**14**

**Error Debugging**

**Objective**

· Turn on Shell debugging options

·Identify conditions which cause a Shell to terminate

·Specify appropriate action within a Shell script when signals are received



Notes

**Termination of a Shell Program**

The following problems cause an active Shell **program** to terminate:

Syntax error in control construct

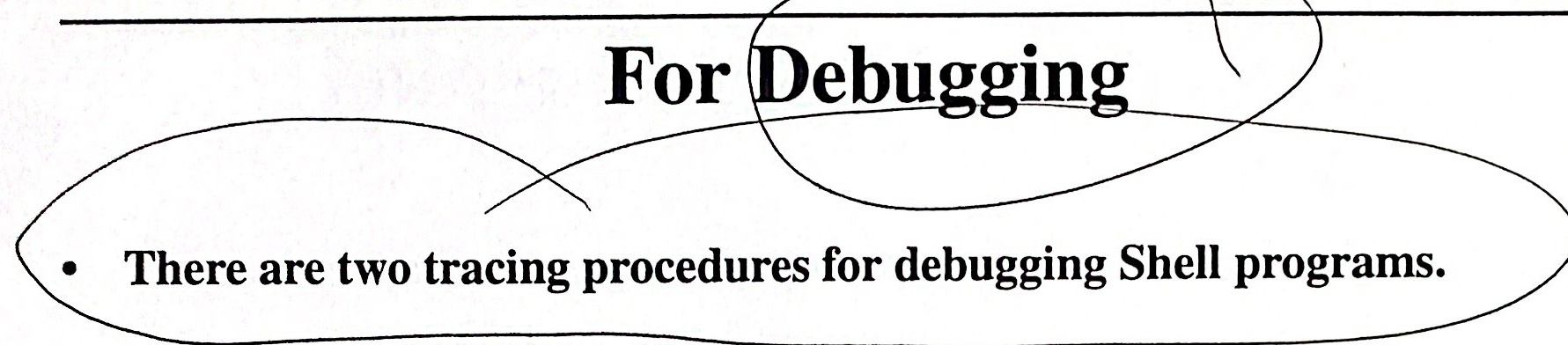
Kill signal

Failure of built-in statement (exceptions: read and test)

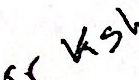
- Syntax error in special substitution

Assignment failure

Notes



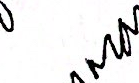
1. sh -v command.file

 or

set -v

**The** **lines** **within** **the** command file **are** printed as they are read.This **command is usually entered directly as** standardinput.

**2.** set -x

**As each parameter substitution is made, the value of** theparameteris **printed along with the command using the parameter.** Thiscommandis  **usually used within a Shell script.**

**Notes**

set options can be eliminated by terminating the current Shell or creating a subshell. Newer versions of the Shell allow set +x to turn off tracing.

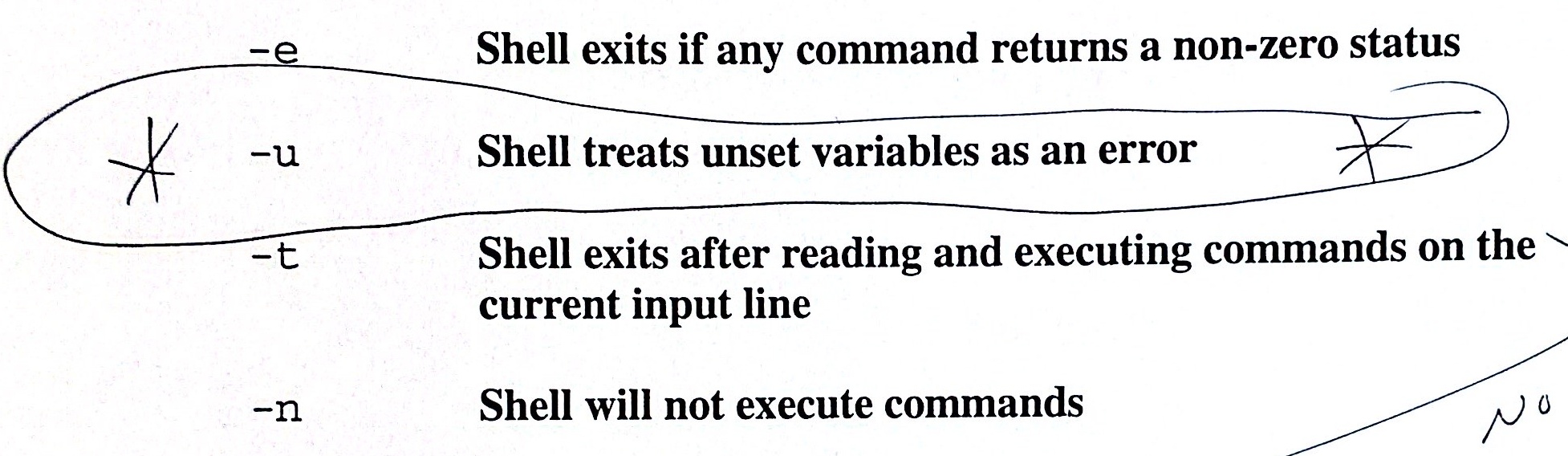
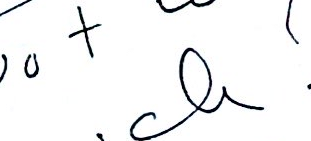
Options currently set can be seen with:

echo $-

**Execution Flags**

**The following flags can be useful** whenplacedinShell procedures.

set **flag**





**Notes**



**Invocation Flags**

**·The following flags are** interpretedbytheShell.

-c string **The commands in the string** areexecuted

-S **Commands are read from stdin**

-i **Interactive** mode

(kill 0 does not kill an interactive Shell)

-r Restricted Shell



**Notes**

**Error Handling**

·A command may fail for one or more of the following reasons:

1. Redirection may fail because the file does not exist or cannot be created.

2.The command does not exist or the user does not have execute permission.

3. The command terminates normally but returns a nonzero status.

4. The command terminates abnormally.

5. Syntax errors

(e.g.no terminating fi for if statement).

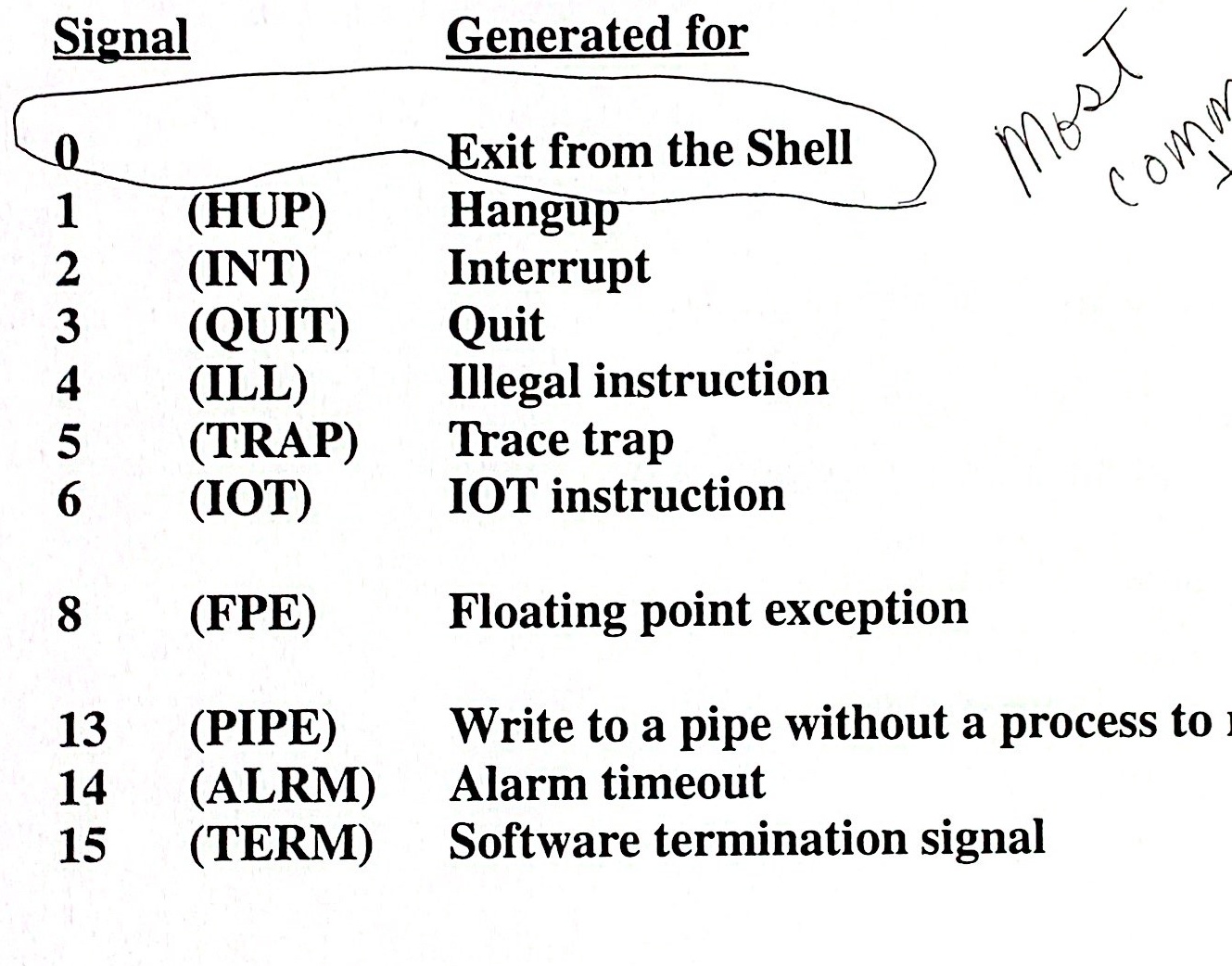
6. Interrupt signals.

7.Failure of internal commands.

Notes

**Signals**

· A signal was originally designed as a means for the operating system to communicate to a process that something had gone wrong. That is still its primary use.



**Notes**

When programs receive certain signals, they terminate. It is possible to trap signals and control the subsequent action.

Signa1 9is not on the list.

kil1 -9 cannot be caught or ignored.

Use kill -1 to see possible signals.

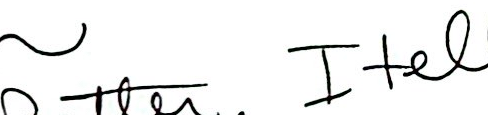
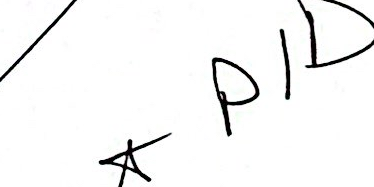
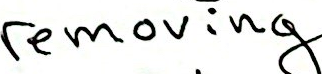
**Traps**

trap proc signal

**Traps allow flow control from one section of code to another upon receiving** **a signal.**

**If a signal occurs after you set the trap, then a procedure is called and** **executed.The Shell will resume execution unless it exits from the** trap **procedure.**

proc **is one or more commands that will be executed whenever one of the** **signals is received.**

**· If the proc is null(" "), the signal is ignored.**$ trap ''1 23 15  remore      ·If the signal is 0, then the proc is executed on exit from the Shell.**$** **trap** 'rm -f(junk.$$; exit' 012315   

**Notes**

·A trap may be reset by entering:

$ **trap <signal>**

$ **trap** 1 3 15

Resets the handling of signals 1,3,and 15 to default values.

trap by itself will list all trap settings.

**$ trap**

**trap Example**

$ cat createfile

**trap** 'rm junk$$; exit' 012 3 15

cat > junk$$ <<!

hello

there

!

cat junk$$

1s

$ sh createfile

hello

there

createfile junk3512

$1s

createfile # the trap removed

# junk3512



**Notes**

·Within the subshell junk$$ exists. When the subshell terminates, the file junk$$ is gone.

The text of a trap procedure is scanned twice by the Shell, once when the trap is defined,and again when it is executed.Therefore, it is usually best to use single quotes around the text of the trap procedure.