**While Statement (35.15)**

- General form. The commands in the loop are performed while the condition is true.

```plaintext
while condition
do
  one-or-more-commands
done
```

**While Statement Examples**

```plaintext
# process commands until a stop is encountered
read cmd
while [ $cmd != "stop" ]
  do
    ...
read cmd
done

# process a loop forever
while :
  do
    ...
done
```

**Until Statement (35.15)**

- General form. The commands in the loop are performed until the condition is true.

```plaintext
until condition
do
  one-or-more-commands
done
```

**Until Statement Example**

```plaintext
read cmd
until [ $cmd = "stop" ]
  do
    ...
read cmd
done
```
Shift (35.22)
- The shift command removes an argument from the script file's argument list by shifting all the others over one ($1=$2; $2=$3; $3=$4; ...). It also decrements the $# value by one.
- Example:
  ```bash
  # print the command line arguments
  # two per line
  while [ $# -gt 0 ]
  do
    echo $1 $2
    shift
    if [ $# -gt 0 ]
    then
      shift
    fi
  done
  ```

Exit Status (35.12, 35.16)
- Zero normally indicates success. Nonzero values normally indicate some type of failure. It is a good practice to end each shell script with an “exit 0” command if everything succeeded.
- Most unix utilities, which are written in C, will call “exit(<value>);” upon termination to pass a value back to the shell or utility that invoked it so the utility's status can be checked.

Exit Command Example
- Below could be a script to get a yes/no answer.
  ```bash
  #!/bin/sh
  echo “Please answer yes or no.”; read answer
  while :
  do
    case $answer in
      “yes”) exit 0 ;;
      “no”) exit 1 ;;
      *) echo "Invalid response."
        echo “Please answer yes or no.”
        read answer ;;
    esac
  done
  ```

Testing the Exit Status
- All conditions tested in control statements can also be the exit status of commands. The condition below uses the exit status of the yes.sh script shown in the previous slide.
  ```bash
  if yes.sh
  then
    echo “Please enter filename:”
    ...
  fi
  ```
Regular Expressions (32.3)

- Many shell commands and Unix utilities use regular expressions.
- A regular expression is a description of a possible sequence of symbols (e.g. characters).

Regular Expressions Operations (32.3)

- Concatenation is implicit.
  - ab    # 'a' followed by 'b'
  - abc   # 'a' followed by 'b' followed by 'c'
- * indicates zero or more instances of the preceding regular expression.
  - a*    # "",  a, aa, aaa, ...
  - a*b   # b, ab, aab, aaab, ...
- + indicates one or more instances.
  - a+    # a, aa, aaa, ...
  - a+b   # ab, aab, aaab, ...

Character Classes (32.8)

- '.' indicates any single character except a newline
  - a.b    # 'a' followed by any character followed by 'b'
- Use [...] to indicate one of a set of characters. The '-' operator within [] indicates a range. The '^' after the '[' means match anything not in the set.
  - [abc]  # a, b, c
  - [0-9]  # any decimal digit
  - [a-z]  # any lowercase letter
  - [A-Z]  # any uppercase letter
  - [a-zA-Z] # any letter
  -[^0-9]  # any character other than a decimal digit
  -[^n]    # same as '.'

Anchors (32.5)

- Anchors can be used to indicate that a pattern will only match when it is at the beginning or end of a line.
  - ^echo    # “echo” at the beginning of the line
  - [A-Za-z]+$ # a name at the end of the line
  - ^done$   # “done” on a line by itself
Alternation and Grouping

- Use the '|' character to choose between alternatives. Use parentheses for grouping.
  
a|b  # a or b
a*|b  # "", a, aa, aaa, ..., b
(ab)*c  # c, abc, ababc, abababc, ...
(a|b)*c  # any combination of a's or b's followed by c

Grep (13.2)

- `grep` searches for strings in files that match a regular expression and prints the lines that contain these matches to stdout.

- General form. The pattern is the regular expression. You can specify zero or more files. If no files are specified, then grep reads from standard input. The [...] below means that whatever is inside the brackets is optional. We will discuss the command line options later.
  
grep [-i] [-w] [-c] [-v] pattern [files]

Grep Examples

```bash
grep [Ww]halley *.txt
  # Where Whalley or whalley occurs in .txt files.

grep interger *.c
  # Where did I misspell integer?

grep ^\.*TS report.tr
  # Did I use tables in the report.tr file?

grep Whalley report.tex | grep David > tmp.out
  # Where did I put both my last and first name on the same line of report.tex? Place output in tmp.out.
```

Additional RE References

<table>
<thead>
<tr>
<th>Symbol</th>
<th>ed</th>
<th>vi</th>
<th>sed</th>
<th>awk</th>
<th>grep</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match any character.</td>
</tr>
<tr>
<td>*</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match zero or more preceding.</td>
</tr>
<tr>
<td>^</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match beginning of line.</td>
</tr>
<tr>
<td>$</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match end of line.</td>
</tr>
<tr>
<td>\</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Escape character following.</td>
</tr>
<tr>
<td>[]</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match one from a set.</td>
</tr>
<tr>
<td>{}</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Match a range of instances.</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td>•</td>
<td></td>
<td>•</td>
<td>Match one or more preceding.</td>
</tr>
<tr>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>Match zero or one preceding.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>Separate choices to match.</td>
</tr>
<tr>
<td>()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>Group expressions to match.</td>
</tr>
</tbody>
</table>

- See 32.21 for additional operations, a quick reference guide, and more examples.
Grep Options (13.2, 13.3)

- `-i` will make the search case insensitive.
- `-c` will cause the number of lines matched to be printed.
- `-w` will make the search look for entire words.
- `-v` will cause the lines that don't match to be output.
- `-A num` will print `num` lines after each line that was matched.
- `-B num` will print `num` lines before each line that was matched.
- `-C num` will print `num` lines before and after each line that was matched.

Grep Examples Using Options

grep -i whalley tmp.tr  # finds both Whalley and whalley
grep -c "do {" prog.c  # counts number of do-whiles in prog.c
grep -w "int abs" *.c  # find implementation of abs function
grep -wc if prog.c     # counts if statements in prog.c
grep -v "^#" prog.c   # prints preprocessor commands in prog.c
grep -c "/\[*/\]" prog.c # counts number of comments in prog.c
grep -C 1 Whalley      # prints three lines (before, matched, after) for each line containing Whalley

Wc (16.6)

- `wc` counts the number of lines, words, and characters in files and prints this information to stdout.

- General form. You can specify zero or more files. Like grep and most Unix utilities, it reads from standard input if no files are specified. Again the [...] form means that the ... inside the brackets is optional. The options indicate to only print the number of lines (-l), words (-w), or characters (-c).

  `wc [-l] [-w] [-c] [files]`

Wc Examples

`wc *.txt *.c`  # How big are all of my .txt and .c files?
`wc -l *.c`     # How many lines of code have I written?
`wc -w report.tex`  # How many words is my report?
`wc -c doc.pdf`  # How many bytes is this document that I downloaded over the internet?
### Touch

- Touch creates a file with no data if it does not exist. Sometimes you need to create an empty file.
  
  ```bash
  touch employee.txt
  ```

- It updates the last modification date/time to be the current date/time if the file does exist. This feature can be useful if you copied files into a directory and you need to have the Makefile recompile everything.
  
  ```bash
  touch *.cpp
  ```

### Command Substitution (28.14)

- A pair of backquotes, `...`, does command substitution. This means that the standard output of the command within the backquotes is used as arguments to another command.
  
  ```bash
  count=`wc -w < $1`
  # assigns to count the number of words in the file $1
  
  if [ `wc -l < $2.txt` -lt 1000 ]
  # checks if the number of lines in the $2.txt file is < 1000
  ```

### Xargs (28.17)

- The xargs command reads a group of arguments from standard input and then runs the specified Unix command with that group of arguments.

- General form.
  
  ```bash
  xargs <command>
  ```

- Example:
  
  ```bash
  ls *.c > all_c_files.txt
  vi all_c_files.txt
  xargs gcc -g -c < all_c_files.txt
  ```

### For Statement (28.9, 35.21)

- General form: The shell variable is assigned each word in the list, where the set of commands is performed each time the word is assigned to the variable. If the “in <word_list>” is omitted, then the variable is assigned each of the command-line arguments.

  ```bash
  for <variable> [ in <word_list> ]
  do
    one-or-more-commands
  done
  ```
For Statement Examples

```bash
# make a backup of each of the C files in the current directory
for file in `ls *.c`
do
    cp $file ""$file".bak"
done

# echo each of the command line arguments to a separate line
for arg
do
    echo "$arg"
done
```

For Statement Examples (cont.)

```bash
# place the command line arguments into a single string variable
# where the arguments are separated by blanks
s=for arg
do
    s= "$s $arg"
done ...

# compile each of the files specified in the list in $s
for file in $s
do
    gcc -g -c $file
done
```

Break and Continue (35.23)

- A `break` statement causes immediate termination of a loop. A `continue` statement skips the rest of the commands in the loop and starts the next iteration.

```bash
for file in `ls *.c`
do
    if [ ! -r $file ] && [ ! -d $file ]
    continue
    elif ! gcc -c $file
    then
        echo "could not compile" $file
        break
    fi
done
```

Expr (36.21)

- Expr evaluates an arithmetic or relational expression and prints its results to standard output. Useful when your shell script needs to perform a calculation. Outputs 1 (true) or 0 (false) when evaluating a relational expression.

- General form. Note that the arguments and the operators must be separated by spaces. At least one argument must be specified.

```
expr arg1 [ oper1 arg2 [ oper2 arg3 ... ] ]
```