

Unix Shells

An Advanced Introduction to Unix/C Programming



Dennis
Ritchie



Ken
Thompson



Linus
Torvalds



Richard
Stallman



Brian
Kernighan

John Dempsey

COMP-232 Programming Languages
California State University, Channel Islands

Unix Shells

When you login, you are assigned a default shell.

For example, John's default login Unix shell is /bin/bash.

```
root@oho:~# grep john /etc/passwd
```

```
john:x:1000:1000:John Dempsey,805-111-2222,Remote:/home/john:/bin/bash
```

You can check to see what Unix shell you're using by typing:

```
% echo $SHELL
```

```
/bin/bash
```

Unix Shells

There are many Unix shells to choose from, such as:

sh	- Bourne Shell
csh	- C Shell
bash	- Born Again Shell
ksh	- Korn Shell
tsh	- tsh
zsh	- zsh
..... and many more!	

Each has its own syntax when writing scripts.

Unix Shell

- A Unix shell is an interpreter and supports a command line interface and a scripting language.
- A command line interface lets you type and run commands.
- A scripting language is a programming language that lets you run Unix commands in the script.

Shell Example

```
% chmod 755 run
```

```
% cat run
```

```
vi c.c
```

```
gcc c.c
```

```
a.out
```

```
% run
```

What shell does the above run script run in?

What does run shell script do when you run it?

Shell Example

```
% chmod 755 run
```

```
% cat run
```

```
cd /home/projects/johnstudent/tictactoe
```

```
pwd
```

```
ls -l
```

```
vi getHumanMove.c
```

```
% run
```

What shell does the above run script run in?

What does run shell script do when you run it?

Bourne sh Example

```
#!/bin/sh
```

← First line can specify the shell to use.

```
# Install script
```

```
cp -p control_pgm /app/bin/control_pgm
```

```
cp -p init_pgm /app/bin/init_pgm
```

```
cp -p my_app /app/bin/my_app
```

To run, you can type:

```
% sh my_app.sh
```

← Suffix .sh isn't necessary, but a good idea.

or make the script executable and then run script:

```
% chmod +x my_app.sh
```

```
% my_app.sh
```

Shell Scripts

- Shell scripts can be started in a crontab and run as a cron job.

Crontab Syntax

The crontab file consists of lines with six fields separated by spaces:

```
* * * * * command
| | | | |
| | | | | ----- Day of week (0 - 7) (Sunday=0 or 7)
| | | | | ----- Month (1 - 12)
| | | | | ----- Day of month (1 - 31)
| | | | | ----- Hour (0 - 23)
| | | | | ----- Minute (0 - 59)
```

Each field can contain:

- **Asterisk (*)**: Any value.
- **Comma (,)**: A list of values.
- **Hyphen (-)**: A range of values.
- **Slash (/)**: Step values.

Examples

1. Run a command every hour: `0 * * * * echo "Hourly task"`
2. Run a script every day at 3 AM: `0 3 * * * /path/to/script.sh`
3. Run a command every 5 minutes: `*/5 * * * * /path/to/command`

Shell Scripts

- Shell scripts are used to generate daily or weekly reports, run backups, upload/download files, run hourly and daily tasks.

```
00 07 * * * /u1/scripts/start_app.sh >> /u1/logs/start_app.log
```

```
00 23 * * * /u1/scripts/stop_app.sh >> /u1/logs/stop_app.log
```

```
00 00 * * 7 /u1/scripts/run_backup.sh >> /u1/logs/run_backup.log
```

```
15 * * * * /u1/scripts/check_system.sh > /u1/logs/check_system.log
```

What does this script do?

```
#!/bin/sh -x
cat files.txt | while read filename
do
    echo ${filename}
    echo "-----"
    diff $1/${filename} $2/${filename}
    echo "-----"
done
```

How would you run this script? What does the `-x` do?

bash Syntax: if/then/else & for examples

```
#!/bin/bash
```

```
echo -n Enter 2 numbers:  
read n1 n2  
echo First number is $n1  
echo Second number is $n2
```

```
if [ $n1 -eq $n2 ]  
then  
    echo "$n1 is equal to $n2."  
elif [ $n1 -gt $n2 ]; then  
    echo "$n1 is greater than $n2."  
else  
    echo "$n1 is less than $n2."  
fi
```

```
% a.bash
```

```
Enter two numbers:25 77  
First number entered is: 25  
Second number entered is: 77  
25 is less than 77.
```

```
#!/bin/bash
```

```
for i in $( ls /etc/ )  
do  
    if [ ! -x /etc/$i ]  
    then  
        echo /etc/$i is not an executable file.  
    fi  
done
```

```
% b.bash | head -4  
/etc/adduser.conf is not an executable file.  
/etc/aliases is not an executable file.  
/etc/aliases.db is not an executable file.  
/etc/at.deny is not an executable file.
```

bash Syntax: case & while examples

```
#!/bin/bash
```

```
echo Enter your color:
```

```
echo 1. Red
```

```
echo 2. Green
```

```
echo 3. Blue
```

```
read color
```

```
case $color in
```

```
  1) echo Red is radical.;;
```

```
  2) echo Green is grand.;;
```

```
  3) echo Blue is blah.;;
```

```
  *) echo Other color.
```

```
esac
```

```
john@oho:~/SHELL$ c.bash
```

```
Enter your color:
```

```
1. Red
```

```
2. Green
```

```
3. Blue
```

```
2
```

```
Green is grand.
```

```
#!/bin/bash
```

```
hi=$1
```

```
echo You entered $1.
```

```
counter=1
```

```
echo "counter = $counter"
```

```
while test $hi -ge $counter  
do
```

```
  echo $counter. Hello!
```

```
  counter=$((counter + 1))
```

```
done
```

```
john@oho:~/SHELL$ d.bash 4
```

```
You entered 4.
```

```
counter = 1
```

```
1. Hello!
```

```
2. Hello!
```

```
3. Hello!
```

```
4. Hello!
```

disk_space.bash

```
john@oho:~/SHELL$ cat disk_space.bash
#!/bin/bash
```

```
DATE=`date '+%m%d'`
TIME=`date '+%H%M'`
```

```
echo Today is $DATE at $TIME.
```

```
echo ----- > disk_report_$DATE
echo Disk Report >> disk_report_$DATE
echo Today is $DATE at $TIME. >> disk_report_$DATE
echo ----- >> disk_report_$DATE
df -k >> disk_report_$DATE
echo ----- >> disk_report_$DATE
```

```
john@oho:~/SHELL$ disk_space.bash
Today is 0726 at 1803.
```

```
john@oho:~/SHELL$ cat disk_report_0726
```

```
-----
Disk Report
```

```
Today is 0726 at 1803.
```

```
-----
Filesystem      1K-blocks      Used    Available Use% Mounted on
rootfs          1951597728 1217742252 733855476 63% /
none            1951597728 1217742252 733855476 63% /dev
none            1951597728 1217742252 733855476 63% /run
none            1951597728 1217742252 733855476 63% /run/lock
none            1951597728 1217742252 733855476 63% /run/shm
none            1951597728 1217742252 733855476 63% /run/user
tmpfs           1951597728 1217742252 733855476 63% /sys/fs/cgroup
C:\              1951597728 1217742252 733855476 63% /mnt/c
D:\              4883466240 3239306496 1644159744 67% /mnt/d
-----
```

```
john@oho:~/SHELL$ crontab -l | grep disk
```

```
00 07 * * * /home/john/SHELL/disk_space.bash > /tmp/disk_space.log
```

bash – Shell Script

```
john@oho:~/LAB5/STOCKS$ cat run_YTD_report.bash
```

```
#!/bin/bash
```

```
YYYYMMDD=`date '+%Y%m%d'`
```

```
filename=/home/john/LAB4/STOCKS/$1.txt
```

```
if [ -f "$filename" ]; then
```

```
    echo "File $filename does not exist. Report not run for $YYYYMMDD." >> /home/john/LAB4/STOCKS/log.txt
```

```
    exit
```

```
fi
```

```
/home/john/LAB4/STOCKS/stocks $1 > /home/john/LAB4/STOCKS/report_YYYYMMDD.txt
```

```
echo "YTD stock report for $1 ran on: `date '+%D %r'`" >> /home/john/LAB4/STOCKS/log.txt
```

```
mail -s "YTD Stock Report for $YYYYMMDD" john.dempsey@csuci.edu < /home/john/LAB4/STOCKS/report_YYYYMMDD.txt
```

```
echo "YTD Report for $YYYYMMDD Done."
```

Job Control

You can push your job into the background, run Unix commands, then bring your job to foreground.

% vi filename.c

^Z

% chmod 644 filename.c

% fg

% jobs

% fg 3

← Edit a file

← Push vi into background.

← Run Unix commands.

← Go back to the vi filename.c.

← Bring job #3 to foreground

Job Control

You can run your program in the “background” by using & at the end.

```
john@oho:~$ cat sleep.c
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Sleep for 5 seconds.\n");
    sleep(5);
}
```

```
john@oho:~$ gcc sleep.c
```

```
john@oho:~$ a.out &
```

```
[1] 155
```

```
john@oho:~$ Sleep for 5 seconds.
```

```
john@oho:~$ date
```

```
Tue Oct 10 20:41:35 PDT 2023
```

```
john@oho:~$
```

```
[1]+ Done
```

```
a.out
```

← Run sleep.c in the background

← Run one or more commands while sleep.c runs.

← Tells you when its done.

Job Control

If you're running a program that takes a long time to complete, you can run the program in the background after starting it.

% analyze_data 2022

% ^Z

% bg

% jobs

← Taking too long to complete

← Now runs like: analyze_data 2022 &

← You can run your next command

tail -f filename

To see updates to a file that is being updated while a program run, use tail -f

% tail -f /home/sherry/output.log