2 - Unix Shells

- A shell is the user interface or set of programs user to interact with Unix and process commands
- Common shells are:

		Prompt	Auto
– Bourne	sh	$	x
– Bourne again	bash	~/rel. path$	√
– C	csh	~/rel. path>	x
– Korn	ksh	not on cs1	

- When a shell is executed, it
  - inherits environmental variables set by the previous shell
  - stops the previous shell
  - becomes the current shell
- Can change shell from by issuing
  - Short shell name : mentioned above sh, bash etc.
  - Or issuing chsh command followed by shell name

Bourne Shell (sh)

- Written by Dr Steven Bourne of Bell Labs
- Both a command interpreter and a high-level programming language
- Typically the default Unix shell
- Fast
  - Approximately 20 times faster than C shell because it is simpler and doesn't carry as much baggage

Korn Shell (ksh)

- Written by David Korn of AT&T, released in 1986
- Includes features of both the Bourne and C shell
- Introduces several new user interface features including command line editing
- Adds features that improve its usefulness as a programming language
  - report formatting capabilities
  - built-in arithmetic
  - data types

CLI shell families - Bourne derived

- sh - original Bourne - after developer
  - About 20 times faster than C-shell :
  - fewer user friendly facilities
- bash - Bourne again shell – the default on cs1
  - the most common & powerful of all,
  - since it incorporates some better 'C-family' shell features
- Other backwardly compatible extensions (availble but not installed on cs1)
  - ksh - Korn shell - again after developer
  - pdksh - public domain Korn shell
    - Command line editing
    - Built-in arithmetic data types
  - mksh - MirKorn shell
    - actively developed successor to pdksh
  - zsh - Z shell – good for interactive use…& script dev

C Shell (csh)

- Written by Bill Joy at University of California at Berkeley
- Slower and more complex than Bourne shell, but has facilities to make it more user friendly
  - Alias
    - for files - saves space
      - one file, many links in different directories
    - for commands - saves learning
      - Uses similar command names as in other system
  - History - saves re-typing commands
    - previously typed commands can be repeated and/or edited (by use of up arrow & edit by backspace/delete and retype – variations exist!)
  - job control
  - filename completion - saves typing

Superuser

- A user with essentially all privileges
  - Often referred to as *root* privileges
- Allowed access to all files
- Allowed to run all commands
- System administrator has superuser privileges
- Accessible by
  - Login
  - Sudo (super user do following command)

Superuser

- Can be very dangerous, all the Unix built in protections are by-passed
- FIRST COMMANDMENT of UNIX
  – DO NOT RUN AS SUPERUSER / ROOT
  – Too dangerous - since system thinks you are 'god', will do what it's told, won't check, thinks you are incapable of error!
    - You might think you know what you are doing
    - You can make a mistake in a command
    - You might encounter some quirk in a command or shell
    - You might run a dodgy script
  – Better to run sudo (**su**peruser **do** 'one at a time') commands
  – Even experienced administrators avoid root & do sudo…
  – Because all made the mistake of running as root once, and never again…unless…they think they're god!
  – Everyone knows it, states it, broke it and got burned!

Password Security

- Unfortunately, password security is a necessity
- Good passwords have several characteristics
  – Minimum of six (6) characters
  – Mixture of alphabetic (upper & lower case) and numeric
  – No real words, names etc. : which are susceptible to a dictionary attack
- Avoid personal information which may be inferred from facebook etc.
  – Family names, Birthdates, Pet's names, other personal data
- Generally good to change passwords frequently – unless trivial incremental change
  – e.g. don't change 'abcde123' to 'abcde124', or 'bcdef123', 'abcde123' etc. as patterns are a weakness in a decryption attack.
- Tricks…
  – Tr1ck5 – no longer a good trick as it is so common, that system will warn dictionary
  – Tayratoot …**t**ie **a** **y**ellow **r**ibbon **a**round **t**he **o**ld **o**ak **t**ree
  – But perhaps not a repetitive chorus…no matter how poignant

Did they beat the drums slowly, did they play the fife lowly,
Did they sound the death march as they lowered you down…
Did the band play the last post and chorus.
Did the pipes play the flowers of the forest.

Do it your way!

- My own suggestion…which some Sys Admin seemed to approve:
  – first/last letters of consecutive words of a phrase
  – e.g. forward (could also be in reverse) : HDsoaw ..or yytnal; etc for "Humpty-Dumpty sat on a wall"*,
  – otherwise Humpty-Dumpty might have a great fall!
- But try not to use a popular lyric, phrase, poem…
- Note 11/1/11: German security researcher 'cracked' Wi-Fi WPA-PSK passwords (usually fairly long ~ 20-30 alphanumeric characters!) in about 10 mins using Amazon EC2 (Elastic Compute) Cloud for ~ €1 cost to him!
  – http://www.blackhat.com/html/bh-dc-11/bh-dc-11-briefings.html#Roth
- So security is not all IT is 'cracked' up to be! (Terrible pun!)

Or in French…"Un petit, d'un petit, s'étonne aux halles!?"

Logging In and Out

- login: *username*
- password: *your_password*
  – To change your password
    - machinename:pathname% *passwd*
    - Changing password for ??? on urmac.
    - Old password: *your_current_password*
    - New password: *your_new_password*
    - Retype new password: *your_new_password*
    - machinename:pathname %
- To logout
  – machinename:pathname % *exit*
  – or machinename:pathname % *logout*

MANual pages - man

- man - Access inbuilt documentation

or

- find reference pages by keyword

Syntax: man [[section] title]
or: man [[-k keyword] | [-f filename]]

- Examples:

```
urmac % man ps
urmac% man -k compile
urmac % man -f /var/spool/mail
urmac % man link
urmac % man 2 link
urmac % man 8 link
```

apropos & whatis

apropos

•Locate commands by keyword lookup
•Syntax: *apropos keyword*
•*Example:*

```
urmac % apropos compiler
```

•Note: *apropos* is no different than *man –k*

whatis

•Displays a one line summary about a command
•Syntax: whatis command
•Example:

```
urmac % whatis vi
```

The Command Prompt

Commands are the way to “do things” in Unix
A command consists of a command name and options called “flags”
Commands are typed at the *command prompt*
In Unix, *everything* (including commands) is case-sensitive

```
[prompt]$ <command> <flags> <args>
```

```
urmc:/filepath $ ls -l -a unix-tutorial
```

Command Prompt Command (Optional) flags (Optional) arguments

Note: In Unix, you’re expected to know what you’re doing. Many commands will print a message only if something went wrong.

Command Components

- Commands consist of:
  - Command name
  - Options, or flags
  - Arguments
- Arguments are the “things” that the command will operate on
- Options modify the behavior of the command

Command Notation

- command name
- [ ] optional arguments or options
  - can be nested
    - [arg1 [arg2]]
    - [...]
- | is an OR condition
  - arg1 | arg2
- Note [,], and | all from regular expressions!

Two Basic Commands

- The most useful commands you’ll ever learn:
  - `man` (short for “*man*ual” - more like vim editor, q & h)
  - `Info (uses emacs editor to move around, inbuilt help)`
  - `Pinfo - if installed, is easier and colourful!`
- They help you find information about other commands
  - `man <cmd>` or `info <cmd>` retrieves detailed information about `<cmd>`
  - `man -k <keyword>` searches the man page summaries (faster, and will probably give better results)
  - `man -K <keyword>` searches the full text of the man pages (so every irrelevant reference is likely to be included too!)

```
$ man -k password
passwd     (5)  - password file
xlock      (1)  - Locks the local X display
                 until a password is entered
urmc:/filepath$ passwd
```

Two Basic Commands (info)

- Info, as opposed to man, is category based
- Documents are hyperlinked
  - `info <cmd>` retrieves detailed information about `<cmd>`
  - `info` by itself will give instructions on its usage
  - Type `q` to quit.
  - Type `h` for help.

What happens when you login?

- Unix runs the *login* program
  - If it exists, .login script is executed
- Then the default shell specified in /etc/passwd is executed to set your preferences and environment and user is placed in his HOME directory
  - *.cshrc* (or *.bashrc*) is executed depending on whether csh (or bash) is your default shell)
  - Others are run for different shells
- So what’s the difference? Why two initialization files?
  - *.login* is executed **only** at login
  - *.cshrc (or equivalent for another shell)* is executed **every** time a new shell is spawned
- NB Don’t confuse
  - the shell change cmd *‘chsh’*
  - *With the .cshrc*

.login file in cs1 ~ 2010

Just for show, don't think you have to understand or learn it now, but should by end of course!
Note also that it is based on old DEC Unix...from ages ago.. if it isn't broken, why fix it!?

```
# *    Copyright (c) Digital Equipment Corporation, 1991, 1996    *
# @(#)$RCSfile: .login,v $ $Revision: 4.1.7.3 $ (DEC) $Date: 1995/10/25 20:03:52 $
#
if ($?path) then     # sets PATH variable … where to find commands
    set path=(/usr/local/bin  /usr/X11R6/bin  $HOME/bin  $path
                                              /users/coursework/cs1100/bin)
else
    set path=($HOME/bin   /usr/bin   .)
endif
if ( ! ${?DT} ) then                          # sets terminal specs
            stty dec new
            tset -I -Q
endif
set prompt="`hostname`> "                     # sets command prompt
set mail=/usr/spool/mail/$USER
```

exit or logout

- *exit* terminates your current shell
  - if it is also your login shell, exit will exit and logout
- *logout* terminates a login shell

who

- Who is on the system and info about their login
  - User_name, terminal_name, login_time
- Syntax: *who [am i]*
- Example:

```
Urmac:pathname% who
rootconsole Jul 22   09:36
krf           ttyp0          Aug 7     10:32   (dal-tx2-33.ix.ne)
```

- *am i* option lists requestor info only
- Related command: *whoami*
  - Lists requestor's user_name only
  - Why bother – surely you know who you are!?
    - Historical : identifies owner process on vacant terminal!
    - Useful in scripts for taking user-specific actions.

.cshrc

Just for show, don't think you have to understand or learn it now, but should by end of course!
Note also that it is based on old DEC Unix...from ages ago.. if it isn't broken, why fix it!?

```
set path = (. ~ $lpath ~/bin /usr/local /usr/ucb /usr/bin /usr/etc)
set noclobber
#        aliases for all shells….-i for interactive…check before exec to be sure
alias cd          'cd \!*;echo $cwd'
alias cp          'cp -i'
alias mv           'mv -i'
alias rm          'rm -i'
alias pwd          'echo $cwd'
#alias del          'rm -i'
set history=40
set ignoreeof
#set notify
#set savehist=40
#set prompt="% "
#set prompt="`hostname`{`whoami`}\!: "
#set time=100
```

Unix – social networking etc. ~ 1970

- Who - is associated with a terminal or process
- Finger - to find out more about a user
- Whois - an IP address or web domain name
- Cal - timing is everything
- Date
- Mail - supports most GUI mail clients
- Talk - sets up a 2-frame chat session
- Write - just writes on the other's screen!
  » Great prank to mess anothers output!
- Mesg - blocks others' interruptions…talk or write

finger

- Displays information about users
- Syntax: *finger [options] user_name*
- By default, finger displays *user_name*'s:
  - login name (& therefore Unix email addr)
  - full name
  - terminal name
  - idle time
  - login time
  - location
  - first line of .plan & .project files
    - These are personal files which contain info. The user is happy to make public… like old-timers blog..!?

finger options

- -m Match arguments only on user name (not first or last name)
- -l Long output format
- -s Short output format
- -q Quick output format, only the login name, terminal, and login time are printed
- -i "idle" output format, prints only the login name, terminal, login time, and idle time.
- -b Suppress printing the user's home directory and shell
- -f Suppress printing the header
- -w Suppress printing the full name
- -h Suppress printing of the .project file
- -p Suppress printing of the .plan file

CALendar - cal

- Prints a calendar for the specified year
  - Default is current year
- Syntax: cal [[*month*] *year*]
  - *month* - number from 1 to 12
  - *year* - number from 1 to 9999
- Note: September 1752 is odd, 11 days were skipped to make up for lack of prior leap year adjustments
- Examples:

  Urmac:pathname % *cal*

  Urmac:pathname % *cal 9 1752*

mail

Read or send electronic mail messages

Syntax, for reading mail:

– *mail [-deHinNUv] [-f [filename* | *+folder]] [-T file]*

or for sending mail:

- *mail [-dFinUv] [-h number] [-r address] [-s subject] recipient ...*

- Restrictions may apply on this teaching system, so these may not work:
  - Read mail: *mail*
  - Send mail: *mail -s "subject" recipient*
  - or: *mail -s "subject" recipient < file_name*

Whois – but not on our system.

- Internet 'white pages'
  - Searches for a TCP/IP directory entry…143.239…
  - Used to find people or domain owners/contacts
- Syntax: who [-h host] identifier
  - host - name of host computer to use for lookup
    - Default is nic.ddn.mil, which no longer supports anything but MILNET, Current host is rs.internic.net
    - Names are stored as
      - last name, first name, titles
- Examples:

  urmac % *whois -h rs.internic.net Earthlink.com*

  urmac % *whois -h rs.internic.net 'Krol, Ed*'*

date

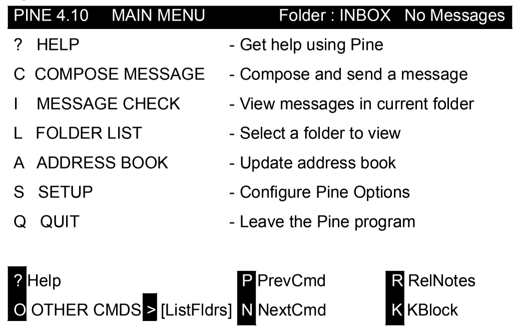
- Display or set the date
- Syntax:*date [-u] [-a [-] sss.fff] [yymmddhhmm [.ss]]*
  - u - display date in GMT, default is local time
  - a - slowly adjust system clock
  - yymmddhhmm [.ss] - set system date and time
    - Only superuser can set the date and time
- If the argument begins with a +, the output of date is under user control
- Examples:

  Urmac:pathname % *date*

  Urmac:pathname % *date -u*

  Urmac:pathname % *date +%T*

pine – more advanced mail client… still used by diehards



```
PINE 4.10   MAIN MENU              Folder : INBOX   No Messages

?  HELP               - Get help using Pine
C  COMPOSE MESSAGE    - Compose and send a message
I  MESSAGE CHECK      - View messages in current folder
L  FOLDER LIST        - Select a folder to view
A  ADDRESS BOOK       - Update address book
S  SETUP              - Configure Pine Options
Q  QUIT               - Leave the Pine program

? Help                      P PrevCmd       R RelNotes
O OTHER CMDS  > [ListFldrs] N NextCmd       K KBlock
```

Talk - Unix instant messaging

- Talk to another user
- Syntax: *talk username [ttyname]*

  username - login name if on the same machine, username@machinename if on a different machine

  ttyname - login session to use if username is logged in more than once

talk Example

```
urmac % talk userid
Message from Talk_Daemon@urmac at
   17:39 ...
talk: connection requested by
   userid@urmac.urplace.ie
talk: respond with:  talk
   userid@urmac.urplace.urland
```

- Respond as shown and a split screen will be displayed with your input in one half and your talk-mate's output in the other
- To end the session, type Control-c

MESsaGe - mesg

- Permit or deny messages on your terminal
- Syntax: *mesg [n] [y]*
  - default reports current state without changing it
  - *n* - forbids messages to be sent to you from talk or write
  - *y* - reinstates permission

write

- Write a message to another user
- Syntax: *write username [ttyname]*

  username - login name of the message recipient

  ttyname - terminal name if user is logged in more than once

  Whatever you type is then copied, line by line, to the recipient's terminal until you enter on EOF (control-d)

write Example:

Again restrictions might apply in this installation:-

```
urmac% write userid
Hi there!
Heard any good Unix jokes lately?
Control-d
```

On *userid*'s terminal, the following appears:

```
urmac%
Message from krf@urmac on ttyp1 at 17:50 ...
Hi there!
Heard any good Unix jokes lately?
urmac%
```

Process Concepts

- Early or simple (DOS) computer systems allowed only one program to be executed at a time
- Modern computer systems like Unix are *multi-tasking*, allowing multiple programs to be loaded and executed concurrently
- This requires more control of programs, leading to the notion of a *process*
- A *process* is a program in some stage of execution
- A modern computer system is a collection of processes
  - Operating system processes and User processes

Unix Processes

- Every command executed on Unix is a process
- Unix processes are hierarchical
  - Parent and child processes
- Every process is automagically assigned three standard files
  - input
  - output
  - error

Process Blocks

- Each process has a process block (admin record) that includes
  - a process identifier – pid – successive numbers..
  - the process state
  - the value of the process's program counter
  - other information specific to the process such as
    - memory limits
    - files in use
    - processor time used
    - ....

Unix Process Management

- Processes are managed and executed by a *scheduler*
- Scheduler simulates simultaneous process execution by:
  - Sharing CPU by time-slicing and giving each active process a time slot
  - May require paging and swapping as processes are activated or de-activated
    - Paging is a function of virtual memory that simulates a large, virtual memory map
    - Swapping is the process of moving a process out of memory to disk to free memory for another process

show ProcesSes - ps

- Displays the status of current processes
- Syntax: *ps [[-] acCegjklnrSuUvwx] [-tx]* | *[num] [kernel-name] [c-dump-file] [swap-file]*
- Default is to display only processes with your effective user ID
- We will only use the following options:
  - *a* - include processes that are not owned by you
  - *x* - show processes that don't have a controlling terminal
  - *r* - show only running processes
  - Example:urmac %

```
$ ps -a
  PID   TTY        TIME           CMD
16698 ttys000   0:00.09        login -pf JamesDoherty
16699 ttys000   0:00.01        bash
16702 ttys000   0:00.00        ps -a
```

Note
Fieldnames
PID's rising
Time passed
Command

Kill a process…

- Processes can be stopped with a kill pid command
  - Essential if a process hangs for whatever reason…else it could be hanging around for ages taking up resources and slowing the system.
  - Some systems will 'shelve' if not terminate processes which overrun their fair-use policy, but that too has problems, if it is a critical sender/listener job!

```
$ ps -a
  PID TTY          TIME CMD
16698 ttys000    0:00.09 login -pf JamesDoherty
16699 ttys000    0:00.01 –bash
16702 ttys000    0:00.00 ps –a

$ kill 16702
-bash: kill: (16702) - No such process

Kill  9      KILL (non-catchable, non-ignorable kill)
```

Print Working Directory - pwd

- Displays the pathname of the current working directory
- Syntax: *pwd*
- Example:

  urmac % *pwd*
  /home/urmac/urid
  urmac %

Disk Free - df

- Reports amount of free disk space on file system
- Syntax:
  df [-a] [-i] [-t type] [filesystem...] [filename…]
  - default is to report on all mounted file systems
  - *a* - report on all file systems, including “uninteresting” ones with zero total blocks
  - *i* - report number of used and free inodes, print * is no information is available
  - *t type* - report on file systems of a given type, such as NFS
  - *filename* - report space used by the file system containing *filename*

df Examples

```
urmac% df

Filesystem    kbytes     used     avail   capacity  Mounted on
/dev/sd0a      30807      6462   21265    23%    /
/dev/sd0g     204535    180363    3719    98%    /usr
/dev/sd0h     239391    180222   35230    84%    /home
/dev/sd0f     425767    296019   87172    77%    /usr/local

urmac % df my_file

/dev/sd0h     239391    180222   35230    84%    /home
```

Disk Utilized - du

- Displays the number of disk blocks used per directory or file
- Syntax: *du [-s] [-a] [filename]*
  - s - only display grand total for each of the specified file names
  - a - display a value for each file

du Examples

```
urmac% du -a
3      ./.cshrc
3      ./.login
1      ./.sunview
3      ./.rootmenu
6      ./ttg_public
1      ./.plan
1      ./mail
10      ./.pinerc
1      ./.project
3      ./mbox
33      .
urmac% du -s
33      .
urmac%
```

Set TTY – stty (tty is short for terminal!)

- Display and set terminal options
  - Was useful in old times to make one terminal behave as another,
    - either across manufacturers
    - Or according to your own terminal or system preferences;
  - Some modern ssh/ftp clients still support it (e.g. PuTTY)
- Syntax: *stty [-ag] [option]* ...
  - *a* - Report all option settings
  - *g* - Report current settings in a format that can be used as an argument to another stty command\
- Most common user use is to remap control keys, such as erase
  - To set the erase command to be a backspace:
    urmac% *stty erase ^h*
  - Key mapping good candidate for inclusion in your .login file

stty Examples

```
urmac% stty
speed 38400 baud; evenp
-inpck imaxbel -tabs
iexten crt
urmac% stty -a
speed 38400 baud, 0 rows, 0 columns
parenb -parodd cs7 -cstopb -hupcl cread -clocal -crtscts
-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr icrnl -iuclc
ixon -ixany -ixoff imaxbel
isig iexten icanon -xcase echo echoe echok -echonl -noflsh -tostop
echoctl -echoprt echoke
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel -tabs
erase  kill   werase rprnt  flush  lnext  susp   intr   quit    stop     eof
  ^?     ^U     ^W     ^R     ^O     ^V    ^Z/^Y  ^C     ^\     ^S/^Q    ^D
urmac% stty -g
2526:1805:1af:8a3b:3:1c:7f:15:4:0:0:0:11:13:1a:19:12:f:17:16:0
urmac%
```