EECS 2031 3.0 A
Software Tools

Week 9: November 6, 2018

BASH

- Written as a replacement for the Bourne shell (its the Bourne Again Shell) in the late 1980's
- One of the most popular 'command line interpreters' in the world
- Is extremely similar to 'sh' that it replaces, and on most UNIX machines /bin/sh is actually a pointer to /bin/bash.
- On Raspian, /bin/sh is a pointer to /bin/dash

Command Line Interpreter

- A (very common) mechanism of using a keyboard and a text window to interface with a computer.
- Basically you interact in terms of 'lines'
  - Each terminated by a carriage return
- The system processes your command, prints its output (if any) and then provides you with a prompt for you to enter the next command.

BASH

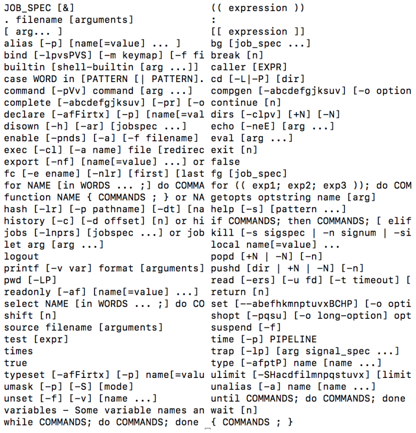
- Part of the GNU project
- Documented in fine detail online
  - https://www.gnu.org/software/bash/manual/
- As in many CLI's, the shell itself supports the ability to write 'script' files that allow for the automation of tasks within the shell.

BASH components

- Simple commands - ls, cat, …
- Pipelines - stringing the output of one command as the input to the next
- Lists - putting together sequences of commands
- Flow control - selection, iteration
- Advanced features (we will touch on these)

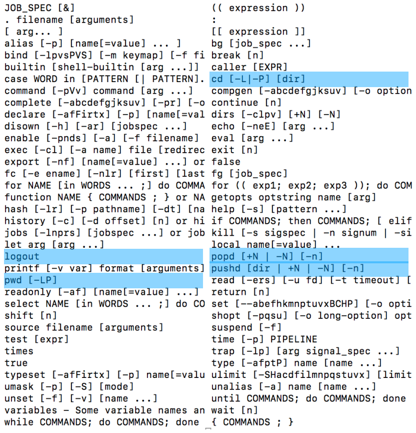
Simple commands

- Fall into two basic groups
  - Built in (executed within the shell itself)
  - External (separate programs that are run)
- Basic syntax is
  - command arg1 arg2 arg3 ….
  - In Unix flags are typically given to commands using dash command (e.g., -o foo.o)



```
JOB_SPEC [&]                       (( expression ))
. filename [arguments]             :
[ arg... ]                         [[ expression ]]
alias [-p] [name[=value] ... ]     bg [job_spec ...]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]]  caller [EXPR]
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...]   compgen [-abcdefgjksuv] [-o option
complete [-abcdefgjksuv] [-pr] [-o continue [n]
declare [-afFirtx] [-p] [name[=val dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...]    echo [-neE] [arg ...]
enable [-pnds] [-a] [-f filename]  eval [arg ...]
exec [-cl] [-a name] file [redirec exit [n]
export [-nf] [name[=value] ...] or false
fc [-e ename] [-nlr] [first] [last fg [job_spec]
for NAME [in WORDS ... ;] do COMMA for (( exp1; exp2; exp3 )); do COM
function NAME { COMMANDS ; } or NA getopts optstring name [arg]
hash [-lr] [-p pathname] [-dt] [na help [-s] [pattern ...]
history [-c] [-d offset] [n] or hi if COMMANDS; then COMMANDS; [ elif
jobs [-lnprs] [jobspec ...] or job kill [-s sigspec | -n signum | -si
let arg [arg ...]                  local name[=value] ...
logout                             popd [+N | -N] [-n]
printf [-v var] format [arguments] pushd [dir | +N | -N] [-n]
pwd [-LP]                          read [-ers] [-u fd] [-t timeout] [
readonly [-af] [name[=value] ...]  return [n]
select NAME [in WORDS ... ;] do CO set [--abefhkmnptuvxBCHP] [-o opti
shift [n]                          shopt [-pqsu] [-o long-option] opt
source filename [arguments]        suspend [-f]
test [expr]                        time [-p] PIPELINE
times                              trap [-lp] [arg signal_spec ...]
true                               type [-afptP] name [name ...]
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvx] [limit
umask [-p] [-S] [mode]             unalias [-a] name [name ...]
unset [-f] [-v] [name ...]         until COMMANDS; do COMMANDS; done
variables - Some variable names an wait [n]
while COMMANDS; do COMMANDS; done  { COMMANDS ; }
```

Bash built in commands (type help)

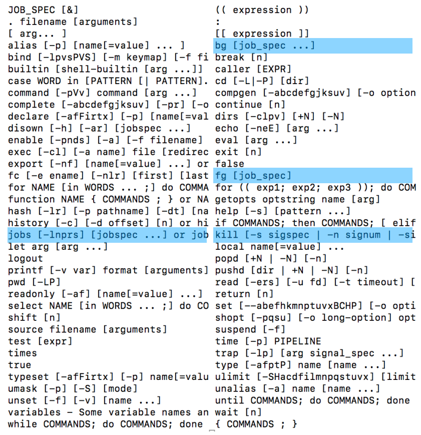


```
JOB_SPEC [&]                       (( expression ))
. filename [arguments]             :
[ arg... ]                         [[ expression ]]
alias [-p] [name[=value] ... ]     bg [job_spec ...]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]]  caller [EXPR]
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...]   compgen [-abcdefgjksuv] [-o option
complete [-abcdefgjksuv] [-pr] [-o continue [n]
declare [-afFirtx] [-p] [name[=val dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...]    echo [-neE] [arg ...]
enable [-pnds] [-a] [-f filename]  eval [arg ...]
exec [-cl] [-a name] file [redirec exit [n]
export [-nf] [name[=value] ...] or false
fc [-e ename] [-nlr] [first] [last fg [job_spec]
for NAME [in WORDS ... ;] do COMMA for (( exp1; exp2; exp3 )); do COM
function NAME { COMMANDS ; } or NA getopts optstring name [arg]
hash [-lr] [-p pathname] [-dt] [na help [-s] [pattern ...]
history [-c] [-d offset] [n] or hi if COMMANDS; then COMMANDS; [ elif
jobs [-lnprs] [jobspec ...] or job kill [-s sigspec | -n signum | -si
let arg [arg ...]                  local name[=value] ...
logout                             popd [+N | -N] [-n]
printf [-v var] format [arguments] pushd [dir | +N | -N] [-n]
pwd [-LP]                          read [-ers] [-u fd] [-t timeout] [
readonly [-af] [name[=value] ...]  return [n]
select NAME [in WORDS ... ;] do CO set [--abefhkmnptuvxBCHP] [-o opti
shift [n]                          shopt [-pqsu] [-o long-option] opt
source filename [arguments]        suspend [-f]
test [expr]                        time [-p] PIPELINE
times                              trap [-lp] [arg signal_spec ...]
true                               type [-afptP] name [name ...]
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvx] [limit
umask [-p] [-S] [mode]             unalias [-a] name [name ...]
unset [-f] [-v] [name ...]         until COMMANDS; do COMMANDS; done
variables - Some variable names an wait [n]
while COMMANDS; do COMMANDS; done  { COMMANDS ; }
```

Ones you have probably been using since day 1

Managing your tasks

- If you have always waited for your current task to complete, great. However, bash supports having multiple tasks (jobs) running within the same terminal.
- task & - runs the job in the background
- ^Z - suspends the current job
- jobs - lists current jobs
- fg %n - brings job n to the foreground (has control of the terminal)
- bg %n - runs job n in the background
- kill %n - kills job n



```
JOB_SPEC [&]                       (( expression ))
. filename [arguments]             :
[ arg... ]                         [[ expression ]]
alias [-p] [name[=value] ... ]     bg [job_spec ...]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]]  caller [EXPR]
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...]   compgen [-abcdefgjksuv] [-o option
complete [-abcdefgjksuv] [-pr] [-o continue [n]
declare [-afFirtx] [-p] [name[=val dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...]    echo [-neE] [arg ...]
enable [-pnds] [-a] [-f filename]  eval [arg ...]
exec [-cl] [-a name] file [redirec exit [n]
export [-nf] [name[=value] ...] or false
fc [-e ename] [-nlr] [first] [last fg [job_spec]
for NAME [in WORDS ... ;] do COMMA for (( exp1; exp2; exp3 )); do COM
function NAME { COMMANDS ; } or NA getopts optstring name [arg]
hash [-lr] [-p pathname] [-dt] [na help [-s] [pattern ...]
history [-c] [-d offset] [n] or hi if COMMANDS; then COMMANDS; [ elif
jobs [-lnprs] [jobspec ...] or job kill [-s sigspec | -n signum | -si
let arg [arg ...]                  local name[=value] ...
logout                             popd [+N | -N] [-n]
printf [-v var] format [arguments] pushd [dir | +N | -N] [-n]
pwd [-LP]                          read [-ers] [-u fd] [-t timeout] [
readonly [-af] [name[=value] ...]  return [n]
select NAME [in WORDS ... ;] do CO set [--abefhkmnptuvxBCHP] [-o opti
shift [n]                          shopt [-pqsu] [-o long-option] opt
source filename [arguments]        suspend [-f]
test [expr]                        time [-p] PIPELINE
times                              trap [-lp] [arg signal_spec ...]
true                               type [-afptP] name [name ...]
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvx] [limit
umask [-p] [-S] [mode]             unalias [-a] name [name ...]
unset [-f] [-v] [name ...]         until COMMANDS; do COMMANDS; done
variables - Some variable names an wait [n]
while COMMANDS; do COMMANDS; done  { COMMANDS ; }
```

Built in operations to manipulate tasks

Bash terminal

- Bash's command line is sophisticated, it has editing capabilities, a history, command completion, etc.
- history - lists your history
- Each command has a number, to re-execute it type !n
- Up and down arrow keys let you walk through the history
- Left/right arrow keys let you move through the currently selected history element (and edit)
- Hit return to execute the command
- The arrows are vi-like, emacs-like motion works too (^a,^e,^p,^n) - can be a problem with screen

Variables

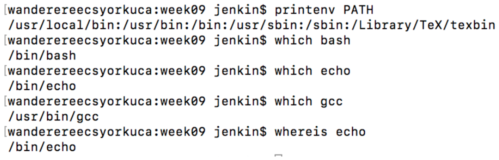
- The shell supports variables, two types 'environment variables' and 'shell variables'
  - Many commands use 'well known environment variables' to control their action.
- printenv - prints all environment variables
- set - prints all variables

PATH

- An environment variable
  - When you type a command junk -o foo -x bar
  - BASH first checks to see if its a built in command
    - If so, executes it
  - It then searches your path for junk that is executable by you
    - If found, executes it
    - If not found, prints an error
  - If you put a slash in the command name, then BASH just looks for the file directly
- Note: bash actually maintains a table of all executable programs to avoid having to search through this list often.

Which command

- which echo - which echo will be run
- whereis echo - path for the echo that will be run



```
[wanderereecsyorkuca:week09 jenkin$ printenv PATH
 /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin
[wanderereecsyorkuca:week09 jenkin$ which bash
 /bin/bash
[wanderereecsyorkuca:week09 jenkin$ which echo
 /bin/echo
[wanderereecsyorkuca:week09 jenkin$ which gcc
 /usr/bin/gcc
[wanderereecsyorkuca:week09 jenkin$ whereis echo
 /bin/echo
```

Top commands

- tar - manipulates an archive (like zip)
- grep - search through a file for records
- find - search the file system for a file
- ssh - secure shell login to remote system
- sed - stream editor
- awk - run the awk command
- vi - run the vi (vim) editor
- diff - find differences in two files
- sort - sorts a file
- export - export an environment variable
- ls - list files
- pwd - print working directory
- cd - change directory

From GeekStuff

Variables

- x=2
  - Note: no spaces. None
- Want to know its value use set or
  - echo “$x” or echo $x (not echo ‘$x’)
- Variables are untyped
- Can set to null (x=)
- let command lets you manipulate variable values (but there are other ways, often better ones)
  - let “x=2+3+4”
  - echo “$x”

Your environment

- When bash starts up it looks in certain places for files that it executes to 'customize' your environment (basically set certain variables and run commands on startup).
- These include
  - /etc/profile
  - ~/.bashrc
- You can basically make your login unusable if you mess these up. On prism, the default tries to make you very safe.

On the Pi

- Who you are is defined in /etc/passwd (standard)

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
pi:x:1000:1000:,,,:/home/pi:/bin/bash
messagebus:x:105:109::/var/run/dbus:/bin/false
statd:x:106:65534::/var/lib/nfs:/bin/false
sshd:x:107:65534::/run/sshd:/usr/sbin/nologin
avahi:x:108:112:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
lightdm:x:109:113:Light Display Manager:/var/lib/lightdm:/bin/false
epmd:x:110:114::/var/run/epmd:/bin/false
```

Your uid=1000,gid=1000 home is /home/pi and your shell is /bin/bash

/bin/bash

```
When bash is invoked as an interactive login shell, or as a  non-inter-
active  shell with the --login option, it first reads and executes com-
mands from the file /etc/profile, if that file exists.   After  reading
that file, it looks for ~/.bash_profile, ~/.bash_login, and ~/.profile,
in that order, and reads and executes commands from the first one  that
exists  and  is  readable.  The --noprofile option may be used when the
shell is started to inhibit this behavior.
```

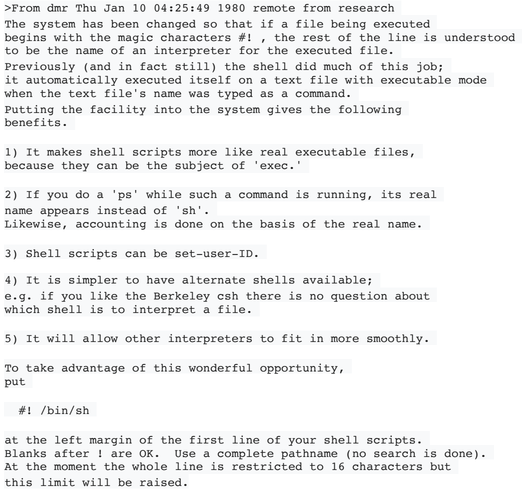
From 'man bash'

/etc/profile

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "`id -u`" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/
games:/usr/games"
fi
export PATH

if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
```

Shell script

- File of shell commands
  - Anything you can do in a script you can do on the console.
- Start with
  - #!/bin/bash
  - #! is known as a hash bang or shebang line.



```
>From dmr Thu Jan 10 04:25:49 1980 remote from research
The system has been changed so that if a file being executed
begins with the magic characters #! , the rest of the line is understood
to be the name of an interpreter for the executed file.
Previously (and in fact still) the shell did much of this job;
it automatically executed itself on a text file with executable mode
when the text file's name was typed as a command.
Putting the facility into the system gives the following
benefits.

1) It makes shell scripts more like real executable files,
because they can be the subject of 'exec.'

2) If you do a 'ps' while such a command is running, its real
name appears instead of 'sh'.
Likewise, accounting is done on the basis of the real name.

3) Shell scripts can be set-user-ID.

4) It is simpler to have alternate shells available;
e.g. if you like the Berkeley csh there is no question about
which shell is to interpret a file.

5) It will allow other interpreters to fit in more smoothly.

To take advantage of this wonderful opportunity,
put

  #! /bin/sh

at the left margin of the first line of your shell scripts.
Blanks after ! are OK.  Use a complete pathname (no search is done).
At the moment the whole line is restricted to 16 characters but
this limit will be raised.
```

Dennis Ritchie's email that made this so

Hello World

- Must have the x bit set in order to run it
  - chmod 755 hello.sh
- ./hello.sh

```
#!/bin/bash
echo "Hello World"
```

```
#!/bin/bash
clear
echo "Hello $USER"
echo
echo "Who is logged in?"
who
echo -n "It is now "
date
```

```
Hello jenkin

Who is logged in?
_mbsetupuser console  Oct 31 23:01
jenkin       console  Oct 31 23:01
jenkin       ttys000  Nov  6 18:46
It is now Tue  7 Nov 2017 00:08:17 EST
wanderereecsyorkuca:~ jenkin$
```

Bash commands

- Any command you can type on the terminal
- Comments start with ‘#’ and continue to the end of the line
- Commands are separated by new lines or by ;
- Indentation does not matter, but spaces do in certain circumstances.

if

- If [<test>] ; then <command> fi

```
#!/bin/bash
if [ $USER == 'jenkin' ]
then
  echo "You are jenkin"
fi
```

```
#!/bin/bash
if [ $USER == 'jenkin' ] ; then
  echo "You are jenkin"
fi
```

If else

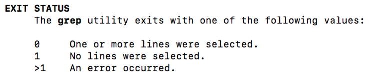
```
#!/bin/bash
if [ $USER == 'root' ] ; then
  echo "You are root"
else
  echo "You are not root"
fi
```

Testing

- The ‘condition’ is actual the ‘exit status’ of a command
  - If the command returns 0 then the status is ‘true’ otherwise ‘false’
  - Think of this as the return value from a C program
    - 0 == success

```
#!/bin/bash
if false; then
  echo "true"
else
  echo "false"
fi
```

```
#!/bin/bash
if grep -q “main” test.c; then
  echo "there was a main in test.c"
else
  echo "no main in test.c"
fi
```



```
EXIT STATUS
     The grep utility exits with one of the following values:

     0     One or more lines were selected.
     1     No lines were selected.
     >1    An error occurred.
```

Test

- [] or test is a command that ‘tests’ some property
  - man test
- test -f foo.c
  - Tests if foo.c is a file that exists
  - $? is the last exit status

Test

- test -f test.c; echo $?
- test -f testxxx.c; echo $?

```
sh-3.2$ test -f testx.c; echo $?
1
sh-3.2$ test -f test.c; echo $?
0
sh-3.2$ test -f textxxx.c; echo $?
1
```

```
#!/bin/bash
if test -f test.c ; then
  echo "test.c exists"
else
  echo "test.c does not exist"
fi
```

Blanks & quotes

- So many characters are valid symbols in commands, so blanks can be problematic
  - Extra blanks are good when you control things
- Remember that things like file names, variables can contain things like blanks or symbols that mean things to the shell.
  - Extra blanks can be bad when you do not

Test and [ ]

- [ ] is a ‘short form’ for ‘test’
- () executes the inner contents in a sub-shell
  - Limits side effects of the inner contents (more on this later)

Test

```
sh-3.2$ test "h" \>  "a"; echo $?
0
sh-3.2$ test "h" \<  "a"; echo $?
1
```

- Huge number of options. A few observations
  - <, > are special symbols in the shell and must be escaped \<, \>
  - test uses different operators for strings and ints

```
sh-3.2$ test 2 \> 3; echo $?
1
sh-3.2$ test 2 \> 03; echo $?
0
```

```
sh-3.2$ test 2 -gt  3; echo $?
1
sh-3.2$ test 2 -gt  03; echo $?
1
```

Test (more observations)

- test is just a command
  - test “a”=“b” != test “a” = “b”
  - test “a” = “b” == test “a” == “b”

```
sh-3.2$ test "a" = "a"; echo $?
0
sh-3.2$ test "a" == "a"; echo $?
0
sh-3.2$ test "a" == "b"; echo $?
1
sh-3.2$ test "a" = "b"; echo $?
1
sh-3.2$ test "a"="b"; echo $?
0
sh-3.2$ test "a"="a"; echo $?
0
```

Let

- There are many ways of doing arithmetic expressions in bash.
- Mechanism #1 ‘let’

```
sh-3.2$ let z=3+5
sh-3.2$ echo $z
8
sh-3.2$ let z = 3 + 5
sh: let: =: syntax error: operand expected (error token is "=")
sh-3.2$ let "z=3+5"
sh-3.2$ echo $z
8
sh-3.2$ let "z = 3   +    5"
sh-3.2$ echo $z
8
sh-3.2$ let "z = z+6"
sh-3.2$ echo $z
14
```

Special variables (again)

```
#!/bin/bash           ./foo.sh all the world is a stage "and the"
echo "$ $" $#         $ $ 7
echo "$ $" $$         $ $ 6144
echo "$ ?" $?         $ ? 0
echo "$ 0 " $0        $ 0  ./foo.sh
echo "$ 1 " $1        $ 1  all
echo "$ 2 " $2        $ 2  the
echo "$ 9 " $9        $ 9
echo "$ @ " $@        $ @  all the world is a stage and the
```

Back tick (quote)

- In Bash, if you execute a command in \`ls\` then the output of the command is returned.
- You can use $(ls) as well.

A real example

- Am I logged in more than once?

```
sh-3.2$ who | grep pi | wc -l
       2
```

```
sh-3.2$ cat foo.sh
#!/bin/bash
z=$(who | grep pi | wc -l)
echo $z
sh-3.2$ ./foo.sh
2
```

```
#!/bin/bash
z=$(who | grep pi | wc -l)
if test "$z" -gt 1
then
  echo "You are logged in more than once"
elif test "$z" -eq 1
then
  echo "You are logged in only one time"
else
  echo "You are not logged in at all?????"
fi
exit 0
sh-3.2$ ./foo.sh
You are logged in more than once
sh-3.2$
```

A real example

- Let us not hard code ‘jenkin’, figure it out

```
sh-3.2$ who am i
jenkin       ttys000  Nov  6 18:46
sh-3.2$
```

Use ‘sed’ stream editor to delete everything from the first blank to the end

```
sh-3.2$ who am i | sed 's/ .*$//'
jenkin
sh-3.2$
```

```
#!/bin/bash
user=$(who am i|sed 's/ .*$//')
z=$(who | grep "$user" | wc -l)
if test "$z" -gt 1
then
  echo "$user is logged in more than once"
elif test "$z" -eq 1
then
  echo "$user is logged in only one time"
else
  echo "$user is not logged in at all?????"
fi
exit 0
sh-3.2$ ./foo.sh
Pi is logged in more than once
sh-3.2$
```

Let the user specify the user to check

```
#!/bin/bash
if test $# -eq 1 ; then
  user=$1
elif test $# -eq 0 ; then
  user=$(who am i|sed 's/ .*$//')
else
  echo "Usage $0: [user]"
  exit 1
fi
z=$(who | grep "$user" | wc -l)
if test "$z" -gt 1
then
  echo "$user is logged in more than once"
elif test "$z" -eq 1
then
  echo "$user is logged in only one time"
else
  echo "$user is not logged in at all"
fi
exit 0
```

```
sh-3.2$ ./foo.sh
jenkin is logged in more than once
sh-3.2$ ./foo.sh mary
mary is not logged in at all
sh-3.2$ who
_mbsetupuser console  Oct 31 23:01
jenkin       console  Oct 31 23:01
jenkin       ttys000  Nov  6 18:46
sh-3.2$ ./foo.sh _mbsetupuser
_mbsetupuser is logged in only one time
sh-3.2$ ./foo.sh a b c
Usage ./foo.sh: [user]
sh-3.2$
```