Lecture overview

- Introduction
- Loops
- Conditionals and their use
- Backticking and similar methods
- Making GUI in scripts
- Service scripts
- Summary
Why command line?

- Use not limited to GUI design
- No need to obey GUI’s rules
- GUI applications tend to be less stable
- Easier to hack command-line tools
- Command line applications usually “do the job” better
- Repeatablilty (no memory from previous session)
- Scriptability
- Automation
When to do scripting in Bash

Do it in Bash...

- ... at shell prompt
- ... when there are a lot of application calls
- ... for system scripts
- ... if you don’t want to get into the Perl vs. Python war

Don’t to it in Bash (use Perl / Python instead) ...

- ... when the script itself should do something nontrivial
- ... when you want setuid root
Common use of bash scripts

- .bashrc, .bash_profile, .bash_logout
- Services going on and off
- ./configure
- In makefiles
- One-liners at prompt
Shebang and friends

- Comments in Bash scripts start with a 
- Bash scripts start with `#!/bin/bash` ("shebang")
- Line breaks are bridged with "\" (backslash, like C)
- Group commands: With '{' and '}'
- Group commands in subshell: With '()''

... and a couple of special parameters:

- `$$` expands to the current process number. Good for temporary files:
  - `tmpfile=delme-tmp-$$`
- `$1, $2, $3,...` are the arguments passed to the script
Loops in bash

- for i in Hello World ; do echo $i ; done
- while [ 1 ] ; do echo Wow ; done
- for ((i=0 ; i<10 ; i++)) ; do echo $i ; done

Note: If you want to kill a loop (in absence of CTRL-C), you have to kill the bash process itself
Conditionals in Bash

- Every executable is a conditional by its return value:
  
  ```bash
  while true ; do echo Wow ; done
  while grep -q audio /proc/modules
    do echo Audio! ; done ;
  ```

  ... but don’t use `true` and `false`!

- If `[ -d /etc ]` ; then echo Yes ; fi
  
  ```bash
  if [ -d /etc ] ; then echo Yes ; fi
  ```

  `'[' and ']' mean Bash `test`, so it’s the same as

  ```bash
  if test -d /etc ; then echo Yes ; fi
  ```

- `'[' and ']'` are “enhanced” but not sh-compatible. These two mean the same:
  
  ```bash
  if [ -d /etc -a -d /bin ] ; then echo Yes ; fi
  ```

  ```bash
  if [[ -d /etc && -d /bin ]] ; then echo Yes ; fi
  ```
Now some binary operations. Below, all “Yes” will be printed, all “No” will not.

```bash
if [[ "12" == 12 ]] ; then echo Yes ; fi
if [ "12" = 012 ] ; then echo Yes ; fi
if [[ "12" == 012 ]] ; then echo No ; fi
if [ "12" -eq 012 ] ; then echo Yes ; fi
if [[ "12" -eq 012 ]] ; then echo No ; fi
if [[ "12" -eq 12 ]] ; then echo Yes ; fi
```

According to the man page, `-eq` and friends are arithmetic and `==` is stringwise lexicographic. (This is not Perl)

`=` is like `==` in test context (Yuck!)

Conclusion: RTFM, and think twice if you want to use this
Using conditionals

- while loops as we’ve seen
- `rm -f *.o && make`
  ... which is the same as
  `if rm -f *.o ; then make ; fi`
- Note the semicolons!
- `''''`,''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
The name of the game is ' ( ( ) ) '

`echo $((1+1))` and `echo $((2**8))`

All arithmetics is with integers

Conditionals and autoincrement (instead of for-loop):

```bash
i=0; while ((i<10)); do echo $((i++)); done
```

```bash
i=1; while ((i<256)); do echo $((i*=2)); done
```

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

if (( $# < 1 ));
    then echo "Usage: $0 destination-path"; exit 1;
fi

if [ -a $1 ];
    then echo "File/dir $1 already exists"; exit 1;
fi

mkdir $1 || { echo "Failed to mkdir $1"; exit 1; } ||

...
The almighty backtick

- Run a command (or commands) and organize standard output as arguments delimited by spaces:

```bash
$ which bash
/usr/bin/bash

$ ls -l `which bash`
-rw xr-xr-x 1 root root 478720 Feb 19 2002 /usr/bin/bash

$ echo `find . -true`
./file1 ./file2 ./file3
```
The “for i in” loop

- for i in file1.c file2.c
do grep -H \#define $i;
done
- for i in *.c; do grep -H \#define $i; done
- for i in {a,b,c}-{d,e,f}; do echo $i; done
- for i in `find . -name \*.c`
do grep -H \#define $i; done
The problems with backticks

- File names with spaces: “my file.doc” looks like two files: “my” and file.doc
- Quotation marks don’t solve this!
- May exceed maximal number of arguments for Bash.
- Loop starts only when backticked command finishes: Slow response
- The solution: Use the `read` builtin command:
  ```bash
  find . -name \*\.c । while read i ;
  do grep -H \#define "\$i" ; done
  ```
- Note the quotation marks – they take care of the spaces in the file names!
Read the “find” man page!

This is not really about Bash, but still...

```bash
for dir in / ; do
    find /$dir -newer /etc/computer-bought-date \\
        ! -type d >> $1/backup-files;
done;

{ tar -c --to-stdout --preserve \
    --files-from $1/backup-files; } | \ 
{ cd $1 && tar --preserve -v -x ; }

... or who’s eating my disk space?
find . -true -printf "%k %p\n" | sort -nr
```
The xargs utility

Show me 20 images at a time:

```
find . -name /*.jpg -print0 | \
  xargs --null -n 20 kview
```

To `xargs` white spaces in the input are delimiters, unless in quotes, or as above: `print0` and `--null`

The `-printf "\"%p\"\n"` is the filename within double quotes (what if the file name includes quotes?)

If we change the second line to

```
xargs --null -P 4 -n 20 kview
```
we get four instances (windows) of `kview`. Close one, another will pop up!

The inserted arguments don’t have to be last ones with `--replace=XXX`
String operations

- find . -name ".*.wav" | while read i ;
  do lame -h "$i" "${i%.*}.mp3" ; done

- Or more specific:
- find . -name ".*.wav" | while read i ;
  do lame -h "$i" "${i%.wav}.mp3" ; done

% and %% chop off suffixes. # and ## chop off prefixes.
%% and ## are greedy. % and # match minimal characters.
Remove path (file name only): ${i##*/}
Remove "./": ${i#/}
If no match is found, the string is left as is
My CD image generation script

... and another string expansion:
#!/bin/bash

for i in cd-* ; do
    item=${i:3:5};
    today=`date +%y%m%d`;
    echo Now creating volume $today$item...
    mkisofs -R -J -graft-points -V $today$item \
            -o $i.iso "/$i/";
done

- \${i:3:5} is character 3 to 5 (counting from zero) in $i.
- Later on we’ll see how Bash is used to burn the images...
The `printf` builtin command

- Of course there’s a `printf`!
- This is how we find a unique `dirXXXX` directory name:
  
  ```bash
  i=1; while name=`printf dir%04d $i` && [ -e $name ]
          do ((i++)); done;
  ```

- Note: No comma between format string and argument(s)
This simple script is for serial CD burning

```bash
for i in *.iso;
    do Xdialog --msgbox "Now burning $i" 0 0;
        cdrecord dev=0,0,0 speed=24 -v -eject -dao $i;
    done;
```

- **Xdialog** prompts the user with an “OK” message box

- **File selection (and then view):**
  ```bash
  Xdialog --stdout --fselect "" 0 0 | \ 
  { read i; kview "$i" ; }
  ```

- **Basically a front end for GTK**

- **The text-based version is** `dialog`

- **Several other widgets (edit boxes, progress meters, log boxes etc.)**
Functions

```
$ Hello() { echo I got $1 ; return 5 ; }
$ Hello World
I got World
$ echo $? 
5

- The function is run in the current environment
- No new process is created
```
The case statement

```bash
#!/bin/bash
case "$1" in
    [Hh]ello)
        echo "Nice to meet you"
        ;;
    [Bb]ye)
        echo "See you later"
        ;;
    *)
        echo "I am so glad to hear!"
esac

* The ;; is not a “break” statement. It’s syntactically necessary.
```
Service scripts

- Scripts can be found somewhere like `/etc/rc.d/init.d` (distribution dependent)
- The scripts are called during bootup according to the services setup
- ... or by `service xxx start`. Or stop. Or restart.
- The scripts are called with one argument, typically `start`, `stop`, `restart`, `status`, or other service-specific commands.
- Let’s see one!
Summary

We have seen:

- Loops and how to make meaningful loop indexes (file names...)
- Conditionals and arithmetics
- Backticking, `xargs` and while-read loops
- String operations
- Basic GUI
- We went to the safari (... service scripts)
- Bash is not Perl – it doesn’t cooperate
- ... but it’s still very useful
Further reading

- `man bash`
- Orna’s lecture about Bash:
  http://www.haifux.org/lectures/92-sil/
- Advanced Bash-Scripting Guide:
  http://tldp.org/LDP/abs/
- Linux Files and Command Reference:
  http://www.comptechdoc.org/os/linux(commands/
Thank you!

The slides were made with \LaTeX
(pr prosper\ class)