Introduction to z/OS UNIX

Introduction to z/OS UNIX - Course Objectives

On successful completion of this class, the student, with the aid of the appropriate reference materials, should be able to:

1. Accomplish work to support applications that use files in the Hierarchical File System (HFS and zFS), using TSO commands or z/OS UNIX System Services shell commands, in particular:
    a. copy / move data between z/OS files
    b. copy / move data between HFS files
    c. copy / move data between z/OS files and HFS files, including converting code pages as necessary
    d. create, display, rename, delete, archive, unarchive, compress, and uncompress data in HFS files
    e. edit and browse HFS data

2. Use the ISPF shell, ISHELL, to work with HFS files from a TSO/ISPF session

3. Display web pages residing in HFS files on the mainframe on his or her local browser on your company intranet

4. Create and maintain simple HTML files to display basic text and images and to link from one page to another

5. Run batch jobs that access files and programs (DLLs) in the HFS

6. Submit jobs to run in the batch from the z/OS UNIX shell

7. Use the ISPF UNIX Directory List utiltiy (3.17), if possible

8. Use telnet to access z/OS UNIX, if possible in his or her installation

9. Understand how UNIX System Services expands and extends the capabilities of the mainframe.

Introduction to z/OS UNIX - Topical Outline

Day One

Introduction to z/OS UNIX
- z/OS and UNIX System Services
- TSO User ID, Profiles, and the UNIX User ID
- z/OS UNIX - The Shell Interface under OMVS
- Computer Exercise: Getting to the z/OS Shell 22

More Identities
- Effective UID, Group ID
- Shell Command Syntax
- Shell Commands: id
- OMVS - Some Options
- Shell Commands: logname, cal, date, echo, man, who, whoami, fc, history, r, alias, hash, unalias
- Computer Exercise: Practice With Commands 55

Variables
- Shell variables
- Environment variables
- Shell Commands: export, env
- The Shell Environment
- Shell Commands: readonly, printenv
- Some Common Shell Variables
- Computer Exercise: Working With Shell Variables 72

UNIX Files and the Hierarchical File System (HFS)
- Hierarchical File System (HFS)
- Paths
- Commands for Files and Paths
- Shell Commands: pwd, cd, basename, dirname
- File Access Permissions

Managing and Deleting Directories and Paths
- Making Directories
- Default Permissions
- Shell Commands: umask, mkdir, rmdir
- TSO CommandL: MKDIR
- Shell Command: ls
- Computer Exercise: Creating and Deleting Directories 121

Introduction to z/OS UNIX - Topical Outline, p.2.

Day One, continued

Data Transfer - Part 1: TSO Commands
- File Organizations Supported
- Record Formats
- Code Pages
- Names
- Copying Files
- TSO Commands: OPUT, OPUTX, OGET, OGETX
- Computer Exercise: Copying Files 143

Day Two

Displaying Data Under the Shell
- Displaying data
- Shell Commands: head, tail
- Shell Commands: Using z/OS Data Set Names
- Shell Commands: cat, pg, more, pr, nl, asa, fold, cut
- Computer Exercise: Displaying Files 160

Managing Files and Directories
- Commands for Files and Directories
- UNIX Files - Review
- Common directories
- Shell Commands: touch, rm, chmod, chaudit, chgrp, ln, link
- Some more Environment Variables
- Shell Commands: du, df
- Computer Exercise: Working With Files 200

Piping and Redirection
- "Standard" files and file descriptors
- Pipes vs. File redirection
- Shell Commands: tee, paste
- Computer Exercise: Piping and Redirection 224

OEDIT and OBROWSE TSO Commands
- More Irish commands
- The TSO OEDIT command
- The TSO OBROWSE command
- Shell Commands: oedit, obrowse
- Computer Exercise: OEDIT 238

Introduction to z/OS UNIX - Topical Outline, p.3.

Day Two, continued

ISHELL: Doing UNIX-y Things in an ISPF-y Way
    The TSO ISHELL command
    Computer Exercise: Using ISHELL 254

Locales and Internationalization in UNIX
    Code Pages, Internationalization, and Locales
    Internationalized Applications
    What's This Got To Do With UNIX?
    Setting Locale Values
    Shell Commands: mkcatdefs, gencat, runcat, dspcat, dspmsg, localedef, locale, iconv, tr
    Computer Exercise: tr command 282

Day Three

More OMVS Features
    OMVS - The Parameters
    Computer Exercise: More Work with OMVS 298

Data Transfer - Part 2: UNIX Commands
    Shell Commands: cp, copytree, mv
    Shell Commands: tso
    Computer Exercise: Copying, Renaming, and Moving Files 315

Compressing and Uncompressing
    Compressing and Uncompressing files
    Shell Commands: compress
    Shell Commands: uncompress
    Shell Commands: zcat

Introduction to The Web
    The Web - Basic Concepts
    Web Servers on z/OS
    Computer Exercise: Displaying a Web Page from the Mainframe . 338

Introduction to Markup Languages
    Overview: Markup Languages
    SGML / HTML / XML
    HTML - An Introduction
    Computer Exercise: Writing Basic HTML 358

Introduction to z/OS UNIX - Topical Outline, p.4.

Managing Archive Files
- Archive Files
- Shell Commands: pax
- Shell Commands: tar
- Computer Exercise: Unwinding an Archive 374

Accessing HFS Files and Programs from Batch and TSO
- Applications
- JCL and HFS files
- ALLOCATE and HFS files
- Programs and subroutines
- DLLs
- UNIX System Services
- C Run Time Library services
- Computer Exercise: Accessing HFS files through non-UNIX applications 403

Submitting jobs from the shell
- Shell commands: submit
- Computer Exercise: Submitting a batch job from the shell 408

Unix directory list support
- HFS file attibutes
- The Udlist utiltity

telnet (Optional)
- telnet and rlogin
- The telnet interface
- Shell commands: stty, tabs, unexpand, expand
- Computer Exercise: The telnet Experience 449
- A Little UNIX Humor

Wrap Up: Where Do We Go From Here
- Further Studies
- Resources

Appendices
- OCOPY
- Course summary
- Index

UNIX Standards

There have been a number of UNIX standards that have become popular over the years, so many that "open source" is almost "open chaos". In an attempt to re-establish common ground, two major organizations, The Open Group, an organization sponsored by many of the major vendors, and the IEEE (Institute of Electrical and Electronics Engineers) a well-respected technical, non-profit organization, have combined to establish the Single UNIX Specification (SUS), a document that both organizations have pledged to adhere to.

This document merges and extends standards by establishing a common vocabulary and set of APIs (Application Programming Interface's, including commands and utilities) that build on the IEEE's POSIX standard and The Open Group's UNIX 95 (XPG) standard.

Some web pages that are of interest for those who want to explore more details:

http://www.unix-systems.org/	- main page to explore the SUS from; the standard may be downloaded from here in hypertext format or PDF
http://www.ieee.org/index.html	- home page for the IEEE
http://www.opengroup.org/	- home page for The Open Group

IBM's z/OS UNIX System Services conforms to various levels of these standards and includes extensions to the standards. Remember that while any given extension may be nice to have / use, using such a feature may make your work less portable to other UNIX platforms (or even not portable to such platforms).

In 2006, with the advent of z/OS 1.8, some commands had to change the meaning of some of their options and flags in order to conform to version 3 of SUS (SUSv3). A special environment variable was defined, _UNIX03, such that if that variable has a value of YES, then the new behavior takes effect; if this variable is undefined or does not have a value of YES, the prior behavior is in effect. Places where this is a concern are documented throughout these materials.

In 2008, SUSV4 was released. No information on how this is implemented for z/OS as of this course publication date.

Section Preview

❒ **Introduction**

- ♦ **z/OS and UNIX System Services**
- ♦ **TSO User ID**
- ♦ **Profiles**
- ♦ **UNIX User ID**
- ♦ **z/OS UNIX - The Shell Interface Under OMVS**
- ♦ **Getting to the z/OS Shell (Machine exercise)**

z/OS and UNIX System Services

❐ **The advent of UNIX System Services in z/OS (and OS/390 earlier) introduced a bit of a cultural divide in mainframe shops**

The z/OS Culture	**The UNIX Culture**
♦ **Control**	♦ **Ease of use**
♦ **Discipline**	♦ **Cleverness**
♦ **Integrity**	♦ **Functionality**
♦ **Serial execution**	♦ **Parallel execution**

❐ **Which is not to say that each culture doesn't have some of the characteristics of the other, they do**

- ♦ **It's a matter of degree and perception**

❐ **In this class, we assume you're already familiar with the z/OS world, so we will work on learning how to function on the UNIX side of things**

- ♦ **Our goal is to get familiar with UNIX as it runs under z/OS, which is quite a bit different than standard UNIX**
  - ✗ You can access z/OS files and services, for example, which doesn't make any sense on other UNIX systems

z/OS and UNIX System Services, continued

❒ **Note that UNIX is always properly spelled with all capital letters.**

- ♦ **The IBM implementation is formally called "z/OS UNIX System Services"**

- ♦ **The official short version of the name is simply "z/OS UNIX"**

- ♦ **An un-official abbreviation, "USS", is in farily wide use, although IBM discourages that term because it has a different meaning in the VTAM component ("Unformatted Session Services")**

❒ **We'll start by accessing the UNIX shell using the *omvs* command, from TSO READY or ISPF option 6**

- ♦ **Note: you will sometimes hear references to "the Irish commands"**

  - ✗ This is a reference to the commands originally called Open Edition commands, so the names are, for example OMVS, OBROWSE, OEDIT, and so on

  - ✗ Coupled with a reference to the fact that many Irish family names begin with "O'" (O'Neil, O'Reilly, and so on, for example)

z/OS and UNIX System Services, continued

- ❒ **You can also access the z/OS UNIX shell through the *telnet* and *rlogin* methods**

  - ♦ **We discuss these briefly later on, but we need to mention them in passing soon, so we wanted to make sure you realized these are alternative options**

- ❒ **In all cases, to use z/OS UNIX, you need to have an ID - an Identity ...**

TSO User ID

❐ **Every user of TSO must have a TSO logon id, referred to by any of the terms "user id", "TSO ID", "TSO user id", and so on**

- ♦ **We will use the term "TSO ID" to distinguish this term from the UNIX term "UID" (discussed shortly)**

❐ **A TSO ID is assigned by your security administrator and must follow these rules:**

- ♦ **1-7 alphanumeric characters, the first of which must be alphabetic**
  - ✗ Although you may enter your TSO ID in lower case, the logon process forces it to upper case so alpha characters in TSO IDs are effectively upper-case
- ♦ **In addition to the TSO restrictions, your installation may have standards regarding the format of TSO IDs**

❐ **Any given TSO ID can only be logged into a particular z/OS system only once at a time**

- ♦ **That is, you cannot establish two TSO sessions on a single system using the same TSO id**

Profiles

❒ **Each TSO ID will have a TSO profile associated with it**

- ♦ **This profile specifies session attributes, for example: if TSO should automatically prefix unquoted data set names with your TSO ID**

❒ **In addition, each user must have a security profile (using RACF, ACF2, Top Secret, or other security package)**

- ♦ **This profile specifies security privileges and authorizations, such as if you are allowed to create or delete certain data sets**

- ♦ **Security groups can be established with common sets of privileges and authorizations, and a TSO user must belong to at least one of these groups**

  - ✗ Note that a user may belong to multiple groups

UNIX Userid

- ❒ **In addition to [or, in some cases, instead of] a TSO ID, a user of z/OS UNIX must have a UNIX Userid, or user name**
  - ♦ **A <u>Userid</u> is a name that is known to the system for login purposes**
    - ✗ UNIX standards require a Userid to be case sensitive and to allow periods, dashes, and underscores in the name
    - ✗ However, in z/OS, the z/OS UNIX Userid is generally the TSO user id in all lowercase (so no special characters, for example)
- ❒ **Users logging in through *telnet* or *rlogin*, do not need a TSO ID, but they must have a UNIX Userid**
- ❒ **Any given z/OS UNIX user may be logged into z/OS UNIX multiple times at once**
  - ♦ **As well as possibly being logged on to the system through TSO once**
- ❒ **z/OS UNIX <u>requires</u> a z/OS security product (one of: RACF, ACF2, Top Secret, etc.) which is used to define both TSO users and UNIX users to the system**

UNIX User ID

- ❐ **A UNIX user must also have a User ID, or "UID" assigned**

- ❐ **UID's must follow these rules**

  - ♦ **A UID is a 4-byte binary integer in the range 0 - 2,147,483,647**

  - ♦ **A user (person) may have multiple Userids (user names), each with one UID; and a given UID may be assigned to multiple Userids (although this is not recommended)**

  - ♦ **UID of 0 (zero) is special: a superuser**

    - ✗ Users with superuser status can access and change all UNIX system resources

    - ✗ A UID of 0 may be assigned to multiple users (although this is not recommended)

- ❐ **A user database is maintained that keeps track of each user, their UID, userid and other information**

z/OS UNIX - The Shell Interface Under OMVS

❒ **So, before you can use *omvs*, the systems staff must have defined the appropriate IDs, profile files, and security segments**

- ♦ **Details on setting these up are beyond the scope of this course, although we will explore aspects of the file system and security later**

- ♦ **So let's see what happens when you enter the *omvs* command ...**

z/OS UNIX - The Shell Interface Under OMVS, p.2.

❒ **The *omvs* interface is a 3270-based interface**

- **The first screen you see will look something like this:**

```
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2006
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.


 - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -       User-specified message, or omitted          -
 - - - - - - - - - - - - - - - - - - - - - - - - - - -

$
 ===> _
                                                                     RUNNING
ESC=¢  1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
       7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

❒ **The major areas of the screen are**

- **The <u>body</u>, everything above the command line**
  - ✗ This displays commands issued and any viewable output

- **The <u>command line</u> (the line with the arrow (===>)**
  - ✗ Key in a command here and press <Enter>

- **The <u>status indicator</u> (RUNNING in the example)**

- **The <u>Escape character(s) indicator</u> (¢ in the example)**

- **The <u>function key display</u>**

z/OS UNIX - The Shell Interface Under OMVS, p.3.

Notes

- **The dollar sign ($) is the z/OS shell prompt character; its presence indicates the shell is ready to accept input**

- **If a command you are entering is too long to fit on the command line, enter a backslash (\) and press <Enter>**

  - ✗ The shell will accept your command as written so far and the prompt changes to a greater than symbol (>) to indicate it understands there is more to come for the current command

- **The status indicator is one of these values:**

RUNNING - either an application program is running or the terminal is being polled

INPUT - the shell is waiting for input from the user

MORE... - the screen is full and is waiting for your request to show the next screen of data (do so by pressing one of these function keys: Scroll, HalfScr, or Bottom)

HIDDEN - input characters are not being echoed back to the terminal (*e.g.:* a password)

NOT ACCEPTED - indicates the application or shell is hung and must be interrupted to move on

SUBCOMMAND - indicates you are operating in subcommand mode

❒ **You may also see more than one of these, separated by slashes (*e.g.*: HIDDEN/MORE)**

z/OS UNIX - The Shell Interface Under OMVS, p.4.

<u>Notes, continued</u>

- **The ESC area indicates what character(s) can be used to generate an escape sequence**
  - ✗ To send control information (a signal) instead of data to the shell, a command, or a program
    - ➢ More on this later

- **The function key area shows you what *omvs* subcommands are currently assigned to function keys**
  - ✗ More on this later, but for now, notice the scrolling related function keys:
    - ➢ Top
    - ➢ Bottom
    - ➢ BackScr
    - ➢ Scroll
  - ✗ As you issue commands and the responses come back to you, the lines scroll up off the top of the screen
  - ✗ If you want to go back to look at earlier interactions, then Top and BackScr are for you; to go forward, press Bottom or Scroll

z/OS UNIX - The Shell Interface Under OMVS, p.5.

❒ We will be exploring more than 60 shell commands over the next several days

❒ But for now, we just mention one command: *exit*

- ♦ Exit terminates the shell and you see this message:

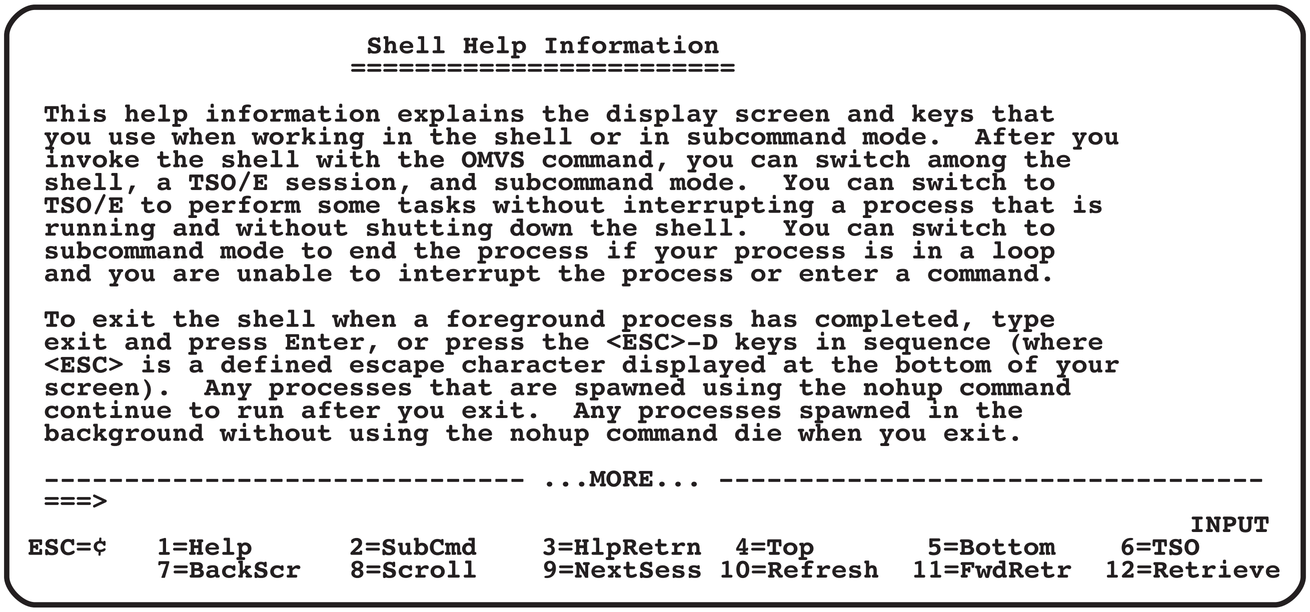
```
$ exit
 >>>> FSUM2331 The session has ended.  Press <Enter> to end OMVS.
                                                                       INPUT
ESC=¢   1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

- ♦ When you press <Enter>, you are back where you previously started (TSO READY or TSO/ISPF option 6)

z/OS UNIX - The Shell Interface Under OMVS, p.6.

❒ **Notice that function key 1 is set to** Help

- ♦ **If you press this key you'll see something like this:**



```
                    Shell Help Information
                   ========================

 This help information explains the display screen and keys that
 you use when working in the shell or in subcommand mode.  After you
 invoke the shell with the OMVS command, you can switch among the
 shell, a TSO/E session, and subcommand mode.  You can switch to
 TSO/E to perform some tasks without interrupting a process that is
 running and without shutting down the shell.  You can switch to
 subcommand mode to end the process if your process is in a loop
 and you are unable to interrupt the process or enter a command.

 To exit the shell when a foreground process has completed, type
 exit and press Enter, or press the <ESC>-D keys in sequence (where
 <ESC> is a defined escape character displayed at the bottom of your
 screen).  Any processes that are spawned using the nohup command
 continue to run after you exit.  Any processes spawned in the
 background without using the nohup command die when you exit.

 ------------------------------ ...MORE... ----------------------------------
 ===>
                                                                       INPUT
ESC=¢   1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

❒ **The ... MORE ... at the bottom of the screen indicates you can scroll down**

- ♦ **At this point, function keys 4, 5, 7, and 8 become useful**

- ♦ **To leave Help, press function key 3**

The Shell at Work

- ❒ **Generally, the shell works with three files...**

  - ♦ stdin **- where commands and data are entered; normally mapped to the keyboard (this is how the command line contents are made accessible to commands and programs)**

  - ♦ stdout **- where the results of commands are written; normally mapped to the screen**

  - ♦ stderr **- where error messages and diagnostics are written; normally mapped to the screen also**

- ❒ **Each file may also be represented by a** file descriptor**, a number used as a shorthand in many commands and programs; usually:**

  - ♦ **file descriptor 0 is mapped to** stdin

  - ♦ **file descriptor 1 is mapped to** stdout

  - ♦ **file descriptor 2 is mapped to** stderr

- ❒ **Basically, the shell writes a prompt to** stdout **then reads from** stdin**, waiting for the user to press <Enter>**

  - ♦ **Input from the command line (**stdin**) is read, parsed, and processed, results being written to** stdout

  - ♦ **If an error is encountered, one or more error messages are written to** stderr

Computer Exercise: Getting to the z/OS Shell

At this point, you should log in to TSO. Then at either the TSO READY prompt or at ISPF option 6, issue the omvs command.

Press the F1 key (Help). Then while in Help use F4, F5, F7, and F8 to explore help about the z/OS shell, then press F3 to exit help.

Exit omvs and return to TSO or ISPF.

Next you need to be in ISPF to do the next step: from ISPF option 6 issue this command:

```
===> ex   '___________.train.library(u510strt)'   exec
```

This will invoke a small dialog to create some files we will use for later exercises. The first thing you will see is a prompt for the high level qualifier to use for the data set names; it is set to be your TSO id and this is probably OK. In any case, set the value you want and press <Enter>. At this point the files you need will be created. The file names will begin with your high level qualifier (<hlq>) followed by TR:

<hlq>.TR.LIBRARY	(PDS for data)
<hlq>.TR.CNTL	(PDS for batch jobs lab)
<hlq>.TR.LOAD	(PDS for executable programs)
<hlq>.TR.PDSE	(PDSE for executable programs)

Although this is a simple exercise, it helps us make sure everyone can get to the z/OS shell. If you have problems we can start getting help now since most problems at this point are incorrect setup by systems staff, network failures, and workstation problems.

Section Preview

❐ More Identities

- ♦ Effective UID
- ♦ Group ID

❐ More shell commands

- ♦ Shell Command Syntax
- ♦ Shell Commands: id
- ♦ OMVS - Some Options
- ♦ Shell Commands: logname, cal, date, echo, man, who, whoami, fc, history, r, alias, hash, unalias
- ♦ Practice With Commands (Machine Exercise)

Effective UID

- ❒ **In some cases, a UNIX user can change the UID they are running under, on the fly**
  - ♦ **In this case, they have an** effective UID **(the UID they are currently using) and a real UID (the UID they logged in under)**
- ❒ **For this to be allowed, the user must be a superuser or have particular permissions**
- ❒ **Similarly, some executable files can be marked as being a SETUID program**
  - ♦ **When anyone runs the program, they run under the UID of the file's owner**
  - ♦ **At this point, the user's real UID is their real UID, but their effective UID is the UID of the program's owner**

Group ID

❒ **Every UNIX system also has a set of <u>groups</u> of users**

- **Collections of related UIDs, authorizations, and permissions**

❒ **Each group is identified by a 4-byte binary integer in the range 0 - 2,147,483,647**

- **This is called the** <u>Group ID</u> **(sometimes "**<u>GID</u>**")**

❒ **Every group also has a** <u>Group name</u> **(alphanumeric)**

❒ **Every user belongs to at least one group (perhaps a department or an application group) and often many groups**

- **Access to files is at least partially controlled by what group(s) a UNIX user belongs to, as we shall see later**

❒ **As with user ID's, there can be a real GID (the GID associated at login time) and an effective GID (the GID associated with the executable being run)**

- **They are usually the same, but not always, since a program can be designated as a SETGID program**

- **When such a program is run, the effective GID is temporarily changed**

❒ **As with the user id, a <u>group database</u> is maintained that keeps track of group related information**

More Shell Commands

❒ **So far we have seen just one shell command:** *exit*

- ♦ **Not very exciting, but useful**

    - ✗ Incidentally, you can code an integer value, or expression that evaluates to an integer, between 0 and 255 on an exit command, for example: *exit 12*

    - ✗ This value is called the <u>exit value</u>

    - ✗ All commands produce an exit value, either 0 (for success) or a message number (for error or unusual completion)

    - ✗ For exit, if you don't specify an exit value, the exit value on the last command run is passed back as the exit value

❒ **There are many more shell commands, so we start to build our repertoire a little bit more; we'll look at commands that:**

- ♦ **Are related to identity and ID's (***id***,** *logname***,** *who***)**

- ♦ **Provide information (***date***,** *cal***,** *man***)**

- ♦ **Assist in command execution (***echo***,** *fc***,** *alias***,** *unalias***)**

❒ **Along the way, we'll try our hand at some of these**

❒ **But first, a word about syntax ..**

Shell Command Syntax

❒ **Typically, a command has this syntax:**

command_name **[**option(s)**]** **[**operand(s)**]**

- ♦ command_name **is case sensitive**
- ♦ **The** options**, or <u>flags</u>, usually begin with a dash(-)**
- ♦ **The number of operands depends on the command; multiple operands are separated by spaces; some commands have no operands**

❒ **In the syntax diagrams, we use brackets ([]) to indicate flags or operands that are optional**

- ♦ **For example: id [*user*]**
  indicates you can supply a user name or not
- ♦ **If an optional piece is omitted, there is always some default implied**
- ♦ **Also, don't code the brackets in these cases**
- ♦ **Unfortunately, many commands can use operands that include brackets as part of the syntax — we will be very careful to make the distinctions clear**

❒ **Terms shown in non-italicized type are keyed as shown**

❒ **Terms that are shown in** *italic* **are to be replaced by values**

- ♦ **For example: id [*user*]**
  means code a <u>value</u> for *user***, don't code the word "user"**

Shell Command Syntax, 2

- **Sometimes there is a string of flags in the brackets; this means you can code some or all of these options**

  - **For example: ls [-AabCcdEFfgHiLlmnopqRrstuWx1] which indicates you can code -A or -Aa or -Aab or ...**

    - If you want, you can code each option with its own dash, for example: ls -A -m -p

  - **The order of options is not important, as long as they all come before the operands of the command**

  - **Note that we may not always include all possible flags for any given command**

    - **We want to focus on the essentials and important features and ignore options that are likely not of interest or use for the applications programmer**

    - **However, we will tend to err on the side of completeness**

  - **In the command write-up, the following narrative will explain any special restrictions on various combinations**

Shell Commands: id

❒ **The id command returns the UID, user name, GID, and group name for a UNIX user**

Syntax

```
    id    [user]
or
    id    -G [-n] [user]
or
    id    -g [-nr] [user]
or
    id    -u [-nr] [user]
```

Where

- ♦ *user* **is the name of the user you are inquiring about (if omitted, it returns your information)**
- ♦ **"G" indicates return all group IDs (real and effective) while "g" indicates only return the effective group ID**
- ♦ **"u" indicates only return the effective UID (default is to return both real and effective)**
- ♦ **"n" indicates return the name only (not the ID)**
- ♦ **"r" indicates return only the real ID, not the effective**

Shell Commands: id, p.2.

Notes

- **So the entire list of valid commands without a user name is:**

```
id
id     -G
id     -Gn
id     -g
id     -gnr
id     -u
id     -unr
```

Let's Do It!

Computer Experiment: Test the id command (postpone if not possible)

If you are not already in omvs, get there.

Issue the above commands.

Issue the same commands using someone else's user name.

Issue an invalid form, such as: `id -Gr` What happens?

OMVS - Some Options

❒ If you enter a command with an invalid format, UNIX comes back and tries to give you a clue about the usage of the command, looking something like this:

```
Usage: id Ÿuser"
   id -G Ÿ-n" Ÿuser"
   id -g Ÿ-nr" Ÿuser"
   id -u Ÿ-nr" Ÿuser"
```

❒ If your screen looks like this, the problem is code page based

❒ A code page defines the character assigned to each code point (hex value) that can be found in a byte

- ♦ When a byte is sent to a screen or a printer, the current code page determines what is seen

❒ Even though all mainframe work is done using the EBCDIC coding scheme, different code pages are used for different languages

- ♦ The default code page for z/OS is usually IBM-037

- ♦ But the default code page for the z/OS UNIX Shell is IBM-1047

❒ There are 13 characters that are variant across IBM mainframe code pages: the hex_value-to-graphics mappings are not the same for all code pages for these characters: { } \ [ ] ^ ~ ! # | $ @ ` ("accent grave" or "backwards apostrophe" or "backquote")

OMVS - Some Options, continued

❒ **There are several ways to fix this problem**

- **Try changing the code page used by your emulator (some 3270 emulator programs let you do this)**

- **Try remapping just specific characters that cause you problems with your emulator**

- **Try changing your terminal type on the ISPF Settings panel**

- **Invoke *omvs* with the CONVERT option, specifying a code page conversion table (in the US, usually BPXFX111)**

  ✗ You do this as part of issuing the omvs command:

  ```
  ==> omvs convert((bpxfx111))
  ```

- **Note the double set of parentheses; the outermost indicates a parameter value is being assigned to the convert option, the innermost indicates the name is a member in the default search order for system libraries (the link list)**

Let's Do It!

Computer Experiment: Solve code page problem

If you had a code page problem, exit out of omvs, and try the various options described above; retry the bad id command:

```
id -Gr
```

to see if the messages are easier to read now.

OMVS - Some Options, continued

❒ **Another strange default IBM supplies is to use the cent sign (¢) as the default escape character - "strange" since this character is not found on most workstation keyboards**

❒ **As with code pages, there are several alternatives here**

- ♦ **Remap your keyboard to be able to enter the cent sign**

- ♦ **Add the ESC option on the *omvs* command**

  - ✗ You may specify up to eight characters, bounded by single quotes, no spaces; any of these characters may be used for an escape character

  - ✗ Do not choose any characters that might be part of your data or commands, if possible

    - ➢ For example:

```
omvs   esc('^@')
```

- ♦ **Assign a function key to the Control *omvs* subcommand:**

```
omvs   pf2(control)
```

<u>Note that these may be combined with the convert option</u>

OMVS - Some Options, continued

❒ **More about escape characters**

- **There are two main uses for escape characters**
  - ✗ 1) To send control signals to the shell, a program, or a command
    - ➢ Control signals might include requests to pause, resume, cancel, raise end of data, and so on
      - ➤ Discussed on the next page
  - ✗ 2) To send special characters and have them interpreted as the characters instead of their special use
    - ➢ Discussed later
- **For a variety of reasons we don't want to get into here, users accessing TSO using an emulator cannot use the <u>Esc</u> or <u>Ctrl</u> keys on their keyboards as their escape key**
  - ✗ You must either designate a function key to have the value of 'control' or specify a key with an assigned glyph (that is, alphanumeric or punctuation) as your escape character
    - ➢ Then use assigned "escape sequences" to send control signals to the shell or a running program

OMVS - Some Options, continued

Escape sequences

- **In the literature you will see references to <EscChar-D> or <Ctrl-D>, for example**
  - ✗ If you have assigned an escape character to a keyboard character, you enter that character followed by the character specified in the reference
    - ➢ For example, if you invoked omvs as **omvs   esc('^@')**, then if you enter the string: ^D on the keyboard
    - ➢ When you press <Enter>, these two characters are converted to the equivalent escape sequence
  - ✗ If you have assigned the control command to a function key, enter the character and press the function key
    - ➢ For example: D <pf2>
    - ➢ At that point, the equivalent escape sequence is passed to the program

❐ **Note that escape sequences are not case sensitive**

- **These escape sequences are useful at this point:**
  - ✗ <Ctrl-c> - cancel the current work
  - ✗ <Ctrl-d> - exit the session
  - ✗ <Ctrl-z> - suspend the current work; resume by entering the *fg* command (details later)

Shell Commands: logname

❐ This command returns the user name of the person issuing the command

Syntax

```
logname
```

♦ The name is returned in all upper case, regardless of how the person logged in or how the name is defined to UNIX, TSO, or the security package

Shell Commands: cal

❐ **The cal command displays a calendar for a specific month**

Syntax

cal [*month*] [*year*]

Where

- **With no arguments, *cal* displays a calendar for the current month of the current year**
- **If one argument is given and it is numeric, *cal* interprets it as a year (for example, 2012)**
- **if a single argument is not numeric, *cal* interprets it as the name of a month, possibly abbreviated (for example, apr)**
- **If two arguments are given, *cal* assumes that the first argument is the month (either a number from 1 to 12 or a month name) and the second is the year**

❐ **Year numbers less than 100 refer to the early Christian era, not the current century**

❐ **This command uses the Gregorian calendar, handling September 1752 correctly**

- **Many cultures observe other calendars.**

Shell Commands: date

❐ **This command returns the current date and time as far as the operating system knows**

Syntax

date [-cu] [+*format*]

Where

- **"c" displays the time using Greenwich Mean Time, referring to it as "CUT" (for Coordinated Universal Time)**
- **"u" displays the time using Greenwich Mean Time referring to it as "GMT"**
- **If neither "c" nor "u" is specified, the time zone choice is determined by an environment variable called TZ; (more on environment variables soon)**
- *format* **is a format string specifying how you want the date formatted**
- **Details on following page**

Shell Commands: date, p.2.

date format picture strings

- %A - weekday name (*e.g.*: Sunday)
- %a - weekday abbrev. (*e.g.*: Sun)
- %B - month name (*e.g.*: August)
- %b - month abbrev. (*e.g.*: Aug)
- %C - century (00-99)
- %c - local date / time format
- %D - date as mm/dd/yy
- %d - day of month (01-31)
- %e - day of month (ƀ1-31)
- %H - hour (00-23)
- %h - same as %b
- %I - hour (01-12)
- %j - day of year (001-366)
- %M - minute (00-59)
- %m - month number (01-12)
- %n - newline character
- %p - local "am" or "pm"
- %r - time with am or pm
- %S - seconds (00-61)
- %T - time
- %t - tab character
- %U - week in year (00-53) (start w/ Sunday)
- %W - week in year (00-53) (start w/ Monday)
- %w - weekday number (Sunday is 0)
- %X - time
- %x - date
- %Y - year (4 digits)
- %y - year (2 digits)
- %Z - time zone name
- %% - a "%" character

- Note that some of these formats vary depending on current settings (locale, environment variables, etc.)

Shell Commands: date, p.3.

❒ **If you omit a format, a default format is used**

❒ **If you specify a format, build it up with format strings:**

- ♦ **Start with a "+"**
- ♦ **Include one or more format pictures**
  - ✗ If you have more than one, the whole string must be bounded by single quotes, for example

```
date '+%d  %B'
```

- ♦ **Anything besides a recognized format picture is displayed as is**
- ♦ **The date routine processes the format string from left to right until the string is exhausted**
- ♦ **The "+" can go before or after the leading quote**

<u>Another example</u>

```
date +'Today is %A.'
```

displays:

```
Today is Monday.
```

perhaps

Shell Commands: echo

❐ **The** echo **command writes a string to standard out (usually the terminal)**

Syntax

echo *argument*

Notes

♦ **The** *argument* **can be any string, including any of the following C-style escape sequences**

✗	\a	sound bell, if present
✗	\b	backspace
✗	\c	remove any following characters
✗	\f	form feed
✗	\n	newline
✗	\r	carriage return
✗	\t	horizontal tab
✗	\v	vertical tab
✗	\0nnn	The byte with the numeric value specified by the octal value
✗	\\	Backslash

Shell Commands: echo, p.2.

Notes, continued

- **If any escape characters are contained in the argument string, the string must be enclosed in single quotes or double quotes, for example**

```
    echo 'test \f form feed'
or
    echo "test \f form feed"
not
    echo test \f form feed
```

- **These escape sequences won't work if you established '\' as one of your escape characters on the omvs command**

  ✗ *e.g.*:   omvs esc('\')

Let's Do It!

Computer Experiment: Test the echo command (postpone if not possible)

Under omvs, issue these commands:

```
echo this is a test

echo "this is a test"

echo 'test \f form feed'

echo 'test \f form feed and \v vertical tab'
```

Shell Commands: man

❒ **The** man **command displays information from the UNIX online manuals (hence the name)**

Syntax

man [-wx] [-M *path***] [***section***]** *entry* **...**

or

man -k [-M *path***]** *keyword* **...**

Where

- **"w" means only display the name of the file where any located information is found**
- **"x" means list the files being searched for** *entry*
  - ✗ The files searched are determined by the environment variable MANPATH as well as some filename-generating logic; files that are found are searched for a topic containing *entry*
- **"M" restricts the search to only manuals found in** *path* **(paths are discussed later)**
- *section* **is a number 0-9 indicating what section of the help docs to search; for z/OS UNIX, only one section is provided (1 - commands)**
- **"k" indicates you are searching pre-generated list for keywords, and you include those keywords as operands**
  - ✗ If you have multiple words, the search is for any entry containing either of them; to find a phrase, put the phrase in quotes or apostrophes

Shell Commands: man, p.2.

❒ **If the output is extensive, the** man **command sends the output through a pager command, which paginates the output in page sized chunks**

- ♦ **See the** ***pg*** **command later**

❒ **To end the display of a** ***man*** **page, type q then press <Enter>**

❒ **If you want to see information on the TSO UNIX commands, prefix the command name with 'tso'**

- ♦ **For example,**

```
man     omvs
```

- ♦ **will not return any information, but**

```
man     tsoomvs
```

- ♦ **will find information**

❒ **The other TSO UNIX commands are discussed later**

Shell Commands: who

❐ **This command returns information about UNIX users currently logged in**

- **The information comes from a default accounting file, or one that you specify**

Syntax

```
    who [-HmTu] [file]
or
    who -q[file]
or
    who am {I|i}
```

Where the flags control what information to display:

- **"H" - include headers on the display**
- **"m" - display information about current terminal only**
- **"T" - terminal status ('+' - terminal allows write access to other users; '-' - terminal does not do this)**
- **"u" - show idle time for terminal (shows '.' if terminal has been used in the last minute), and show PID (process ID)**
- **"q" - quick list; just user names and count**
- **"who am I" or "who am i" returns just your information**

❐ **There are some other options, but they are extensions to the POSIX standard**

Shell Commands: whoami

- ❐ **This command displays a user name associated with your effective UID**
  - ♦ **Usually this is the same as your user name**

  Syntax

  ```
  whoami
  ```

- ❐ **Remember, your effective UID could be the UID of a program's owner**
  - ♦ **And any UID could have more than one user name associated with it**
    - ✗ For example, UID 0 (superuser) almost always has several user's assigned to it

- ❐ whoami **is an extension to the UNIX standards, so it is not portable**

Shell Commands: fc

- **You may be set up to keep a history file of commands**
  - **This file may be permanent or temporary**
  - **If the file is temporary, it is created fresh each session**
  - **If the file is permanent, the latest lines are appended each session (there may be a maximum number of lines specified for the file; if that size is exceeded, truncation may occur)**

Shell Commands: fc, p.2.

❒ **The** fc **command lets you work with the history of commands you have entered**

- ♦ **"fc" stands for "fix commands", for what it's worth**

Syntax

fc -l [-nr] [*first* [*last*]]

or

fc -s [*old*=*new*] *spec*

Where

- ♦ **"l" produces a numbered list of commands from the history file, from line number *first* to line number *last***
  - ✗ If *last* is not specified, the value of *first* is used
  - ✗ If *first* is not specified, begin with the first command in the current session
  - ✗ "n" lists the lines with no line numbers
  - ✗ "r" produces the list in reverse order

Shell Commands: fc, p.3.

<u>Where, continued</u>

- **"s" recalls the specified command, optionally changes part of it, then executes it**

  - ✗ "spec" is an unsigned number (meaning the command on that specified line), a number prefixed by a minus sign (the command that many lines before the current line), or a string (the first command containing that string); examples:

```
fc   -s   293
fc   -s   -5
fc   -s   tsoomvs
```

  - ✗ If spec is not coded, the default is the most recent command entered

  - ✗ "old=new" indicates that the first occurrence of string *old* in the retrieved command is to be replaced by the string *new* before the command is executed, for example:

```
fc   -s   stnt329=stnt330   -2
```

- **Note that "fc" with no operands (or certain combinations of operands) puts the command lines into an editor**

  - ✗ Since we don't discuss UNIX editors in this course, we won't explore this further; if you find yourself at a question mark prompt (?), enter q to quit the editor

Shell Commands: history, r

❒ **The *history* and *r* commands are <u>command aliases</u>: alternate ways of coding a complete command**

- **coding *history* is the same as coding** fc -l

  - ✗ You can also include the same [*first* [*last*]] operands

  - ✗ This might be a more natural way to remember the command

- **coding *r* is the same as coding** fc -s

  - ✗ You can also include the same [*old*=*new*] *spec* operands

  - ✗ Which might be easy to remember ("r" for "recall last command"), as well as saving keystrokes

Shell Commands: alias

❒ **The** alias **command lets you assign one string as equivalent to another; usually the second string is a command**

- **When** name **is entered as the <u>first word</u> on the <u>command line</u>, or in a <u>shell script</u>, the equivalent string (the** *value***) is substituted**

<u>Syntax</u>

```
alias   [-tx] [name ...]
alias   [name[=value] ... ]
alias   -r
```

<u>Where</u>

- **<u>t</u> requests the alias to be a <u>tracked alias</u>: the shell will keep track of the full pathname to avoid future checking of the PATH directories to find an alias when it is referenced later**
- **<u>x</u> marks the alias name to be exported to shells that will run scripts (discussed later)**
- **<u>r</u> removes all tracked aliases from the list**
- *name* **is the alias you are creating; if omitted, display all current tracked (-t) or untracked (no -t) aliases**
- *value* **is the string to equate to** *name***; if =***value* **is omitted, display current value of** *name*
  - ✗ If *value* has special characters, bound it in single quotes or double quotes; If the string has a backslash, precede the backslash with another backslash; if such a string is double quoted precede such double backslashes with another backslash (or just use single quotes instead of double quotes)

Shell Commands: alias, p.2.

❒ **You can use a command name for an alias name, for example**

```
alias   id="id  -G"
```

- ♦ **Now, whenever you issue the id command, you get the id -G format automatically**

❒ **If you want to override your alias for one time, you can Escape the alias by prefixing it with a backslash:**

```
\id  -g
```

Notes

- ♦ **Use tracked aliases for frequently referenced alias commands**
- ♦ **Do not code spaces around the equals sign (=)**

More examples

```
alias Howdy='echo Hello'
```
- may use single quotes or double quotes here

```
alias Adios="echo Don\'t go yet"
```
- need the backslash to escape the single quote

Shell Commands: hash

❒ **The** hash **command lets you identify an alias as a tracked alias**

Syntax

```
hash [name ... ]
hash -r
```

Where

- ♦ **Each** name **is an already defined alias**

- ♦ **If no** *name***(s) is/are specified, you get the list of tracked aliases currently in effect**

- ♦ **r says remove all tracked aliases**

❒ hash**, itself, is defined as a built-in alias:**

```
alias   hash='alias -t'
```

Shell Commands: unalias

❒ **To remove an alias entirely, issue the** unalias **command**

Syntax

unalias *name*

Example

```
unalias   id
```

Computer Exercise: Practice With Commands

Take some time and explore the commands we've talked about. Try different options and combinations.

For reference, in this section we discussed:

id	(but we already tried all the options)
logname	
cal	
date	
echo	
hash	
man	
who	
whoami	
fc	(and its aliases "history" and "r')
alias	(and its alias "hash")
unalias	

So now...

* display all current aliases
* set an alias of "Now" to be a date string that formats as

  Today is _month_ _day_of_month_, _year_; it is now _hour_ : _min_

hint: get the date command right first; then set up the alias

hint: the date format string will require single quotes, so the alias string will require double quotes

hint: no spaces around the '=' sign in the alias command

note: command names are case sensitive (note that it is Now)

* issue the command Now
* display all current aliases

**** more ****

Computer Exercise: Practice With Commands, p.2.

Issue the command

```
man tsoomvs
```

Press <Enter> to step through a couple of pages of output, then:

Interrupt the work (Ctrl-z)

issue a date command

resume the interrupted work (fg command) and observe that you pick up where you left off

remember, to exit a 'man' display early, key: q<Enter>

Once you are out from the man command, enter the following string on the command line:

```
echo "here is" ; Now ; id
```

press <Enter>. This is to demonstrate you can string multiple distinct commands on the command line, separated by semi-colons. This is called a command string. All the commands will be run in parallel (although these commands end so fast you can't really tell the difference here between running at once and running one after the other.)

Press key F12; this is omvs 'retrieve' subcommand; nicer than fc, right?

To exit your session this time, use the escape sequence Ctrl-d.

When you are done, take a break.