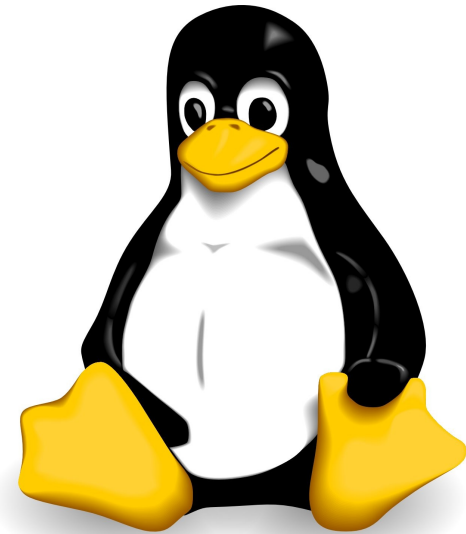


The **BASH**ing Admins

The ICS Shell Scripting Class

William Malchisky Jr.
Effective Software Solutions, LLC



Agenda - Building a Solid Scripting Building



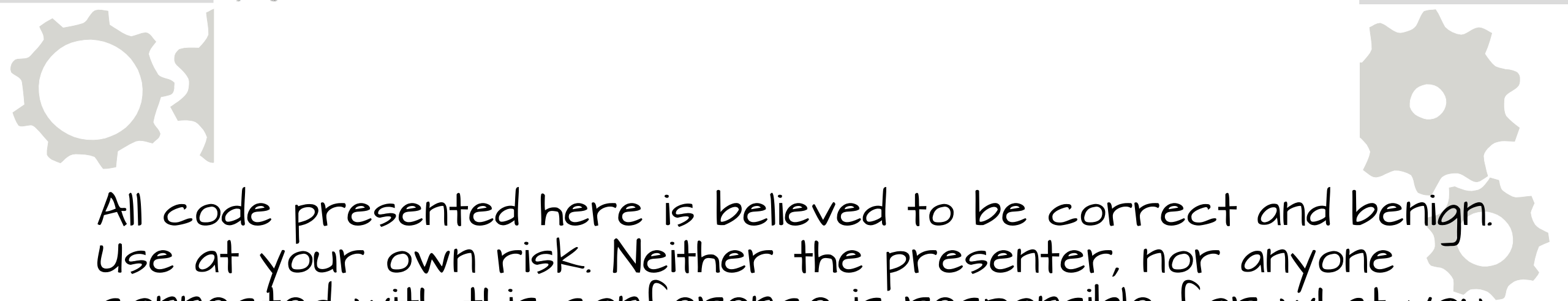
- ← Goodies
- ← Domino start script
- ← Housekeeping
- ← Audience Participation
- ← DB2 Assistance
- ← Interacting w/IBM and Domino
- ← Non-production backup
- ← Remote access tips
- ← Foundational knowledge

Agenda - Building a Solid Scripting Building



Foundational Knowledge

Disclaimer



All code presented here is believed to be correct and benign. Use at your own risk. Neither the presenter, nor anyone connected with this conference is responsible for what you do or might do to your environment.

Have a nice day... :)

Scripting Wisdom

- Even as admins, avoid bad practices
 - Avoid hard coding names and paths, as they can change
 - System upgrades, patches can change a mount point (e.g. local NAS)
- Use IP addresses for testing on network issues primarily
Hostnames whenever possible
- Integer math in BASH; no floating point

"Please note that you should quote patterns as a matter of course, otherwise the shell will expand any wildcard characters in them." -
find man page

Getting Started – “Shebang!”

All scripts start this way

```
#!/<path_to_shell>
```

```
malchw@san-domino:~$ which bash  
/bin/bash
```

```
#!/bin/bash
```

Time Saving Preparation Steps

1. Get organized!

```
$mkdir scripts; cd scripts
```

Copy `template.sh` to `<new_script_name.sh>` each time

```
$cp template.sh selstatus.sh && vi selstatus.sh
```

2. Create directory trees on-the-fly

```
malchw@san-domino:~$ mkdir -p scripts/test/pub/{beer/{ale,stout},wine/{red,white},music,snacks,softdrinks/{soda,juice,water},pos} && tree scripts/test
```

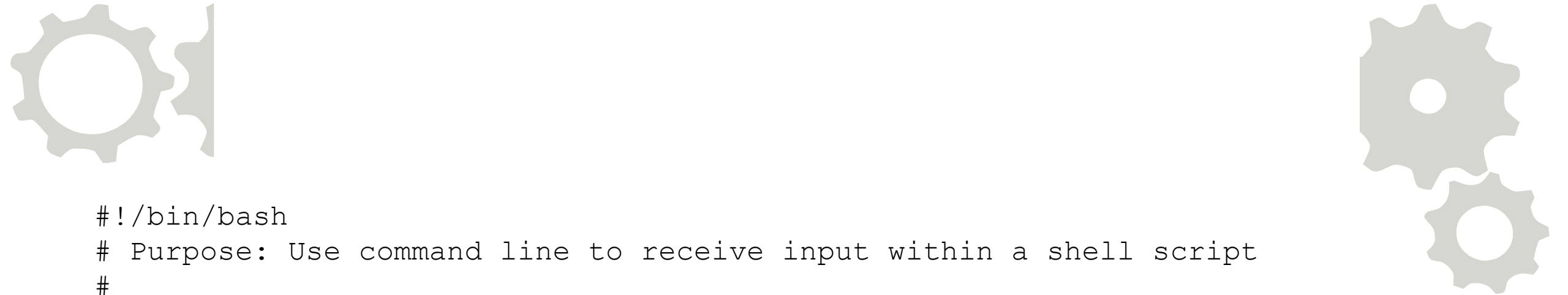
Note: Be certain to make your scripts executable

```
scripts/test
├── pub
│   ├── beer
│   │   ├── ale
│   │   └── stout
│   ├── music
│   ├── pos
│   ├── snacks
│   ├── softdrinks
│   │   ├── juice
│   │   ├── soda
│   │   └── water
│   └── wine
│       ├── red
│       └── white
```


- When executing your script, each space delimited value is passed to the script
- Access the values numerically: \$1, \$2 for two arguments

```
$ ./backup.san-domino.full.sh WD4TB-EXT4 2015.mar.24
```

Receiving Input – Ask for Data



```
#!/bin/bash
# Purpose: Use command line to receive input within a shell script
#
echo "What's your favorite beverage?";
read a;
echo "It's great you like $a. What is your second favorite beverage?";
read b;
echo "Excellent to know you appreciate $a and $b";
echo "*****"
```

Viewing Output – Ask for Data

```
malchw@san-domino:~/scripts$ ./input.example.sh
```

```
What's your favorite beverage?
```

```
Beer
```

```
It's great you like Beer. What is your second favorite beverage?
```

```
Wine
```

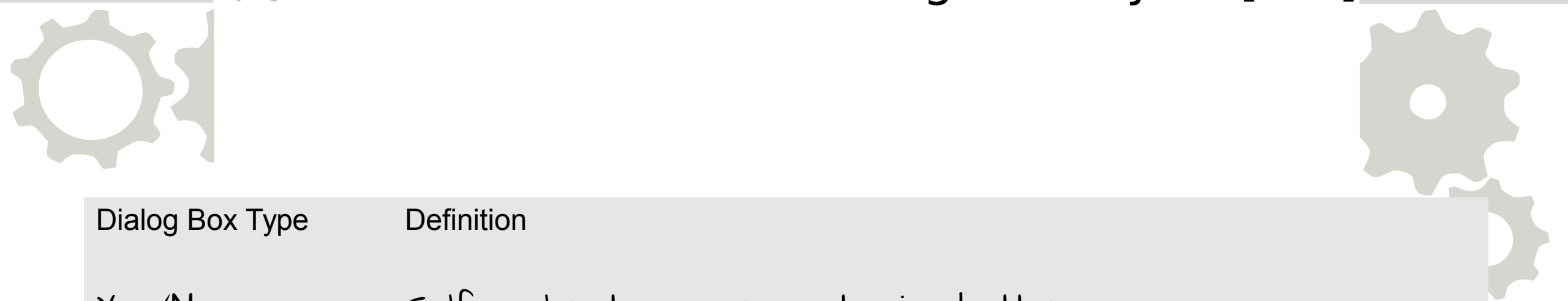
```
Excellent to know you appreciate Beer and Wine
```

```
*****
```

Dialog Boxes with whiptail

- *whiptail* runs on all current Linux systems
- Many customization options (e.g. `--defaultno`)
`$man whiptail` for details
- Like `@prompt` in Formula language, several types


Ten Dialog Box Styles [1-5]



Dialog Box Type	Definition
Yes/No	Self-explanatory; can customize buttons
menu	Self-explanatory
input	Easy way to receive input from user
text	Display a text file's contents; arrow keys work; l/r to scroll
message	Single OK button

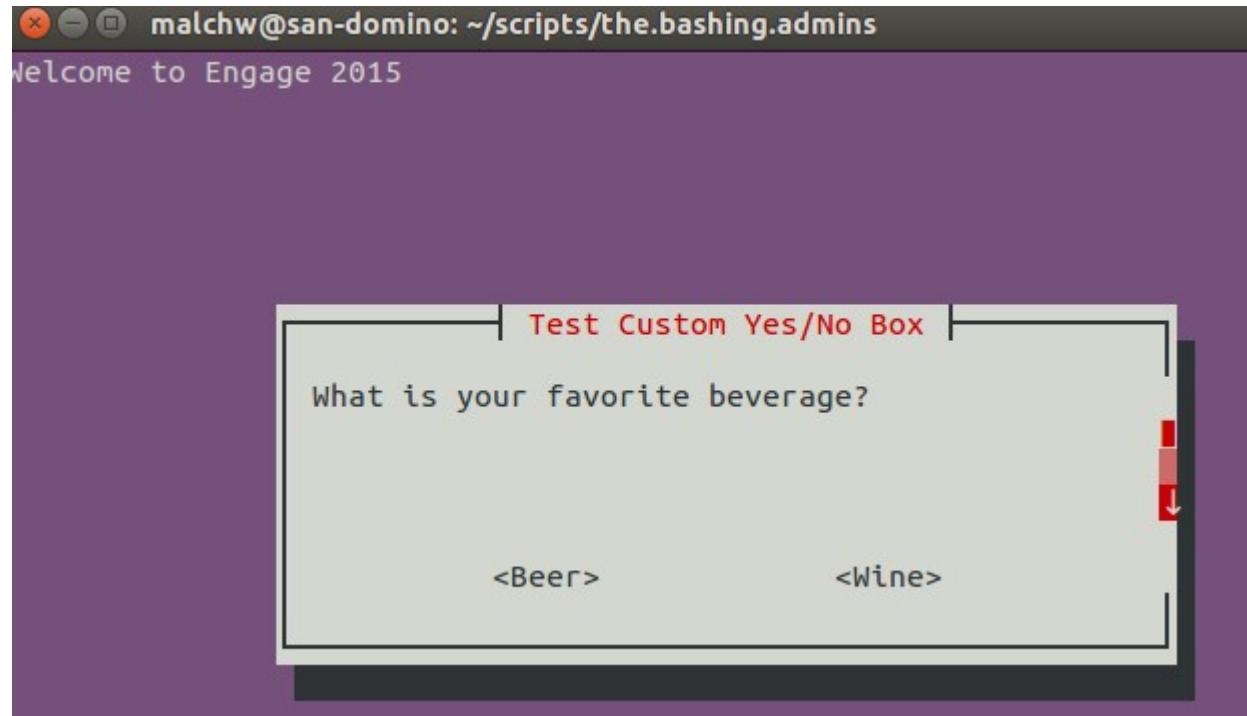
Ten Dialog Box Styles [6-10]

Dialog Box Type	Definition
info	Display message, whiptail exits, cleared later by script
checklist	Menu box; select multiple options via space bar
radiolist	Menu box; select only one
gauge	Display a progress bar; static image; run in loop
password	Input box, entered text invisible; don't use init with it



```
#!/bin/bash
# File: whiptail.example.sh
# Notes:
#
if (whiptail --title "Test Custom Yes/No Box" --backtitle "Welcome to Engage 2015"
--scrolltext --yes-button "Beer" --no-button "Wine" --yesno "What is your favorite
beverage?" 10 50) then
    echo "You love beer (Exit status for 'yes' is $?.)"
else
    echo "You love wine (Exit status for 'no' is $?.)"
fi
```

Whiptail – Display Dialog



A terminal window with a purple background. The title bar shows 'malchw@san-domino: ~/scripts/the.bashing.admins'. The terminal text reads 'Welcome to Engage 2015'. A dialog box titled 'Test Custom Yes/No Box' is displayed in the center. The dialog box has a light gray background and a black border. It contains the text 'What is your favorite beverage?' and two options: '<Beer>' and '<Wine>'. A red vertical bar with a downward arrow is on the right side of the dialog box.

```
malchw@san-domino: ~/scripts/the.bashing.admins
Welcome to Engage 2015


Test Custom Yes/No Box
What is your favorite beverage?
<Beer>           <Wine>
```


Whiptail – Terminal Output

```
malchw@san-domino:~/scripts$ ./whiptail.example.sh
You love beer (Exit status for 'yes' is 0.)
malchw@san-domino:~/scripts$ ./whiptail.example.sh
You love wine (Exit status for 'no' is 1.)
```

Detailed code samples for each box type are available here:
<http://xmodulo.com/create-dialog-boxes-interactive-shell-script.html>

Seeing Double? Clarifying Duplicate Symbols



Operand	Definition
&&	Run second command if first completes successfully
	Run second command if first <u>fails</u>
(())	Arithmetic expression: +, -, *, / are used with assignments (=) or tests (<, >)
\$(())	Arithmetic expansion - expression replaced with result of the mathematic evaluation: $\$(((x-y) * 4))$



Conditional
Execution

Best Practice – Avoid creating a chain of multiple commands connected by conditional operands

Seeing Double?

Clarifying Duplicate Symbols: [, [[

Keywords	Definition
[Test; simple command for simple tests; compound arguments can break it; all arguments treated the same-no special considerations
[[New test; no word splitting, properly parses commands; only used with BASH, zsh, ksh

Best Practice – Unless you need non-BASH portable code, [[is preferred to [

Seeing Double?

Clarifying Duplicate Symbols: [, [[

Subset of supported tests by [(or test)

Test Case	Result	Test Case	Result
-e <file>	File exists?	-h <file>	Is symbolic link?
-f <file>	Regular file?	-r <file>	Can user running command read file?
-d <file>	File is a directory?	-s <file>	File exists and contains data?
-w <file>	Writable by user running command?	<f1> -nt <f2>	F1 newer than F2?
		<f1> -ot <f2>	F1 older than F2?

Seeing Double?

Clarifying Duplicate Symbols: [, [[

Subset of supported tests by [[(or new test)

New Test Case	Result
String = PATTERN	By default, pattern match occurs against, rather than a comparison, when special symbols exist
EXP {&&, } EXP	So, logical and or logical or within a test condition between two expressions

Note: if you leave the variable quoted on the right side of && or ||, [[will evaluate as a literal

Seeing Double?

Clarifying Duplicate Symbols: [, [[

Best Practice - Avoid using -a and -o with [
Instead, link multiple tests with either a conditional
and, or symbol, to avoid unpredictable results on
some shells

```
if [ "$job" = admin ] && [ "$role" = senior ]; then  
    echo "You are a senior admin."  
fi
```

--OR--

```
if [[ "$job" = "admin" && "$role" = "senior" ]]  
then  
    echo "You are a senior admin."  
fi
```

Contrasting Duplicate Symbols: ', ""

Symbol

Definition

Single quote

Preserves the literal value of all characters;
no ' inside expression

Views each character literally

Double quote

Preserves the literal value of all characters,
excepting \$, ;, \

Interpolates string as expression.

Seeing Double?

Clarifying Duplicate Symbols: ', ""

Knowing which to construct to use for the desired need is key to obtaining accurate results, while minimizing troubleshooting

```
#!/bin/bash
PUB="is open"
echo "double quotes: pub $PUB"
echo 'single quotes: pub $PUB'

malchw@san-domino:~/scripts$ ./quotes.sh
double quotes: pub is open
single quotes: pub $PUB
```


Seeing Double?

Clarifying Duplicate Symbols: ', ""

Two more examples - contrasting alias usage for quotes

```
alias purgeoldkernels='~/scripts/purgeoldkernels.sh'  
alias installmissingkernelfiles='sudo apt-get install linux-headers-  
`uname -r`'
```

Versus

```
alias top6="ps -eo pcpu,user,pid,cmd | sort -r | head -7"  
alias latestfile="ls -t1 | head -n1"
```

Agenda - Building a Solid Scripting Building



Remote access tips

Quick SSH Tips

- Domino, Mobile Connect, Sametime, and Connections have multiple servers
Here's how you can access them easily and frequently

- Use aliases for each server to login easily and intuitively to each box
Store in `~/.bashrc` or `~/.bash_aliases`

- Naming construct examples - `<action><hostname | hostcode>`

```
alias sshcxn1='ssh awesomeadmin@172.20.10.10'  
alias sshcxn2='ssh user25@172.20.10.11 -p 9999'  
alias sshmail01='ssh user25@172.20.11.12 -p 8888'  
alias sshimc='ssh user25@172.20.11.17 -p 10101'  
alias sshstsc='ssh user25@172.20.12.26'
```

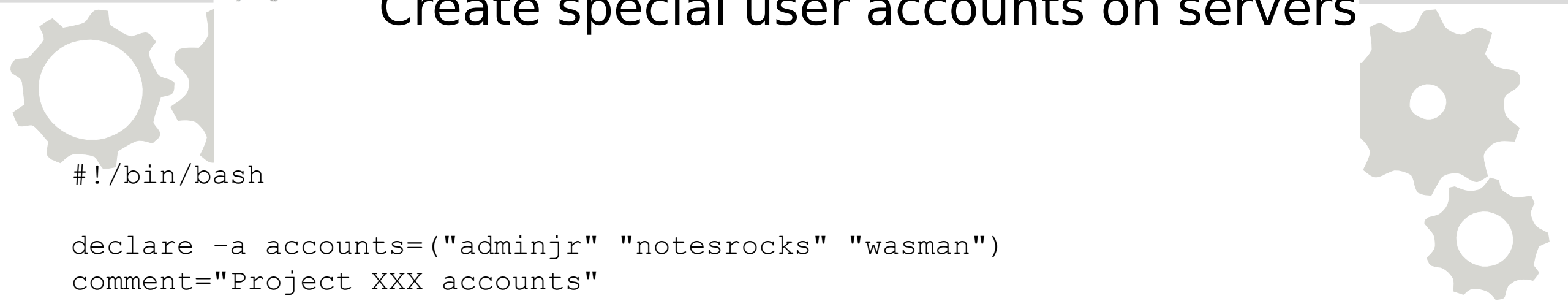
Quick SCP Tips

- Adding to the list... ensure you can move files up to those same boxes
- Use functions for each server to handle easily file management.
Store in `~/.bashrc` or `~/.bash_aliases`

- Naming construct examples - `function <action><hostname | hostcode>`

```
function scpcxn1() { scp $1 awesomeadmin@172.20.10.10:/dl/cxn5.x ; }  
function scpcxn2() { scp -P 9999 $1 user25@172.20.10.11:/dl/cxn5.x ; }  
function scpmail01 () { scp -P 8888 $1 user25@172.20.11.12:/dl/ibm/domino9.x ; }
```

Create special user accounts on servers



```
#!/bin/bash
```

```
declare -a accounts=("adminjr" "notesrocks" "wasman")  
comment="Project XXX accounts"
```

```
#for loop through array to create all accounts on the fly  
for i in "${accounts[@]}"  
do  
    useradd -c "$comment" "$i"  
done
```

```
malchw@san-domino:~/scripts/the.bashing.admins$ sudo ./createaccounts.sh
```

Use Case: Create special user account – Verification

```
malchw@san-domino:~/scripts/the.bashing.admins$ tail -n 3 /etc/passwd  
/etc/group
```

```
==> /etc/passwd <==
```

```
adminjr:x:1003:1004:Project XXX accounts:/home/adminjr:  
notesrocks:x:1004:1005:Project XXX accounts:/home/notesrocks:  
wasman:x:1005:1006:Project XXX accounts:/home/wasman:
```

```
==> /etc/group <==
```

```
adminjr:x:1004:  
notesrocks:x:1005:  
wasman:x:1006:
```

Building Blocks – Add user Accounts to Multiple Servers

- Load *.bashrc* with *ssh* and *scp* aliases for target servers
- Use *scp* variants to send *createaccounts.sh* to all servers
- Engage *ssh* variants to each server
- Run script locally

Huge time saver!


Agenda - Building a Solid Scripting Building



← Non-production backup

- Getting data protected before servers go live, testing and development servers, or local virtual machines
- GUI tools can be unreliable
 - Speaking from experience
- Write a script

Non-production Backups – Comment Section



```
#!/bin/bash
# Backup /home/[malchw,vmware]; /media/[big-data,virtual-machines]
# documents to a mounted <n>TB USB NAS drive
# Arguments:
# 1 -- Mounted NAS name: [WD4TB-EXT4, "My Passport"]
# 2 -- Date for inclusion into filename
# Updated 9 Dec 2014
# #####
```

Non-production Backups – Code Review

```
mkdir /media/$USER/$1/backup/$2
tar cvzf "/media/$USER/$1/backup/$2/home.malchw.$2.tgz" /home/malchw
tar tvzf "/media/$USER/$1/backup/$2/home.malchw.$2.tgz" > "/media/$USER/$1/backup/$2/home.malchw.$2.toc"
tar cvzf "/media/$USER/$1/backup/$2/home.vmware.$2.tgz" /home/vmware
tar tvzf "/media/$USER/$1/backup/$2/home.vmware.$2.tgz" > "/media/$USER/$1/backup/$2/home.vmware.$2.toc"
tar cvzf "/media/$USER/$1/backup/$2/virtual.machines.$2.tgz" /media/virtual-machines
tar tvzf "/media/$USER/$1/backup/$2/virtual.machines.$2.tgz" > "/media/$USER/$1/backup/$2/virtual.machines.$2.toc"
tar cvzf "/media/$USER/$1/backup/$2/big-data.$2.tgz" /media/big-data
tar tvzf "/media/$USER/$1/backup/$2/big-data.$2.tgz" > "/media/$USER/$1/backup/$2/big-data.$2.toc"
```

Non-production Backups – Customize It

- Use `date` command instead of passing an argument

```
malchw@san-domino:~$ date "+%A %-m-%d-%y %-I:%M:%S %p %Z"  
Saturday 3-14-15 9:26:53 AM CDT
```

- Try utilizing a symbolic link in your data directory to the mount point

Agenda - Building a Solid Scripting Building



← Interacting w/IBM and Domino

Script Ideas

- IBM released *imcsupport.sh* which collects log files and FTPs to their site for IMC
 - Extrapolates for other products quite easily
- Domino system maintenance scripts
 - Setup for *fixup*, *updall*, and *compact* via IND files
 - Run during system maintenance to quickly process all targeted files

Agenda - Building a Solid Scripting Building



DB2 Assistance

Working with Connections, DB2 Gets Easier

- Courtesy of Christoph Stoettner

- Two resources

1. His Github DB2 code repository

<https://github.com/stoeps13/ibmcdnxscripting/tree/master/DB2>

2. Engage 2014 deck with Sharon Bellamy

DB2 specific code around IBM Connections

<http://de.slideshare.net/ChristophStoettner/practical-solutions-for-connections-administrators-extended>

<http://www.slideshare.net/ChristophStoettner/practical-solutions-for-connections-administrators-extended>

Agenda - Building a Solid Scripting Building



← Aggregate Knowledge


Audience Participation Time

What's wrong with the code?

Example 1 - Error

```
malchw@san-domino:~/scripts/the.bashing.admins$ ./echo2.sh  
bash: ./echo2.sh: Permission denied
```

Example 1 - Answer

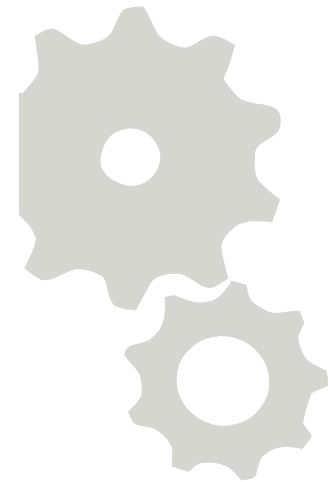


```
malchw@san-domino:~/scripts/the.bashing.admins$ ls -l echo2.sh  
-rw-r--r-- 1 malchw malchw 27 Mar 29 09:50 echo2.sh
```

Need to set executable status

```
malchw@san-domino:~/scripts/the.bashing.admins$ chmod 744 echo2.sh  
malchw@san-domino:~/scripts/the.bashing.admins$ ls -l echo2.sh  
-rwxr--r-- 1 malchw malchw 27 Mar 29 09:50 echo2.sh
```

```
malchw@san-domino:~/scripts/the.bashing.admins$ ./echo2.sh  
malchw
```



Example 2 - Error

```
[ "$pub" = bar && "$bar" = beer ]
```

Example 2 - Answer

- Can't use "&&" inside test
- The && operand requires new test only
- Correct code

```
[[ "$pub" = bar && "$bar" = beer ]]
```

Example 3 - Error

```
[ $foo = "bar" ]
```

Example 3a - Error

```
malchw@san-domino:~/scripts/the.bashing.admins$ ./foobar.sh  
./foobar.sh: line 10: [: == unary operator expected
```


Example 3a - Answer

- Quotes in wrong place.
- Variables can contain metacharacters or pattern characters
Use quotes to protect your script
- Definitely if whitespace or wildcards are present
- If `"foo" = ""` then

```
[ $foo = "bar" ] --> [ = "bar" ]
```

"=" is binary, needing two operands to compare, ergo the unary error

Thus, ["\$foo" = bar]

Example 3b - Error

Changed value for foo, same code

```
malchw@san-domino:~/scripts/the.bashing.admins$ ./foobar.sh  
./foobar.sh: line 12: [: too many arguments
```

Example 3b - Answer

- If "foo" = "the cat" then

```
[ $foo = "bar" ] --> [ the cat = "bar" ]
```

Too many arguments for the binary operator

- Thus, ["\$foo" = bar] or [[\$foo = "bar"]]

Example 4 - Error

```
cp $file $target
```

Example 4 - Answer

- Sans quotes, expansion issues can be problematic, especially if moving files from Windows servers (they usually contain spaces)
- `cp System Reports Q2.docx /mnt/nas/archive/`
- Which breaks as indicated
- `malchw@san-domino:~/Documents/scripts$ cp System Reports Q2.docx /mnt/nas/archive/`

```
cp: cannot stat 'System': No such file or directory
cp: cannot stat 'Reports': No such file or directory
cp: cannot stat 'Q2.docx': No such file or directory
```
- If wildcards exist, filenames get expanded, unless inside quotes

Example 4 - Answer (Continued)

If the file commences with a '-', you have other problems...

1. Reference file via a relative path `"./-file1.docx"`
2. Tell the command to ignore the dash to avoid having it interpreted as a parameter

e.g. `cp -- <source> <target>`

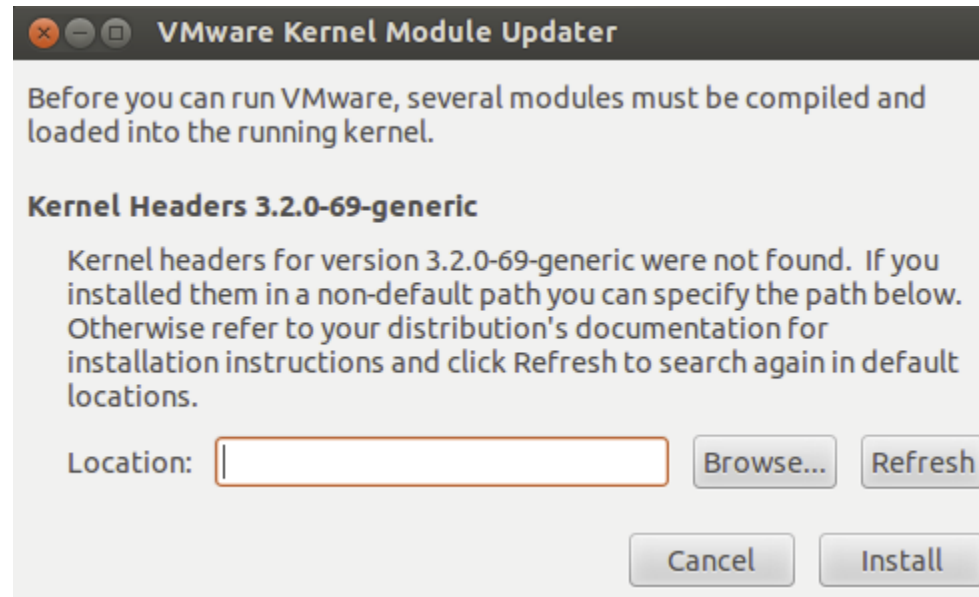
Agenda - Building a Solid Scripting Building



← Housekeeping

Installing Missing Kernel Files

- When VMware is unable to find the kernel header's *include* directory
- Some distros install the kernel sans the source code
- Here's a quick fix to save time



```
alias installmissingkernelfiles='sudo apt-get install  
linux-headers-`uname -r`'
```


Freeing Disk Space - Purge Old Kernels



Here's the code

```
#!/bin/bash
sudo apt-get remove --purge $(dpkg -l 'linux-*' | sed
'/^ii/!d;/'"$(uname -r | sed "s/\(.*\)-\([^0-9]\
+\)/\1/"'"')"/d;s/^[^
]* [^
]* \([^\
]*\) .*/\1/;/[0-9]/!d')
sudo update-grub2
```

Freeing Disk Space - Purge Old Kernels

Dissecting the code

```
#!/bin/bash
sudo apt-get remove --purge $(dpkg -l 'linux-*' | sed
'/^ii/!d;/'"$(uname -r | sed "s/\(.*\) - \([0-9]\
+\)/\1/"'"'/d;s/^[^ ]* [^ ]* \([^\ ]*\)\.*/\1/;/[0-9]/!d')
sudo update-grub2
```

Remove files, and
purge configuration files

Create removal list
From Linux Kernel
List based upon
pattern

"\$(uname -r | ...)" provides the current kernel to use as a baseline

- Sed is a powerful amazing single pass data stream filtering parser
- With some finesse, you can extract precisely what you want from streamed data

Running the code

```
malchw@san-domino:~$ purgeoldkernels
```

```
Building dependency tree
```

```
Reading state information... Done
```

```
The following packages will be REMOVED:
```

```
linux-headers-3.13.0-43* linux-headers-3.13.0-43-generic*
```

```
linux-headers-3.13.0-44* linux-headers-3.13.0-44-generic*
```

```
linux-image-3.13.0-43-generic* linux-image-3.13.0-44-generic*
```

```
linux-image-extra-3.13.0-43-generic* linux-image-extra-3.13.0-44-generic*
```

```
0 upgraded, 0 newly installed, 8 to remove and 31 not upgraded.
```

```
After this operation, 542 MB disk space will be freed.
```

```
Do you want to continue? [Y/n]
```

Freeing Disk Space - Purge Old Kernels

Contrasting the results - Before

```
malchw@san-domino:~$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/vg--01-vol1--root	69G	52G	14G	80%	/
none	4.0K	0	4.0K	0%	/sys/fs/cgroup
udev	16G	4.0K	16G	1%	/dev
tmpfs	3.2G	1.8M	3.2G	1%	/run
none	5.0M	0	5.0M	0%	/run/lock
none	16G	11M	16G	1%	/run/shm
none	100M	36K	100M	1%	/run/user
/dev/sda1	180M	119M	48M	72%	/boot
/dev/sda5	11G	28M	11G	1%	/free
/dev/mapper/vg--02-vol3--home	80G	56G	20G	75%	/home
/dev/mapper/vg--01-vol2--vmware	47G	28G	17G	64%	/home/vmware
/dev/sdb1	321G	229G	77G	75%	/media/virtual-machines
/dev/sdb2	138G	90G	42G	69%	/media/big-data

Freeing Disk Space - Purge Old Kernels

Contrasting the results - After

```
malchw@san-domino:~$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/vg--01-vol1--root	69G	51G	15G	78%	/
none	4.0K	0	4.0K	0%	/sys/fs/cgroup
udev	16G	12K	16G	1%	/dev
tmpfs	3.2G	2.9M	3.2G	1%	/run
none	5.0M	0	5.0M	0%	/run/lock
none	16G	11M	16G	1%	/run/shm
none	100M	40K	100M	1%	/run/user
/dev/sda1	180M	46M	122M	28%	/boot
/dev/sda5	11G	28M	11G	1%	/free
/dev/mapper/vg--02-vol3--home	80G	56G	20G	75%	/home
/dev/mapper/vg--01-vol2--vmware	47G	28G	17G	64%	/home/vmware
/dev/sdb1	321G	229G	77G	75%	/media/virtual-machines
/dev/sdb2	138G	90G	42G	69%	/media/big-data

Agenda - Building a Solid Scripting Building



← Domino start

Daniel Nashed's Code – New Version Launch!

- Today, Daniel Nashed is officially releasing the next version of his famed Domino Start/Stop Script
- Announcement here -
<http://blog.nashcom.de/nashcomblog.nsf/dx/find-us-at-engage-conference-next-week.htm>
- Promised systemd support at ConnectED's Linuxfest VI - RHEL 7 and SLES 12
Kept his word
- If you see Daniel, say, "Thanks!" This is a tremendous gift to the ICS/Lotus Community, which he continually patches and evolves
significant effort

Logic and Limitations

- Process ID is that of the `rc_domino_script`, not of the Domino server process
 - New change in `systemd` support and welcomed
 - Process ID is needed for `systemd` to work properly
 - Extra file in new release to define the service → `"domino.service"`
- Workflow
 - `'rc_domino'` starts the `domino.service` (via `systemd`) → invokes the `rc_domino_script` → calls Domino
- No *restart live* support with `systemd`
 - Adds too much code complexity for what is a nice-to-have feature


Documentation is Enhanced

- *resources*, now display server's resources:
Process, shared memory, message queues and semaphores - for starters
- Readme file is completely updated
- Newly expanded *Known Issues* section
- Look for a RHEL7 and systemd primer from BillMal in Q2 '15 on my blog
- To get Daniel's code → <http://tinyurl.com/dominostartstop>

Agenda – Building a Solid Scripting Building

← Goodies

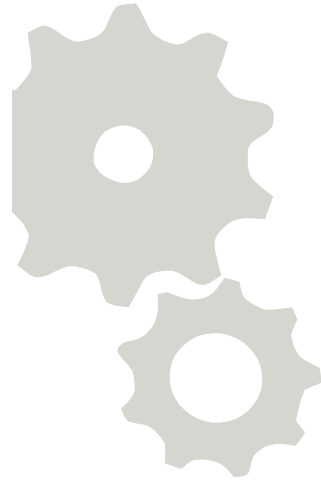


- 
- The A-Z of Programming Languages - Interview with Steve Bourne
http://www.computerworld.com.au/article/279011/a-z_programming_languages_bourne_shell_sh/
 - History with time stamp
 - <http://linux.byexamples.com/archives/467/list-command-line-history-with-timestamp/>
 - Very easy to setup - takes less than five minutes
 - Reserved word list
https://www.gnu.org/software/bash/manual/html_node/Reserved-Word-Index.html

- What is the difference between test, [, and [[?
 - <http://mywiki.woledge.org/BashFAQ/031>
 - <http://stackoverflow.com/questions/3427872/whats-the-difference-between-and-in-bash>
- Arithmetic Expression in BASH
 - <http://mywiki.woledge.org/ArithmeticExpression>
- Writing Shell Scripts
 - http://linuxcommand.org/lc3_writing_shell_scripts.php



- Co-founder of Linuxfest at Lotusphere/Connect
- Speaker at 20+ Lotus® related events/LUGs
- Co-authored two IBM® Redbooks on Linux®
- IBM Champion for Collaboration Solutions
2015, 2014, 2013, 2012, 2011
- Linux aficionado



How to contact me:
Bill Malchisky Jr.

william.malchisky@effectivesoftware.com

@billmalchisky
Skype: FairTaxBill

