TEXT

ANDREW WOODS

- Seattle Web Developer
- SeaPHP Organizer
- PNWPHP Organizer

andrewwoods.net

@awoods

Seattle PHP

www.seaphp.com

Pacific Northwest PHP

Sept 15–17, 2016

www.pnwphp.com

ASSUMPTIONS

- You know what the command line is.
- You're familiar with the UNIX Philosophy
- You're comfortable with the essential 15.
- You've probably used tar, gzip, chmod, ssh
- You’re using Bash as your shell
- Your familiar with PHP code

OTHER SHELLS

- Z Shell
- C Shell
- Korn Shell
- Bourne Shell

UNIX PHILOSOPHY

- Do one thing and do it well
- Everything is a file
- Small is beautiful
- Store data and configuration in flat text files
- Use shell scripts to increase leverage and portability
- Chain programs together to complete complex task
- Choose portability over efficiency
- Keep it Simple, Stupid (KISS)

TEXT

ESSENTIAL 15 (OR SO) COMMANDS

cd, ls, man, pwd, rm

cat, cp, date, echo, mv

clear, head*, less*, mkdir, rmdir

SHELL CONFIGURATION

SHELL CONFIGURATION

- .bash_profile
- .bashrc
- .vimrc
- .gitconfig

TEXT

BASHRC

- Loaded by any shell
- Use to setup custom configurations
  - Aliases
  - Variables
  - Functions
  - Source files

TEXT

BASH PROFILE

- Anything that .bashrc can do
- Loaded once - by login shell
- Allows for customization

VARIABLES

- There are no strict data types
  - Number
  - String
  - Array
- Read using a $

TEXT

USING A VARIABLE

```
first_name = "Andrew"

echo $first_name

# This is a comment
```

SETTING VARIABLES

- declared/assigned
- when exported
- in an arithmetic expression ((...))
- in a read statement
- head of a loop

TEXT

SPACES VS. NO SPACES

```
# This is Correct
no_spaces = “no spaces around equal sign"

# Don’t Do This
spaces   =   “has spaces around equal sign"
```

TEXT

DIRECTORY VARIABLES

```
# In your .bashrc
client_proj = "/var/www/client/project/wp-content"
fun_proj = “/var/www/fun/project/wp-content”

# Later on the command line
$ cd  $client_proj
```

TEXT

AD HOC VARIABLES

```
# cd to your projects plugin directory
$  p =`pwd`

# cd to your projects theme directory
$  t =`pwd`


$ cd $p
```

ALIAS

ALIASES

- Use them where you'd use a command
- Reduce typing
- Increase simplicity

TEXT

ALIASES

```
alias staged='git diff --staged'
alias wpvi='vim -u ~/.vimrc_wordpress'
alias mute='osascript -e "set volume with output muted"'
alias unmute='osascript -e "set volume without output muted"'
alias please='sudo'
```

FUNCTIONS

TEXT

BENEFITS OF FUNCTIONS

- Encapsulate functionality
- Promote reusability
- Reduce need to memorize

TEXT

WHAT CAN YOU DO

- Execute Commands
- Execute Functions
- Define Loops
- Control Structures
- Define and Use Variables

TEXT

EXAMPLE FUNCTION

```
wp_siteurl () {
        wp option update home  $1
        wp option update siteurl  $1
}
```

TEXT

PREFERRED STYLE OF DEFINITION

```
function wp_siteurl {
        wp option update home $1
        wp option update siteurl $1
}
```

USING PARAMETERS

- Positional
- No names
- Not required

TEXT

PHP FUNCTION EXAMPLE

```
 <?php
function main( $one,  $two, $three="" ) {
    echo  "one=$one   two= $two\n";
    if ( $three ){
        echo  "three=$three \n";
    }
}
?>
```

BASH FUNCTION EXAMPLE

```
<?php
 function  main {
      echo  “one=$1    two= $2\n” ;
      if [  -n “$3 “ ]; then
          echo   “three=$3 \n” ;
      fi
}
?>
```

TEXT

CONTROL YOUR SCOPE

```
function example {
        second = “two”
        local third = “three”
}

first = “one”
example
```

REDIRECTION

FILE DESCRIPTORS

- 0 STDIN Standard Input
- 1 STDOUT Standard Output
- 2 STDERR Standard Error

REDIRECTION OPERATORS

- Commands redirection
  - uses the |
- File redirection
  - Input <
  - Output >
  - Append >>

REDIRECT TO A FILE

```
grep -RHn function . > output.txt
```

GET INPUT FROM A FILE

```
sort < data.txt
```

```
sort -k 2 -n -r < data.txt
```

TEXT

GET INPUT FROM ONE FILE OUTPUT ANOTHER

```
sort < data.txt > output.txt
```

```
sort -k 2 -n -r < data.txt > output.txt
```

PIPING COMMANDS TOGETHER

```
cat  my-blog-post.txt  |  wc -w
```

```
find  .  -name  “*.php”  |  grep ‘class-’
```

```
grep ‘@gmail.com’ file.csv | cut -f 3 |  sort  > email.txt
```

USEFUL COMMANDS

```
pwd | pbcopy

cat /dev/null > debug.log

find / -name “*.php" 2>/dev/null

pbpaste | say
```

COMMAND SUBSTITUTION

- Use the output of one command inside of another
- Back ticks and Subshells
- Capture output into a variable

USING BACKTICKS

- Simple to type
- Cannot be nested
- Allow for variable interpolation

`which php`

USING SUBSHELLS

- Allow for nesting
- Can do everything back ticks can do
- Pass arguments to them

$(which php)

```
653 #
654 # Display lines 100 through 140.
655 # $ seg error.log 100 140
656 #
657 function seg()
658 {
659     range=10
660     filename=$1
661
662     if [[ -z $3 ]]; then
663         start=$( calc $2-$range)
664         end=$(calc $2+$range)
665     else
666         start=$2
667         end=$3
668     fi
669
670     awk "NR >= $start  && NR <=  $end  " $file
671 }
672
673
674 #
675 # cal3 - Display the previous month, current m
676 #
677 function cal3()
678 {
679     cal -my $(date -v-1m "+%m %Y")
680     cal
681     cal -my $(date -v+1m "+%m %Y")
682 }
683
684 #
685 # ncal3 - Display the previous month, current
686 #
687 function ncal3()
688 {
689     ncal -my $(date -v-1m "+%m %Y")
690     echo ' '
```

SCRIPTING

FUNDAMENTALS

- The things you know are valuable
- The fundamentals are the same
- Your previous experience will guide you well

TEXT

SOURCE YOUR FILES

- Equivalent to include in PHP
- Logically organize your code

source ~/dotfiles/bash/functions

CONTROL STRUCTURES

- if/elif/else/fi
- case/esac

TEXT

```
if [[ -n $1 ]]; then
    message=$@
else
    message="It is Done!"
fi
```

TEXT

```
case "$1" in
one|first)       echo "the 1st thing" ;;
two|second)  echo "the 2nd thing" ;;
*)                    echo “the default " ;;
esac
```

LOOPS

- for
- while
- until

TEXT

FOR IN LOOP

```
# A C-style for loop
for (( i=1; i<20; i+=2 ))
do
    echo "do stuff $i"
done
```

TEXT

FOR LOOP

```
for  plugin_id in `cat plugins.txt`
do
    wp plugin install $plugin_id
done
```

TEXT

FOR LOOP – LIST OF NUMBERS

```
for i in $(seq 1 2 20)
do
    echo "do stuff $i"
done
```

TEXT

DIFFERENT WAYS TO WRITE FOR LOOP

```
for i in 1 1 2 3 5 8 13 21 34
for i in `seq 1 50`
for i in $(cat ~/file.csv)
for i in $(ls ~/data)
for i in {1..15}
for i in EWR  JFK  LAX  LGB  SAN  SEA  SNA
```

TEXT

WHILE LOOP

```
while [ condition ]; do
    # your commands here
done
```

TEXT

WHILE - LIST OF NUMBERS

```
n=1
while [ $n -le 20 ]
do
    echo $n
    n=$(( n+2 ))
done
```

DEBUGGING SCRIPTS

- set -x Turn on debug mode
- set -n Read commands but do not execute
- set -v Verbose output

LOGICAL OPERATORS

- And &&
- Or ||
- Not !

OPERATOR EXAMPLES

```
function mkcd
{
    mkdir -p "$1" && cd "$1"
}
```

GOTCHAS

- Math
- Whitespace

BASH DOES INTEGER MATH

- No Floating Point numbers

```
echo $(( 16 / 3 ))      #  5

echo $(( 16 % 3 ))      #  1
```

WHITESPACE

- Used for separating arguments
- Sometimes you’ll need to use quotes to simplify

KEEP
CALM
AND
DO YOUR
HOMEWORK

HOMEWORK

HOMEWORK

- Look at Opal - https://github.com/andrewwoods/opal
- Create your own dot files project
  - Aliases
  - Variables
  - Functions

GENERAL COMMANDS TO KNOW

- xargs
- set -o vi
- wp-cli
- curl
- grep / ag

OS X COMMANDS TO KNOW

- pbcopy
- pbpaste
- open
- mdfind
- say