ITE315 Module 2 Part E - Advanced Bash and Bash Scripting

Athens State University

Useful Bash Advanced Built-in Commands

```
alias necho="echo -n"
unalias command
pushd; popd;
ulimit
wait
wait 2000
```

Functions in Bash

```
#!/bin/sh

function usage ()
{
        echo "Usage:"
        echo "        myprog.sh [--test|--help|--
            version]"
}

case $1 in
        --test|-t)
                echo "you used the --test option"
                exit 0
        ;;
        --help|-h)
                usage
        ;;
        --version|-v)
                echo "myprog.sh version 0.0.2"
                exit 0
        ;;
        -*)
```

Traps

- There are a number of special characters understood by the shell
- In particular, we have "control characters": key chords prefaced with the `CTRL` key
- Important examples;
  - `CTRL-d`: Interpreted as indicator of the end of file
  - `CTRL-C`: Break execution of a program
  - `CTRL-Z`: Send the currently running application to the background

Traps

```
#!/bin/sh

function on_hangup ()
{
    echo 'Hangup (SIGHUP) signal recieved'
}

trap on_hangup SIGHUP

while true ; do
    sleep 1
done

exit 0
```

Traps

```
#!/bin/sh
function on_exit ()
{
    echo 'I should remove temp files now'
}
trap on_exit EXIT
while true ; do
    sleep 1
done
exit 0
```

Patching and file differences

```
diff -u <old-file> <new-file>
diff -u --recursive --newfile <old-dir> <new-dir> > <
    patchfile>.diff
cd <old-dir>
patch -p1 -s < <patch-file>diff
```

Working With Text

- What does a software developer actually do while they're at work?
- Snark aside, we spend most of our day doing things the require the manipulation of text
- And a lot of this work can be automated

Remember our discussion about editor history

- Both `vi` and `emacs` trace their history back to the classic line editors like `ed` and `TECO`
- Another tool in that lineage is the Unix stream editor `sed`
  - This tool reads text line by line and applies a editor command script to each line
  - You use it as a filter on output from commands
  - The commands understood by `sed` are very similar to the "`ed`-mode" commands in `vi`

The Stream Editor

```
# Filter out empty lines from a file
sed -e '/^$/d' myfile.txt
# Add the computer "client123" to every line
# in /etc/exports
cat /etc/exports | \
  sed -e 's/$/ mycomputer/' > /etc/exports
# Add the computer "comp2" only to the entries
# beginning with /data in /etc/exports
cat /etc/exports | \
  sed -e '/^\/data\//s/$/ comp2/' > /etc/exports
# Remove first word on each input line
cat myfile.txt | sed -e 's/^ *[^ ]* //'
```

Field-Based Text Manipulation

- Data has structure
- In text files, it's common to store one record of data per line with some sort of field separator
  - This is often whitespace or punctuation (mostly commas)
- Linux (other POSIX-based systems) provide tools that allow you to extract field data from a text line

Consider This Data File

```
[~] : cat dataset.csv
harry,25,16200
gill,46,17500
bill,45,20000
john,43,100000
barry,27,42000
paul,18,26500
```

The cut command

```
[~] : cut -d, -f 1,3 dataset.csv
harry,16200
gill,17500
bill,20000
john,100000
barry,42000
paul,26500
```

The paste

```
[~] : paste dataset.csv dataset.csv
harry,16200 harry,16200
gill,17500  gill,17500
bill,20000  bill,20000
john,100000 john,100000
barry,42000 barry,42000
paul,26500  paul,26500
```

```
[~] : paste -d, dataset.csv dataset.csv
harry,16200,harry,16200
gill,17500,gill,17500
bill,20000,bill,20000
john,100000,john,100000
barry,42000,barry,42000
paul,26500,paul,26500
```

The AWK Scripting Language

- The AWK scripting language is one of earliest special-purpose scripting languages for UNIX-like operating systems
  - Developed in 1977 by Aho, Weinberger, and Kernighan (get it, AWK?)
  - Significantly enhanced by the GNU project
- Tries to be more of a programming language than `sed`

The Structure of an AWK script

```
1 condition {action}
  condition {action}
```

- The condition can be a simple expression (such as boolean expression) or a regular expression (pattern to match)
- You can define a BEGIN and END pattern-action when define what to do at the start and end of the program
- Or can be `pattern1,pattern2` when says work on the section of input that is found between the first instance of first regexp up until you reach the first instance of the second regexp

A Sample AWK script

```
BEGIN {
  print "--- checking sulog"
  failed=0
  }
{
  if ($4 == "-") {
    print "failed su:\t"$6"\tat\t"$2"\t"$3
    failed=failed+1
    }
}
END {
  print "-----------------------------------------"
  printf("\ttotal number of records:\t%d\n", NR)
  printf("\ttotal number of failed su's:\t%d\n",failed
      )
}
```

Mixing bash and AWK

```
1 ls -l /tmp/foobar | awk '{print $1"\t"$9}'
```

- Command-line parameters passed to AWk are treated as scripts
- So it's often used as part of a filter chain in bash scripts as it's more flexible than combining `sed`, `cut`, and `paste`

The Power of Scripting Languages

- People realized that the things that they were trying to do with `sed`, `cut`, and `paste` was the same thing that you could with languages like `C` (and `C++`)
- So we begin to see languages like `awk`, `PERL`, and `Python` appear on the market
- Languages of this type are optimized for the building of scripts
  - But many like `Python` or `Javascript` can be used as general purpose programming languages