UNIX System Programming

Shell programming

UNIX – Linux…

- UNIX is the name of the proprietary operating system licensed through a number of separate vendors (Bell labs).
- Linux is an operating system which, in its initial form, is meant to make available a free UNIX type of environment.
- In this course we will use UNIX and Linux interchangeably or just call it *NIX.

The *NIX shell

- A shell is a program that allows you to interact with the operating system. Just like your Windows or MAC desktop gives you access to the system.

- As we will see, all processes in *NIX inherit three file descriptors from their parent: STDIN, STDOUT and STDERR. The shell is no different.

The *NIX shell

- When you connect to the server through ssh, it creates a new process to deal with the interaction between your computer and the target host.
- Shells are just running programs. In our case, the program doesn’t recognize mice or know that it is in a window (if using ssh).
- It only knows it has a display and a keyboard
- It does this by connecting your remote keyboard to the STDIN for the shell and both STDOUT and STDERR are connected to your remote display.

The *NIX shell

- Instead of clicking on icons, a shell requires that you type in commands (verbs).
- Each verb will perform a different action just like clicking on different icons on your desktop.
- Sometimes you have to tell the shell program where to go find the verb you are attempting to run.
- For some standard commands, no path to the executable is required.

The *NIX shell

- As you issue commands, the shell will search your PATH variable for an executable by that name.
  - If it finds one, it creates a new process to run the command.
  - If it cannot find an executable which matches the verb, it returns an error message.

- Though there is a basic set of universal commands recognized by *NIX: *ls, mkdir, ps*….
- Each flavour of shell deals with scripting and variables names in a slightly different way.

The *NIX shell

- The shell can also allow for the creation of “environment” variables.
- The default set of these variables contain information that is used by the various programs that you run.
- You can use the *env* command to get a list of existing variables (for the current process).
- You can also create and use variables of your own.
- When you use a variable, it is prefixed by a $. (e.g. $HOME and $USERNAME)

The *NIX shell

UNIX does not, for the most part, recognize what in the "windows" world is know as a file extension.

In UNIX, a file is a file is a file.

That means that to UNIX, a file is just a collection of bytes.

What makes a file "executable". i.e. what makes a file something that UNIX recognizes as a set of instructions that can be executed?

*NIX shell

File permissions:

For a file to be seen as something the shell might try to run, you must first turn on the “executable” flag for the file.

For compiled code, such as the code you will write in C for this course, the compiler does this for you.

```
[jacques@loki sample_code]$ ls -lt hello*
-rwxrwxr-x. 1 jacques jacques 8520 Jul 24 11:49 hello
-rw-rw-r--. 1 jacques jacques   69 Jul 24 11:49 hello_world.c
```

*NIX shell

When you want to use a scripting language, there are two thing you need to do:

1) Set the executable flag yourself
2) Include in the first line of the script, an identifier of the program you want to use as the command interpreter for your script.

For option (1), you can use the *chmod* command.

```
[jacques@loki lab1]$ ls -lt
-rw-rw-r--. 1 jacques jacques 130 Jul 12 11:38 find_top_10.sh

[jacques@loki lab1]$ chmod u+x find_top_10.sh

[jacques@loki lab1]$ ls -lt
-rwxrw-r--. 1 jacques jacques 130 Jul 12 11:38 find_top_10.sh
[jacques@loki lab1]$
```

*NIX shell

Defining the command interpreter for your scripting language is done by using the #! (she-bang) prefix on the first line of your code.

She-bang is then followed by the full path of the interpreter for the language you selected.

You can find the full path by using the which command followed by the name of the language you want to use

```
[jacques@loki lab1]$ which bash
/usr/bin/bash
[jacques@loki lab1]$ which csh
/usr/bin/csh
```

```
[jacques@loki lab1]$ which perl
/usr/bin/perl
[jacques@loki lab1]$ which php
/usr/bin/php
```

*NIX shell

Some examples of defining different interpreters:

```
[root@loki]# head -n 3 create_course_accounts.pl
#!/bin/perl -w
#
#  Script Name: Create_Course_accounts.pl


root@loki account_tools]# cat monitor_account_sizes.sh
#!/bin/bash -x
echo Top 20 account sizes in /home
echo " "
du  --summarize -c /home/* | sort -n -r | head -n 20
echo " "
```

NOTE: the "-x" or "--debug" on the /bin/bash line can be QUITE useful !!!

*NIX shell

Binaries (compiled code) also have their own “magic bytes” at the beginning of the file. This tells the interpreter what type of binary it is:

```
[jacques@loki sample_code]$ ls -lt shared
-rwxrwxr-x. 1 jacques jacques 13120 Aug  1 09:58 shared

[jacques@loki sample_code]$ hexdump -C shared | head -n 10
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 3e 00 01 00 00 00  d0 06 40 00 00 00 00 00  |..>.......@.....|
00000020  40 00 00 00 00 00 00 00  c0 2b 00 00 00 00 00 00  |@........+......|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1e 00 1b 00  |....@.8...@.....|
00000040  06 00 00 00 05 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000050  40 00 40 00 00 00 00 00  40 00 40 00 00 00 00 00  |@.@.....@.@.....|
00000060  f8 01 00 00 00 00 00 00  f8 01 00 00 00 00 00 00  |................|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
00000080  38 02 00 00 00 00 00 00  38 02 40 00 00 00 00 00  |8.......8.@.....|
00000090  38 02 40 00 00 00 00 00  1c 00 00 00 00 00 00 00  |8.@.............|
[jacques@loki sample_code]$
```

* ELF => Executable and Linkable Format

Shell commands

- The list of commands that you can use is quite large.
- There are probably a dozen or so that you will use all the time:
  - ls – list files
  - cd – change directory
  - gcc – invoke the GNU C compiler
  - mkdir – make a new directory
  - zip – create an archive of multiple files
  - grep – search for data which meet a specific search criteria.
- You can get a list of commands using *apropos* and *man*.

Shell commands

A shell command is made up of 3 parts:

- The verb or command you are issuing
- switches or flags that you are including to alter the behaviour of the command.
- The arguments that the command will act on

Switches/flags: are typically optional

Arguments can be:

- Mandatory: single or multiple
- Optional

In the man pages optional items are placed in square brackets [ ].

Shell commands

The *NIX shell that you are using will provide to you a prompt. Just so you know it is ready for you to type something in.

On Loki, the prompt is: [jacques@loki 3380]$

- An open square bracket: [
- your username
- @
- The name of the host (Loki)
- a space
- The name of the directory you are in
- A close square bracket:]
- A symbol to show your security access (for lack of a better description). Either "$"==Normal or "#"==root

Shell Commands

As an example the ls command to list your files:

```
[jacques@loki mid-term]$ ls
extracts  mail_all_extracts.sh  midterm_mailer.pl  w2021_3380_Midterm_email_test.csv
jb.tmp    make_individual_user_extracts.sh  README.TXT         wip
```

with some switches:

```
[jacques@loki mid-term]$ ls -lt
total 408
-rw-rw-r--. 1 jacques jacques    503 Mar 10  2021 README.TXT
-rwxrw-r--. 1 jacques jacques   1937 Mar 10  2021 midterm_mailer.pl
-rw-rw-r--. 1 jacques jacques   8416 Mar 10  2021 jb.tmp
drwxrwxr-x. 2 jacques jacques   4096 Mar 10  2021 extracts
-rwxrw-r--. 1 jacques jacques    440 Mar 10  2021 mail_all_extracts.sh
drwxrwxr-x. 2 jacques jacques   4096 Mar 10  2021 wip
-rwxrw-r--. 1 jacques jacques    549 Mar 10  2021 make_individual_user_extracts.sh
-rwx------. 1 jacques jacques 380106 Mar 10  2021 w2021_3380_Midterm_email_test.csv
[jacques@loki mid-term]$
```

Shell Commands

With a parameter:

```
[jacques@loki mid-term]$ ls README.TXT
README.TXT
```

With both switches and parameters:

```
[jacques@loki mid-term]$ ls -lt README.TXT
-rw-rw-r--. 1 jacques jacques 503 Mar 10  2021 README.TXT
[jacques@loki mid-term]$
```

The copy command with switches and two parameters:

```
[jacques@loki mid-term]$ cp -v README.TXT readme.copy
‘README.TXT’ -> ‘readme.copy’
[jacques@loki mid-term]$
```

STDIN, STDOUT and STDERR

- Might also be mentioned as SYSIN, SYSOUT and SYSERR
- These are the three I/O channels opened for you by default when you create a process.
- You can, for commands that you issue from your shell, change the default target of these file descriptors.
- This is known as redirection.

Redirection and PIPEs

- You can use the > and/or the < symbols as operators to redirect STDIN and STDOUT.

e.g.

- `command < input_filename`
- `command > output_filename`

e.g. who –a > all_logged_in_users.txt

- In addition, you can use the PIPE operator | to tie the STDOUT of one command to the STDIN of another command

e.g.

```
ls -lt | head  -n 10
```

Data streams

One of the main philosophies behind UNIX is to write a piece of code once and make it flexible enough to be used in a variety of situations.

The idea behind this is that you can then deal with large volumes of data and extract different results using the exact same tools.

Once you develop familiarity with the tools, there's no need to go off and write applications specific to your current problem (in many cases).

Shell scripts

A shell script is simply a flat text file which contains a series of commands that you want to execute.

It is a “program” made up of normal command line verbs for the shell to interpret.

You do not need to include the system prompt inside your shell scripts.

The objective of the script file is to eliminate the repetition required to run a sequence of commands over and over again.

References:
https://www.tldp.org/LDP/abs/html/index.html
https://www.gnu.org/software/bash/manual/

Shell scripts - variables

Shell scripts have access to three types of variables:

1. Environment variables for the current process
    1. $LOGNAME
    2. $HOME
    3. $HOSTNAME

2. Parameters passed to the script via the command line.

```
if [ -z "$1" ]
 then
  echo “missing argument on the command line"
  exit 1;
fi
```

The –z tests the string to see if it is of zero length.

Shell scripts - variables

3. Shell scripts can build their own local variables from either the output of certain commands or by combining existing variables.

```
ARCHIVE_NAME=${FIRSTNAME}_${LASTNAME}_$1.zip
```

All variables are prefixed with a $. When combining existing variables it is safest to enclose the name in curly braces ${HOME}.

$1, $2, $3… are the command line parameters. The “words” which follow the script name on the command line

Note: There can’t be ANY spaces on either side of the equal sign.

Shell scripts - Conditionals

Shell scripts can also contain flow control structures:

```
if ! grep -q regex options; then
     printf '%s\n' 'myscript: Pattern not found!' >&2
     exit 1
fi
```

and:

internal field separator

```
cat cities.csv | while IFS=\, read country name lat long
do
   echo " $country --> city: $name ------> $lat $long "
done
echo "all done..."
```

The Shell Environment

Environment Variables:

```
jacques@UBU64vm:~$ env
SESSION=ubuntu
GPG_AGENT_INFO=/run/user/1000/keyring-ZtvBwH/gpg:0:1
TERM=xterm
SHELL=/bin/bash
…
USER=jacques
…
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
   sr/games:/usr/local/games
DESKTOP_SESSION=ubuntu
PWD=/home/jacques
HOME=/home/jacques
LOGNAME=jacques
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
```

The Shell Environment

PS1 and PS2: You can change your command prompt to a string or combinations of a number of preset shortcut values: (PS2 is used for line continuations after a \)

Typically, PS1 is set to: PS1="\u@\h> “

You can type: echo $PS1 to find its current setting.

```
jacques@UBU64vm:~$ PS1="\u> "
jacques> PS1="\h> “
UBU64vm> cd Music/
jacques@UBU64vm> PS1="\w> "
~/Music>
```

\u – Username

\h – Hostname

\w – Full pathname of current directory. Please note that when you are in the home directory, this will display only ~ as shown above

Note that there is a space at the end in the value of PS1. Personally, I prefer a space at the end of the prompt for better readability.

(from: http://www.thegeekstuff.com/2008/09/bash-shell-take-control-of-ps1-ps2-ps3-ps4-and-prompt_command/)

The Shell Environment

What shell am I running and what gets set by default?

```
jacques@UBU64vm> cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
…
pulse:x:115:122:PulseAudio daemon,,,:/var/run/pulse:/bin/false
jacques:x:1000:1000:jacques,,,:/home/jacques:/bin/bash
sshd:x:116:65534::/var/run/sshd:/usr/sbin/nologin
postfix:x:117:125::/var/spool/postfix:/bin/false
statd:x:118:65534::/var/lib/nfs:/bin/false
mysql:x:119:127:MySQL Server,,,:/nonexistent:/bin/false
jamie:x:1001:1001:Jamie Mitchell,113,,:/home/jamie:/bin/bash
cert:x:1002:1002::/home/cert:
```

Or:

```
grep -e jacques /etc/passwd !!!
```

The Shell Environment

Your shell environment allows you to “create” your own command names/verbs. These are called aliases.

Let’s say you always want to see the last 10 files modified in your current directory. Nice to have if your code creates files or if you can’t remember what you changed last. Maybe where you left off last session?

To do this, you would issue:

```
jacques@UBU64vm> ls -lt | head -n 10
```

The Shell Environment

It can get quite boring retyping this all of the time. Even retrieving the command using up arrow can be tedious.

In your shell environment, you can create an ALIAS for this command. You can then use the alias instead of retyping the whole line.

```
jacques@UBU64vm> alias t10="ls -lt | head -n 10"
```

You can then type in the verb "t10" when you want to see the top 10 last modified files.

The Shell Environment

Since shell scripting is just like writing programs, you can also define "functions" that can be used in your code.

Aliases are static in their definition.

Functions can be passed parameters.

let's say you know that a filename you created contains the word "test" in it.

At the command line you would type: ls –lt *test*

That's a lot of typing if you need to check for a number of different name patterns: test, exam or was it quiz?

The Shell Environment

In bash, you can define a function using this syntax:

```
functionName()  { the bash code you want to run including $1, $2…; }
```

In our example, we could write something like:

```
lt() { /bin/ls -lt *${1}*; };
```

we can then, at the command line, execute this function (much like an alias) but now, we can add parameters to the command line!

```
[jacques@loki ~]$ lt student
-rwxr--r--. 1 jacques jacques 2119 Oct  4 15:45 restore_student_account.sh
-rwxr--r--. 1 jacques jacques 2175 Oct  4 15:45 remove_student_account.sh

[jacques@loki ~]$ lt test
-rwxr--r--. 1 jacques jacques   858 Aug  9 13:05 mime_mail_test.pl
-rwxrwx---. 1 jacques jacques 18064 Jan 28  2021 test3.html
-rwxrwx---. 1 jacques jacques    59 Jan  5  2021 test.txt
-rwxrwx---. 1 jacques jacques   305 Sep 14  2020 test_colours.pl
[jacques@loki ~]$
```

The Shell Environment

When you first log onto Loki, the O/S automatically runs a "set-up" script for you.

It is called .bashrc **(notice the period which makes it a hidden file! use ls –a)**

You can define functions and aliases inside your .bashrc.

The definitions will therefore be reinstated every time you log into the server.

You can now customize your command line experience!

Just remember: it is NOT a good idea to incorporate aliases and functions defined in your .bashrc inside your independent shell script!!!

.bashrc

```
jacques@UBU64vm> ls -lt .bashrc
-rw-r--r-- 1 jacques jacques 3669 Nov  5 16:54 .bashrc

jacques@UBU64vm> cat .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac
…

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000
…
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'
alias t10='ls -lt | head -n 10'
```

The System Environment

```
jacques@UBU64vm> top

top - 09:20:49 up 19 min,  2 users,  load average: 0.01, 0.02, 0.05
Tasks: 327 total,   1 running, 326 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   2042668 total,   919764 used,  1122904 free,    66856 buffers
KiB Swap:  1046524 total,        0 used,  1046524 free.   374252 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1895 root      20   0  297476  49264  15744 S   1.3  2.4   0:05.27 Xorg
 2931 jacques   20   0  649060  18816  12432 S   0.7  0.9   0:02.35 gnome-term+
 2555 jacques   20   0  361584   4168   2908 S   0.3  0.2   0:00.89 ibus-daemon
 2655 jacques   20   0  205152   3312   2736 S   0.3  0.2   0:00.29 ibus-engin+
 2783 jacques   20   0 1293568  73520  38468 S   0.3  3.6   0:03.61 compiz
    1 root      20   0   33888   3256   1476 S   0.0  0.2   0:01.70 init
```

The System Environment

```
jacques@UBU64vm> jacques@loki ~]$ uname -a
Linux loki.trentu.ca 3.10.0-514.26.2.el7.x86_64 #1 SMP Tue Jul 4
   15:04:05 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
jacques@UBU64vm>

[jacques@loki ~]$ ls /etc -1 | grep -e release
centos-release
centos-release-upstream
os-release
redhat-release
system-release
system-release-cpe


[jacques@loki ~]$ cat /etc/centos-release
CentOS Linux release 7.3.1611 (Core)
```

The System Environment

The filesystem organization:

```
jacques@UBU64vm:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            987M  4.0K  987M   1% /dev
tmpfs           200M  1.2M  199M   1% /run
/dev/sda1        58G   25G   31G  45% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            5.0M     0  5.0M   0% /run/lock
none            998M  152K  998M   1% /run/shm
none            100M   36K  100M   1% /run/user
jacques@UBU64vm:~$
```

The System Environment

The file system table:

```
jacquesabeland@loki:~> cat /etc/fstab
/dev/sda1            swap                 swap       defaults            0 0
/dev/sda2            /                    ext3       acl,user_xattr      1 1
proc                 /proc                proc       defaults            0 0
sysfs                /sys                 sysfs      noauto              0 0
debugfs              /sys/kernel/debug    debugfs    noauto              0 0
devpts               /dev/pts             devpts     mode=0620,gid=5     0 0
/dev/sdb1            /home/common         ext3       acl,user_xattr      1 2
jacquesabeland@loki:~>
```

On Loki:

```
jacques@loki ~]$ cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Sun Feb 19 19:19:13 2017
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=e2c92a30-4550-4fbc-aeec-e67712e82084 /                  ext4    defaults        1 1
UUID=6fba1999-27ea-4faa-8523-f35ab22bc7cd /home              ext4    defaults,usrquota  1 2
UUID=6164653f-2cd5-4b92-89d7-9ac6f160d304 swap               swap    defaults        0 0
[jacques@loki ~]$
```

Basic Commands

```
jacques@UBU64vm> ls
AMS_0_1.gp                eicar.txt                    perl             SOM_BMU_Animation_gnuplot.7z
AMS_0_1.gp~               examples.desktop             Pictures         ssl-ccs-injection.nse
AMS_10_1000.gp            loki_logs                   Public           stellarium
…
Desktop                   Jamies_solution_Archive.zip  set_proxy.sh     t.sql
Documents                 linux_3.13.0-35.62.diff.gz   set_proxy.sh~    update-manager.sh
dosbox                    linux_3.13.0-35.62.dsc       shellshock       Videos
Downloads                 linux_3.13.0.orig.tar.gz     shellshock.txt   vmware_tools
eicar.com                 Music                        SMTP_Syntax.txt
eicar_com.zip             old_school                   software.txt
jacques@UBU64vm>
jacques@UBU64vm> ls -a
.                         .gconf                       README.txt
..                        get_zone1_memberships.sql    README.txt~
AMS_0_1.gp                get_zone1_memberships.sql~   Research
AMS_0_1.gp~               .gimp-2.8                    set_proxy.sh
AMS_10_1000.gp            gnuplot                      set_proxy.sh~
AMS_10_1000.gp~           .gnuplot_history             shellshock
apt_proxy_settings        .gnuplot-wxt                 shellshock.txt
.bash_history             .gstreamer-0.10              SMTP_Syntax.txt
.bash_logout              .gvfs                        software.txt
.bashrc                   Hello_survey.html            SOM_BMU_Animation_gnuplot.7z
.bashrc~                  hist.1                       .ssh
binary_file_as_html.html  hist.2                       ssl-ccs-injection.nse
.cache                    .hplip                       stellarium
```

Basic Commands

```
jacques@UBU64vm> ls -lt
total 124896
drwxrwxr-x 6 jacques jacques     20480 Jan 13 08:10 loki_logs
-rwxrwx--- 1 jacques jacques       911 Jan 12 12:54 commands.info
-rw-rw-r-- 1 jacques jacques       349 Jan 12 12:26 test_perl.pl~
-rwxrwxr-x 1 jacques jacques        70 Jan 12 12:11 test_script.sh~
-rw-r--r-- 1 jacques jacques      9650 Jan 12 08:48 q
drwxrwxr-x 5 jacques jacques      4096 Jan 12 08:42 Trent_teaching
-rw-rw-r-- 1 jacques jacques       238 Jan 12 08:32 hist.2
-rw-rw-r-- 1 jacques jacques      2235 Jan 12 08:31 hist.1
drwxr-xr-x 7 jacques jacques      4096 Jan  7 12:12 public_html
drwxr-xr-x 3 jacques jacques      4096 Jan  1 15:22 Desktop
drwxrwxr-x 4 jacques jacques      4096 Dec 16 14:46 dosbox
drwxrwxr-x 4 jacques jacques     12288 Dec  7 15:30 random
drwxrwxr-x 2 jacques jacques     69632 Dec  7 11:22 perl
drwxr-xr-x 3 jacques jacques      4096 Nov 25 20:22 vmware_tools
-rw-r--r-- 1 jacques jacques     53012 Oct 27 20:52 software.txt
---------- 1 jacques jacques     36223 Sep 20 16:25 Jamies_solution_Archive.
```

Basic Commands

Pipes, Filters and squeezing out the results you want:

```
jacques@UBU64vm> alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
alias t10='ls -lt | head -n 10`

jacques@UBU64vm> t10
total 124896
drwxrwxr-x 6 jacques jacques     20480 Jan 13 08:10 loki_logs
-rwxrwx--- 1 jacques jacques       911 Jan 12 12:54 commands.info
-rw-rw-r-- 1 jacques jacques       349 Jan 12 12:26 test_perl.pl~
-rwxrwxr-x 1 jacques jacques        70 Jan 12 12:11 test_script.sh~
-rw-r--r-- 1 jacques jacques      9650 Jan 12 08:48 q
drwxrwxr-x 5 jacques jacques      4096 Jan 12 08:42 Trent_teaching
-rw-rw-r-- 1 jacques jacques       238 Jan 12 08:32 hist.2
-rw-rw-r-- 1 jacques jacques      2235 Jan 12 08:31 hist.1
drwxr-xr-x 7 jacques jacques      4096 Jan  7 12:12 public_html
```

Basic Commands

Redirecting of input or output:

command > destination_filename

command < input_data_stream

```
jacques@UBU64vm> ls -lt > myfiles.txt
jacques@UBU64vm> wc < myfiles.txt
  62   551  4014  ←――――――――――――――――――――――――――――――――――  (new-lines, words, bytes)
jacques@UBU64vm>
jacques@UBU64vm> ls -lS | head -n 5
total 125536
-rw-r--r-- 1 jacques jacques 116419243 Feb  3  2014 linux_3.13.0.orig.tar.gz
-rw-r--r-- 1 jacques jacques   8059281 Aug 19  2014 linux_3.13.0-35.62.diff.gz
-rwxr--r-- 1 jacques jacques   1844888 Sep  2  2014 SOM_BMU_Animation_gnuplot.7z
-rw-rw-r-- 1 jacques jacques    877094 Jul 17  2014 binary_file_as_html.html
jacques@UBU64vm>
```

More Basic Commands

Other commands:

grep: search within files for specific patterns.

awk: filter/extract portions of a file

date: returns the current system date

wc: word count (counts lines, words and bytes)

sort: sorts a file (based on command line parameters)

touch: Creates an empty file in the target directory

rsync: a fast, versatile, remote (and local) file-copying tool

tar – Creates a tar archive of the source files.

grep

grep allows you to look into a stream of data (or a file), and extract “lines” which meet a search criteria.

grep –e “pattern” filename

e.g.

```
[jacques@loki ~]$ grep -e jacques /etc/passwd
jacquesabeland:x:1001:1001::/home/jacquesabeland:/bin/bash
jacques:x:1002:1003::/home/jacques:/bin/bash
```

awk

awk is VERY powerful. We could probably spend a whole lecture on each of *awk* and *grep*.

For our purposes, we can use *awk* to extract specific columns of data from a stream.

Columns are defined using a “field separator” definition. If we don’t specify one, BLANK is used.

Let’s take the output from our grep and only show the first and last column (if you don’t want to count, $NF is a variable which holed the numeric value of the last column)

e.g.

```
$ grep -e jacques /etc/passwd | awk -F: '{print $1,$NF}'
jacquesabeland /bin/bash
jacques /bin/bash
```

More Basic Commands

We'll leave exploring date, wc and a few others for lab0.

You should really sign on and play with this, it will help you not panic when you hit the first assignment! Not that it is overly hard. It is however alien to most of you.

Basic Commands - Variables

Capturing command output to a shell variable.

```
jacques@UBU64vm> VARIABLE=`date -I`
jacques@UBU64vm> echo $VARIABLE
2016-01-13
jacques@UBU64vm> tar -zcvf my_backup_${VARIABLE}.tar.gz *.txt
eicar.txt
myfiles.txt
new_filename.txt
new_names.txt
README.txt
shellshock.txt
SMTP_Syntax.txt
software.txt
student_names.txt
jacques@UBU64vm> ls -lt *.gz
-rw-rw-r-- 1 jacques jacques 232933 Jan 13 13:21 my_backup_2016-01-13.tar.gz
jacques@UBU64vm>
```

Shell scripts and executable mode

A shell script is just a collection of commands you would normally enter at the keyboard. Simply add them to a text file.

```
jacques@UBU64vm> cat backup_text.sh
#!/bin/bash
VARIABLE =`date -I`
echo $VARIABLE
tar -zcvf my_backup_${VARIABLE}.tar.gz *.txt
echo .....
echo
ls -lt my_backup_${VARIABLE}.tar.gz
echo Done !

jacques@UBU64vm:~$ ls -lt backup_text.sh
-rw-rw-r-- 1 jacques jacques 124 Oct 27 14:03 backup_text.sh
```

Shell scripts and executable mode

```
jacques@UBU64vm:~$ bash backup_text.sh
2016-10-27
eicar.txt
myfiles.txt
new_names.txt
README.txt
shellshock.txt
SMTP_Syntax.txt
software.txt
student_names.txt
.....

-rw-rw-r-- 1 jacques jacques 237774 Oct 27 14:05 my_backup_2016-10-27.tar.gz
Done !
jacques@UBU64vm:~$
```

```
VARIABLE=`date -I`
echo ${VARIABLE}
tar -zcvf my_backup_${VARIABLE}.tar.gz *.txt
echo .....
echo
ls -lt my_backup_${VARIABLE}.tar.gz
echo Done !
```

Shell scripts and executable mode

Had to call up a copy of *bash* and use the filename as the name of the script file for *bash* to execute.

Change the file mode to allow for execution:

```
jacques@UBU64vm> ls -lt backup_text.sh
-rw-rw-r-- 1 jacques jacques 124 Jan 13 13:27 backup_text.sh

jacques@UBU64vm> chmod u+x backup_text.sh   make executable

jacques@UBU64vm> ls -lt backup_text.sh
-rwxrw-r-- 1 jacques jacques 124 Jan 13 13:27 backup_text.sh

jacques@UBU64vm> ./backup_text.sh   execute the file
eicar.txt
...
student_names.txt
-rw-rw-r-- 1 jacques jacques 232933 Jan 13 13:32 my_backup_2016-01-13.tar.gz
Done !
jacques@UBU64vm>
```

Shell scripts and executable mode

Perl is also a scripting language....

```
jacques@UBU64vm> cat ./loki_logs/code_and_data/test_perl.pl

#!/usr/bin/perl   specify the language of execution
use strict;
while ( 1==1 )
{
  my $now = time();
  print "$now -> ";

  my($seconds, $minutes, $hours, $day_of_month, $month, $year, $wday, $yday,
   $isdst) = localtime($now);

  my $ts = sprintf("%4d-%02d-%02d %02d:%02d:%02d
   ",$year+1900,$month+1,$day_of_month,$hours,$minutes,$seconds);
  print      "$ts \n";

 sleep(3);

} # loop forever
```

Shell scripts and executable mode

```
jacques@UBU64vm> ./test_perl.pl
bash: ./test_perl.pl: Permission denied

jacques@UBU64vm> chmod u+x test_perl.pl

jacques@UBU64vm> ./test_perl.pl
./test_perl.pl: line 1: use: command not found
./test_perl.pl: line 5: syntax error near unexpected token `('
./test_perl.pl: line 5: `  my $now = time();'
jacques@UBU64vm>
```

The first line is ANY script should start with a pointer to which script engine should be used to interpret the code.

This is known as the she-bang line: #!

Shell scripts and executable mode

For a perl script... Or bash, how do you know?

```
jacques@UBU64vm> which perl
/usr/bin/perl
```

use 'which' to find the path/she-bang line

```
jacques@UBU64vm> which bash
/bin/bash
```

The first line in the perl script should then be:

#!/usr/bin/perl

When you do this:

```
jacques@UBU64vm> ./test_perl.pl
1452710561 -> 2016-01-13 13:42:41
1452710564 -> 2016-01-13 13:42:44
1452710567 -> 2016-01-13 13:42:47
```

(what other way could you get this script to run without the she-bang line?)

Bash scripts should then start with #!/usr/bin/bash.

Shell script – command line parameters

Here’s a quick script:

```
#!/bin/bash

if [  !  -z $1 ];
 then   echo "First parameter is $1";
fi

if [ ! -z $2 ];
 then   echo -n "Second parameter is $2  ";
fi

if [ ! -z $3 ];
 then   echo "and the third parameter is $3";
 else   echo " ";
Fi

verb="tar -zcvf my_backup_$1_$2.tar.gz $3"
echo "The command that would be executed is: $verb"

echo "Done..."
```

```
if [ -f “$FILENAME” ] ; then
   The file exists!
fi
```
check if a file exists

```
if [ -d “$DIRNAME” ] ; then
  directory exists
fi
```
check if the directory exists/passed directory name actually exists

see: https://linuxize.com/post/bash-check-if-file-exists/

In bash, the command line parameters can be accessed as $1, $2, $3...

Shell Scripts - Loops

```
jacques@newton$ cat do_all.sh

#!/bin/bash
for f in nr_0* ; do
    TS=`date`
    echo Starting ${f}  at ${TS} >> do_all.journal
    perl /research/5_get_bulk_SED.pl ${f}
    perl /research/6_load_SED_table.pl
    COUNT=`ls -1 ./SED/Processed/*.SED | wc -l`
    echo Found ${COUNT} SEDs >> do_all.journal
    7z a ${f}_SED.7z ./SED
    rm *.log
    rm ./SED/Processed/*
done

NOTE: Notice the lack of documentation! Bad style!
```

Building
a
shell
script

Building a shell script

A shell script is just a flat text file that contains lines of code you could have typed in at the command line.

You can crate these lines in a file using the “nano” editor.

How do you do that?

```
[jacques@loki 3380]$ nano myscript.sh
```

An example – Step by Step

In this course, you'll be required to "compile" your C programs.

At the command line, the syntax for compiling is:

```
[jacques@loki 3380]$ gcc -o binary  source_code.c
```

where the –o option identifies the name of the compiled program (**binary** image) your source_code.c will create.

- If you get the order of the fields wrong, you might overwrite your code.
- If you make TONS of changes between compiles, you can't go back to a previous version. Linux doesn't have "versioning" on files like VMS does.

An example – Step by Step

Let’s make a script that:

- Uses a source code filename from the command line
- Makes a backup copy
- Names the binary with the same name as the source (without the .c)

The following slides show us snippets of the whole bash script file.

INCLUDING PROPER DOCUMENTATION !

An example – Step by Step

```
#!/usr/bin/bash
#
#  This routine accepts from the command line a filename
#  it assumes that the file is the prefix for a .c source
#  code file.
#
#  It then makes a backup copy and then compiles the code into a binary
#
#  Did the user put a filename on the command line?
#
if [ -z $1 ]; then
   echo  Usage:
   echo      compile filename
   exit 1
fi
#
```

error condition "1" because user did not enter the argument
this 1 is used to set the exit status

An example – Step by Step

```
#
#  take the first parameter on the command line as the name of the file we want to
#  compile. If the user accidentally uses the source code filename, strip off
#  everything after the first period.
#
FILENAME=`echo ${1} |   awk   -F.   '{print $1}'`
SOURCE="${FILENAME}.c"
#
#
#  Does the source code file exist?
#
if [ ! -f ${SOURCE} ] ; then
  echo the filename ${SOURCE} does not exist?
  exit 2
fi
#
#
```

awk is going to use "." as a field seperator and create 2 columns

display the name of the file

if the source code file does not exists then exit status = 2

An example – Step by Step

```
#
#  Now we make a backup copy of the source. We tag to the end of the filename
#  a timestamp (# of seconds since the beginning of the year). This will give
#  us multiple fallback points. We'll have to clean those up later.
#
TIMESTAMP=`date +%s`
#
BACKUP_NAME=${SOURCE}_${TIMESTAMP}
#
echo Backing up ${SOURCE} to ${BACKUP_NAME}
#
cp    ${SOURCE}    ${BACKUP_NAME}
```

create a backup of the file name_timestamp

An example – Step by Step

```
# Now we compile the code
# Notice that if there are extra libraries required this
# routine will need to be modified to include them.
#
# The routine captures any compile time errors into a file
# for review
#
LOGFILE=${FILENAME}_compile.log
#
gcc -o ${FILENAME} ${SOURCE} > ${LOGFILE} 2>&1
#
# Find out how many lines were output to the log file
LOGSIZE=`wc -l ${LOGFILE} | awk '{print $1}'`
#
if [ ${LOGSIZE} -gt 0 ]; then
   echo There were errors found during the compile. Press q to exit
   less ${LOGFILE}
  else
   echo -e "\tyour code compiled cleanly"
fi
#
```

gcc - name of the compiler

There are 3 file descriptors
STDERR is file descriptor number 2
file descriptor number 1 will redirect everything
into the file mentioned before

if there are errors then everythign is redirected to the LOGFILE

output file name - this is a binary file

file being compiled name

we capture everything into the file so
we don't miss any errors as error show while script
continues running (if its not showstopping)

An example – Step by Step

```
#
#
#  Now we suggest to the user that they cleanup their backup copies
#  should they have more than 4 backups.
#
COUNT=` ls -1 ${SOURCE}_* | wc -l `
#
echo -e "\tI found ${COUNT} backup files"
#
if [ ${COUNT} -gt 4 ]; then
   TO_BE_REMOVED=$(( ${COUNT} - 4))
   echo
   echo -e "\tTime to clean up. ${TO_BE_REMOVED} file to be purged"
   echo -e "\t to delete them copy-paste this command"
   RMLIST=`ls -1 ${SOURCE}_* | tail -n ${TO_BE_REMOVED}`
   echo
   echo  rm ${RMLIST}
   echo
fi

echo  All Done
```

_* : anything after underscore
so all timestamps will be shown

if we have 8 files then we are doing 8-4 and strong the number
in the variable to_be_removed

An example – Step by Step

Let’s test out code:

Notice a lot of different testing just like you should do for your assignments!

there should be if blocks to test different cases such as directory name does not exists

```
[jacques@loki 3380]$ ./compile.sh snorlax.c
```

the filename snorlax.c does not exist?

An example – Step by Step

```
[jacques@loki 3380]$ ./compile.sh fred.c
Backing up fred.c to fred.c_1673370328
       your code compiled cleanly
       I found 6 backup files

       Time to clean up. 2 files to be purged
        to delete them copy-paste this command

rm    fred.c_1673370059    fred.c_1673370328

All Done
```

```
[jacques@loki 3380]$ ./compile.sh fred
Backing up fred.c to fred.c_1673370331
       your code compiled cleanly
       I found 7 backup files

       Time to clean up. 3 files to be purged
        to delete them copy-paste this command

rm fred.c_1673370059 fred.c_1673370328 fred.c_1673370331

All Done
[jacques@loki 3380]$
```

An example – Step by Step

With errors in the code

```
[jacques@loki 3380]$ ./compile.sh fred
Backing up fred.c to fred.c_1673370593
There were errors found during the compile. Press q to exit
        I found 8 backup files

        Time to clean up. 4 file to be purged
         to delete them copy-paste this command

rm fred.c_1673370059 fred.c_1673370328 fred.c_1673370331 fred.c_1673370593

All Done
[jacques@loki 3380]$
```

Output from the less command

```
fred.c: In function 'main':
fred.c:7:3: warning: incompatible implicit declaration of built-in function 'printf' [enabled by default]
   printf("Hello World !!!\n);
   ^
fred.c:7:10: warning: missing terminating " character [enabled by default]
   printf("Hello World !!!\n);
          ^
fred.c:7:3: error: missing terminating " character
   printf("Hello World !!!\n);
   ^
fred.c:9:1: error: expected expression before '}' token
 }
 ^
fred.c:9:1: error: expected ';' before '}' token
fred_compile.log (END)
```

An example – Step by Step

This is all well and good if the compile.sh script resides in the directory you are working in.

If not, you have to type in the complete path to the compile.sh every time you want to run it

```
[jacques@loki Assignment1]$ /home/jacques/3380/compile.sh fred.c
```

That’s a LOT of typing. Maybe it is not worth it?

try to stay in the directory of the file so we don't have to type the entire path for it to find the file

An example – Step by Step

Let’s define a *function* in our .bashrc file which includes the path.

This way we create our own VERB for bash an we can run it from anywhere.

```
cc() {/home/jacques/3380/compile.sh  *${1}*; };
```

An example – Step by Step

The new .bashrc function in action:

```
jacques@loki 3380]$ cc hello_world
Backing up 1_hello_world.c to 1_hello_world.c_1673371633
        your code compiled cleanly
        I found 1 backup files
All Done



[jacques@loki 3380]$ cd lab4
[jacques@loki lab4]$ ls *.c
lab4_fork_and_copy.c
```

every time i write 'cc' it runs the path i mentioned above

```
[jacques@loki lab4]$ cc lab4_fork_and_copy.c
Backing up lab4_fork_and_copy.c to lab4_fork_and_copy.c_1673371667
        your code compiled cleanly
        I found 1 backup files
All Done
```

Combining commands and verbs as filters for a data stream

Everything is a file !

Everything in *NIX is treated as a stream of bytes.

So you can take the stream and pass it through multiple "filters" (processes) to massage the data to get what you are looking for.

this file contains the last 10 accounts that were created. The awk command separates everything by the field separator and creates like a table. So the first column is the username and the last column is the /bin/bash. So when we use that to print we ca only print relevant columns

Streams of data - mining

A simple example:

```
jacques@UBU64vm> grep -e jacques /etc/passwd
jacques:x:1000:1000:jacques,,,:/home/jacques:/bin/bash
jacques@UBU64vm> grep -e jacques /etc/passwd | wc -l
1
jacques@UBU64vm> grep -e jacques /etc/passwd | awk -F : '{print $7}'
/bin/bash
jacques@UBU64vm>
jacques@UBU64vm> tail -n 10 /etc/passwd | awk -F : '{print $1, $7}'
colord /bin/false
hplip /bin/false
pulse /bin/false
jacques /bin/bash
sshd /usr/sbin/nologin
postfix /bin/false
statd /bin/false
mysql /bin/false
jamie /bin/bash
cert
```

field separator is colon
print 7th column

tail shows last 10 lines in passwd file

show column 1 & 7

Systems Administration - mining

```
jacques@UBU64vm> du -sch access*
1.6M     access_log
1.6M     access_log~
4.6M     access_log-20141218 …
7.5M     access_log-20151208
5.7M     access_log-20151210
4.1M     access_log-20151217
243M     total
jacques@UBU64vm>
jacques@UBU64vm> wc -l access*
     5679 access_log
    17394 access_log-20141218
    16862 access_log-20141222
    18121 access_log-20141230…
    27324 access_log-20151208
    17364 access_log-20151210
    11293 access_log-20151217
   819447 total
jacques@UBU64vm>
```

Systems Administration - mining

```
jacques@UBU64vm> grep -h -e beland access_log | head -n 3
24.235.228.125 - - [18/Dec/2015:04:24:05 -0500] "GET
   /~jacquesabeland/add_card.php HTTP/1.1" 200 1105 "-" "Mozilla/5.0 (Windows
   NT 10.0; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0"
24.235.228.125 - - [23/Dec/2015:10:15:39 -0500] "GET
   /~jacquesabeland/duck.html HTTP/1.1" 200 436 "-" "Mozilla/5.0 (Windows NT
   10.0; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0"
24.235.228.125 - - [23/Dec/2015:10:15:40 -0500] "GET /~jacquesabeland/Duck.jpg
   HTTP/1.1" 200 48245 "http://loki.trentu.ca/~jacquesabeland/duck.html"
   "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0"
jacques@UBU64vm>
```

checking one log file

```
jacques@UBU64vm> grep -h -e beland access_log | wc -l
355
```

checking all log files

```
jacques@UBU64vm> grep -h -e beland access_log* | wc -l
3211
jacques@UBU64vm>
```

Systems Administration - mining

```
jacques@UBU64vm> grep -h -e beland access* | awk -F \ '{print $1}'|sort -u
104.158.13.39
157.55.39.210
172.25.130.63  🡐 Private non-routable network (a.k.a. Trent)
172.25.131.213
…
172.25.65.53
172.25.66.197
172.25.66.86
172.25.67.139
172.25.67.157
209.42.109.235
212.71.238.108
24.235.128.223 🡐 Known Cogeco subnet (Home network)
24.235.149.239
24.235.196.174
24.235.213.52
24.235.228.125
68.235.177.16
jacques@UBU64vm>
```

sort all IP addresses and give only unique values

There's a blank here!!!

Systems Administration - mining

Let’s look at ALL of the access logs we have:

The slash at the end allows us to continue more long line of command in next line

```
jacques@UBU64vm> grep -h -e beland access* |awk -F \ '{print $1}’|sort -u \
> | while read i; do wget -O $i.txt  http://ipinfo.io/$i 2>NULL ; done
jacques@UBU64vm>
```

The single > above is PS2 (the second prompt string). It is triggered because I finished the first line with a \.

Not to be confused with an output redirection!

read command uses STDIN

Systems Administration - mining

```
Results:
jacques@UBU64vm> ls *.txt
104.158.13.39.txt   172.25.140.228.txt  172.25.159.248.txt  172.25.67.139.txt   24.235.213.52.txt
157.55.39.210.txt   172.25.140.75.txt   172.25.64.135.txt   172.25.67.157.txt   24.235.228.125.txt
172.25.130.63.txt   172.25.144.205.txt  172.25.64.136.txt   209.42.109.235.txt  68.235.177.16.txt
172.25.131.213.txt  172.25.144.83.txt   172.25.65.24.txt    212.71.238.108.txt
172.25.133.130.txt  172.25.148.203.txt  172.25.65.53.txt    24.235.128.223.txt
172.25.137.102.txt  172.25.151.40.txt   172.25.66.197.txt   24.235.149.239.txt
172.25.137.145.txt  172.25.153.4.txt    172.25.66.86.txt    24.235.196.174.txt
jacques@UBU64vm> rm 24.235*
jacques@UBU64vm> grep -e "country\|city" *.txt
104.158.13.39.txt:  "city": "Peterborough",
104.158.13.39.txt:  "country": "CA",
157.55.39.210.txt:  "city": "Redmond",
157.55.39.210.txt:  "country": "US",
209.42.109.235.txt:  "city": "Peterborough",
209.42.109.235.txt:  "country": "CA",
212.71.238.108.txt:  "city": "",
212.71.238.108.txt:  "country": "GB",
68.235.177.16.txt:  "city": "Greater Sudbury",
68.235.177.16.txt:  "country": "CA",
jacques@UBU64vm>
```

rm is remove

line that has a country or a city in it

That is the OR symbol in the grep command and then put slash so command line does not think it is a pipe symbol

Hmmm. I've never been to Washington State and I haven't been in Great Britain since 1985!!!

Systems Administration - mining

What about the warn log files?

Build a regular expression of what a pattern is for an IP address
this is like creating of what IP addresses look like.

```
jacques> grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' warn
222.255.46.44
222.255.46.44
222.255.46.44
222.255.46.44
…
222.255.46.44
222.255.46.44
91.201.236.114
91.201.236.114

…
jacques@UBU64vm>
```

so looking for an IP address inside a data file where it is not exactly the first column.

[] - define the characters allowed inside
[0-9]\ numbers allowed
{1,3\} i want 1 digit to 3 digits long
\. followed by period

grep -e
grep ' ' -> this is the pattern we want
so we are not using e

grep -n shakespeare -> will tell line number followed by line

warn is the file name

Systems Administration - mining

```
jacques@UBU64vm> wget -O t.t http://ipinfo.io/222.255.46.44
--2016-01-13 10:30:17--  http://ipinfo.io/222.255.46.44
Resolving ipinfo.io (ipinfo.io)... 54.209.230.199, 54.164.24.149, 52.6.165.90
Connecting to ipinfo.io (ipinfo.io)|54.209.230.199|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 221 [application/json]
Saving to: ‘t.t’

100%[======================================================================>] 221         --.-K/s   in 0s

2016-01-13 10:30:17 (45.2 MB/s) - ‘t.t’ saved [221/221]

jacques@UBU64vm> cat t.t
{
  "ip": "222.255.46.44",
  "hostname": "dynamic.vdc.vn",
  "city": "Hanoi",
  "region": "Thanh Pho Ha Noi",
  "country": "VN",
  "loc": "21.0333,105.8500",
  "org": "AS7643 Vietnam Posts and Telecommunications (VNPT)"
}
jacques@UBU64vm>
```

Systems Administration - mining

```
jacques@UBU64vm> wget -O t.t http://ipinfo.io/91.201.236.114
--2016-01-13 10:31:11--  http://ipinfo.io/91.201.236.114
Resolving ipinfo.io (ipinfo.io)... 54.164.24.149, 52.6.165.90, 54.209.230.199
Connecting to ipinfo.io (ipinfo.io)|54.164.24.149|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 175 [application/json]
Saving to: ‘t.t’

100%[===========================================================>] 175         --.-K/s   in 0s

2016-01-13 10:31:11 (10.1 MB/s) - ‘t.t’ saved [175/175]

jacques@UBU64vm>  cat t.t
{
  "ip": "91.201.236.114",
  "hostname": "No Hostname",
  "city": "",
  "region": "",
  "country": "UA",           ← B.T.W.  UA = Ukraine
  "loc": "50.4500,30.5233",
  "org": "AS44446 Qwalarty Corporation"
}
jacques@UBU64vm>
```

Systems Administration - mining

Let’s do ALL of the WARN logs

all warn files

```
jacques> PS2="--> "
Jacques>grep -h -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' warn* \
→   | sort -u | while read i; do wget -O $i.txt  http://ipinfo.io/$i 2>NULL ; done

jacques@UBU64vm> grep -h -e country *.txt | sort -u
  "country": "AE",
  "country": "AT",
  "country": "BR",
… "country": "EC",
  "country": "ES",
… "country": "HK",
  "country": "HU",
  "country": "ID",
  "country": "RU",
… "country": "US",
  "country": "VE",
  "country": "VN",
  "country": "ZA",
jacques@UBU64vm>  # Traffic from 50 different countries !!!
```

Systems Administration - mining

```
jacques@UBU64vm> grep -h -e jacquesabeland access_log* | grep -e 404 |
   grep -o "GET /~jacquesabeland/[a-z]*" | sort -u | head -n 100
GET /~jacquesabeland/
GET /~jacquesabeland/a
GET /~jacquesabeland/abcdef
GET /~jacquesabeland/about
GET /~jacquesabeland/aboutus
GET /~jacquesabeland/access
GET /~jacquesabeland/accessibility
GET /~jacquesabeland/account
GET /~jacquesabeland/action
GET /~jacquesabeland/activ

…
GET /~jacquesabeland/whois
GET /~jacquesabeland/wiki
GET /~jacquesabeland/win
GET /~jacquesabeland/window
GET /~jacquesabeland/windows
GET /~jacquesabeland/wireless
GET /~jacquesabeland/wlan
GET /~jacquesabeland/wordpress
GET /~jacquesabeland/world
GET /~jacquesabeland/wp
GET /~jacquesabeland/write
GET /~jacquesabeland/ws
…
```

GET command comes when we were trying to access a web page so that is one way to find who was trying to access a particular web page

Systems Administration - mining

```
jacques@UBU64vm> grep -h -e jacquesabeland access_log* | grep -e 404 |
   grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}'
   | sort | uniq -c
      1 172.25.130.63
      1 172.25.137.145
      1 172.25.140.228
      2 172.25.140.75
      1 172.25.144.205
      2 172.25.148.203
      1 172.25.151.40
      1 172.25.65.24
      1 172.25.67.139
      5 209.42.109.235
   2525 212.71.238.108
     43 24.235.228.125
      1 68.235.177.16
jacques@UBU64vm>
```

```
jacques@UBU64vm> cat 212.71.238.108.txt
{
  "ip": "212.71.238.108",
  "hostname": "li670-108.members.linode.com",
  "city": "",
  "region": "",
  "country": "GB",
  "loc": "51.5000,-0.1300",
  "org": "AS15830 TELECITYGROUP INTERNATIONAL LIMITED"
}jacques@UBU64vm>
```

Demo…

- Basic commands
  - mkdir & cd
  - ls with qualifiers: -1, -lt -lS
  - rm
  - who – whoami
  - env
  - **grep**: search within files for specific patterns.
  - xargs: uses the data from a previous command as argument for the next.
  - **awk**: filter/extract portions of a file/stream
  - **date**: returns the current system date
  - **wc**: word count (counts lines, words and bytes)
  - **sort**: sorts a file/stream (based on command line parameters)
  - **touch**: Creates an empty file in the target directory
  - **rsync**: a fast, versatile, remote (and local) file-copying tool
  - **tar** – Creates a tar archive of the source files.
  - gcc –o binary_name source_code.c
  - bash

The Shell Environment

After you log in:

```
jacques@UBU64vm:~$ whoami
Jacques
```

$LOGNAME
$USER

```
jacques@UBU64vm:~$ pwd
/home/jacques
```

print working directory

The Shell Environment

```
jacques@UBU64vm> who -a
           system boot  2016-01-13 09:01
           run-level 2  2016-01-13 09:01
LOGIN      tty4         2016-01-13 09:01                1123 id=4
LOGIN      tty5         2016-01-13 09:01                1128 id=5
LOGIN      tty2         2016-01-13 09:01                1135 id=2
LOGIN      tty3         2016-01-13 09:01                1136 id=3
LOGIN      tty6         2016-01-13 09:01                1140 id=6
LOGIN      tty1         2016-01-13 09:02                1844 id=1
jacques  ? :0           2016-01-13 09:02   ?            2455 (:0)
jacques  + pts/9        2016-01-13 09:02   .            2931 (:0)
jacques  + pts/0        2016-01-13 09:56 00:05          2931 (:0)
cert     + pts/23       2016-01-13 10:01   .            3199 (192.168.56.1)
```

Basic Commands - rsync

```
D:\Trent_Teaching> ssh -l jacquesabeland loki.trentu.ca
Password:
Last login: Wed Jan 13 09:25:39 2016 from d24-235-228-125.home1.cgocable.net
jacquesabeland@loki:~>
jacquesabeland@loki:~> mkdir target
jacquesabeland@loki:~> cd target
jacquesabeland@loki:~/target> ls
jacquesabeland@loki:~/target>
```

Meanwhile back on UBU64vm:

```
jacques@UBU64vm> cd source
jacques@UBU64vm> pwd
/home/jacques/source
jacques@UBU64vm> ls
eicar.txt     new_names.txt  shellshock.txt   software.txt
myfiles.txt   README.txt     SMTP_Syntax.txt  student_names.txt
jacques@UBU64vm>
```

Basic Commands - rsync

rsync: a VERY useful remote backup tool

```
jacques@UBU64vm> rsync -rav * jacquesabeland@loki.trentu.ca:target/
Password:
sending incremental file list
README.txt
SMTP_Syntax.txt
eicar.txt
myfiles.txt
new_names.txt
shellshock.txt
software.txt
student_names.txt

sent 568,082 bytes  received 171 bytes  87,423.54 bytes/sec
total size is 567,414  speedup is 1.00
jacques@UBU64vm>
```

rsync is used for sunchronizing files and directories between different locations.

-rav
r tells copy recuresively
a is shortcut for several options that preserve permissions, ownerships, timestamps and recuresive copying
v tells verbose output
* all files in current directory
rest is the destination directory

Basic Commands - rsync

On loki:

```
jacquesabeland@loki:~/target> ls
eicar.txt  myfiles.txt  new_names.txt  README.txt  shellshock.txt  SMTP_Syntax.txt
   software.txt  student_names.txt
jacquesabeland@loki:~/target>
```

On UBU64vm:

```
jacques@UBU64vm> touch new_filename.txt
jacques@UBU64vm> ls
eicar.txt    new_filename.txt  README.txt      SMTP_Syntax.txt  student_names.txt
myfiles.txt  new_names.txt     shellshock.txt  software.txt

jacques@UBU64vm> rsync -rav * jacquesabeland@loki.trentu.ca:target/
Password:
sending incremental file list
new_filename.txt

sent 273 bytes  received 34 bytes  68.22 bytes/sec
total size is 567,414  speedup is 1,848.25
jacques@UBU64vm>
```

touch updates the modification time for a file without changing contents of the file