Linux Fundamentals by Commands and Examples

version 1.0

Editor: Ahmed Baraka

Document Purpose

This document is edited to be a quick reference of Linux essential commands. It can be used by Linux beginners as a reminder of basic Linux commands usage. It cannot be used to learn Linux from scratch.

The document is oriented based on the required task, the command(s) to do the task, basic syntax of the command, and examples. No explanation will be presented.

Version History

Version	Date	Updates
1.0	25-June-2009	Initial document.

Resources

Resource Name
Linux Essentials article by Thomas Girke.
The Linux Cookbook: Tips and Techniques for Everyday Use by Michael Stutz
Red Hat Linux Getting Started Guide, 2003
Red Hat Essentials RH033 (courseware material)
Linux Reviews website
The Linux Tutorial website
Zytrax Info website
Academic Computing And Communications Center ACCC website
The Linux Information Project website
UNIX & Linux Shell Scripting Tutorial website

Contents

Introduction ____ *7*
Unix variants ____ 7
GNU/Linux distributions ____ 7

Getting Started ____ *8*
Virtual Consoles ____ 8
Changing password ____ 8
Logging-In ____ 8
Date and Time Commands ____ 8
Making Arithmetic Calculations ____ 8
Generating Sequential Numbers ____ 8
Getting Help ____ 9
Handy shortcuts ____ 9

Managing Files and Directories ____ *10*
Files and Directories Commands ____ 10
Determining File Content ____ 11
Viewing Files ____ 11
Hard and Symbolic (Soft) Links ____ 12
Checking Free Space ____ 12
Searching Files By Names ____ 12
Searching Files By Names and Attributes ____ 12
Archiving Files ____ 14
Compression Utilities ____ 14
Text File Processing Tools ____ 14

Users, Groups and Permissions ____ *15*
Change Password ____ 15
Change Your Identity ____ 15
User Information Commands ____ 15
Changing File Ownership ____ 15
Changing Permissions ____ 15
Default File Permission ____ 15
Special Permission ____ 16

bash Shell Basics ____ *17*
File Blobbing ____ 17
History Tricks ____ 17
Command Line Expansion ____ 17
Local Shell Variables ____ 18
Aliases ____ 19
Type ____ 19
Environment Variables ____ 19
Showing Path of Executable ____ 19
Login and Non-Login Shells ____ 20
Startup and Logout Scripts ____ 20
Recording a Shell Session ____ 20

Standard I/O and Pipes ____ *21*
Redirecting Output to a File ____ 21
Redirecting STDOUT to a Program (Piping) ____ 21
Redirecting to Multiple Targets ____ 21
Redirecting STDIN from a File ____ 21
Sending Multiple Lines to STDIN ____ 21

Text Files and String Manipulation ____ *22*
Viewing File Contents ____ 22
Viewing File Excerpts ____ 22
Extracting Text by Column ____ 22
Gathering Text Statistics ____ 22
Sorting Text ____ 23
Eliminating Duplicates ____ 23
Comparing Files ____ 23
Spell Checking with aspell ____ 23
Converting Characters ____ 24
Combining Files ____ 24
Expanding Tabs Into Spaces ____ 24
Regular Expressions ____ 24
Extended Regular Expressions ____ 25
Extracting Text by Keyword ____ 25
Search and Replace ____ 26

Editing Text by Programming Language ____ 26

Using the Text Editor vi ____ ***27***

Modes ____ 27

Search and Replace (Command Mode) ____ 27

Manipulating Text (Command Mode) ____ 27

Undoing Changes (Command Mode) ____ 28

Visual Mode ____ 28

Using Multiple "windows" ____ 28

Configuring vi and vim ____ 28

Managing Processes ____ ***29***

Listing Processes ____ 29

Sending Signals to Processes ____ 31

Changing Process Scheduling Priority ____ 31

Listing Background and Suspended Jobs ____ 31

Resuming Suspended Jobs ____ 31

Compound Commands ____ 32

Scheduling a Process ____ 32

Scheduling a Process Periodically ____ 32

bash Shell Scripting Basics ____ ***33***

Creating Shell Scripts ____ 33

Handling Input ____ 33

Shell Script Debugging ____ 33

Handling Positional Parameters (Arguments) ____ 33

Using Functions ____ 34

Exit Status ____ 34

Conditional Execution ____ 34

Using the if Statement ____ 34

Using the Case Statement ____ 35

Using the For Loop ____ 35

Using the While loop ____ 35

Disrupting Loops ____ 36

File Tests ____ 36

String Tests ____ 37

Shell Option Test _____ **37**
Logical Tests _____ **37**
Comparison _____ **37**

Introduction

Unix variants

- Unix, GNU/Linux, Solaris, IRIX, HP-UX, FreeBSD, OpenBSD, NetBSD, Darwin (Mac), and more...

GNU/Linux distributions

- Ubuntu, Edubuntu, Debian, RedHat, Fedora, Slackware, SuSE, Darwin, and more...
- Family tree of the GNU/Linux distributions

Getting Started

Virtual Consoles

- In Red Hat: available through CTRL+ALT+F[1-6]
- If X is running, it is available as CTRL+ALT+F7

Changing password

- passwd

Logging-In

- From Mac or LINUX

```
ssh -X your_username@hostname
```

- From Windows: Open Putty and select ssh.
- Use WinSCP software for file exchange.

Date and Time Commands

- `date`
  - `u` display date and time in UTC
  - `R` display date and time in RFC822 (used in email messages)
- `chrony` package maintains time by connecting to servers over the Internet.
- `cal` output a calendar for the current month
  - `y` print calendar of current year

```
cal 2010   #output a calendar for the year 2010
```

Making Arithmetic Calculations

- `bc`
- supported operators: + - * / % ^ sqrt()

Generating Sequential Numbers

- seq
  - `w` make all generated numbers of same width
  - s '*b*' make *b* character as the separator between numbers

```
seq 7
seq -5 5
```

```
seq 1 2 10          # from 1 to 10 increment by 2
seq –s ' ' 1 23     # separated by spaces
```

Getting Help

- `man`
- `info`
- `apropos` search for only exact matches
- `whatis` to list a one–line description of a program
- Software packages may store its help files in /usr/share/doc
- Online help: SuperMan Pages, Linux Documentation Project (LDP)
- LinuxQuestions.org

```
man ls
man –k copy          # search for "copy" in the whatis database
apropos copy         # search for "copy" (not "xcopy") in the whatis database
man –f copy          # restrict search for the whole word
ls -- help           # brief help usage
info cp              # information is organized into sections
whatis who
```

Handy shortcuts

up(down)_key scrolls through command history
<something-incomplete> TAB completes path/file_name
Ctrl+a # cursor to beginning of command line
Ctrl+e # cursor to end of command line
Ctrl+d # delete character under cursor
Ctrl+k # cut line from cursor into kill buffer
Ctrl+y # paste content from Ctrl k

Managing Files and Directories

Files and Directories Commands

- `pwd`
- `cd`
- `ls` (see the options in the example below)
- File types that may be listed by ls –l :
  - `-` regular file
  - `d` directory
  - `l` symbolic link
  - `b` block special file
  - `c` character special file
  - `p` named pipe
  - `s` socket
- `cp` `# for copying between hosts, see next section`
- `rm`
- `mv`
- `mkdir`
- `rmdir`
- `touch` create empty files or update file timestamps
- File Name Expansion characters can be used with these commands.

```
cd ..                          # one level up
cd                             # home directory
cd -                           # previous directory

ls –a                         # include hidden files
ls –l                         # long listing
ls –R                         # recurses through subdirectories with contents
ls –d                         # directory names without their contents
ls –lh                        # print sizes in human readable format
ls –ld                        # avoid listing directory contents
ls –i                         # print index number
ls –S                         # sort by file size
ls –t                         # sort by modification time (newest first)
ls –r                         # reverse order while sorting
ls –l --time-style=STYLE      # STYLE: full-iso, long-iso, iso, locale, +FORMAT

cp file1 file2                # timestamp changes for the new file
```

```
cp -p file1 file2            # all of the attributes are preserved
cp file1 dir1
cp file1 file2 dir1
cp ./dir1/* dir2
cp -r dir1 dir2      # -r (same as -R) copy entire directory tree
                     #  links aren't copied, permissions aren't preserved
cp -a dir1 dir2      # copy the entire tree including permissions and links

mv file1 file2               # renames directories or files
mv file1 ./dir1              # moves file/directory as specified in path

rm file1                     # removes file name
rm -r dir1                   # removes directory including its content,
rm -rf dir                   # 'f' argument turns confirmation off
rm -- -myfile                # the fielename containes hyphen

touch {doc,sheet}_{jan,feb,mar}
touch file{0..6}.txt
```

Determining File Content

- `file`

```
file myfile
```

Viewing Files

- `cat`
- `less`
- `less` navigation commands:

space	ahead one full screen
ENTER	ahead one line
b	back one full screen
k	back one line
g	top of the file
G	bottom of the file
/text	search forward for text (Regular Expressions can be used)
n	repeat last search
N	repeat backward last search
q	quit

Hard and Symbolic (Soft) Links

- `ln`
- `ls -l` in case of soft link, it displays the link name and the referenced file

```
ln –s filename
```

Checking Free Space

- `df` space usage by file systems
- `du` disk space by directories and subdirectories

```
df –h                          # -h prints size in readable format
du –h –s ~                     # -s reports single directory summary
```

Searching Files By Names

- `locate [options] name(s)`
- `slocate [options] name(s)`
- Only the files you own are searched
- Some options are shown in the example below.
- `locate.db or slocate.db` databases are used
- `updatedb` or `locate -u` to manually update the database

```
locate "*.png"        # wildcard characters can be used
locate "*.png" -q     # supress errors
locate -n 15 "*.html" # only 15 resultes returned
locate -i "*.HtmL"    # case-insensitive search
```

Searching Files By Names and Attributes

- `find <dirs> [conditions] [-exec cmd {} \;]`

`-atime n`	File was last accessed n days ago
`-ctime n`	File was last changed n days ago
`-user uname`	File is owned by user uname (or user ID)
`-group gname`	File belongs to group gname (or group ID)
`-size n[cwbkMG]`	b 512-byte blocks (default), c in bytes, w two-byte words, k kilobyte
`-iname`	case -insensitive version of –name
`-o`	logical operator between criteria (by default it is AND)
`-not`	negate (logical NOT)
`-perm mode`	permission bits are exactly mode (octal or symbolic).
`-perm -mode`	ALL of the permission bits mode are set for the file.

`-perm +mode`	Any of the permission bits mode are set for the file.
`-regex pattern`	Full path filename (not only filename) matches regular expression pattern.
`-mtime n`	Files was last modified Exactly n*24 hours ago.
`-mtime +n`	Files was last modified >= n*24 hours ago.
`-mtime -n`	Files was last modified <= n*24 hours ago.
`-mmin n`	Files was last modified n minutes ago.
`-daystart`	measure time in the options above from the beginning of the current day instead of 24 hours ago.
`-newer <file>`	Files newer than <file> modification date

```
find . –name "*.html"
find -iname snow.png
find -user peter -group peter
find -user joe -not -group joe
find -user joe -o -user jane
find -not \( -user joe -o -user jane \)

find -perm 755  # matches if mode is exactly 755
find -perm +222 # matches if anyone can write
find -perm -222 # matches if everyone can write
find -perm -002 # matches if other can write

find -size 1024k  # excatly 1 MB
find -size +1024k # over 1 MB
find -size -1024k # less than 1 MB
find ~ –empty     # find empty regular files or directories

find -size +102400k -ok gzip {} \; # OK prompte before acting
find . -regex '.*[124].*ms$'

find ~ –mtime 1            # files modified exactly 24 hours ago
find ~ –mtime 1  –daystart # modified yesterday
find ~ –mtime +356         # one year or longer ago
find ~ –mtime 2 –mtime –4 –daystart # two to four days ago
# files that were modified after May 4 of the current year
touch –t 05040000 /tmp/timestamp
find ~ –newer /tmp/timestamp
```

Archiving Files

- `tar cvf archive_name files …` to create an archive file
  - `c` create a new archivd
  - `v` produces verbose messages
  - `f` archive file name
  - j use bzip2 compression
  - z use gzip compression
- `tar tf archive_name` to inspect files in an archive, if `v` option is used, long file list
- `tar xvf archive_name` to extract an archive file (always to current directory)

```
tar cvf mywork.tar .bas_profile /tmp
tar cvf myHome.tar ~
```

Compression Utilities

- `gzip –v file(s)` v option displays compression percentage, original file replaced
  only regular files are compressed
- `bzip2 –v file` better compression
- `gunzip filename.gz` uncompress the file
- `gzip –d filename.gz` uncompress the file
- `gunzip –c filename.gz` list contents of the compressed file in STDOUT, the file unchanged
- `bunzip2 –v file`

Text File Processing Tools

- Check the section Text File Processing Tools

Users, Groups and Permissions

Change Password

- `passwd`

Change Your Identity

- `su username`
- `su – username` # start a login shell

User Information Commands

- `whoami` # who you are
- `groups, id` # what groups you belong to
- `users, who, w` # who is logged on
- `last` # login/reboot history

Changing File Ownership

- `chown user_name file|directory`
- `chgrp group_name file|directory`

```
chown john myfile
chown –R john dir              # operate on files and directories recursively
```

Changing Permissions

- `chmod mode file`

  where mode is: [u,g or o] [+ or -] [r, w or x] (Symbolic Method)

  where mode is: 4:r 2:w 1:x (Numeric Method)

```
chmod o-rwx file1
chmod u-w,go-x file1
chmod +x file1  # the file is executable to all security levels
chomod 775 file1
```

Default File Permission

- `umask` # if case of 0002, 664 permission for files, 775 for directories

Special Permission

- `chmod Xnnn` # X: 4 for suid, 2 for sgid, 1 for sticky bit
- suid and sgid are effective on **executable files:** the program runs with permissions of the owner, not the executor.
- sgid and sticky `bit` are effective on **directories**:
    - sticky bit: files in the directory can be deleted by the owner or the root, regardless of the directory write permission.
    - Sgid: files created in the directory will inherit its group affiliation from the directory, rather than the user.

```
ls -l /usr/bin/passwd
-r-s--x--x  1 root root
ls -ld /tmp/
drwxrwxrwt  10 root root
chmod 2770 GroupDir
```

bash Shell Basics

File Blobbing

*	matches zero of more characters
?	matches any single character
[a-z]	matches a range of characters
[^a-z]	matches all except the range

```
ls file*
ls ??file
ls file[1-9]
ls file[^6-0]
```

History Tricks

- history
- Use the up and down keys to scroll through previous commands.
- Type Ctrl-r to search for a command in command history.
- Esc+. to recall last argument from previous command.
- !n re-execute command number n

Command Line Expansion

- Command Expansion: `$()` or ``` `` ```
- Brace Expansion: `{ }`
- Arithmetic: `$[]`
- Arithmetic Evaluations: `+ - * / ** %` (Full list in Arithmetic Evaluation section in bash man page)
- `\` backslash makes the next character literal and can be used as last character on line to continue command on next line.
- To pass special characters as a string: `$'string'`
- Special backslash escape sequences:

`\a`	Alert (rings the system bell).
`\b`	Backspace.
`\e`	Escape.
`\f`	Form feed.
`\n`	Newline.
`\r`	Carriage return.
`\t`	Horizontal tab.

`\v` Vertical tab.

`\\` Backslash.

`\NNN` Character whose ASCII code is NNN in octal (base 8).

- Filename Expansion Characters: used with commands handling files and directories:

  `*` zero or more characters

  `?` exactly one character

  `[list]` one character in the list. Examples: [abc],[a-m], a[-b]c

  `[!list]` except the characters in the list. For example: a[!b]c matches aac a-c adc, but not abc.

```
echo "This system's name is $(hostname)"
echo $'Note the space below\n'     # doesn't work if you use double quote
echo current date is `date`
echo file{1,3,5}
echo Area: $[ $X * $Y ]     # equivalent to $[$X*$Y]
echo Your cost: \$8.00
find / -name myfile\*
mv /usr/tmp/song[0–9].cdda ~/music
```

Local Shell Variables

- VARIABLE=value
- `echo $VARIABLE`
- To see list of the local variables that configure the shell, see the **Shell Variables** section of the `bash` man page.
- Common Local Configuration Variables
  - `PS1` the prompt

    `\d` date

    `\h` short hostname

    `\t` time

    `\u` user name

    `\w` current working directory

    `\!` history number of the current command

    `\$` $ for superusers, # for non-privileged user
  - `HISTFILESIZE` how many commands to save in history
  - `COLUMNS` width of the terminal
  - `LINES` height of the terminal

- Common Local Information Variables
  - `HOME` user's home directory
  - `EUID` user's effective UID

```
PS1="\u\w\$"
```

Aliases

- `alias`

```
alias lf="ls -Fca"
alias rm="rm -i"
\rm -r myfile       #to run the command, not the alias
```

Type

- `type` to ask the shell what it is using to fulfill the command

```
type rm
tyupe cate
```

Environment Variables

- `export VARIABLE=value`
- Common Environment Variables:
  - `HOME` user home directory path
  - `LANG` default language (like en_US.UTF-8)
  - `PWD` current working directory
  - `EDITOR` default editor used by programs
  - `LESS` options to pass to the `less` command
- `reset` when the screen become corrupted

```
EDITOR=/usr/bin/vim;   export EDITOR
export EDITOR=/usr/bin/pico
EDITOR=       # once exported, no need to export it again to change its value
```

Showing Path of Executable

- `which`

```
which xboard
```

Login and Non-Login Shells

- **Login shells are:**
  - Any shell created at login (includes X login)
  - `su -`
- **Non-login shells are:**
  - `su`
  - graphical terminals
  - executed scripts
  - any other bash instances

Startup and Logout Scripts

- **Login Sells**

```
/etc/profile
   /etc/profile.d
~/.bash_profile
   ~/.bashrc
          /etc/bashrc
```

- **Non-Login Shells**

```
~/.bashrc
   /etc/bashrc
          /etc/profile.d
```

- **Logout Script**

```
~/.bash_logout
```

Recording a Shell Session

- `script <filename>` record session typescript records in filename
- `exit` end session recording

Standard I/O and Pipes

Redirecting Output to a File

- > Redirect STDOUT to file (overwrite)
- >> Redirect STDOUT to file (append)
- 2> Redirect STDERR to file
- &> Redirect all output to file
- 2>&1: Redirects STDERR to STDOUT
- (): Combines STDOUTs of multiple programs

```
find /etc -name passwd > find.out 2> find.err
find /etc -name passwd &> find.all
( cal 2007 ; cal 2008 )
```

Redirecting STDOUT to a Program (Piping)

- Pipes (the | character) can connect commands

```
ls -l /etc | less
```

Redirecting to Multiple Targets

- `tee` - read from standard input and write to standard output and files
- command1 | tee filename | command2 store STDOUT of command1 in filename, then pipes to command2

```
find /etc -name "r*" | tee foundlist.txt | less
```

Redirecting STDIN from a File

- Redirect standard input with <

```
$ tr 'A-Z' 'a-z' < myfile   # equivalent to cat myfile | tr 'A-Z' 'a-z'
```

Sending Multiple Lines to STDIN

- Redirect multiple lines from keyboard to STDIN with <<WORD
- All text until WORD is sent to STDIN

```
$ mail -s "Please Call" test@mydomain.com <<END
```

Text Files and String Manipulation

Viewing File Contents

- `cat` dump one or more files to STDOUT
- `less` view file or STDIN one page at a time
- `less` navigation commands:

space	ahead one full screen
ENTER	ahead one line
b	back one full screen
k	back one line
g	top of the file
G	bottom of the file
/text	search forward for text
n	repeat last search
N	repeat backward last search
q	quit

Viewing File Excerpts

- `head`
- `tail`

```
head -n 5 .bash_profile
tail -n 5 .bash_profile
tail -n 5 -f mylogfile        # follow subsequent additions to the file
                              # useful for monitoring log files!
```

Extracting Text by Column

- `cut`

`-d`	to specify the column delimiter (default is TAB)
`-f`	to specify the column to print
`-c`	to cut by characters

```
cut -d: -f1 /etc/passwd
grep root /etc/passwd | cut -d: -f7
cut -c2-5 /usr/share/dict/words
```

Gathering Text Statistics

- `wc`
  - `-l` for only line count
  - `-w` for only word count

`-c` for only byte count
`-m` for character count

```
wc story.txt     # words, lines, bytes
39 237 1901 story.txt
```

Sorting Text

- `sort`
  `-r` performs a reverse (descending) sort
  `-n` performs a numeric sort
  `-f` ignores (folds) case of characters in strings
  `-u` (unique) removes duplicate lines in output
  `-t` *c* uses *c* as a field separator
  `-k` *X* sorts by *c*-delimited field X

```
grep bash /etc/passwd | sort
sort –t : -k 3 –n /etc/passwd    # sort by uid
```

Eliminating Duplicates

- `sort -u`: removes duplicate lines from input
- `uniq`: removes duplicate adjacent lines
  -c prefix lines by the number of occurrences
  -d only print duplicate lines
  -u only print unique lines

```
cut –d: -f7 /etc/passwd | sort | uniq
```

Comparing Files

- `diff`

```
diff file1 file2
```

Spell Checking with aspell

- `aspell`

```
aspell check letter.txt
aspell list < letter.txt
aspell list < letter.txt | wc -l
```

Converting Characters

- `tr` converts characters in one set to corresponding characters in another set

```
tr 'a-z' 'A-Z' < lowercase.txt
```

Combining Files

- `paste` combines files horizontally and separate the pasted lines by TAB by default.

```
paste –d: ids.txt names.txt > merged.txt  # separate the data with colon
```

Expanding Tabs Into Spaces

- `expand` convert the tabs in the file to spaces

```
expand tabfile.txt > tabfile.expanded.txt
```

Regular Expressions

- **Wildcard Characters**
  another single character
  - . any single character
  - [abc] any single character in the set
  - [a-c] any single character in the range
  - [^abc] any single character ***not*** in the set
  - [^a-c] any single character ***not*** in the range

- **Modifiers**
  number of the previous character
  - * zero or more of the previous char
  - \+ one or more of the previous char
  - \? zero or one of pervious char
  - \{i\} exactly i of the previous char
  - \{i,\} i or more of the previous char
  - \{i,j\} i to j of the previous char

- **Anchors**
  match the beginning or end of a line or word
  - ^ line begins with
  - $ line ends with
  - \< word begins with
  - \> word ends with

- **Other expressions**
  - [:alnum:] Alpha-numeric characters 0 to 9 OR A to Z or a to z
  - [:alpha:] Alpha character a-z A-Z
  - [:cntrl:] Control characters
  - [:digit:] Digits 0 to 9

[:graph:]	Graphics characters
[:print:]	Printable characters
[:punct:]	Punctuation " ' ? ! ; : # $ % & ( ) * + - / < > = @ [ ] \ ^ _ { } | ~
[:space:]	White spaces (space, tab, NL, FF, VT, CR)
[:blank:]	Space and Tab characters only

```
^S[^ ]* R     # the last name begins with S and first name begins with R.
^[M-Z].*[12]  # the last name begins with a letter from M to Z and where the
              #   phone number ends with a 1 or 2.
'^..$'        # any word of only two characters
'^.\{17\}$'   #  words of exactly seventeen characters wide
[0-9]\{5,10\} # all number combinations between 5 and 10 number long
[a-z]\)$      # The \ is an escape characher
\(.*l         # contains ls and preceeded by an open bracket
```

Extended Regular Expressions

- Except word anchors, basic regular expressions requiring a preceding backslash do not require backslash
- Used by:
    - egrep
    - grep –E
    - awk

Extracting Text by Keyword

- grep [OPTION]... PATTERN [FILE] ...

-i	to search case-insensitively
-n	to print line numbers of matches
-v	to print lines not containing pattern
-AX	to include the X lines after each match
-BX	to include the X lines before each match

- grep uses by default basic Regular Expressions
- egrep uses Extended Regular Expressions

```
grep 'root' file*.doc         # this will list the file name
grep –h 'root' file*.doc      # to avoid listing the file names
grep 'ahmed' /etc/passwd      # highly advisable to use sing quote
date --help | grep year
egrep 'a{2,5}' myfile    # search for counter 2,3,4 or 5 letter a's
egrep '\<bye\>' myfile
```

Search and Replace

- `sed` (stream editor) uses regular expressions in search string (but not in replace)

```
sed 's/cat/dog/' petsfile            # makes the replacement once per line
sed 's/cat/dog/g' petsfile           # multiple changes per line
sed 's/[Cc]at/dog/g' petsfile
sed 's/\<[Cc]at\>/dog/g' petsfile    # search by word (not string)
sed 's/\<[Cc]at\>/& and dog/g' petsfile # whatever found (Cat or cat), it will
                                        # be replaced with cat and dog
sed '10,40s/cat/dog/g' petsfile      # only lines from 10 and 40 searched
sed '/begin/,/end/s/cat/dog/' petfiel # search will start from the line
                                      #  containing "begin" to the line
                                      #  containing "end"
sed –e 's/cat/dog/g' –e 's/cow/goat/g' petsfile # multiple find and replaces
```

Editing Text by awk

- `awk`
- All extended regular expressions work except curly brace counters. To use them, use `--posix` or `--re-interval` options.

```
awk ' { print } ' myfile                 # equivalent to cat command
awk '/bye/ { print } ' myfile            # print lines containing the pattern
awk '/[2-5]+/ { print } ' myfile
awk ' { print $2, $1 } ' myfile  # print fields 2 and 1 in a space separated
                                 #  text file.
awk ' { print $2 "  " $1 } ' myfile  # in a tab separated file
```

Using the Text Editor vi

Modes

Command Mode

- Default mode of vim
- Move by character: Arrow Keys, h, j, k, l
- Move by word: w, b
- Move by sentence:), (
- Move by paragraph: }, {
- Jump to line x: xG
- Jump to end: G

Insert mode

- i begins insert mode at the cursor
- A append to end of line
- I insert at beginning of line
- o insert new a line (below)
- O insert new line (above

Ex Mode

- :w writes (saves) the file to disk
- :wq writes and quits
- :q! quits, even if changes are lost

Search and Replace (Command Mode)

- /, n, N Search
- <>/<>/<> Search/Replace (as in sed command)

```
:1,5s/cat/dog/g  # search in lines 1 to 5 and replace all words in any line
:%s/cat/dog/gi   # the whole file
```

Manipulating Text (Command Mode)

Action followed by Target

Possible actions:

- change (c)
- cut (d)
- yank (y)
- paste (p) without target

Possible target:

- Line as in action
- Letter l
- Word w
- Sentence ahead)
- Sentence behind (
- Paragraph above {
- Paragraph below }

Undoing Changes (Command Mode)

- u undo most recent change.
- U undo all changes to the current line since the cursor landed on the line.
- Ctrl-r redo last "undone" change

Visual Mode

- Allows selection of blocks of text
- v starts character-oriented highlighting
- V starts line-oriented highlighting
- Highlighted text can be deleted, yanked, changed, filtered, search/replaced, etc.

Using Multiple "windows"

- Multiple documents can be viewed in a single vim screen.
- Ctrl-w, s splits the screen horizontally
- Ctrl-w, v splits the screen vertically
- Ctrl-w, Arrow moves between windows
- :q close the current window
- Ex-mode instructions always affect the current window

Configuring vi and vim

- :set or :set all Configuring on the fly
- ~/.vimrc or ~/.exrc Configuring permanently
- :set showmode show when you are in insert mode
- :set ic ignore case when searching
- :set noic turn ignore case off
- :set nu turn on line numbering
- :set nonu turn line numbering off

Managing Processes

Listing Processes

- `top` continuously updated list
- `ps` shows processes from the current terminal by default
  - -a all processes except session leaders and processes not associated with a terminal.
  - `-A` prints all processes. Identical to -e.
  - `-e` prints all processes. Identical to -A.
  - -H show process hierarchy
  - `-u` prints process owner information
  - –l show log-listing format
  - –L show thread information
  - `-a` exclude processes not associated with a terminal
  - `-x` includes processes not attached to terminals
  - `-f` prints process parentage
  - -- sort some sorting options are:

c	cmd	simple name of executable
C	pcpu	cpu utilization
r	rss	resident set size
R	resident	resident pages
s	size	memory size in kilobytes
S	share	amount of shared pages
T	start_time	time process was started
U	uid	user ID number
u	user	user name
v	vsize	total VM size in kB

  - -o CODE prints custom information where CODE taken from the following list:

Code	Header	Description
%cpu	%CPU	cpu utilization of the process in "##.#" format. (alias `pcpu`)
%mem	%MEM	physical memory in percentage. (alias `pmem`)
Bsdstart	START	time the command started.
bsdtime	TIME	accumulated cpu time, user + system. "MMM:SS"
comm.	COMMAND	command name (only the executable name)
Egid	EGID	effective group ID number of the process (alias `gid`)
egroup	EGROUP	effective group ID of the process. (alias `group`)
etime	ELAPSED	elapsed time since the process was started, [[dd-]hh:]mm:ss.
euid	EUID	effective user ID. (alias `uid`)
euser	EUSER	effective user name.
fgid	FGID	filesystem access group ID. (alias `fsgid`)
fname	COMMAND	first 8 bytes of the base name of the proces executable file.
fuid	FUID	filesystem access user ID. (alias `fsuid`)
fuser	FUSER	filesystem access user ID.
label	LABEL	security label (used for SE Linux context data).
lstart	STARTED	time the command started.
lwp	LWP	lwp (light weight process, or thread)

		ID of the lwp being reported. (alias `spid, tid`)
ni	NI	nice value. This ranges from 19 (nicest) to -20 (not nice to others) (alias `nice`)
nlwp	NLWP	number of lwps (threads) in the process. (alias `thcount`)
pgid PGID	process	group ID or, equivalently, the process ID of the process group leader. (alias `pgrp`)
pid	PID process	ID number of the process.
ppid	PPID	parent process ID.
psr	PSR	processor that process is currently assigned to.
rgid	RGID	real group ID.
rss	RSS	resident set size, the non-swapped physical memory that a task has used (in kiloBytes). (alias rssize, rsz).
ruid	RUID	real user ID.
ruser	RUSER	real user ID (textual, if possible)
s	S	minimal state display (one character). See sub-section below (alias state)
sched	SCH	scheduling policy of the process (0,1,2)
sess	SESS	session ID (alias session, sid).
sig	PENDING	pending. (alias pending, sig_pend).
sigcatch	CAUGHT	caught. (alias caught, sig_catch).
sigignore	IGNORED	ignored. (alias ignored, sig_ignore).
sigmask	BLOCKED	blocked. (alias blocked, sig_block).
start_time	START	starting time or date of the process.
stat	STAT	multi-character process state.
suid	SUID	saved user ID. (alias svuid).
suser	SUSER	saved user name (textual, if possible) (alias svuser).
time	TIME	cumulative CPU time
tname	TTY	controlling tty (terminal). (alias tt, tty).
tt	TT	controlling tty (terminal). (alias tname, tty).
tty	TT	controlling tty (terminal). (alias tname, tt).
vsize	VSZ	virtual memory usage of entire process.
vsz	VSZ	see vsize. (alias vsize).

Process statuses:

R Runnable: executing

S Sleeping: waiting for an event to occur to wake up

T Stopped: not executing

D Uninterruptible sleep

Z Zombie: just before a process dies. It no notification acknowledgment received from parent, all resources except PID are released.

When the `stat` keyword is used, additional characters may be displayed:

<	high-priority (not nice to other users)
N	low-priority (nice to other users)
L	has pages locked into memory (for real-time and custom IO)
s	is a session leader
l	is multi-threaded
+	is in the foreground process group

```
ps aux          # commonly used and equivalent to -aux
ps –e           # to see every process on the system
ps –ef          # to see every process on the system
ps –eH          # to print a process tree
```

```
ps –eLf        # to show threads
ps –el         # to display long listing format

# To see every process with a user-defined format:
ps -eo pid,euser,ruser,lstart,stat,pcpu,pmem,rss,vsize --sort -rss
ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm.
ps -eo euser,ruser,suser,fuser,f,comm,label   # To get security info.
```

Sending Signals to Processes

- `kill [-signal] pid(s)` the default signal is `TERM` (15)
- `kill –l` lists the signals (for more info use `man 7 signal`)

```
# following commands send TERM signal (normal exiting)
kill 3453
kill -15 3453
kill –TERM 3453
# following commands send KILL signal (can be used if TERM failed)
kill –KILL 3453
kill -9 3453
```

Changing Process Scheduling Priority

- `nice [-n adj ] command` where `adj` between -20 (highest) and 19 (lowest)
- `renice adj [[-p|-g] PID [[-u] user]` for running processes

```
nice –n 10 myapp
renice -15 –p 201    # only superuser can increase priority
renice 15 –u john    # for all processes owned by john
```

Listing Background and Suspended Jobs

- `jobs`

Resuming Suspended Jobs

- `bg [%job_number]` or Ctl+Z brings the current process into background
- `fg [%job_number]` brings the background process into foreground

You can use the following code example to test the commands:

```
(while true; do echo –n B >> file.log; sleep 1; done) &
```

Compound Commands

- List of commands separated by semi-colons
- Put the list between () to run them all in a subshell (treat it all as a one command)

```
date; who | wc –l >> mylogfile.txt  # only the second command will be logged
( date; who | wc –l) >> mylogfile.txt  # all output will be logged
```

Scheduling a Process

- `at time <commands>` commands entered one per line, terminate with Ctl+D
- `atq [user]` lists the current `at` jobs
- `atrm [user|atJobID]` removes at jobs

```
at 8:00pm December 7
at 7 am Thursday
at now + 5 minutes
at midnight + 23 minutes
```

Scheduling a Process Periodically

- `crontab` used to install, deinstall or list the tables (crontabs).
  - `-u user` the user whose crontab is to be tweaked
  - `-l` display the current crontab file
  - `-r` remove the current crontab file
  - `-e` edit. After exit from the editor, the modified crontab will be installed
- Cronttab file:
  - Space delimited
  - Fields: minute, hour (0-23), day of month (0-31), month (1-12), and day of week (0=Sun to 6).

```
Min  Hour  DoM  Month  DoW   Commands
3    4     *    *      1,3,5 find ~ -name core | xargs rm –f P{}
```

bash Shell Scripting Basics

Creating Shell Scripts

- First line contains the magic "shbang" `#!/bin/bash`
- Comments start with #
- One command spans multiple lines with \
- By convenient, they have sh extension

Handling Input

- `read` assigns an input word(s) to a shell variable
- words are separated by default with space. `IFS` variable controls the separator.

```
#!/bin/bash
read –p "Enter the words:" word1 word2 word3
echo "Word1 : $word1"
echo "Word2 : $word2"
echo "Word3 : $word3"
```

Shell Script Debugging

- Modify the shebang as follows

  `#!/bin/bash –x`

  `#!/bin/bash -v`

- Alternatively, Invoke the shell interpreter with debug options

  `bash –x scriptname`

  `bash –v scriptname`

Handling Positional Parameters (Arguments)

- accessed by `$1, $2, …, $9, ${10},${11},…`
- `$0` reserved for the program name
- $* holds all command line parameters
- $# holds number of command line parameters

```
#!/bin/bash
printf "First  Parameter :%s\n" $1
printf "Second Parameter :%s\n" $2
echo –e "\nAll   Parameters: $*\n" # -e option enables interpretation of the
                                  # backslash-escaped characters
```

Using Functions

- `functionname() { [return ...] }`
- Arguments passed to a function are accessed by its positional parameters $1, $2 … etc.
- `return` keyword sets the special variable `$?`
- Variables are made local in a function using `local` keyword.

```
#!/bin/bash
printname(){
  local firstname=$1 lastname=$2
  echo –e "Full name: $lastname $firstname\n"
  return 1
}
printname Ahmed Baraka
retval=$?
echo "Returned value: $retval"
```

Exit Status

- `$?` contains exit status of the most recently executed command.
- It takes values 0 for success, 1-255 for failure
- `exit` sets an exist status in a script

Conditional Execution

- `<cmd1> && <cmd2>` execute cmd2 if cmd1 succeeds
- `<cmd1> || <cmd2>` execute cmd2 if cmd1 fails

```
ping -c1 -W2 pc1 &> /dev/null \
> && echo "pc1 is up" \
> || $(echo 'pc1 is unreachable'; exit 1)
```

Using the if Statement

```
if [ condition ]; then
 …
elif [condition]; the
 …
else
 …
fi
```

```
if [ $retval != 0 ]; then
  echo "There was an error running the application"
  exit $retval
fi
```

Using the Case Statement

```
case variable in
 pattern1)
  <command>;;
 pattern2)
  <command>;;
esac
```

```
#!/bin/bash
. ~/lib/funcs
case $1 in
  start)
   start_func;;
  stop)
   stop_func;;
  restart)
   stop_func
   start_func;;
  status)
   status_func;;
  *)
   echo "Use Command"
 esac
```

Using the For Loop

```
for variable in list-of-vlaues
do
  commands...
done
```

```
#!/bin/sh
echo "Please enter a list of numbers between 1 and 100. "
read NUMBERS
for NUM in $NUMBERS
do
  if [ "$NUM" -lt 1 ] || [ "$NUM" -gt 100 ]; then
    echo "Invalid Number ($NUM) - Must be between 1 and 100!"
  else
    echo "$NUM is valid."
  fi
done
```

Using the While loop

```
while condition
do
  commands...
done
```

```
#!/bin/sh
# Guess the number game.

ANSWER=5          # The correct answer
CORRECT=false     # The correct flag

while [ "$CORRECT" != "true" ]
do
        # Ask the user for the number...
        echo "Guess a number between 1 and 10. "
        read NUM

        # Validate the input...
        if [ "$NUM" -lt 1 ] || [ "$NUM" -gt 10 ]; then
                echo "The number must be between 1 and 10!"
        elif [ "$NUM" -eq "$ANSWER" ]; then
                echo "You got the answer correct!"
                CORRECT=true
        else
                echo "Sorry, incorrect."
        fi
done
```

```
(while true; do echo -n B >> file.log; sleep 1; done)
```

Disrupting Loops

- `continue` jump back to the initial condition
- `break` jump to the command past the `done`

File Tests

- Common file tests are:
  - `-e` file exists
  - `-f` file exists and is a regular file
  - `-d` file exists and is a directory
  - `-x` file exists and is an executable
  - `-h` file exists and is symbolic link
  - `-r` file exists and is readable by you
  - `-s` file exists and is not empty
  - `-w` file exists and is writable by you
  - `-O` file exists and is effectively owned by you
  - `-G` file exists and is effectively owned by your group
- `help test` for the complete list

```
if [ -f $HOME/lib/functions ]; then
   ...
fi
```

String Tests

- String operators:

`-z STRING`	True if string is empty.
`-n STRING`	True if string is not empty.
`STRING1 = STRING2`	True if the strings are equal.
`STRING1 != STRING2`	True if the strings are not equal.
`STRING1 < STRING2`	True if STRING1 sorts before STRING2 lexicographically.
`STRING1 > STRING2`	True if STRING1 sorts after STRING2 lexicographically.

Shell Option Test

- Shell option operator

`-o OPTION`	True if the shell option OPTION is enabled.

Logical Tests

- Logical Operators

`! EXPR`	True if expr is false.
`EXPR1 -a EXPR2`	True if both expr1 AND expr2 are true.
`EXPR1 -o EXPR2`	True if either expr1 OR expr2 is true.

Comparison

- Comparison Operators

`arg1 OP arg2`	OP is one of: -eq, -ne, -lt, -le, -gt, or -ge.