Kernighan & Pike, *The UNIX Programming Environment*
A Roadmap for Chapter 5

Section	Scripts	Ideas
1	cal	case, set, sh builtins, case's patterns
2	which	test, exit status, $?, if-then, ||, &&
3	watchwho checkmail watchfor	while, until, piping loops, conditional variables (also called "special substitutions")
4		traps and interrupts
5	overwrite, replace	shift, "$@"
6	zap	IFS
7	pick	read, "$*" vs "$@"
8	news	touch
9	get, put	diff -e

```
# cal cal
# cal:  nicer interface to /usr/bin/cal

case $# in
0)      set `date`; m=$2; y=$6 ;;  # no args: use today
1)      m=$1; set `date`; y=$6 ;;  # 1 arg: use this year
*)      m=$1; y=$2 ;;              # 2 args: month and year
esac

case $m in
jan*|Jan*)      m=1 ;;
feb*|Feb*)      m=2 ;;
mar*|Mar*)      m=3 ;;
apr*|Apr*)      m=4 ;;
may*|May*)      m=5 ;;
jun*|Jun*)      m=6 ;;
jul*|Jul*)      m=7 ;;
aug*|Aug*)      m=8 ;;
sep*|Sep*)      m=9 ;;
oct*|Oct*)      m=10 ;;
nov*|Nov*)      m=11 ;;
dec*|Dec*)      m=12 ;;
[1-9]|10|11|12) ;;                 # numeric month
*)              y=$m; m="" ;;      # plain year
esac

/usr/bin/cal $m $y                 # run the real one
$
```

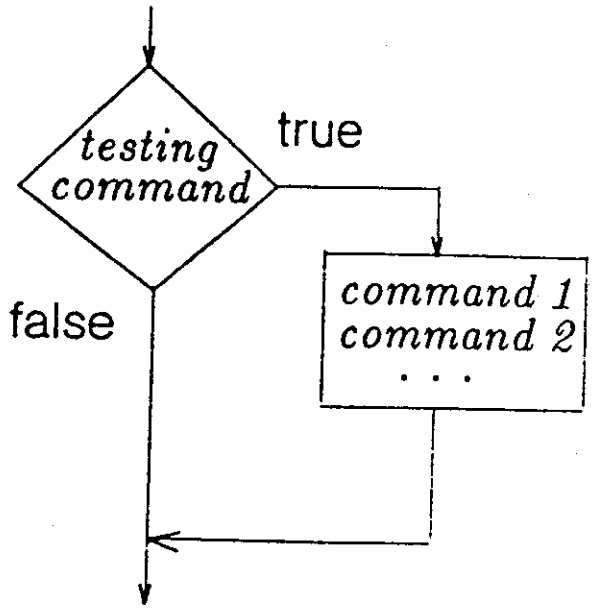
```
$ cat which
# which cmd:  which cmd in PATH is executed, version 1

case $# in
0)      echo 'Usage: which command' 1>&2; exit 2
esac
for i in `echo $PATH | sed 's/^:/.:/
                            s/::/:.:/g
                            s/:$/:./
                            s/:/ /g'`
do
        if test -f $i/$1        # use test -x if you can
        then
                echo $i/$1
                exit 0          # found it
        fi
done
exit 1          # not found
$
```

```
$ cat which
# which cmd:  which cmd in PATH is executed, final version

opath=$PATH
PATH=/bin:/usr/bin

case $# in
0)      echo 'Usage: which command' 1>&2; exit 2
esac
for i in `echo $opath | sed 's/^:/.:/
                             s/::/:.:/g
                             s/:$/:./
                             s/:/ /g'`
do
        if test -f $i/$1        # this is /bin/test
        then                    # or /usr/bin/test only
                echo $i/$1
                exit 0          # found it
        fi
done
exit 1          # not found
$
```

CONDITIONAL CONSTRUCT
if-then

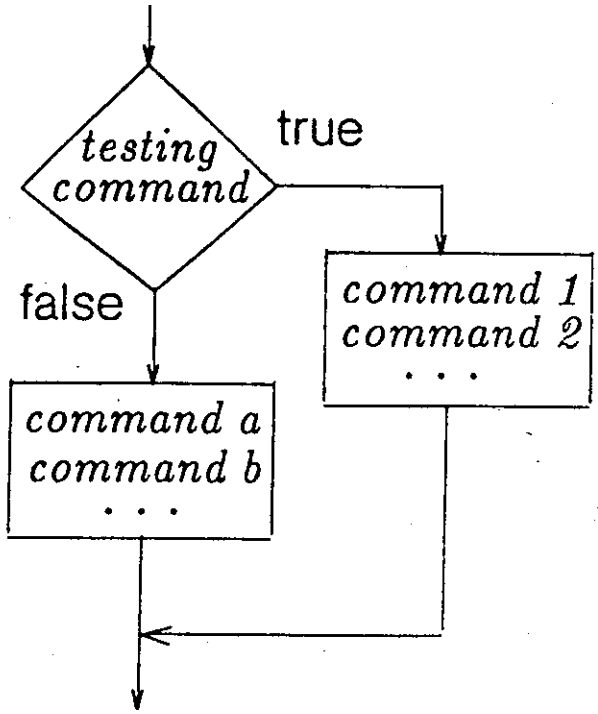
- Used to test a condition
- Changes the program flow based on the test result

```
if
        testing command
then
        command 1
        command 2
           . . .
fi
```



CONDITIONAL CONSTRUCT
if-then-else

```
if
        testing command
then
        command 1
        command 2
            . . .
else
        command a
        command b
            . . .
fi
```



test STATEMENT

- CHECKS FILE STATUS
- COMPARES STRINGS
- COMPARES NUMERIC VALUES CONTAINED IN STRINGS
- RETURNS 0 IF TEST CONDITION IS TRUE
- RETURNS 1 IF TEST CONDITION IS FALSE
- USED WITH CONSTRUCTS
- ARGUMENTS TO test DESCRIBE CONDITION BEING TESTED

test STATEMENT — FILE STATUS EVALUATION

- TESTS REQUESTER'S PERMISSIONS AGAINST THOSE OF THE SPECIFIED FILE
- FILE NAME CAN COME FROM CAPABILITY OF THE SHELL THAT GENERATES CHARACTER STRINGS OR FROM THE COMMAND LINE
- FILE NAME CANNOT BE OMITTED
- IF FILE NAME IS GENERATED BY SUBSTITUTION REQUEST, ENCLOSE IN DOUBLE QUOTES
- FILE STATUS CONDITIONS

```
test -r file_name  # exists and readable
test -w file_name  # exists and writable
test -x file_name  # exists and executable
test -s file_name  # exists and nonzero size
test -f file_name  # exists and ordinary
test -d file_name  # exists and directory
```

THE test COMMAND

- Performs string comparisons
- Performs numeric comparisons
- Determines file attributes

contents of **nprint1**

```
echo 'Enter the file you want to print?'
read name
echo 'Print file with line numbers?'
read ans
if
  test "$ans" = y
then
  pr -n -t $name
fi
if
  test "$ans" = n
then
  cat $name
fi
```

test STATEMENT — STRING COMPARISON

- TWO STRING COMPARISON TESTS
  - — EQUALITY (=)
  - — INEQUALITY (!=)
- OPERATORS MUST BE SURROUNDED WITH SPACE

EXAMPLE

```
$ echo ${TERM}<CR>
2645

$ test "${TERM}" = "2645"<CR>
$ echo ${?}<CR>
0

$ test "${TERM}" != "2645"<CR>
$ echo ${?}<CR>
1
```

test STATEMENT — STRING COMPARISON

EXAMPLE (Cont'd.)

- ARGUMENTS TO test SHOULD USUALLY BE QUOTED
- IF NOT QUOTED, AND VALUE OF STRING IS NULL, STRING IS REMOVED FROM COMMAND LINE BY IFS PROCESSING

EXAMPLE

```
$ echo ${NOTSET}<CR>


$ test ${NOTSET} != "hello"<CR>
test: argument expected

$ test "${NOTSET}" != "hello"<CR>

$ echo ${?}<CR>
0
```

test STATEMENT — STRING SIZE TESTING

- OPERATORS:
  - — LENGTH OF STRING ZERO (**-z**)
  - — LENGTH OF STRING NONZERO (**-n**)
  - — IF NO OPTION, IMPLIES **-n**

FORMAT

```
test -z "string"

test -n "string"

test "string"

test "-d file_name -a -f file_name"
```

test STATEMENT — STRING SIZE TESTING

EXAMPLE

```
$ cat test.args<CR>
test -z "${1}"
echo ${?}

$ test.args hello there<CR>
1

$ test.args<CR>
0
```

test STATEMENT — NUMERIC COMPARISON (Cont'd.)

FORMAT

```
test "string1" -eq "string2"

test "string1" -ne "string2"

test "string1" -gt "string2"

test "string1" -ge "string2"

test "string1" -lt "string2"

test "string1" -le "string2"
```

- STRINGS ON EACH SIDE SHOULD CONTAIN NUMERIC INFORMATION
- OPERATORS MUST BE SURROUNDED WITH SPACES

```
$ cat watchwho
# watchwho:  watch who logs in and out

PATH=/bin:/usr/bin
new=/tmp/wwho1.$$
old=/tmp/wwho2.$$
>$old              # create an empty file

while :            # loop forever
do
        who >$new
        diff $old $new
        mv $new $old
        sleep 60
done | awk '/>/ { $1 = "in:      "; print }
            /</ { $1 = "out:     "; print }'
$
```

```
$ cat checkmail
# checkmail:

PATH=/bin:/usr/bin
MAIL=/usr/spool/mail/`getname`   # system dependent

t=${1-60}

x="`ls -l $MAIL`"
while :
do
        y="`ls -l $MAIL`"
        echo $x $y
        x="$y"
        sleep $t
done | awk '$4 < $12 { print "You have mail" }'
$
```

Table 5.3: Evaluation of Shell Variables

`$var`	value of `var`; nothing if `var` undefined
`${var}`	same; useful if alphanumerics follow variable name
`${var-thing}`	value of `var` if defined; otherwise `thing`. `$var` unchanged.
`${var=thing}`	value of `var` if defined; otherwise `thing`. If undefined, `$var` set to `thing`
`${var?message}`	if defined, `$var`. Otherwise, print `message` and exit shell. If `message` empty, print: `var: parameter not set`
`${var+thing}`	`thing` if `$var` defined, otherwise nothing

```
$ cat watchfor
# watchfor:  watch for someone to log in

PATH=/bin:/usr/bin

case $# in
0)      echo 'Usage: watchfor person' 1>&2; exit 1
esac

until who | egrep "$1"
do
        sleep 60
done
$ cx watchfor
$ watchfor you
you      tty0     Oct  1 08:01        It works
$ mv watchfor /usr/you/bin             Install it
$
```

changed grep to egrep so you can type

```
$ watchfor 'joe|mary'
```

Table 5.4: Shell Signal Numbers

0	shell exit (for any reason, including end of file)
1	hangup
2	interrupt (DEL key)
3	quit (*ctl*-\; causes program to produce core dump)
9	kill (cannot be caught or ignored)
15	terminate, default signal generated by `kill`(1)

SIGNALS

- EXTERNAL CONDITIONS THAT MAY AFFECT THE EXECUTION OF A RUNNING PROGRAM
- MAY COME FROM
  - THE USER'S TERMINAL (DELETE KEY)
  - THE OPERATING SYSTEM
  - OTHER SOFTWARE (VIA THE kill COMMAND)
- NORMALLY TERMINATES PROCESS UNLESS "TRAPPED"
- SIGNALS MAY BE
  - CAUGHT
  - IGNORED
  - RESET

trap STATEMENT

trap ["commands"] [signal...]

EXAMPLE

```
trap 'echo "Interrupted routine" ;
          rm tmp$$ ; exit 4'  1 2 3
```

- IF SPECIFIED SIGNALS ARE RECEIVED
  - SPECIFIED COMMAND EXECUTES
  - CONTROL PASSES TO THE COMMAND FOLLOWING THE COMMAND THAT WAS INTERRUPTED UNLESS THE TRAP EXECUTED THE exit STATEMENT
- IF NO COMMAND SPECIFIED OR EXPLICITLY NULL, THEN

```
trap '' 2 3        # IGNORE
trap ':' 2 3       # DO NOTHING
```

IGNORING VS. THE NULL STATEMENT

- IN CHILD PROCESSES, IGNORED SIGNALS ARE IGNORED
- IN CHILD PROCESSES, ALL TRAP SETTINGS ARE RESET IN CHILD PROCESSES
- EXAMPLES

  trap "" 2 ignore intr and tell children to do same

  trap ":" 2 execute : statement but do not modify children's response to signal

RESETTING TRAPS

- IF NO COMMAND SPECIFIED, WILL RESET SETTINGS FOR SPECIFIED SIGNALS
- THE COMMAND LINE

  trap 1 2 3 removes previous trap settings for these signals

- TRAP WITHOUT ANY ARGUMENTS LISTS CURRENT TRAP SETTINGS

trap STATEMENT – QUOTING CONSIDERATIONS

- COMMANDS WITHIN trap ARE SCANNED TWICE
  - ONCE WHEN READ
  - AGAIN WHEN EXECUTED
- USING DOUBLE QUOTES
  - ALL VARIABLE AND COMMAND SUBSTITUTION EXPRESSIONS ARE EVALUATED WHEN THE SHELL FIRST READS THE SIGNAL PROCESSING ROUTINE
- USING ~~SIGNAL~~ SINGLE QUOTES
  - PREVENTS THE SHELL FROM INITIALLY DOING ANY SUBSTITUTIONS WHEN THE COMMAND LINE IS READ

CHILD INHERITANCE

```
$ trap "echo process ending" 0<CR>
$ trap "" 16<CR>
$ trap ":" 17<CR>
$ trap<CR>
0: echo process ending
16:
17: :
$ (trap)<CR>
16:
```

```
$ sed 's/UNIX/UNIX(TM)/g' ch2 | overwrite ch2
```

```
# overwrite:  copy standard input to output after EOF
# version 1.  BUG here

PATH=/bin:/usr/bin

case $# in
1)      ;;
*)      echo 'Usage: overwrite file' 1>&2; exit 2
esac

new=/tmp/overwr.$$
trap 'rm -f $new; exit 1' 1 2 15

cat >$new                   # collect the input
cp $new $1                  # overwrite the input file
rm -f $new
```

```
# overwrite:  copy standard input to output after EOF
# version 2.  BUG here too

PATH=/bin:/usr/bin

case $# in
1)      ;;
*)      echo 'Usage: overwrite file' 1>&2; exit 2
esac

new=/tmp/overwr1.$$
old=/tmp/overwr2.$$
trap 'rm -f $new $old; exit 1' 1 2 15

cat >$new                   # collect the input
cp $1 $old                  # save original file

trap '' 1 2 15              # we are committed; ignore signals
cp $new $1                  # overwrite the input file

rm -f $new $old
```

```
# overwrite:  copy standard input to output after EOF
# final version

opath=$PATH
PATH=/bin:/usr/bin

case $# in
0|1)    echo 'Usage: overwrite file cmd [args]' 1>&2; exit 2
esac

file=$1; shift
new=/tmp/overwr1.$$; old=/tmp/overwr2.$$
trap 'rm -f $new $old; exit 1' 1 2 15

if PATH=$opath "$@" >$new
then
        cp $file $old
        trap '' 1 2 15
        cp $new $file
else
        echo "overwrite: $1 failed, $file unchanged" 1>&2
        exit 1
fi
rm -f $new $old
```

```
$ cat replace
# replace:  replace str1 in files with str2, in place

PATH=/bin:/usr/bin

case $# in
0|1|2)  echo 'Usage: replace str1 str2 files' 1>&2; exit 1
esac

left="$1"; right="$2"; shift; shift

for i
do
        overwrite $i sed "s@$left@$right@g" $i
done
$ cat footnote
UNIX is not an acronym
$ replace UNIX Unix footnote
$ cat footnote
Unix is not an acronym
$
```

```
$ cat zap
# zap pattern:  kill all processes matching pattern
# BUG in this version

PATH=/bin:/usr/bin

case $# in
0)      echo 'Usage: zap pattern' 1>&2; exit 1
esac

kill `pick \`ps -ag | grep "$*"\` | awk '{print $1}'`
$
```

Note the nested backquotes, protected by backslashes. The `awk` program selects the process-id from the `ps` output selected by the `pick`:

```
$ sleep 1000 &
22126
$ ps -ag
   PID TTY TIME CMD
...
 22126 0   0:00 sleep 1000
...
$ zap sleep
22126?
0? q                          What's going on?
$
```

```
$ echo 'echo $#' >nargs
$ cx nargs
$ who
you      tty0    Oct  1 05:59
pjw      tty2    Oct  1 11:26
$ nargs `who`
10                                  Ten blank and newline-separated fields
$ IFS='
'                                   Just a newline
$ nargs `who`
2                                   Two lines, two fields
$
```

With `IFS` set to newline, `zap` works fine:

```
$ cat zap
# zap pat:  kill all processes matching pat
# final version

PATH=/bin:/usr/bin
IFS='
'                           # just a newline
case $1 in
"")     echo 'Usage: zap [-2] pattern' 1>&2; exit 1 ;;
-*)     SIG=$1; shift
esac

echo '   PID TTY   TIME CMD'
kill $SIG `pick \`ps -ag | egrep "$*"\` | awk '{print $1}'`
$ ps -ag
   PID TTY TIME CMD
...
 22126 0   0:00 sleep 1000
...
$ zap sleep
   PID TTY TIME CMD
 22126 0   0:00 sleep 1000? y
 23104 0   0:02 egrep sleep? n
$
```

```
# pick:  select arguments

PATH=/bin:/usr/bin

for i                                   # for each argument
do
        echo -n "$i? " >/dev/tty
        read response
        case $response in
        y*)     echo $i ;;
        q*)     break
        esac
done </dev/tty
```

echo "$i? \c"

for i in $*

for i in "$*"

for i in "$@"

In summary, here are the rules:

- `$*` and `$@` expand into the arguments, and are rescanned; blanks in arguments will result in multiple arguments.
- `"$*"` is a single word composed of all the arguments to the shell file joined together with spaces.
- `"$@"` is identical to the arguments received by the shell file: blanks in arguments are ignored, and the result is a list of words identical to the original arguments.

```
$ cat news
# news:  print news files, version 1

HOME=.          # debugging only
cd .            # place holder for /usr/news
for i in `ls -t * $HOME/.news_time`
do
        case $i in
        */.news_time)   break ;;
        *)              echo news: $i
        esac
done
touch $HOME/.news_time
$ touch .news_time
$ touch x
$ touch y
$ news
news: y
news: x
$


$ cat news
# news:  print news files, version 2

HOME=.          # debugging only
cd .            # place holder for /usr/news
IFS='
'               # just a newline
for i in  `ls -t * $HOME/.news_time 2>&1`
do
        case $i in
        *' not found')  ;;
        */.news_time)   break ;;
        *)              echo news: $i ;;
        esac
done
touch $HOME/.news_time
$ rm .news_time
$ news
news: news
news: y
news: x
```

```
# news:  print news files, final version

PATH=/bin:/usr/bin
IFS='
'                           # just a newline
cd /usr/news

for i in `ls -t * $HOME/.news_time 2>&1`
do
        IFS=' '
        case $i in
        *' not found')  ;;
        */.news_time)   break ;;
        *)      set X`ls -l $i`
                echo "
$i: ($3) $5 $6 $7
"
                cat $i
        esac
done
touch $HOME/.news_time
```

```
$ echo a line of text >junk
$ put junk
Summary: make a new file                    Type the description
get: no file junk.H                         History doesn't exist...
put: creating junk.H                        ... so put creates it
$ cat junk.H
a line of text
@@@ you Sat Oct  1 13:31:03 EDT 1983 make a new file
$ echo another line >>junk
$ put junk
Summary: one line added
$ cat junk.H
a line of text
another line
@@@ you Sat Oct  1 13:32:28 EDT 1983 one line added
2d
@@@ you Sat Oct  1 13:31:03 EDT 1983 make a new file
$
```

```
$ rm junk
$ get junk                                   Most recent version
$ cat junk
a line of text
another line
$ get -1 junk
$ cat junk                                   Newest-but-one version
a line of text
$ get junk                                   Most recent again
$ replace another 'a different' junk         Change it
$ put junk
Summary: second line changed
$ cat junk.H
a line of text
a different line
@@@ you Sat Oct  1 13:34:07 EDT 1983 second line changed
2c
another line
.
@@@ you Sat Oct  1 13:32:28 EDT 1983 one line added
2d
@@@ you Sat Oct  1 13:31:03 EDT 1983 make a new file
$
```

```
# put:  install file into history

PATH=/bin:/usr/bin

case $# in
        1)      HIST=$1.H ;;
        *)      echo 'Usage: put file' 1>&2; exit 1 ;;
esac
if test ! -r $1
then
        echo "put: can't open $1" 1>&2
        exit 1
fi
trap 'rm -f /tmp/put.[ab]$$; exit 1' 1 2 15
echo -n 'Summary: '
read Summary

if get -o /tmp/put.a$$ $1               # previous version
then                    # merge pieces
        cp $1 /tmp/put.b$$              # current version
        echo "@@@ `getname` `date` $Summary" >>/tmp/put.b$$
        diff -e $1 /tmp/put.a$$ >>/tmp/put.b$$  # latest diffs
        sed -n '/^@@@/,$p' <$HIST >>/tmp/put.b$$ # old diffs
        overwrite $HIST cat /tmp/put.b$$        # put it back
else                    # make a new one
        echo "put: creating $HIST"
        cp $1 $HIST
        echo "@@@ `getname` `date` $Summary" >>$HIST
fi
rm -f /tmp/put.[ab]$$
```

```
# get:

PATH=/bin:/usr/bin

VERSION=0
while test "$1" != ""
do
    case "$1" in
        -i) INPUT=$2; shift ;;
        -o) OUTPUT=$2; shift ;;
        -[0-9]) VERSION=$1 ;;
        -*) echo "get: Unknown argument $i" 1>&2; exit 1 ;;
        *)  case "$OUTPUT" in
            "") OUTPUT=$1 ;;
            *)  INPUT=$1.H ;;
            esac
    esac
    shift
done
OUTPUT=${OUTPUT?"Usage: get [-o outfile] [-i file.H] file"}
INPUT=${INPUT-$OUTPUT.H}
test -r $INPUT || { echo "get: no file $INPUT" 1>&2; exit 1; }
trap 'rm -f /tmp/get.[ab]$$; exit 1' 1 2 15

sed <$INPUT -n '1,/^@@@/w /tmp/get.a'$$'
                /^@@@/,$w /tmp/get.b'$$

awk </tmp/get.b$$ '
    /^@@@/  { count++ }
    !/^@@@/ && count > 0 && count <= - '$VERSION'
    END { print "$d"; print "w", "'$OUTPUT'" }
' | ed - /tmp/get.a$$
rm -f /tmp/get.[ab]$$
```

```
# get:  extract file from history

PATH=/bin:/usr/bin

VERSION=0
while test "$1" != ""
do
    case "$1" in
        -i) INPUT=$2; shift ;;
        -o) OUTPUT=$2; shift ;;
        -[0-9]) VERSION=$1 ;;
        -*) echo "get: Unknown argument $i" 1>&2; exit 1 ;;
        *)  case "$OUTPUT" in
            "") OUTPUT=$1 ;;
            *)  INPUT=$1.H ;;
            esac
    esac
    shift
done
OUTPUT=${OUTPUT?"Usage: get [-o outfile] [-i file.H] file"}
INPUT=${INPUT-$OUTPUT.H}
test -r $INPUT || { echo "get: no file $INPUT" 1>&2; exit 1; }
trap 'rm -f /tmp/get.[ab]$$; exit 1' 1 2 15
# split into current version and editing commands
sed <$INPUT -n '1,/^@@@/w /tmp/get.a'$$'
                /^@@@/,$w /tmp/get.b'$$
# perform the edits
awk </tmp/get.b$$ '
    /^@@@/  { count++ }
    !/^@@@/ && count > 0 && count <= - '$VERSION'
    END { print "$d"; print "w", "'$OUTPUT'" }
' | ed - /tmp/get.a$$
rm -f /tmp/get.[ab]$$
```