- Command substitution
- Background processing
- Subshells
- Sequences (conditional, unconditional)
- Pipes
- Wildcards
- Redirection
- Variables (local, environment)
- Scripts
- Built-in commands

CORE SHELL FUNCTIONALITY:

```
$ echo $SHELL
```

- To examine your default shell, type
- To change your default shell use the `chsh` utility

```
CH. 3 THE UNIX SHELLS (Bourne shell, Korn shell, C shell)
```
User commands:

1. It reads a special startup file (.shrc) for each user.
2. It displays a prompt and waits for a user command.
3. If user enters CTRL-D (end of input), the shell terminates.
4. Otherwise, it executes the user command.

When a shell is invoked, either automatically upon login or manually from the keyboard or script, the following takes place:

What does the shell do?
cd dir

) in csh)

echo by default appends a new line (to inhibit new line use -n option

How are you?

$ echo How are you?

- echo arguments

Shells have built-in command; two important ones: echo, cd

Built-in commands

in the title system (using PATH variable)

interactively (ex. ls, vi etc); the shell has to locate the utility

Most Unix commands invoke utility programs stored in the title
```
//<<tok Input redirection; read std. input until tok.
>><tok Prevent special interpretation of characters that follows
  \Expand the value of a variable
  \Comment (rest of characters ignored by shell)
  $Run command in background
  $&Group commands (⋯)
  &&Conditional execution; execute command if previous one succeeds
  ||Conditional execution; execute command if previous one fails
  ;Used to sequence commands
  |Pipe; send output of one process to the input of another
  |command, command substitution; replaced by the output of command
  |command substitution with character within brackets
  [⋯]Pipe-substitution wildcard; matches any character
  |⋯]Pipe-substitution wildcard; matches any single character
  ?Pipe-substitution wildcard; matches 0 or more characters
  *Input redirection (reads std. input from file)
  >Output redirection (appends std. output to file)
  <<Output redirection (writes std. output to file)
```

[\{backslashes\} extracted]
Redirection will see this later.

- The << operator is almost exclusively used in shell scripts.
- matl tony > thom
  - cat x2.2 > y.c
  - cat x1.2 > y.c

  Use the contents of a file as input to a command.
  - Store the output of a command to a file.
  - The shell redirection facility allows you to...
cat /etc/passwd | awk -F: \{print $1\} | sort

dm
adm
bin

adm:adm:adm:adm
:daemon:daemon:daemon
:bin:bin:bin:bin

head -4 /etc/passwd

4

ls -w $PPIP00 $PPIP24 $PPIP48 $PPIP72

ls $command

| command1 | command2 | command3 $
there are 2 users on the system
$ echo there are \`who \-l\` users on the system

today is Sat Jun 19 22:23:28 EDT 1999
$ echo today is \`date\`

There
standard output is inserted in the command's place in the command
A command surrounded by grave accents (, ) is executed and its

Command Substitution
If the previous one fails (returns a non-zero exit code or that command is executed), a set of commands separated by semi-colons (or commands or pipes) separated by semicolons, the next command is executed.

```bash
$ cc myprog.c || echo compilation failed
$ cc myprog.c && a.out

- Conditional sequences:

```bash
$ date > date.txt; ls pwd > pwd.txt
```

- Each command in a sequence may be individually I/O redirected.

```bash
$ date; pwd; ls /proc/self/status
```

- Sequences
(a sub shell is created to execute the grouped commands)
Commands can be grouped by putting them within parentheses

```bash
who > ttt.txt
ps -ef > procs.log
who > ttt.txt
ps -ef > procs.log
```

Sat Jun 19 22:40:43 EDT 1999
```bash
$ more out.txt
$ (date; ls; pwd) > out.txt
```
- Attempt to read from standard input; they terminate.
- Background processes cannot read from standard input; if they
  the terminal, you may redirect the output to a file.
- To prevent output from a background process to come to
  a process id is displayed when a background process begins
  control of the keyboard.
- This starts a sub-shell as a background process which does not take
  or a group of commands
- An & sign can follow a simple command, pipeline, sequence

Background Processing
Comment Line

Note: pound sign on first column in any other line implies a
* else the script is interpreted by the Bourne shell.
then the appropriate shell is used to interpret the script
   \# or /bin/bash etc
* if the first line of the script is of the form
      #! /bin/sh
  the script is executed.
then the script is interpreted by the shell from which
   \#)
* if the first line of the script is a pound sign (#),
script was written for; the rules are:
When a script runs, the system determines which shell the

Before executing it by simply typing the title name.
   \# use the chmod utility to set execute permissions on the title
for execution.
   \# Any series of shell commands may be stored in a text file
-
Shell Programs/Scripts
- Always recommended to use #!path

#!/bin/csh
# A simple C-shell script
echo -n "The date today is ".date
with an empty local-variable space. Gets a copy of the parent’s environment space but starts a local-variable space; when a child shell is created it

Every shell has two data areas: an environment space and

A subshell has its own working directory; cd commands in subshells

- Background processes
- Script execution
  - Grouped command (ls; pwd; date)

Within a login shell there are several ways a subshell can

Subshells
name with a $ sign.

- Accessing variables in all shells is done by prefixing the

$HOME, $PATH, $MAIL, $USER, $SHELL, $TERM

in all shells:

local variables. Some pre-defined environment variables and

- Every shell has a set of pre-defined environment variables and

variables. Both kinds hold data in string format.

- A shell supports two kinds of variables: local and environment

Variables
To assign environment variables:

```
set variable = "value"
```

csh: set variable=value

export variable

To make a variable an environment variable in sh, ksh, csh:

```
variable = "value"
```

sh, ksh: variable=value

different shells:

- Assigning values to variables is done differently in different shells:
a list of all command line arguments

* $ if applicable

$n refers to the nth command line argument

$name of shell script (if applicable)

$0 process ID of shell

$1 $2 $3 $4 $5 ...

- Common built-in variables with special meaning:
29 Jun 20 21:33 paul.554

This script places the date into a temporary file called paul.554.

the list of all arguments is paul ringo george john

declaration is paul

the name of this file is ./script2.csh

$ cat script2.csh

# cat script2.csh

```bash
rm .1
ls -l .1
date > .1.
```

```bash
$ echo this script places the date into a temporary file called $1.
$ echo the list of all arguments is $*
$ echo the first argument is $1
```

```bash
$ cat script2.csh
```
My name is Ray; the date is Sun Jun 20 21:59:13 EDT 1999

```
$ echo "My name is $USER; the date is date."
 3 * 4 = 12
3 * 4 = 12
```

When quotes are nested only the outer quotes have any effect

- Double quotes (") inhibits wildcard replacement
- Single quotes (') inhibits only variable substitution, and command substitution

Quoting
mail sent to ra

here. cash ra

$ echo mail sent to $1

ENDOPTEXT

USER

Please see me regarding some exciting news!

dear $1,

mail $1 >> ENDOPTEXT

$ cat here. cash

Here Documents
$ mail
"/var/spool/mail/raj": 6 messages 1 new
      5 raj@kamakshi.gsu.edu Sun Jun 20 22:13 18/420
  >N  6 raj@kamakshi.gsu.edu Sun Jun 20 22:14 14/377

&
Message 6:
From raj   Sun Jun 20 22:14:31 1999
Date: Sun, 20 Jun 1999 22:14:31 -0400
From: raj@kamakshi.gsu.edu
To: raj@kamakshi.gsu.edu

Dear raj,
Please see me regarding some exciting news!
raj
nohup command $ 

- To keep the background processes to continue in $n
and Ksh, use

when you log out (csh allows them to continue)

- Bourne and Ksh automatically terminate background processes

```bash
PID : process ID
```

```bash
# include long listing
# include full listing
ps -ef | include all running processes
```

terminate.

- wait allows the shell to wait for one of its child processes to

id terminate processes based on process ID

- kill command terminates processes and their attributes

- ps command generates a list of processes and their attributes
SIGKILL (9) is used if the process refuses to die. If SIGTERM is not specified the default signal is SIGTERM (15).

kill -signal pid

$" SIGINT
   SIGILL
   SIGSEGV
   SIGBUS
   SIGPWR
   SIGXFSZ
   SIGXCPU
   SIGPROF
   SIGWINCH
   SIGPWR
   SIGALRM
   SIGHUP
   SIGTERM
   SIGINT
   SIGQUIT
   SIGILL
   SIGINT
   SIGQUIT
   SIGILL
This feature is used in advanced shell scripts.

```
( )

`sleep 30; echo done 2`

( )

`sleep 30; echo done 1`

`done 4`

`done 3`

`done 2`

`done 1`

`done 0`

Waiting for child processes
```
PATH variable.

Normally, the current working directory is included in the
PATH environment variable from left to right. For an executable
that matches the command, the shell searches the directory names that are stored in the
PATH environment variable before any directory above it.

- If the command is not a built-in and not a full pathname,
  - Error occurs if the executable is not found.
  - Command is the absolute path name of an executable.
  - If the command begins with a / the shell assumes that the
directly interpreted by the shell.

- If the command is a shell built-in such as echo or cd it is

Finding a command: $PATH
- Homebrewed utilities: Some Unix users create their own versions of some Unix utilities and store them in their bin directory; then they place their bin directory ahead of all other directories so that their version of the utility is executed.

- If PATH is empty or is not set, only the current working directory is searched for the executable.
the last command is returned by default.
If the script does not exit with an exit code, the exit code of:

- exit number

Any script written by you should contain the exit command:

- The special variable $? contains the exit code of the last command execution. In csh $status also contains the exit code.

- All built-in commands return 1 when they fail.

- By convention, a 0 value means success and a non-zero value means failure.

Every Unix process terminates with an exit value.

Termination and Exit Codes:
the login session terminates.
If the shell was a login shell,
As a result, if the command terminates, the shell also
replaced with the command in the process’ memory space.
The exec shell command causes the shell’s image to be
- exec command

5
$ echo $x
5
$ eval "echo x=5",
- eval command

Common Core Built-in commands
First argument is b, all args are a b c d
First argument is a, all args are a b c d
$ script.csh a b c d

*echo first argument is $1, all args are
shift
*echo first argument is $1, all args are
# ./bin/csh

Useful in processing command line parameters.
To be renamed $1.$(n-1) and $1 is lost.
This command causes all of the positional parameters $2..$n

- shift
umask $ 

- To see current umask value use the command

  umask octalValue $ 

- To change umask value use the command

  umask octalValue $ 

  umask value say 022 to produce the permission 644 with the permissions, which is usually 666, is masked (xor) whenever a file is created (say by vi or by redirection), the default value is 022 octal.

  Every Unix process has a special quantity called umask value.