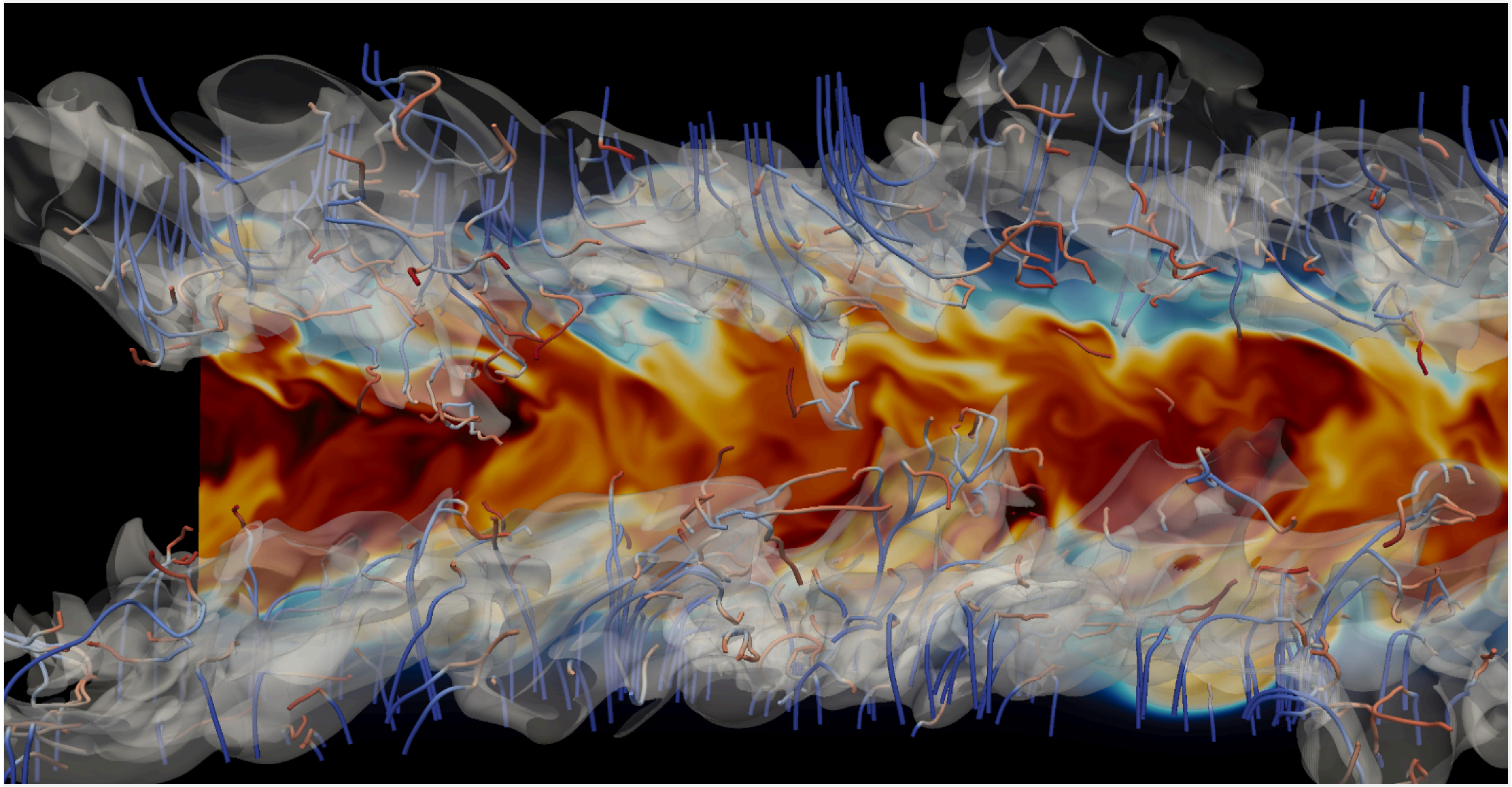
Software Tools for UNIX/Linux Systems

Part 6: Shell Scripting

C. Hasse

TECHNISCHE UNIVERSITÄT DARMSTADT

STFS



Agenda

Introduction

A *script* is a set of instructions. The computer executes the instructions, then returns a value
–
which could be a number, a string, a list, or another data type.

Frontier.userland.com/tutorial/whatIsAScript

Introduction: script vs. program

The line between both is blurred and not always a clear distinction can be drawn.

Generally, a *program* is preprocessed (*interpreted*) by a *compiler* allowing it to run fast and efficient during *runtime* – usually these compiled programming languages are more complicated to learn.

Scripting is programming inside a program. The *script* itself is interpreted and executed at runtime, rendering it slow. Scripting languages are commonly easy to learn because much of the abstraction is done for you.

Introduction

- Automate (your) reoccurring tasks
- Automate system tasks (e.g. boot, cron-jobs)
- Standardize workflow
- Simplify communication to "users"
- Documentation of certain procedure

Introduction

Best Practice Guide:

http://google-styleguide.googlecode.com/svn/trunk/shell.xml

Further Reading (highly recommended):

http://www.tldp.org/LDP/abs/html/

Disclaimer

Here, we will look at *bash* (*GNU Bourne-Again Shell*) syntax only, although many (all) presented concepts are easily adaptable for different interpreters

(sh, dash, csh, python, …)

Most of the things discussed on the next slides are directly available to the user:

```
$>   man 1 bash
$>   help <builtin>
```

Review: 02 - Basics

Q: How to define variables?

Q: Whats the difference between „“ and ‚‘

Q: How do you use
mkdir, mv, cp, cd, tr, cut, rev, tac, cat,
sort, uniq, paste, join, head, tail, wc, colrm
nl, split, tee

Q: BASH: set –x, set –o vi, shopt

Q: Use &, bg, fg, wait

Q: How to combine programs?

The start of all scripts

- SHEBANG:
- first line in every SHELL-script
  - also for python etc.
- What is SHEBANG?

```
#!
```

Basics

Create and edit file:

```
$>    cat example01.sh

#!/bin/bash

# this is a comment

# lets give some feedback...
echo "hello world"

# and return
exit 0
```

Save and run the code:

```
$>    ./example01.sh
```

What happens?

Variables

Variables are a named chunk of memory.

Variables can be used for temporary storage of runtime values and represent the state of a program or a script. They are valid only in the scope of their declaration.

Arguments are variables that are passed to other programs, scripts or functions.

An ***exit status*** or ***return value*** is reported back to the caller of a program, a script or a function.

Variables and arguments

```
$>  cat example02.sh

#!/bin/bash

LOCALVAR="hallo welt"
echo "LOCALVAR = $LOCALVAR"

echo "GLOBALVAR = $GLOBALVAR"

echo "FIRST ARGUMENT = $1"
exit 0

$>  chmod 744; ./example02.sh
```

What happens?

Special Variables

Special variables available in scripts: (refer-only!)

Variable	Meaning
$1…$9	Arguments 1…9 respectively
$@	All arguments seperated by SPACE
$#	Number of arguments
$?	Exit status of most recent foreground pipeline
$-	Current option flag of shell (refer to set)
$$	PID of calling shell
$!	PID of most recent background job
$0	Name of the script e.g. "./example01.sh"
$_	Last command visible to caller (in subshell)

Conditionals

```
$> cat example03.sh

#!/bin/bash

if [ "$1" ]; then
  echo "argument 1: $1"
else
  echo "no arguments"
fi
exit 0

#Q: What happens when you leave out the „“??
```

Conditionals

Primary	Meaning
[-d FILE]	True if FILE exists and is a directory.
[-f FILE]	True if FILE exists and is a regular file.
[-z STRING]	True of the length if "STRING" is zero.
[STRING]	True if the length of "STRING" is non-zero.
[STRING1 == STRING2]	True if the strings are equal.
[STRING1 != STRING2]	True if the strings are not equal.
[STRING1 < STRING2]	True if "STRING1" sorts before "STRING2“ lexicographically in the current locale.
[STRING1 > STRING2]	True if "STRING1" sorts after "STRING2“ lexicographically in the current locale.
[ARG1 OP ARG2]	"OP" is one of -eq, -ne, -lt, -le, -gt or -ge.

Conditionals

```
$> cat example03.sh

#!/bin/bash

if [ -f $0 ]; then
  echo "$0: i am here"
else
  echo "i am lost"
fi
exit 0
```

Conditionals

```
$>  cat example04.sh
#!/bin/bash

case $1 in
  1)
    echo "you entered 1"
    ;;
  h[ae]llo)
    echo "hello world"
    ;;
  '')
    echo "no argument"
    ;;
  *)
    echo "unknown argument $1"
    ;;
esac
exit 0
```

Loops

```
$>  cat example05.sh
    #!/bin/bash
    TESTFILE=example05.dat
    if [ -f $TESTFILE ]; then rm $TESTFILE; fi
    touch $TESTFILE

    for MYVAR in $@; do
      echo $MYVAR >> $TESTFILE
    done

    while read MYVAR; do
      echo $MYVAR
    done < $TESTFILE

    for MYVAR in {1..5}; do
      echo $MYVAR;
    done
    exit 0
```

Loops

```
$>    cat example06.sh
#!/bin/bash
COUNTER=0
until [ $COUNTER -gt 3 ]; do
  echo $((COUNTER++))
done
exit 0
```

Select

```
$>    cat example07.sh
#!/bin/bash

echo „Whats your favorite color?“
select MYSEL in red green blue; do
   echo $MYSEL;
   break;
done
exit 0
```

```
# Q: why break?
# Q: continue vs. break
```

Command Line Options

```
$>    cat example08.sh

#!/bin/bash
while getopts  "abc:def:ghi" flag; do
  echo "$flag $OPTIND $OPTARG"
done
exit 0
```

Command Line Options

```
$>    cat example09.sh

#!/bin/bash

while getopts  "abc:def:ghi" flag; do
  echo "$flag $OPTIND $OPTARG"
done

echo "resetting…"
OPTIND=1

while getopts  "abc:def:ghi" flag; do
  echo "$flag $OPTIND $OPTARG"
done
```

Substitution

```
$>    cat example10.sh

#!/bin/bash

# old way:

for FILE in `ls`; do
  echo $FILE
done
exit 0
```

Substitution

```
$>    cat example11.sh

#!/bin/bash

# better:

head -n1 $( \
   for FILE in $(ls); do \
      if [ -f $FILE ]; then \
         echo $FILE; \
      fi; \
   done )
exit 0
```

Function

```
$>    cat example12.sh

#!/bin/bash

function calc {
 echo "scale=4; $1" |  bc
}

RESULT=$(calc 2+2)
RESULT=$(calc $RESULT/12)

echo "RESULT = $RESULT"

exit 0
```

Spare me the details!

```
$>    cat example13.sh

#!/bin/bash

A=2
echo $((++A))
echo $A
let "A = 50 % 6"
echo ${A+5}
echo $(date | rev)
read -p "Whats your age? “
echo $REPLY
export A=2; echo $A; (echo $((++A)); echo $A);
echo $A
```

Spare me the details!

```
Brackets

if [ CONDITION ]    Test construct
if [[ CONDITION ]]  Extended test construct
Array[1]=element1   Array initialization
[a-z]               Range of characters within a Regular Expression

Curly Brackets

${variable}                             Parameter substitution
${!variable}                            Indirect variable reference
{ command1; command2; . . . commandN; } Block of code
{string1,string2,string3,...}           Brace expansion
{a..z}                                  Extended brace expansion
{}                                      Text replacement, after find and xargs

Parentheses

( command1; command2 )             Command group executed within a subshell
Array=(element1 element2 element3) Array initialization
result=$(COMMAND)                  Command substitution, new style
>(COMMAND)                         Process substitution
<(COMMAND)                         Process substitution

Double Parentheses

(( var = 78 ))            Integer arithmetic
var=$(( 20 + 5 ))         Integer arithmetic, with variable assignment
(( var++ ))               C-style variable increment
(( var-- ))               C-style variable decrement
(( var0 = var1<98?9:21 )) C-style trinary operation
```

Spare me the details!

```
BASH
BASHPID
$$
CDPATH
EDITOR
FUNCNAME
GROUPS
HOME
HOSTNAME
HOSTTYPE
MACHTYPE
IFS
IGNOREEOF
PWD
OLDPWD
OSTYPE
PATH
PIPESTATUS
$?
PS1..PS4
REPLY
SECONDS
SHELLOPTS
TMOUT
UID
RANDOM
```

Spare me the details!

```
shopt
shopt -s cdspell
echo $SHELLOPTS
set -x
echo $SHELLOPTS
export PS4="# "
export PS4="+ "
set +x
set -o vi
echo $SHELLOPTS
```

Review: 02 - Basics

Q: How to define variables?

Q: Whats the difference between „“ and ‚‘

Q: How do you use
mkdir, mv, cp, cd, tr, cut, rev, tac, cat,
sort, uniq, paste, join, head, tail, wc, colrm
nl, split, tee

Q: BASH: set –x, set –o vi, shopt

Q: Use &, bg, fg, wait, disown

Q: How to combine programs?

Q: check out nc

Real-world example

```
$>   view /etc/bash_completion.d/apt
$>   view /etc/init.d/ssh

### FILE: /root/scripts/startServer
#!/bin/bash

INITDIR=/etc/init.d/

# USAGE: startServer COMMON_NAME INIT_SCRIPT
#      COMMON_NAME as it appears in status notifications
#      INIT_SCRIPT as it is named in $INITDIR

function startServer
{
     echo "[INFO] Starting $1 license server..."
     $INITDIR/$2 start &&
      echo "[ OK ] $1 license server successfully started" ||
      echo "[FAIL] $1 license server failed to start!"
}

startServer PGI               pgi_lmgrd
startServer ToolWorks         toolworks
startServer MATLAB            matlab
startServer COMSOL            lm_comsol
startServer ALLINEA           allinea_licensing_init
```

What should you have learned?

What should you have learned today?

- What is the difference between script and program?
- What is SHEBANG?
- How to execute scripts?
- How to pass arguments?
- Which loops are available and used?
- What is a subshell?
- Which are the most common variables for input arguments?
- Where to find help?
- Which are the most common commands in BASH