APPENDIX C
SHELL COMMAND SUMMARY

This section describes the syntax of the Bourne Shell. It lists special characters, variables, and built-in programming commands used by the shell.

Special Files

`$HOME/.profile` Executed at shell startup.

Special Characters for Filename Generation

Note to reviewers: Markup problem: how to indent these lists.

`*` Match any string of characters.

`?` Match any single character.

`[...]` Match any of the enclosed characters. A pair of characters separated by a minus will match any character lexically between the pair.

Special Characters For Control Flow

Note to reviewers: Book errata: inconsistent over use of italics for file.

`|` Perform pipeline (use output of preceding command as input of following command, e.g., `cat file | lpr`).

`;` Separate sequential commands on the same line.

`&` Run command in background (e.g., `lpr` *file* `&`).

`&&` Execute command if previous command was successful (e.g., `grep` *string file* `&&` `lpr` *file*).

`||` Execute command if previous command was unsuccessful (e.g., `grep` *string1 file* `||` `grep` *string2 file*).

`( )` Execute commands enclosed in `( )` in a subshell; output from the entire set can then be redirected as a unit or placed in the background.

`'...'` Take all characters between single quotation marks literally. (Don't allow special character meaning.)

`\` Take following character literally.

`"..."` Take enclosed characters literally but allow variable and command substitution.

`` `cmd` `` Use output of *cmd* as argument to another command.

`#` Begin a comment in a shell file.

<*file* Take input from *file*.

<*string* Read standard input up to a line identical to *string*.

>*file* Redirect output to *file* (overwrite).

>>*file* Redirect output to end of *file* (append).

Note to reviewers: Book errata: >&digit explanation is plain wrong and example doesn't match.

`>&`*digit* Duplicate standard input from *digit* e.g., `2>&1`.

`<&-` Close standard input.

`>&-` Close standard output.

Variable Substitution

Note to reviewers: Book errata: I think are missing braces in the left hand column.

variable=*value* Set *variable* to *value*.

`$`*variable* Use value of *variable*.

Note to reviewers: Markup problem: how to have two lines for the IP item. Did the book use a table?

`$`*variable*−*value*

`$`*variable*[**:**]−*value* Use *variable* if set; otherwise set to *value*. For example: `TERM=${1:-$TERM)` will set the `TERM` variable to the value of the first argument to a shell script, if given, or else to the existing (default) value of `TERM`.

`$`*variable*=*value* Use *variable* if not set; otherwise set to *value*.

`$`*variable*[**:**]=*value*

`$`*variable*`?`*value* Use *variable* if set; otherwise print *value* then exit.

`$`*variable*[**:**]`?`*value*

`$`*variable*+*value* Use *value* if variable is set; otherwise nothing.

`$`*variable*[**:**]+*value*

If the colon (**:**) is included in these expressions, a test is performed to see if the variable is non-null as well as set.

Shell Parameters Set by the Shell under Execution

`$#` Number of command-line arguments.

`$-` Options supplied in invocation or by the `set` command.

`$?` Return value of last executed command.

`$$` Return process number of current process.

`$!` Return process number of last background command.

Shell Variables Initially Set By `profile`

`$HOME` Default (home directory) value for the `cd` command.

`$IFS` Internal field separators.

`$MAIL` Default mail file.

`$PATH` Default search path for commands.

`$PS1` Prime prompt string; default is `$`.

`$PS2` Second prompt string; default is >.

`$TERM` Specifies the type of terminal.

Shell Functions

name `()` `{`*command1*`;` *...*`;` *commandn*`)`

Create a function called *name* that consists of the commands enclosed in braces. The function can be invoked by name within the current script.

Built-in Commands

file — *file*
Execute contents of *file*.

`break` — `break`[*n*]
Exit from a `for`, `while`, or `until` loop in *n* levels.

`case` —
`case` *value* `in`
pattern1`)` *commands*`;;`
.
.
.
patternn`)` *commands*`;;`
`esac`
For each item in *list* that matches *pattern*, execute *command*.

`cd` — `cd` [*dir*]
Change current directory to *dir*.

`continue` — `continue` [*n*]
Resume *n*th iteration of a *for*, *while*, or *until* loop.

`echo` — `echo` *args*
Print *args* on standard output.

`eval` — `echo` [*arg* . . .]
Evaluate arguments, then execute results.

`exec` — `exec` [*cmd*]
Execute *cmd* in place of current shell.

`exit` — `exit` [*n*]
Exit the shell with exit status *n*, e.g., `exit 1`.

`export` — `export` [*var* . . .]
Export variable *var* to environment.

Note to reviewers: Book errata: what's x?

`for` —
`for` *variable* [`in` *list* . . .]
`do`
commands
`done`
For variable *x* (in optional *list*) do *commands*.

`if` —
`if` *condition*
`then` *commands*
[`elif` *condition2*
`then` *commands2*] . . .
[`else` *commands3*]

	`fi` If *condition* is met, do list of *commands*, or else if *condition2* is met, do *commands2*, otherwise do *commands3*. (See `test` for a list of conditions.)
`hash`	`hash` *cmds* Temporarily add *cmds* to search path.
`login`	`login` [*user* . . .] Log in as another user.
`newgrp`	`newgrp` [*group* . . .] Change your group ID to *group*; if no argument, change back to your default group.
`pwd`	`pwd` Print current working directory.
`read`	`read` [*var* . . .] Read value of *var* from standard input,
`readonly`	`readonly` [*var* . . .] Mark variable *var* as read only.
`return`	`return` Stop execution of current shell function and return to calling level.
`set`	`set` [*t*] [*options*] [*arg* . . .] With no arguments, `set` prints the values of all variables known to the current shell. The following options can be enabled (-option) or disabled (+option). **Note to reviewers:** Markup problem: don't know -ms well enough to copy book. `−−` Don't treat subsequent arguments beginning with − as options. `−a` Automatically export all subsequently defined variables. `−e` Exit shell if any command has a nonzero exit status. `−k` Put keywords in an environment for a command. `−n` Read but do not execute commands. `−t` Exit after one command is executed. `−u` Treat unset variables as an error. `−v` Print commands as they are executed. `−x` Turn on `trace` mode in current shell (echo commands in scripts as they are executed). *arg*. . . Assigned in order to `$1`, `$2`, `. . . $9`.
`shift`	`shift` Perform a shift for arguments, e.g., `$2` becomes `$1`. **Note to reviewers:** Markup problem: don't know -ms well enough to copy book.
`test`	`test` *exp* | `[`*exp*`]` Evaluate the expression *exp*. An alternate form of the command uses `[ ]` rather than the word *test*. The following primitives are used to construct *expression*. `−b` *file* True if *file* exists and is a block special file. `−c` *file* True if *file* exists and is a character special file. `−d` *file* True if *file* exists and is a directory. `−f` *file* True if and*file*exists `−g` *file* True if *file* exists and its set-group-id bit is set. **Note to reviewers:** Book errata: why s1 and not just s? There is no s2 here. `−k` *file* True if *file* exists and its sticky bit is set.

`-n` *s1* True if the length of string *s1* is nonzero.
`-r` *file* True if *file* exists and is readable.
`-s` *file* True if *file* exists and has a size greater than zero.
`-t` [*n*] True if the open file whose file descriptor number is *n* (default is 1) is associated with a terminal device.
`-u` *file* True if *file* exists and its set-user-id bit is set.
`-w` *file* True if *file* exists and is writable.
`-x` *file* True if *file* exists and is executable.
`-z` *s1* True if the length of string *s1* is zero.
s1 = *s2* True if strings *s1* and *s2* are identical.
s1 `!=` *s2* True if strings *s1* and *s2* are not identical.
s1 True if string *s1* is not the null string.
n1 `-eq` *n2* True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, and `-le` may be used in place of `-eq`.

`times` — `times`
Print accumulated process times.

`trap` — `trap` [*cmd*] [*n*]
Execute *cmd* if signal *n* is received. Useful signals include:
0 Successful exit of command.
1 Hangup of terminal line.
2 Interrupt
15 Process is killed.

`type` — `type` *commands*
Print information about *commands*.

`until` — `until` *condition*
[`do` *commands*]
`done`
Until *condition* is met, do *commands* (see `test` for conditions).

`ulimit` — `ulimit` [*size*]
Set maximum size of file that can be created to *size*; if no arguments, print current limit.

`umask` — `umask` [*nnn*]
Set file creation mask to octal value *nnn*.

Note to reviewers: Book errata: isn't there only one definition being removed?

`unset` — `unset` *vars* . . .
Remove definitions for variable *var*.

Note to reviewers: Book errata: why put parenthesis around n?

`wait` — `wait` [*n*]
Wait for specified process with identification number (*n*) to terminate and report its status.

`while` — `while` *condition*
[`do` *commands*]
`done`
While *condition* is met, do *commands* (see `test` for conditions).

filename — *filename*
Read and execute commands from executable file *filename*.

Notes to Reviewers

□ Page 1: Markup problem: how to indent these lists.
□ Page 1: Book errata: inconsistent over use of italics for file.
□ Page 2: Book errata: >&digit explanation is plain wrong and example doesn't match.
□ Page 2: Book errata: I think are missing braces in the left hand column.
□ Page 2: Markup problem: how to have two lines for the IP item. Did the book use a table?
□ Page 3: Book errata: what's x?
□ Page 4: Markup problem: don't know -ms well enough to copy book.
□ Page 4: Markup problem: don't know -ms well enough to copy book.
□ Page 4: Book errata: why s1 and not just s? There is no s2 here.
□ Page 5: Book errata: isn't there only one definition being removed?
□ Page 5: Book errata: why put parenthesis around n?