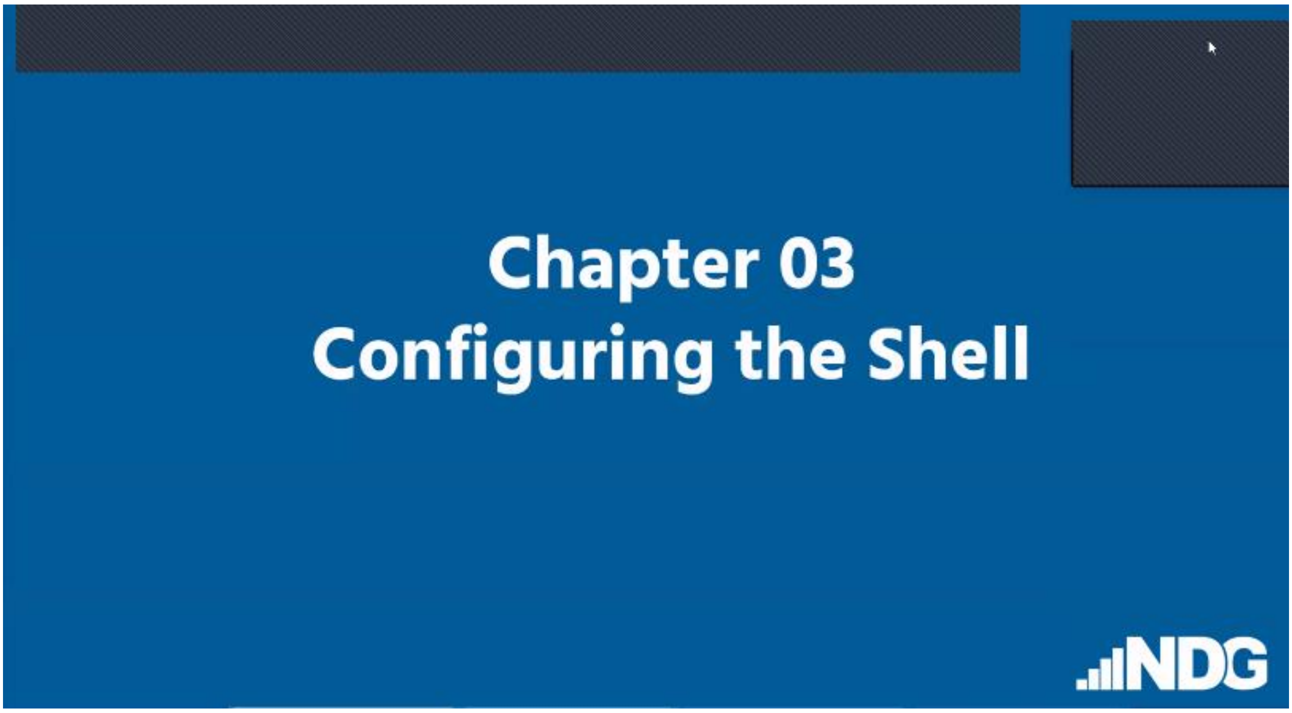
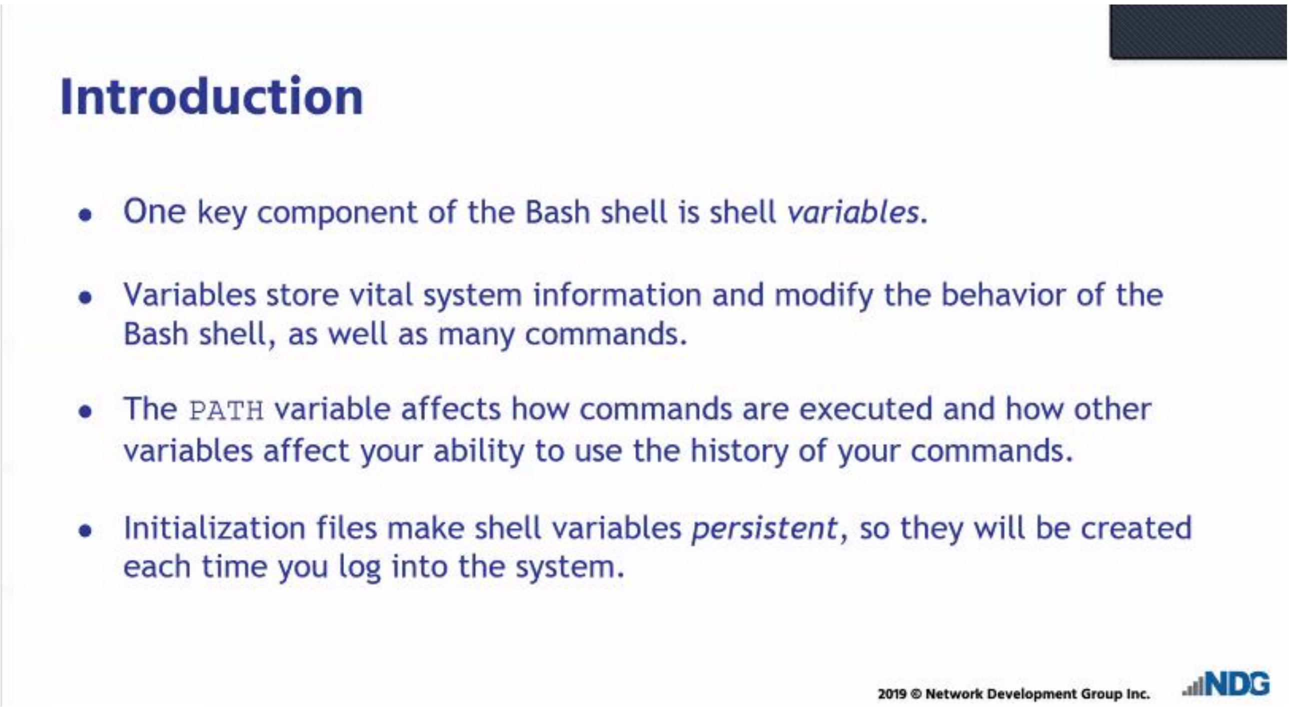
MODULO 3

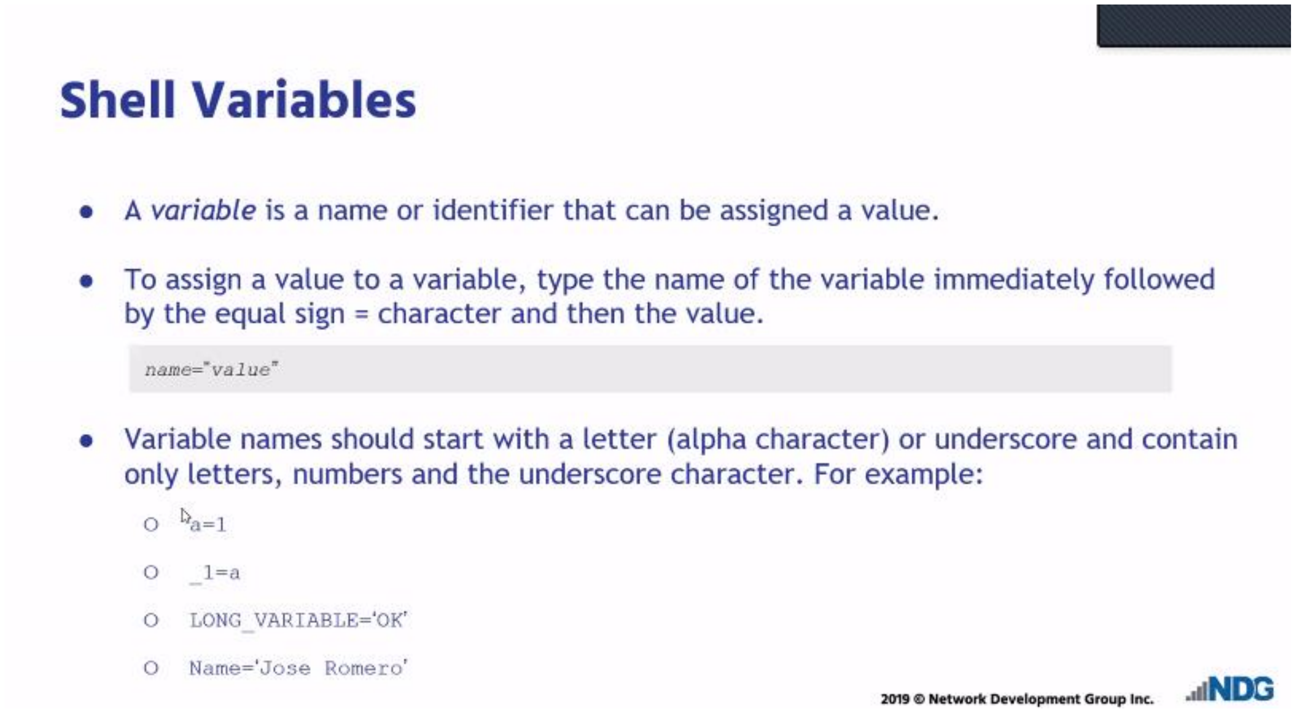


Chapter 03
Configuring the Shell



Introduction

- One key component of the Bash shell is shell *variables*.
- Variables store vital system information and modify the behavior of the Bash shell, as well as many commands.
- The `PATH` variable affects how commands are executed and how other variables affect your ability to use the history of your commands.
- Initialization files make shell variables *persistent*, so they will be created each time you log into the system.



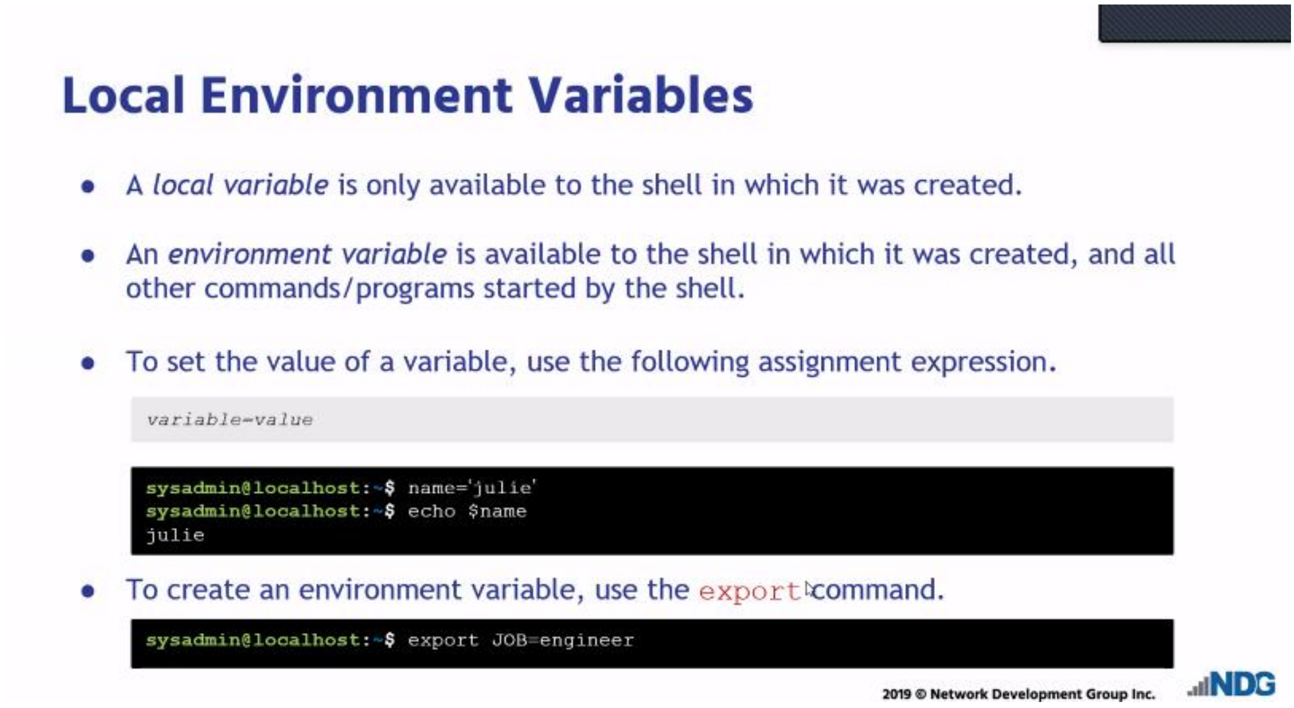
Shell Variables

- A *variable* is a name or identifier that can be assigned a value.
- To assign a value to a variable, type the name of the variable immediately followed by the equal sign = character and then the value.

```
name="value"
```

- Variable names should start with a letter (alpha character) or underscore and contain only letters, numbers and the underscore character. For example:
  - a=1
  - _1=a
  - LONG_VARIABLE='OK'
  - Name='Jose Romero'

2019 © Network Development Group Inc.



Local Environment Variables

- A *local variable* is only available to the shell in which it was created.
- An *environment variable* is available to the shell in which it was created, and all other commands/programs started by the shell.
- To set the value of a variable, use the following assignment expression.

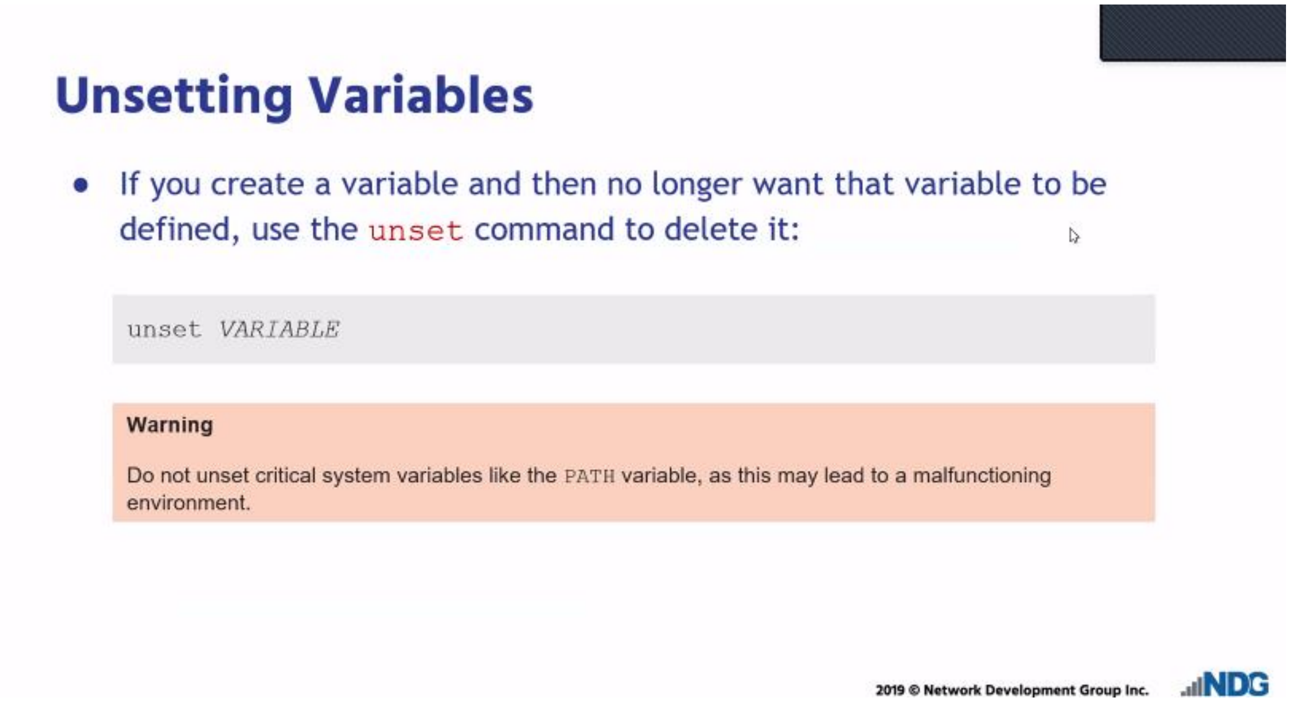
```
variable=value
```

```
sysadmin@localhost:~$ name='julie'
sysadmin@localhost:~$ echo $name
julie
```

- To create an environment variable, use the `export` command.

```
sysadmin@localhost:~$ export JOB=engineer
```

2019 © Network Development Group Inc.



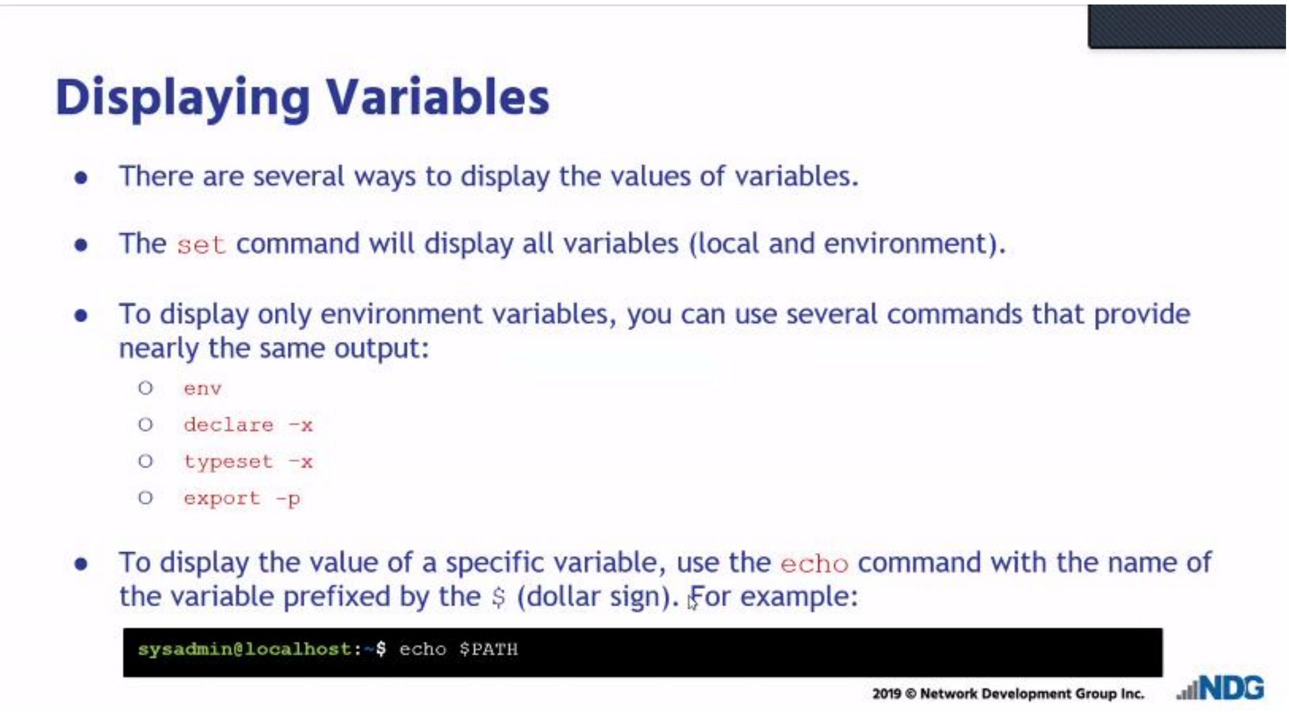
Unsetting Variables

- If you create a variable and then no longer want that variable to be defined, use the unset command to delete it:

```
unset VARIABLE
```

Warning

Do not unset critical system variables like the PATH variable, as this may lead to a malfunctioning environment.



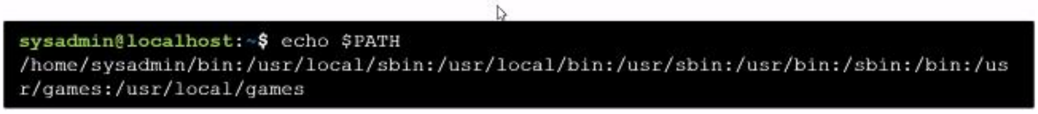
Displaying Variables

- There are several ways to display the values of variables.
- The set command will display all variables (local and environment).
- To display only environment variables, you can use several commands that provide nearly the same output:
  - env
  - declare -x
  - typeset -x
  - export -p
- To display the value of a specific variable, use the echo command with the name of the variable prefixed by the $ (dollar sign). For example:

```
sysadmin@localhost:~$ echo $PATH
```

PATH Variable

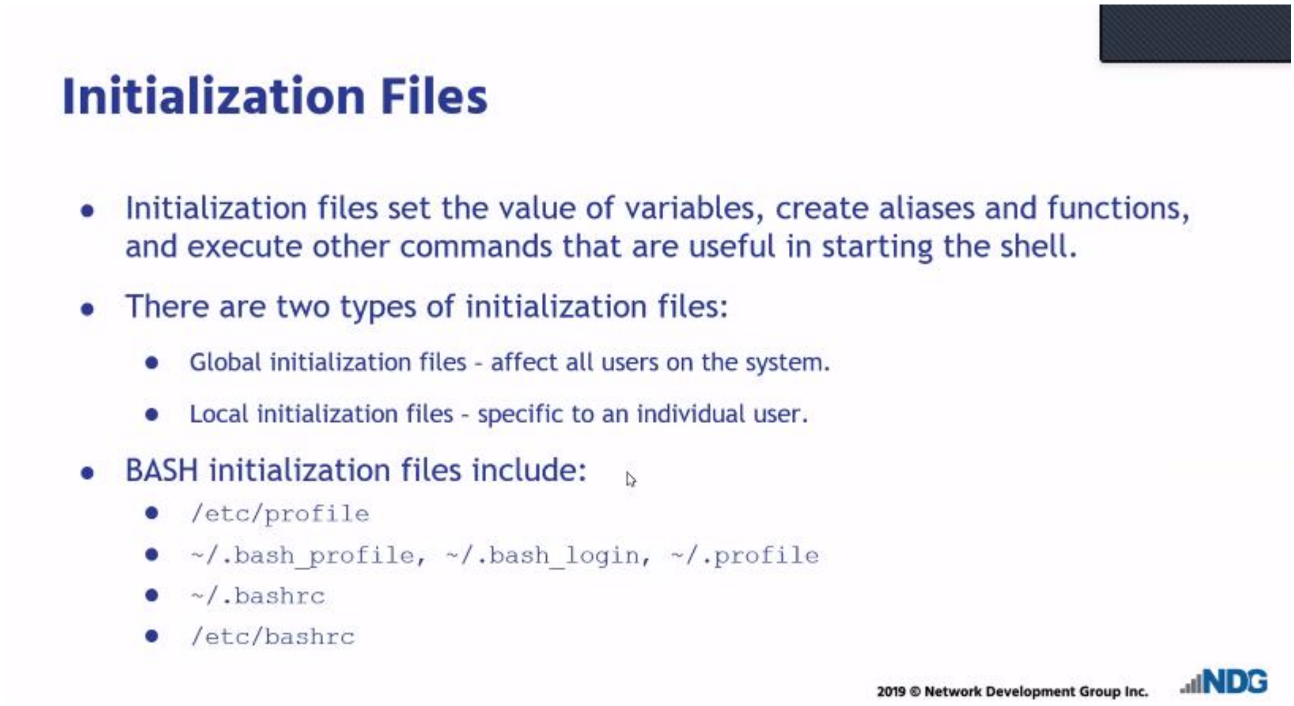
- The PATH variable contains a list of directories that are used to search for commands entered by the user.
- The PATH directories are searched for an executable file that matches the command name.
- The following example displays a typical PATH variable:



```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/us
r/games:/usr/local/games
```

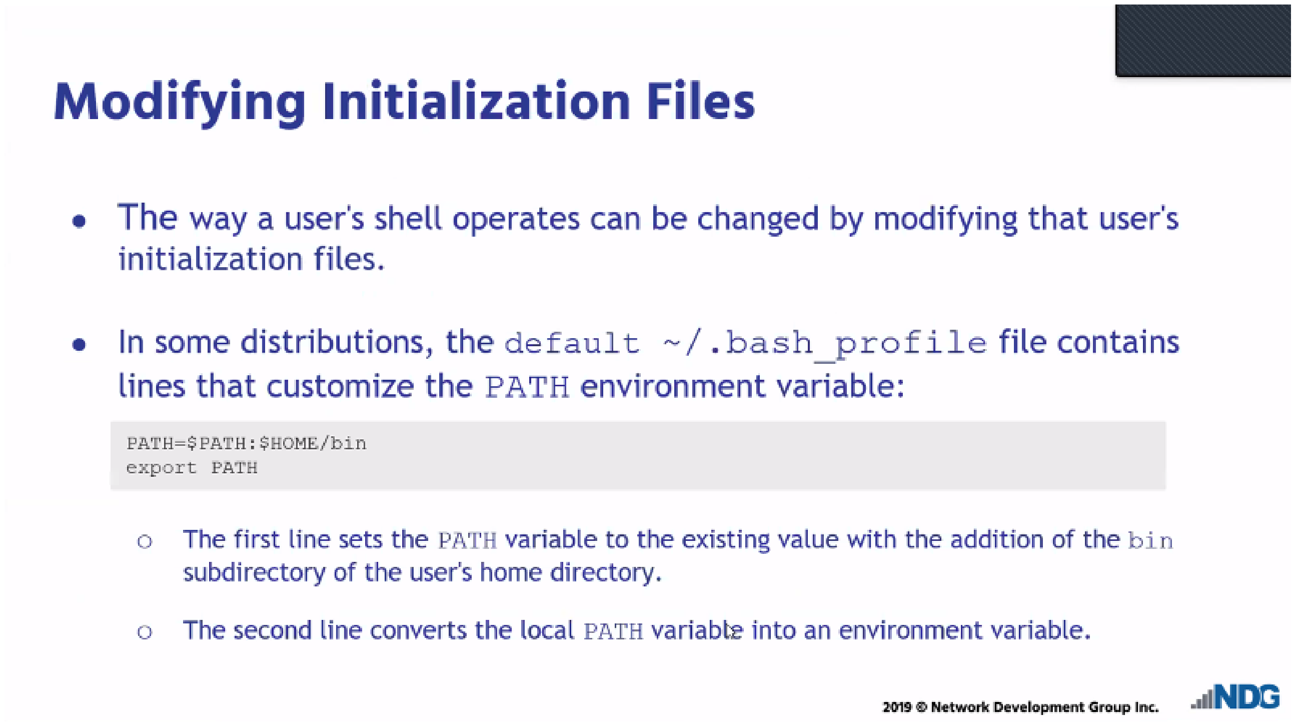
PATH Variable

- To execute commands that are not contained in the directories that are listed in the PATH variable, several options exist:
  - Type the *absolute path* to the command.
  - Use the *relative path* to the command.
  - The PATH variable can be set to include the directory where the command is located.
  - Copy command to a directory that is listed in the PATH variable.
- An *absolute path* specifies the location of a file or directory from the top-level directory (i.e. /usr/bin/ls).
- A *relative path* specifies the location of a file or directory relative to the current directory (i.e. test/newfile).



Initialization Files

- Initialization files set the value of variables, create aliases and functions, and execute other commands that are useful in starting the shell.
- There are two types of initialization files:
  - Global initialization files - affect all users on the system.
  - Local initialization files - specific to an individual user.
- BASH initialization files include:
  - `/etc/profile`
  - `~/.bash_profile, ~/.bash_login, ~/.profile`
  - `~/.bashrc`
  - `/etc/bashrc`

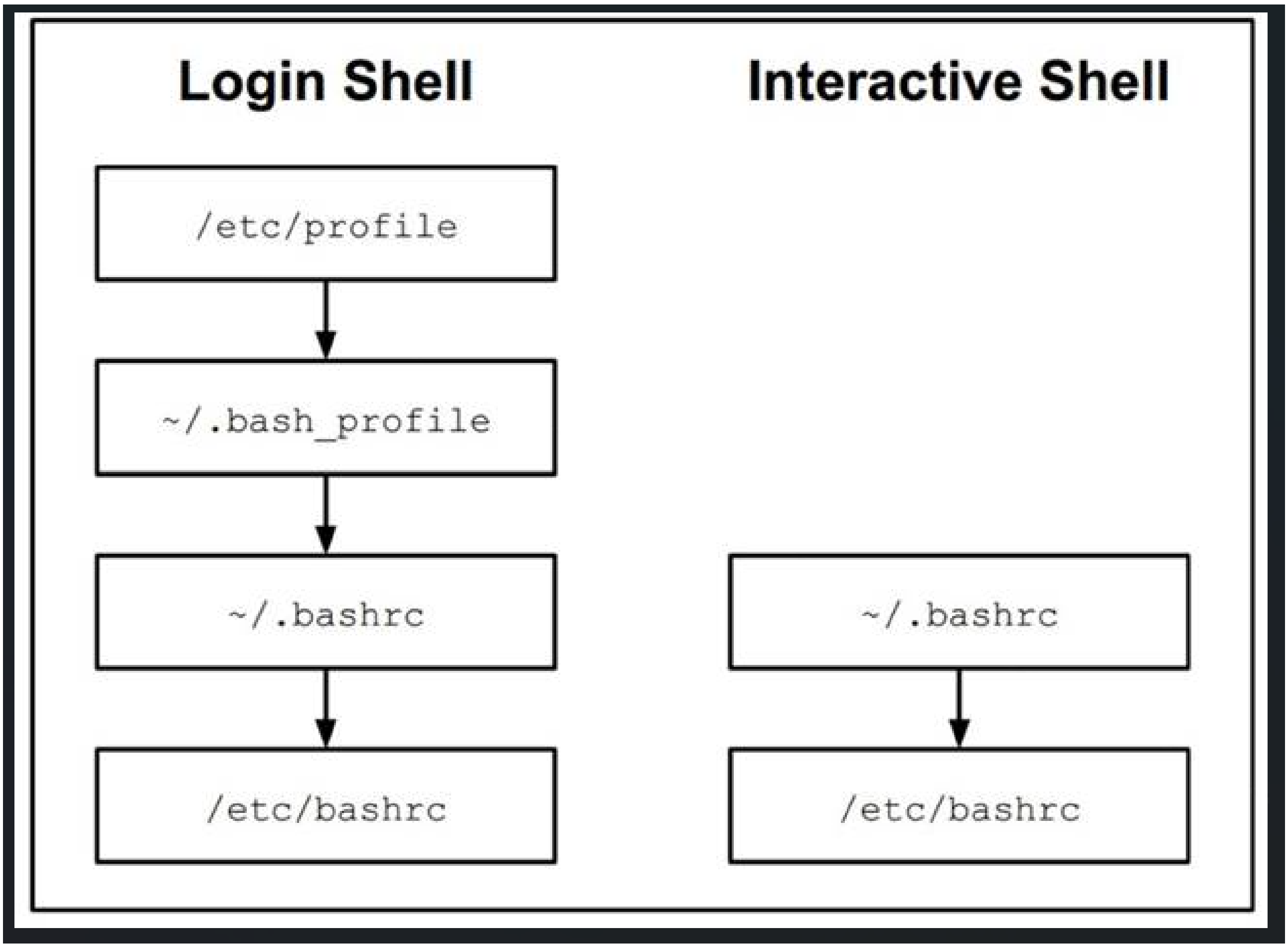


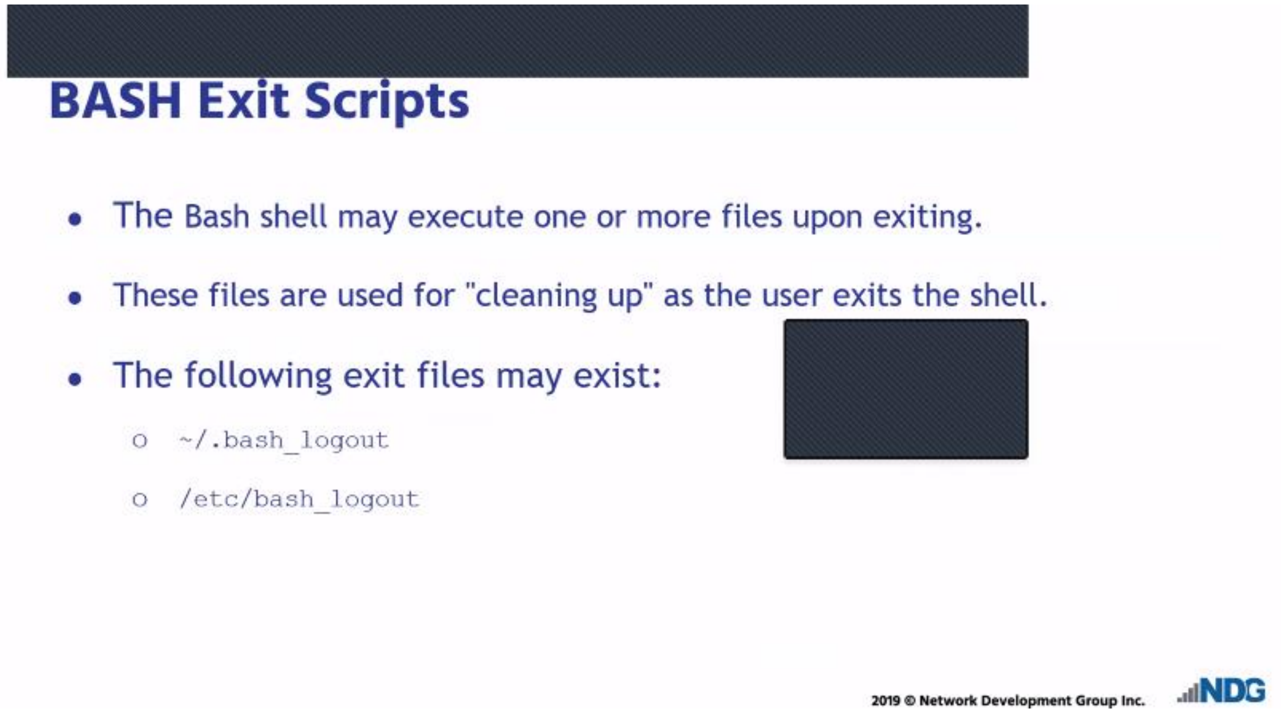
Modifying Initialization Files

- The way a user's shell operates can be changed by modifying that user's initialization files.
- In some distributions, the `default ~/.bash_profile` file contains lines that customize the `PATH` environment variable:

```
PATH=$PATH:$HOME/bin
export PATH
```

  - The first line sets the `PATH` variable to the existing value with the addition of the `bin` subdirectory of the user's home directory.
  - The second line converts the local `PATH` variable into an environment variable.


Login Shell
Interactive Shell
/etc/profile
~/.bash_profile
~/.bashrc
/etc/bashrc
~/.bashrc
/etc/bashrc

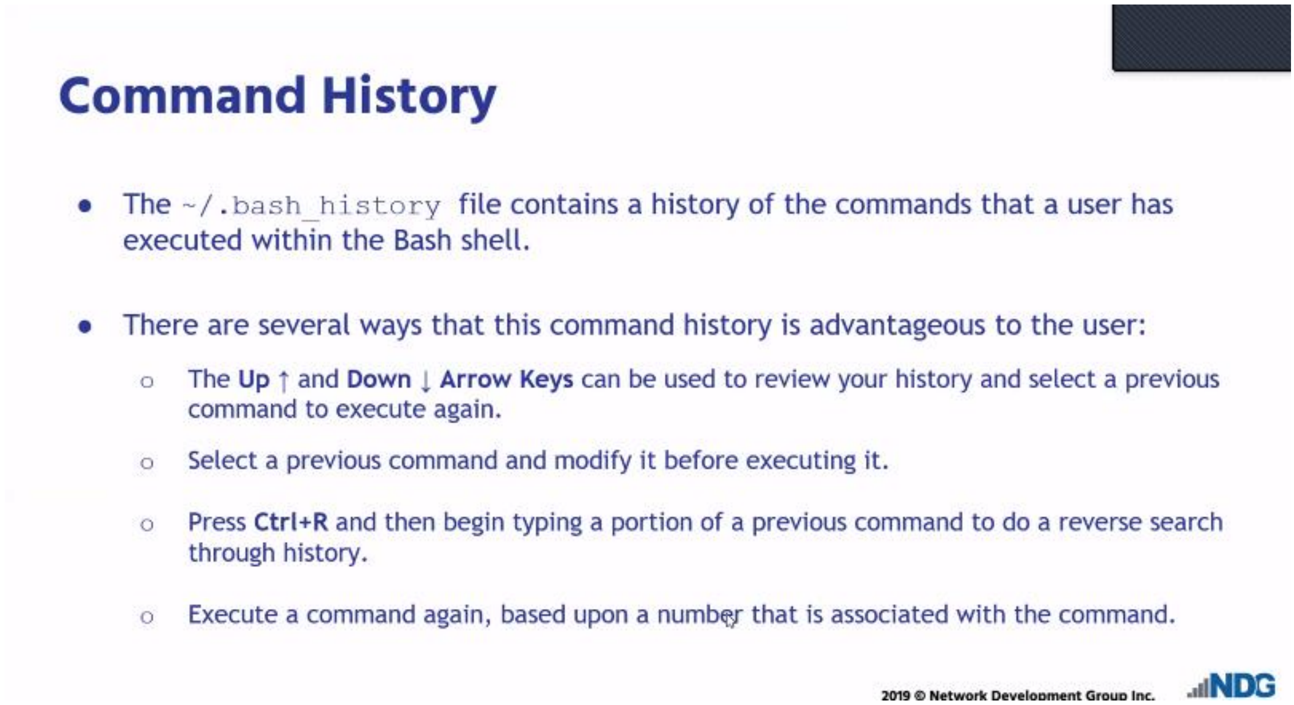


[redacted]

BASH Exit Scripts

- The Bash shell may execute one or more files upon exiting.
- These files are used for "cleaning up" as the user exits the shell.
- The following exit files may exist:
  - `~/.bash_logout`
  - `/etc/bash_logout`

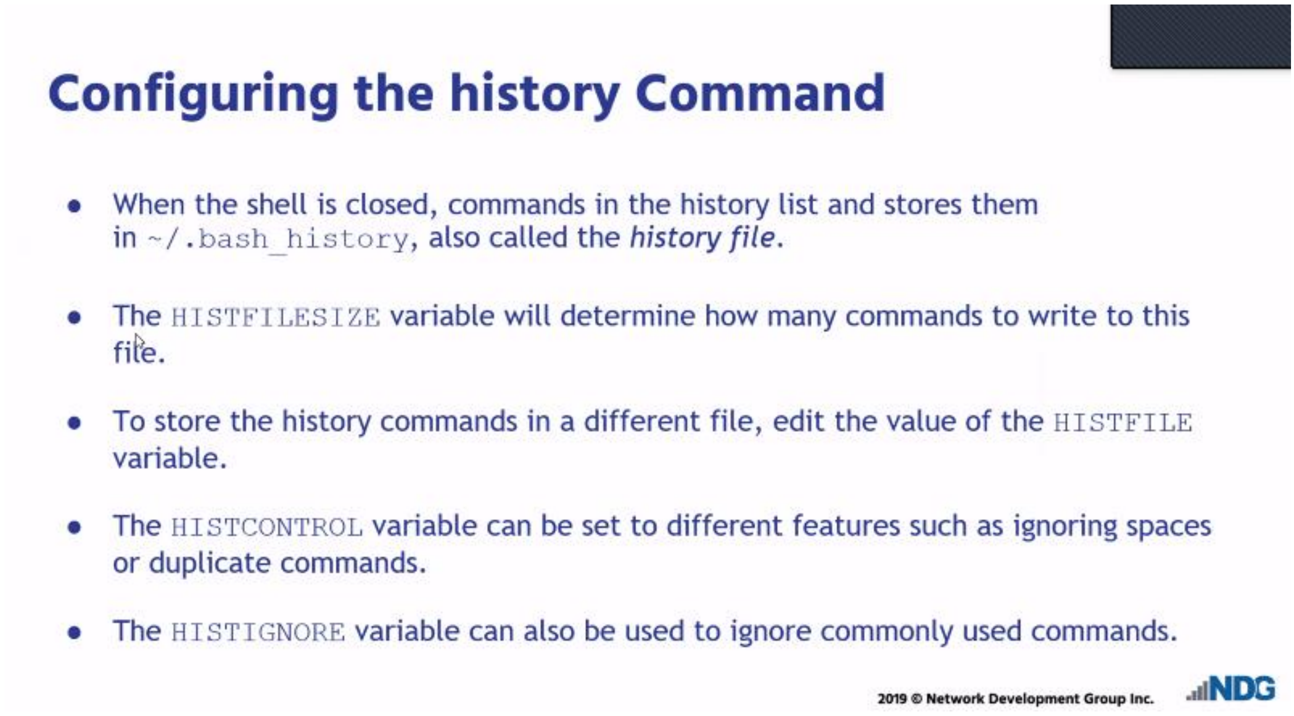
[redacted]



[redacted]

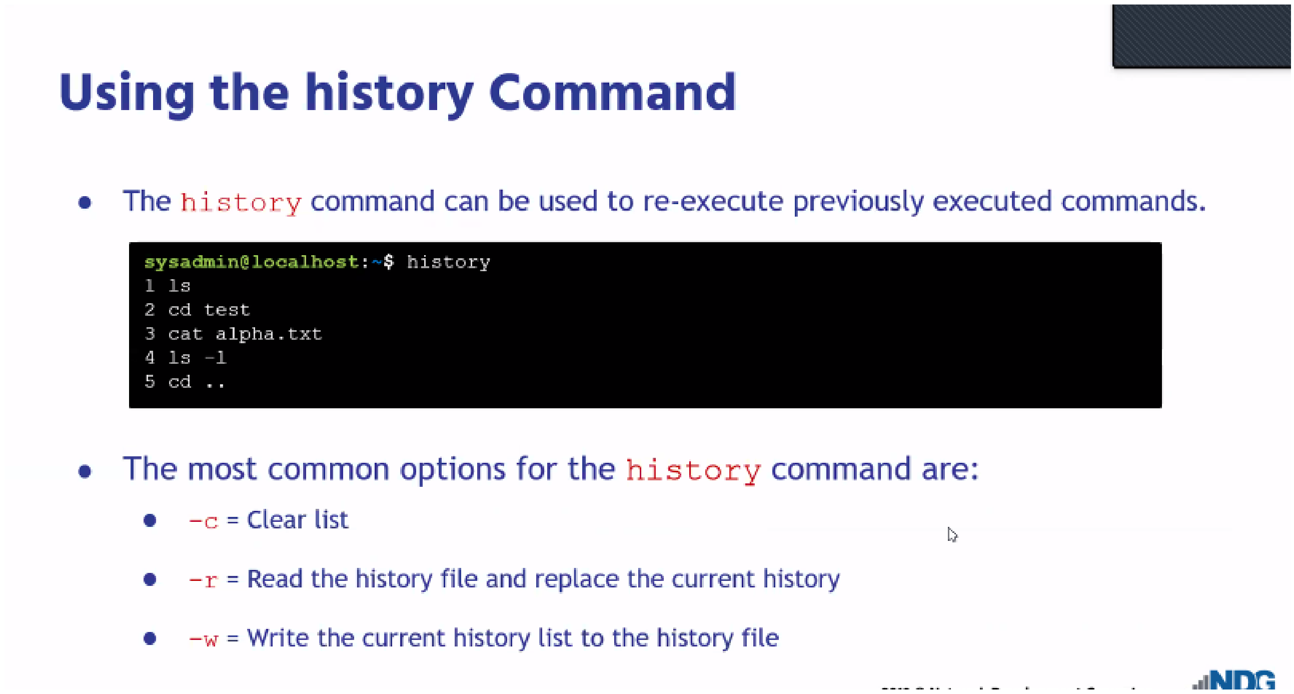
Command History

- The `~/.bash_history` file contains a history of the commands that a user has executed within the Bash shell.
- There are several ways that this command history is advantageous to the user:
  - The **Up ↑** and **Down ↓ Arrow Keys** can be used to review your history and select a previous command to execute again.
  - Select a previous command and modify it before executing it.
  - Press **Ctrl+R** and then begin typing a portion of a previous command to do a reverse search through history.
  - Execute a command again, based upon a number that is associated with the command.



Configuring the history Command

- When the shell is closed, commands in the history list and stores them in `~/.bash_history`, also called the *history file*.
- The `HISTFILESIZE` variable will determine how many commands to write to this file.
- To store the history commands in a different file, edit the value of the `HISTFILE` variable.
- The `HISTCONTROL` variable can be set to different features such as ignoring spaces or duplicate commands.
- The `HISTIGNORE` variable can also be used to ignore commonly used commands.

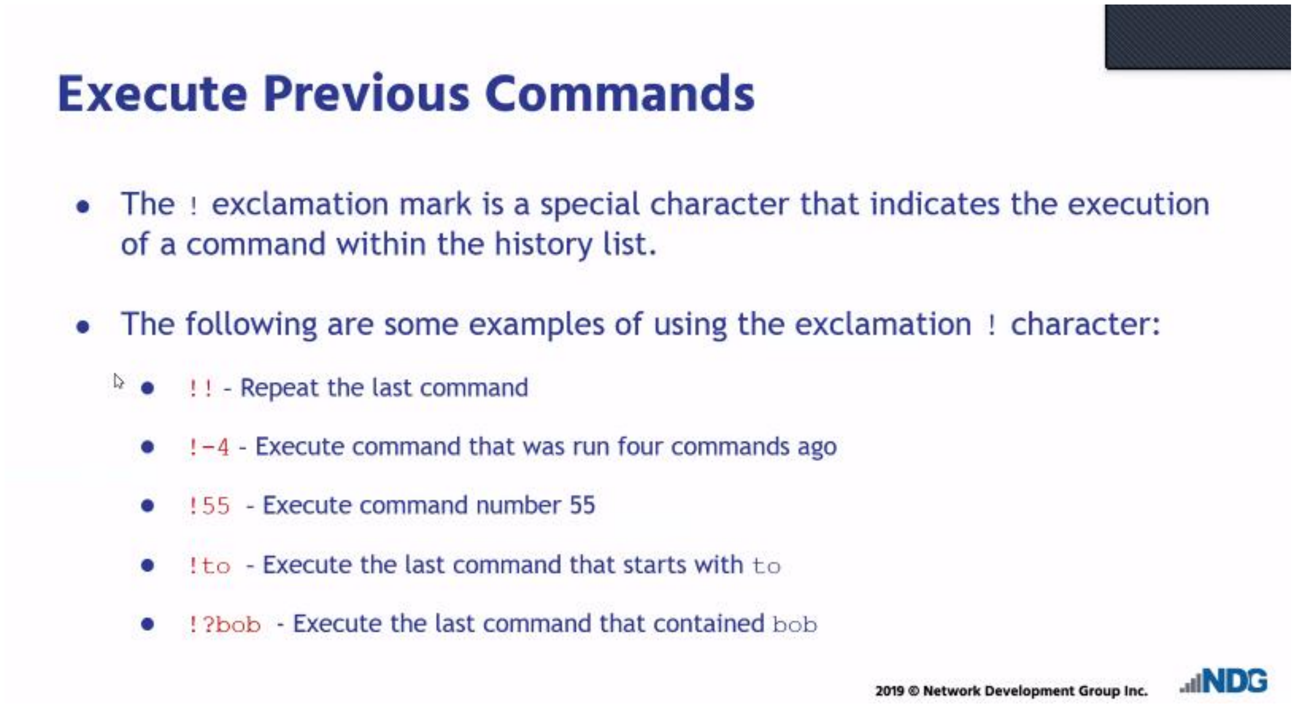


Using the history Command

- The `history` command can be used to re-execute previously executed commands.

```
sysadmin@localhost:~$ history
1 ls
2 cd test
3 cat alpha.txt
4 ls -l
5 cd ..
```

- The most common options for the `history` command are:
  - `-c` = Clear list
  - `-r` = Read the history file and replace the current history
  - `-w` = Write the current history list to the history file

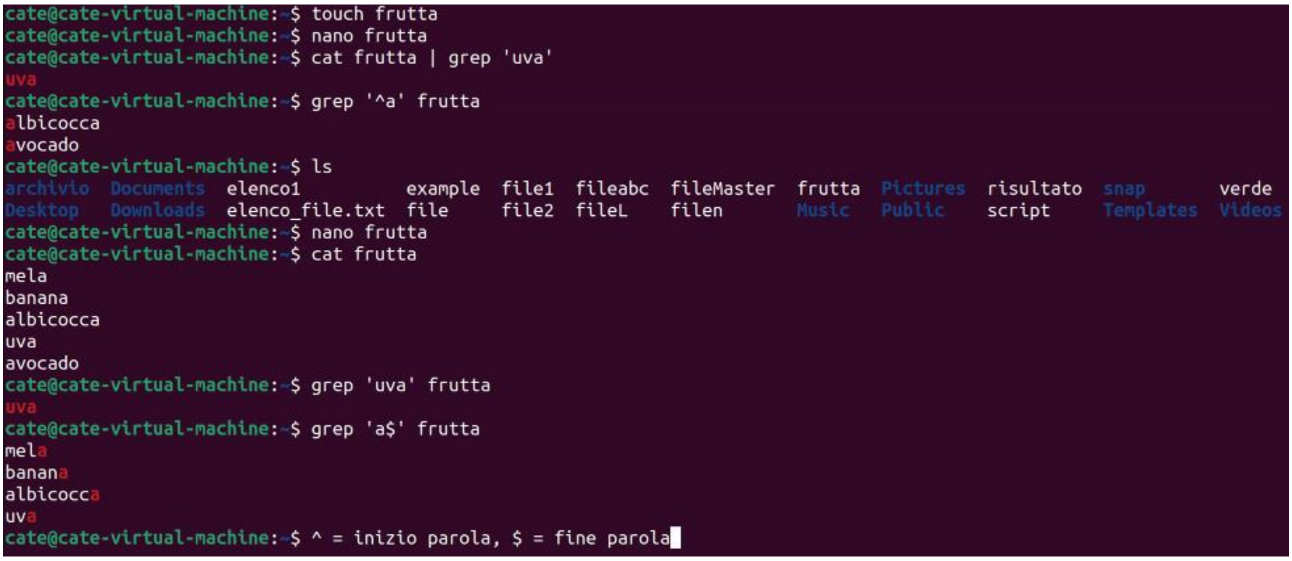


Execute Previous Commands

- The ! exclamation mark is a special character that indicates the execution of a command within the history list.
- The following are some examples of using the exclamation ! character:
  - `!!` - Repeat the last command
  - `!-4` - Execute command that was run four commands ago
  - `!55` - Execute command number 55
  - `!to` - Execute the last command that starts with `to`
  - `!?bob` - Execute the last command that contained `bob`

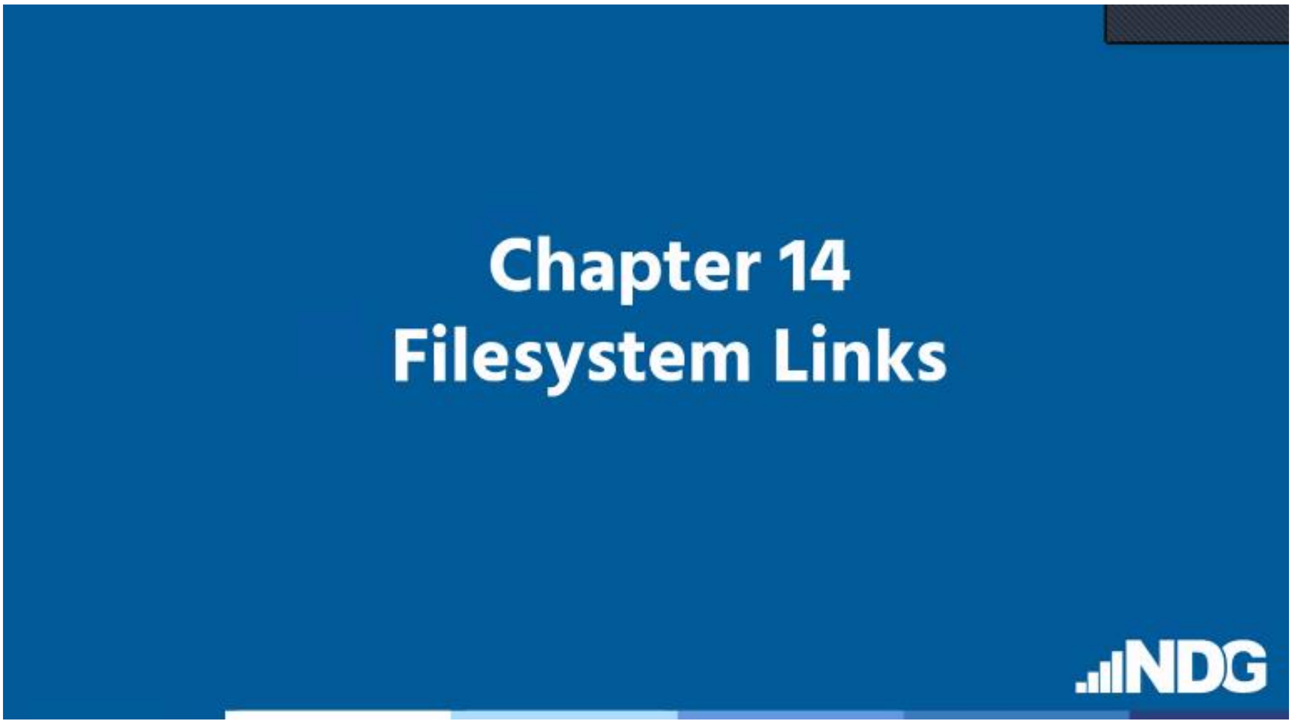
MODULO 8

Grep premette di cercare all’interno di un testo delle parole specifiche (grep ‘uva’ file.txt, grep ‘^a’ file.txt, grep ‘a$’ file.txt)

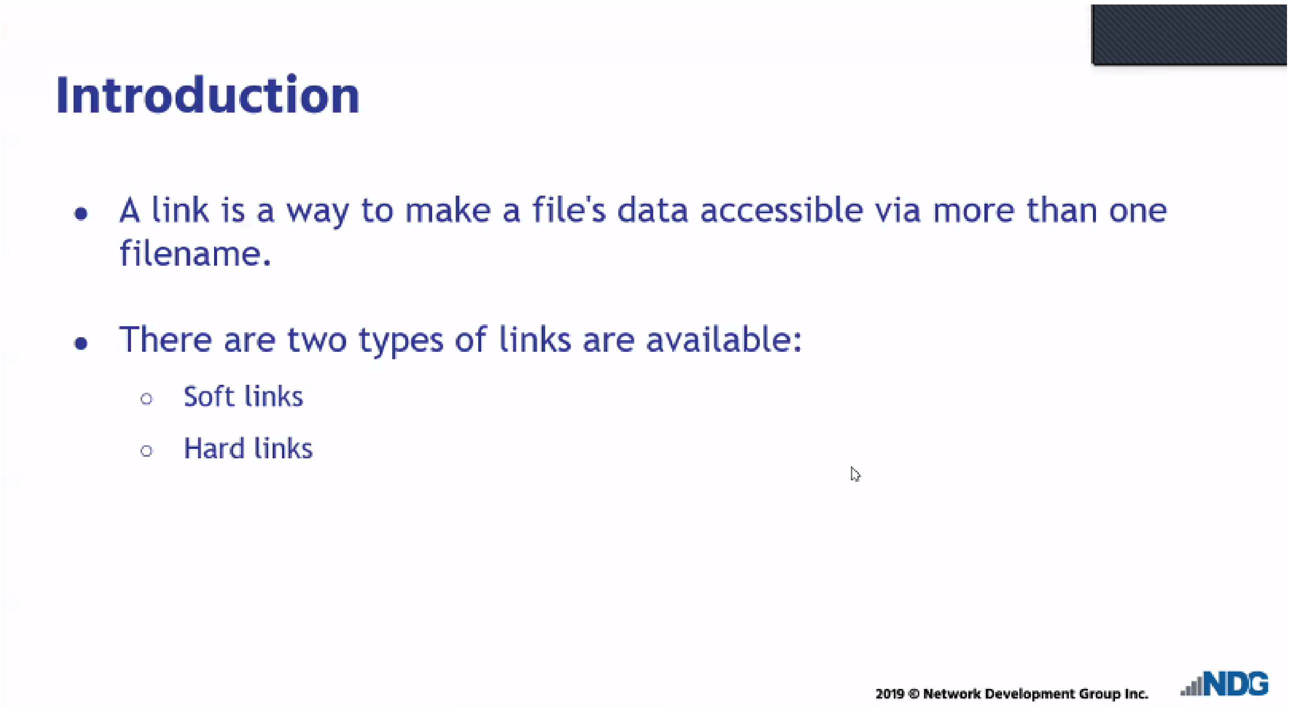


```
cate@cate-virtual-machine:~$ touch frutta
cate@cate-virtual-machine:~$ nano frutta
cate@cate-virtual-machine:~$ cat frutta | grep 'uva'
uva
cate@cate-virtual-machine:~$ grep '^a' frutta
albicocca
avocado
cate@cate-virtual-machine:~$ ls
archivio  Documents  elenco1          example  file1  fileabc  fileMaster  frutta  Pictures  risultato  snap       verde
Desktop   Downloads  elenco_file.txt  file     file2  fileL    filen       Music   Public    script     Templates  Videos
cate@cate-virtual-machine:~$ nano frutta
cate@cate-virtual-machine:~$ cat frutta
mela
banana
albicocca
uva
avocado
cate@cate-virtual-machine:~$ grep 'uva' frutta
uva
cate@cate-virtual-machine:~$ grep 'a$' frutta
mela
banana
albicocca
uva
cate@cate-virtual-machine:~$ ^ = inizio parola, $ = fine parola
```

MODULO 14

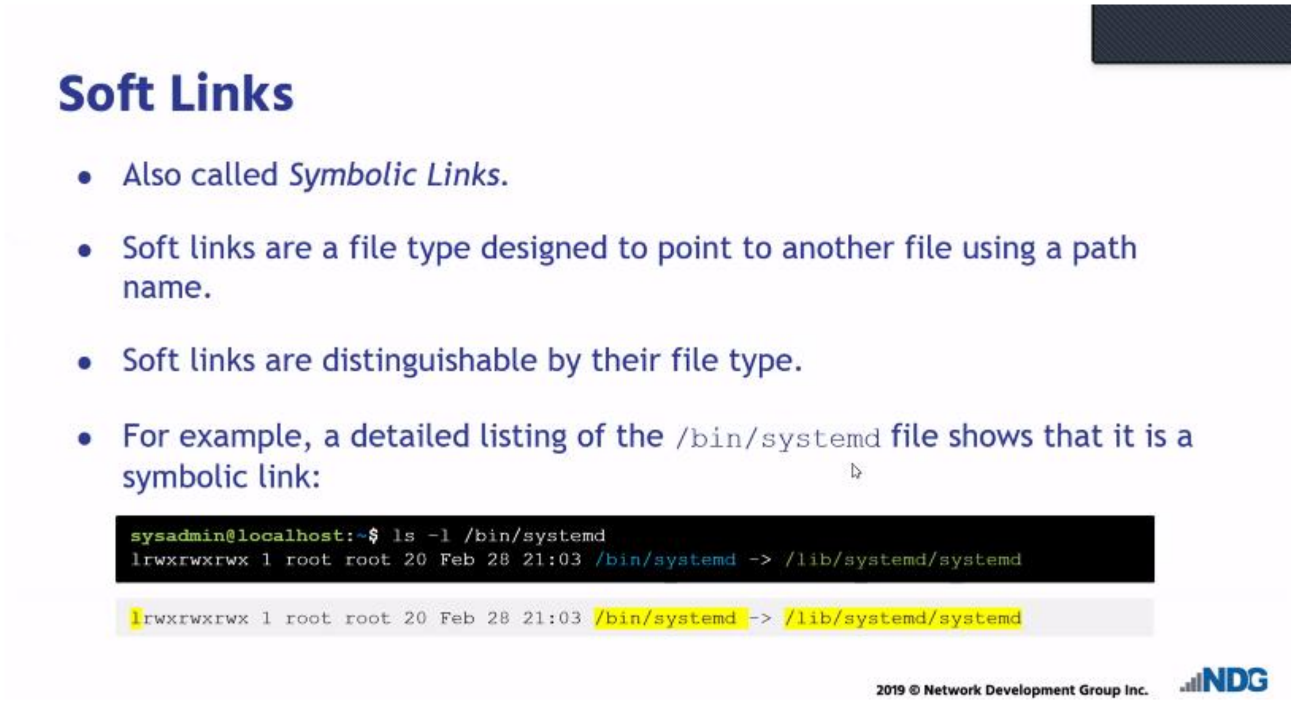


Chapter 14
Filesystem Links



Introduction

- A link is a way to make a file's data accessible via more than one filename.
- There are two types of links are available:
  - Soft links
  - Hard links

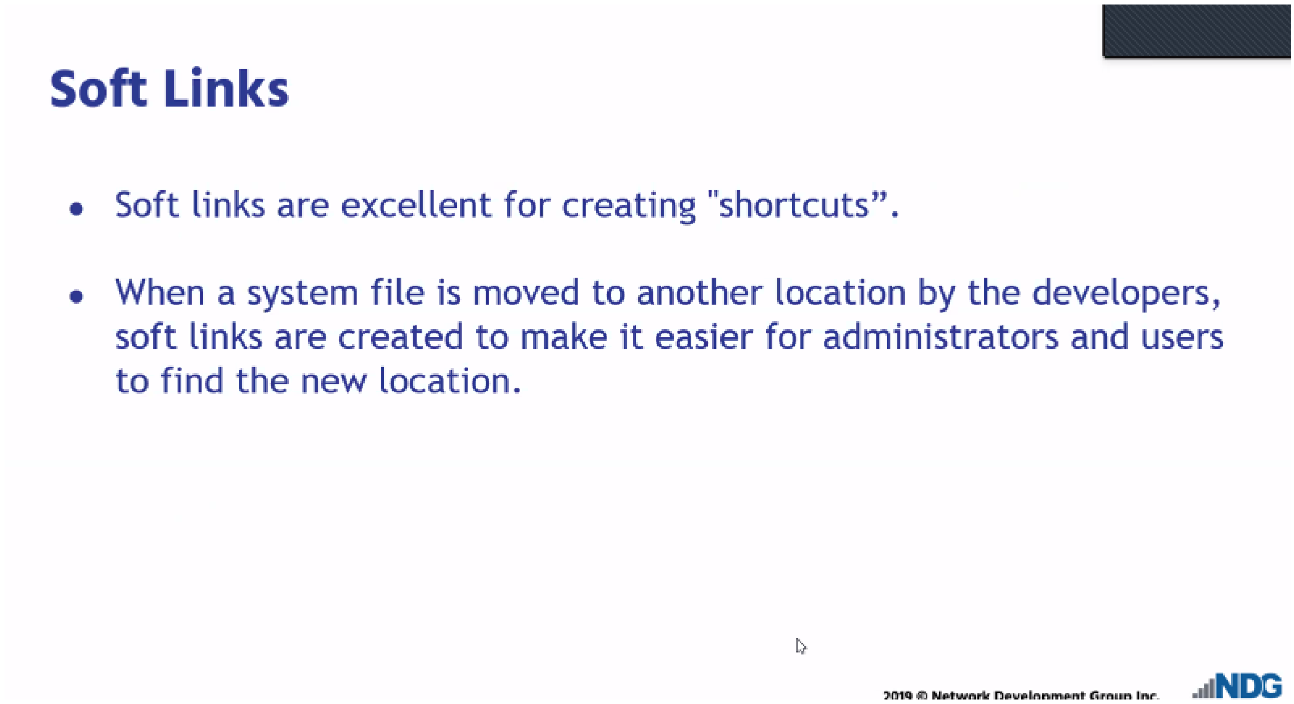


Soft Links

- Also called *Symbolic Links*.
- Soft links are a file type designed to point to another file using a path name.
- Soft links are distinguishable by their file type.
- For example, a detailed listing of the `/bin/systemd` file shows that it is a symbolic link:

```
sysadmin@localhost:~$ ls -l /bin/systemd
lrwxrwxrwx 1 root root 20 Feb 28 21:03 /bin/systemd -> /lib/systemd/systemd
```

```
lrwxrwxrwx 1 root root 20 Feb 28 21:03 /bin/systemd -> /lib/systemd/systemd
```



Soft Links

- Soft links are excellent for creating "shortcuts”.
- When a system file is moved to another location by the developers, soft links are created to make it easier for administrators and users to find the new location.

Soft Links

- To create a soft link file, use the `ln` command with the `-s` option.

```
ln -s target link_name
```

```
sysadmin@localhost:~$ ln -s file1.txt file2.txt
sysadmin@localhost:~$ ls -l file*
-rw-rw-r-- 1 sysadmin sysadmin 0 May 9 02:48 file1.txt
lrwxrwxrwx 1 sysadmin sysadmin 9 May 9 02:49 file2.txt -> file1.txt
```

- Soft link files *do not* increase the *link count* number associated with a regular file.

```
-rw-r--r-- 1 sysadmin sysadmin May 9 14:39 file1.txt
lrwxrwxrwx 1 sysadmin sysadmin May 9 14:39 file2.txt -> file1.txt
```

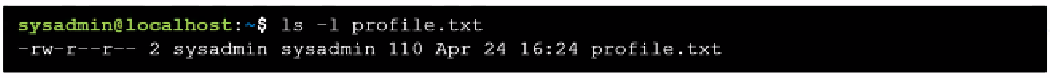
Hard Links

- Hard links are two or more files that share the same inode number.
  - Inode number: index node - contains file info including location on the disk.
- Hard links are exactly identical to the original in every way except the file name
- Hard links are created using the `ln` command without using the `-s` option

```
ln target link_name
```

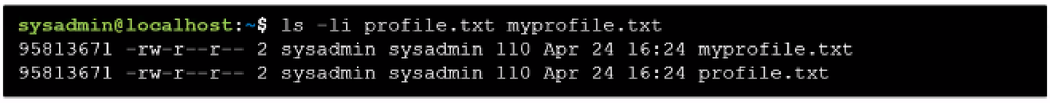
Hard Links

- Hard links identified using `ls -l`.



```
sysadmin@localhost:~$ ls -l profile.txt
-rw-r--r-- 2 sysadmin sysadmin 110 Apr 24 16:24 profile.txt
```

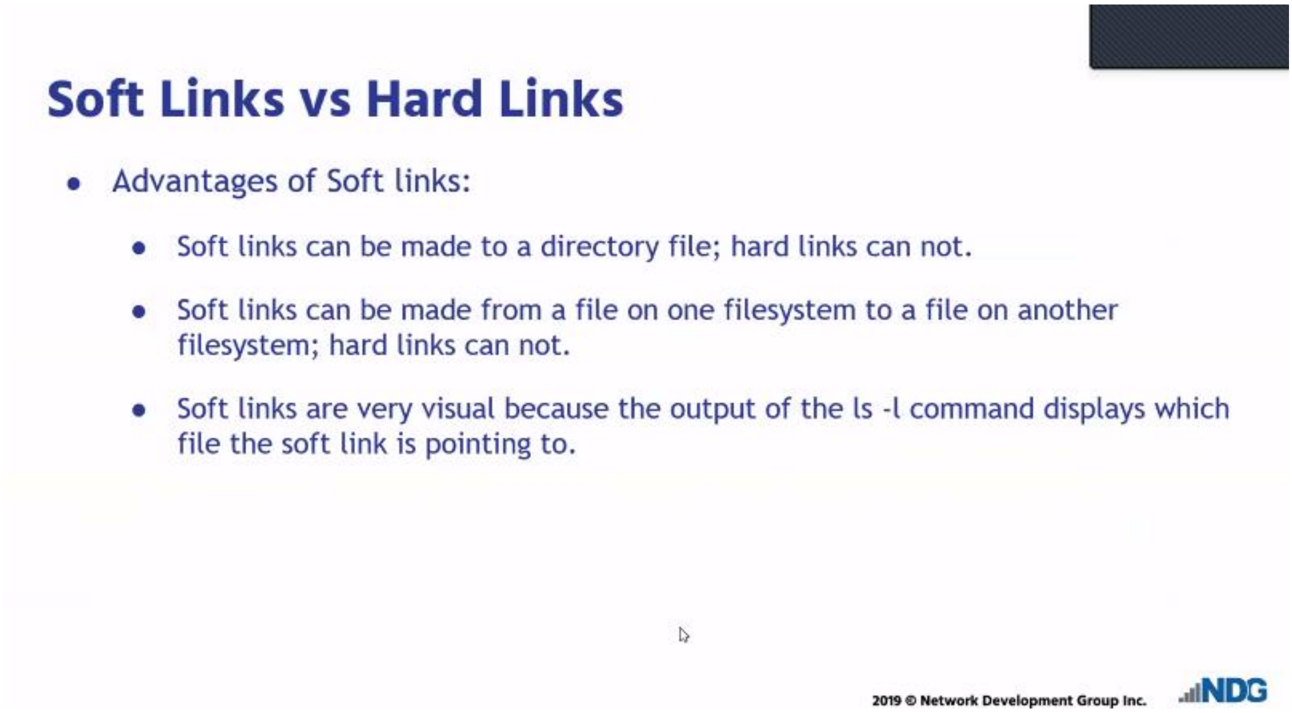
- The link count (2) indicates how many hard links there are to the file.
- Since hard links share the same inode, they will have the same inode number.
- The `ls -i` option can be helpful to validate that the files are sharing an inode:



```
sysadmin@localhost:~$ ls -li profile.txt myprofile.txt
95813671 -rw-r--r-- 2 sysadmin sysadmin 110 Apr 24 16:24 myprofile.txt
95813671 -rw-r--r-- 2 sysadmin sysadmin 110 Apr 24 16:24 profile.txt
```

Soft Links vs Hard Links

- Advantages of Hard links:
  - Hard linked files are indistinguishable by programs from regular files
  - If files are hard linked then they are always contained within one filesystem
  - It is easy to find files that are hard linked
  - Removing hard links doesn't remove the actual data unless you remove all of the hard links



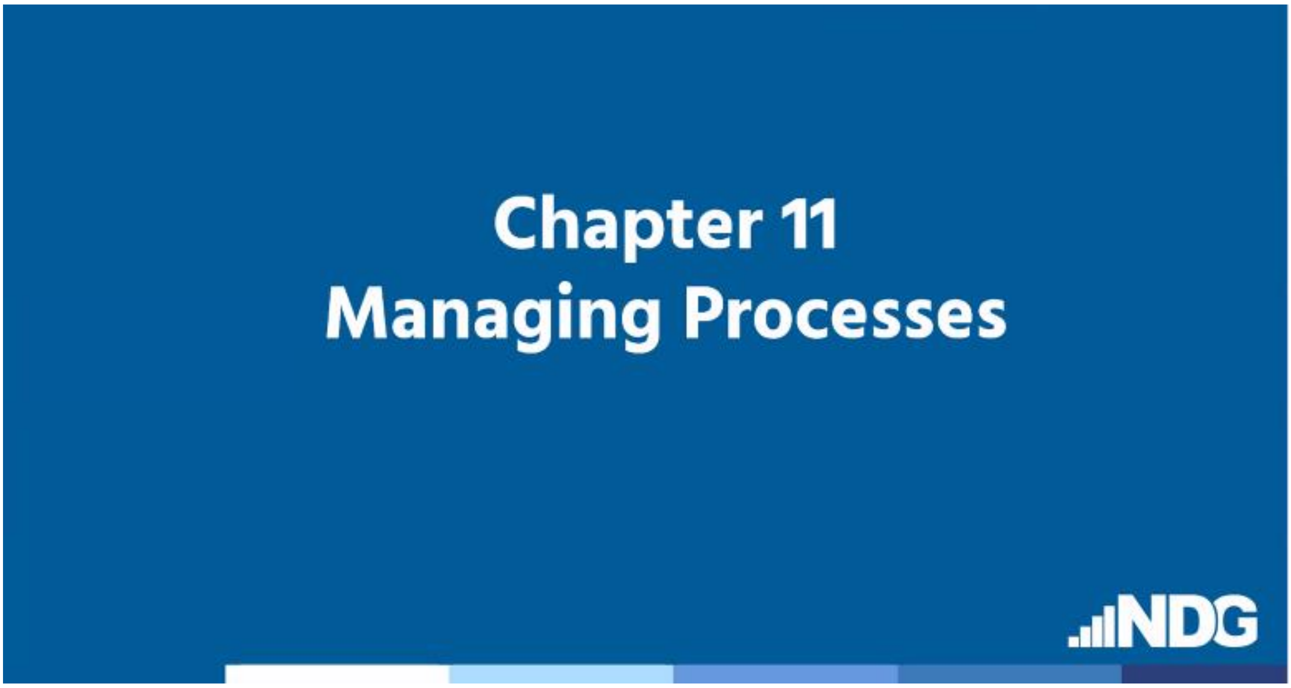
Soft Links vs Hard Links

- Advantages of Soft links:
  - Soft links can be made to a directory file; hard links can not.
  - Soft links can be made from a file on one filesystem to a file on another filesystem; hard links can not.
  - Soft links are very visual because the output of the ls -l command displays which file the soft link is pointing to.

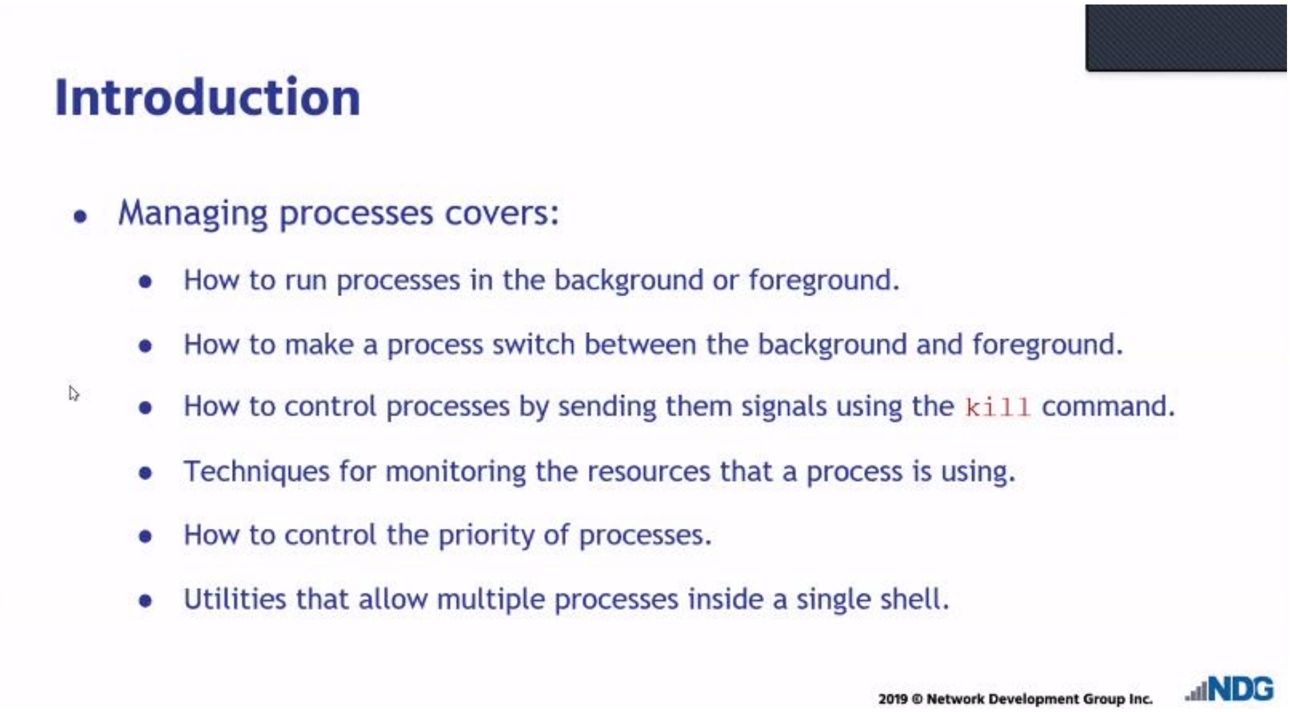
PRO:

Ottimizzazione dello spazio e come e cosa volete creare

MODULO 11

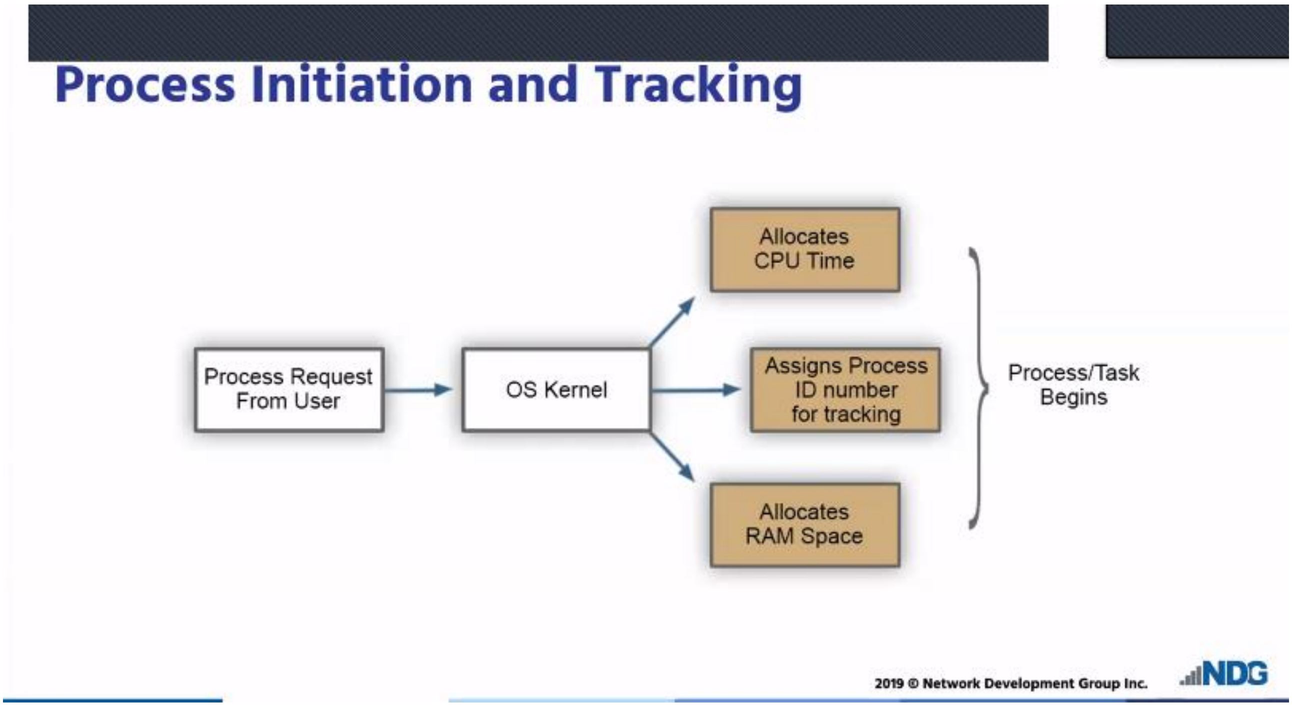


Chapter 11
Managing Processes



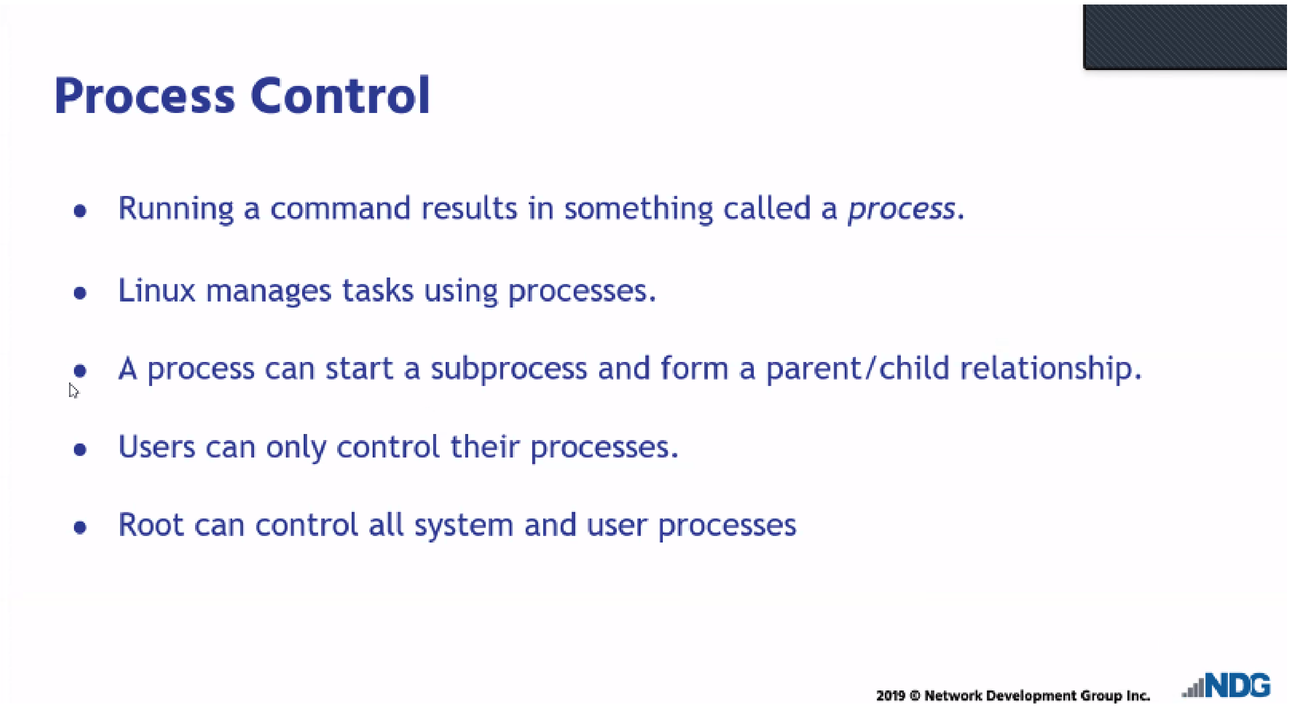
Introduction

- Managing processes covers:
  - How to run processes in the background or foreground.
  - How to make a process switch between the background and foreground.
  - How to control processes by sending them signals using the `kill` command.
  - Techniques for monitoring the resources that a process is using.
  - How to control the priority of processes.
  - Utilities that allow multiple processes inside a single shell.



Process Initiation and Tracking

Process Request From User → OS Kernel → Allocates CPU Time / Assigns Process ID number for tracking / Allocates RAM Space → Process/Task Begins



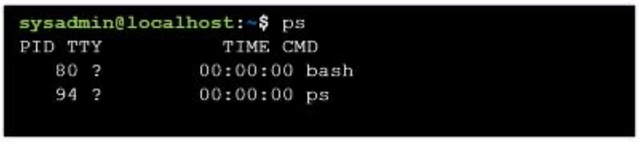
Process Control

- Running a command results in something called a *process*.
- Linux manages tasks using processes.
- A process can start a subprocess and form a parent/child relationship.
- Users can only control their processes.
- Root can control all system and user processes

Listing Processes

- The ps (process status) command lists running processes
- By itself, lists processes running in the current terminal:

```
Ps [OPTION]…
```



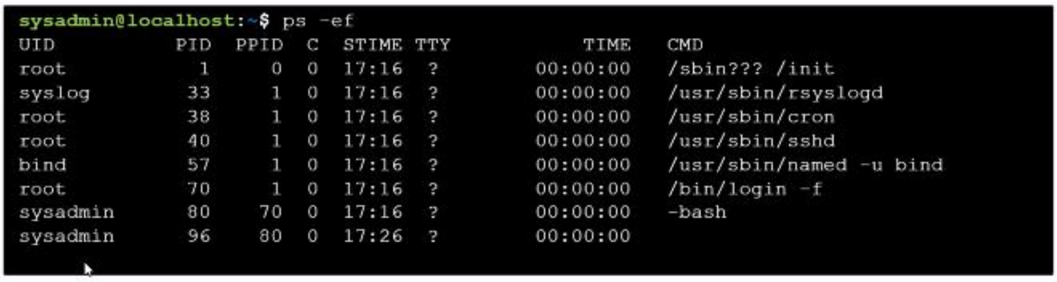
```
sysadmin@localhost:~$ ps
PID TTY          TIME CMD
   80 ?        00:00:00 bash
   94 ?        00:00:00 ps
```

PID	Process identifier unique to each process.
TTY	Name of the terminal or pseudo-terminal where the process is running.
TIME	Total processor time used by the process.
CMD	Command that started the process.

Listing Processes

- Use the -e (every) and -f (full) options with the ps command to list all processes on the system:

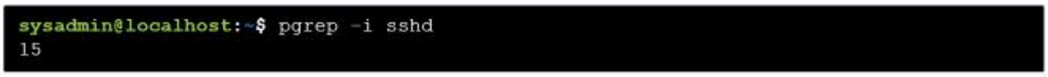
```
ps -ef
```



```
sysadmin@localhost:~$ ps -ef
UID          PID  PPID  C STIME TTY          TIME  CMD
root           1     0  0 17:16  ?       00:00:00  /sbin??? /init
syslog        33     1  0 17:16  ?       00:00:00  /usr/sbin/rsyslogd
root          38     1  0 17:16  ?       00:00:00  /usr/sbin/cron
root          40     1  0 17:16  ?       00:00:00  /usr/sbin/sshd
bind          57     1  0 17:16  ?       00:00:00  /usr/sbin/named -u bind
root          70     1  0 17:16  ?       00:00:00  /bin/login -f
sysadmin      80    70  0 17:16  ?       00:00:00  -bash
sysadmin      96    80  0 17:26  ?       00:00:00
```

Searching For Processes

- The pgrep command looks for the specified names or other criteria of processes.
- The -i option makes the pgrep command case insensitive.
- For example, to find all instances of the sshd command running on the system:

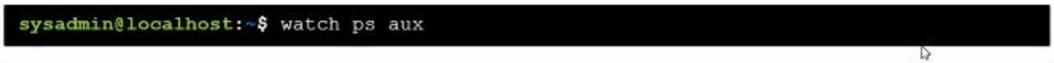


```
sysadmin@localhost:~$ pgrep -i sshd
15
```

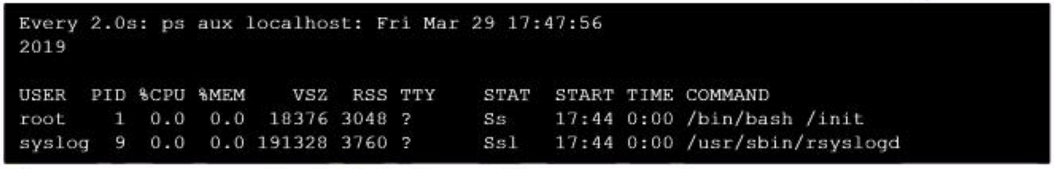
- The -l option can be used with the pgrep command to list the process name along with the PID
- The -u option can be used to search for processes owned by a specific user.

Watching Processes

- The watch command can be used to monitor recurring processes.
- The watch command can be used with the ps command to monitor running processes in the shell:



```
sysadmin@localhost:~$ watch ps aux
```



```
Every 2.0s: ps aux localhost: Fri Mar 29 17:47:56
2019

USER   PID %CPU %MEM    VSZ  RSS TTY    STAT  START TIME COMMAND
root     1  0.0  0.0  18376 3048 ?      Ss    17:44 0:00 /bin/bash /init
syslog  9   0.0  0.0 191328 3760 ?      Ssl   17:44 0:00 /usr/sbin/rsyslogd
```

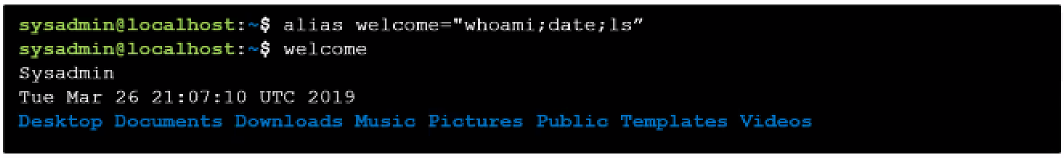
- The watch command's default interval is two seconds.
- To change the interval, use the -n option.

Executing Multiple Commands

- It can be useful to execute two or more commands in a single command line.
- By using the semicolon ; character as a delimiter between commands, a user can type multiple commands on one command line:

```
COMMAND;COMMAND[;COMMAND]...
```

- For example, to create an alias called `welcome` which outputs the current user, the date, and the current directory listing, execute the following command:



```
sysadmin@localhost:~$ alias welcome="whoami;date;ls”
sysadmin@localhost:~$ welcome
Sysadmin
Tue Mar 26 21:07:10 UTC 2019
Desktop Documents Downloads Music Pictures Public Templates Videos
```

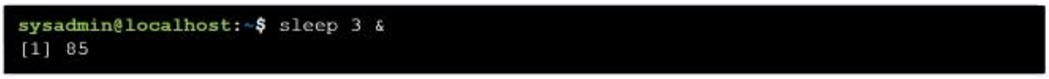
Foreground Processes

- A foreground process is one that prevents the user from using the shell until the process is complete.
- When one process starts another, the first process referred to as the *parent* process and the new process is called a *child* process.

Background Processes

- When executed in the background, a child process releases control back to the parent process
- To have a command execute as a background process, add the ampersand `&` character after the command:

```
sleep 3 &
```

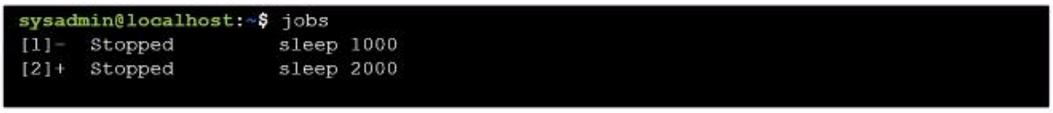


```
sysadmin@localhost:~$ sleep 3 &
[1] 85
```

- When executing commands in the background, it outputs a *job number* followed by a space and then the *process identification number (PID)*.

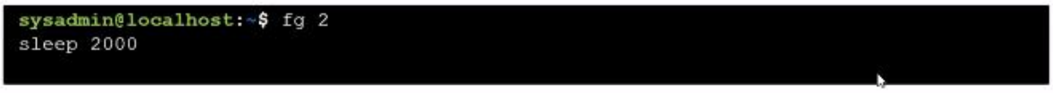
Managing Jobs

- The `jobs` command displays background jobs:

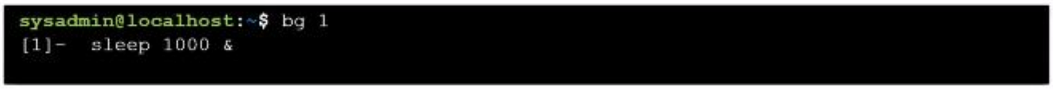


```
sysadmin@localhost:~$ jobs
[1]-  Stopped          sleep 1000
[2]+  Stopped          sleep 2000
```

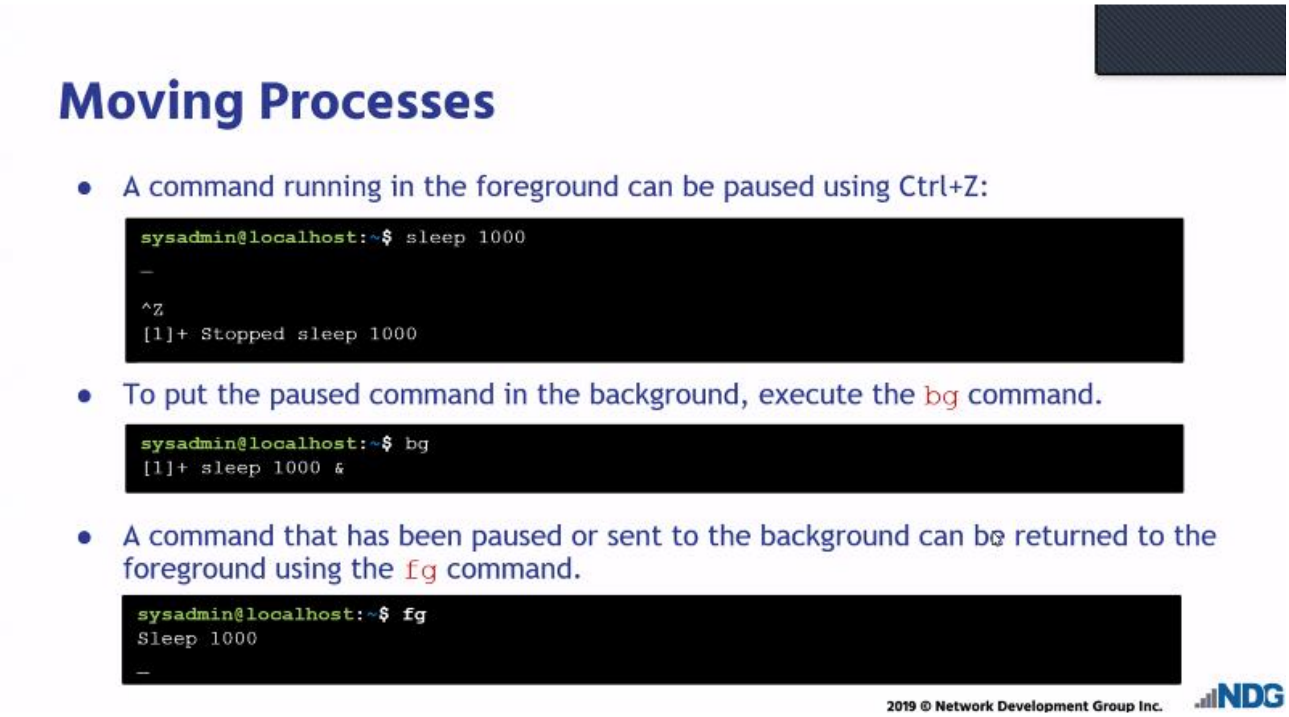
- The `fg` (foreground) and `bg` (background) commands provide the ability to multi-task:



```
sysadmin@localhost:~$ fg 2
sleep 2000
```



```
sysadmin@localhost:~$ bg 1
[1]-  sleep 1000 &
```



Moving Processes

- A command running in the foreground can be paused using Ctrl+Z:

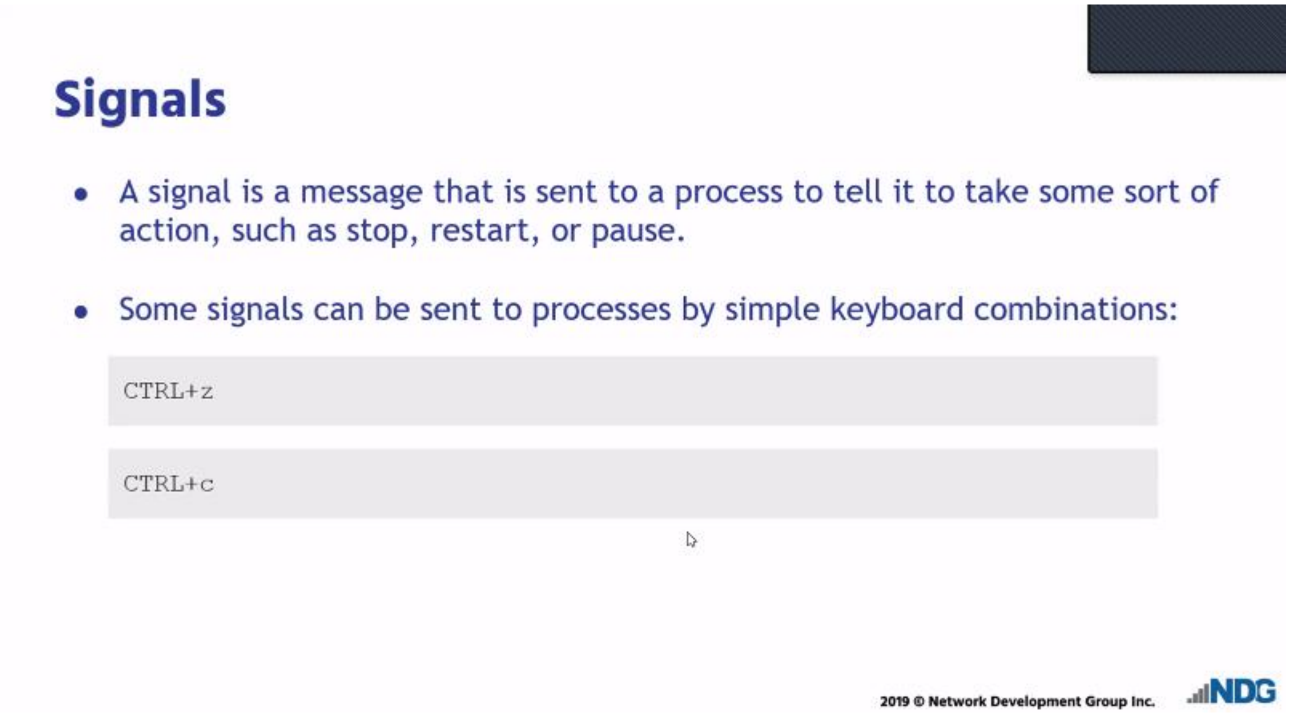
```
sysadmin@localhost:~$ sleep 1000
_

^Z
[1]+ Stopped sleep 1000
```

- To put the paused command in the background, execute the `bg` command.

```
sysadmin@localhost:~$ bg
[1]+ sleep 1000 &
```

- A command that has been paused or sent to the background can be returned to the foreground using the `fg` command.

```
sysadmin@localhost:~$ fg
Sleep 1000
_
```



Signals

- A signal is a message that is sent to a process to tell it to take some sort of action, such as stop, restart, or pause.

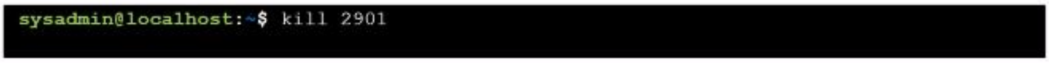
- Some signals can be sent to processes by simple keyboard combinations:

```
CTRL+z
```

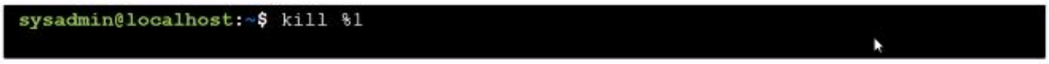
```
CTRL+c
```

Signals

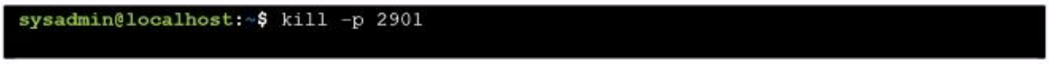
- The `bg` command sends a process a signal to execute in the background.
- To list all signals, use the `kill -l` command.
- To send a process a signal, use the `kill` command followed by the PID# or Job# (prefixed with the percent % sign):



```
sysadmin@localhost:~$ kill 2901
```



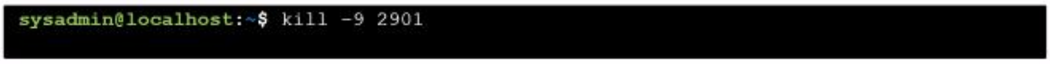
```
sysadmin@localhost:~$ kill %1
```



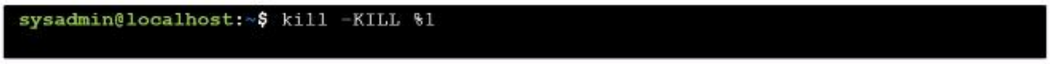
```
sysadmin@localhost:~$ kill -p 2901
```

Force Kill

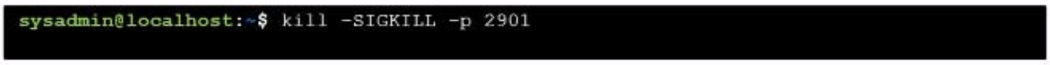
- If other signals have failed to end a process, use the `SIGKILL` signal to force the process to end:



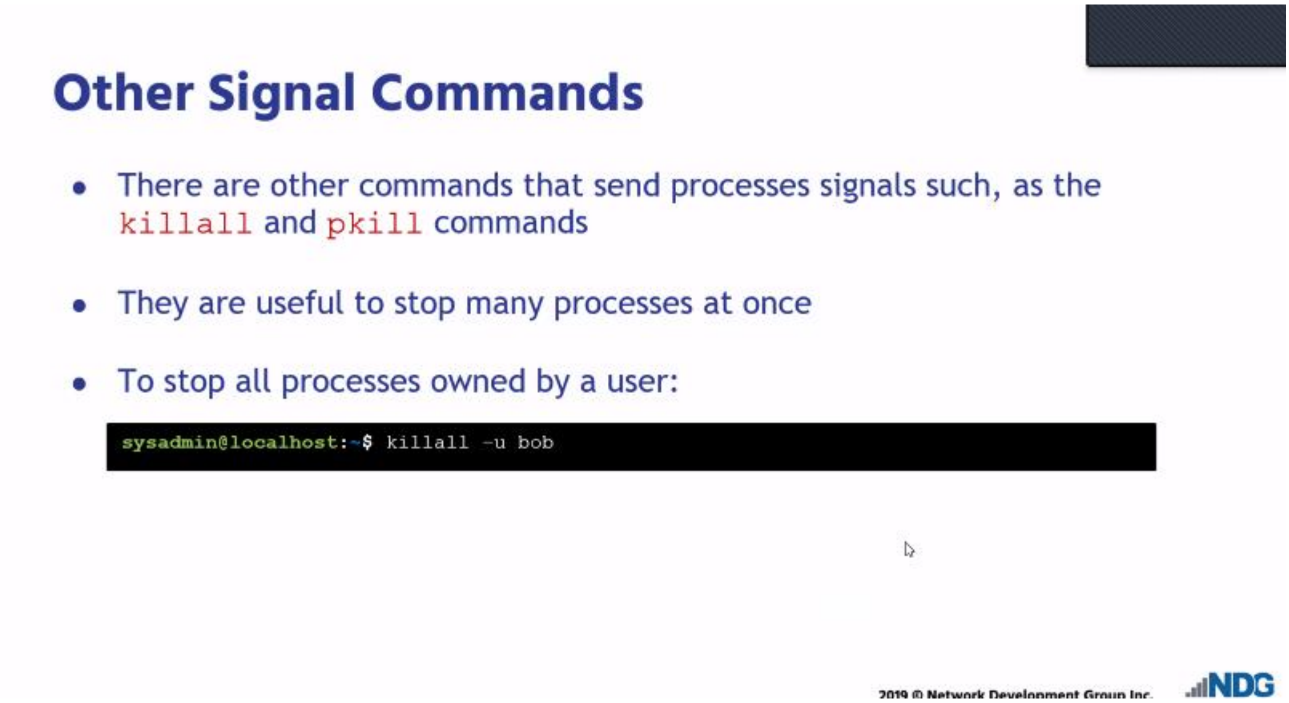
```
sysadmin@localhost:~$ kill -9 2901
```



```
sysadmin@localhost:~$ kill -KILL %1
```



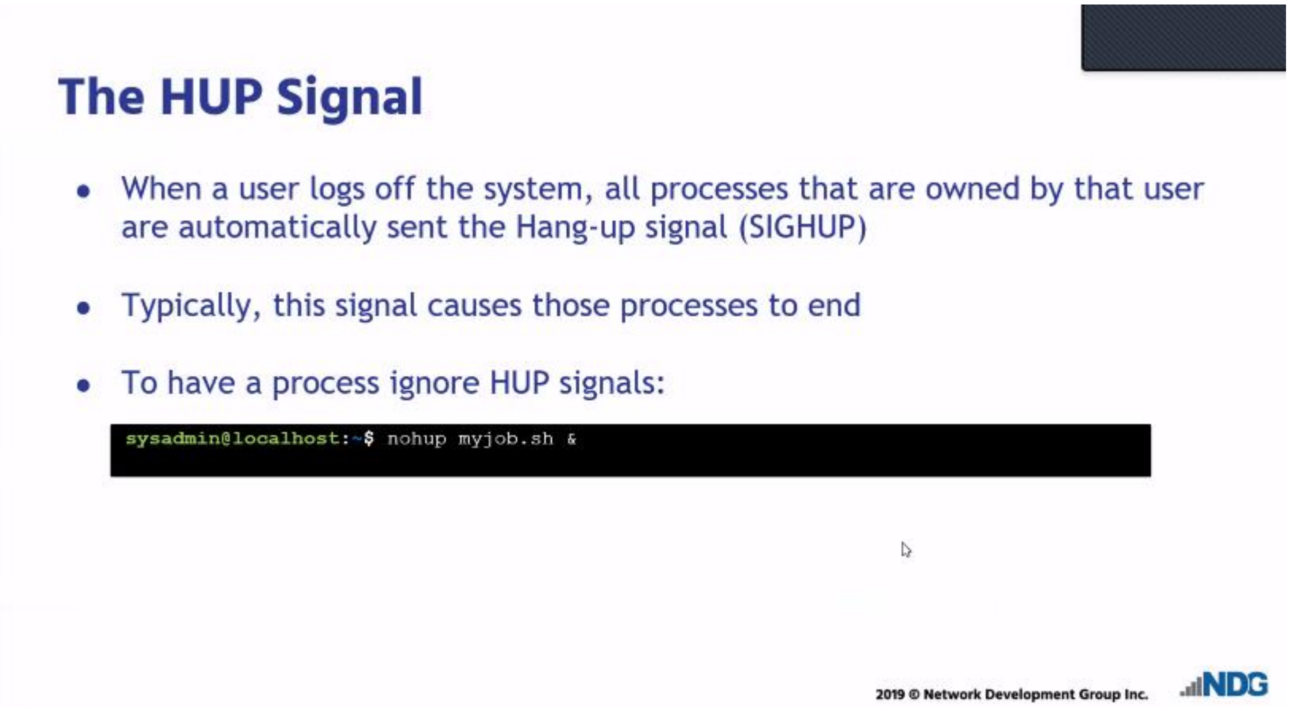
```
sysadmin@localhost:~$ kill -SIGKILL -p 2901
```



Other Signal Commands

- There are other commands that send processes signals such, as the `killall` and `pkill` commands
- They are useful to stop many processes at once
- To stop all processes owned by a user:

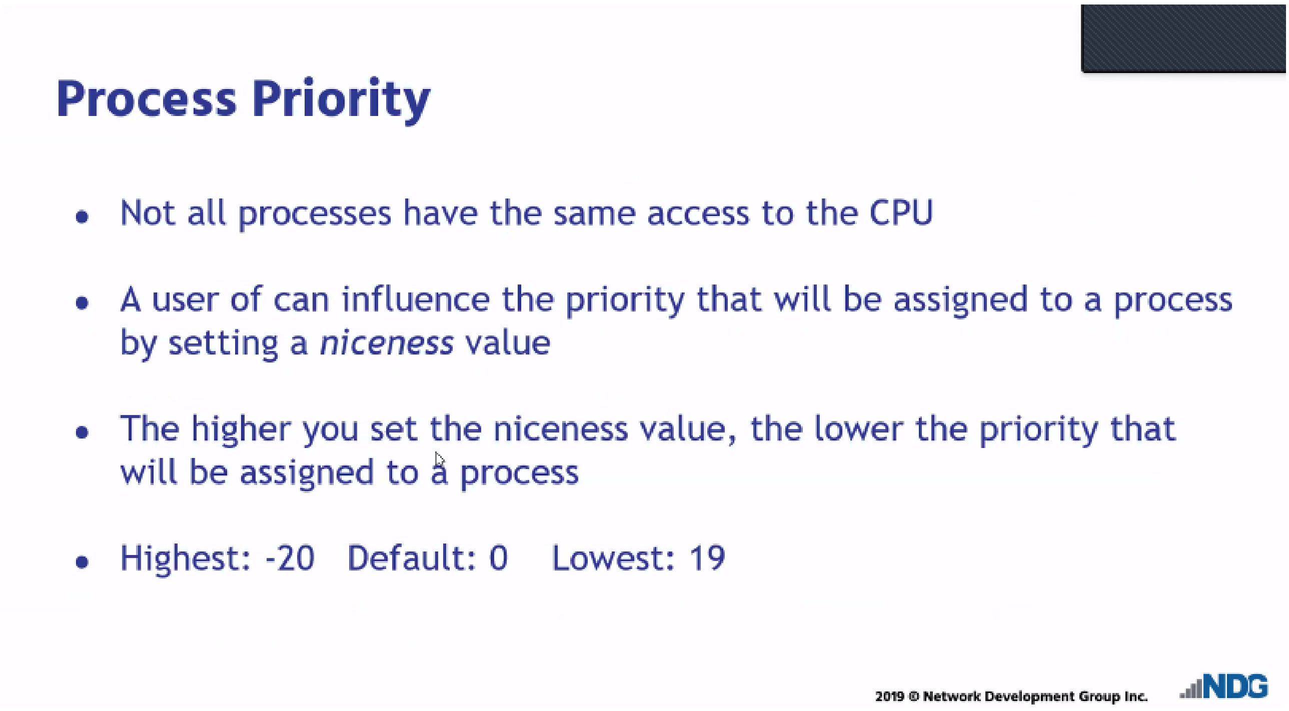
```
sysadmin@localhost:~$ killall -u bob
```



The HUP Signal

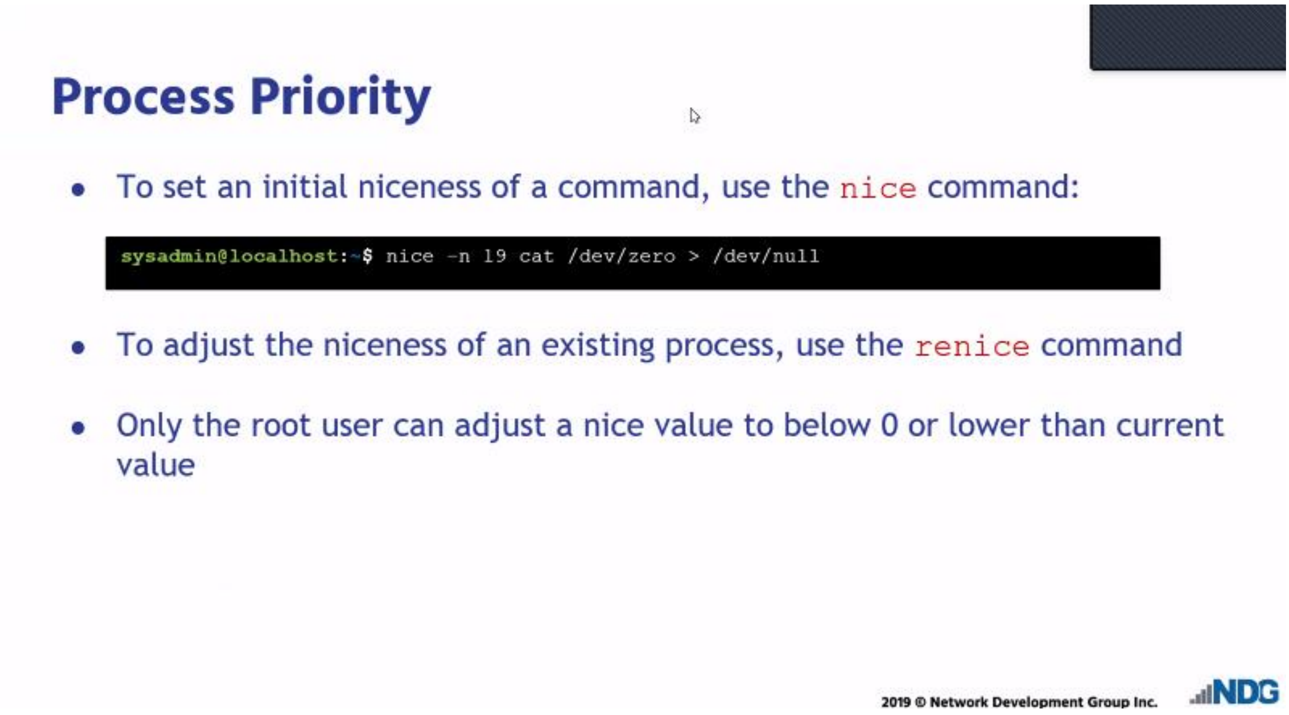
- When a user logs off the system, all processes that are owned by that user are automatically sent the Hang-up signal (SIGHUP)
- Typically, this signal causes those processes to end
- To have a process ignore HUP signals:

```
sysadmin@localhost:~$ nohup myjob.sh &
```



Process Priority

- Not all processes have the same access to the CPU
- A user of can influence the priority that will be assigned to a process by setting a *niceness* value
- The higher you set the niceness value, the lower the priority that will be assigned to a process
- Highest: -20   Default: 0   Lowest: 19



Process Priority

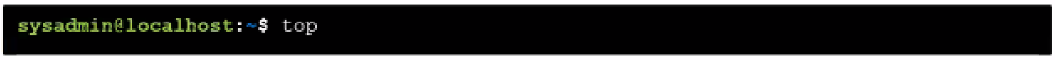
- To set an initial niceness of a command, use the `nice` command:

```
sysadmin@localhost:~$ nice -n 19 cat /dev/zero > /dev/null
```

- To adjust the niceness of an existing process, use the `renice` command
- Only the root user can adjust a nice value to below 0 or lower than current value

Monitoring Processes

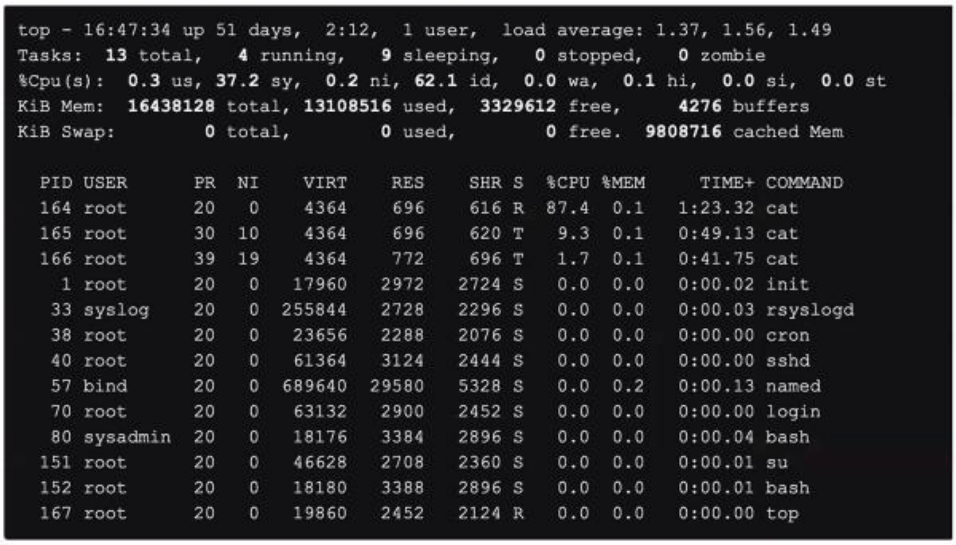
- The `top` command provides the ability to monitor processes in real-time, as well as manage the processes.



```
sysadmin@localhost:~$ top
```

- Monitors processes in real time using the interactive keys in `top`:
  - Press `h` to see all available options
  - Press `r` for renice
  - Press `q` to quit

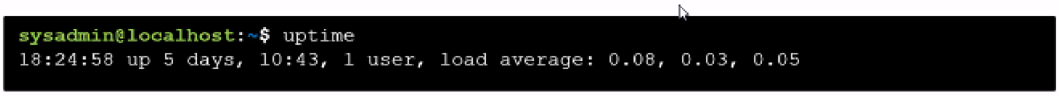
Monitoring Processes



```
top - 16:47:34 up 51 days,  2:12,  1 user,  load average: 1.37, 1.56, 1.49
Tasks:  13 total,   4 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us, 37.2 sy,  0.2 ni, 62.1 id,  0.0 wa,  0.1 hi,  0.0 si,  0.0 st
KiB Mem:  16438128 total, 13108516 used,  3329612 free,     4276 buffers
KiB Swap:        0 total,        0 used,        0 free.  9808716 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  164 root      20   0    4364    696    616 R  87.4  0.1   1:23.32 cat
  165 root      30  10    4364    696    620 T   9.3  0.1   0:49.13 cat
  166 root      39  19    4364    772    696 T   1.7  0.1   0:41.75 cat
    1 root      20   0   17960   2972   2724 S   0.0  0.0   0:00.02 init
   33 syslog    20   0  255844   2728   2296 S   0.0  0.0   0:00.03 rsyslogd
   38 root      20   0   23656   2288   2076 S   0.0  0.0   0:00.00 cron
   40 root      20   0   61364   3124   2444 S   0.0  0.0   0:00.00 sshd
   57 bind      20   0  689640  29580   5328 S   0.0  0.2   0:00.13 named
   70 root      20   0   63132   2900   2452 S   0.0  0.0   0:00.00 login
   80 sysadmin  20   0   18176   3384   2896 S   0.0  0.0   0:00.04 bash
  151 root      20   0   46628   2708   2360 S   0.0  0.0   0:00.01 su
  152 root      20   0   18180   3388   2896 S   0.0  0.0   0:00.01 bash
  167 root      20   0   19860   2452   2124 R   0.0  0.0   0:00.00 top
```

Monitoring the System

- The uptime command displays:
  - The current time
  - The amount of time the system has been running
  - The number of users who are currently logged in
  - The load averages during the past one, five and fifteen minute.



```
sysadmin@localhost:~$ uptime
18:24:58 up 5 days, 10:43, 1 user, load average: 0.08, 0.03, 0.05
```