

UNIX for Programmers and Users

“UNIX for Programmers and Users”

Third Edition, Prentice-Hall, GRAHAM GLASS, KING ABLES

Listing Group: groups

- `groups`
- The `groups` utility allows you to list all of the groups that you're a member of, and it works like this:

`groups` `userId`

- When invoked with no arguments, the group utility displays a list of all of the groups that you are a member of.
- If the name of a user is specified, a list of the groups to which that user belongs are displayed.
- Example:

```
$ groups userId
```

--> list my groups

```
cs      music
```

```
$ _
```

Changing File Group: chgrp

- Changing a File's group : chgrp

chgrp -R groupname fileName

- The **chgrp** utility allows a user to change the group of files that he/she owns.
- A super-user can change the group of any file.
- All of the files that follow the groupname argument are affected.
- The **-R option** recursively changes the group of the files in a directory.

Changing File Group: example

```
$ ls -lg heart.final
```

```
-rw-r--r--  1  glass  cs  213 Jan 31 00:12 heart.final
```

```
$ chgrp music heart.final --> change the group.
```

```
$ ls -lg heart.final --> confirm the changes.
```

```
-rw-r--r--  1  glass  music 213 Jan 31 00:12 heart.final
```

```
$ _
```

- The `chgrp` utility is also used to change the group of a directory.

Changing File Owner: chown

`chown -R newUserId fileName`

- The `chown` utility allows a super-user to change the ownership of files.
- Some Unix versions allow the owner of the file to reassign ownership to another user.
- All of the files that follow the `newUserId` argument are affected.
- The `-R` option recursively changes the owner of the files in directories.

Changing File Owner: chown

- Example: change the ownership of “heart.final” to “tim” and then back to “glass” again:

\$ **ls -lg heart.final** --> to view the owner before the change.

```
-rw-r----- 1 glass music 213 Jan 31 00:12 heart.final
```

\$ **chown tim heart.final** --> change the owner to “tim”.

\$ **ls -lg heart.final** --> to view the owner after the change.

```
-rw-r----- 1 tim music 213 Jan 31 00:12 heart.final
```

\$ **chown glass heart.final** --> change the owner back to “glass”.

\$ _

Change User Groups: newgrp

`newgrp [-][groupname]`

- The `newgrp` utility with a groupname as an argument, **creates a new shell with an effective group ID** corresponding to the groupname.
- The old shell sleeps until the termination of the newly created shell.
- User must be **a member of the specified group**.
- If the argument is a dash(-) instead of **a groupname**, **a shell is created with the same settings** as those of the shell that was created by logging into the system.

Changing Groups: example

\$ **date > test1** --> create from a “cs” group shell.

\$ **newgrp music** --> create a “music” group shell.

\$ **date > test2** --> create from a “music” group shell.

^D

\$ **ls -lg test1 test2** --> look at each file's attributes.

| | | | | | | |
|------------|---|-------|----|-----------|-------|-------|
| -rw-r--r-- | 1 | glass | cs | 29 Jan 31 | 22:57 | test1 |
|------------|---|-------|----|-----------|-------|-------|

| | | | | | | |
|------------|---|-------|-------|-----------|-------|-------|
| -rw-r--r-- | 1 | glass | music | 29 Jan 31 | 22:57 | test2 |
|------------|---|-------|-------|-----------|-------|-------|

\$ _

Adding group & assigning to user

- `sudo groupadd -g 2000 ug1`
- `sudo tail /etc/group`
- `groups srd`
- `sudo adduser srd ug1`
- `groups srd`

Shells

• INTRODUCTION

A shell is a program that is an interface between a user and the raw operating system.

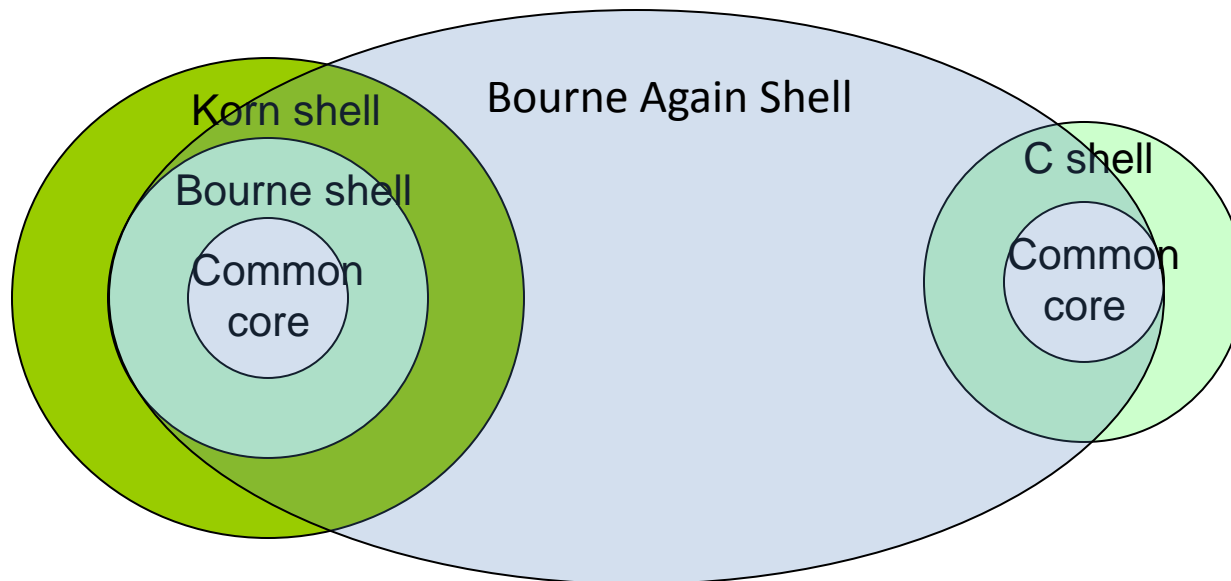
It makes basic facilities such as multitasking and piping easy to use, and it adds useful file-specific features such as wildcards and I/O redirection.

There are four common shells in use:

- the Bourne shell
- the Korn shell
- the C shell
- the Bash shell (Bourne Again Shell)

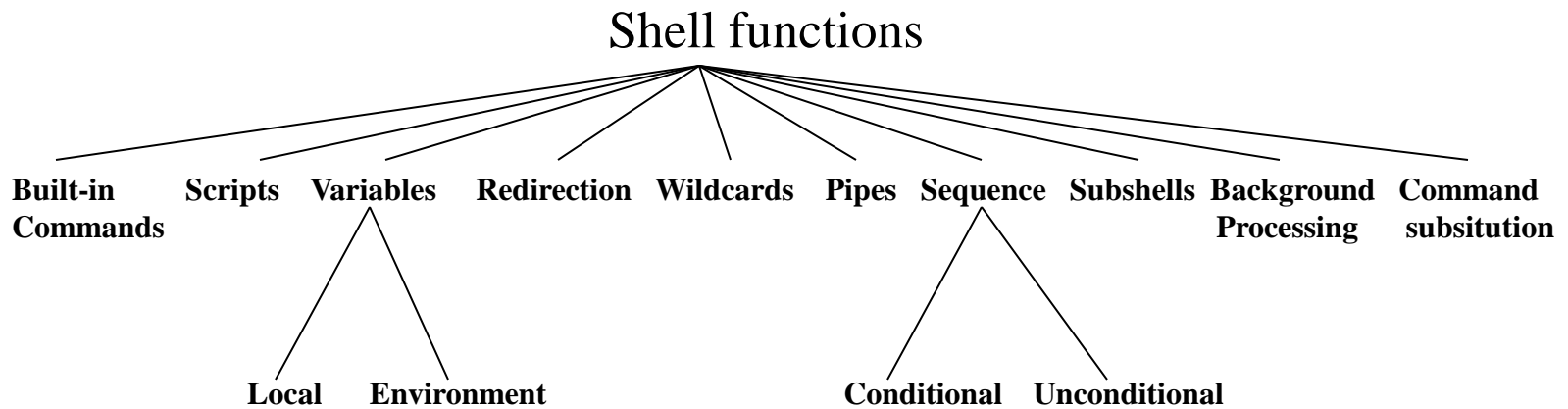
- **SHELL FUNCTIONALITY**

- Here is a diagram that illustrates the relationship among the four shells:



- **SHELL FUNCTIONALITY**

- The features shared by the four shells



- **SELECTING A SHELL**

The system administrator chooses a shell for any UNIX user.

\$ prompt represents probably a Bash, Bourne or a Korn shell.

% prompt represents probably a C shell.

- **Utility : chsh**

- **chsh** allows you to change your default login shell.

It prompts you for the full pathname of the new shell,
which is then used as your shell for subsequent logins.

- In order to use **chsh**, you must know the full pathnames of the four shells. Here they are:

| Shell | Full pathname |
|--------|---------------|
| Bourne | /bin/sh |
| Bash | /bin/bash |
| Korn | /bin/ksh |
| C | /bin/csh |

- **SELECTING A SHELL**

Change the default login shell from a Bourne shell to a Bash shell:

```
$ echo $SHELL      ---> display the name of current login shell.  
/bin/bash         ---> full pathname of the Bash shell.
```

```
$ chsh             ---> change the login shell from bash to sh.  
Changing login shell for glass  
Old shell : /bin/bash ---> pathname of old shell is displayed.  
New shell: /bin/sh  ---> enter full pathname of new shell.
```

```
$ echo $SHELL  
/bin/sh           ---> full pathname of the Bourne shell.  
$
```

• SHELL OPERATIONS

When a shell is invoked, either automatically during a login or manually from a keyboard or script, it follows a preset sequence:

1. It reads a special startup file, typically located in the user's home directory, that contains some initialization information.
2. It displays a prompt and waits for a user command.
3. If the user enters a Control-D character on a line of its own, this command is interpreted by the shell as meaning "end of input", and it causes the shell to terminate;

otherwise, the shell executes the user's command and returns to step 2.

- **SHELL OPERATIONS**

Commands range from simple utility invocations like:

```
$ ls
```

to complex-looking pipeline sequences like:

```
$ ps -ef | sort | wc -l
```

- a command with a backslash(\) character, and the shell will allow you to continue the command on the next line:

```
$ echo this is a very long shell command and needs to \
  be extended with the line-continuation character. Note \
  that a single command may be extended for several lines.
```

```
$ _
```

- **EXECUTABLE FILES VERSUS BUILT-IN COMMANDS**

Most UNIX commands **invoke utility programs** that are stored in the directory hierarchy.

Utilities are stored in files that have **execute permission**.

For example, when you type

```
$ ls
```

the shell locates the executable program called “ls”, which is typically found in the **“/bin”** directory, and executes it.