# Unix Command Line Basics

Linux System Administration
Fall 2023

# bash command line

- Default shell on modern Linux
  - Editing, history, autocompletion
- Built-in commands
  - if, for, while, case, select, alias, ...
- Parameter manipulation
- Shell variables
- Compound with | and ;

# Bash startup Files

https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

- Interactive login
  - `/etc/profile`
  - First existing from
    `~/.bash_profile`
    `~/.bash_login`
    `~/.profile`
- Interactive non-login shell
  - `~/.bashrc`
- Non-interactively
  - Defined by `$BASH_ENV`

- Interactive login should explicitly source `~/.bashrc` in profile
- Profile typically sets `$PATH` and `umask`
- `~/.bashrc` sets most things
  - prompt (`$PS1`)
  - terminal (`$TERM`)
  - aliases (`~/.bash_aliases`)
  - other variables as needed

# ssh (secure shell)

- replaces telnet & rlogin/rsh
  - encrypted connection
  - basis for scp, rsync, etc
- Our go-to tool for access
  - Major goal of exams
  - ssh tunnel (-L and -R)
- Password-less login
  - ~/.ssh/authorized_keys

- `ssh user@host`
- Server side sshd
- Default port 22

  -p for other port
- `ssh user@host 'command'`
- Forward X11 with -X or -Y
- Debug with -vvvv
- root logins may be disabled

# Editing the command line

- Left and Right arrow, Home and End moves cursor
- Backspace and delete
- Tab completes file name
  - Can be context sensitive
- Ctrl-C aborts editing
- Up and down arrows scroll through the command history
- Ctrl-R searches command history for pattern
  - Ctrl-R searches for next
  - ESC to stop and edit
  - ENTER to stop and execute

# Script arguments

- Positional parameters
  - **`$0`** name of script
  - **`$`*`n`*** parameter *n*
  - **`$*`** all parameters from 1
- Wildcard expansion is done by the shell
  - Script sees expanded

- Useful shell parameters
  - `$EUID` effective UID
    - 0 when run by sudo
  - `$PWD` current directory
  - `$HOSTNAME` host
  - `$LINES` window height
  - `$COLUMNS` window width

# Compound commands

- Unix approach:  commands should do one thing well. Compound commands allow complex behavior
- Separate with ;
```
for file in *.txt;do mv $file $file.0;done
```
- Pipeline with |
```
awk -F: '{print $7}' /etc/passwd|sort|uniq -c
```
- Convert with xargs
```
grep -l -dskip foo *|xargs sed -i s/foo/bar/
```

# One liners

- Login shells in /etc/passwd

```
awk -F: '{print $7}' /etc/passwd |sort|uniq -c
```

- Do something on multiple hosts

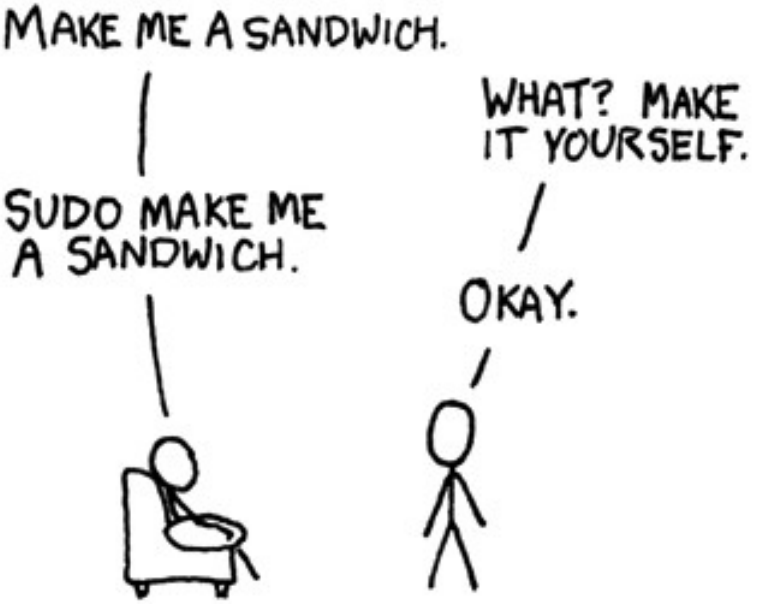
```
for host in foo bar;do ssh $host dnf install mlocate;done
```

- Do something on specific files

```
file *|grep CRLF|sed 's/: .*//'|xargs -l dos2unix
```

# sudo

- Run command as root
  - Training wheels for root
  - Can limit commands
  - Provides audit trail
- /etc/sudoers
  - Group **sudo** or **wheel**
- ***USE IT!!!***
  - D.F.I.U.




https://xkcd.com/149/

# Files and directories

- Everything is a file
- Unified directory tree
  - / is the directory separator
  - file systems mount as directories
- File names are arbitrary strings
  - White space in file names are a nightmare
  - Extensions are not definitive

# Special file types

- directory - list of file names and inode pointers
  - hard link - directory entry to existing inode
- symbolic link - path to another file
- pseudo files
  - FIFO (named pipe)
  - socket
  - device file (block or character)

# Where are stuff?

`/boot` – boot loader files

`/etc` – system configuration

`/bin` – essential binaries (`ls, cat , cp, mv`)

`/sbin` – essential system binaries (`mount, mkfs, fsck, modprobe`)

`/usr` – user utilities and files (`/usr/bin, /usr/share, /usr/lib`)

`/var` – variable files (`/var/log, /var/mail, /var/spool`)

`/tmp` – temporary files (cleaned out on boot)

`/dev` – device files

`/proc` – virtual system files (kernel access)

`/home` – user data

# Manipulating the file system

- Directories
  - `cd` - change
  - `ls` - list
  - `pwd` - show current
  - `mkdir` - create new
  - `rmdir` - remove
- Files
  - `cp` - copy
  - `mv` - move
  - `rm` - remove
  - `ln` - link
  - `cat` - concatenate
  - `scp` - remote copy
  - `rsync` -remote sync

# ls flags

-l  long listing

-a also show dot files

-h show human format

-t  sort by time

-S sort by size

-d show directory

-i  show inode

-r  reverse

-1 list one file per line

- By default, files starting with . (period) are not shown
- List file by type

  --color  color code

  -F append */=>@|
- Show files in human readable format ordered by time in reverse

  ls -ltrh

# Special directory names

/ root directory

. current directory `$PWD`

.. parent directory

~ home directory `$HOME`

\- previous directory on cd `$OLDPWD`

# Moving files locally

- Copy file(s)

  `cp` *`from to`*

  -i prompt on overwrite

  -a preserve attributes

  -r recursive

  Copy multiple files if *to* is a directory

- Move file or directory

  *`mv from to`*

  -i prompt on overwrite

  Preserves attributes by default

  Move multiple files if *to* is a directory

# Copying files remotely

- `scp` *`from to`*
  - -p preserve attributes
  - -r recursive
  - -P port
- `rsync` *`from to`*
  - -a preserve attributes
  - -r recursive
  - --delete-exlcuded

- Layered on ssh
- Specify remote as
  `user@host:dir`
- rsync copies only when changed files very efficiently

# Locating files

- `find` *`dirs`*
  - Search directories

    `-name` *`pattern`*

    `-iname` *`pattern`*
  - Lots of options to find by type, time, permissions, etc.
- `which` *`command`*
  - Uses $PATH to find executable for *command*
- `locate` *`pattern`*
  - Not installed by default
  - Database of files
    - updatedb scans filesystems

# Finding out more about files

- `file`
  - Determines file properties by examining the contents of the file
  - Extensions are not definitive
- `stat`
  - Shows details of the file returned by stat() system call
    - Permissions
    - Disk usage
    - Times

# Comparing files

- `diff file1 file2`
  - compare two files
- `diff –r dir1 dir2`
  - compare two trees
- `sdiff file1 file2`
  - show differences side-by-side
- `zdiff file1 file2.gz`
  - compare gzipped files

**Useful diff flags**

-q only report that files differ

-i ignore case

-b ignore changes in the amount of whitespace

-w ignore all white space

-Z ignore trailing whitespace

-B ignore blank lines

# File permissions

- Owner:Group:Other
  - r - read (4)
  - w - write (2)
  - x - execute (1)
    - Directory x=list
- Example
  - rwxr-xr--

- `ls -l`
  - show permissions
- `chmod`
  - `chmod ug=rwx`
  - `chmod 775`
    - Values are octal
  - `chmod g+w`

# Showing what is in a file

- `cat` – show all
- `more` - paginate
- `head` – first n lines
- `tail` – last n lines

  -f to follow changes

- `grep pattern file(s)`
  - lines with pattern

    -v to invert selection
- `awk action file(s)`
  - Show users in passwd

    ```
    awk –F: '{print $1}' /etc/passwd
    ```

# aliases

- Creates new or modifies commands
- Applies when it is the first word of a simple command
- Expanded by bash
- `~/.bash_aliases`

- `alias cp='cp -i'`
- `alias mv='mv -i'`
- `alias rm='rm -i'`
- `alias ll='ls -l'`
- `alias vi='vim'`
- `alias -p`
- `unalias ls`

# Bash if and for

```
for x in *
do
  if [ -f "$x" ]
  then
    echo "$x is a file"
  else
    echo "$x is something else"
  fi
done
```

- Loop over all files
- If it is of type file
- say it is a file
- else
- say it is something else

# if

### Conditional action

```
if condition
then
   commands
fi
```

### Alternative actions

```
if condition
then
   commands
else
   commands
fi
```

### Nested decisions

```
if condition
then
   commands
   if condition
   then
   commands
   else
   commands
   fi
else
   commands
fi
```

### Cascading decisions

```
if condition
then
   commands
elif condition
then
   commands
else
   commands
fi
```

# Conditional expressions

- `if [ $x == 0 ]`
  - Is the string '0'
- `if [ $x -gt 0 ]`
  - Is $x positive
- `if [ "$x" != abc ]`
  - Is the string not abc
- `if [ -z "$x" ]`
  - Is this a blank string
- `if [ -r foo.txt ]`
  - Is foo.txt readable

- Hints
  - Beware string vs. numeric comparisons
    - Bash defaults to strings
  - Whitespace matters
  - Quote strings that may be blank
  - 0 means true
- Lots more on man page
  - CONDITIONAL EXPRESSIONS

# case

```
case $variable in
pattern1)
  commands;;
pattern2)
  commands;;
patternN)
  commands;;
*)
  commands;;
esac
```

- Hints
  - Comparison is based on string matching
  - Wildcards
    - * matches anything
  - Clauses are done in order
    - First match wins
  - No action if none match
    - Can be a convenient if

# Iteration

- **`for`** `var` **`in`** `list`**`; do`** `commands`**`; done`**
  - Execute commands for values in list
- **`while`** `cmd1`**`; do`** `cmd2`**`; done`**
  - Do cmd2 while cmd1 returns true
- **`until`** `cmd1`**`; do`** `cmd2`**`; done`**
  - Do cmd2 until cmd1 returns true

# Return values

- Generally 0 means everything is OK
  - This is just like most system function calls
- Return (exit) value of the last command is `$?`
- 0 is true in bash
- Example:  Can we ping 8.8.8.8?

```
ping –c 1 8.8.8.8
if [ $? ]; then
   echo "8.8.8.8 pings OK"
fi
```

# Whitespace Matters

- Assignment

  `date=3` vs. `date = 3`

- Expressions

  `if [ $? ];then` vs. `if [$?];then`

- Range

  `for x in {1..5}` vs. `for x in {1 .. 5}`

# Quotes matter

```
X=`date`
for i in {1..3}
do
   echo "$X"
   sleep 1
done
```

```
X="date"
for i in {1..3}
do
   echo `$X`
   sleep 1
done
```

```
X=`date`
for i in {1..3}
do
   echo '$X'
   sleep 1
done
```

```
X='date'
for i in {1..3}
do
   echo `$X`
   sleep 1
done
```

# Bash arithmetic

```
x=5
y=6
z=$x+$y
echo $z
```

5+6

```
((z=$x+$y))
echo $z
```

11

- Hints:
  - Bash treats variables as strings
  - Double parentheses means it is an arithmetic expressions
    - `((z=$x+$y))`
    - `z=$(($x+$y))`
  - Result is a string to bash

# Scripts

- Comment start with #
- Initial line #! invokes an interpreter
  - `#!/bin/bash` - bash
  - `#!/usr/bin/python3` - python 3
- File must be executable
- **You should get good at bash and another scripting language like Python or Perl**

# bash arrays

- Initialize

```
days=('M' 'W' 'F')
```

- Assign

```
days[2]='Saturday'
```

- Append

```
days+='Sunday'
```

- Access one element

```
echo ${days[1]}
```

- Access all elements

```
echo ${days[*]}
echo ${days[@]}
```

- Number of elements

```
echo ${#days[*]
```

- Index of elements

```
echo ${!days[*]}}
```

- Declaration

```
declare -a days
```

- Associative arrays

```
declare -A name
```

# Bash functions

- Definition

```
myfunc () {commands;}
```

or

```
function myfunc {commands;}
```

- Arguments are by position
  - `$1 $2 $3`
- Return value in `$?`
- Use `local` to make variables local to function

# Parameter expansion

- Isolate variable
  - ${*variable*}
  - `dir/${file}a.txt`
- Substring
  - `${name:off:len}`
- Substitution
  - `FILE=foo.txt`
  - `${FILE/txt/pdf}` = foo.pdf
- Remove prefix
  - `${parameter#word}`
- Remove suffix
  - `${parameter%word}`
- Set default value
  - `${parameter:-value}`
- Parameter length
  - `${#parameter}`

# Basic networking

- `ifconfig` configure interfaces
- `route` configure routing
- `ip` configure networking
- `ping` test connectivity
- `traceroute` test routing
- `dig` test DNS
- `nslookup` test DNS

# Downloading files

- Get data from a web or ftp server
  - ftp supports wildcards
- `wget`

  -m is great for mirroring entire directory tree
- `curl` (cURL)
  - Supports many more protocols