

DEV NULL

- Written as /dev/null
- Often used with stderr

```
command 2> /dev/null

cat /dev/null > error.log
```

OVERVIEW: VARIABLES

- `name=“value”`
- `echo $name`

OVERVIEW: ALIASES

- Name and value
- Takes no parameters

```
alias statmod="git status | grep 'modified: ' | cut -f2 -d:"
```

OVERVIEW: FUNCTION

- Name
- 1 or more parameters
- body

OVERVIEW: COMMANDS

- Name
- Options
- Arguments

PITFALLS

- Whitespace
- Global Scope by Default
- Lack of Parameter Names
- exit()

POSIX

- “portable operating system” code
- If you have 2 POSIX-compliant systems, the code written on one should work for the other
-

DEFINING FUNCTIONS

```
public function greet(User $user): void {
    echo “Hello {$user→getName()}”
}
```

DEFINING FUNCTIONS

```
function greet(User $user): void {
    echo “Hello {$user->getName()}”
}
```

DEFINING FUNCTIONS

```
function greet(User $user) {
    echo “Hello {$user→getName()}”
}
```

DEFINING FUNCTIONS

```
function greet($user) {
    echo “Hello {$user→getName()}”
}
```

DEFINING FUNCTIONS

```
function greet($user) {
    echo “Hello {$user}”
}
```

DEFINING FUNCTIONS

```
function greet($user) {
    echo “Hello ${user}”
}
```

DEFINING FUNCTIONS

```
function greet() {
    echo “Hello ${user}”
}
```

DEFINING FUNCTIONS

```
function greet() {
    local user=“$1”
    echo “Hello ${user}”
}
```

DEFINING FUNCTIONS

```
function greet() {
```

DEFINING FUNCTIONS

```
function greet {


         greet() {
```

CALLING FUNCTIONS

- Return success/failure, not data
- Need to capture output

```
greeting="$(greet John)"
```

RETURN STATUS

- `return 0    for success`
- `return 2    for error any # > 0`

$? captures the status of last run command

COMMAND SUBSTITUTION

```
echo “There are `ls | wc -l` files”
```

```
echo “There are $(ls | wc -l) files”
```

MAKING FUNCTIONS

- Prototype on the command line
- Wrap it
- Substitute values
- Add argument handling

MAKE A FUNCTION: CHANGED FILES

```
ls -1rt
```

MAKE A FUNCTION: CHANGED FILES

```
ls -1rt | tail -n 10
```

```
function changed {

    ls -1rt | tail -n 10

}
```

```
function dir:changed {
   ls -1rt | tail -n 10
}
```

```
function dir:changed {

    local -i quantity=10

    ls -1rt | tail -n $quantity

}
```

```
function dir:changed {
    local -i quantity=10
    if [[ -n $1 ]]; then
        quantity="$1"
    fi
    ls -1rt | tail -n $quantity
}
```

```
function dir:changed {
    local -i quantity=10
    if opal:is_set “$1”; then
        quantity="$1"
    fi
    ls -1rt | tail -n $quantity
}
```

```
function dir:changed {
    local -i quantity
    if [[ -z “$1” ]]; then
        echo “How many files?” >&2
        return 1
    fi
    quantity="$1"
    ls -1rt | tail -n $quantity
}
```

```
function dir:changed {
    local -i quantity
    if opal:is_unset “$1”; then
        opal:std_error “How many files?”
        return 1
    fi
    quantity="$1"
    ls -1rt | tail -n $quantity
}
```

MISSING FEATURES

- Shell documentation
- Coding Standard
- shfmt uses EditorConfig

DEBUGGING BASH

- Syntax Check

```
bash -n filename.bash
```

DEBUGGING BASH

- Debug Options
  - set -x   Display the expanded value of PS4
  - set -v   Print input lines as they're read
  - set -u   Unset variables are an error

```
function tek:ps4 {
    PS4="\n"
    PS4+='source-file: ${BASH_SOURCE}\n'
    PS4+='Function: ${FUNCNAME[0]:+${FUNCNAME[0]}} \n'
    PS4+='Line: ${LINENO} \n'
    PS4+="> "
    export PS4
}
```

SHELLCHECK

- Static Analysis tool
- Available in Neovim and PhpStorm

TEXT

THANK YOU

- Andrew Woods
- @awoodsnet@phpc.social
- andrewwoods.net